Deckhouse Code Documentation

Generated: September 22, 2026

Documentation

About Deckhouse Code

Purpose and features

Deckhouse Code organizes collaborative work between software development and operations teams: it covers code storage and versioning, code review, CI/CD pipelines, and delivery to target environments.

Deckhouse Code covers the following scenarios:

- storing and versioning Git repositories, including forks, protected branches, and pull mirroring;
- reviewing and merging changes through merge requests with approval rules and merge trains;
- running CI/CD pipelines defined in `.gitlab-ci.yml`;
- managing projects, groups, and access rights;
- scanning code for vulnerabilities and tracking dependencies, in Enterprise Edition.

Installation types

Deckhouse Code is delivered as a Linux package, an Omnibus Docker image, a Helm Chart or a Deckhouse Kubernetes Platform module.

Installation type	What it is	When it applies	Installation documentation
Linux package	Deckhouse Code installed from a `.deb` or `.rpm` package on a single host	For a host running a supported Linux distribution directly, without a container or a Kubernetes cluster	Quick start
Omnibus Docker	Deckhouse Code running as a single container built from the same omnibus package	For a host running Docker instead of the package directly	Quick start
Helm Chart	Deckhouse Code deployed to a Kubernetes cluster with a Helm chart, without the Deckhouse Kubernetes Platform operator	For a Kubernetes cluster managed independently of Deckhouse Kubernetes Platform	Quick start
Deckhouse Kubernetes Platform module	Deckhouse Code installed and managed by an operator through the CodeInstance resource	For a Deckhouse Kubernetes Platform cluster	Deckhouse Kubernetes Platform module

Integrations

Deckhouse Code supports the following integrations:

- Jira — GitLab's Jira integration documentation.
- Jenkins — GitLab's Jenkins integration documentation.
- EvaProject — the EvaProject integration guide.

Next steps

Deckhouse Code is available in Standard Edition and Enterprise Edition; see Editions for the feature differences.

For instance administration, see the Administration Overview. For everyday work with projects, repositories, and merge requests, see the Usage Overview.

Editions

Deckhouse Code ships in Standard Edition (SE) and Enterprise Edition (EE).

Feature availability

Category	Features	SE	EE
Installation types	Linux package, Omnibus Docker, Helm Chart, Deckhouse Kubernetes Platform module	Yes	Yes
Projects and groups	Groups and members, project visibility levels, project import, issues, labels, milestones, boards, project and group wikis	Yes	Yes
Source code	Git repositories with forks and the web editor, protected branches, push rules, pull mirroring, CODEOWNERS, Git access over SSH and HTTPS	Yes	Yes
Code review and merge requests	Merge requests, approval rules, external status checks, merge trains, code review analytics, merge request analytics	Yes	Yes
CI/CD	Pipelines from `.gitlab-ci.yml`, secrets from an external Vault or Deckhouse Stronghold, service accounts for pipelines	Yes	Yes
Security scanning and reporting	SAST scanning with Semgrep, a dependency list from	No	Yes

Category	Features	SE	EE
	the CycloneDX SBOM, license compliance		
Analytics reports	Issue analytics, CI/CD analytics, group repository analytics	No	Yes
Project repository analytics	Programming languages, test coverage and commit statistics of a project	Yes	Yes
Security settings and audit	Group and project security settings, instance security settings, security credentials, audit events	Yes	Yes
Identity and access	Sign-in with personal two-factor authentication, OmniAuth providers, LDAP synchronization, sign-in restrictions, group and project creation restrictions, protocol restrictions, instance-wide two-factor authentication	Yes	Yes
Search	Advanced search, search indexing with OpenSearch	Yes	Yes
Integrations	Webhooks for group, project, and subgroup events	Yes	Yes
Operations	Resource limits and performance settings, monitoring,	Yes	Yes

Category	Features	SE	EE
	maintenance mode, backup and restore, upgrade, troubleshooting		

Pages available only in Enterprise Edition carry an “EE” badge in the documentation sidebar.

Installation

Linux package

Requirements

The Linux package deploys Deckhouse Code on a single server: the web interface, Git, the database and the background jobs all run on that machine.

Operating system

Supported distributions, the architecture is `x86_64`:

- RED OS 8.x
- Ubuntu 24.04 LTS

System dependencies are installed from the OS repositories, so access to them is required during the installation. Internet access is not required.

Resources

The values apply to a typical instance:

Resource	Value
Memory	8 GB; 6 GB with a swap partition at the minimum
Disk	20 GB or more for the `/var/opt/gitlab` data directory

For an instance serving hundreds of active users or more, request sizing recommendations from the vendor.

Installation package

Obtain the installation package from the vendor; the file name depends on the OS:

OS	Package file
RED OS 8.x	`deckhouse-code-<VERSION>.el8.x86_64.rpm`
Ubuntu 24.04 LTS	`deckhouse-code_<VERSION>_amd64.deb`

`<VERSION>` is the version of the package you received, for example `1.0.0-0`.

Network ports

The instance accepts connections on the ports that are opened on the server firewall:

Port	Purpose
`22`	Git over SSH, served by the system sshd
`80`	Redirect to HTTPS
`443`	Web interface and Git over HTTPS

Quick start

The instance first starts over HTTP, then TLS is enabled on it. Before you start, make sure the server meets the requirements.

In the commands and configuration files below, replace `<HOSTNAME>` with the FQDN of the instance and `<VERSION>` with the version of the package you received, for example `1.0.0-0`.

> The CLI utilities (`gitlab-ctl`, `gitlab-backup`), the configuration file (`/etc/gitlab/gitlab.rb`) and the service directories (`/opt/gitlab`, `/var/opt/gitlab`) keep the `gitlab-` prefix: Deckhouse Code runs on the GitLab engine, and the system names are left unchanged.

Preparation

The instance name `<HOSTNAME>` is used in the web interface address, in the repository addresses for Git, in the `external_url` setting and in the CN and SAN fields of the certificate — the same name everywhere.

1. Check that the name resolves to the IP address of the machine:

   ```
   getent hosts <HOSTNAME>
   ```

2. If there is no A or AAAA DNS record, add the name to the `/etc/hosts` file on the server:

   ```
   echo "<IP_ADDRESS> <HOSTNAME>" | sudo tee -a /etc/hosts
   ```

   `<IP_ADDRESS>` is the IP address of the instance machine. Add the same line on every machine that will open the web interface and work with Git, including your workstation.

Installing the package

The installation starts the initial configuration (`gitlab-ctl reconfigure`) on its own, which takes several minutes. The instance address is taken from the `EXTERNAL_URL` variable during the first installation; TLS is enabled at the next step.

RED OS 8

Install the system dependencies:

```
sudo dnf install -y libxcrypt-compat policycoreutils-python-utils
```

The `libxcrypt-compat` package provides the `libcrypt.so.1` library, without which the application does not start: the minimal RED OS installation does not contain it. The `policycoreutils-python-utils` package provides the `semanage` utility, which sets the SELinux contexts during the initial configuration.

Install the Deckhouse Code package:

```
sudo EXTERNAL_URL="http://<HOSTNAME>" \
  rpm -i ./deckhouse-code-<VERSION>.el8.x86_64.rpm
```

The `enforcing` SELinux mode is supported: the policy module is installed during the initial configuration, and no manual action is needed.

Ubuntu 24.04

There is no need to install the system dependencies separately: they are declared in the package, and `apt` takes them from the OS repositories.

Install the Deckhouse Code package:

```
sudo EXTERNAL_URL="http://<HOSTNAME>" \
  apt install -y ./deckhouse-code_<VERSION>_amd64.deb
```

The `./` prefix in the file path is required: without it, `apt` looks for a package with that name in the repositories.

What you get

The instance is running at `http://<HOSTNAME>` : the web interface, Git over HTTP, the database and the background jobs. Check its state — the commands run on the instance machine itself:

```
# Every service is in the run state.
sudo gitlab-ctl status
# Code 200 is expected.
curl -s -o /dev/null -w "%{http_code}\n" http://localhost/-/health
# The first administrator password.
sudo cat /etc/gitlab/initial_root_password
```

Open `http://<HOSTNAME>` in a browser and sign in as the `root` user with the password from the `/etc/gitlab/initial_root_password` file.

> ⚠ Save the password: the `initial_root_password` file is kept for 24 hours, and any further `gitlab-ctl reconfigure` run deletes it once more than a day has passed since the installation. The password value itself does not change.

TLS

Over HTTP, passwords and repository contents travel in plain text, so enable TLS right after the first sign-in.

Prepare the directory for the certificates:

```
sudo mkdir -p /etc/gitlab/ssl && sudo chmod 755 /etc/gitlab/ssl
```

Place the certificate and the key in it under the names `<HOSTNAME>.crt` and `<HOSTNAME>.key`.

Your own certificate

Issue a certificate for the `<HOSTNAME>` name in your certificate authority and copy the files to the server:

```
sudo install -m 600 <KEY_FILE> /etc/gitlab/ssl/<HOSTNAME>.key
# The certificate together with its chain.
sudo install -m 644 <CERT_FILE> /etc/gitlab/ssl/<HOSTNAME>.crt
```

`<KEY_FILE>` and `<CERT_FILE>` are the paths to the key and certificate files you received.

Self-signed certificate

On a test stand, issue the certificate on the spot. The instance name in the SAN field is required, otherwise clients reject the certificate:

```
sudo openssl req -x509 -nodes -newkey rsa:2048 -days 825 \
  -keyout /etc/gitlab/ssl/<HOSTNAME>.key \
  -out /etc/gitlab/ssl/<HOSTNAME>.crt \
  -subj "/CN=<HOSTNAME>/O=Deckhouse Code/C=RU" \
  -addext "subjectAltName=DNS:<HOSTNAME>"
sudo chmod 600 /etc/gitlab/ssl/<HOSTNAME>.key
```

Enable HTTPS in the `/etc/gitlab/gitlab.rb` file:

```
external_url 'https://<HOSTNAME>'
letsencrypt['enable'] = false
nginx['redirect_http_to_https'] = true
nginx['ssl_certificate'] = "/etc/gitlab/ssl/<HOSTNAME>.crt"
nginx['ssl_certificate_key'] = "/etc/gitlab/ssl/<HOSTNAME>.key"
```

Without the `letsencrypt['enable'] = false` line, the configuration with an `https://` address starts obtaining a certificate through Let’s Encrypt over HTTP-01 and fails in an isolated network.

Apply the configuration:

```
sudo gitlab-ctl reconfigure
```

Verification

Check the state of the services, the response over HTTPS and the redirect from HTTP:

```
# Every service is in the run state.
sudo gitlab-ctl status
# The --resolve option sends the request to the loopback address: /-/health answers
# only on the machine itself, while the name check against the certificate is kept.
# health=200 tls=0 is expected.
curl --cacert /etc/gitlab/ssl/<HOSTNAME>.crt \
     --resolve '<HOSTNAME>:443:127.0.0.1' \
     -o /dev/null -w "health=%{http_code} tls=%{ssl_verify_result}\n" \
     https://<HOSTNAME>/-/health
# The redirect from HTTP to HTTPS, https://<HOSTNAME>/ is expected.
curl -sI http://<HOSTNAME>/ | grep -i location
```

A browser and Git reject a self-signed certificate until its certificate authority is trusted. For Git, set the path to the certificate file on the client machine:

```
git config --global http.https://<HOSTNAME>.sslCAInfo <CERT_FILE>
```

Change the password of the `root` user in the web interface and delete the initial password file:

```
sudo rm -f /etc/gitlab/initial_root_password
```

Post-installation setup

These settings are made before the instance is handed over to its users. Changes in the `/etc/gitlab/gitlab.rb` file are applied with the `sudo gitlab-ctl reconfigure` command.

Interface language

The language is set for the whole instance by an administrator and separately by each user for themselves.

The instance language applies to the sign-in page and to every user who has not chosen a language manually. Sign in as an administrator, select “Admin” in the top right corner, go to “Settings” → “Preferences” in the left pane, scroll to the “Localization” section, select the language in the “Default language” field, for example “Russian - русский”, and click “Save changes”.

A user changes the language for themselves: the avatar in the top right corner → “Preferences” → “Localization” → “Language” → the language → “Save changes”. The interface switches after the page is reloaded. The personal choice takes precedence over the instance language.

Email (SMTP)

Without email, password reset, invitations and notifications do not work. Set the connection parameters of the mail server in the `/etc/gitlab/gitlab.rb` file:

```
gitlab_rails['smtp_enable'] = true
gitlab_rails['smtp_address'] = "<SMTP_HOST>"
gitlab_rails['smtp_port'] = 587
gitlab_rails['smtp_user_name'] = "<SMTP_USER>"
gitlab_rails['smtp_password'] = "<SMTP_PASSWORD>"
gitlab_rails['smtp_domain'] = "<DOMAIN>"
gitlab_rails['smtp_authentication'] = "login"
gitlab_rails['smtp_enable_starttls_auto'] = true
gitlab_rails['gitlab_email_from'] = "deckhouse-code@<DOMAIN>"
```

Apply the configuration and send a test message to your own address:

```
sudo gitlab-ctl reconfigure
sudo gitlab-rails runner \
  "Notify.test_email('<ADMIN_EMAIL>', 'Test', 'It works').deliver_now"
```

Network access (firewall)

The instance needs ports `22` (Git over SSH through the system sshd), `80` (redirect to HTTPS) and `443` (web interface and Git over HTTPS). Open them with the tools of your OS.

firewalld (RED OS)

```
sudo systemctl enable --now firewalld
sudo firewall-cmd --permanent --add-service={ssh,http,https}
sudo firewall-cmd --reload
```

ufw (Ubuntu)

```
sudo ufw allow OpenSSH
sudo ufw allow 80,443/tcp
sudo ufw enable
```

Resources

With 6 to 8 GB of memory, fix the number of worker processes so that the automatic tuning does not take all the memory. Set the values in the `/etc/gitlab/gitlab.rb` file:

```
puma['worker_processes'] = 2
sidekiq['concurrency'] = 10
```

Apply the configuration:

```
sudo gitlab-ctl reconfigure
```

Check that a swap partition is configured on the server: it covers the memory peaks during the migrations of an upgrade and during heavy repository operations.

SSH port

If the SSH service of the instance listens on a port other than 22, name it in `/etc/gitlab/gitlab.rb`, so that the clone URLs in the web interface carry that port:

```
gitlab_rails['gitlab_shell_ssh_port'] = 2222
```

Apply the configuration:

```
sudo gitlab-ctl reconfigure
```

Advanced search (OpenSearch)

Advanced search runs on an external OpenSearch cluster. Set the connection in `/etc/gitlab/gitlab.rb` and route the indexing jobs to their own Sidekiq queue:

```
gitlab_rails['fe_search'] = {
  'advanced_search_enabled' => true,
  'opensearch' => {
    'url' => 'https://<OPENSEARCH_HOST>:9200',
    'username' => '<OPENSEARCH_USER>',
    'password' => '<OPENSEARCH_PASSWORD>'
  }
}
sidekiq['routing_rules'] = [
  ['feature_category=fe_global_search', 'global-search-indexing'],
  ['*', 'default']
]
```

`<OPENSEARCH_HOST>` is the address of the OpenSearch cluster, `<OPENSEARCH_USER>` and `<OPENSEARCH_PASSWORD>` are the credentials of the account that owns the indexes. Apply the configuration:

```
sudo gitlab-ctl reconfigure
```

Then reindex the instance and enable advanced search in the settings, as described in Advanced search.

Troubleshooting

The errors below occur during the Linux package installation and the initial configuration of the instance. If your case is not here, collect the output of the commands from the General diagnostics section and contact the vendor.

Application does not start: the libcrypt.so.1 library is missing

Where it appears. RED OS, the output of the package installation and the initial configuration:

```
libcrypt.so.1: cannot open shared object file
```

Cause. The minimal RED OS installation does not contain the `libcrypt.so.1` library.

Action. Install the package with the library and repeat the configuration:

```
sudo dnf install -y libxcrypt-compat
sudo gitlab-ctl reconfigure
```

Configuration fails with an SELinux error

Where it appears. RED OS, the `gitlab-ctl reconfigure` run, the `bash[Set proper security context on ssh files for selinux]` resource:

```
ValueError: Type gitlab_shell_t is invalid
semanage: command not found
```

Cause. The server has no SELinux utilities from the `policycoreutils-python-utils` package, or the policy module was not loaded during the initial configuration.

Action. Install the utilities, repeat the configuration and check that the policy module is loaded:

```
sudo dnf install -y policycoreutils-python-utils
sudo gitlab-ctl reconfigure
sudo semodule -l | grep -i gitlab
```

Instance name is unreachable: curl returns 000

Cause. The `<HOSTNAME>` name does not resolve on the server or on the machine that opens the web interface, or port `80` or `443` is unreachable: closed by the firewall or nginx is not running.

Action.

1. Check the name with the `getent hosts <HOSTNAME>` command. If the name is missing, create an A or AAAA DNS record pointing to the IP address of the machine, or add a line to `/etc/hosts` on a stand without DNS. The name must match `external_url` and the CN and SAN fields of the certificate.
2. If the name resolves to the expected IP address, check the state of nginx and the listening ports on the server:

   ```
   sudo gitlab-ctl status nginx
   sudo ss -tlnp | grep -E ':(80|443)\b'
   ```

3. Check the HTTP entry point from the workstation:

   ```
   curl -sS -o /dev/null -w "HTTP %{http_code}\n" \
     http://<HOSTNAME>/
   ```

> Any HTTP code (usually 301 or 302 once TLS is enabled) means that nginx is reachable. Code 000 means that the connection was not established. If nginx is running and the port is closed from the outside, open it as described in Network access (firewall).

Web interface returns 502 after configuration or restart

Cause. The application is still starting: during the first minute or two after `reconfigure` or `restart`, code 502 is expected.

Action. Wait and check the state of the services with the `sudo gitlab-ctl status` command. If 502 lasts longer than a few minutes, read the log with `sudo gitlab-ctl tail puma`: a frequent cause is a lack of memory, see Resources.

Configuration fails with a Let's Encrypt error

Where it appears. The `gitlab-ctl reconfigure` run after an `https://` address has been set in `external_url` while the certificate files are not on disk.

Cause. Without a certificate on disk, obtaining a certificate through Let's Encrypt is turned on, which is impossible in an isolated network.

Action. Place the certificate and set `letsencrypt['enable'] = false` as described in TLS, then run `sudo gitlab-ctl reconfigure`.

Reading an error in the reconfigure output

Where it appears. The output of a `gitlab-ctl reconfigure` command that ended with an error.

Action. At the end of the output, find the block with the resource name and the cause:

```
Error executing action ... on resource '...'
```

The full run log is kept in the `/var/log/gitlab/reconfigure/` directory.

General diagnostics

These commands apply to any of the errors listed above:

```
# The state of every service.
sudo gitlab-ctl status
# The live log of a service, for example puma or nginx.
sudo gitlab-ctl tail <SERVICE>
# The instance self-check.
sudo gitlab-rake gitlab:check SANITIZE=true
# The log directories per service.
ls /var/log/gitlab/
```

Omnibus Docker

Requirements

Deckhouse Code runs as one container that holds the same omnibus package as the Linux package installation. The host provides Docker Engine, the resources of a single instance, the directories behind the volumes and the network ports.

Host

The container runs on a Linux host with the x86_64 architecture and Docker Engine.

Memory and disk are those of the Linux package installation, listed in Requirements: 8 GB of RAM (6 GB plus swap as the minimum) and at least 20 GB of free space for the instance data. The image takes another 2.5–3 GB on the host.

Volumes

The image declares volumes for the configuration, the data and the logs. Mount all of them: they hold the state that outlives the container, and an upgrade replaces the container while keeping the volumes.

Path in the container	Content
`/etc/gitlab`	Configuration file `gitlab.rb`, encryption keys `gitlab-secrets.json`, TLS certificates, SSH host keys, the initial `root` password file.
`/var/opt/gitlab`	Repositories, database, uploaded files, CI artifacts, package registry, backup archives in `/var/opt/gitlab/backups`.
`/var/log/gitlab`	Logs of every service and the log of each configuration run in `/var/log/gitlab/reconfigure`.

In the examples on these pages the volumes are host directories: `/srv/code/config`, `/srv/code/data` and `/srv/code/logs`.

Network ports

The exposed ports carry Git over SSH, HTTP and HTTPS traffic.

Port	What uses it
`22`	Git over SSH. The container runs its own `sshd`, which accepts the `git` user and public-key authentication.
`80`	Web interface and Git over HTTP, redirect to HTTPS once TLS is configured.
`443`	Web interface and Git over HTTPS.

Publish the ports when you create the container. A host port may differ from the container port: in that case the address of the instance names the host port that users open, and an SSH port other than 22 is named in the configuration, as described in Post-installation setup.

Shared memory

Create the container with `--shm-size 256m`. Docker gives `/dev/shm` 64 MB by default.

Quick start

A running instance takes one `docker run` command and the first start that configures it. The instance answers over HTTP first; TLS is the last step of this page.

Before you start

Check the host against Requirements and pick `<HOSTNAME>`: the name users open in the browser and put into Git remotes. The same name goes into the address of the instance and into the certificate.

Check that the name resolves to the address of the host:

```
getent hosts <HOSTNAME>
```

Without a DNS record, add the name to `/etc/hosts` on the host and on every machine that opens the web interface or works with Git.

Getting the image

The image is published per flavor: a flavor names the base distribution and stands for `<FLAVOR>` in the image reference.

Flavor	Base distribution
`ubuntu_22.04`	Ubuntu 22.04
`ubuntu_24.04`	Ubuntu 24.04
`redos_8`	RED OS 8

Every tag names a version, and no tag follows the latest one, so both the first run and every upgrade name the version explicitly:

```
docker pull <REGISTRY>/<FLAVOR>:<VERSION>
```

Starting the container

Create the container with the volumes, the published ports and the address of the instance:

```
docker run -d --name code \
  --shm-size 256m \
  -p 80:80 -p 443:443 -p 22:22 \
  -e GITLAB_OMNIBUS_CONFIG="external_url 'http://<HOSTNAME>'" \
  -v /srv/code/config:/etc/gitlab \
  -v /srv/code/logs:/var/log/gitlab \
  -v /srv/code/data:/var/opt/gitlab \
  <REGISTRY>/<FLAVOR>:<VERSION>
```

The name `code` is used by every command below. The left-hand port of each `-p` is the host port and may differ from the container port; port 22 on the host is usually taken by the host `sshd`, and Post-installation setup describes the setting that names another port in clone URLs.

Ports, volumes and environment variables are fixed when the container is created. Changing any of them means removing the container and creating a new one with the same volumes.

Address of the instance

Without configuration the address is `http://<CONTAINER_HOSTNAME>`, and Docker sets the container host name to the container identifier unless `--hostname` is given. That identifier would then appear in every link and clone URL, so set the address on the first run.

The address is the `external_url` setting of `gitlab.rb`, and the image takes it from the environment variable and from the configuration file:

- `GITLAB_OMNIBUS_CONFIG` holds configuration lines and is evaluated on every start;
- `/etc/gitlab/gitlab.rb` on the configuration volume is read after that, so a setting written into the file overrides the same setting in the variable.

`EXTERNAL_URL`, the variable the package installer reads, has no effect in the container.

First start

The start-up script prepares the instance before the services accept requests:

- copies `gitlab.rb` from the template when the configuration volume has no such file;
- generates the SSH host keys into `/etc/gitlab` when they are absent;
- runs `gitlab-ctl reconfigure`, the step that takes several minutes on the first start;
- brings the database to the current PostgreSQL version;
- tails the service logs into the container output.

Follow the start in the container output:

```
docker logs -f code
```

The image carries a health check that runs every 60 seconds. Until the first successful check the container health is `starting`, and five failed checks in a row turn it into `unhealthy`, which the first start reaches while the configuration is still running:

```
docker inspect --format '{{.State.Health.Status}}' code
```

The start has finished when the health is `healthy` and every service is in state `run`:

```
docker exec code gitlab-ctl status
```

First sign-in

The first `root` password is written to the configuration volume. Read it from the container:

```
docker exec code cat /etc/gitlab/initial_root_password
```

Open `http://<HOSTNAME>` and sign in as `root` with that password.

> ⚠ The password file is deleted by the first configuration run that happens more than 24 hours after the file was written, and the TLS step below is such a run. Change the `root` password in the web interface and delete the file: `docker exec code rm -f /etc/gitlab/initial_root_password`.

HTTP carries the sign-in data and the repository content unencrypted. An instance that serves users runs over TLS.

TLS

Put the certificate and the key on the configuration volume, so that the container reads them from `/etc/gitlab/ssl`:

```
sudo mkdir -p /srv/code/config/ssl && sudo chmod 755 /srv/code/config/ssl
sudo install -m 644 <CERT_FILE> /srv/code/config/ssl/<HOSTNAME>.crt
sudo install -m 600 <KEY_FILE> /srv/code/config/ssl/<HOSTNAME>.key
```

`<CERT_FILE>` is the certificate file in PEM format, `<KEY_FILE>` is its private key file.

Issuing the certificate, the requirement for a `subjectAltName` and the reason Let's Encrypt is turned off are described in Quick start.

Add the settings to `/srv/code/config/gitlab.rb`, which the container reads as `/etc/gitlab/gitlab.rb`:

```
external_url 'https://<HOSTNAME>'
letsencrypt['enable'] = false
nginx['redirect_http_to_https'] = true
nginx['ssl_certificate'] = "/etc/gitlab/ssl/<HOSTNAME>.crt"
nginx['ssl_certificate_key'] = "/etc/gitlab/ssl/<HOSTNAME>.key"
```

The paths in the settings are container paths. Apply the change by restarting the container, because every start runs the configuration:

```
docker restart code
```

Verification

Check the services, the health check and the redirect from HTTP:

```
docker exec code gitlab-ctl status
docker exec code gitlab-healthcheck --fail --max-time 10
docker inspect --format '{{.State.Health.Status}}' code
curl -sI http://<HOSTNAME>/ | grep -i location
```

Open `https://<HOSTNAME>` in the browser, sign in and create a test project to check Git access over HTTPS and SSH.

Post-installation setup

The settings of an instance in a container are the settings of the Linux package installation, and the container adds its own layer: where the configuration file lives, which variables the start-up script reads and which ports the host publishes.

Applying a configuration change

The configuration file is `/etc/gitlab/gitlab.rb` on the configuration volume, that is `/srv/code/config/gitlab.rb` on the host. Edit it on the host, or in the container:

```
docker exec -it code editor /etc/gitlab/gitlab.rb
```

Apply the change without stopping the container:

```
docker exec code gitlab-ctl reconfigure
```

A restart has the same effect, because every start of the container runs the configuration:

```
docker restart code
```

Interface language, SMTP, worker counts and backup retention are the same settings as on a host installation; what each of them does is described in Post-installation setup.

Start-up variables

The start-up script reads its variables when the container starts, so a change to any of them means creating the container anew with the same volumes.

Variable	What it does
`GITLAB_OMNIBUS_CONFIG`	Holds configuration lines, which are evaluated before `/etc/gitlab/gitlab.rb` is read. A setting present in the file overrides the same setting in the variable.
`GITLAB_SKIP_RECONFIGURE`	With the value `true` on an already configured instance, the start skips `gitlab-ctl reconfigure`, and configuration changes stay unapplied until the next configuration run.
`GITLAB_SKIP_PG_UPGRADE`	With the value `true`, the start does not run `gitlab-ctl pg-upgrade` and the database stays on its current PostgreSQL version.
`GITLAB_DISABLE_OPENSSH`	With the value `true`, the start does not prepare the `sshd` service, and Git over SSH is unavailable.
`GITLAB_SKIP_TAIL_LOGS`	With the value `true`, the service logs are not written to the container output; they stay on the log volume.
`GITLAB_PRE_RECONFIGURE_SCRIPT`	When set, its value runs as a shell command before the services start.
`GITLAB_POST_RECONFIGURE_SCRIPT`	When set, its value runs as a shell command after the PostgreSQL step.
`GITLAB_ALLOW_SHA1_RSA`	With the value `true`, the `sshd` configuration accepts the `ssh-rsa` host key algorithm and public key type.
`OPENSSL_FORCE_FIPS_MODE`	Its value is written to the environment of the `sshd` service.

SSH host keys

The RSA, ECDSA and Ed25519 host keys are generated on the first start into `/etc/gitlab` and linked into `/etc/ssh`, where the instance reads them. They are on the configuration volume, so a container replaced during an upgrade keeps them and Git clients see the same host key as before. A new configuration volume means new keys, and every client reports a changed host key on the next connection.

Git over SSH uses the keys that users add in the web interface: `sshd` in the container listens on port 22, accepts the `git` user only and has password authentication turned off.

Memory and workers

The container gets the memory of the host unless `--memory` limits it at creation, and it needs `--shm-size 256m` as described in Requirements.

On a host with 6–8 GB of RAM, fix the number of workers instead of leaving it to autotuning. The settings go into `/etc/gitlab/gitlab.rb` or into `GITLAB_OMNIBUS_CONFIG`, and their values are described in Post-installation setup:

```
puma['worker_processes'] = 2
sidekiq['concurrency'] = 10
```

Ports and the host firewall

Ports 80, 443 and 22 are published when the container is created. A host port may differ from the container port, as in `-p 8080:80 -p 8443:443 -p 2222:22`.

The address of the instance names the host port that users open; the SSH port that the web interface shows in clone URLs is the `gitlab_rails['gitlab_shell_ssh_port']` setting, described in SSH port.

Open the published ports on the host for the networks of the users; the commands for `firewalld` and `ufw` are in Post-installation setup. Docker publishes ports through its own `iptables` rules, so check the ports from a workstation after each change of the firewall rules:

```
curl -sI http://<HOSTNAME>/ | head -n 1
ssh -T -p 22 git@<HOSTNAME>
```

Advanced search (OpenSearch)

The OpenSearch connection is the `gitlab_rails['fe_search']` setting and the Sidekiq routing rule described in Advanced search (OpenSearch). Put them into `/etc/gitlab/gitlab.rb` in the configuration volume or into `GITLAB_OMNIBUS_CONFIG`, then restart the container or run `docker exec code gitlab-ctl reconfigure`.

Troubleshooting

Most failures of an instance in a container appear at start, and the reason is in the container output or in the logs on the log volume. Each section below gives the symptom, the cause and the action.

The web interface answers 502 after a start

Cause. every start runs `gitlab-ctl reconfigure`, and the application accepts requests only after it. The first start takes several minutes, a later one takes one or two.

Action. follow the start in the container output and wait for the configuration to finish:

```
docker logs -f code
docker exec code gitlab-ctl status
```

A 502 that holds for longer means the application is not starting: read its log and check the free memory on the host.

```
docker exec code gitlab-ctl tail puma
```

The container health stays unhealthy

Cause. the health check runs every 60 seconds and turns the container `unhealthy` after five failures in a row, which a first start reaches while the configuration is still running. A health that stays `unhealthy` after that means the services have not started.

Action. run the same check by hand and look at the services and at the configuration log:

```
docker exec code gitlab-healthcheck --fail --max-time 10
docker exec code gitlab-ctl status
docker exec code ls /var/log/gitlab/reconfigure
```

Services do not start after the volumes were copied or restored

Cause. the owner and the mode of the files in the volumes do not match the accounts inside the image, which are created with fixed identifiers.

Action. restore the ownership with the command the image ships and restart the container:

```
docker exec -it code update-permissions
docker restart code
```

The database upgrade failed and the container exited

Where it appears. the container output holds `Upgrading the existing database failed and was reverted.`

Cause. the start runs `gitlab-ctl pg-upgrade`, the upgrade failed and the database was rolled back to its previous version.

Action. create the container again with `-e GITLAB_SKIP_PG_UPGRADE=true` added to the command from Quick start. The instance starts on the current PostgreSQL version, and the database upgrade is then handled as described in Upgrade.

The instance does not answer over HTTPS

Cause. the image exposes port 443, but a port is reachable only when the container publishes it, and ports are fixed when the container is created.

Action. remove the container and create it again from the command in Quick start, with `-p 443:443` and the same volumes; the data stays on the volumes:

```
docker stop code
docker rm code
```

Links and clone URLs name the container identifier

Cause. the address of the instance is not set, and the image builds it from the host name of the container, which Docker sets to the container identifier.

Action. set `external_url` in `/etc/gitlab/gitlab.rb` and restart the container, or create the container again with the address in `GITLAB_OMNIBUS_CONFIG`, as described in Quick start.

The container stops at the upgrade check

Where it appears. the container output reports the version found in the data and the version to upgrade to first.

Cause. the start compares the version on the data volume with the version in the image and stops when the instance cannot move to it in one step.

Action. run the tag of the intermediate version first, then the target one, as described in Upgrade.

A configuration change has no effect

Where it appears. the container output holds `Skipped reconfigure because GITLAB_SKIP_RECONFIGURE is set.`

Cause. the variable `GITLAB_SKIP_RECONFIGURE` is set to `true`, so the start leaves the configuration as it is. The same symptom without that line means the setting is present in `GITLAB_OMNIBUS_CONFIG` and is overridden by `/etc/gitlab/gitlab.rb`.

Action. run the configuration by hand, or create the container without the variable:

```
docker exec code gitlab-ctl reconfigure
```

Git over SSH is refused

Cause. port 22 of the container is not published, or `GITLAB_DISABLE_OPENSSH` is set to `true` and the `sshd` service is not prepared, or the client connects to a host port that the clone URL does not name.

Action. check the published ports and the variables of the container, then set the SSH port as described in Post-installation setup:

```
docker port code
docker inspect --format '{{.Config.Env}}' code
```

A client that reports a changed host key after a restart works against a new configuration volume: the host keys were generated anew.

General diagnostics

These commands read the state of a running container:

```
docker logs -f code
docker exec code gitlab-ctl status
docker exec code gitlab-ctl tail <SERVICE>
docker exec code gitlab-rake gitlab:check SANITIZE=true
docker inspect --format '{{.State.Health.Status}}' code
```

The logs of every service are on the log volume, `/srv/code/logs` on the host, and the log of each configuration run is in `/var/log/gitlab/reconfigure`.

Helm Chart

Requirements

The Helm Chart deploys Deckhouse Code directly to a Kubernetes cluster, without the Deckhouse Kubernetes Platform module. The cluster and the services below must already exist before you install the chart.

Kubernetes cluster

The cluster runs Kubernetes version `<VALUE>` or later.

Helm

The chart installs with Helm 3.

Ingress controller

An ingress controller is already running in the cluster. The chart creates Ingress objects for the web interface and for the optional container registry and Pages components; each one uses the ingress class named in `global.ingress.class`.

Default StorageClass

A default StorageClass, or a StorageClass named explicitly in `gitaly.persistence.storageClass`, so the cluster can provision the persistent volume claim each Gitaly pod uses for Git repository storage.

External PostgreSQL

An external PostgreSQL server, addressed by `global.psql.host`, that stores Deckhouse Code's application data. The chart does not deploy a database.

External Redis

An external Redis server, addressed by `global.redis.host`, used for background job queues, caching and session storage. The chart does not deploy Redis.

Object storage

An S3-compatible object storage bucket, configured through `global.appConfig.object_store`, that stores CI/CD artifacts, uploads and backup archives. The chart does not deploy object storage.

DNS name and TLS certificate

A DNS name for the instance, set in `global.hosts.domain`, that resolves to the ingress controller's address. A TLS certificate for that name, supplied as a Kubernetes Secret named in `global.ingress.tls.secretName`, whether issued manually or by a certificate manager already running in the cluster.

Quick start

Complete the checks in Requirements before you start. The steps below deploy Deckhouse Code with the values every installation needs at minimum.

Obtain the chart

Add the chart repository and update it:

```
helm repo add deckhouse-code <CHART_REGISTRY>
helm repo update
```

Prepare the values file

Create `values.yaml` with the domain, the ingress class, the TLS Secret, the StorageClass for Git repository storage, and the external PostgreSQL, Redis and object storage connections:

```
global:
  hosts:
    domain: code.example.com
  ingress:
    class: <INGRESS_CLASS_NAME>
    tls:
      secretName: <TLS_SECRET_NAME>
  psql:
    host: <POSTGRESQL_HOST>
    port: 5432
    database: <POSTGRESQL_DATABASE>
    username: <POSTGRESQL_USERNAME>
    password:
      secretName: <POSTGRESQL_PASSWORD_SECRET>
      key: password
  redis:
    host: <REDIS_HOST>
    port: 6379
    password:
      secretName: <REDIS_PASSWORD_SECRET>
      key: password
  appConfig:
    object_store:
      enabled: true
      connection:
        secretName: <OBJECT_STORE_CONNECTION_SECRET>
gitaly:
  persistence:
    size: <VALUE>
    storageClass: <STORAGE_CLASS_NAME>
```

Replace every placeholder with the actual class, Secret and host names. Configuration names every value shown here and what it configures.

Install the release

```
helm upgrade --install code deckhouse-code \
  -n code --create-namespace \
  -f values.yaml
```

The command creates the `code` namespace when it does not exist yet, and installs the release under the name `code`.

Point DNS at the ingress

Get the address the ingress controller assigned to the release's Ingress objects:

```
d8 k -n code get ingress
```

Point the DNS name from `global.hosts.domain` at the address shown in the `ADDRESS` column, as an A, AAAA or CNAME record depending on what the ingress controller returns.

Wait for the release to become ready

A migrations job runs once, before the pods that depend on the database schema start. Confirm every pod reaches the `Running` state:

```
d8 k -n code get pods
```

Sign in for the first time

The chart creates a Secret with the initial password for the `root` account, named after the release (`code-initial-root-password`, following the chart's initial-password naming convention). Retrieve it:

```
d8 k -n code get secret code-initial-root-password -o jsonpath='{.data.password}' |
base64 -d
```

Open `https://code.example.com` (the domain from `global.hosts.domain`) and sign in as `root` with that password.

Verify the installation

Confirm the sign-in page loads over HTTPS with the certificate from `global.ingress.tls.secretName`, and that every pod and the migrations job report a healthy state:

```
d8 k -n code get pods,jobs
```

Configuration

The values below configure a Deckhouse Code Helm release beyond the minimal install from Quick start. Every value name here is the same as in the chart's `values.yaml`; Advanced configuration lists every value the chart accepts.

Hosts and ingress

`global.hosts.domain` sets the base domain for the instance. `global.ingress.class` selects the ingress controller that serves it. The chart creates an Ingress object for the web interface at this domain, and one more for each optional component that is enabled, such as the container registry or Pages.

```
global:
  hosts:
    domain: code.example.com
  ingress:
    class: <INGRESS_CLASS_NAME>
```

TLS

`global.ingress.tls.secretName` names the Kubernetes Secret with the certificate and private key for `global.hosts.domain`. Every Ingress object the chart creates references this Secret.

```
global:
  ingress:
    tls:
      secretName: <TLS_SECRET_NAME>
```

The Secret must exist in the release's namespace before you run `helm upgrade`; the chart does not create or renew it.

PostgreSQL

`global.psql.host` and `global.psql.port` address the external PostgreSQL server. `global.psql.database` and `global.psql.username` select the database and the role Deckhouse Code connects as. `global.psql.password.secretName` and `global.psql.password.key` point to the Secret with the password.

```
global:
  psql:
    host: <POSTGRESQL_HOST>
    port: 5432
    database: <POSTGRESQL_DATABASE>
    username: <POSTGRESQL_USERNAME>
    password:
      secretName: <POSTGRESQL_PASSWORD_SECRET>
      key: password
```

Every component that reads or writes application data connects to this server; the chart does not deploy PostgreSQL.

Redis

`global.redis.host` and `global.redis.port` address the external Redis server. `global.redis.password.secretName` and `global.redis.password.key` point to the Secret with the password, when the server requires one.

```
global:
  redis:
    host: <REDIS_HOST>
    port: 6379
    password:
      secretName: <REDIS_PASSWORD_SECRET>
      key: password
```

Redis holds background job queues, caches and sessions; the chart does not deploy Redis.

Object storage

`global.appConfig.object_store.enabled` turns on S3-compatible object storage for CI/CD artifacts, uploads and backup archives. `global.appConfig.object_store.connection.secretName` names the Secret with the endpoint, region and access keys.

```
global:
  appConfig:
    object_store:
      enabled: true
      connection:
        secretName: <OBJECT_STORE_CONNECTION_SECRET>
```

Git storage (Gitaly persistence)

`gitaly.persistence.size` sets the size of the persistent volume claim each Gitaly pod mounts for Git repository storage. `gitaly.persistence.storageClass` selects the StorageClass it is provisioned from.

```
gitaly:
  persistence:
    size: <VALUE>
    storageClass: <STORAGE_CLASS_NAME>
```

Container registry

`registry.enabled` turns on the container registry component. `registry.storage.bucket` names the object storage bucket it stores image layers in, using the connection Secret from `global.appConfig.object_store.connection.secretName`.

```
registry:
  enabled: true
  storage:
    bucket: <REGISTRY_BUCKET_NAME>
```

Pages

`pages.enabled` turns on the Pages component. `pages.storage.bucket` names the object storage bucket its published sites are stored in.

```
pages:
  enabled: false
  storage:
    bucket: <PAGES_BUCKET_NAME>
```

SMTP

`global.smtp.enabled` turns on outgoing notification email. `global.smtp.address` and `global.smtp.port` address the SMTP server. `global.smtp.credentialsSecret` names the Secret with the user name and password.

```
global:
  smtp:
    enabled: true
    address: <SMTP_HOST>
    port: 587
    credentialsSecret: <SMTP_CREDENTIALS_SECRET>
```

Advanced configuration

The tables below list every chart value referenced on Quick start and Configuration, one table per key group. Value names are those of the chart's `values.yaml`. Default column values are placeholders until the chart ships.

Hosts and ingress

Key	Type	Default	Meaning
`global.hosts.domain`	string	—	Base domain for the instance and its Ingress objects.
`global.ingress.class`	string	—	Ingress class name the chart's Ingress objects use.

TLS

Key	Type	Default	Meaning
`global.ingress.tls.secretName`	string	—	Name of the Secret with the TLS certificate and key for `global.hosts.domain`.

PostgreSQL

Key	Type	Default	Meaning
`global.psql.host`	string	—	Address of the external PostgreSQL server.
`global.psql.port`	integer	`<VALUE>`	Port of the external PostgreSQL server.
`global.psql.database`	string	—	Name of the database Deckhouse Code uses.
`global.psql.username`	string	—	PostgreSQL role Deckhouse Code connects as.
`global.psql.password.secretName`	string	—	Name of the Secret with the PostgreSQL password.
`global.psql.password.key`	string	`password`	Key inside that Secret that holds the password.

Redis

Key	Type	Default	Meaning
`global.redis.host`	string	—	Address of the external Redis server.
`global.redis.port`	integer	`<VALUE>`	Port of the external Redis server.
`global.redis.password.secretName`	string	—	Name of the Secret with the Redis password, when the server requires one.
`global.redis.password.key`	string	`password`	Key inside that Secret that holds the password.

Object storage

Key	Type	Default	Meaning
`global.appConfig.object_store.enabled`	boolean	`<VALUE>`	Whether Deckhouse Code stores CI/CD artifacts, uploads and backup archives in S3-compatible object storage.
`global.appConfig.object_store.connection.secretName`	string	—	Name of the Secret with the object storage endpoint, region and access keys.

Git storage (Gitaly persistence)

Key	Type	Default	Meaning
`gitaly.persistence.size`	string	`<VALUE>`	Size of the persistent volume claim each Gitaly pod mounts for Git repository storage.
`gitaly.persistence.storageClass`	string	—	StorageClass the Gitaly persistent volume claims are provisioned from.

Container registry

Key	Type	Default	Meaning
`registry.enabled`	boolean	`<VALUE>`	Whether the chart deploys the container registry component.
`registry.storage.bucket`	string	—	Object storage bucket the container registry stores image layers in.

Pages

Key	Type	Default	Meaning
`pages.enabled`	boolean	`<VALUE>`	Whether the chart deploys the Pages component.
`pages.storage.bucket`	string	—	Object storage bucket Pages content is stored in.

SMTP

Key	Type	Default	Meaning
`global.smtp.enabled`	boolean	`<VALUE>`	Whether outgoing notification email is sent through SMTP.
`global.smtp.address`	string	—	Address of the SMTP server.
`global.smtp.port`	integer	`<VALUE>`	Port of the SMTP server.
`global.smtp.credentialsSecret`	string	—	Name of the Secret with the SMTP user name and password.

Troubleshooting

Each entry below describes one symptom, its cause and the action that resolves it.

Pods stay in Pending

`d8 k -n code get pods` shows one or more pods in the `Pending` state.

Cause. No StorageClass satisfies `gitaly.persistence.storageClass` (or the cluster has no default StorageClass), or no node has enough CPU and memory for the pod's resource requests.

Action. Read the pod's events:

```
d8 k -n code describe pod <POD_NAME>
```

A `FailedScheduling` event naming a persistent volume claim points to the StorageClass; check it with `d8 k get storageclass`. A `FailedScheduling` event naming CPU or memory points to node resources.

Ingress has no address

`d8 k -n code get ingress` shows no value in the `ADDRESS` column.

Cause. `global.ingress.class` does not match an ingress controller installed in the cluster, or the controller's own Service has no external address yet.

Action. Confirm the ingress controller is running and that its class name matches `global.ingress.class`. If the controller's Service is of type `LoadBalancer`, check that the surrounding infrastructure assigned it an address.

TLS certificate is not issued

The browser reports the certificate as invalid or self-signed when opening `global.hosts.domain`.

Cause. The Secret named in `global.ingress.tls.secretName` does not exist in the `code` namespace, or it does not hold a certificate valid for `global.hosts.domain`.

Action. Check the Secret and the certificate it holds:

```
d8 k -n code get secret <TLS_SECRET_NAME> -o yaml
```

Confirm the certificate's subject and Subject Alternative Names cover `global.hosts.domain`.

Database connection refused

Pods restart and their logs show `PG::ConnectionBad: could not connect to server: Connection refused`.

Cause. `global.psql.host` or `global.psql.port` does not reach the external PostgreSQL server, or network policies block the connection from the release's namespace.

Action. Test connectivity to the server from inside the cluster, and confirm the database in `global.psql.database` and the role in `global.psql.username` exist on it.

Object storage credentials rejected

Uploads and artifacts fail, and pod logs show an access-denied response from the object storage endpoint.

Cause. The Secret named in `global.appConfig.object_store.connection.secretName` holds an outdated endpoint, region or access key, or the bucket does not exist.

Action. Check the Secret's content against the object storage provider's console, and confirm the bucket exists and the credentials have write access to it.

Migrations job failing

The migrations job's pod exits with a non-zero status, and pods that depend on the database schema stay `Pending` or restart in a loop.

Cause. PostgreSQL is not reachable yet, the credentials in `global.psql.password.secretName` are wrong, or the release skipped a Deckhouse Code version that a later migration depends on.

Action. Read the job's log:

```
d8 k -n code logs job/<MIGRATIONS_JOB_NAME>
```

Resolve the database connection first, then check Upgrade for the version order when skipping a version is the cause.

Helm upgrade times out

`helm upgrade` exits with `Error: UPGRADE FAILED: timed out waiting for the condition`.

Cause. A pod or the migrations job didn't reach a ready or complete state within Helm's wait timeout — usually one of the causes above.

Action. Inspect the release's pods and jobs:

```
d8 k -n code get pods,jobs
```

Resolve the underlying cause, then run `helm upgrade` again, adding `--timeout` when the release legitimately needs more time.

Deckhouse Kubernetes Platform module

The Deckhouse Kubernetes Platform module delivers Deckhouse Code through the `code-operator` component, which installs and maintains the Deckhouse Code instance. Every setting of the instance is a field of the CodeInstance resource; for an empty field the operator takes the platform's global setting: the public domain template, the IngressClass or the HTTPS mode. The module's own ModuleConfig carries only the `logLevel` setting.

The web interface of the module delivery is available at the `code` subdomain of the platform's public domain template, for example `https://code.example.com`. Setting `spec.network.web.hostname` in CodeInstance uses that address instead.

Requirements

The module requires:

- Deckhouse Kubernetes Platform 1.68 or later;
- Kubernetes 1.29 or later;
- the `cert-manager` module enabled;
- PostgreSQL 17 or later;
- Redis 7.0 or later;
- S3 storage with the `YCloud` or `Generic` provider;
- a StorageClass for the Git repositories, named in the `gitData.storageClass` field;
- an IngressClass for the traffic to the web interface.

The cluster resources for the instance size are given in Minimum requirements in the module documentation, and Getting started there describes the setup of PostgreSQL, Redis and the S3 storage.

Installation

Enabling the module installs `code-operator`; once PostgreSQL, Redis and the S3 storage are ready, applying a CodeInstance resource installs Deckhouse Code with the given parameters. Getting started in the module documentation covers the installation steps, from enabling the module to the first sign-in.

> Disabling the module deletes the Code configuration, the Git data and every Secret and ConfigMap owned by the operator, including the encryption keys of the data stored in PostgreSQL. Create a backup before disabling the module.

Configuration

Every setting of the module delivery is a field of the CodeInstance resource. Custom Resources in the module documentation lists every field.

Operations

Backup, scaling and network configuration of the module delivery are CodeInstance fields, described in the module documentation: Maintenance, Scaling and Network. The module is updated by the platform; the backup before an update is described on the Upgrade page. Moving an instance between the Linux package or Omnibus Docker and the module is described in Migration.

Administration

A module administrator uses the product documentation Overview and the rest of the Administration section for instance-level tasks: users and access, authentication, security, monitoring, backup and upgrade.

Administration

Overview

The Administration section covers instance-level configuration of Deckhouse Code: access control, authentication, security, monitoring, maintenance mode, backup, and upgrade.

Pages that differ by installation type

Deckhouse Code is delivered as a Linux package, Omnibus Docker, a Helm Chart, or a Deckhouse Kubernetes Platform module. In Monitoring and limits the built-in monitoring of each installation type is a tab. Backup and restore and Upgrade are single pages: the part shared by every installation type comes first, and each step whose command differs is a tab per type.

Other pages

- Users and access — restricting who can create groups and projects, and instance-wide access controls.
- Authentication — configuring sign-in through OmniAuth providers and LDAP.
- Security settings — instance-wide security settings, including sign-up restrictions and administrator two-factor authentication.
- Security credentials — reviewing and revoking tokens, SSH keys, and GPG keys issued across the instance.
- Service accounts — creating service accounts for CI/CD pipelines and integrations.
- Audit events — viewing the audit events recorded for the instance.
- Maintenance mode — reducing write operations during maintenance tasks.
- Advanced search — operating full-text search indexing after OpenSearch is connected.

Cluster-level operations of the Deckhouse Kubernetes Platform module — sizing, network configuration, migration from the Linux package or Omnibus Docker to the module and back — are described in the module documentation.

Users and access

Restricting group and project creation

1. Go to “Admin” → “Settings” → “General”.
2. Expand “Visibility and access controls” and set “Default minimum role required to create projects” to the lowest role that may create projects on the instance.
3. Expand “Account and limit”. In the “User restrictions” group, clear the “Allow new users to create top-level groups” and “Allow users with up to Guest role to create groups and personal projects” checkboxes.

4. Click “Save changes” in every section you have changed.

The “Allow new users to create top-level groups” checkbox sets the default for accounts created after the change. For an account that already exists, the permission is set in “Admin” → “Overview” → “Users”: open the user, click “Edit” and set the “Can create top-level groups” value.

Enabled Git access protocols

The instance accepts Git operations over SSH and over HTTP(S). To leave one protocol:

1. Go to “Admin” → “Settings” → “General”.
2. Expand “Visibility and access controls”.
3. In the “Enabled Git access protocols” list, select “Only SSH” or “Only HTTP(S)”.
4. Click “Save changes”.

Git operations over the protocol that is left out are refused.

Password authentication for Git over HTTPS

1. Go to “Admin” → “Settings” → “General”.
2. Expand “Sign-in restrictions”.
3. Clear the “Allow password authentication for Git over HTTP(S)” checkbox.
4. Click “Save changes”.

A Git client then authenticates over HTTPS with a personal access token instead of the account password. When LDAP is configured, the LDAP password is accepted as well.

Two-factor authentication for the whole instance

1. Go to “Admin” → “Settings” → “General”.
2. Expand “Sign-in restrictions”.
3. Select “Enforce two-factor authentication” to require the second factor from every user of the instance, or “Enforce two-factor authentication for administrators” to require it from administrators only.
4. In the “Two-factor grace period” field, enter the number of hours during which a user may postpone the setup. The value `0` requires the setup at the next sign-in.
5. Click “Save changes”.

A user who has not set the second factor up is asked to do so on the next sign-in. The methods a user can choose from are described on the Signing in page.

Push rules at the instance level

Push rules set in “Admin” → “Settings” → “Repository” → “Push rules” apply to every project of the instance: a group or a project receives them as its default values and changes them only while the rule allows an override. The rules themselves and the override are described on the Push rules page.

Authentication

Overview

Deckhouse Code signs users in through external authentication providers (OmniAuth) and takes group memberships and access rights from an LDAP directory. The provider behaviour, the synchronization rules and the decisions that grant or withdraw access hold for every installation type; the place where the settings are written differs:

- Linux package — the `/etc/gitlab/gitlab.rb` file;
- Omnibus Docker — the `GITLAB_OMNIBUS_CONFIG` variable or `/etc/gitlab/gitlab.rb` on the configuration volume;
- Helm Chart — the chart values;
- Deckhouse Kubernetes Platform module — the `CodeInstance` resource.

The keys and manifests for every type are on Configuration.

The settings below are named as the product reads them. The page of your installation type gives the key each one is written under.

Sign-in restrictions and password authentication

Sign-in of local accounts is controlled in “Admin” → “Settings” → “General” → “Sign-in restrictions”. The “Allow password authentication for the web interface” checkbox shows whether an account with a password signs in to the web interface alongside the external providers; in the interface the checkbox is read-only.

To turn password authentication for the web interface on, open the Rails console:

Linux package

```
sudo gitlab-rails console
```

Omnibus Docker

```
docker exec -it code gitlab-rails console
```

Helm Chart

```
d8 k -n code exec -it deploy/code-toolbox -- gitlab-rails console -e production
```

Deckhouse Kubernetes Platform module

```
d8 k -n d8-code exec -it -c toolbox deploy/toolbox -- gitlab-rails console -e
production
```

Run the following command in it:

```
Gitlab::CurrentSettings.update!(password_authentication_enabled_for_web: true)
```

Password authentication for Git over HTTPS and two-factor authentication for the whole instance are set in the same section of the admin area, see the Users and access page.

OmniAuth configuration

The general OmniAuth parameters apply to every provider; OpenID Connect and SAML add parameters of their own.

Supported providers

The `name` parameter of a provider entry accepts one of the following values:

- `openid_connect` — OpenID Connect (described below);
- `saml` — SAML (described below);
- `oauth2_generic` — any OAuth 2.0 provider;
- `jwt` — JWT authentication;
- `github` — GitHub;
- `gitlab` — GitLab.com;
- `google_oauth2` — Google;
- `azure_activedirectory_v2` — Microsoft Entra ID (Azure AD);
- `atlassian_oauth2` — Atlassian;
- `crowd` — Atlassian Crowd;
- `auth0` — Auth0;
- `alicloud` — AliCloud;
- `salesforce` — Salesforce;
- `shibboleth` — Shibboleth.

Signing in through LDAP is configured separately, in the LDAP server section (see LDAP synchronization).

General OmniAuth parameters

The following parameters hold for every provider:

- `enabled`: Allows signing in through external providers. Default — `true`.
- `providers`: The list of providers users are allowed to sign in through. Default — `[]`.
- `allow_single_sign_on`: The list of providers for which an account is created automatically on the first sign-in (for example, `['openid_connect']`). Also accepts `true` (all providers) and `false`. If automatic creation is disabled, the user must first get a Deckhouse Code account and then link it to the provider. Default — `false`.
- `block_auto_created_users`: If `true`, automatically created accounts are blocked pending administrator approval. Default — `true`.
- `auto_link_ldap_user`: Links the account to an LDAP account on the first sign-in (see Linking OIDC accounts to LDAP). Default — `false`.
- `auto_link_user`: Links a sign-in through a provider to an existing Deckhouse Code account by email address. Accepts a list of providers or the `true` and `false` values. Default — `false`.
- `auto_sign_in_with_provider`: The name of the provider whose sign-in page the user is redirected to automatically, bypassing the Deckhouse Code sign-in page. Default — `false`.
- `external_providers`: The list of providers whose accounts are created as external. Default — `[]`.
- `allow_bypass_two_factor`: The list of providers that do not require two-factor authentication on sign-in. Also accepts the `true` and `false` values. Default — `false`.
- `sync_profile_from_provider`: The list of providers whose data updates the user profile on every sign-in. Also accepts the `true` and `false` values. Default — `false`.
- `sync_profile_attributes`: The list of profile attributes to update during synchronization: `name`, `email`, `location`. The synchronized attributes become read-only. Default — `['email']`.

OpenID Connect (OIDC)

Providers are listed in the `providers` parameter. The following parameters are available for an OIDC provider:

- `name`: The provider type. For OIDC, it is always `'openid_connect'`.
- `label`: The sign-in button label. Default — `'Openid Connect'`.
- `icon`: The address of the image shown on the sign-in button.
- `args`: The provider connection parameters:
  - `name`: The OmniAuth strategy name, matching the value of the provider `name` parameter;
  - `scope`: The list of requested scopes, for example `['openid', 'profile', 'email']`;
  - `response_type`: The OAuth 2.0 response type. For the Authorization Code flow, it is `'code'`;
  - `issuer`: The OIDC provider address;

    - `discovery`: If `true`, the provider settings are retrieved automatically from `<issuer>/.well-known/openid-configuration`;
    - `client_auth_method`: The client authentication method at the token endpoint: `'basic'` or `'query'`;
    - `uid_field`: The field from the user data used as the account `uid` (for example, `preferred_username`). If the parameter is not set or the field is missing, the `sub` field is used;
    - `send_scope_to_token_endpoint`: Whether to pass the `scope` parameter in requests to the token endpoint. Set it to `false` if the provider does not accept this parameter. Default — `true`;
    - `pkce`: Enables Proof Key for Code Exchange (PKCE);
    - `client_options`:
        - `identifier`: The identifier of the client registered with the provider;
        - `secret`: The client secret;
        - `redirect_uri`: The address of your Deckhouse Code installation with the `/users/auth/openid_connect/callback` path. The same address must be specified in the client settings on the provider side.

Additionally, Deckhouse Code supports the following parameters. They are set at the top level of the provider entry, next to the `name` parameter:

- `allowed_groups`: A list of groups whose users are allowed to log in. Users not in these groups will be denied access. Default — `null` (all groups are allowed).
- `admin_groups`: A list of groups whose users are granted administrative privileges. Default — `null` (no groups are granted admin rights).
- `auditor_groups`: A list of groups whose users are granted the auditor role: read-only access to all groups and projects, without access to the admin area. Default — `null` (no groups are granted the auditor role).
- `groups_attribute`: The name of the attribute used to extract user group information. Default — `'groups'`.

> The `admin_groups` and `auditor_groups` parameters are taken into account only if the `allowed_groups` parameter is set. If a user belongs to both `admin_groups` and `auditor_groups`, administrative privileges are granted.

Provider entry examples for every installation type: OpenID Connect provider and SAML provider.

SAML

The same parameters are available for SAML providers:

- `allowed_groups`: A list of groups whose members are allowed to log in. Default — `null` (all groups are allowed).
- `admin_groups`: Groups whose members are granted administrative privileges. Default — `null` (no groups are granted admin rights).
- `auditor_groups`: Groups whose members are granted the auditor role: read-only access to all groups and projects. Default — `null` (no groups are granted the auditor role).
- `groups_attribute`: The name of the attribute that contains group information. Default — `'Groups'`.

> If a user belongs to `admin_groups` but is not listed in `allowed_groups`, access will be denied. In this case, administrative privileges will not be granted either.

LDAP synchronization

Deckhouse Code supports synchronization of users, groups, and access rights with an LDAP server. Synchronization runs automatically every hour, or at a custom interval set by the schedule of the `ldap_sync_worker` job.

LDAP server entry examples for every installation type: LDAP servers and Synchronization schedule.

LDAP server-side limitations

During synchronization, LDAP queries are executed for all users and groups defined in the configuration. Pagination is used automatically if necessary. If the LDAP server enforces limits on the number of returned entries, this may cause synchronization errors or lead to user access rights being removed.

Multiple LDAP servers

The LDAP section can describe several servers. The key of an entry is the server name; the provider name is derived from it as `ldap<key>` in lowercase. For example, the `main` key corresponds to the `ldapmain` provider. This name is stored in the account identity and appears in the `server` field of the synchronization log entries.

Each server has its own connection parameters. The synchronization parameters — the `group_sync` section with everything in it, including `role_mapping` — are taken into account only for the server under the `main` key; on the other servers the section is ignored (see Synchronization scope).

Signing in works through any of the described servers: the sign-in page and the admin area sign-in page show a separate tab for each server. No extra setting is needed to enable this.

Which servers are used depends on the scenario:

Scenario	Servers used
Signing in through the LDAP form in the web interface	All servers, one tab each
Looking up the account when linking to OIDC (`auto_link_ldap_user`)	All servers, one by one until the first match
Synchronization of users, groups, and access rights	The `ldapmain` server only
Periodic access re-check	`ldapmain` , if the account has its identity; otherwise, the directories of the account's own identities

Synchronization scope

Synchronization of users, groups, and access rights covers a single server — the one whose provider name is `ldapmain` , that is, the server under the `main` key. The other servers remain a source of sign-in, but not a source of groups and rights. The limitation applies both to the scheduled synchronization task and to the synchronization that runs when a user signs in.

> ⚠ The `ldapmain` name is fixed and cannot be configured. If there is no server under the `main` key in the section, synchronization does not run for any of the described servers.

As a result:

- a user who exists only in a non-main directory can sign in to Deckhouse Code but receives no groups or rights from LDAP — they have to be assigned manually;
- for a user with two identities (the same primary email address in both directories), groups and rights always come from `ldapmain` , no matter which server they signed in through;
- the `group_sync` section of non-main servers is not read, so an error in their `group_sync.filter` parameter does not prevent the application from starting. An error in the filter of the `main` server is still detected at startup and keeps the application down.

When synchronization reaches a non-main server, it skips the server and writes a single `INFO` level entry about it, with the name of the skipped server in the `server` field:

```
Server is not synchronized: users, groups and permissions come from 'ldapmain' only
```

User identification with multiple servers

A user is looked up first by account identity, then by primary email address:

- if the primary address is the same in both directories, signing in through the second server adds a second LDAP identity to the existing account; no new account is created;

- if the addresses differ, two separate accounts are created.

The list of account identities is available in the admin area, on the `/admin/users/<username>/identities` page.

Access decision with multiple identities

Deckhouse Code periodically, at most once an hour, re-checks the access of an LDAP user: it queries the directory and blocks the account if the entry is not found there, or unblocks it if the entry is found again. If the account has several LDAP identities, the directory for the check is chosen as follows:

- if the `ldapmain` identity is present, only it decides, and the other directories are not queried at all. If the user exists in `ldapmain` and is not disabled there, access is granted; if the user is missing or disabled there, access is denied regardless of what happens in the other directories;
- if the `ldapmain` identity is absent, meaning the user exists only in non-main directories, the user is checked against their own identities, and access is kept as long as the entry is found in at least one of the corresponding directories.

“The entry is found” means that it exists in the directory and is not disabled through Active Directory. The latter is checked only for directories configured as AD.

The rule is aligned with the synchronization scope: synchronization blocks users based on the `ldapmain` data, and the access re-check blocks exactly the same users. Because of that, signing in through another directory does not lift a block set by synchronization.

> ⚠ When moving a user from the `ldapmain` directory to another directory, remove the stale `ldapmain` identity from their account on the `/admin/users/<username>/identities` page. As long as that identity is in place, the user stays blocked even if they exist in another directory. The identity is not removed automatically.

Removing or disabling a user in a non-main directory does not revoke access on its own while the user remains in `ldapmain`. Access has to be revoked in the `ldapmain` directory.

Groups and access rights

LDAP groups are mapped to GitLab groups. You can assign roles to users based on group names.

Required parameters:

- `group_sync.base`: The DN from which LDAP group search starts.

Optional parameters:

- `group_sync.create_groups`: If `true`, groups will be created in Deckhouse Code.
- `group_sync.filter`: LDAP filter used to find groups.
- `group_sync.scope`: Scope of group search (0 — Base, 1 — SingleLevel, 2 — WholeSubtree).
- `group_sync.prefix`: Defines which attribute to use for determining the parent group name. If missing, the default value is used.

- `group_sync.top_level_group`: The top-level group to which all synchronized groups will be added.
- `group_sync.name_mask`: Regular expression used to extract the group name from the CN (Common Name) attribute.
- `group_sync.owner`: Name of the user to be assigned as group owner (default is `root`).

role_mapping section

Assigns roles to users based on group names (`cn`):

- `role_mapping.by_name`: A regular expression; if the group name matches, the corresponding role is assigned to the user.
- `role_mapping.gitlab_role`: The name of a role available on the instance.

The list of available roles comes from the standard role catalog — the same one used to decide which roles can be granted to a group or project member. The catalog is read anew on every synchronization run, so a role disabled at the instance level is not in it and cannot be used in `role_mapping`.

Currently the catalog provides seven names:

`gitlab_role`	Role	Access level
`guest`	Guest	10
`planner`	Planner	15
`reporter`	Reporter	20
`security_manager`	Security Manager	25
`developer`	Developer	30
`maintainer`	Maintainer	40
`owner`	Owner	50

The role name is written exactly as shown in the `gitlab_role` column: lowercase, with words separated by underscores. Case and spaces are not normalized, so `Security Manager`, for example, counts as an unrecognized name.

The `security_manager` role is present in the catalog only if the role is enabled on the instance (the `GITLAB_SECURITY_MANAGER_ROLE` environment variable, enabled by default). If the role is disabled, its name counts as unrecognized.

If several rules match an LDAP group name, the numerically highest access level is assigned. For example, if a group matches both a rule with the `security_manager` role (level 25) and a rule with the `developer` role (level 30), its members get the Developer role.

Role name validation

Role names are validated once, at the start of membership distribution — including the rules that no LDAP group matched in that run. This also catches a dormant typo that would otherwise show up only once a matching group appears.

The validation does not interrupt membership distribution:

- LDAP groups whose matching rules are all recognized are processed in full and receive memberships;
- an unrecognized rule does not take part in the access level calculation;
- if only unrecognized rules match an LDAP group, no access level is determined for it, and the group is skipped entirely in that run: its current members are neither recalculated nor removed.

The run itself, however, ends with an error — after the groups, the users, and the valid memberships have been written. As a result:

- the run counts as abnormally finished on the metrics page of the synchronization task, and the successful synchronization mark is not updated (see Manual synchronization run);
- an `ERROR` level entry is written to the logs, listing both the unrecognized names and the full set of valid ones:

```
Unknown gitlab_role in group_sync.role_mapping: 'security_manger'. Available
roles: guest, planner, reporter, security_manager, developer, maintainer, owner
```

Memberships are also distributed when a user signs in through an LDAP provider. On that path the same error is only written to the logs: the sign-in works as usual, and the remaining memberships are assigned.

Group membership resolution

> ⚠ LDAP Sync does not support transitivity for nested groups. See Nested groups and transitivity section for details and workarounds.

Deckhouse Code supports the following attributes to determine group membership (all values are arrays of DNs):

- `member`
- `uniquemember`
- `memberof`
- `memberuid`
- `submember`

User synchronization

During synchronization, usernames, email addresses, and account lock status are updated.

Optional parameters:

- `sync_name` — if `true`, the username will be updated based on LDAP data.

Blocking users based on LDAP data

If a user is removed from LDAP, the next scheduled synchronization blocks their account. If the user is restored in LDAP, the next synchronization unblocks the account automatically.

Blocking always happens and requires no additional settings: LDAP remains the source of truth for the account status. A blocked user will be denied access even if they sign in through an OIDC provider where the account is still active.

If your directory does not delete users but blocks them using an attribute, exclude blocked users from LDAP results using the `user_filter` parameter of the server. Specify a single filter matching your directory:

Directory	`user_filter` value
OpenLDAP with ppolicy	`(!(pwdAccountLockedTime=*))`
389-DS	`(!(nsAccountLock=TRUE))`
Active Directory	`(!(userAccountControl:1.2.840.113556.1.4.803:=2))`
A custom attribute	`(!(employeeType=blocked))`

Linking OIDC accounts to LDAP

If users sign in through an OIDC provider (for example, Keycloak) while permissions are granted based on LDAP groups, enable automatic linking of the OIDC account to the LDAP account with the `auto_link_ldap_user` parameter.

How the LDAP account is found

On the first sign-in through OIDC, Deckhouse Code looks the user up in LDAP. The search uses two values from the OIDC provider data:

- `uid`: The value of the field specified in the provider `uid_field` parameter (for example, `preferred_username`).
- email: The user email address.

The configured LDAP servers are queried one by one. On each server, up to four search attempts are made, until the first match:

Value being searched for	LDAP attribute searched
`uid`	The attribute specified in the LDAP server `uid` parameter (for example, `cn`)
`uid`	Mail attributes: `mail`, `email`, `userPrincipalName`
email	The same mail attributes
`uid`	DN — if the `uid` value is a DN itself

If the user is found, an LDAP identity with the discovered DN is added to their account. If none of the attempts succeeds, the account is created without an LDAP link.

This means linking works only if the `uid` or email from the OIDC provider matches the value of the corresponding attribute in LDAP.

First and subsequent sign-ins

The first successful sign-in links the account to LDAP. On subsequent sign-ins:

- LDAP is not queried — the previously established link is used;
- access and administrative privileges are re-evaluated based on the groups provided by the OIDC provider (the `allowed_groups` and `admin_groups` parameters). If the user is removed from an allowed group, the account is blocked;
- group and project memberships, as well as roles in them, do not change on an OIDC sign-in — they are updated by the scheduled background LDAP synchronization.

As a result, right after the first sign-in a user can log in but has no group or project memberships yet: they appear after the next synchronization. To avoid waiting for it, run the synchronization manually (see Manual synchronization run).

> Group and membership synchronization on sign-in runs only when a user signs in through an LDAP provider (a provider whose name starts with `ldap`). An OIDC sign-in does not trigger it, even if the account is already linked to LDAP.

LDAP as the source of truth

To allow users found in LDAP to sign in immediately and send everyone else to administrator approval, combine two settings:

- `block_auto_created_users` at the OmniAuth level set to `true`: a user who is not found in LDAP gets an account that is blocked pending administrator approval;
- `block_auto_created_users` on the `main` LDAP server set to `false`: a user who is found in LDAP signs in immediately.

Linking specifics

- If a user is blocked by renaming their `cn` in the directory, on the first sign-in they may still be found by email and linked to the LDAP account. To block users, remove them from the directory or use the `user_filter` parameter (see Blocking users based on LDAP data).
- If a user was not linked to LDAP and was blocked by the `Ldap::BlockNonLdapUsersWorker` job, automatic unblocking will not work. Such a user must be unblocked manually and linked to an LDAP account.

Troubleshooting synchronization issues

If a previous sync job was not completed successfully, Redis may retain a lock preventing the next job from starting (the default `concurrency` is set to 1).

To remove the lock:

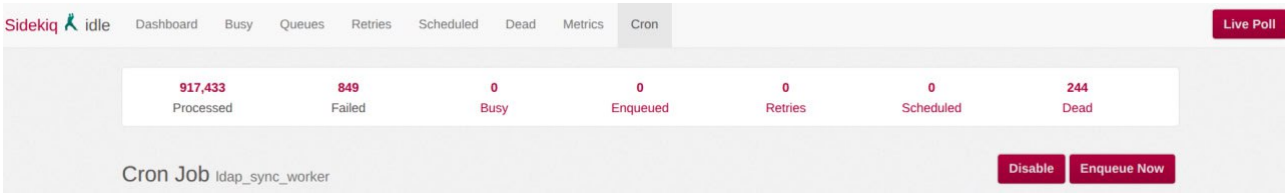
1. Connect to Redis using the databases specified in `config/redis.shared_state.yml` and `config/redis.queues.yml`.
2. Delete the key `sidekiq:concurrency_limit:throttled_jobs:{ldap/sync_worker}` using the following commands:

```
keys *ldap*
del "sidekiq:concurrency_limit:throttled_jobs:{ldap/sync_worker}"
```

Manual synchronization run

To synchronize groups immediately after they are changed on the LDAP side, follow these steps:

1. Go to the LDAP synchronization worker page `/admin/sidekiq/cron/namespaces/default/jobs/ldap_sync_worker`.
2. In the upper-right corner, click the “Enqueue Now” button and confirm in the dialog.



To see how the triggered synchronization finished, open the metrics page for the LDAP synchronization task: `/admin/sidekiq/metrics?substr=SyncWorker&period=8h`. The chart displays call statistics; the table below shows the number of successful and failed LDAP synchronization runs.

To view the full synchronization logs:

1. On the worker page `/admin/sidekiq/cron/namespaces/default/jobs/ldap_sync_worker`, find the run events table named “History”. The first row corresponds to the most recent run. Copy the value in the JID (Job ID) column — you will need it to search the logs.

2. Collect the Sidekiq log entries of that run, substituting the copied JID:

Linux package

```
sudo grep '<JID>' /var/log/gitlab/sidekiq/current
```

Omnibus Docker

```
docker exec code grep '<JID>' /var/log/gitlab/sidekiq/current
```

Helm Chart

```
d8 k -n code logs -l app.kubernetes.io/component=sidekiq | jq 'select(.jid=="
<JID>")'
```

Deckhouse Kubernetes Platform module

```
d8 k -n d8-code -l app.kubernetes.io/component=sidekiq get pod -o NAME
d8 k -n d8-code logs <POD_NAME> -c sidekiq | jq 'select(.jid=="<JID>")'
```

Old logs are removed by rotation over time, so they may become unavailable. If needed, rerun the synchronization and collect the latest logs.

LDAP Sync behavior

Synchronization algorithm

LDAP Sync uses a flat, non-recursive synchronization algorithm:

1. Group retrieval. An LDAP query retrieves all groups based on the configured `base`, `filter`, and `scope` parameters.
2. Member extraction. For each discovered group, LDAP Sync reads the membership attributes: `member`, `uniquemember`, `memberof`, `memberuid`, `submember`.
3. User matching. Each DN from the membership attributes is matched against `Identity.extern_uid` in the database.
4. Ignoring unknown DNs. If a DN does not match a known user, it is skipped. For example, this may be the DN of a nested group.

Cyclic group dependencies

Cyclic dependencies in the LDAP group hierarchy do not cause synchronization errors. LDAP Sync processes groups in a flat way, according to the configured filter, and does not attempt to reconstruct the LDAP tree structure.

Because recursive traversal of nested groups is not performed, cycles do not affect the synchronization result.

Nested groups and transitivity

LDAP Sync does not support transitivity for nested groups.

During synchronization, LDAP Sync processes only the direct values of a group's membership attributes and does not recursively traverse nested groups.

If one LDAP group contains another group as a member, users from the nested group are not automatically added to the parent group.

For synchronization to work correctly, all required user DNs must be present directly in the group's membership attributes.

If nested groups must be taken into account, this must be implemented on the LDAP server side. For example, the `submember` attribute can be populated with the full list of transitive members.

This approach simplifies synchronization and avoids issues related to recursive group processing.

Creating a local account when LDAP synchronization is enabled

Local accounts can still be created and used even when LDAP synchronization is enabled.

Such users sign in through the web interface while password authentication for the web interface is on (see Sign-in restrictions and password authentication).

Configuration

What each setting does is described in Overview. This page gives, for every installation type, the place the setting is written in and the shape of its value.

Where the configuration lives

Linux package. The settings are keys of `/etc/gitlab/gitlab.rb`.

Omnibus Docker. The container reads the same `gitlab.rb` keys. They reach it from the `GITLAB_OMNIBUS_CONFIG` environment variable, which is evaluated on every start, and from `/etc/gitlab/gitlab.rb` on the configuration volume, which is read after the variable. A key written in the file overrides the same key in the variable. The variable is set when the container is created:

```
docker run -d --name code \
  --shm-size 256m \
  -p 80:80 -p 443:443 -p 22:22 \
  -e GITLAB_OMNIBUS_CONFIG="external_url 'https://<HOSTNAME>';
gitlab_rails['omniauth_enabled'] = true; gitlab_rails['omniauth_allow_single_sign_on']
= ['openid_connect']; gitlab_rails['omniauth_auto_link_ldap_user'] = true" \
  -v /srv/code/config:/etc/gitlab \
  -v /srv/code/logs:/var/log/gitlab \
  -v /srv/code/data:/var/opt/gitlab \
  <REGISTRY>/<FLAVOR>:<VERSION>
```

The variable is fixed when the container is created, so changing it means removing the container and creating a new one with the same volumes. The provider entries and the LDAP servers are multi-line

values, and the configuration volume keeps them across container replacements. With the volumes from the quick start, `/etc/gitlab/gitlab.rb` inside the container is `/srv/code/config/gitlab.rb` on the host, and it is edited on the host:

```
sudo vi /srv/code/config/gitlab.rb
```

Helm Chart. The settings are chart values, written in the `values.yaml` file of the release.

Deckhouse Kubernetes Platform module. The settings are parameters of the `CodeInstance` resource, described on the OmniAuth and LDAP setup and Custom Resources pages of the module documentation.

OmniAuth general parameters

The parameters and their effect are described in General OmniAuth parameters.

Linux package

Each general OmniAuth parameter has a `gitlab.rb` key of its own:

Parameter	`gitlab.rb` key
`enabled`	`gitlab_rails['omniauth_enabled']`
`providers`	`gitlab_rails['omniauth_providers']`
`allow_single_sign_on`	`gitlab_rails['omniauth_allow_single_sign_on']`
`block_auto_created_users`	`gitlab_rails['omniauth_block_auto_created_users']`
`auto_link_ldap_user`	`gitlab_rails['omniauth_auto_link_ldap_user']`
`auto_link_user`	`gitlab_rails['omniauth_auto_link_user']`
`auto_sign_in_with_provider`	`gitlab_rails['omniauth_auto_sign_in_with_provider']`
`external_providers`	`gitlab_rails['omniauth_external_providers']`
`allow_bypass_two_factor`	`gitlab_rails['omniauth_allow_bypass_two_factor']`
`sync_profile_from_provider`	`gitlab_rails['omniauth_sync_profile_from_provider']`
`sync_profile_attributes`	`gitlab_rails['omniauth_sync_profile_attributes']`

```
gitlab_rails['omniauth_enabled'] = true
gitlab_rails['omniauth_allow_single_sign_on'] = ['openid_connect']
gitlab_rails['omniauth_block_auto_created_users'] = true
gitlab_rails['omniauth_auto_link_ldap_user'] = true
gitlab_rails['omniauth_sync_profile_attributes'] = ['email']
```

Omnibus Docker

The keys and their values are the ones of the Linux package tab. A single-line key fits into `GITLAB_OMNIBUS_CONFIG`, separated from the next one by `;`, as in the `docker run` command above; a key whose value spans several lines is written in `/srv/code/config/gitlab.rb`.

Helm Chart

The general parameters are values under `global.appConfig.omniauth`:

```
global:
  appConfig:
    omniauth:
      enabled: true
      allowSingleSignOn: ['openid_connect']
      blockAutoCreatedUsers: true
      autoLinkLdapUser: true
      syncProfileAttributes: ['email']
```

Deckhouse Kubernetes Platform module

The general parameters are set in the `spec.appConfig.omniauth` section, and the providers are listed in its `providers` parameter:

```
enabled: true
allowSingleSignOn: ['openid_connect']
blockAutoCreatedUsers: true
autoLinkLdapUser: true
syncProfileAttributes: ['email']
```

OpenID Connect provider

The connection parameters of the provider and their effect are described in OpenID Connect (OIDC).

Linux package

A provider is an entry of the `gitlab_rails['omniauth_providers']` array. The `allowed_groups`, `admin_groups`, `auditor_groups` and `groups_attribute` keys sit next to `name`, at the top level of the entry; the connection parameters sit inside `args`:

```
gitlab_rails['omniauth_providers'] = [
  {
    name: 'openid_connect',   # Do not change this value.
    label: 'Keycloak',        # Sign-in button label.
    allowed_groups: ['gitlab'],
    admin_groups: ['admin'],
    auditor_groups: ['audit'],
    groups_attribute: 'gitlab_group',
    args: {
      name: 'openid_connect',
      scope: ['openid', 'profile', 'email'],
      response_type: 'code',
      issuer: 'https://keycloak.example.com/realms/example',
      discovery: true,
      client_auth_method: 'query',
      uid_field: 'preferred_username',
      send_scope_to_token_endpoint: false,
      pkce: true,
      client_options: {
        identifier: '<client_id>',
        secret: '<client_secret>',
        redirect_uri: 'https://code.example.com/users/auth/openid_connect/
callback'
      }
    }
  }
]
```

Omnibus Docker

The entry is the one of the Linux package tab. It spans several lines, so write it in `/srv/code/config/gitlab.rb` on the configuration volume.

Helm Chart

A provider entry is stored in a Kubernetes Secret and referenced from the values, so the client secret stays out of `values.yaml`. Create the Secret with the provider block as its content:

```
d8 k -n code create secret generic code-omniauth-keycloak \
  --from-file=provider=keycloak.yaml
```

The file holds the same entry as on the other installation types:

```
name: 'openid_connect'
label: 'Keycloak'
allowed_groups:
  - 'gitlab'
admin_groups:
  - 'admin'
groups_attribute: 'gitlab_group'
args:
  name: 'openid_connect'
  scope:
    - 'openid'
    - 'profile'
    - 'email'
  response_type: 'code'
  issuer: 'https://keycloak.example.com/realms/example'
  discovery: true
  client_auth_method: 'query'
  uid_field: 'preferred_username'
  send_scope_to_token_endpoint: false
  pkce: true
  client_options:
    identifier: '<client_id>'
    secret: '<client_secret>'
    redirect_uri: 'https://code.example.com/users/auth/openid_connect/callback'
```

Reference the Secret from the values:

```
global:
  appConfig:
    omniauth:
      providers:
        - secret: code-omniauth-keycloak
          key: provider
```

Deckhouse Kubernetes Platform module

The provider is an entry of the `providers` parameter in the `spec.appConfig.omniauth` section. The `allowedGroups`, `adminGroups` and `groupsAttribute` keys sit next to `name`; the CodeInstance resource has no parameter for auditor groups. The connection parameters sit inside `args`:

```
providers:
  - name: 'openid_connect'   # Do not change this value.
    label: 'Keycloak'        # Sign-in button label.
    allowedGroups:
      - 'gitlab'
    adminGroups:
      - 'admin'
    groupsAttribute: 'gitlab_group'
    args:
      name: 'openid_connect'
      scope:
        - 'openid'
        - 'profile'
        - 'email'
      response_type: 'code'
      issuer: 'https://keycloak.example.com/realms/example'
      discovery: true
      client_auth_method: 'query'
      uid_field: 'preferred_username'
      send_scope_to_token_endpoint: false
      pkce: true
      client_options:
        identifier: '<client_id>'
        secret: '<client_secret>'
        redirect_uri: 'https://code.example.com/users/auth/openid_connect/
callback'
```

SAML provider

The parameters of a SAML provider and their effect are described in SAML.

Linux package

A SAML provider is written as one more entry of the `gitlab_rails['omniauth_providers']` array:

```
gitlab_rails['omniauth_providers'] = [
  {
    name: 'saml',
    allowed_groups: ['gitlab'],
    admin_groups: ['admin'],
    groups_attribute: 'gitlab_group'
  }
]
```

Omnibus Docker

The entry is the one of the Linux package tab, written in `/srv/code/config/gitlab.rb` on the configuration volume.

Helm Chart

A SAML entry is stored in a Secret and referenced from `global.appConfig.omniauth.providers` the same way as the OpenID Connect entry.

Deckhouse Kubernetes Platform module

The provider is an entry of the `providers` parameter in the `spec.appConfig.omniauth` section:

```
providers:
  - name: 'saml'
    allowedGroups:
      - 'gitlab'
    adminGroups:
      - 'admin'
    groupsAttribute: 'gitlab_group'
```

LDAP servers

The key of a server entry is the server name. What each connection parameter does is described in LDAP synchronization; the `group_sync` section, which creates the groups and assigns the roles, is described in Groups and access rights and role_mapping section. The `group_sync` section is taken into account only for the server under the `main` key (see Synchronization scope).

Linux package

Signing in through LDAP is turned on by `gitlab_rails['ldap_enabled']`. The servers are described in `gitlab_rails['ldap_servers']`, whose value is a YAML document.

> ⚠ The value is parsed as YAML, so the indentation inside the block is significant and tab characters break it.

```
gitlab_rails['ldap_enabled'] = true
gitlab_rails['ldap_servers'] = YAML.load <<-'EOS'
  main:
    label: 'Head office'
    host: ldap-main.example.com
    port: 3389
    uid: 'cn'
    bind_dn: 'uid=viewer,ou=People,dc=example,dc=com'
    password: 'viewer123'
    base: 'ou=People,dc=example,dc=com'
    # Ignore users blocked in the directory (optional).
    user_filter: '(!(nsAccountLock=TRUE))'
    # The user is found in LDAP — sign-in is allowed immediately.
    block_auto_created_users: false
    sync_name: true
    group_sync:
      create_groups: true
      base: 'ou=Groups,dc=example,dc=com'
      filter: '(objectClass=groupOfNames)'
      prefix:
        attribute: 'businessCategory'
        default: 'default-program'
      top_level_group: 'LdapGroups'
      name_mask: '(?<=-)[A-z0-9]*$'
      owner: 'root'
      role_mapping:
        - by_name: '.*-project_manager-.*'
          gitlab_role: 'maintainer'
        - by_name: '.*-developer-.*'
          gitlab_role: 'developer'
        - by_name: '.*-participant-.*'
          gitlab_role: 'reporter'
  contractors:
    label: 'Contractors'
    host: ldap-contractors.example.com
    port: 3389
    uid: 'cn'
    bind_dn: 'uid=viewer,ou=People,dc=contractors,dc=example,dc=com'
    password: 'viewer123'
    base: 'ou=People,dc=contractors,dc=example,dc=com'
EOS
```

Omnibus Docker

The keys are `gitlab_rails['ldap_enabled']` and `gitlab_rails['ldap_servers']` of the Linux package tab. The value of `gitlab_rails['ldap_servers']` spans several lines, so write it in `/srv/code/config/gitlab.rb` on the configuration volume.

Helm Chart

The servers are values under `global.appConfig.ldap.servers`, with the same keys the product reads. The bind password is taken from a Secret:

```
d8 k -n code create secret generic code-ldap-password \
  --from-literal=password='viewer123'
```

```
global:
  appConfig:
    ldap:
      servers:
        main:
          label: 'Head office'
          host: ldap-main.example.com
          port: 3389
          uid: 'cn'
          bind_dn: 'uid=viewer,ou=People,dc=example,dc=com'
          password:
            secret: code-ldap-password
            key: password
          base: 'ou=People,dc=example,dc=com'
          user_filter: '(!(nsAccountLock=TRUE))'
          block_auto_created_users: false
          sync_name: true
          group_sync:
            create_groups: true
            base: 'ou=Groups,dc=example,dc=com'
            filter: '(objectClass=groupOfNames)'
            top_level_group: 'LdapGroups'
            owner: 'root'
            role_mapping:
              - by_name: '.*-developer-.*'
                gitlab_role: 'developer'
        contractors:
          label: 'Contractors'
          host: ldap-contractors.example.com
          port: 3389
          uid: 'cn'
          bind_dn: 'uid=viewer,ou=People,dc=contractors,dc=example,dc=com'
          password:
            secret: code-ldap-password
            key: password
          base: 'ou=People,dc=contractors,dc=example,dc=com'
```

Deckhouse Kubernetes Platform module

The servers are keys of the `servers` parameter in the `spec.appConfig.ldap` section:

```
servers:
  main:
    label: ldap
    host: 127.0.0.1
    port: 3389
    bindDn: 'uid=viewer,ou=People,dc=example,dc=com'
    base: 'ou=People,dc=example,dc=com'
    uid: 'cn'
    password: 'viewer123'
    syncName: true
    groupSync:
      createGroups: true
      base: 'ou=Groups,dc=example,dc=com'
      filter: '(objectClass=groupOfNames)'
      prefix:
        attribute: 'businessCategory'
        default: 'default-program'
      topLevelGroup: 'LdapGroups'
      nameMask: '(?<=-)[A-z0-9]*$'
      owner: 'root'
      roleMapping:
        - byName: '.*-project_manager-.*'
          gitlabRole: 'Maintainer'
        - byName: '.*-developer-.*'
          gitlabRole: 'Developer'
        - byName: '.*-participant-.*'
          gitlabRole: 'Reporter'
```

Multiple LDAP servers

Several servers are described by repeating the server entry under another key. Each server has its own connection parameters; the `group_sync` section with everything in it is taken into account only for the server under the `main` key (see Synchronization scope). How a user is identified and how the access decision is taken when the same user is found on several servers is described in Multiple LDAP servers.

Linux package

The entries are keys of the same YAML document in `gitlab_rails['ldap_servers']`, as `main` and `contractors` in the LDAP servers section above.

Omnibus Docker

The entries are the ones of the Linux package tab, written in `/srv/code/config/gitlab.rb` on the configuration volume.

Helm Chart

The entries are keys under `global.appConfig.ldap.servers`, as `main` and `contractors` in the LDAP servers section above.

Deckhouse Kubernetes Platform module

The entries are keys of the `servers` parameter in the `spec.appConfig.ldap` section:

```
servers:
  main:
    label: 'Head office'
    host: ldap-main.example.com
    base: 'ou=People,dc=example,dc=com'
    uid: 'cn'
    # Other connection parameters.
    groupSync:
      base: 'ou=Groups,dc=example,dc=com'
      roleMapping:
        - byName: '.*-developer-.*'
          gitlabRole: 'Developer'
  contractors:
    label: 'Contractors'
    host: ldap-contractors.example.com
    base: 'ou=People,dc=contractors,dc=example,dc=com'
    uid: 'cn'
    # Other connection parameters.
```

Linking OIDC accounts to LDAP

Automatic linking of the OIDC account to the LDAP account is turned on by the `auto_link_ldap_user` parameter. How the LDAP account is found, and what happens on the first and the subsequent sign-ins, is described in Linking OIDC accounts to LDAP.

Linux package

```
gitlab_rails['omniauth_auto_link_ldap_user'] = true
```

Omnibus Docker

The key is the one of the Linux package tab. Its value is a single line, so it fits into `GITLAB_OMNIBUS_CONFIG` as well as into `/srv/code/config/gitlab.rb`.

Helm Chart

```
global:
  appConfig:
    omniauth:
      autoLinkLdapUser: true
```

Deckhouse Kubernetes Platform module

The parameter is set in the `spec.appConfig.omniauth` section:

```
autoLinkLdapUser: true
```

LDAP as the source of truth

To allow users found in LDAP to sign in immediately and to send everyone else for administrator approval, combine `auto_link_ldap_user` and `block_auto_created_users` of the OmniAuth parameters with `block_auto_created_users` of the `main` LDAP server (see LDAP as the source of truth).

Linux package

```
gitlab_rails['omniauth_auto_link_ldap_user'] = true
# The user is not found in LDAP — the account is created blocked,
# pending administrator approval.
gitlab_rails['omniauth_block_auto_created_users'] = true
gitlab_rails['ldap_servers'] = YAML.load <<-'EOS'
  main:
    # The user is found in LDAP — sign-in is allowed immediately.
    block_auto_created_users: false
    # Other connection parameters.
EOS
```

Omnibus Docker

The keys are the ones of the Linux package tab, written in `/srv/code/config/gitlab.rb` on the configuration volume.

Helm Chart

```
global:
  appConfig:
    omniauth:
      autoLinkLdapUser: true
      # The user is not found in LDAP — the account is created blocked,
      # pending administrator approval.
      blockAutoCreatedUsers: true
    ldap:
      servers:
        main:
          # The user is found in LDAP — sign-in is allowed immediately.
          block_auto_created_users: false
```

Deckhouse Kubernetes Platform module

The parameters are set in the `spec.appConfig` section:

```
omniauth:
  autoLinkLdapUser: true
  # The user is not found in LDAP — the account is created blocked,
  # pending administrator approval.
  blockAutoCreatedUsers: true
ldap:
  servers:
    main:
      # The user is found in LDAP — sign-in is allowed immediately.
      blockAutoCreatedUsers: false
```

Synchronization schedule

The schedule of the synchronization job is written in cron format. What the job does on each run is described in LDAP synchronization; a run started by hand is described in Manual synchronization run.

Linux package

```
gitlab_rails['ldap_sync_worker_cron'] = "0 * * * *"
```

Omnibus Docker

The key is the one of the Linux package tab. Its value is a single line, so it fits into `GITLAB_OMNIBUS_CONFIG` as well as into `/srv/code/config/gitlab.rb`.

Helm Chart

```
global:
  appConfig:
    cron_jobs:
      ldap_sync_worker:
        cron: "0 * * * *"
```

Deckhouse Kubernetes Platform module

The interval is set by the `cronJobs` parameter of the `spec.appConfig` section:

```
cronJobs:
  ldapSyncWorker:
    cron: "0 * * * *"
```

Applying the configuration

Linux package

Every change to `/etc/gitlab/gitlab.rb` takes effect after:

```
sudo gitlab-ctl reconfigure
```

Omnibus Docker

Apply the edited file inside the running container:

```
docker exec code gitlab-ctl reconfigure
```

A container restart applies the configuration as well, and it is the way a changed `GITLAB_OMNIBUS_CONFIG` of a recreated container takes effect:

```
docker restart code
```

Helm Chart

Apply the edited `values.yaml` file to the release:

```
helm upgrade code deckhouse-code \
  -n code \
  -f values.yaml
```

Deckhouse Kubernetes Platform module
The operator applies a changed `CodeInstance` itself.

Complete example: signing in through OIDC with permissions from LDAP

The steps below describe a setup where users sign in through an OIDC provider (for example, Keycloak), while groups, memberships, and roles come from LDAP.

The setup includes the following steps:

1. Configure the LDAP provider.
2. Configure the OIDC provider and enable the LDAP integration.
3. Verify the integration upon first login.
4. Synchronize permissions.

Prerequisites

To set this up, you'll need:

- An OIDC provider.
- An LDAP directory with the same users and with groups whose names allow determining the role.
- An LDAP service account with read access to the directory (the `bind_dn` and `password` parameters).

> The `uid` or email value in the OIDC provider must match the value of the corresponding attribute in LDAP, otherwise linking will not work.

How the account is matched is described in How the LDAP account is found.

Configure the LDAP provider

The server entry names the directory, the search base and the service account, and its `group_sync` section turns the group names into roles.

Linux package

```
gitlab_rails['ldap_enabled'] = true
gitlab_rails['ldap_servers'] = YAML.load <<-'EOS'
  main:
    label: ldap
    host: ldap.example.com
    port: 3389
    bind_dn: 'uid=viewer,ou=People,dc=example,dc=com'
    password: 'viewer123'
    base: 'ou=People,dc=example,dc=com'
    uid: 'cn'
    sync_name: true
    # Ignore users blocked in the directory (optional).
    user_filter: '(!(nsAccountLock=TRUE))'
    # The user is found in LDAP — sign-in is allowed immediately.
    block_auto_created_users: false
    group_sync:
      create_groups: true
      base: 'ou=Groups,dc=example,dc=com'
      filter: '(objectClass=groupOfNames)'
      top_level_group: 'LdapGroups'
      name_mask: '(?<=-)[A-z0-9]*$'
      owner: 'root'
      role_mapping:
        - by_name: '.*-maintainer-.*'
          gitlab_role: 'maintainer'
        - by_name: '.*-developer-.*'
          gitlab_role: 'developer'
        - by_name: '.*-participant-.*'
          gitlab_role: 'reporter'
EOS
```

Omnibus Docker

The keys are `gitlab_rails['ldap_enabled']` and `gitlab_rails['ldap_servers']` of the Linux package tab, written in `/srv/code/config/gitlab.rb` on the configuration volume.

Helm Chart

```
global:
  appConfig:
    ldap:
      servers:
        main:
          label: ldap
          host: ldap.example.com
          port: 3389
          bind_dn: 'uid=viewer,ou=People,dc=example,dc=com'
          password:
            secret: code-ldap-password
            key: password
          base: 'ou=People,dc=example,dc=com'
          uid: 'cn'
          sync_name: true
          # Ignore users blocked in the directory (optional).
          user_filter: '(!(nsAccountLock=TRUE))'
          # The user is found in LDAP — sign-in is allowed immediately.
          block_auto_created_users: false
          group_sync:
            create_groups: true
            base: 'ou=Groups,dc=example,dc=com'
            filter: '(objectClass=groupOfNames)'
            top_level_group: 'LdapGroups'
            name_mask: '(?<=-)[A-z0-9]*$'
            owner: 'root'
            role_mapping:
              - by_name: '.*-maintainer-.*'
                gitlab_role: 'maintainer'
              - by_name: '.*-developer-.*'
                gitlab_role: 'developer'
              - by_name: '.*-participant-.*'
                gitlab_role: 'reporter'
```

Deckhouse Kubernetes Platform module

Set up the configuration in the `servers` parameter of the `spec.appConfig.ldap` section:

```
servers:
  main:
    label: ldap
    host: ldap.example.com
    port: 3389
    bindDn: 'uid=viewer,ou=People,dc=example,dc=com'
    password: 'viewer123'
    base: 'ou=People,dc=example,dc=com'
    uid: 'cn'
    syncName: true
    # Ignore users blocked in the directory (optional).
    userFilter: '(!(nsAccountLock=TRUE))'
    # The user is found in LDAP — sign-in is allowed immediately.
    blockAutoCreatedUsers: false
    groupSync:
      createGroups: true
      base: 'ou=Groups,dc=example,dc=com'
      filter: '(objectClass=groupOfNames)'
      topLevelGroup: 'LdapGroups'
      nameMask: '(?<=-)[A-z0-9]*$'
      owner: 'root'
      roleMapping:
        - byName: '.*-maintainer-.*'
          gitlabRole: 'Maintainer'
        - byName: '.*-developer-.*'
          gitlabRole: 'Developer'
        - byName: '.*-participant-.*'
          gitlabRole: 'Reporter'
```

Configure the OIDC provider and enable linking to LDAP

Pay attention to the `uid_field` parameter: the field it points to becomes the account `uid`, and this value is used when looking the user up in LDAP. It must match the value of the attribute specified in the LDAP server `uid` parameter (`cn` in this example), or the email address — for details, see How the LDAP account is found.

The remaining provider parameters are described in the OpenID Connect (OIDC) section.

Linux package

```
gitlab_rails['omniauth_enabled'] = true
gitlab_rails['omniauth_auto_link_ldap_user'] = true
gitlab_rails['omniauth_providers'] = [
  {
    name: 'openid_connect',   # Do not change this value.
    label: 'Keycloak',        # Sign-in button label.
    groups_attribute: 'gitlab_group',
    args: {
      name: 'openid_connect',
      scope: ['openid', 'profile', 'email'],
      response_type: 'code',
      issuer: 'https://keycloak.example.com/realms/example',
      discovery: true,
      client_auth_method: 'query',
      uid_field: 'preferred_username',
      send_scope_to_token_endpoint: false,
      pkce: true,
      client_options: {
        identifier: '<client_id>',
        secret: '<client_secret>',
        redirect_uri: 'https://code.example.com/users/auth/openid_connect/
callback'
      }
    }
  }
]
```

Omnibus Docker

The keys are `gitlab_rails['omniauth_enabled']`, `gitlab_rails['omniauth_auto_link_ldap_user']` and `gitlab_rails['omniauth_providers']` of the Linux package tab, written in `/srv/code/config/gitlab.rb` on the configuration volume.

Helm Chart

The provider entry of this example is stored in a Secret and referenced from the values, with linking turned on beside it:

```
global:
  appConfig:
    omniauth:
      enabled: true
      autoLinkLdapUser: true
      providers:
        - secret: code-omniauth-keycloak
          key: provider
```

Deckhouse Kubernetes Platform module

Set up the configuration in the `spec.appConfig.omniauth` section:

```
autoLinkLdapUser: true
providers:
  - name: 'openid_connect'   # Do not change this value.
    label: 'Keycloak'        # Sign-in button label.
    groupsAttribute: 'gitlab_group'
    args:
      name: 'openid_connect'
      scope:
        - 'openid'
        - 'profile'
        - 'email'
      response_type: 'code'
      issuer: 'https://keycloak.example.com/realms/example'
      discovery: true
      client_auth_method: 'query'
      uid_field: 'preferred_username'
      send_scope_to_token_endpoint: false
      pkce: true
      client_options:
        identifier: '<client_id>'
        secret: '<client_secret>'
        redirect_uri: 'https://code.example.com/users/auth/openid_connect/
callback'
```

Verify linking on the first sign-in

To verify linking, follow these steps:

1. Sign in with a test user through the OIDC provider.
2. Open the user page in the admin area (`/admin/users/<username>/identities`). A linked account must have two identities: `openid_connect` and `ldapmain` (the LDAP identity name consists of the `ldap` prefix and the LDAP server name, `main` in this example).

If the LDAP identity is missing, check the following:

- the `uid` or email values match in the OIDC provider and in LDAP;
- the user falls within the `base` search scope and is not filtered out by `user_filter` ;
- the `auto_link_ldap_user` parameter is enabled.

Rights synchronization

Right after the first sign-in, the user has an account but no group or project memberships. Wait for the next synchronization or run it manually (see Manual synchronization run), then check that:

- the groups are created inside the group specified in `group_sync.top_level_group` ;
- the user is added to them with the role matching `role_mapping` .

Security settings

Administrator account

The instance starts with the `root` administrator account, whose password is generated during the installation.

1. Sign in as `root` with the initial password.
2. Open the user menu and select “Edit profile”.
3. Go to “Access” → “Password and authentication” and set a new password.
4. Keep the new password outside the instance.

After the password is changed, remove the initial password from the place the delivery keeps it:

Linux package
The `/etc/gitlab/initial_root_password` file, deleted with `sudo rm -f /etc/gitlab/initial_root_password` . A `gitlab-ctl reconfigure` run more than 24 hours after the installation deletes the file on its own, and the password value stays as it was.

Omnibus Docker

The `/etc/gitlab/initial_root_password` file inside the container, deleted with `docker exec code rm -f /etc/gitlab/initial_root_password`. The file lies on the configuration volume, so with the volumes of Quick start it is `/srv/code/config/initial_root_password` on the host.

Helm Chart

The `code-initial-root-password` secret of the release namespace, read with `d8 k -n code get secret code-initial-root-password -o jsonpath='{.data.password}' | base64 -d`. Once the password is changed, the value in the secret no longer opens the account.

Deckhouse Kubernetes Platform module

The `initial-root-password` secret of the `d8-code` namespace, read with `d8 k -n d8-code get secret initial-root-password -o jsonpath='{.data.password}' | base64 -d`. Once the password is changed, the value in the secret no longer opens the account.

Sign-up and sign-in

New user accounts

After an installation of the Linux package, of Omnibus Docker or of the Helm Chart, anyone who reaches the instance over the network creates an account for themselves. To close registration:

1. Go to “Admin” → “Settings” → “General”.
2. Expand “New user account restrictions”.
3. Clear the “Allow new user accounts” checkbox.
4. Click “Save changes”.

In the Deckhouse Kubernetes Platform module, registration is closed after the installation; the operator sets it from the `appConfig.signUpEnabled` field of the CodeInstance resource.

In the other installation types the same is done from the console:

Linux package

```
sudo gitlab-rails runner 'ApplicationSetting.current.update!(signup_enabled:
false)'
sudo gitlab-rails runner 'puts ApplicationSetting.current.signup_enabled'
```

Omnibus Docker

```
docker exec code gitlab-rails runner 'ApplicationSetting.current.update!
(signup_enabled: false)'
docker exec code gitlab-rails runner 'puts
ApplicationSetting.current.signup_enabled'
```

Helm Chart

```
d8 k -n code exec deploy/code-toolbox -- gitlab-rails runner 'ApplicationSetting.c
urrent.update!(signup_enabled: false)'
d8 k -n code exec deploy/code-toolbox -- gitlab-rails runner 'puts
ApplicationSetting.current.signup_enabled'
```

The second command prints `false` while registration is closed.

While registration stays open, the same section of the admin area limits who gets an account:

- “Require admin approval for new user accounts” — an account waits for an administrator before its first sign-in;
- “Email confirmation settings” — in the “Hard” mode the address is confirmed before the first sign-in;
- “Domains allowed for new users” and “Domain denylist” — the email domains that may and may not register;
- “Minimum password length (number of characters)” — applies to accounts that sign in with a password, 8 characters by default.

None of these settings applies to an account created through an external provider.

Sign-in restrictions

1. Go to “Admin” → “Settings” → “General”.
2. Expand “Sign-in restrictions”.
3. Select “Enforce two-factor authentication for administrators”.
4. Click “Save changes”.

The same section holds:

- “Enable Admin Mode” — the administrator authenticates once more before the admin area opens;
- “Email notification for unknown sign-ins” — the user gets a message when a sign-in comes from a device or an IP address that was not used before;

- “Require email verification when account is locked” — a locked account confirms its mailbox before it signs in again.

Two-factor authentication for every user of the instance, the enabled Git access protocols and password authentication for Git over HTTPS are set in the same section, see the Users and access page. Password authentication for the web interface is described on the Authentication page.

External authentication provider

With LDAP, SAML or OIDC, the accounts and the group memberships live in the directory, and an account closed there loses access to Deckhouse Code without an action inside the instance. The providers, the synchronization and the linking of accounts are described on the Authentication page.

Visibility and access control

The settings are in “Admin” → “Settings” → “General” → “Visibility and access controls”:

- “Default project visibility”, “Default snippet visibility” and “Default group visibility” — set them to “Private”, so that a new project, snippet or group is closed until someone opens it;
- “Restricted visibility levels” — the levels a non-administrator cannot choose. With the public and internal levels selected here, a regular user creates private projects only;
- the retention period for a deleted group or project — the time during which a deletion is still reversible;
- “Default minimum role required to create projects” and “Enabled Git access protocols” — see the Users and access page;
- “RSA SSH keys”, “DSA SSH keys”, “ECDSA SSH keys” and the other key lists — each one forbids its algorithm or sets the minimum key length the instance accepts from a user.

Forbid DSA keys, set the minimum length of an RSA key to 3072 bits and of an ECDSA key to 256 bits. Leave the `ECDSA_SK` and `ED25519_SK` lists enabled while hardware FIDO2 or U2F keys are in use.

User account restrictions

The settings are in “Admin” → “Settings” → “General” → “Account and limit”:

- “Default projects limit” — the maximum number of projects one user creates;
- “Session timeout duration” — the session length in minutes, 7 days by default; the change applies after the instance is restarted;
- “Remember me” → “Allow users to extend their session” — with the checkbox cleared, a session ends when the timeout expires, whatever the user does;
- “Require expiration date” — a new personal, group or project token gets an expiry date;
- “User OAuth applications” — with the checkbox cleared, a user does not register an OAuth client of their own on the instance;
- the “User restrictions” group — group and project creation, see the Users and access page.

Restart the instance after changing the session length:

Linux package

```
sudo gitlab-ctl restart
```

Omnibus Docker

```
docker restart code
```

Helm Chart

```
d8 k -n code rollout restart deploy/<WEBSERVICE_DEPLOYMENT>
```

Deckhouse Kubernetes Platform module

```
d8 k -n d8-code rollout restart deploy -l 'app.kubernetes.io/component in
(webservice, sidekiq)'
```

Import and export

The settings are in “Admin” → “Settings” → “General” → “Import and export settings”:

- “Import sources” — the systems a project is imported from; every source is off by default, and the “Repository by URL” source accepts any Git repository by its address;
- “Project export” — with the checkbox cleared, a user does not build a `.tar.gz` archive of a project to carry it to another instance;
- “Allow migrating GitLab groups and projects by direct transfer” — off by default;
- “Silent exports by admins” — while the checkbox is selected, an export made by an administrator leaves no audit event.

Repository settings

Default branch

1. Go to “Admin” → “Settings” → “Repository”.
2. Expand “Default branch”.

3. In the “Initial default branch name” field, enter the branch name every new repository starts with.
4. In “Initial default branch protection”, select the protection a new repository’s default branch starts with, then set “Allowed to push” and “Allowed to merge” to the role the branch accepts changes from.
5. Clear “Allowed to force push” and “Allow developers to push the initial commit”.
6. Click “Save changes”.

The settings apply to repositories created after the change; an existing repository keeps its branch rules.

Push rules

The “Push rules” section of the same page sets the rules every project of the instance starts with: the regular expressions for the commit message, the author email address, the file name and the branch name, the committer checks, and the block on committing secrets such as `pem` or `id_rsa` files. A rule whose override is cleared cannot be changed in a group or a project. The rules are described on the Push rules page, and the instance level on the Users and access page.

CI/CD settings

Runner authentication tokens

“Admin” → “Settings” → “CI/CD” → “Continuous Integration and Deployment” sets the lifetime of the authentication tokens of instance, group and project runners; the lifetime is unlimited until a value is entered. An online runner rotates its token as the expiry approaches; a runner that was offline at that moment is registered anew.

Runners

“Admin” → “Settings” → “CI/CD” → “Runners” holds:

- “Fetch GitLab Runner release version data from GitLab.com” — while the checkbox is selected, the instance requests the runner release versions from GitLab.com; clear it on an installation that has no access to the internet;
- “Allow runner registration token” — with the checkbox cleared, a runner is registered with an authentication token of its own instead of a registration token shared by the instance, the group or the project.

Job token permissions

“Admin” → “Settings” → “CI/CD” → “Job token permissions” holds “Enable and enforce job token allowlist for all projects”. While it is selected, a job reaches another project only when that project lists the job’s project in its allowlist, and the option that opens a project to every group and project is hidden.

Spam and bot protection

“Admin” → “Settings” → “Reporting” → “Spam and Anti-bot Protection” holds reCAPTCHA and the limit on the number of accounts per IP address. The section applies to an instance that is reachable from the internet.

Usage statistics

In “Admin” → “Settings” → “Metrics and profiling”, the “Usage statistics” section holds the version check and Service Ping, and the “Event tracking” section holds the sending of events. Event tracking is off and its checkbox is read-only; a version check that is off is not turned back on from the interface.

Network rate limits

“Admin” → “Settings” → “Network” holds the request limits: “User and IP rate limits” for the instance as a whole, “Git HTTP rate limits”, “Git LFS rate limits”, “Search rate limits”, “Pipeline creation rate limits” and “Import and export rate limits” for their own kind of request, each overriding the general limit. “Outbound requests” restricts the addresses that hooks and integrations reach from the instance.

Platform-level hardening

In the Deckhouse Kubernetes Platform module, the traffic between the components, access to the `d8-code` namespace, the network policies and the export of audit events are set in the cluster. They are described in Security recommendations in the module documentation.

Security credentials

The Security credentials section provides a comprehensive overview of all tokens (personal, group, project) and keys (SSH, GPG) on the Code platform. Administrators can switch tabs of different credential types, apply filters and sorting options, and delete or revoke any credential to limit user access and enhance security.

Purpose

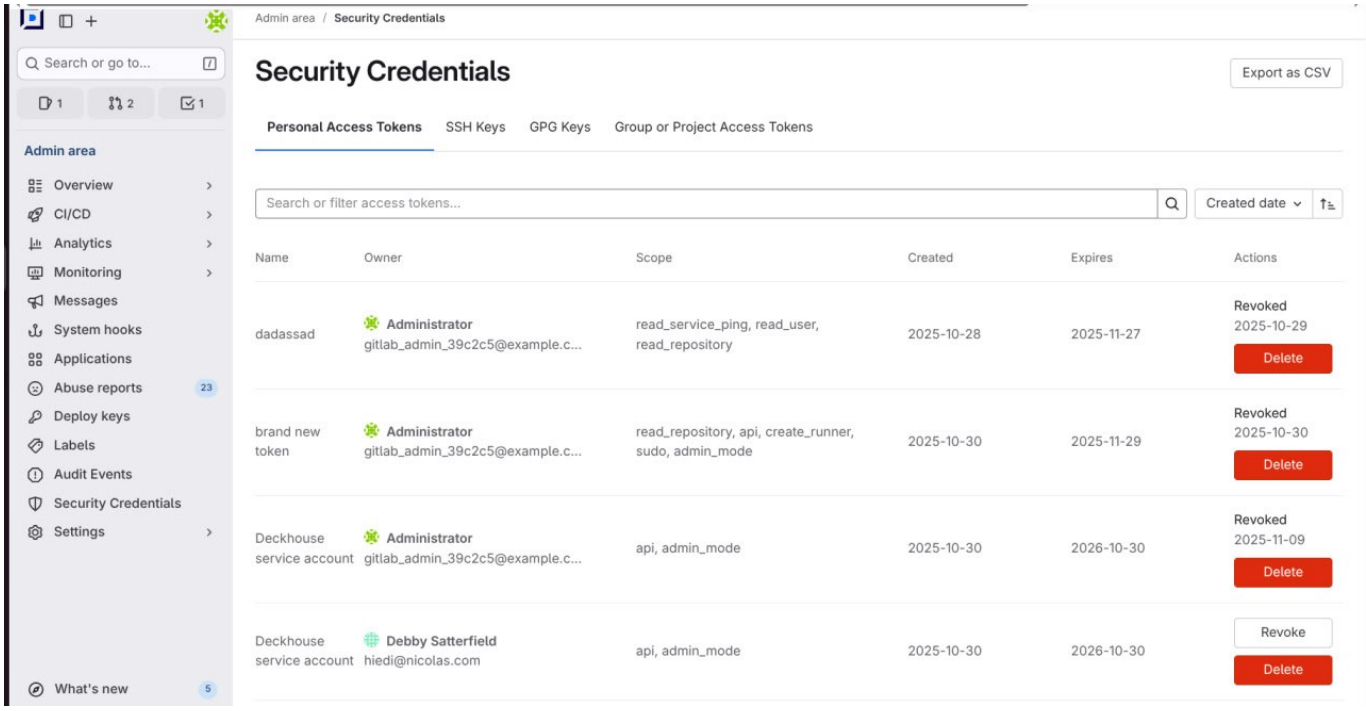
The security credentials section is used for:

- Providing a complete overview of all security credentials (ownership, scope, creation/expiration/revocation dates, etc.).
- Filtering and sorting credentials.
- Managing the credential lifecycle: revocation, deletion.

Accessing security credentials via UI

To access security credentials, switch to the admin mode and select “Security credentials” in the sidebar. This opens the security credentials table. The section consists of 4 tabs:

- **Personal Access Tokens**: List of tokens, which belong to real users.
- **SSH Keys**: List of SSH keys, which belong to real users.
- **GPG Keys**: List of GPG keys, which belong to real users.
- **Group/Project Access Keys**: List of tokens, which belong to top-level groups or projects.



Available actions

The set of actions that can be performed on a credential depends on its type. The table shows the relationship between actions and the credential type.

Action\Credential type	Personal Access Token	SSH Key	GPG Key	Group/Project Access Token
Removal	Yes	Yes	Yes	Yes
Revocation	Yes	No	No	Yes

Service accounts

A service account is a user account intended for use in automated scripts. These accounts are typically used in CI/CD pipelines and integrations. A service account cannot be used to authenticate via the web interface or to perform actions through impersonation.

Creating a service account

Rails console

A service account is created from the Rails console. Open the console for the installation type in use:

Linux package

```
sudo gitlab-rails console
```

Omnibus Docker

```
docker exec -it code gitlab-rails console
```

`code` is the container name from the `docker run` command in Quick start.

Helm Chart

```
d8 k -n code exec -it deploy/code-toolbox -- gitlab-rails console
```

Deckhouse Kubernetes Platform module

The console is part of the Toolbox utility set:

```
d8 k -n d8-code exec -it -c toolbox deploy/toolbox -- gitlab-rails console -e
production
```

Creating an account

1. In the Rails console, prepare the parameters defining the account to be created. Fill in the `name`, `username`, `email`, and `admin` fields, and define the rest of the parameters as shown in the example below:

```
user_args = {
name: 'kaiten_sa',
username: 'kaiten_sa',
email: 'kaiten_sa@flant.com',
admin: false,
user_type: :service_account,
organization_id: Organizations::Organization.default_organization.id,
password_automatically_set: true,
force_random_password: true,
skip_confirmation: true
}
```

2. Select the user on whose behalf the service account will be created and execute the account creation:

```
user = User.find_by_username('root')
Users::CreateService.new(user, user_args).execute
```

Generating an access token

To generate an access token, use GitLab's Personal access tokens API.

Audit events

Security audit is a detailed analysis of your infrastructure aimed at identifying potential vulnerabilities and unsafe practices.
Code simplifies the auditing process with audit events, which allow you to track a wide range of activities happening in the system.

Purpose and scope

The security event logging mechanism (audit events) is used to:

- Record user and administrator actions related to system configuration changes.
- Register information security incidents.
- Provide the ability to investigate incidents and reconstruct a complete picture of actions in the system.

The scope of application is all levels of the Code platform — from individual projects and groups to instance configuration.
Events are recorded regardless of user role (if they have the right to perform the action) and stored centrally.

Technical capabilities for event logging

The audit system features the following capabilities:

- **Centralized event logging**: All events are stored in a single audit log.
- **Real-time collection**: Events are logged instantly at the time of business operation execution. This includes, for example, user login, email change, or enabling `force push` in a protected branch.
- **Integrity protection**: The audit log is read-only for administrators. Logged events can't be deleted or modified.
- **Access via UI or API**: Events can be viewed and filtered both in the administrator interface and through the dedicated API.

Use cases

Audit events let you track:

- Who and when changed a user's access level in a Code project.
- Cases of user creation and deletion.
- Traces of suspicious activity — for example, cases of a bulk employee email change or a repository deletion.
- Changes made to CI/CD environment variables.
- Changes to project and group visibility levels.

Audit events help you:

- Assess risks and strengthen security measures.
- Respond promptly to incidents.

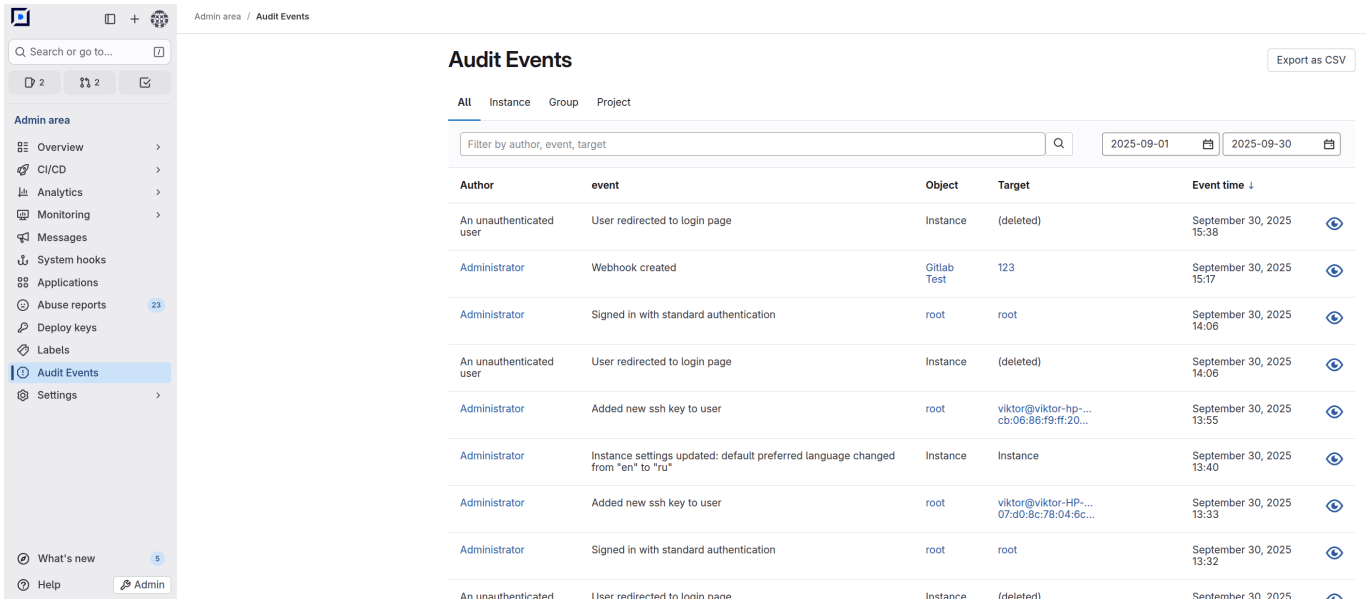
Accessing audit events

Accessing via UI

To access audit events, switch to the admin mode and select "Audit events" in the sidebar. This opens the audit event table.

Description of the table columns:

- **Author**: User who triggered the event.
- **Event**: System message with event details.
- **Object**: Related scope of the event (instance, user, group, or project name).
- **Target**: Entity that was changed (project, user, protected branch, token, or CI variable name, etc.).
- **Event time**: Date and time when the event occurred.



Accessing via API

Deckhouse Code provides the following API method to retrieve the list of audit events:

`POST /api/v4/admin/audit_events/search`

The method supports filtering by dates, full-text search, and entity types.

> ⚠ The date range must be within a single calendar month. If `created_after` and `created_before` belong to different months, the `created_before` value will be automatically adjusted to the last day of the month of `created_after`.

Request parameters

Parameter	Type	Required	Description
`created_after`	String	No	Start date (inclusive) in ISO8601 format. Default: beginning of the current month
`created_before`	String	No	End date (inclusive) in ISO8601 format. Default: end of the current month
`q`	String	No	Full-text search in the event message
`sort`	String	No	Sort by creation date. Possible values: `created_asc`, `created_desc` (default)
`entity_types`	Array	No	List of entity types for filtering: `User`, `Project`, `Group`, `Gitlab::Audit::InstanceScope`

Example request:

```
curl --request POST "https://example.com/api/v4/admin/audit_events/search" \
     --header "PRIVATE-TOKEN: <ACCESS_TOKEN>" \
     --header "Content-Type: application/json" \
     --data '{
       "created_after": "2025-08-01",
       "created_before": "2025-08-31",
       "q": "repository",
       "sort": "created_desc",
       "entity_types": ["Project"]
     }'
```

Audit event contents

Each audit event contains a date, time, IP address and user account details, as well as all required information about the scope of changes, object, and what exactly has been changed.

List of audit events

The table below shows example system messages.
Audit events in a production environment contain full information either directly in the message or in an additional JSON field with data.

Name	System message	Purpose	Audited attributes
`2fa_login_failed`	User 2fa login failed	A failed attempt to log in with two-factor authentication was detected.	
`access_approved`	User access was approved	User's access request to the instance was approved.	
`access_token_created`	Project/Group access token created	An access token for a project or group was created.	
`access_token_revoked`	Project/Group access token revoked	An access token was revoked.	
`added_gpg_key`	Added new gpg key to user	A user added a new GPG key.	
`added_ssh_key`	User added new ssh key	A user added a new SSH key.	
`application_created`	Application was created	A new application (OAuth or integration) was created.	
`application_deleted`	Application deleted	An application was deleted.	
`application_secret_renew`	Application secret renew	The application secret was renewed.	
`application_updated`	Application Updated	Application parameters were updated.	
`ci_cd_job_token_removed_from_allowlist`	Disallow group to use job token	A project restricted a specific group from using CI/CD Job Token.	
`ci_cd_job_token_added_to_allowlist`	Allow group to use job token	A project allowed a specific group to use CI/CD Job Token.	

Name	System message	Purpose	Audited attributes
`ci_variable_created`	Ci variable `#{key}` created	A new CI/CD variable was created.	
`ci_variable_deleted`	Ci variable `#{key}` deleted	A CI/CD variable was deleted.	
`ci_variable_updated`	Ci variable updated (Value, Protected)	The value or protection status of a CI/CD variable was updated.	
`deploy_key_created`	Deploy key added	A new deploy key was added for a project or instance.	
`deploy_key_deleted`	Deploy key was deleted	A deploy key was deleted.	
`deploy_key_disabled`	Deploy key disabled	A deploy key was disabled.	
`deploy_key_enabled`	Deploy key enabled	A deploy key was re-enabled.	
`deploy_token_created`	Deploy token created	A deploy token was created for data access.	
`deploy_token_deleted`	Deploy token deleted	A deploy token was deleted.	
`deploy_token_revoked`	Deploy token revoked	A deploy token was revoked by a user or the system.	
`feature_flag_created`	Created feature flag with description	A new feature flag was created.	
`feature_flag_deleted`	Feature flag was deleted	A feature flag was deleted.	
`feature_flag_updated`	Feature flag was updated	A feature flag was updated.	
`group_created`	Group was created		

Name	System message	Purpose	Audited attributes
		A new group was created.	
`group_export_created`	Group file export was created	A group export file was generated.	
`group_invite_via_group_link_created`	Invited group to group	Another group was invited to the group using a group link.	
`group_invite_via_group_link_deleted`	Revoked group from group	Group access through a group link was revoked.	
`group_invite_via_group_link_updated`	Group access changed	Group access parameters through a group link were updated.	
`group_invite_via_project_group_link_created`	Invited group to project	A group was invited to a project using a group link.	
`group_invite_via_project_group_link_deleted`	Revoked group from project	Group access to a project was revoked.	
`group_invite_via_project_group_link_updated`	Group access for project changed	Group access parameters to a project were updated.	
`group_updated`	Group updated (visibility, 2FA grace period)	Changes in group settings (visibility, security, limits, access policy).	• `repository_size_limit` • `two_factor_grace_period` • `lfs_enabled` • `membership_lock` • `path` • `require_two_factor_authentication`

Name	System message	Purpose	Audited attributes
			• `request_access_enabled` • `shared_runners_minutes_limit` • `share_with_group_lock` • `mentions_disabled` • `max_personal_access_token_lifetime` • `visibility_level` • `name` • `description` • `project_creation_level` • `default_branch_protected` • `seat_control` • `duo_features_enabled` • `prevent_forking_outside_group` • `allow_mfa_for_subgroups` • `default_branch_name` • `resource_access_token_creation_allowed` • `new_user_signups_cap` • `show_diff_preview_in_email` • `enabled_git_access_protocol`

Name	System message	Purpose	Audited attributes
			• `runner_registration_enabled` • `allow_runner_registration_token` • `emails_enabled` • `service_access_tokens_expiration_enforced` • `enforce_ssh_certificates` • `disable_personal_access_tokens` • `remove_dormant_members` • `remove_dormant_members_period` • `prevent_sharing_groups_outside_hierarchy` • `default_branch_protection_defaults` • `wiki_access_level`
`impersonation_initiated`	User root impersonated another user	An administrator started a session impersonating another user.	
`impersonation_stopped`	User root stopped impersonation	An administrator stopped impersonating another user.	
`instance_settings_updated`		Global instance settings were updated.	

Name	System message	Purpose	Audited attributes
	Instance settings updated: Signup enabled turned on		All instance settings except encrypted fields.
`login_failed`	Attempt to login failed	A login attempt failed.	
`manually_trigger_housekeeping`	Housekeeping task	A housekeeping task for a repository was triggered manually.	
`member_permissions_created`	New member access granted	A user was granted membership (role) in a group or project.	
`member_permissions_destroyed`	Member access revoked	A user's membership was revoked from a group or project.	
`member_permissions_updated`	Member access updated	A user's role or membership expiration date was updated.	
`merge_request_closed_by_project_bot`	Merge request `#{merge_request.title}` closed by project bot	A merge request was closed by a project bot.	
`merge_request_created_by_project_bot`	Merge request `#{merge_request.title}` created by project bot	A merge request was created by a project bot.	
`merge_request_merged_by_project_bot`	Merge request `#{merge_request.title}` merged by project bot	A merge request was merged by a project bot.	
`merge_request_reopened_by_project_bot`	Merge request `#{merge_request.title}` reopened by project bot	A merge request was reopened by a project bot.	
`omniauth_login_failed`			

Name	System message	Purpose	Audited attributes
	Omniauth login failed for `#{user}` `#{provider}`	Failed login attempt via external OAuth/ Omniauth provider.	
`password_reset_failed`	Password reset failed	Failed password reset attempt by a user.	
`personal_access_token_issued`	Personal access token issued	A new personal access token was issued.	
`personal_access_token_revoked`	Personal access token revoked	A personal access token was revoked.	
`pipeline_deleted`	Pipeline deleted	A CI/CD pipeline was deleted.	
`project_blobs_removal`	Project blobs removed	Bulk removal of repository blobs in a project.	
`project_created`	Project was created	A new project was created.	
`project_default_branch_changed`	Project default branch updated	The default branch of a project was changed.	
`project_export_created`	Project export created	A project export file was generated.	
`project_feature_updated`	Project features updated	Feature access levels for a project were updated (issues, wiki, etc.).	
`project_setting_updated`	Project settings updated	Project merge commit and squash commit templates were updated.	
`project_text_replacement`	Project text replaced	Bulk text replacement in a project.	
	Project topic changed		

Name	System message	Purpose	Audited attributes
`project_topic_changed`		Project topic was updated.	
`project_updated`	Project updated (name, namespace)	Project settings were updated (name, namespace, policies).	• `name` • `packages_enabled` • `reset_approvals_on_push` • `path` • `merge_requests_author_approval` • `merge_requests_disable_committers_approval` • `only_allow_merge_if_all_discussions_are_resolved` • `only_allow_merge_if_pipeline_succeeds` • `require_password_to_approve` • `disable_overriding_approvers_per_merge_request` • `repository_size_limit` • `project_namespace_id` • `namespace_id` • `printing_merge_request_link_enabled` • `resolve_outdated_diff_discussions`

Name	System message	Purpose	Audited attributes
			• `merge_requests_ff_only_enabled` • `merge_requests_rebase_enabled` • `remove_source_branch_after_merge` • `merge_requests_template` • `visibility_level` • `builds_access_level` • `container_registry_access_level` • `environments_access_level` • `feature_flags_access_level` • `forking_access_level` • `infrastructure_access_level` • `issues_access_level` • `merge_requests_access_level` • `metrics_dashboard_access_level` • `monitor_access_level` • `operations_access_level` • `package_registry_access_level`

Name	System message	Purpose	Audited attributes
			• `pages_access_level` • `releases_access_level` • `repository_access_level` • `requirements_access_level` • `security_and_compliance_access_level` • `snippets_access_level` • `wiki_access_level` • `merge_commit_template` • `squash_commit_template` • `runner_registration_enabled` • `show_diff_preview_in_email` • `selective_code_owner_removals`
`protected_branch_created`	Protected branch created	A protected branch was created.	
`protected_branch_deleted`	Protected branch was deleted	A protected branch was deleted.	
`protected_branch_updated`	Protected branch was updated	Protected branch rules were updated.	
`protected_tag_created`	Protected tag created	A protected tag was created.	
`protected_tag_deleted`	Protected tag was deleted	A protected tag was deleted.	

Name	System message	Purpose	Audited attributes
`protected_tag_updated`	Protected tag updated	Protected tag rules were updated.	
`removed_gpg_key`	Removed gpg key from user	A user's GPG key was removed.	
`removed_ssh_key`	User removed ssh key	A user's SSH key was removed.	
`requested_password_reset`	User requested password change	A user requested a password reset.	
`revoked_gpg_key`	Revoked gpg key from user	A user's GPG key was revoked.	
`unban_user`	User was unban	A user was unbanned.	
`unblock_user`	User was unblocked	A user was unblocked.	
`user_access_locked`	User access locked	A user account was locked.	
`user_access_unlocked`	User access unlocked	A user account was unlocked.	
`user_activated`	User was activated	A user account was activated.	
`user_banned`	User was banned	A user account was banned.	
`user_blocked`	User was blocked	A user account was blocked.	
`user_created`	User was created	A new user account was created.	
`user_deactivated`	User was deactivated	A user account was deactivated.	
`user_destroyed`	User was destroyed	A user account was deleted.	
`user_email_updated`	User email updated	A user's email address was updated.	

Name	System message	Purpose	Audited attributes
`user_logged_in`	User logged in	A user logged in successfully.	
`user_password_updated`	Password updated	A user's password was changed.	
`user_rejected`	User was rejected	A user account was rejected (for example, during a registration).	
`user_removed_two_factor`	Two factor disabled	A user disabled two-factor authentication.	
`user_settings_updated`	User settings updated	A user updated their profile settings.	• `name` • `public_email` • `otp_secret` • `otp_required_for_login` • `admin` • `private_profile`
`user_signup`	User was registered	A new user registered.	
`user_switched_to_admin_mode`	User switched to admin mode	A user switched to admin mode.	
`user_username_updated`	Username updated	A user's username was updated.	
`webhook_created`	Webhook was created	A webhook for a project, group, or instance was created.	
`webhook_destroyed`	System hook removed	A webhook was removed.	
`group_deleted`	Group was deleted	A group was deleted.	
`project_deleted`	Project was deleted	A project was deleted.	
`logout`	User logged out	A user logged out of the system.	

Name	System message	Purpose	Audited attributes
`unauthenticated_session`	Redirected to login	An unauthenticated user was redirected to the login page.	
`ci_runners_bulk_deleted`	CI runner bulk deleted: Errors:	Multiple CI runners were deleted.	
`ci_runner_registered`	CI runner created via API	A CI runner was registered via API.	
`ci_runner_unregistered`	CI runner unregistered	A CI runner was unregistered.	
`ci_runner_token_reset`	CI runner registration token reset	A CI runner's registration token was reset.	
`ci_runner_assigned_to_project`	CI runner assigned to project	A CI runner was assigned to a project.	
`ci_runner_unassigned_from_project`	CI runner unassigned from project	A CI runner was unassigned from a project.	
`ci_runner_created`	CI runner created via UI	A CI runner was created via the user interface.	
`package_registry_package_published`	`#{name}` package version `#{version}` has been published	A new package was published to the package registry.	
`package_registry_package_deleted`	package version `#{package.version}` has been deleted	A package was deleted from the package registry.	
`group_access_token_destroyed`	Group access token destroyed	A group access token was destroyed.	
`group_access_token_revoked`	Group access token revoked	A group access token was revoked.	
`personal_access_token_destroyed`	Personal access token destroyed	A personal access token was destroyed.	

Name	System message	Purpose	Audited attributes
`project_access_token_destroyed`	Project access token destroyed	A project access token was destroyed.	
`project_access_token_revoked`	Project access token revoked	A project access token was revoked.	
`approval_rule_created`	Merge request\Project\Group approval rule created	A merge request, project, or group approval rule was created.	
`approval_rule_updated`	Merge request\Project\Group approval rule updated	A merge request, project, or group approval rule was updated.	
`approval_rule_deleted`	Merge request\Project\Group approval rule deleted	A merge request, project, or group approval rule was deleted.	
`approval_rule_propagated`	Approval rules `#{rule_names}` propagated to `#{project_and_group_names}`	Approval rules were propagated to these projects and groups.	
`variable_viewed_api`	User viewed all CI/CD variables via API for this scope or User viewed CI/CD variable `#{variable.key}` via API	All CI/CD variables were viewed via API for this scope or A CI/CD variable `#{variable.key}` was viewed via API.	
`group_unarchived`	User unarchived group	A group was unarchived.	
`group_archived`	User archived group	A group was archived.	
`comment_deleted`	Comment deleted	A comment was deleted.	
`comment_created`	Comment created	A comment was created.	

Name	System message	Purpose	Audited attributes
`comment_updated`	Comment updated	A comment was updated.	
`user_records_migrated_to_ghost`	User records migrated to ghost	User records were migrated to a ghost user.	
`user_enable_passkey`	User enable new passkey	A new passkey was enabled.	
`user_disable_passkey`	User disable passkey	A passkey was disabled.	
`push_rule_propagated`	Push rule propagated	A push rule was propagated.	
`push_rule_updated`	Push rule updated	An existing push rule was updated.	

Monitoring and limits

Monitoring of a Deckhouse Code instance consists of the limits set in the web interface and of the built-in Prometheus of each installation type.

Resource management

The limits and the performance settings of the instance are set by an administrator in the web interface:

1. Configure repository and artifact limits:
    - Go to “Admin” → “Settings” → “Account and Limits”.
2. Optimize system performance:
    - Go to “Admin” → “Settings” → “Network”.

Built-in monitoring

Each installation type ships its own metrics endpoint or alerting; the tab names the type.

Linux package

Prometheus and the exporters (node, postgres, redis and others) are installed together with the package and listen on the `127.0.0.1` address only, so no port is open to the outside. Check their state:

```
sudo gitlab-ctl status | grep -E "prometheus|exporter"
# The readiness of the built-in Prometheus.
curl -s http://127.0.0.1:9090/-/ready
```

To let an external monitoring system collect the metrics, set the listen address of Prometheus in the `/etc/gitlab/gitlab.rb` file:

```
prometheus['listen_address'] = '0.0.0.0:9090'
```

Apply the setting:

```
sudo gitlab-ctl reconfigure
```

Restrict access to port `9090` on the firewall to the addresses of the monitoring system.

firewalld (RED OS)

```
sudo firewall-cmd --permanent --new-zone=monitoring
sudo firewall-cmd --permanent --zone=monitoring --add-source=<MONITORING_IP>/32
sudo firewall-cmd --permanent --zone=monitoring --add-port=9090/tcp
# The zone takes all traffic from this address, so the base services are opened
as well.
sudo firewall-cmd --permanent --zone=monitoring --add-service={ssh,http,https}
sudo firewall-cmd --reload
```

ufw (Ubuntu)

```
sudo ufw allow from <MONITORING_IP> to any port 9090 proto tcp
```

Omnibus Docker

The container runs the same Prometheus and the same exporters except `node_exporter`, which the image turns off. The image exposes ports `80`, `443` and `22` only, so the Prometheus port has to be published when the container is created: add `-p 9090:9090` to the `docker run` command, as the list of published ports of a running container cannot be changed.

Set `prometheus['listen_address']` in the `/etc/gitlab/gitlab.rb` file in the configuration volume; the value, its meaning and the access restriction on the port are described on the “Linux package” tab. The setting is applied at the next start of the container: the `gitlab-ctl reconfigure` command runs at every start.

Helm Chart

The Helm Chart creates a ServiceMonitor object (the Prometheus Operator custom resource) for each component that exposes Prometheus metrics, when `metrics.serviceMonitor.enabled` is `true`. `metrics.serviceMonitor.labels` sets the labels a Prometheus instance matches to discover them.

```
metrics:
  serviceMonitor:
    enabled: true
    labels:
      release: <PROMETHEUS_RELEASE_LABEL>
```

A Prometheus instance configured to watch ServiceMonitor objects with these labels, in the `code` namespace or across the cluster depending on its own configuration, discovers and scrapes them without further setup on the release side.

Deckhouse Kubernetes Platform module

The module ships its own alerting rules: the list of alerts and the action for each of them are described in the Alerts section of the module documentation.

Maintenance mode

Maintenance mode allows administrators to reduce write operations to a minimum while maintenance tasks are performed. The main goal is to block all external actions that change the internal state of the PostgreSQL database, as well as files, Git repositories, and container image registries.

When maintenance mode is enabled, in-progress actions finish shortly because no new actions are coming in, and internal state changes are minimal. In that state, various maintenance tasks are easier.

Maintenance mode allows most external actions that do not change the internal state. At the HTTP level, requests with `POST`, `PUT`, `PATCH`, and `DELETE` methods are blocked.

Enabling maintenance mode

Enable maintenance mode as an administrator in one of these ways:

- **Via Web UI**:
  1. Go to “Admin” → “Settings” → “General”.
  2. Expand the “Maintenance mode” section, and select the “Enable maintenance mode” checkbox.
  3. If necessary, add a message to be displayed in the banner.
  4. Select “Save changes”.
- **Via API**:

  Run the following request:

```
curl --request PUT --header "PRIVATE-TOKEN: $ADMIN_TOKEN" \
  "<INSTANCE_URL>/api/v4/application/settings?maintenance_mode=true"
```

Disabling maintenance mode

Disable maintenance mode in one of these ways:

- **Via Web UI**:
  1. Go to “Admin” → “Settings” → “General”.
  2. Expand the “Maintenance mode” section and clear the “Enable maintenance mode” checkbox.
  3. Select “Save changes”.
- **Via API**:

  Run the following request:

```
curl --request PUT --header "PRIVATE-TOKEN: $ADMIN_TOKEN" \
  "<INSTANCE_URL>/api/v4/application/settings?maintenance_mode=false"
```

Maintenance mode considerations

When maintenance mode is enabled, a banner is displayed at the top of the interface. If necessary, the banner message can be customized.

When trying to perform an unallowed write operation, the user gets an error message.

> In some cases, visual feedback from an action might be misleading. For example, when starring a project, the “Star” button changes to “Unstar”. This is due to the UI being updated prior to obtaining the result of the `POST` request.

Administrator actions

Administrators can still edit the application settings. This allows them to disable maintenance mode after it’s been enabled.

Authentication

All users can sign in and out of the system, but the new user creation is blocked.

If an LDAP synchronization is scheduled during the maintenance, it will fail because user creation is disabled. Similarly, a new user creation based on SAML and other OmniAuth providers will fail as well.

> When user creation is blocked during an LDAP sign-in, the user sees an error message indicating that the instance is in read-only (maintenance) mode, rather than a generic message about the access being denied.

Git actions

All read-only Git operations continue to work, including `git clone` and `git pull`.

All write operations fail, both through the CLI and the Web IDE, accompanied by the following message:

```
Git push is not allowed because this instance is currently in (read-only) maintenance mode.
```

Merge requests and issues

All write actions besides the exceptions fail. For example, a user cannot edit merge requests or issues.

Incoming email

Creating new issue replies, issues (including new Service Desk issues), and merge requests by email fails. The incoming mail workers skip processing while maintenance mode is enabled.

Outgoing email

Notification emails continue to arrive, but emails that require database writes, like resetting the password, do not arrive.

REST API

For most JSON requests, the `POST`, `PUT`, `PATCH`, and `DELETE` methods are blocked. The API returns a `503` response with the error message `system is in maintenance mode` and a `Retry-After` header.

Only the following requests are allowed:

HTTP request	Allowed routes	Notes
`POST`	`/admin/application_settings/general`	Updating application settings in the administrator UI
`PUT`	`/api/v4/application/settings`	Updating application settings via the API
`POST`	`/users/sign_in`	Sign in for users
`POST`	`/users/sign_out`	Sign out for users
`POST`	`/oauth/token`	Obtaining OAuth tokens
`POST`	`/admin/session`, `/admin/session/destroy`	Using Admin Mode for administrators
`POST`	Paths ending with `/compare`	Git revision routes
`POST`	`*.git/git-upload-pack`	Running `git pull` and `git clone`
`POST`	`/api/v4/internal`	Internal API routes
`POST`	`/admin/sidekiq`	Management of background jobs in the “Admin” area

GraphQL API

The `POST /api/graphql` requests are allowed, but GraphQL mutations are blocked with the following error message: `You cannot perform write operations on a read-only instance`.

Continuous integration

During the maintenance, the following restrictions apply:

- No new jobs or pipelines start, scheduled or otherwise.
- Jobs that were already running when the maintenance was enabled continue to have a `running` status in the UI, even if they finish running on the runner.
- Jobs in the `running` state do not time out if the project’s time limit is exceeded.
- Pipelines cannot be started, retried, or canceled. No new jobs in the existing pipelines can be created either.
- The status of the runners in the “Admin” → “Runners” section isn’t updated.

After maintenance mode is disabled, new jobs are picked up again.

Jobs that were in the `running` state before enabling maintenance mode resume, and their logs start updating again.

> When the maintenance mode is disabled, it is recommended that you restart pipelines that were in the `running` state when the maintenance was enabled.

Deployments

Deployments don't go through because associated pipelines can't be finished. Disable automated deployments during maintenance mode, and enable them when it is disabled.

Container registry

The `docker push` command fails with the following error message:

```
denied: requested access to the resource is denied
```

However, the `docker pull` command still works.

Package registry

The package registry allows you to install but not publish packages.

Background jobs

Background jobs (including cron and Sidekiq jobs) continue running as is, because background jobs are not automatically disabled.

As background jobs perform operations that can change the internal state of your instance, you may want to disable some or all of them while maintenance mode is enabled.

To monitor queues and disable jobs:

1. Go to "Admin" → "Monitoring" → "Background jobs".
2. In the Sidekiq dashboard, select "Cron" and disable jobs individually or all at once by selecting "Disable All".

Incident management

Incident management functions are limited. The creation of alerts and incidents is paused entirely, while associated notifications are disabled.

Feature flags

- Development feature flags cannot be turned on or off through the API, but can be toggled through the Rails console.
- The feature flag service responds to feature flag checks, but feature flags cannot be toggled.

Advanced search

Administrator documentation for advanced search in Deckhouse Code: operating indexing after OpenSearch is connected. In the Deckhouse Kubernetes Platform module, OpenSearch is connected from the CodeInstance resource, described in Global search in the module documentation. For user instructions, see the user guide.

Operations

Manage indexing, monitoring, and troubleshooting.

Instance-level management

Go to “Admin” → “Settings” → “Search”.

This section is available after OpenSearch is connected.

Pause indexing

Enable “Pause OpenSearch indexing” to pause background indexing and reindexing jobs. To enable indexing again, disable the “Pause OpenSearch indexing” flag. After removing the pause, Sidekiq pause control resumes jobs automatically within a few minutes.

Branch indexing mode

The following branch indexing modes can be used in projects:

Mode	Description
Default branch only	Only the default branch is indexed for all projects
Allow per-project branch regex	For projects, you can configure a regex to index additional branches

> ⚠ After changing the branch indexing mode, manually enqueue a code reindex. Search results may be incomplete until indexing completes.

OpenSearch index status

The page displays a table of indices:

Index suffix	Search scope
`code`	Code (`blobs`)
`commits`	Commits
`wiki`	Wiki pages
`notes`	Comments
`milestones`	Milestones
`merge-requests`	Merge requests
`work-items`	Work items

An OpenSearch index name consists of the shared instance prefix and the suffix from the table, for example `deckhouse-development-code` .

For each index, the table shows the OpenSearch index name, presence, document count, and index health.

Index operations

The following operations are available on the same page:

- **Reindex** — reindex a single index (removes existing documents and enqueues background jobs).
- **Reindex all indices** — reindex all indices.

> ⚠ The **Reindex** operation removes existing documents. Search results may be incomplete until background indexing catches up.

The `commits` index is reindexed together with the `code` index: there is no separate operation for commits. For the same reason, commits have no dedicated `schema_class` value.

You can also perform operations on indices using queries to the OpenSearch API.

Monitoring

Metrics

OpenSearch requests are tracked in Prometheus separately for HTTP requests (search in the UI and API) and for Sidekiq background jobs (indexing). Metric names contain `elasticsearch` — a historical GitLab name; the metrics refer to OpenSearch.

HTTP requests (user search):

Metric	Description
`http_elasticsearch_requests_total`	Number of OpenSearch requests per HTTP request
`http_elasticsearch_requests_duration_seconds`	Total OpenSearch request time per HTTP request
`http_elasticsearch_requests_failed_total`	Failed OpenSearch requests per HTTP request (connection or authorization errors) — **added in Deckhouse Code**

Sidekiq (background indexing):

Metric	Description
`sidekiq_elasticsearch_requests_total`	Number of OpenSearch requests per Sidekiq job
`sidekiq_elasticsearch_requests_duration_seconds`	Total OpenSearch request time per Sidekiq job
`sidekiq_elasticsearch_requests_failed_total`	Failed OpenSearch requests per Sidekiq job (connection or authorization errors) — **added in Deckhouse Code**

Repository indexer (`Search::RepositoryIndexerWorker` — code, commits, wiki):

Metric	Labels	Description
`search_repository_indexer_starts_total`	`indexer_class`	Indexing runs that started after the `advanced_search_enabled` check
`search_repository_indexer_runs_total`	`outcome` , `indexer_class`	Completed runs after obtaining an exclusive lock (`outcome` : `success` or `error`)
`search_repository_indexer_duration_seconds`	`outcome` , `indexer_class`	Duration of the indexing phase under exclusive lock
`search_repository_indexer_lock_contention_total`	—	Times the lock was not obtained and the job was rescheduled

The `indexer_class` label indicates the indexing type:

`indexer_class`	When used
`Search::RepositoryIndexer::IncrementalIndexService`	Incremental indexing after repository changes
`Search::RepositoryIndexer::FullIndexService`	Full reindex (`force: true`)
`Search::RepositoryIndexer::MaintainsService`	Index update triggered by an event
`Search::RepositoryIndexer::DeleteService`	Remove documents from the index (empty or deleted repository/wiki)

An increase in `search_repository_indexer_lock_contention_total` indicates lock contention between jobs for the same project. An increase in `search_repository_indexer_runs_total{outcome="error"}` indicates go-indexer or indexing service errors; see Sidekiq logs for details.

The `*_failed_total` metrics increase on OpenSearch connection or authorization errors. An increase in `*_failed_total` indicates OpenSearch is unavailable or credentials are invalid. An increase in `*_duration_seconds` with a stable `*_total` indicates slow OpenSearch responses.

For repository indexing, use `search_repository_indexer_*` ; for OpenSearch requests from Sidekiq, use `sidekiq_elasticsearch_*` . For user search, use `http_elasticsearch_*` .

The indexing progress widget on “Admin” → “Settings” → “Search” shows the number of remaining full reindex jobs. The same data is available through the `indexing_queue_stats` endpoint.

Sidekiq queue

OpenSearch indexing jobs run in the dedicated `global-search-indexing` queue, not in the shared `default` queue. Routing is configured with a Sidekiq rule: all workers with the `fe_global_search` category go to this queue. A separate queue isolates indexing load from other Deckhouse Code background jobs.

Cron jobs

The following cron jobs are regularly performed to automate indexing:

Schedule	Purpose
Every minute	Comment indexing — processes the accumulated notes change queue
Daily at 03:00	Enqueues project indexing

Cron jobs do not index directly: they start or resume the corresponding workers in the `global-search-indexing` queue.

Logs

OpenSearch indexing jobs are written to Sidekiq logs. Filter by queue name `global-search-indexing`.

Troubleshooting

OpenSearch is unavailable

- The “Admin” → “Settings” → “Search” page displays a connection failure message.
- Search returns an error.

The address and the credentials of OpenSearch are set in the configuration of the installation type, not in the interface:

Linux package

The `gitlab_rails['fe_search']` key of `/etc/gitlab/gitlab.rb`, applied with `sudo gitlab-ctl reconfigure`.

Omnibus Docker

The same key of `/etc/gitlab/gitlab.rb` on the configuration volume, applied with `docker exec code gitlab-ctl reconfigure`.

Helm Chart

The `global.appConfig.search` values of the release, applied with `helm upgrade`.

Deckhouse Kubernetes Platform module

The CodeInstance resource, described in Global search in the module documentation.

Incomplete search results

- Wait for background indexing to complete (progress widget on “Admin” → “Settings” → “Search”).
- Run reindexing at the project level or “Reindex” for the required index in “Admin” → “Settings” → “Search”.

Indexing jobs are not appearing

If new jobs are not enqueued to the `global-search-indexing` queue:

1. Check whether “Pause OpenSearch indexing” is enabled in “Admin” → “Settings” → “Search”. Clear the flag and wait for jobs to resume (the cron job runs every 5 minutes).
2. If the pause is cleared but jobs still do not appear, run Redis cleanup — stuck leases or Sidekiq duplicate keys are possible:

Linux package

```
sudo gitlab-rails runner /opt/gitlab/embedded/service/gitlab-rails/fe/
scripts/clear_search_opensearch_worker_redis.rb
```

Omnibus Docker

```
docker exec code gitlab-rails runner /opt/gitlab/embedded/service/gitlab-
rails/fe/scripts/clear_search_opensearch_worker_redis.rb
```

Helm Chart

```
d8 k -n code exec deploy/code-toolbox -- gitlab-rails runner /srv/gitlab/fe/
scripts/clear_search_opensearch_worker_redis.rb
```

Deckhouse Kubernetes Platform module

```
d8 k -n d8-code exec -it -c toolbox deploy/toolbox -- gitlab-rails runner /
srv/gitlab/fe/scripts/clear_search_opensearch_worker_redis.rb
```

The script clears the exclusive lease for `Search::RepositoryIndexerWorker`, concurrency limits, and dedup keys for the `global-search-indexing` queue.

OpenSearch API

This section documents Deckhouse Code admin OpenSearch endpoints. For user-facing search parameters, see “Search API”.

POST /api/v4/admin/opensearch/recreate_indices

Synchronously recreates OpenSearch index(es) and enqueues background reindex jobs. To rebuild (reindex) all indexes, run the query without a body. To rebuild a specific index, specify it in the query body.

Access rights: admin only (`authenticated_as_admin!`).

Request body

Field	Type	Required	Allowed values
`schema_class`	string	Yes	`recreate_all`, `Search::Opensearch::IndicesSchema::Code`, `Search::Opensearch::IndicesSchema::Wiki`, `Search::Opensearch::IndicesSchema::Note`, `Search::Opensearch::IndicesSchema::Milestone`, `Search::Opensearch::IndicesSchema::WorkItem`, `Search::Opensearch::IndicesSchema::MergeRequest`

Responses

- `202 Accepted`

```
{
  "message": "OpenSearch indices were reset; reindex jobs were enqueued."
}
```

- `400 Bad Request` (for example, if OpenSearch is disabled or there is a service error)

```
{
  "message": "OpenSearch is disabled"
}
```

Request example

This query rebuilds all indexes:

```
curl --request POST \
  --header "PRIVATE-TOKEN: <ACCESS_TOKEN>" \
  --header "Content-Type: application/json" \
  --data '{"schema_class":"recreate_all"}' \
  --url "https://gitlab.example.com/api/v4/admin/opensearch/recreate_indices"
```

GET /api/v4/admin/opensearch/indexing_queue_stats

Returns Sidekiq queue stats for OpenSearch indexing.

Access rights: authenticated user with permission `read_admin_search_indexing_queue_stats` on `:global`.

Response (200 OK)

```
{
  "total": 42,
  "updated_at": "2026-07-01T12:34:56.789Z"
}
```

Response fields:

- `total`: Total number of indexing jobs in the queue.
- `updated_at`: ISO8601 timestamp with milliseconds (or `null`).

Request example

```
curl --request GET \
  --header "PRIVATE-TOKEN: <ACCESS_TOKEN>" \
  --url "https://gitlab.example.com/api/v4/admin/opensearch/indexing_queue_stats"
```

Backup and restore

A Deckhouse Code backup consists of an archive with the instance data and of files that the archive does not contain. The contents of the archive are the same for the Linux package and for Omnibus Docker: both deliveries run on the same engine, and the archive in them is created by the `gitlab-backup` command. In a container the same commands run through `docker exec`, and the archive is written to the data volume.

What the backup contains

The `gitlab-backup create` command collects into a single archive:

- Repositories
- The database
- Uploads
- CI artifacts
- LFS objects
- The package registry

What is stored separately

The archive does not contain these files, so copy them along with every archive:

- `/etc/gitlab/gitlab-secrets.json`: Encryption keys of the database. Without this file the restore is incomplete: two-factor authentication settings, tokens and encrypted CI variables are lost.
- `/etc/gitlab/gitlab.rb`: The instance configuration.
- Certificates and keys from the `/etc/gitlab/ssl/` directory.

Store the secret files with the same level of protection as passwords.

Linux package

The files are in the `/etc/gitlab/` directory on the instance machine. Copy them together with every archive.

Omnibus Docker

The files are on the configuration volume. Archive them with a command of its own:

```
docker exec -t code gitlab-ctl backup-etc
```

The command writes `/etc/gitlab/config_backup/gitlab_config_<TIMESTAMP>_<DATE>.tar` with the content of `/etc/gitlab`: `gitlab.rb`, `gitlab-secrets.json` and the TLS certificates. The directory is on the configuration volume, `/srv/code/config/config_backup` on the host.

Creating a backup

Linux package

A backup is taken with a single command:

```
sudo gitlab-backup create
```

The archive appears in the `/var/opt/gitlab/backups/` directory under a name of the form `<TIMESTAMP>_<DATE>_<ENGINE_VERSION>_gitlab_backup.tar`. The version in the name is the version of the built-in engine, which differs from the version of the `deckhouse-code` package, so record the package version next to the archive when you take a backup:

```
# RED OS.
rpm -q deckhouse-code
# Ubuntu.
dpkg -l deckhouse-code
```

Omnibus Docker

Run the backup in the running container:

```
docker exec -t code gitlab-backup create
```

The archive appears in `/var/opt/gitlab/backups` inside the container, which is `/srv/code/data/backups` on the host with the volumes from Quick start. Copy it off the host together with the archive of the configuration and the secrets.

Helm Chart

The `backup-utility` utility runs inside the toolbox pod the Helm Chart deploys:

```
d8 k -n code exec deploy/code-toolbox -- backup-utility
```

The command creates a backup archive and uploads it to the object storage bucket configured through `global.appConfig.object_store` (Object storage).

Archive retention and schedule

Archives older than `backup_keep_time` are deleted when the next backup is created.

Linux package

The retention time is set in the `/etc/gitlab/gitlab.rb` file in seconds:

```
# Seven days.
gitlab_rails['backup_keep_time'] = 604800
```

The change takes effect after `sudo gitlab-ctl reconfigure`.

Regular backups are set up with a cron job, for example daily at 02:00:

```
echo '0 2 * * * root /opt/gitlab/bin/gitlab-backup create CRON=1' \
  | sudo tee /etc/cron.d/deckhouse-code-backup
```

Omnibus Docker

How long archives are kept is set by `gitlab_rails['backup_keep_time']` in the configuration on the configuration volume; its value and behaviour are described in the Linux package tab.

For a schedule, run the same command from the host `cron`:

```
echo '0 2 * * * root /usr/bin/docker exec -t code gitlab-backup create CRON=1' \
  | sudo tee /etc/cron.d/deckhouse-code-backup
```

Copying off the host

Copy fresh archives to a separate server regularly.

Linux package

Move the archives from the `/var/opt/gitlab/backups/` directory, for example with the `rsync` command. Copy the `/etc/gitlab/gitlab-secrets.json` and `/etc/gitlab/gitlab.rb` files together with the archive: without them the archive is not restored completely.

Omnibus Docker

Move the archives from the `/srv/code/data/backups` directory on the host together with the configuration archive from `/srv/code/config/config_backup`: without the configuration and the secrets the archive is not restored completely.

Restoring

Restoring replaces the instance data with the contents of the archive. Before you start, check that the conditions are met:

- The installed version is the version the backup was taken on.
- The `gitlab.rb` and `gitlab-secrets.json` files of the source instance are in place. When moving to a new server, copy them before the restore and run the configuration.
- The certificates and keys from the `/etc/gitlab/ssl/` directory are in place: `gitlab.rb` refers to them by path, and without these files the configuration run fails to start nginx.
- There is enough free space for the unpacked archive.

In the commands of the Linux package and Omnibus Docker, `<BACKUP_NAME>` is the archive file name without the `_gitlab_backup.tar` suffix: for the `1784800000_2026_07_23_19.1.2_gitlab_backup.tar` archive it is `1784800000_2026_07_23_19.1.2`.

Linux package

Check the installed package version with `rpm -q deckhouse-code` on RED OS or `dpkg -l deckhouse-code` on Ubuntu. When the configuration and the secrets come from another machine, run `sudo gitlab-ctl reconfigure` after copying them.

1. Copy the archive to the backup directory and hand the file over to the `git` user:

   ```
   sudo cp <BACKUP_NAME>_gitlab_backup.tar /var/opt/gitlab/backups/
   sudo chown git:git /var/opt/gitlab/backups/<BACKUP_NAME>_gitlab_backup.tar
   ```

2. Stop the services that write to the database; the other services keep running:

   ```
   sudo gitlab-ctl stop puma
   sudo gitlab-ctl stop sidekiq
   ```

3. Restore the data from the archive:

   ```
   sudo gitlab-backup restore BACKUP=<BACKUP_NAME>
   ```

   The command asks for confirmation twice: before recreating the `authorized_keys` file and before dropping the current database tables. Answer `yes` both times.

4. Restart the instance and run the self-check:

   ```
   sudo gitlab-ctl reconfigure
   sudo gitlab-ctl restart
   sudo gitlab-rake gitlab:check SANITIZE=true
   ```

Omnibus Docker

Check that the container runs the image tag of the version the backup was taken on and that the configuration and the secrets of the source instance are on the configuration volume.

1. Copy the archive into the backup directory on the data volume and give it the owner the instance uses:

   ```
   sudo cp <BACKUP_NAME>_gitlab_backup.tar /srv/code/data/backups/
   docker exec code chown git:git /var/opt/gitlab/backups/
   <BACKUP_NAME>_gitlab_backup.tar
   ```

2. Stop the services that write to the database:

   ```
   docker exec code gitlab-ctl stop puma
   docker exec code gitlab-ctl stop sidekiq
   ```

3. Start the restore. The command asks for confirmation twice, before the `authorized_keys` file is rebuilt and before the current database tables are dropped; answer `yes` both times:

   ```
   docker exec -it code gitlab-backup restore BACKUP=<BACKUP_NAME>
   ```

4. Apply the configuration and restart the container:

   ```
   docker exec code gitlab-ctl reconfigure
   docker restart code
   ```

5. Check the instance:

   ```
   docker exec code gitlab-ctl status
   docker exec code gitlab-rake gitlab:check SANITIZE=true
   ```

Helm Chart

1. Put the instance in maintenance mode.
2. Confirm the object storage bucket configured through `global.appConfig.object_store` holds the archive you want to restore.
3. Run the restore command inside the toolbox pod:

   ```
   d8 k -n code exec deploy/code-toolbox -- backup-utility restore --backup-id <BACKUP_ID>
   ```

4. Wait for the command to finish and confirm every pod is `Running`:

   ```
   d8 k -n code get pods
   ```

5. Take the instance out of maintenance mode.

After the restore, sign in to the web interface and clone a repository to check that the instance works with its own data.

Deckhouse Kubernetes Platform module

In the module delivery, backups are created by the operator on the schedule from the CodeInstance resource and uploaded to object storage: the contents of the archive, the restore preconditions and the command matrix are described in the Backups and restore section of the module documentation.

Upgrade

An upgrade of Deckhouse Code installs a new version over the current one: the configuration and the data are kept, and the configuration run and the database migrations start automatically. The rules below apply to every installation type.

Version order

Upgrade sequentially, without skipping major versions: the database migrations are designed for the transition from the previous major version. If several major releases lie between the current and the target version, install them one by one and wait for the background migrations to finish between the steps. Ask the vendor for the upgrade path for your pair of versions when you obtain the new version.

Background migrations

An upgrade applies the main database migrations and does not wait for the background ones: they keep running on the instance that is already serving users. All background migrations must finish before the next upgrade, otherwise the upgrade stops with an error.

To check the state of the migrations, select “Admin” in the top right corner, then go to “Monitoring” → “Background migrations” in the left pane. The “Queued” and “Finalizing” tabs must have no jobs left and the “Failed” tab must be empty: every migration is in the “Finished” status. On a typical instance this takes minutes, and longer on large amounts of data.

Before the upgrade

Linux package

Create a backup as described in Backup and restore and check that fresh copies of the `/etc/gitlab/gitlab-secrets.json` and `/etc/gitlab/gitlab.rb` files exist:

```
sudo gitlab-backup create
```

Omnibus Docker

Create a backup of the data and of the configuration, as described in Backup and restore, and check that the archives are outside the host.

Write down the tag the container runs, because the rollback returns to it:

```
docker inspect --format '{{.Config.Image}}' code
```

Helm Chart

Every chart release states the Deckhouse Code version it deploys. Check it before upgrading:

```
helm show chart deckhouse-code --version <CHART_VERSION>
```

Create a backup as described in Backup and restore: the rollback of a schema change needs it.

Upgrading

Linux package

The configuration in `/etc/gitlab/gitlab.rb` and the data are kept.

1. Install the new package; the configuration run and the migrations start automatically:

```
# RED OS.
sudo rpm -Uvh ./deckhouse-code-<NEW_VERSION>.el8.x86_64.rpm
# Ubuntu.
sudo apt install -y ./deckhouse-code_<NEW_VERSION>_amd64.deb
```

2. Check the state of the instance and the installed version:

```
# Every service is in the run state.
sudo gitlab-ctl status
# The installed version on RED OS.
rpm -q deckhouse-code
# The installed version on Ubuntu.
dpkg -l deckhouse-code
# health=200 is expected.
curl --cacert /etc/gitlab/ssl/<HOSTNAME>.crt \
     --resolve '<HOSTNAME>:443:127.0.0.1' -o /dev/null \
     -w "health=%{http_code}\n" https://<HOSTNAME>/-/health
```

While the migrations run, the instance answers more slowly; on a typical instance the upgrade takes minutes.

Omnibus Docker

The container is replaced with one created from the new image tag; the volumes stay, and the instance data and configuration on them are migrated on the first start. Every tag names a version, and no tag follows the latest one, so the target version is named explicitly.

1. Pull the image of the target version:

   ```
   docker pull <REGISTRY>/<FLAVOR>:<VERSION>
   ```

2. Stop and remove the container. The volumes are host directories and stay in place:

   ```
   docker stop code
   docker rm code
   ```

3. Create the container from the new tag with the same name, ports, volumes and environment variables as in Quick start:

   ```
   docker run -d --name code \
     --shm-size 256m \
     -p 80:80 -p 443:443 -p 22:22 \
     -e GITLAB_OMNIBUS_CONFIG="external_url 'http://<HOSTNAME>'" \
     -v /srv/code/config:/etc/gitlab \
     -v /srv/code/logs:/var/log/gitlab \
     -v /srv/code/data:/var/opt/gitlab \
     <REGISTRY>/<FLAVOR>:<VERSION>
   ```

4. Follow the start. The database migrations run during the configuration, and the instance answers 502 until it finishes:

   ```
   docker logs -f code
   ```

5. Check the version and the services:

   ```
   docker logs code | grep 'Current version'
   docker exec code gitlab-ctl status
   docker inspect --format '{{.State.Health.Status}}' code
   ```

The start-up script reads the version of the data on the volume and compares it with the version in the image. When the instance cannot move to the new version in one step, the output names the version to install first and the start stops: run the tag of that version, wait for the background migrations to finish, then run the target tag.

After the configuration, the script brings the database to the PostgreSQL version of the new image. A failed database upgrade is rolled back, and the container exits with the message `Upgrading the existing database failed and was reverted.` Create the container with `-e GITLAB_SKIP_PG_UPGRADE=true` to bring the instance up on the current PostgreSQL version, and upgrade the database separately.

Helm Chart

Upgrade the release:

```
helm repo update
helm upgrade code deckhouse-code \
  -n code \
  -f values.yaml \
  --version <CHART_VERSION>
```

The command reuses the existing `values.yaml`. Follow the version order when the mapping spans more than one Deckhouse Code version; skipping a version can leave a later migration unable to run.

A migrations job runs once per upgrade, before the components that depend on the new database schema restart. Check its progress:

```
d8 k -n code get jobs
d8 k -n code logs job/<MIGRATIONS_JOB_NAME>
```

Background migrations continue on the running instance after the upgrade and have to finish before the next one.

Rollback

A rollback is only possible from a backup taken before the upgrade: the new version has migrated the data.

Linux package

Install the package version on which the backup was taken and restore the data as described in Restoring. Ask the vendor for the exact downgrade procedure for your configuration.

Omnibus Docker

1. Stop and remove the container:

   ```
   docker stop code
   docker rm code
   ```

2. Create the container from the previous tag with the same volumes.
3. Restore the data from the backup taken before the upgrade, as described in Restoring.

Helm Chart

Roll the release back to the previous revision:

```
helm rollback code <REVISION>
```

> ⚠ `helm rollback` reverts the release's Kubernetes objects only. It does not reverse a database schema change made by the upgrade's migrations.

When the upgrade you are rolling back changed the schema, restore the backup taken before the upgrade after the rollback completes, as described in Restoring.

Moving between the package, Docker and the module

Moving an instance from the Linux package or Omnibus Docker to the Deckhouse Kubernetes Platform module and back is described in the Migration section of the module documentation.

Deckhouse Kubernetes Platform module

In the module delivery, the upgrade is started by the platform. With `backup.enabled` and `backup.backupBeforeUpdate` set in the CodeInstance resource, the operator creates a backup first and updates the remaining components after the backup job has finished successfully; a failed job postpones the update and raises the `D8CodeOperatorUpdatePostpone` alert. The order is described in the Automatic backup creation before module updates section of the module documentation.

Usage

Overview

Deckhouse Code is where a team keeps its source code, reviews changes, and runs CI/CD pipelines. The Usage section covers the everyday work of a project member or a group owner, from the first sign-in to searching across the instance.

Sign in to the web interface with your account credentials or through an external identity provider, then clone a repository and push changes over SSH or HTTPS. See Signing in.

A group brings projects together under a shared role model and shared policies such as protected branches and push rules; a project holds a repository, its issues, and its wiki. See Groups and members.

A repository stores the Git history of a project: branches, tags, and files edited through the web interface or Git LFS. Protected branches, push rules, and pull mirroring control how changes reach it. See Managing Git repositories.

A merge request combines changes from one branch into another through review: comments, approval rules, and external status checks decide when it can merge. See Code review.

A pipeline defined in `.gitlab-ci.yml` builds, tests, and deploys a project's code, with secrets available from an external Vault or Deckhouse Stronghold. See CI/CD pipelines.

Group owners and project maintainers apply security settings such as visibility levels, push rule enforcement, and merge checks to protect the codebase. See Group and project security settings.

Webhooks notify external systems about events in a group, project, or subgroup. See Webhooks.

Advanced search finds code, commits, issues, and merge requests across the projects you have access to. See Advanced search.

Getting started

Signing in

1. Open the Deckhouse Code web interface.

   Linux package

   The address is the `external_url` value set during installation. For details, see Quick start.

Omnibus Docker

The address is the `external_url` value set during installation. For details, see Quick start.

Deckhouse Kubernetes Platform module

Enter the address of the web interface in your browser's address bar. It is the hostname set for the web interface in the CodeInstance resource or, when none is set, the platform's global `publicDomainTemplate` with `%s` replaced by `code`: for the template `%s.example.com` the address is `code.example.com`.

Helm Chart

The address is set in the release values. For details, see Quick start.

2. Sign in with your account credentials.

   If the administrator configured an external identity provider, sign in through it instead. See Authentication.

Two-factor authentication

You can enable two-factor authentication for your account to require a second confirmation step at sign-in, in addition to your password. Deckhouse Code supports authenticator apps (for example, Google Authenticator or Authy) and hardware security keys (U2F, WebAuthn).

To enable two-factor authentication, go to "Profile" → "Account" → "Two-factor authentication" and follow the setup steps for the method you choose.

Working with Git

Access methods

Git operations from the command line authenticate with an SSH key or with a personal access token over HTTPS.

- SSH keys: Generate a key pair on your machine and add the public key in "Profile" → "SSH Keys". Use an SSH key to clone and push over a `git@` URL without entering a password on every request.
- HTTPS with a personal access token: Create a token in "Profile" → "Access Tokens" and use it as the password when Git prompts for credentials over an `https://` URL.

The administrator can restrict which of the two protocols are available on the instance. See Users and access.

Repository cloning

Cloning allows you to copy a remote Git repository to your local machine.
To clone, run the following command in the terminal:

```
git clone <REPOSITORY_URL>
```

Replace `<REPOSITORY_URL>` with the HTTPS or SSH URL of your repository.

Getting updates

To fetch the latest changes from the remote repository, navigate to the repository folder on your local machine and run:

```
git pull origin <BRANCH_NAME>
```

Replace `<BRANCH_NAME>` with the name of the branch, e.g., `main` or `feature/my-new-feature`.

Pushing changes

1. Make necessary changes in your codebase.
2. Commit the changes with the command:

   ```
   git commit -am "Description of your changes"
   ```

3. Push the changes to the remote repository:

   ```
   git push origin <BRANCH_NAME>
   ```

Replace `<BRANCH_NAME>` with the name of the branch to which you are pushing.

Creating a feature branch

Feature branches are used to develop new features or fixes. Below are the steps to create a feature branch and push it to Deckhouse Code:

1. Clone the repository:

   ```
   git clone <REPOSITORY_URL>
   ```

   Replace `<REPOSITORY_URL>` with the actual repository URL.

2. Change directory to the repository folder:

   ```
   cd <REPOSITORY_PATH>
   ```

3. Create a new feature branch:

   ```
   git checkout -b <BRANCH_NAME>
   ```

   Replace `<BRANCH_NAME>` with the name of your new branch. The `-b` flag tells Git to create the new branch and switch to it immediately.

4. Push the new branch to the remote repository:

   ```
   git push origin <BRANCH_NAME>
   ```

   Make sure the branch appears in the remote repository.
   To verify, open the project in Deckhouse Code and find the new branch in the branch selection menu.

Merge request

A merge request is used to combine changes from one branch into another.

Benefits of using merge requests:

1. Code quality control:
   - Allows code review before merging into the main branch.
   - Helps catch errors and maintain consistent coding style.
2. Collaboration organization:
   - Developers can comment on code, suggest fixes, and discuss changes.
3. Testing and CI/CD automation:
   - MRs can integrate with CI/CD pipelines for automatic testing of changes.
4. Change history and documentation:
   - MRs include descriptions of changes, reviewer comments, and discussions.
5. Development flexibility and version control:
   - Enables easy rollback or amendments before merging.
6. Enhanced security:
   - Prevents vulnerabilities through checks and reviews.

Creating a merge request

1. Navigate to the Merge Requests section:
   - Open your project in Deckhouse Code.

    - In the sidebar, select: “Code” → “Merge requests”.

2. Create a new merge request:

    - Click the “New merge request” button.
    - Choose the source branch (with your changes) and the target branch (usually main).
    - Click “Compare branches and continue”.

3. Fill in merge request details:

    - Title: briefly describe the changes.
    - Description: provide details and add screenshots if needed.
    - Assignee: assign a user to review the request.

4. Submit the merge request:

    - Click “Create merge request”.

Reviewing and merging the request

The merge can proceed once the following conditions are met:

- The merge request has passed review and received the required approvals.
- There are no conflicts between branches.
- All CI/CD pipelines have successfully completed.

To complete the process, click the “Merge” button on the merge request page.

Projects and issues

Groups and members

Every member of a group or project has one of the predefined roles: Guest, Reporter, Developer, Maintainer or Owner. The role determines what the member can see and change, both in the group itself and in every project inside it.

A group also carries settings that apply to all of its projects: protected branch and tag rules restrict who can push directly or delete them, and push rules and other group-wide policies set defaults that a project inherits unless the group allows a project to override them.

1. Create a group:

    - Go to “Groups” → “New Group”.
    - Specify the group name (e.g., “Development Team”).
    - Set the visibility level:
        - “Private” — visible only to group members.
        - “Public” — visible to everyone without exception.
        - “Internal” — visible to all registered users.

Creating a group requires the permission to create groups, which every registered user has by default. The administrator can restrict this permission. See Users and access.

2. Invite users:
    - Open the created group.
    - Go to the “Manage Access” section.
    - Click “Invite members”, enter the user’s email or username, and assign a role (e.g., “Developer”).

> ⚠ When using SSO integrations, the access setup process may differ.

3. Assign roles to group members:
    - Go to the desired group.
    - Select “Manage” → “Members”.
    - Assign the appropriate role to each invited user.

Depending on the assigned role, users have different levels of access:

Guest:

- Group: Can view public items. No access to confidential data.
- Project: Can view issues in public repositories, but cannot comment or manage them.

Reporter:

- Group: Can view all information, including issues and public projects.
- Project: Can read code, CI/CD pipelines, and reports without making changes.

Developer:

- Group: Can create projects and modify repositories.
- Project: Has access to branches, commits, merge requests, CI/CD, and development tasks.

Maintainer:

- Group: Full control over projects and group members.
- Project: Can manage project settings, branches, tags, and merge requests.

Owner:

- Group: Full control, including deletion and role management.
- Project: Has group-level access with full project management rights.

Creating a project

1. Go to the “Projects” section in the sidebar.
2. Click the “New Project” button.

3. Choose one of the available creation methods:
    - “Create blank project” — creates a new repository from scratch.
    - “Import project” — imports an existing project from another system. With the “Repository by URL” method, you can select the “Mirror repository” checkbox to keep the new project synchronized with the source repository through pull mirroring.
    - “Create from template” — creates a new project based on a predefined template.
4. In the form that opens, specify the project parameters:
    - “Project name” — enter a clear and unique name, for example: `my-project`.
    - “Visibility level” — define who will have access to view the project:
        - “Private” — only members of the group or project can access it.
        - “Internal” — visible to all registered users.
5. Click “Create Project”.
6. After creation:
    - Navigate to the newly created project.
    - Click the “Code” button in the top right corner.
    - Copy the clone URL — choose between HTTPS or SSH.

Project visibility levels

Level	Description
“Private”	The project is visible only to its members or members of the parent group
“Internal”	The project is visible to all registered users
“Public”	The project is visible to everyone, including unregistered users

Importing a project

When the source GitLab instance has no network access to Deckhouse Code, importing a project by URL is not possible. Export the project to a file on the source instance, then import that file into Deckhouse Code.

Prerequisites

- The GitLab version on the source instance is the same as, or up to two minor versions older than, the GitLab version Deckhouse Code runs. For example, if Deckhouse Code runs GitLab 19.1, exports from GitLab 19.1, 19.0, and 18.11 are compatible.
- You have the `Maintainer` or `Owner` role on the project you want to export.

- You have the `Maintainer` or `Owner` role on the destination group in Deckhouse Code.

Import steps

Step 1. Enable project export on the source instance

An administrator of the source GitLab instance enables the export feature:

1. Go to “Admin” → “Settings” → “General”.
2. Expand “Import and export settings”.
3. Under “Project export”, select “Enabled”.
4. Select “Save changes”.

> If project export is already enabled on the source instance, skip this step.

Step 2. Export the project

1. Open the project you want to transfer.
2. Go to “Settings” → “General”.
3. Expand “Advanced”.
4. Select “Export project”.
5. After the export is generated, download the resulting file (a `.tar.gz` archive) through the email link, or by selecting “Download export” on the same page.

Save the exported file to your workstation: you upload it to Deckhouse Code in the next step.

Step 3. Enable GitLab export as an import source

An administrator of the Deckhouse Code instance enables the import source:

1. Go to “Admin” → “Settings” → “General”.
2. Expand “Import and export settings”.
3. Under “Import sources”, select the “GitLab export” checkbox.
4. Select “Save changes”.

Step 4. Import the project

1. In the top-right corner, select “Create new” (+) → “New project/repository”.
2. Select “Import project”.
3. In “Import project from”, select “GitLab export”.
4. Enter the project name and URL, then select the exported file.
5. Select “Import project”.

You can check the import status through the project import API.

> For large projects, use Rake tasks for project import instead of the web interface.

What transfers and what does not

Transferred: repository, wiki, issues, merge requests, labels, milestones, snippets, releases, direct project members, LFS objects, issue boards, archived CI/CD pipelines, protected branches and tags, push rules.

Not transferred: CI/CD variables, webhooks, CI/CD job traces and artifacts, package and container registry images, encrypted tokens, deploy keys, pipeline triggers, security policies.

For the full list of exported and non-exported items, see the GitLab project export and import reference.

Issues and boards

Deckhouse Code provides built-in tools for managing project tasks.

Features:

- Create issues with descriptions, labels, and assigned users.
- Set priorities and due dates.
- Support for task lists to break down issues into smaller steps.
- Automatically close issues using commit messages (e.g., `Fixes #123`).

Development milestones

Milestones allow grouping issues and merge requests into releases or sprints.

Features:

- Organize issues and requests based on deadlines.
- Visual representation of progress.
- Automatic milestone closure once all linked items are completed.

Issue board (Kanban)

A tool for visual task management using the Kanban methodology.

Features:

- Customizable columns (e.g., To Do, In Progress, Done).
- Drag-and-drop interface for changing issue status.
- Filtering by labels, assignees, and states.

Project documentation (Wiki)

A built-in Markdown-based Wiki for maintaining project documentation.

Features:

- Organize content into pages and sections.
- Import and export pages for sharing or backup purposes.

Group wiki

Enabling wiki at the group level

To enable or configure access to the Wiki, open the desired group page and go to: “Settings” → “General” → “Permissions and group features” → “Wiki access level”.

Available options:

- Enabled — the Wiki is available to everyone (for public groups) or only to authenticated users (for internal groups).
- Private — the Wiki is accessible only to group members.
- Disabled — the Wiki is completely disabled and cannot be accessed.

Default value — Enabled.

Accessing the wiki

To open the group Wiki, go to the group page and select “Plan” → “Wiki”.

Roles and wiki

Access level is determined by the user’s group membership:

Role	Actions
Guest	View the Wiki
Reporter	View the Wiki, download code
Developer	Create wiki pages
Maintainer	Administer the Wiki
Planner	Administer the Wiki
Anonymous / external user	Access granted if the group is public or internal, and the user is not external — only code download allowed

Features

1. Structure and nesting — creates a tree navigation and helps organize content:
    - Create a hierarchical page structure using the `/` symbol in names. Example:

```
devops/ci-pipelines
devops/kubernetes
product/design
```

2. Markdown, rich text, attachments, and diagrams:
    - Markdown (GitLab Flavored):
        - Mentions (`@username`), references to issues/MRs (`#123`, `!456`), checklists (`- [ ]`), tables, code blocks.
    - Rich text editor:
        - WYSIWYG editor for users who prefer visual formatting.
        - Built-in support for Mermaid and Draw.io.
    - Attachments:
        - Upload and embed images, PDFs, and other files.
        - Stored in the Wiki's Git repository.
3. Change history:
    - Complete history of changes for each page:
        - Who made the changes.
        - When the changes were made.
    - View diffs, revert changes, and track revisions.
4. Discussions and comments on pages.
5. Git access:
    - Each Wiki is a separate Git repository.
    - Clone via SSH or HTTPS:

      ```
      git clone git@code.example.com:groupname/wiki.git
      ```

    - Full access to `.md` files, branches, and history for local editing, backups, or automation.
6. PDF export:
    - Export any Wiki page to PDF via the web interface.
    - Useful for offline access, sharing, or printing.
7. Page templates:
    - Reusable templates stored in the `templates/` directory.
    - Applied when creating or editing pages to standardize content.
8. Customizable sidebar:
    - The sidebar displays pages in a nested tree structure.
    - Fully customizable via the special `_sidebar` page.

    - You can add links, sections, and improve navigation.

Issue analytics

The “Issue analytics” report shows how many issues were opened and closed in each month of the specified period, and lists the issues of that period in decreasing order. The report is available for a group and for a project.

Report availability

The report is available to members of a group or a project with the “Guest” role and above, including those who hold that role in a parent group, and to users of the “Auditor” type.

In a group or a project with the “Public” visibility level, the report is open to any visitor, including one who has not signed in.

With the “Internal” visibility level, the report is open to any user, even one who is not a member of the group or project.

Viewing the report

To view the report of a group:

1. In the top bar, select “Search or go to…” and pick the group.
2. In the group menu, go to “Analyze” → “Issue analytics”.

To view the report of a project:

1. In the top bar, select “Search or go to…” and pick the project.
2. In the project menu, go to “Analyze” → “Issue analytics”.

The report is also available at a direct address:

Report	Address
Group	`https://<HOST>/groups/<GROUP>/-/issues_analytics`
Project	`https://<HOST>/<GROUP>/<PROJECT>/-/analytics/issues_analytics`

The report of a group covers the issues of its projects and the projects of its subgroups, except archived ones.

Chart overview

The “Overview” chart holds one column per month of the period, split into series:

- “Opened” — issues by the month they were created;

- “Closed” — issues by the month they were closed.

An issue created in one month and closed in another falls into both series, so a month can show more closed issues than opened ones. Months are counted in UTC, and a month with no issues is shown as an empty column.

The line above the chart sums up the period: `Opened: 128 in total, 10.7 a month. Closed: 94 in total, 7.8 a month.` The average is the total divided by the number of months in the period.

Reporting period

The period is 12 months, including the current one. To set another duration, add the `months_back` parameter to the address of the page:

```
https://<HOST>/groups/<GROUP>/-/issues_analytics?months_back=3
```

The parameter takes a value from 1 to 36 months, including the current one. A value outside the range is brought to the nearest boundary, and a value that is not a number, to 12.

The caption under the horizontal axis names the period: `Over 3 months (Jul 2026 – Sep 2026)`, and for a period of one month — “Current month”.

Filter bar

The filter bar above the chart narrows both the chart and the table:

- “Author” — one author;
- “Assignee” — one or several assignees;
- “Milestone” — one milestone;
- “Label” — one or several labels;
- “My reaction” — an emoji the current user reacted with;
- search field — search across issues.

Every filter except “My reaction” also accepts the `!=` operator, which excludes the selected value from the report.

The filters and the period are kept in the address of the page, so a configured report can be shared as a link.

Issues table

The “Issues” table lists up to 100 issues created within the period, newest first. The line above the table shows how many issues are shown and how many match the filters: `Newest issues of the period: 100 of 240`. The total is counted up to 1000; past that the line reads `Newest issues of the period: 100 of more than 1000`.

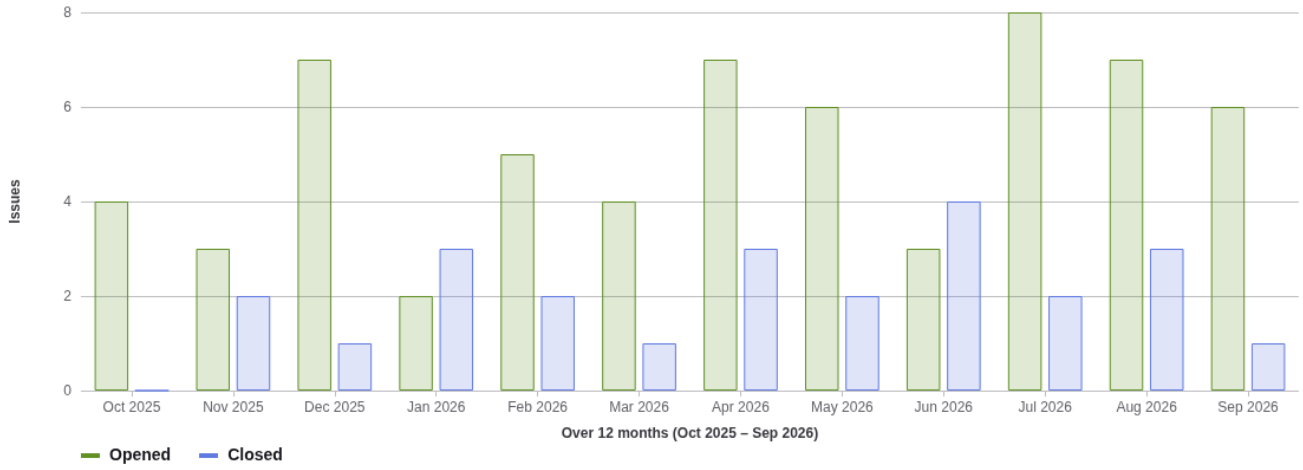
Issue analytics

How many issues were opened and closed each month, and the newest issues behind the numbers.

Search or filter results...

Overview

Opened: 62 in total, 5.2 a month. Closed: 24 in total, 2.0 a month.



Issues

Newest 62 issues of the period

Issue	Age	Status	Milestone	Due date	Assignees	Created by
Archive inactive projects automatically #62 · 3	0 days	Open	Release 2.4	Oct 17, 2026	IV	AK
Reduce N+1 queries on the members page #61 · 2	2 days	Open	Release 2.5	Oct 13, 2026	MS	IV
Add a health check for the object storage connection #60 · 3	4 days	Open	Release 2.3	Oct 9, 2026	DO	MS
Snippet clone URLs use the wrong port behind a proxy #59 · 2	6 days	Open	Release 2.4	Oct 5, 2026	AK	DO

Column	Content
“Issue”	Title with a link to the issue, its number, and the number of its labels. Pointing at the label count shows the labels, and selecting a label opens the issue list filtered by it
“Age”	Number of days since the issue was created
“Status”	“Open” or “Closed”
“Milestone”	Milestone of the issue, with a link to it
“Due date”	Due date of the issue
“Assignees”	Users assigned to the issue
“Created by”	Author of the issue

Repositories

Managing Git repositories

Code storage and versioning

Deckhouse Code provides tools for working with Git repositories. Each project includes a repository for storing source code and tracking change history:

- Full support for the distributed Git version control system.
- Execution of standard commands: clone (`git clone`), commit (`git commit`), push (`git push`), pull (`git pull`).
- Automatic tracking and preservation of change history.

Fork support

A fork is an independent copy of a repository, intended for development without affecting the main codebase:

1. A developer creates a fork of the main repository.
2. Makes changes in their fork.
3. Submits a merge request (MR) to the original repository.
4. After review, the changes can be merged.

This approach is widely used in open source projects and when access to the main repository is restricted.

Working with branches and merging

Branches allow you to develop new features and fixes in isolation from the main codebase:

- Create branches: `git branch <BRANCH_NAME>`, `git checkout -b <BRANCH_NAME>`.
- Switch between branches: `git checkout <BRANCH_NAME>`.
- Merge changes: `git merge <BRANCH_NAME>`.
- Support for protected branches — changes must go through a merge request.
- Automatic deletion of merged branches (if enabled).

Built-in web editor

Allows you to modify code directly in the browser without needing to clone the repository:

- Create and edit files through the web interface.
- Automatically create commits when saving changes.
- Markdown support with preview capability.

Git LFS (Large File Storage) support

Git LFS enables efficient management of large files, such as:

- Images;
- Video;
- Audio files;
- Machine learning datasets.

Git LFS replaces the contents of large files with references, while the actual files are stored outside the Git repository. This reduces load on commit history and improves repository performance.

Protected branches

Protected branches restrict changes to important branches of a repository: only those who are explicitly allowed can push to or merge into such a branch. Protection is typically used for `main`, `master`, release, and stable branches — so that changes reach them only through a merge request and code review.

The default branch of a project is protected automatically when the project is created.

Protected branch restrictions

The following restrictions apply to a protected branch:

- Direct pushes (`git push`) are rejected for everyone except those listed in “Allowed to push and merge”. Other members contribute changes through merge requests.
- Force pushes (`git push --force`) are rejected for everyone unless the “Allow force push” setting is enabled in the branch rule.

- The branch cannot be deleted with Git commands. It can only be deleted through the web interface by users with the `Maintainer` role or higher.

Default branch protection

When a project is created, its default branch becomes protected automatically. The initial protection level is controlled by the “Initial default branch protection” setting:

- **At the instance level**: “Admin area” → “Settings” → “Repository”, the “Default branch” section. Available to instance administrators.
- **At the group level**: On the group page, go to “Settings” → “Repository”, the “Default branch” section. Available to users with the `Owner` role in the group.

The group setting takes precedence over the instance setting: if it is not set for the group, the instance setting applies. Projects in a user’s personal namespace always use the instance setting.

By default, full protection is applied: push and merge are allowed only for members with the `Maintainer` role or higher, and force pushes are rejected.

Branch rules

Branch protection is configured through branch rules. To access rules, open the project page and go to “Settings” → “Repository” → “Branch rules”. Configuration is available to users with the `Maintainer` and `Owner` roles.

To create a rule:

1. Click “Add branch rule”.
2. Select “Branch name or pattern”.
3. Enter the name of a specific branch (for example, `main`) or a pattern with the `*` wildcard (for example, `release/*`). A rule with a pattern protects all matching branches, including ones created later.
4. Click “Create branch rule”.

To open the settings of an existing rule, click “View details” next to it in the list.

Configuring access to a protected branch

On the rule page, the “Protect branch” section contains two access lists:

- **“Allowed to merge”**: Who can merge merge requests into this branch.
- **“Allowed to push and merge”**: Who can push changes directly to the branch (`git push`) and also merge.

To change the corresponding list, click “Edit allowed to merge” or “Edit allowed to push and merge”.

Each list can combine roles, individual users, and groups. The “Allowed to push and merge” list additionally supports deploy keys. Access is granted to anyone who matches at least one of the selected entries.

Available options:

List entry	Who gets access
“Maintainers”	Project members with the `Maintainer` role or higher
“Developers and Maintainers”	Project members with the `Developer` role or higher
“No one”	Nobody. Role-based access is fully disabled
“Administrators”	Instance administrators
“Users”	The listed project members with write access to the repository (the `Developer` role or higher). For details, see “Access for individual users and groups”
“Groups”	Direct members of the listed groups with the `Developer` role or higher in the group. The group must be invited to the project with at least the `Developer` role. For details, see “Access for individual users and groups”
“Deploy keys”	The owner of the selected deploy key, including pushes made with the key itself. The key must be enabled in the project with write access, and its owner must be a project member. Only available in the “Allowed to push and merge” list

Access for individual users and groups

Besides roles, access to a protected branch can be granted selectively — to specific users and groups. For example, you can close the branch to everyone (by selecting “No one”) and allow merging only for the release manager, without raising their role in the project.

Users

In the “Users” selector you can pick project members with write access to the repository — the `Developer` role or higher, including members inherited from the parent group.

Adding a user to the list does not elevate their permissions in the project. A member without the `Developer` role or higher cannot push to the protected branch even if they are listed.

These lists let you narrow down the set of people whose role already grants them write access. For example, select “No one” for roles and list specific users — then only they can work with this branch.

Groups

The “Groups” selector only shows groups invited to the project with the `Developer` role or higher. Group invitations are managed in the project’s “Manage” → “Members” section.

Ancestor groups of the project and groups invited with a lower role cannot be selected.

Only direct members of the invited group with the `Developer` role or higher in that group get access through it. Members of nested subgroups do not get access.

When access stops working

Permissions are checked at the moment of each operation, so membership changes take effect immediately:

- If a user is no longer a project member, their access to the protected branch stops working.
- If the group’s invitation to the project is revoked or its role is downgraded below `Developer`, access for the group’s members stops working.

The entry remains in the access list and becomes effective again if the membership or invitation is restored. To revoke access permanently, remove the user or group from the list in the branch rule.

Allow force push

The “Allow force push” toggle on the rule page allows force pushes (`git push --force`) to the protected branch. Users who can force push are determined by the “Allowed to push and merge” list. The toggle is off by default, and force pushes are rejected for everyone, including maintainers and administrators.

Notes and limitations

- Access granted on a protected branch does not change the user’s role in the project and does not grant any other permissions. It only affects push and merge operations on branches matched by the rule.
- A rule with a pattern (for example, `release/*`) applies to all existing and future branches whose names match the pattern.
- Selecting “No one” disables role-based access but does not clear the lists of users, groups, and deploy keys. The listed entities still have access.
- Only users with the `Maintainer` role or higher can configure branch rules.

Push rules

Deckhouse Code provides a set of push rules to enforce specific policies on commits and Git pushes. These rules help maintain repository integrity, improve code quality, and ensure compliance with your organization’s security standards.

Available rules

- ***Verify committer email***

  Committer email must be verified in the Deckhouse Code profile.

- ***Prevent tag deletion***

  Tags cannot be deleted via git push. Deletion through the Web UI is available.

- ***Member check***

  Commit author must be a Deckhouse Code member.

- ***Prevent secret leaks***

  Any attempt to commit files that match the following patterns will be rejected:

  - `aws/credentials`,
  - `ssh/personal_rsa`, `ssh/personal_dsa`, `ssh/personal_ed25519`, `ssh/personal_ecdsa`, `ssh/personal_ecdsa_sk`, `ssh/personal_ed25519_sk`,
  - `ssh/server_rsa`, `ssh/server_dsa`, `ssh/server_ed25519`, `ssh/server_ecdsa`, `ssh/server_ecdsa_sk`, `ssh/server_ed25519_sk`,
  - `id_rsa`, `id_dsa`, `id_ed25519`, `id_ecdsa`, `id_ecdsa_sk`, `id_ed25519_sk`,
  - files with extensions `.pem` or `.key`,
  - files `.history` or `_history`,
  - files with extensions `.keystore` or `.jks`,
  - `keystore.p12`, `keystore.pfx`.

- ***Require GPG signature***

  Only GPG-signed commits are allowed. This rule also rejects changes via the Web UI if GPG signing is not configured for the UI.

- ***Require DCO (Developer Certificate of Origin)***
  Commits, including merge commits and squash commits, must contain the DCO “Signed-off-by” line. With this rule enabled, the **revert** option for merge requests in the Web UI will be unavailable, since the auto-generated commit message does not include a DCO sign-off.

- ***Verify committer name***

  Committer’s name must match the Deckhouse Code profile name.

- ***Commit message regex***

  Commits must match the specified regex pattern. Leaving the field empty disables the rule.

- ***Author email regex***

  Author email must match the specified regex pattern. Leaving the field empty disables the rule.

- ***File name regex***

File names must match the specified regex pattern. Leaving the field empty disables the rule.

- ***Branch name regex***

  Branch names must match the specified regex pattern. Leaving the field empty disables the rule.

- ***Negative commit message regex***

  Commits will be rejected if their message matches the specified regex pattern. Leaving the field empty disables the rule.

- ***Max file size (MB)***

  Files larger than the specified size (in megabytes) will be rejected. To disable this rule, set the value to `0`.

Configuring push rules

To configure push rules for a project, open the project page and go to “Settings → Repository → Push Rules”.

Only users with the `Maintainer` or `Owner` role can configure push rules.

Configuring push rules at group or instance level

To configure push rules at the group level, open the group page and go to “Settings → Repository → Push Rules”. This option is available to users with the `Owner` role.

Instance-level push rules apply to every project on the instance and are set by the administrator. See Users and access.

When a rule is changed at the instance level, the new values are automatically applied to all groups and projects.

When a rule is changed at the group level, the new values are automatically applied to all subgroups and their projects.

Rules defined at the group or instance level are used as default settings for projects. When a project is created, it inherits the rules from its parent group or from the instance (if created in a personal namespace).

Restricting rule overrides

You can prevent overriding of rules in child projects or groups. To do this, uncheck the option “Allow override at group/project level”. Such a rule cannot be modified in child groups or projects. At the instance level, this applies to all groups and projects.

Examples:

1. If you enable the *“Verify committer email”* rule at the group level and disallow overriding, this rule will be enabled in all projects of that group and its subgroups, with no option to disable it.

2. If you disable the same rule at the group level and disallow overriding, it will be disabled in all projects of that group and its subgroups, with no option to enable it.
3. If overriding is allowed, the rule will still be automatically updated when it changes at the parent level, but child groups and projects will be able to modify it afterwards.

The inheritance hierarchy is as follows:
Instance → Group → Subgroup → Project

Pull mirroring

To configure repository mirroring, on the project page, go to “Settings” → “Repository” → “Mirroring repositories”.

If the repository is empty, import it first. All hooks are triggered during mirroring, and pulling a large repository may significantly impact system performance.

Configuring pull mirroring of a repository

> ⚠ Only one pull mirroring task can be configured per project. Multiple push mirroring tasks are allowed.

To configure pull mirroring of a repository, follow these steps:

1. Go to the project page:
    - Open your project in the Deckhouse Code interface.
    - In the left-hand menu, select “Settings” → “Repository”.
    - Scroll down to the “Mirroring repositories” section.
2. Specify the repository URL. Credentials in the URL are ignored. Use the fields in the “Authentication method” block below for authorization.
3. Configure authentication:
    - If using HTTP(S) access, in the “Authentication method” field, select “Username and password” and enter:
        - “Username”: Your username.
        - “Password”: Your password or access token.
    - If using SSH mirroring, specify the username (typically `git`). After saving the configuration, Deckhouse Code will generate an SSH key to be used for access.

Branch filter

The “Branch filter” block in the mirroring settings defines which branches are pulled from the source repository:

- “Mirror all branches”: All branches of the source repository are mirrored.
- “Mirror only protected branches”: Only the branches protected in this project are mirrored. The option is unavailable until the project has at least one protected branch (protected branches of the parent group are taken into account as well). For more information, see Protected branches.
- “Mirror matching branches”: Only the branches whose names match the specified regular expression (RE2 syntax) are mirrored. The expression is matched against the whole branch name. While you are typing the expression, the form shows how many branches of this repository match it.

How the branch filter affects tag synchronization

> ⚠ The branch filter applies to tags as well. If “Mirror only protected branches” or “Mirror matching branches” is selected, a tag of the source repository is synchronized only if its commit is present in one of the branches of this repository. All other tags are skipped.

Consider the following specifics:

- With “Mirror all branches”, all tags of the source repository are synchronized.
- A tag skipped because of the filter is not lost: it is re-checked during every synchronization, and as soon as its commit reaches one of the mirrored branches, the tag is created in the mirror.
- The check is performed against all branches of the mirror repository, including branches created directly in Deckhouse Code, and not only against the branches that match the filter.
- Tags that have already been synchronized remain in the repository even if the branch filter is changed later so that it no longer covers them.

Importing a repository by URL as a pull mirror

You can set up pull mirroring while importing a repository by URL, so that the project is created and immediately configured as a pull mirror of the source repository. This way, you do not need to configure mirroring separately after the import.

> ℹ Repository mirroring must be enabled for the instance by an administrator. In the “Admin area”, go to “Settings” → “Repository” → “Repository mirroring” (the `mirror_available` setting). If mirroring is not enabled for the instance, the “Mirror repository” checkbox is shown but disabled (grayed out) and cannot be selected.

To import a repository by URL as a pull mirror, follow these steps:

1. Create a new project:
    - Go to “New project” → “Import project” → “Repository by URL”.
    - Specify the URL of the source repository.
    - If the source repository requires authentication, provide the credentials used to access it.
2. Select the “Mirror repository” checkbox.
3. Create the project.

During the initial import, the repository is created and populated from the source. After that, the project is kept in sync with the source repository automatically, on the pull mirroring schedule (for more information, see the “Scheduling and error handling” section below).

To restrict mirroring to protected branches or to branches matching a pattern, configure the branch filter after the import on the project’s “Settings” → “Repository” → “Mirroring repositories” page (for more information, see the “Branch filter” section above).

You can also turn an existing project into a pull mirror by running an import by URL into it; this requires the Maintainer role. In either case, mirroring runs on behalf of the user who configured it.

Scheduling and error handling

- Pull mirroring tasks are scheduled once per hour (`Projects::PullMirrorScheduleWorker`).
- Each mirror is updated no more than once every 6 hours.
- If a mirroring task fails, the next attempt will be during the next run of `Projects::PullMirrorScheduleWorker` . The maximum number of retry attempts is **5**. Clicking “Update now” resets the failure counter.
- If a Sidekiq task terminates unexpectedly (for example, due to a Sidekiq restart), its status is updated after 3 hours, and a new synchronization attempt is scheduled.

LFS (Large File Storage) specifics

When using pull mirroring, LFS objects are fetched **only if** LFS is enabled in the target Deckhouse Code project.

CODEOWNERS

The **CODEOWNERS** feature allows you to determine individuals responsible for specific parts of the repository. Users and groups can be determined as responsible parties. Changes affecting files specified in `CODEOWNERS` must be approved by the code owners.

Using CODEOWNERS helps:

- Require approval from domain experts for important directories.
- Simplify the process of finding those responsible for a specific section of code.

CODEOWNERS complements the approval rules mechanism, but does not replace it. Unlike approval rules, which are set manually in the UI, CODEOWNERS works through the `CODEOWNERS` file in the repository.

How CODEOWNERS works

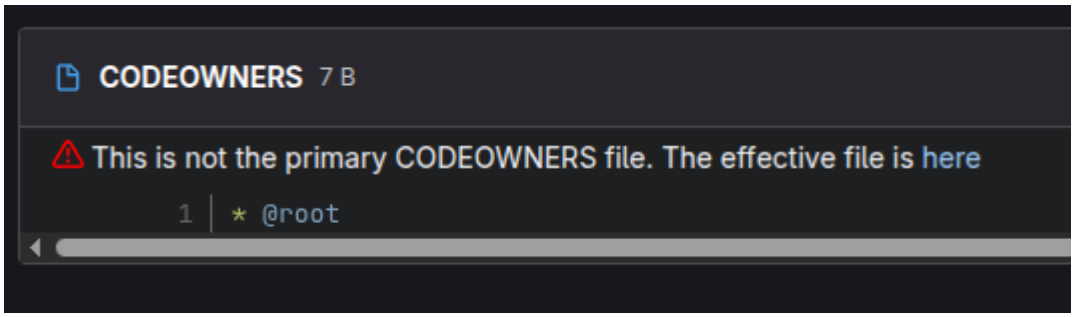
`CODEOWNERS` is a text file in the repository. Deckhouse Code uses it to determine the owners of files involved in a merge request.

It is searched in three places (in order of priority):

1. `./CODEOWNERS`
2. `./docs/CODEOWNERS`
3. `./.gitlab/CODEOWNERS`

Only the first file found is used.

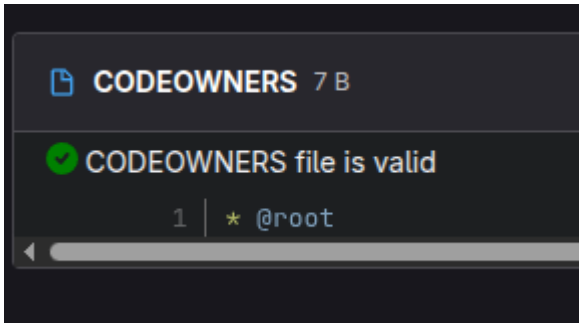
If there are several files in the project and you are currently viewing a non-priority file, you will see a corresponding message.



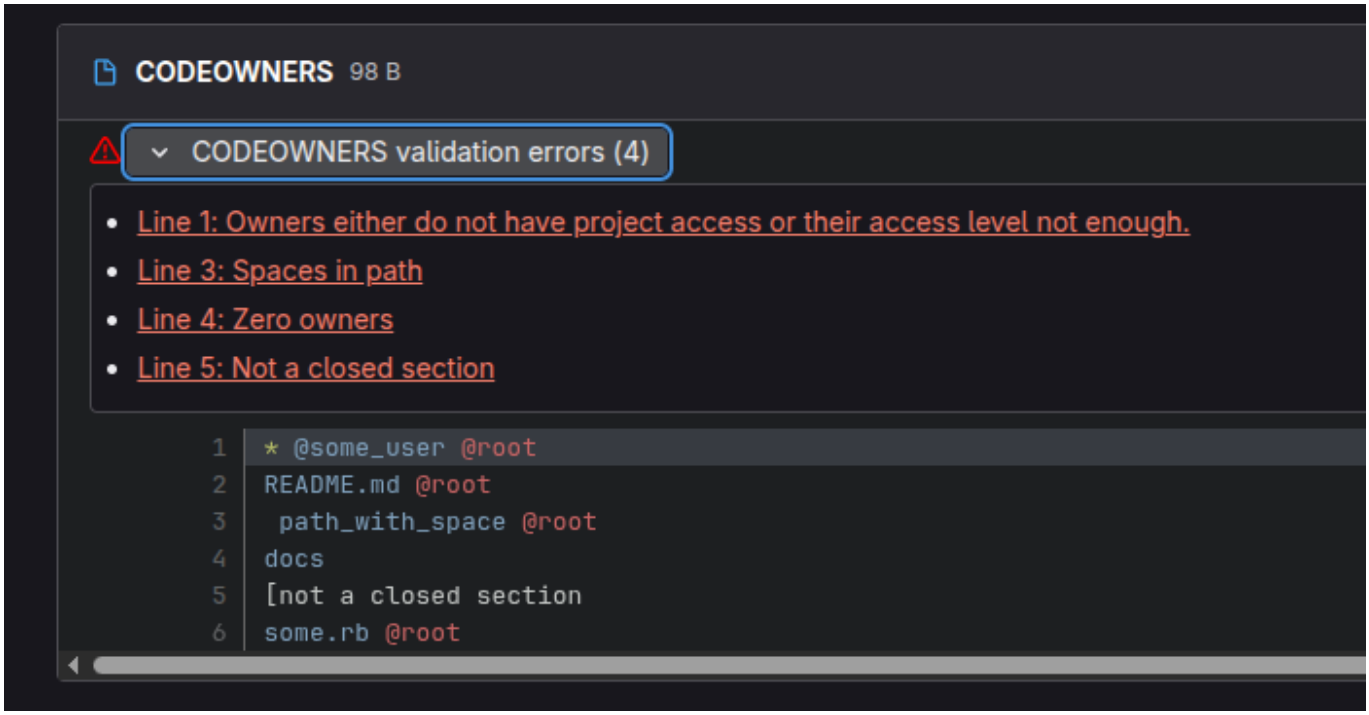
Validating the CODEOWNERS file

Validation helps you find errors in the file during editing.

If the file is valid, the following message is displayed:



If the file is invalid, the problematic lines and the type of error will be displayed:



Validation reports the following types of errors:

- *Owners either do not have project access or their access level not enough*: At least one of the owners does not have sufficient rights in the project.
- *Spaces in path*: If the path contains spaces, they must be escaped with the `\` character.
- *Zero owners*: No owners are specified.
- *Not a closed section*: The closing section character `]` is missing.

Where CODEOWNERS is used

If the MR modifying the file falls under the CODEOWNERS rules, approval must be obtained from the owners.

CODEOWNERS format

Rule string:

```
<path or pattern> <list of owners>
```

Owners:

- `@user`
- `@group`
- `@group/subgroup`
- Roles: `@@developer`, `@@maintainer`, `@@owner`

Examples

```
# Owner of all files.
* @default-team

# README.md only in the root directory.
/README.md @docs-team

# All Ruby files.
*.rb @backend-team

# The entire config directory.
/config/ @devops

# README.md anywhere.
README.md @docs
```

Path patterns

Deckhouse Code uses wildcards:

- `*`: Any characters except `/`.
- `**`: Matches multiple directory levels.
- `/dir/`: Only the specified directory.
- `*.md`: All Markdown files.
- `/config/**/*.rb`: All Ruby files inside `config/*` at all levels.

Pattern examples

```
/docs/*       @docs-one      # one level
/docs/**      @docs-two      # recursively
/app/**/*.rb  @ruby-team
```

CODEOWNERS sections

In the `CODEOWNERS` file, sections are named areas that are analyzed separately and always applied. Until you define a section, Deckhouse Code treats the entire `CODEOWNERS` file as a single section.

Features of using `CODEOWNERS`:

- Deckhouse Code treats entries without sections, including rules defined before the first section, as a separate, unnamed section.
- Each section processes its rules separately.
- If the file path matches multiple entries within a single section, only the last matching entry in that section is used.
- If the file path matches entries in multiple sections, the last matching entry in each section is used.

For example, in a `CODEOWNERS` file with the following sections defining owners for README:

```
* @root

[README Owners]
README.md @user1 @user2
internal/README.md @user4

[README other owners]
README.md @user3
```

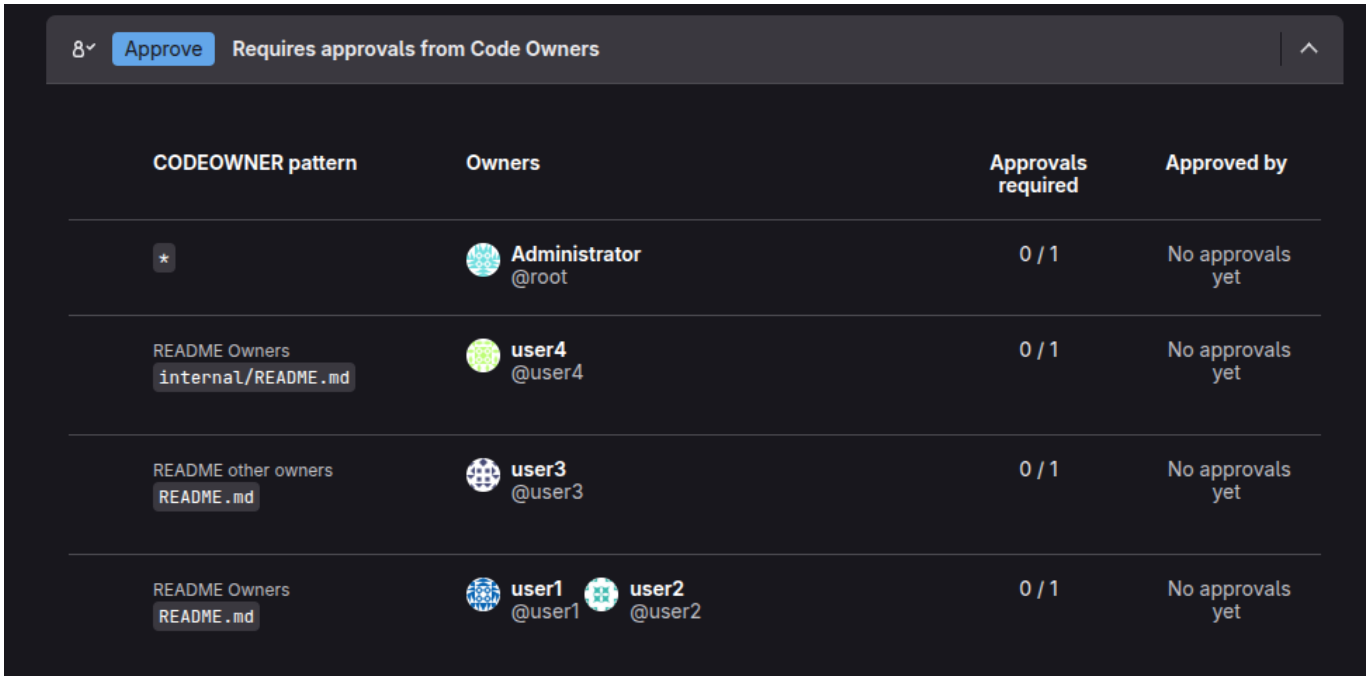
Owners for `README.md` in the root directory:

- `@root`: From the unnamed section.
- `@user1` and `@user2`: From the `[README Owners]` section.
- `@user3`: From the `[README other owners]` section.

Owners for `internal/README.md`:

- `@root`: From the unnamed section.
- `@user4`: From the `[README Owners]` section. (Both `README.md` and `internal/README.md` match the section rules, but only the last matching entry in this section is used).
- `@user3`: From the `[README other owners]` section.

In the merge request widget, each code owner is displayed under their own label.



Detailed example: complex combination of sections

Below is an example of a real industrial file showing:

- Sections with different numbers of approvals
- Redefining owners
- Optional sections
- Exceptions
- Inheritance of sections with identical names

```
# Global settings.
* @fallback-team
!*.lock                    # lock files do not require approval
!**/generated/**           # automatic generation

[Backend][2] @backend-core
# Requires 2 approvals from backend-core for any Ruby code.
app/**/*.rb

# But for critical models, we define a different order.
app/models/**/*.rb @backend-core @security-team

# But this model is an exception; the owner is different.
app/models/legacy/**/*.rb @migration-team

[Backend]
# Section update: Backend now also includes SQL.
db/**/*.sql @db-team

[Ruby Optional]
# Additional highlighting for owners, approvals are NOT required.
^[Ruby Optional]
*.rb @ruby-advisors

[Frontend][3]
# The UI requires 3 approvals.
*.vue @frontend-team
*.js  @frontend-team

# Redefining: specific file → specific person.
frontend/critical_entry.vue @frontend-lead

[Docs]
*.md @technical-writers

[Docs]
# Redefining: README always belongs to docs-lead.
README.md @docs-lead
```

Things to note about the example:

- The `[Backend]` section is declared twice → Deckhouse Code will combine it.
- `[Backend][2] @backend-core` specifies **2 mandatory approvals**.
- `[Frontend][3]` requires **3 approvals** from the frontend.
- `[Ruby Optional]` is marked as optional — owners are visible, but their approval is not required.
- The exceptions `!*.lock` and `!**/generated/**` mean that these files do not require approvals at all.

Exclusions (!)

Used to exclude files within a section:

```
* @default
!package-lock.json
!**/generated/**
```

After exclusion, the file **cannot be re-included** in the same section.

Rule processing order

Rules are processed in the following order:

- Deckhouse Code reads the file from top to bottom.
- Within a single section, rules with the same path **override** previous ones.
- If a file fits into several sections, the owners from all these sections are added.
- Sections do not override each other: They are *independent*.

Who can be an owner (eligible owners)

1. Users (via @username)

A user is considered a valid owner if:

- They have **direct** membership in the project (Developer, Maintainer, or Owner).
- They have membership in a project group (including inherited: project group, parent groups, ancestors of parent groups).
- They are a direct or inherited member of a group that is directly invited to the project.
- They are a direct but not inherited member of a group that is invited to the project group.
- They are a direct but not inherited member of a group that is invited to an ancestor of the project group.

To sum up, all users who are displayed on the project participants page with the role of Developer or higher are eligible.

A user is not considered an owner if:

- They are blocked.

- Their access has been revoked.
- Their role is lower than Developer.

2. Groups (via @group or @group/subgroup)

A group can be an owner **only if:**

- It is directly invited to the project with the Developer+ role (via “Invite a group”).
- Only *direct members* of this group are considered owners.

Inherited members of parent groups are not considered owners.

3. Roles (via @@role)

Format:

```
@@developer
@@maintainer
@@owner
```

Features:

- **Only direct project participants** are taken into account.
- Group members are not included in the calculation.
- Roles are not inherited from each other (`@@developer` does NOT include maintainers).

You can specify multiple roles in one line:

```
file.md @@developer @@maintainer
```

Interaction with approval rules

CODEOWNERS and approval rules work independently, but their requirements are cumulative.

Example:

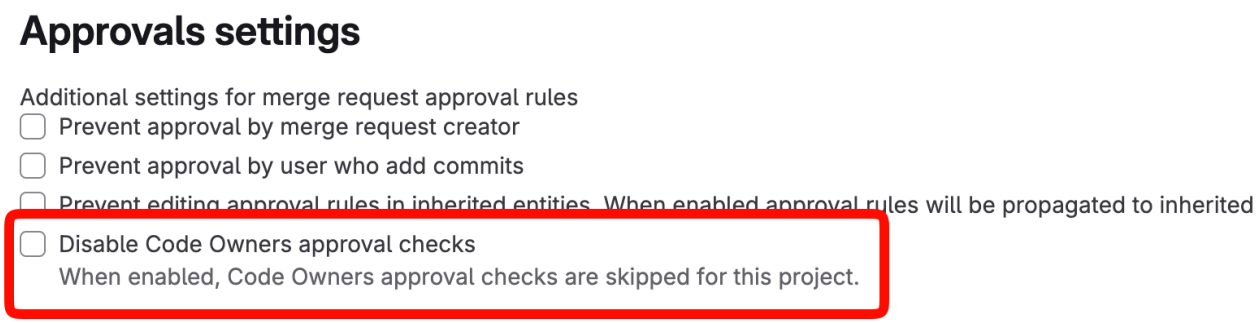
- The `Security` rule requires 1 approval.
- The `[Backend][2]` section requires 2 approvals.
- MR changes the backend file.

Result: **3 approvals** are required.

Disabling CODEOWNERS checks

CODEOWNERS checks, which are enabled by default in Deckhouse Code, can be disabled at the project level. This is useful when a project has a `CODEOWNERS` file, but approvals from code owners are not required.

To disable these checks, navigate to “Project” → “Settings” → “Merge requests” → the additional approval rule settings, and select “Disable Code Owners approval checks”.



Once the checks are disabled, CODEOWNERS are skipped across the whole project:

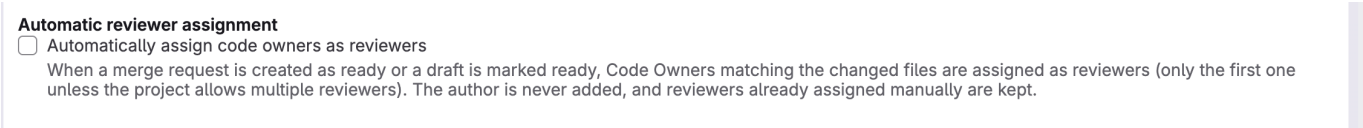
- Merge requests are not blocked by missing approvals from code owners.
- Code owners are not defined for the changed files — the list of applicable entries is empty.
- Enabling or disabling the setting is immediately applied to already open merge requests.

Validation of the `CODEOWNERS` file syntax keeps working. The file can be validated even when the checks are disabled.

Automatically assigning code owners as reviewers

Code owners matching the changed files can be automatically assigned as reviewers of a merge request.

To enable that, navigate to “Project” → “Settings” → “Merge requests” → “Automatic reviewer assignment”, and select “Automatically assign code owners as reviewers”.



This setting is disabled by default. When it is enabled, code owners are assigned as reviewers:

- When a merge request is created as ready (not a draft).
- When a draft merge request is marked ready.

Restrictions applied when the automatic assignment is enabled:

- Assignment is not performed for drafts.
- If a merge request already has reviewers, automatic assignment is skipped — manually assigned reviewers are kept.
- The merge request author can never be assigned as a reviewer.
- Only users with read access to the merge request are assigned.
- If the project forbids to assign multiple reviewers, only the first code owner is assigned.

Formatting recommendations

When formatting the `CODEOWNERS` file, follow these recommendations:

- First, set broad rules (`* @team`). Then specify more detailed rules (`app/**/*.rb @backend-team`).
- Use sections for better structure and clarity.
- Avoid duplicates — the last rule in a section takes precedence.
- Optional sections are useful for highlighting experts without mandatory approvals.

Example of the final file

```
# Global owners.
* @default-team

# Exclusions.
!yarn.lock
!**/generated/**

[Backend][2]
app/**/*.rb @backend-core

[Frontend][3]
*.vue @frontend-team
*.js @frontend-team

^[Docs]
*.md @docs-team

[Critical Overrides]
/config/production.yml @ops-lead
```

Group repository analytics

The “Repository analytics” report is available at group level. It shows test code coverage for projects in the group and its subgroups. Coverage appears after a pipeline that reports it finishes on a project’s default branch.

Report availability

The report is available to group members with the “Reporter” role and above, and to users of the “Auditor” type. Guests and anonymous users cannot open it, including in a public group.

Viewing the report

1. In the top bar, select “Search or go to…” and pick the group.
2. In the group menu, go to “Analyze” → “Repository analytics”.

The report is also available at a direct address:

```
https://<HOST>/groups/<GROUP>/-/analytics/repository_analytics
```

Test code coverage

The “Test Code Coverage” section shows data from the most recent day that has coverage data:

- “Projects with Coverage” — number of projects with coverage data;
- “Average Coverage by Job” — mean coverage across the jobs that report it;
- “Jobs with Coverage” — number of jobs that report coverage.

The “Last updated” label names that day.

Average test coverage

The “Average test coverage” chart plots the mean coverage of all jobs by day for the last 30 days. Days without coverage data have no point on the chart.

Point at a chart point to see its date, the coverage percentage, and the number of jobs with coverage.

Latest test coverage results

The “Latest test coverage results” table shows up to 20 projects with coverage by default. Rows are sorted alphabetically. To replace this set, select one or more projects in the “Projects” list. The list includes projects in subgroups.

Column	Content
“Project”	Project name with a link to its coverage chart
“Coverage”	Mean coverage of the project’s jobs
“Coverage Jobs”	Number of jobs that report coverage
“Last Update”	Date of the most recent coverage data

Downloading coverage data

The report can export historical coverage data for all projects in the group and its subgroups, or for selected projects. The CSV file contains up to 1000 most recent records, ordered from newest to oldest. Each row holds the date, the job group name, the project name, and the coverage percentage.

To download a CSV file:

1. Select “Download historic test coverage data (.csv)”.
2. In the “Projects” list, select projects or leave “All projects”.
3. In “Date range”, select 7, 14, 30, 60, or 90 days. The default is 30 days.
4. Select “Download test coverage data (.csv)”.

Repository analytics for a project

The “Repository analytics” report is available at project level. It shows the programming languages used in the repository’s default branch, and test code coverage and commit statistics for a selected branch or tag.

Report availability

The report is available to users who can download the project’s code. The project must have an initialized repository and at least one commit in its default branch. Wiki commits are not included in the report.

Viewing the report

1. In the top bar, select “Search or go to…” and pick the project.
2. In the project menu, go to “Analyze” → “Repository analytics”.

The report is also available at a direct address:

```
https://<HOST>/<GROUP>/<PROJECT>/-/graphs/<REF>/charts
```

To view test coverage and commit statistics for another branch or tag, select it in the list next to “Commit statistics”. The programming languages chart always uses the default branch.

Programming languages

The “Programming languages used in this repository” chart shows the share of code in each programming language in the project’s default branch. The calculation uses code size in bytes and excludes generated and vendored code.

Test code coverage

The “Code coverage statistics” chart shows coverage data for the selected branch or tag for the last 90 days. Data appears after a pipeline with coverage data completes.

If more than one job group publishes coverage, select its name from the list above the chart. Point at a chart point to see the date and coverage percentage. When the chart has data, select “Download raw data (.csv)” to download it.

Coverage data is shown only to users who can view the project’s pipeline and job results. For a public project, anonymous users can view coverage when public pipelines are enabled.

Commit statistics

The “Commit statistics” section uses up to 2000 non-merge commits from the selected branch or tag. It shows the total number of commits, the daily average, and the number of authors, followed by charts of commits:

- by day of the month;
- by day of the week;
- by hour in UTC.

Merge requests

Code review

Merge requests (MR)

A key tool for ensuring code quality before merging changes into the main branch.

Main features:

- Creating an MR after committing to a feature branch.
- Automatic comparison of changes with the target branch.
- Built-in discussions and code review.
- Support for auto-merge after successful checks (CI/CD, review).
- Visual conflict display and tools for resolution.

Diff view

Allows visual comparison between code versions.

Supported modes:

- Standard diff — commit-by-commit comparison.
- Side-by-side diff — line-by-line comparison.
- Inline diff — highlights changes within lines.

Comments and discussions

Tools for collaborative work on code changes.

Features:

- Commenting on specific lines of code.
- Group discussions with user mentions.
- “Mark as resolved” functionality for closing threads.
- Automatic creation based on comments.

Event notifications

Notification system for project events.

Supported notifications:

- Email alerts about new commits, MRs, and comments.
- Integration with external messengers: Slack, Mattermost, Telegram (configurable alerts).

Merge request approvals

A quality control mechanism that enforces required approvals before merging.

Features:

- Configurable minimum number of approvals.
- Assigning required reviewers.
- Support for auto-approval based on conditions.
- Merge blocking if required approvals are missing.
- Integration with the Codeowners mechanism for mandatory review by code owners.
- Automatic reset of approvals on MR changes.
- Logging of all approval-related actions.

Codeowners

A mechanism for assigning responsibility over parts of the codebase.

Functionality:

- Defining owners for specific files and directories.
- Using a `CODEOWNERS` file to declare ownership.
- Automatically assigning owners as MR reviewers.
- Integration with the approvals system: owners become mandatory reviewers.
- Support for wildcard path patterns (`*`, `**`) and owner groups.
- Ability to restrict changes in protected branches to owners only.

Approval rules for merge requests

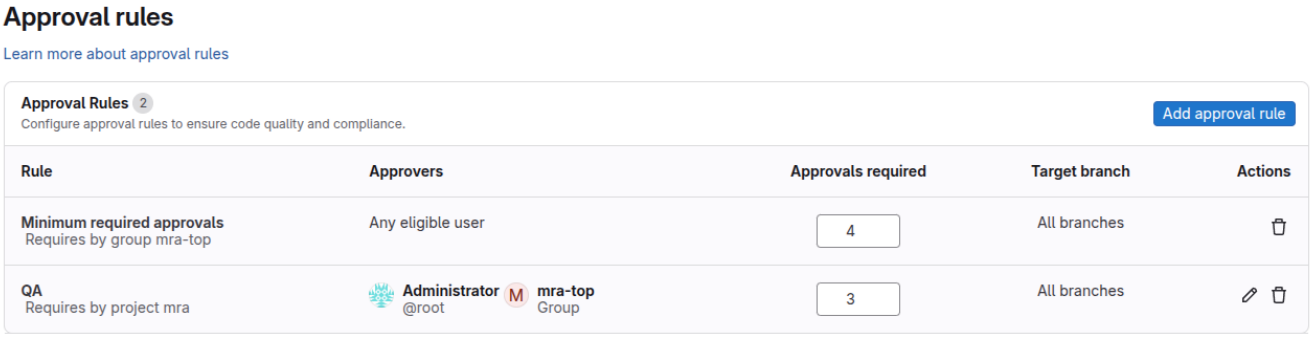
Approval rules help control code quality and ensure mandatory review before merging. They can be used to determine how many approvals are required, who should participate in the review, and in which cases the rule should be triggered.

These rules operate at the group, project, and merge request levels, are inherited down the hierarchy, and enable centralized management of the review process. Together with CODEOWNERS, they form a flexible and reliable change control system, protecting important parts of the repository from unwanted or unreviewed edits.

Defining rules

Rules can be defined at the following levels:

- Groups (available for the **Maintainer** role). To define rules for a group, go to “Group” → “Settings” → “Merge requests” → “Approval rules”.
- Project (available for the **Maintainer** role). To define rules for a project, go to “Settings” → “Merge Requests” → “Approve Rules”.
- Merge requests (available for the **Developer** role). To define rules for a merge request, expand the **“Approval rules”** section on the MR creation or editing page.



Description of columns in the approval rules table

- **Rule**: Contains the name of the rule and inheritance attribute. The entry “Required by Example Org group” means that the rule was created in the *Example Org* group.
- **Approvers**: Contains a list of users and groups whose approvals are taken into account when checking compliance with the rule.
- **Required number of approvals**: The number of approvals that must be obtained from users or group members specified in the “Approvers” column. The value `Any eligible user` means that an approval from any user who has the right to do so will be counted.
- **Target branch**: The rule applies only if the merge request is sent to the specified branch. A wildcard (`*`) can be used as the branch name.

Rule inheritance

Approval rules are inherited in a cascade: group → subgroup → project → merge request. The inheritance mechanism has the following features:

- When a subgroup, project, or merge request is created, the rules of the parent entity automatically appear in the new entity as inherited. The table of rules displays the level at which the rule was originally created next to its name.
- When new rules are created at the group or project level, they are automatically added to all existing lower-level subgroups and projects. **New rules are not added to merge requests that have already been created.**

- If you change a rule at the parent level, the changes are automatically applied to all inherited rules at lower levels. For example, if a project has inherited a rule from a group and you change the number of required approvals in the group, this change will also be reflected in the project.
- If you change an inherited rule in a child entity, **it becomes an independent rule**. That is, the rule is no longer associated with the parent: a separate copy with new parameters remains at the child level, and further changes to the rule in the group will no longer affect this rule in the project.

Rules for the group

Rules created at the group level are automatically added to new projects and subgroups as inherited. To open them, go to “Group” → “Settings” → “Merge requests” → “Approval rules”.

Group rules can be managed by group members with the **Maintainer** role.

The “Approval rules” section displays a list of rules configured at the group level.

Creating a rule in a group

You can create a new rule by clicking the **“Add approval rule”** button and specifying:

- The name of the rule.
- The target branch (presets available: “All branches”, “All protected branches”, “Enter branch name”, or wildcard `*`).
- The required number of approvals.
- Approvers: Direct members of the group or group on the instance. When adding a group, only approvals from direct members of the added group are taken into account.

Project rules

Rules created at the project level are automatically added to new merge requests as inherited. To open them, go to “Project” → “Settings” → “Merge requests” → “Approval rules”.

Project rules can be managed by project members with the **Maintainer** role.

Creating a rule in a project

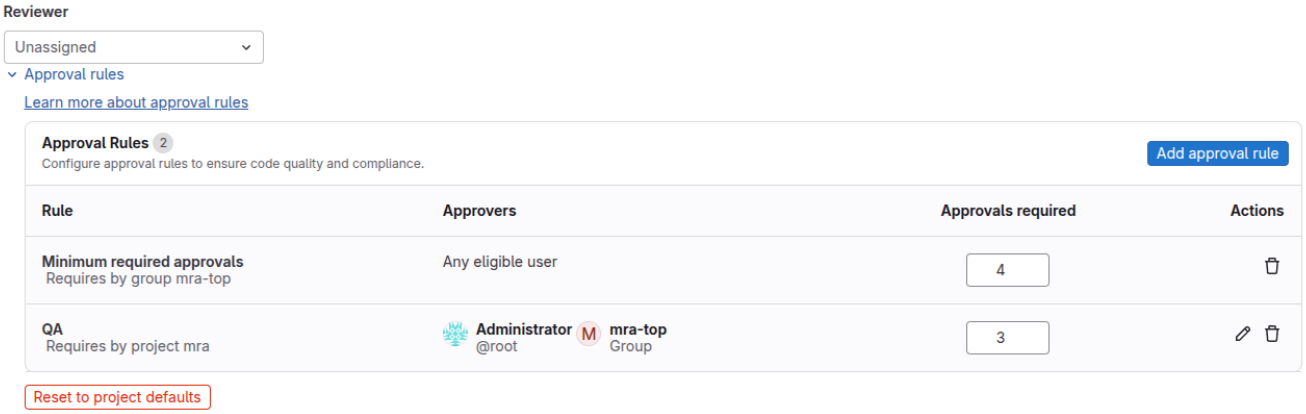
When creating a new rule, you can specify:

- The name of the rule.
- The target branch (you can select a preset, specify the branch name or a regular expression; you can also select a protected project branch).
- The required number of approvals.
- Approvers: Direct or inherited project members, as well as groups invited to the project with the **Reporter** role. For added groups, only approvals from direct group members are counted.

Rules for merge requests

To open the rules of a merge request, expand the **“Approval rules”** section on the page for creating or editing it.

Merge request rules can be managed by users with the **Developer** role.



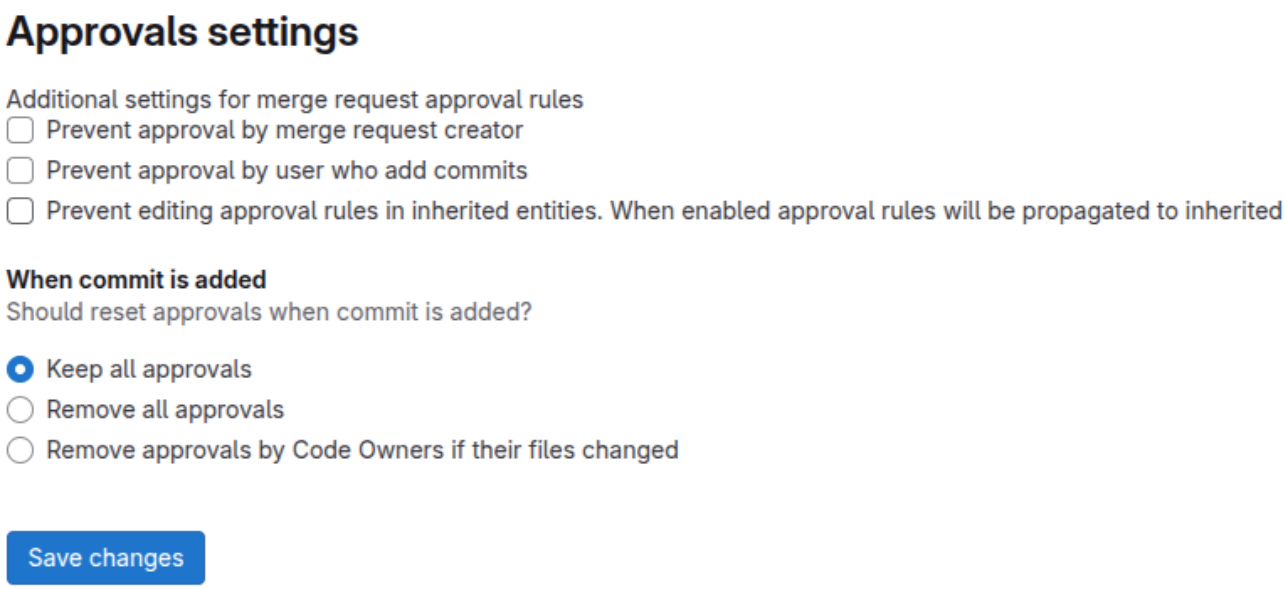
The MR page has a **“Reset to project defaults”** button that deletes the current rules and adds all project rules that apply to the target MR branch.

Creating a rule in MR

When creating a rule, you can specify:

- The name of the rule.
- The required number of approvals.
- Approvers: Direct or inherited project members, as well as groups with the **Reporter** role or higher. Only approvals from direct group members are taken into account on the group side.

Additional approval settings



At the project or group level, you can configure additional settings:

- **Prevent approval by merge request creator**: The MR author cannot approve.
- **Prevent approval by user who add commits**: Committers cannot approve.
- **Prevent editing approval rules in inherited entities**: When enabled:
  - Rules are copied to child entities and become unavailable for editing or deletion.
  - Lower-level groups and projects can only create their own rules.
  - Rules are not automatically added to existing MRs.
  - Current approval settings also apply to all child entities.
- **When commit is added**: This section determines whether existing approvals should be reset when a new commit appears.

Who can perform approvals

If the approval settings do not prohibit it, approvals can be performed by:

- All project members with the **Developer** role or higher.
- Members with the **Planner** role or higher, if they are assigned as responsible in the merge request.

External status checks in merge requests

External status checks let project maintainers send merge request data to an external service and use the service response as an additional merge condition. Use them when a project must pass an external compliance check, quality gate, or security validation before a merge request can be merged.

External status checks are configured for each project separately and are not shared between projects.

How external status checks work

When a merge request event occurs, Deckhouse Code sends information about the merge request to every external status check that applies to the target branch. The external service checks the merge request and sends a result back to Deckhouse Code.

A check result always applies to the current `HEAD` SHA of the merge request source branch. If new commits are pushed to the merge request, the previous result no longer applies to the new state and Deckhouse Code waits for a new result.

External status checks can affect merging only if the project setting **Status checks must succeed** is enabled. If the setting is disabled, the widget still shows check results, but failed or pending checks do not block merging.

Configuring external status checks

To configure external status checks, open the project and go to “Settings” → “Merge requests”.

The page contains two related configuration areas:

- “Merge checks”: Project-level settings that affect mergeability.

- “External status checks”: A table for creating, updating, and deleting external status check services.

Users need permission to manage merge request settings in the project. In standard roles, this is available to users with the **Maintainer** or **Owner** role.

Merge checks settings

Status checks must succeed

The “Status checks must succeed” checkbox blocks merge requests until all applicable external status checks have the `passed` status.

If the checkbox is cleared, external status checks are still shown in merge requests but do not block merging.

Status checks timeout

The “Status checks timeout” field sets the default timeout for project status checks.

Behavior:

- The default value is `5` minutes.
- The value must be `1` minute or greater.
- The value applies to checks without an individual timeout.
- If the external service does not respond before the timeout expires, the check response becomes `failed`.

External status checks table

The “External status checks” table shows all status check services configured for the project.

The table contains:

- The status check name.
- The external service URL.
- The target branch scope.
- The status check timeout or the project default timeout.
- The HMAC shared secret status.
- “Update” and “Delete” actions.

If a check does not have an individual timeout, the table shows the project default timeout with the `(default)` label.

Adding an external status check

To add an external status check:

1. Open the project.
2. Go to “Settings” → “Merge requests”.

3. In “External status checks”, select “Add external status check”.
4. Fill in the fields.
5. Select “New external status check”.

Field	Description
Service name	Name of the external service. The value is required, must be unique in the project, and must not exceed 255 characters
API to check	URL of the external service endpoint. The value is required, must be unique in the project, and must use the `http` or `https` protocol
Target branch	Scope that defines which merge request target branches use the check
Timeout minutes	Individual timeout for this check. If set, it overrides the project-level **Status checks timeout** value
HMAC Shared Secret	Optional secret used to sign requests sent from Deckhouse Code to the external service

After a check is created, Deckhouse Code creates `pending` check responses for matching open merge requests. Requests to the external service are not sent for those existing merge requests. These responses become `failed` after the timeout expires unless a user retries the check.

For new merge request events, Deckhouse Code sends requests to the matching external status check services automatically.

Branch scope

The “Target branch” field defines which merge requests use the status check.

Scope	Description
All branches	Applies the check to merge requests targeting any branch
All protected branches	Applies the check to merge requests targeting protected branches
Selected protected branches	Applies the check only to merge requests targeting the selected protected branches. Wildcard protected branches are supported

If a merge request target branch does not match a check scope, the check is not shown in the merge request widget.

When the target branch changes, Deckhouse Code recalculates the applicable checks. Checks that no longer apply are removed from the merge request, and newly applicable checks are added in the `pending` status.

HMAC shared secret

If “HMAC Shared Secret” is set, Deckhouse Code adds the `X-Gitlab-Signature` header to requests sent to the external service.

The header value is an HMAC-SHA256 digest of the request body, generated with the shared secret.

The secret is stored encrypted. After it is saved, the UI only shows that a secret exists. To replace the secret:

1. In the status check row, select “Update”.
2. Select “Edit secret”.
3. Enter a new value.
4. Select “Update status check”.

Check lifecycle

When a merge request event occurs, Deckhouse Code sends a merge request payload to every applicable external status check service. The payload includes the `external_approval_rule` object with the check `id`, `name`, and `external_url`.

A check response can have one of the following statuses:

Status	Description
`pending`	Deckhouse Code is waiting for the external service response
`passed`	The external service approved the merge request state
`failed`	The external service rejected the merge request state, the request to the external URL failed, or the timeout expired

The external service updates a response by using the API endpoint:

```
POST /projects/:id/merge_requests/:merge_request_iid/status_check_responses
```

The request must include:

- `external_status_check_id`: Status check ID.
- `sha`: Current `HEAD` SHA of the merge request source branch.
- `status`: `passed` or `failed`.

Deckhouse Code does not update the response if:

- `sha` does not match the current source branch `HEAD`.
- The response is no longer in the `pending` status.
- The timeout has already expired.

Timeouts

Timeout counting starts when Deckhouse Code sends the request to the external service. If a user retries a failed check, timeout counting starts from the retry time.

Timeout values are selected in this order:

1. The “Timeout minutes” value of the status check.
2. The project-level “Status checks timeout” value.

A background worker checks pending responses every minute. When a response exceeds its timeout, Deckhouse Code changes its status to `failed` and records the reason as `Automatically closed after timeout`.

If the request to the external service fails, Deckhouse Code changes the response status to `failed` and stores the error reason. Users can see the error reason in the merge request widget.

Merge request widget

The “External status checks” widget is shown on the merge request page when the merge request has applicable external checks.

The widget shows:

- Check name.
- Check status.
- External service URL, if your role allows viewing it.
- Error details for failed checks, if Deckhouse Code has an error reason.
- “Retry” action for failed checks, if your role allows retrying them.

The widget helps authors and reviewers understand which external systems still need to respond before the merge request can be merged.

Pending checks

A check stays in `pending` while Deckhouse Code waits for the external service response.

While at least one check is `pending`, the merge request widget refreshes the external status checks approximately every 10 seconds. Polling stops when no checks are in `pending`.

A pending check can become `failed` automatically if the external service does not respond before the configured timeout. The timeout is configured at the project level or for a specific external status check.

Failed checks

A check can become `failed` when:

- The external service returns `failed` for the current merge request `HEAD` SHA.
- Deckhouse Code cannot send a request to the external service.
- The external service URL is blocked or unavailable.
- The external service does not respond before the timeout expires.

If “Status checks must succeed” is enabled for the project, a failed check blocks merging until the check passes or the project settings are changed.

Retry a failed check

You can retry a failed external status check if you have the **Developer** role or higher in the project.

Retry is available only for failed checks that belong to the current `HEAD` SHA of the merge request source branch.

To retry a failed check:

1. Open the merge request.
2. Find the “External status checks” widget.
3. In the failed check row, select “Retry”.

After retry, Deckhouse Code changes the check back to `pending` and sends the current merge request payload to the external service again.

Use retry after the external service issue has been fixed or when the failure was temporary.

Check URLs

The widget can show the external service URL for a check. The URL is visible only to users whose role allows viewing external status check URLs. In standard project roles, this is available to users with the **Developer** role or higher.

If you cannot see the URL, you can still see the check status if your role allows viewing status check responses.

Access to check results

Access to external status check information depends on your project role:

Action	Minimum role
Create, update, delete, or list project status check services	**Maintainer** or **Owner**
Send a status check callback	**Developer**
View check responses in the widget	**Reporter**
View the external service URL	**Developer**
Retry a failed check	**Developer**

For projects with `internal` visibility, an authenticated user who is not marked as external can read external status check responses in the merge request page widget if they have read access to that merge request.

The external check URL is shown only to users with permission to view external status check URLs (in standard roles, `Developer` or higher), so it is not shown to a regular user who is not a project member.

Deleting an external status check

To delete a check:

1. Open the project.
2. Go to “Settings” → “Merge requests”.
3. In “External status checks”, select “Delete” for the required check.
4. Confirm the deletion.

After deletion, the check no longer applies to project merge requests.

Audit events

Deckhouse Code records audit events for status check management and response changes.

Audit event	Description
`external_status_check_created`	An external status check was created
`external_status_check_updated`	An external status check was updated
`external_status_check_destroyed`	An external status check was deleted
`external_status_check_response_updated`	A response was changed by an external callback, retry, request failure, or timeout

Troubleshooting

Merge request is blocked by an external status check

Check the following:

- The “Status checks must succeed” checkbox is enabled.
- The check applies to the merge request target branch.
- The external service sent `passed` for the current `HEAD` SHA.
- The timeout did not expire before the external service callback.

Check failed because of timeout

Check the following:

- The project-level “Status checks timeout” value.
- The individual “Timeout minutes” value of the check.
- The external service response time.
- Whether the check should be retried after the external service is fixed.

A check stays pending

A check can stay pending while Deckhouse Code waits for the external service.

Check the following:

- Whether the external service is available.
- Whether the service can process the merge request payload.
- Whether the service sends a response for the current `HEAD` SHA.
- Whether the configured timeout is long enough for the service to finish processing.

If the timeout expires, the check becomes `failed`.

Request to the external service failed

Check the following:

- The URL in “API to check”.
- Network access from Deckhouse Code to the external service.
- Whether the URL uses `http` or `https`.
- The error text in the merge request widget.

Retry is not available

The “Retry” action is shown only when all of the following conditions are true:

- The check has the `failed` status.
- The check belongs to the current merge request `HEAD` SHA.
- You have the **Developer** role or higher in the project.

If retry is not available, ask a project maintainer to check the project settings and your role.

The external service URL is not visible

The URL is hidden if your role does not allow viewing external status check URLs. Ask a project maintainer if you need access to the external service URL.

Duplicate name or URL error

A project cannot have two external status checks with the same “Service name” or “API to check” value. Use a unique value for each check.

Merge trains

A merge train forms a queue of merge requests targeting the same branch. Before merging, each merge request is validated together with the merge requests ahead of it in the queue. This ensures that only changes validated in their merge order are merged into the target branch.

Merged results pipelines are used for validation. For the first merge request in the queue, such a pipeline validates its changes against the current state of the target branch. For each subsequent merge request, changes from the preceding merge requests in the queue are also taken into account. Therefore, if merge requests pass validation separately but their changes are incompatible, the issue is detected before the changes are merged into the target branch.

Merge trains are configured separately for each project. A separate merge train is used for each target branch in the project.

How merge trains work

Merge requests are arranged in the queue in the order they were added. Merge requests added earlier are placed at the front of the queue.

Merge requests in the queue are processed as follows:

1. A merge request with auto-merge enabled is added to the end of the queue.
2. Deckhouse Code creates a temporary Git ref for the merge request and runs a pipeline for it. The Git ref contains the state of the target branch, changes from the preceding merge requests in the queue, and changes from the current merge request.
3. If the pipeline for the first merge request in the queue completes successfully, the merge request is merged into the target branch and leaves the queue.
4. The next merge request moves to the front of the queue. If merging the previous merge request makes the state for which the pipeline was run outdated, Deckhouse Code recreates the pipeline for the new state. Pipelines for subsequent merge requests are also recreated.

A merge train is not stored as a separate object. It is formed from merge requests added to the queue for the same target branch of a project. When a merge request is merged or removed from the queue, the positions of the remaining merge requests are recalculated.

Prerequisites

Before configuring merge trains, make sure the following requirements are met:

- CI/CD is enabled in the project.
- The CI/CD configuration file is configured to create pipelines for merge requests.
- Merged results pipelines are enabled in the project.
- The user has the **Maintainer** or **Owner** role to modify project settings.

Enabling merge trains

To enable merge trains:

1. Open the project.
2. Go to “Settings” → “Merge requests”.
3. In the “Merge options” section, select “Enable merged results pipelines”.
4. Select “Enable merge trains”.
5. Click “Save changes”.

After merge trains are enabled, the following additional settings become available:

Setting	Description
Merge immediately without restarting the merge train	Adds the immediate merge options described in Merging immediately. This setting is unavailable if the project requires fast-forward or semi-linear merges, or if merging through a merge train is enforced
Require merge requests to merge through a merge train	Rejects all direct merges in the project. For details, refer to Enforcing merge trains
Maximum parallel pipelines per merge train	Limits the number of merge train pipelines that can run simultaneously. For details, refer to Limiting parallel pipelines

Clearing “Enable merge trains” does not immediately clear the existing queue. Merge requests are removed gradually as the update cycle processes them, so they may remain visible in the queue for some time after the setting is saved.

Working with the queue

You can add merge requests to the queue, view or remove them, or merge them immediately.

Adding a merge request to a merge train

You can add a merge request to a merge train using the web interface or API.

Adding via web UI

To add a merge request to a merge train, click “Set to auto-merge” on the merge request page. If merge trains are enabled in the project, the merge request is added to the merge train instead of being merged directly.

Depending on the state of the merge request, one of the following options is available:

Option	Description
Add to merge train	Adds the merge request to the queue immediately. This option is available if the merge request is ready to merge and the pipeline for the current `HEAD` SHA has completed
Add to merge train when all merge checks pass	Adds the merge request to the queue after all merge checks pass

The option that will be used is displayed next to the merge button. If the merge request has already been validated by a merge train pipeline, “Re-add to merge train” is displayed.

If the latest pipeline for the merge request failed, Deckhouse Code asks for confirmation before adding the merge request to the queue.

Adding via API

To add a merge request to a merge train via API, run the following request:

```
curl --request POST --header "PRIVATE-TOKEN: <TOKEN>" \
  "https://<HOST>/api/v4/projects/<PROJECT_ID>/merge_trains/merge_requests/
<MERGE_REQUEST_IID>"
```

Where:

- `<TOKEN>`: Token used to authenticate with Deckhouse Code.
- `<HOST>`: Deckhouse Code instance address.
- `<PROJECT_ID>`: Project ID.
- `<MERGE_REQUEST_IID>`: Internal ID of the merge request within the project.

The following parameters can be specified in the request:

Parameter	Description
`sha`	SHA expected in the source branch. If the current SHA does not match the specified value, the request is rejected
`squash`	Squashes the source branch commits into a single commit when the merge request is merged. When merging through a merge train, the merge request's squash setting is also taken into account
`auto_merge`	Adds the merge request to the queue only after all merge checks pass

Possible response codes:

Code	Description
`201`	Merge request added to the queue
`202`	Merge request is waiting for merge checks to pass and has not yet been added to the queue
`400`	Merge trains are disabled in the project, or the current state of the merge request does not allow it to be added to the queue
`401`	Request was made without authentication
`403`	User role does not allow viewing the merge train or merging this merge request
`404`	Project or merge request not found
`409`	Auto-merge is already enabled for the merge request

Viewing a merge train

You can get information about a merge train using the web interface or API.

Viewing via web UI

You can open the project's merge trains page in one of the following ways:

- On the page of a merge request added to the queue, follow the "View merge train" link.
- On the merge train pipeline page, follow the "View merge train details" link in the page header.
- Open the page directly at `https://<HOST>/<NAMESPACE>/<PROJECT>/-/merge_trains`.

The merge trains page displays:

- A filter for selecting the target branch. The project's default branch is selected initially.
- The "Active" tab with merge requests currently in the queue and the "Merged" tab with merge requests that have already been merged. Each tab shows the number of merge requests.
- Information about each merge request: pipeline status, title, time when the merge request was added to the queue or merged, and the user who added or merged it.

While a merge request is in the queue, its position is displayed on the merge request page:

- For the first merge request: "A new merge train has started, and this merge request is first in the queue".
- For other merge requests, the position in the queue is displayed, for example, "This merge request is number 2 of 5 in the queue".

Viewing via API

To get information about merge trains via API, run one of the following requests:

```
# All merge trains in the project.
curl --header "PRIVATE-TOKEN: <TOKEN>" \
  "https://<HOST>/api/v4/projects/<PROJECT_ID>/merge_trains"

# Merge train for a specific target branch.
curl --header "PRIVATE-TOKEN: <TOKEN>" \
  "https://<HOST>/api/v4/projects/<PROJECT_ID>/merge_trains/<TARGET_BRANCH>"

# Status of a specific merge request in the queue.
curl --header "PRIVATE-TOKEN: <TOKEN>" \
  "https://<HOST>/api/v4/projects/<PROJECT_ID>/merge_trains/merge_requests/
<MERGE_REQUEST_IID>"
```

Where:

- `<TOKEN>`: Token used to authenticate with Deckhouse Code.
- `<HOST>`: Deckhouse Code instance address.
- `<PROJECT_ID>`: Project ID.
- `<TARGET_BRANCH>`: Target branch name.
- `<MERGE_REQUEST_IID>`: Internal ID of the merge request within the project.

The first two endpoints accept the following parameters:

Parameter	Description
`scope`	Limits the response to merge requests that are still in the queue (`active`) or have already left it (`complete`)
`sort`	Specifies the order of merge requests by their position in the queue: `asc` sorts from oldest to newest, and `desc` from newest to oldest. The default value is `desc`

Both endpoints support pagination using the `page` and `per_page` parameters. The first endpoint returns merge requests for all target branches. To get the queue for a specific branch only, use the second endpoint and specify the branch name.

You can also get information about project merge trains and queues using the GraphQL API.

Removing a merge request from a merge train

To remove a merge request from a merge train, use one of the following methods:

- Cancel auto-merge on the merge request page.
- On the merge trains page, click the remove icon in the merge request row, then click “Remove from merge train”.

When a merge request is removed from the queue, pipelines for the merge requests behind it are recreated because the state of the merge train has changed.

A merge request cannot be removed from the merge train if its merge has already started. In this case, Deckhouse Code rejects the removal and completes the merge.

Deckhouse Code also automatically removes a merge request from the merge train if it can no longer be merged. For details, refer to Merge request removed from the merge train.

Merging immediately

You can merge a merge request immediately without waiting for its turn in the merge train. Immediate merging is available through the web interface and API.

Merging via web UI

If “Merge immediately without restarting the merge train” is enabled in the project, two additional options are available next to the merge button on the page of a merge request in the queue:

Option	Description
Merge now and restart train	Merges the merge request immediately and recreates the pipelines for subsequent merge requests to include the merged changes
Merge now and don’t restart train	Merges the merge request immediately while existing pipelines continue running without being restarted. As a result, merge requests already in the queue are not validated together with the merged changes

> ⚠ Use “Merge now and don’t restart train” only if the merged changes cannot affect merge requests already in the queue.

Both options require confirmation and allow the merge request to be merged outside the established queue order.

Immediate merge options are unavailable if merging through a merge train is enforced in the project. The “Merge now and restart train” setting is also unavailable if the project requires fast-forward or semi-linear merges.

The result of an immediate merge depends on the selected option:

- When merging without restarting the train, the merge request appears on the “Merged” tab with the `skip_merged` status.
- When merging with a train restart, the merge request is merged directly and does not appear on the “Merged” tab. After the merge, it is removed from the queue, and the removal is recorded in the merge request activity.

Merging via API

To merge a merge request immediately via API, use the merge endpoint with the `skip_merge_train` parameter:

```
curl --request PUT --header "PRIVATE-TOKEN: <TOKEN>" \
  "https://<HOST>/api/v4/projects/<PROJECT_ID>/merge_requests/<MERGE_REQUEST_IID>/
merge?skip_merge_train=true"
```

Where:

- `<TOKEN>`: Token used to authenticate with Deckhouse Code.
- `<HOST>`: Deckhouse Code instance address.
- `<PROJECT_ID>`: Project ID.
- `<MERGE_REQUEST_IID>`: Internal ID of the merge request within the project.

The `skip_merge_train` parameter accepts the following values:

Value	Description
`true`	Merges the merge request immediately without restarting the train
`false`	Default. Merges the merge request immediately and recreates the pipelines for subsequent merge requests

The parameter takes effect only when “Merge immediately without restarting the merge train” is enabled. Otherwise, the parameter is ignored. If merging through a merge train is enforced in the project, the merge request is rejected.

Configuring merge trains

In the project settings, you can enforce merging through a merge train and limit the number of pipelines that can run simultaneously.

Enforcing merge trains

To prevent merge requests from being merged directly in the project, select “Require merge requests to merge through a merge train”.

After this setting is enabled:

- Immediate merge options become unavailable.
- An API merge request is rejected if auto-merge is not enabled for it and it has not been added to a merge train.

- Merges performed by the merge train itself continue to work as usual.

When merge trains are enforced, “Merge immediately without restarting the merge train” is automatically disabled and becomes unavailable. After the settings are saved with merge trains enforced, this setting remains disabled even if enforcement is later turned off. To use it again, enable the setting manually.

Merge train enforcement applies only when merge trains are enabled. If merge trains are disabled, merge requests can be merged directly.

The restriction applies to all users, including project owners and administrators.

Limiting parallel pipelines

Pipelines for multiple merge requests in a merge train can run simultaneously. The number of pipelines that can run at the same time is determined by two limits:

- The Deckhouse Code instance limit configured by the administrator. The default is `20`.
- The “Maximum parallel pipelines per merge train” project setting.

The lower of the two values is used. For example, if the instance limit is `20` and the project limit is `10`, no more than 10 pipelines can run simultaneously.

To use the instance limit, leave the project setting empty. You can specify a project value from `1` to `500`, but it cannot increase the limit configured for the instance.

Merge train pipelines

For each merge request in the queue, Deckhouse Code runs a pipeline for a Git ref. This pipeline does not run for the merge request’s source branch. In the pipeline list, it is labeled `merge train`, which distinguishes it from a regular merged results pipeline.

A completed merge train pipeline cannot be restarted because the repository state for which it ran may have changed. To run a new pipeline, re-add the merge request to the merge train.

A pipeline is recreated in the following cases:

- A merge request ahead of it in the queue is merged, removed from the queue, or gets a new pipeline.
- The pipeline for the preceding merge request in the queue fails.
- Changes are pushed directly to the target branch. In this case, pipelines for all merge requests in the queue are recreated, starting with the first one.

Running jobs only in merge train pipelines

In a merge train pipeline, the `CI_MERGE_REQUEST_EVENT_TYPE` variable is set to `merge_train`. You can use it to run a job only in merge train pipelines:

```
integration-tests:
  script: ./run-integration-tests.sh
  rules:
    - if: $CI_MERGE_REQUEST_EVENT_TYPE == "merge_train"
```

To skip a job in merge train pipelines instead, use the opposite condition:

```
quick-lint:
  script: ./lint.sh
  rules:
    - if: $CI_MERGE_REQUEST_EVENT_TYPE != "merge_train"
```

Merge request statuses in the queue

Each merge request in a merge train has a status that can be retrieved through the REST API or GraphQL API. The statuses are not displayed directly in the web interface. Instead, merge requests are grouped under the “Active” and “Merged” tabs.

Status	Description
`idle`	Merge request is in the queue, but its pipeline has not started yet
`fresh`	Pipeline corresponds to the current merged result
`stale`	The state of the merge train has changed and the pipeline is being recreated
`merging`	Merge request is being merged
`merged`	Merge request was merged through the merge train and removed from the queue
`skip_merged`	Merge request was merged immediately without restarting the merge train and removed from the queue

Merge requests with the `idle`, `fresh`, and `stale` statuses are displayed on the “Active” tab and can be removed from the queue. Merge requests with the `merged` and `skip_merged` statuses are displayed on the “Merged” tab. A merge request with the `merging` status is not displayed on either tab until the merge is complete.

Merge request activity

Deckhouse Code records events related to a merge request in a merge train as system notes in the merge request's "Activity" section. The following events are recorded:

- The merge request started a new merge train or was added to an existing merge train at a specific position in the queue.
- The merge request was removed from the merge train by a user.
- The merge request was removed from the merge train by the system, with the reason specified.
- Automatic addition to the merge train after merge checks pass was enabled, canceled, or interrupted, with the reason specified.

If a merge request is removed from the merge train by the system, its participants also receive a corresponding to-do item in their to-do list.

Access to merge trains

Available merge train actions depend on the user's role and project settings:

Action	Requirements
Viewing the merge trains page and retrieving information via API	Merge trains are enabled in the project, and the user has read access to merge requests and pipelines. With standard roles, this is available to users with the **Reporter** role or higher
Retrieving the merge train status for a specific merge request	Same requirements as for viewing a merge train
Adding a merge request to a merge train	The user has permission to merge the merge request. With standard roles, this is available to users with the **Developer** role or higher
Removing a merge request from the queue on the merge trains page	The user has permission to cancel pipelines, modify the merge request, and merge into the target branch. With standard roles, this is available to users with the **Developer** role or higher
Modifying merge train settings in the project	**Maintainer** or **Owner** role

Authentication is required to view merge trains through the web interface or API. Merge trains are not available to anonymous users, even in public projects.

Audit events

Deckhouse Code records the following audit events when merge train settings are changed in a project:

Audit event	Description
`project_cicd_merge_trains_enabled_updated`	The setting that enables merge trains in the project was changed
`project_cicd_merge_train_enforced_updated`	The setting that enforces merging through a merge train was changed

Troubleshooting

This section describes common issues with merge trains and how to resolve them.

Merge request removed from the merge train

If a merge request can no longer be merged while its pipeline is running, Deckhouse Code removes it from the queue to prevent it from blocking other merge requests. The reason for removal is specified in a system note in the “Activity” section.

The most common reasons for removal are:

- Merge trains are disabled in the project.
- New commits were added to the merge request’s source branch.
- The merge request was closed.
- The merge request was marked as a draft.
- The target branch of the merge request was changed.
- The merge request can no longer be merged, for example, due to a conflict.
- Auto-merge was canceled for the merge request.
- The merge train pipeline for the merge request failed.
- The account of the user who added the merge request to the queue was deleted.

If a merge request is waiting for merge checks to pass before being added to the merge train, the process is also interrupted if the merge request becomes a draft or its pipeline fails.

Discussions and approvals are checked when a merge request is added to the queue. A new discussion opened afterward does not block the merge, even if the project requires all discussions to be resolved. This preserves the state already validated by the pipeline and avoids recreating pipelines for subsequent merge requests.

If a required approval is revoked, the merge is blocked. The merge request retains its position in the queue but is removed from the merge train when it reaches the front of the queue.

Resolve the issue specified in the system note and re-add the merge request to the merge train.

Merge button does not offer merge train options

Check the following:

- Merged results pipelines and merge trains are enabled in the project settings.
- The CI/CD configuration file creates pipelines for merge requests.

- The merge request has a completed pipeline for the current `HEAD` SHA of the source branch.
- The user's role allows them to merge the merge request.

Merge rejected in a project that requires merge trains

If "Require merge requests to merge through a merge train" is enabled, direct merges are rejected, including merges through the API. Instead, add the merge request to the merge train.

The restriction applies to all users, including the project owner. To allow direct merges again, disable this setting.

Merge train pipeline cannot be restarted

A completed merge train pipeline cannot be restarted because the state for which it ran may have changed. To create a pipeline for the current state, re-add the merge request to the merge train.

Merge appears to be stuck

If the merge process is interrupted, Deckhouse Code automatically returns the merge request to the queue and continues processing the merge train. No additional action is required.

Merge trains are disabled, but the queue is still displayed

After merge trains are disabled, merge requests are not removed from the existing queue immediately. They may remain visible on the page for some time while Deckhouse Code gradually clears the queue.

Merge trains page is empty

Check the following:

- The target branch whose queue you want to view is selected in the filter.
- Auto-merge is enabled for the merge requests.
- Merge trains are enabled in the project. If they are disabled, the page is unavailable even if the queue remains from its previous state.

Remove action is unavailable

A merge request can be removed from the merge train only on the "Active" tab and only if you have permission to cancel pipelines, modify the merge request, and merge into the target branch.

A merge request cannot be removed from the merge train if its merge has already started.

Code review analytics

The report shows how long the open merge requests of a project have been waiting for review.

What reaches the report

The report lists the open merge requests of the project that have at least one comment from a reviewer of the request (any user other than the author).

Merged, closed, and merge requests without comments are absent from the report.

Bot comments and system notes (a label change, a reviewer assignment) do not count as the start of review, so a merge request that has only such comments stays out of the report.

Viewing the report

Open the project and select “Analyze” → “Code review analytics” in the sidebar.

Counting review time

Review time runs from the first comment left by a reviewer to the current moment.

The value is approximate and is shown in minutes, hours, or days. The start of review is recorded when the comment is saved. A merge request that waited eight months for review, had system comments, and got a comment from a reviewer an hour ago shows a review time of “about 1 hour”.

Filters

The filter bar above the table lets you filter the report by milestone and by label. The “Milestone” filter accepts one value, the “Label” filter accepts several values.

List of merge requests

The “Merge requests in review” table lists the merge requests under review, ordered from the earliest by creation date. The counter next to the table heading shows how many merge requests the filters leave under review.

A page holds no more than 20 merge requests; when the total exceeds that, pagination controls are displayed below the table. Changing a filter returns the table to its first page.

The table has the following columns:

Column	Contents
“Merge request”	Title with a link to the merge request, its number, the number of labels and comments, and the number of approvals if there are any
“Review time”	Time since the first comment left by a reviewer
“Author”	Author of the merge request
“Approvers”	Users who approved the merge request
“Comments”	Number of comments
“Commits”	Number of commits
“Line changes”	Number of added and removed lines

When the filters leave no merge request under review, the table is replaced with the message “No merge requests are under review”.

Access to the report

The report is available to users with the “Reporter” role or higher. In a public project it is available to everyone who can see the project, including anonymous users.

When the “Analytics” feature is turned off in the project settings, the report and its menu item become unavailable.

Merge request analytics

The report shows how many merge requests of a project were merged over a selected period and how long merging took. The report contains the “Mean time to merge” card, the “Throughput” chart, and the “Merge requests” table.

What reaches the report

The report counts the merged merge requests of the project that fall inside the selected date range. Open merge requests and merge requests closed without merging are absent from the report.

A merge request created before the range and merged inside it is counted in full: the whole interval from creation to merge goes into the mean time to merge.

Viewing the report

Open the project and select “Analyze” → “Merge request analytics” in the sidebar.

Date range

The range is set by one of the following options, counting back from the current moment:

- “Last 7 days”
- “Last 30 days”
- “Last 90 days”
- “Last 180 days”
- “Last 365 days” (the default)

With “Custom range”, the dates are set in the “From” and “To” fields. The range spans at most 365 days.

The selected range is displayed next to the headings “Mean time to merge”, “Throughput”, and “Merge requests”.

Throughput chart

The “Throughput” chart plots the number of merged merge requests by month. In the first and the last month of the range, only the days inside the range are counted: for a range that starts on 15 March, the first point of the chart covers 15–31 March.

When the range and the filters leave no merged merge requests, the chart is replaced with the message “No merge requests were merged in this date range”.

Mean time to merge

The “Mean time to merge” card shows the mean time from the creation of a merge request to its merge over the whole range, in days.

The mean covers merge requests whose merge is later than their creation. A merge request whose merge time matches its creation time or precedes it, for example one imported with `merged_at` already filled in, is counted in the chart and listed in the table. It stays out of the mean. When the range holds no merge request with a merge later than its creation, the card shows a dash (`-`).

Filters

The filter bar narrows the chart, the mean time to merge, and the table at once. The following filters are available:

- “Source branch”
- “Target branch”
- “Milestone”
- “Label”
- “Author”
- “Assignee”

The “Label” filter accepts several values, and the report keeps the merge requests that carry all the selected labels.

Merge request table

The “Merge requests” table lists the merged merge requests of the range, starting with the most recently merged one. A page holds 20 rows. When the range holds more merge requests, pagination controls are displayed below the table. Changing a filter returns the table to its first page.

The table has the following columns:

Column	Contents
“Merge request”	Title with a link to the merge request, its number, the pipeline status, the number of labels and comments, and the number of approvals if there are any
“Date merged”	Date the merge request was merged
“Time to merge”	Approximate interval between the creation of the merge request and its merge, shown in minutes, hours, or days
“Milestone”	Milestone of the merge request with a link to it
“Commits”	Number of commits in the merge request
“Pipelines”	Number of pipelines of the merge request
“Line changes”	Number of added and removed lines
“Assignees”	Avatars of the assignees

When the range and the filters leave no merged merge requests, the table is replaced with the message “No merge requests were merged in this date range” and a suggestion to widen the range or clear the filters.

Access to the report

The report is available to users with the “Reporter” role or higher. Users with the “Guest” role and anonymous users cannot open the report, even in a public project.

When the “Analytics” feature is turned off in the project settings, the report and its menu item become unavailable.

CI/CD

CI/CD pipelines

Deckhouse Code uses the `.gitlab-ci.yml` file to automate testing, build, and deployment processes through CI/CD pipelines. This enables the following functionality:

- Automatic code checks on every commit.
- Flexible configuration of execution stages and jobs.
- Parallel job execution to speed up build and test processes.
- Custom test steps tailored to project requirements.

- Pipeline triggers on various repository events such as commits, merge requests, tag creation, and more.

External Vault integration

This feature allows you to configure integration with a Vault server and use secrets in CI pipelines. Before getting started, you need to configure the Vault server and create the appropriate roles and policies.

Vault configuration

1. Enable JWT authentication:

```
vault auth enable jwt

vault write auth/jwt/config \
  oidc_discovery_url="https://code.example.com" \
  bound_issuer="https://code.example.com" \
  default_role="code-role"
```

2. Create a role:

```
vault write auth/jwt/role/code-role - <<EOF
{
  "role_type": "jwt",
  "user_claim": "sub",
  "bound_audiences": ["vault"],
  "bound_claims": {
    "project_id": "23"
  },
  "policies": ["code-policy"],
  "ttl": "1h"
}
EOF
```

> Always use `bound_claims` to restrict access to the role. Otherwise, any JWT issued by the platform will be able to authenticate using this role.

3. Create a policy:

```
vault policy write code-policy - <<EOF
path "kv/data/code/vault-demo" {
  capabilities = ["read"]
}
EOF
```

CI configuration

Environment variables

To work correctly with Vault in a CI/CD pipeline, you need to define the following environment variables:

- `VAULT_SERVER_URL` — **required**. The URL of the Vault server (e.g., `https://vault.example.com`).
- `VAULT_AUTH_ROLE` — *optional*. The name of the role in Vault. If not specified, the default role configured for the authentication method will be used.
- `VAULT_AUTH_PATH` — *optional*. Path to the authentication method in Vault. Defaults to `jwt`.
- `VAULT_NAMESPACE` — *optional*. Vault namespace, if a multi-level hierarchy is used.

Alternative variable names

Each of these variables also has an alternative name with the `STRONGHOLD_` prefix. A project that keeps CI/CD secrets in Deckhouse Stronghold can configure the connection under these names:

Variable	Alternative name
`VAULT_SERVER_URL`	`STRONGHOLD_SERVER_URL`
`VAULT_AUTH_ROLE`	`STRONGHOLD_AUTH_ROLE`
`VAULT_AUTH_PATH`	`STRONGHOLD_AUTH_PATH`
`VAULT_NAMESPACE`	`STRONGHOLD_NAMESPACE`

Only the names in the table configure the connection. A variable such as `STRONGHOLD_TOKEN` is not read.

The values are resolved by the following rules:

- Each setting is resolved separately, so the server URL can come from `STRONGHOLD_SERVER_URL` while the role comes from `VAULT_AUTH_ROLE`.
- When both names are set for the same setting, the value of the variable with the `VAULT_` prefix is used.
- An empty value in a `STRONGHOLD_` variable counts as unset, so an empty `STRONGHOLD_AUTH_PATH` keeps the default path `jwt`.

The alternative names affect the Vault connection only. A job script that reads `$VAULT_SERVER_URL` gets a value only when a variable with that name is set.

Using secrets in CI

To retrieve secrets from Vault, you can use the following job template:

```
stages:
  - test
vault-login:
  stage: test
  image: ruby:3.2
  id_tokens:
    VAULT_ID_TOKEN:
      aud: vault
  secrets:
    DATABASE_PASSWORD:
      vault: code/vault-demo/DATABASE_PASSWORD@kv
      token: $VAULT_ID_TOKEN
  script: echo $DATABASE_PASSWORD
```

Secret parameters

Example:

```
DATABASE_PASSWORD:
  vault: code/vault-demo/DATABASE_PASSWORD@kv
  token: $VAULT_ID_TOKEN
  file: false
```

Parameter details:

1. `vault` (required) — the path to the secret in the string format `path/to/secret/KEY@ENGINE`, where:
    - `code/vault-demo/` — the path to the secret in Vault;
    - `DATABASE_PASSWORD` — the name of the field inside the secret;
    - `kv` — the mount point of the secret engine (default is `secret`).

By default, the `kv-v2` engine is used. If you need to use a different engine, you can specify it as an object instead of a string:

```
DATABASE_PASSWORD:
  vault:
    path: code/vault-demo
    field: DATABASE_PASSWORD
    engine:
      name: 'kv-v1'
      path: 'kv1'
  token: $VAULT_ID_TOKEN
  file: false
```

1. `token` (required) — a JWT token from the `id_tokens` section used to authenticate with Vault.

2. `file` (optional, defaults to `true`) — defines how the secret is provided:
    - `true` — the secret is saved to a temporary file;
    - `false` — the secret is passed as a string to an environment variable.

JWT claims

The following fields are automatically included in the JWT token and can be used by Vault to validate access rights:

Claim	Availability condition	Description
`jti`	always	Unique token identifier
`iss`	always	Token issuer (typically the Deckhouse Code URL)
`iat`	always	Token issue time (`Issued At`)
`nbf`	always	Time before which the token is not valid
`exp`	always	Token expiration time
`sub`	always	Token subject (usually the CI job ID)
`namespace_id`	always	ID of the namespace (group or user space)
`namespace_path`	always	Path to the namespace (e.g., `groups/dev`)
`project_id`	always	Project ID
`project_path`	always	Path to the project
`user_id`	always	User ID
`user_login`	always	User login
`user_email`	always	User email
`pipeline_id`	always	CI pipeline ID
`pipeline_source`	always	Pipeline trigger source (push, schedule, merge request, etc.)
`job_id`	always	CI job ID
`ref`	always	Git reference (e.g., `main` , `v1.2`)
`ref_type`	always	Git reference type (`branch` or `tag`)
`ref_path`	always	Full Git reference path (e.g., `refs/heads/main`)

Claim	Availability condition	Description
`ref_protected`	always	Indicates if the Git reference is protected
`environment`	if available	Environment name (if used)
`groups_direct`	if available (<200 groups)	Paths to groups the user is directly a member of
`environment_protected`	if available	Indicates if the environment is protected
`deployment_tier`	if available	Environment type (`production` , `staging` , etc.)
`environment_action`	if available	Action being performed on the environment (e.g., `deploy`)

Quick start

This section provides an example of the minimum required configuration for integrating HashiCorp Vault with Deckhouse Code and verifying that a CI job can retrieve secrets from Vault.

> ⚠ This example is provided for demonstration purposes only. It does not reflect security best practices and uses a simplified configuration to allow you to quickly verify that the integration works.

Step 1. Set environment variables

Set the environment variables for Vault and Deckhouse Code. Some parameters can be left unchanged, but `VAULT_ADDR` , `VAULT_TOKEN` , `CODE_URL` , and `PROJECT_PATH` must be set manually.

```
export VAULT_ADDR="https://vault.example.com"
export VAULT_TOKEN="<TOKEN>"

# Deckhouse Code URL.
export CODE_URL="https://code.example.com"

# Vault role and policy names.
export VAULT_ROLE="code-role"
export VAULT_POLICY="code-policy"

# Secret path and data.
export VAULT_SECRET_PATH="code/vault-demo"
export VAULT_SECRET_FIELD="DATABASE_PASSWORD"
export VAULT_SECRET_VALUE="super-secret-password"

# Value of the project_path claim that Vault will validate.

export PROJECT_PATH="root/my-pr"
```

Step 2. Enable the JWT authentication method

Enable the JWT authentication method in Vault. Without it, Vault will not be able to accept ID tokens that Deckhouse Code passes to CI jobs.

```
curl \
  -H "X-Vault-Token: $VAULT_TOKEN" \
  -X POST "$VAULT_ADDR/v1/sys/auth/jwt" \
  -d '{"type":"jwt"}'
```

Step 3. Configure JWT and OIDC

The following request:

- Sets the OIDC discovery URL (`$CODE_URL`).
- Specifies the expected issuer.
- Defines the *default role* that Vault issues during authentication.

```
curl \
  -H "X-Vault-Token: $VAULT_TOKEN" \
  -X POST "$VAULT_ADDR/v1/auth/jwt/config" \
  --data @- <<EOF
{
  "oidc_discovery_url": "$CODE_URL",
  "bound_issuer": "$CODE_URL",
  "default_role": "$VAULT_ROLE"
}
EOF
```

Step 4. Mount the KV v2 secret engine

KV v2 is the most commonly used Vault secret engine. The following request enables it at the `/kv` path:

```
curl \
  -H "X-Vault-Token: $VAULT_TOKEN" \
  -X POST "$VAULT_ADDR/v1/sys/mounts/kv" \
  -d '{"type": "kv-v2"}'
```

Step 5. Create a test secret

Create a secret that will later be read by a CI job. The secret is stored at the `code/vault-demo` path and contains a single field.

```
curl \
  -H "X-Vault-Token: $VAULT_TOKEN" \
  -X POST "$VAULT_ADDR/v1/kv/data/$VAULT_SECRET_PATH" \
  --data @- <<EOF
{
  "data": {
    "$VAULT_SECRET_FIELD": "$VAULT_SECRET_VALUE"
  }
}
EOF
```

Step 6. Create an ACL policy

The policy defines which paths in Vault can be accessed. In the following example, the policy grants read-only access to the specified secret.

```
curl \
  -H "X-Vault-Token: $VAULT_TOKEN" \
  -X PUT "$VAULT_ADDR/v1/sys/policies/acl/$VAULT_POLICY" \
  --data @- <<EOF
{
  "policy": "path \"kv/data/$VAULT_SECRET_PATH\" { capabilities = [\"read\"] }"
}
EOF
```

Step 7. Create a Vault role

The role defines:

- Authentication type
- Required claims in the token (`project_path`)
- Policies granted to the authenticated subject
- Allowed audiences (`aud`)

- Token TTL

Deckhouse Code will issue an ID token with `aud=vault`, and Vault will verify that the `project_path` value matches the configured one.

```
curl \
  -H "X-Vault-Token: $VAULT_TOKEN" \
  -X POST "$VAULT_ADDR/v1/auth/jwt/role/$VAULT_ROLE" \
  --data @- <<EOF
{
  "role_type":   "jwt",
  "user_claim":  "sub",
  "bound_audiences": ["vault"],

  "bound_claims": {
    "project_path": "$PROJECT_PATH"
  },

  "policies": ["$VAULT_POLICY"],
  "ttl": "1h"
}
EOF
```

At this point, the Vault configuration is complete.

Testing the integration in Deckhouse Code

1. Open the project specified in `PROJECT_PATH`. The project CI token must match the `project_path` claim. Otherwise, Vault will deny access to secrets.
2. In the project CI/CD settings, add the `VAULT_SERVER_URL` variable with the value of `$VAULT_ADDR` used earlier. This variable tells Deckhouse Code where to send Vault API requests.
3. Create the `.gitlab-ci.yml` file. The file runs a test CI job that:
    - Obtains an ID token with `aud=vault`.
    - Passes it to Vault.
    - Retrieves a secret from KV.
    - Outputs the secret value.

```
stages:
  - test

vault-demo:
  stage: test
  image: alpine
  id_tokens:
    VAULT_ID_TOKEN:
      aud: vault
  secrets:
    DATABASE_PASSWORD:
      vault: code/vault-demo/DATABASE_PASSWORD@kv
      token: $VAULT_ID_TOKEN
      file: false
  script:
    - echo "Raw value (masked by Deckhouse Code):"
    - echo "$DATABASE_PASSWORD"

    - echo
    - echo "Value with spaces (not masked):"
    - printf '%s\n' "$DATABASE_PASSWORD" | sed 's/./& /g'
```

Result

If the integration is configured correctly:

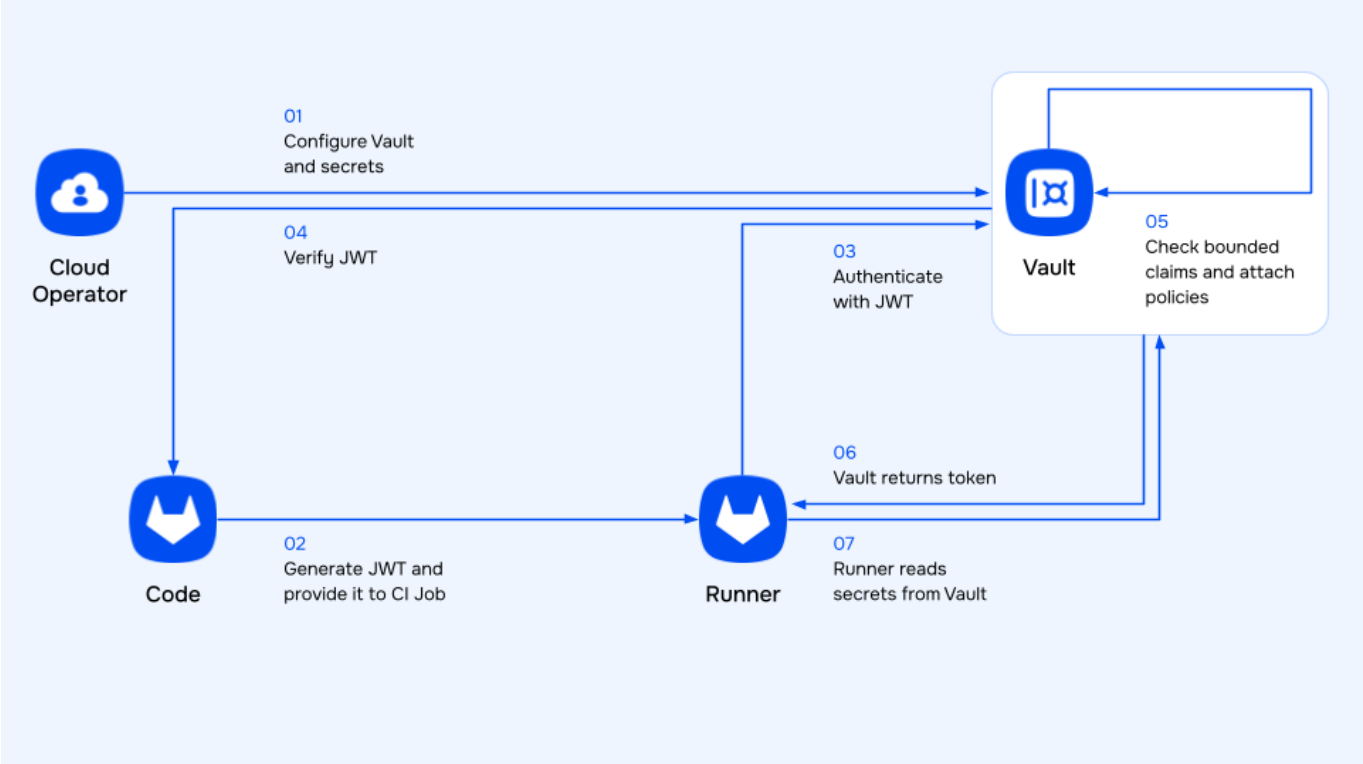
- The CI job successfully obtains an ID token.
- Vault validates the `project_path` value and grants access.
- The secret is retrieved and printed to the log.

In the job output, you will see the value of the `DATABASE_PASSWORD` field loaded directly from Vault.

Deckhouse Stronghold integration

Deckhouse Stronghold is an enterprise secrets management solution built on top of HashiCorp Vault. Integrating Stronghold with Deckhouse Code allows you to securely use secrets in CI/CD pipelines without storing them in project variables.

Interaction flow



Integration benefits:

- **Centralized secrets management** — all secrets are stored in one place with access auditing.
- **Automatic rotation** — secrets can be automatically rotated without modifying pipelines.
- **Granular access control** — access to secrets is restricted based on project, branch, environment, and other parameters.
- **No secrets in the repository** — secrets never end up in the code or CI variables.

Usage examples

This section provides common scenarios for using Vault secrets in CI/CD pipelines.

Deploying an application with database credentials

Example of retrieving PostgreSQL credentials during deployment:

```
stages:
  - deploy

deploy-production:
  stage: deploy
  image: alpine/k8s:1.28.0
  environment:
    name: production
  id_tokens:
    VAULT_ID_TOKEN:
      aud: vault
  secrets:
    DB_HOST:
      vault: apps/myapp/production/DB_HOST@kv
      token: $VAULT_ID_TOKEN
      file: false
    DB_USER:
      vault: apps/myapp/production/DB_USER@kv
      token: $VAULT_ID_TOKEN
      file: false
    DB_PASSWORD:
      vault: apps/myapp/production/DB_PASSWORD@kv
      token: $VAULT_ID_TOKEN
      file: false
  script:
    - kubectl create secret generic db-credentials \
        --from-literal=host=$DB_HOST \
        --from-literal=username=$DB_USER \
        --from-literal=password=$DB_PASSWORD \
        --dry-run=client -o yaml | kubectl apply -f -
    - kubectl rollout restart deployment/myapp
  rules:
    - if: $CI_COMMIT_BRANCH == "main"
```

Using API keys for external services

Example of integration with external APIs (Slack, Telegram, S3):

```
stages:
  - notify
  - backup

notify-slack:
  stage: notify
  image: curlimages/curl:latest
  id_tokens:
    VAULT_ID_TOKEN:
      aud: vault
  secrets:
    SLACK_WEBHOOK_URL:
      vault: integrations/slack/WEBHOOK_URL@kv
      token: $VAULT_ID_TOKEN
      file: false
  script:
    - |
      curl -X POST "$SLACK_WEBHOOK_URL" \
        -H "Content-Type: application/json" \
        -d "{\"text\": \"Deployment completed for $CI_PROJECT_NAME\"}"

backup-to-s3:
  stage: backup
  image: amazon/aws-cli:latest
  id_tokens:
    VAULT_ID_TOKEN:
      aud: vault
  secrets:
    AWS_ACCESS_KEY_ID:
      vault: cloud/aws/backup-user/ACCESS_KEY_ID@kv
      token: $VAULT_ID_TOKEN
      file: false
    AWS_SECRET_ACCESS_KEY:
      vault: cloud/aws/backup-user/SECRET_ACCESS_KEY@kv
      token: $VAULT_ID_TOKEN
      file: false
  script:
    - aws s3 sync ./artifacts s3://my-backup-bucket/$CI_PROJECT_NAME/$CI_COMMIT_SHA/
```

Signing Docker images with Cosign

Example of using a private key from Vault to sign images:

```
stages:
  - build
  - sign

build-image:
  stage: build
  image: docker:24.0.5
  services:
    - docker:24.0.5-dind
  script:
    - docker build -t $CI_REGISTRY_IMAGE:$CI_COMMIT_SHA .
    - docker push $CI_REGISTRY_IMAGE:$CI_COMMIT_SHA

sign-image:
  stage: sign
  image: bitnami/cosign:latest
  id_tokens:
    VAULT_ID_TOKEN:
      aud: vault
  secrets:
    COSIGN_PRIVATE_KEY:
      vault: signing/cosign/PRIVATE_KEY@kv
      token: $VAULT_ID_TOKEN
      file: true
  script:
    - cosign sign --key $COSIGN_PRIVATE_KEY $CI_REGISTRY_IMAGE:$CI_COMMIT_SHA
```

Branch-based access control

Example of configuring different secrets for `develop` and `main` branches using bound claims on branch (`ref`):

```
stages:
  - deploy

.deploy-template:
  image: alpine/k8s:1.28.0
  id_tokens:
    VAULT_ID_TOKEN:
      aud: vault
  script:
    - echo "Deploying from branch $CI_COMMIT_BRANCH with database $DB_HOST"
    - kubectl set env deployment/myapp DB_HOST=$DB_HOST DB_PASSWORD=$DB_PASSWORD

deploy-develop:
  extends: .deploy-template
  stage: deploy
  variables:
    VAULT_AUTH_ROLE: myapp-develop
  secrets:
    DB_HOST:
      vault: apps/myapp/develop/DB_HOST@kv
      token: $VAULT_ID_TOKEN
      file: false
    DB_PASSWORD:
      vault: apps/myapp/develop/DB_PASSWORD@kv
      token: $VAULT_ID_TOKEN
      file: false
  rules:
    - if: $CI_COMMIT_BRANCH == "develop"

deploy-main:
  extends: .deploy-template
  stage: deploy
  variables:
    VAULT_AUTH_ROLE: myapp-main
  secrets:
    DB_HOST:
      vault: apps/myapp/main/DB_HOST@kv
      token: $VAULT_ID_TOKEN
      file: false
    DB_PASSWORD:
      vault: apps/myapp/main/DB_PASSWORD@kv
      token: $VAULT_ID_TOKEN
      file: false
  rules:
    - if: $CI_COMMIT_BRANCH == "main"
```

Example of configuring a Vault role with bound claims based on branch for access control:

```
# Role for `develop` — access only from `develop` branch.
vault write auth/jwt/role/myapp-develop - <<EOF
{
  "role_type": "jwt",
  "user_claim": "sub",
  "bound_audiences": ["vault"],
  "bound_claims": {
    "project_path": "myteam/myapp",
    "ref": "develop",
    "ref_type": "branch"
  },
  "policies": ["myapp-develop-policy"],
  "ttl": "1h"
}
EOF

# Role for `main` — access only from `main` branch (protected).
vault write auth/jwt/role/myapp-main - <<EOF
{
  "role_type": "jwt",
  "user_claim": "sub",
  "bound_audiences": ["vault"],
  "bound_claims": {
    "project_path": "myteam/myapp",
    "ref": "main",
    "ref_type": "branch",
    "ref_protected": "true"
  },
  "policies": ["myapp-main-policy"],
  "ttl": "1h"
}
EOF
```

CI/CD analytics

The “CI/CD analytics” report shows how many of the pipelines a group or a project created over the selected period have completed, how long they took, and how their statuses are distributed.

Report availability

The report of a project is available to members with the “Reporter” role and above, including those who hold that role in a parent group.

The “Project-based pipeline visibility” setting of the project opens the report to everyone who can see the project itself: to users of the “Auditor” type, to members with the “Guest” role, in a public project to any visitor, including one who has not signed in, and in an internal project to any signed-in user. With the setting off, the report stays with the members holding the “Reporter” role and above.

Reading the report of a project requires access to the project's pipelines as well. Narrowing the visibility of the "CI/CD" feature to project members closes the report to everyone outside the project, including a visitor to a public project. Users of the "Auditor" type retain access regardless of that setting.

Narrowing the visibility of the "Analytics" feature to project members closes the report the same way, and users of the "Auditor" type keep it the same way. Turning the "Analytics" feature off closes the report for everyone, the members of the project included.

Turning the "CI/CD" feature off does not close the report: the project keeps its accumulated pipeline history, and the report continues to display it. The "CI/CD analytics" entry, though, disappears from the "Analyze" menu — the same happens when the project's repository is empty. In either case the report stays reachable at its direct address.

The report of a group is available to members with the "Reporter" role and above, including those who hold that role in a parent group, and to users of the "Auditor" type.

Viewing the report

To view the report of a group:

1. In the top bar, select "Search or go to…" and pick the group.
2. In the group menu, go to "Analyze" → "CI/CD analytics".

To view the report of a project:

1. In the top bar, select "Search or go to…" and pick the project.
2. In the project menu, go to "Analyze" → "CI/CD analytics".

The report is also available at a direct address:

Report	Address
Group	`https://<HOST>/groups/<GROUP>/-/analytics/ci_cd_analytics`
Project	`https://<HOST>/<GROUP>/<PROJECT>/-/analytics/ci_cd_analytics`

The report of a group covers the pipelines of its projects and the projects of its subgroups, archived projects included.

Metrics

The line above the charts sums up the period:

Metric	Content
“Total pipeline runs”	Number of completed pipelines created within the period
“Median duration”	Median duration (p50) of the runs that took time
“Failure rate”	Share of failed runs among the runs that succeeded or failed
“Success rate”	Share of successful runs among the runs that succeeded or failed
“Other rate”	Share of canceled and skipped runs among all runs of the period

“Failure rate” and “Success rate” divide by the runs that succeeded or failed. Each of them is rounded to a whole number on its own, so together they come to about a hundred: a period with 5 successful and 3 failed runs shows a failure rate of 38 % and a success rate of 63 %. “Other rate” divides by every run of the period and adds on top of those. The “How this is calculated?” icon next to “Failure rate” and “Other rate” states the formula.

A rate whose denominator holds no run of the period is shown as a dash. A period of canceled and skipped runs alone therefore shows a dash in “Failure rate” and “Success rate”, and a hundred per cent in “Other rate”.

When the project has failed runs, “View all” under “Failure rate” opens the pipeline list of the project filtered to the failed runs. A branch selected in the report is carried over to the list as a filter; with “All branches” the list opens without one.

Charts

The “Duration” chart holds a line per percentile of the pipeline duration:

- “Median (50th percentile)”;
- “95th percentile”.

The vertical axis of the chart is in minutes.

The “Status” chart holds a column per day, split into series:

- “Successful”;
- “Failed”;
- “Other (Cancelled, Skipped)”.

The vertical axis of the chart counts pipelines.

Filters

The filters above the charts narrow the metrics and both charts:

Filter	Values
“Source”	Event that created the pipeline: “Push”, “Schedule”, “Merge Request Event” and the rest. “Any source” by default
“Branch”	Branch of the pipeline. The default branch of the project by default; “All branches” drops the filter. The report of a group has no such filter
“Date range”	“Last week”, “Last 30 days”, “Last 90 days”, “Last 180 days”. “Last week” by default

The “Branch” filter also takes the pipelines of the merge requests this branch is the source of. Those pipelines are counted while “Source” stays at “Any source” or is set to “Merge Request Event”; with any other source the report counts the runs of the branch itself.

The selected filters are kept in the address of the page as the `source` and `time` parameters, so a configured report can be shared as a link. The report of a project keeps the branch there as well, as the `branch` parameter.

How pipelines are counted

The report counts the pipelines created within the period that have reached a completed status: successful, failed, canceled and skipped. A pipeline belongs to the day it was created on, counted in UTC, so a pipeline created on one day and completed on the next is counted on the day it was created.

The period ends at the start of the current day, so the pipelines created today stay out of the report and the charts carry no point for today.

The percentiles of the “Duration” chart and the “Median duration” metric skip the runs without a duration and the runs with a zero duration. A day with no such run carries no point on the chart, and a period with no such run shows a dash in place of the median.

The report of a group with the same filters and period is refreshed at most once a minute, so opening it again can show a state up to a minute old. The report of a project is built on every request.

When the report takes too long to build, the page shows the failure in place of the charts and asks for a narrower reporting window. Select a shorter “Date range” and open the report again.

Security

Group and project security settings

Group owners and project maintainers apply these settings to protect projects on a daily basis. Open the group's or the project's "Settings" page in the left sidebar, then the tab named in each section below.

Group-level settings

Settings a group owner applies to every project in the group.

General

Set the group visibility level to "Private" wherever possible, in "Settings" → "General": the group and its projects are visible only to members, which lowers the risk of unintended access to source code and issues.

Group access rights and capabilities

Configure access rights and capabilities for the group in "Settings" → "General":

- Whether members can invite groups outside this group and its subgroups.
- Whether projects in this group can be transferred to other groups.
- The wiki access level for the group.
- The Git access protocols (SSH, HTTP(S)) enabled for the group.
- The minimum role required to create projects in the group.
- The minimum role required to create subgroups.
- Whether members can create group and project access tokens in the group. A group access token can only be created by a member with the `Owner` role; a project access token created by a group member is limited by that member's role in the project.
- Whether two-factor authentication is required for every member of the group, and the grace period before it is enforced. Subgroups cannot set their own two-factor authentication rules once this is enabled.
- Whether Pages content is restricted to project members for every project in the group.

Repository

Settings that apply to the Git repository of every project in the group.

Deploy tokens

Deploy tokens grant access to the group's repositories and packages without a personal account. Review deploy tokens regularly, remove the ones no longer in use, and give the remaining tokens a clear description, a limited scope, and an expiration date.

Default branch

Configure default branch protection for every project in the group in "Settings" → "Repository". A project inherits this setting from the group unless the group allows a project to override it.

Push rules

Configure push rules for every project in the group in “Settings” → “Repository”. If overriding the instance-level rules is allowed, override them here for a policy specific to the group. See Push rules for the available rules.

Prohibit overriding these rules at the project level unless a project needs its own policy.

CI/CD

Settings that apply to CI/CD pipelines and variables in every project in the group.

Pipelines

Enable “Enable JWT format for CI/CD job tokens” in “Settings” → “CI/CD”.

Variables

Mask CI/CD variables that hold sensitive values, in “Settings” → “CI/CD”. A masked variable’s value does not appear in job logs or audit events.

Project-level settings

Settings a project maintainer or owner applies to a single project.

General

General project settings control who can view the project and what they can do without an assigned role.

Project visibility, features, and permissions

Set the project visibility to “Private” wherever possible, in “Settings” → “General”. Enable “Require authentication to view media files” to prevent unauthenticated access to media embedded in issues and merge requests, and restrict access to LFS objects, merge requests, branches, package and container registries, issues, environments, and releases according to your organization’s requirements. Downloading from the package registry is disabled for every user by default.

Repository

Settings that apply to the project’s Git repository.

Push rules

Configure push rules for the project in “Settings” → “Repository”. If overriding the group-level rules is allowed, override them here for a policy specific to the project. See Push rules for the available rules.

Protected branches

Use protected branches, in “Settings” → “Repository”, to control which roles can merge into the main and stable branches or push to them directly. By default, only members with the `Maintainer` or `Owner` role can push directly to a protected branch. See Protected branches.

Protected tags

If tags mark releases or other significant points in the project's history, restrict which role can create tags matching a pattern, in "Settings" → "Repository". For example, restrict version tags matching `*_v*` to the `Maintainer` role.

Deploy tokens

Deploy tokens grant access to the project's repository and packages without a personal account. Review deploy tokens regularly, remove the ones no longer in use, and give the remaining tokens a clear description, a limited scope, and an expiration date.

Deploy keys

Deploy keys are an alternative to deploy tokens for granting access to the repository without a personal account, in "Settings" → "Repository". The same review applies to them: scope, description, and expiration date.

Merge requests

Settings that control how a merge request is reviewed and merged into the project's main or stable branches, in "Settings" → "Merge requests".

Merge checks

- Enable "Pipelines must succeed" and add the security checks the project needs to its pipeline: secret scanning, static and dynamic analysis, and vulnerability scanning for dependencies and container images.
- Enable "All discussions must be resolved" so that code still under review cannot reach the main or stable branches.

Assign code paths to reviewers with a `CODEOWNERS` file, and configure merge request approval rules to set the minimum number of approvers, to prohibit an author from approving their own merge request, and to restrict who can change the approval rules themselves, for example by including the review policy file from another project through the `include` mechanism. See CODEOWNERS and Approval rules for merge requests.

CI/CD

Settings that apply to CI/CD pipelines, variables, and tokens in the project, in "Settings" → "CI/CD".

Pipelines

Enable "Use separate caches for protected branches". Assign a `CODEOWNERS` entry to `.gitlab-ci.yml` to control who can change the pipeline configuration, and require manual approval before a critical pipeline or job runs.

Variables

Mark CI/CD variables that hold sensitive values as protected and masked, and store secrets in CI/CD variables or in an external Vault or Deckhouse Stronghold instead of the repository. See External Vault

integration. Enable “Show pipeline variables” to display pipeline variable values in job logs, which helps diagnose pipeline failures and detect unexpected values.

Pipeline trigger tokens

A pipeline trigger token runs a pipeline with the permissions of the member who issued it. Review trigger tokens with the same care as CI/CD job token permissions, below.

CI/CD job token permissions

Restrict which groups and projects can access this project’s data from their own pipelines, in “Settings” → “CI/CD”, using an allowlist.

Packages and registries

Protect packages published to the project’s package registry with the protected packages mechanism, in “Settings” → “Packages and registries”, restricting access to members above a given role. The same applies to the container registry.

SAST scanning with Semgrep

Static Application Security Testing (SAST) in Deckhouse Code looks for vulnerabilities in the source code of a project. The scanner behind it is Semgrep, and the scan is called `sast`.

The scan is **mandated**: it is added to a project’s pipeline by a scan execution policy that lives in a separate security policy project, not by the project’s own `.gitlab-ci.yml`. The policy — that is, the security team — decides what is scanned and what fails the pipeline, and the scanned project cannot override it from its own CI/CD settings.

> ⚠ Security scanning is gated behind the instance-level feature flag `fe_security_scan_policies`, which is **disabled by default**. While it is off, policy pages and scanner integration pages are hidden and no scan jobs are added to pipelines. Ask an instance administrator to enable it.

What the scan does

The `sast` scan adds two jobs to the pipeline, both in the `fe-security-scanner` stage:

Job	What it does
`semgrep_sast_scan`	Runs the scanner over the repository and publishes its raw JSON output as an artifact.
`semgrep_sast_junit`	Reads that JSON, builds a JUnit test report and a security report from it, and decides the outcome: the job succeeds when no finding reaches the blocking threshold, and fails otherwise. It runs on a Ruby image.

Both jobs upload their artifacts with `when: always`, so a blocked pipeline still carries the findings that blocked it.

Findings reach the same places as those of every other policy scanner:

Report	Where to view
Tests (JUnit)	“Tests” tab of the pipeline and the test summary in the merge request
Security report (SAST)	Vulnerability report page
Findings forwarded to DefectDojo	DefectDojo, when that integration is configured

Prerequisites

Before the scan can run, make sure that:

- an instance administrator has enabled the `fe_security_scan_policies` feature flag;
- a security policy project is linked to the project or to a group above it;
- a GitLab Runner with a `docker` executor is available, and it can pull both images the scan uses: the scanner image (see “Where the scanner image comes from” below) and the Ruby image the second job runs on, `ruby:3.3.10-slim` by default, which an installation can redirect with the instance variable `FE_SCANS_REPORT_CONVERTER_IMAGE`.

Turning the scan on

Add a `sast` action to `.gitlab/security-policies/policy.yml` in the security policy project:

```
scan_execution_policy:
  - name: SAST everywhere
    enabled: true
    rules:
      - type: pipeline
        branches: ["*"]
    actions:
      - scan: sast
```

Then link the policy project to the target project or group in “Settings” → “Security policy”. Every pipeline the rules match then gains the two jobs above.

The scan options can also be edited in the policy editor, which renders the form described below.

What the policy sets

The form for the `sast` action is built from the groups “Scanner image”, “Rules”, “Subjects to leave out”, “Blocking”, “Paths”, “Rule levels” and “Additional arguments”. Every option belongs to the policy: the server writes the values into the job while the pipeline is rendered, so the scanned project’s own CI/CD variables cannot change them.

Each group opens and closes on its own title, and all of them start closed. A closed group carries a statement of its own: the policy writes nothing into it, so that part of the scan keeps the answer the server gives without it — for the scanner image, the one the Semgrep integration names where it names one, and the value shipped with this product everywhere else. Open a group and fill in a field, and the policy takes that decision over for every project it covers. Save the policy and open it again, and the groups it holds something in come back open, so a saved policy always shows everything it sets.

A group is named after the choice it is built around, and that choice is drawn without its own name repeated inside the group: opening “Scanner image” goes straight to the image sources, “Rules” to the rule sources, “Paths” to the filter sources, “Blocking” to the threshold, “Rule levels” to the three levels. The fields a choice reveals keep their own names underneath it, and the hint ("?") of the leading choice sits beside the group’s title.

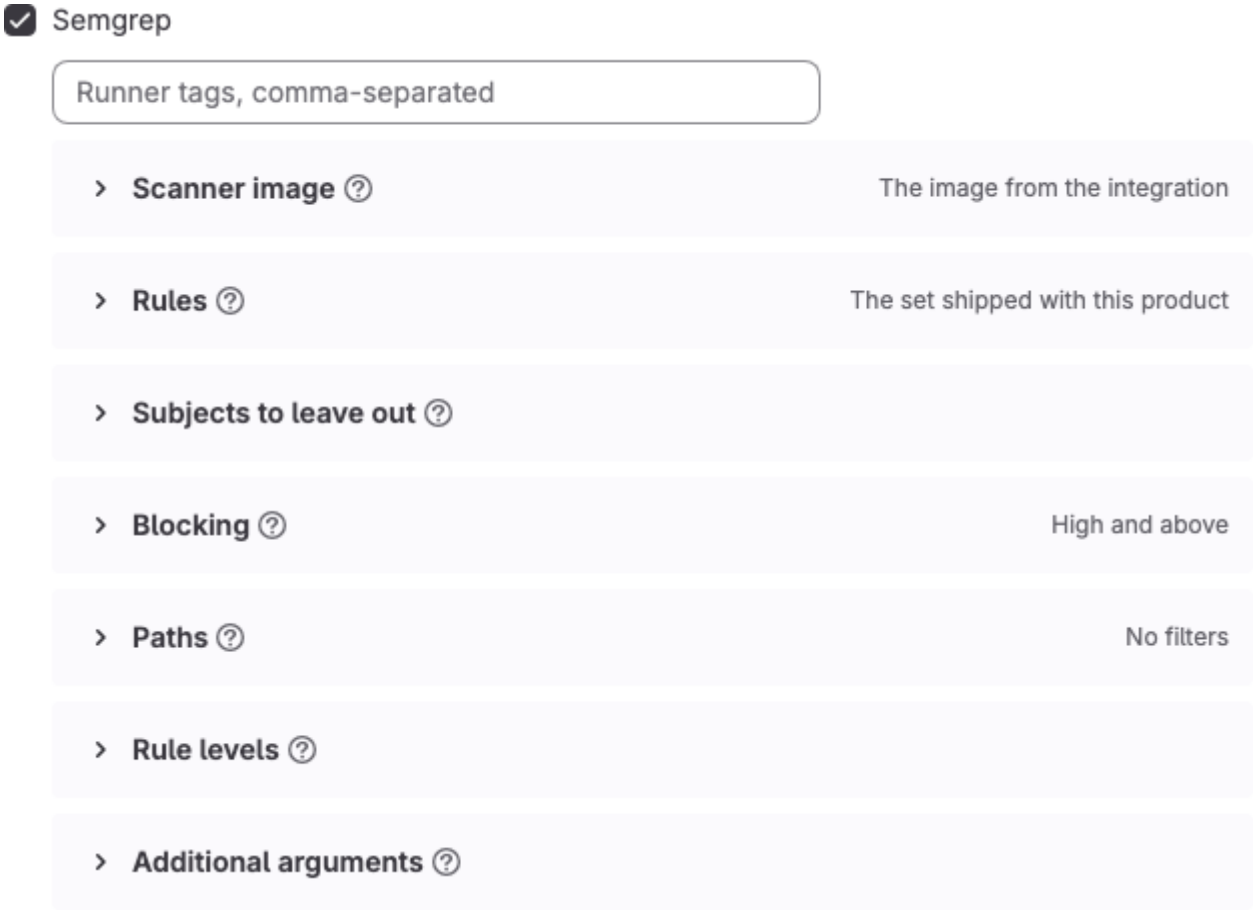
What a closed group has settled is stated at the end of its title line. That is what makes the closed form readable: the threshold and the rule source can be read without opening anything.

Group	What its closed line states
“Scanner image”	Which of the three sources the image comes from — “The image from the integration” where the integration names an image, and “The image shipped with this product” everywhere else
“Rules”	Where the rules come from — “The set shipped with this product” until the policy picks another source
“Subjects to leave out”	Nothing. The group holds checkboxes, and the line states a choice out of a list
“Blocking”	The threshold — “High and above” until the policy picks another
“Paths”	Where the path filters come from — “No filters” until the policy picks another
“Rule levels”	The levels the policy switched on, such as “High, Medium”. Blank while the policy has set none
“Additional arguments”	Nothing. Free text carries no choice out of a list, so there is nothing to state

The lines of “Scanner image”, “Rules”, “Blocking” and “Paths” always carry a value, because their control is a set of radio buttons, which always shows an answer: the policy’s, or the one the server would give while the policy has said nothing. The line and the control therefore agree. “Rule levels” holds a set of levels, and while the policy has set none there is nothing to name — the three boxes come up ticked instead, which is the same state.

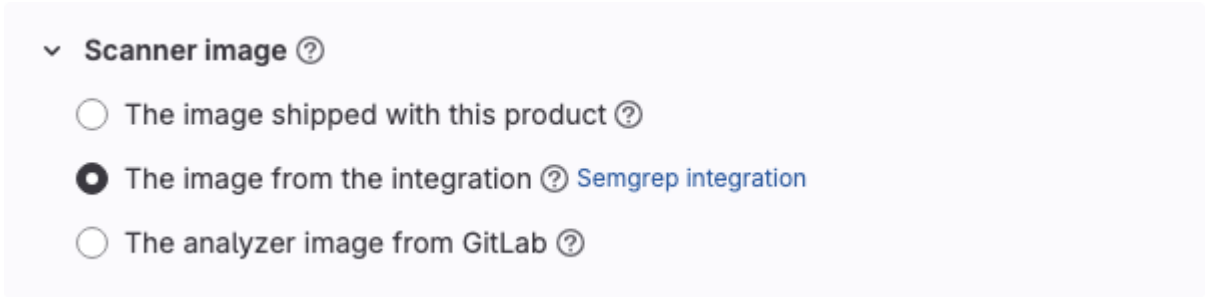
No group here carries a checkbox beside its title. Where another scanner’s group carries one, that group is a module that can be switched off; these groups are the scan itself. Removing the `sast` action from the policy is what turns the scan off. The checkboxes inside “Subjects to leave out” and “Paths” each carry one setting of their own.

A field that answers to one choice is drawn under that choice and nowhere else. Picking “A file in the security policy project” as the rule source puts the rule file path beneath that option; picking another source takes the field away.



Scanner image

This group sets which image the scan runs in. It opens on “Image source”, a choice of three:



Source	Which image the scan runs
“The image shipped with this product”	The image this release pins, or the one an administrator set for the whole installation with the instance variable `FE_SCANS_SEMGREP_IMAGE`. The image carries its own version, so this source has no version field.
“The image from the integration”	The registry the Semgrep integration mirrors the scanner in, or the one prepared image that integration names — see “The Semgrep integration”.
“The analyzer image from GitLab”	`registry.gitlab.com/security-products/semgrep`, at the tag the policy names. Needs an installation that can reach that registry.

Field	What it sets
“Image version”	The tag the analyzer image is pulled at, as the registry lists it: three numbers, such as `6.25.0`, or a digest. Drawn under “The analyzer image from GitLab” and nowhere else, because the other two sources carry their own version. Left empty, the tag this release ships is used.

The image tag and the semgrep version are two different numberings: a larger image tag has meant an older semgrep. The hint on “Image version” names the semgrep version that ships with this product and the image tag carrying it, and the field’s list of suggestions names the same for every tag this release knows of. The hint beside the group’s title states which image this installation will run: the prepared image the integration names, the registry it mirrors the scanner in, or the image the installation itself is set to use.

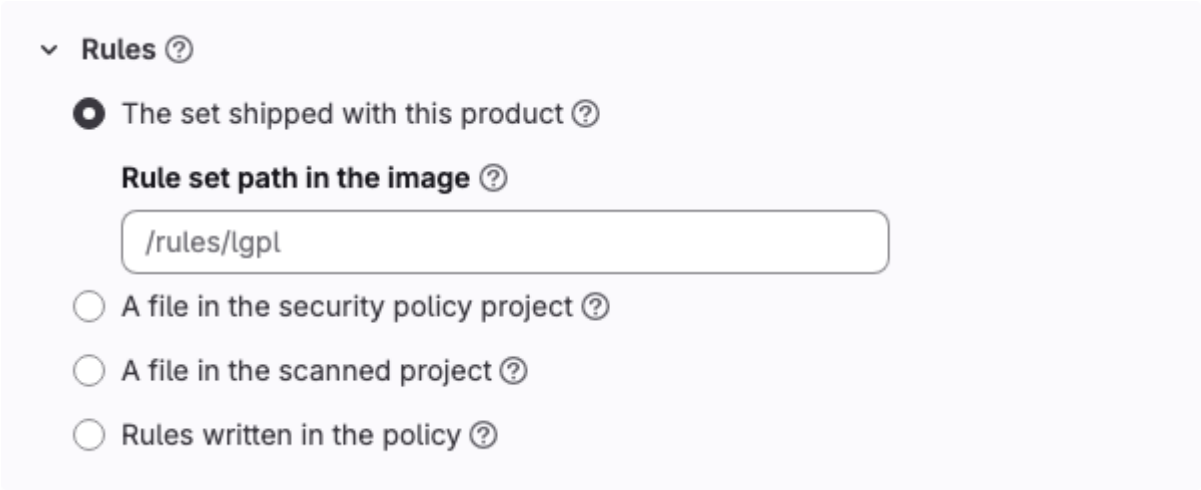
Beside “The image from the integration” stands a link to the Semgrep integration page, in every state of the choice, for a policy owned by a group. A policy owned by a project is settled by the group above it, whose settings the project’s maintainer may not be able to open, so that choice carries no link there.

“The image from the integration” is the one source that can be unavailable, and there is a single reason for it: the integration names no image for the scan to pull, either because no Semgrep integration is set up for the group, or because one is set up and names neither a registry nor a prepared image. The choice then stays in the list, greyed out, with the reason written on the choice itself. The other two sources are always available.

A policy that names no source shows the answer the server would give without it: “The image from the integration” where the integration names an image, and “The image shipped with this product” everywhere else — see “Where the scanner image comes from”.

Rules

This group sets which rules the scan runs.



Field	What it sets
“Rule set”	Where the rules come from. The sources are described below. Default: “The set shipped with this product”.
“Rule set path in the image”	Which of the sets carried inside the scanner image to run (see “Rule sets in the image”). Default: `/rules/lgpl`. The scan stops if the image has no set at this path.
“Rules”	The rules themselves, in the scanner’s own YAML. Shown when the source is “Rules written in the policy”.
“Rule file path”	Path to the rule file inside the project the rules are read from. Default: `.semgrep.yml`.
“Replace the shipped set”	Off by default: your own rules run **alongside** the shipped set. Ticked, they run **instead** of it, and everything the shipped set covered stops being checked.

Every field carries a hint behind the “?” beside its name; the hint of “Rule set”, which the group is named after, sits beside the group’s title.

The rule sources differ in who owns the file:

Source	Where the rules live	Who controls them
“The set shipped with this product”	Inside the scanner image, at the path named above	The product release
“A file in the security policy project”	The project this policy lives in, on its default branch	The security team
“A file in the scanned project”	The repository being scanned, at the commit being built	**The scanned project**
“Rules written in the policy”	The policy itself	The policy author

The server reads “The set shipped with this product”, “A file in the security policy project” and “Rules written in the policy”, and writes them into the job. “A file in the scanned project” is read by the job itself, from its own checkout, at the commit being built. That is also the one source the scanned side writes: a commit that rewrites that file rewrites the rules of the scan that checks it. Leaving “Replace the shipped set” unticked keeps a baseline in place even when the project’s file is emptied.

A rule file the policy named but the scan could not be given stops the job: it prints why and fails, and the shipped set does not step in for the missing file.

Subjects to leave out

This group sets which subjects of the shipped set the scan leaves out.

Field	What it sets
“Leave out the secret rules”	Off by default. Ticked, the rules that look for credentials committed to the repository stay out of the run.
“Leave out the configuration rules”	Off by default. Ticked, the rules that read configuration and infrastructure code — Terraform, Dockerfiles, Kubernetes manifests, CI configuration — stay out of the run.
“Leave out the malicious code rules”	Off by default. Ticked, the rules that look for the marks of deliberately hidden code stay out of the run.

The boxes appear while the rules come from the shipped set, because rules of your own carry no subject to leave out. A ticked box leaves the subject out; an empty one runs it, which is what a policy written before these fields existed goes on doing. The group’s hint states what the choice is for: where another mandated scan already covers one of these subjects, both report it, and the same line is a finding twice.

What each subject holds, and the rule set an exclusion needs, are in “Subjects inside the shipped set”.

Blocking

This group sets what a finding does to the pipeline.

Field	What it sets
“Blocking threshold”	What fails the pipeline. Findings below the threshold are still reported and still reach the vulnerability report; the threshold decides the pipeline’s outcome and nothing else. Default: “High and above”.

Threshold values:

Value	What fails the pipeline
“High and above” (default)	A high finding. Medium and info are reported.
“Medium and above”	Medium and high. Info is reported.
“Any finding”	Any finding, whatever its level.
“Report only, block nothing”	Nothing. Findings are still reported and still reach the vulnerability report and DefectDojo.

Semgrep reports at ERROR, WARNING and INFO, which the scan shows as high, medium and info. Semgrep has no critical and no low, so a threshold named after either would be one no finding could reach: “critical” would block nothing while looking strict, and “low” would repeat “medium”. “Report only, block nothing” puts the decision to block nothing into the policy as a value of its own.

Paths

This group sets what the scan looks at, and what it leaves out.

Field	What it sets
“Path filters”	Where the lists of paths to leave out come from. Default: “No filters”.
“Filter file path”	Path to the filter file inside the project the filters are read from. Default: `.semgrepignore`.
“Skip these paths”	One path or pattern per line: third-party code, fixtures, generated files.
“Look only at these paths”	One path or pattern per line. Everything else is left out of the scan entirely.
“Honor the repository’s .gitignore”	Off by default.

Whichever source the filters come from, the job applies its own built-in exclusions — `node_modules/`, `vendor/`, `dist/`, `build/`, `.venv/` and `.git/` — on top of whatever the policy set, so dependency trees stay out of the scan even where the policy’s own filters say nothing about them.

Filters take their source the same way rules do:

Source	Where the list lives	Who controls it
“No filters”	—	—
“A file in the security policy project”	Beside the policy, applied to every project it covers	The security team
“A file in the scanned project”	The repository being scanned	**The scanned project**
“Lists written in the policy”	The two lists in the form	The policy author

By default the job sets aside any `.semgrepignore` it finds in the scanned repository — including nested ones in subdirectories — before the scan starts. Picking “A file in the scanned project” is what turns that off:

> ⚠ “A file in the scanned project” and “Honor the repository’s .gitignore” both hand the scanned project the say over what the mandated scan examines. A commit in that project can then narrow the scan that checks it, and the narrowing stays invisible in the policy. A security team may well decide that each project knows its own third-party directories best; the delegation is then recorded in the policy, since both settings stay off until this group is opened and one of them is set.

About the two lists:

- “Look only at these paths” is stronger than “Skip these paths”. An exclusion removes what it names; an inclusion removes everything else. A single `src/**` quietly takes `lib`, `config` and your migrations out of the scan.
- They are applied in that order — inclusions first, then exclusions — so an exclusion still applies inside what an inclusion let through.

A pattern that matches nothing passes the form’s validation. The job catches it: it prints how many files the scanner read, and fails when that number is zero.

Rule levels

This group sets which levels of rule run at all.

Field	What it sets
“Rule levels”	“High” (ERROR), “Medium” (WARNING), “Info” (INFO). All three boxes come up ticked, which is what a scan runs while the policy has set no levels of its own.

A level is exactly the rules the scanner marks with it: “High” the ERROR rules, “Medium” the WARNING rules, “Info” the INFO rules — and a finding is reported at the level of the rule that found it. Unticking a level takes exactly that level’s rules out of the run.

The last level left ticked cannot be unticked. An emptied list is removed from the policy, and a policy that names no levels is the state that runs every level.

“Rule levels” and “Blocking threshold” look alike and do different things:

Field	Effect	A finding below the level
“Rule levels”	Which rules are **evaluated**	Does not exist: the rule never ran
“Blocking threshold”	Which findings **fail the pipeline**	Is in the report, but does not block

A threshold covers its own level and every level above it, which is what “and above” says; a rule level covers the rules the scanner marks with that level and no others.

Leaving a level out therefore removes its findings from the report as well as from blocking: the vulnerability report and DefectDojo both get smaller, and the job log says nothing about it. A threshold that no ticked level can reach never fires; the form says so when that happens.

Additional arguments

This group sets the flags the rest of the form does not name.

Field	What it sets
“Additional arguments”	Flags passed to the scan as written.

“Additional arguments” tunes the resources of the run: timeouts, memory limits, maximum file size. Flags that redirect or silence the report are refused, as are the flags this form already owns: output and report formats (`--json` , `--sarif` , `--junit-xml` , `--gitlab-sast` , `--gitlab-secrets` , `--output` , `--text` and their paired `*-output` forms), `--config` , `--metrics` and `--baseline-commit` .

The hint on this field also states where the scan’s findings end up: they are published as a security report, so they reach the vulnerability report and whatever the administrator has connected to it.

Rule sets in the image

By default the scan runs the rule set carried inside the scanner image, at `/rules/lgpl` . The image carries more than one set, and “Rule set path in the image” chooses between them:

Path in the image	Rules	Rules per language
`/rules/lgpl` (default)	152	`javascript` 83, `kotlin` 58, `swift` 5, `java` 4, `typescript` 4, `generic` 2
`/rules/lgpl-cc`	97	`ruby` 40, `java` 39, `php` 9, `python` 6, `javascript` 1, `typescript` 1, `generic` 1, `yaml` 1
`/rules/gitlab`	5	`java` 2, `generic` 1, `javascript` 1, `kotlin` 1
`/rules`	592	the sets above, plus the rule files stored beside them: `scala` 86, `python` 79, `c` 62, `cpp` 62, `go` 27, `csharp` 22 and more

A rule may name several languages, so the per-language figures add up to more than the rule count. The `/rules` row is the whole directory: pointing the scan at it loads every set below it and the files beside them.

These figures were measured in `registry.gitlab.com/security-products/semgrep:6.25.0` and describe that image; a newer image may carry a different set. Each directory in the image carries its own license file, and “Rule set path in the image” is where you name the one to run.

> ⚠ **The default set covers `javascript`, `kotlin`, `swift`, `java`, `typescript` and `generic`. Python, Go, Ruby, PHP, C, C++, C# and Scala are outside it.** A repository written in one of those, scanned at the default path, finishes successfully with no findings: no rule applied to it. Such a repository has no SAST coverage at all, and its green pipeline says nothing about the code.
> Point “Rule set path in the image” at a set that covers its language, add rules of your own through the “Rule set” field, or have an administrator name an image carrying a suitable set on the Semgrep integration and pick “The image from the integration” in the policy. Whoever brings a set of their own owns what is in it and the terms it comes under.

Rules of your own are added through the “Rule set” field and, with “Replace the shipped set” left unticked, run alongside whichever set the image provides.

Subjects inside the shipped set

The rules in the set are marked by subject: what a rule judges. Measured on a rule set of 2826 rules:

Subject	Rules	What its rules judge
Code	1988	The program’s own code
Configuration	522	Configuration and infrastructure code: Terraform, Dockerfiles, Kubernetes manifests, CI configuration
Secrets	225	Credentials committed to the repository
Malicious code	91	The marks of deliberately hidden code: obfuscation, code assembled at run time, execution routed through indirection

A newer image may carry a different set. Configuration, secrets and malicious code are the subjects the policy can leave out, in the “Subjects to leave out” group; the code rules run in every scan.

An exclusion needs a rule set that carries an index of its subjects. The set published in GitLab’s own image carries none, so a ticked box there stops the job, naming the subjects it was asked to leave out and the rule set that has no index to leave them out by — see “Troubleshooting”. Rules written in the policy carry no subjects either, which is why the boxes are hidden while they are the source.

⚠ The secret rules and the `secret_detection` scan cover the same ground, and the same line then arrives as two findings at two severities. Measured on one run: semgrep filed a token committed to the repository into the SAST report at Info, while `secret_detection` filed the same token into its own report at Critical. Neither set of findings sits inside the other — an AWS key was found by semgrep alone, a Slack webhook URL by `secret_detection` alone — so leaving the secret rules out moves the subject to `secret_detection` and takes those findings of its own off the SAST report with it.
The same reading applies to the configuration rules where a dedicated infrastructure scanner already covers that ground, and to the malicious code rules, which read intent: a repository that obfuscates or assembles code at run time for reasons of its own sees them fire.

Where the scanner image comes from

The image the scan runs in is resolved by the server once, before the job starts, and written into the job as a literal value. It never becomes a CI/CD variable, so the scanned project's own settings cannot point the scan at a different scanner.

Which image that is follows from the source the policy names in its "Scanner image" group:

Source in the policy	What the scan pulls
"The image shipped with this product"	The image an administrator set for the whole installation with the instance environment variable `FE_SCANS_SEMGREP_IMAGE`, or the image this release pins where that variable is unset.
"The image from the integration"	The prepared image the Semgrep integration names, exactly as written; where the integration names a registry instead, the scanner from that registry, at the version the policy carries or the one this installation runs.
"The analyzer image from GitLab"	`registry.gitlab.com/security-products/semgrep`, at the version named in "Image version", or at the version this release ships while that field is empty.

A prepared image on the integration answers for one source and no other: the administrator who named it named what "The image from the integration" pulls, including its version, so a version set in the policy does not apply to it. A policy asking for the shipped image, or for GitLab's analyzer image, gets the image it asked for.

A policy that names no source at all was written before this field existed, and it keeps the answer it has always had: the image the integration names, then the image the instance was set to, then the one the release ships. A version such a policy carries still applies, unless the integration names a prepared image.

The scanned project's own integration record is ignored: the image comes from the side that mandates the scan.

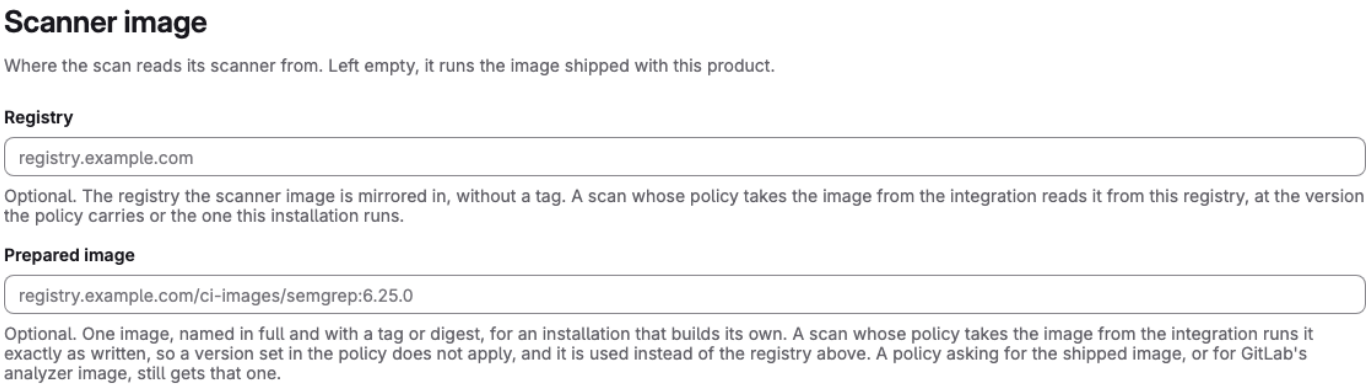
The Semgrep integration

The integration holds where the scanner image is read from. It is set on a **group**, so an installation names the registry once for every policy below it; on a project the integration shows no fields, which is how a reader learns the scan is configured above it.

Open "Settings" → "Integrations" → "Semgrep" on the group and fill in either field — both are optional:

Field	What it sets
"Registry"	The registry the scanner image is mirrored in, without a tag. For example, `registry.example.com` or `registry.example.com/mirror`. A scan whose policy takes the image from the integration reads it from here, at the version the policy carries or the one this installation runs.
"Prepared image"	One image, named in full and with a tag or a digest, for an installation that builds its own. For example, `registry.example.com/ci-images/semgrep:6.25.0`. A scan whose policy takes the image from the integration runs it exactly as written, so a version set in the policy does not apply. Takes precedence over "Registry".

With both fields empty the integration names no image, so "The image from the integration" cannot be picked in a policy and the scan runs the image shipped with this product. What the scan looks for is set in the policy.



The version shipped with this product

This release ships `registry.gitlab.com/security-products/semgrep:6.25.0`, pinned to that exact version. One tag pins both halves of the scanner at once — the engine and the rule sets carried inside the image — because both live in the same image.

The “Image version” field in the policy takes either form of pin:

- a version number, such as `6.25.0`, which a person can read and compare;
- a digest, such as `sha256:aafbcaff…`, which names one build and cannot be moved to another.

Floating tags (`latest`, `stable`, a tag that is a major number on its own, such as `6`) and references that carry a registry address are refused: the registry is the integration’s to name, and the policy names only the version.

The version shipped with a release is reviewed when Deckhouse Code is updated, so it moves forward with the product.

Installations with no route to the internet

An installation with no route to the internet provides both images the scan uses:

Image	Default	How to redirect it
Scanner	`registry.gitlab.com/security-products/semgrep:6.25.0`	The instance variable `FE_SCANS_SEMGREP_IMAGE`, which redirects “The image shipped with this product”; or the “Registry” or “Prepared image” field of the Semgrep integration, for a policy that takes the image from the integration
Report converter	`ruby:3.3.10-slim`, from Docker Hub	The instance variable `FE_SCANS_REPORT_CONVERTER_IMAGE`

Copy both images into your own registry, as GitLab’s own offline documentation prescribes for its analyzer images, then point the corresponding setting at that registry. Deckhouse Code names both images and pulls them from outside.

> The GitLab dependency proxy leaves the scanner image outside its reach. The proxy is a pull-through cache for images stored on Docker Hub, and the scanner image is published on `registry.gitlab.com`; of the two images, it can cache the Ruby converter. An installation that wants to stop pulling the scanner image from outside on every run — with or without a route to the internet — copies it into its own registry and names that copy in `FE_SCANS_SEMGREP_IMAGE` or in the “Registry” field.

Semgrep’s paid features

Everything described on this page runs on the open version of the scanner. Semgrep capabilities that lie outside it:

Capability	In the open version	What Deckhouse Code offers instead
Cross-file data-flow analysis	No, within a single file only	Nothing; the limitation stands
Pro rules	No	Rules of your own, through the “Rule set” field
Dependency analysis (Supply Chain)	No	The CodeScoring integration, or the `dependency_scanning` scan
Validity checking of discovered secrets	No	Nothing

> ⚠ The paid capabilities work through the vendor’s cloud. Turning them on means the scan uploads its findings — and the code context around them — to Semgrep’s servers, outside your installation. An installation with no route to the internet has nothing to reach that cloud through, so the capabilities are unavailable there.

The mandated scan keeps everything inside the job. It runs `semgrep scan` with `--metrics=off` and `--disable-version-check`, so findings and usage telemetry stay where the job put them. Before the scanner is even located, the job clears every `SEMGREP_*` and `PYTHON*` variable it inherited: `SEMGREP_RULES` and `SEMGREP_BASELINE_COMMIT` among them are documented equivalents of `--config` and `--baseline-commit`, and left in place they would let the scanned project’s CI/CD settings replace the rule set or the comparison base of the scan that checks it.

Troubleshooting

The scan does not appear in the pipeline

Check that:

- the `fe_security_scan_policies` feature flag is enabled on the instance;
- the security policy project is linked to the project, and `policy.yml` contains a `- scan: sast` action;
- the policy’s rules match the pipeline (branch, pipeline type);
- a GitLab Runner with a `docker` executor is available.

The job fails with “no rule set at … in this image”

The path in “Rule set path in the image” does not exist in the image the scan resolved. Check the path against the table above, and check which image is actually in use: with “The image from the integration” picked, a “Prepared image” set on the integration may carry a different layout.

The job fails with “carries no class index to switch them off by”

A box in “Subjects to leave out” is ticked, and the rule set the scan resolved carries no index of its subjects — the set published in GitLab’s own image carries none. Untick the boxes, or point the scan at a rule set that carries the index. The job prints the subjects it was asked to leave out and the rule set it read.

The neighbouring message, that the index holds no rule in any of the named subjects, is answered the same way.

The job fails with “semgrep read no files at all”

The scan examined nothing, so it refuses to report success. The usual causes are a “Look only at these paths” pattern that matches nothing, filters that removed the whole repository, or a checkout with nothing in it. The job’s log prints how many files it read and how many the filters removed.

The scan succeeds but finds nothing

Most often the rule set does not cover the language of the repository — see the warning in “Rule sets in the image”. Check the job log: it prints the rule set path in use and the number of rules loaded. Narrowing “Rule levels” has the same effect for a different reason: a level the policy unticked never runs, so its findings are absent from the report entirely.

Dependency list and license compliance

A security scan uploads a Software Bill of Materials (SBOM) in CycloneDX format to the pipeline. Two project pages read that artifact and show what is in it:

Page	Address	What it shows
“Dependency list”	`/-/security/dependencies`	The components of the SBOM: name, version, type, license, package identifier
“License compliance”	`/-/security/licenses`	The licenses across those components, with the number of components under each

Both pages are read-only and open from the “Secure” section of the project sidebar.

Availability

Both pages appear in the “Secure” section only while all of these hold; otherwise their addresses answer 404:

- the “Security and compliance” feature is enabled in the project settings;
- the user has the Developer, Maintainer or Owner role in the project;
- the feature flag `fe_security_scan_policies` is enabled on the instance;

- the license of the installation includes security scanning.

> ⚠ The feature flag `fe_security_scan_policies` is disabled by default. While it is off, both pages stay hidden, along with the policy and scanner integration pages. Ask an instance administrator to enable it.

Where the data comes from

Both pages read the latest pipeline of the project's **default branch**, whatever its status: a running pipeline has no artifact yet, and a failed one still carries the reports its jobs uploaded with `artifacts:when: always`.

From that pipeline they take every job artifact of the `cyclonedx` report type. The `codescoring` scan writes one: its `codescoring_sca` job publishes `gl-sbom.cdx.json` as `artifacts:reports:cyclonedx`. A job in the project's own `.gitlab-ci.yml` that uploads a report of the same type reaches these pages the same way.

Components from every such artifact are merged into one table. A component is kept once, recognized by its package identifier, and by its name and version when it declares no identifier. Both pages therefore show what all the analyzers of the pipeline reported together.

Nothing is stored in the database: every request reads and parses the artifacts again. Refreshing the data means running a new pipeline on the default branch.

Above the table both pages show where the SBOM came from: the pipeline status, its number, the commit, when it ran, and a download button per CycloneDX artifact — “Download SBOM”, or “Download SBOM (`<job name>`)” when the pipeline holds more than one. A user without access to the project's job artifacts sees no button. The line below names the branch: “Data comes from the most recent pipeline on main. Run a new pipeline to refresh it.”

This block is absent for a project whose default branch has no pipeline at all.

Dependency list

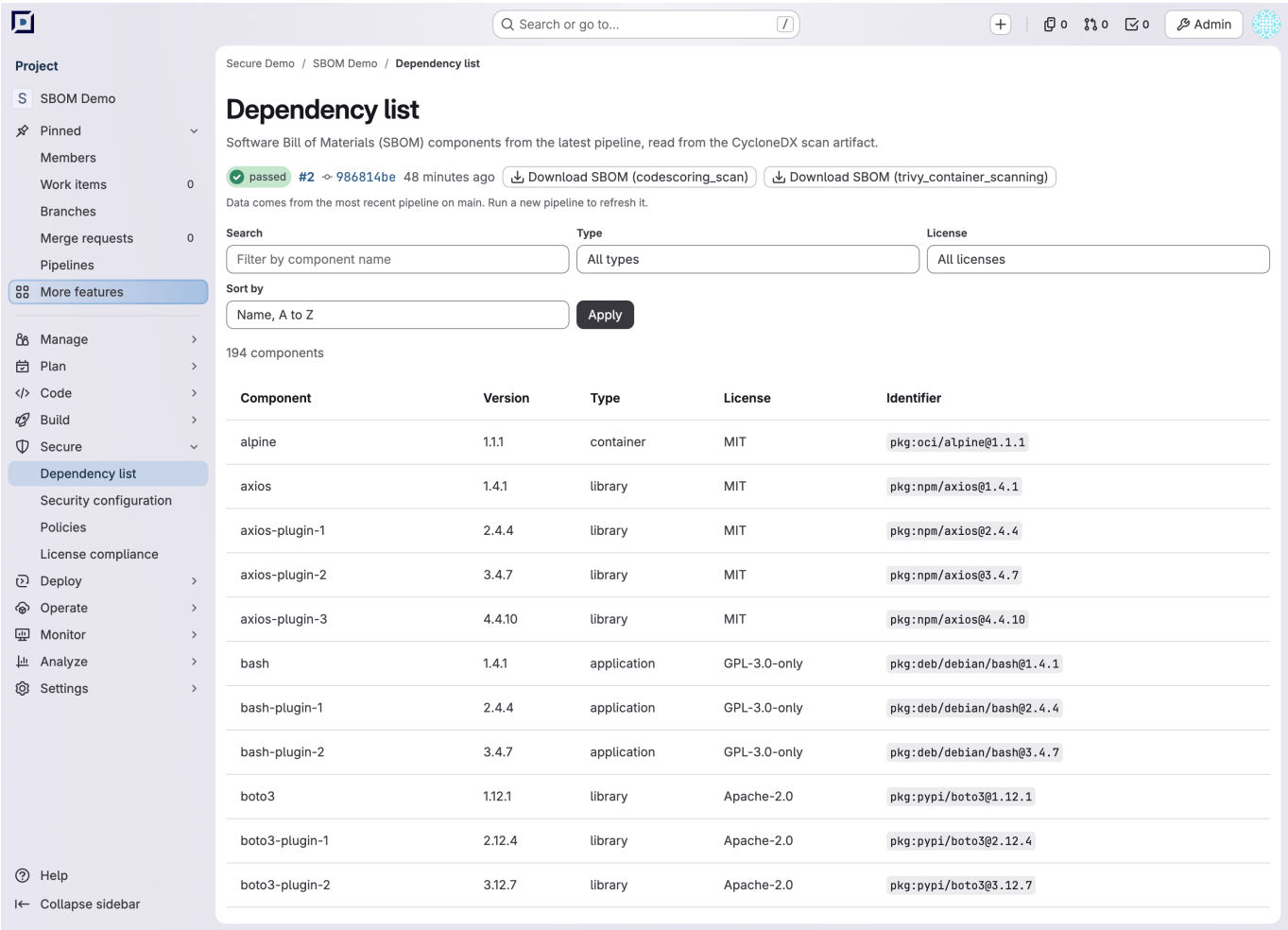


Table columns:

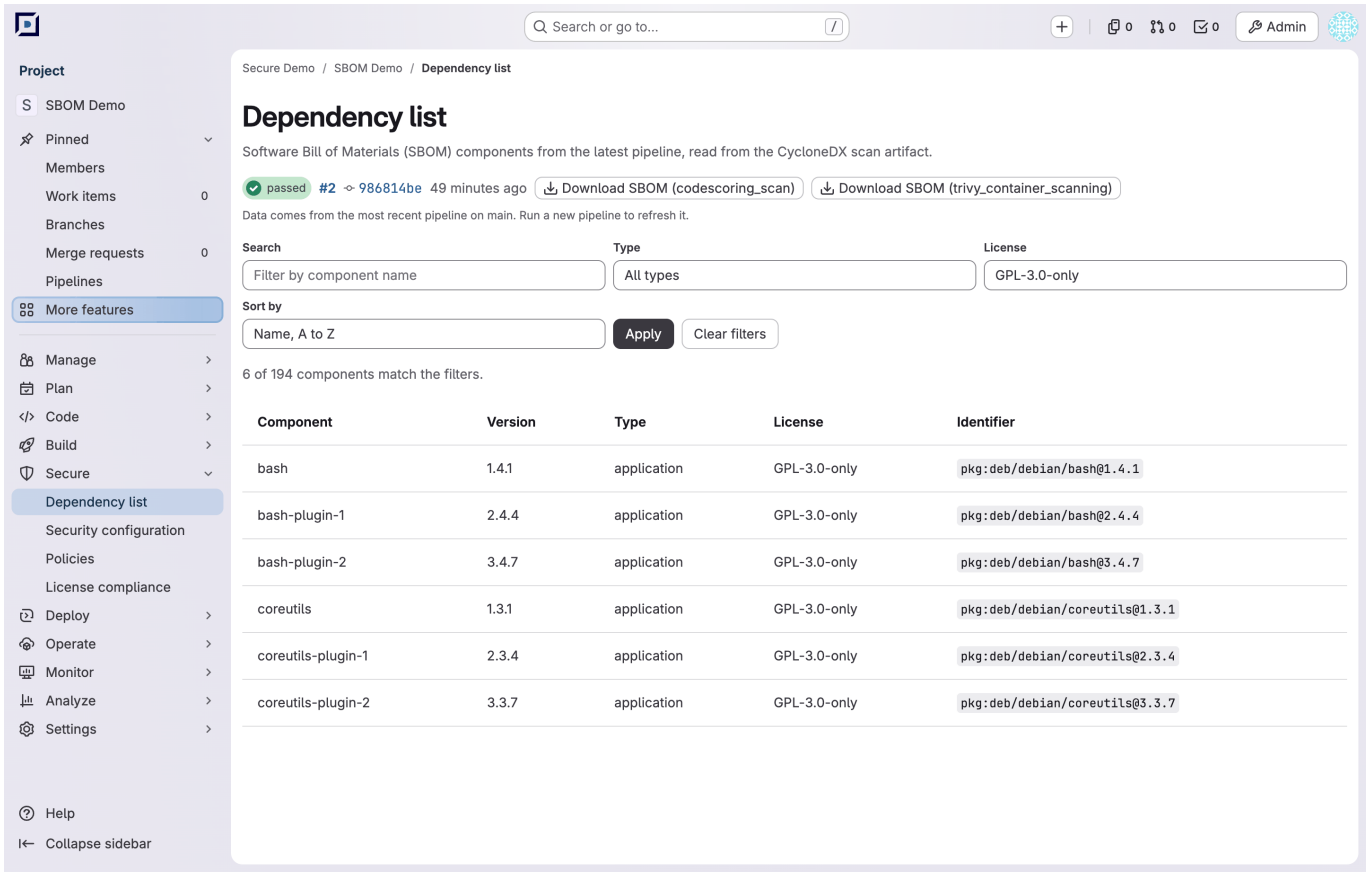
Column	What it holds
“Component”	The component name from the SBOM
“Version”	The component version
“Type”	The CycloneDX component type: `library`, `application`, `container` and others
“License”	The licenses the component declares, separated by commas; “Unknown” when it declares none
“Identifier”	The package identifier of the component in purl format

The table holds 50 rows per page. Above it stands the number of components in the SBOM, replaced by “6 of 194 components match the filters.” while a filter is set.

Controls above the table:

Control	What it does
“Search”	Keeps the components whose name contains the entered text, in any case
“Type”	Keeps the components of one type
“License”	Keeps the components under one license; “Unknown” keeps the ones that declare none
“Sort by”	Orders the table: “Name, A to Z” (default), “Name, Z to A”, “Type”, “License”

“Apply” applies what the controls hold. “Clear filters” appears once a filter is set and returns the whole list. The “Type” and “License” lists are built from the whole SBOM, so narrowing one of them keeps the choices of the other.



The controls are kept in the address, so a filtered list can be bookmarked or sent to a colleague: `/-/security/dependencies?license=GPL-3.0-only&sort=name_desc`.

A filter that matches nothing shows “No dependencies match the filters” with the hint “Change or clear the filters to see more components.”

License compliance

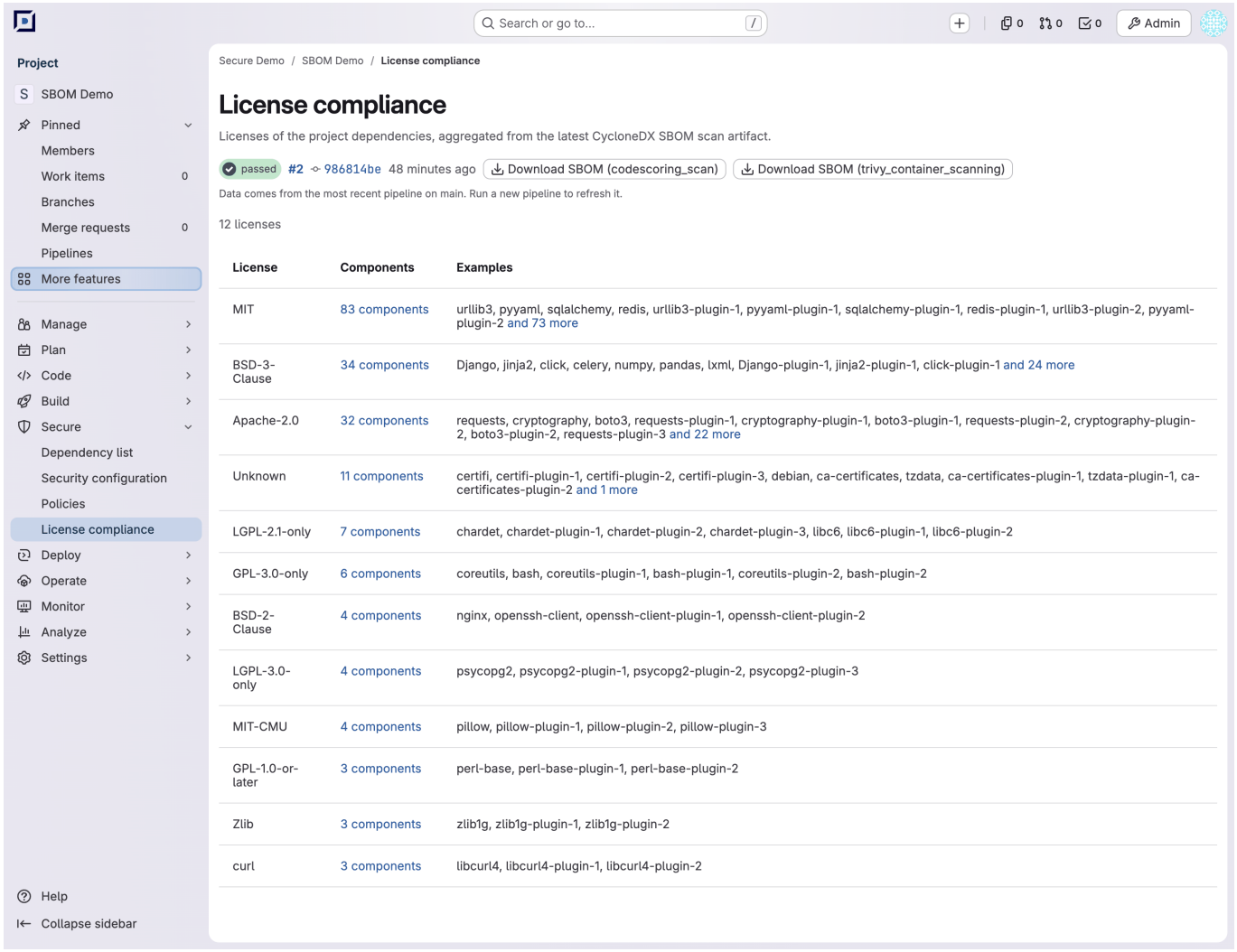


Table columns:

Column	What it holds
“License”	The license identifier from the SBOM; “Unknown” gathers the components that declare none
“Components”	How many components are under the license. The number links to the dependency list filtered by that license
“Examples”	Up to ten component names; “and N more” links to the same filtered list

Rows are ordered by the number of components, largest first, and alphabetically between equal counts. A component that declares several licenses is counted under each of them, so the counts add up to more than the number of components in the SBOM.

“Unknown” holds the same components on both pages, so the count in a row and the filtered list it opens agree.

Report states

What the pipeline holds	What the pages show
No CycloneDX artifact	“No SBOM results yet” and “Run a pipeline with the CodeScoring scanner enabled to see project dependencies here.”
Artifacts, none of which could be read	“Could not read the SBOM report” and “The latest scan artifact could not be read. Check the CodeScoring job logs and re-run the pipeline.”
At least one artifact that was read	The tables. An SBOM with no components gives “No dependencies found” or “No licenses found”

One artifact that cannot be read leaves the rest in place: the pages show what the readable ones reported. The pipeline block with the download buttons is shown in all three states, so an unreadable artifact can be downloaded and examined.

Troubleshooting

The pages are missing from the “Secure” section

Check the conditions listed in “Availability”: the project setting, your role in the project, the feature flag and the license of the installation.

“No SBOM results yet” after a pipeline that passed

The latest pipeline of the default branch uploaded no artifact of the `cyclonedx` report type. Check that:

- the pipeline that ran the scan is on the default branch: a pipeline of another branch or of a merge request leaves these pages empty;
- the scan job finished and uploaded its artifacts, which the “Jobs” tab of the pipeline shows;
- the path in the job’s `artifacts:reports:cyclonedx` key matches a file the job wrote.

“Could not read the SBOM report”

Download the artifact from the pipeline block and open it: the pages read a CycloneDX document in JSON. The `codescoring_sca` job writes the format named by the `FE_SCANS_CODESCORING_BOM_FORMAT` variable, `cyclonedx_v1_6_json` by default; another format is uploaded to the same artifact and cannot be read here.

Integrations

Webhooks

Webhooks are an event-driven integration mechanism with external systems. They enable automatic sending of HTTP requests when specific events occur in Deckhouse Code:

- Support for a wide range of events: Push, Merge Request, Issue, Pipeline, Release, and others.
- Request configuration: choice of method (POST, PUT), JSON format, header customization.
- Security features: Secret Token, SSL/TLS, event filtering.
- Support at the project, group, and instance levels.
- Integration with CI/CD, monitoring, messaging platforms, and tracking systems.
- Retry mechanism for connection failures.

> Project webhooks are available in Deckhouse Code.

Group webhooks

To add a webhook at the group level, open the group page and navigate to “Settings” → “Webhooks”. Then select the desired events. Group webhooks support all project events plus:

- Member events;
- Project events;
- Subgroup events.

Member events trigger upon creation, modification, or deletion of group or project members.

> If the user has no public email specified, the email in the request body will appear as “[REDACTED]”.

Creating group webhooks

Request header:

```
X-Gitlab-Event: Member Hook
```

Example request body:

```
{
  "created_at": "2025-07-02T15:23:25Z",
  "updated_at": "2025-07-02T15:35:51Z",
  "group_name": "agriculture",
  "group_path": "agriculture",
  "group_id": 1130,
  "user_username": "reported_user_barabara",
  "user_name": "Estella Gleason",
  "user_email": "[DELETED]",
  "user_id": 58,
  "group_access": "Guest",
  "expires_at": "2025-07-09T00:00:00Z",
  "event_name": "user_add_to_group"
}
```


Updating group webhooks

Request header:

```
X-Gitlab-Event: Member Hook
```

Example request body:


```
{
  "created_at": "2025-07-02T15:23:25Z",
  "updated_at": "2025-07-02T15:36:21Z",
  "group_name": "agriculture",
  "group_path": "agriculture",
  "group_id": 1130,
  "user_username": "reported_user_barabara",
  "user_name": "Estella Gleason",
  "user_email": "[DELETED]",
  "user_id": 58,
  "group_access": "Guest",
  "expires_at": null,
  "event_name": "user_update_for_group"
}
```


Deleting group webhooks

Request header:

```
X-Gitlab-Event: Member Hook
```

Example request body:

```
{
  "created_at": "2025-07-02T15:23:25Z",
  "updated_at": "2025-07-02T15:36:21Z",
  "group_name": "agriculture",
  "group_path": "agriculture",
  "group_id": 1130,
  "user_username": "reported_user_barabara",
  "user_name": "Estella Gleason",
  "user_email": "[DELETED]",
  "user_id": 58,
  "group_access": "Guest",
  "expires_at": null,
  "event_name": "user_remove_from_group"
}
```


Project events

Trigger on creation or deletion of projects in groups and subgroups.

Creating project webhooks

Request header:

```
X-Gitlab-Event: Project Hook
```

Example request body:


```
{
  "event_name": "project_create",
  "created_at": "2025-07-02T15:40:09Z",
  "updated_at": "2025-07-02T15:40:09Z",
  "name": "rspec",
  "path": "rspec",
  "path_with_namespace": "flant-development/agriculture/rspec",
  "project_id": 28,
  "project_namespace_id": 1130,
  "owners": [
    {
      "name": "Administrator",
      "email": "[DELETED]"
    }
  ],
  "project_visibility": "private"
}
```

Deleting project webhooks

Request header:

```
X-Gitlab-Event: Project Hook
```

Example request body:


```
{
  "event_name": "project_destroy",
  "created_at": "2025-07-02T15:40:09Z",
  "updated_at": "2025-07-02T15:42:04Z",
  "name": "rspec",
  "path": "rspec",
  "path_with_namespace": "flant-development/agriculture/rspec",
  "project_id": 28,
  "project_namespace_id": 1130,
  "owners": [
    {
      "name": "Administrator",
      "email": "[REDACTED]"
    }
  ],
  "project_visibility": "private"
}
```


Subgroup events

Trigger on creation or deletion of subgroups.

Creating subgroup webhooks

Request header:

```
X-Gitlab-Event: Subgroup Hook
```

Example request body:

```
{
  "created_at": "2025-07-02T15:44:02Z",
  "updated_at": "2025-07-02T15:44:02Z",
  "event_name": "subgroup_create",
  "name": "finances",
  "path": "finances",
  "full_path": "flant-development/finances",
  "group_id": 1659,
  "parent_group_id": 1123,
  "parent_name": "Flant development",
  "parent_path": "flant-development",
  "parent_full_path": "flant-development"
}
```

Deleting subgroup webhooks

Request header:

```
X-Gitlab-Event: Subgroup Hook
```

Example request body:

```
{
  "created_at": "2025-07-02T15:44:02Z",
  "updated_at": "2025-07-02T15:44:02Z",
  "event_name": "subgroup_destroy",
  "name": "finances",
  "path": "finances",
  "full_path": "flant-development/finances",
  "group_id": 1659,
  "parent_group_id": 1123,
  "parent_name": "Flant development",
  "parent_path": "flant-development",
  "parent_full_path": "flant-development"
}
```

Advanced search

Search in Deckhouse Code helps you quickly find the information you need across projects, groups, or the entire instance. Results are ranked by relevance and allow you to jump directly to the source object.

Advanced search powered by OpenSearch allows you to:

- find code patterns across all accessible projects;
- track usage of deprecated functions and libraries;
- search comments on issues and merge requests;

- find commits by message or SHA;
- search wiki page content.

Use advanced search

- An administrator must enable advanced search (OpenSearch).

To search:

1. In the top bar, select “Search”.
2. Enter your search term.
3. Press “Enter”.

You can also use advanced search in a project or group context.

When OpenSearch is enabled, Deckhouse Code uses it as the backend for advanced search scopes (issues, merge requests, code, and others). For REST API parameters, filters, and response format, see the “Search API” section.

Search scopes

Scopes describe the type of data you are searching.

Basic search

The following scopes are available for basic search (without OpenSearch):

Scope	Global	Group	Project
Code	✗	✗	✓
Comments	✗	✗	✓
Commits	✗	✗	✓
Issues	✓	✓	✓
Merge requests	✓	✓	✓
Milestones	✓	✓	✓
Projects	✓	✓	✗
Users	✓	✓	✓
Wiki	✗	✗	✓

Advanced search

When OpenSearch is enabled, the following scopes are available:

Scope	Global	Group	Project
Code	✓	✓	✓
Comments	✓	✓	✓
Commits	✓	✓	✓
Issues	✓	✓	✓
Merge requests	✓	✓	✓
Milestones	✓	✓	✓
Projects	✓	✓	✗
Users	✓	✓	✓
Wiki	✓	✓	✓

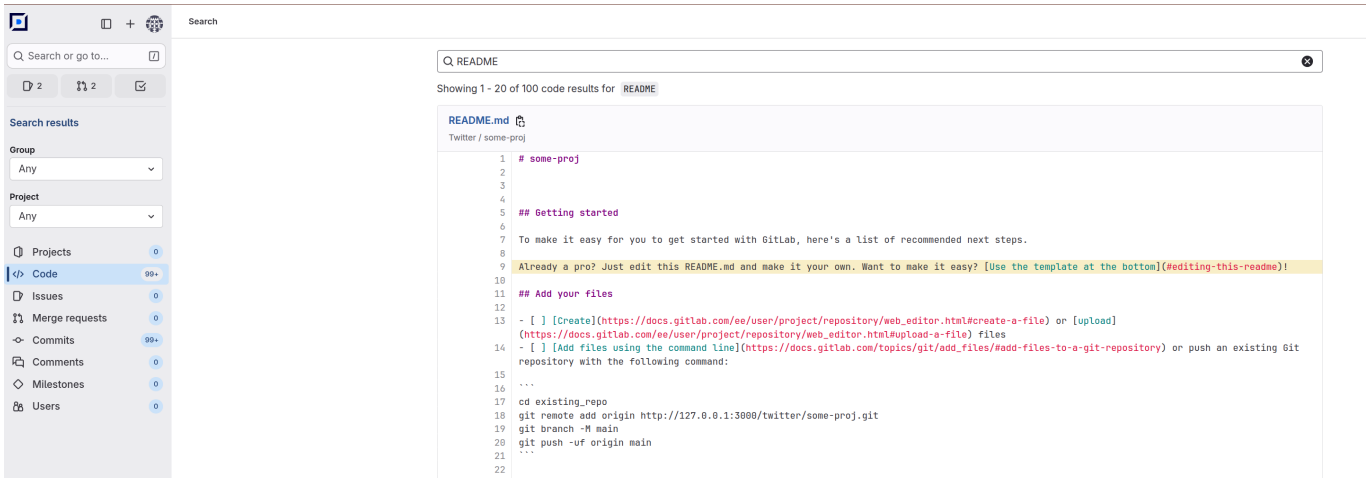
When OpenSearch is enabled, search for code, commits, wiki, and comments runs through OpenSearch and respects the **access matrix**. Users see only objects they have permission to read. Search for issues, merge requests, and other entities runs through the database.

> The tables above describe search scopes in the web interface. The set of scopes in the REST API differs: there, search for code, commits, comments, and wiki is only available at the project level. For details, see “API search scopes”.

Using search

General procedure for searching in Deckhouse Code:

1. Click “Search” in the top navigation bar.
2. Enter a search query.
3. Press “Enter” — results appear on the search page.
4. Use filters to refine results by group, project, or object type.



Global search

Allows searching across all projects and groups within the instance.

1. In the left menu, select “Search”.
2. Enter a query and press “Enter”.

Project search

1. Open the target project.
2. In the left menu, select “Search”.
3. Enter a query and press “Enter”.

Group search

1. Open the target group.
2. In the left menu, select “Search”.
3. Enter a query and press “Enter”.

Additional features

- Search supports query completion for projects, groups, and users.
- When advanced search is enabled, query completion also works for commit messages, filenames, code, issues, and merge requests.
- When searching, you can quickly navigate to a commit by its SHA.

Syntax

Advanced search supports extended query syntax: exact and fuzzy matching, logical operators, and filters.

Syntax	Description	Example
`"`	Exact search	`"gem sidekiq"`
`~`	Fuzzy search	`J~ Doe`
`|`	Or	`display | banner`
`+`	And	`display +banner`
`-`	Exclude	`display -banner`
`*`	Partial match	`bug error 50*`
`\`	Escape	`\*md`
`#`	Issue ID (in comments)	`#23456`
`!`	Merge request ID (in comments)	`!23456`

Code search

Syntax	Description	Example
`filename:`	Filename	`filename:*spec.rb`
`path:`	Repository location (full or partial matches)	`path:spec/workers/`
`extension:`	File extension without `.`	`extension:js`
`blob:`	Git object ID	`blob:998707*`

The code search UI also provides a language filter.

Examples

Query	Description
`rails -filename:gemfile.lock`	Returns `rails` in all files except `gemfile.lock`
`RSpec.describe Resolvers -*builder`	Returns `RSpec.describe Resolvers` excluding matches starting with `builder`
`bug | (display +banner)`	Returns `bug` or both `display` and `banner`
`helper -extension:yml -extension:js`	Returns `helper` in all files except `.yml` and `.js` files
`helper path:lib/git`	Returns `helper` in files with a `lib/git*` path (for example, `spec/lib/gitlab`)

Indexing settings

Project settings

A project maintainer can go to “Settings” → “Search”.

Branch regex

When **Allow per-project branch regex** is enabled at the instance level, the maintainer can specify a regex for additional branches. The default branch is always indexed.

Example regex: `(feature|hotfix)/.*`

> ⚠ Changing the regex triggers a full project reindex.

Reindex code and wiki

- **Reindex code** — full reindex of the repository code.
- **Reindex wiki** — full reindex of the wiki (if a wiki repository exists).

The “Index up to date” badge shows whether indexing is complete for the current repository state.

Group settings

A group owner can go to “Settings” → “Search”.

Group wiki reindexing is available: index status and the “Reindex wiki” button.

Search API

Search REST API lets you conduct a search across a Deckhouse Code instance, a specific group, or a project.

Endpoints

The following endpoints are available for searching:

- `GET /api/v4/search`: Search across a Deckhouse Code instance.
- `GET /api/v4/groups/:id/search` (or `/api/v4/groups/:id/-/search`): Search in a group.
- `GET /api/v4/projects/:id/search` (or `/api/v4/projects/:id/-/search`): Search in a project.

All endpoints require authentication.

API search scopes

A search scope is defined via the required parameter `scope`. Supported values differ by endpoints:

`scope` value	Instance	Group	Project	Backend when OpenSearch is enabled
`projects`				PostgreSQL
`users`				PostgreSQL
`snippet_titles`				PostgreSQL
`issues`				OpenSearch (`advanced`)
`work_items`				OpenSearch (`advanced`)
`merge_requests`				OpenSearch (`advanced`)
`milestones`				OpenSearch (`advanced`)
`notes`				OpenSearch (`advanced`)
`wiki_blobs`				OpenSearch (`advanced`)
`commits`				OpenSearch (`advanced`)
`blobs`				OpenSearch (`advanced`)

The response header `X-Search-Type` returns the resolved search type.

The set of API scopes differs from the search scopes in the web interface: the `blobs`, `commits`, `notes`, and `wiki_blobs` values are only supported by the project endpoint, while in the interface these scopes are also searchable globally and per group.

Request parameters

Common parameters

Parameter	Type	Required	Endpoints	Notes
`search`	string	Yes	All	Search query
`scope`	string	Yes	All	Search scope. See the earlier table for available values
`confidential`	boolean	No	All	Passed to search service
`include_archived`	boolean	No	Instance, group	Not available for searching in a project
`page` / `per_page`	integer	No	All	Offset pagination
`ref`	string	No	Project	Branch or tag for project search
`state`	string	No	All	Object state: `all`, `opened`, `closed`, `merged`
`type`	array[string]	No	All	Work item type filter (effective for `work_items`)

Additional parameters

Support for additional parameters depends on the selected search scope. If a parameter is submitted with invalid `scope` values, API returns the `400` response with the message `<PARAM_NAME> is supported only for <SCOPE_LIST>`.

Parameter	Type	Applies to `scope`	Restrictions
`author_username`	string	`merge_requests`	Author filter
`exclude_forks`	boolean	`work_items`, `issues`	Only in these `scope` values
`fields`	array[string]	`work_items`, `issues`	Only `title` is supported. For other values, API returns `400`
`label_name`	array[string]	`work_items`, `issues`, `merge_requests`	Comma-separated values are supported
`language`	array[string]	`blobs`	Comma-separated values are supported
`not_author_username`	string	`merge_requests`	Author exclusion filter
`not_source_branch`	string	`merge_requests`	Exclusion filter
`not_target_branch`	string	`merge_requests`	Exclusion filter
`num_context_lines`	integer	`blobs`	Supported range `0..20`
`source_branch`	string	`merge_requests`	Exact branch filter
`target_branch`	string	`merge_requests`	Exact branch filter

Response headers

The API can return the following headers:

- `X-Search-Type`: Resolved search type for current request.
- `X-Search-Aggregations`: Present only when OpenSearch is enabled and aggregations exist for the requested scope.

The aggregation scope depends on the `scope` value:

`Scope` value	Aggregations
`blobs`	`language`
`work_items`, `issues`	`work_item_type_ids`, `labels`
`merge_requests`	`labels`

Response body

The endpoint returns a JSON array of scope-specific entities:

`Scope` value	Entity type
`issues`	`IssueBasic`
`work_items`	`WorkItem`
`merge_requests`	`MergeRequestBasic`
`milestones`	`Milestone`
`notes`	`Note`
`commits`	`Commit`
`blobs`	`Blob`
`wiki_blobs`	`Blob`
`projects`	`BasicProjectDetails`
`users`	`UserBasic`
`snippet_titles`	`Snippet`

Request examples

Instance search: issues/work items with labels and fields

```
curl --request GET \
  --header "PRIVATE-TOKEN: <ACCESS_TOKEN>" \
  --url "https://code.example.com/api/v4/search?
scope=issues&search=deploy&fields=title&label_name=team%3Aplatform&exclude_forks=true"
```

Group search: merge requests with filters

```
curl --request GET \
  --header "PRIVATE-TOKEN: <ACCESS_TOKEN>" \
  --url "https://code.example.com/api/v4/groups/my-group/-/search?
scope=merge_requests&search=release&source_branch=release%2F1.2&not_author_username=bo
t"
```

Project search: code blobs with context lines

```
curl --request GET \
  --header "PRIVATE-TOKEN: <ACCESS_TOKEN>" \
  --url "https://code.example.com/api/v4/projects/my-group%2Fmy-project/-/search?
scope=blobs&search=deploy&num_context_lines=5&language=Ruby"
```