Deckhouse Code Documentation

Generated: September 09, 2026

Documentation

Deckhouse Code administrator guide

This document is an administrator’s guide for Deckhouse Code and is part of the operational documentation for the Deckhouse Code tool, which is a component of the Deckhouse Kubernetes Platform ecosystem responsible for implementing a version control system for software code.

This guide provides step-by-step instructions for configuring Deckhouse Code, including access setup, maintenance, and configuration of MergeRequestApprovals.

Access configuration

Configuring permissions for working with projects

1. Restrict group and project creation:
    - Go to “Admin” → “Settings” → “General” in the left navigation menu.
    - Locate the “Visibility” and “Access Control” section.
    - Make sure that project creation is allowed only for authenticated users.
2. Configure project settings:
    - Go to “Admin” → “Settings” → “Repository”.
    - Ensure that the default branch name and other repository operation settings comply with your security policies.

Create a group for users

1. Log in as an administrator:
    - Go to your Deckhouse Code installation using the public URL.
    - Use administrator credentials.
    - During initial setup, use the `root` user. The password is available as a secret in your cluster namespace:

      ```
      secrets/initial-root-token
      ```
2. Create a group:
    - Go to the “Admin” section — the bottom button in the left sidebar.
    - Navigate 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.

3. 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.

4. Assign roles to group members.

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.

1. Go to the desired group.
2. Select “Manage” → “Members”.
3. Assign the appropriate role to each invited user.

Monitoring and control

Resource management

1. Configure repository and artifact limits:
    - Go to “Admin” → “Settings” → “Account and Limits”.
2. Optimize system performance:
    - Go to “Admin” → “Settings” → “Network”.
3. Audit:
    - Go to “Admin” → “Event Audit”. Make sure all your actions are recorded in the audit events.

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.

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

- Check the connection settings. For more information see the `code` module documentation.
- The “Admin” → “Settings” → “Search” page displays a connection failure message.
- Search returns an error.

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:

   ```
   bundle exec rails runner 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"
```

Service account

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

To create a service account, use the Rails console provided in the Toolbox utility set. Open the console by running the following command:

```
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.

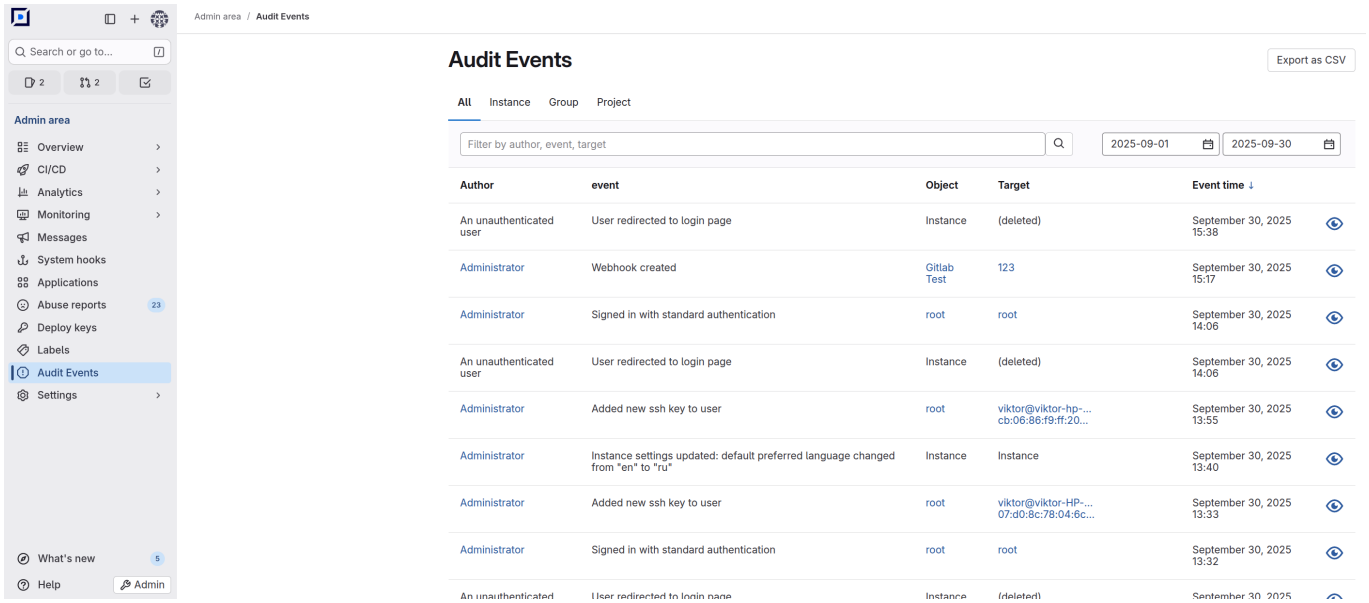
Accessing audit events

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.	

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

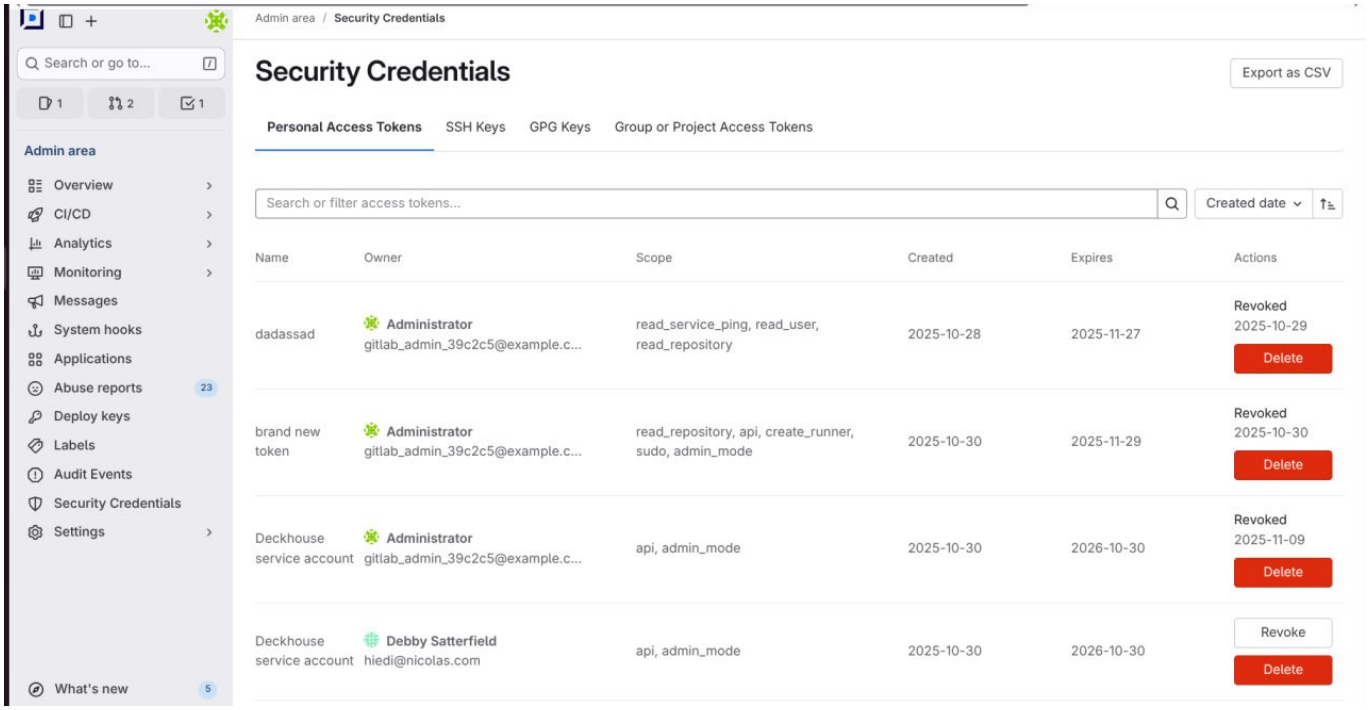
The security credentials section is used for:

- 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

Deckhouse Code user guide

This documentation is a user guide for working with Deckhouse Code — a component of the Deckhouse Kubernetes Platform that provides a version control system.

The guide includes step-by-step instructions for performing common tasks such as creating repositories, working with feature branches, managing files using Git commands, and creating merge requests.

Login

1. Open the Deckhouse Code web interface.

   To open the web interface, enter `code.<CLUSTER_NAME_TEMPLATE>` in your browser's address bar, where `<CLUSTER_NAME_TEMPLATE>` is a string that matches the DNS name template of the cluster specified in the global `modules.publicDomainTemplate` parameter. The address format may vary depending on your system configuration.

2. Log in using your existing credentials.

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

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.

Working with the codebase and 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.

Task and project management

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.

Delivery (CI/CD)

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.

User, access and change management

Security is one of the key aspects of Deckhouse Code. The tool provides built-in mechanisms for source code protection, access control, and audit logging.

Deckhouse Code implements a multi-level access control model that ensures the security of both individual repositories and the entire infrastructure.

Features:

- **Role-based access control (RBAC)** — predefined roles are supported: Guest, Reporter, Developer, Maintainer, Owner.
- **Protected branches** — direct changes are prohibited; modifications are allowed only through merge requests.
- **Protected tags** — control over tag creation and modification.
- **Authentication via external providers** — support for SAML, LDAP, and OIDC.
- **Group-wide access policies** — centralized security management for all projects within a group.

Audit and activity logging

Deckhouse Code records user and administrator actions for security auditing and analysis.

Tracked events:

- Changes in project or group settings.
- Branch and tag creation, deletion, and modification.

- User permission changes.
- SSH key addition and removal.
- Successful and failed login attempts.

Logs are available via the web interface or can be exported to external monitoring systems.

Two-factor authentication (2FA) support

To enhance account security, two-factor authentication can be enabled.

Supported methods:

- Authenticator apps (e.g., Google Authenticator, Authy).
- Hardware security keys (U2F, WebAuthn).

The administrator can enforce 2FA for all project users.

SSH and HTTPS access control

Deckhouse Code allows secure connections to repositories using the following methods:

- **SSH keys** — a secure method of authentication without using a password.
- **HTTPS + Personal Access Tokens** — an alternative method using access tokens.

Additional configuration options:

- Restrict allowed protocols (HTTPS-only or SSH-only).
- Disable password login — only keys or tokens are allowed.

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.

API

REST API and GraphQL

Deckhouse Code provides REST API and GraphQL interfaces that allow you to extend the platform's capabilities and automate routine tasks.

Example use cases:

- Automating project management: creating, updating, and retrieving project information.
- Retrieving data about users, merge requests, and commits.
- Automatically rotating user secrets and access keys.
- Programmatic search with OpenSearch-backed filters is described in "Search API".

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.

Approval rules for merger 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.

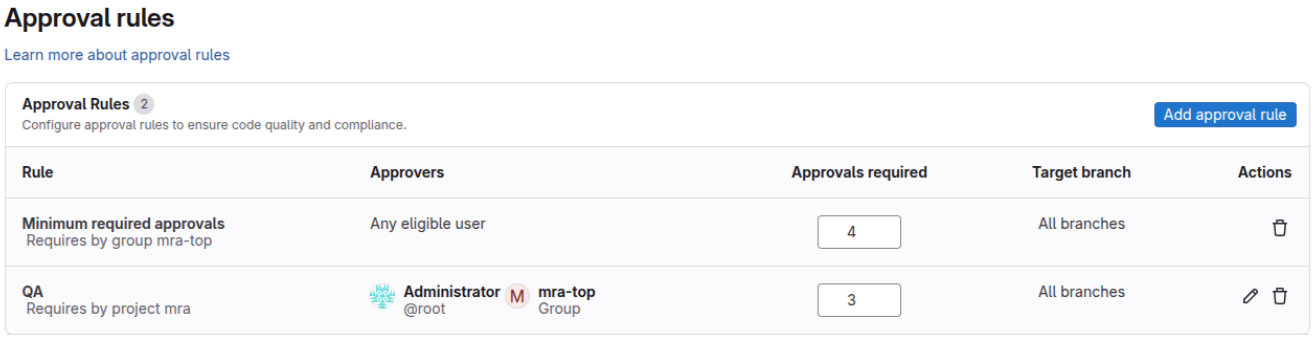
These rules operate at the group, project, and merge request levels, are inherited down the hierarchy, and enable centralized management of the review process. Together with CODEOWNERS, they form a flexible and reliable change control system, protecting important parts of the repository from unwanted 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 Gitlab Org group” means that the rule was created in the *Gitlab 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

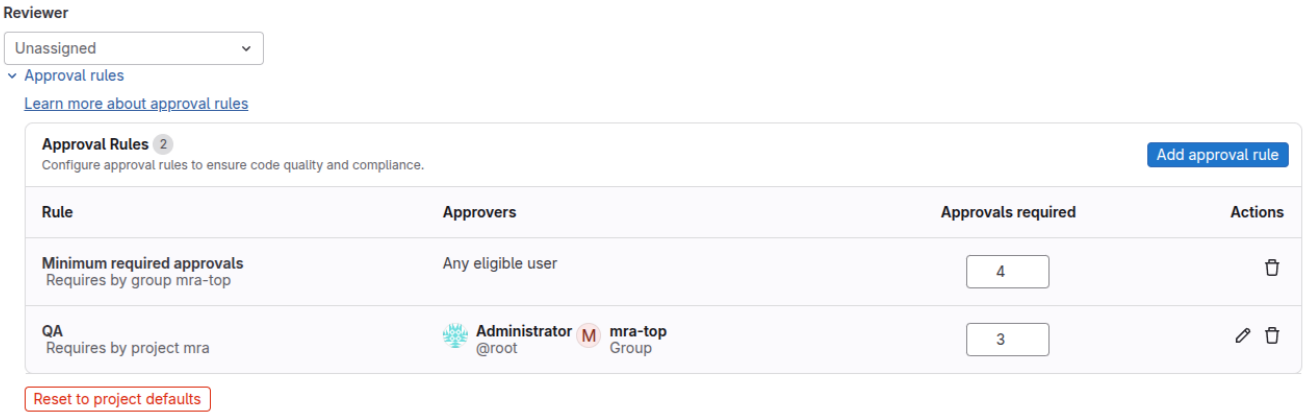
When creating a new rule, you can specify:

- 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.



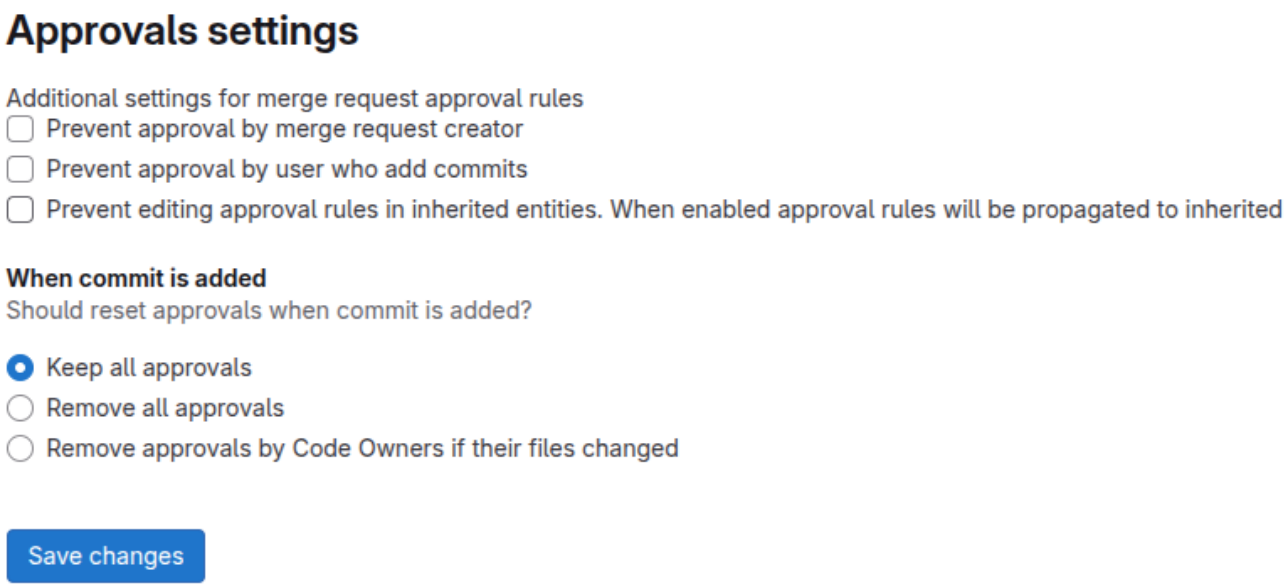
The MR page has a **“Reset to project defaults”** button that deletes the current rules and adds all project rules that apply to the target MR branch.

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.

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

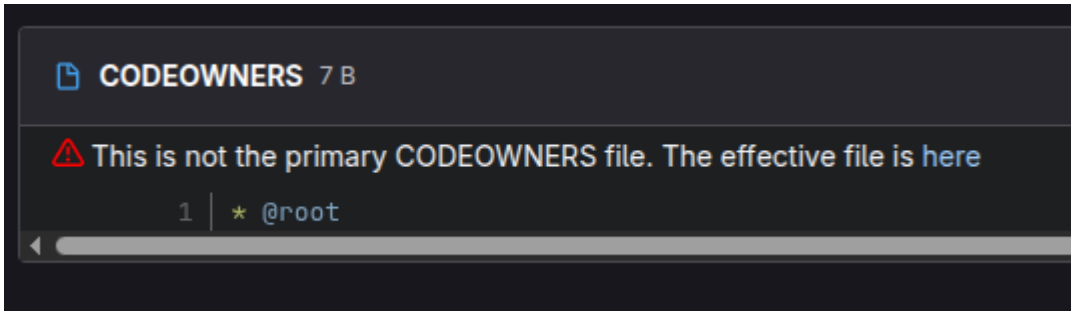
`CODEOWNERS` is a text file in the repository. Deckhouse Code uses it to determine the owners of files involved in a merge request.

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.

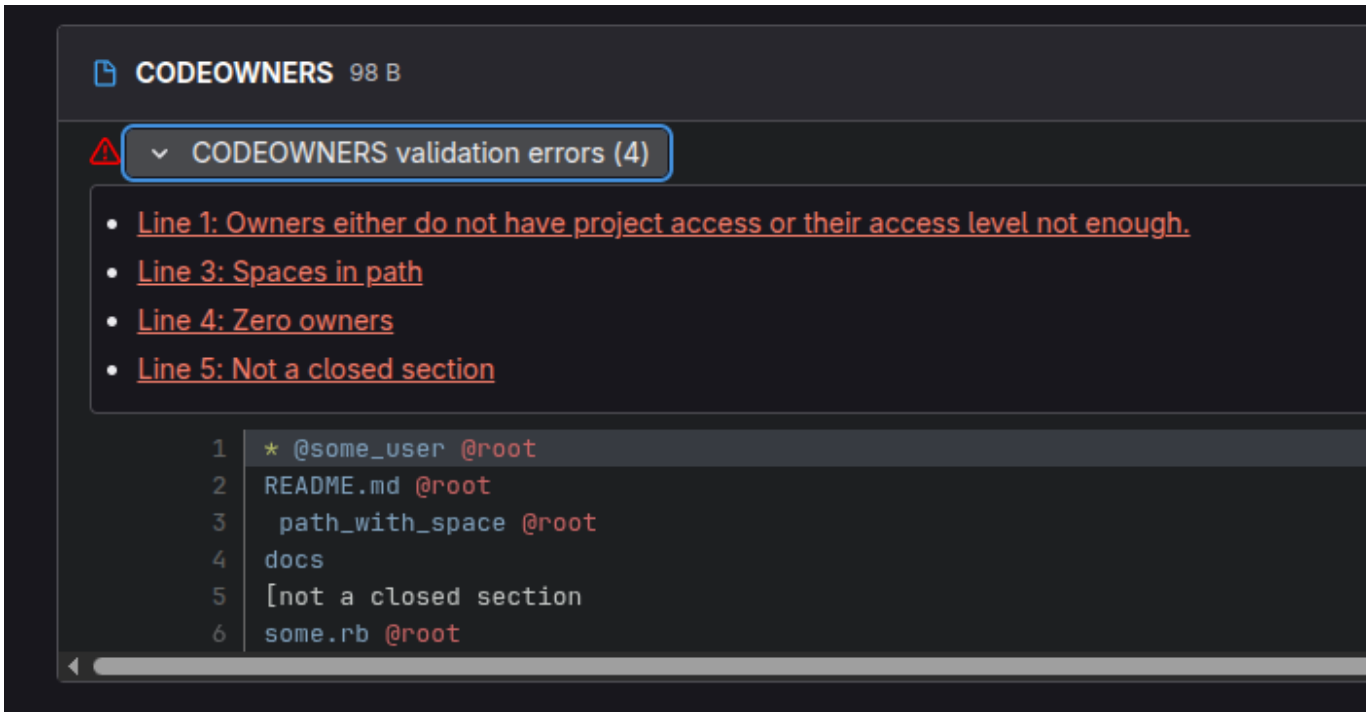


Validating the CODEOWNERS file

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
```

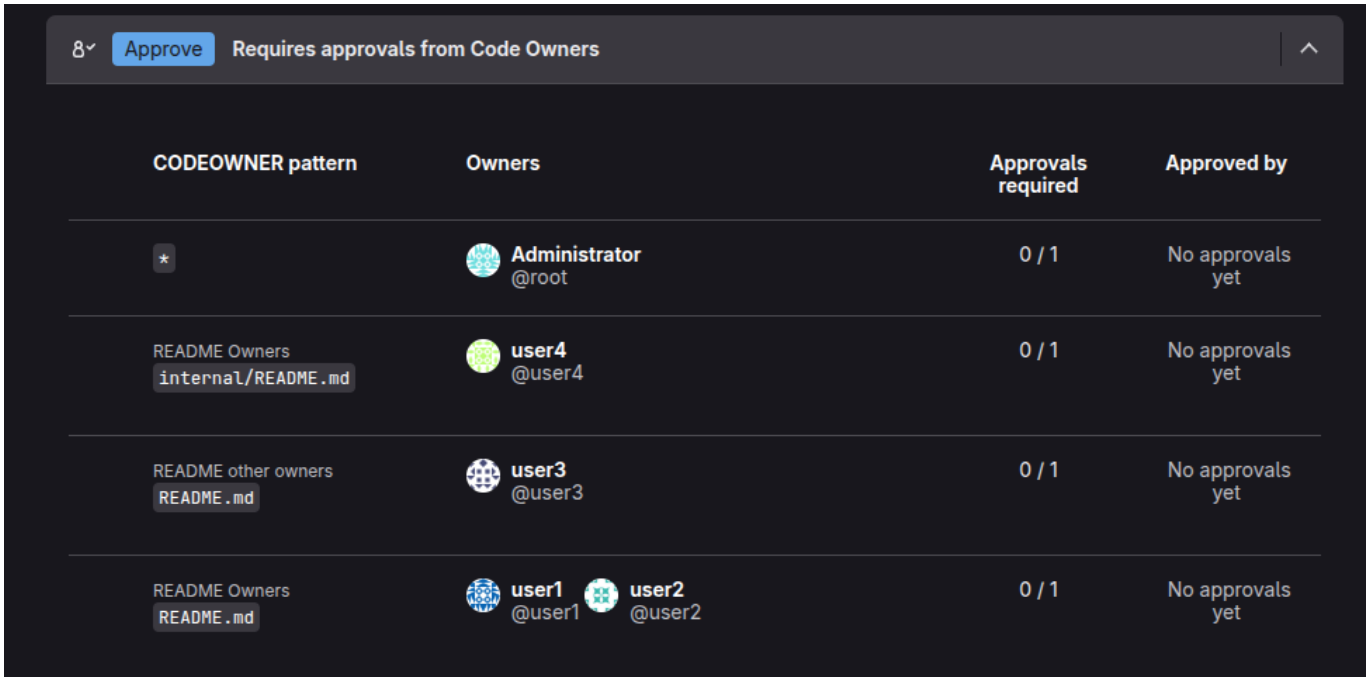
Owners for `README.md` in the root directory:

- `@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:

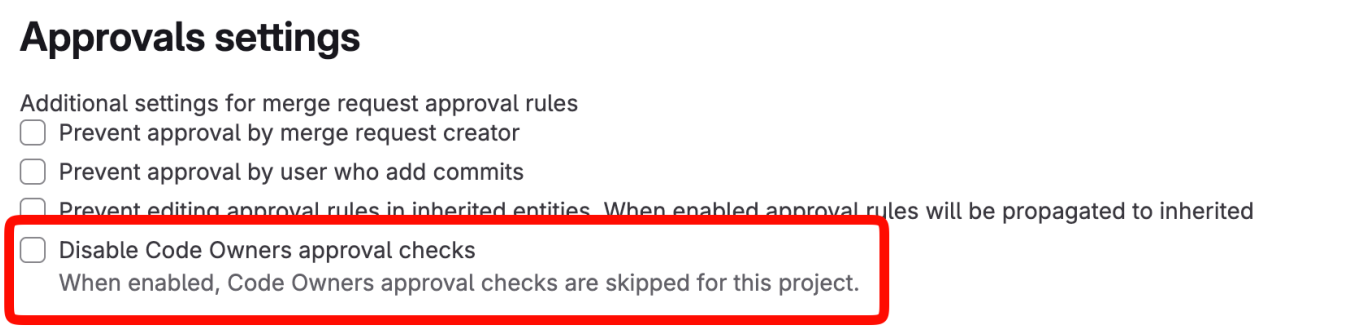
- The `Security` rule requires 1 approval.
- The `[Backend][2]` section requires 2 approvals.
- MR changes the backend file.

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:

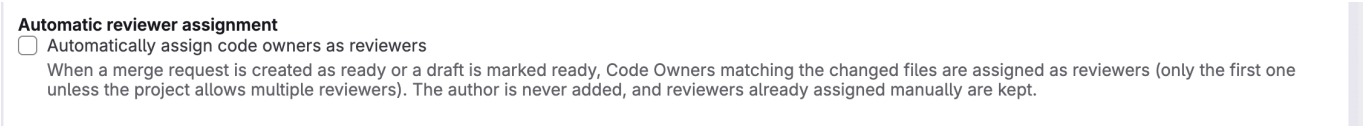
- Merge requests are not blocked by missing approvals from code owners.
- Code owners are not defined for the changed files — the list of applicable entries is empty.
- Enabling or disabling the setting is immediately applied to already open merge requests.

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
```

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="gitlab-role"
```

2. Create a role:

```
vault write auth/jwt/role/gitlab-role - <<EOF
{
  "role_type": "jwt",
  "user_claim": "sub",
  "bound_audiences": ["vault"],
  "bound_claims": {
    "project_id": "23"
  },
  "policies": ["gitlab-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 gitlab-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.

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: $VAULT_ID_TOKEN
      file: false
  script:
    - echo "Raw value (masked by GitLab):"
    - echo "$DATABASE_PASSWORD"

    - echo
    - echo "Value with spaces (not masked):"
    - printf '%s\n' "$DATABASE_PASSWORD" | sed 's/./& /g'
```

Result

If the integration is configured correctly:

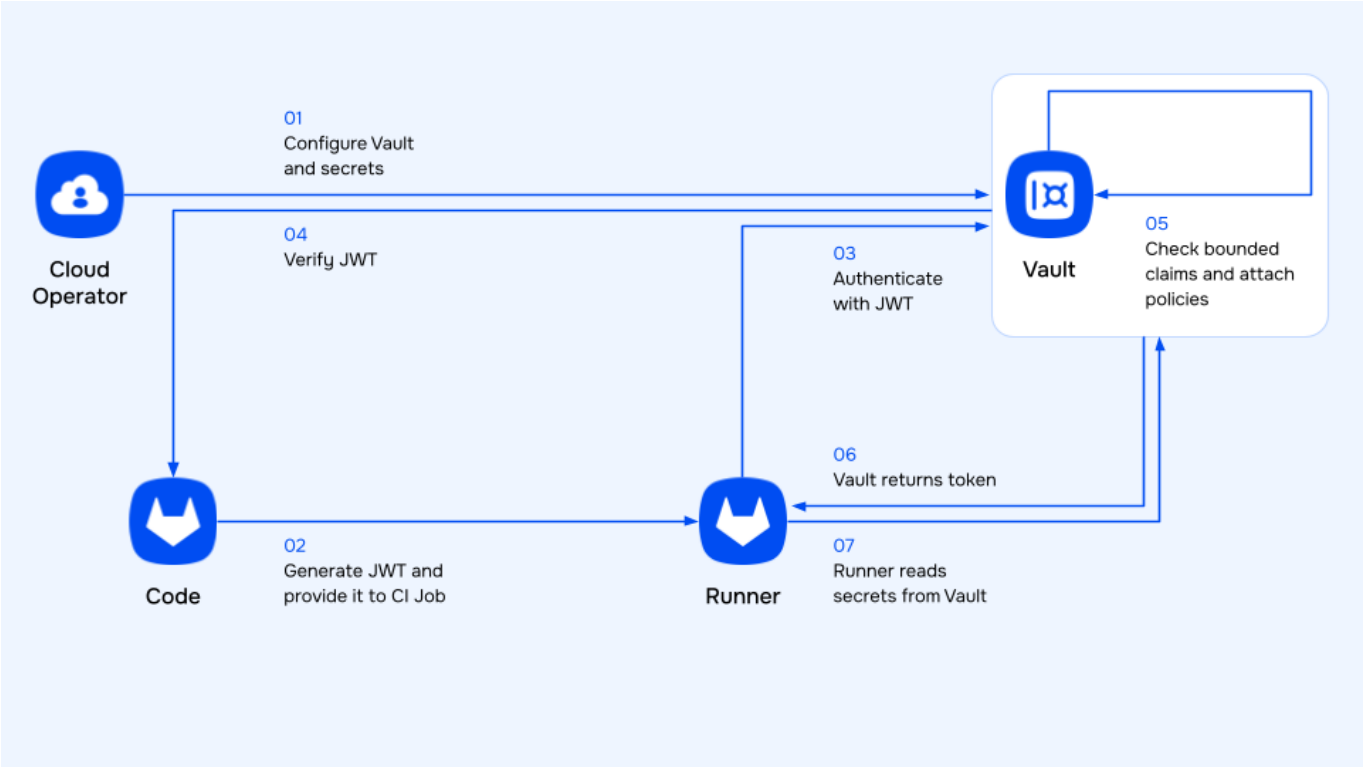
- The CI job successfully obtains an ID token.
- Vault validates the `project_path` value and grants access.
- The secret is retrieved and printed to the log.

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
```

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.

OmniAuth & LDAP

OmniAuth configuration

Deckhouse Code supports OmniAuth configuration in accordance with the GitLab official documentation. Additionally, it provides extended functionality described below.

OpenID Connect (OIDC)

The following parameters are available for integrating with OIDC providers:

- `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).
- `groups_attribute` — the name of the attribute used to extract user group information.
  Default — `'groups'`.

OIDC configuration example

This configuration is set in the `spec.appConfig.omniauth.` section:

```
providers:
  - name: 'openid_connect'
    allowed_groups:
      - 'gitlab'
    admin_groups:
      - 'admin'
    groups_attribute: 'gitlab_group'
```

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).
- `groups_attribute` — the name of the attribute that contains group information.
  Default — `'Groups'`.

SAML configuration example

This configuration is set in the `spec.appConfig.omniauth.` section:

```
providers:
  - name: 'saml'
    allowed_groups:
      - 'gitlab'
    admin_groups:
      - 'admin'
    groups_attribute: 'gitlab_group'
```

⚠ 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.

You can configure the synchronization interval using the `cronJobs` parameter in the `spec.appConfig.` section:

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

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.

Example LDAP provider configuration

The configuration is defined in `spec.appConfig.ldap.`:

```
main:
  label: ldap
  host: 127.0.0.1
  port: 3389
  bind_dn: 'uid=viewer,ou=People,dc=example,dc=com'
  base: 'ou=People,dc=example,dc=com'
  uid: 'cn'
  password: 'viewer123'
  sync_name: true
  group_sync: {
    create_groups: true,
    base: 'ou=Groups,dc=example,dc=org',
    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' }
    ]
  }
```

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 attribute.
- `group_sync.owner` — name of the user to be assigned as group owner (default is `root`).

The 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 role name in Deckhouse Code (e.g., `guest`, `reporter`, `developer`, `maintainer`, `owner`).

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 must be arrays of user 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.

Troubleshooting synchronization issues

Incorrect synchronization process

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:

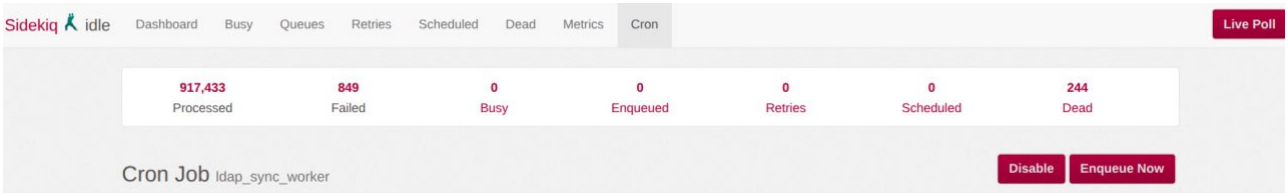
1. Connect to Redis using the databases specified in `config/redis.shared_state.yml` and `config/redis.queues.yml`.
2. Delete the key `sidekiq:concurrency_limit:throttled_jobs:{ldap/sync_worker}` using the following commands:

```
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. Connect to the cluster and determine the Sidekiq pod name: `d8 k -n d8-code -l app.kubernetes.io/component=sidekiq get pod -o NAME`
3. Run the log collection command, substituting the copied JID and pod name (POD_NAME): `d8 k -n d8-code logs POD_NAME | 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.

To allow such users to sign in through the web interface, the "Enable password and passkey authentication for the web interface" setting must be enabled.

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.

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 supported in GitLab CE.

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	✓	✓	✓

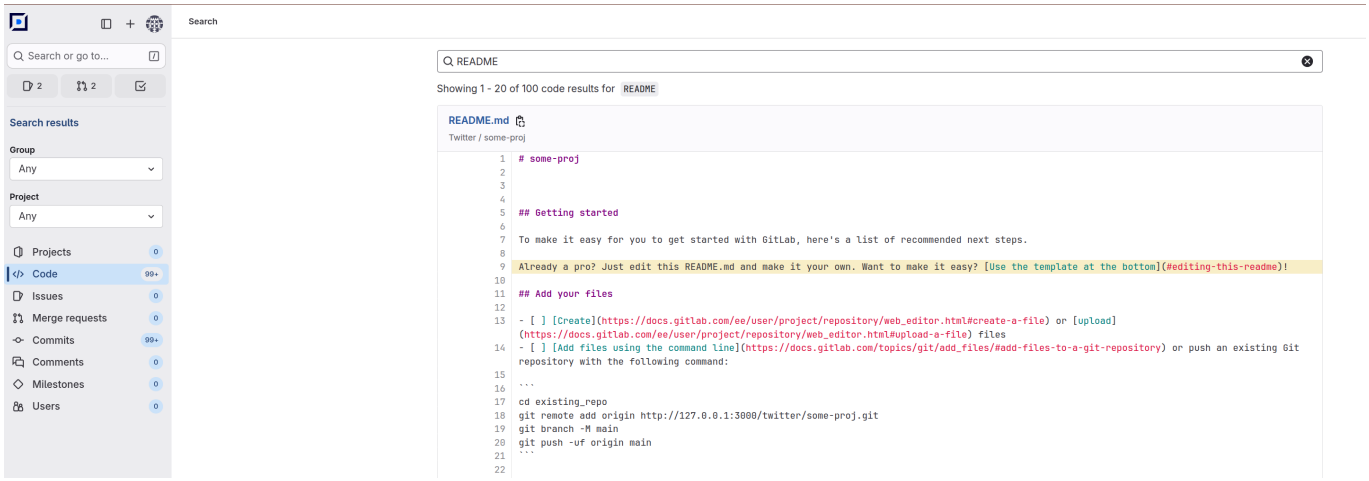
When OpenSearch is enabled, search for code, commits, wiki, and comments runs through OpenSearch and respects the **access matrix**. Users see only objects they have permission to read. Search for issues, merge requests, and other entities runs through the database.

> 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://gitlab.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://gitlab.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://gitlab.example.com/api/v4/projects/my-group%2Fmy-project/-/search?
scope=blobs&search=deploy&num_context_lines=5&language=Ruby"
```

Push Rules

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 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 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 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.

To configure push rules at the instance level, go to “Admin → Settings → Repository → Push Rules”. This option is available only to administrators.

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

Working with Git

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.