Deckhouse Galleon Documentation

Generated: September 07, 2026

Documentation

Concepts

This section explains how Deckhouse Galleon works: repositories and artifact formats, the access control model, and how proxy repositories cache external registries.

The pages read best in order: repositories and formats explain what the storage consists of and how clients address it; access control covers who may do what with that content; proxying and caching describe how Galleon works with external registries. The practice is in the administration guide and the user guide.

Repositories and formats

A repository is the basic unit of storage in Galleon. Every repository has a name, a format, and a type.

Formats

The format defines which artifacts a repository stores and which protocol clients use to work with it.

Format	Contents	Typical client
Docker (OCI)	Container images, OCI artifacts (such as Helm charts)	`docker`, `crane`, `skopeo`, `helm`
Maven	Java libraries and other build artifacts	`mvn`, `gradle`
Go	Go modules	`go`
npm	npm packages	`npm`, `yarn`, `pnpm`
PyPI	Python packages	`pip`, `uv`, `twine`
RubyGems	Ruby gems	`gem`, `bundle`
Cargo	Rust crates	`cargo`
raw	Arbitrary files	`curl`, a browser

Repository types

Two repository types are supported: hosted and proxy.

Hosted

A hosted repository stores the artifacts you publish: build results, internal libraries, release archives. The repository write policy defines what is allowed: overwriting published versions, first publication of each

version only (what is published cannot be overwritten), or a complete write ban (the repository is read-only).

Proxy

A proxy repository is a caching mirror of an external registry, such as Maven Central or Docker Hub. On the first request an artifact is downloaded from the external registry and stored; subsequent requests are served from the cache. A proxy repository is read-only; use a hosted repository for publishing. See Proxying and caching for details.

> Group repositories (several repositories under a single address) are not supported yet.

Repository address

Galleon works with the standard clients of each format. The client is configured with a repository address of the form:

```
https://<GALLEON_HOST>/repository/<REPOSITORY>/...
```

The exception is Docker (OCI) repositories: container clients access them without the `/repository/` prefix, and an image address looks like `<GALLEON_HOST>/<REPOSITORY>/<IMAGE>:<TAG>`.

How to configure the client for each format is described in the User guide.

Components and assets

Repository content consists of components and assets:

- **Component** — the unit of versioning: a package version, a tagged image, a Go module of a specific version.
- **Asset** — an individual file belonging to a component: a package archive, a POM file, an image layer.

In raw repositories every individual file is a component of its own — its name is the full file path.

A container image follows the same model: the component is the tagged image, and its assets are the manifest and the layers. Layers are shared between images: a base layer of several images is stored once. Besides tagged images, a repository may hold manifests referenced only by their checksum — the content tree marks them as `digest`.

Identical content is stored only once: a file is identified by its checksum, and the same archive published to two repositories occupies space once. Space is freed once no reference to the content is left.

The web interface shows repository content as a tree of components and assets. Deleting artifacts is available there as well, subject to permissions, and for the Maven, npm, RubyGems, PyPI, and raw formats — uploading.

Starter set of repositories

On the first start with an empty database, Galleon creates a starter set of repositories: proxy mirrors of public registries (Maven Central, proxy.golang.org, the Go checksum database, Docker Hub) and hosted repositories for Maven (separate ones for releases without overwriting and for snapshot versions), Go, and container images.

The set is created only when the repository list is empty: a repository deleted from it does not reappear on restart. The proxy mirrors of public registries have anonymous read rights; they take effect once anonymous access is enabled (see Access control).

The exact contents of the set and the settings of each repository are described in Repositories of the Administration guide.

Access control

Access to every Galleon operation, from downloading artifacts to administration, is governed by a single model: users are assigned roles, and roles bundle privileges.

Model

The model is built on three entities:

- **Privilege** — the right to perform specific actions on an object: a repository or an administrative area.
- **Role** — a named set of privileges. A role can include other roles; a user with the role gets their privileges as well.
- **User** — an account with assigned roles.

Built-in roles:

- `admin` — full access to all operations;
- `anonymous` — the privileges of this role apply to unauthenticated requests (see Anonymous access).

Privileges

A privilege allows a set of actions on one object: a repository or an administrative area — users, roles, settings, tasks.

Privileges are created automatically: the administrative ones at the first start, the ones for every action on a repository when that repository is created. There are wildcard privileges as well, such as read access in every repository.

The privilege types, their fields, how the names are built, and how to assemble the typical roles are in Privileges of the administration guide.

Actions

Repository privileges operate on five independent actions:

Action	What it allows
`browse`	Seeing the repository and its content in the web interface
`read`	Downloading artifacts
`add`	Publishing new artifacts
`edit`	Modifying already published content
`delete`	Deleting artifacts

> ⚠ Actions are independent: `read` does not imply `browse`, and `edit` does not imply `add`. There is no “administration implies write implies read” hierarchy — the required set of actions is granted explicitly.

Publishing container images is the exception: the client asks the registry for read and write access at once, so `docker push` needs both actions — `read` and `add`.

In the web interface and its API a repository without the `browse` or `read` right looks nonexistent: the server answers with `404`.

Anonymous access

Anonymous access (reading without authentication) is a combination of two settings:

1. The global anonymous access switch — disabled by default.
2. The privileges assigned to the built-in `anonymous` role — they define what exactly is available without authentication.

The privileges of the `anonymous` role apply only to unauthenticated requests: an authenticated user gets exactly the privileges of their roles, even if anonymous access is configured to be broader.

Publishing artifacts always requires authentication, regardless of the anonymous access settings.

Credentials

Two kinds of credentials are used for authentication:

- **Password** — used to sign in to the web interface; also works for command-line clients.
- **User token** — a generated pair of values used as the username and password in command-line clients and CI without exposing the account password. Each user can have one active token; it can be revoked and re-issued in the profile.

In the web interface the session persists after signing in; command-line clients pass credentials with every request.

The first administrator account is created automatically on the first start from the server configuration (see Initial setup).

Proxying and caching

A proxy repository is a caching mirror of an external registry: clients always talk to Galleon, and Galleon downloads artifacts from the external registry on demand and stores them locally.

How a request is handled

A request to a proxy repository is handled as follows:

- the artifact is in the cache and its retention period has not expired — Galleon serves it from the cache, the external registry is not involved;
- the period has expired — Galleon asks the external registry whether the artifact has changed. If it has not, the client gets the cached copy and the retention period starts over; if it has, Galleon downloads the new version, replaces the cache with it, and serves it to the client;
- the artifact is not in the cache — Galleon downloads it from the external registry, stores it, and serves it to the client.

When the external registry is unreachable

As long as a copy is in the cache, clients receive it — even if the retention period has expired and the freshness check failed. Only requests for something absent from the cache fail.

If the failures persist and automatic blocking is enabled in the repository settings, Galleon stops contacting the registry and periodically checks whether it has recovered; the cache keeps being served all that time. Outbound requests can also be paused manually — then the repository works from the cache only.

The current connection state is shown in the “Remote” column of the repository list; the values of that column are described in Repositories of the Administration guide.

Retention periods

The cache distinguishes two kinds of data:

- **Content** — the artifacts themselves: package archives, image layers. The content of such files usually never changes for a given version.
- **Metadata** — version lists, indexes, and package descriptors that the external registry updates as new versions are released.

The periods are configured per repository with two parameters: `contentMaxAge` for the content and `metadataMaxAge` for the metadata. Both accept three kinds of values:

- **A positive number** — the retention period in minutes. The default is 1440, that is a day: during that time requests are served from the cache, after it Galleon asks the external registry whether the artifact has changed.
- `0` — check the external registry on every request. The content comes from the cache if the registry answers “not modified”, but every request waits for the external registry.
- `-1` — cache indefinitely and never contact the external registry again. Suitable for immutable content and for isolated networks.

Which files belong to which class:

Format	Metadata (`metadataMaxAge`)	Content (`contentMaxAge`)
Docker (OCI)	Tags: which image a tag points to, the tag list	— (manifests and layers are immutable, see below)
Maven	`maven-metadata.xml` and its checksums	Build artifacts (`.jar` , `.pom`) and their checksums
Go	The version list and the latest-version request	— (version files are immutable, see below)
npm	The package descriptor	Version tarballs
PyPI	The project page in the index	Distribution files
RubyGems	The Compact Index files (version and name lists)	Gem files
Cargo	The crate index	Crate files
raw	—	All files

Files requested by checksum are cached indefinitely regardless of both settings: their content cannot change. These are Go module version files and the layers and manifests of container images. For images, `metadataMaxAge` sets how often Galleon checks with the external registry whether a tag points to another image.

For Maven repositories with the RELEASE version policy, content is cached indefinitely by default: release versions are never republished.

When the retention period expires, the artifact is not removed from the cache: Galleon merely checks its freshness on the next request. Cached content can be evicted with a cleanup policy — it deletes components by age, by how long ago they were downloaded, and by path (Cleanup policies).

The cache can also be dropped without waiting for the periods to expire — with the “Invalidate cache” button in the settings of any proxy repository. The content is not deleted but marked stale: every following request is first checked against the external registry and only then served. The negative cache is cleared along with it. The mark applies under any retention period, `-1` included; content requested by checksum is not affected.

Negative caching

“Not found” responses from the external registry are cached as well — for 15 minutes by default. This protects the external registry from a stream of repeated requests to nonexistent paths.

If a package has just been published to the external registry but Galleon responds that it does not exist, the negative cache is probably in effect: it expires on its own, after 15 minutes by default, and it can be dropped at once with the “Invalidate cache” button in the repository settings.

Requests to the external registry

Transient errors of the external registry (overload, network failures) are handled with retries and increasing backoff; the number of attempts is configured in the repository settings.

If the external registry requires authentication, a private container registry for example, the credentials are configured in the same place, the proxy repository settings.

Administration guide

This section is for the Galleon administrator: deploying and configuring the server, managing repositories and access, operating the storage, backups, and observability.

Overview

The administrator is responsible for the Galleon instance: deploying and configuring the server, repositories, user access, storage, and observability.

Tools

The administration tools:

- **The “Settings” section of the web interface** — repositories, users, roles, and privileges, the authentication chain and anonymous access, garbage collection, cleanup policies. All entity operations are performed here.
- **Server configuration** — the `config.yaml` file and `GALLEON_*` environment variables: addresses, the database, storage, limits, keys, and background process parameters. Applied by restarting the server.

The same operations are available programmatically — through the built-in API.

> The operations of the “Settings” section are available to any user with the corresponding privileges. Items unavailable with your permissions are rendered inactive with the “Not available with your current privileges” hint.

The administrator’s route

The typical order of bringing an instance into service:

1. Install Galleon: PostgreSQL, the storage directory, TLS.
2. Perform the initial setup: sign in as the bootstrap administrator, create working accounts.
3. Create repositories and configure access.
4. Set up backups and observability.

Installation

Deploying Galleon takes the server itself and PostgreSQL. The server is a single process, it accepts requests on port `8080` by default and can serve HTTPS itself once a certificate and a key are set (TLS). PostgreSQL holds the metadata of the installation. A reverse proxy is deployed in front of the server when needed (Reverse proxy).

The artifacts themselves are stored as files in the storage directory on the server disk (see Storage and garbage collection).

> To obtain the Galleon distribution, contact the vendor.

System requirements

Minimum server requirements:

- CPU and memory: 4 vCPU and 4 GB RAM.
- Disk: the server itself needs almost no disk space — the volume is determined by the stored artifacts (the storage directory) and the database.
- PostgreSQL version 16 or newer. With the default server settings, the database needs `max_connections` of at least 110.
- The database and the role named in the configuration are created in advance. Galleon connects to the database and runs migrations as that role, so the role needs rights to create tables and the `pg_stat_statements` extension.

First start

The minimal configuration is four blocks (the full parameter reference is in Configuration parameters):

```
database:
  host: localhost
  port: 5432
  user: galleon
  password_env: GALLEON_DATABASE_PASSWORD
  dbname: galleon

storage:
  default:
    type: local
    local:
      root: /var/lib/galleon/data

security:
  secret_key_file: /etc/galleon/secret-key.yaml

auth:
  bootstrap_admin:
    username: admin
    password_env: GALLEON_BOOTSTRAP_ADMIN_PASSWORD
```

To start the server, follow these steps:

1. Save the configuration to a file, for example `/etc/galleon/config.yaml`.

2. Create the encryption key and store it in the key file (see Secrets and the encryption key for details):

```
openssl rand -base64 32
```

```
# /etc/galleon/secret-key.yaml
active: 1
keys:
  - id: 1
    key: "<BASE64>"
```

3. Set the passwords in the environment variables named in the configuration:

```
export GALLEON_DATABASE_PASSWORD='<database password>'
export GALLEON_BOOTSTRAP_ADMIN_PASSWORD='<first administrator password>'
```

4. Start the server:

```
galleon --config /etc/galleon/config.yaml
```

On the first start, Galleon creates the first administrator and the starter set of repositories.

Migrations at startup

By default, the server brings the database schema to the required version and continues serving — no separate step is needed. When the instance has several servers, one of them runs the migrations while the others wait their turn and start once the schema is ready.

The migrations can be moved into a separate run — to update the schema before rolling out a new version, for example. Two mutually exclusive flags do that:

- `--migrate-only` — apply the migrations and exit without starting to serve;
- `--skip-migrate` — do not apply the migrations, but check that the schema is not older than the required one and start serving. If it is older, the server does not start.

If the database is still booting at startup, the server exits at once. To make the server wait for the database, set a waiting budget:

```
database:
  migrate_wait: 5m
```

The waiting covers only the signs of “the database is not ready yet”: a refused connection, an unresolvable name, a connection timeout, a database server that is starting up or recovering, a missing database. Wrong credentials are not covered — the server exits at once.

The exit code separates a temporary cause from one that needs attention:

Code	Meaning
`0`	The migrations are applied
`1`	The start failed: wrong credentials, an SQL failure. The cause needs looking into
`75`	The start was interrupted by an external cause: a stop signal or the exhausted database waiting budget. Starting the server again is enough

The migrations can be interrupted at any moment. The unfinished step is rolled back whole, and the next start resumes from it.

Verification

Make sure the server responds:

```
curl http://localhost:8080/healthz
curl http://localhost:8080/readyz
```

`healthz` confirms the process is alive; `readyz` — that the database is reachable. See Monitoring for details.

Initial setup

On the first start with an empty database, Galleon creates the first administrator and the starter set of repositories.

Bootstrap administrator

The first administrator account is described by the `auth.bootstrap_admin` configuration block: the username and the password — via an environment variable (`password_env`) or directly as a bcrypt hash (`password_hash` , takes priority).

The account is created only if the `admin` role is assigned to no user.

> ⚠ Once the administrator exists, the `bootstrap_admin` block is ignored: changing the password variable does not change the password of the existing account (the server log shows the “bootstrap_admin is configured but an admin already exists” warning). The password is changed only through the web interface.
> Galleon has no means of recovering a lost administrator password: only another user with administrative rights can reset it. Create a second administrative account right away — otherwise the only way to regain access is to deploy the instance anew.

The `force_password_change: true` option makes the bootstrap password one-time: until the password is changed, the user is blocked both in the web interface and in the format clients.

Starting state

On a fresh database, the following is created:

- the built-in `admin` role with full access and the `anonymous` role for anonymous reads;
- the privilege catalog (see Access control);
- the starter repositories (see the table in Repositories): proxy mirrors of Maven Central, proxy.golang.org, the Go checksum database, and Docker Hub — with automatic blocking enabled for an unreachable external registry — and hosted repositories for Maven (releases and snapshot versions), Go, and container images.

The `anonymous` role is pre-granted read privileges for the four public proxy mirrors, but anonymous access has no effect until the global switch is enabled (“Settings” → “Anonymous Access”). The hosted starter repositories are never anonymously accessible.

The starter set is created only when the repository list is empty: repositories deleted one by one do not come back on restart. If, however, every last repository is deleted, the set is created anew on the next start, together with their privileges and anonymous read rights.

First steps

After the first sign-in as the bootstrap administrator, do the following:

1. Create personal accounts for everyone who administers the instance and assign them the `admin` role (“Settings” → “Users”). Do not work under the bootstrap account permanently.
2. Create roles for development teams from the privileges of the repositories they need (“Settings” → “Roles”, see Access control).
3. Decide whether users may change their own passwords: the `nx-userschangepw` privilege is not part of any role by default — grant it to user roles, otherwise only the administrator changes passwords.
4. Set up TLS, the reverse proxy if you use one, and backups.

Configuration

The server is configured with a `config.yaml` file (the path is passed with the `--config` flag). Settings are applied at startup — changes require a server restart. Entities (repositories, users, roles) are not described in the file — they are managed through the web interface.

The list of all parameters with their defaults is in Configuration parameters.

Value formats

Durations and sizes are written the same way in the file and in the environment variables:

- a duration is a string with a unit of measurement: `s` for seconds, `m` for minutes, `h` for hours; composite values (`1h30m`) are allowed. A number without a unit of measurement is rejected at startup;
- a size is a string with a unit of measurement: `K` , `M` , `G` , `T` (or `KB` , `MB` , `GB` , `TB`); the multipliers are binary, so `256K` is 262,144 bytes. A number without a unit of measurement means bytes, and `0` removes the limit.

Environment variables

Any file key can be overridden with a `GALLEON_*` variable: dots become underscores, `database.host` → `GALLEON_DATABASE_HOST` , `gc.grace_period` → `GALLEON_GC_GRACE_PERIOD` .

Rules:

- an environment variable takes precedence over the file value;
- lists are comma-separated: `GALLEON_SERVER_MONITORING_ALLOWLIST="127.0.0.0/8,::1/128"` ;
- do not leave empty parent blocks in the file (for example, a bare `security:` with no keys) — they may prevent nested environment variables from applying.

Mandatory parameters

The following keys have no defaults — the server does not start without them:

- `database.user` and `database.dbname` — the database credentials;
- `storage.default.type` and the storage path for the chosen type;
- `security.secret_key_file` — the path to the encryption key file;
- `auth.bootstrap_admin` — only on the first start with an empty database.

What is configured where

The parameters are grouped into blocks:

Block	What it configures
`server`	The address and port, request limits, reverse proxy trust, access to the health endpoints
`database`	The PostgreSQL connection and connection pool sizes
`storage`	The type and root directory of the artifact file storage
`security`	The path to the encryption key file
`auth`	The first administrator, the verified credential cache, session lifetimes
`oci`	Timeouts for pulling container images from external registries
`gc`	The schedule and parameters of storage garbage collection
`index`	Background building of the service repository indexes
`logging`	The level, format, and sources of log records
`proxy`	Protection of the outbound requests of proxy repositories

Typical tasks

Cap the artifact upload size

Galleon has no limit of its own by default (`server.max_upload_size: 0`). If a reverse proxy stands in front of the server, keep its request body size in sync with this one (Reverse proxy).

To set the limit on the Galleon side:

```
server:
  max_upload_size: 10GB
```

Exceeding it reaches the client as a `413` error.

Enable a TLS connection to the database

The certificate validation mode is set by the `database.sslmode` key:

```
database:
  sslmode: verify-full
```

The values match the PostgreSQL modes: `require` encrypts the connection, `verify-ca` and `verify-full` also validate the certificate.

Collect verbose logs for diagnostics

Two keys of the `logging` block raise the log verbosity:

```
logging:
  level: debug
  registry_level: info
```

The first key raises the verbosity of the Galleon logs, the second — of the records about container image operations; it is needed when investigating image problems. Restore the previous values after the diagnostics: at the `debug` level the record volume is much higher.

Allow proxying into an internal network

Requests from proxy repositories to private IP addresses are forbidden by default: loopback, the RFC 1918 ranges (`10/8` , `172.16/12` , `192.168/16`), link-local, CGNAT (`100.64/10`), multicast, broadcast, and IPv6 ULA. The ban is checked when the repository is saved (against the addresses the external registry name resolves to), on every connection, and on redirects: the address from the `Location` header goes through the same check.

To allow a specific internal address:

```
proxy:
  ssrf:
    allowlist:
      ips:
        - 10.0.0.10
      domains:
        - registry.internal.example.com
```

The exception list accepts both IP addresses and host names; a name is matched in full, so subdomains are not covered by the exception. Private addresses outside the list stay blocked. The metadata service addresses of cloud providers (for example, `169.254.169.254`) are blocked always — they cannot be put on the exception list.

Grow the connection pools under load

Grow the pools if the database connections stopped being enough under load. Connections are taken by parallel builds, by the background index building, and especially by proxying container images: while a layer is being fetched from the external registry it keeps a connection (up to two per `docker pull`).

The signs of a shortage:

- requests respond slowly while the database itself is not loaded: there are few active queries in `pg_stat_activity`, but the number of Galleon connections sits at `max_open_conns` — meaning requests are queuing for a free connection;
- the database itself is out of slots: PostgreSQL stops accepting new connections, requests fail, and the access log starts carrying the `sqlstate` field with a class `53` code — resource exhaustion (see Logging).

The pool usage is visible in PostgreSQL:

```
SELECT application_name, count(*) FROM pg_stat_activity GROUP BY 1 ORDER BY 2 DESC;
```

The `galleon` label is the main pool, `galleon-oci-meta` is the image metadata read pool; the connections serving container images carry no label.

```
database:
  max_open_conns: 80
  registry_max_open_conns: 40
```

The defaults are 40 and 25. After the change, recalculate the database `max_connections`. Galleon connects to it with three pools: two are capped by `max_open_conns` (hence the doubling in the formula), the third by `registry_max_open_conns`. PostgreSQL also keeps a few connections in reserve for the superuser (`superuser_reserved_connections`, 3 by default):

```
max_connections >= max_open_conns × 2 + registry_max_open_conns +
superuser_reserved_connections
```

At the defaults that is 40 + 40 + 25 + 3 = 108, so for a production installation set it to at least 110. Leave a few spare slots for `psql` and one-off operations.

Configuration parameters

The complete list of `config.yaml` parameters, grouped by block. The environment variable override rules, the list of mandatory parameters, and the typical configuration tasks are described in Configuration.

Parameters without a default carry an example value: it shows the form of the value, not a recommended setting.

server

Network parameters, limits, and reverse proxy trust.

See Reverse proxy and Monitoring for details.

Parameter	Description
`listen`	The address and port the server accepts HTTP requests on. **Default:** `:8080`
`public_base_url`	The external address of the instance. It is used in the addresses the server reports to clients. **Default:** none. **Example:** `https://galleon.example.com`
`max_upload_size`	The size limit of a file uploaded to a repository. The `0` value removes the limit, and then the effective limit is set by the reverse proxy. **Default:** `0`. **Example:** `10GB`
`tls.cert_file`	The path to the certificate file in PEM format. Inside the file the server certificate comes first, the intermediate certificates after it. TLS is on only when `tls.key_file` is set as well (TLS). **Default:** none. **Example:** `/etc/galleon/tls/fullchain.pem`
`tls.key_file`	The path to the private key file in PEM format. Keys with a passphrase are not accepted. TLS is on only when `tls.cert_file` is set as well. **Default:** none. **Example:** `/etc/galleon/tls/privkey.pem`
`max_control_body`	The body size limit of a management API request — creating repositories, users, roles. **Default:** `256K`
`max_query_value_bytes`	The length limit of a single value in the query string, after the `?` sign. The `0` value removes the limit. **Default:** `4096`
`max_header_bytes`	The total size limit of request headers, in bytes. **Default:** `1048576`
`read_timeout`	The limit on reading the whole request, body included. The `0s` value removes the limit so that slow uploads can finish.

Parameter	Description
	Default: `0s`. **Example:** `30m`
`read_header_timeout`	The limit on reading the headers. It protects against connections that are opened but send no request. **Default:** `10s`
`idle_timeout`	How long an idle keep-alive connection stays open. **Default:** `120s`
`trusted_proxies`	The addresses of reverse proxies in CIDR notation or as individual IP addresses. The headers of their requests may be trusted. **Default:** `[]`. **Example:** `["10.0.0.0/8"]`
`proxy_auth.header`	The header in which the reverse proxy passes the shared secret. **Default:** `X-Origin-Token`
`proxy_auth.secret_env`	The name of the environment variable the secret is read from. **Default:** none. **Example:** `GALLEON_ORIGIN_TOKEN`
`trust_forwarded_headers`	Honor the `X-Forwarded-Host` and `X-Forwarded-Proto` headers when building addresses for clients. **Default:** `false`
`monitoring_allowlist`	The addresses allowed to reach the health endpoints. Everyone else gets the nonexistent-path response. **Default:** `["127.0.0.0/8", "::1/128"]`. **Example:** `["127.0.0.0/8", "10.0.0.0/8"]`

database

The PostgreSQL connection and connection pools.

When changing the pool sizes, recalculate the database `max_connections`: it needs `max_open_conns × 2 + registry_max_open_conns + superuser_reserved_connections`. Where the summands come from is explained in Configuration.

Parameter	Description
`host`	The database server address. **Default:** `localhost`. **Example:** `postgres.internal`
`port`	The database server port. **Default:** `5432`. **Example:** `6432`
`user`	The database role name. **Default:** none. **Example:** `galleon`
`dbname`	The database name. **Default:** none. **Example:** `galleon`
`password_env`	The name of the environment variable with the password. Without this key the connection is passwordless. **Default:** none. **Example:** `GALLEON_DATABASE_PASSWORD`
`sslmode`	The TLS mode of the database connection: `disable`, `require`, `verify-ca`, or `verify-full`. **Default:** `disable`. **Example:** `verify-full`
`connect_timeout`	The limit on establishing a database connection. **Default:** `5s`
`migrate_wait`	How long to wait for a database that is still booting before the migrations. `0` — do not wait and exit at once. **Default:** `0s`. **Example:** `5m`
`max_open_conns`	The size of each of the two main connection pools. **Default:** `40`. **Example:** `80`
`max_idle_conns`	

Parameter	Description
	How many connections these pools keep open while idle. **Default:** `5`
`registry_max_open_conns`	The size of the separate connection pool serving container images. **Default:** `25`. **Example:** `40`
`statement_timeout`	The execution limit of a single SQL query. It does not apply to migrations. **Default:** `30s`
`lock_timeout`	The limit on waiting for a database lock. **Default:** `10s`
`idle_in_transaction_session_timeout`	The idle limit inside an open transaction. **Default:** `60s`
`max_conn_lifetime`	The connection lifetime after which it is recreated. **Default:** `30m`
`max_conn_idle_time`	The connection idle time after which it is closed. **Default:** `5m`

storage

The artifact file storage. The keys are set per named storage; a backend named `default` is mandatory.

The directory layout is described in Storage and garbage collection.

Parameter	Description
`type`	The storage type. Only `local` is supported. **Default:** none. **Example:** `local`
`local.root`	The storage root directory. **Default:** none. **Example:** `/var/lib/galleon/data`

security

Encryption of the stored secrets.

See Secrets and the encryption key for details.

Parameter	Description
`secret_key_file`	The path to the encryption key file. The parameter is mandatory: without it the server does not start. **Default:** none. **Example:** `/etc/galleon/secret-key.yaml`

auth

The first administrator, the credential cache, and web interface sessions.

See Initial setup for details.

Parameter	Description
`bootstrap_admin.username`	The name of the first administrator. **Default:** none. **Example:** `admin`
`bootstrap_admin.password_env`	The name of the environment variable with the first administrator's password. **Default:** none. **Example:** `GALLEON_BOOTSTRAP_ADMIN_PASSWORD`
`bootstrap_admin.password_hash`	The bcrypt hash of the password. It takes priority over `password_env`. **Default:** none. **Example:** `$2a$12$LQv3c1yqBWVHxkd0...`
`bootstrap_admin.force_password_change`	Require a password change on the first sign-in. **Default:** `false`
`credential_cache_ttl`	How long a password check result is kept in memory. The `0s` value disables the cache, and the password is verified again on every client request. **Default:** `60s`. **Example:** `0s`
`session_max_lifetime`	The maximum lifetime of a web interface session regardless of activity. **Default:** `168h`. **Example:** `24h`
`session_idle_timeout`	The inactivity time after which a session ends. **Default:** `168h`. **Example:** `8h`

oci

Parameters for pulling container images from external registries.

Parameter	Description
`proxy.blob_idle_window`	The allowed pause in data transfer while pulling a layer from an external registry. After it the connection is dropped. **Default:** `60s`
`proxy.db_mirror_timeout`	The limit on writing the metadata of a pulled layer to the database. **Default:** `60s`

gc

Storage garbage collection. The semantics are described in Storage and garbage collection.

Parameter	Description
`auto`	Scheduled cycles. The manual run works regardless of this value. **Default:** `true`. **Example:** `false`
`interval`	The pause between scheduled cycles. **Default:** `5m`. **Example:** `15m`
`grace_period`	The minimum file age that allows its deletion. **Default:** `24h`. **Example:** `5m`
`batch_size`	How many candidate files are selected per database query. **Default:** `500`. **Example:** `1000`
`max_duration`	The duration limit of a single cycle. The remainder is handled by the next cycle. **Default:** `15m`. **Example:** `5m`

index

Background building of repository indexes — the service files Galleon generates itself: version lists, package metadata.

The number of workers is chosen by the load: the more of them, the faster the indexes are updated after publications. **The `workers: 0` value stops background index building entirely** and must not be set in a production installation.

Parameter	Description
`workers`	The number of build queue workers. **Default:** `4`. **Example:** `8`
`poll_interval`	The pause between job queue polls. **Default:** `1s`. **Example:** `5s`
`max_backoff`	The delay limit before retrying a failed job. **Default:** `1h`. **Example:** `15m`
`lease_duration`	How long a worker holds a job. If it does not finish, the job returns to the queue. **Default:** `5m`. **Example:** `10m`
`backfill_on_start`	Build the missing indexes when the server starts. **Default:** `true`. **Example:** `false`

logging

The server log.

See Logging for details.

Parameter	Description
`level`	The Galleon log threshold: `debug`, `info`, `warn`, or `error`. **Default:** `info`. **Example:** `debug`
`format`	The log record format. **Default:** `json`. **Example:** `text`
`registry_level`	A separate threshold for the records about container image operations. **Default:** `warn`. **Example:** `info`

proxy

Protection of the outbound requests of proxy repositories.

The metadata service addresses of cloud providers are always blocked, regardless of these settings.

Parameter	Description
`ssrf.block_private_networks`	Forbid requests from proxy repositories to private networks. **Default:** `true`. **Example:** `false`
`ssrf.allowlist.ips`	Addresses allowed to bypass the ban. **Default:** `[]`. **Example:** `["10.0.0.10"]`
`ssrf.allowlist.domains`	Domains allowed to bypass the ban. **Default:** `[]`. **Example:** `["registry.internal.example.com"]`

Secrets and the encryption key

Secret values are never written into `config.yaml`. The configuration file carries only environment variable names or file paths:

Key	Purpose
`database.password_env`	The name of the variable with the PostgreSQL password
`auth.bootstrap_admin.password_env`	The name of the variable with the first administrator's password
`server.proxy_auth.secret_env`	The name of the variable with the reverse proxy secret
`security.secret_key_file`	The path to the encryption key file

If a `*_env` key is set but the variable itself is empty or missing, the server does not start — the error message names the variable.

The encryption key file

`security.secret_key_file` is a mandatory parameter: without the key file the server does not start. The key encrypts the secrets stored in the database: the external registry credentials of proxy repositories and the service key that signs container image uploads. User passwords and tokens are stored as hashes and are not encrypted with this key.

The file format:

```
active: 1
keys:
  - id: 1
    key: "<base64 value of 32 bytes>"
```

The value is generated with `openssl rand -base64 32`. The server does not check the file permissions — restrict them with the means of your deployment system.

> The key file is part of the instance data. If it is lost, the server will not start, and the stored external registry credentials are lost irrecoverably. Do not regenerate the file during deployment: if the deployment tool re-creates the secret on every update, all servers stop starting. Back the file up together with the database (see Backup).

Key rotation

Rotation is performed in three phases, restarting the servers after the first two:

1. **Add.** Add the new key to `keys` with the next `id`, leaving `active` unchanged. Roll the file out to all servers and restart them.

2. **Promote.** Switch `active` to the `id` of the new key, roll out, and restart. New records are encrypted with the new key; the image upload service key is re-encrypted automatically at startup.
3. **Re-encrypt the credentials.** Open “Edit” for every proxy repository with stored external registry credentials and enter the password or token again — that is the only way they are re-saved under the new key. This does not happen automatically.

While some records are still not re-encrypted, the server writes the `secrets: upstream credentials still sealed under a non-primary key` warning at startup, with the number of such records in the `stragglers` field. Remove the old key from the file only after the warning is gone.

> ⚠ If the old key is removed without completing the third phase, repositories with stored credentials stop working: every request to them fails, and the secrets can only be restored by entering them anew.

All servers of an instance must use the same key file.

Repositories

Repositories are managed on the “Settings” → “Repositories” page. Repository types, formats, and caching are described in Repositories and formats and Proxying and caching.

Repository list

The list shows the repositories you have administration rights for. Columns: “Name”, “Format”, “Type”, “URL”, “Online”, “Remote”.

- “URL” — the address for clients; for Docker (OCI) repositories, the pull reference is shown with the “pull ref” mark.
- “Online” — whether the repository serves clients. With the `no` value, requests to it are rejected while the content stays in place: this is how a repository is taken out of service without deleting it.
- “Remote” — the state of the connection to the external registry (proxy only):

State	Meaning
“Ready”	No exchange with the external registry yet
“Available”	The external registry responds
“Auto-blocked”	Outbound requests are paused automatically: the registry is unreachable; the tooltip shows the next retry time
“Blocked”	Outbound requests are blocked manually in the repository settings
“Unavailable”	The registry is unreachable (auto-blocking is off)

The cause is printed under the state when it is known: `unauthorized` — the registry rejected the credentials; `service unavailable`, `bad gateway`, `gateway timeout` — the answer of the registry itself; `upstream connection failure` — the request never reached the registry; `background probe failed` — the background availability probe failed. A cause on the “Available” state means a special case: the registry responds but rejects the credentials while auto-blocking is off. The state is highlighted amber in that case, and uncached requests answer `502`.

Auto-blocking clears itself once the external registry responds again. To clear it early, when the registry is already fixed but the next retry window has not come yet, save the repository settings with the “Save” button: the auto-blocking counter is reset.

Creating

Click “+ New repository” and fill in the form:

1. “Name”, “Format”, “Type”, and the storage are fixed at creation — they cannot be changed later.
2. The “online” checkbox keeps the repository in service; clearing it lets you create a repository in advance and open it to clients later.
3. `writePolicy` (hosted only) — what clients are allowed to do with the content:
   - `ALLOW` — publish, overwrite, and delete;
   - `ALLOW_ONCE` — publish and delete, but not overwrite what is already published. Deleting is allowed, so “delete and publish again” remains a way to replace a version: the policy forbids overwriting rather than guaranteeing immutability;
   - `DENY` — read-only: publishing, overwriting, and deleting are all refused.

   The policy applies on every write path: in the format clients, `docker push` included, and on deletion in the web interface. For npm, RubyGems, PyPI, and Cargo the form suggests `ALLOW_ONCE` — versions of these formats are conventionally immutable.

4. For a proxy repository, fill in the caching block:
    - “Remote URL” — the external registry address; the placeholder suggests the standard address for each format;
    - `contentMaxAge` (minutes) and `metadataMaxAge` (minutes) — cache retention periods: a positive number sets minutes (1440 by default), `0` — check the external registry on every request, `-1` — cache indefinitely. Which files count as content and which as metadata is described in Proxying and caching;
    - `negativeCache.enabled` and `negativeCache.timeToLive` (minutes) — the negative cache (15 minutes by default).
5. “Block outbound connections” — a manual freeze of requests to the external registry: only the cache is served, everything else answers `404`.
6. “Auto-block outbound connections” — an automatic pause of requests while the external registry is unreachable: the cache is served, uncached requests answer `502`. Enabled by default in the form.
7. “Authentication” — the external registry credentials: type `username` (a username and password) or `bearerToken` (a static token).
8. “HTTP request settings” — the number of retries (`retries`), the timeout, and the User-Agent suffix of outbound requests.
9. “Cleanup policies” — the binding of automatic deletion policies; policies of the same format are offered. What they delete and how they run is described in Cleanup policies.
10. Format-specific fields:
    - Maven — `maven.versionPolicy` (`RELEASE` / `SNAPSHOT` / `MIXED`);
    - raw — `raw.contentDisposition` (`INLINE` — the browser renders files, `ATTACHMENT` — forced download);
    - Cargo proxy — `cargo.requireAuthentication` (the `cargo` client will send credentials);
    - Docker proxy — `insecureSkipTlsVerify` (disables TLS certificate validation of the external registry — only for trusted private registries with self-signed certificates).

Creating a repository automatically produces its privilege set (see Access control).

Editing and maintenance

The “Edit” button opens the same settings, except those fixed at creation. Stored secrets are not displayed: an empty password field means “keep the stored one” (the “Leave empty to keep the stored secret” hint); when the registry address or the authentication type changes, the secret must be re-entered.

For any proxy repository, the edit dialog offers the “Invalidate cache” button. It does not delete the cached content but marks it stale: every following request is first checked against the external registry

and only then served to the client. The negative cache, the remembered “not found” responses, is cleared as well. The operation requires the repository edit privilege; repeating it changes nothing.

The “Run cleanup” button runs the bound cleanup policies over this repository right away, without waiting for the scheduled pass. It asks for confirmation: the components that match the criteria of those policies are deleted irreversibly.

Deleting a repository

The “Delete” button removes the repository together with its artifacts. The repository disappears from the catalog at once: from that moment requests to it are answered with “not found”. The disk space is reclaimed by a subsequent garbage collection run.

If the deletion of a docker repository was not carried through (the server was stopped at that moment, for example), creating a repository under the same name answers with the `cleanup_in_progress` error. The garbage collection completes the deletion: start it with the “Run” button on the “Settings” → “Garbage Collection” page.

Starter set

On an empty database, eight repositories are created:

Repository	Format	Type	Write policy	Purpose
`maven-central`	`maven2`	`proxy`	—	Maven Central mirror; content is cached indefinitely
`maven-releases`	`maven2`	`hosted`	`ALLOW_ONCE`	Release Java artifacts (immutable)
`maven-snapshots`	`maven2`	`hosted`	`ALLOW`	Snapshot versions
`go-proxy`	`go`	`proxy`	—	proxy.golang.org mirror
`go-hosted`	`go`	`hosted`	`ALLOW`	Internal Go modules
`go-sumdb`	`raw`	`proxy`	—	The Go checksum database (sum.golang.org)
`docker-hosted`	`docker`	`hosted`	`ALLOW`	Internal container images (tags are overwritable)
`docker-hub-proxy`	`docker`	`proxy`	—	Docker Hub mirror, no credentials

All proxy repositories of the set have auto-blocking enabled. The `anonymous` role is pre-granted read privileges for the four public mirrors — the access is activated by the global anonymous access switch.

Access control

Access is managed on the “Users”, “Roles”, “Privileges”, “Anonymous Access”, and “Realms” pages of the “Settings” section. This page collects the operations: creating and disabling a user, assembling a role, enabling anonymous access, and changing the authentication chain. How the access model itself works is covered in Access control.

Users

The “Settings” → “Users” page. The “+ New user” button creates a local account: an identifier, name, email, password, status, and an initial role.

Statuses: “Active”, “Disabled”, and “Force password change” — the password becomes one-time: until it is changed, the user is blocked both in the web interface and in the format clients.

Switching to “Disabled” is the standard way to revoke access immediately: the user’s sessions are revoked, their token is deleted, and requests stop passing both in the web interface and in the clients (if there are several servers, on the others — with a delay of up to a minute). Switching back to “Active” restores access, but the token has to be generated anew.

Roles are assigned right in the list (the “Add role” button; the cross next to a role removes it) or in the “Edit” dialog (there the role list is set as a whole; at least one role).

Changing another user’s password (the “Change password” button) is available only with the `nx-all` privilege (the `admin` role); the `users` management area privileges are not enough. The user’s existing sessions and token keep working after the password change — to stop the access immediately, switch the account to “Disabled”.

Roles

The “Settings” → “Roles” page. A role is assembled from privileges (by name) and nested roles; the privileges of the nested roles reach the user as well. Built-in roles are marked “read-only”.

> Galleon will not let you remove the last administrator: an operation after which no active user would hold the `admin` role (directly or through nested roles) is rejected. The protection covers the `admin` role only: a custom role with the same privileges is not covered.

Privileges

The “Settings” → “Privileges” page shows the privilege catalog of the installation with a filter by type and lets you create your own with the “+ New privilege” button.

The privilege types and their fields, the actions on the content and on the repository itself, how the names are built, the restrictions on custom privileges, and the ready-made sets for the typical tasks are in Privileges.

Anonymous access

The “Settings” → “Anonymous Access” page — the global “Enable anonymous access” switch (off by default). What exactly is available anonymously is defined by the privileges granted to the `anonymous` role (via the “Roles”/“Users” pages).

> ⚠ Before enabling it on a public installation, audit the privilege set of the `anonymous` role: every unauthenticated request runs with these rights.

Anonymous access is disabled with the same switch. The change takes effect at once; if there are several servers, on the others — with a delay of up to a minute.

> ℹ Do not disable anonymous access by stripping the privileges off the `anonymous` role: if the role is left with neither privileges nor users, the server grants it the starter privileges anew on the next restart. Disable anonymous access only with the “Enable anonymous access” switch.

Authentication chain

The “Settings” → “Realms” page manages the composition and order of realms: a request is checked along the chain top to bottom; the first matching realm wins. Realms marked `stub` do not work yet: they reject every request.

> ✖ Do not remove the `cookie-session` realm from the active chain — without it, signing in to the web interface becomes impossible. Saving such a configuration is blocked.

Privileges

A privilege is the smallest unit of rights: it allows a set of actions on one object, a repository or an administrative area. Privileges are not granted to a user directly — a role lists them and a user is given roles (the whole model is in Access control).

Every privilege of the installation is visible on the “Settings” → “Privileges” page, and the type filter narrows the list. The catalog fills itself: the administrative privileges are created at the first start, and the privileges for a repository — together with the repository. For most tasks it is enough to pick the names you need from the catalog.

Types

The type defines what a privilege works on and which fields it has.

Type	What it works on	Fields
`repository-view`	The content of a repository: the artifacts	Format, repository, actions
`repository-admin`	The repository itself: its settings and its existence	Format, repository, actions
`application`	An administrative area: users, roles, privileges, settings, tasks	Domain, actions

The same action means different things in the two repository families: `add` in `repository-view` is publishing an artifact, `add` in `repository-admin` is creating a repository.

Actions on the content

`repository-view` privileges operate on five actions:

Action	What it allows
`browse`	Seeing the repository and its content in the web interface
`read`	Downloading artifacts with a format client
`add`	Publishing new artifacts
`edit`	Modifying already published content
`delete`	Deleting artifacts

What `read` and `browse` allow depends on where the request goes, and it is the same for every format.

`read` covers everything a format client gets in ordinary work: the file itself and the service listings it resolves dependencies from — `maven-metadata.xml` with its checksums, the version list of a Go module, an npm package descriptor, the PyPI, RubyGems, and Cargo indexes, the tag list of an image. A build agent that only downloads needs `read` alone.

`browse` is for the web interface: without it the repository does not appear in the “Browse” section even when downloading from it is allowed. The repository card and the component and asset lists open with either of the two actions.

> ⚠ Actions are independent: `read` does not imply `browse`, and `edit` does not imply `add`. There is no “administration implies write implies read” hierarchy — the required set of actions is granted explicitly.

Publishing container images is the exception: the client asks the registry for read and write access at once, so `docker push` needs both actions — `read` and `add`.

Actions on the repository

`repository-admin` privileges govern the repository itself: its settings and its existence.

Action	What it allows
`read`	Viewing the repository settings
`add`	Creating repositories
`edit`	Changing the settings, invalidating the cache of a proxy repository included
`delete`	Deleting the repository

The `add` action is the only one that does not apply to an existing repository: at the moment of creation there is none yet. The right to create repositories is therefore granted by a privilege that covers not one repository but all of them at once: `nx-repository-admin-*-*-add` allows creating repositories of any format. How such names are built is covered below.

Administrative areas

`application` privileges cover the operations that are not tied to a repository. The area is called a domain:

Domain	What it covers
`users`	Accounts: creating, viewing, changing, deleting
`roles`	Roles
`privileges`	Privileges
`settings`	System settings: anonymous access, the realm chain, cleanup policies
`tasks`	Tasks: viewing the state, running and pausing
`userschangepw`	Changing one’s own password

Administrative privileges have verbs of their own: `create`, `read`, `update`, `delete`, and `all`. A modification privilege includes reading its area: a user with `nx-users-update` sees the user list, and a separate `nx-users-read` is not needed. The `settings` domain has no `create` and `delete` — settings are not created or deleted, they are read and changed.

Privilege names

Privilege names follow a pattern made of the area or format, the repository, and the action. Knowing the pattern, you can compose the name you need yourself and find it in the catalog by search:

- `nx-<DOMAIN>-<VERB>` — administrative: `nx-users-create`, `nx-roles-read`, `nx-settings-update`;
- `nx-repository-view-<FORMAT>-<REPOSITORY>-<ACTION>` — the content: `nx-repository-view-maven2-maven-releases-read`;
- `nx-repository-admin-<FORMAT>-<REPOSITORY>-<ACTION>` — the repository itself: `nx-repository-admin-docker-docker-hosted-edit`.

The format and repository positions accept the `*` wildcard: `nx-repository-view-*-*-read` is reading in every repository, `nx-repository-view-docker-*-browse` is viewing every Docker (OCI) repository.

Two privileges fall outside the pattern:

- `nx-tasks-run` — running tasks and pausing the garbage collection. Only this privilege grants that right: `nx-tasks-read` allows viewing the state and nothing more;
- `nx-userschangepw` — changing one's own password; not part of any role by default (see Initial setup).

Full access to every operation is granted by the built-in `nx-all` privilege — it is part of the `admin` role.

Where the privileges come from

The catalog is filled without the administrator's involvement:

- the first start creates the administrative privileges for every domain and the privileges with a wildcard: for all repositories at once and for all repositories of each format;
- creating a repository produces the set of privileges for it: one per content action and one per action on the repository itself. They are removed together with the repository.

Built-in privileges are read-only: they cannot be changed or deleted. They are put into roles the same way as any other privilege.

Custom privileges

Before creating a privilege of your own, look for a suitable one in the catalog: most tasks are covered by the built-in names.

The built-in catalog leaves one case uncovered — **the administrator of all repositories of one format**. Ready-made `repository-admin` privileges with a wildcard exist only for all formats at once (`nx-repository-admin-*-*-<ACTION>`), while a privilege for "every Maven repository and no other" is created by hand: type `repository-admin`, format `maven2`, repository `*`, and the actions you need.

A privilege is created on the “Settings” → “Privileges” page with the “+ New privilege” button. The rules:

- **The name.** Names starting with `nx-` are reserved for the built-in catalog. Pick a different prefix for your own privileges, `team-` for example.
- **The format and the repository.** If a specific repository is named, it has to exist and to belong to the chosen format: a privilege with the format `maven2` and the repository `docker-hosted` is not created. If the repository position holds `*`, the format still has to be one of those Galleon supports.
- **The domain.** The domain of an administrative privilege is not checked against the list of domains. A typo such as `user` instead of `users` raises no error: the privilege is stored, it can be put into a role, and it grants nothing at all. Check it against the domain table above.

> ⚠ When deleting a repository, delete the privileges you created for it by hand as well. The automatic ones go away with it, while yours stay in the roles and start working again if a repository with the same name and format is created anew.

Typical roles

What to put into a role for a particular task:

Task	Privileges
A build agent that only downloads	`nx-repository-view-*-*-read`
A build agent that publishes to one repository	`nx-repository-view-<FORMAT>-<REPOSITORY>-add`; for container images add `-read`
A developer working through the web interface	`browse` and `read` on the repositories in question
An administrator of all repositories of one format	A custom `repository-admin` privilege with the format `docker` and `*` in the repository position, actions `read`, `edit`, `delete` — managing every Docker (OCI) repository and no other
An operator on maintenance duty	`nx-tasks-read` and `nx-tasks-run` — seeing the state of the garbage collection, running it and pausing it

A role is assembled on the “Settings” → “Roles” page. A role can include other roles: a user with such a role gets both its own privileges and the privileges of the roles included in it. Those may include roles as well, and the privileges are collected along the whole chain. This is how a base role with the common rights is made and added to the team roles. A role cannot include itself, including through intermediate roles: the server refuses such nesting.

Storage and garbage collection

Galleon stores data in two tiers: metadata (the artifact catalog, users, settings) — in PostgreSQL, artifact content — as files on disk in the storage directory (`storage.default.local.root`).

Directory layout

The storage root contains:

- `blobs/` — the content of the package formats. A file is named by the SHA-256 hash of its content, so identical data is stored once, no matter how many repositories and paths reference it.
- `docker/` — the manifests and layers of container images, with its own layout. Layers are addressed by checksum as well and are shared between images; they are tracked in separate tables of the same database.
- `galleon-storage.json` — the marker that binds the directory to the database. The server creates it on the first start; copy it together with the directory.

On a foreign or unmounted directory the server does not start: the error message starts with `storage identity:` and names the cause and the action.

An asset record is stored only in the database and points at a file in the storage. Any number of assets from different repositories may reference the same file, so deleting an asset frees space only when the last reference to the file disappears. The component and asset model is described in Repositories and formats.

A sudden power loss leaves no half-written files that the database already points at.

Garbage collection

The garbage collection does not decide what is outdated: it takes the content nothing references any more. What components to delete is decided by the administrator by hand or by cleanup policies (Cleanup policies).

An artifact deleted from the artifact catalog is not removed from disk immediately: files that are no longer referenced are found and deleted by the background garbage collection. It serves both storage trees in one cycle on one schedule. Uploads do not have to be stopped for the duration of a cycle.

The parameters are the `gc` configuration block:

Key	Default	Purpose
`auto`	`true`	Scheduled cycles; `false` disables only the schedule — the manual run always works
`interval`	`5m`	The pause between scheduled cycles
`grace_period`	`24h`	The minimum age of deleted content after which its space is reclaimed
`batch_size`	`500`	Candidates per scan query
`max_duration`	`15m`	The ceiling of a single cycle; the remainder is picked up by the next one

Because of the grace period, space is not reclaimed at once: deleted content stays on disk for at least `grace_period`. The same value applies to container images.

Observation and manual runs

The “Settings” → “Garbage Collection” page shows the state of the collection: whether a cycle is running right now, when the previous one was (“Last run”), how it ended (“Last result” — a summary such as “OK: N objects reclaimed”), and when the next one is due.

The “Run” button starts a cycle immediately. The run is asynchronous: the interface confirms that the cycle has started while the cycle itself continues in the background — closing the page does not interrupt it. Only one cycle runs at a time; a repeated start is rejected.

Viewing the page requires the `nx-tasks-read` privilege, running — `nx-tasks-run`.

The garbage collection parameters are not changed from the page — only in the server configuration, with a restart.

Pausing for maintenance

The “Pause” button on the same page stops both the scheduled cycles and the manual run. The pause lasts a limited time: a day by default, seven days at most. When the period expires it lifts itself, so a forgotten pause will not stop reclaiming space forever. The “Resume” button lifts it early. The state survives a server restart; if there are several servers, it is shared by all of them.

The pause exists first of all for backups — see Backup for details. It does not affect artifact deletion by users: a component or a repository can be deleted during the pause, the space is reclaimed later.

When the disk is full

A full disk shows up as errors on artifact uploads; in the server log, such records carry `no space left on device` in the `err` field. Reading keeps working.

To reclaim space, follow these steps:

1. Delete the unneeded artifacts in the “Browse” section of the web interface — large and outdated builds first.
2. Run the garbage collection with the “Run” button on the “Settings” → “Garbage Collection” page.
3. If that frees too little, reduce `grace_period` in the configuration and restart the server: the garbage collection never touches files younger than the grace period.

> Do not delete files from the storage directory manually. A file name is the hash of its content, so the name tells nothing about which artifact it belongs to; a file deleted by mistake is not recoverable, is not detected by the garbage collection, and requests for the corresponding artifact start failing.

Cleanup policies

Cleanup policies set the retention of repository content: the administrator describes in rules what to keep and for how long, and Galleon applies those rules itself, with no manual tidying.

The typical jobs they cover: not keeping snapshot builds longer than a month, removing versions nobody has downloaded in a long time, bounding the size of a proxy repository cache, keeping only the few latest releases of every package.

A policy is a set of criteria: age, time since the last download, path. A component is deleted only when it matches every criterion of the policy at once: a policy with the age “older than 30 days” and the path `.*-SNAPSHOT.*` deletes only what is both older than thirty days and sits under a matching path.

A policy is created for one format and applies to repositories of that format only: a Maven policy cannot be bound to an npm repository. Several policies can be bound to one repository — a component is deleted when any of them selects it. Their order does not matter: the results add up.

A policy deletes components from the catalog, while the disk space is returned by the garbage collection — it takes the content nothing references any more, no earlier than `gc.grace_period` (Storage and garbage collection).

Hosted and proxy

Policies work with repositories of both types, but the outcome differs.

In a hosted repository the deletion is irreversible: the artifact was published into Galleon, and the only way to get it back is to publish it again. For hosted repositories, therefore, choose the criteria carefully and check them with a dry run.

In a proxy repository only the cached copy is deleted. The artifact itself stays in the external registry and the next request fetches it again — users notice nothing beyond that first request being slower. Cleanup works as cache eviction here: it frees the space taken by what nobody asks for any more.

> ⚠ Cleanup does not obey the repository write policy. A repository with the `DENY` write policy is closed for client writes, but a bound cleanup policy still cleans it.

Creating a policy

The “Settings” → “Cleanup Policies” page, the “+ New policy” button.

A policy name starts with a letter, a digit, or a hyphen, contains letters, digits, hyphens, underscores, and dots, and is at most 255 characters long. Names are compared ignoring case, and the name cannot be changed after creation.

The format can be changed while the policy is not bound to any repository. For a bound policy the change is refused: remove the bindings first, then change the format.

> ℹ A policy without a single criterion deletes nothing — the list marks it as “selects nothing”. The same holds for a policy that only sets “Retain newest versions”: there is nothing to select and therefore nothing to exclude.

Criteria

A component is deleted only when it matches every criterion that is set. The criteria common to all formats:

“Component age (days)” — selects the components whose last content change across all assets is older than the given number of days. The value is an integer between 1 and 24,855.

“Component usage (days)” — selects the components none of whose assets have been downloaded for the given number of days. An asset that has never been downloaded counts from its last content change. The range is the same.

“Asset path regex” — selects the components at least one of whose assets has a matching path. The match is complete: the expression is anchored on both sides and is compared against the full path, the leading slash included. The syntax is RE2, up to 1024 characters; constructs RE2 does not have are refused when the policy is saved — `.*(?!-SNAPSHOT)\.jar`, for example.

> ⚠ The `antlr.*` expression matches no path, because it is compared against the whole path. Write `.*antlr.*` instead.

Format-dependent criteria

“Release type” is available for the Maven and npm formats and selects either releases only or prereleases only. A prerelease is defined differently: for Maven it is a version with the `-SNAPSHOT` suffix, for npm — a version carrying a semver prerelease suffix.

“Retain newest versions” keeps the given number of the newest versions of a package: they are not deleted even when they match the other criteria. The count is per package, not per repository as a whole. The criterion is available to two formats:

- Maven — versions are ordered by version number, and the criterion requires the “RELEASES” release type: the exclusion applies to release versions only;
- Docker — tags are ordered by date and are only counted in the tags scope.

Docker specifics

A Docker component is an image-and-tag pair, so a policy has two scopes and at least one of them has to be on:

- “Tags” — tagged images, one component per tag. On by default. “Retain newest versions” works in this scope only;
- “Untagged manifests” — standalone manifests that are not part of a live image index. Off by default.

Cleanup removes tags and deletes manifests; the image files themselves are collected later by the garbage collection.

When both scopes are enabled in one policy, they are evaluated in turn: the untagged manifests first, the tags second. A manifest whose last tag was removed by this very pass therefore comes up for deletion only on the next one — the nightly pass or a manual run. When planning a first large cleanup, count on an image disappearing over two passes: the tags first, the manifest after.

If the tags and the untagged manifests are split across different policies, it all depends on the binding order: a manifest policy listed after a tag policy sees the freed manifests within the current pass.

The age of a Docker component is counted from its tag: creating the tag and a push that changes what the tag points at reset the count, while a repeated push of the same content does not. The age of an untagged manifest counts from the moment it was uploaded to the repository.

The regex selects a Docker component by the manifest path only: `/v2/<IMAGE>/manifests/<TAG>` for a tagged component and the digest path for an untagged manifest. Layer paths never select a component.

Signatures and attestations

Signatures and attestations live in the repository next to the image, and cleanup treats them differently depending on how they are attached to it.

A signature attached to an image with the `subject` reference from the OCI 1.1 standard is kept exactly as long as the image itself: while the image is in the repository no policy deletes the signature, and together with the image it deletes the signature too. Signatures are published this way by `oras attach` and by `cosign` run with `--registry-referrers-mode=oci-1-1`. Nothing has to be configured for such signatures — enabling the untagged manifests scope is enough.

> A signature published under a separate tag such as `sha256-<DIGEST>.sig` (and likewise `.att` and `.sbom`) is not linked to the image in any way: to Galleon it is a separate image with a tag of its own, and a policy evaluates it on its own. Tags like these are produced by `cosign` in the `legacy` mode: that mode predates the OCI 1.1 standard and stays on until the client is told to use `oci-1-1`. A signature is pulled rarely, only when it is verified, so by time since the last download it falls under deletion even when the image itself is pulled daily. In a hosted repository the signature cannot be brought back afterwards: the policy deletes it, the image stays, and signature verification starts failing where it used to pass. The only way to get it back is to sign the image again. In a proxy repository there are no consequences: the cached copy is deleted, and the next verification fetches the signature from the external registry again.
> If a hosted repository holds signatures like these, write the regex so that only the images you mean fall under the policy, set the day windows with a margin, and always do a dry run before binding.

Binding to repositories

A policy takes effect once it is bound to a repository: the repository form (Repositories) has a “Cleanup policies” field where policies of the same format are picked.

Deleting a policy removes the binding from every repository that used it.

Running

The ways to run cleanup:

1. **On schedule** — daily at 01:00 UTC over every repository with bound policies. A repository taken out of service (“online” cleared) is skipped by the scheduled pass.
2. **Manually over all repositories** — the “Run now” button on the task card of the cleanup policies page.
3. **Manually over one repository** — the “Run cleanup” button in the repository form. A manual run works for a repository taken out of service as well.

Only one run per repository happens at a time; a repeated one answers that the cleanup is already running. The outcome of the scheduled pass is shown on the task card in the “Last run” field.

Dry run

The “Dry run” button counts the components a policy would delete from the chosen repository and deletes nothing. The policy does not have to be bound to that repository — this is the way to check the criteria before they start working.

A dry run counts per policy, so a component selected by two bound policies is counted twice; a real run deletes it once.

Images carry one more discrepancy: when a pass deletes an untagged index, the manifests it held become candidates within that same pass, and a real run may delete more than its dry run showed.

Rights

Policies belong to the `settings` domain: viewing the list requires the `nx-settings-read` privilege, while creating, changing, and deleting require `nx-settings-update` (Privileges).

Running a cleanup requires `nx-tasks-run`, the task state — `nx-tasks-read`. The run button works without the right to read the task state: only the card with the outcome of the previous pass stays hidden.

Backup

A Galleon backup consists of three parts — all three are mandatory:

- **The PostgreSQL database** — the artifact catalog of all formats (including container image metadata), users and permissions, settings.
- **The storage directory** (`storage.default.local.root` as a whole: `blobs/`, `docker/`, and the `galleon-storage.json` marker) — the artifact content.
- **The encryption key file** (`security.secret_key_file`) — without it a database dump is not restorable: the external registry credentials and service secrets cannot be decrypted (see Secrets and the encryption key).

> A backup of the database alone is useless: without the storage directory there are no artifacts, without the key file — no access to the encrypted settings.

Backup order

The consistency rule: **the database must never be ahead of the storage**.

Extra files in the storage without database records only take up space; database records without files in the storage mean lost artifacts: every download of such an artifact fails.

Therefore a backup is taken in this order: **the database first, the storage directory second**. Everything published between the two moments lands in the copy as extra files.

A concurrently running garbage collection can break this rule: if someone deletes an artifact between the database dump and the storage copy, the garbage collection may also delete the file that the already-taken dump references.

Deleted content stays on disk for at least `gc.grace_period` (a day by default), so a backup that fits into that period is consistent on its own.

If the backup window is longer than the grace period, or the period has been reduced, pause the garbage collection for the duration of the work with the “Pause” button on the “Settings” → “Garbage Collection” page (Storage and garbage collection). The same can be done with an API request when the backup is driven by a script. The pause stops both the scheduled cycles and the manual run. It lasts a limited time, a day by default, and lifts itself automatically, so a forgotten pause will not stop reclaiming space forever. The “Resume” button lifts it as soon as the backup is complete.

Both copies can also be taken as atomic snapshots of the same moment (LVM, ZFS, cloud snapshots), pausing writes.

Restore

The restore order:

1. Restore **the key file** — the same one that was in effect at the backup moment.
2. Restore **the database**.
3. Restore **the storage directory** — from a copy taken at the same time as the database dump or later. A storage copy taken earlier than the dump leaves records with no files behind, and those artifacts stop downloading.
4. Restart all servers of the instance.

Restore the storage directory together with the `galleon-storage.json` marker: without it the server does not start.

Restore the database as a whole: selectively restoring individual tables desynchronizes the service secrets, the artifact catalog, and the image metadata. Extra storage files that got into the copy remain on disk: the garbage collection does not delete them. They must not be deleted manually (Storage and garbage collection).

TLS

Galleon serves HTTPS itself when the configuration names a certificate and a key. A reverse proxy is not needed for that (Reverse proxy).

Enabling it

TLS is enabled by the `server.tls` block with two paths:

```
server:
  tls:
    cert_file: /etc/galleon/tls/fullchain.pem
    key_file: /etc/galleon/tls/privkey.pem
```

TLS is enabled when both paths are set. The `GALLEON_SERVER_TLS_CERT_FILE` and `GALLEON_SERVER_TLS_KEY_FILE` environment variables take the same values.

The requirements for the files:

- the certificate in PEM format. Inside the file the server certificate comes first, the intermediate certificates of the authority after it; ACME clients and cert-manager call such a file `fullchain`;
- the key in PEM format. Keys with a passphrase and keys in the OpenSSH format are not accepted;
- the key algorithm — RSA, ECDSA, or Ed25519.

Once TLS is on, Galleon reports addresses to clients with the `https` scheme. If the instance is reachable under a different name, set it in `server.public_base_url`.

Configuration errors

The server does not start when:

- only one of the two paths is set;
- a file cannot be read or holds the wrong format;
- the key is protected by a passphrase;
- an unknown key appears inside `server.tls`;
- the `tls` block ended up outside `server`.

Rotation

A certificate is replaced by a restart:

1. Put the new files at the same paths.
2. Restart Galleon; if there are several servers, restart each of them.
3. Make sure the intended certificate is live: at startup the log carries a `tls: server certificate loaded` record with an `sha256` field.

Protocol settings

Connections are accepted over TLS 1.2 and newer. The cipher suites:

- `TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256`;
- `TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256`;
- `TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384`;
- `TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384`;

- `TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256`;
- `TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256`;
- `TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256`;
- `TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256`.

In TLS 1.3 the cipher suites are determined by the protocol itself.

Responses served over TLS carry the `Strict-Transport-Security` header with a one-year term and `includeSubDomains`. If the instance is later moved back to HTTP, browsers that already visited it over HTTPS will open the web interface only after the HSTS state is cleared.

Diagnostics

Failed TLS handshakes are written to the log at the `warn` level with the `server.http` source: the client could not negotiate a protocol version or a cipher. The record format is described in Logging.

Reverse proxy

Galleon can serve HTTPS itself (TLS), so a reverse proxy in front of it is not required. If a proxy does stand in front of the server, set it up to the requirements below: with default settings part of the repository traffic breaks.

Requirements

Any reverse proxy must:

- proxy requests to the Galleon port; terminate TLS at the proxy, or pass the traffic on over HTTPS when TLS is configured in Galleon as well;
- **not decode percent-encoded sequences in paths**: expanding `%2F` into `/` changes the request path, and the artifact is not found under it;
- pass large request bodies through: keep the size the proxy allows in sync with `server.max_upload_size` in the Galleon configuration;
- set read and write timeouts with a margin for slow uploads of large artifacts;
- forward the `Host`, `X-Real-IP`, `X-Forwarded-For`, `X-Forwarded-Proto` headers.

Header trust

By default Galleon does not trust the `X-Forwarded-*` headers: a direct client can put any address in them, and it would reach the log as `client_ip`. Trust is enabled by two independent conditions:

```
server:
  # Condition 1: proxy addresses in CIDR notation.
  trusted_proxies: ["172.18.0.0/16"]
  # Condition 2: a shared secret in a header.
  proxy_auth:
    header: X-Origin-Token
    secret_env: GALLEON_ORIGIN_TOKEN
```

How it works:

- only `trusted_proxies` is set — the address the request came from is checked;
- only `proxy_auth` is set — the secret in the header (by default `X-Origin-Token`) is checked; the address does not matter;
- both are set — both are required at once;
- nothing is set — the forwarded headers of every request are ignored.

A request that fails the check is handled as a direct client's request: its forwarded headers are ignored.

The absolute addresses Galleon reports to clients (the `url` field in API responses, the package addresses in npm metadata) are built from the `Host` header and the scheme Galleon itself serves. When TLS is terminated at the proxy, those addresses reach clients with the `http` scheme. To have the host and the scheme taken from `X-Forwarded-Host` and `X-Forwarded-Proto`, turn on `server.trust_forwarded_headers`. The headers are honoured only for requests that pass the check above, so without the trust conditions this parameter changes nothing.

The `server.public_base_url` parameter fixes the address and bypasses the headers:

```
server:
  public_base_url: https://galleon.example.com
```

It is needed when clients have to be told the same address no matter which name the request arrived at. For example, the instance is also reachable under an internal name: a client that comes in through it receives internal package addresses in the npm metadata, and those end up in `package-lock.json`.

nginx example

The configuration below terminates TLS, proxies requests to Galleon, and passes the headers the server uses to identify the originating client:

```
upstream galleon {
  server 127.0.0.1:8080;
  keepalive 32;
}

server {
  listen 443 ssl;
  server_name galleon.example.com;

  # The upload limit is set by the proxy: without this line nginx cuts bodies at 1 MB.
  client_max_body_size 0;

  # Timeouts for large artifacts.
  proxy_read_timeout 1h;
  proxy_send_timeout 1h;

  location / {
    proxy_http_version 1.1;
    proxy_set_header Connection "";
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto $scheme;
    proxy_set_header X-Origin-Token "<GALLEON_ORIGIN_TOKEN>";
    # proxy_pass without a URI part: with one, nginx decodes %2F in paths.
    proxy_pass http://galleon;
  }
}
```

⚠ A `proxy_set_header` directive is not inherited from the previous configuration level if at least one such directive is declared on the current level. When adding separate locations, repeat the whole header set in each of them, including the secret.

Galleon ignores an inbound `X-Request-ID` header and always returns a request identifier of its own. To match the proxy's log records with Galleon's, log it in nginx through the `$upstream_http_x_request_id` variable.

Monitoring

For liveness checks, Galleon provides two endpoints without authentication:

Endpoint	Purpose	Responses
`GET /healthz`	The process is alive and responding	`200`
`GET /readyz`	The database is reachable	`200`; `503` when the database is unreachable

`healthz` suits liveness probes; `readyz` — readiness probes and the post-deployment readiness check.

Restricting access

Access to the health endpoints is restricted by the `server.monitoring_allowlist` address list; the default is loopback only, so local checks work without setup while network requests are denied.

Rules:

- the TCP connection address is checked; `X-Forwarded-*` headers are not considered — a request through a reverse proxy is judged by the address of the proxy itself;
- a denial is indistinguishable from a nonexistent path (`404`); the reason is visible only in the log (`deny_reason=monitoring_ip_denied`);
- an explicitly set empty list denies everyone, including local checks;
- the `0.0.0.0/0` value opens the endpoints to everyone.

```
server:
  monitoring_allowlist: ["127.0.0.0/8", "::1/128", "10.0.0.0/8"]
```

If the checks do not come from loopback, their addresses have to be on the list: for external monitoring through a reverse proxy that is the proxy's own address. The exact source address is visible in the access log of the rejected request — the `client_ip` field in the record with `deny_reason=monitoring_ip_denied` .

Observing the instance

Observation is built on logs and PostgreSQL tooling:

- **The server log** — three fields absent on healthy traffic provide ready-made monitors: `sqlstate` (database errors), `deny_reason` (access denials), `served_stale` (external registry degradation). See Logging for details.
- **The state of external registry connections** — the "Remote" column on the "Settings" → "Repositories" page (see Repositories).
- **PostgreSQL** — the standard tooling: activity by the `galleon` and `galleon-oci-meta` application names (the connections serving container images carry no label), database size, slow queries.
- **Disk** — the storage directory usage.

Logging

All Galleon subsystems write a single structured log stream to stdout — JSON format, one record per line. Rotation and delivery to a log storage system are the deployment platform's concern.

Configuration

The `logging` configuration block:

Key	Default	Purpose
`level`	`info`	The Galleon log threshold: `debug`, `info`, `warn`, `error`
`format`	`json`	`json` or `text`
`registry_level`	`warn`	A separate threshold for the records about container image operations

The audit events of container image operations (manifest, blob, and tag uploads and deletions) are always written, regardless of `registry_level`.

Access log

Every request writes one `msg="http request"` record with the fields: `source` (the subsystem), `method`, `path`, `status`, `size`, `user`, `client_ip`, `duration`, the database time and query count, and the trailing `request_id`. `user` carries the authenticated account; a request without successful authentication shows `-` there.

Galleon returns the same identifier to the client in the `X-Request-ID` header. It finds every record of one request in the log. `client_ip` behind a reverse proxy is correct only with trust configured (Reverse proxy).

The errors of the HTTP server itself are logged with the `server.http` source at the `warn` level. When Galleon serves HTTPS itself, failed TLS handshakes land in that source (TLS).

The fields that appear only on problem traffic:

- `sqlstate` — the request hit a PostgreSQL error;
- `served_stale=true` — a proxy repository served a stale copy because the external registry did not respond;
- `deny_reason` — an authentication or access denial; the value dictionary is below.

The `attempted_user` field is the login the client presented: the credentials were rejected, or not checked because the request was refused earlier. The `?` value means a non-Basic authorization or an

empty login. Look for failed sign-ins by this field: `user` shows `-` on such records. The field appears on normal traffic as well, for example when monitoring polls an open endpoint with credentials.

Denial reasons

The `deny_reason` field explains every denial without changing the client response (response codes do not reveal the existence of others' repositories):

Value	Reason
`no_credentials`	A request without credentials where they are mandatory
`bad_credentials`	Credentials were presented and rejected
`anonymous_disabled`	Anonymous access is off
`anonymous_forbidden`	The `anonymous` role lacks a privilege
`forbidden`	The user lacks a privilege on the repository
`admin_required`	An administrative privilege is missing
`anon_no_admin`	An anonymous container image request that requires administrative rights
`password_change_required`	Blocked until the forced password change
`repository_not_found`	The target repository was not found; also stamped on a `404` answered before the credential check
`repository_name_invalid`	The repository name could not be parsed, so no lookup was made
`wrong_format`	The repository exists but has a different format
`monitoring_ip_denied`	An address outside `monitoring_allowlist`
`upstream_manually_blocked`	Requests to the external registry are frozen manually
`upstream_auto_blocked`	Requests are paused by auto-blocking

A `401` / `403` record without `deny_reason` indicates an infrastructure failure: check the `err` and `sqlstate` fields. A `404` on a repository carries a reason as well; only the answers of a repository taken out of service and the refusals of a malformed request go without one.

Audit events

Administrative and domain events are written at the `info` level:

- artifact publications — `msg="<FORMAT> artifact published"` records with the coordinates and repository, one per artifact;
- container image operations — manifest, blob, and tag uploads and deletions;
- repository cache invalidation (`source=audit.repository`);
- bootstrap administrator creation, creation of the starter set of repositories, garbage collection cycle summaries, external registry connection state changes.

Secrets never reach the log at any level: authorization header values, passwords, tokens, and body contents are not logged. Control characters and invalid UTF-8 in client-supplied values (the path, headers, file names) are replaced with `?` in the log.

API

Galleon has a built-in REST API: every web interface operation is available programmatically. It is used to create repositories and accounts during a deployment, to export the instance state into an inventory system, and to plug in configuration management tools.

The API reference is built into the server: Swagger UI opens at `https://<GALLEON_HOST>/docs`, and the machine-readable OpenAPI specification is available at `https://<GALLEON_HOST>/openapi.yaml`. Both addresses require authentication — any account will do (the browser asks for a username and password).

Authentication

Programmatic requests are authenticated the same way as format clients: with a username and password or with a user token (Credentials). For automation, create a dedicated account and work with its token — it is revoked without touching the password.

An example: the list of the instance repositories.

```
curl -u <USERNAME>:<PASSWORD> https://<GALLEON_HOST>/service/rest/v1/repositories
```

Here `<USERNAME>` and `<PASSWORD>` are the account username and password, or the “Name code” and “Pass code” of a token.

Rights

API operations are checked against the same privileges as the buttons in the web interface: to create a repository programmatically an account needs the repository creation privilege, and to create a user — a privilege of the `users` domain (Privileges). A `403` answer means a missing privilege rather than a malformed request.

User guide

This section is for the Galleon user: working in the web interface, credentials, and connecting the standard clients for every format.

Overview

A Galleon user works with artifacts through the web interface and the standard format clients. In the web interface they browse repository contents, upload files, and manage their own profile. The standard clients of each format (`docker`, `mvn`, `npm`, and others) work with the repositories directly, configured with the Galleon address.

Getting started

To start working, follow these steps:

1. Get an account from your administrator. If anonymous access is enabled for the repositories you need, you can read them without an account.
2. Sign in to the web interface and generate a user token — it is used instead of your username and password in clients and CI (see Credentials).
3. Configure the client of your format with the repository address — per-format instructions are collected in the Formats section.

What is available to a user

User capabilities in Galleon:

- **Browsing** — the tree of components and assets of any accessible repository and file downloads: the “Browse” section of the web interface.
- **Publishing** — with the standard format clients or through the “Upload” section of the web interface.
- **Profile** — changing the password, the user token, managing sessions.

The set of visible repositories and allowed actions is determined by the roles assigned by the administrator. The access model is described in Access control.

Web interface

The Galleon web interface is available at the instance address (`https://<GALLEON_HOST>`). It lets a user browse repository contents, download and upload artifacts, and manage their own profile.

The interface is available in English and Russian. The language is switched with the “Interface language” list in the header, and on the sign-in page — on the sign-in card. The choice is remembered in the browser: it is not tied to the account and does not travel to another browser or device. On the first visit

the language follows the browser settings. Server messages, the text of a rejection, for example, stay in English.

Signing in

To sign in, enter “Username” and “Password” on the login page and click “Sign in”. After signing in, the “Browse” section opens.

Without signing in, only browsing is available — and only if the administrator has enabled anonymous access. The “Upload” and “Settings” links appear in the header after signing in.

Browsing content

The “Browse” section shows cards of the repositories available to you (name, format, type). Repository content is displayed as a tree (for the component and asset model, see Repositories and formats):

- **folders** (▪) — group the content: Maven groups, package namespaces, images;
- **components** (◆) — package versions and image tags;
- **index files** (▤) — service files that Galleon generates itself (for example, `maven-metadata.xml`).

In Docker (OCI) repositories the image content is laid out in groups: `tags/` — the tags, `manifests/` — the manifests those tags resolve to (several tags of one manifest are shown as a single row). The layers and configs are collected into a separate `blobs/` branch at the repository root: they belong to the repository rather than to an image, and the same layer may be used by several images.

Selecting a node opens a detail panel on the right:

- for a component — coordinates, format metadata (package description and dependencies, image platforms and layers), and the list of assets;
- for an asset — size, content type, checksums (SHA-256, SHA-1, MD5, SHA-512), dates, and for proxy repositories — the source URL and the cache expiry.

One of the dates is “Last downloaded” — when the file was downloaded last. The “Never” value means it has not been downloaded a single time, and “Not tracked” means downloads are not recorded: container image layers are marked this way, because they are shared across all the images of the repository.

The “Download” button downloads an asset. The “Load more” button loads the next page of long lists.

The “Delete component” and per-asset “Delete” buttons require the `delete` privilege. Under the `DENY` write policy a deletion is rejected regardless of privileges: such a repository is read-only. Deletion is irreversible; when the last asset is deleted, the component is removed automatically. Container images have deletion rules of their own — see Docker (OCI).

Uploading

The “Upload” section uploads files to hosted repositories of the Maven, npm, RubyGems, PyPI, and raw formats. The form adapts to the format of the selected repository:

- **Maven** — coordinates (`groupId` , `artifactId` , `version`), files with the `classifier` and `extension` fields, the `generate-pom` flag;
- **npm** and **RubyGems** — only the package file (`.tgz` or `.gem`): the name and version are read from its metadata;
- **PyPI** — distribution files (a wheel or an sdist); the name and version may be omitted — they are read from the metadata;
- **raw** — the destination directory (`directory`), the file name (`filename`), and the file itself.

Uploading requires the `add` privilege on the repository. Repositories of the Go, Docker (OCI), and Cargo formats accept publications only from the standard clients (see Formats).

Profile

The “Profile” page opens from the avatar menu and contains the following sections:

- **Roles** — the roles assigned to you;
- **Account** — account details; for local users they are editable with the “Edit” button;
- **Password** — changing the password with the “Change password” button (the current password is required);
- **User token** — the token for clients and CI (see Credentials);
- **Sessions** — active sessions: the current one can be ended (“Logout”), any of the others can be revoked (“Revoke”), and all but the current one — at once (“Log out other sessions”).

The “Settings” section in the header is meant for administration — it is described in the Administration guide.

Credentials

Galleon authentication uses a password and a user token; the access model is described in Access control.

Password

The password is used to sign in to the web interface. It can be changed in the profile: the avatar menu → “Profile” → “Change password” (the current password is required). If the administrator has required a password change, all other actions in the web interface are blocked until it is completed.

User token

In command-line clients and CI, use the token instead of the password: it is supplied as a regular username/password pair, and if compromised, it can be revoked without changing the account password.

To generate a token, follow these steps:

1. Open “Profile”, the “User token” section.
2. Click “Generate”.
3. Save the displayed values: use “Name code” as the username and “Pass code” as the password.

> ⚠ The pass code is shown only once — right after generation. If the value is lost, generate the token anew.

Each user can have one active token:

- “Regenerate” issues a new token in place of the old one — the old credentials stop working immediately;
- “Delete” revokes the token without issuing a new one.

Using the credentials in clients

The token (or the password) is supplied wherever a format client expects a username and password: in `docker login`, Maven `settings.xml`, `.npmrc`, `~/.pypirc`, `~/.netrc`, and others. Ready-made examples for every client are collected in the Formats section.

RubyGems (the `~/.gem/credentials` file) and Cargo (the `cargo login` command) take the credentials as a ready authorization header string `Basic <BASE64>`, where `<BASE64>` is `<USERNAME>:<PASSWORD>` encoded in base64. See the RubyGems and Cargo pages for details.

If the client gets an error

A `401` answer means the credentials were not accepted: most often the token was regenerated with “Regenerate” or removed with “Delete”, and the old values stopped working. A `403` answer means a privilege for the operation is missing — ask your administrator for the rights. A `404` answer comes both for a missing artifact and for missing access rights (Access control), and for proxy repositories — from the negative cache as well (Proxying and caching).

Formats

This section collects the instructions for every format: connecting the client to a proxy repository, publishing to a hosted repository, and format specifics.

In the example commands, substitute your own values:

- `<GALLEON_HOST>` — the address of your Galleon instance;
- `<USERNAME>` and `<PASSWORD>` — the account username and password, or the “Name code” and “Pass code” of a user token (Credentials);
- `<REPOSITORY>` — the repository name. The names in the examples are conventional; check the actual ones in the “Browse” section or with your administrator.

Common rules for all formats:

- the repository address is `https://<GALLEON_HOST>/repository/<REPOSITORY>/`; the exception is Docker (OCI), covered in Repositories and formats;
- if the administrator has enabled anonymous access to a repository, the credentials in read commands can be omitted (`<USERNAME>:<PASSWORD>@` in the address, `docker login`, and the like);
- why a package just released in an external registry may not be visible immediately is explained in Proxying and caching.

Docker (OCI)

Docker (OCI) repositories store container images and OCI artifacts, such as Helm charts. A hosted repository serves as a private registry, a proxy repository caches an external registry. The `docker-hosted` and `docker-hub-proxy` (Docker Hub cache) repositories are part of the starter set.

Docker (OCI) is the only format addressed without the `/repository/` prefix:

```
<GALLEON_HOST>/<REPOSITORY>/<IMAGE>:<TAG>
```

Signing in to the registry

Sign in with your username and password or a user token:

```
docker login <GALLEON_HOST>
```

Compatible tools sign in the same way: `crane auth login`, `skopeo login`, `helm registry login`, `oras login`.

Pulling

Through a proxy repository, external registry images are pulled with the common pattern — the image path follows the repository name (official Docker Hub images need the `library/` prefix):

```
docker pull <GALLEON_HOST>/docker-hub-proxy/library/alpine:3.20
```

Pulling from a hosted repository:

```
docker pull <GALLEON_HOST>/docker-hosted/alpine:1.0
helm pull oci://<GALLEON_HOST>/docker-hosted/charts/mychart --version 0.1.0
```

Publishing

To publish an image, tag it with the repository address and push:

```
docker tag alpine:3.20 <GALLEON_HOST>/docker-hosted/alpine:1.0
docker push <GALLEON_HOST>/docker-hosted/alpine:1.0
```

Helm charts are published as OCI artifacts — the chart version becomes the tag:

```
helm push mychart-0.1.0.tgz oci://<GALLEON_HOST>/docker-hosted/charts
```

Publishing requires both privileges on the repository — `read` and `add`: when pushing an image the client asks the registry for read and write access at once, so the `add` privilege alone is not enough.

Deleting

Images are deleted in the “Browse” section of the web interface (the `delete` privilege is required):

- deleting a tag removes only that tag — the image itself stays available by digest;
- deleting a manifest by digest is refused while a tag or a multi-platform index still points at it: remove the referencing tags or the index first;
- layers and configs (the `blobs/` branch) cannot be deleted individually: the content is shared across all the images of the repository;
- with the `DENY` write policy, deletion is refused even with the `delete` privilege — the policy is changed by an administrator.

The disk space is returned by the garbage collection, so it is not freed right after the deletion (Storage and garbage collection).

Specifics

Format specifics:

- Re-pushing an existing tag overwrites it — tags are mutable; the immutable reference to an image is its digest. Overwriting obeys the repository write policy: with `ALLOW_ONCE`, moving an existing tag to another image is refused, and with `DENY` any publishing is refused.
- Multi-platform images and OCI artifacts are supported: Helm charts, `oras` artifacts.
- Images pulled through a proxy repository are cached in full — the manifest and all layers; the general cache behavior is described in Proxying and caching.
- The list of allowed OCI artifact media types is fixed: publishing an artifact with another media type is rejected.
- Publishing through the web interface is not available — use the clients listed above.

Maven

The starter set includes the `maven-central` proxy repository (a Maven Central mirror) and the `maven-releases` (releases) and `maven-snapshots` (snapshot versions) hosted repositories.

Connecting to the proxy

Point Maven at the mirror in `~/.m2/settings.xml`:

```
<settings>
  <mirrors>
    <mirror>
      <id>galleon-central</id>
      <url>https://<GALLEON_HOST>/repository/maven-central</url>
      <mirrorOf>central</mirrorOf>
    </mirror>
  </mirrors>
</settings>
```

If anonymous access to the repository is disabled, add a `<server>` block with the same `id` as the mirror and your credentials — following the example in the Publishing section.

Publishing

Specify the target repository in `pom.xml` and the credentials in `settings.xml` — the `id` values in both blocks must match:

```
<!-- pom.xml -->
<distributionManagement>
  <repository>
    <id>galleon-releases</id>
    <url>https://<GALLEON_HOST>/repository/maven-releases</url>
  </repository>
</distributionManagement>
```

```
<!-- ~/.m2/settings.xml -->
<settings>
  <servers>
    <server>
      <id>galleon-releases</id>
      <username><USERNAME></username>
      <password><PASSWORD></password>
    </server>
  </servers>
</settings>
```

```
mvn deploy
```

Publishing from Gradle:

```
publishing {
  repositories {
    maven {
      url = uri("https://<GALLEON_HOST>/repository/maven-releases")
      credentials {
        username = "<USERNAME>"
        password = "<PASSWORD>"
      }
    }
  }
}
```

Consuming published artifacts

The mirror covers Maven Central only, so hosted repositories are added to the build with separate entries — in `pom.xml` or in a `settings.xml` profile:

```
<repositories>
  <repository>
    <id>galleon-releases</id>
    <url>https://<GALLEON_HOST>/repository/maven-releases</url>
  </repository>
  <repository>
    <id>galleon-snapshots</id>
    <url>https://<GALLEON_HOST>/repository/maven-snapshots</url>
    <snapshots>
      <enabled>true</enabled>
    </snapshots>
  </repository>
</repositories>
```

For Gradle:

```
repositories {
  maven { url = uri("https://<GALLEON_HOST>/repository/maven-releases") }
  maven { url = uri("https://<GALLEON_HOST>/repository/maven-snapshots") }
}
```

The credentials are the same `<server>` (Maven) or `credentials` (Gradle) blocks as for publishing.

Releases and snapshot versions

Re-publishing an existing release version is rejected with `409`: releases are immutable.

Every upload of a snapshot version gets a timestamp (`YYYYMMDD.HHMMSS-N`) and lands in `maven-metadata.xml`; requesting the `X-SNAPSHOT` version returns the latest build.

Specifics

Format specifics:

- `maven-metadata.xml` is generated by the server automatically — do not publish it.
- A hosted repository serves only the checksum files uploaded by the client: Maven publishes `.sha1` and `.md5` by default. To publish all four, deploy with `-Daether.checksums.algorithms=SHA-1,MD5,SHA-256,SHA-512`.
- Classifiers (`sources`, `javadoc`) work as usual.

Go

The starter set includes the `go-proxy` proxy repository (a proxy.golang.org mirror), the `go-hosted` hosted repository (private modules), and `go-sumdb` — a proxy repository of the sum.golang.org checksum database.

Connecting to the proxy

Set the environment variables (or store them with `go env -w`):

```
export GOPROXY="https://<GALLEON_HOST>/repository/go-proxy"
export GOSUMDB="sum.golang.org https://<GALLEON_HOST>/repository/go-sumdb"
go get rsc.io/quote
```

`GOSUMDB` takes two space-separated values — the database name and the proxy repository address: the `go` client will verify checksums through Galleon without reaching the internet directly.

Private modules

Private modules are published to `go-hosted`, while public ones keep coming through `go-proxy` — list both sources in `GOPROXY` (on a `404` response the client moves on to the next address):

```
export GOPROXY="https://<GALLEON_HOST>/repository/go-hosted,https://<GALLEON_HOST>/
repository/go-proxy"
export GOPRIVATE="example.com/private-*"
export GONOPROXY="none"
go mod download example.com/private-x/lib@v1.0.0
```

Private modules are absent from the public checksum database, so `GOPRIVATE` disables checksum verification for their paths.

> ⚠ `GONOPROXY="none"` is mandatory: without it, `GOPRIVATE` routes private module requests directly to the version control system, bypassing Galleon.

The `go` client reads credentials from the `~/.netrc` file:

```
machine <GALLEON_HOST>
login <USERNAME>
password <PASSWORD>
```

Publishing

Go has no standard publish command — module files are uploaded to the module proxy protocol paths:

```
curl -u <USERNAME>:<PASSWORD> -T v1.0.0.mod https://<GALLEON_HOST>/repository/go-hosted/example.com/foo/@v/v1.0.0.mod
curl -u <USERNAME>:<PASSWORD> -T v1.0.0.zip https://<GALLEON_HOST>/repository/go-hosted/example.com/foo/@v/v1.0.0.zip
```

The `.mod` file must match the `go.mod` inside the `.zip` archive byte for byte. Re-uploading an already published version is rejected with `409` — module versions are immutable; publish a new version for a new build.

Specifics

Format specifics:

- Version files (`.zip` , `.mod` , `.info`) are immutable by the Go protocol rules; how proxy repositories cache content and refresh version lists is described in Proxying and caching.
- If `GOSUMDB` is not pointed at `go-sumdb` , the client verifies checksums directly against sum.golang.org — in an isolated network, configure both variables.

npm

npm repositories work with the standard `npm` , `yarn` , and `pnpm` clients. The examples use the conventional names `npm-proxy` and `npm-hosted` .

Connecting to the proxy

Set the registry in `.npmrc` :

```
registry=https://<GALLEON_HOST>/repository/npm-proxy/
```

```
npm install lodash
```

Private packages from the hosted repository are wired by binding a scope to the registry — public packages keep coming through the proxy:

```
@myscope:registry=https://<GALLEON_HOST>/repository/npm-hosted/
```

If reading requires authentication, add the credential lines for the repository being read as well — following the example in the Publishing section, with the corresponding address.

Publishing

Add the hosted repository credentials to `.npmrc` — a username and password or the “Name code” and “Pass code” of a token (the password is base64-encoded):

```
//<GALLEON_HOST>/repository/npm-hosted/:username=<USERNAME>
//<GALLEON_HOST>/repository/npm-hosted/:_password=<base64(<PASSWORD>)>
//<GALLEON_HOST>/repository/npm-hosted/:email=you@example.com
//<GALLEON_HOST>/repository/npm-hosted/:always-auth=true
```

```
npm publish --registry https://<GALLEON_HOST>/repository/npm-hosted/
```

Published versions are immutable: re-publishing the same version is rejected.

Specifics

Format specifics:

- Distribution tags (`npm dist-tag add/rm/ls`) are supported, but the `latest` tag cannot be moved manually — it is updated only by publishing.
- `npm deprecate` works, including for already published versions.
- Scoped packages (`@scope/name`) work in all clients.
- Lock files record Galleon addresses — external registry links do not leak out.

PyPI

PyPI repositories work with the `pip` , `uv` , `poetry` , and `twine` clients. The examples use the conventional names `pypi-proxy` and `pypi-hosted` .

Connecting to the proxy

Point the client at the repository index — the path ends with `/simple/` :

```
pip install --index-url https://<USERNAME>:<PASSWORD>@<GALLEON_HOST>/repository/pypi-proxy/simple/ six==1.16.0
```

For `uv`, add the index to `pyproject.toml` (credentials go in the `UV_INDEX_GALLEON_USERNAME` and `UV_INDEX_GALLEON_PASSWORD` variables):

```
[[tool.uv.index]]
name = "galleon"
url = "https://<GALLEON_HOST>/repository/pypi-proxy/simple/"
default = true
```

For `poetry` (credentials — `POETRY_HTTP_BASIC_GALLEON_USERNAME` and `POETRY_HTTP_BASIC_GALLEON_PASSWORD`):

```
poetry source add --priority=primary galleon https://<GALLEON_HOST>/repository/pypi-proxy/simple/
```

Private packages

Packages from the hosted repository are wired as an additional index next to the proxy.

For `pip` — an extra index:

```
pip install --index-url https://<GALLEON_HOST>/repository/pypi-proxy/simple/ \
  --extra-index-url https://<GALLEON_HOST>/repository/pypi-hosted/simple/ mypkg
```

> ⚠ With several indexes, pip may pick a package from any of them — private package names must be unique relative to the public PyPI (dependency confusion protection).

For `uv` — a second `[[tool.uv.index]]` block with the `pypi-hosted` address and pinning the private package to it via `[tool.uv.sources]`:

```
[[tool.uv.index]]
name = "galleon-private"
url = "https://<GALLEON_HOST>/repository/pypi-hosted/simple/"

[tool.uv.sources]
mypkg = { index = "galleon-private" }
```

Publishing

`twine`, `uv publish`, and `poetry publish` are supported. Authentication is username and password only (or the “Name code” and “Pass code” of a token); the single-secret `__token__` format is not supported.

Publishing with `twine` (`~/.pypirc`):

```
[distutils]
index-servers = galleon

[galleon]
repository = https://<GALLEON_HOST>/repository/pypi-hosted/
username = <USERNAME>
password = <PASSWORD>
```

```
python -m build
twine upload --repository galleon dist/*
```

Publishing with `uv`:

```
UV_PUBLISH_USERNAME=<USERNAME> UV_PUBLISH_PASSWORD=<PASSWORD> \
  uv publish --publish-url https://<GALLEON_HOST>/repository/pypi-hosted/ dist/*
```

Specifics

Format specifics:

- Published versions are immutable: re-uploading a file is rejected with “File already exists”. For idempotent CI publishing use `uv publish --check-url` or `poetry publish --skip-existing` — they succeed on byte-identical already-uploaded files.
- Project names are normalized per the PyPI rules: any spelling (`Foo_Bar`, `foo.bar`) leads to the single canonical page `foo-bar`.
- Wheel metadata is served as separate files (PEP 658) — `uv lock` gathers dependency information without downloading the packages themselves.

RubyGems

RubyGems repositories work with the `gem` and `bundle` clients. The examples use the conventional names `rubygems-proxy` and `rubygems-hosted`.

Connecting to the proxy

Point Bundler at the proxy repository with a mirror:

```
bundle config mirror.https://rubygems.org https://<GALLEON_HOST>/repository/rubygems-
proxy
```

Installing a single gem — the flag order matters, `--clear-sources` must precede `--source`:

```
gem install colorize --clear-sources \
  --source https://<USERNAME>:<PASSWORD>@<GALLEON_HOST>/repository/rubygems-proxy
```

Installing from the hosted repository

Gems published to the hosted repository are wired as a source in the `Gemfile`:

```
source "https://<GALLEON_HOST>/repository/rubygems-hosted" do
  gem "foo"
end
```

Bundler credentials are set with a command (it works for the mirror from the previous section as well):

```
bundle config set --global <GALLEON_HOST> <USERNAME>:<PASSWORD>
```

Publishing

The `gem` client sends the authorization key value verbatim, so `~/.gem/credentials` stores a ready `Basic <BASE64>` string (the file must have `0600` permissions):

```
---
:galleon: Basic <BASE64>
```

```
gem build foo.gemspec
gem push foo-1.0.0.gem --key galleon --host https://<GALLEON_HOST>/repository/
rubygems-hosted
```

Specifics

Format specifics:

- Published versions are immutable: re-publishing the same version is rejected.
- The `gem yank` command is not supported — a version is deleted through the web interface with the “Delete component” button (the `delete` privilege is required).
- Dependency resolution uses the Compact Index protocol; the classic commands (`gem fetch`, `gem list --remote`) work as well.

Cargo

Cargo repositories use the sparse index (Cargo 1.70 and newer). The examples use the conventional names `cargo-hosted` and `cargo-proxy`.

Connecting

Describe the registry in `~/.cargo/config.toml`; for repositories that require authentication, list the credential provider explicitly:

```
[registry]
global-credential-providers = ["cargo:token"]

[registries.galleon]
index = "sparse+https://<GALLEON_HOST>/repository/cargo-hosted/"
```

The token is a ready `Basic <BASE64>` string; store it with `cargo login`:

```
printf 'Basic <BASE64>\n' | cargo login --registry galleon
```

Proxy for crates.io

To route all public dependencies through Galleon, configure source replacement — the registry is also declared under `[registries]` so a token can be attached to it:

```
[registries.galleon-proxy]
index = "sparse+https://<GALLEON_HOST>/repository/cargo-proxy/"

[source.crates-io]
replace-with = "galleon-proxy"

[source.galleon-proxy]
registry = "sparse+https://<GALLEON_HOST>/repository/cargo-proxy/"
```

If the proxy repository requires authentication, store a token for it as well: `cargo login --registry galleon-proxy` (the same `Basic <BASE64>` string).

Usage

A dependency from the hosted registry is declared in `Cargo.toml` with the registry name:

```
mycrate = { version = "1", registry = "galleon" }
```

Publishing

A crate is sent to the hosted registry under the name declared in `registries.galleon`:

```
cargo publish --registry galleon
```

Specifics

Format specifics:

- Re-publishing an existing version is rejected with `409`; names are compared case-insensitively, with `-` and `_` treated as the same character.

- `cargo yank` is supported: the version is flagged in the index, but the file is not deleted — builds with the exact version in `Cargo.lock` keep working. The `edit` privilege is required.
- `cargo search` and `cargo owner` are not supported.

raw

raw repositories store arbitrary files: release archives, models, documents. The examples use the conventional name `raw-hosted`.

Publishing

Upload a file with a `PUT` request to the desired path:

```
curl -u <USERNAME>:<PASSWORD> -T release.tar.gz \
  https://<GALLEON_HOST>/repository/raw-hosted/releases/v1.0.0/release.tar.gz
```

The response carries checksum headers (`X-Checksum-Sha256` and others). Uploading is also available through the web interface — the “Upload” section.

Downloading and deleting

Downloading and deleting a file:

```
curl -u <USERNAME>:<PASSWORD> -O https://<GALLEON_HOST>/repository/raw-hosted/
releases/v1.0.0/release.tar.gz
curl -u <USERNAME>:<PASSWORD> -X DELETE https://<GALLEON_HOST>/repository/raw-hosted/
releases/v1.0.0/release.tar.gz
```

Checksums are returned in the `X-Checksum-*` headers on download and upload. A separate checksum file (for example, `file.tar.gz.sha256`) is available only if it was uploaded as a standalone file.

Proxy repositories

A raw repository can also be a caching mirror of an external HTTP source: files are downloaded with the same `GET` paths, while uploading and deleting are unavailable. The starter set example is `go-sumdb`, the proxy repository of the Go checksum database.

Specifics

Format specifics:

- How the browser handles the files (display or save) is set by the administrator in the repository settings. In repositories with the `inline` display mode, publishing HTML, SVG, and JavaScript files is rejected for security reasons.
- If the repository write policy forbids overwriting, re-uploading to an existing path is rejected with `409`.

- Constructs like `..`, control characters, and a number of special characters are rejected in paths.