Red Hat Quay 3.6

Configure Red Hat Quay

Customizing Red Hat Quay using configuration options

Last Updated: 2022-07-05

Red Hat Quay 3.6 Configure Red Hat Quay

Customizing Red Hat Quay using configuration options

Legal Notice

Abstract

Configure Red Hat Quay

Table of Contents

CHAPTER 1. GETTING STARTED WITH CONFIGURATION

Red Hat Quay can be deployed in a standalone manner, or on an existing OpenShift cluster using the Operator. The methods you use to create, retrieve, update and validate the Red Hat Quay configuration vary slightly, depending on the type of deployment you are using. However, the core configuration options are fundamentally the same for all types of deployment, and these options can be manipulated:

- Directly, by editing the **config.yaml** file. See the section Editing the configuration file.
- Programmatically, using the configuration API. See the section Using the configuration API.
- Visually, using the configuration tool UI. See the section Using the configuration tool.

You can install Quay on OpenShift using the Operator, without the need to supply any initial configuration, as the Operator will supply sensible defaults to deploy the registry. For a standalone deployment, however, you must supply a minimal level of configuration before the registry can be started. The minimal requirements can be determined using the configuration API and are documented in the section

Once you have Quay deployed with your initial configuration, you should retrieve and save the full configuration from the running system as it may contain extra, generated values that you will need in future when restarting or upgrading your system.

1.1. CONFIGURATION UPDATES FOR QUAY 3.6

1.1.1. New configuration fields

- **FEATURE_EXTENDED_REPOSITORY_NAMES:** Support for nested repositories and extended repository names has been added. This change allows the use of / in repository names needed for certain OpenShift Container Platform use cases. For more information, see Configuring nested repositories
- **FEATURE_USER_INITIALIZE:** If set to true, the first User account may be created via API **/api/v1/user/initialize**. For more information, see Pre-configuring Quay for automation
- **ALLOWED_OCI_ARTIFACT_TYPES:** Helm, cosign, and ztsd compression scheme artifacts are built into Red Hat Quay 3.6 by default. For any other OCI media type that is not supported by default, you can add them to the **ALLOWED_OCI_ARTIFACT_TYPES** configuration in Quay's **config.yaml** For more information, see Adding other OCI media types to Quay
- **CREATE_PRIVATE_REPO_ON_PUSH:** Registry users now have the option to set **CREATE_PRIVATE_REPO_ON_PUSH** in their config.yaml to **True** or **False** depending on their security needs.
- **CREATE_NAMESPACE_ON_PUSH:** Pushing to a non-existent organization can now be configured to automatically create the organization.

1.1.2. Deprecated configuration fields

- **FEATURE_HELM_OCI_SUPPORT**: This option has been deprecated and will be removed in a future version of Red Hat Quay. In Red Hat Quay 3.6, Helm artifacts are supported by default and included under the **FEATURE_GENERAL_OCI_SUPPORT** property. Users are no longer required to update their config.yaml files to enable support.

1.2. EDITING THE CONFIGURATION FILE

Deploying the registry in standalone mode requires a minimal configuration - see section Section 1.5, "Minimal configuration".

The configuration file is validated on startup of the registry, and any issue will be highlighted in the output:

It is possible to use the configuration API to validate the configuration, but this requires starting the Quay container in config mode

For changes to take effect, the registry needs to be restarted.

1.3. LOCATION OF CONFIGURATION FILE IN A STANDALONE DEPLOYMENT

For a standalone deployment, the **config.yaml** file must be specified when starting the Quay registry. This file is located in the config volume, so in the following example, the config file is located at **$QUAY/config/config.yaml**:

```
$ sudo podman run -d --rm -p 80:8080 -p 443:8443 \
  --name=quay \
  -v $QUAY/config:/conf/stack:Z \
  -v $QUAY/storage:/datastorage:Z \
  registry.redhat.io/quay/quay-rhel8:v3.6.8
```

1.4. CONFIGURING QUAY ON OPENSHIFT USING THE COMMAND LINE AND API

Once deployed, you can configure the Quay application by editing the Quay configuration bundle secret **spec.configBundleSecret** and you can also change the managed status of components in the **spec.components** object of the QuayRegistry resource

The Operator does not watch the **spec.configBundleSecret** resource for changes, so it is recommended that configuration changes be made to a new **Secret** resource and that the **spec.configBundleSecret** field is updated to reflect the change. In the event there are issues with the new configuration, it is simple to revert the value of **spec.configBundleSecret** to the older **Secret**.

The procedure for changing the configuration involves:

1. Determining the current endpoints and secrets
2. Downloading the existing configuration bundle, if Red Hat Quay has already been deployed on OpenShift
3. Creating or updating the **config.yaml** configuration file
4. Assembling any SSL certs required for Quay, or custom SSL certs needed for services
5. Creating a new config bundle secret, using the config file and any certs
6. Creating or updating the registry, referencing the new config bundle secret and specifying any over-rides for managing components

7. Monitoring the deployment to ensure successful completion and that the configuration changes have taken effect

Alternatively, you can use the config editor UI to configure the Quay application, as described in the section Chapter 4, *Using the config tool to reconfigure Quay on OpenShift* .

1.4.1. Determining QuayRegistry endpoints and secrets

You can examine the QuayRegistry resource, using **oc describe quayregistry** or **oc get quayregistry -o yaml**, to determine the current endpoints and secrets:

```
$ oc get quayregistry example-registry -n quay-enterprise -o yaml

apiVersion: quay.redhat.com/v1
kind: QuayRegistry
metadata:
  ...
  name: example-registry
  namespace: quay-enterprise
  ...
spec:
  components:
  ...
  configBundleSecret: example-registry-quay-config-bundle-fjpnm
status:
  configEditorCredentialsSecret: example-registry-quay-config-editor-credentials-kk55dc7299
  configEditorEndpoint: https://example-registry-quay-config-editor-quay-
enterprise.apps.docs.quayteam.org
  currentVersion: 3.6.0
  lastUpdated: 2021-09-21 11:18:13.285192787 +0000 UTC
  registryEndpoint: https://example-registry-quay-quay-enterprise.apps.docs.quayteam.org
  unhealthyComponents: {}
```

The relevant fields are:

- **registryEndpoint**: The URL for your registry, for browser access to the registry UI, and for the registry API endpoint
- **configBundleSecret**: The config bundle secret, containing the **config.yaml** file and any SSL certs
- **configEditorEndpoint**: The URL for the config editor tool, for browser access to the config tool, and for the configuration API
- **configEditorCredentialsSecret**: The secret containing the username (typically **quayconfig**) and the password for the config editor tool

To determine the username and password for the config editor tool:

1. Retrieve the secret:

```
$ oc get secret -n quay-enterprise example-registry-quay-config-editor-credentials-
kk55dc7299 -o yaml

apiVersion: v1
data:
```

```
  password: SkZwQkVKTUN0a1BUZmp4dA==
  username: cXVheWNvbmZpZw==
kind: Secret
```

2. Decode the username:

```
$ echo 'cXVheWNvbmZpZw==' | base64 --decode

quayconfig
```

3. Decode the password:

```
$ echo 'SkZwQkVKTUN0a1BUZmp4dA==' | base64 --decode

JFpBEJMCtkPTfjxt
```

1.4.2. Downloading the existing configuration

There are a number of methods for accessing the current configuration:

1. Using the config editor endpoint, specifying the username and password for the config editor:

```
$ curl -k -u quayconfig:JFpBEJMCtkPTfjxt https://example-registry-quay-config-editor-quay-enterprise.apps.docs.quayteam.org/api/v1/config
```


```
{
   "config.yaml": {
      "ALLOW_PULLS_WITHOUT_STRICT_LOGGING": false,
      "AUTHENTICATION_TYPE": "Database",
      ...
      "USER_RECOVERY_TOKEN_LIFETIME": "30m"
   },
   "certs": {
      "extra_ca_certs/service-ca.crt":
"LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSURVVENDQWptZ0F3SUJBZ0lJRE9k
WFhuUXFjMUF3RFFZSktvWklodmNOQVFFTEJRQXdOakUwTURJR0ExVUUKQXd3cmIzQ
mxibk5vYVdaMExYTmxjblpwWTJVdGMyVnlkbWx1WnkxemFXZHVaWEpBTVRZek1UYzNPRE
V3TXpBZQpGdzB5TVRBNU1UWXdOelF4TkRKYUZ..."
   }
}
```


2. Using the config bundle secret

   a. Get the secret data:

```
$ oc get secret -n quay-enterprise example-registry-quay-config-bundle-jkfhs -o jsonpath='{.data}'
```


```
{
   "config.yaml":
"QUxMT1dfUFVMTFNfV0lUSE9VVF9TVFJJQ1RfTE9HR0lORzogZmFsc2UKQVVUSEVO
VElDQVRJT05fVFlQRTogRGF0YWJhc2UKQVZBVEFSX0tJTkQ6IGxvY2FsCkRBVEFCQ
VNFX1NFQ1JFVF9LRVk6IHhlOEc1VDBNbklaGxNQzNkTjd3MWR5WWxwVmo0a0R2enl
```

```
xZ3l6Ulp5ZjFpODBmWWU3VDUxU1FPZ3hXelpocFlqYlVxNzRKaDllVVVEVWpyCkRFR
...
OgotIDJ3ClRFQU1fUkVTWU5DX1NUQUxFX1RJTUU6IDYwbQpURVNUSU5HOiBmYWx
zZQpVU0VSX1JFQ09WRVJZX1RPS0VOX0xJRkVUSU1FOiAzMG0K",
  "extra_ca_cert_service-ca.crt":
"LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSURVVENDQWptZ0F3SUJBZ0lJR
E9kWFhuUXFjMUF3RFFZSktvWklodmNOQVFFTEJRQXdOakUwTURJR0ExVUUKQXd3
cmIzQmxibk5vYVdaMExYTmxjblpwWTJVdGMyVnlkbWx1WnkxemFXZHVaWEpBTVRZek1
UYzNPREV3TXpBZQpGdzB5TVRBNU1UWXdOelF4TkRKYUZ3MHI
...
XSW1jaApkQXZTWGpFUnZOZEZzN3pHK1VzTmZwN0ZIQkJVWkY4L2RZNWJCR2Mw
WTVaY0J6bFNjQT09Ci0tLS0tRU5EIENFUlRJRklDQVRFLS0tLS0K"
}
```

b. Decode the data:

```
$ echo 'QUxMT1dfUFVMTFN...U1FOiAzMG0K' | base64 --decode
```

```
ALLOW_PULLS_WITHOUT_STRICT_LOGGING: false
AUTHENTICATION_TYPE: Database
...
TAG_EXPIRATION_OPTIONS:
- 2w
TEAM_RESYNC_STALE_TIME: 60m
TESTING: false
USER_RECOVERY_TOKEN_LIFETIME: 30m
```

1.4.3. Using the config bundle to configure custom SSL certs

You can configure custom SSL certs either before initial deployment or after Red Hat Quay is deployed on OpenShift, by creating a new config bundle secret. If you are adding the cert(s) to an existing deployment, you must include the complete existing **config.yaml** in the new config bundle secret, even if you are not making any configuration changes.

1. Create the secret using embedded data or using files:

   a. Embed the configuration details directly in the Secret resource YAML file, for example:

   custom-ssl-config-bundle.yaml

```
apiVersion: v1
kind: Secret
metadata:
  name: custom-ssl-config-bundle-secret
  namespace: quay-enterprise
data:
  config.yaml: |
    ALLOW_PULLS_WITHOUT_STRICT_LOGGING: false
    AUTHENTICATION_TYPE: Database
    ...
  extra_ca_cert_my-custom-ssl.crt: |
    -----BEGIN CERTIFICATE-----
    MIIDsDCCApigAwIBAgIUCqlzkHjF5i5TXLFy+sepFrZr/UswDQYJKoZIhvcNAQEL

BQAwbzELMAkGA1UEBhMCSUUxDzANBgNVBAgMBkdBTFdBWTEPMA0GA1UEBwwG
```

```
R0FM
    ....
    -----END CERTIFICATE-----
```

Next, create the secret from the YAML file:

```
$ oc create  -f custom-ssl-config-bundle.yaml
```

b. Alternatively, you can create files containing the desired information, and then create the secret from those files:

```
$ oc create secret generic custom-ssl-config-bundle-secret \
  --from-file=config.yaml \
  --from-file=extra_ca_cert_my-custom-ssl.crt=my-custom-ssl.crt
```

2. Create or update the QuayRegistry YAML file **quayregistry.yaml**, referencing the created Secret, for example:

quayregistry.yaml

```
apiVersion: quay.redhat.com/v1
kind: QuayRegistry
metadata:
  name: example-registry
  namespace: quay-enterprise
spec:
  configBundleSecret: custom-ssl-config-bundle-secret
```

3. Deploy or update the registry using the YAML file:

```
oc apply -f quayregistry.yaml
```

1.5. MINIMAL CONFIGURATION

For a standalone deployment, configuration options are required for the following features:

- Server hostname
- HTTP or HTTPS
- Authentication type, for example, Database or LDAP
- Secret keys for encrypting data
- Storage for images
- Database for metadata
- Redis for build logs and user events
- Tag expiration options

1.5.1. Sample minimal configuration file

A sample minimal config file, using local storage for images, is shown below:

$QUAY/config/config.yaml

```
AUTHENTICATION_TYPE: Database
BUILDLOGS_REDIS:
    host: quay-server.example.com
    password: strongpassword
    port: 6379
DATABASE_SECRET_KEY: 0ce4f796-c295-415b-bf9d-b315114704b8
DB_URI: postgresql://quayuser:quaypass@quay-server.example.com:5432/quay
DEFAULT_TAG_EXPIRATION: 2w
DISTRIBUTED_STORAGE_CONFIG:
    default:
        - LocalStorage
        - storage_path: /datastorage/registry
DISTRIBUTED_STORAGE_DEFAULT_LOCATIONS: []
DISTRIBUTED_STORAGE_PREFERENCE:
    - default
PREFERRED_URL_SCHEME: http
SECRET_KEY: e8f9fe68-1f84-48a8-a05f-02d72e6eccba
SERVER_HOSTNAME: quay-server.example.com
SETUP_COMPLETE: true
TAG_EXPIRATION_OPTIONS:
    - 0s
    - 1d
    - 1w
    - 2w
    - 4w
USER_EVENTS_REDIS:
    host: quay-server.example.com
    password: strongpassword
    port: 6379
```

NOTE

The **SETUP_COMPLETE** field indicates that the configuration has been validated. You should use the config editor tool to validate your configuration before starting the registry.

1.5.2. Local storage

Using local storage for images is only recommended when deploying a registry for proof of concept purposes. In this case, storage is specified on the command line when starting the registry, mapping a local directory **$QUAY/storage** to the **/datastorage** path in the container:

```
$ sudo podman run -d --rm -p 80:8080 -p 443:8443 \
  --name=quay \
  -v $QUAY/config:/conf/stack:Z \
  -v $QUAY/storage:/datastorage:Z \
  registry.redhat.io/quay/quay-rhel8:v3.6.8
```

1.5.3. Cloud storage

Storage configuration is detailed in the section Image storage. It is useful to compare the difference when using cloud storage, for example, on Google Cloud Platform:

$QUAY/config/config.yaml

```
DISTRIBUTED_STORAGE_CONFIG:
    default:
        - GoogleCloudStorage
        - access_key: GOOGQIMFB3ABCDEFGHIJKLMN
          bucket_name: quay_bucket
          secret_key: FhDAYe2HeuAKfvZCAGyOioNaaRABCDEFGHIJKLMN
          storage_path: /datastorage/registry
DISTRIBUTED_STORAGE_DEFAULT_LOCATIONS: []
DISTRIBUTED_STORAGE_PREFERENCE:
    - default
```

When starting the registry using cloud storage, no configuration is required on the command line:

```
$ sudo podman run -d --rm -p 80:8080 -p 443:8443 \
   --name=quay \
   -v $QUAY/config:/conf/stack:Z \
   registry.redhat.io/quay/quay-rhel8:v3.6.8
```

CHAPTER 2. CONFIGURATION FIELDS

2.1. REQUIRED CONFIGURATION FIELDS

Required fields are covered in the following sections:

- General required fields
- Storage for images
- Database for metadata
- Redis for build logs and user events
- Tag expiration options

2.2. AUTOMATION OPTIONS

- Pre-configuring Quay for automation
- Using the API to create the first user

2.3. OPTIONAL CONFIGURATION FIELDS

Optional fields are covered in the following sections:

- Basic configuration
- SSL
- LDAP
- Repository mirroring
- Security scanner
- OCI and Helm
- Action log
- Build logs
- Dockerfile build
- OAuth
- Configuring nested repositories
- Adding other OCI media types to Quay
- Mail
- User
- Recaptcha

- ACI
- JWT
- App tokens
- Miscellaneous
- Legacy options

2.4. GENERAL REQUIRED FIELDS

Table 2.1. General required fields

Field	Type	Description
AUTHENTICATION_TYPE (Required)	String	The authentication engine to use for credential authentication **Values:** One of **Database**, **LDAP**, **JWT**, **Keystone**, **OIDC** **Default:** **Database**
PREFERRED_URL_SCHEME (Required)	String	The URL scheme to use when accessing Red Hat Quay **Values:** One of **http**, **https** **Default:** **http**
SERVER_HOSTNAME (Required)	String	The URL at which Red Hat Quay is accessible, without the scheme **Example:** **quay-server.example.com**
DATABASE_SECRET_KEY (Required)	String	Key used to encrypt sensitive fields within the database. This value should never be changed once set, otherwise all reliant fields, for example, repository mirror username and password configurations, are invalidated.

Field	Type	Description
SECRET_KEY (Required)	String	Key used to encrypt sensitive fields within the database and at run time. his value should never be changed once set, otherwise all reliant fields, for example, encrypted password credentials, are invalidated.
SETUP_COMPLETE (Required)	Boolean	This is an artefact left over from earlier versions of the software and currently it **must** be specified with a value of **true**.

2.5. DATABASE CONFIGURATION

You configure the connection to the database using the required DB_URI field and optional connection arguments in the DB_CONNECTION_ARGS structure. Some key-value pairs defined under DB_CONNECTION_ARGS are generic while others are database-specific. In particular, SSL configuration depends on the database you are deploying, and examples for PostgreSQL and MySQL are given below.

2.5.1. Database URI

Table 2.2. Database URI

Field	Type	Description
DB_URI (Required)	String	The URI for accessing the database, including any credentials

Example:

```
postgresql://quayuser:quaypass@quay-server.example.com:5432/quay
```

2.5.2. Database connection arguments

Table 2.3. Database connection arguments

Field	Type	Description
DB_CONNECTION_ARGS	Object	Optional connection arguments for the database, such as timeouts and SSL

Field	Type	Description
.autorollback	Boolean	Whether to use thread-local connections Should **ALWAYS** be **true**
.threadlocals	Boolean	Whether to use auto-rollback connections Should **ALWAYS** be **true**

2.5.2.1. PostgreSQL SSL connection arguments

A sample PostgreSQL SSL configuration is given below:

```
DB_CONNECTION_ARGS:
  sslmode: verify-ca
  sslrootcert: /path/to/cacert
```

The **sslmode** option determines whether or with what priority a secure SSL TCP/IP connection will be negotiated with the server. There are six modes:

- **disable:** only try a non-SSL connection
- **allow:** first try a non-SSL connection; if that fails, try an SSL connection
- **prefer:** (default) first try an SSL connection; if that fails, try a non-SSL connection
- **require:** only try an SSL connection. If a root CA file is present, verify the certificate in the same way as if verify-ca was specified
- **verify-ca:** only try an SSL connection, and verify that the server certificate is issued by a trusted certificate authority (CA)
- **verify-full:** only try an SSL connection, verify that the server certificate is issued by a trusted CA and that the requested server host name matches that in the certificate

More information on the valid arguments for PostgreSQL is available at https://www.postgresql.org/docs/current/libpq-connect.html.

2.5.2.2. MySQL SSL connection arguments

A sample MySQL SSL configuration follows:

```
DB_CONNECTION_ARGS:
  ssl:
    ca: /path/to/cacert
```

Information on the valid connection arguments for MySQL is available at https://dev.mysql.com/doc/refman/8.0/en/connecting-using-uri-or-key-value-pairs.html.

2.6. IMAGE STORAGE

2.6.1. Image storage features

Table 2.4. Storage config features

Field	Type	Description
FEATURE_REPO_MIRROR	Boolean	If set to true, enables repository mirroring **Default: False**
FEATURE_PROXY_STORAGE	Boolean	Whether to proxy all direct download URLs in storage via the registry nginx **Default: False**
FEATURE_STORAGE_REPLICATION	Boolean	Whether to automatically replicate between storage engines **Default: False**

2.6.2. Image storage configuration fields

You specify a list of all storage engines using the DISTRIBUTED_STORAGE_CONFIG field, and choose you preferred storage engine(s) using the DISTRIBUTED_STORAGE_PREFERENCE field.

The DISTRIBUTED_STORAGE_DEFAULT_LOCATIONS field is used to control which locations will have their images replicated by default.

Table 2.5. Storage config fields

Field	Type	Description
DISTRIBUTED_STORAGE_CONFIG (Required)	Object	Configuration for storage engine(s) to use in Red Hat Quay. Each key represents an unique identifier for a storage engine. The value consists of a tuple of (key, value) forming an object describing the storage engine parameters. **Default: []**

Field	Type	Description
DISTRIBUTED_STORAGE_DEFAULT_LOCATIONS (Required)	Array of string	The list of storage engine(s) (by ID in DISTRIBUTED_STORAGE_CONFIG) whose images should be fully replicated, by default, to all other storage engines.
DISTRIBUTED_STORAGE_PREFERENCE (Required)	Array of string	The preferred storage engine(s) (by ID in DISTRIBUTED_STORAGE_CONFIG) to use. A preferred engine means it is first checked for pulling and images are pushed to it. **Default:** **false**
MAXIMUM_LAYER_SIZE	String	Maximum allowed size of an image layer **Pattern**: **^[0-9]+(G|M)$** **Example**: **100G** **Default:** **20G**

2.6.3. Local storage

```
DISTRIBUTED_STORAGE_CONFIG:
  default:
    - LocalStorage
    - storage_path: /datastorage/registry
DISTRIBUTED_STORAGE_DEFAULT_LOCATIONS: []
DISTRIBUTED_STORAGE_PREFERENCE:
    - default
```

2.6.4. OCS/NooBaa

```
DISTRIBUTED_STORAGE_CONFIG:
  rhocsStorage:
    - RHOCSStorage
    - access_key: access_key_here
      secret_key: secret_key_here
      bucket_name: quay-datastore-9b2108a3-29f5-43f2-a9d5-2872174f9a56
      hostname: s3.openshift-storage.svc.cluster.local
      is_secure: 'true'
      port: '443'
      storage_path: /datastorage/registry
```

2.6.5. Ceph / RadosGW Storage / Hitachi HCP

```
DISTRIBUTED_STORAGE_CONFIG:
  radosGWStorage:
    - RadosGWStorage
    - access_key: access_key_here
      secret_key: secret_key_here
      bucket_name: bucket_name_here
      hostname: hostname_here
      is_secure: 'true'
      port: '443'
      storage_path: /datastorage/registry
DISTRIBUTED_STORAGE_DEFAULT_LOCATIONS: []
DISTRIBUTED_STORAGE_PREFERENCE:
    - default
```

2.6.6. AWS S3 storage

```
DISTRIBUTED_STORAGE_CONFIG:
  s3Storage:
    - S3Storage
    - host: s3.us-east-2.amazonaws.com
      s3_access_key: ABCDEFGHIJKLMN
      s3_secret_key: OL3ABCDEFGHIJKLMN
      s3_bucket: quay_bucket
      storage_path: /datastorage/registry
DISTRIBUTED_STORAGE_DEFAULT_LOCATIONS: []
DISTRIBUTED_STORAGE_PREFERENCE:
    - s3Storage
```

2.6.7. Google cloud storage

```
DISTRIBUTED_STORAGE_CONFIG:
    googleCloudStorage:
        - GoogleCloudStorage
        - access_key: GOOGQIMFB3ABCDEFGHIJKLMN
          bucket_name: quay-bucket
          secret_key: FhDAYe2HeuAKfvZCAGyOioNaaRABCDEFGHIJKLMN
          storage_path: /datastorage/registry
DISTRIBUTED_STORAGE_DEFAULT_LOCATIONS: []
DISTRIBUTED_STORAGE_PREFERENCE:
    - googleCloudStorage
```

2.6.8. Azure storage

```
DISTRIBUTED_STORAGE_CONFIG:
  azureStorage:
    - AzureStorage
    - azure_account_name: azure_account_name_here
      azure_account_key: azure_account_key_here
      azure_container: azure_container_here
      sas_token: some/path/
      storage_path: /datastorage/registry
```

```
DISTRIBUTED_STORAGE_DEFAULT_LOCATIONS: []
DISTRIBUTED_STORAGE_PREFERENCE:
    - azureStorage
```

2.6.9. Swift storage

```
DISTRIBUTED_STORAGE_CONFIG:
  swiftStorage:
    - SwiftStorage
    - swift_user: swift_user_here
      swift_password: swift_password_here
      swift_container: swift_container_here
      auth_url: https://example.org/swift/v1/quay
      auth_version: 1
      ca_cert_path: /conf/stack/swift.cert"
      storage_path: /datastorage/registry
DISTRIBUTED_STORAGE_DEFAULT_LOCATIONS: []
DISTRIBUTED_STORAGE_PREFERENCE:
    - swiftStorage
```

2.7. REDIS CONFIGURATION FIELDS

2.7.1. Build logs

Table 2.6. Build logs configuration

Field	Type	Description
BUILDLOGS_REDIS (Required)	Object	Redis connection details for build logs caching
.host (Required)	String	The hostname at which Redis is accessible **Example:** **quay-server.example.com**
.port (Required)	Number	The port at which Redis is accessible **Example:** **6379**
.password	String	The port at which Redis is accessible **Example:** **strongpassword**

2.7.2. User events

Table 2.7. User events config

Field	Type	Description
USER_EVENTS_REDIS (Required)	Object	Redis connection details for user event handling
.host (Required)	String	The hostname at which Redis is accessible **Example:** **quay-server.example.com**
.port (Required)	Number	The port at which Redis is accessible **Example:** **6379**
.password	String	The port at which Redis is accessible **Example:** **strongpassword**

2.7.3. Example redis configuration

```
BUILDLOGS_REDIS:
    host: quay-server.example.com
    password: strongpassword
    port: 6379

USER_EVENTS_REDIS:
    host: quay-server.example.com
    password: strongpassword
    port: 6379
```

2.8. TAG EXPIRATION OPTIONS

Table 2.8. Tag expiration configuration

Field	Type	Description
FEATURE_GARBAGE_COLLECTION	Boolean	Whether garbage collection of repositories is enabled **Default:** True

Field	Type	Description
TAG_EXPIRATION_OPTIONS (Required)	Array of string	The options that users can select for expiration of tags in their namespace (if enabled) **Pattern:** **^[0-9]+(w|m|d|h|s)$**
DEFAULT_TAG_EXPIRATION (Required)	String	The default, configurable tag expiration time for time machine. **Pattern:** **^[0-9]+(w|m|d|h|s)$** **Default:** **2w**
FEATURE_CHANGE_TAG_EXPIRATION	Boolean	Whether users and organizations are allowed to change the tag expiration for tags in their namespace **Default:** True

Example:

```
DEFAULT_TAG_EXPIRATION: 2w
TAG_EXPIRATION_OPTIONS:
  - 0s
  - 1d
  - 1w
  - 2w
  - 4w
```

2.9. PRE-CONFIGURING QUAY FOR AUTOMATION

Quay has a number of configuration options that support automation. These options can be set before deployment, to minimize the need to interact with the user interface.

2.9.1. Allowing the API to create the first user

Set the config option **FEATURE_USER_INITIALIZE** to **true**, so that you can use the API **/api/v1/user/initialize** to create the first user. This API endpoint does not require authentication, unlike all other registry API calls which require an OAuth token which is generated by an OAuth application in an existing organization.

Once you have deployed Quay, you can use the API to create a user, for example, **quayadmin**, provided no other users have already been created. For more information, see the section on Creating the first user using the API

2.9.2. Enabling general API access

Set the config option **BROWSER_API_CALLS_XHR_ONLY** to **false**, to allow general access to the Quay registry API.

2.9.3. Adding a super user

While you cannot create a user until after deployment, it is convenient to ensure that first user is an administrator with full permissions. It is easier to configure this in advance, using the **SUPER_USER** configuration object.

2.9.4. Restricting user creation

Once you have configured a super user, you can restrict the ability to create new users to the super user group. Set the **FEATURE_USER_CREATION** to **false** to restrict user creation.

2.9.5. Suggested configuration for automation

Create a **config.yaml** configuration file that includes the appropriate settings:

config.yaml

```
...
FEATURE_USER_INITIALIZE: true
BROWSER_API_CALLS_XHR_ONLY: false
SUPER_USERS:
- quayadmin
FEATURE_USER_CREATION: false
...
```

2.9.6. Deploying the Operator using the initial configuration

1. Create a Secret using the configuration file

   ```
   $ oc create secret generic --from-file config.yaml=./config.yaml init-config-bundle-secret
   ```

2. Create a QuayRegistry YAML file **quayregistry.yaml**, identifying the unmanaged components and also referencing the created Secret, for example:

   quayregistry.yaml

   ```
   apiVersion: quay.redhat.com/v1
   kind: QuayRegistry
   metadata:
     name: example-registry
     namespace: quay-enterprise
   spec:
     configBundleSecret: init-config-bundle-secret
   ```

3. Deploy the registry:

   ```
   $ oc create -f quayregistry.yaml
   ```

4. Create the first user, **quayadmin**, using the API

2.9.7. Using the API to create the first user

When using the API to create the first user, the following conditions must be met:

- The config option **FEATURE_USER_INITIALIZE** must be set to **true**
- No users can already exist in the database

For more information on pre-configuring the deployment, see the section Pre-configuring Quay for automation

2.9.7.1. Invoking the API

Using the **status.registryEndpoint** URL, invoke the **/api/v1/user/initialize** API, passing in the username, password and email address. You can also request an OAuth token by specifying **"access_token": true**.

```
$ curl -X POST -k https://example-registry-quay-quay-enterprise.apps.docs.quayteam.org/api/v1/user/initialize --header 'Content-Type: application/json' --data '{ "username": "quayadmin", "password":"quaypass123", "email": "quayadmin@example.com", "access_token": true}'
```

```
{"access_token":"6B4QTRSTSD1HMIG915VPX7BMEZBVB9GPNY2FC2ED", "email":"quayadmin@example.com","encrypted_password":"1nZMLH57RIE5UGdL/yYpDOHLqiNCgimb6W9kfF8MjZ1xrfDpRyRs9NUnUuNuAitW","username":"quayadmin"}
```

If successful, the method returns an object with the username, email and encrypted password. If a user already exists in the database, an error is returned:

```
$ curl -X POST -k https://example-registry-quay-quay-enterprise.apps.docs.quayteam.org/api/v1/user/initialize --header 'Content-Type: application/json' --data '{ "username": "quayuser2", "password":"quaypass123", "email": "quayuser2@example.com"}'
```

```
{"message":"Cannot initialize user in a non-empty database"}
```

The password must be at least 8 characters and contain no whitespace:

```
$ curl -X POST -k https://example-registry-quay-quay-enterprise.apps.docs.quayteam.org/api/v1/user/initialize --header 'Content-Type: application/json' --data '{ "username": "quayadmin", "password":"pass123", "email": "quayadmin@example.com"}'
```

```
{"message":"Failed to initialize user: Invalid password, password must be at least 8 characters and contain no whitespace."}
```

2.9.7.2. Using the OAuth token

You can now invoke the rest of the Quay API specifying the returned OAuth code. For example, to get a list of the current users:

```
$ curl -X GET -k -H "Authorization: Bearer
6B4QTRSTSD1HMIG915VPX7BMEZBVB9GPNY2FC2ED" https://example-registry-quay-quay-
enterprise.apps.docs.quayteam.org/api/v1/superuser/users/
```


```
{
   "users": [
      {
         "kind": "user",
         "name": "quayadmin",
         "username": "quayadmin",
         "email": "quayadmin@example.com",
         "verified": true,
         "avatar": {
            "name": "quayadmin",
            "hash": "3e82e9cbf62d25dec0ed1b4c66ca7c5d47ab9f1f271958298dea856fb26adc4c",
            "color": "#e7ba52",
            "kind": "user"
         },
         "super_user": true,
         "enabled": true
      }
   ]
}
```


In this instance, the details for the **quayadmin** user are returned as it is the only user that has been created so far.

2.9.7.2.1. Create organization

To create an organization, use a POST call to **api/v1/organization/** endpoint:

```
$ curl -X POST -k --header 'Content-Type: application/json' -H "Authorization: Bearer
6B4QTRSTSD1HMIG915VPX7BMEZBVB9GPNY2FC2ED" https://example-registry-quay-quay-
enterprise.apps.docs.quayteam.org/api/v1/organization/ --data '{"name": "testorg", "email":
"testorg@example.com"}'
```

```
"Created"
```

2.9.7.2.2. Get organization details

To retrieve the details of the organization you created:

```
$ curl -X GET -k --header 'Content-Type: application/json' -H "Authorization: Bearer
6B4QTRSTSD1HMIG915VPX7BMEZBVB9GPNY2FC2ED" https://min-registry-quay-quay-
enterprise.apps.docs.quayteam.org/api/v1/organization/testorg
```


```
{
   "name": "testorg",
   "email": "testorg@example.com",
   "avatar": {
      "name": "testorg",
      "hash": "5f113632ad532fc78215c9258a4fb60606d1fa386c91b141116a1317bf9c53c8",
      "color": "#a55194",
```

```
        "kind": "user"
    },
    "is_admin": true,
    "is_member": true,
    "teams": {
        "owners": {
            "name": "owners",
            "description": "",
            "role": "admin",
            "avatar": {
                "name": "owners",
                "hash": "6f0e3a8c0eb46e8834b43b03374ece43a030621d92a7437beb48f871e90f8d90",
                "color": "#c7c7c7",
                "kind": "team"
            },
            "can_view": true,
            "repo_count": 0,
            "member_count": 1,
            "is_synced": false
        }
    },
    "ordered_teams": [
        "owners"
    ],
    "invoice_email": false,
    "invoice_email_address": null,
    "tag_expiration_s": 1209600,
    "is_free_account": true
}
```


2.10. BASIC CONFIGURATION

Table 2.9. Basic configuration

Field	Type	Description
REGISTRY_TITLE	String	If specified, the long-form title for the registry **Default:** **Quay Enterprise**
REGISTRY_TITLE_SHORT	String	If specified, the short-form title for the registry. **Default:** **Quay Enterprise**
BRANDING	Object	Custom branding for logos and URLs in the Red Hat Quay UI.

Field	Type	Description
.logo (Required)	String	Main logo image URL **Example:** **/static/img/quay-horizontal-color.svg**
.footer_img	String	Logo for UI footer **Example:** **/static/img/RedHat.svg**
.footer_url	String	Link for footer image **Example:** **https://redhat.com**
CONTACT_INFO	Array of String	If specified, contact information to display on the contact page. If only a single piece of contact information is specified, the contact footer will link directly.
[0]	String	Adds a link to send an e-mail **Pattern:** **^mailto:(.)+$** **Example:** **mailto:support@quay.io**
[1]	String	Adds a link to visit an IRC chat room **Pattern:** **^irc://(.)+$** **Example:** **irc://chat.freenode.net:6665/quay**
[2]	String	Adds a link to call a phone number+ **Pattern:** **^tel:(.)+$** **Example:** **tel:+1-888-930-3475**

Field	Type	Description
[3]	String	Adds a link to a defined URL **Pattern:** **^http(s)?://(.)+$** **Example:** **https://twitter.com/quayio**

2.11. SSL CONFIGURATION FIELDS

Table 2.10. SSL configuration

Field	Type	Description
PREFERRED_URL_SCHEME	String	One of **http**, **https** **Default: http**
SERVER_HOSTNAME (Required)	String	The URL at which Red Hat Quay is accessible, without the scheme **Example:** **quay-server.example.com**
SSL_CIPHERS	Array of String	If specified, the nginx-defined list of SSL ciphers to enabled and disabled **Example:** [**CAMELLIA**, **!3DES**]
SSL_PROTOCOLS	Array of String	If specified, nginx is configured to enabled a list of SSL protocols defined in the list. Removing an SSL protocol from the list disables the protocol during Red Hat Quay startup. **Example:** **['TLSv1','TLSv1.1','TLSv1.2']**
SESSION_COOKIE_SECURE	Boolean	Whether the **secure** property should be set on session cookies **Default:** False **Recommendation:** Set to True for all installations using SSL

2.11.1. Configuring SSL

1. Copy the certificate file and primary key file to your configuration directory, ensuring they are named **ssl.cert** and **ssl.key** respectively:

```
$ cp ~/ssl.cert $QUAY/config
$ cp ~/ssl.key $QUAY/config
$ cd $QUAY/config
```

2. Edit the **config.yaml** file and specify that you want Quay to handle TLS:

   config.yaml

```
...
SERVER_HOSTNAME: quay-server.example.com
...
PREFERRED_URL_SCHEME: https
...
```

3. Stop the **Quay** container and restart the registry

2.12. ADDING TLS CERTIFICATES TO THE RED HAT QUAY CONTAINER

To add custom TLS certificates to Red Hat Quay, create a new directory named **extra_ca_certs/** beneath the Red Hat Quay config directory. Copy any required site-specific TLS certificates to this new directory.

2.12.1. Add TLS certificates to Red Hat Quay

1. View certificate to be added to the container

```
$ cat storage.crt
-----BEGIN CERTIFICATE-----
MIIDTTCCAjWgAwIBAgIJAMVr9ngjJhzbMA0GCSqGSIb3DQEBCwUAMD0xCzAJBgNV
[...]
-----END CERTIFICATE-----
```

2. Create certs directory and copy certificate there

```
$ mkdir -p quay/config/extra_ca_certs
$ cp storage.crt quay/config/extra_ca_certs/
$ tree quay/config/
├── config.yaml
├── extra_ca_certs
│   ├── storage.crt
```

3. Obtain the **Quay** container's **CONTAINER ID** with **podman ps**:

```
$ sudo podman ps
CONTAINER ID  IMAGE                           COMMAND               CREATED
STATUS            PORTS
5a3e82c4a75f  <registry>/<repo>/quay:v3.6.8 "/sbin/my_init"  24 hours ago  Up
18 hours  0.0.0.0:80->80/tcp, 0.0.0.0:443->443/tcp, 443/tcp  grave_keller
```

4. Restart the container with that ID:

```
$ sudo podman restart 5a3e82c4a75f
```

5. Examine the certificate copied into the container namespace:

```
$ sudo podman exec -it 5a3e82c4a75f cat /etc/ssl/certs/storage.pem
-----BEGIN CERTIFICATE-----
MIIDTTCCAjWgAwIBAgIJAMVr9ngjJhzbMA0GCSqGSIb3DQEBCwUAMD0xCzAJBgNV
```

2.13. LDAP CONFIGURATION FIELDS

Table 2.11. LDAP configuration

Field	Type	Description
AUTHENTICATION_TYPE (Required)	String	Must be set to **LDAP**
FEATURE_TEAM_SYNCING	Boolean	Whether to allow for team membership to be synced from a backing group in the authentication engine (LDAP or Keystone) **Default: true**
FEATURE_NONSUPERUSER_TEAM_SYNCING_SETUP	Boolean	If enabled, non-superusers can setup syncing on teams using LDAP **Default: false**
LDAP_ADMIN_DN	String	The admin DN for LDAP authentication.
LDAP_ADMIN_PASSWD	String	The admin password for LDAP authentication.
LDAP_ALLOW_INSECURE_FALLBACK	Boolean	Whether or not to allow SSL insecure fallback for LDAP authentication.
LDAP_BASE_DN	Array of String	The base DN for LDAP authentication.
LDAP_EMAIL_ATTR	String	The email attribute for LDAP authentication.

Field	Type	Description
LDAP_UID_ATTR	String	The uid attribute for LDAP authentication.
LDAP_URI	String	The LDAP URI.
LDAP_USER_FILTER	String	The user filter for LDAP authentication.
LDAP_USER_RDN	Array of String	The user RDN for LDAP authentication.
TEAM_RESYNC_STALE_TIME	String	If team syncing is enabled for a team, how often to check its membership and resync if necessary **Pattern:** **^[0-9]+(w|m|d|h|s)$** **Example:** **2h** **Default:** **30m**

2.13.1. LDAP configuration example

$QUAY/config/config.yaml

```
AUTHENTICATION_TYPE: LDAP
...
LDAP_ADMIN_DN: uid=testuser,ou=Users,o=orgid,dc=jumpexamplecloud,dc=com
LDAP_ADMIN_PASSWD: samplepassword
LDAP_ALLOW_INSECURE_FALLBACK: false
LDAP_BASE_DN:
    - o=orgid
    - dc=example
    - dc=com
LDAP_EMAIL_ATTR: mail
LDAP_UID_ATTR: uid
LDAP_URI: ldap://ldap.example.com:389
LDAP_USER_RDN:
    - ou=Users
```

2.14. MIRRORING CONFIGURATION FIELDS

Table 2.12. Mirroring configuration

Field	Type	Description
FEATURE_REPO_MIRROR	Boolean	Enable or disable repository mirroring **Default:** **false**
REPO_MIRROR_INTERVAL	Number	The number of seconds between checking for repository mirror candidates **Default:** 30
REPO_MIRROR_SERVER_HOSTNAME	String	Replaces the **SERVER_HOSTNAME** as the destination for mirroring. **Default:** None **Example**: **openshift-quay-service**
REPO_MIRROR_TLS_VERIFY	Boolean	Require HTTPS and verify certificates of Quay registry during mirror. **Default:** **false**

2.15. SECURITY SCANNER CONFIGURATION FIELDS

Table 2.13. Security scanner configuration

Field	Type	Description
FEATURE_SECURITY_SCANNER	Boolean	Enable or disable the security scanner **Default:** **false**
FEATURE_SECURITY_NOTIFICATIONS	Boolean	If the security scanner is enabled, turn on or turn off security notifications **Default:** **false**

Field	Type	Description
SECURITY_SCANNER_V4_REINDEX_THRESHOLD	String	This parameter is used to determine the minimum time, in seconds, to wait before re-indexing a manifest that has either previously failed or has changed states since the last indexing. The data is calculated from the **last_indexed datetime** in the **manifestsecuritystatus** table. This parameter is used to avoid trying to re-index every failed manifest on every indexing run. The default time to re-index is 300 seconds.
SECURITY_SCANNER_V4_ENDPOINT	String	The endpoint for the V4 security scanner **Pattern:** **^http(s)?://(.)+$** **Example:** **http://192.168.99.101:6060**
SECURITY_SCANNER_V4_PSK	String	The generated pre-shared key (PSK) for Clair
SECURITY_SCANNER_INDEXING_INTERVAL	Number	The number of seconds between indexing intervals in the security scanner **Default:** 30
SECURITY_SCANNER_ENDPOINT	String	The endpoint for the V2 security scanner **Pattern:** **^http(s)?://(.)+$** **Example:** **http://192.168.99.100:6060**

Field	Type	Description
SECURITY_SCANNER_INDEXING_INTERVAL	String	This parameter is used to determine the number of seconds between indexing intervals in the security scanner. When indexing is triggered, Red Hat Quay will query its database for manifests that must be indexed by Clair. These include manifests that have not yet been indexed and manifests that previously failed indexing.

The following is a special case for re-indexing:

When Clair v4 indexes a manifest, the result should be deterministic. For example, the same manifest should produce the same index report. This is true until the scanners are changed, as using different scanners will produce different information relating to a specific manifest to be returned in the report. Because of this, Clair v4 exposes a state representation of the indexing engine (**/indexer/api/v1/index_state**) to determine whether the scanner configuration has been changed.

Red Hat Quay leverages this index state by saving it to the index report when parsing to Quay's database. If this state has changed since the manifest was previously scanned, Quay will attempt to re-index that manifest during the periodic indexing process.

By default this parameter is set to 30 seconds. Users might decrease the time if they want the indexing process to run more frequently, for example, if they did not want to wait 30 seconds to see security scan results in the UI after pushing a new tag. Users can also change the parameter if they want more control over the request pattern to Clair and the pattern of database operations being performed on the Quay database.

2.16. OCI AND HELM CONFIGURATION

Support for Helm is now supported under the **FEATURE_GENERAL_OCI_SUPPORT** property. If you need to explicitly enable the feature, for example, if it has previously been disabled or if you have upgraded from a version where it is not enabled by default, you need to add two properties in the Quay configuration to enable the use of OCI artifacts:

```
FEATURE_GENERAL_OCI_SUPPORT: true
FEATURE_HELM_OCI_SUPPORT: true
```

Table 2.14. OCI and Helm configuration

Field	Type	Description
FEATURE_GENERAL_OCI_SUPPORT	Boolean	Enable support for OCI artifacts **Default:** True

Field	Type	Description
FEATURE_HELM_OCI_SUPPORT	Boolean	Enable support for Helm artifacts **Default:** True

IMPORTANT

As of Red Hat Quay 3.6, **FEATURE_HELM_OCI_SUPPORT** has been deprecated and will be removed in a future version of Red Hat Quay. In Red Hat Quay 3.6, Helm artifacts are supported by default and included under the **FEATURE_GENERAL_OCI_SUPPORT** property. Users are no longer required to update their config.yaml files to enable support.

2.17. ACTION LOG CONFIGURATION FIELDS

2.17.1. Action log storage configuration

Table 2.15. Action log storage configuration

Field	Type	Description
FEATURE_LOG_EXPORT	Boolean	Whether to allow exporting of action logs **Default: True**
LOGS_MODEL	String	Enable or disable the security scanner **Values:** One of **database**, **transition_reads_both_writes_es**, **elasticsearch** **Default: database**
LOGS_MODEL_CONFIG	Object	Logs model config for action logs

- **LOGS_MODEL_CONFIG** [object]: Logs model config for action logs
  - **elasticsearch_config** [object]: Elasticsearch cluster configuration
    - **access_key** [string]: Elasticsearch user (or IAM key for AWS ES)
      - Example: **some_string**
    - **host** [string]: Elasticsearch cluster endpoint
      - Example: **host.elasticsearch.example**
    - **index_prefix** [string]: Elasticsearch's index prefix

- Example: **logentry_**

- **index_settings** [object]: Elasticsearch's index settings
- **use_ssl** [boolean]: Use ssl for Elasticsearch. Defaults to True
  - Example: **True**
- **secret_key** [string]: Elasticsearch password (or IAM secret for AWS ES)
  - Example: **some_secret_string**
- **aws_region** [string]: Amazon web service region
  - Example: **us-east-1**
- **port** [number]: Elasticsearch cluster endpoint port
  - Example: **1234**

- **kinesis_stream_config** [object]: AWS Kinesis Stream configuration
  - **aws_secret_key** [string]: AWS secret key
    - Example: **some_secret_key**
  - **stream_name** [string]: Kinesis stream to send action logs to
    - Example: **logentry-kinesis-stream**
  - **aws_access_key** [string]: AWS access key
    - Example: **some_access_key**
  - **retries** [number]: Max number of attempts made on a single request
    - Example: **5**
  - **read_timeout** [number]: Number of seconds before timeout when reading from a connection
    - Example: **5**
  - **max_pool_connections** [number]: The maximum number of connections to keep in a connection pool
    - Example: **10**
  - **aws_region** [string]: AWS region
    - Example: **us-east-1**
  - **connect_timeout** [number]: Number of seconds before timeout when attempting to make a connection
    - Example: **5**

- **producer** [string]: Logs producer if logging to Elasticsearch
  - **enum**: kafka, elasticsearch, kinesis_stream

- Example: **kafka**

- **kafka_config** [object]: Kafka cluster configuration
  - **topic** [string]: Kafka topic to publish log entries to
    - Example: **logentry**
  - **bootstrap_servers** [array]: List of Kafka brokers to bootstrap the client from
  - **max_block_seconds** [number]: Max number of seconds to block during a **send()**, either because the buffer is full or metadata unavailable
    - Example: **10**

2.17.2. Action log rotation and archiving configuration

Table 2.16. Action log rotation and archiving configuration

Field	Type	Description
FEATURE_ACTION_LOG_ROTATION	Boolean	Enabling log rotation and archival will move all logs older than 30 days to storage Default: **false**
ACTION_LOG_ARCHIVE_LOCATION	String	If action log archiving is enabled, the storage engine in which to place the archived data Example:: **s3_us_east**
ACTION_LOG_ARCHIVE_PATH	String	If action log archiving is enabled, the path in storage in which to place the archived data Example: **archives/actionlogs**
ACTION_LOG_ROTATION_THRESHOLD	String	The time interval after which to rotate logs Example: **30d**

2.18. BUILD LOGS

Table 2.17. Build logs

Field	Type	Description
FEATURE_READER_BUILD_LOGS	Boolean	If set to true, build logs may be read by those with read access to the repo, rather than only write access or admin access. **Default:** **False**
LOG_ARCHIVE_LOCATION	String	The storage location, defined in DISTRIBUTED_STORAGE_CONFIG, in which to place the archived build logs **Example:** **s3_us_east**
LOG_ARCHIVE_PATH	String	The path under the configured storage engine in which to place the archived build logs in JSON form **Example:** **archives/buildlogs**

2.19. DOCKERFILE BUILD TRIGGERS FIELDS

Table 2.18. Dockerfile build support

Field	Type	Description
FEATURE_BUILD_SUPPORT	Boolean	Whether to support Dockerfile build. **Default:** False
SUCCESSIVE_TRIGGER_FAILURE_DISABLE_THRESHOLD	Number	If not None, the number of successive failures that can occur before a build trigger is automatically disabled **Default:** 100
SUCCESSIVE_TRIGGER_INTERNAL_ERROR_DISABLE_THRESHOLD	Number	If not None, the number of successive internal errors that can occur before a build trigger is automatically disabled **Default:** 5

2.19.1. GitHub build triggers

Table 2.19. GitHub build triggers

Field	Type	Description
FEATURE_GITHUB_BUILD	Boolean	Whether to support GitHub build triggers **Default:** False
GITHUB_TRIGGER_CONFIG	Object	Configuration for using GitHub (Enterprise) for build triggers
.GITHUB_ENDPOINT (Required)	String	The endpoint for GitHub (Enterprise) **Example:** **https://github.com/**
.API_ENDPOINT	String	The endpoint of the GitHub (Enterprise) API to use. Must be overridden for **github.com** **Example**: **https://api.github.com/**
.CLIENT_ID (Required)	String	The registered client ID for this Red Hat Quay instance; this cannot be shared with GITHUB_LOGIN_CONFIG.
.CLIENT_SECRET (Required)	String	The registered client secret for this Red Hat Quay instance.

2.19.2. BitBucket build triggers

Table 2.20. BitBucket build triggers

Field	Type	Description
FEATURE_BITBUCKET_BUILD	Boolean	Whether to support Bitbucket build triggers **Default:** False

Field	Type	Description
BITBUCKET_TRIGGER_CONFIG	Object	Configuration for using BitBucket for build triggers
.CONSUMER_KEY (Required)	String	The registered consumer key (client ID) for this Quay instance
.CONSUMER_SECRET (Required)	String	The registered consumer secret (client secret) for this Quay instance

2.19.3. GitLab build triggers

Table 2.21. GitLab build triggers

Field	Type	Description
FEATURE_GITLAB_BUILD	Boolean	Whether to support GitLab build triggers **Default:** False
GITLAB_TRIGGER_CONFIG	Object	Configuration for using Gitlab for build triggers
.GITLAB_ENDPOINT (Required)	String	The endpoint at which Gitlab (Enterprise) is running
.CLIENT_ID (Required)	String	The registered client ID for this Quay instance
.CLIENT_SECRET (Required)	String	The registered client secret for this Quay instance

2.20. OAUTH CONFIGURATION

Table 2.22. OAuth fields

Field	Type	Description
DIRECT_OAUTH_CLIENTID_WHITELIST	Array of String	A list of client IDs for **Quay-managed** applications that are allowed to perform direct OAuth approval without user approval.

2.20.1. GitHub OAuth

Table 2.23. GitHub OAuth fields

Field	Type	Description
FEATURE_GITHUB_LOGIN	Boolean	Whether GitHub login is supported **Default: **False**
GITHUB_LOGIN_CONFIG	Object	Configuration for using GitHub (Enterprise) as an external login provider.
.ALLOWED_ORGANIZATIONS	Array of String	The names of the GitHub (Enterprise) organizations whitelisted to work with the ORG_RESTRICT option.
.API_ENDPOINT	String	The endpoint of the GitHub (Enterprise) API to use. Must be overridden for github.com **Example:** **https://api.github.com/**
.CLIENT_ID (Required)	String	The registered client ID for this Red Hat Quay instance; cannot be shared with GITHUB_TRIGGER_CONFIG **Example:** **0e8dbe15c4c7630b6780**
.CLIENT_SECRET (Required)	String	The registered client secret for this Red Hat Quay instance **Example:** **e4a58ddd3d7408b7aec109e85564a0d153d3e846**
.GITHUB_ENDPOINT (Required)	String	The endpoint for GitHub (Enterprise) **Example**: **https://github.com/**
.ORG_RESTRICT	Boolean	If true, only users within the organization whitelist can login using this provider.

2.20.2. Google OAuth

Table 2.24. Google OAuth fields

Field	Type	Description
FEATURE_GOOGLE_LOGIN	Boolean	Whether Google login is supported **Default: **False**
GOOGLE_LOGIN_CONFIG	Object	Configuration for using Google for external authentication
.CLIENT_ID (Required)	String	The registered client ID for this Red Hat Quay instance **Example:** **0e8dbe15c4c7630b6780**
.CLIENT_SECRET (Required)	String	The registered client secret for this Red Hat Quay instance **Example:** e4a58ddd3d7408b7aec109e85564a0d153d3e846

2.21. CONFIGURING NESTED REPOSITORIES

With Red Hat Quay 3.6, support for nested repository path names has been added under the **FEATURE_EXTENDED_REPOSITORY_NAMES** property. This optional configuration must be manually added to the config.yaml by the user to enable support. Enablement allows the use of / in repository names.

```
FEATURE_EXTENDED_REPOSITORY_NAMES: true
```

Table 2.25. OCI and nested repositories configuration

Field	Type	Description
FEATURE_EXTENDED_REPOSITORY_NAMES	Boolean	Enable support for nested repositories **Default:** False

2.22. ADDING OTHER OCI MEDIA TYPES TO QUAY

Helm, cosign, and ztsd compression scheme artifacts are built into Red Hat Quay 3.6 by default. For any other OCI media type that is not supported by default, you can add them to the **ALLOWED_OCI_ARTIFACT_TYPES** configuration in Quay's config.yaml using the following format:

```
ALLOWED_OCI_ARTIFACT_TYPES:
  <oci config type 1>:
```

```
  - <oci layer type 1>
  - <oci layer type 2>

  <oci config type 2>:
  - <oci layer type 3>
  - <oci layer type 4>
...
```

For example, you can add Singularity (SIF) support by adding the following to your config.yaml:

```
...
ALLOWED_OCI_ARTIFACT_TYPES:
  application/vnd.oci.image.config.v1+json
  - application/vnd.dev.cosign.simplesigning.v1+json
  application/vnd.cncf.helm.config.v1+json
  - application/tar+gzip
  application/vnd.sylabs.sif.config.v1+json
  - application/vnd.sylabs.sif.layer.v1+tar
...
```

NOTE

When adding OCI media types that are not configured by default, users will also need to manually add support for cosign and Helm if desired. The ztsd compression scheme is supported by default, so users will not need to add that OCI media type to their config.yaml to enable support.

2.23. MAIL CONFIGURATION

Table 2.26. Mail fields

Field	Type	Description
FEATURE_MAILING	Boolean	Whether emails are enabled **Default: False**
MAIL_DEFAULT_SENDER	String	If specified, the e-mail address used as the **from** when Red Hat Quay sends e-mails. If none, defaults to **support@quay.io** **Example: support@example.com**
MAIL_PASSWORD	String	The SMTP password to use when sending e-mails
MAIL_PORT	Number	The SMTP port to use. If not specified, defaults to 587.

Field	Type	Description
MAIL_SERVER	String	The SMTP server to use for sending e-mails. Only required if FEATURE_MAILING is set to true. Example: **smtp.example.com**
MAIL_USERNAME	String	The SMTP username to use when sending e-mails
MAIL_USE_TLS	Boolean	If specified, whether to use TLS for sending e-mails Default: **True**

2.24. USER CONFIGURATION FIELDS

Table 2.27. User configuration

Field	Type	Description
FEATURE_SUPER_USERS	Boolean	Whether superusers are supported Default: **true**
FEATURE_USER_CREATION	Boolean	Whether users can be created (by non-superusers) Default: **true**
FEATURE_USER_LAST_ACCESSED	Boolean	Whether to record the last time a user was accessed Default: **true**
FEATURE_USER_LOG_ACCESS	Boolean	If set to true, users will have access to audit logs for their namespace Default: **false**
FEATURE_USER_METADATA	Boolean	Whether to collect and support user metadata Default: **false**

Field	Type	Description
FEATURE_USERNAME_CONFIRMATION	Boolean	If set to true, users can confirm their generated usernames Default: **true**
FEATURE_USER_RENAME	Boolean	If set to true, users can rename their own namespace Default: **false**
FEATURE_INVITE_ONLY_USER_CREATION	Boolean	Whether users being created must be invited by another user Default: **false**
FRESH_LOGIN_TIMEOUT	String	The time after which a fresh login requires users to re-enter their password Example: **5m**
USERFILES_LOCATION	String	ID of the storage engine in which to place user-uploaded files Example: **s3_us_east**
USERFILES_PATH	String	Path under storage in which to place user-uploaded files Example: **userfiles**
USER_RECOVERY_TOKEN_LIFETIME	String	The length of time a token for recovering a user accounts is valid Pattern: **^[0-9]+(w|m|d|h|s)$** Default: **30m**

2.25. RECAPTCHA CONFIGURATION

Table 2.28. Recaptcha fields

Field	Type	Description
FEATURE_RECAPTCHA	Boolean	Whether Recaptcha is necessary for user login and recovery **Default:** False
RECAPTCHA_SECRET_KEY	String	If recaptcha is enabled, the secret key for the Recaptcha service
RECAPTCHA_SITE_KEY	String	If recaptcha is enabled, the site key for the Recaptcha service

2.26. ACI CONFIGURATION

Table 2.29. ACI configuration

Field	Type	Description
FEATURE_ACI_CONVERSION	Boolean	Whether to enable conversion to ACIs **Default:** False
GPG2_PRIVATE_KEY_FILENAME	String	The filename of the private key used to decrypte ACIs
GPG2_PRIVATE_KEY_NAME	String	The name of the private key used to sign ACIs
GPG2_PUBLIC_KEY_FILENAME	String	The filename of the public key used to encrypt ACIs

2.27. JTW CONFIGURATION

Table 2.30. JWT configuration

Field	Type	Description
JWT_AUTH_ISSUER	String	The endpoint for JWT users **Pattern:** **^http(s)?://(.)+$** **Example:** **http://192.168.99.101:6060**

Field	Type	Description
JWT_GETUSER_ENDPOINT	String	The endpoint for JWT users **Pattern: ^http(s)?://(.)+$** **Example:** **http://192.168.99.101:6060**
JWT_QUERY_ENDPOINT	String	The endpoint for JWT queries **Pattern: ^http(s)?://(.)+$** **Example:** **http://192.168.99.101:6060**
JWT_VERIFY_ENDPOINT	String	The endpoint for JWT verification **Pattern: ^http(s)?://(.)+$** **Example:** **http://192.168.99.101:6060**

2.28. APP TOKENS

Table 2.31. App tokens configuration

Field	Type	Description
FEATURE_APP_SPECIFIC_TOKENS	Boolean	If enabled, users can create tokens for use by the Docker CLI **Default:** True
APP_SPECIFIC_TOKEN_EXPIRATION	String	The expiration for external app tokens. **Default** None **Pattern: ^[0-9]+(w|m|d|h|s)$**
EXPIRED_APP_SPECIFIC_TOKEN_GC	String	Duration of time expired external app tokens will remain before being garbage collected **Default:** 1d

2.29. MISCELLANEOUS FIELDS

Table 2.32. Miscellaneous fields

Field	Type	Description
ALLOW_PULLS_WITHOUT_STRICT_LOGGING	String	If true, pulls will still succeed even if the pull audit log entry cannot be written . This is useful if the database is in a read-only state and it is desired for pulls to continue during that time. **Default:** False
AVATAR_KIND	String	The types of avatars to display, either generated inline (local) or Gravatar (gravatar) **Values:** local, gravatar
BROWSER_API_CALLS_XHR_ONLY	Boolean	If enabled, only API calls marked as being made by an XHR will be allowed from browsers **Default:** True
DEFAULT_NAMESPACE_MAXIMUM_BUILD_COUNT	Number	The default maximum number of builds that can be queued in a namespace. **Default:** None
ENABLE_HEALTH_DEBUG_SECRET	String	If specified, a secret that can be given to health endpoints to see full debug info when not authenticated as a superuser
EXTERNAL_TLS_TERMINATION	Boolean	Set to true if TLS is supported, but terminated at a layer before Quay
FRESH_LOGIN_TIMEOUT	String	The time after which a fresh login requires users to re-enter their password **Example:** **5m**
HEALTH_CHECKER	String	The configured health check **Example:** **('RDSAwareHealthCheck', {'access_key': 'foo', 'secret_key': 'bar'})**

Field	Type	Description
PROMETHEUS_NAMESPACE	String	The prefix applied to all exposed Prometheus metrics **Default:** **quay**
PUBLIC_NAMESPACES	Array of String	If a namespace is defined in the public namespace list, then it will appear on **all** users' repository list pages, regardless of whether the user is a member of the namespace. Typically, this is used by an enterprise customer in configuring a set of "well-known" namespaces.
REGISTRY_STATE	String	The state of the registry **Values:** **normal** or **read-only**
SEARCH_MAX_RESULT_PAGE_COUNT	Number	Maximum number of pages the user can paginate in search before they are limited **Default:** 10
SEARCH_RESULTS_PER_PAGE	Number	Number of results returned per page by search page **Default:** 10
V1_PUSH_WHITELIST	Array of String	The array of namespace names that support V1 push if FEATURE_RESTRICTED_V1_PUSH is set to true
V2_PAGINATION_SIZE	Number	The number of results returned per page in V2 registry APIs
WEBHOOK_HOSTNAME_BLACKLIST	Array of String	The set of hostnames to disallow from webhooks when validating, beyond localhost
CREATE_PRIVATE_REPO_ON_PUSH	Boolean	Whether new repositories created by push are set to private visibility **Default:** True

Field	Type	Description
CREATE_NAMESPACE_ON_PUSH	Boolean	Whether new push to a non-existent organization creates it **Default:** False
NON_RATE_LIMITED_NAMESPACES	Array of String	If rate limiting has been enabled using **FEATURE_RATE_LIMITS**, you can override it for specific namespace that require unlimited access.

2.30. LEGACY CONFIGURATION FIELDS

Some fields are deprecated or obsolete:

Table 2.33. Legacy fields

Field	Type	Description
FEATURE_BLACKLISTED_EMAILS	Boolean	If set to true, no new User accounts may be created if their email domain is blacklisted
BLACKLISTED_EMAIL_DOMAINS	Array of String	The list of email-address domains that is used if FEATURE_BLACKLISTED_EMAILS is set to true Example: **"example.com", "example.org"**
BLACKLIST_V2_SPEC	String	The Docker CLI versions to which Red Hat Quay will respond that V2 is **unsupported** Example: **<1.8.0** Default: **<1.6.0**
DOCUMENTATION_ROOT	String	Root URL for documentation links
SECURITY_SCANNER_V4_NAMESPACE_WHITELIST	String	The namespaces for which the security scanner should be enabled

CHAPTER 3. ENVIRONMENT VARIABLES

Red Hat Quay supports a limited number of environment variables for dynamic configuration.

3.1. GEO-REPLICATION

The exact same configuration should be used across all regions, with exception of the storage backend, which can be configured explicitly using the **QUAY_DISTRIBUTED_STORAGE_PREFERENCE** environment variable.

Table 3.1. Geo-replication configuration

Variable	Type	Description
QUAY_DISTRIBUTED_STORAGE_PREFERENCE	String	The preferred storage engine (by ID in DISTRIBUTED_STORAGE_CONFIG) to use.

3.2. DATABASE CONNECTION POOLING

Red Hat Quay is composed of many different processes which all run within the same container. Many of these processes interact with the database.

If enabled, each process that interacts with the database will contain a connection pool. These per-process connection pools are configured to maintain a maximum of 20 connections. Under heavy load, it is possible to fill the connection pool for every process within a Red Hat Quay container. Under certain deployments and loads, this may require analysis to ensure Red Hat Quay does not exceed the database's configured maximum connection count.

Overtime, the connection pools will release idle connections. To release all connections immediately, Red Hat Quay requires a restart.

Database connection pooling may be toggled by setting the environment variable DB_CONNECTION_POOLING={true|false}

Table 3.2. Database connection pooling configuration

Variable	Type	Description
DB_CONNECTION_POOLING	Boolean	Enable or disable database connection pooling

If database connection pooling is enabled, it is possible to change the maximum size of the connection pool. This can be done through the following config.yaml option:

config.yaml

```
...
DB_CONNECTION_ARGS:
  max_connections: 10
...
```

3.3. HTTP CONNECTION COUNTS

It is possible to specify the quantity of simultaneous HTTP connections using environment variables. These can be specified as a whole, or for a specific component. The default for each is 50 parallel connections per process.

Table 3.3. HTTP connection counts configuration

Variable	Type	Description
WORKER_CONNECTION_COUNT	Number	Simultaneous HTTP connections **Default:** 50
WORKER_CONNECTION_COUNT_REGISTRY	Number	Simultaneous HTTP connections for registry **Default:** WORKER_CONNECTION_COUNT
WORKER_CONNECTION_COUNT_WEB	Number	Simultaneous HTTP connections for web UI **Default:** WORKER_CONNECTION_COUNT
WORKER_CONNECTION_COUNT_SECSCAN	Number	Simultaneous HTTP connections for Clair **Default:** WORKER_CONNECTION_COUNT

3.4. WORKER COUNT VARIABLES

Table 3.4. Worker count variables

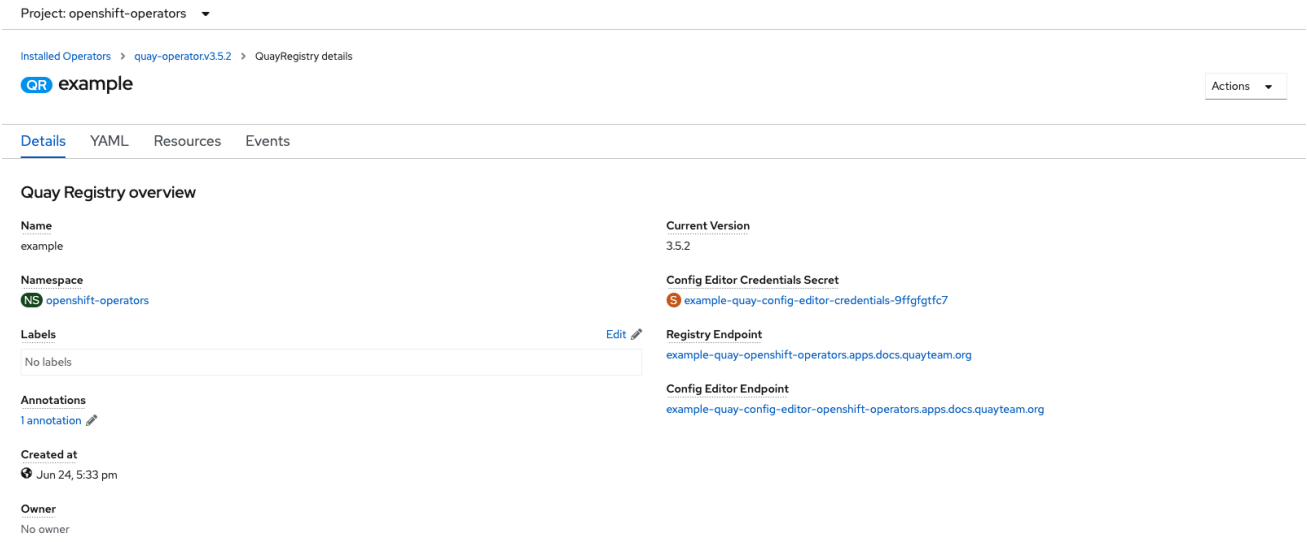
Variable	Type	Description
WORKER_COUNT	Number	Generic override for number of processes
WORKER_COUNT_REGISTRY	Number	Specifies the number of processes to handle Registry requests within the **Quay** container **Values:** Integer between 8 and 64

Variable	Type	Description
WORKER_COUNT_WEB	Number	Specifies the number of processes to handle UI/Web requests within the container **Values:** Integer between 2 and 32
WORKER_COUNT_SECSCAN	Number	Specifies the number of processes to handle Security Scanning (e.g. Clair) integration within the container **Values:** Integer between 2 and 4

CHAPTER 4. USING THE CONFIG TOOL TO RECONFIGURE QUAY ON OPENSHIFT

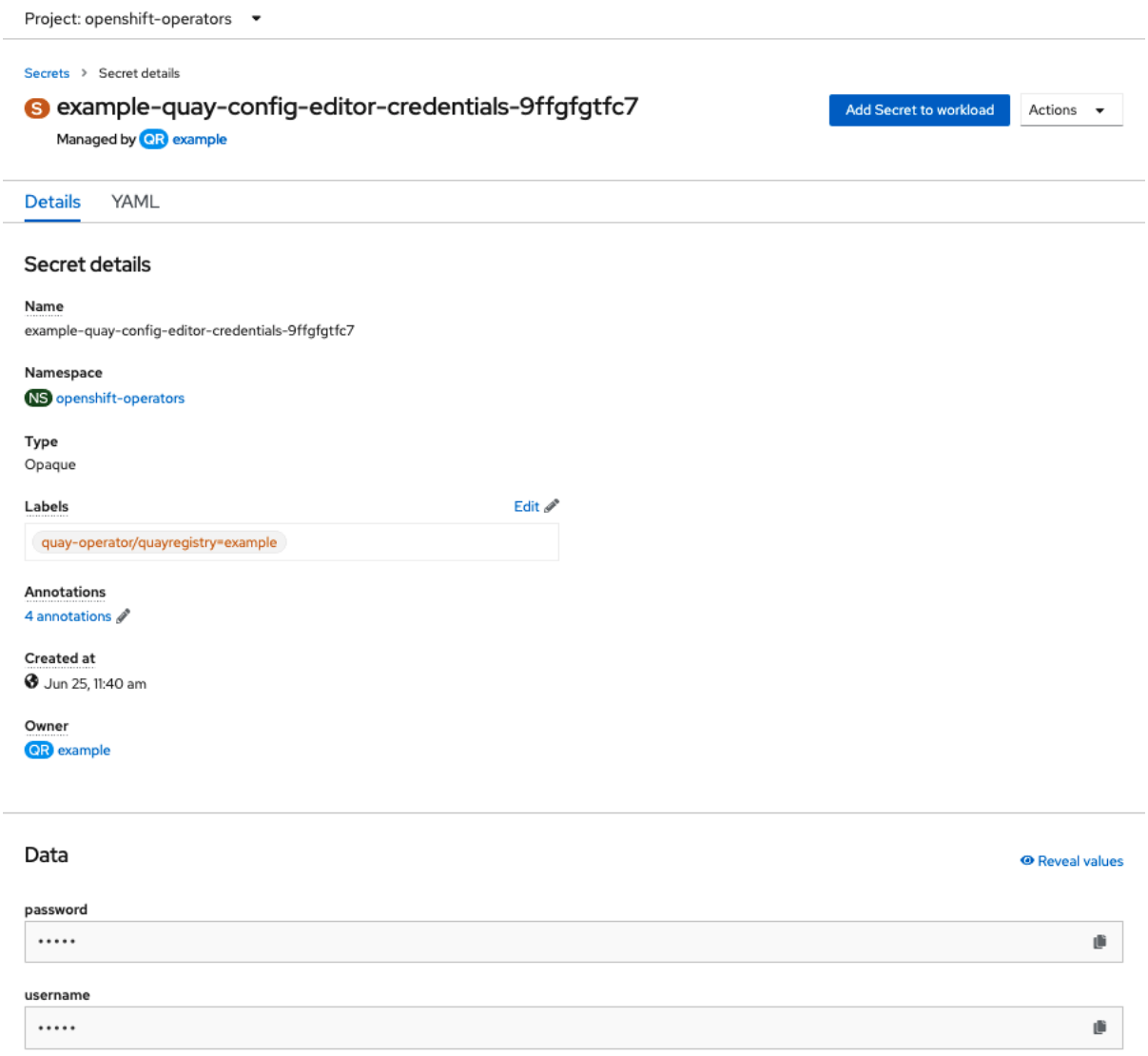
4.1. ACCESSING THE CONFIG EDITOR

In the Details section of the QuayRegistry screen, the endpoint for the config editor is available, along with a link to the secret containing the credentials for logging into the config editor:

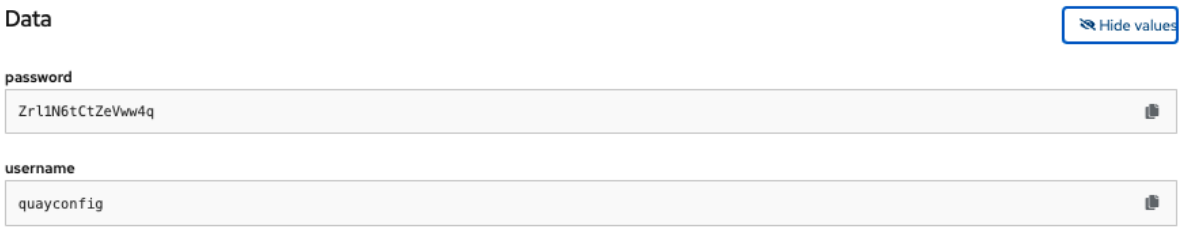


4.1.1. Retrieving the config editor credentials

1. Click on the link for the config editor secret:

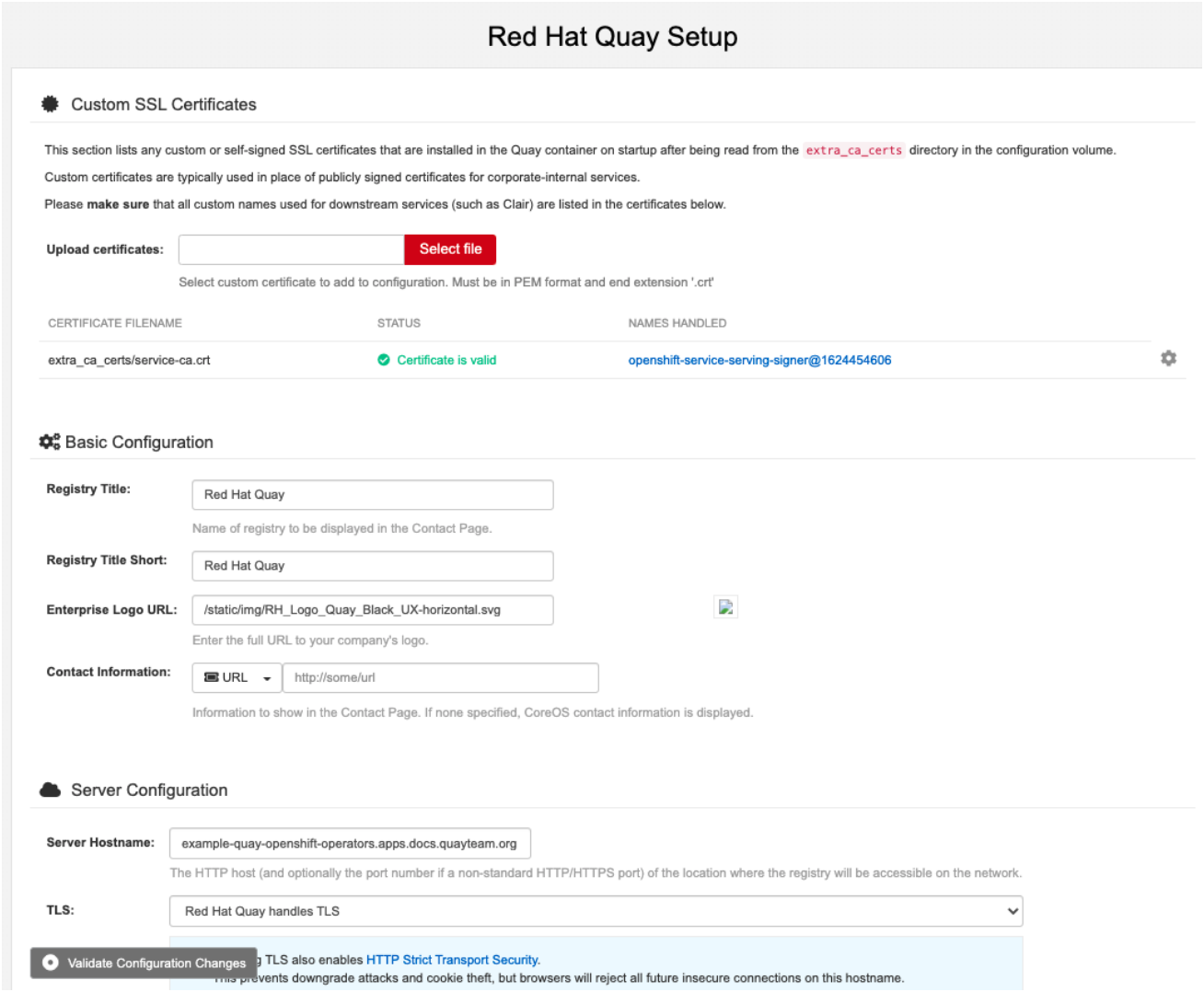


2. In the Data section of the Secret details screen, click **Reveal values** to see the credentials for logging in to the config editor:



4.1.2. Logging in to the config editor

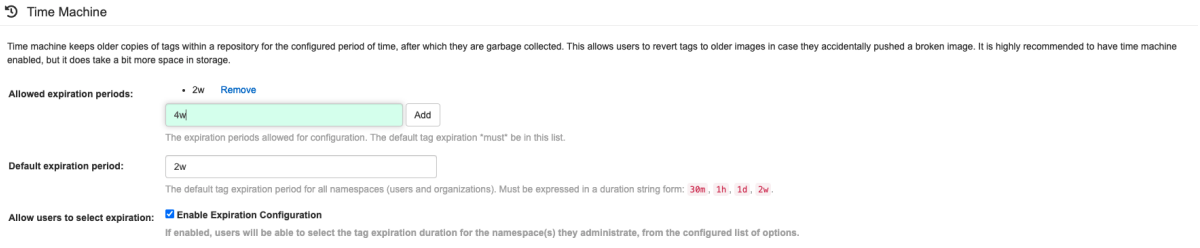
Browse to the config editor endpoint and then enter the username, typically **quayconfig**, and the corresponding password to access the config tool:



4.1.3. Changing configuration

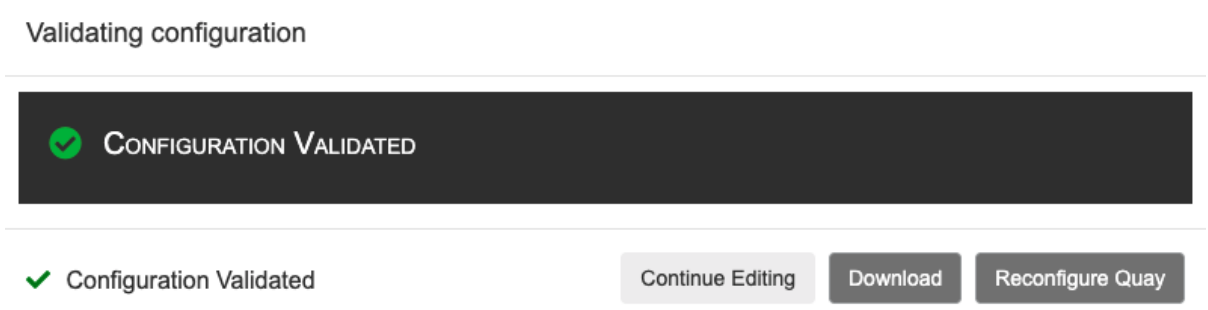
In this example of updating the configuration, a superuser is added via the config editor tool:

1. Add an expiration period, for example **4w**, for the time machine functionality:

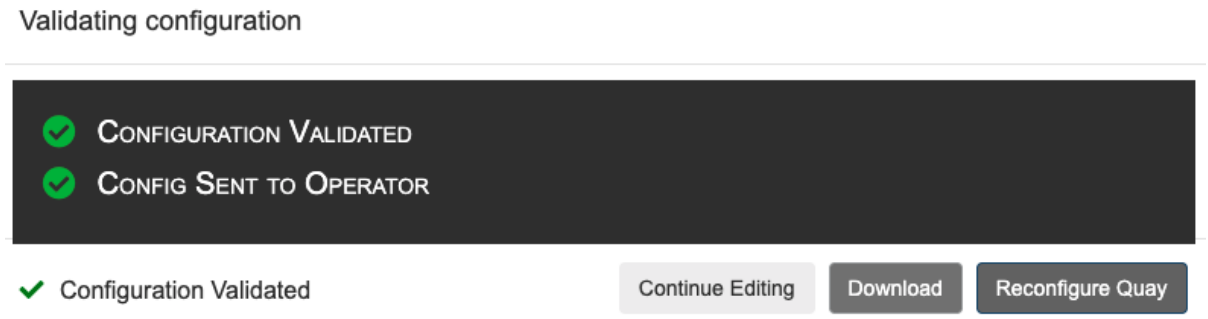


2. Select **Validate Configuration Changes** to ensure that the changes are valid

3. Apply the changes by pressing the **Reconfigure Quay** button:



4. The config tool notifies you that the change has been submitted to Quay:



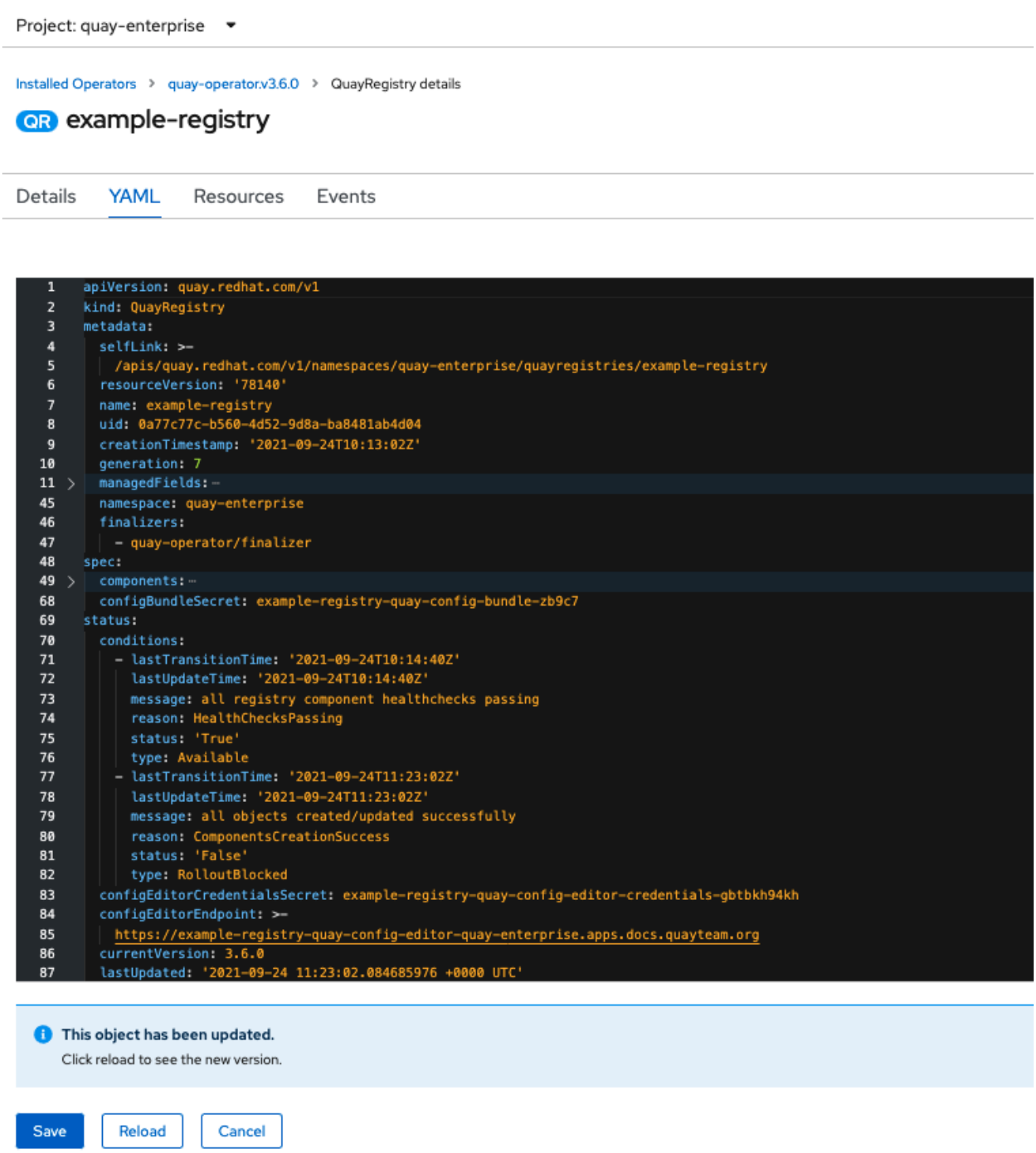
NOTE

Reconfiguring Red Hat Quay using the config tool UI can lead to the registry being unavailable for a short time, while the updated configuration is applied.

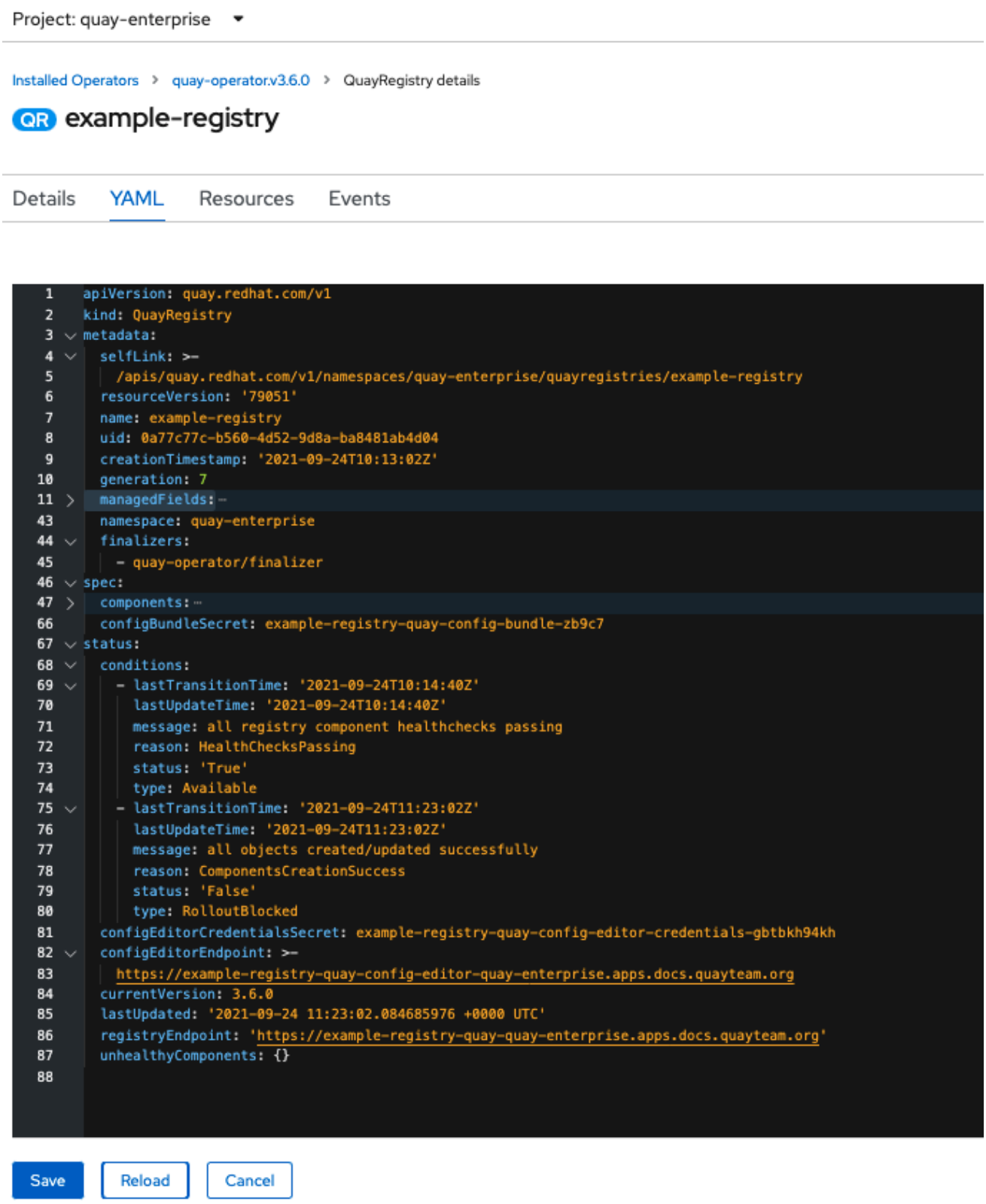
4.2. MONITORING RECONFIGURATION IN THE UI

4.2.1. QuayRegistry resource

After reconfiguring the Operator, you can track the progress of the redeployment in the YAML tab for the specific instance of QuayRegistry, in this case, **example-registry**:

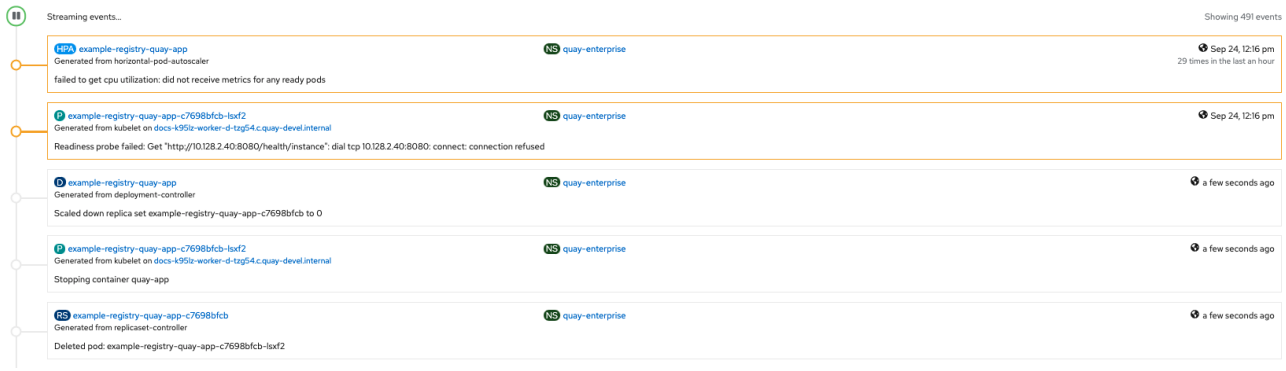


Each time the status changes, you will be prompted to reload the data to see the updated version. Eventually, the Operator will reconcile the changes, and there will be no unhealthy components reported.

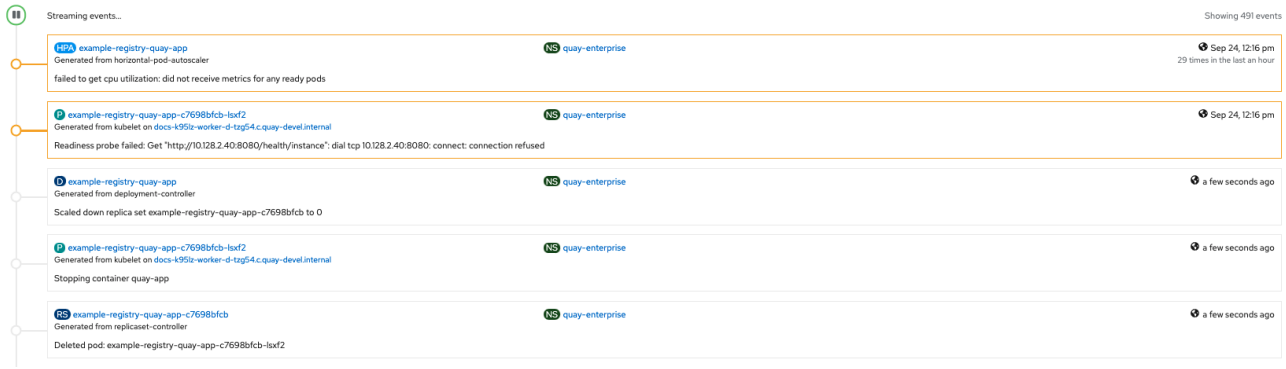


4.2.2. Events

The Events tab for the QuayRegistry shows some events related to the redeployment:



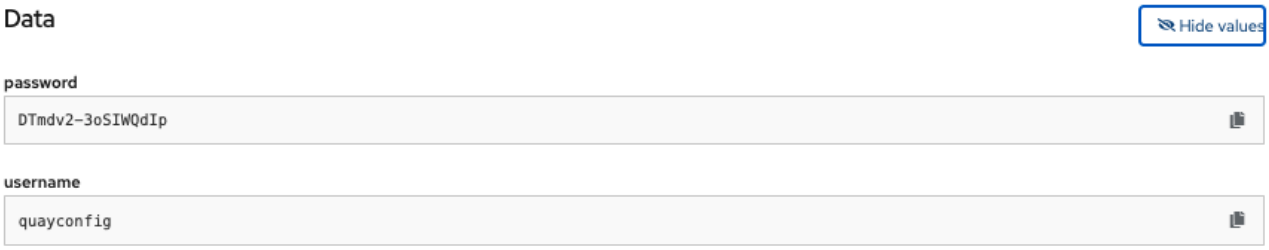
Streaming events, for all resources in the namespace that are affected by the reconfiguration, are available in the OpenShift console under Home → Events:



4.3. ACCESSING UPDATED INFORMATION AFTER RECONFIGURATION

4.3.1. Accessing the updated config tool credentials in the UI

Since a new pod has been created for the config tool, a new secret will have been created, and you will need to use the updated password when you next attempt to login:



4.3.2. Accessing the updated config.yaml in the UI

Use the config bundle to access the updated **config.yaml** file.

1. On the QuayRegistry details screen, click on the Config Bundle Secret
2. In the Data section of the Secret details screen, click Reveal values to see the **config.yaml** file
3. Check that the change has been applied. In this case, **4w** should be in the list of **TAG_EXPIRATION_OPTIONS**:

```
...
SERVER_HOSTNAME: example-quay-openshift-operators.apps.docs.quayteam.org
SETUP_COMPLETE: true
SUPER_USERS:
- quayadmin
TAG_EXPIRATION_OPTIONS:
- 2w
- 4w
...
```

CHAPTER 5. QUAY OPERATOR COMPONENTS

Quay is a powerful container registry platform and as a result, has a significant number of dependencies. These include a database, object storage, Redis, and others. The Quay Operator manages an opinionated deployment of Quay and its dependencies on Kubernetes. These dependencies are treated as *components* and are configured through the **QuayRegistry** API.

In the **QuayRegistry** custom resource, the **spec.components** field configures components. Each component contains two fields: **kind** - the name of the component, and **managed** - boolean whether the component lifecycle is handled by the Operator. By default (omitting this field), all components are managed and will be autofilled upon reconciliation for visibility:

```
spec:
  components:
    - managed: true
      kind: clair
    - managed: true
      kind: postgres
    - managed: true
      kind: objectstorage
    - managed: true
      kind: redis
    - managed: true
      kind: horizontalpodautoscaler
    - managed: true
      kind: route
    - managed: true
      kind: mirror
    - managed: true
      kind: monitoring
    - managed: true
      kind: tls
```

5.1. USING MANAGED COMPONENTS

Unless your **QuayRegistry** custom resource specifies otherwise, the Operator will use defaults for the following managed components:

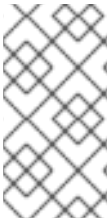
- **postgres:** For storing the registry metadata, uses a version of Postgres 10 from the Software Collections
- **redis:** Handles Quay builder coordination and some internal logging
- **objectstorage:** For storing image layer blobs, utilizes the **ObjectBucketClaim** Kubernetes API which is provided by Noobaa/RHOCS
- **clair:** Provides image vulnerability scanning
- **horizontalpodautoscaler:** Adjusts the number of Quay pods depending on memory/cpu consumption
- **mirror:** Configures repository mirror workers (to support optional repository mirroring)
- **route:** Provides an external entrypoint to the Quay registry from outside OpenShift

- **monitoring:** Features include a Grafana dashboard, access to individual metrics, and alerting to notify for frequently restarting Quay pods
- **tls:** Configures whether Red Hat Quay or OpenShift handles TLS

The Operator will handle any required configuration and installation work needed for Red Hat Quay to use the managed components. If the opinionated deployment performed by the Quay Operator is unsuitable for your environment, you can provide the Operator with **unmanaged** resources (overrides) as described in the following sections.

5.2. USING UNMANAGED COMPONENTS FOR DEPENDENCIES

If you have existing components such as Postgres, Redis or object storage that you would like to use with Quay, you first configure them within the Quay configuration bundle (**config.yaml**) and then reference the bundle in your **QuayRegistry** (as a Kubernetes **Secret**) while indicating which components are unmanaged.



NOTE

The Quay config editor can also be used to create or modify an existing config bundle and simplifies the process of updating the Kubernetes **Secret**, especially for multiple changes. When Quay's configuration is changed via the config editor and sent to the Operator, the Quay deployment will be updated to reflect the new configuration.

5.2.1. Using an existing Postgres database

1. Create a configuration file **config.yaml** with the necessary database fields:

   config.yaml:

   ```
   DB_URI: postgresql://test-quay-database:postgres@test-quay-database:5432/test-quay-database
   ```

2. Create a Secret using the configuration file:

   ```
   $ kubectl create secret generic --from-file config.yaml=./config.yaml config-bundle-secret
   ```

3. Create a QuayRegistry YAML file **quayregistry.yaml** which marks the **postgres** component as unmanaged and references the created Secret:

   quayregistry.yaml

   ```
   apiVersion: quay.redhat.com/v1
   kind: QuayRegistry
   metadata:
     name: example-registry
     namespace: quay-enterprise
   spec:
     configBundleSecret: config-bundle-secret
     components:
       - kind: postgres
         managed: false
   ```

4. Deploy the registry as detailed in the following sections.

5.2.2. NooBaa unmanaged storage

1. Create a NooBaa Object Bucket Claim in the console at Storage → Object Bucket Claims.
2. Retrieve the Object Bucket Claim Data details including the Access Key, Bucket Name, Endpoint (hostname) and Secret Key.
3. Create a **config.yaml** configuration file, using the information for the Object Bucket Claim:

```
DISTRIBUTED_STORAGE_CONFIG:
  default:
    - RHOCSStorage
    - access_key: WmrXtSGk8B3nABCDEFGH
      bucket_name: my-noobaa-bucket-claim-8b844191-dc6c-444e-9ea4-87ece0abcdef
      hostname: s3.openshift-storage.svc.cluster.local
      is_secure: true
      port: "443"
      secret_key: X9P5SDGJtmSuHFCMSLMbdNCMfUABCDEFGH+C5QD
      storage_path: /datastorage/registry
DISTRIBUTED_STORAGE_DEFAULT_LOCATIONS: []
DISTRIBUTED_STORAGE_PREFERENCE:
  - default
```

5.2.3. Disabling the Horizontal Pod Autoscaler

HorizontalPodAutoscalers have been added to the Clair, Quay, and Mirror pods, so that they now automatically scale during load spikes.

As HPA is configured by default to be **managed**, the number of pods for Quay, Clair and repository mirroring is set to two. This facilitates the avoidance of downtime when updating / reconfiguring Quay via the Operator or during rescheduling events.

If you wish to disable autoscaling or create your own **HorizontalPodAutoscaler**, simply specify the component as unmanaged in the **QuayRegistry** instance:

```
apiVersion: quay.redhat.com/v1
kind: QuayRegistry
metadata:
  name: example-registry
  namespace: quay-enterprise
spec:
  components:
    - kind: horizontalpodautoscaler
      managed: false
```

5.3. ADD CERTS WHEN DEPLOYED ON KUBERNETES

When deployed on Kubernetes, Red Hat Quay mounts in a secret as a volume to store config assets. Unfortunately, this currently breaks the upload certificate function of the superuser panel.

To get around this error, a base64 encoded certificate can be added to the secret *after* Red Hat Quay has been deployed. Here's how:

1. Begin by base64 encoding the contents of the certificate:

```
$ cat ca.crt
-----BEGIN CERTIFICATE-----
MIIDljCCAn6gAwIBAgIBATANBgkqhkiG9w0BAQsFADA5MRcwFQYDVQQKDA5MQUlu
TElCQ09SRS5TTzEeMBwGA1UEAwwVQ2VydGlmaWNhdGUgQXV0aG9yaXR5MB4XDTE2
MDExMjA2NTkxMFoXDTM2MDExMjA2NTkxMFowOTEXMBUGA1UECgwOTEFCLkxJQkNP
UkUuU08xHjAcBgNVBAMMFUNlcnRpZmljYXRlIEF1dGhvcml0eTCCASIwDQYJKoZI
[...]
-----END CERTIFICATE-----

$ cat ca.crt | base64 -w 0
[...]
c1psWGpqeGlPQmNEWkJPMjJ5d0pDemVnR2QNCnRsbW9JdEF4YnFSdVd3PT0KLS0tLS1F
TkQgQ0VSVElGSUNBVEUtLS0tLQo=
```

2. Use the **kubectl** tool to edit the quay-enterprise-config-secret.

```
$ kubectl --namespace quay-enterprise edit secret/quay-enterprise-config-secret
```

3. Add an entry for the cert and paste the full base64 encoded string under the entry:

```
  custom-cert.crt:
c1psWGpqeGlPQmNEWkJPMjJ5d0pDemVnR2QNCnRsbW9JdEF4YnFSdVd3PT0KLS0tLS1F
TkQgQ0VSVElGSUNBVEUtLS0tLQo=
```

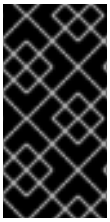
4. Finally, recycle all Red Hat Quay pods. Use **kubectl delete** to remove all Red Hat Quay pods. The Red Hat Quay Deployment will automatically schedule replacement pods with the new certificate data.

5.4. CONFIGURING OCI AND HELM WITH THE OPERATOR

Customizations to the configuration of Quay can be provided in a secret containing the configuration bundle. Execute the following command which will create a new secret called **quay-config-bundle**, in the appropriate namespace, containing the necessary properties to enable OCI support.

quay-config-bundle.yaml

```
apiVersion: v1
stringData:
  config.yaml: |
    FEATURE_GENERAL_OCI_SUPPORT: true
    FEATURE_HELM_OCI_SUPPORT: true
kind: Secret
metadata:
  name: quay-config-bundle
  namespace: quay-enterprise
type: Opaque
```



IMPORTANT

As of Red Hat Quay 3.6, **FEATURE_HELM_OCI_SUPPORT** has been deprecated and will be removed in a future version of Red Hat Quay. In Red Hat Quay 3.6, Helm artifacts are supported by default and included under the **FEATURE_GENERAL_OCI_SUPPORT** property. Users are no longer required to update their config.yaml files to enable support.

Create the secret in the appropriate namespace, in this example **quay-enterprise**:

```
$ oc create -n quay-enterprise -f quay-config-bundle.yaml
```

Specify the secret for the **spec.configBundleSecret** field:

quay-registry.yaml

```
apiVersion: quay.redhat.com/v1
kind: QuayRegistry
metadata:
  name: example-registry
  namespace: quay-enterprise
spec:
  configBundleSecret: quay-config-bundle
```

Create the registry with the specified configuration:

```
$ oc create -n quay-enterprise -f quay-registry.yaml
```

5.5. VOLUME SIZE OVERRIDES

As of Red Hat Quay v3.6.2, you can specify the desired size of storage resources provisioned for managed components. The default size for Clair and Quay PostgreSQL databases is **50Gi**. You can now choose a large enough capacity upfront, either for performance reasons or in the case where your storage backend does not have resize capability.

In the following example, the volume size for the Clair and the Quay PostgreSQL databases has been set to **70Gi**:

```
apiVersion: quay.redhat.com/v1
kind: QuayRegistry
metadata:
  name: quay-example
  namespace: quay-enterprise
spec:
  configBundleSecret: config-bundle-secret
  components:
    - kind: objectstorage
      managed: false
    - kind: route
      managed: true
    - kind: tls
      managed: false
    - kind: clair
      managed: true
```

```
    overrides:
      volumeSize: 70Gi
  - kind: postgres
    managed: true
    overrides:
      volumeSize: 70Gi
```

CHAPTER 6. USING THE CONFIGURATION API

The configuration tool exposes 4 endpoints that can be used to build, validate, bundle and deploy a configuration. The config-tool API is documented at https://github.com/quay/config-tool/blob/master/pkg/lib/editor/API.md. In this section, you will see how to use the API to retrieve the current configuration and how to validate any changes you make.

6.1. RETRIEVING THE DEFAULT CONFIGURATION

If you are running the configuration tool for the first time, and do not have an existing configuration, you can retrieve the default configuration. Start the container in config mode:

```
$ sudo podman run --rm -it --name quay_config \
  -p 8080:8080 \
  registry.redhat.io/quay/quay-rhel8:v3.6.8 config secret
```

Use the **config** endpoint of the configuration API to get the default:

```
$ curl -X GET -u quayconfig:secret http://quay-server:8080/api/v1/config  | jq
```

The value returned is the default configuration in JSON format:

```
{
  "config.yaml": {
    "AUTHENTICATION_TYPE": "Database",
    "AVATAR_KIND": "local",
    "DB_CONNECTION_ARGS": {
      "autorollback": true,
      "threadlocals": true
    },
    "DEFAULT_TAG_EXPIRATION": "2w",
    "EXTERNAL_TLS_TERMINATION": false,
    "FEATURE_ACTION_LOG_ROTATION": false,
    "FEATURE_ANONYMOUS_ACCESS": true,
    "FEATURE_APP_SPECIFIC_TOKENS": true,
    ....
  }

}
```

6.2. RETRIEVING THE CURRENT CONFIGURATION

If you have already configured and deployed the Quay registry, stop the container and restart it in configuration mode, loading the existing configuration as a volume:

```
$ sudo podman run --rm -it --name quay_config \
  -p 8080:8080 \
  -v $QUAY/config:/conf/stack:Z \
  registry.redhat.io/quay/quay-rhel8:v3.6.8 config secret
```

Use the **config** endpoint of the API to get the current configuration:

```
$ curl -X GET -u quayconfig:secret http://quay-server:8080/api/v1/config  | jq
```

The value returned is the current configuration in JSON format, including database and Redis configuration data:

```
{
  "config.yaml": {
    ....
    "BROWSER_API_CALLS_XHR_ONLY": false,
    "BUILDLOGS_REDIS": {
      "host": "quay-server",
      "password": "strongpassword",
      "port": 6379
    },
    "DATABASE_SECRET_KEY": "4b1c5663-88c6-47ac-b4a8-bb594660f08b",
    "DB_CONNECTION_ARGS": {
      "autorollback": true,
      "threadlocals": true
    },
    "DB_URI": "postgresql://quayuser:quaypass@quay-server:5432/quay",
    "DEFAULT_TAG_EXPIRATION": "2w",
    ....


  }

}
```

6.3. VALIDATING CONFIGURATION USING THE API

You can validate a configuration by posting it to the **config/validate** endpoint:

```
curl -u quayconfig:secret --header 'Content-Type: application/json' --request POST --data '
{
  "config.yaml": {
    ....
    "BROWSER_API_CALLS_XHR_ONLY": false,
    "BUILDLOGS_REDIS": {
      "host": "quay-server",
      "password": "strongpassword",
      "port": 6379
    },
    "DATABASE_SECRET_KEY": "4b1c5663-88c6-47ac-b4a8-bb594660f08b",
    "DB_CONNECTION_ARGS": {
      "autorollback": true,
      "threadlocals": true
    },
    "DB_URI": "postgresql://quayuser:quaypass@quay-server:5432/quay",
    "DEFAULT_TAG_EXPIRATION": "2w",
    ....


  }

} http://quay-server:8080/api/v1/config/validate | jq
```

The returned value is an array containing the errors found in the configuration. If the configuration is valid, an empty array **[]** is returned.

6.4. DETERMINING THE REQUIRED FIELDS

You can determine the required fields by posting an empty configuration structure to the **config/validate** endpoint:

```
curl -u quayconfig:secret --header 'Content-Type: application/json' --request POST --data '
{
  "config.yaml": {
  }

} http://quay-server:8080/api/v1/config/validate | jq
```

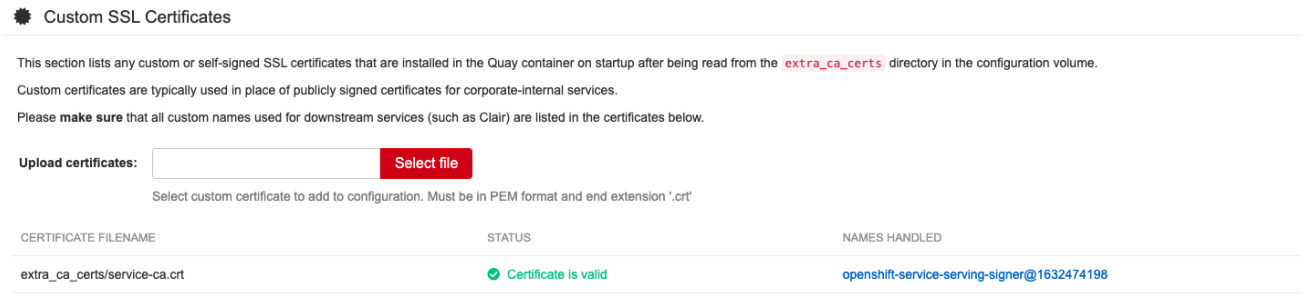
The value returned is an array indicating which fields are required:

```
[
  {
    "FieldGroup": "Database",
    "Tags": [
      "DB_URI"
    ],
    "Message": "DB_URI is required."
  },
  {
    "FieldGroup": "DistributedStorage",
    "Tags": [
      "DISTRIBUTED_STORAGE_CONFIG"
    ],
    "Message": "DISTRIBUTED_STORAGE_CONFIG must contain at least one storage location."
  },
  {
    "FieldGroup": "HostSettings",
    "Tags": [
      "SERVER_HOSTNAME"
    ],
    "Message": "SERVER_HOSTNAME is required"
  },
  {
    "FieldGroup": "HostSettings",
    "Tags": [
      "SERVER_HOSTNAME"
    ],
    "Message": "SERVER_HOSTNAME must be of type Hostname"
  },
  {
    "FieldGroup": "Redis",
    "Tags": [
      "BUILDLOGS_REDIS"
    ],
    "Message": "BUILDLOGS_REDIS is required"
  }
]
```

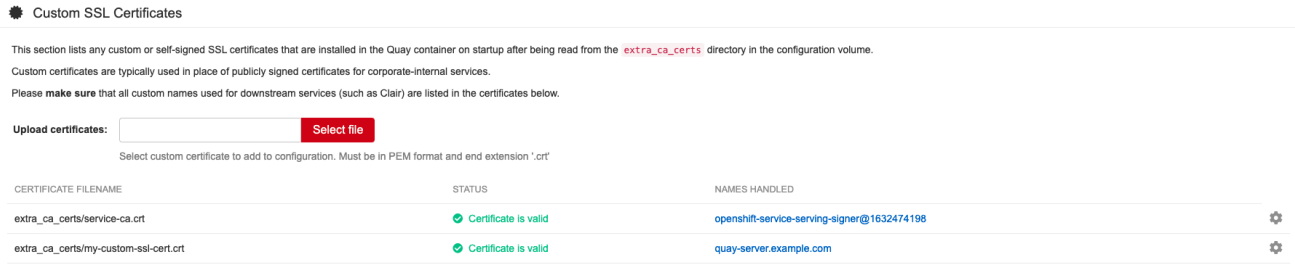
CHAPTER 7. USING THE CONFIGURATION TOOL

7.1. CUSTOM SSL CERTIFICATES UI

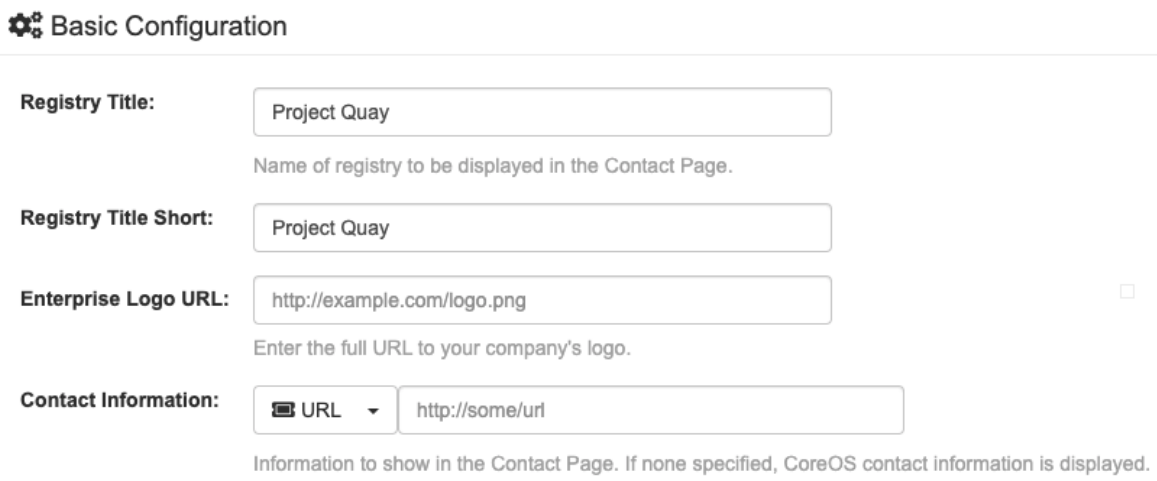
The config tool can be used to load custom certificates to facilitate access to resources such as external databases. Select the custom certs to be uploaded, ensuring that they are in PEM format, with an extension **.crt**.



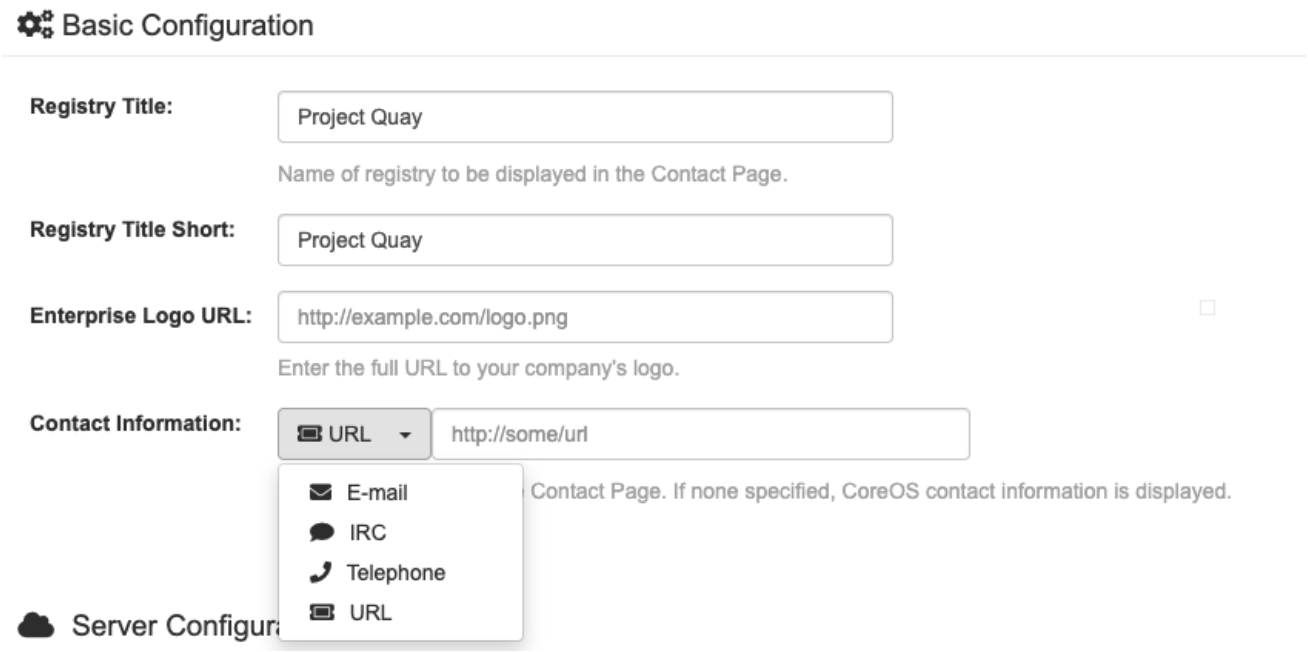
The config tool also displays a list of any uploaded certificates. Once you upload your custom SSL cert, it will appear in the list:



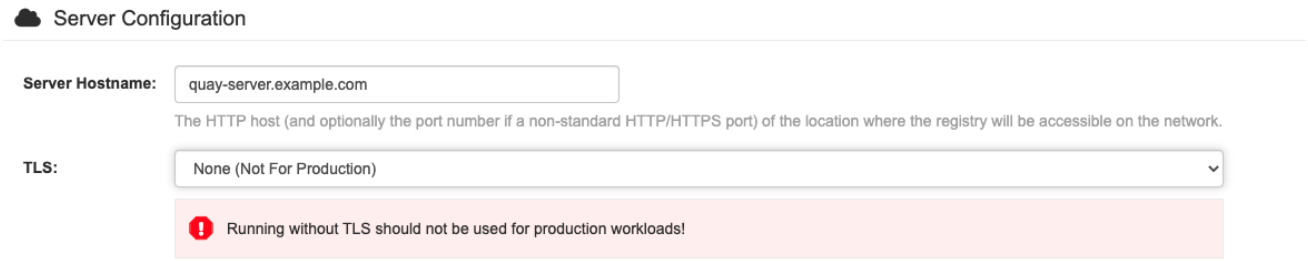
7.2. BASIC CONFIGURATION



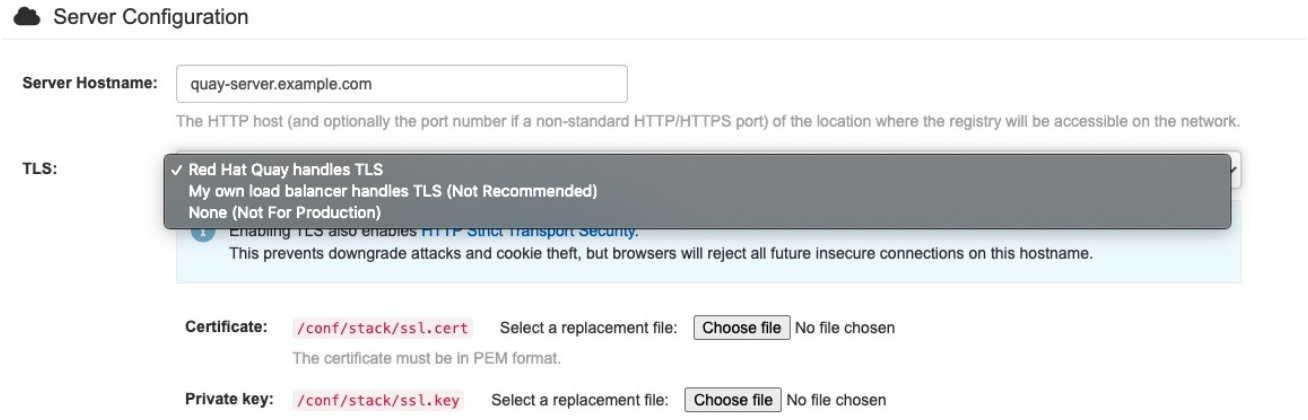
7.2.1. Contact information



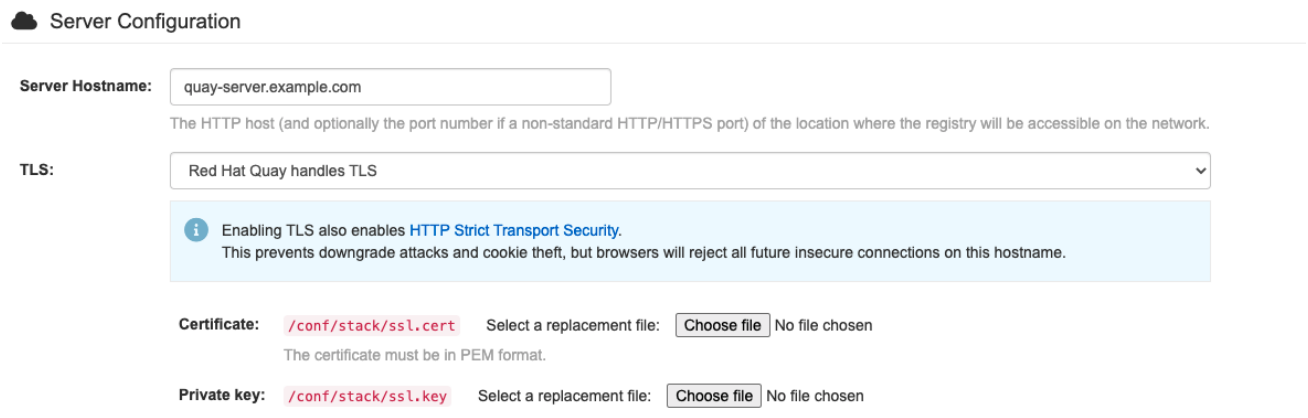
7.3. SERVER CONFIGURATION



7.3.1. Server configuration choice

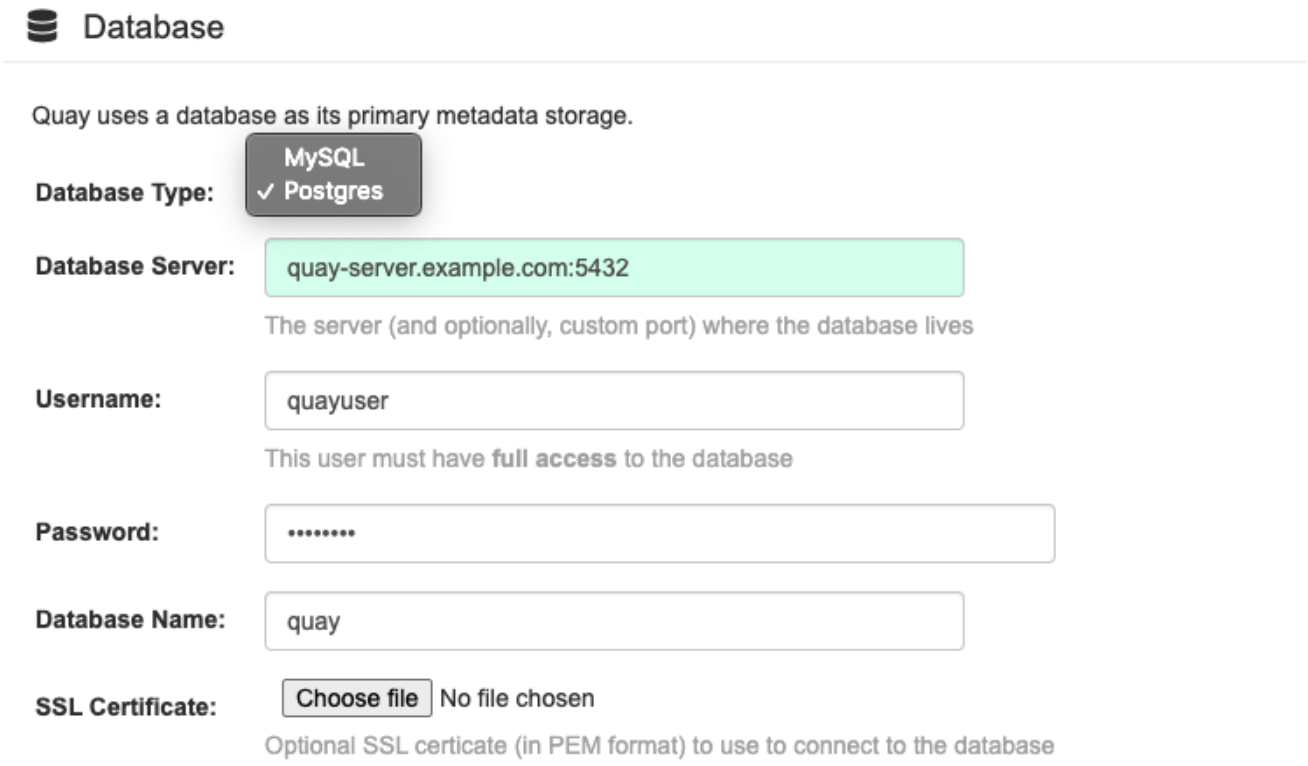


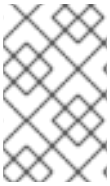
7.3.2. TLS configuration



7.4. DATABASE CONFIGURATION

You can choose between PostGreSQL and MySQL:



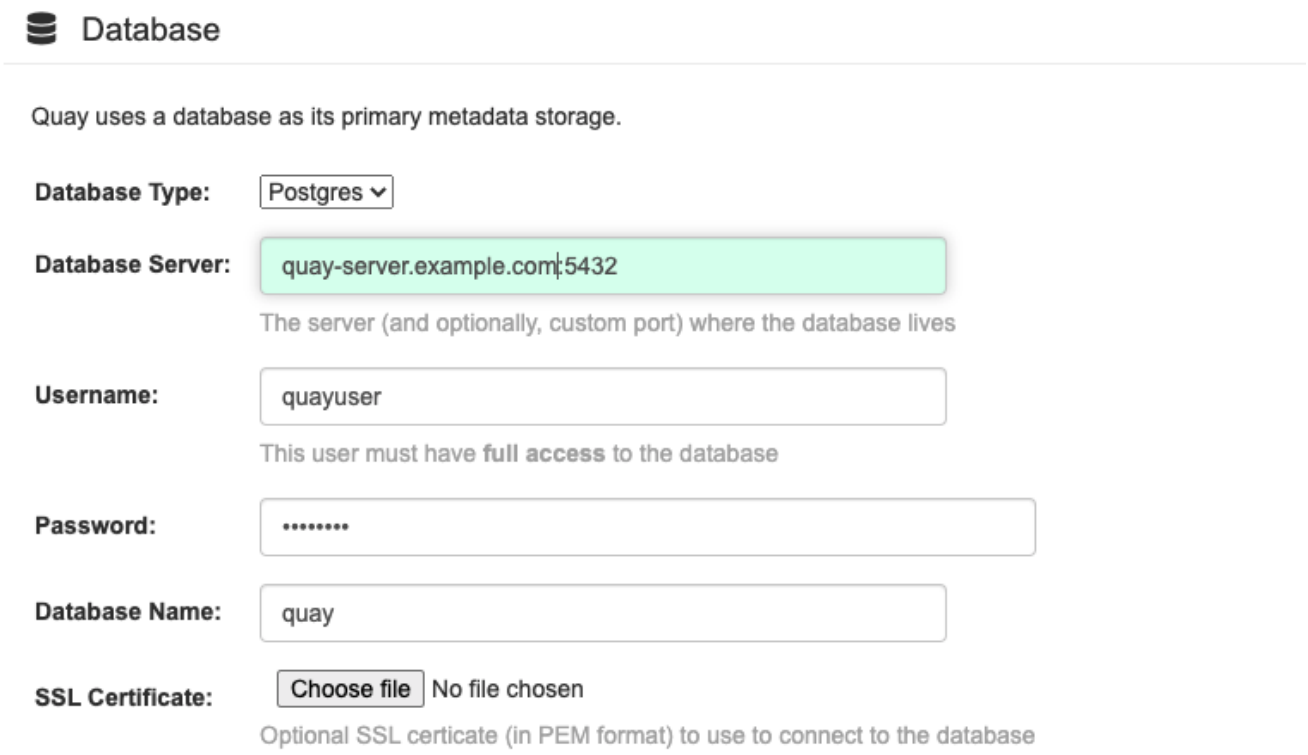


NOTE

The MySQL and MariaDB databases have been deprecated as of Red Hat Quay 3.6. Support for these databases will be removed in a future version of Red Hat Quay. If starting a new Red Hat Quay installation, it is strongly recommended to use PostgreSQL.

7.4.1. PostgreSQL configuration

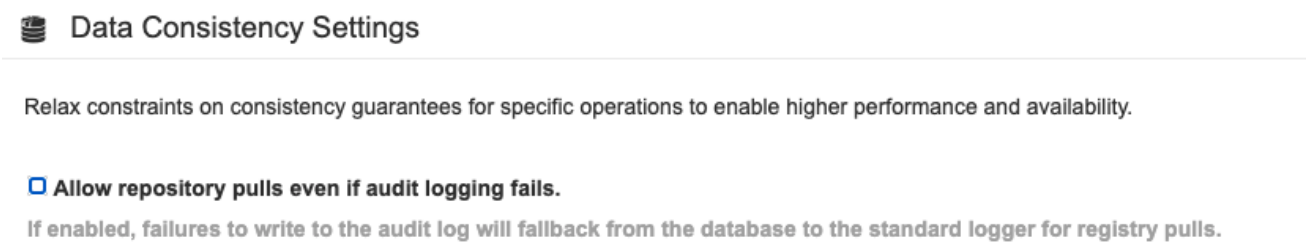
Enter the details for connecting to the database:



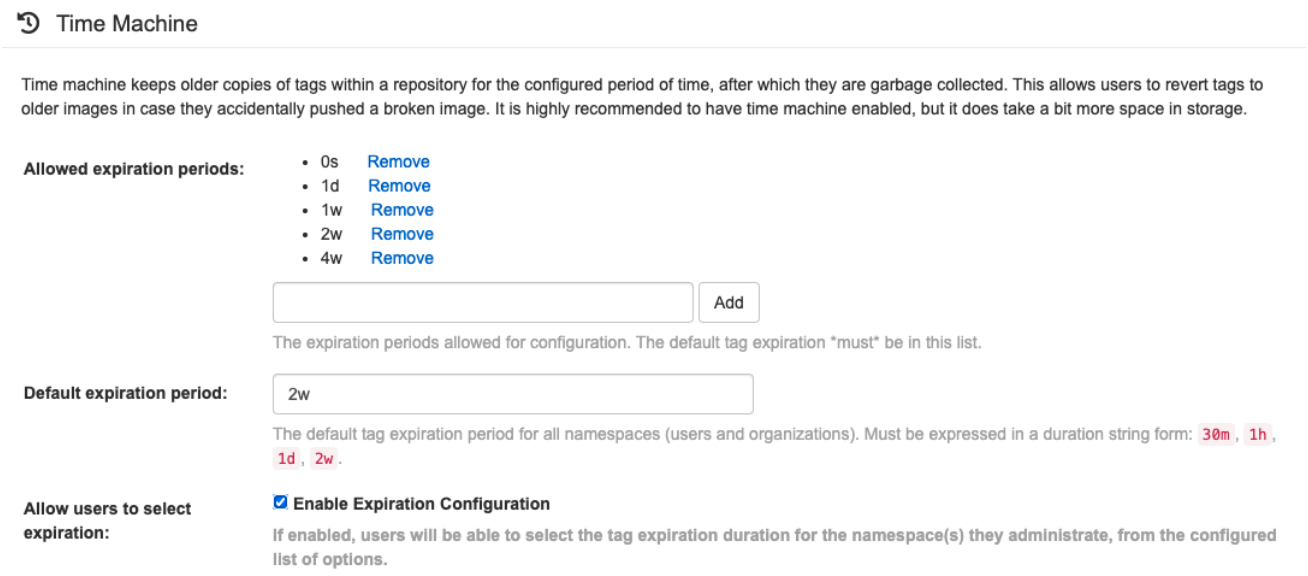
This will generate a DB_URI field of the form **postgresql://quayuser:quaypass@quay-server.example.com:5432/quay**.

If you need finer-grained control of the connection arguments, see the section "Database connection arguments" in the Configuration Guide.

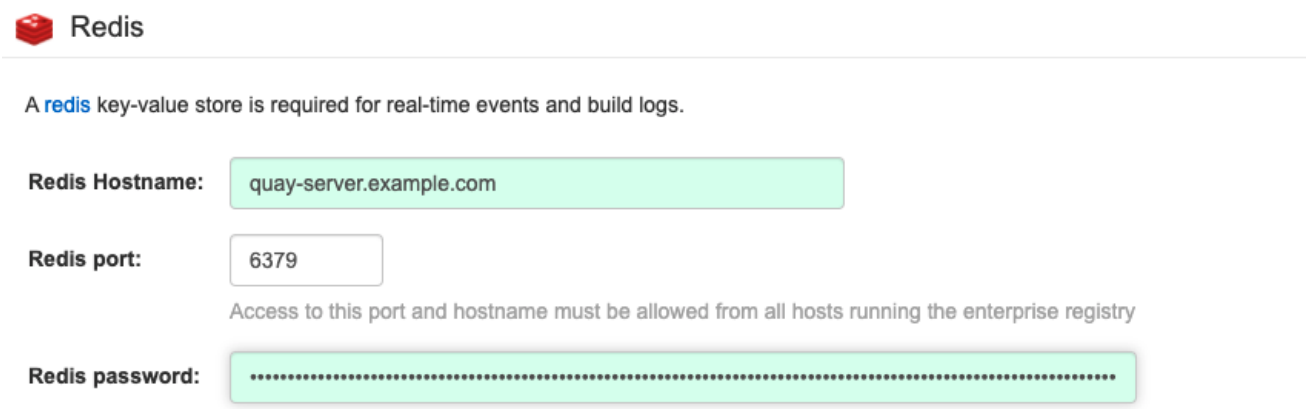
7.5. DATA CONSISTENCY



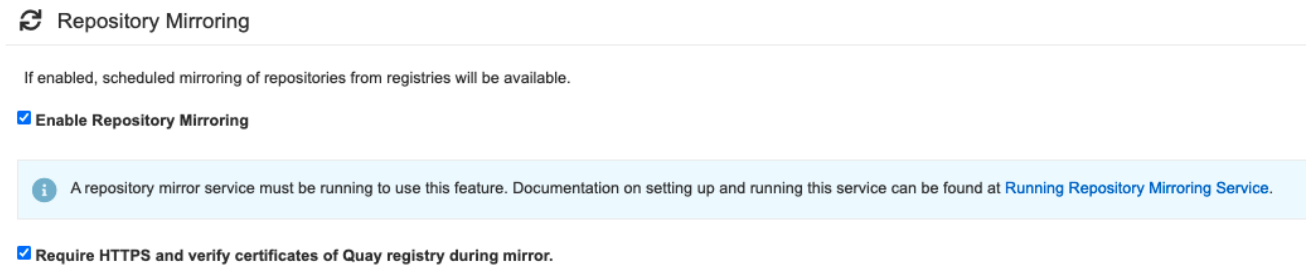
7.6. TIME MACHINE CONFIGURATION



7.7. REDIS CONFIGURATION



7.8. REPOSITORY MIRRORING CONFIGURATION



7.9. REGISTRY STORAGE CONFIGURATION

- Proxy storage
- Storage georeplication
- Storage engines

7.9.1. Enable storage replication

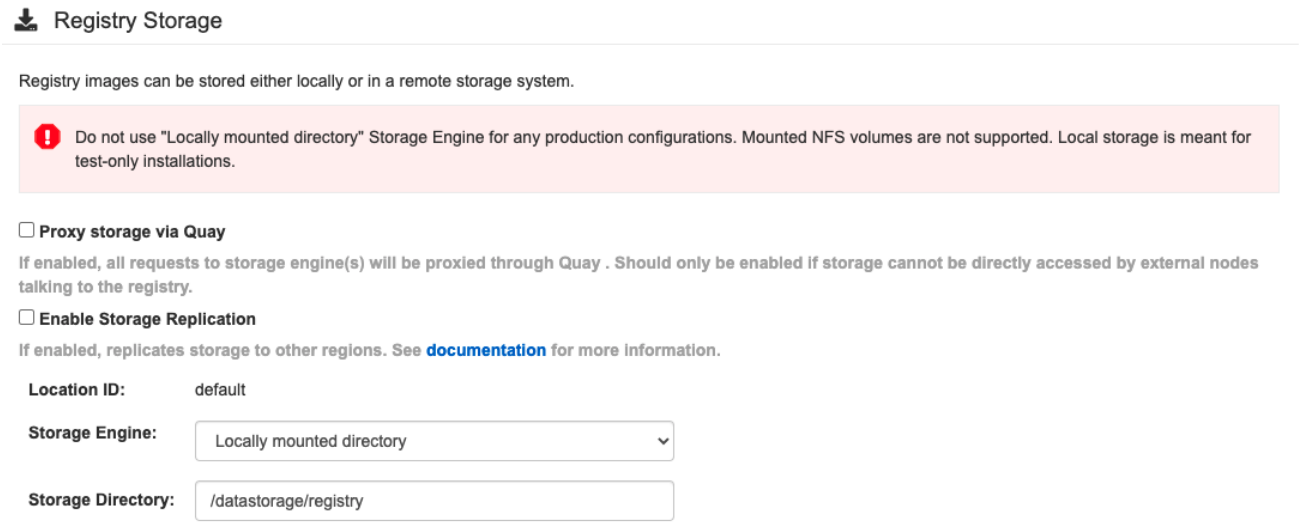
1. Scroll down to the section entitled **Registry Storage**.
2. Click **Enable Storage Replication**.
3. Add each of the storage engines to which data will be replicated. All storage engines to be used must be listed.
4. If complete replication of all images to all storage engines is required, under each storage engine configuration click **Replicate to storage engine by default**. This will ensure that all images are replicated to that storage engine. To instead enable per-namespace replication, please contact support.
5. When you are done, click **Save Configuration Changes**. Configuration changes will take effect the next time Red Hat Quay restarts.
6. After adding storage and enabling "Replicate to storage engine by default" for Georeplications, you need to sync existing image data across all storage. To do this, you need to **oc exec** (or docker/kubectl exec) into the container and run:

```
# scl enable python27 bash
# python -m util.backfillreplication
```

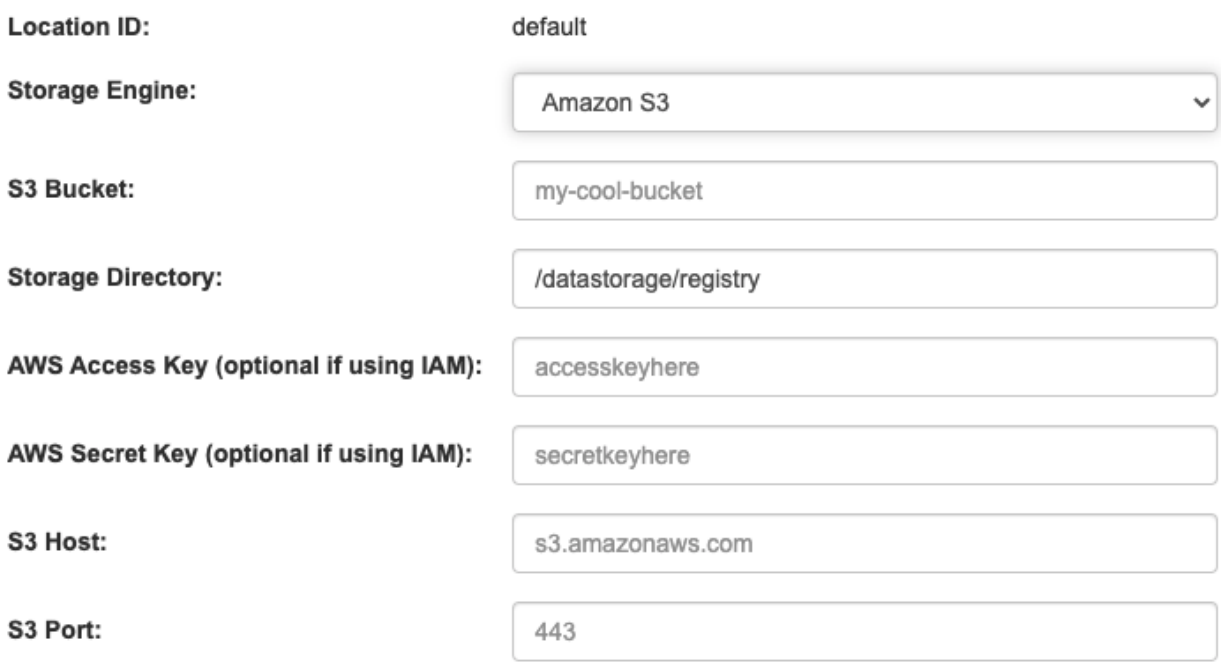
This is a one time operation to sync content after adding new storage.

7.9.2. Storage engines

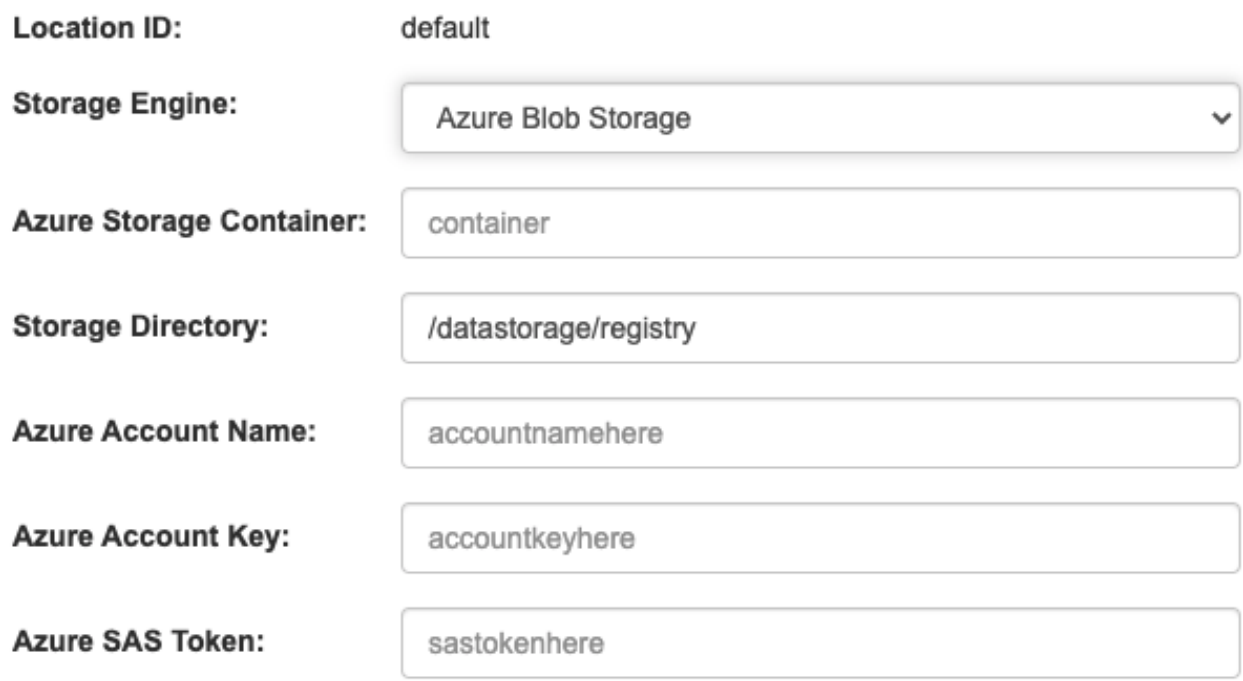
7.9.2.1. Local storage



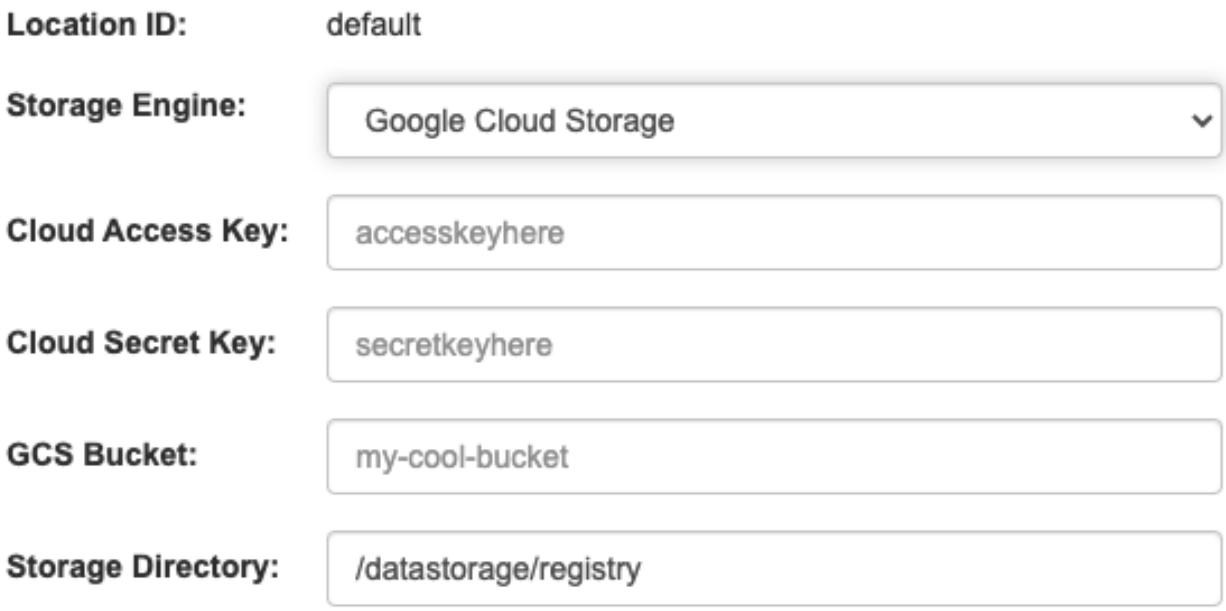
7.9.2.2. Amazon S3 storage



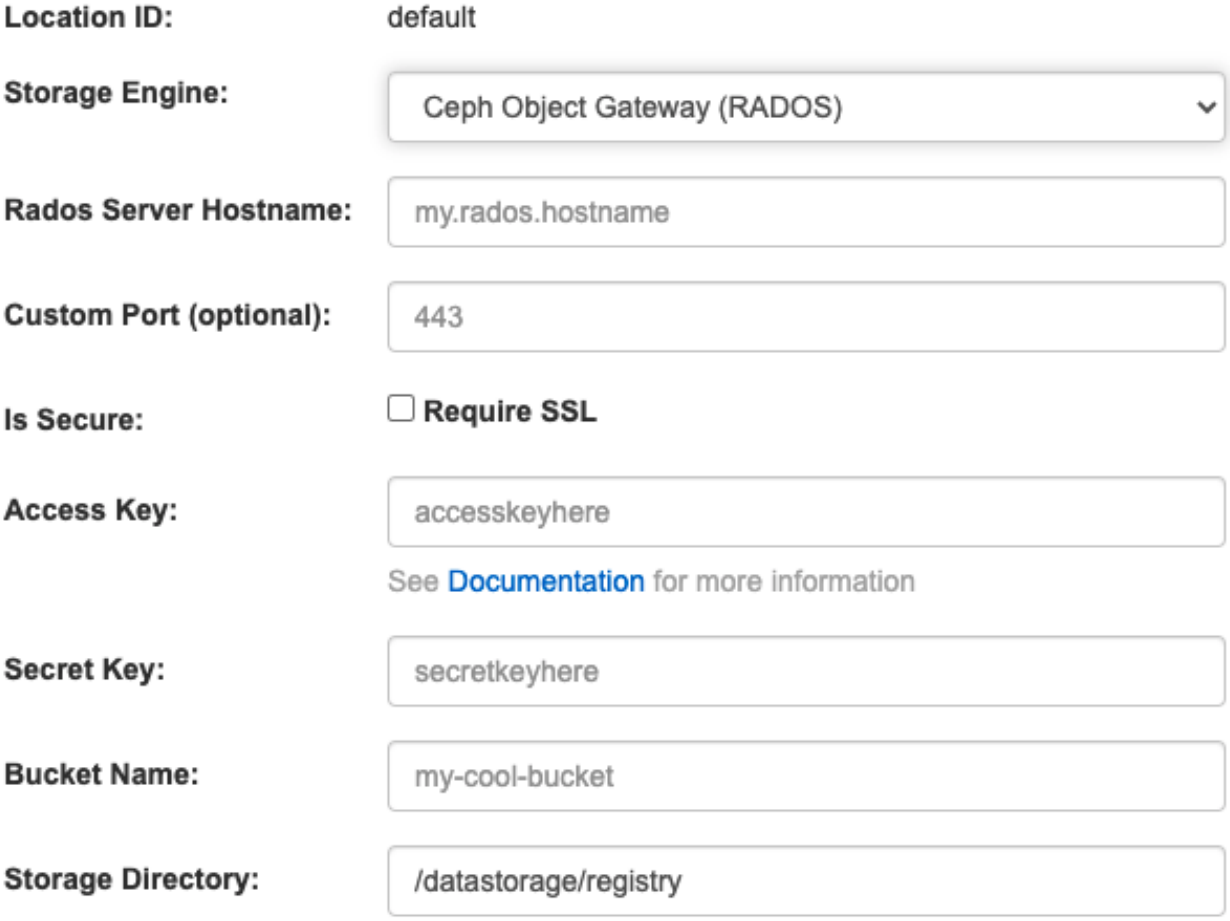
7.9.2.3. Azure blob storage



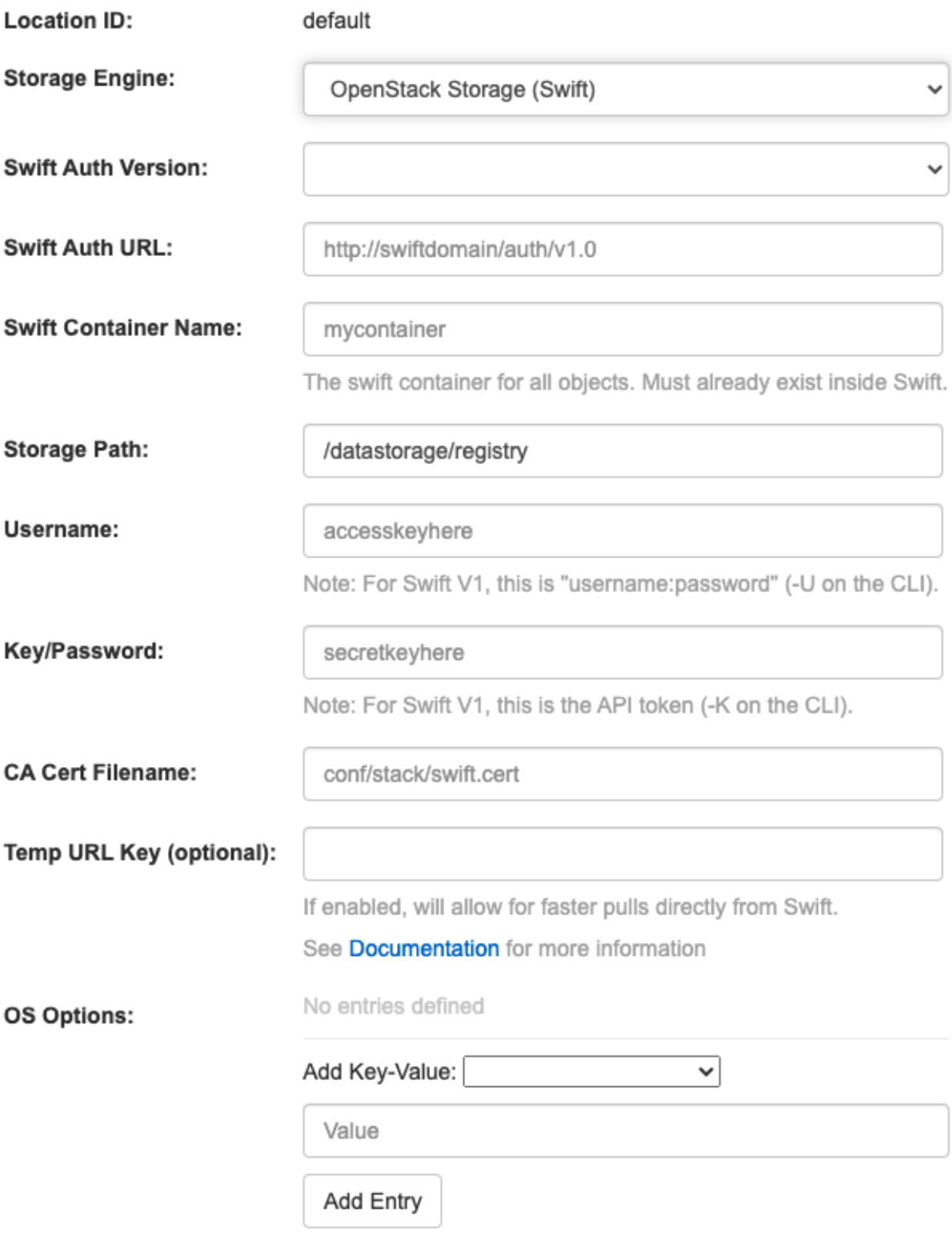
7.9.2.4. Google cloud storage



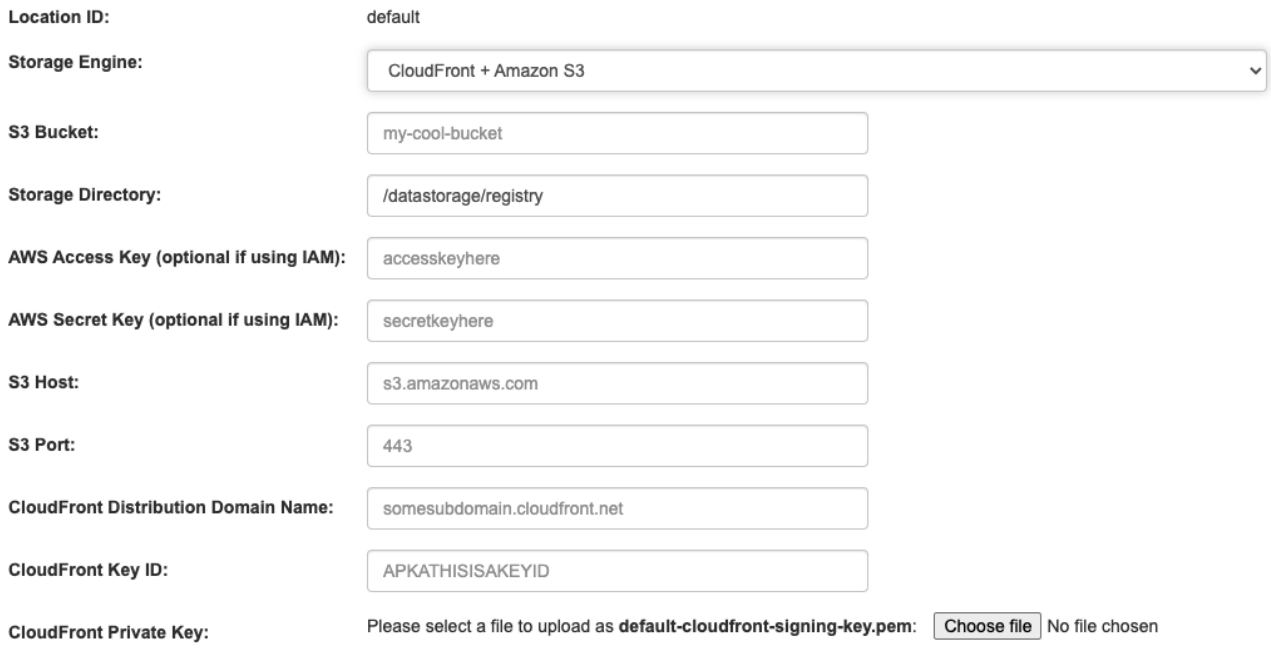
7.9.2.5. Ceph object gateway (RADOS) storage



7.9.2.6. OpenStack (Swift) storage configuration



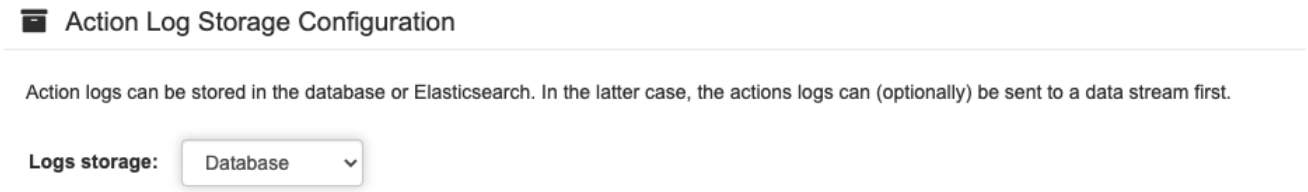
7.9.2.7. Cloudfront + Amazon S3 storage configuration



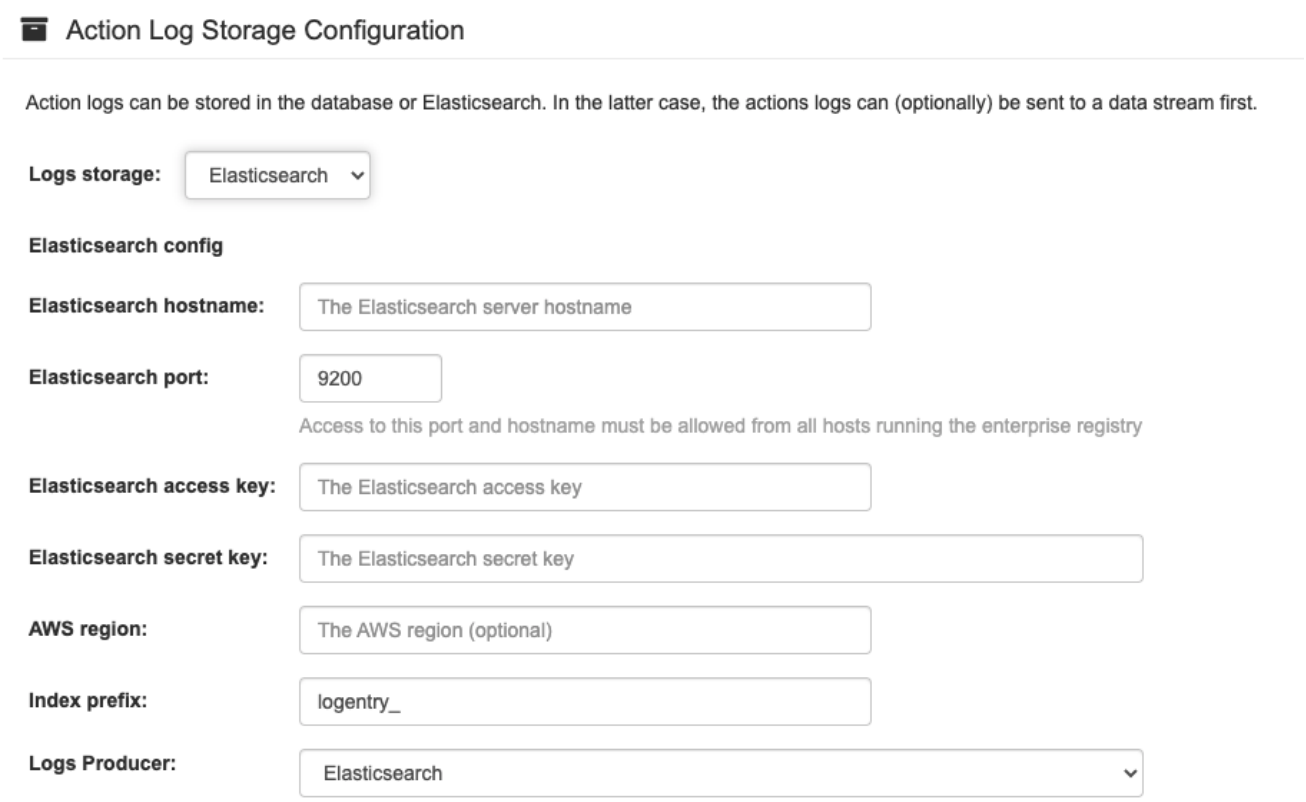
7.10. ACTION LOG CONFIGURATION

7.10.1. Action log storage configuration

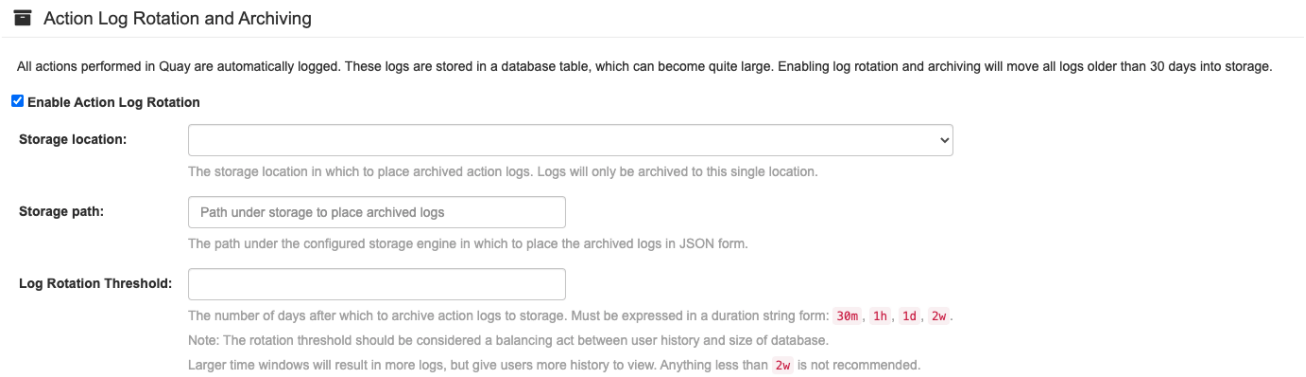
7.10.1.1. Database action log storage

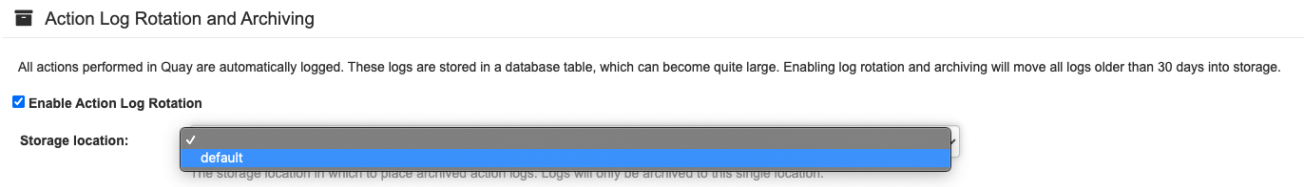


7.10.1.2. Elasticsearch action log storage

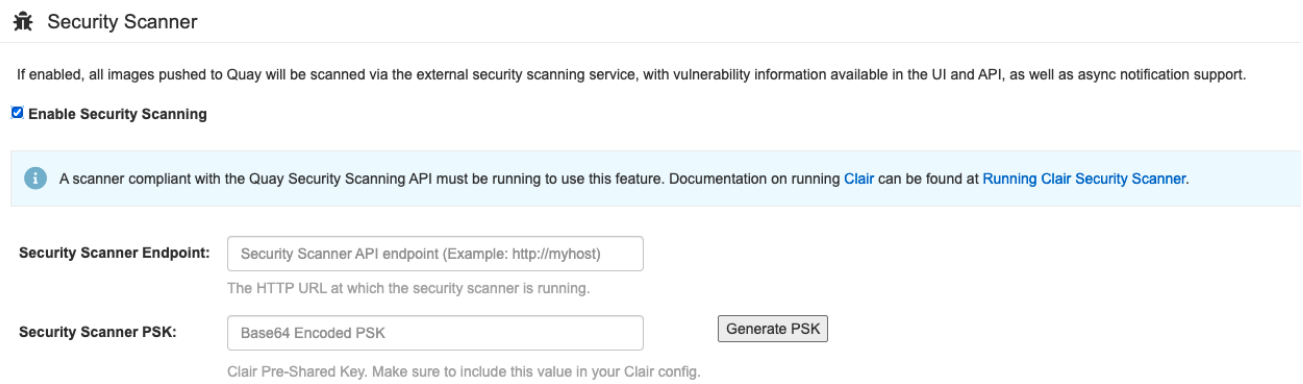


7.10.2. Action log rotation and archiving

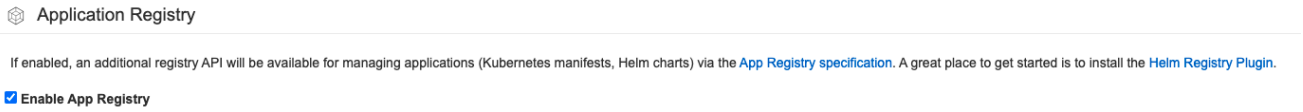




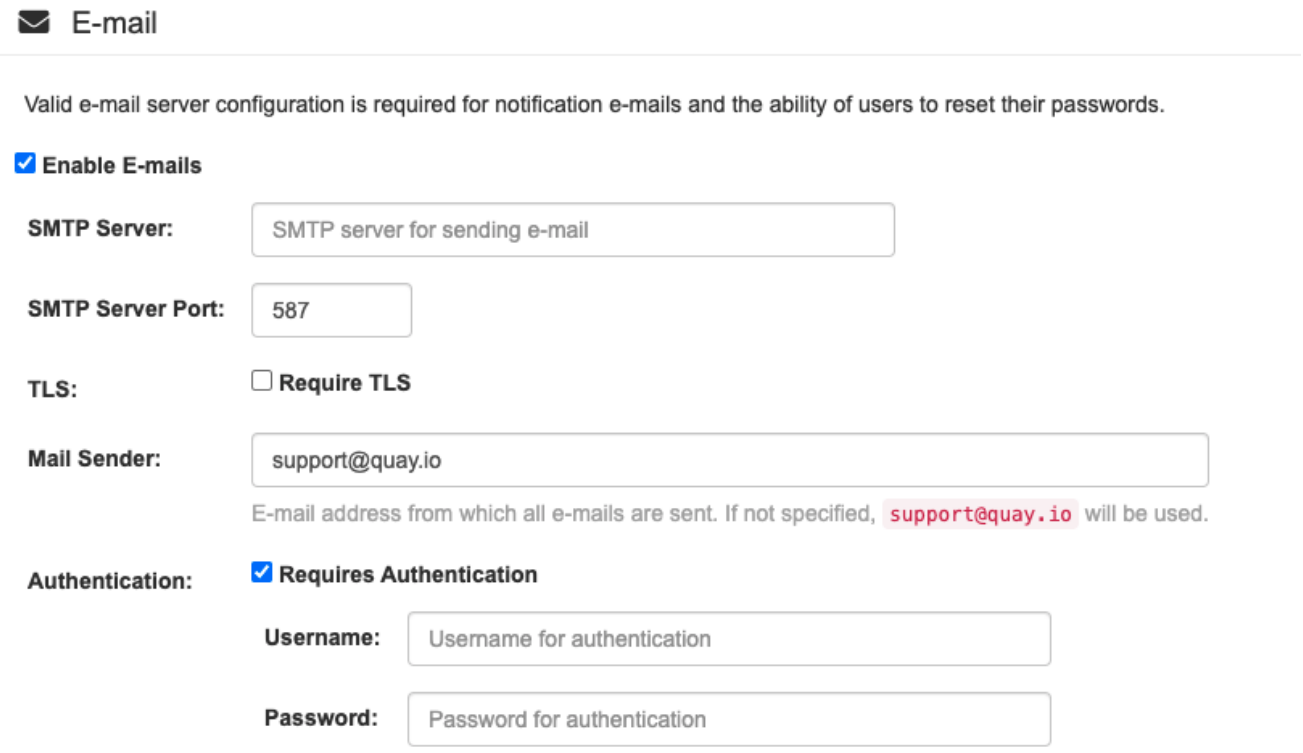
7.11. SECURITY SCANNER CONFIGURATION



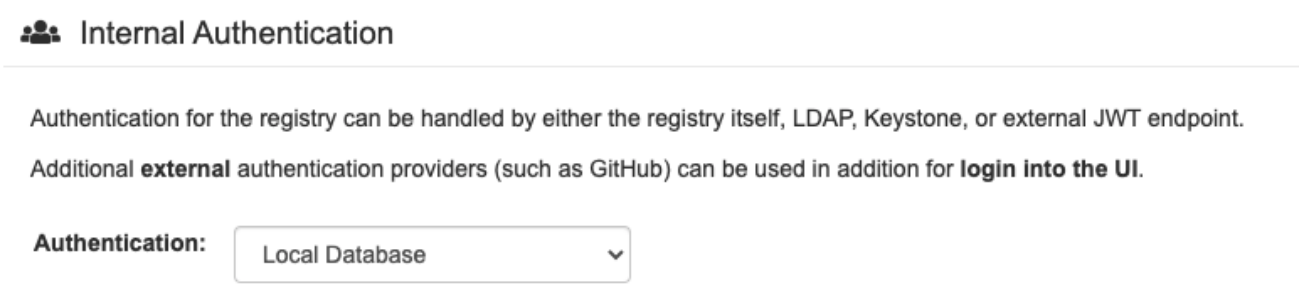
7.12. APPLICATION REGISTRY CONFIGURATION

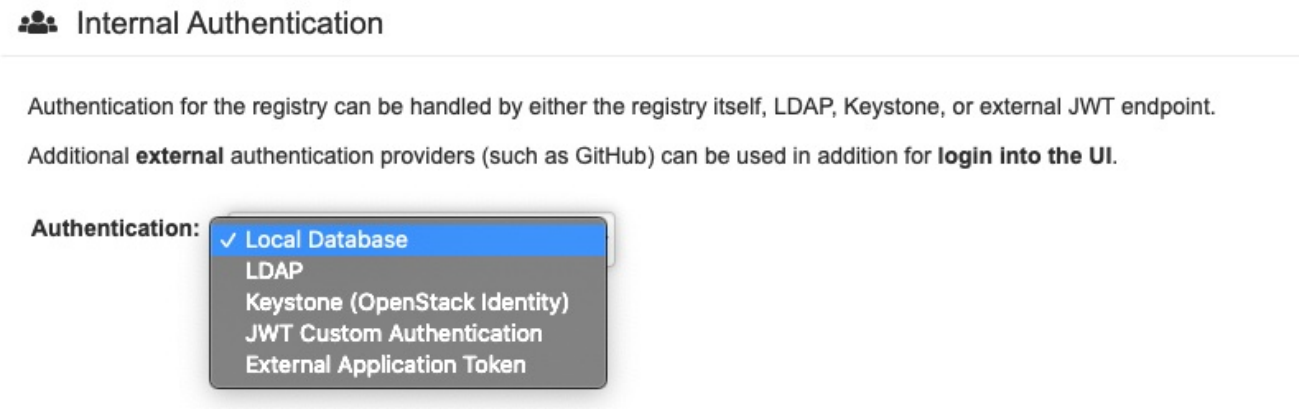


7.13. EMAIL CONFIGURATION

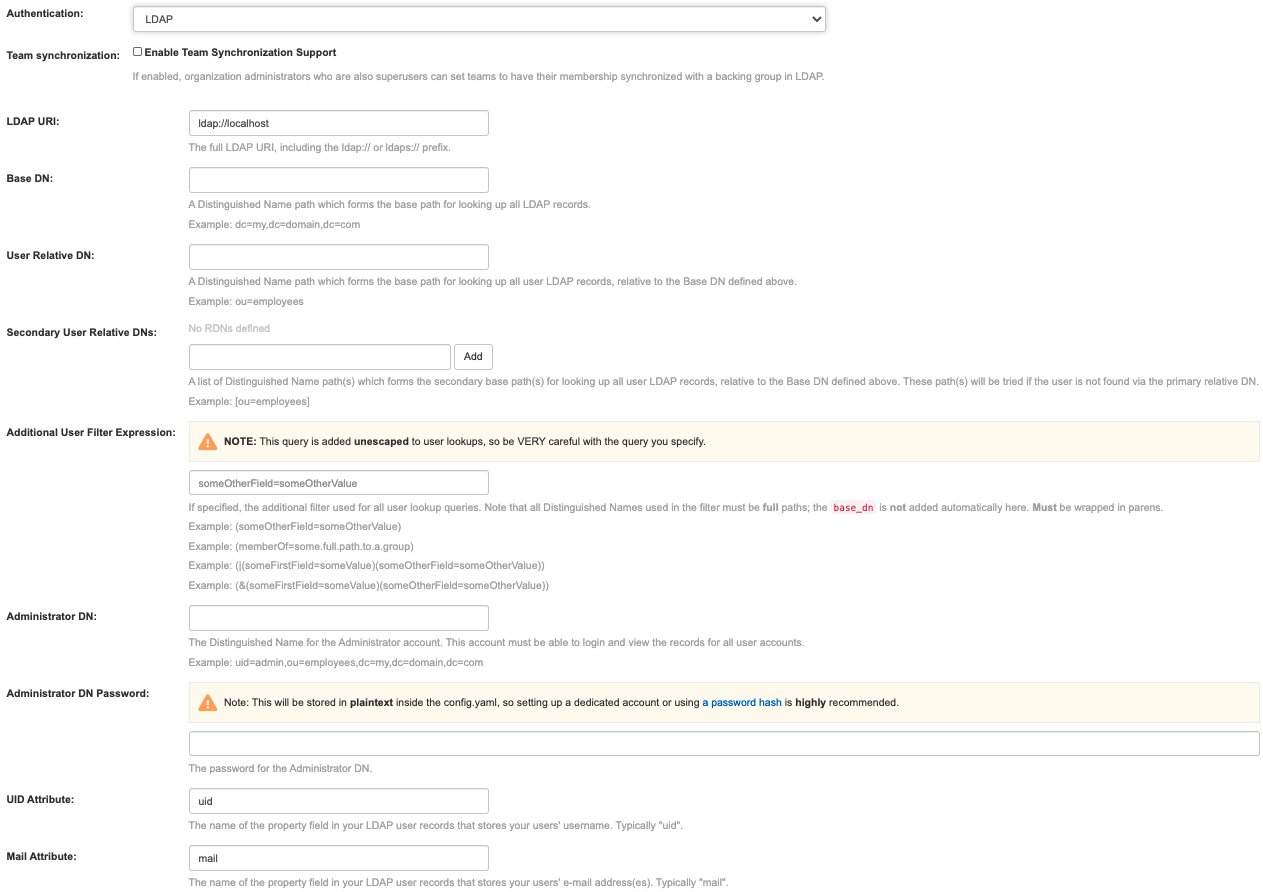


7.14. INTERNAL AUTHENTICATION CONFIGURATION

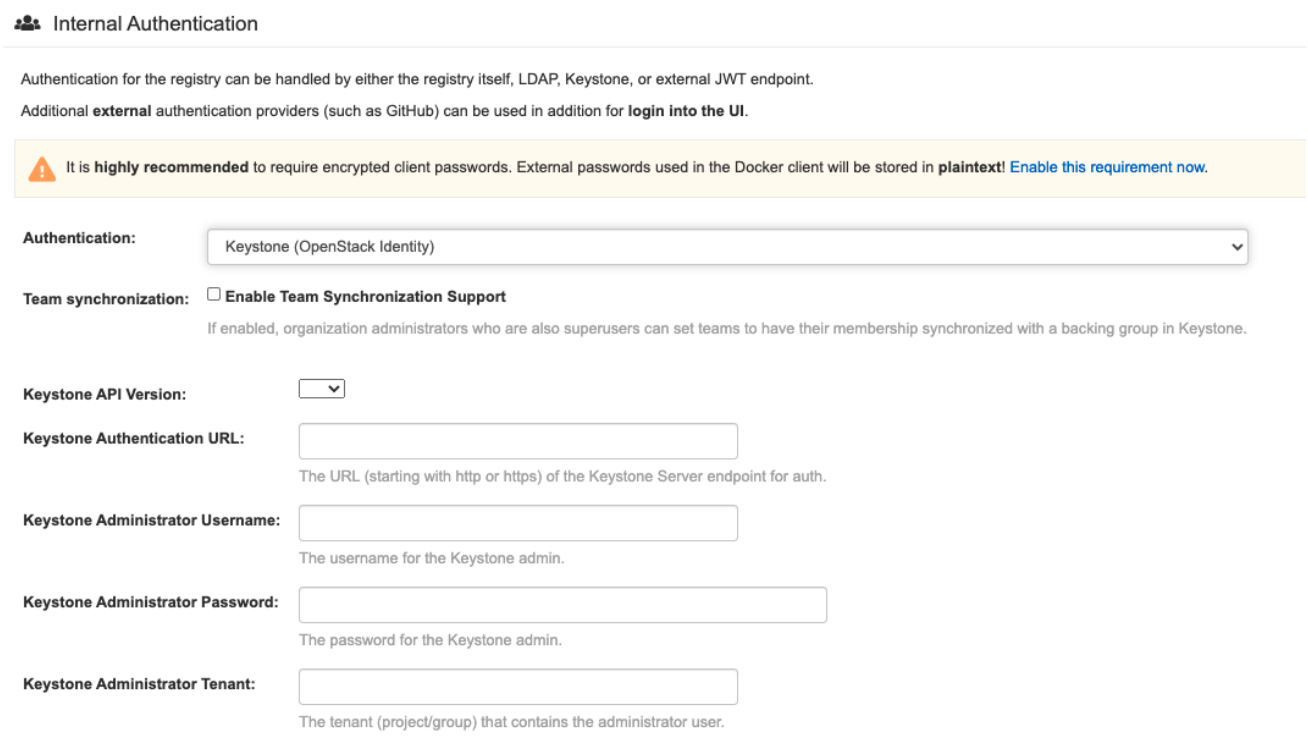




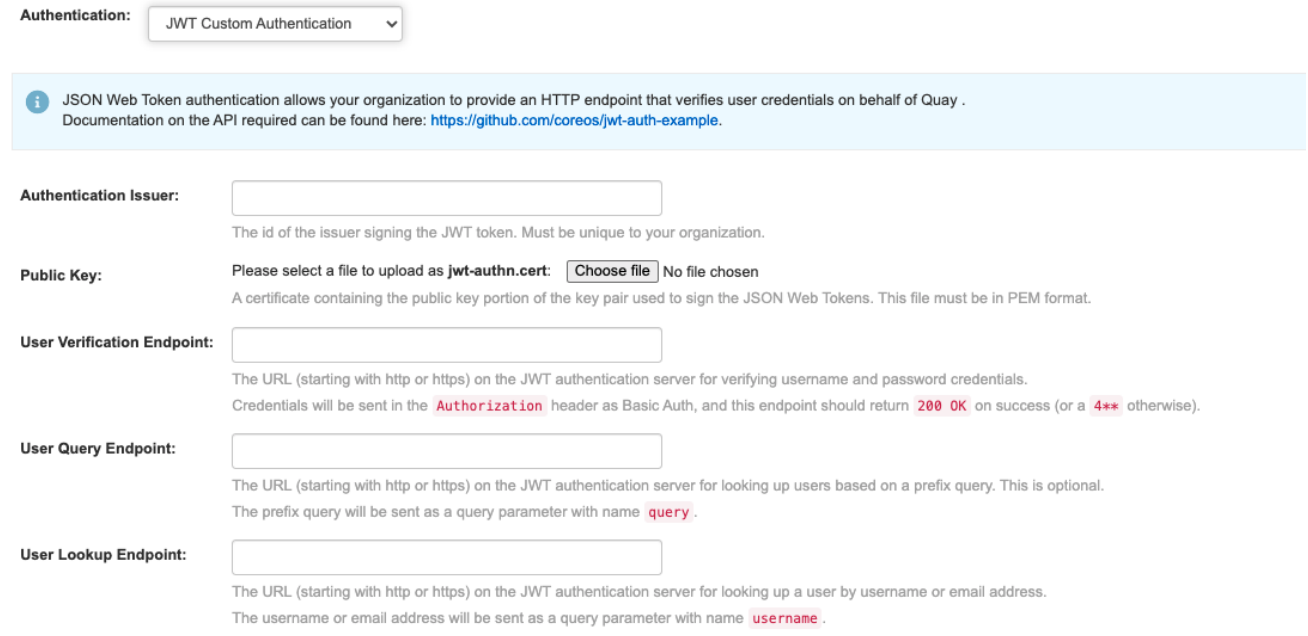
7.14.1. LDAP



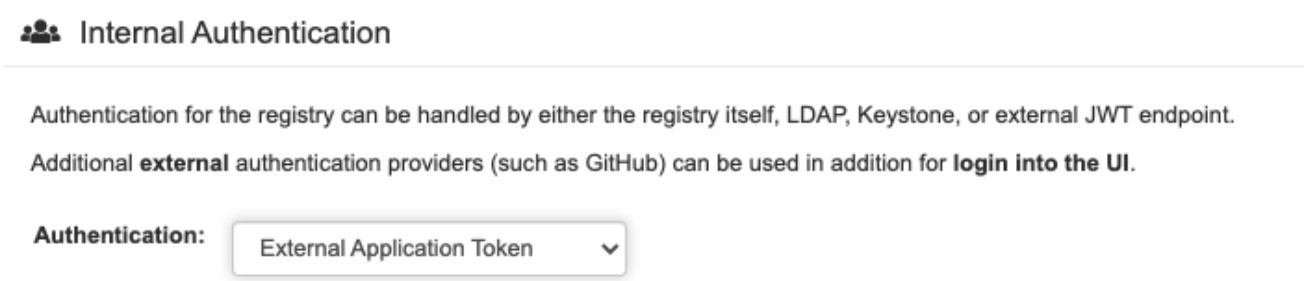
7.14.2. Keystone (OpenStack identity)



7.14.3. JWT custom authentication

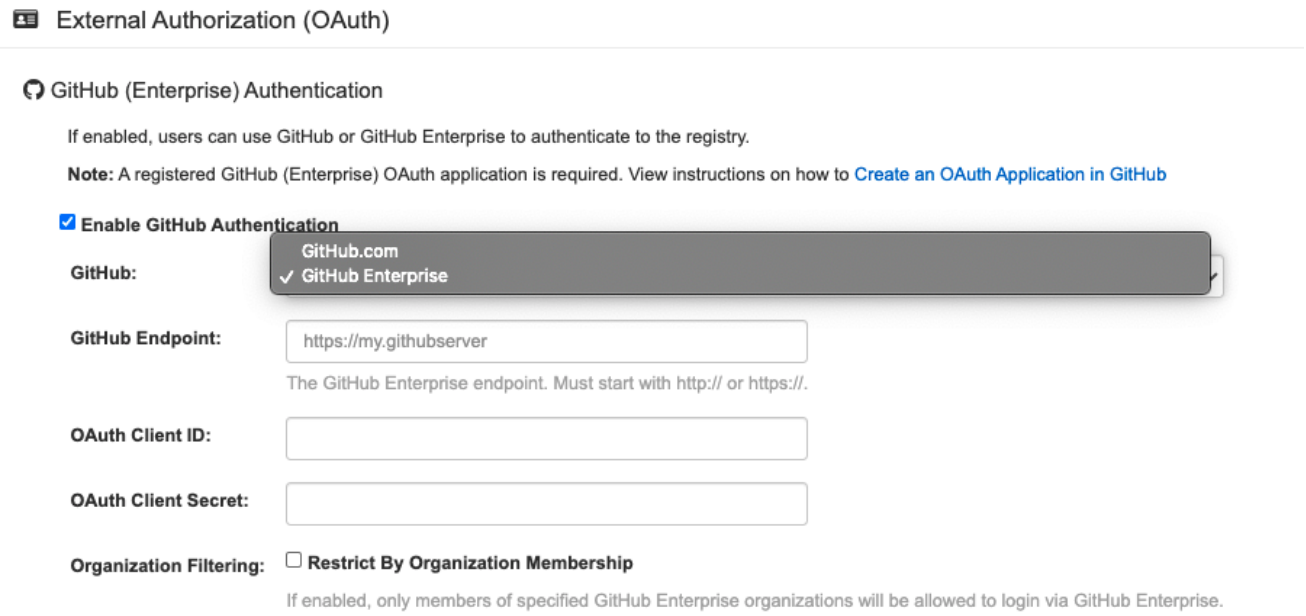


7.14.4. External application token

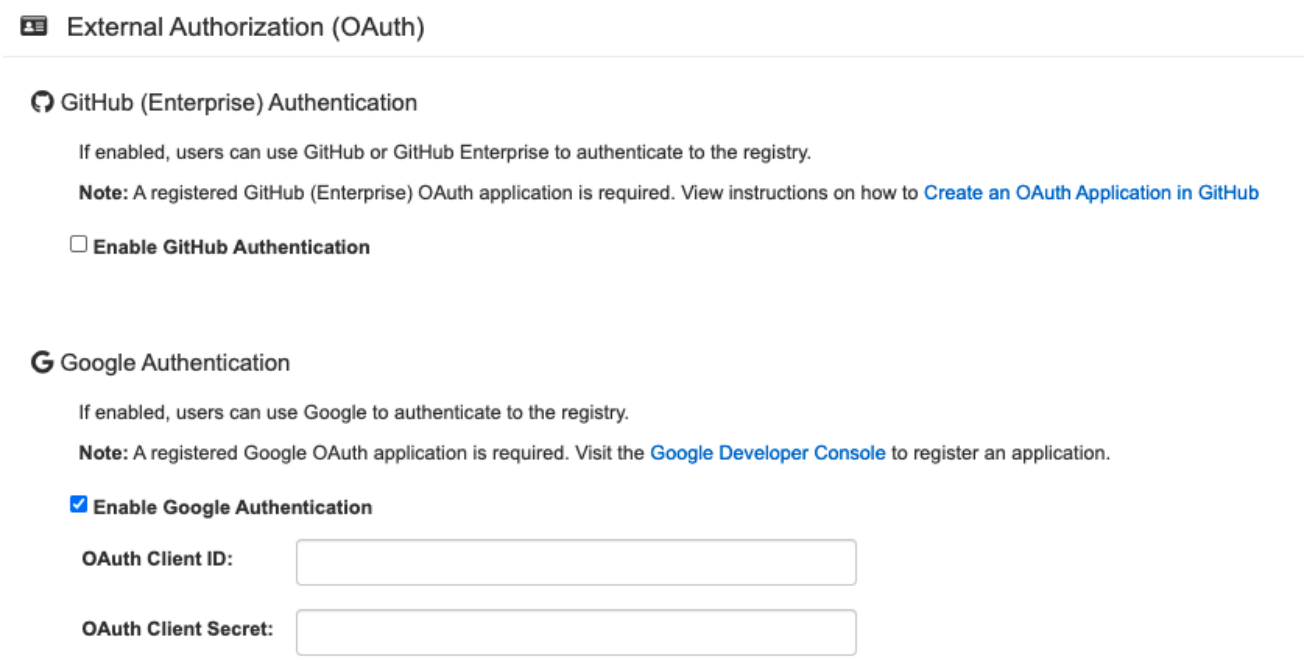


7.15. EXTERNAL AUTHENTICATION (OAUTH) CONFIGURATION

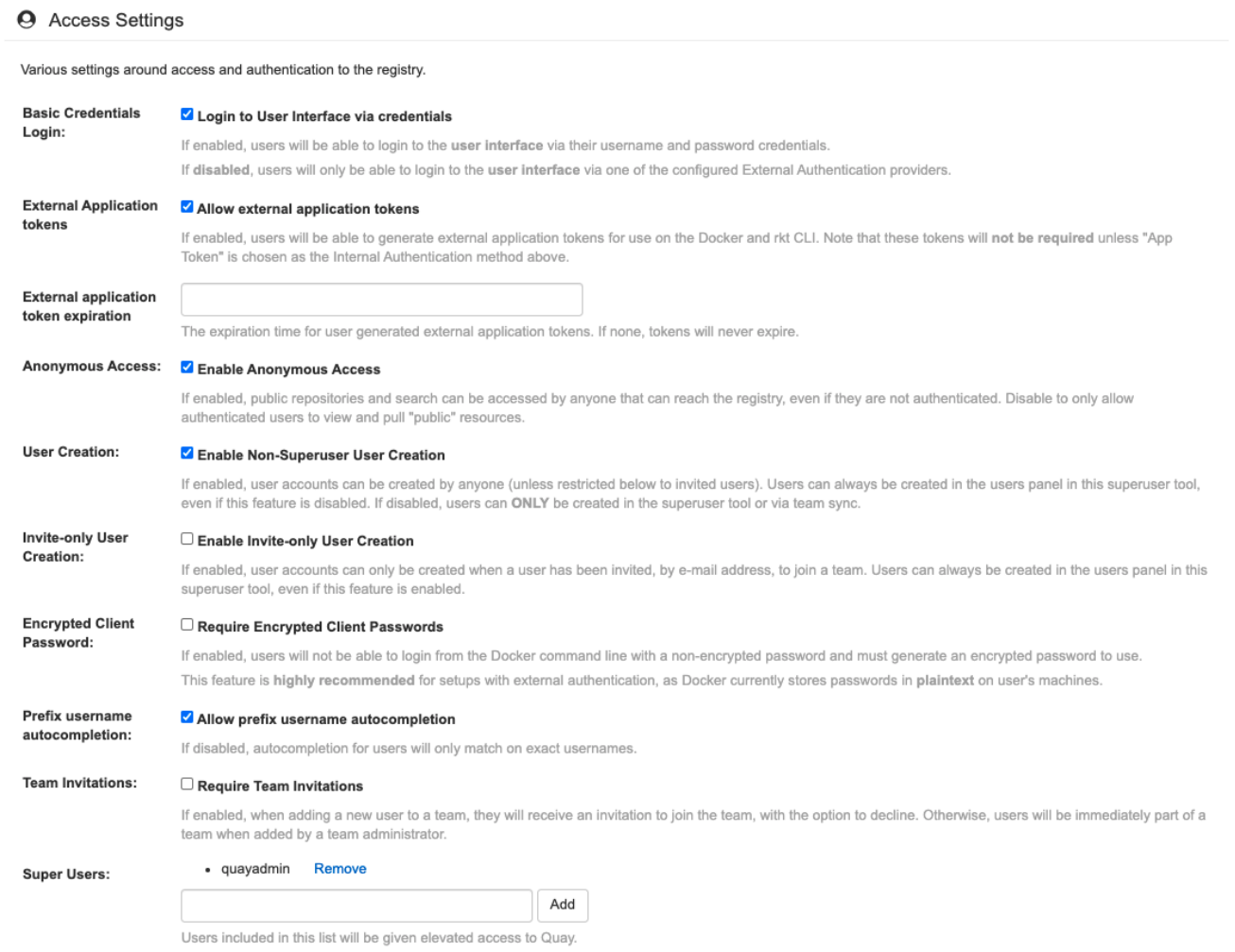
7.15.1. GitHub (Enterprise) authentication



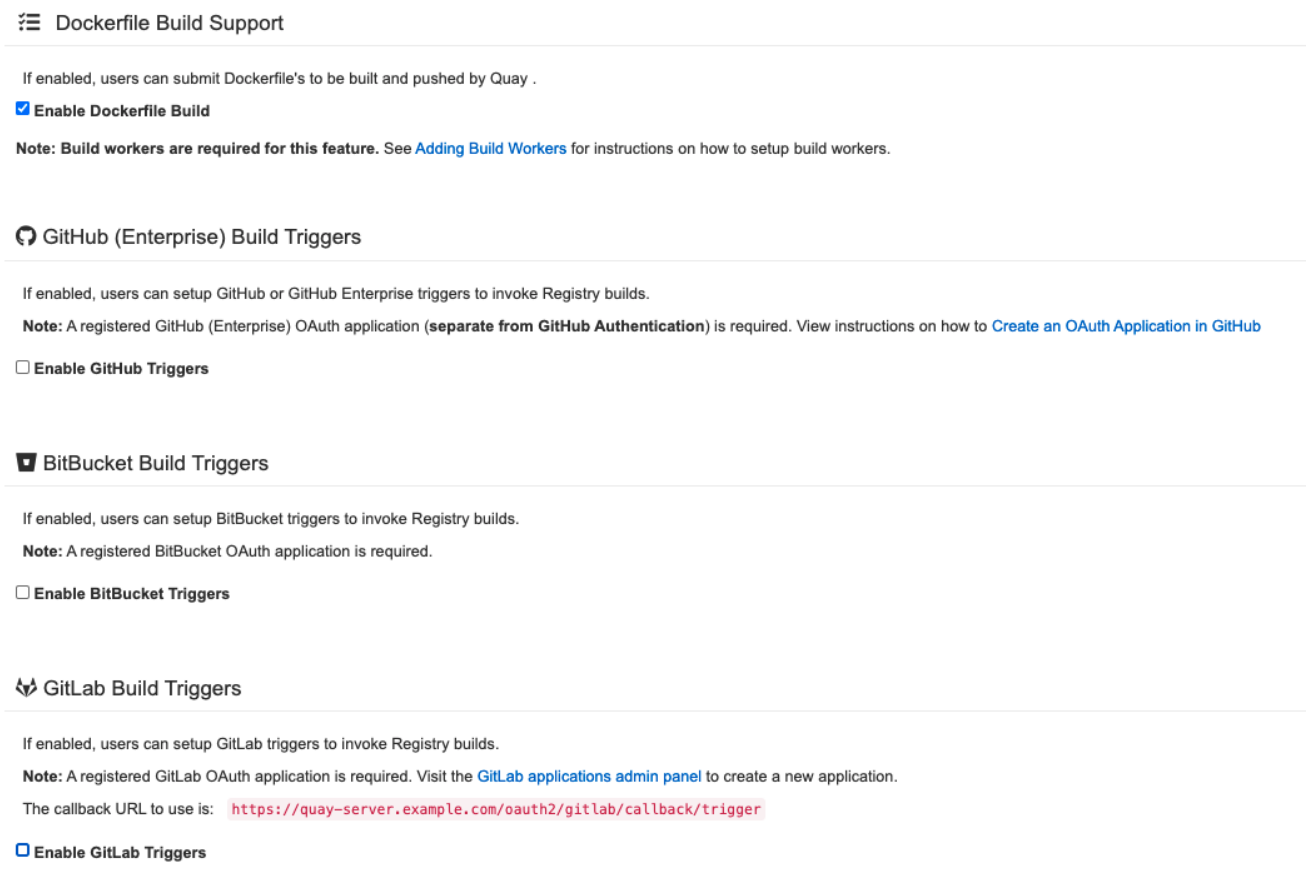
7.15.2. Google authentication



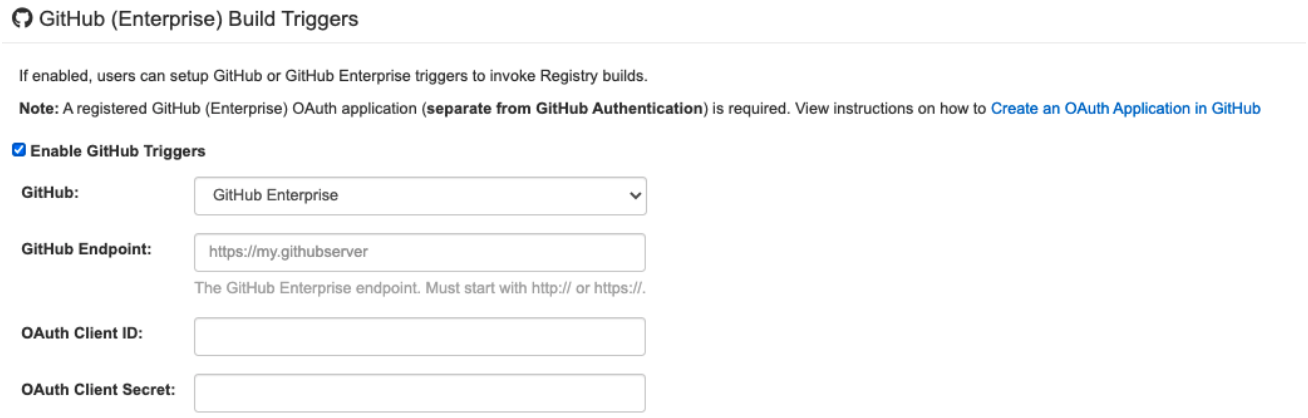
7.16. ACCESS SETTINGS CONFIGURATION



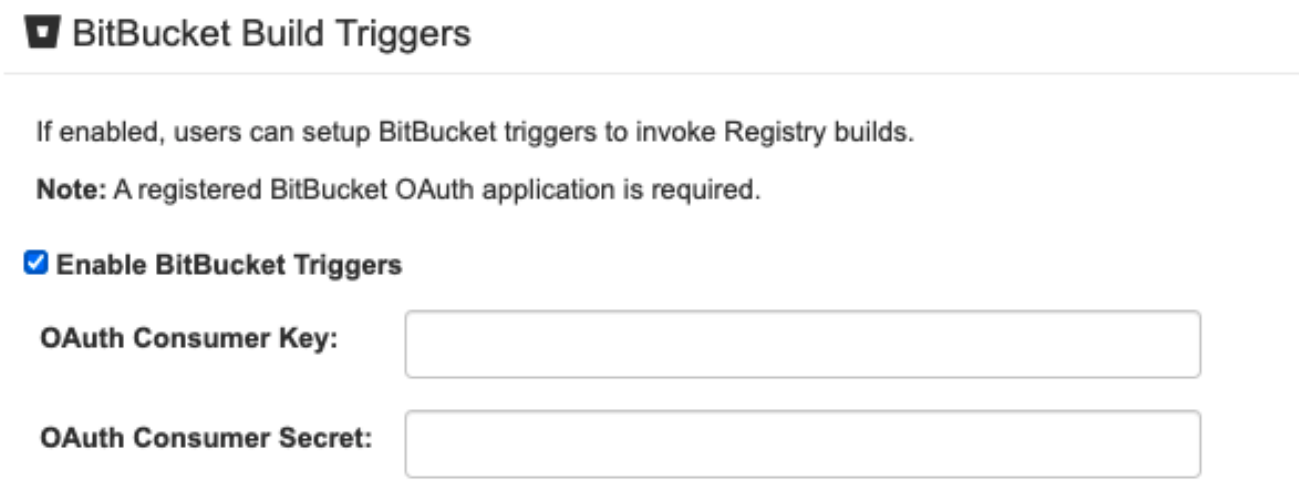
7.17. DOCKERFILE BUILD SUPPORT



7.17.1. GitHub (Enterprise) Build Triggers



7.17.2. BitBucket Build Triggers



7.17.3. GitLab Build Triggers