



## Red Hat Quay 3.7

### Configure Red Hat Quay

Customizing Red Hat Quay using configuration options



## Red Hat Quay 3.7 Configure Red Hat Quay

---

Customizing Red Hat Quay using configuration options

## Legal Notice

Copyright © 2023 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux<sup>®</sup> is the registered trademark of Linus Torvalds in the United States and other countries.

Java<sup>®</sup> is a registered trademark of Oracle and/or its affiliates.

XFS<sup>®</sup> is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL<sup>®</sup> is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js<sup>®</sup> is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack<sup>®</sup> Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

## Abstract

Configure Red Hat Quay

## Table of Contents

<b>CHAPTER 1. GETTING STARTED WITH RED HAT QUAY CONFIGURATION</b> .....	<b>6</b>
1.1. CONFIGURATION UPDATES FOR QUAY 3.7	6
1.1.1. New configuration fields for Red Hat Quay 3.7.7	6
1.1.2. New configuration fields	6
1.2. CONFIGURATION UPDATES FOR RED HAT QUAY 3.6	7
1.2.1. New configuration fields	7
1.2.2. Deprecated configuration fields	8
1.3. EDITING THE CONFIGURATION FILE	8
1.4. LOCATION OF CONFIGURATION FILE IN A STANDALONE DEPLOYMENT	8
1.5. MINIMAL CONFIGURATION	9
1.5.1. Sample minimal configuration file	9
1.5.2. Local storage	10
1.5.3. Cloud storage	10
<b>CHAPTER 2. CONFIGURATION FIELDS</b> .....	<b>11</b>
2.1. REQUIRED CONFIGURATION FIELDS	11
2.2. AUTOMATION OPTIONS	11
2.3. OPTIONAL CONFIGURATION FIELDS	11
2.4. GENERAL REQUIRED FIELDS	12
2.5. DATABASE CONFIGURATION	13
2.5.1. Database URI	13
2.5.2. Database connection arguments	13
2.5.2.1. PostgreSQL SSL connection arguments	14
2.5.2.2. MySQL SSL connection arguments	15
2.6. IMAGE STORAGE	15
2.6.1. Image storage features	15
2.6.2. Image storage configuration fields	16
2.6.3. Local storage	16
2.6.4. OCS/NooBaa	17
2.6.5. Ceph / RadosGW Storage / Hitachi HCP	17
2.6.6. AWS S3 storage	17
2.6.7. Google Cloud Storage	18
2.6.8. Azure Storage	18
2.6.9. Swift storage	18
2.7. REDIS CONFIGURATION FIELDS	19
2.7.1. Build logs	19
2.7.2. User events	19
2.7.3. Example Redis configuration	20
2.8. MODELCACHE CONFIGURATION OPTIONS	20
2.8.1. Memcache configuration option	20
2.8.2. Single Redis configuration option	21
2.8.3. Clustered Redis configuration option	21
2.9. TAG EXPIRATION CONFIGURATION FIELDS	21
2.9.1. Example tag expiration configuration	22
2.10. PRE-CONFIGURING RED HAT QUAY FOR AUTOMATION	22
2.10.1. Allowing the API to create the first user	22
2.10.2. Enabling general API access	23
2.10.3. Adding a super user	23
2.10.4. Restricting user creation	23
2.10.5. Enabling new functionality	23
2.10.6. Suggested configuration for automation	23

2.10.7. Deploying the Red Hat Quay Operator using the initial configuration	24
2.10.8. Using the API to deploy Red Hat Quay	24
2.10.8.1. Using the API to create the first user	24
2.10.8.2. Using the OAuth token	25
2.10.8.3. Using the API to create an organization	26
2.11. BASIC CONFIGURATION FIELDS	27
2.12. SSL CONFIGURATION FIELDS	29
2.12.1. Configuring SSL	30
2.13. ADDING TLS CERTIFICATES TO THE RED HAT QUAY CONTAINER	31
2.13.1. Add TLS certificates to Red Hat Quay	31
2.14. LDAP CONFIGURATION FIELDS	31
2.14.1. LDAP configuration example	33
2.15. MIRRORING CONFIGURATION FIELDS	33
2.16. SECURITY SCANNER CONFIGURATION FIELDS	34
2.17. OCI AND HELM CONFIGURATION FIELDS	36
2.18. ACTION LOG CONFIGURATION FIELDS	37
2.18.1. Action log storage configuration	37
2.18.2. Action log rotation and archiving configuration	39
2.19. BUILD LOGS CONFIGURATION FIELDS	39
2.20. DOCKERFILE BUILD TRIGGERS FIELDS	40
2.20.1. GitHub build triggers	41
2.20.2. BitBucket build triggers	41
2.20.3. GitLab build triggers	42
2.21. OAUTH CONFIGURATION FIELDS	42
2.21.1. GitHub OAuth configuration fields	43
2.21.2. Google OAuth configuration fields	43
2.22. NESTED REPOSITORIES CONFIGURATION FIELDS	44
2.23. ADDING OTHER OCI MEDIA TYPES TO QUAY	44
2.24. MAIL CONFIGURATION FIELDS	45
2.25. USER CONFIGURATION FIELDS	46
2.26. RECAPTCHA CONFIGURATION FIELDS	47
2.27. ACI CONFIGURATION FIELDS	48
2.28. JWT CONFIGURATION FIELDS	48
2.29. APP TOKENS CONFIGURATION FIELDS	49
2.30. MISCELLANEOUS CONFIGURATION FIELDS	49
2.31. LEGACY CONFIGURATION FIELDS	52
<b>CHAPTER 3. ENVIRONMENT VARIABLES</b> .....	<b>54</b>
3.1. GEO-REPLICATION	54
3.2. DATABASE CONNECTION POOLING	54
3.3. HTTP CONNECTION COUNTS	55
3.4. WORKER COUNT VARIABLES	55
<b>CHAPTER 4. USING THE CONFIG TOOL TO RECONFIGURE QUAY ON OPENSIFT</b> .....	<b>57</b>
4.1. ACCESSING THE CONFIG EDITOR	57
4.1.1. Retrieving the config editor credentials	57
4.1.2. Logging in to the config editor	58
4.1.3. Changing configuration	59
4.2. MONITORING RECONFIGURATION IN THE UI	60
4.2.1. QuayRegistry resource	60
4.2.2. Events	62
4.3. ACCESSING UPDATED INFORMATION AFTER RECONFIGURATION	63
4.3.1. Accessing the updated config tool credentials in the UI	63

---

4.3.2. Accessing the updated config.yaml in the UI	63
<b>CHAPTER 5. QUAY OPERATOR COMPONENTS</b>	<b>64</b>
5.1. USING MANAGED COMPONENTS	64
5.2. USING UNMANAGED COMPONENTS FOR DEPENDENCIES	65
5.2.1. Using an existing Postgres database	65
5.2.2. NooBaa unmanaged storage	66
5.2.3. Disabling the Horizontal Pod Autoscaler	66
5.3. ADD CERTS WHEN DEPLOYED ON KUBERNETES	67
5.4. CONFIGURING OCI AND HELM WITH THE OPERATOR	67
5.5. VOLUME SIZE OVERRIDES	68
<b>CHAPTER 6. USING THE CONFIGURATION API</b>	<b>70</b>
6.1. RETRIEVING THE DEFAULT CONFIGURATION	70
6.2. RETRIEVING THE CURRENT CONFIGURATION	70
6.3. VALIDATING CONFIGURATION USING THE API	71
6.4. DETERMINING THE REQUIRED FIELDS	72
<b>CHAPTER 7. USING THE CONFIGURATION TOOL</b>	<b>73</b>
7.1. CUSTOM SSL CERTIFICATES UI	73
7.2. BASIC CONFIGURATION	73
7.2.1. Contact information	73
7.3. SERVER CONFIGURATION	74
7.3.1. Server configuration choice	74
7.3.2. TLS configuration	74
7.4. DATABASE CONFIGURATION	75
7.4.1. PostgreSQL configuration	75
7.5. DATA CONSISTENCY	76
7.6. TIME MACHINE CONFIGURATION	76
7.7. REDIS CONFIGURATION	77
7.8. REPOSITORY MIRRORING CONFIGURATION	77
7.9. REGISTRY STORAGE CONFIGURATION	77
7.9.1. Enable storage replication - standalone Quay	77
7.9.2. Storage engines	78
7.9.2.1. Local storage	78
7.9.2.2. Amazon S3 storage	78
7.9.2.3. Azure blob storage	79
7.9.2.4. Google cloud storage	79
7.9.2.5. Ceph object gateway (RADOS) storage	80
7.9.2.6. OpenStack (Swift) storage configuration	80
7.9.2.7. Cloudfront + Amazon S3 storage configuration	81
7.10. ACTION LOG CONFIGURATION	82
7.10.1. Action log storage configuration	82
7.10.1.1. Database action log storage	82
7.10.1.2. Elasticsearch action log storage	82
7.10.2. Action log rotation and archiving	83
7.11. SECURITY SCANNER CONFIGURATION	83
7.12. APPLICATION REGISTRY CONFIGURATION	84
7.13. EMAIL CONFIGURATION	84
7.14. INTERNAL AUTHENTICATION CONFIGURATION	84
7.14.1. LDAP	85
7.14.2. Keystone (OpenStack identity)	85
7.14.3. JWT custom authentication	86
7.14.4. External application token	86

---

7.15. EXTERNAL AUTHENTICATION (OAUTH) CONFIGURATION	87
7.15.1. GitHub (Enterprise) authentication	87
7.15.2. Google authentication	87
7.16. ACCESS SETTINGS CONFIGURATION	88
7.17. DOCKERFILE BUILD SUPPORT	88
7.17.1. GitHub (Enterprise) Build Triggers	89
7.17.2. BitBucket Build Triggers	89
7.17.3. GitLab Build Triggers	90





# CHAPTER 1. GETTING STARTED WITH RED HAT QUAY CONFIGURATION

Red Hat Quay can be deployed by an independent, standalone configuration, or by using the OpenShift Container Platform Red Hat Quay Operator.

How you create, retrieve, update, and validate the Red Hat Quay configuration varies depending on the type of deployment you are using. However, the core configuration options are the same for either deployment type. Core configuration can be set by one of the following options:

- Directly, by editing the **config.yaml** file. See [Editing the configuration file](#) for more information.
- Programmatically, by using the configuration API. See [Using the configuration API](#) for more information.
- Visually, by using the configuration tool UI. See [Using the configuration tool](#) for more information.

For standalone deployments of Red Hat Quay, you must supply the minimum required configuration parameters before the registry can be started. The minimum requirements to start a Red Hat Quay registry can be found in the [Retrieving the current configuration](#) section.

If you install Red Hat Quay on OpenShift Container Platform using the Red Hat Quay Operator, you do not need to supply configuration parameters because the Red Hat Quay Operator supplies default information to deploy the registry.

After you have deployed Red Hat Quay with the desired configuration, you should retrieve, and save, the full configuration from your deployment. The full configuration contains additional generated values that you might need when restarting or upgrading your system.

## 1.1. CONFIGURATION UPDATES FOR QUAY 3.7

### 1.1.1. New configuration fields for Red Hat Quay 3.7.7

Field	Type	Description
REPO_MIRROR_ROLLBACK	Boolean	When set to <b>true</b> , the repository rolls back after a failed mirror attempt.  <b>Default: false</b>

### 1.1.2. New configuration fields

The following configuration fields have been introduced with Red Hat Quay 3.7:

Parameter	Description
-----------	-------------

Parameter	Description
<code>FEATURE_QUOTA_MANAGEMENT</code>	Quota management is now supported. With this feature, users have the ability to report storage consumption and to contain registry growth by establishing configured storage quota limits. For more information about quota management, see <a href="#">Red Hat Quay Quota management and enforcement</a> .
<code>DEFAULT_SYSTEM_REJECT_QUOTA_BYTES</code>	The quota size to apply to all organizations and users. For more information about quota management, see <a href="#">Red Hat Quay Quota management and enforcement</a> .
<code>FEATURE_PROXY_CACHE</code>	Using Red Hat Quay to proxy a remote organization is now supported. With this feature, Red Hat Quay will act as a proxy cache to circumvent pull-rate limitations from upstream registries. For more information about quota management, see <a href="#">Red Hat Quay as proxy cache for upstream registries</a> .

## 1.2. CONFIGURATION UPDATES FOR RED HAT QUAY 3.6

### 1.2.1. New configuration fields

The following configuration fields have been introduced with Red Hat Quay 3.6:

Parameter	Description
<code>FEATURE_EXTENDED_REPOSITORY_NAMES</code>	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 <a href="#">Configuring nested repositories</a> .
<code>FEATURE_USER_INITIALIZE</code>	If set to true, the first <b>User</b> account can be created by the API <code>/api/v1/user/initialize</code> . For more information, see <a href="#">Pre-configuring Red Hat Quay for automation</a> .
<code>ALLOWED_OCI_ARTIFACT_TYPES</code>	Helm, cosign, and ztsd compression scheme artifacts are built into Red Hat Quay 3.6 by default. For any other Open Container Initiative (OCI) media types that are not supported by default, you can add them to the <b>ALLOWED_OCI_ARTIFACT_TYPES</b> configuration in Quay's <b>config.yaml</b> . For more information, see <a href="#">Adding other OCI media types to Quay</a> .

Parameter	Description
<code>CREATE_PRIVATE_REPO_ON_PUSH</code>	Registry users now have the option to set <b>CREATE_PRIVATE_REPO_ON_PUSH</b> in their <b>config.yaml</b> to <b>True</b> or <b>False</b> depending on their security needs.
<code>CREATE_NAMESPACE_ON_PUSH</code>	Pushing to a non-existent organization can now be configured to automatically create the organization.

## 1.2.2. Deprecated configuration fields

The following configuration fields have been deprecated with Red Hat Quay 3.6:

Parameter	Description
<code>FEATURE_HELM_OCI_SUPPORT</code>	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 <b>FEATURE_GENERAL_OCI_SUPPORT</b> property. Users are no longer required to update their <b>config.yaml</b> files to enable support.

## 1.3. EDITING THE CONFIGURATION FILE

To deploy a standalone instance of Red Hat Quay, you must provide the minimal configuration information. The requirements for a minimal configuration can be found in [Red Hat Quay minimal configuration](#).

After supplying the required fields, you can validate your configuration. If there are any issues, they will be highlighted.



### NOTE

It is possible to use the configuration API to validate the configuration, but this requires starting the Quay container in configuration mode. For more information, see [Using the configuration tool](#).

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

## 1.4. LOCATION OF CONFIGURATION FILE IN A STANDALONE DEPLOYMENT

For standalone deployments of Red Hat Quay, the **config.yaml** file must be specified when starting the Red Hat Quay registry. This file is located in the configuration volume. For example, the configuration file is located at **\$QUAY/config/config.yaml** when deploying Red Hat Quay by the following command:

```
$ sudo podman run -d --rm -p 80:8080 -p 443:8443 \
  --name=quay \
  -v $QUAY/config:/conf/stack:Z \
  -v $QUAY/storage:/datastorage:Z \
  {productrepo}/{quayimage}:{productminv}
```

## 1.5. MINIMAL CONFIGURATION

The following configuration options are required for a standalone deployment of Red Hat Quay:

- Server hostname
- HTTP or HTTPS
- Authentication type, for example, Database or Lightweight Directory Access Protocol (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

The following example shows a sample minimal configuration file that uses local storage for images:

```
AUTHENTICATION_TYPE: Database
BUILDLOGS_REDIS:
  host: quay-server.example.com
  password: strongpassword
  port: 6379
  ssl: false
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
  port: 6379
  ssl: false
```



## NOTE

The **SETUP\_COMPLETE** field indicates that the configuration has been validated. You should use the configuration 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.

When configuring local storage, storage is specified on the command line when starting the registry. The following command maps 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 \
  {productrepo}/{quayimage}:{productminv}
```

### 1.5.3. Cloud storage

Storage configuration is detailed in the [Image storage](#) section. For some users, it might be useful to compare the difference between Google Cloud Platform and local storage configurations. For example, the following YAML presents a Google Cloud Platform storage configuration:

#### **\$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. For example:

```
$ sudo podman run -d --rm -p 80:8080 -p 443:8443 \
  --name=quay \
  -v $QUAY/config:/conf/stack:Z \
  {productrepo}/{quayimage}:{productminv}
```

---

## CHAPTER 2. CONFIGURATION FIELDS

This section describes the both required and optional configuration fields when deploying Red Hat Quay.

### 2.1. REQUIRED CONFIGURATION FIELDS

The fields required to configure Red Hat Quay 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

The following sections describe the available automation options for Red Hat Quay deployments:

- [Pre-configuring Red Hat Quay for automation](#)
- [Using the API to create the first user](#)

### 2.3. OPTIONAL CONFIGURATION FIELDS

Optional fields for Red Hat Quay can be found 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

The following table describes the required configuration fields for a Red Hat Quay deployment:

**Table 2.1. General required fields**

Field	Type	Description
<b>AUTHENTICATION_TYPE</b> (Required)	String	The authentication engine to use for credential authentication.  <b>Values:</b> One of <b>Database, LDAP, JWT, Keystone, OIDC</b>  <b>Default: Database</b>
<b>PREFERRED_URL_SCHEME</b> (Required)	String	The URL scheme to use when accessing Red Hat Quay.  <b>Values:</b> One of <b>http, https</b>  <b>Default: http</b>
<b>SERVER_HOSTNAME</b> (Required)	String	The URL at which Red Hat Quay is accessible, without the scheme.  <b>Example:</b> <b>quay-server.example.com</b>
<b>DATABASE_SECRET_KEY</b> (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
<b>SECRET_KEY</b> (Required)	String	Key used to encrypt sensitive fields within the database and at run time. This value should never be changed once set, otherwise all reliant fields, for example, encrypted password credentials, are invalidated.
<b>SETUP_COMPLETE</b> (Required)	Boolean	This is an artefact left over from earlier versions of the software and currently it <b>must</b> be specified with a value of <b>true</b> .

## 2.5. DATABASE CONFIGURATION

This section describes the database configuration fields available for Red Hat Quay deployments.

### 2.5.1. Database URI

With Red Hat Quay, connection to the database is configured by using the required **DB\_URI** field.

The following table describes the **DB\_URI** configuration field:

**Table 2.2. Database URI**

Field	Type	Description
<b>DB_URI</b> (Required)	String	The URI for accessing the database, including any credentials.  Example <b>DB_URI</b> field:  <code>postgresql://quayuser:quaypas s@quay- server.example.com:5432/quay</code>

### 2.5.2. Database connection arguments

Optional connection arguments are configured by the **DB\_CONNECTION\_ARGS** parameter. Some of the key-value pairs defined under **DB\_CONNECTION\_ARGS** are generic, while others are database specific.

The following table describes database connection arguments:

**Table 2.3. Database connection arguments**

Field	Type	Description
<b>DB_CONNECTION_ARGS</b>	Object	Optional connection arguments for the database, such as timeouts and SSL.
<b>.autorollback</b>	Boolean	Whether to use thread-local connections. Should always be <b>true</b>
<b>.threadlocals</b>	Boolean	Whether to use auto-rollback connections. Should always be <b>true</b>

### 2.5.2.1. PostgreSQL SSL connection arguments

With SSL, configuration depends on the database you are deploying. The following example shows a PostgreSQL SSL configuration:

```
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:

Table 2.4. SSL options

Mode	Description
<b>disable</b>	Your configuration only tries non-SSL connections.
<b>allow</b>	Your configuration first tries a non-SSL connection. Upon failure, tries an SSL connection.
<b>prefer</b> (Default)	Your configuration first tries an SSL connection. Upon failure, tries a non-SSL connection.
<b>require</b>	Your configuration only tries an SSL connection. If a root CA file is present, it verifies the certificate in the same way as if verify-ca was specified.
<b>verify-ca</b>	Your configuration only tries an SSL connection, and verifies that the server certificate is issued by a trusted certificate authority (CA).

Mode	Description
<code>verify-full</code>	Only tries an SSL connection, and verifies that the server certificate is issued by a trusted CA and that the requested server host name matches that in the certificate.

For more information on the valid arguments for PostgreSQL, see [Database Connection Control Functions](#).

### 2.5.2.2. MySQL SSL connection arguments

The following example shows a sample MySQL SSL configuration:

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

Information on the valid connection arguments for MySQL is available at [Connecting to the Server Using URI-Like Strings or Key-Value Pairs](#).

## 2.6. IMAGE STORAGE

This section details the image storage features and configuration fields that are available with Red Hat Quay.

### 2.6.1. Image storage features

The following table describes the image storage features for Red Hat Quay:

Table 2.5. Storage config features

Field	Type	Description
<code>FEATURE_REPO_MIRROR</code>	Boolean	If set to true, enables repository mirroring.  <b>Default: false</b>
<code>FEATURE_PROXY_STORAGE</code>	Boolean	Whether to proxy all direct download URLs in storage through NGINX.  <b>Default: false</b>
<code>FEATURE_STORAGE_REPLICATION</code>	Boolean	Whether to automatically replicate between storage engines.  <b>Default: false</b>

## 2.6.2. Image storage configuration fields

The following table describes the image storage configuration fields for Red Hat Quay:

Table 2.6. Storage config fields

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

## 2.6.3. Local storage

The following YAML shows a sample configuration using local storage:

```
DISTRIBUTED_STORAGE_CONFIG:
  default:
    - LocalStorage
    - storage_path: /datastorage/registry
```

```
DISTRIBUTED_STORAGE_DEFAULT_LOCATIONS: []
DISTRIBUTED_STORAGE_PREFERENCE:
- default
```

### 2.6.4. OCS/NooBaa

The following YAML shows a sample configuration using an Open Container Storage/NooBaa instance:

```
DISTRIBUTED_STORAGE_CONFIG:
rhocsStorage:
- RHOCSSStorage
- 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

The following YAML shows a sample configuration using Ceph/RadosGW and Hitachi HCP storage:

```
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

The following YAML shows a sample configuration using 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

The following YAML shows a sample configuration using 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

The following YAML shows a sample configuration using Azure Storage:

```
DISTRIBUTED_STORAGE_CONFIG:
  azureStorage:
    - AzureStorage
    - azure_account_name: azure_account_name_here
      azure_container: azure_container_here
      storage_path: /datastorage/registry
      azure_account_key: azure_account_key_here
      sas_token: some/path/
      endpoint_url: https://[account-name].blob.core.usgovcloudapi.net 1
DISTRIBUTED_STORAGE_DEFAULT_LOCATIONS: []
DISTRIBUTED_STORAGE_PREFERENCE:
  - azureStorage
```

**1** The **endpoint\_url** parameter for Azure storage is optional and can be used with Microsoft Azure Government (MAG) endpoints. If left blank, the **endpoint\_url** will connect to the normal Azure region.

As of Red Hat Quay 3.7, you must use the Primary endpoint of your MAG Blob service. Using the Secondary endpoint of your MAG Blob service will result in the following error:

**AuthenticationErrorDetail:Cannot find the claimed account when trying to GetProperties for the account whusc8-secondary.**

## 2.6.9. Swift storage

The following YAML shows a sample configuration using 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

This section details the configuration fields available for Redis deployments.

### 2.7.1. Build logs

The following build logs configuration fields are available for Redis deployments:

**Table 2.7. Build logs configuration**

Field	Type	Description
<b>BUILDLOGS_REDIS</b> (Required)	Object	Redis connection details for build logs caching.
<b>.host</b> (Required)	String	The hostname at which Redis is accessible. <b>Example:</b> <b>quay-server.example.com</b>
<b>.port</b> (Required)	Number	The port at which Redis is accessible. <b>Example:</b> <b>6379</b>
<b>.password</b>	String	The port at which Redis is accessible. <b>Example:</b> <b>strongpassword</b>
<b>.port</b> (Required)	Number	The port at which Redis is accessible. <b>Example:</b> <b>6379</b>
<b>ssl</b>	Boolean	Whether to enable TLS communication between Redis and Quay. Defaults to false.

### 2.7.2. User events

The following user event fields are available for Redis deployments:

**Table 2.8. User events config**

Field	Type	Description
<b>USER_EVENTS_REDIS</b> (Required)	Object	Redis connection details for user event handling.
<b>.host</b> (Required)	String	The hostname at which Redis is accessible. <b>Example:</b> <b>quay-server.example.com</b>
<b>.port</b> (Required)	Number	The port at which Redis is accessible. <b>Example:</b> <b>6379</b>
<b>.password</b>	String	The port at which Redis is accessible. <b>Example:</b> <b>strongpassword</b>
<b>ssl</b>	Boolean	Whether to enable TLS communication between Redis and Quay. Defaults to false.

### 2.7.3. Example Redis configuration

The following YAML shows a sample configuration using Redis:

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

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



#### NOTE

If your deployment uses Azure Cache for Redis and **ssl** is set to **true**, the port defaults to **6380**.

## 2.8. MODELCACHE CONFIGURATION OPTIONS

The following options are available on Red Hat Quay for configuring ModelCache.

### 2.8.1. Memcache configuration option



Memcache is the default ModelCache configuration option. With Memcache, no additional configuration is necessary.

### 2.8.2. Single Redis configuration option

The following configuration is for a single Redis instance with optional read-only replicas:

```
DATA_MODEL_CACHE_CONFIG:
engine: redis
redis_config:
  primary:
    host: <host>
    port: <port>
    password: <password if ssl is true>
    ssl: <true | false >
  replica:
    host: <host>
    port: <port>
    password: <password if ssl is true>
    ssl: <true | false >
```

### 2.8.3. Clustered Redis configuration option

Use the following configuration for a clustered Redis instance:

```
DATA_MODEL_CACHE_CONFIG:
engine: rediscluster
redis_config:
  startup_nodes:
    - host: <cluster-host>
      port: <port>
  password: <password if ssl: true>
  read_from_replicas: <true|false>
  skip_full_coverage_check: <true | false>
  ssl: <true | false >
```

## 2.9. TAG EXPIRATION CONFIGURATION FIELDS

The following tag expiration configuration fields are available with Red Hat Quay:

Table 2.9. Tag expiration configuration fields

Field	Type	Description
FEATURE_GARBAGE_COLLECTION	Boolean	Whether garbage collection of repositories is enabled.  <b>Default:</b> True

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

### 2.9.1. Example tag expiration configuration

The following YAML shows a sample tag expiration configuration:

```

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

```

## 2.10. PRE-CONFIGURING RED HAT QUAY FOR AUTOMATION

Red Hat Quay has several configuration options that support automation. These options can be set before deployment to minimize the need to interact with the user interface.

### 2.10.1. Allowing the API to create the first user

To create the first user using the `/api/v1/user/initialize` API, set the **FEATURE\_USER\_INITIALIZE** parameter to **true**. Unlike all other registry API calls which require an OAuth token that is generated by an OAuth application in an existing organization, the API endpoint does not require authentication.

After you have deployed Red Hat Quay, you can use the API to create a user, for example, **quayadmin**, assuming that no other users have already been created. For more information see [Using the API to create the first user](#).

### 2.10.2. Enabling general API access

Set the config option **BROWSER\_API\_CALLS\_XHR\_ONLY** to **false** to allow general access to the Red Hat Quay registry API.

### 2.10.3. Adding a super user

After deploying Red Hat Quay, you can create a user. We advise that the first user be given administrator privileges with full permissions. Full permissions can be configured in advance by using the **SUPER\_USER** configuration object. For example:

```
...
SERVER_HOSTNAME: quay-server.example.com
SETUP_COMPLETE: true
SUPER_USERS:
  - quayadmin
...
```

### 2.10.4. Restricting user creation

After 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. For example:

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

### 2.10.5. Enabling new functionality

To use new Red Hat Quay 3.7 functionality, enable some or all of the following features:

```
...
FEATURE_QUOTA_MANAGEMENT: true
FEATURE_BUILD_SUPPORT: true
FEATURE_PROXY_CACHE: true
FEATURE_STORAGE_REPLICATION: true
DEFAULT_SYSTEM_REJECT_QUOTA_BYTES: 102400000
...
```

### 2.10.6. Suggested configuration for automation

The following **config.yaml** parameters are suggested for automation:

```
...
FEATURE_USER_INITIALIZE: true
BROWSER_API_CALLS_XHR_ONLY: false
SUPER_USERS:
```

```
- quayadmin  
FEATURE_USER_CREATION: false  
...
```

### 2.10.7. Deploying the Red Hat Quay Operator using the initial configuration

Use the following procedure to deploy Red Hat Quay on OpenShift Container Platform using the initial configuration.

#### Prerequisites

- You have installed the **oc** CLI.

#### Procedure

1. Create a secret using the configuration file:

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

2. Create a **quayregistry.yaml** file. Identify the unmanaged components and reference the created secret, for example:

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

3. Deploy the Red Hat Quay registry:

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

#### Next Steps

- [Using the API to create the first user](#)

### 2.10.8. Using the API to deploy Red Hat Quay

This section introduces using the API to deploy Red Hat Quay.

#### Prerequisites

- 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 your Red Hat Quay deployment, see the section [Pre-configuring Red Hat Quay for automation](#)

#### 2.10.8.1. Using the API to create the first user

Use the following procedure to create the first user in your Red Hat Quay organization.



## NOTE

This procedure requests an OAuth token by specifying **"access\_token": true**.

- Using the **status.registryEndpoint** URL, invoke the **/api/v1/user/initialize** API, passing in the username, password and email address by entering the following command:

```
$ 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}'
```

If successful, the command returns an object with the username, email, and encrypted password. For example:

```
{"access_token":"6B4QTRSTSD1HMIG915VPX7BMEZBVB9GPNY2FC2ED",
"email":"quayadmin@example.com","encrypted_password":"1nZMLH57RIE5UGdL/yYpDOHL
qiNCgimb6W9kfF8MjZ1xrfDpRyRs9NUnUuNuAitW","username":"quayadmin"}
```

If a user already exists in the database, an error is returned:

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

If your password is not at least eight characters or contains whitespace, an error is returned:

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

### 2.10.8.2. Using the OAuth token

After invoking the API, you can call out the rest of the Red Hat Quay API by specifying the returned OAuth code.

#### Prerequisites

- You have invoked the **/api/v1/user/initialize** API, and passed in the username, password, and email address.

#### Procedure

- Obtain the list of current users by entering the following command:

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

Example output:

```
{
  "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.

### 2.10.8.3. Using the API to create an organization

The following procedure details how to use the API to create a Red Hat Quay organization.

#### Prerequisites

- You have invoked the **/api/v1/user/initialize** API, and passed in the username, password, and email address.
- You have called out the rest of the Red Hat Quay API by specifying the returned OAuth code.

#### Procedure

1. 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"}'

```

Example output:

```
"Created"
```

2. You can retrieve the details of the organization you created by entering the following command:

```

$ 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

```

Example output:

```

{
  "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
}

```

## 2.11. BASIC CONFIGURATION FIELDS

Table 2.10. Basic configuration

Field	Type	Description
REGISTRY_TITLE	String	If specified, the long-form title for the registry.  <b>Default:</b> <b>Quay Enterprise</b>

Field	Type	Description
REGISTRY_TITLE_SHORT	String	If specified, the short-form title for the registry.  <b>Default:</b> <b>Quay Enterprise</b>
BRANDING	Object	Custom branding for logos and URLs in the Red Hat Quay UI.
.logo (Required)	String	Main logo image URL.  <b>Example:</b> <b>/static/img/quay-horizontal-color.svg</b>
.footer_img	String	Logo for UI footer.  <b>Example:</b> <b>/static/img/RedHat.svg</b>
.footer_url	String	Link for footer image.  <b>Example:</b> <a href="https://redhat.com">https://redhat.com</a>
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.  <b>Pattern:</b> <b>^mailto:(.)+\$</b> <b>Example:</b> <b>mailto:support@quay.io</b>
[1]	String	Adds a link to visit an IRC chat room.  <b>Pattern:</b> <b>^irc://(.)+\$</b> <b>Example:</b> <a href="irc://chat.freenode.net:6665/quay">irc://chat.freenode.net:6665/quay</a>



Field	Type	Description
[2]	String	Adds a link to call a phone number.+ <b>Pattern:</b> <b><code>^tel:(.)+\$</code></b> <b>Example:</b> <b><code>tel:+1-888-930-3475</code></b>
[3]	String	Adds a link to a defined URL.  <b>Pattern:</b> <b><code>^http(s)?://(.)+\$</code></b> <b>Example:</b> <b><a href="https://twitter.com/quayio">https://twitter.com/quayio</a></b>

## 2.12. SSL CONFIGURATION FIELDS

Table 2.11. SSL configuration

Field	Type	Description
<b>PREFERRED_URL_SCHEME</b>	String	One of <b>http</b> or <b>https</b> . Note that users only set their <b>PREFERRED_URL_SCHEME</b> to <b>http</b> when there is no TLS encryption in the communication path from the client to Quay.  + Users must set their <b>PREFERRED_URL_SCHEME</b> to <b>https</b> when using a TLS-terminating load balancer, a reverse proxy (for example, Nginx), or when using Quay with custom SSL certificates directly. In most cases, the <b>PREFERRED_URL_SCHEME</b> should be <b>https</b> . <b>Default: http</b>
<b>SERVER_HOSTNAME</b> (Required)	String	The URL at which Red Hat Quay is accessible, without the scheme  <b>Example:</b> <b><code>quay-server.example.com</code></b>

Field	Type	Description
<b>SSL_CIPHERS</b>	Array of String	If specified, the nginx-defined list of SSL ciphers to enabled and disabled  <b>Example:</b> [ <b>CAMELLIA, !3DES</b> ]
<b>SSL_PROTOCOLS</b>	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.  <b>Example:</b> [ <b>'TLSv1','TLSv1.1','TLSv1.2','TLSv1.3'</b> ]
<b>SESSION_COOKIE_SECURE</b>	Boolean	Whether the <b>secure</b> property should be set on session cookies  <b>Default:</b> False  <b>Recommendation:</b> Set to True for all installations using SSL

### 2.12.1. 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

## 2.13. 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.

### 2.13.1. 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:v3.7.10 "/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
```

## 2.14. LDAP CONFIGURATION FIELDS

Table 2.12. LDAP configuration

Field	Type	Description
<b>AUTHENTICATION_TYPE</b> (Required)	String	Must be set to <b>LDAP</b>

Field	Type	Description
FEATURE_TEAM_SYNCING	Boolean	Whether to allow for team membership to be synced from a backing group in the authentication engine (LDAP or Keystone)  <b>Default: true</b>
FEATURE_NONSUPERUSER_TEAM_SYNCING_SETUP	Boolean	If enabled, non-superusers can setup syncing on teams using LDAP  <b>Default: false</b>
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.
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.

Field	Type	Description
TEAM_RESYNC_STALE_TIME	String	<p>If team syncing is enabled for a team, how often to check its membership and resync if necessary</p> <p><b>Pattern:</b>  <code>^[0-9]+(w m d h s)\$</code></p> <p><b>Example:</b>  <b>2h</b></p> <p><b>Default:</b>  <b>30m</b></p>

### 2.14.1. LDAP configuration example

\$QUAY/config/config.yamll

```

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

```

## 2.15. MIRRORING CONFIGURATION FIELDS

Table 2.13. Mirroring configuration

Field	Type	Description
FEATURE_REPO_MIRROR	Boolean	<p>Enable or disable repository mirroring</p> <p><b>Default: false</b></p>
REPO_MIRROR_INTERVAL	Number	<p>The number of seconds between checking for repository mirror candidates</p> <p><b>Default: 30</b></p>

Field	Type	Description
REPO_MIRROR_SERVER_HOSTNAME	String	Replaces the <b>SERVER_HOSTNAME</b> as the destination for mirroring.  <b>Default:</b> None  <b>Example:</b> <b>openshift-quay-service</b>
REPO_MIRROR_TLS_VERIFY	Boolean	Require HTTPS and verify certificates of Quay registry during mirror.  <b>Default:</b> <b>false</b>
REPO_MIRROR_ROLLBACK	Boolean	When set to <b>true</b> , the repository rolls back after a failed mirror attempt.  <b>Default:</b> <b>false</b>

## 2.16. SECURITY SCANNER CONFIGURATION FIELDS

Table 2.14. Security scanner configuration

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

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 <b>last_indexed datetime</b> in the <b>manifestsecuritystatus</b> 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  <b>Pattern:</b> <b><code>^http(s)?://(.)+\$</code></b>  <b>Example:</b> <b><a href="http://192.168.99.101:6060">http://192.168.99.101:6060</a></b>
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  <b>Default:</b> 30
SECURITY_SCANNER_ENDPOINT	String	The endpoint for the V2 security scanner  <b>Pattern:</b> <b><code>^http(s)?://(.)+\$</code></b>  <b>Example:</b> <b><a href="http://192.168.99.100:6060">http://192.168.99.100:6060</a></b>

Field	Type	Description
<code>SECURITY_SCANNER_INDEXING_INTERVAL</code>	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.

## 2.17. OCI AND HELM CONFIGURATION FIELDS

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 2.15. OCI and Helm configuration fields

Field	Type	Description
<code>FEATURE_GENERAL_OCI_SUPPORT</code>	Boolean	Enable support for OCI artifacts  <b>Default:</b> True



Field	Type	Description
FEATURE_HELM_OCI_SUPPORT	Boolean	Enable support for Helm artifacts  <b>Default:</b> 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.

## 2.18. ACTION LOG CONFIGURATION FIELDS

### 2.18.1. Action log storage configuration

Table 2.16. Action log storage configuration

Field	Type	Description
FEATURE_LOG_EXPORT	Boolean	Whether to allow exporting of action logs  <b>Default:</b> True
LOGS_MODEL	String	Enable or disable the security scanner  <b>Values:</b> One of <b>database</b> , <b>transition_reads_both_writes_es</b> , <b>elasticsearch</b> <b>Default:</b> <b>database</b>
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**

## 2.18.2. Action log rotation and archiving configuration

Table 2.17. Action log rotation and archiving configuration

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

## 2.19. BUILD LOGS CONFIGURATION FIELDS

Table 2.18. Build logs configuration fields

Field	Type	Description
-------	------	-------------

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.  <b>Default: False</b>
LOG_ARCHIVE_LOCATION	String	The storage location, defined in DISTRIBUTED_STORAGE_CONFIG, in which to place the archived build logs  <b>Example: s3_us_east</b>
LOG_ARCHIVE_PATH	String	The path under the configured storage engine in which to place the archived build logs in JSON form  <b>Example: archives/buildlogs</b>

## 2.20. DOCKERFILE BUILD TRIGGERS FIELDS

Table 2.19. Dockerfile build support

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

### 2.20.1. GitHub build triggers

Table 2.20. GitHub build triggers

Field	Type	Description
FEATURE_GITHUB_BUILD	Boolean	Whether to support GitHub build triggers  <b>Default:</b> False
GITHUB_TRIGGER_CONFIG	Object	Configuration for using GitHub (Enterprise) for build triggers
.GITHUB_ENDPOINT (Required)	String	The endpoint for GitHub (Enterprise)  <b>Example:</b> <a href="https://github.com/">https://github.com/</a>
.API_ENDPOINT	String	The endpoint of the GitHub (Enterprise) API to use. Must be overridden for <b>github.com</b>  <b>Example:</b> <a href="https://api.github.com/">https://api.github.com/</a>
.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.

### 2.20.2. BitBucket build triggers

Table 2.21. BitBucket build triggers

Field	Type	Description
FEATURE_BITBUCKET_BUILD	Boolean	Whether to support Bitbucket build triggers  <b>Default:</b> False

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

### 2.20.3. GitLab build triggers

Table 2.22. GitLab build triggers

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

## 2.21. OAUTH CONFIGURATION FIELDS

Table 2.23. OAuth fields

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

## 2.21.1. GitHub OAuth configuration fields

Table 2.24. GitHub OAuth fields

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

## 2.21.2. Google OAuth configuration fields

Table 2.25. Google OAuth fields

Field	Type	Description
FEATURE_GOOGLE_LOGIN	Boolean	Whether Google login is supported  **Default: <b>False</b>
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  <b>Example:</b> <b>0e8dbe15c4c7630b6780</b>
.CLIENT_SECRET (Required)	String	The registered client secret for this Red Hat Quay instance  <b>Example:</b> e4a58ddd3d7408b7aec109e855 64a0d153d3e846

## 2.22. NESTED REPOSITORIES CONFIGURATION FIELDS

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 is added to the config.yaml by default. Enablement allows the use of / in repository names.

```
FEATURE_EXTENDED_REPOSITORY_NAMES: true
```

Table 2.26. OCI and nested repositories configuration fields

Field	Type	Description
FEATURE_EXTENDED_REPOSITORY_NAMES	Boolean	Enable support for nested repositories  <b>Default:</b> True

## 2.23. 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.

## 2.24. MAIL CONFIGURATION FIELDS

Table 2.27. Mail configuration fields

Field	Type	Description
FEATURE_MAILING	Boolean	Whether emails are enabled  <b>Default: False</b>
MAIL_DEFAULT_SENDER	String	If specified, the e-mail address used as the <b>from</b> when Red Hat Quay sends e-mails. If none, defaults to <b>support@quay.io</b>  <b>Example:</b> <b>support@example.com</b>
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.  <b>Example: smtp.example.com</b>
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  <b>Default: True</b>

## 2.25. USER CONFIGURATION FIELDS

Table 2.28. User configuration fields

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

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

## 2.26. RECAPTCHA CONFIGURATION FIELDS

Table 2.29. Recaptcha configuration fields

Field	Type	Description
-------	------	-------------

Field	Type	Description
FEATURE_RECAPTCHA	Boolean	Whether Recaptcha is necessary for user login and recovery  <b>Default:</b> 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

## 2.27. ACI CONFIGURATION FIELDS

Table 2.30. ACI configuration fields

Field	Type	Description
FEATURE_ACI_CONVERSION	Boolean	Whether to enable conversion to ACIs  <b>Default:</b> 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

## 2.28. JWT CONFIGURATION FIELDS

Table 2.31. JWT configuration fields

Field	Type	Description
JWT_AUTH_ISSUER	String	The endpoint for JWT users  <b>Pattern:</b> <code>^http(s)?://(.)+\$</code> <b>Example:</b> <a href="http://192.168.99.101:6060">http://192.168.99.101:6060</a>

Field	Type	Description
JWT_GETUSER_ENDPOINT	String	The endpoint for JWT users <b>Pattern:</b> <code>^http(s)?://(.)+\$</code> <b>Example:</b> <a href="http://192.168.99.101:6060">http://192.168.99.101:6060</a>
JWT_QUERY_ENDPOINT	String	The endpoint for JWT queries  <b>Pattern:</b> <code>^http(s)?://(.)+\$</code> <b>Example:</b> <a href="http://192.168.99.101:6060">http://192.168.99.101:6060</a>
JWT_VERIFY_ENDPOINT	String	The endpoint for JWT verification  <b>Pattern:</b> <code>^http(s)?://(.)+\$</code> <b>Example:</b> <a href="http://192.168.99.101:6060">http://192.168.99.101:6060</a>

## 2.29. APP TOKENS CONFIGURATION FIELDS

Table 2.32. App tokens configuration fields

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

## 2.30. MISCELLANEOUS CONFIGURATION FIELDS

Table 2.33. Miscellaneous configuration fields

Field	Type	Description
-------	------	-------------

Field	Type	Description
<code>ALLOW_PULLS_WITHOUT_STRICT_LOGGING</code>	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.  <b>Default:</b> False
<code>AVATAR_KIND</code>	String	The types of avatars to display, either generated inline (local) or Gravatar (gravatar)  <b>Values:</b> local, gravatar
<code>BROWSER_API_CALLS_XHR_ONLY</code>	Boolean	If enabled, only API calls marked as being made by an XHR will be allowed from browsers  <b>Default:</b> True
<code>DEFAULT_NAMESPACE_MAXIMUM_BUILD_COUNT</code>	Number	The default maximum number of builds that can be queued in a namespace.  <b>Default:</b> None
<code>ENABLE_HEALTH_DEBUG_SECRET</code>	String	If specified, a secret that can be given to health endpoints to see full debug info when not authenticated as a superuser
<code>EXTERNAL_TLS_TERMINATION</code>	Boolean	Set to <b>true</b> if TLS is supported, but terminated at a layer before Quay. Set to <b>false</b> when Quay is running with its own SSL certificates and receiving TLS traffic directly.
<code>FRESH_LOGIN_TIMEOUT</code>	String	The time after which a fresh login requires users to re-enter their password  <b>Example:</b> 5m

Field	Type	Description
HEALTH_CHECKER	String	The configured health check  <b>Example:</b> <code>('RDSAwareHealthCheck', {'access_key': 'foo', 'secret_key': 'bar'})</code>
PROMETHEUS_NAMESPACE	String	The prefix applied to all exposed Prometheus metrics  <b>Default: quay</b>
PUBLIC_NAMESPACES	Array of String	If a namespace is defined in the public namespace list, then it will appear on <b>all</b> 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  <b>Values: normal or read-only</b>
SEARCH_MAX_RESULT_PAGE_COUNT	Number	Maximum number of pages the user can paginate in search before they are limited  <b>Default: 10</b>
SEARCH_RESULTS_PER_PAGE	Number	Number of results returned per page by search page  <b>Default: 10</b>
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  <b>Default: 50</b>
WEBHOOK_HOSTNAME_BLACKLIST	Array of String	The set of hostnames to disallow from webhooks when validating, beyond localhost

Field	Type	Description
CREATE_PRIVATE_REPO_ON_PUSH	Boolean	Whether new repositories created by push are set to private visibility  <b>Default:</b> True
CREATE_NAMESPACE_ON_PUSH	Boolean	Whether new push to a non-existent organization creates it  <b>Default:</b> False
NON_RATE_LIMITED_NAMESPACES	Array of String	If rate limiting has been enabled using <b>FEATURE_RATE_LIMITS</b> , you can override it for specific namespace that require unlimited access.

## 2.31. LEGACY CONFIGURATION FIELDS

Some fields are deprecated or obsolete:

Table 2.34. Legacy configuration 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 <b>FEATURE_BLACKLISTED_EMAILS</b> is set to true  <b>Example:</b> "example.com", "example.org"
BLACKLIST_V2_SPEC	String	The Docker CLI versions to which Red Hat Quay will respond that V2 is <b>unsupported</b>  <b>Example:</b> <1.8.0 <b>Default:</b> <1.6.0



Field	Type	Description
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
<b>QUAY_DISTRIBUTED_STORAGE_PREFERENCE</b>	String	The preferred storage engine (by ID in <code>DISTRIBUTED_STORAGE_CONFIG</code> ) 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
<b>DB_CONNECTION_POOLING</b>	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  <b>Default:</b> 50
WORKER_CONNECTION_COUNT_REGISTRY	Number	Simultaneous HTTP connections for registry  <b>Default:</b> WORKER_CONNECTION_COUNT
WORKER_CONNECTION_COUNT_WEB	Number	Simultaneous HTTP connections for web UI  <b>Default:</b> WORKER_CONNECTION_COUNT
WORKER_CONNECTION_COUNT_SECSCAN	Number	Simultaneous HTTP connections for Clair  <b>Default:</b> 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 <b>Quay</b> container  <b>Values:</b> Integer between 8 and 64

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

Project: openshift-operators ▾

Installed Operators > quay-operator.v3.5.2 > QuayRegistry details

**QR** example Actions ▾

[Details](#) [YAML](#) [Resources](#) [Events](#)

### Quay Registry overview

<b>Name</b> example	<b>Current Version</b> 3.5.2
<b>Namespace</b> openshift-operators	<b>Config Editor Credentials Secret</b> example-quay-config-editor-credentials-9ffggtfc7
<b>Labels</b> No labels <a href="#">Edit</a>	<b>Registry Endpoint</b> example-quay-openshift-operators.apps.docs.quayteam.org
<b>Annotations</b> 1 annotation	<b>Config Editor Endpoint</b> example-quay-config-editor-openshift-operators.apps.docs.quayteam.org
<b>Created at</b> Jun 24, 5:33 pm	
<b>Owner</b> No owner	

### 4.1.1. Retrieving the config editor credentials


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

Project: openshift-operators ▾

---

Secrets > Secret details

**S** example-quay-config-editor-credentials-9ffgfgtfc7 Add Secret to workload Actions ▾

Managed by  example


---

[Details](#) [YAML](#)


---

### Secret details


**Name**  
example-quay-config-editor-credentials-9ffgfgtfc7


**Namespace**  
 openshift-operators


**Type**  
Opaque

**Labels** Edit 

quay-operator/quayregistry=example

**Annotations**  
4 annotations 


**Created at**  
 Jun 25, 11:40 am

**Owner**  
 example


---

**Data** [Reveal values](#)

**password**

..... 

**username**

..... 

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

**Data** [Hide values](#)

**password**

Zr1iN6tCtZeVww4q 

**username**

quayconfig 

## 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:

## Red Hat Quay Setup

### Custom SSL Certificates

This section lists any custom or self-signed SSL certificates that are installed in the Quay container on startup after being read from the `extra_ca_certs` directory in the configuration volume. Custom certificates are typically used in place of publicly signed certificates for corporate-internal services. Please **make sure** that all custom names used for downstream services (such as Clair) are listed in the certificates below.

Upload certificates:  Select file

Select custom certificate to add to configuration. Must be in PEM format and end extension '.crt'

CERTIFICATE FILENAME	STATUS	NAMES HANDLED
extra_ca_certs/service-ca.crt	✔ Certificate is valid	openshift-service-serving-signer@1624454606

### Basic Configuration

**Registry Title:**   
Name of registry to be displayed in the Contact Page.

**Registry Title Short:**

**Enterprise Logo URL:**    
Enter the full URL to your company's logo.

**Contact Information:** URL   
Information to show in the Contact Page. If none specified, CoreOS contact information is displayed.

### Server Configuration

**Server Hostname:**   
The HTTP host (and optionally the port number if a non-standard HTTP/HTTPS port) of the location where the registry will be accessible on the network.

**TLS:** Red Hat Quay handles TLS ▼

Validate Configuration Changes Enabling TLS also enables [HTTP Strict Transport Security](#). This prevents downgrade attacks and cookie theft, but browsers will reject all future insecure connections on this hostname.

### 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:

🔍 Time Machine

Time machine keeps older copies of tags within a repository for the configured period of time, after which they are garbage collected. This allows users to revert tags to older images in case they accidentally pushed a broken image. It is highly recommended to have time machine enabled, but it does take a bit more space in storage.

**Allowed expiration periods:** 2w Remove

The expiration periods allowed for configuration. The default tag expiration "must" be in this list.

**Default expiration period:**

The default tag expiration period for all namespaces (users and organizations). Must be expressed in a duration string form: `30m`, `1h`, `1d`, `2w`.

**Allow users to select expiration:**  **Enable Expiration Configuration**

If enabled, users will be able to select the tag expiration duration for the namespace(s) they administrate, from the configured list of options.

2. Select **Validate Configuration Changes** to ensure that the changes are valid
3. Apply the changes by pressing the **Reconfigure Quay** button:

## Validating configuration



✓ Configuration Validated

Continue Editing

Download

Reconfigure Quay

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

## Validating configuration

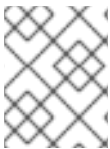


✓ Configuration Validated

Continue Editing

Download

Reconfigure 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**:



Project: quay-enterprise ▾

Installed Operators &gt; quay-operator.v3.6.0 &gt; QuayRegistry details

 example-registryDetails YAML Resources Events

```

1  apiVersion: quay.redhat.com/v1
2  kind: QuayRegistry
3  metadata:
4    selfLink: >=
5    /apis/quay.redhat.com/v1/namespaces/quay-enterprise/quayregistries/example-registry
6    resourceVersion: '78140'
7    name: example-registry
8    uid: 0a77c77c-b560-4d52-9d8a-ba8481ab4d04
9    creationTimestamp: '2021-09-24T10:13:02Z'
10   generation: 7
11  > managedFields: --
45   namespace: quay-enterprise
46   finalizers:
47     - quay-operator/finalizer
48   spec:
49  > components: --
68   configBundleSecret: example-registry-quay-config-bundle-zb9c7
69   status:
70     conditions:
71     - lastTransitionTime: '2021-09-24T10:14:40Z'
72       lastUpdateTime: '2021-09-24T10:14:40Z'
73       message: all registry component healthchecks passing
74       reason: HealthChecksPassing
75       status: 'True'
76       type: Available
77     - lastTransitionTime: '2021-09-24T11:23:02Z'
78       lastUpdateTime: '2021-09-24T11:23:02Z'
79       message: all objects created/updated successfully
80       reason: ComponentsCreationSuccess
81       status: 'False'
82       type: RolloutBlocked
83   configEditorCredentialsSecret: example-registry-quay-config-editor-credentials-gbtbkh94kh
84   configEditorEndpoint: >=
85     https://example-registry-quay-config-editor-quay-enterprise.apps.docs.quayteam.org
86   currentVersion: 3.6.0
87   lastUpdated: '2021-09-24 11:23:02.084685976 +0000 UTC'

```

**i** This object has been updated.

Click reload to see the new version.

Save

Reload

Cancel

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.

Project: quay-enterprise ▾

Installed Operators &gt; quay-operator.v3.6.0 &gt; QuayRegistry details

 example-registry

[Details](#)
[YAML](#)
[Resources](#)
[Events](#)

```

1  apiVersion: quay.redhat.com/v1
2  kind: QuayRegistry
3  metadata:
4  ▾ selfLink: >-
5    /apis/quay.redhat.com/v1/namespaces/quay-enterprise/quayregistries/example-registry
6    resourceVersion: '79051'
7    name: example-registry
8    uid: 0a77c77c-b560-4d52-9d8a-ba8481ab4d04
9    creationTimestamp: '2021-09-24T10:13:02Z'
10   generation: 7
11  > managedFields:--
12  namespace: quay-enterprise
13  finalizers:
14  - quay-operator/finalizer
15  spec:
16  > components:--
17  configBundleSecret: example-registry-quay-config-bundle-zb9c7
18  status:
19  conditions:
20  - lastTransitionTime: '2021-09-24T10:14:40Z'
21    lastUpdateTime: '2021-09-24T10:14:40Z'
22    message: all registry component healthchecks passing
23    reason: HealthChecksPassing
24    status: 'True'
25    type: Available
26  - lastTransitionTime: '2021-09-24T11:23:02Z'
27    lastUpdateTime: '2021-09-24T11:23:02Z'
28    message: all objects created/updated successfully
29    reason: ComponentsCreationSuccess
30    status: 'False'
31    type: RolloutBlocked
32  configEditorCredentialsSecret: example-registry-quay-config-editor-credentials-gbtbkh94kh
33  configEditorEndpoint: >-
34    https://example-registry-quay-config-editor-quay-enterprise.apps.docs.quayteam.org
35  currentVersion: 3.6.0
36  lastUpdated: '2021-09-24 11:23:02.084685976 +0000 UTC'
37  registryEndpoint: 'https://example-registry-quay-quay-enterprise.apps.docs.quayteam.org'
38  unhealthyComponents: {}

```

Save

Reload

Cancel

## 4.2.2. Events

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

Event Type	Message	Source	Timestamp
Warning	failed to get cpu utilization: did not receive metrics for any ready pods	horizontal-pod-autoscaler	Sep 24, 12:16 pm (29 times in the last an hour)
Warning	Readiness probe failed: Get "http://10.128.2.40:8080/health/instance": dial tcp 10.128.2.40:8080: connect: connection refused	docs-k95iz-worker-d-tgz54.c.quay-devel.internal	Sep 24, 12:16 pm
Normal	Scaled down replica set example-registry-quay-app-c7698bfc to 0	deployment-controller	a few seconds ago
Normal	Stopping container quay-app	docs-k95iz-worker-d-tgz54.c.quay-devel.internal	a few seconds ago
Normal	Deleted pod: example-registry-quay-app-c7698bfc-lsx2	replicaset-controller	a few seconds ago

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

Event Type	Message	Source	Timestamp
Warning	failed to get cpu utilization: did not receive metrics for any ready pods	horizontal-pod-autoscaler	Sep 24, 12:16 pm (29 times in the last an hour)
Warning	Readiness probe failed: Get "http://10.128.2.40:8080/health/instance": dial tcp 10.128.2.40:8080: connect: connection refused	docs-k95iz-worker-d-tgz54.c.quay-devel.internal	Sep 24, 12:16 pm
Normal	Scaled down replica set example-registry-quay-app-c7698bfc to 0	deployment-controller	a few seconds ago
Normal	Stopping container quay-app	docs-k95iz-worker-d-tgz54.c.quay-devel.internal	a few seconds ago
Normal	Deleted pod: example-registry-quay-app-c7698bfc-lsx2	replicaset-controller	a few seconds ago

## 4.3. ACCESSING UPDATED INFORMATION AFTER RECONFIGURATION

### 4.3.1. Accessing the updated config tool credentials in the UI

With Red Hat Quay 3.7, reconfiguring Quay through the UI no longer generates a new login password. The password now generates only once, and remains the same after reconciling **QuayRegistry** objects.

### 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:
    - kind: quay
      managed: true
    - kind: postgres
      managed: true
    - kind: clair
      managed: true
    - kind: redis
      managed: true
    - kind: horizontalpodautoscaler
      managed: true
    - kind: objectstorage
      managed: true
    - kind: route
      managed: true
    - kind: mirror
      managed: true
    - kind: monitoring
      managed: true
    - kind: tls
      managed: true
    - kind: clairpostgres
      managed: true
```

### 5.1. USING MANAGED COMPONENTS

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

- **quay**: Holds overrides for the Quay deployment, for example, environment variables and number of replicas. This component is new in Red Hat Quay 3.7 and cannot be set to unmanaged.
- **postgres**: For storing the registry metadata, uses a version of Postgres 10 from the [Software Collections](#)
- **clair**: Provides image vulnerability scanning
- **redis**: Handles Quay builder coordination and some internal logging
- **horizontalpodautoscaler**: Adjusts the number of Quay pods depending on memory/cpu consumption

- **objectstorage:** For storing image layer blobs, utilizes the **ObjectBucketClaim** Kubernetes API which is provided by Noobaa/RHOCS
- **route:** Provides an external endpoint to the Quay registry from outside OpenShift
- **mirror:** Configures repository mirror workers (to support optional repository mirroring)
- **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
- **clairpostgres:** Configures a managed Clair database

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:
    - RHOCSSStorage
    - 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
TEICQ09SRS5TTzEeMBwGA1UEAwwVQ2VydGhmaWNhdGUgQXV0aG9yaXR5MB4XDTE2
MDExMjA2NTkxMFoXDTM2MDExMjA2NTkxMFowOTEXMBUGA1UECgwOTEFCLkxJQkNP
UkUuU08xHjAcBgNVBAMMFUNlcnRpZmlyYXRlIEF1dGhvcml0eTCCASlwDQYJKoZI
[...]
-----END CERTIFICATE-----

$ cat ca.crt | base64 -w 0
[...]
c1psWGpqeGIPQmNEWkJPMjJ5d0pDemVnR2QNCnRsbW9JdEF4YnFSdVd3PT0KLS0tLS1F
TkQgQ0VSVEIGSUNBVEUtLS0tLQo=
```

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:
c1psWGpqeGIPQmNEWkJPMjJ5d0pDemVnR2QNCnRsbW9JdEF4YnFSdVd3PT0KLS0tLS1F
TkQgQ0VSVEIGSUNBVEUtLS0tLQo=
```

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.

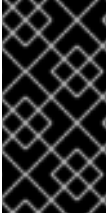
### 5.4. CONFIGURING OCI AND HELM WITH THE OPERATOR

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 3.7, **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:v3.7.10 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:v3.7.10 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**.

### Custom SSL Certificates

This section lists any custom or self-signed SSL certificates that are installed in the Quay container on startup after being read from the `extra_ca_certs` directory in the configuration volume.

Custom certificates are typically used in place of publicly signed certificates for corporate-internal services.

Please **make sure** that all custom names used for downstream services (such as Clair) are listed in the certificates below.

Upload certificates:

Select custom certificate to add to configuration. Must be in PEM format and end extension '.crt'

CERTIFICATE FILENAME	STATUS	NAMES HANDLED
extra_ca_certs/service-ca.crt	✔ Certificate is valid	openshift-service-serving-signer@1632474198

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

### Custom SSL Certificates

This section lists any custom or self-signed SSL certificates that are installed in the Quay container on startup after being read from the `extra_ca_certs` directory in the configuration volume.

Custom certificates are typically used in place of publicly signed certificates for corporate-internal services.

Please **make sure** that all custom names used for downstream services (such as Clair) are listed in the certificates below.

Upload certificates:

Select custom certificate to add to configuration. Must be in PEM format and end extension '.crt'

CERTIFICATE FILENAME	STATUS	NAMES HANDLED	
extra_ca_certs/service-ca.crt	✔ Certificate is valid	openshift-service-serving-signer@1632474198	⚙
extra_ca_certs/my-custom-ssl-cert.crt	✔ Certificate is valid	quay-server.example.com	⚙

## 7.2. BASIC CONFIGURATION

### Basic Configuration

Registry Title:

Name of registry to be displayed in the Contact Page.

Registry Title Short:

Enterprise Logo URL:

Enter the full URL to your company's logo.

Contact Information:

Information to show in the Contact Page. If none specified, CoreOS contact information is displayed.

### 7.2.1. Contact information

## Basic Configuration

**Registry Title:**

Name of registry to be displayed in the Contact Page.

**Registry Title Short:**

**Enterprise Logo URL:**

Enter the full URL to your company's logo.

**Contact Information:** URL

- E-mail
- IRC
- Telephone
- URL

Contact Page. If none specified, CoreOS contact information is displayed.

## Server Configuration

### 7.3. SERVER CONFIGURATION

#### Server Configuration

**Server Hostname:**

The HTTP host (and optionally the port number if a non-standard HTTP/HTTPS port) of the location where the registry will be accessible on the network.

**TLS:** None (Not For Production)

Running without TLS should not be used for production workloads!

#### 7.3.1. Server configuration choice

#### Server Configuration

**Server Hostname:**

The HTTP host (and optionally the port number if a non-standard HTTP/HTTPS port) of the location where the registry will be accessible on the network.

**TLS:** 
 Red Hat Quay handles TLS  
 My own load balancer handles TLS (Not Recommended)  
 None (Not For Production)

Enabling TLS also enables [HTTP Strict Transport Security](#). This prevents downgrade attacks and cookie theft, but browsers will reject all future insecure connections on this hostname.

**Certificate:** /conf/stack/ssl.cert Select a replacement file:  No file chosen  
 The certificate must be in PEM format.

**Private key:** /conf/stack/ssl.key Select a replacement file:  No file chosen

#### 7.3.2. TLS configuration

## Server Configuration

**Server Hostname:**

The HTTP host (and optionally the port number if a non-standard HTTP/HTTPS port) of the location where the registry will be accessible on the network.

**TLS:**

**i** Enabling TLS also enables [HTTP Strict Transport Security](#). This prevents downgrade attacks and cookie theft, but browsers will reject all future insecure connections on this hostname.

**Certificate:**  Select a replacement file:  No file chosen  
The certificate must be in PEM format.

**Private key:**  Select a replacement file:  No file chosen

## 7.4. DATABASE CONFIGURATION

You can choose between PostgreSQL and MySQL:

### Database

Quay uses a database as its primary metadata storage.

**Database Type:**  MySQL  Postgres

**Database Server:**

The server (and optionally, custom port) where the database lives

**Username:**

This user must have **full access** to the database

**Password:**

**Database Name:**

**SSL Certificate:**  No file chosen

Optional SSL certificate (in PEM format) to use to connect to the database



### 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:

## Database

---

Quay uses a database as its primary metadata storage.

**Database Type:**

**Database Server:**

The server (and optionally, custom port) where the database lives

**Username:**

This user must have **full access** to the database

**Password:**

**Database Name:**

**SSL Certificate:**  No file chosen

Optional SSL certicate (in PEM format) to use to connect 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

### Data Consistency Settings

---

Relax constraints on consistency guarantees for specific operations to enable higher performance and availability.

**Allow repository pulls even if audit logging fails.**

If enabled, failures to write to the audit log will fallback from the database to the standard logger for registry pulls.

## 7.6. TIME MACHINE CONFIGURATION



## Time Machine

Time machine keeps older copies of tags within a repository for the configured period of time, after which they are garbage collected. This allows users to revert tags to older images in case they accidentally pushed a broken image. It is highly recommended to have time machine enabled, but it does take a bit more space in storage.

**Allowed expiration periods:**

- 0s [Remove](#)
- 1d [Remove](#)
- 1w [Remove](#)
- 2w [Remove](#)
- 4w [Remove](#)

The expiration periods allowed for configuration. The default tag expiration \*must\* be in this list.

**Default expiration period:**

The default tag expiration period for all namespaces (users and organizations). Must be expressed in a duration string form: [30m](#), [1h](#), [1d](#), [2w](#).

**Allow users to select expiration:**

**Enable Expiration Configuration**

If enabled, users will be able to select the tag expiration duration for the namespace(s) they administrate, from the configured list of options.

## 7.7. REDIS CONFIGURATION

### Redis

A [redis](#) key-value store is required for real-time events and build logs.

**Redis Hostname:**

**Redis port:**

Access to this port and hostname must be allowed from all hosts running the enterprise registry


**Redis password:**

## 7.8. REPOSITORY MIRRORING CONFIGURATION

### Repository Mirroring

If enabled, scheduled mirroring of repositories from registries will be available.

**Enable Repository Mirroring**

 A repository mirror service must be running to use this feature. Documentation on setting up and running this service can be found at [Running Repository Mirroring Service](#).

**Require HTTPS and verify certificates of Quay registry during mirror.**

## 7.9. REGISTRY STORAGE CONFIGURATION

- Proxy storage
- Storage georeplication
- Storage engines

### 7.9.1. Enable storage replication - standalone Quay

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

#### Registry Storage

Registry images can be stored either locally or in a remote storage system.

**!** Do not use "Locally mounted directory" Storage Engine for any production configurations. Mounted NFS volumes are not supported. Local storage is meant for test-only installations.

#### Proxy storage via Quay

If enabled, all requests to storage engine(s) will be proxied through Quay . Should only be enabled if storage cannot be directly accessed by external nodes talking to the registry.

#### Enable Storage Replication

If enabled, replicates storage to other regions. See [documentation](#) for more information.

Location ID: default

Storage Engine:

Storage Directory:

### 7.9.2.2. Amazon S3 storage

<b>Location ID:</b>	default
<b>Storage Engine:</b>	<input type="text" value="Amazon S3"/>
<b>S3 Bucket:</b>	<input type="text" value="my-cool-bucket"/>
<b>Storage Directory:</b>	<input type="text" value="/datastorage/registry"/>
<b>AWS Access Key (optional if using IAM):</b>	<input type="text" value="accesskeyhere"/>
<b>AWS Secret Key (optional if using IAM):</b>	<input type="text" value="secretkeyhere"/>
<b>S3 Host:</b>	<input type="text" value="s3.amazonaws.com"/>
<b>S3 Port:</b>	<input type="text" value="443"/>

### 7.9.2.3. Azure blob storage

<b>Location ID:</b>	default
<b>Storage Engine:</b>	<input type="text" value="Azure Blob Storage"/>
<b>Azure Storage Container:</b>	<input type="text" value="container"/>
<b>Storage Directory:</b>	<input type="text" value="/datastorage/registry"/>
<b>Azure Account Name:</b>	<input type="text" value="accountnamehere"/>
<b>Azure Account Key:</b>	<input type="text" value="accountkeyhere"/>
<b>Azure SAS Token:</b>	<input type="text" value="sastokenhere"/>
<b>Azure Storage Endpoint URL:</b>	<input type="text" value="Optional, must include http(s)://"/>

### 7.9.2.4. Google cloud storage

<b>Location ID:</b>	default
<b>Storage Engine:</b>	<input type="text" value="Google Cloud Storage"/>
<b>Cloud Access Key:</b>	<input type="text" value="accesskeyhere"/>
<b>Cloud Secret Key:</b>	<input type="text" value="secretkeyhere"/>
<b>GCS Bucket:</b>	<input type="text" value="my-cool-bucket"/>
<b>Storage Directory:</b>	<input type="text" value="/datastorage/registry"/>

#### 7.9.2.5. Ceph object gateway (RADOS) storage

<b>Location ID:</b>	default
<b>Storage Engine:</b>	<input type="text" value="Ceph Object Gateway (RADOS)"/>
<b>Rados Server Hostname:</b>	<input type="text" value="my.rados.hostname"/>
<b>Custom Port (optional):</b>	<input type="text" value="443"/>
<b>Is Secure:</b>	<input type="checkbox"/> <b>Require SSL</b>
<b>Access Key:</b>	<input type="text" value="accesskeyhere"/> See <a href="#">Documentation</a> for more information
<b>Secret Key:</b>	<input type="text" value="secretkeyhere"/>
<b>Bucket Name:</b>	<input type="text" value="my-cool-bucket"/>
<b>Storage Directory:</b>	<input type="text" value="/datastorage/registry"/>

#### 7.9.2.6. OpenStack (Swift) storage configuration

**Location ID:** default

**Storage Engine:**

**Swift Auth Version:**

**Swift Auth URL:**

**Swift Container Name:**   
The swift container for all objects. Must already exist inside Swift.

**Storage Path:**

**Username:**   
Note: For Swift V1, this is "username:password" (-U on the CLI).

**Key/Password:**   
Note: For Swift V1, this is the API token (-K on the CLI).

**CA Cert Filename:**

**Temp URL Key (optional):**   
If enabled, will allow for faster pulls directly from Swift.  
See [Documentation](#) for more information

**OS Options:** No entries defined

---

Add Key-Value:

### 7.9.2.7. Cloudfront + Amazon S3 storage configuration

Location ID:	default
Storage Engine:	<input type="text" value="CloudFront + Amazon S3"/>
S3 Bucket:	<input type="text" value="my-cool-bucket"/>
Storage Directory:	<input type="text" value="/datastorage/registry"/>
AWS Access Key (optional if using IAM):	<input type="text" value="accesskeyhere"/>
AWS Secret Key (optional if using IAM):	<input type="text" value="secretkeyhere"/>
S3 Host:	<input type="text" value="s3.amazonaws.com"/>
S3 Port:	<input type="text" value="443"/>
CloudFront Distribution Domain Name:	<input type="text" value="somesubdomain.cloudfront.net"/>
CloudFront Key ID:	<input type="text" value="APKATHISISAKEYID"/>
CloudFront Private Key:	Please select a file to upload as <b>default-cloudfront-signing-key.pem</b> : <input type="button" value="Choose file"/> No file chosen

## 7.10. ACTION LOG CONFIGURATION

### 7.10.1. Action log storage configuration

#### 7.10.1.1. Database action log storage

##### Action Log Storage Configuration

Action logs can be stored in the database or Elasticsearch. In the latter case, the actions logs can (optionally) be sent to a data stream first.

Logs storage:

#### 7.10.1.2. Elasticsearch action log storage

## 📄 Action Log Storage Configuration

Action logs can be stored in the database or Elasticsearch. In the latter case, the actions logs can (optionally) be sent to a data stream first.

Logs storage:

### Elasticsearch config

Elasticsearch hostname:

Elasticsearch port:

Access to this port and hostname must be allowed from all hosts running the enterprise registry

Elasticsearch access key:

Elasticsearch secret key:

AWS region:

Index prefix:

Logs Producer:

## 7.10.2. Action log rotation and archiving

### 📄 Action Log Rotation and Archiving

All actions performed in Quay are automatically logged. These logs are stored in a database table, which can become quite large. Enabling log rotation and archiving will move all logs older than 30 days into storage.

Enable Action Log Rotation

Storage location:

The storage location in which to place archived action logs. Logs will only be archived to this single location.

Storage path:

The path under the configured storage engine in which to place the archived logs in JSON form.

Log Rotation Threshold:

The number of days after which to archive action logs to storage. Must be expressed in a duration string form: **30m**, **1h**, **1d**, **2w**.

Note: The rotation threshold should be considered a balancing act between user history and size of database.

Larger time windows will result in more logs, but give users more history to view. Anything less than **2w** is not recommended.

### 📄 Action Log Rotation and Archiving

All actions performed in Quay are automatically logged. These logs are stored in a database table, which can become quite large. Enabling log rotation and archiving will move all logs older than 30 days into storage.

Enable Action Log Rotation

Storage location:


The storage location in which to place archived action logs. Logs will only be archived to this single location.

## 7.11. SECURITY SCANNER CONFIGURATION

## Security Scanner

If enabled, all images pushed to Quay will be scanned via the external security scanning service, with vulnerability information available in the UI and API, as well as async notification support.

Enable Security Scanning

 A scanner compliant with the Quay Security Scanning API must be running to use this feature. Documentation on running Clair can be found at [Running Clair Security Scanner](#).

**Security Scanner Endpoint:**

The HTTP URL at which the security scanner is running.

**Security Scanner PSK:**

Clair Pre-Shared Key. Make sure to include this value in your Clair config.

## 7.12. APPLICATION REGISTRY CONFIGURATION

### Application Registry

If enabled, an additional registry API will be available for managing applications (Kubernetes manifests, Helm charts) via the [App Registry specification](#). A great place to get started is to install the [Helm Registry Plugin](#).

Enable App Registry

## 7.13. EMAIL CONFIGURATION

### E-mail

Valid e-mail server configuration is required for notification e-mails and the ability of users to reset their passwords.

Enable E-mails

**SMTP Server:**

**SMTP Server Port:**

**TLS:**  Require TLS

**Mail Sender:**

E-mail address from which all e-mails are sent. If not specified, `support@quay.io` will be used.

**Authentication:**  Requires Authentication

**Username:**

**Password:**

## 7.14. INTERNAL AUTHENTICATION CONFIGURATION



## Internal Authentication

Authentication for the registry can be handled by either the registry itself, LDAP, Keystone, or external JWT endpoint.

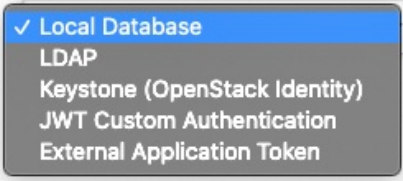
Additional **external** authentication providers (such as GitHub) can be used in addition for **login into the UI**.

Authentication:

## Internal Authentication

Authentication for the registry can be handled by either the registry itself, LDAP, Keystone, or external JWT endpoint.

Additional **external** authentication providers (such as GitHub) can be used in addition for **login into the UI**.

Authentication: 

### 7.14.1. LDAP

Authentication:


Team synchronization:  Enable Team Synchronization Support  
If enabled, organization administrators who are also superusers can set teams to have their membership synchronized with a backing group in LDAP.

LDAP URI:   
The full LDAP URI, including the ldap:// or ldaps:// prefix.


Base DN:   
A Distinguished Name path which forms the base path for looking up all LDAP records.  
 Example: dc=my,dc=domain,dc=com

User Relative DN:   
A Distinguished Name path which forms the base path for looking up all user LDAP records, relative to the Base DN defined above.  
 Example: ou=employees

Secondary User Relative DNs: No RDNs defined  
   
A list of Distinguished Name path(s) which forms the secondary base path(s) for looking up all user LDAP records, relative to the Base DN defined above. These path(s) will be tried if the user is not found via the primary relative DN.  
 Example: [ou=employees]

Additional User Filter Expression:  **NOTE:** This query is added **unescaped** to user lookups, so be **VERY** careful with the query you specify.  
  
If specified, the additional filter used for all user lookup queries. Note that all Distinguished Names used in the filter must be full paths; the `base_dn` is not added automatically here. Must be wrapped in parens.  
 Example: (someOtherField=someOtherValue)  
 Example: (memberOf=some.full.path.to.a.group)  
 Example: ((someFirstField=someValue)(someOtherField=someOtherValue))  
 Example: (&(someFirstField=someValue)(someOtherField=someOtherValue))

Administrator DN:   
The Distinguished Name for the Administrator account. This account must be able to login and view the records for all user accounts.  
 Example: uid=admin,ou=employees,dc=my,dc=domain,dc=com

Administrator DN Password:  **Note:** This will be stored in **plaintext** inside the config.yaml, so setting up a dedicated account or using a **password hash** is **highly** recommended.  
  
The password for the Administrator DN.

UID Attribute:   
The name of the property field in your LDAP user records that stores your users' username. Typically "uid".

Mail Attribute:   
The name of the property field in your LDAP user records that stores your users' e-mail address(es). Typically "mail".

### 7.14.2. Keystone (OpenStack identity)

## Internal Authentication

Authentication for the registry can be handled by either the registry itself, LDAP, Keystone, or external JWT endpoint.

Additional **external** authentication providers (such as GitHub) can be used in addition for **login into the UI**.

**It is highly recommended** to require encrypted client passwords. External passwords used in the Docker client will be stored in **plaintext!** [Enable this requirement now.](#)

**Authentication:**

**Team synchronization:**  **Enable Team Synchronization Support**  
 If enabled, organization administrators who are also superusers can set teams to have their membership synchronized with a backing group in Keystone.

**Keystone API Version:**

**Keystone Authentication URL:**   
 The URL (starting with http or https) of the Keystone Server endpoint for auth.

**Keystone Administrator Username:**   
 The username for the Keystone admin.

**Keystone Administrator Password:**   
 The password for the Keystone admin.

**Keystone Administrator Tenant:**   
 The tenant (project/group) that contains the administrator user.

### 7.14.3. JWT custom authentication

**Authentication:**

**JSON Web Token authentication** allows your organization to provide an HTTP endpoint that verifies user credentials on behalf of Quay . Documentation on the API required can be found here: <https://github.com/coreos/jwt-auth-example>.

**Authentication Issuer:**   
 The id of the issuer signing the JWT token. Must be unique to your organization.

**Public Key:** Please select a file to upload as **jwt-authn.cert**:  No file chosen  
 A certificate containing the public key portion of the key pair used to sign the JSON Web Tokens. This file must be in PEM format.

**User Verification Endpoint:**   
 The URL (starting with http or https) on the JWT authentication server for verifying username and password credentials. Credentials will be sent in the **Authorization** header as Basic Auth, and this endpoint should return **200 OK** on success (or a **4\*\*** otherwise).

**User Query Endpoint:**   
 The URL (starting with http or https) on the JWT authentication server for looking up users based on a prefix query. This is optional. The prefix query will be sent as a query parameter with name **query**.

**User Lookup Endpoint:**   
 The URL (starting with http or https) on the JWT authentication server for looking up a user by username or email address. The username or email address will be sent as a query parameter with name **username**.

### 7.14.4. External application token

## Internal Authentication

Authentication for the registry can be handled by either the registry itself, LDAP, Keystone, or external JWT endpoint. Additional **external** authentication providers (such as GitHub) can be used in addition for **login into the UI**.

**Authentication:**

## 7.15. EXTERNAL AUTHENTICATION (OAUTH) CONFIGURATION

### 7.15.1. GitHub (Enterprise) authentication

#### External Authorization (OAuth)

##### GitHub (Enterprise) Authentication

If enabled, users can use GitHub or GitHub Enterprise to authenticate to the registry.

**Note:** A registered GitHub (Enterprise) OAuth application is required. View instructions on how to [Create an OAuth Application in GitHub](#)

**Enable GitHub Authentication**

**GitHub:**   
 GitHub Enterprise

**GitHub Endpoint:**

The GitHub Enterprise endpoint. Must start with http:// or https://.

**OAuth Client ID:**

**OAuth Client Secret:**

**Organization Filtering:**  **Restrict By Organization Membership**

If enabled, only members of specified GitHub Enterprise organizations will be allowed to login via GitHub Enterprise.

### 7.15.2. Google authentication

## External Authorization (OAuth)

### GitHub (Enterprise) Authentication

If enabled, users can use GitHub or GitHub Enterprise to authenticate to the registry.

**Note:** A registered GitHub (Enterprise) OAuth application is required. View instructions on how to [Create an OAuth Application in GitHub](#)

Enable GitHub Authentication

### Google Authentication

If enabled, users can use Google to authenticate to the registry.

**Note:** A registered Google OAuth application is required. Visit the [Google Developer Console](#) to register an application.

Enable Google Authentication

OAuth Client ID:

OAuth Client Secret:

## 7.16. ACCESS SETTINGS CONFIGURATION

### Access Settings

Various settings around access and authentication to the registry.

**Basic Credentials**  **Login to User Interface via credentials**

**Login:**

If enabled, users will be able to login to the user interface via their username and password credentials.

If disabled, users will only be able to login to the user interface via one of the configured External Authentication providers.

**External Application tokens**  **Allow external application tokens**

If enabled, users will be able to generate external application tokens for use on the Docker and rkt CLI. Note that these tokens will not be required unless "App Token" is chosen as the Internal Authentication method above.

**External application token expiration**

The expiration time for user generated external application tokens. If none, tokens will never expire.

**Anonymous Access:**  **Enable Anonymous Access**

If enabled, public repositories and search can be accessed by anyone that can reach the registry, even if they are not authenticated. Disable to only allow authenticated users to view and pull "public" resources.

**User Creation:**  **Enable Non-Superuser User Creation**

If enabled, user accounts can be created by anyone (unless restricted below to invited users). Users can always be created in the users panel in this superuser tool, even if this feature is disabled. If disabled, users can **ONLY** be created in the superuser tool or via team sync.

**Invite-only User Creation:**  **Enable Invite-only User Creation**

If enabled, user accounts can only be created when a user has been invited, by e-mail address, to join a team. Users can always be created in the users panel in this superuser tool, even if this feature is enabled.

**Encrypted Client Password:**  **Require Encrypted Client Passwords**

If enabled, users will not be able to login from the Docker command line with a non-encrypted password and must generate an encrypted password to use. This feature is highly recommended for setups with external authentication, as Docker currently stores passwords in plaintext on user's machines.

**Prefix username autocompletion:**  **Allow prefix username autocompletion**

If disabled, autocompletion for users will only match on exact usernames.

**Team Invitations:**  **Require Team Invitations**

If enabled, when adding a new user to a team, they will receive an invitation to join the team, with the option to decline. Otherwise, users will be immediately part of a team when added by a team administrator.

**Super Users:**

- quayadmin [Remove](#)

Users included in this list will be given elevated access to Quay.

## 7.17. DOCKERFILE BUILD SUPPORT

## ☰ Dockerfile Build Support

If enabled, users can submit Dockerfile's to be built and pushed by Quay .

**Enable Dockerfile Build**

**Note:** Build workers are required for this feature. See [Adding Build Workers](#) for instructions on how to setup build workers.

## 🔗 GitHub (Enterprise) Build Triggers

If enabled, users can setup GitHub or GitHub Enterprise triggers to invoke Registry builds.

**Note:** A registered GitHub (Enterprise) OAuth application (**separate from GitHub Authentication**) is required. View instructions on how to [Create an OAuth Application in GitHub](#)

**Enable GitHub Triggers**

## 🔑 BitBucket Build Triggers

If enabled, users can setup BitBucket triggers to invoke Registry builds.

**Note:** A registered BitBucket OAuth application is required.

**Enable BitBucket Triggers**

## 🚀 GitLab Build Triggers

If enabled, users can setup GitLab triggers to invoke Registry builds.

**Note:** A registered GitLab OAuth application is required. Visit the [GitLab applications admin panel](#) to create a new application.

The callback URL to use is: `https://quay-server.example.com/oauth2/gitlab/callback/trigger`

**Enable GitLab Triggers**

## 7.17.1. GitHub (Enterprise) Build Triggers

### 🔗 GitHub (Enterprise) Build Triggers

If enabled, users can setup GitHub or GitHub Enterprise triggers to invoke Registry builds.

**Note:** A registered GitHub (Enterprise) OAuth application (**separate from GitHub Authentication**) is required. View instructions on how to [Create an OAuth Application in GitHub](#)

**Enable GitHub Triggers**

**GitHub:**

**GitHub Endpoint:**

The GitHub Enterprise endpoint. Must start with http:// or https://.

**OAuth Client ID:**

**OAuth Client Secret:**

## 7.17.2. BitBucket Build Triggers

## BitBucket Build Triggers

---

If enabled, users can setup BitBucket triggers to invoke Registry builds.

**Note:** A registered BitBucket OAuth application is required.

### Enable BitBucket Triggers

OAuth Consumer Key:

OAuth Consumer Secret:

## 7.17.3. GitLab Build Triggers

### GitLab Build Triggers

---

If enabled, users can setup GitLab triggers to invoke Registry builds.

**Note:** A registered GitLab OAuth application is required. Visit the [GitLab applications admin panel](#) to create a new application.

The callback URL to use is: `https://quay-server.example.com/oauth2/gitlab/callback/trigger`

### Enable GitLab Triggers

GitLab:

GitLab CE/EE



GitLab Endpoint:

https://my.gitlabserver

The GitLab Enterprise endpoint. Must start with http:// or https://.

Application Id:

Secret: