Red Hat Quay 3.8

Configure Red Hat Quay

Customizing Red Hat Quay using configuration options

Last Updated: 2024-01-15

Red Hat Quay 3.8 Configure Red Hat Quay

Customizing Red Hat Quay using configuration options

Legal Notice

Abstract

Configure Red Hat Quay

Table of Contents

CHAPTER 1. GETTING STARTED WITH RED HAT QUAY CONFIGURATION 6
1.1. CONFIGURATION UPDATES FOR QUAY 3.8 6
1.2. CONFIGURATION UPDATES FOR QUAY 3.7 8
1.2.1. New configuration fields for Red Hat Quay 3.7.7 8
1.2.2. New configuration fields 8
1.3. CONFIGURATION UPDATES FOR RED HAT QUAY 3.6 8
1.3.1. New configuration fields 9
1.3.2. Deprecated configuration fields 9
1.4. EDITING THE CONFIGURATION FILE 9
1.5. LOCATION OF CONFIGURATION FILE IN A STANDALONE DEPLOYMENT 10
1.6. MINIMAL CONFIGURATION 10
1.6.1. Sample minimal configuration file 10
1.6.2. Local storage 11
1.6.3. Cloud storage 11

CHAPTER 2. CONFIGURATION FIELDS 13
2.1. REQUIRED CONFIGURATION FIELDS 13
2.2. AUTOMATION OPTIONS 13
2.3. OPTIONAL CONFIGURATION FIELDS 13
2.4. GENERAL REQUIRED FIELDS 14
2.5. DATABASE CONFIGURATION 15
2.5.1. Database URI 15
2.5.2. Database connection arguments 15
2.5.2.1. PostgreSQL SSL connection arguments 16
2.5.2.2. MySQL SSL connection arguments 17
2.6. IMAGE STORAGE 17
2.6.1. Image storage features 17
2.6.2. Image storage configuration fields 18
2.6.3. Local storage 19
2.6.4. OCS/NooBaa 19
2.6.5. Ceph / RadosGW Storage / Hitachi HCP 19
2.6.6. AWS S3 storage 20
2.6.7. Google Cloud Storage 20
2.6.8. Azure Storage 20
2.6.9. Swift storage 21
2.7. REDIS CONFIGURATION FIELDS 21
2.7.1. Build logs 21
2.7.2. User events 22
2.7.3. Example Redis configuration 23
2.8. MODELCACHE CONFIGURATION OPTIONS 24
2.8.1. Memcache configuration option 24
2.8.2. Single Redis configuration option 24
2.8.3. Clustered Redis configuration option 24
2.9. TAG EXPIRATION CONFIGURATION FIELDS 25
2.9.1. Example tag expiration configuration 25
2.10. PRE-CONFIGURING RED HAT QUAY FOR AUTOMATION 25
2.10.1. Allowing the API to create the first user 25
2.10.2. Enabling general API access 26
2.10.3. Adding a superuser 26
2.10.4. Restricting user creation 26
2.10.5. Enabling new functionality 26

2.10.6. Enabling new functionality 27
2.10.7. Suggested configuration for automation 27
2.10.8. Deploying the Red Hat Quay Operator using the initial configuration 27
2.10.9. Using the API to deploy Red Hat Quay 28
2.10.9.1. Using the API to create the first user 28
2.10.9.2. Using the OAuth token 29
2.10.9.3. Using the API to create an organization 30
2.11. BASIC CONFIGURATION FIELDS 31
2.12. SSL CONFIGURATION FIELDS 33
2.12.1. Configuring SSL 34
2.13. ADDING TLS CERTIFICATES TO THE RED HAT QUAY CONTAINER 34
2.13.1. Add TLS certificates to Red Hat Quay 34
2.14. LDAP CONFIGURATION FIELDS 35
2.14.1. LDAP configuration field references 37
2.14.1.1. Basic LDAP user configuration 37
2.14.1.2. LDAP restricted user configuration 38
2.14.1.3. LDAP superuser configuration reference 38
2.15. MIRRORING CONFIGURATION FIELDS 39
2.16. SECURITY SCANNER CONFIGURATION FIELDS 39
2.17. OCI AND HELM CONFIGURATION FIELDS 41
2.18. ACTION LOG CONFIGURATION FIELDS 42
2.18.1. Action log storage configuration 42
2.18.2. Action log rotation and archiving configuration 44
2.19. BUILD LOGS CONFIGURATION FIELDS 44
2.20. DOCKERFILE BUILD TRIGGERS FIELDS 45
2.20.1. GitHub build triggers 45
2.20.2. BitBucket build triggers 46
2.20.3. GitLab build triggers 47
2.21. OAUTH CONFIGURATION FIELDS 47
2.21.1. GitHub OAuth configuration fields 47
2.21.2. Google OAuth configuration fields 48
2.22. OIDC CONFIGURATION FIELDS 49
2.22.1. OIDC configuration 51
2.23. NESTED REPOSITORIES CONFIGURATION FIELDS 51
2.24. ADDING OTHER OCI MEDIA TYPES TO QUAY 51
2.25. MAIL CONFIGURATION FIELDS 52
2.26. USER CONFIGURATION FIELDS 53
2.26.1. User configuration fields references 55
2.26.1.1. FEATURE_SUPERUSERS_FULL_ACCESS configuration reference 55
2.26.1.2. GLOBAL_READONLY_SUPER_USERS configuration reference 55
2.26.1.3. FEATURE_RESTRICTED_USERS configuration reference 55
2.26.1.4. RESTRICTED_USERS_WHITELIST configuration reference 56
2.27. RECAPTCHA CONFIGURATION FIELDS 56
2.28. ACI CONFIGURATION FIELDS 56
2.29. JWT CONFIGURATION FIELDS 57
2.30. APP TOKENS CONFIGURATION FIELDS 57
2.31. MISCELLANEOUS CONFIGURATION FIELDS 58
2.31.1. Miscellaneous configuration field references 60
2.31.1.1. v2 user interface configuration 60
2.31.1.1.1. Creating a new organization in the Red Hat Quay 3.8 beta UI 61
2.31.1.1.2. Deleting an organization using the Red Hat Quay 3.8 beta UI 62
2.31.1.1.3. Creating a new repository using the Red Hat Quay 3.8 beta UI 62
2.31.1.1.4. Deleting a repository using the Red Hat Quay 3.8 beta UI 62

2.31.1.1.5. Pushing an image to the Red Hat Quay 3.8 beta UI 63
2.31.1.1.6. Deleting an image using the Red Hat Quay 3.8 beta UI 63
2.31.1.1.7. Enabling the Red Hat Quay legacy UI 64
2.32. LEGACY CONFIGURATION FIELDS 64
2.33. USER INTERFACE V2 CONFIGURATION FIELD 65
2.34. IPV6 CONFIGURATION FIELD 65
2.35. BRANDING CONFIGURATION FIELDS 65
2.35.1. Example configuration for Red Hat Quay branding 66
2.36. SESSION TIMEOUT CONFIGURATION FIELD 66
2.36.1. Example session timeout configuration 67

CHAPTER 3. ENVIRONMENT VARIABLES 68
3.1. GEO-REPLICATION 68
3.2. DATABASE CONNECTION POOLING 68
3.3. HTTP CONNECTION COUNTS 69
3.4. WORKER COUNT VARIABLES 69
3.5. DEBUG VARIABLES 70

CHAPTER 4. USING THE CONFIG TOOL TO RECONFIGURE QUAY ON OPENSHIFT 72
4.1. ACCESSING THE CONFIG EDITOR 72
4.1.1. Retrieving the config editor credentials 72
4.1.2. Logging in to the config editor 73
4.1.3. Changing configuration 74
4.2. MONITORING RECONFIGURATION IN THE UI 75
4.2.1. QuayRegistry resource 75
4.2.2. Events 77
4.3. ACCESSING UPDATED INFORMATION AFTER RECONFIGURATION 78
4.3.1. Accessing the updated config tool credentials in the UI 78
4.3.2. Accessing the updated config.yaml in the UI 78

CHAPTER 5. QUAY OPERATOR COMPONENTS 79
5.1. USING MANAGED COMPONENTS 79
5.2. USING UNMANAGED COMPONENTS FOR DEPENDENCIES 80
5.2.1. Using an existing Postgres database 80
5.2.2. NooBaa unmanaged storage 81
5.2.3. Horizontal Pod Autoscaler 81
5.2.3.1. Disabling the Horizontal Pod Autoscaler 81
5.3. ADD CERTS WHEN DEPLOYED ON KUBERNETES 82
5.4. CONFIGURING OCI AND HELM WITH THE OPERATOR 82
5.5. VOLUME SIZE OVERRIDES 83

CHAPTER 6. CLAIR FOR RED HAT QUAY 85
6.1. CLAIR CONFIGURATION OVERVIEW 85
6.1.1. Clair configuration reference 85
6.1.2. Clair general fields 86
6.1.3. Clair indexer configuration fields 87
6.1.4. Clair matcher configuration fields 89
6.1.5. Clair matchers configuration fields 90
6.1.6. Clair updaters configuration fields 91
6.1.7. Clair notifier configuration fields 92
6.1.8. Clair authorization configuration fields 96
6.1.9. Clair trace configuration fields 97
6.1.10. Clair metrics configuration fields 98

CHAPTER 7. SCANNING POD IMAGES WITH THE CONTAINER SECURITY OPERATOR 99
7.1. DOWNLOADING AND RUNNING THE CONTAINER SECURITY OPERATOR IN OPENSHIFT CONTAINER PLATFORM 99
7.2. QUERY IMAGE VULNERABILITIES FROM THE CLI 101

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 3.8

The following configuration fields have been introduced with Red Hat Quay 3.8:

Table 1.1. Red Hat Quay 3.8 configuration fields

Field	Type	Description
FEATURE_UI_V2	Boolean	When set, allows users to try the beta UI environment. **Default: False**
FEATURE_LISTEN_IP_VERSION	String	Enables IPv4, IPv6, or dual-stack protocol family. This configuration field must be properly set, otherwise Red Hat Quay fails to start. **Default: IPv4** **Additional configurations: IPv6, dual-stack**

Field	Type	Description
LDAP_SUPERUSER_FILTER	String	Subset of the **LDAP_USER_FILTER** configuration field. When configured, allows Red Hat Quay administrators the ability to configure Lightweight Directory Access Protocol (LDAP) users as superusers when Red Hat Quay uses LDAP as its authentication provider. With this field, administrators can add or remove superusers without having to update the Red Hat Quay configuration file and restart their deployment.
LDAP_RESTRICTED_USER_FILTER	String	Subset of the **LDAP_USER_FILTER** configuration field. When configured, allows Red Hat Quay administrators the ability to configure Lightweight Directory Access Protocol (LDAP) users as restricted users when Red Hat Quay uses LDAP as its authentication provider.
FEATURE_SUPERUSERS_FULL_ACCESS	Boolean	Grants superusers the ability to read, write, and delete content from other repositories in namespaces that they do not own or have explicit permissions for. Default: **False**
GLOBAL_READONLY_SUPER_USERS	String	When set, grants users of this list read access to all repositories, regardless of whether they are public repositories.
FEATURE_RESTRICTED_USERS	Boolean	When set with **RESTRICTED_USERS_WHITELIST**, restricted users cannot create organizations or content in their own namespace. Normal permissions apply for an organization's membership, for example, a restricted user will still have normal permissions in organizations based on the teams that they are members of. Default: **False**

Field	Type	Description
RESTRICTED_USERS_WHITELIST	String	When set with **FEATURE_RESTRICTED_USERS: true**, specific users are excluded from the **FEATURE_RESTRICTED_USERS** setting.

1.2. CONFIGURATION UPDATES FOR QUAY 3.7

1.2.1. New configuration fields for Red Hat Quay 3.7.7

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

1.2.2. New configuration fields

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

Parameter	Description
FEATURE_QUOTA_MANAGEMENT	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 Red Hat Quay Quota management and enforcement.
DEFAULT_SYSTEM_REJECT_QUOTA_BYTES	The quota size to apply to all organizations and users. For more information about quota management, see Red Hat Quay Quota management and enforcement
FEATURE_PROXY_CACHE	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 Red Hat Quay as proxy cache for upstream registries.

1.3. CONFIGURATION UPDATES FOR RED HAT QUAY 3.6

1.3.1. New configuration fields

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

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

1.3.2. Deprecated configuration fields

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

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

1.4. 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.

1.5. 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 \
  registry.redhat.io/quay/quay-rhel8:v3.8.15
```

1.6. 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

1.6.1. 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.

1.6.2. 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 \
  registry.redhat.io/quay/quay-rhel8:v3.8.15
```

1.6.3. 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 \
   registry.redhat.io/quay/quay-rhel8:v3.8.15
```

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
- User interface v2
- IPv6 configuration field

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
AUTHENTICATION_TYPE (Required)	String	The authentication engine to use for credential authentication. **Values:** One of **Database**, **LDAP**, **JWT**, **Keystone**, **OIDC** **Default: Database**
PREFERRED_URL_SCHEME (Required)	String	The URL scheme to use when accessing Red Hat Quay. **Values:** One of **http**, **https** **Default: http**
SERVER_HOSTNAME (Required)	String	The URL at which Red Hat Quay is accessible, without the scheme. **Example:** **quay-server.example.com**

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

2.5. DATABASE CONFIGURATION

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
DB_URI (Required)	String	The URI for accessing the database, including any credentials. Example **DB_URI** field: **postgresql://quayuser:quaypass@quay-server.example.com:5432/quay**

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
DB_CONNECTION_ARGS	Object	Optional connection arguments for the database, such as timeouts and SSL.
.autorollback	Boolean	Whether to use thread-local connections. Should always be **true**
.threadlocals	Boolean	Whether to use auto-rollback connections. Should always be **true**

2.5.2.1. PostgreSQL SSL connection arguments

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
disable	Your configuration only tries non-SSL connections.
allow	Your configuration first tries a non-SSL connection. Upon failure, tries an SSL connection.
prefer (Default)	Your configuration first tries an SSL connection. Upon failure, tries a non-SSL connection.
require	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.

Mode	Description
verify-ca	Your configuration only tries an SSL connection, and verifies that the server certificate is issued by a trusted certificate authority (CA).
verify-full	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
FEATURE_REPO_MIRROR	Boolean	If set to true, enables repository mirroring. **Default: false**
FEATURE_PROXY_STORAGE	Boolean	Whether to proxy all direct download URLs in storage through NGINX. **Default: false**

Field	Type	Description
FEATURE_STORAGE_REPLICATION	Boolean	Whether to automatically replicate between storage engines. Default: **false**

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
DISTRIBUTED_STORAGE_CONFIG (Required)	Object	Configuration for storage engine(s) to use in Red Hat Quay. Each key represents an unique identifier for a storage engine. The value consists of a tuple of (key, value) forming an object describing the storage engine parameters. Default: **[]**
DISTRIBUTED_STORAGE_DEFAULT_LOCATIONS (Required)	Array of string	The list of storage engine(s) (by ID in **DISTRIBUTED_STORAGE_CONFIG**) whose images should be fully replicated, by default, to all other storage engines.
DISTRIBUTED_STORAGE_PREFERENCE (Required)	Array of string	The preferred storage engine(s) (by ID in **DISTRIBUTED_STORAGE_CONFIG**) to use. A preferred engine means it is first checked for pulling and images are pushed to it. Default: **false**

Field	Type	Description
MAXIMUM_LAYER_SIZE	String	Maximum allowed size of an image layer. Pattern: **^[0-9]+(G|M)$** Example: **100G** Default: **20G**

2.6.3. Local storage

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

2.6.5. Ceph / RadosGW Storage / Hitachi HCP

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

Field	Type	Description
.password	String	The password to connect to the Redis instance. **Example:** **strongpassword**
.ssl (Optional)	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
USER_EVENTS_REDIS (Required)	Object	Redis connection details for user event handling.
.host (Required)	String	The hostname at which Redis is accessible. **Example:** **quay-server.example.com**
.port (Required)	Number	The port at which Redis is accessible. **Example:** **6379**
.password	String	The password to connect to the Redis instance. **Example:** **strongpassword**
.ssl	Boolean	Whether to enable TLS communication between Redis and Quay. Defaults to false.
.ssl_keyfile (Optional)	String	The name of the key database file, which houses the client certificate to be used. **Example:** **ssl_keyfile: /path/to/server/privatekey.pem**

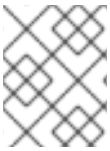
Field	Type	Description
.ssl_certfile (Optional)	String	Used for specifying the file path of the SSL certificate. **Example:** **ssl_certfile: /path/to/server/certificate.pem**
.ssl_cert_reqs (Optional)	String	Used to specify the level of certificate validation to be performed during the SSL/TLS handshake. **Example:** **ssl_cert_reqs: CERT_REQUIRED**
.ssl_ca_certs (Optional)	String	Used to specify the path to a file containing a list of trusted Certificate Authority (CA) certificates. **Example:** **ssl_ca_certs: /path/to/ca_certs.pem**
.ssl_ca_data (Optional)	String	Used to specify a string containing the trusted CA certificates in PEM format. **Example:** **ssl_ca_data: <certificate>**
.ssl_check_hostname (Optional)	Boolean	Used when setting up an SSL/TLS connection to a server. It specifies whether the client should check that the hostname in the server's SSL/TLS certificate matches the hostname of the server it is connecting to. **Example:** **ssl_check_hostname: true**

2.7.3. Example Redis configuration

The following YAML shows a sample configuration using Redis with optional SSL/TLS fields:

```
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
  ssl_*: <path_location_or_certificate>
```



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

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 superuser

After deploying Red Hat Quay, you can create a user. It is suggested 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 3.8 functionality, enable some or all of the following features:

```
...
FEATURE_UI_V2: true
FEATURE_LISTEN_IP_VERSION:
FEATURE_SUPERUSERS_FULL_ACCESS: true
GLOBAL_READONLY_SUPER_USERS:
    -
FEATURE_RESTRICTED_USERS: true
RESTRICTED_USERS_WHITELIST:
    -
...
```

2.10.6. 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
...
```

2.10.7. 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
...
```

2.10.8. 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

2.10.9. Using the API to deploy Red Hat Quay

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

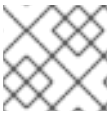
Prerequisites

- 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

2.10.9.1. 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**.

1. As the root user, install **python39** by entering the following command:

```
$ sudo yum install python39
```

2. Upgrade the **pip** package manager for Python 3.9:

```
$ python3.9 -m pip install --upgrade pip
```

3. Use the **pip** package manager to install the **bcrypt** package:

```
$ pip install bcrypt
```

4. Generate a secure, hashed password using the **bcrypt** package in Python 3.9 by entering the following command:

```
$ python3.9 -c 'import bcrypt; print(bcrypt.hashpw(b"subquay12345",
bcrypt.gensalt(12)).decode("utf-8"))'
```

5. Open your Red Hat Quay configuration file and update the following configuration fields:

```
FEATURE_USER_INITIALIZE: true
SUPER_USERS:
    - quayadmin
```

6. Stop the Red Hat Quay service by entering the following command:

```
$ sudo podman stop quay
```

7. Start the Red Hat Quay service by entering the following command:

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

8. Run the following **CURL** command to generate a new user with a username, password, email, and access token:

```
$ curl -X POST -k  http://quay-server.example.com/api/v1/user/initialize --header 'Content-
Type: application/json' --data '{ "username": "quayadmin", "password":"quaypass12345",
"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"} #
gitleaks:allow
```

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."}
```

9. Log in to your Red Hat Quay deployment by entering the following command:

```
$ sudo podman login -u quayadmin -p quaypass12345 http://quay-server.example.com --tls-
verify=false
```

Example output

```
Login Succeeded!
```

2.10.9.2. 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.

2.10.9.3. 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. Displayed in frontend of your Red Hat Quay deployment, for example, at the sign in page of your organization. Should not exceed 35 characters. **Default:** **Red Hat Quay**
REGISTRY_TITLE_SHORT	String	If specified, the short-form title for the registry. Title is displayed on various pages of your organization, for example, as the title of the tutorial on your organization's **Tutorial** page. **Default:** **Red Hat Quay**
CONTACT_INFO	Array of String	If specified, contact information to display on the contact page. If only a single piece of contact information is specified, the contact footer will link directly.
[0]	String	Adds a link to send an e-mail. **Pattern:** **^mailto:(.)+$** **Example:** **mailto:support@quay.io**
[1]	String	Adds a link to visit an IRC chat room. **Pattern:** **^irc://(.)+$** **Example:** **irc://chat.freenode.net:6665/quay**
[2]	String	Adds a link to call a phone number.+ **Pattern:** **^tel:(.)+$** **Example:** **tel:+1-888-930-3475**

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

2.12. SSL CONFIGURATION FIELDS

Table 2.11. SSL configuration

Field	Type	Description
PREFERRED_URL_SCHEME	String	One of **http** or **https**. Note that users only set their **PREFERRED_URL_SCHEME** to **http** when there is no TLS encryption in the communication path from the client to Quay. + Users must set their **PREFERRED_URL_SCHEME `to `https** 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 **PREFERRED_URL_SCHEME** should be **https**. **Default: http**
SERVER_HOSTNAME (Required)	String	The URL at which Red Hat Quay is accessible, without the scheme **Example:** **quay-server.example.com**
SSL_CIPHERS	Array of String	If specified, the nginx-defined list of SSL ciphers to enabled and disabled **Example:** [**CAMELLIA**, **!3DES**]

Field	Type	Description
SSL_PROTOCOLS	Array of String	If specified, nginx is configured to enabled a list of SSL protocols defined in the list. Removing an SSL protocol from the list disables the protocol during Red Hat Quay startup. **Example:** **['TLSv1','TLSv1.1','TLSv1.2', `TLSv1.3`]`**
SESSION_COOKIE_SECURE	Boolean	Whether the **secure** property should be set on session cookies **Default:** False **Recommendation:** Set to True for all installations using SSL

2.12.1. 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:v3.8.15 "/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
AUTHENTICATION_TYPE (Required)	String	Must be set to **LDAP**
FEATURE_TEAM_SYNCING	Boolean	Whether to allow for team membership to be synced from a backing group in the authentication engine (LDAP or Keystone) **Default: true**

Field	Type	Description
FEATURE_NONSUPERUSER_TEAM_SYNCING_SETUP	Boolean	If enabled, non-superusers can setup syncing on teams using LDAP **Default:** **false**
LDAP_ADMIN_DN	String	The admin DN for LDAP authentication.
LDAP_ADMIN_PASSWD	String	The admin password for LDAP authentication.
LDAP_ALLOW_INSECURE_FALLBACK	Boolean	Whether or not to allow SSL insecure fallback for LDAP authentication.
LDAP_BASE_DN	Array of String	The base DN for LDAP authentication.
LDAP_EMAIL_ATTR	String	The email attribute for LDAP authentication.
LDAP_UID_ATTR	String	The uid attribute for LDAP authentication.
LDAP_URI	String	The LDAP URI.
LDAP_USER_FILTER	String	The user filter for LDAP authentication.
LDAP_USER_RDN	Array of String	The user RDN for LDAP authentication.
TEAM_RESYNC_STALE_TIME	String	If team syncing is enabled for a team, how often to check its membership and resync if necessary **Pattern:** **^[0-9]+(w|m|d|h|s)$** **Example:** **2h** **Default:** **30m**

Field	Type	Description
LDAP_SUPERUSER_FILTER	String	Subset of the **LDAP_USER_FILTER** configuration field. When configured, allows Red Hat Quay administrators the ability to configure Lightweight Directory Access Protocol (LDAP) users as superusers when Red Hat Quay uses LDAP as its authentication provider. With this field, administrators can add or remove superusers without having to update the Red Hat Quay configuration file and restart their deployment. This field requires that your **AUTHENTICATION_TYPE** is set to **LDAP**.
LDAP_RESTRICTED_USER_FILTER	String	Subset of the **LDAP_USER_FILTER** configuration field. When configured, allows Red Hat Quay administrators the ability to configure Lightweight Directory Access Protocol (LDAP) users as restricted users when Red Hat Quay uses LDAP as its authentication provider. This field requires that your **AUTHENTICATION_TYPE** is set to **LDAP**.

2.14.1. LDAP configuration field references

Use the following references to update your **config.yaml** file with the desired configuration field.

2.14.1.1. Basic LDAP user configuration

```
---
AUTHENTICATION_TYPE: LDAP
---
LDAP_ADMIN_DN: uid=testuser,ou=Users,o=orgid,dc=jumpexamplecloud,dc=com
LDAP_ADMIN_PASSWD: samplepassword
LDAP_ALLOW_INSECURE_FALLBACK: false
LDAP_BASE_DN:
    - o=orgid
```

```
    - dc=example
    - dc=com
LDAP_EMAIL_ATTR: mail
LDAP_UID_ATTR: uid
LDAP_URI: ldap://ldap.example.com:389
LDAP_USER_RDN:
    - ou=Users
```

2.14.1.2. LDAP restricted user configuration

```
---
AUTHENTICATION_TYPE: LDAP
---
LDAP_ADMIN_DN: uid=<name>,ou=Users,o=<organization_id>,dc=
<example_domain_component>,dc=com
LDAP_ADMIN_PASSWD: ABC123
LDAP_ALLOW_INSECURE_FALLBACK: false
LDAP_BASE_DN:
    - o=<organization_id>
    - dc=<example_domain_component>
    - dc=com
LDAP_EMAIL_ATTR: mail
LDAP_UID_ATTR: uid
LDAP_URI: ldap://<example_url>.com
LDAP_USER_FILTER: (memberof=cn=developers,ou=Users,o=<example_organization_unit>,dc=
<example_domain_component>,dc=com)
LDAP_RESTRICTED_USER_FILTER: (<filterField>=<value>)
LDAP_USER_RDN:
    - ou=<example_organization_unit>
    - o=<organization_id>
    - dc=<example_domain_component>
    - dc=com
---
```

2.14.1.3. LDAP superuser configuration reference

```
---
AUTHENTICATION_TYPE: LDAP
---
LDAP_ADMIN_DN: uid=<name>,ou=Users,o=<organization_id>,dc=
<example_domain_component>,dc=com
LDAP_ADMIN_PASSWD: ABC123
LDAP_ALLOW_INSECURE_FALLBACK: false
LDAP_BASE_DN:
    - o=<organization_id>
    - dc=<example_domain_component>
    - dc=com
LDAP_EMAIL_ATTR: mail
LDAP_UID_ATTR: uid
LDAP_URI: ldap://<example_url>.com
LDAP_USER_FILTER: (memberof=cn=developers,ou=Users,o=<example_organization_unit>,dc=
<example_domain_component>,dc=com)
LDAP_SUPERUSER_FILTER: (<filterField>=<value>)
LDAP_USER_RDN:
```

```
- ou=<example_organization_unit>
- o=<organization_id>
- dc=<example_domain_component>
- dc=com
```

2.15. MIRRORING CONFIGURATION FIELDS

Table 2.13. Mirroring configuration

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

2.16. SECURITY SCANNER CONFIGURATION FIELDS

Table 2.14. Security scanner configuration

Field	Type	Description

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

Field	Type	Description
SECURITY_SCANNER_INDEXING_INTERVAL	Number	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. **Default:** 30

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
FEATURE_GENERAL_OCI_SUPPORT	Boolean	Enable support for OCI artifacts **Default:** True

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

IMPORTANT

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

2.18. 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 **Default:** **True**
LOGS_MODEL	String	Enable or disable the security scanner **Values:** One of **database**, **transition_reads_both_writes_es**, **elasticsearch** **Default:** **database**
LOGS_MODEL_CONFIG	Object	Logs model config for action logs

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

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

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

- **enum**: kafka, elasticsearch, kinesis_stream
- **Example**: **kafka**

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

2.18.2. Action log rotation and archiving configuration

Table 2.17. Action log rotation and archiving configuration

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

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

2.20. DOCKERFILE BUILD TRIGGERS FIELDS

Table 2.19. Dockerfile build support

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

2.20.1. GitHub build triggers

Table 2.20. GitHub build triggers

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

2.20.2. BitBucket build triggers

Table 2.21. BitBucket build triggers

Field	Type	Description
FEATURE_BITBUCKET_BUILD	Boolean	Whether to support Bitbucket build triggers **Default:** False
BITBUCKET_TRIGGER_CONFIG	Object	Configuration for using BitBucket for build triggers

Field	Type	Description
.CONSUMER_KEY (Required)	String	The registered consumer key (client ID) for this Quay instance
.CONSUMER_SECRET (Required)	String	The registered consumer secret (client secret) for this Quay instance

2.20.3. GitLab build triggers

Table 2.22. GitLab build triggers

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

2.21. OAUTH CONFIGURATION FIELDS

Table 2.23. OAuth fields

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

2.21.1. GitHub OAuth configuration fields

Table 2.24. GitHub OAuth fields

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

2.21.2. Google OAuth configuration fields

Table 2.25. Google OAuth fields

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

2.22. OIDC CONFIGURATION FIELDS

Table 2.26. OIDC fields

Field	Type	Description
<string>_LOGIN_CONFIG (Required)	String	The parent key that holds the OIDC configuration settings. Typically the name of the OIDC provider, for example, **AZURE_LOGIN_CONFIG**, however any arbitrary string is accepted.
.CLIENT_ID (Required)	String	The registered client ID for this Red Hat Quay instance. **Example:** **0e8dbe15c4c7630b6780**
.CLIENT_SECRET (Required)	String	The registered client secret for this Red Hat Quay instance. **Example:** **e4a58ddd3d7408b7aec109e85564a0d153d3e846**

.DEBUGLOG	Boolean	Whether to enable debugging.
.LOGIN_BINDING_FIELD	String	Used when the internal authorization is set to LDAP. Red Hat Quay reads this parameter and tries to search through the LDAP tree for the user with this username. If it exists, it automatically creates a link to that LDAP account.
.LOGIN_SCOPES	Object	Adds additional scopes that Red Hat Quay uses to communicate with the OIDC provider.
.OIDC_ENDPOINT_CUSTOM_PARAMS	String	Support for custom query parameters on OIDC endpoints. The following endpoints are supported: **authorization_endpoint**, **token_endpoint**, and **user_endpoint**.
.OIDC_ISSUER	String	Allows the user to define the issuer to verify. For example, JWT tokens container a parameter known as **iss** which defines who issued the token. By default, this is read from the **.well-know/openid/configuration** endpoint, which is exposed by every OIDC provider. If this verification fails, there is no login.
.OIDC_SERVER (Required)	String	The address of the OIDC server that is being used for authentication. **Example: https://sts.windows.net/6c878…/**
.PREFERRED_USERNAME_CLAIM_NAME	String	Sets the preferred username to a parameter from the token.
.SERVICE_ICON	String	Changes the icon on the login screen.
.SERVICE_NAME (Required)	String	The name of the service that is being authenticated. **Example: Azure AD**

.VERIFIED_EMAIL_CLAIM_NAME	String	The name of the claim that is used to verify the email address of the user.

2.22.1. OIDC configuration

The following example shows a sample OIDC configuration.

Example OIDC configuration

```
AZURE_LOGIN_CONFIG:
    CLIENT_ID: <client_id>
    CLIENT_SECRET: <client_secret>
    OIDC_SERVER: <oidc_server_address_>
    DEBUGGING: true
    SERVICE_NAME: Azure AD
    VERIFIED_EMAIL_CLAIM_NAME: <verified_email>
    OIDC_ENDPOINT_CUSTOM_PARAMS":
            "authorization_endpoint":
               "some": "param",
```

2.23. 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 2.27. OCI and nested repositories configuration fields

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

2.24. 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.

2.25. MAIL CONFIGURATION FIELDS

Table 2.28. Mail configuration fields

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

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

2.26. USER CONFIGURATION FIELDS

Table 2.29. User configuration fields

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

Field	Type	Description
FEATURE_USERNAME_CONFIRMATION	Boolean	If set to true, users can confirm and modify their initial usernames when logging in via OpenID Connect (OIDC) or a non-database internal authentication provider like LDAP. **Default:** **true**
FEATURE_USER_RENAME	Boolean	If set to true, users can rename their own namespace **Default:** **false**
FEATURE_INVITE_ONLY_USER_CREATION	Boolean	Whether users being created must be invited by another user **Default:** **false**
FRESH_LOGIN_TIMEOUT	String	The time after which a fresh login requires users to re-enter their password **Example**: **5m**
USERFILES_LOCATION	String	ID of the storage engine in which to place user-uploaded files **Example**: **s3_us_east**
USERFILES_PATH	String	Path under storage in which to place user-uploaded files **Example**: **userfiles**
USER_RECOVERY_TOKEN_LIFETIME	String	The length of time a token for recovering a user accounts is valid **Pattern**: **^[0-9]+(w|m|d|h|s)$** **Default**: **30m**
FEATURE_SUPERUSERS_FULL_ACCESS	Boolean	Grants superusers the ability to read, write, and delete content from other repositories in namespaces that they do not own or have explicit permissions for. **Default:** **False**

Field	Type	Description
FEATURE_RESTRICTED_USERS	Boolean	When set with **RESTRICTED_USERS_WHITELIST**, restricted users cannot create organizations or content in their own namespace. Normal permissions apply for an organization's membership, for example, a restricted user will still have normal permissions in organizations based on the teams that they are members of. Default: **False**
RESTRICTED_USERS_WHITELIST	String	When set with **FEATURE_RESTRICTED_USERS: true**, specific users are excluded from the **FEATURE_RESTRICTED_USERS** setting.
GLOBAL_READONLY_SUPER_USERS	String	When set, grants users of this list read access to all repositories, regardless of whether they are public repositories.

2.26.1. User configuration fields references

Use the following references to update your **config.yaml** file with the desired configuration field.

2.26.1.1. FEATURE_SUPERUSERS_FULL_ACCESS configuration reference

```
---
SUPER_USERS:
- quayadmin
FEATURE_SUPERUSERS_FULL_ACCESS: True
---
```

2.26.1.2. GLOBAL_READONLY_SUPER_USERS configuration reference

```
---
GLOBAL_READONLY_SUPER_USERS:
    - user1
---
```

2.26.1.3. FEATURE_RESTRICTED_USERS configuration reference

```
---
```

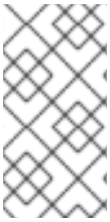
```
AUTHENTICATION_TYPE: Database
---
---
FEATURE_RESTRICTED_USERS: true
---
```

2.26.1.4. RESTRICTED_USERS_WHITELIST configuration reference

Prerequisites

- **FEATURE_RESTRICTED_USERS** is set to **true** in your **config.yaml** file.

```
---
AUTHENTICATION_TYPE: Database
---
---
FEATURE_RESTRICTED_USERS: true
RESTRICTED_USERS_WHITELIST:
    - user1
---
```



NOTE

When this field is set, whitelisted users can create organizations, or read or write content from the repository even if **FEATURE_RESTRICTED_USERS** is set to **true**. Other users, for example, **user2**, **user3**, and **user4** are restricted from creating organizations, reading, or writing content

2.27. RECAPTCHA CONFIGURATION FIELDS

Table 2.30. Recaptcha configuration fields

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

2.28. ACI CONFIGURATION FIELDS

Table 2.31. ACI configuration fields

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

2.29. JWT CONFIGURATION FIELDS

Table 2.32. JWT configuration fields

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

2.30. APP TOKENS CONFIGURATION FIELDS

Table 2.33. App tokens configuration fields

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

2.31. MISCELLANEOUS CONFIGURATION FIELDS

Table 2.34. Miscellaneous configuration fields

Field	Type	Description
ALLOW_PULLS_WITHOUT_STRICT_LOGGING	String	If true, pulls will still succeed even if the pull audit log entry cannot be written . This is useful if the database is in a read-only state and it is desired for pulls to continue during that time. **Default:** False
AVATAR_KIND	String	The types of avatars to display, either generated inline (local) or Gravatar (gravatar) **Values:** local, gravatar
BROWSER_API_CALLS_XHR_ONLY	Boolean	If enabled, only API calls marked as being made by an XHR will be allowed from browsers **Default:** True
DEFAULT_NAMESPACE_MAXIMUM_BUILD_COUNT	Number	The default maximum number of builds that can be queued in a namespace. **Default:** None

Field	Type	Description
ENABLE_HEALTH_DEBUG_SECRET	String	If specified, a secret that can be given to health endpoints to see full debug info when not authenticated as a superuser
EXTERNAL_TLS_TERMINATION	Boolean	Set to **true** if TLS is supported, but terminated at a layer before Quay. Set to **false** when Quay is running with its own SSL certificates and receiving TLS traffic directly.
FRESH_LOGIN_TIMEOUT	String	The time after which a fresh login requires users to re-enter their password **Example: 5m**
HEALTH_CHECKER	String	The configured health check **Example: ('RDSAwareHealthCheck', {'access_key': 'foo', 'secret_key': 'bar'})**
PROMETHEUS_NAMESPACE	String	The prefix applied to all exposed Prometheus metrics **Default: quay**
PUBLIC_NAMESPACES	Array of String	If a namespace is defined in the public namespace list, then it will appear on **all** users' repository list pages, regardless of whether the user is a member of the namespace. Typically, this is used by an enterprise customer in configuring a set of "well-known" namespaces.
REGISTRY_STATE	String	The state of the registry **Values: normal** or **read-only**
SEARCH_MAX_RESULT_PAGE_COUNT	Number	Maximum number of pages the user can paginate in search before they are limited **Default:** 10

Field	Type	Description
SEARCH_RESULTS_PER_PAGE	Number	Number of results returned per page by search page **Default:** 10
V2_PAGINATION_SIZE	Number	The number of results returned per page in V2 registry APIs **Default:** 50
WEBHOOK_HOSTNAME_BLACKLIST	Array of String	The set of hostnames to disallow from webhooks when validating, beyond localhost
CREATE_PRIVATE_REPO_ON_PUSH	Boolean	Whether new repositories created by push are set to private visibility **Default:** True
CREATE_NAMESPACE_ON_PUSH	Boolean	Whether new push to a non-existent organization creates it **Default:** False
NON_RATE_LIMITED_NAMESPACES	Array of String	If rate limiting has been enabled using **FEATURE_RATE_LIMITS**, you can override it for specific namespace that require unlimited access.
FEATURE_UI_V2	Boolean	When set, allows users to try the beta UI environment. **Default: True**

2.31.1. Miscellaneous configuration field references

Use the following references to update your **config.yaml** file with the desired configuration field.

2.31.1.1. v2 user interface configuration

With **FEATURE_UI_V2** enabled, you can toggle between the current version of the user interface and the new version of the user interface.

IMPORTANT

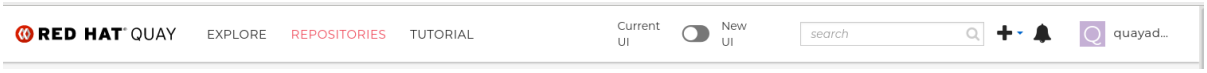
- This UI is currently in beta and subject to change. In its current state, users can only create, view, and delete organizations, repositories, and image tags.
- When running Red Hat Quay in the old UI, timed-out sessions would require that the user input their password again in the pop-up window. With the new UI, users are returned to the main page and required to input their username and password credentials. This is a known issue and will be fixed in a future version of the new UI.
- There is a discrepancy in how image manifest sizes are reported between the legacy UI and the new UI. In the legacy UI, image manifests were reported in mebibytes. In the new UI, Red Hat Quay uses the standard definition of megabyte (MB) to report image manifest sizes.

Procedure

1. In your deployment's **config.yaml** file, add the **FEATURE_UI_V2** parameter and set it to **true**, for example:

```
---
FEATURE_TEAM_SYNCING: false
FEATURE_UI_V2: true
FEATURE_USER_CREATION: true
---
```

2. Log in to your Red Hat Quay deployment.
3. In the navigation pane of your Red Hat Quay deployment, you are given the option to toggle between **Current UI** and **New UI**. Click the toggle button to set it to new UI, and then click **Use Beta Environment**, for example:



2.31.1.1.1. Creating a new organization in the Red Hat Quay 3.8 beta UI

Prerequisites

- You have toggled your Red Hat Quay deployment to use the 3.8 beta UI.

Use the following procedure to create an organization using the Red Hat Quay 3.8 beta UI.

Procedure

1. Click **Organization** in the navigation pane.
2. Click **Create Organization**.
3. Enter an **Organization Name**, for example, **testorg**.
4. Click **Create**.

Now, your example organization should populate under the **Organizations** page.

2.31.1.1.2. Deleting an organization using the Red Hat Quay 3.8 beta UI

Use the following procedure to delete an organization using the Red Hat Quay 3.8 beta UI.

Procedure

1. On the **Organizations** page, select the name of the organization you want to delete, for example, **testorg**.
2. Click the **More Actions** drop down menu.
3. Click **Delete**.

NOTE

On the **Delete** page, there is a **Search** input box. With this box, users can search for specific organizations to ensure that they are properly scheduled for deletion. For example, if a user is deleting 10 organizations and they want to ensure that a specific organization was deleted, they can use the **Search** input box to confirm said organization is marked for deletion.

4. Confirm that you want to permanently delete the organization by typing **confirm** in the box.
5. Click **Delete**.

After deletion, you are returned to the **Organizations** page.

NOTE

You can delete more than one organization at a time by selecting multiple organizations, and then clicking **More Actions → Delete**.

2.31.1.1.3. Creating a new repository using the Red Hat Quay 3.8 beta UI

Use the following procedure to create a repository using the Red Hat Quay 3.8 beta UI.

Procedure

1. Click **Repositories** on the navigation pane.
2. Click **Create Repository**.
3. Select a namespace, for example, **quayadmin**, and then enter a **Repository name**, for example, **testrepo**.
4. Click **Create**.

Now, your example repository should populate under the **Repositories** page.

2.31.1.1.4. Deleting a repository using the Red Hat Quay 3.8 beta UI

Prerequisites

- You have created a repository.

Procedure

1. On the **Repositories** page of the Red Hat Quay 3.8 beta UI, click the name of the image you want to delete, for example, **quay/admin/busybox**.
2. Click the **More Actions** drop-down menu.
3. Click **Delete**.

NOTE

If desired, you could click **Make Public** or **Make Private**.

4. Type **confirm** in the box, and then click **Delete**.
5. After deletion, you are returned to the **Repositories** page.

2.31.1.1.5. Pushing an image to the Red Hat Quay 3.8 beta UI

Use the following procedure to push an image to the Red Hat Quay 3.8 beta UI.

Procedure

1. Pull a sample image from an external registry:

```
$ podman pull busybox
```

2. Tag the image:

```
$ podman tag docker.io/library/busybox quay-server.example.com/quayadmin/busybox:test
```

3. Push the image to your Red Hat Quay registry:

```
$ podman push quay-server.example.com/quayadmin/busybox:test
```

4. Navigate to the **Repositories** page on the Red Hat Quay UI and ensure that your image has been properly pushed.
5. You can check the security details by selecting your image tag, and then navigating to the **Security Report** page.

2.31.1.1.6. Deleting an image using the Red Hat Quay 3.8 beta UI

Use the following procedure to delete an image using theRed Hat Quay 3.8 beta UI.

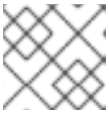
Prerequisites

- You have pushed an image to your Red Hat Quay registry.

Procedure

1. On the **Repositories** page of the Red Hat Quay 3.8 beta UI, click the name of the image you want to delete, for example, **quay/admin/busybox**.

2. Click the **More Actions** drop-down menu.
3. Click **Delete**.



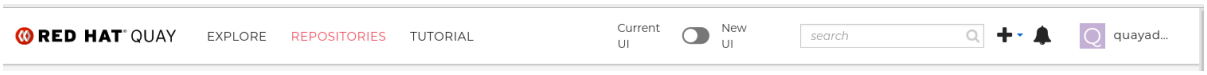
NOTE

If desired, you could click **Make Public** or **Make Private**.

4. Type **confirm** in the box, and then click **Delete**.
5. After deletion, you are returned to the **Repositories** page.

2.31.1.1.7. Enabling the Red Hat Quay legacy UI

1. In the navigation pane of your Red Hat Quay deployment, you are given the option to toggle between **Current UI** and **New UI**. Click the toggle button to set it to **Current UI**.



2.32. LEGACY CONFIGURATION FIELDS

Some fields are deprecated or obsolete:

Table 2.35. 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 FEATURE_BLACKLISTED_EMAILS is set to true **Example: "example.com", "example.org"**
BLACKLIST_V2_SPEC	String	The Docker CLI versions to which Red Hat Quay will respond that V2 is **unsupported** **Example: <1.8.0** **Default: <1.6.0**
DOCUMENTATION_ROOT	String	Root URL for documentation links
SECURITY_SCANNER_V4_NAMESPACE_WHITELIST	String	The namespaces for which the security scanner should be enabled

Field	Type	Description
FEATURE_RESTRICTED_V1_PUSH	Boolean	If set to true, only namespaces listed in V1_PUSH_WHITELIST support V1 push **Default:** True
V1_PUSH_WHITELIST	Array of String	The array of namespace names that support V1 push if FEATURE_RESTRICTED_V1_PUSH is set to true

2.33. USER INTERFACE V2 CONFIGURATION FIELD

Table 2.36. User interface v2 configuration field

Field	Type	Description
FEATURE_UI_V2	Boolean	When set, allows users to try the beta UI environment. **Default: False**

2.34. IPV6 CONFIGURATION FIELD

Table 2.37. IPv6 configuration field

Field	Type	Description
FEATURE_LISTEN_IP_VERSION	String	Enables IPv4, IPv6, or dual-stack protocol family. This configuration field must be properly set, otherwise Red Hat Quay fails to start. **Default: IPv4** **Additional configurations: IPv6, dual-stack**

2.35. BRANDING CONFIGURATION FIELDS

Table 2.38. Branding configuration fields

Field	Type	Description
BRANDING	Object	Custom branding for logos and URLs in the Red Hat Quay UI.

Field	Type	Description
.logo (Required)	String	Main logo image URL. The header logo defaults to 205x30 PX. The form logo on the Red Hat Quay sign in screen of the web UI defaults to 356.5x39.7 PX. **Example:** **/static/img/quay-horizontal-color.svg**
.footer_img	String	Logo for UI footer. Defaults to 144x34 PX. **Example:** **/static/img/RedHat.svg**
.footer_url	String	Link for footer image. **Example:** **https://redhat.com**

2.35.1. Example configuration for Red Hat Quay branding

Branding config.yaml example

```
BRANDING:
    logo: https://www.mend.io/wp-content/media/2020/03/5-tips_small.jpg
    footer_img: https://www.mend.io/wp-content/media/2020/03/5-tips_small.jpg
    footer_url: https://opensourceworld.org/
```

2.36. SESSION TIMEOUT CONFIGURATION FIELD

The following configuration field relies on on the Flask API configuration field of the same name.

Table 2.39. Session logout configuration field

Field	Type	Description

Field	Type	Description
PERMANENT_SESSION_LIFETIME	Integer	A **timedelta** which is used to set the expiration date of a permanent session. The default is 31 days, which makes a permanent session survive for roughly one month. Default: **2678400**

2.36.1. Example session timeout configuration

The following YAML is the suggest configuration when enabling session lifetime.

IMPORTANT

Altering session lifetime is not recommended. Administrators should be aware of the allotted time when setting a session timeout. If you set the time too early, it might interrupt your workflow.

Session timeout YAML configuration

```
PERMANENT_SESSION_LIFETIME: 3000
```

CHAPTER 3. ENVIRONMENT VARIABLES

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

3.1. GEO-REPLICATION

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

Table 3.1. Geo-replication configuration

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

3.2. DATABASE CONNECTION POOLING

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

If enabled, each process that interacts with the database will contain a connection pool. These per-process connection pools are configured to maintain a maximum of 20 connections. Under heavy load, it is possible to fill the connection pool for every process within a Red Hat Quay container. Under certain deployments and loads, this might require analysis to ensure that Red Hat Quay does not exceed the configured database's 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 can be toggled by setting the environment variable **DB_CONNECTION_POOLING** to **true** or **false**.

Table 3.2. Database connection pooling configuration

Variable	Type	Description
DB_CONNECTION_POOLING	Boolean	Enable or disable database connection pooling

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

config.yaml

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

3.3. HTTP CONNECTION COUNTS

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

Table 3.3. HTTP connection counts configuration

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

3.4. WORKER COUNT VARIABLES

Table 3.4. Worker count variables

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

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

3.5. DEBUG VARIABLES

The following debug variables are available on Red Hat Quay.

Table 3.5. Debug configuration variables

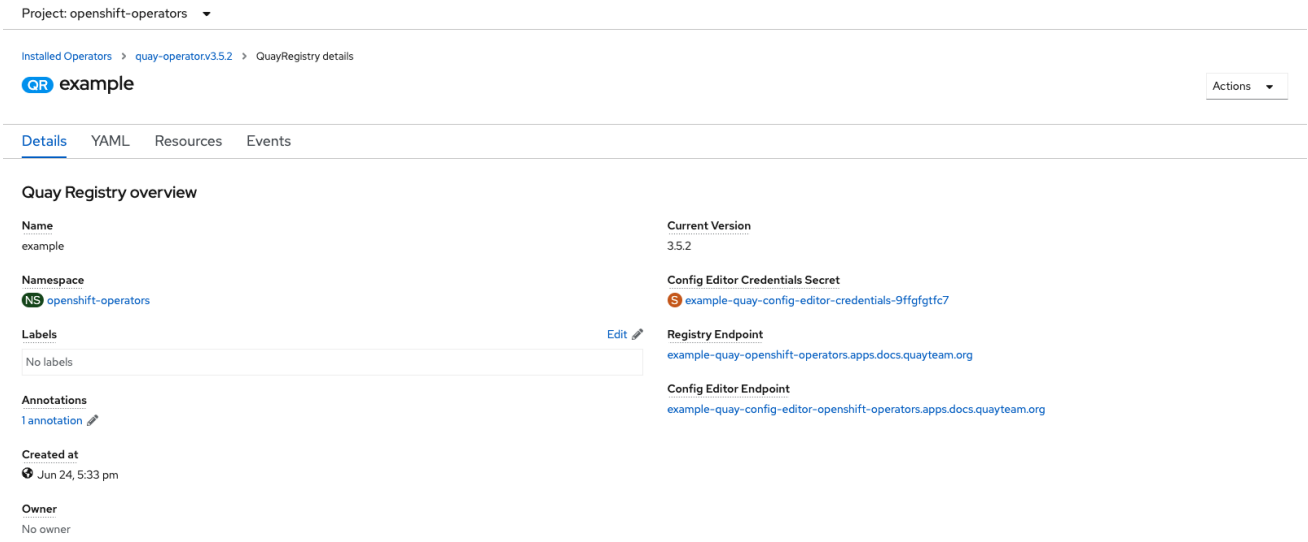
Variable	Type	Description
DEBUGLOG	Boolean	Whether to enable or disable debug logs.

Variable	Type	Description
USERS_DEBUG	Integer. Either **0** or **1**.	Used to debug LDAP operations in clear text, including passwords. Must be used with **DEBUGLOG=TRUE**.  **IMPORTANT** Setting **USERS_DEBUG=1** exposes credentials in clear text. This variable should be removed from the Red Hat Quay deployment after debugging. The log file that is generated with this environment variable should be scrutinized, and passwords should be removed before sending to other users. Use with caution.

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

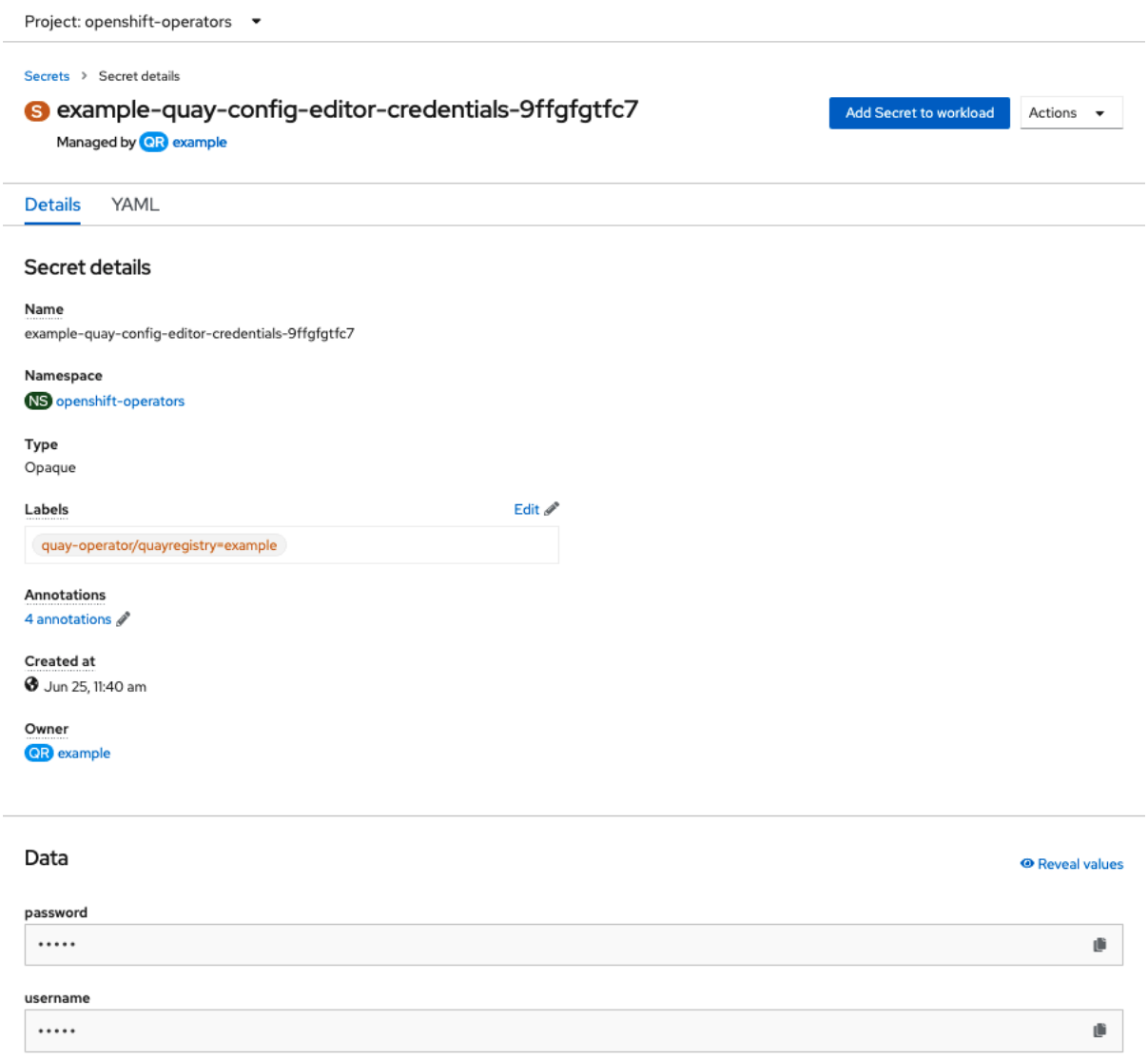
4.1. ACCESSING THE CONFIG EDITOR

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

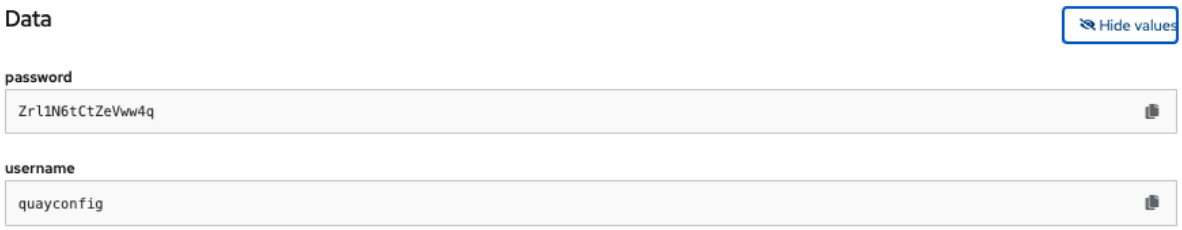


4.1.1. Retrieving the config editor credentials

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

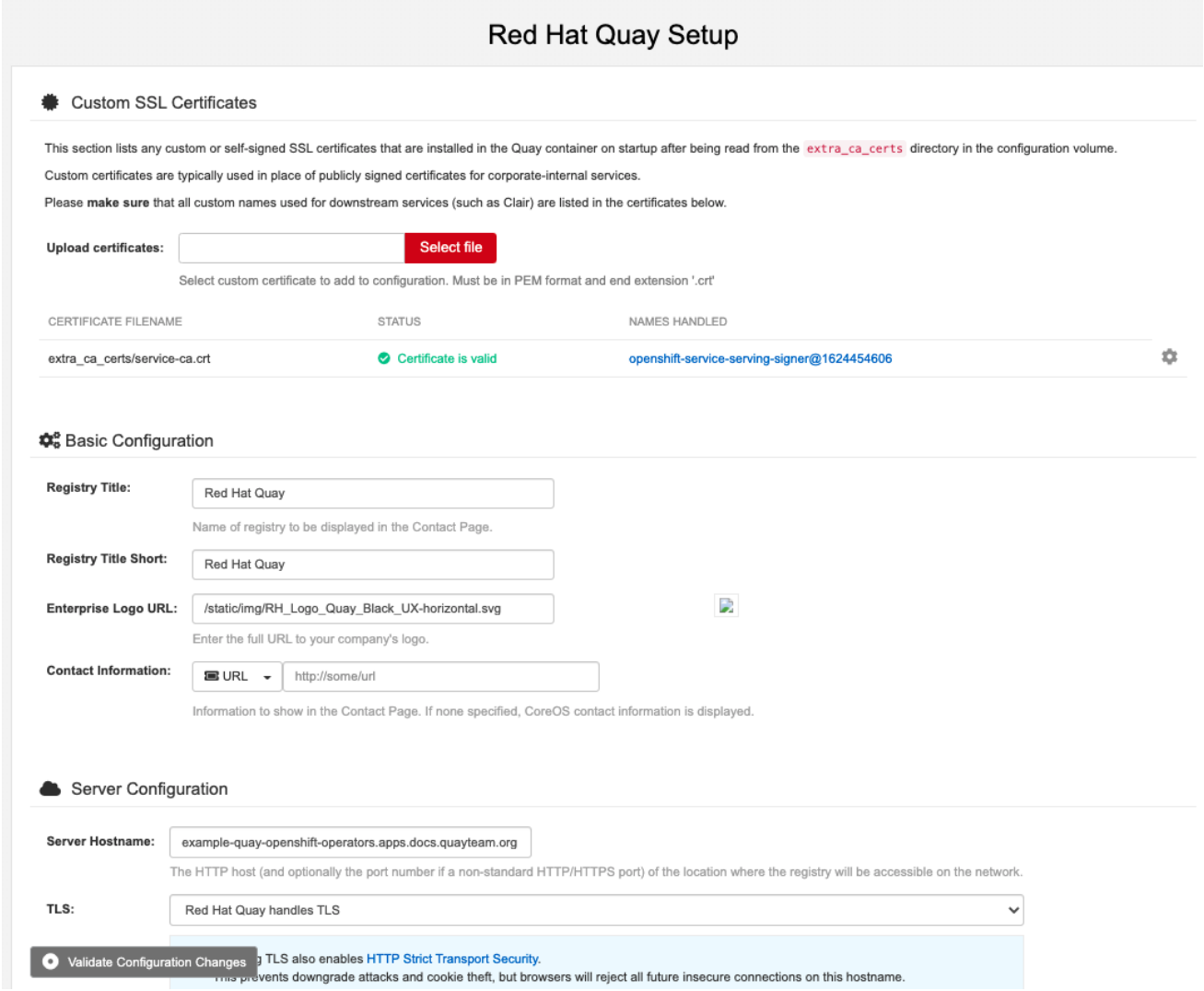


2. In the Data section of the Secret details screen, click **Reveal values** to see the credentials for logging in to the config editor:



4.1.2. Logging in to the config editor

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

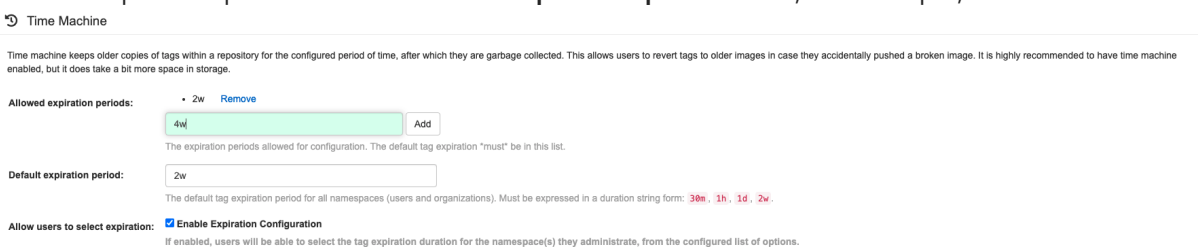


4.1.3. Changing configuration

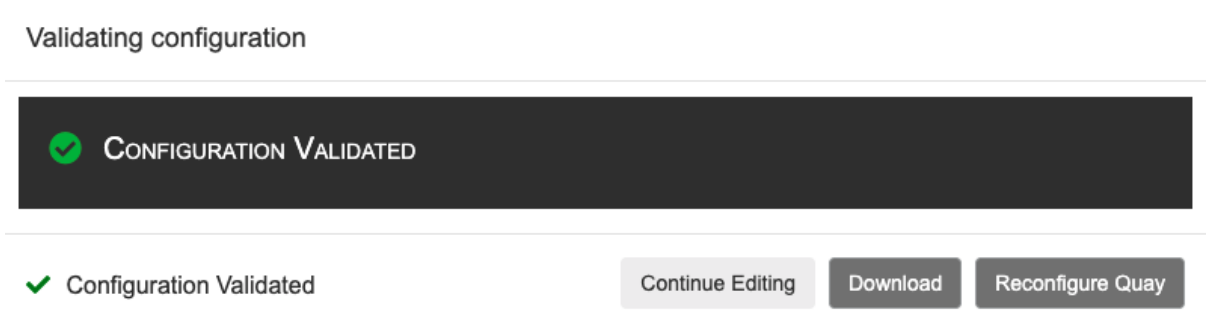
In the following example, you will update your configuration file by changing the default expiration period of deleted tags.

Procedure

1. On the config editor, locate the **Time Machine** section.
2. Add an expiration period to the **Allowed expiration periods** box, for example, **4w**:

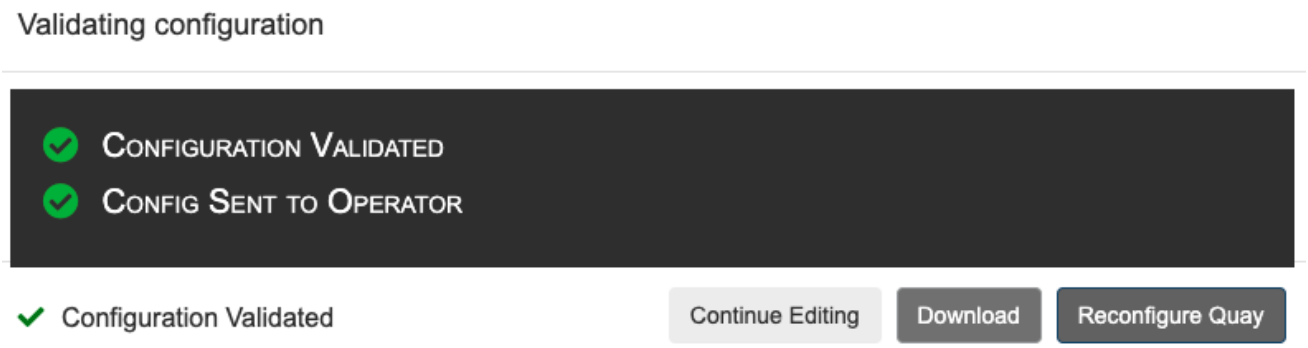


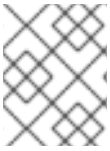
3. Select **Validate Configuration Changes** to ensure that the changes are valid.
4. Apply the changes by pressing **Reconfigure Quay**.



After applying the changes, the config tool notifies you that the changes made have been submitted to your Red Hat Quay deployment:

+





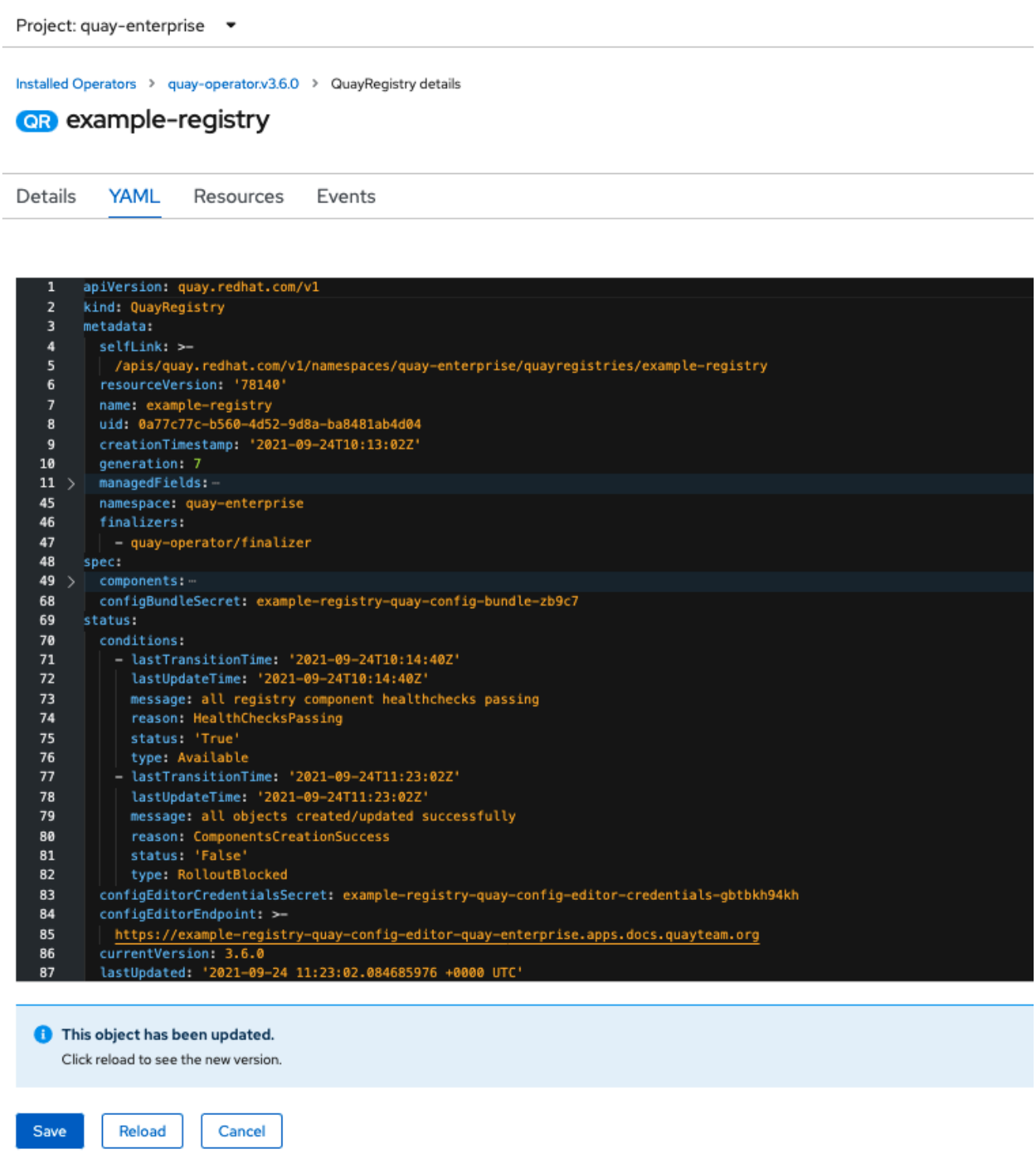
NOTE

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

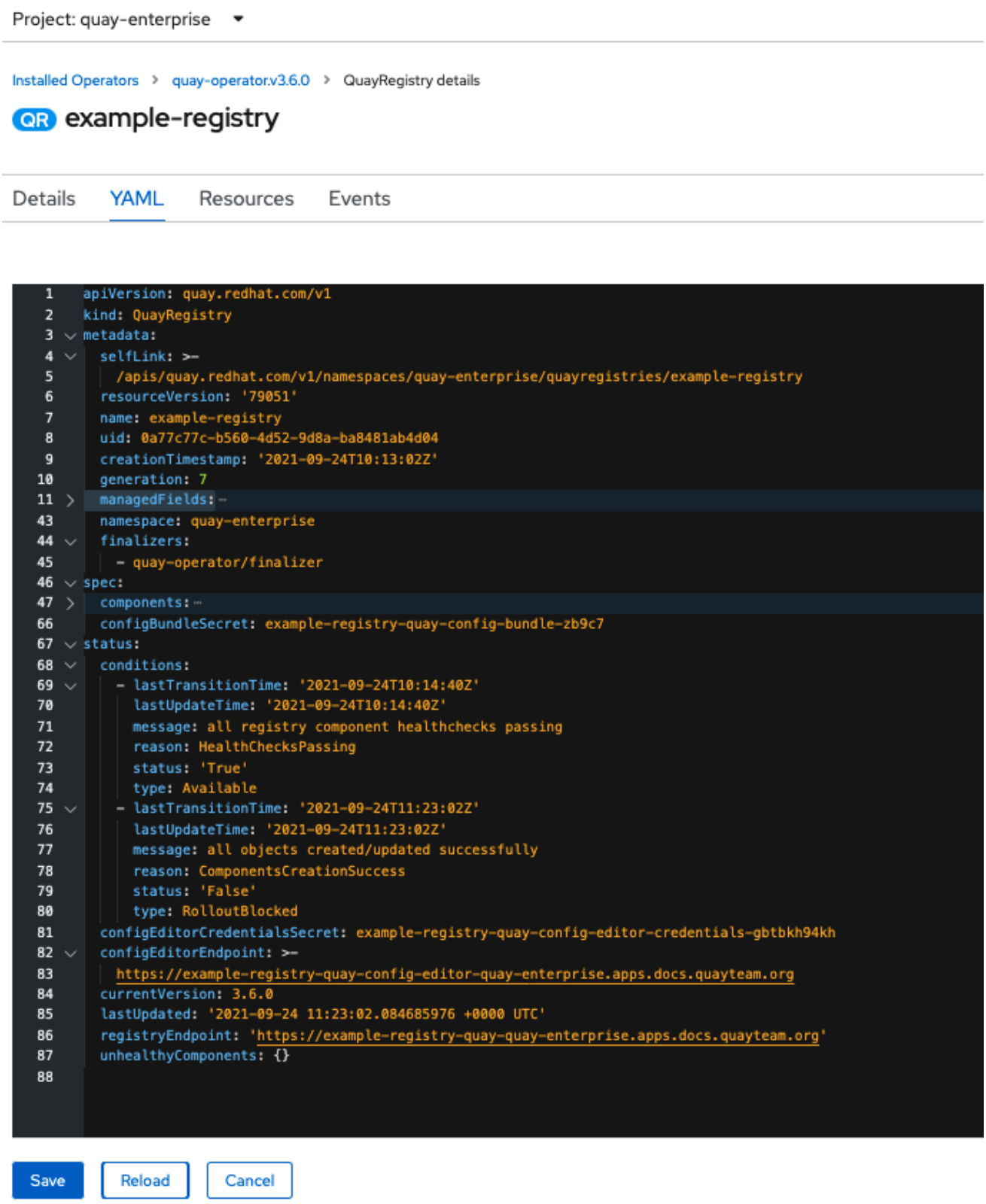
4.2. MONITORING RECONFIGURATION IN THE UI

4.2.1. QuayRegistry resource

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

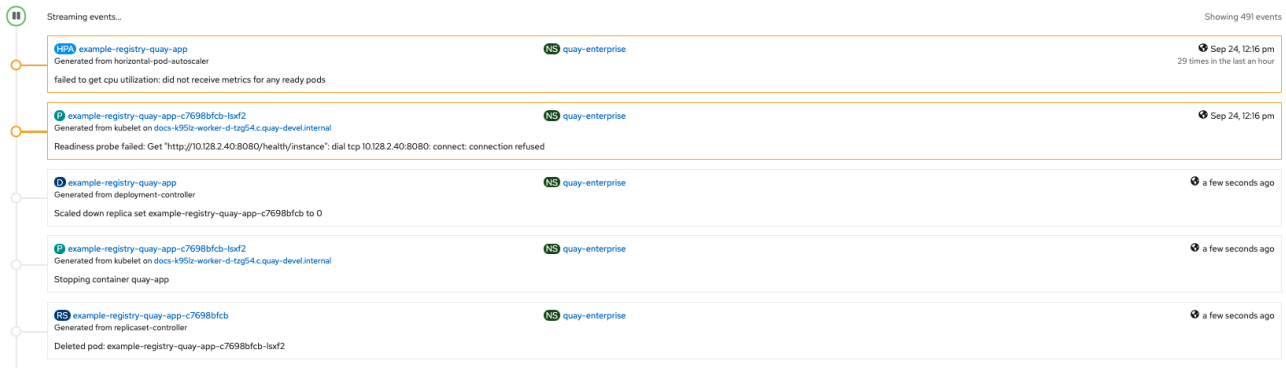


Each time the status changes, you will be prompted to reload the data to see the updated version. Eventually, the Operator will reconcile the changes, and there will be no unhealthy components reported.

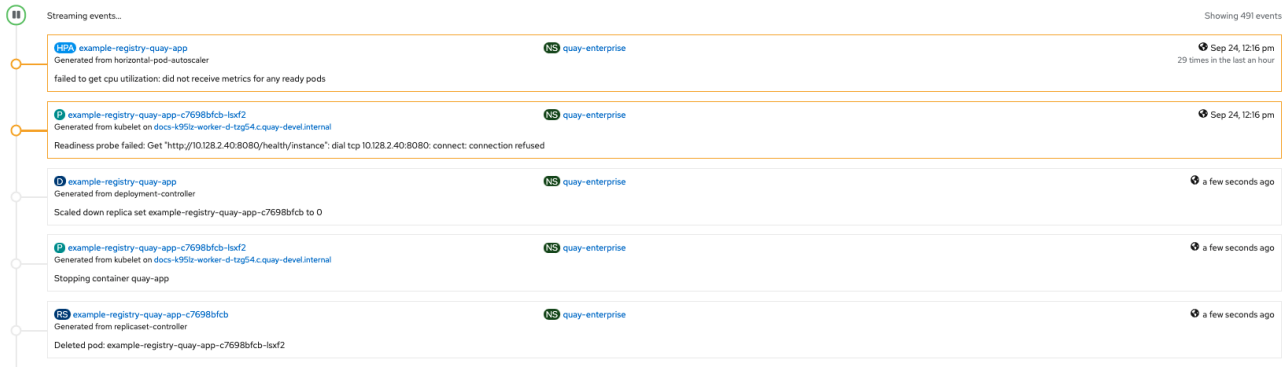


4.2.2. Events

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



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



4.3. ACCESSING UPDATED INFORMATION AFTER RECONFIGURATION

4.3.1. Accessing the updated config tool credentials in the UI

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 Red Hat Quay Operator uses defaults for the following managed components:

- **quay:** Holds overrides for the Red Hat 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:** Stores live builder logs and the Red Hat Quay tutorial. Also includes the locking mechanism that is required for garbage collection.

- **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 entrypoint to the Red Hat Quay registry from outside of OpenShift Container Platform
- **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 Container Platform handles SSL/TLS
- **clairpostgres:** Configures a managed Clair database

The Red Hat Quay Operator handles any required configuration and installation work needed for Red Hat Quay to use the managed components. If the opinionated deployment performed by the Red Hat Quay Operator is unsuitable for your environment, you can provide the Red Hat Quay 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

Requirements:

If you are using an externally managed PostgreSQL database, you must manually enable pg_trgm extension for a successful deployment.

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

   config.yaml:

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

2. Create a Secret using the configuration file:

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

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

   quayregistry.yaml

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

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

5.2.2. NooBaa unmanaged storage

1. Create a NooBaa Object Bucket Claim in the console at Storage → Object Bucket Claims.

2. Retrieve the Object Bucket Claim Data details including the Access Key, Bucket Name, Endpoint (hostname) and Secret Key.

3. Create a **config.yaml** configuration file, using the information for the Object Bucket Claim:

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

5.2.3. Horizontal Pod Autoscaler

Horizontal Pod Autoscalers (HPAs) 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 **Clair**, **Quay**, and **Mirror** pods is set to two. This facilitates the avoidance of downtime when updating or reconfiguring Red Hat Quay by the Operator or during rescheduling events.

5.2.3.1. Disabling the Horizontal Pod Autoscaler

To disable autoscaling or create your own **HorizontalPodAutoscaler**, specify the component as **unmanaged** in the **QuayRegistry** instance. For example:

```
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-----
MIIDljCCAn6gAwIBAgIBATANBgkqhkiG9w0BAQsFADA5MRcwFQYDVQQKDA5MQUIu
TElCQ09SRS5TTzEeMBwGA1UEAwwVQ2VydGlmaWNhdGUgQXV0aG9yaXR5MB4XDTE2
MDExMjA2NTkxMFoXDTM2MDExMjA2NTkxMFowOTEXMBUGA1UECgwOTEFCLkxJQkNP
UkUuU08xHjAcBgNVBAMMFUNlcnRpZmljYXRlIEF1dGhvcml0eTCCASIwDQYJKoZI
[...]
-----END CERTIFICATE-----

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

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

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

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

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

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

5.4. CONFIGURING OCI AND HELM WITH THE OPERATOR

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

quay-config-bundle.yaml

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

IMPORTANT

As of Red Hat Quay 3.8, **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
```

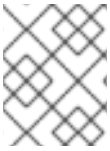
5.5. VOLUME SIZE OVERRIDES

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
    - kind: clairpostgres
      managed: true
```



NOTE

The volume size of the **clairpostgres** component cannot be overridden. This is a known issue and will be fixed in a future version of Red Hat Quay.(PROJQUAY-4301)

CHAPTER 6. CLAIR FOR RED HAT QUAY

Clair v4 (Clair) is an open source application that leverages static code analyses for parsing image content and reporting vulnerabilities affecting the content. Clair is packaged with Red Hat Quay and can be used in both standalone and Operator deployments. It can be run in highly scalable configurations, where components can be scaled separately as appropriate for enterprise environments.

6.1. CLAIR CONFIGURATION OVERVIEW

Clair is configured by a structured YAML file. Each Clair node needs to specify what mode it will run in and a path to a configuration file through CLI flags or environment variables. For example:

```
$ clair -conf ./path/to/config.yaml -mode indexer
```

or

```
$ clair -conf ./path/to/config.yaml -mode matcher
```

The aforementioned commands each start two Clair nodes using the same configuration file. One runs the indexing facilities, while other runs the matching facilities.

Environment variables respected by the Go standard library can be specified if needed, for example:

- **HTTP_PROXY**
- **HTTPS_PROXY**
- **SSL_CERT_DIR**

If you are running Clair in **combo** mode, you must supply the indexer, matcher, and notifier configuration blocks in the configuration.

6.1.1. Clair configuration reference

The following YAML shows an example Clair configuration:

```
http_listen_addr: ""
introspection_addr: ""
log_level: ""
tls: {}
indexer:
    connstring: ""
    scanlock_retry: 0
    layer_scan_concurrency: 0
    migrations: false
    scanner: {}
    airgap: false
matcher:
    connstring: ""
    indexer_addr: ""
    migrations: false
    period: ""
    disable_updaters: false
    update_retention: 2
```

```
matchers:
    names: nil
    config: nil
updaters:
    sets: nil
    config: nil
notifier:
    connstring: ""
    migrations: false
    indexer_addr: ""
    matcher_addr: ""
    poll_interval: ""
    delivery_interval: ""
    disable_summary: false
    webhook: null
    amqp: null
    stomp: null
auth:
  psk: nil
trace:
    name: ""
    probability: null
    jaeger:
        agent:
            endpoint: ""
        collector:
            endpoint: ""
            username: null
            password: null
        service_name: ""
        tags: nil
        buffer_max: 0
metrics:
    name: ""
    prometheus:
        endpoint: null
    dogstatsd:
        url: ""
```

NOTE

The above YAML file lists every key for completeness. Using this configuration file as-is will result in some options not having their defaults set normally.

6.1.2. Clair general fields

The following section describes the general configuration fields available for a Clair deployment:

Field	Typhttp_listen_ae	Description

Field	Typhttp_listen_ae	Description
http_listen_addr	String	Configures where the HTTP API is exposed. Default: **:6060**
introspection_addr	String	Configures where Clair's metrics and health endpoints are exposed.
log_level	String	Sets the logging level. Requires one of the following strings: **debug-color**, **debug**, **info**, **warn**, **error**, **fatal**, **panic**
tls	String	A map containing the configuration for serving the HTTP API of TLS/SSL and HTTP/2.
.cert	String	The TLS certificate to be used. Must be a full-chain certificate.

6.1.3. Clair indexer configuration fields

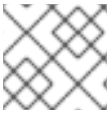
The following indexer configuration fields are available for Clair.

Field	Type	Description
indexer	Object	Provides Clair indexer node configuration.
.airgap	Boolean	Disables HTTP access to the internet for indexers and fetchers. Private IPv4 and IPv6 addresses are allowed. Database connections are unaffected.
.connstring	String	A Postgres connection string. Accepts format as a URL or libpq connection string.

Field	Type	Description
.index_report_request_concurrency	Integer	Rate limits the number of index report creation requests. Setting this to **0** attemps to auto-size this value. Setting a negative value means unlimited. The auto-sizing is a multiple of the number of available cores. The API returns a **429** status code if concurrency is exceeded.
.scanlock_retry	Integer	A positive integer representing seconds. Concurrent indexers lock on manifest scans to avoid clobbering. This value tunes how often a waiting indexer polls for the lock.
.layer_scan_concurrency	Integer	Positive integer limiting the number of concurrent layer scans. Indexers will match a manifest's layer concurrently. This value tunes the number of layers an indexer scans in parallel.
.migrations	Boolean	Whether indexer nodes handle migrations to their database.
.scanner	String	Indexer configuration. Scanner allows for passing configuration options to layer scanners. The scanner will have this configuration pass to it on construction if designed to do so.
.scanner.dist	String	A map with the name of a particular scanner and arbitrary YAML as a value.
.scanner.package	String	A map with the name of a particular scanner and arbitrary YAML as a value.
.scanner.repo	String	A map with the name of a particular scanner and arbitrary YAML as a value.

6.1.4. Clair matcher configuration fields

The following matcher configuration fields are available for Clair.



NOTE

Differs from **matchers** configuration fields.

Field	Type	Description
matcher	Object	Provides Clair matcher node configuration.
.cache_age	String	Controls how long users should be hinted to cache responses for.
.connstring	String	A Postgres connection string. Accepts format as a URL or libpq connection string.
.max_conn_pool	Integer	Limits the database connection pool size. Clair allows for a custom connection pool size. This number directly sets how many active database connections are allowed concurrently. This parameter will be ignored in a future version. Users should configure this through the connection string.
.indexer_addr	String	A matcher contacts an indexer to create a **VulnerabilityReport**. The location of this indexer is required. Defaults to **30m**.
.migrations	Boolean	Whether matcher nodes handle migrations to their databases.
.period	String	Determines how often updates for new security advisories take place. Defaults to **30m**.

Field	Type	Description
.disable_updaters	Boolean	Whether to run background updates or not.
.update_retention	Integer	Sets the number of update operations to retain between garbage collection cycles. This should be set to a safe MAX value based on database size constraints. Defaults to **10m**. If a value of less than **0** is provided, garbage collection is disabled. **2** is the minimum value to ensure updates can be compared to notifications.

6.1.5. Clair matchers configuration fields

The following matchers configuration fields are available for Clair.

NOTE

Differs from **matcher** configuration fields.

Field	Type	Description
matchers	Array of strings	Provides configuration for the in-tree **matchers** and **remotematchers**.
.names	String	A list of string values informing the matcher factory about enabled matchers. If value is set to **null**, the default list of matchers run: **alpine**, **aws**, **debian**, **oracle**, **photon**, **python**, **python**, **rhel**, **suse**, **ubuntu**, **crda**

Field	Type	Description

.config	String	Provides configuration to a specific matcher.
		
		A map keyed by the name of the matcher containing a sub-object which will be provided to the matchers factory constructor. For example:
		
		`config:`
		`  python:`
		`  ignore_vulns:`
		`  - CVE-XYZ`
		`  - CVE-ABC`

6.1.6. Clair updaters configuration fields

The following updaters configuration fields are available for Clair.

Field	Type	Description
updaters	Object	Provides configuration for the matcher's update manager.
.sets	String	A list of values informing the update manager which updaters to run. If value is set to **null**, the default set of updaters runs the following: **alpine**, **aws**, **debian**, **oracle**, **photon**, **pyupio**, **rhel**, **suse**, **ubuntu** If left blank, zero updaters run.

Field	Type	Description

.config	String	Provides configuration to specific updater sets.
		
		A map keyed by the name of the updater set containing a sub-object which will be provided to the updater set's constructor. For example:
		
		`config:`
		`  ubuntu:`
		`  security_tracker_url: http://security.url`
		`  ignore_distributions:`
		`  - cosmic`

6.1.7. Clair notifier configuration fields

The following notifier configuration fields are available for Clair.

Field	Type	Description
notifier	Object	Provides Clair notifier node configuration.
.connstring	String	Postgres connection string. Accepts format as URL, or libpq connection string.
.migrations	Boolean	Whether notifier nodes handle migrations to their database.
.indexer_addr	String	A notifier contacts an indexer to create or obtain manifests affected by vulnerabilities. The location of this indexer is required.
.matcher_addr	String	A notifier contacts a matcher to list update operations and acquire diffs. The location of this matcher is required.
.poll_interval	String	The frequency at which the notifier will query a matcher for update operations.

Field	Type	Description
.delivery_interval	String	The frequency at which the notifier attempts delivery of created, or previously failed, notifications.
.disable_summary	Boolean	Controls whether notifications should be summarized to one per manifest.
.webhook	Object	Configures the notifier for webhook delivery.
.webhook.target	String	URL where the webhook will be delivered.
.webhook.callback	String	The callback URL where notifications can be retrieved. The notification ID will be appended to this URL. This will typically be where the Clair notifier is hosted.
.webhook.headers	String	A map associating a header name to a list of values.
.amqp	Object	Configures the notifier for AMQP delivery.  **NOTE** Clair does not declare any AMQP components on its own. All attempts to use an exchange or queue are passive only and will fail. Broker administrators should setup exchanges and queues ahead of time.
.amqp.direct	Boolean	If **true**, the notifier will deliver individual notifications (not a callback) to the configured AMQP broker.

Field	Type	Description
.amqp.rollup	Integer	When **amqp.direct** is set to **true**, this value informs the notifier of how many notifications to send in a direct delivery. For example, if **direct** is set to **true**, and **amqp.rollup** is set to **5**, the notifier delivers no more than 5 notifications in a single JSON payload to the broker. Setting the value to **0** effectively sets it to **1**.
.amqp.exchange	Object	The AMQP exchange to connect to.
.amqp.exchange.name	String	The name of the exchange to connect to.
.amqp.exchange.type	String	The type of the exchange. Typically one of the following: **direct**, **fanout**, **topic**, **headers**.
.amqp.exchange.durability	Boolean	Whether the configured queue is durable.
.amqp.exchange.auto_delete	Boolean	Whether the configured queue uses an **auto_delete_policy**.
.amqp.routing_key	String	The name of the routing key each notification is sent with.
.amqp.callback	String	If **amqp.direct** is set to **false**, this URL is provided in the notification callback sent to the broker. This URL should point to Clair's notification API endpoint.
.amqp.uris	String	A list of one or more AMQP brokers to connect to, in priority order.
.amqp.tls	Object	Configures TLS/SSL connection to an AMQP broker.

Field	Type	Description
.amqp.tls.root_ca	String	The filesystem path where a root CA can be read.
.amqp.tls.cert	String	The filesystem path where a TLS/SSL certificate can be read.  **NOTE** Clair also allows **SSL_CERT_DIR**, as documented for the Go **crypto/x509** package.
.amqp.tls.key	String	The filesystem path where a TLS/SSL private key can be read.
.stomp	Object	Configures the notifier for STOMP delivery.
.stomp.direct	Boolean	If **true**, the notifier delivers individual notifications (not a callback) to the configured STOMP broker.
.stomp.rollup	Integer	If **stomp.direct** is set to **true**, this value limits the number of notifications sent in a single direct delivery. For example, if **direct** is set to **true**, and **rollup** is set to **5**, the notifier delivers no more than 5 notifications in a single JSON payload to the broker. Setting the value to **0** effectively sets it to **1**.
.stomp.callback	String	If **stomp.callback** is set to **false**, the provided URL in the notification callback is sent to the broker. This URL should point to Clair's notification API endpoint.

Field	Type	Description
.stomp.destination	String	The STOMP destination to deliver notifications to.
.stomp.uris	String	A list of one or more STOMP brokers to connect to in priority order.
.stomp.tls	Object	Configured TLS/SSL connection to STOMP broker.
.stomp.tls.root_ca	String	The filesystem path where a root CA can be read.  **NOTE** Clair also respects **SSL_CERT_DIR**, as documented for the Go **crypto/x509** package.
.stomp.tls.cert	String	The filesystem path where a TLS/SSL certificate can be read.
.stomp.tls.key	String	The filesystem path where a TLS/SSL private key can be read.
.stomp.user	String	Configures login details for the STOMP broker.
.stomp.user.login	String	The STOMP login to connect with.
.stomp.user.passcode	String	The STOMP passcode to connect with.

6.1.8. Clair authorization configuration fields

The following authorization configuration fields are available for Clair.

Field	Type	Description

Field	Type	Description
auth	Object	Defines Clair's external and intra-service JWT based authentication. If multiple **auth** mechanisms are defined, Clair picks one. Currently, multiple mechanisms are unsupported.
.psk	String	Defines pre-shared key authentication.
.psk.key	String	A shared base64 encoded key distributed between all parties signing and verifying JWTs.
.psk.iss	String	A list of JWT issuers to verify. An empty list accepts any issuer in a JWT claim.

6.1.9. Clair trace configuration fields

The following trace configuration fields are available for Clair.

Field	Type	Description
trace	Object	Defines distributed tracing configuration based on OpenTelemetry.
.name	String	The name of the application traces will belong to.
.probability	Integer	The probability a trace will occur.
.jaeger	Object	Defines values for Jaeger tracing.
.jaeger.agent	Object	Defines values for configuring delivery to a Jaeger agent.
.jaeger.agent.endpoint	String	An address in the **<host>:<post>** syntax where traces can be submitted.
.jaeger.collector	Object	Defines values for configuring delivery to a Jaeger collector.

Field	Type	Description
.jaeger.collector.endpoint	String	An address in the **<host>:<post>** syntax where traces can be submitted.
.jaeger.collector.username	String	A Jaeger username.
.jaeger.collector.password	String	A Jaeger password.
.jaeger.service_name	String	The service name registered in Jaeger.
.jaeger.tags	String	Key-value pairs to provide additional metadata.
.jaeger.buffer_max	Integer	The maximum number of spans that can be buffered in memory before they are sent to the Jaeger backend for storage and analysis.

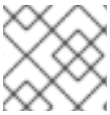
6.1.10. Clair metrics configuration fields

The following metrics configuration fields are available for Clair.

Field	Type	Description
metrics	Object	Defines distributed tracing configuration based on OpenTelemetry.
.name	String	The name of the metrics in use.
.prometheus	String	Configuration for a Prometheus metrics exporter.
.prometheus.endpoint	String	Defines the path where metrics are served.

CHAPTER 7. SCANNING POD IMAGES WITH THE CONTAINER SECURITY OPERATOR

The Container Security Operator (CSO) is an addon for the Clair security scanner available on OpenShift Container Platform and other Kubernetes platforms. With the CSO, users can scan container images associated with active pods for known vulnerabilities.



NOTE

The CSO does not work without Red Hat Quay and Clair.

The Container Security Operator (CSO) performs the following features:

- Watches containers associated with pods on either specified or all namespaces.
- Queries the container registry where the containers came from for vulnerability information (provided that an image's registry supports image scanning, such a a Red Hat Quay registry with Clair scanning).
- Exposes vulnerabilities via the **ImageManifestVuln** object in the Kubernetes API.

NOTE

To see instructions on installing the CSO on Kubernetes, select the Install button from the Container Security OperatorHub.io page.

7.1. DOWNLOADING AND RUNNING THE CONTAINER SECURITY OPERATOR IN OPENSHIFT CONTAINER PLATFORM

Use the following procedure to download the Container Security Operator (CSO).

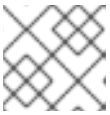
NOTE

In the following procedure, the CSO is installed in the **marketplace-operators** namespace. This allows the CSO to be used in all namespaces of your OpenShift Container Platform cluster.

Procedure

1. On the OpenShift Container Platform console page, select **Operators → OperatorHub** and search for **Container Security Operator**.
2. Select the Container Security Operator, then select **Install** to go to the **Create Operator Subscription** page.
3. Check the settings (all namespaces and automatic approval strategy, by default), and select **Subscribe**. The **Container Security** appears after a few moments on the **Installed Operators** screen.
4. Optional: you can add custom certificates to the CSO. In this example, create a certificate named **quay.crt** in the current directory. Then, run the following command to add the certificate to the CSO:

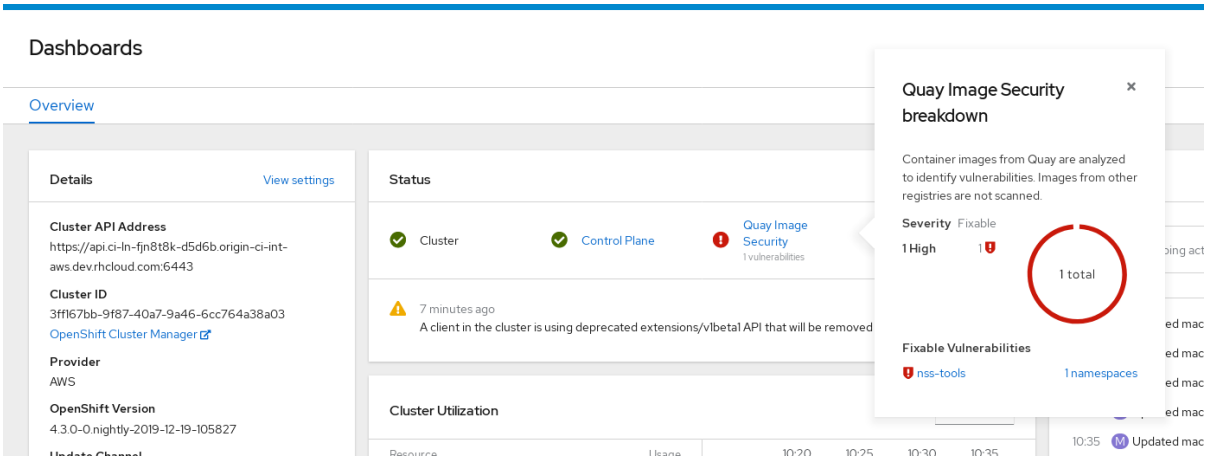
```
$ oc create secret generic container-security-operator-extra-certs --from-file=quay.crt -n
openshift-operators
```

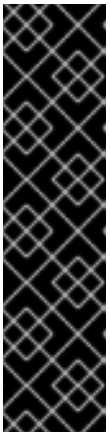


NOTE

You must restart the Operator pod for the new certificates to take effect.

5. Navigate to **Home → Dashboards**. A link to **Image Security** appears under the status section, with a listing of the number of vulnerabilities found so far. Select the link to see a security breakdown, as shown in the following image:



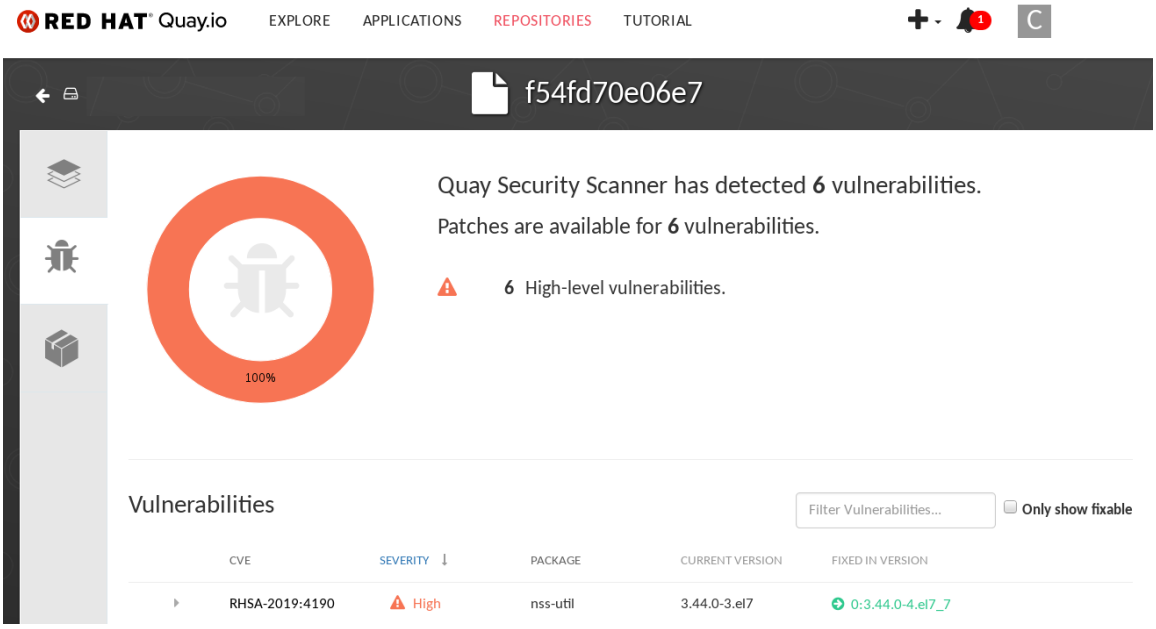


IMPORTANT

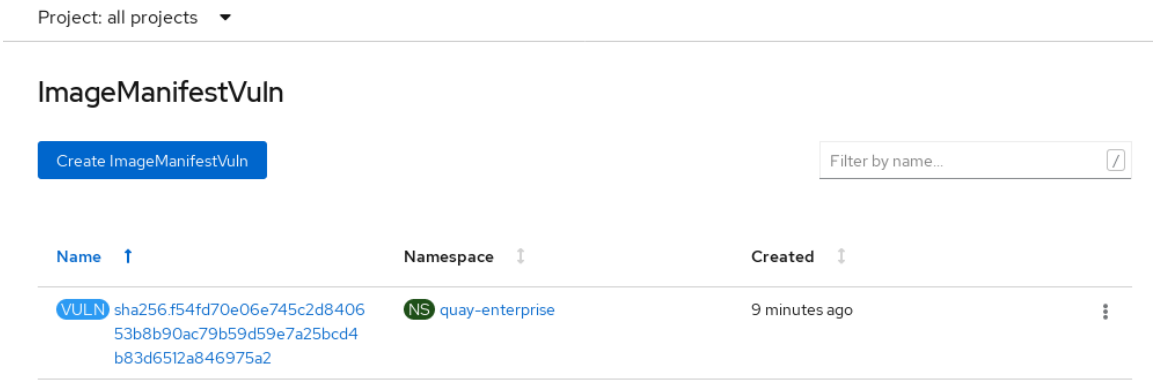
The Container Security Operator currently provides broken links for Red Hat Security advisories. For example, the following link might be provided: **https://access.redhat.com/errata/RHSA-2023:1842%20https://access.redhat.com/security/cve/CVE-2023-23916**. The **%20** in the URL represents a space character, however it currently results in the combination of the two URLs into one incomplete URL, for example, **https://access.redhat.com/errata/RHSA-2023:1842** and **https://access.redhat.com/security/cve/CVE-2023-23916**. As a temporary workaround, you can copy each URL into your browser to navigate to the proper page. This is a known issue and will be fixed in a future version of Red Hat Quay.

6. You can do one of two things at this point to follow up on any detected vulnerabilities:

    a. Select the link to the vulnerability. You are taken to the container registry, Red Hat Quay or other registry where the container came from, where you can see information about the vulnerability. The following figure shows an example of detected vulnerabilities from a Quay.io registry:

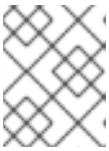


b. Select the namespaces link to go to the **ImageManifestVuln** screen, where you can see the name of the selected image and all namespaces where that image is running. The following figure indicates that a particular vulnerable image is running in two namespaces:



After executing this procedure, you are made aware of what images are vulnerable, what you must do to fix those vulnerabilities, and every namespace that the image was run in. Knowing this, you can perform the following actions:

- Alert users who are running the image that they need to correct the vulnerability.
- Stop the images from running by deleting the deployment or the object that started the pod that the image is in.



NOTE

If you delete the pod, it might take a few minutes for the vulnerability to reset on the dashboard.

7.2. QUERY IMAGE VULNERABILITIES FROM THE CLI

You can query information on security from the command line. To query for detected vulnerabilities, type:

```
$ oc get vuln --all-namespaces
```

```
NAMESPACE     NAME              AGE
default       sha256.ca90...    6m56s
skynet        sha256.ca90...    9m37s
```

To display details for a particular vulnerability, identify one of the vulnerabilities, along with its namespace and the **describe** option. This example shows an active container whose image includes an RPM package with a vulnerability:

```
$ oc describe vuln --namespace mynamespace sha256.ac50e3752...
Name:         sha256.ac50e3752...
Namespace:    quay-enterprise
...
Spec:
  Features:
    Name:            nss-util
    Namespace Name:  centos:7
    Version:         3.44.0-3.el7
    Versionformat:   rpm
    Vulnerabilities:
      Description: Network Security Services (NSS) is a set of libraries...
```