Red Hat Quay 3

Deploy Red Hat Quay on OpenShift with the Quay Operator

Deploy Red Hat Quay on OpenShift with Quay Operator

Last Updated: 2023-01-24

Red Hat Quay 3 Deploy Red Hat Quay on OpenShift with the Quay Operator

Deploy Red Hat Quay on OpenShift with Quay Operator

Abstract

Deploy Red Hat Quay on an OpenShift Cluster with the Red Hat Quay Operator

Table of Contents

PREFACE

Red Hat Quay is an enterprise-quality container registry. Use Red Hat Quay to build and store container images, then make them available to deploy across your enterprise.

The Red Hat Quay Operator provides a simple method to deploy and manage Red Hat Quay on an OpenShift cluster.

As of Red Hat Quay 3.4.0, the Operator has been completely re-written to provide an improved out of the box experience as well as support for more Day 2 operations. As a result the new Operator is simpler to use and is more opinionated. The key differences from earlier versions of the Operator are:

- The **QuayEcosystem** custom resource has been replaced with the **QuayRegistry** custom resource
- The default installation options produces a fully supported Quay environment with all managed dependencies (database, caches, object storage, etc) supported for production use (some components may not be highly available)
- A new robust validation library for Quay’s configuration which is shared by the Quay application and config tool for consistency
- Object storage can now be managed by the Operator using the **ObjectBucketClaim** Kubernetes API (Red Hat OpenShift Data Foundation can be used to provide a supported implementation of this API on OpenShift)
- Customization of the container images used by deployed pods for testing and development scenarios

CHAPTER 1. INTRODUCTION TO THE RED HAT QUAY OPERATOR

This document outlines the steps for configuring, deploying, managing and upgrading Red Hat Quay on OpenShift using the Red Hat Quay Operator.

It shows you how to:

- Install the Red Hat Quay Operator
- Configure object storage, either managed or unmanaged
- Configure other unmanaged components, if required, including database, Redis, routes, TLS, etc.
- Deploy the Red Hat Quay registry on OpenShift using the Operator
- Use advanced features supported by the Operator
- Upgrade the registry by upgrading the Operator

1.1. QUAYREGISTRY API

The Quay Operator provides the **QuayRegistry** custom resource API to declaratively manage **Quay** container registries on the cluster. Use either the OpenShift UI or a command-line tool to interact with this API.

- Creating a **QuayRegistry** will result in the Operator deploying and configuring all necessary resources needed to run Quay on the cluster.
- Editing a **QuayRegistry** will result in the Operator reconciling the changes and creating/updating/deleting objects to match the desired configuration.
- Deleting a **QuayRegistry** will result in garbage collection of all previously created resources and the **Quay** container registry will no longer be available.

The **QuayRegistry** API is fairly simple, and the fields are outlined in the following sections.

1.2. 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
```

1.3. USING MANAGED COMPONENTS

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

- **quay:** Holds overrides for the Quay deployment, for example, environment variables and number of replicas. This component is new in Red Hat Quay 3.7 and cannot be set to unmanaged.
- **postgres:** For storing the registry metadata, uses a version of Postgres 10 from the Software Collections
- **clair:** Provides image vulnerability scanning
- **redis:** Handles Quay builder coordination and some internal logging
- **horizontalpodautoscaler:** Adjusts the number of Quay pods depending on memory/cpu consumption
- **objectstorage:** For storing image layer blobs, utilizes the **ObjectBucketClaim** Kubernetes API which is provided by Noobaa/RHOCS
- **route:** Provides an external entrypoint to the Quay registry from outside OpenShift
- **mirror:** Configures repository mirror workers (to support optional repository mirroring)
- **monitoring:** Features include a Grafana dashboard, access to individual metrics, and alerting to notify for frequently restarting Quay pods
- **tls:** Configures whether Red Hat Quay or OpenShift handles TLS
- **clairpostgres:** Configures a managed Clair database

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

1.4. 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.

1.5. CONFIG BUNDLE SECRET

The **spec.configBundleSecret** field is a reference to the **metadata.name** of a **Secret** in the same namespace as the **QuayRegistry**. This **Secret** must contain a **config.yaml** key/value pair. This **config.yaml** file is a Quay config YAML file. This field is optional, and will be auto-filled by the Operator if not provided. If provided, it serves as the base set of config fields which are later merged with other fields from any managed components to form a final output **Secret**, which is then mounted into the Quay application pods.

1.6. PREREQUISITES FOR RED HAT QUAY ON OPENSHIFT

Before you begin the deployment of Red Hat Quay Operator on OpenShift, you should consider the following.

1.6.1. OpenShift cluster

You need a privileged account to an OpenShift 4.5 or later cluster on which to deploy the Red Hat Quay Operator. That account must have the ability to create namespaces at the cluster scope.

1.6.2. Resource Requirements

Each Red Hat Quay application pod has the following resource requirements:

- 8Gi of memory
- 2000 millicores of CPU.

The Red Hat Quay Operator will create at least one application pod per Red Hat Quay deployment it manages. Ensure your OpenShift cluster has sufficient compute resources for these requirements.

1.6.3. Object Storage

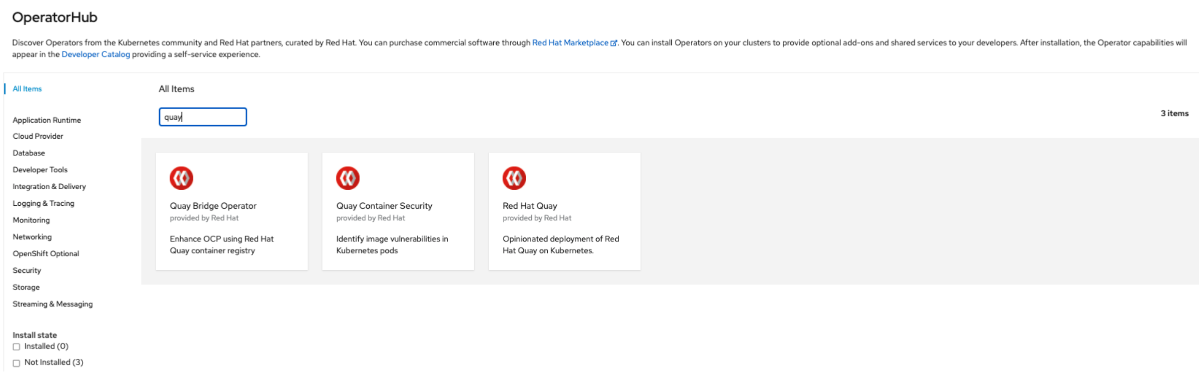
By default, the Red Hat Quay Operator uses the **ObjectBucketClaim** Kubernetes API to provision object storage. Consuming this API decouples the Operator from any vendor-specific implementation. Red Hat OpenShift Data Foundation provides this API via its NooBaa component, which will be used in this example.

Red Hat Quay can be manually configured to use any of the following supported cloud storage options:

- Amazon S3 (see S3 IAM Bucket Policy for details on configuring an S3 bucket policy for Red Hat Quay)
- Azure Blob Storage
- Google Cloud Storage
- Ceph Object Gateway (RADOS)
- OpenStack Swift
- CloudFront + S3

CHAPTER 2. INSTALLING THE QUAY OPERATOR FROM OPERATORHUB

1. Using the OpenShift console, Select Operators → OperatorHub, then select the Red Hat Quay Operator. If there is more than one, be sure to use the Red Hat certified Operator and not the community version.



OperatorHub

Discover Operators from the Kubernetes community and Red Hat partners, curated by Red Hat. You can purchase commercial software through Red Hat Marketplace. You can install Operators on your clusters to provide optional add-ons and shared services to your developers. After installation, the Operator capabilities will appear in the Developer Catalog providing a self-service experience.

All Items

Application Runtime
Cloud Provider
Database
Developer Tools
Integration & Delivery
Logging & Tracing
Monitoring
Networking
OpenShift Optional
Security
Storage
Streaming & Messaging

Install state
☐ Installed (0)
☐ Not Installed (3)

All Items

quay

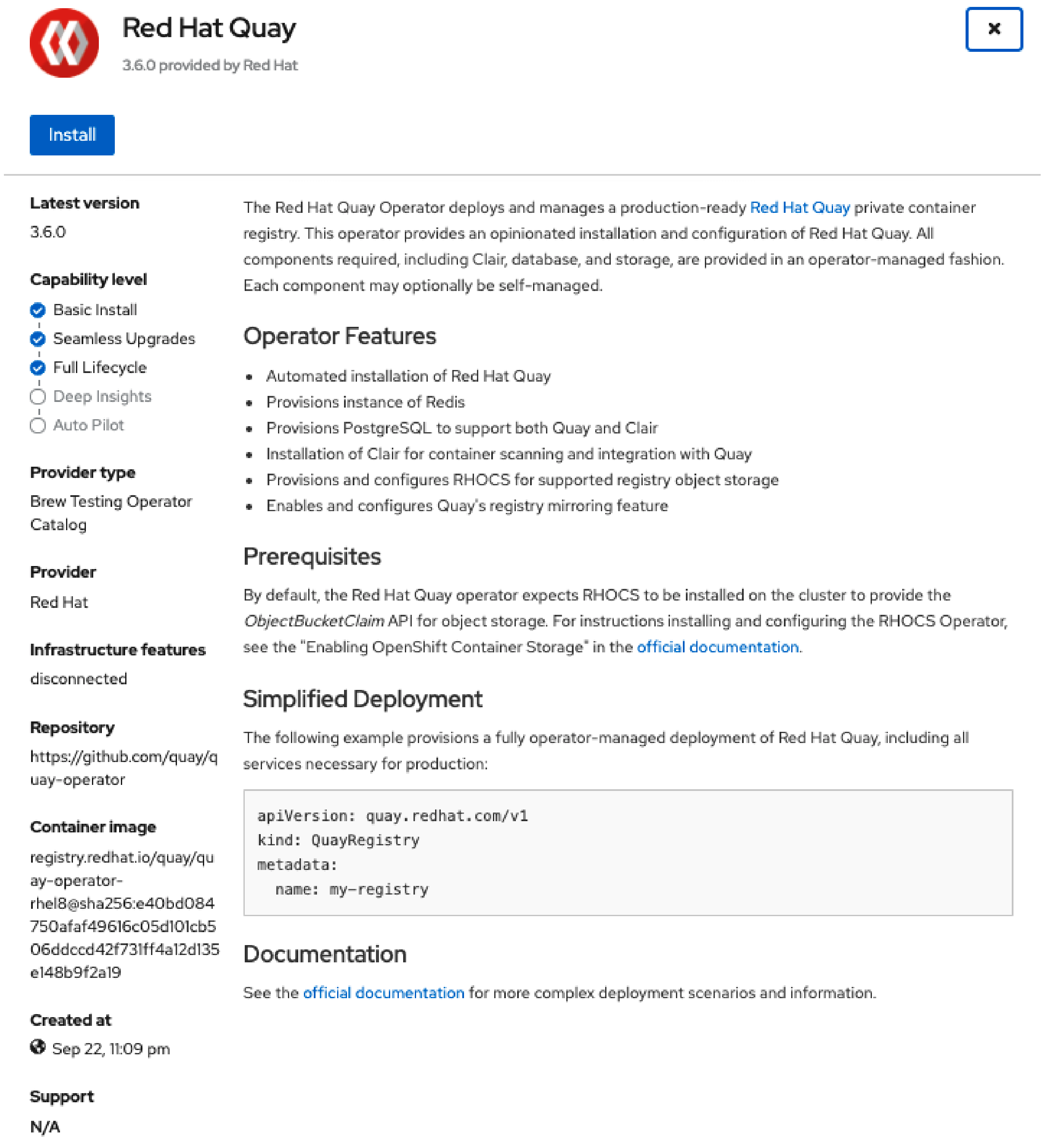
3 items

Quay Bridge Operator
provided by Red Hat
Enhance OCP using Red Hat Quay container registry

Quay Container Security
provided by Red Hat
Identify image vulnerabilities in Kubernetes pods

Red Hat Quay
provided by Red Hat
Opinionated deployment of Red Hat Quay on Kubernetes.

2. The Installation page outlines the features and prerequisites:


Red Hat Quay

3.6.0 provided by Red Hat

Install

Latest version
3.6.0

Capability level
- ✓ Basic Install
- ✓ Seamless Upgrades
- ✓ Full Lifecycle
- ○ Deep Insights
- ○ Auto Pilot

Provider type
Brew Testing Operator Catalog

Provider
Red Hat

Infrastructure features
disconnected

Repository
https://github.com/quay/quay-operator

Container image
registry.redhat.io/quay/quay-operator-rhel8@sha256:e40bd084750afaf49616c05d101cb506ddccd42f731ff4a12d135e148b9f2a19

Created at
Sep 22, 11:09 pm

Support
N/A

The Red Hat Quay Operator deploys and manages a production-ready Red Hat Quay private container registry. This operator provides an opinionated installation and configuration of Red Hat Quay. All components required, including Clair, database, and storage, are provided in an operator-managed fashion. Each component may optionally be self-managed.

Operator Features

- Automated installation of Red Hat Quay
- Provisions instance of Redis
- Provisions PostgreSQL to support both Quay and Clair
- Installation of Clair for container scanning and integration with Quay
- Provisions and configures RHOCS for supported registry object storage
- Enables and configures Quay's registry mirroring feature

Prerequisites

By default, the Red Hat Quay operator expects RHOCS to be installed on the cluster to provide the *ObjectBucketClaim* API for object storage. For instructions installing and configuring the RHOCS Operator, see the "Enabling OpenShift Container Storage" in the official documentation.

Simplified Deployment

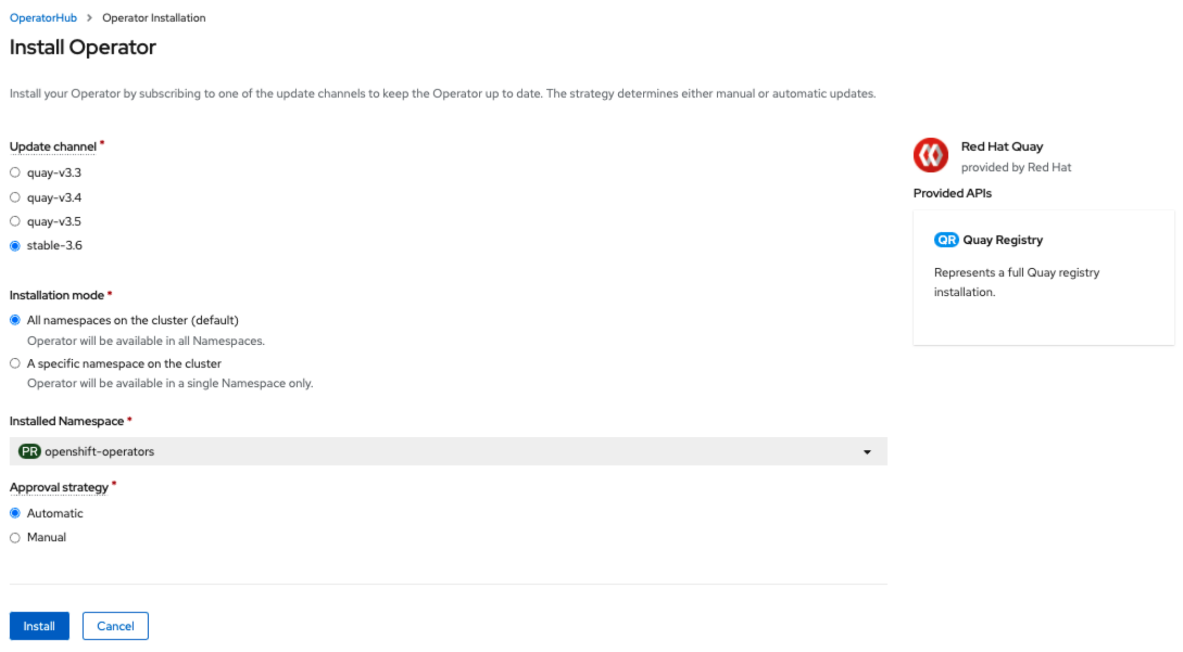
The following example provisions a fully operator-managed deployment of Red Hat Quay, including all services necessary for production:

```
apiVersion: quay.redhat.com/v1
kind: QuayRegistry
metadata:
  name: my-registry
```

Documentation

See the official documentation for more complex deployment scenarios and information.

3. Select Install. The Operator Installation page appears.


OperatorHub > Operator Installation

Install Operator

Install your Operator by subscribing to one of the update channels to keep the Operator up to date. The strategy determines either manual or automatic updates.

Update channel *
- ○ quay-v3.3
- ○ quay-v3.4
- ○ quay-v3.5
- ● stable-3.6

Installation mode *
- ● All namespaces on the cluster (default)
  Operator will be available in all Namespaces.
- ○ A specific namespace on the cluster
  Operator will be available in a single Namespace only.

Installed Namespace *
PR openshift-operators

Approval strategy *
- ● Automatic
- ○ Manual

Red Hat Quay
provided by Red Hat

Provided APIs

QR Quay Registry
Represents a full Quay registry installation.

4. The following choices are available for customizing the installation:
   - **Update Channel:** Choose the update channel, for example, **stable-3.7** for the latest release.
   - **Installation Mode:** Choose **All namespaces on the cluster** if you want the Operator to be available cluster-wide. Choose **A specific namespace on the cluster** if you want it deployed only within a single namespace. It is recommended that you install the Operator cluster-wide. If you choose a single namespace, the monitoring component will not be available by default.
   - **Approval Strategy:** Choose to approve either automatic or manual updates. Automatic update strategy is recommended.
5. Select Install.
6. After a short time, you will see the Operator installed successfully in the Installed Operators page.

CHAPTER 3. CONFIGURING QUAY BEFORE DEPLOYMENT

The Operator can manage all the Red Hat Quay components when deploying on OpenShift, and this is the default configuration. Alternatively, you can manage one or more components externally yourself, where you want more control over the set up, and then allow the Operator to manage the remaining components.

The standard pattern for configuring unmanaged components is:

1. Create a **config.yaml** configuration file with the appropriate settings
2. Create a Secret using the configuration file

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

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

   quayregistry.yaml

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

4. Deploy the registry using the YAML file:

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

3.1. 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.

3.1.1. 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.

3.1.2. 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.

3.1.3. Adding a super user

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

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

3.1.4. 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
...
```

3.1.5. 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:
    -
...
```

3.1.6. 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
...
```

3.1.7. 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
...
```

3.2. CONFIGURING OBJECT STORAGE

You need to configure object storage before installing Red Hat Quay, irrespective of whether you are allowing the Operator to manage the storage or managing it yourself.

If you want the Operator to be responsible for managing storage, see the section on Managed storage for information on installing and configuring the NooBaa / RHOCS Operator.

If you are using a separate storage solution, set **objectstorage** as **unmanaged** when configuring the Operator. See the following section. Unmanaged storage, for details of configuring existing storage.

3.2.1. Unmanaged storage

Some configuration examples for unmanaged storage are provided in this section for convenience. See the Red Hat Quay configuration guide for full details for setting up object storage.

3.2.1.1. 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
```

3.2.1.2. 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
```

3.2.1.3. 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**.

3.2.1.4. Ceph / RadosGW Storage / 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
```

3.2.1.5. 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
```

3.2.1.6. NooBaa unmanaged storage

Use the following procedure to deploy NooBaa as your unmanaged storage configuration.

Procedure

1. Create a NooBaa Object Bucket Claim in the {product-title} console by navigating to **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
```

For more information about configuring an Object Bucket Claim, see Object Bucket Claim .

3.2.2. Managed storage

If you want the Operator to manage object storage for Quay, your cluster needs to be capable of providing object storage via the **ObjectBucketClaim** API. Using the Red Hat OpenShift Data Foundation (ODF) Operator, there are two supported options available:

- A standalone instance of the Multi-Cloud Object Gateway backed by a local Kubernetes **PersistentVolume** storage
  - Not highly available
  - Included in the Quay subscription
  - Does not require a separate subscription for ODF

- A production deployment of ODF with scale-out Object Service and Ceph
  - Highly available
  - Requires a separate subscription for ODF

To use the standalone instance option, continue reading below. For production deployment of ODF, please refer to the official documentation.

NOTE

Object storage disk space is allocated automatically by the Operator with 50 GiB. This number represents a usable amount of storage for most small to medium Red Hat Quay installations but may not be sufficient for your use cases. Resizing the RHOCS volume is currently not handled by the Operator. See the section below on resizing managed storage for more details.

3.2.2.1. About The Standalone Object Gateway

As part of a Red Hat Quay subscription, users are entitled to use the *Multi-Cloud Object Gateway* (MCG) component of the Red Hat OpenShift Data Foundation Operator (formerly known as OpenShift Container Storage Operator). This gateway component allows you to provide an S3-compatible object storage interface to Quay backed by Kubernetes **PersistentVolume**-based block storage. The usage is limited to a Quay deployment managed by the Operator and to the exact specifications of the MCG instance as documented below.

Since Red Hat Quay does not support local filesystem storage, users can leverage the gateway in combination with Kubernetes **PersistentVolume** storage instead, to provide a supported deployment. A **PersistentVolume** is directly mounted on the gateway instance as a backing store for object storage and any block-based **StorageClass** is supported.

By the nature of **PersistentVolume**, this is not a scale-out, highly available solution and does not replace a scale-out storage system like Red Hat OpenShift Data Foundation (ODF). Only a single instance of the gateway is running. If the pod running the gateway becomes unavailable due to rescheduling, updates or unplanned downtime, this will cause temporary degradation of the connected Quay instances.

3.2.2.1.1. Create A Standalone Object Gateway

To install the ODF (formerly known as OpenShift Container Storage) Operator and configure a single instance Multi-Cloud Gateway service, follow these steps:

1. Open the OpenShift console and select Operators → OperatorHub, then select the OpenShift Data Foundation Operator.
2. Select Install. Accept all default options and select Install again.
3. Within a minute, the Operator will install and create a namespace **openshift-storage**. You can confirm it has completed when the **Status** column is marked **Succeeded**.

   > When the installation of the ODF Operator is complete, you are prompted to create a storage system. Do not follow this instruction. Instead, create NooBaa object storage as outlined the following steps.

4. Create NooBaa object storage. Save the following YAML to a file called **noobaa.yaml**.

   ▪

```
apiVersion: noobaa.io/v1alpha1
kind: NooBaa
metadata:
  name: noobaa
  namespace: openshift-storage
spec:
 dbResources:
   requests:
     cpu: '0.1'
     memory: 1Gi
 dbType: postgres
 coreResources:
   requests:
     cpu: '0.1'
     memory: 1Gi
```

This will create a single instance deployment of the *Multi-cloud Object Gateway*.

5. Apply the configuration with the following command:

```
$ oc create -n openshift-storage -f noobaa.yaml
noobaa.noobaa.io/noobaa created
```

6. After a couple of minutes, you should see that the MCG instance has finished provisioning (**PHASE** column will be set to **Ready**):

```
$ oc get -n openshift-storage noobaas noobaa -w
NAME     MGMT-ENDPOINTS              S3-ENDPOINTS                IMAGE
PHASE   AGE
noobaa   [https://10.0.32.3:30318]   [https://10.0.32.3:31958]   registry.redhat.io/ocs4/mcg-
core-
rhel8@sha256:56624aa7dd4ca178c1887343c7445a9425a841600b1309f6deace37ce6b8678d
Ready   3d18h
```

7. Next, configure a backing store for the gateway. Save the following YAML to a file called **noobaa-pv-backing-store.yaml**.

noobaa-pv-backing-store.yaml

```
apiVersion: noobaa.io/v1alpha1
kind: BackingStore
metadata:
  finalizers:
  - noobaa.io/finalizer
  labels:
    app: noobaa
  name: noobaa-pv-backing-store
  namespace: openshift-storage
spec:
  pvPool:
    numVolumes: 1
    resources:
      requests:
```

```
      storage: 50Gi (1)
    storageClass: STORAGE-CLASS-NAME (2)
  type: pv-pool
```

(1) The overall capacity of the object storage service, adjust as needed

(2) The **StorageClass** to use for the **PersistentVolumes** requested, delete this property to use the cluster default

8. Apply the configuration with the following command:

```
$ oc create -f noobaa-pv-backing-store.yaml
backingstore.noobaa.io/noobaa-pv-backing-store created
```

   This creates the backing store configuration for the gateway. All images in Quay will be stored as objects through the gateway in a **PersistentVolume** created by the above configuration.

9. Finally, run the following command to make the **PersistentVolume** backing store the default for all **ObjectBucketClaims** issued by the Operator.

```
$ oc patch bucketclass noobaa-default-bucket-class --patch '{"spec":{"placementPolicy":
{"tiers":[{"backingStores":["noobaa-pv-backing-store"]}]}}}' --type merge -n openshift-storage
```

This concludes the setup of the *Multi-Cloud Object Gateway* instance for Red Hat Quay. Note that this configuration cannot be run in parallel on a cluster with Red Hat OpenShift Data Foundation installed.

3.3. CONFIGURING THE DATABASE

3.3.1. 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.

3.3.2. Database configuration

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

3.3.2.1. 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 3.1. 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**

3.3.2.2. 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 3.2. Database connection arguments

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

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

3.3.2.2.1. 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 3.3. 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.
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.

3.3.2.2.2. 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.

3.3.3. Using the managed PostgreSQL

Recommendations:

- Database backups should be performed regularly using either the supplied tools on the Postgres image or your own backup infrastructure. The Operator does not currently ensure the Postgres database is backed up.
- Restoring the Postgres database from a backup must be done using Postgres tools and procedures. Be aware that your Quay **Pods** should not be running while the database restore is in progress.
- Database disk space is allocated automatically by the Operator with 50 GiB. This number represents a usable amount of storage for most small to medium Red Hat Quay installations but may not be sufficient for your use cases. Resizing the database volume is currently not handled by the Operator.

3.4. CONFIGURING TLS AND ROUTES

Support for OpenShift Container Platform Edge-Termination Routes has been added by way of a new managed component, **tls**. This separates the **route** component from TLS and allows users to configure both separately. **EXTERNAL_TLS_TERMINATION: true** is the opinionated setting. Managed **tls** means that the default cluster wildcard cert is used. Unmanaged **tls** means that the user provided cert/key pair will be injected into the **Route**.

ssl.cert and **ssl.key** are now moved to a separate, persistent Secret, which ensures that the cert/key pair is not re-generated upon every reconcile. These are now formatted as **edge** routes and mounted to the same directory in the Quay container.

Multiple permutations are possible when configuring TLS and Routes, but the following rules apply:

- If TLS is **managed**, then route must also be **managed**
- If TLS is **unmanaged** then you must supply certs, either with the config tool or directly in the config bundle

The following table describes the valid options:

Table 3.4. Valid configuration options for TLS and routes

Option	Route	TLS	Certs provided	Result
My own load balancer handles TLS	Managed	Managed	No	Edge Route with default wildcard cert
Red Hat Quay handles TLS	Managed	Unmanaged	Yes	Passthrough route with certs mounted inside the pod
Red Hat Quay handles TLS	Unmanaged	Unmanaged	Yes	Certificates are set inside the quay pod but route must be created manually

NOTE

Red Hat Quay 3.7 does not support builders when TLS is managed by the Operator.

3.4.1. Creating the config bundle secret with TLS cert, key pair:

To add your own TLS cert and key, include them in the config bundle secret as follows:

```
$ oc create secret generic --from-file config.yaml=./config.yaml --from-file ssl.cert=./ssl.cert --from-file
ssl.key=./ssl.key config-bundle-secret
```

3.5. CONFIGURING OTHER COMPONENTS

3.5.1. Using external Redis

If you wish to use an external Redis database, set the component as unmanaged in the **QuayRegistry** instance:

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

```
BUILDLOGS_REDIS:
    host: quay-server.example.com
    port: 6379
    ssl: false

USER_EVENTS_REDIS:
    host: quay-server.example.com
    port: 6379
    ssl: false
```

2. Create a Secret using the configuration file

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

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

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

4. Deploy the registry

3.5.1.1. Redis configuration fields

This section details the configuration fields available for Redis deployments.

3.5.1.1.1. Build logs

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

Table 3.5. Build logs configuration

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

3.5.1.1.2. User events

The following user event fields are available for Redis deployments:

Table 3.6. User events config

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

3.5.1.1.3. Example Redis configuration

The following YAML shows a sample configuration using Redis:

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

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

NOTE

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

3.5.2. Disabling the Horizontal Pod Autoscaler

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

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

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

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

3.5.3. Disabling Route Component

To prevent the Operator from creating a **Route**:

1. Mark the component as unmanaged in the **QuayRegistry**:

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

2. Specify that you want Quay to handle TLS in the configuration, by editing the **config.yaml** file:

   config.yaml

   ```
   ...
   EXTERNAL_TLS_TERMINATION: false
   ...
   SERVER_HOSTNAME: example-registry-quay-quay-enterprise.apps.user1.example.com
   ...
   PREFERRED_URL_SCHEME: https
   ...
   ```

   If you do not configure the unmanaged Route correctly, you will see an error similar to the following:

   ```
   {
     {
       "kind":"QuayRegistry",
   ```

```
    "namespace":"quay-enterprise",
    "name":"example-registry",
    "uid":"d5879ba5-cc92-406c-ba62-8b19cf56d4aa",
    "apiVersion":"quay.redhat.com/v1",
    "resourceVersion":"2418527"
  },
  "reason":"ConfigInvalid",
  "message":"required component `route` marked as unmanaged, but `configBundleSecret` is
missing necessary fields"
}
```

NOTE

Disabling the default **Route** means you are now responsible for creating a **Route**, **Service**, or **Ingress** in order to access the Quay instance and that whatever DNS you use must match the **SERVER_HOSTNAME** in the Quay config.

3.5.4. Unmanaged monitoring

If you install the Quay Operator in a single namespace, the monitoring component is automatically set to 'unmanaged'. To enable monitoring in this scenario, see the section Section 8.2, “Enabling monitoring when Operator is installed in a single namespace”.

To disable monitoring explicitly:

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

3.5.5. Unmanaged mirroring

To disable mirroring explicitly:

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

CHAPTER 4. DEPLOYING QUAY USING THE QUAY OPERATOR

The Operator can be deployed from the command line or from the OpenShift console, but the fundamental steps are the same.

4.1. DEPLOYING RED HAT QUAY FROM THE COMMAND LINE

1. Create a namespace, for example, **quay-enterprise**.
2. Create a secret for the config bundle, if you want to pre-configure any aspects of the deployment
3. Create a **QuayRegistry** custom resource in a file called **quayregistry.yaml**
    a. For a minimal deployment, using all the defaults:

    quayregistry.yaml:

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

    b. If you want to have some components unmanaged, add this information in the **spec** field. For example, a minimal deployment might look like:

    quayregistry.yaml:

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

    c. If you have created a config bundle, for example, **init-config-bundle-secret**, reference it in the **quayregistry.yaml** file:

    quayregistry.yaml:

    ```
    apiVersion: quay.redhat.com/v1
    kind: QuayRegistry
    metadata:
    ```

```
  name: example-registry
  namespace: quay-enterprise
spec:
  configBundleSecret: init-config-bundle-secret
```

d. If you have a proxy configured, you can add the information using overrides for Quay, Clair, and mirroring:

quayregistry.yaml:

```
  kind: QuayRegistry
  metadata:
    name: quay37
  spec:
    configBundleSecret: config-bundle-secret
    components:
      - kind: objectstorage
        managed: false
      - kind: route
        managed: true
      - kind: mirror
        managed: true
        overrides:
          env:
            - name: DEBUGLOG
              value: "true"
            - name: HTTP_PROXY
              value: quayproxy.qe.devcluster.openshift.com:3128
            - name: HTTPS_PROXY
              value: quayproxy.qe.devcluster.openshift.com:3128
            - name: NO_PROXY
              value:
svc.cluster.local,localhost,quay370.apps.quayperf370.perfscale.devcluster.openshift.com
      - kind: tls
        managed: false
      - kind: clair
        managed: true
        overrides:
          env:
            - name: HTTP_PROXY
              value: quayproxy.qe.devcluster.openshift.com:3128
            - name: HTTPS_PROXY
              value: quayproxy.qe.devcluster.openshift.com:3128
            - name: NO_PROXY
              value:
svc.cluster.local,localhost,quay370.apps.quayperf370.perfscale.devcluster.openshift.com
      - kind: quay
        managed: true
        overrides:
          env:
            - name: DEBUGLOG
              value: "true"
            - name: NO_PROXY
              value:
svc.cluster.local,localhost,quay370.apps.quayperf370.perfscale.devcluster.openshift.com
            - name: HTTP_PROXY
          value: quayproxy.qe.devcluster.openshift.com:3128
        - name: HTTPS_PROXY
          value: quayproxy.qe.devcluster.openshift.com:3128
```

4. Create the **QuayRegistry** in specified namespace:

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

5. See the section Monitoring and debugging the deployment process for information on how to track the progress of the deployment.

6. Wait until the **status.registryEndpoint** is populated.

   ```
   $ oc get quayregistry -n quay-enterprise example-registry -o jsonpath="
   {.status.registryEndpoint}" -w
   ```

4.1.1. Viewing created components using the command line

Use the **oc get pods** command to view the deployed components:

```
$ oc get pods -n quay-enterprise

NAME                                                   READY   STATUS      RESTARTS   AGE
example-registry-clair-app-5ffc9f77d6-jwr9s            1/1     Running     0          3m42s
example-registry-clair-app-5ffc9f77d6-wgp7d            1/1     Running     0          3m41s
example-registry-clair-postgres-54956d6d9c-rgs8l       1/1     Running     0          3m5s
example-registry-quay-app-79c6b86c7b-8qnr2             1/1     Running     4          3m42s
example-registry-quay-app-79c6b86c7b-xk85f             1/1     Running     4          3m41s
example-registry-quay-app-upgrade-5kl5r                0/1     Completed   4          3m50s
example-registry-quay-config-editor-597b47c995-svqrl   1/1     Running     0          3m42s
example-registry-quay-database-b466fc4d7-tfrnx         1/1     Running     2          3m42s
example-registry-quay-mirror-6d9bd78756-6lj6p          1/1     Running     0          2m58s
example-registry-quay-mirror-6d9bd78756-bv6gq          1/1     Running     0          2m58s
example-registry-quay-postgres-init-dzbmx              0/1     Completed   0          3m43s
example-registry-quay-redis-8bd67b647-skgqx            1/1     Running     0          3m42s
```

4.1.2. Horizontal Pod Autoscaling (HPA)

A default deployment shows the following running pods:

- Two pods for the Quay application itself (**example-registry-quay-app-***`)
- One Redis pod for Quay logging (**example-registry-quay-redis-***)
- One database pod for PostgreSQL used by Quay for metadata storage (**example-registry-quay-database-***)
- One pod for the Quay config editor (**example-registry-quay-config-editor-***)
- Two Quay mirroring pods (**example-registry-quay-mirror-***)
- Two pods for the Clair application (**example-registry-clair-app-***)
- One PostgreSQL pod for Clair (**example-registry-clair-postgres-***)

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

```
$ oc get hpa -n quay-enterprise
NAME                         REFERENCE                                TARGETS          MINPODS   MAXPODS
REPLICAS   AGE
example-registry-clair-app     Deployment/example-registry-clair-app     16%/90%, 0%/90%   2
10        2          13d
example-registry-quay-app      Deployment/example-registry-quay-app      31%/90%, 1%/90%   2
20        2          13d
example-registry-quay-mirror   Deployment/example-registry-quay-mirror   27%/90%, 0%/90%   2
20        2          13d
```

4.1.3. 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

4.1.3.1. Using the API to create the first user

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

NOTE

This procedure requests an OAuth token by specifying **"access_token": true**.

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

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

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

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

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

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

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

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

4.1.4. Monitoring and debugging the deployment process

Users can now troubleshoot problems during the deployment phase. The status in the **QuayRegistry** object can help you monitor the health of the components during the deployment an help you debug any problems that may arise:

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

Immediately after deployment, the QuayRegistry object will show the basic configuration:

```
apiVersion: v1
items:
- apiVersion: quay.redhat.com/v1
  kind: QuayRegistry
  metadata:
    creationTimestamp: "2021-09-14T10:51:22Z"
    generation: 3
    name: example-registry
    namespace: quay-enterprise
    resourceVersion: "50147"
    selfLink: /apis/quay.redhat.com/v1/namespaces/quay-enterprise/quayregistries/example-registry
    uid: e3fc82ba-e716-4646-bb0f-63c26d05e00e
  spec:
    components:
    - 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
    configBundleSecret: example-registry-config-bundle-kt55s
kind: List
metadata:
  resourceVersion: ""
  selfLink: ""
```

Use the **oc get pods** command to view the current state of the deployed components:

```
$ oc get pods -n quay-enterprise

NAME                                               READY   STATUS              RESTARTS   AGE
example-registry-clair-app-86554c6b49-ds7bl           0/1     ContainerCreating   0          2s
example-registry-clair-app-86554c6b49-hxp5s           0/1     Running             1          17s
example-registry-clair-postgres-68d8857899-lbc5n      0/1     ContainerCreating   0          17s
example-registry-quay-app-upgrade-h2v7h               0/1     ContainerCreating   0          9s
example-registry-quay-config-editor-5f646cbcb7-lbnc2  0/1     ContainerCreating   0          17s
example-registry-quay-database-66f495c9bc-wqsjf       0/1     ContainerCreating   0          17s
example-registry-quay-mirror-854c88457b-d845g         0/1     Init:0/1            0          2s
example-registry-quay-mirror-854c88457b-fghxv         0/1     Init:0/1            0          17s
example-registry-quay-postgres-init-bktdt             0/1     Terminating         0          17s
example-registry-quay-redis-f9b9d44bf-4htpz           0/1     ContainerCreating   0          17s
```

While the deployment is in progress, the QuayRegistry object will show the current status. In this instance, database migrations are taking place, and other components are waiting until this completes.

```
  status:
    conditions:
    - lastTransitionTime: "2021-09-14T10:52:04Z"
      lastUpdateTime: "2021-09-14T10:52:04Z"
      message: all objects created/updated successfully
      reason: ComponentsCreationSuccess
      status: "False"
      type: RolloutBlocked
    - lastTransitionTime: "2021-09-14T10:52:05Z"
      lastUpdateTime: "2021-09-14T10:52:05Z"
      message: running database migrations
      reason: MigrationsInProgress
      status: "False"
      type: Available
    configEditorCredentialsSecret: example-registry-quay-config-editor-credentials-btbkcg8dc9
    configEditorEndpoint: https://example-registry-quay-config-editor-quay-
enterprise.apps.docs.quayteam.org
    lastUpdated: 2021-09-14 10:52:05.371425635 +0000 UTC
    unhealthyComponents:
      clair:
      - lastTransitionTime: "2021-09-14T10:51:32Z"
        lastUpdateTime: "2021-09-14T10:51:32Z"
        message: 'Deployment example-registry-clair-postgres: Deployment does not have minimum
availability.'
        reason: MinimumReplicasUnavailable
        status: "False"
        type: Available
      - lastTransitionTime: "2021-09-14T10:51:32Z"
        lastUpdateTime: "2021-09-14T10:51:32Z"
        message: 'Deployment example-registry-clair-app: Deployment does not have minimum
availability.'
        reason: MinimumReplicasUnavailable
        status: "False"
        type: Available
      mirror:
      - lastTransitionTime: "2021-09-14T10:51:32Z"
        lastUpdateTime: "2021-09-14T10:51:32Z"
        message: 'Deployment example-registry-quay-mirror: Deployment does not have minimum
```

```
availability.'
        reason: MinimumReplicasUnavailable
        status: "False"
        type: Available
```

When the deployment process finishes successfully, the status in the QuayRegistry object shows no unhealthy components:

```
  status:
    conditions:
    - lastTransitionTime: "2021-09-14T10:52:36Z"
      lastUpdateTime: "2021-09-14T10:52:36Z"
      message: all registry component healthchecks passing
      reason: HealthChecksPassing
      status: "True"
      type: Available
    - lastTransitionTime: "2021-09-14T10:52:46Z"
      lastUpdateTime: "2021-09-14T10:52:46Z"
      message: all objects created/updated successfully
      reason: ComponentsCreationSuccess
      status: "False"
      type: RolloutBlocked
    configEditorCredentialsSecret: example-registry-quay-config-editor-credentials-hg7gg7h57m
    configEditorEndpoint: https://example-registry-quay-config-editor-quay-
enterprise.apps.docs.quayteam.org
    currentVersion: {producty}
    lastUpdated: 2021-09-14 10:52:46.104181633 +0000 UTC
    registryEndpoint: https://example-registry-quay-quay-enterprise.apps.docs.quayteam.org
    unhealthyComponents: {}
```

4.2. DEPLOYING RED HAT QUAY FROM THE OPENSHIFT CONSOLE

1. Create a namespace, for example, **quay-enterprise**.

2. Select Operators → Installed Operators, then select the Quay Operator to navigate to the Operator detail view.

3. Click 'Create Instance' on the 'Quay Registry' tile under 'Provided APIs'.

4. Optionally change the 'Name' of the **QuayRegistry**. This will affect the hostname of the registry. All other fields have been populated with defaults.

5. Click 'Create' to submit the **QuayRegistry** to be deployed by the Quay Operator.

6. You should be redirected to the **QuayRegistry** list view. Click on the **QuayRegistry** you just created to see the details view.

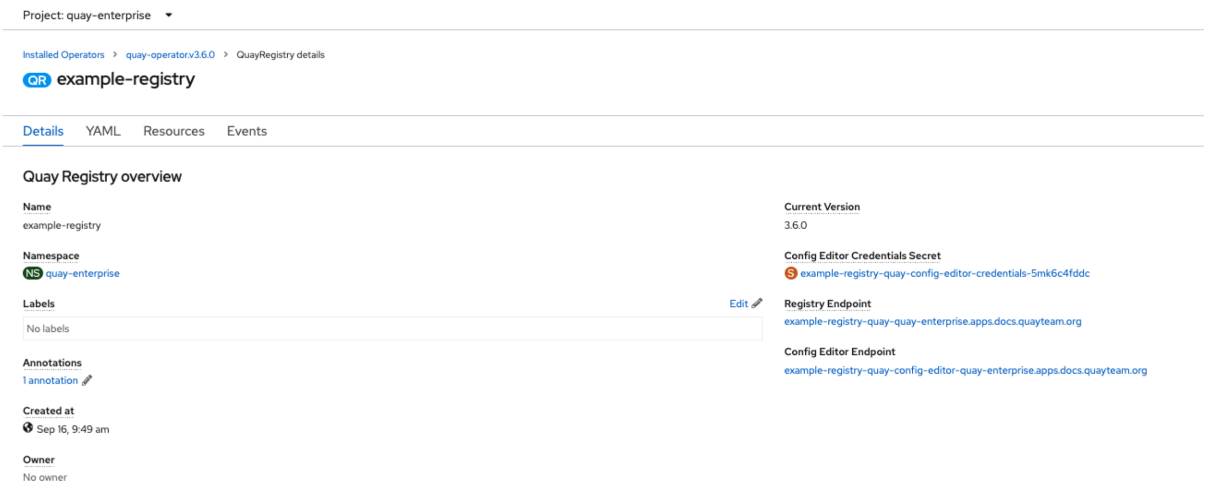
7. Once the 'Registry Endpoint' has a value, click it to access your new Quay registry via the UI. You can now select 'Create Account' to create a user and sign in.

4.2.1. Using the Quay UI to create the first user

NOTE

This procedure assumes that the **FEATURE_USER_CREATION** config option has not been set to **false.** If it is **false**, then the **Create Account** functionality on the UI will be disabled, and you will have to use the API to create the first user.

1. In the OpenShift console, navigate to Operators → Installed Operators, with the appropriate namespace / project.
2. Click on the newly installed QuayRegistry, to view the details:



Project: quay-enterprise

Installed Operators > quay-operator.v3.6.0 > QuayRegistry details

QR example-registry

Details YAML Resources Events

Quay Registry overview

Name
example-registry

Namespace
NS quay-enterprise

Labels Edit
No labels

Annotations
1 annotation

Created at
Sep 16, 9:49 am

Owner
No owner

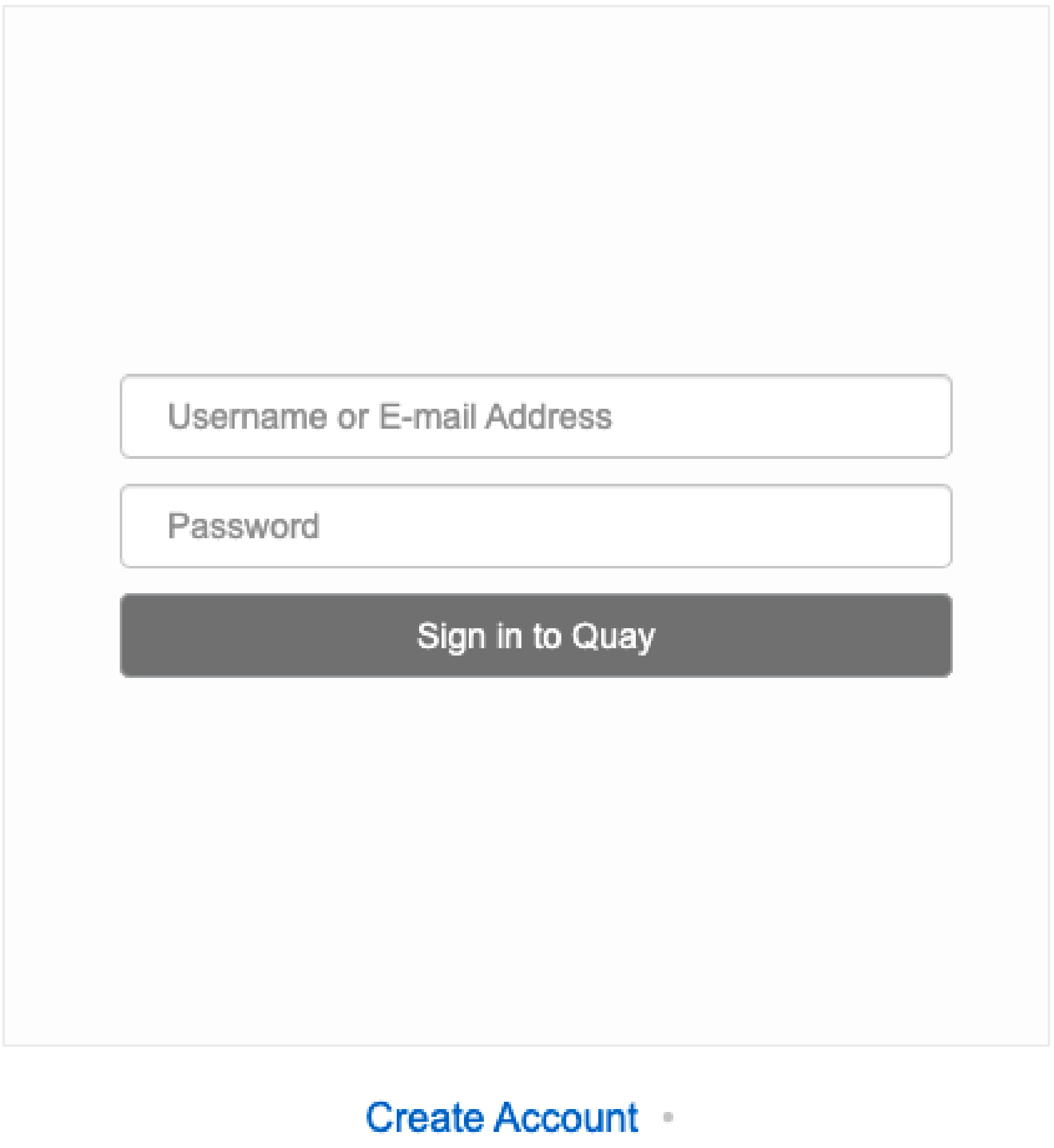
Current Version
3.6.0

Config Editor Credentials Secret
S example-registry-quay-config-editor-credentials-5mk6c4fddc

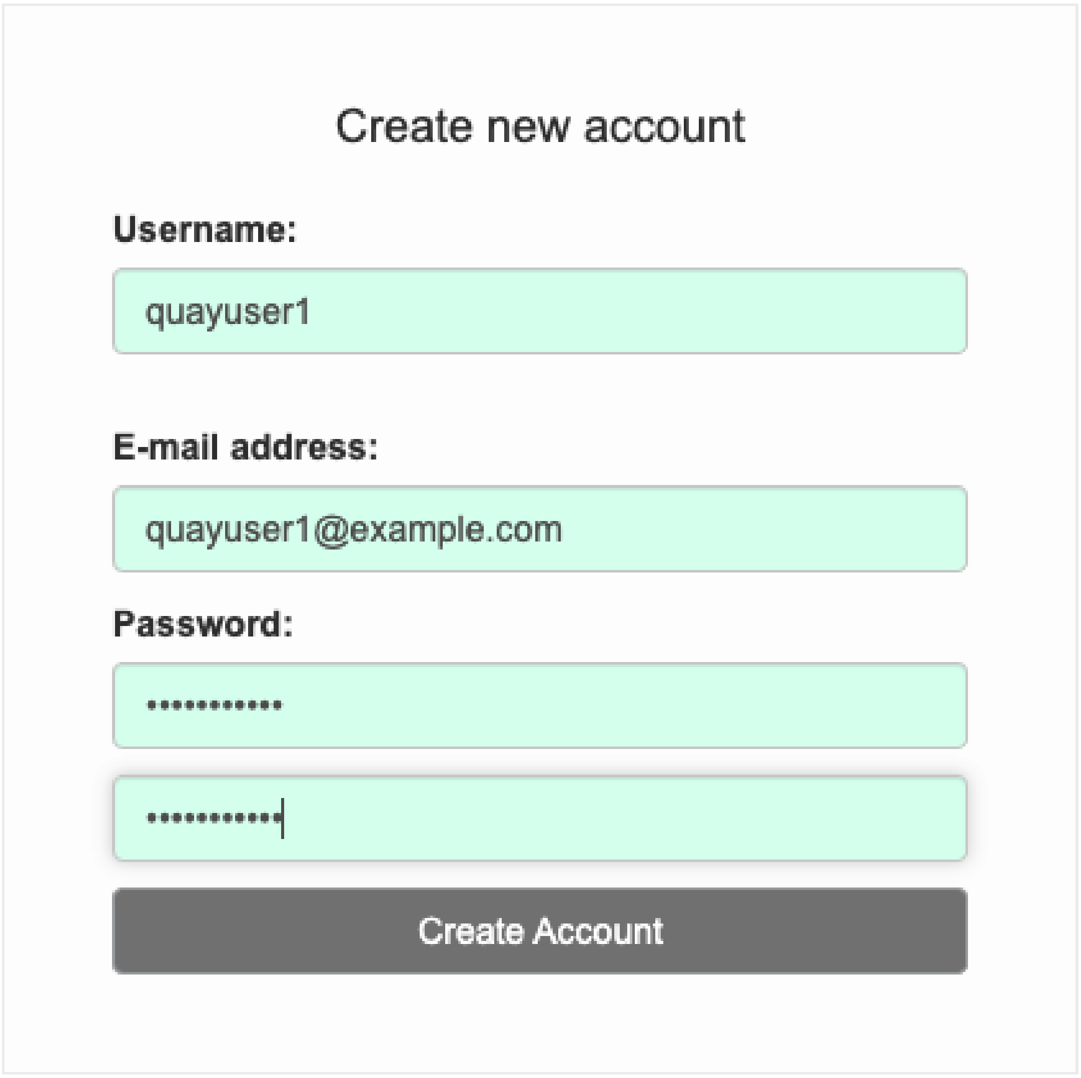
Registry Endpoint
example-registry-quay-quay-enterprise.apps.docs.quayteam.org

Config Editor Endpoint
example-registry-quay-config-editor-quay-enterprise.apps.docs.quayteam.org

3. Once the **Registry Endpoint** has a value, navigate to this URL in your browser
4. Select 'Create Account' in the Quay registry UI to create a user



5. Enter details for username, password, email and click **Create Account**



Create new account

Username:

quayuser1

E-mail address:

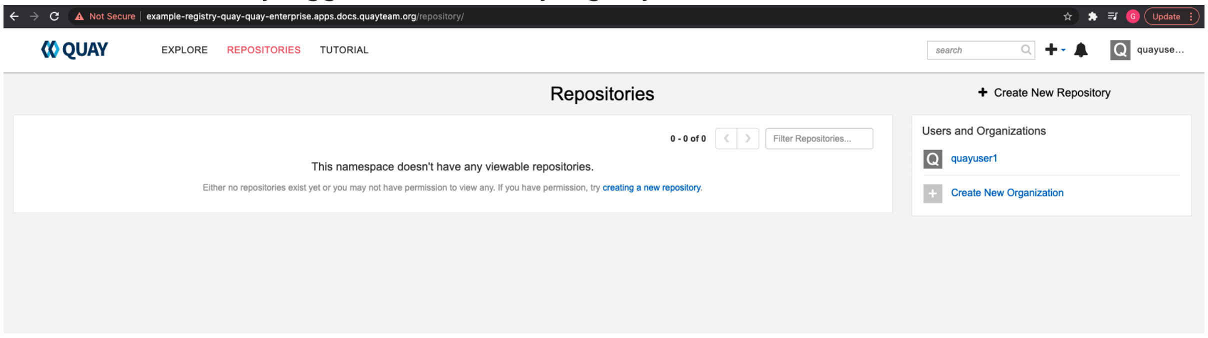
quayuser1@example.com

Password:

•••••••••••

•••••••••••

6. You are automatically logged in to the Quay registry



Not Secure | example-registry-quay-quay-enterprise.apps.docs.quayteam.org/repository/

Repositories

Create New Repository

0 - 0 of 0

Filter Repositories...

This namespace doesn't have any viewable repositories.

Either no repositories exist yet or you may not have permission to view any. If you have permission, try creating a new repository.

Users and Organizations

quayuser1

Create New Organization

CHAPTER 5. CONFIGURING QUAY ON OPENSHIFT

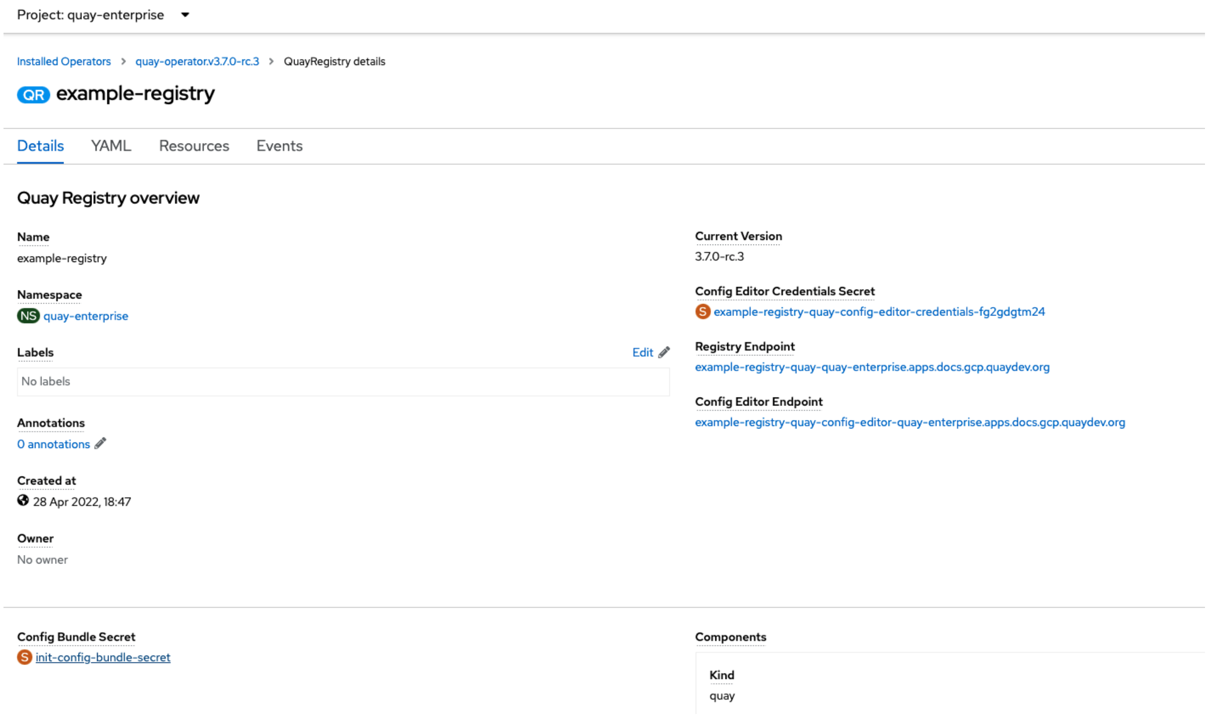
Once deployed, you can configure the Quay application by editing the Quay configuration bundle secret **spec.configBundleSecret** and you can also change the managed status of components in the **spec.components** object of the QuayRegistry resource

Alternatively, you can use the config editor UI to configure the Quay application, as described in the section Chapter 6, *Using the config tool to reconfigure Quay on OpenShift* .

5.1. EDITING THE CONFIG BUNDLE SECRET IN THE OPENSHIFT CONSOLE

Procedure

1. On the Quay Registry overview screen, click the link for the Config Bundle Secret:


Project: quay-enterprise

Installed Operators > quay-operator.v3.7.0-rc.3 > QuayRegistry details

QR example-registry

Details YAML Resources Events

Quay Registry overview

Name
example-registry

Namespace
NS quay-enterprise

Labels Edit
No labels

Annotations
0 annotations

Created at
28 Apr 2022, 18:47

Owner
No owner

Current Version
3.7.0-rc.3

Config Editor Credentials Secret
S example-registry-quay-config-editor-credentials-fg2gdgtm24

Registry Endpoint
example-registry-quay-quay-enterprise.apps.docs.gcp.quaydev.org

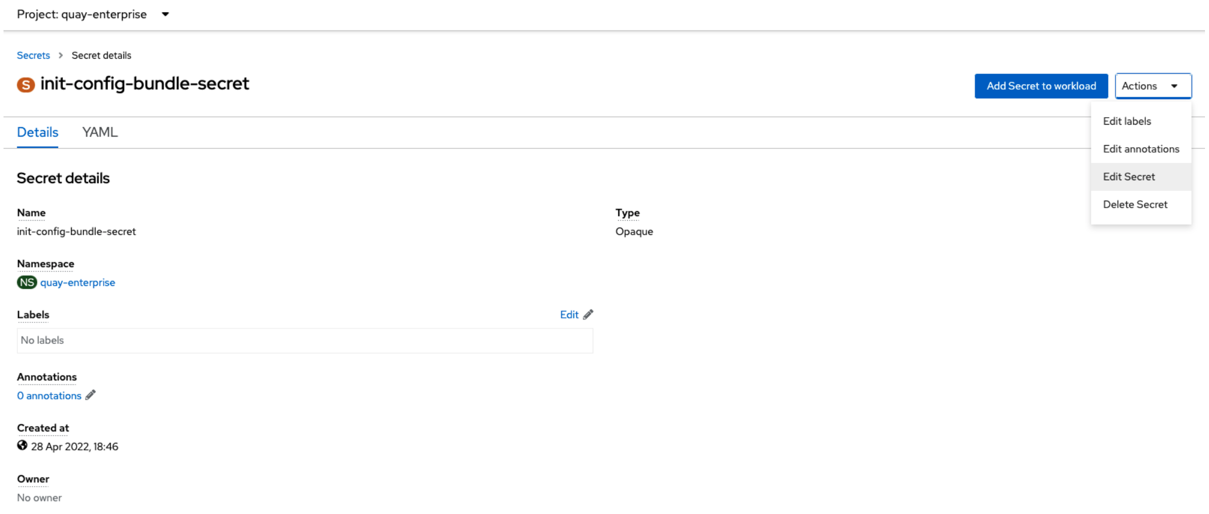
Config Editor Endpoint
example-registry-quay-config-editor-quay-enterprise.apps.docs.gcp.quaydev.org

Config Bundle Secret
S init-config-bundle-secret

Components

Kind
quay

2. To edit the secret, click **Actions → Edit Secret**


Project: quay-enterprise

Secrets > Secret details

S init-config-bundle-secret

Add Secret to workload | Actions

Edit labels
Edit annotations
Edit Secret
Delete Secret

Details YAML

Secret details

Name
init-config-bundle-secret

Type
Opaque

Namespace
NS quay-enterprise

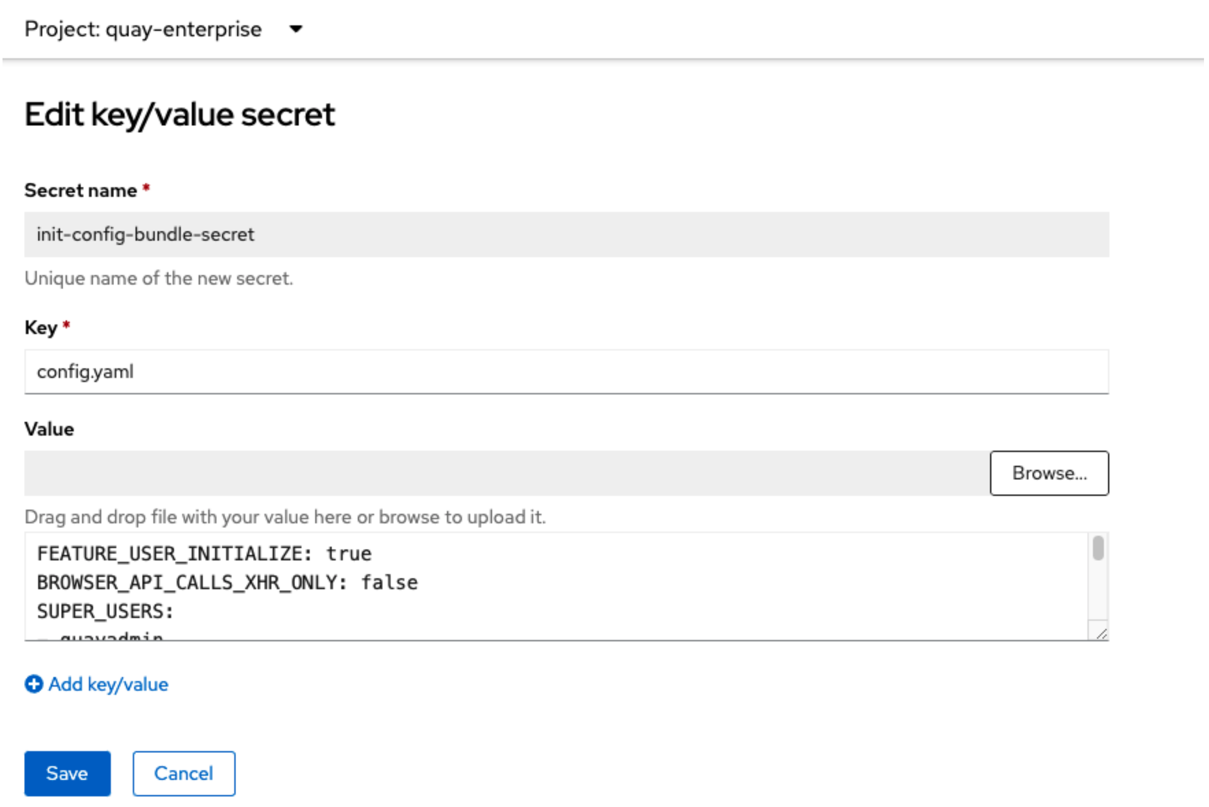
Labels Edit
No labels

Annotations
0 annotations

Created at
28 Apr 2022, 18:46

Owner
No owner

3. Modify the configuration and save the changes


Project: quay-enterprise

Edit key/value secret

Secret name *

init-config-bundle-secret

Unique name of the new secret.

Key *

config.yaml

Value

Drag and drop file with your value here or browse to upload it.

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

4. Monitor the deployment to ensure successful completion and that the configuration changes have taken effect

5.2. DETERMINING QUAYREGISTRY ENDPOINTS AND SECRETS

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

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

apiVersion: quay.redhat.com/v1
kind: QuayRegistry
metadata:
  ...
  name: example-registry
  namespace: quay-enterprise
  ...
spec:
  components:
  - kind: quay
    managed: true
  ...
  - kind: clairpostgres
    managed: true
  configBundleSecret: init-config-bundle-secret
status:
  configEditorCredentialsSecret: example-registry-quay-config-editor-credentials-fg2gdgtm24
  configEditorEndpoint: https://example-registry-quay-config-editor-quay-
enterprise.apps.docs.gcp.quaydev.org
```

```
currentVersion: 3.7.0
lastUpdated: 2022-05-11 13:28:38.199476938 +0000 UTC
registryEndpoint: https://example-registry-quay-quay-enterprise.apps.docs.gcp.quaydev.org
```

The relevant fields are:

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

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

1. Retrieve the secret:

```
$ oc get secret -n quay-enterprise example-registry-quay-config-editor-credentials-fg2gdgtm24 -o yaml

apiVersion: v1
data:
  password: SkZwQkVKTUN0a1BUZmp4dA==
  username: cXVheWNvbmZpZw==
kind: Secret
```

2. Decode the username:

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

quayconfig
```

3. Decode the password:

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

JFpBEJMCtkPTfjxt
```

5.3. DOWNLOADING THE EXISTING CONFIGURATION

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

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

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

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

2. Using the config bundle secret

   a. Get the secret data:

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

Sample output

```
{
    "config.yaml": "RkVBVFVSRV9VU0 ... MDAwMAo="
}
```

   b. Decode the data:

```
$ echo 'RkVBVFVSRV9VU0 ... MDAwMAo=' | base64 --decode
```

```
FEATURE_USER_INITIALIZE: true
BROWSER_API_CALLS_XHR_ONLY: false
SUPER_USERS:
- quayadmin
FEATURE_USER_CREATION: false
FEATURE_QUOTA_MANAGEMENT: true
FEATURE_PROXY_CACHE: true
FEATURE_BUILD_SUPPORT: true
DEFAULT_SYSTEM_REJECT_QUOTA_BYTES: 102400000
```

5.4. USING THE CONFIG BUNDLE TO CONFIGURE CUSTOM SSL CERTS

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

5.4.1. Set TLS to unmanaged

In your Quay Registry yaml, set **kind: tls** to **managed: false**:

```
- kind: tls
  managed: false
```

In the events, you should see that the change is blocked until you set up the appropriate config:

```
  - lastTransitionTime: '2022-03-28T12:56:49Z'
    lastUpdateTime: '2022-03-28T12:56:49Z'
    message: >-
      required component `tls` marked as unmanaged, but `configBundleSecret`
      is missing necessary fields
    reason: ConfigInvalid
    status: 'True'
```

5.4.2. Add certs to config bundle

Procedure

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

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

   custom-ssl-config-bundle.yaml

```
apiVersion: v1
kind: Secret
metadata:
  name: custom-ssl-config-bundle-secret
  namespace: quay-enterprise
data:
  config.yaml: |
    FEATURE_USER_INITIALIZE: true
    BROWSER_API_CALLS_XHR_ONLY: false
    SUPER_USERS:
    - quayadmin
    FEATURE_USER_CREATION: false
    FEATURE_QUOTA_MANAGEMENT: true
    FEATURE_PROXY_CACHE: true
    FEATURE_BUILD_SUPPORT: true
    DEFAULT_SYSTEM_REJECT_QUOTA_BYTES: 102400000
  extra_ca_cert_my-custom-ssl.crt: |
    -----BEGIN CERTIFICATE-----
    MIIDsDCCApigAwIBAgIUCqlzkHjF5i5TXLFy+sepFrZr/UswDQYJKoZIhvcNAQEL

BQAwbzELMAkGA1UEBhMCSUUxDzANBgNVBAgMBkdBTFdBWTEPMA0GA1UEBwwG
R0FM
    ....
    -----END CERTIFICATE-----
```

   Next, create the secret from the YAML file:

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

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

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

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

quayregistry.yaml

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

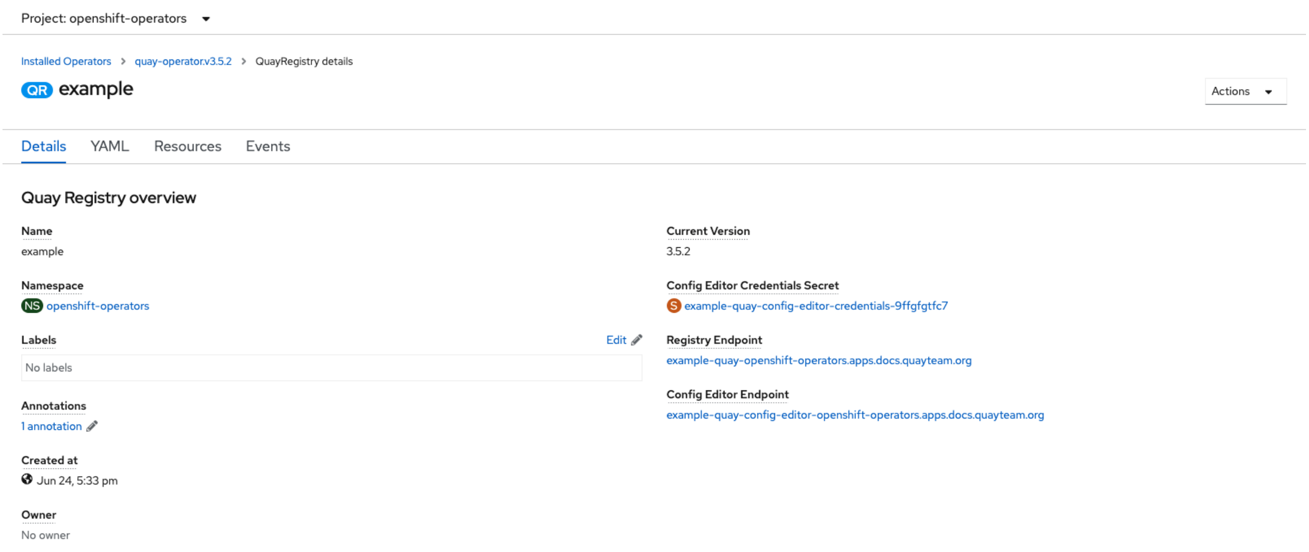
3. Deploy or update the registry using the YAML file:

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

CHAPTER 6. USING THE CONFIG TOOL TO RECONFIGURE QUAY ON OPENSHIFT

6.1. ACCESSING THE CONFIG EDITOR

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


Project: openshift-operators
Installed Operators > quay-operator.v3.5.2 > QuayRegistry details
QR example
Actions
Details YAML Resources Events
Edit
1 annotation

Quay Registry overview

Name
example

Namespace
NS openshift-operators

Labels
No labels

Annotations
1 annotation

Created at
Jun 24, 5:33 pm

Owner
No owner

Current Version
3.5.2

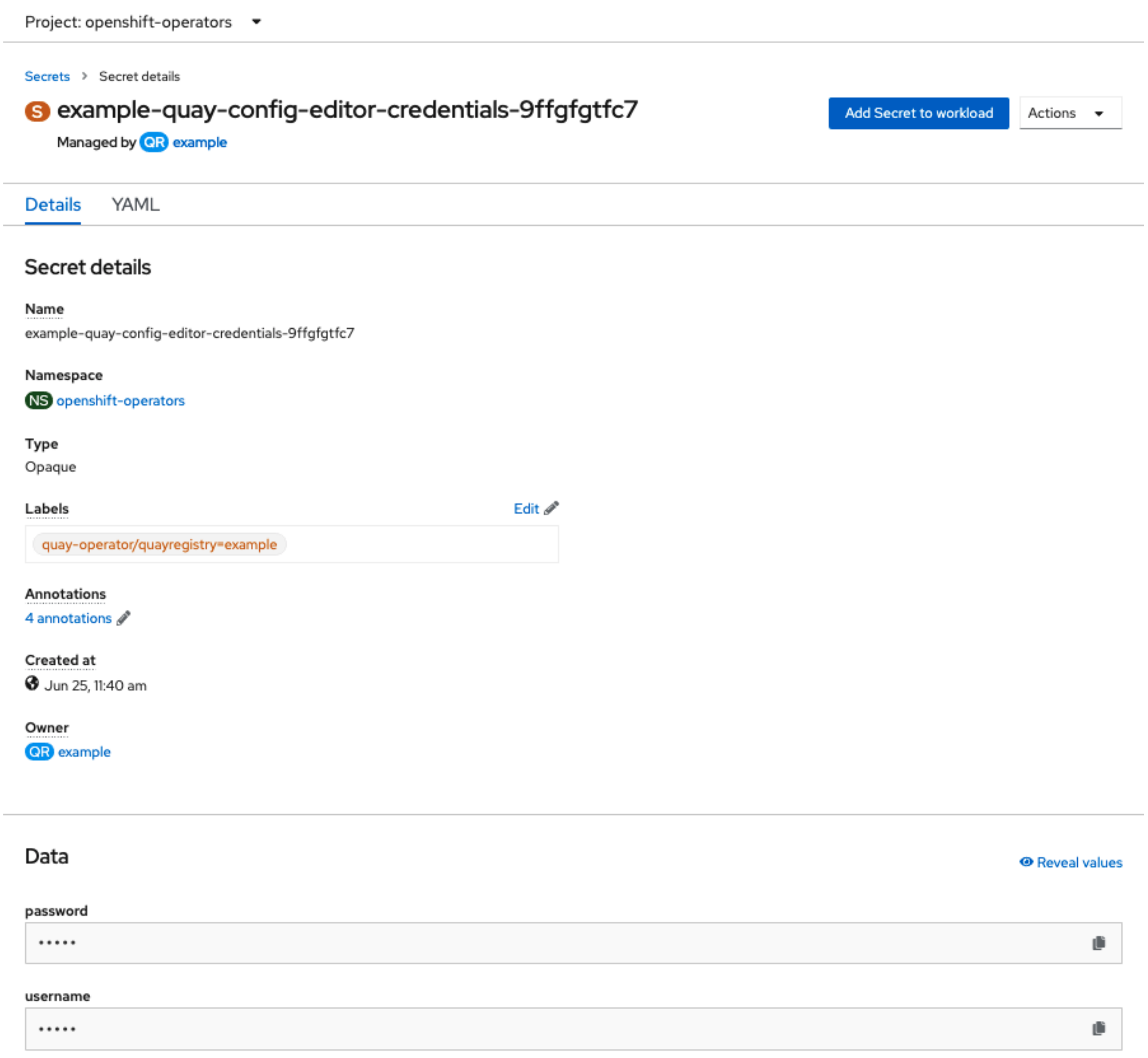
Config Editor Credentials Secret
S example-quay-config-editor-credentials-9ffgfgtfc7

Registry Endpoint
example-quay-openshift-operators.apps.docs.quayteam.org

Config Editor Endpoint
example-quay-config-editor-openshift-operators.apps.docs.quayteam.org

6.1.1. Retrieving the config editor credentials

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


Project: openshift-operators

Secrets > Secret details

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

Managed by QR example

Add Secret to workload | Actions

Details | YAML

Secret details

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

Namespace
NS openshift-operators

Type
Opaque

Labels
Edit
quay-operator/quayregistry=example

Annotations
4 annotations

Created at
Jun 25, 11:40 am

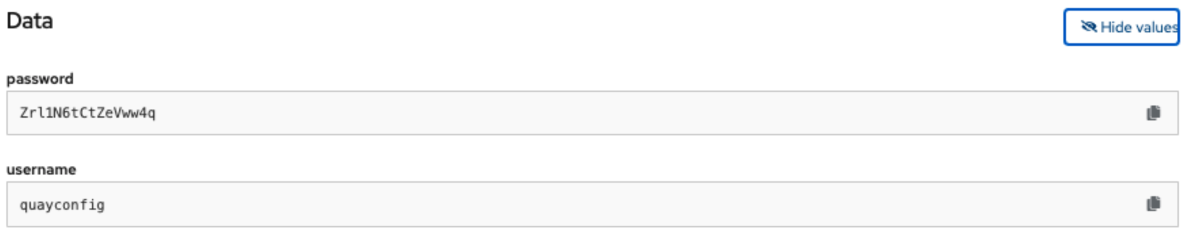
Owner
QR example

Data
Reveal values

password
•••••

username
•••••

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

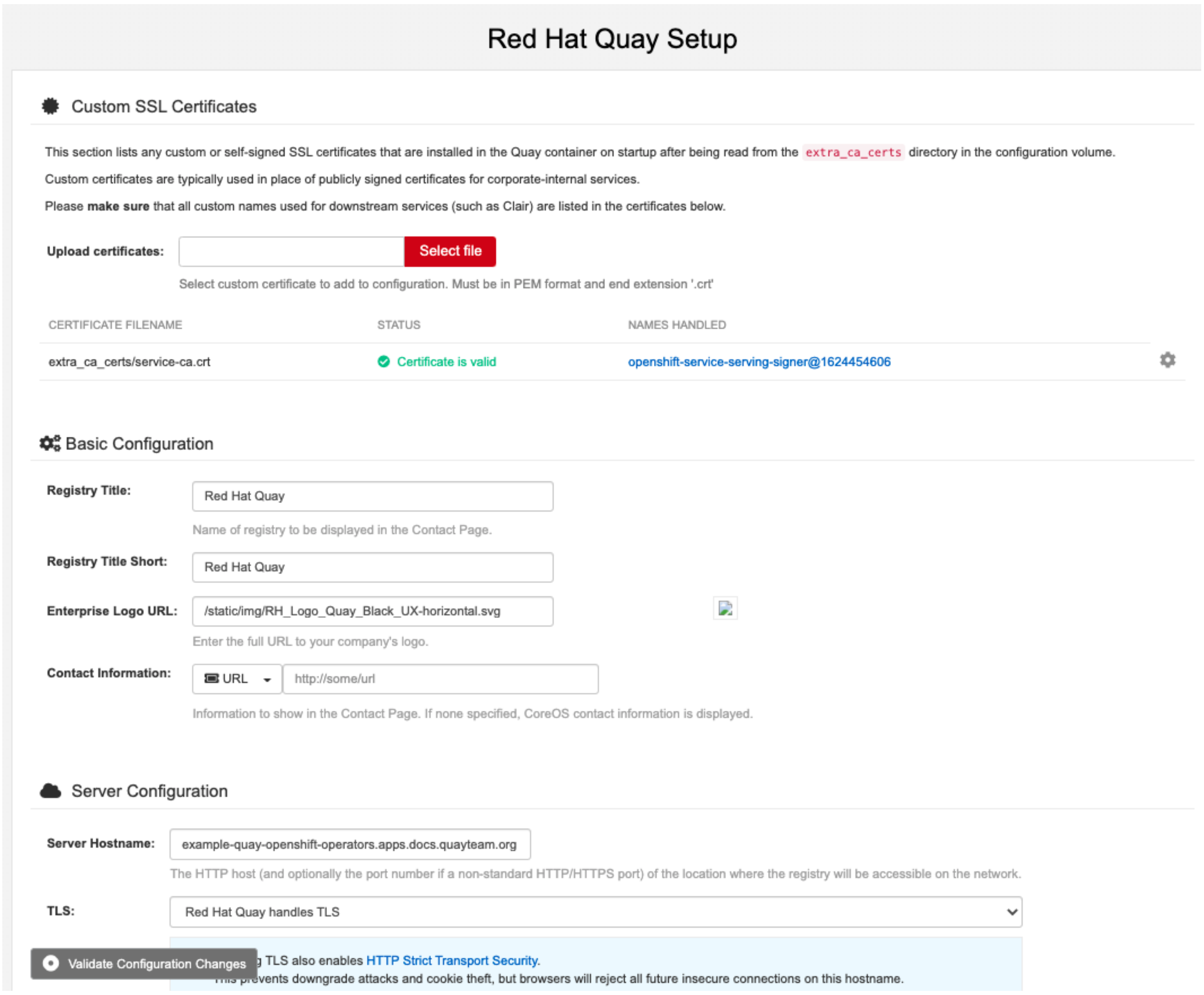

Data
Hide values

password
```
Zrl1N6tCtZeVww4q
```

username
```
quayconfig
```

6.1.2. Logging in to the config editor

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



Red Hat Quay Setup

Custom SSL Certificates

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

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

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

Upload certificates: Select file

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

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

Basic Configuration

Registry Title: Red Hat Quay

Name of registry to be displayed in the Contact Page.

Registry Title Short: Red Hat Quay

Enterprise Logo URL: /static/img/RH_Logo_Quay_Black_UX-horizontal.svg

Enter the full URL to your company's logo.

Contact Information: URL http://some/url

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

Server Configuration

Server Hostname: example-quay-openshift-operators.apps.docs.quayteam.org

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

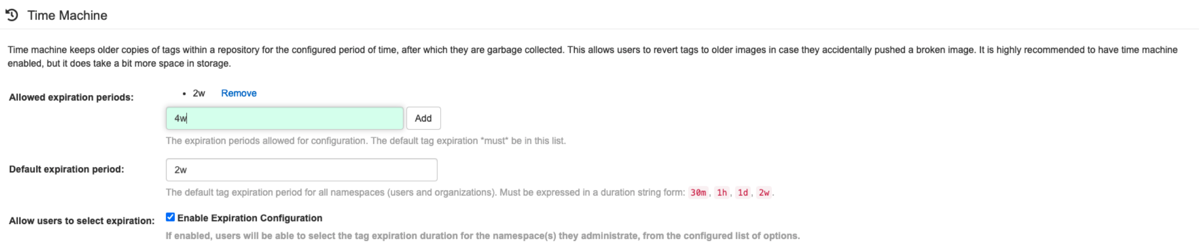
TLS: Red Hat Quay handles TLS

g TLS also enables HTTP Strict Transport Security.
This prevents downgrade attacks and cookie theft, but browsers will reject all future insecure connections on this hostname.

6.1.3. Changing configuration

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

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



Time Machine

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

Allowed expiration periods:
- 2w Remove

4w Add

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

Default expiration period: 2w

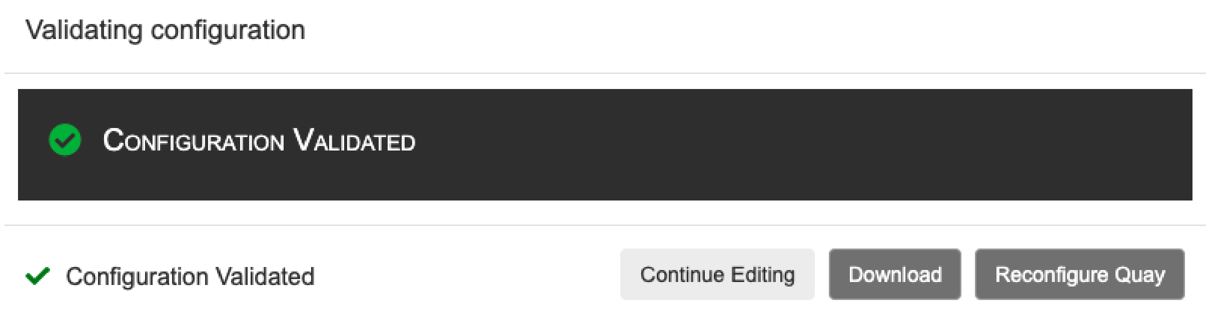
The default tag expiration period for all namespaces (users and organizations). Must be expressed in a duration string form: `30m`, `1h`, `1d`, `2w`.

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

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

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

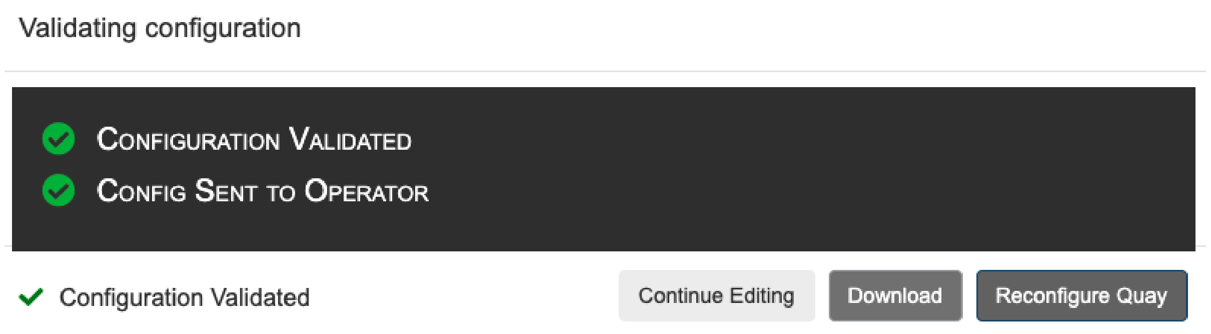
3. Apply the changes by pressing the **Reconfigure Quay** button:


Validating configuration

CONFIGURATION VALIDATED

✓ Configuration Validated

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


Validating configuration

CONFIGURATION VALIDATED

CONFIG SENT TO OPERATOR

✓ Configuration Validated

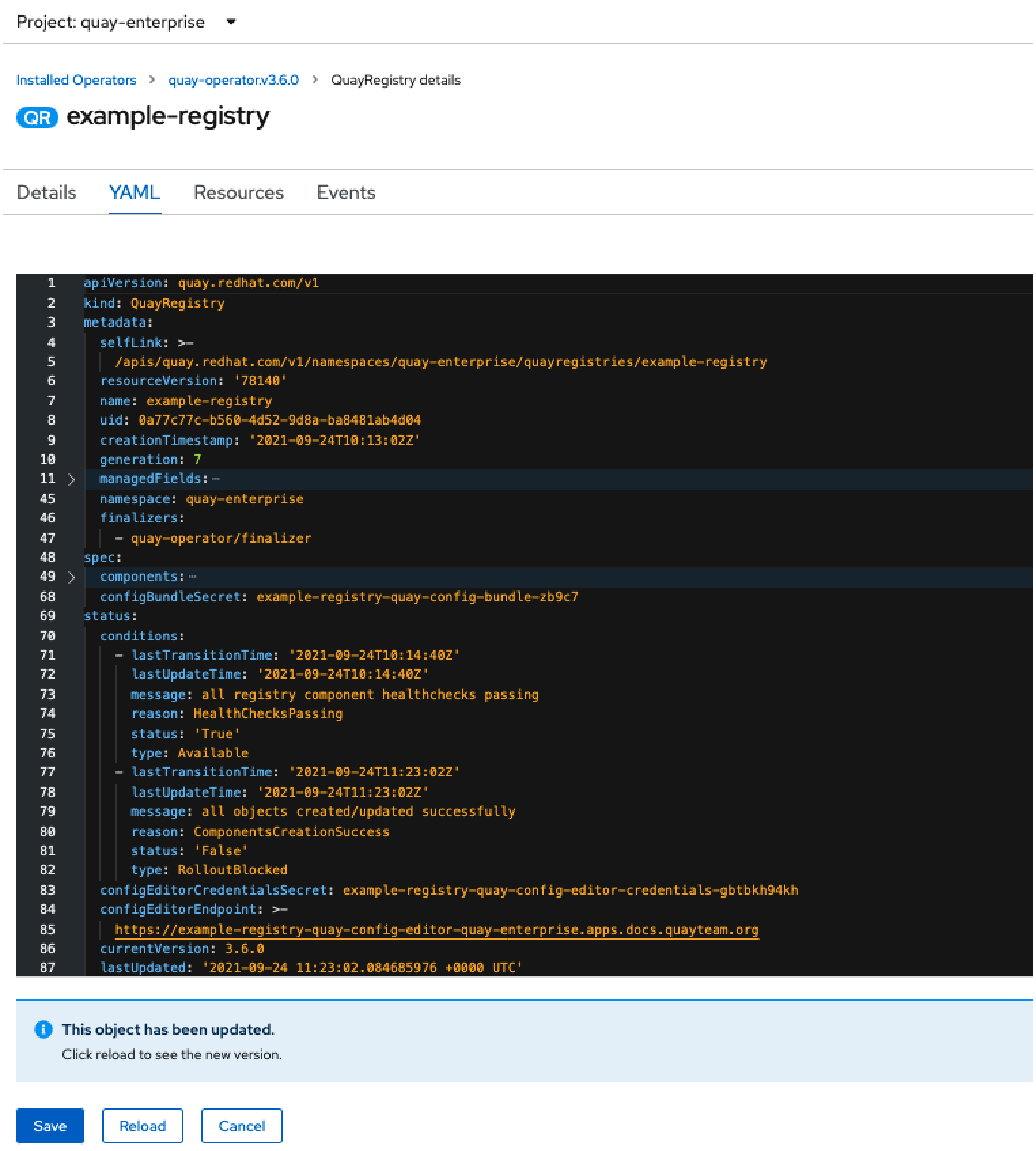
NOTE

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.

6.2. MONITORING RECONFIGURATION IN THE UI

6.2.1. QuayRegistry resource

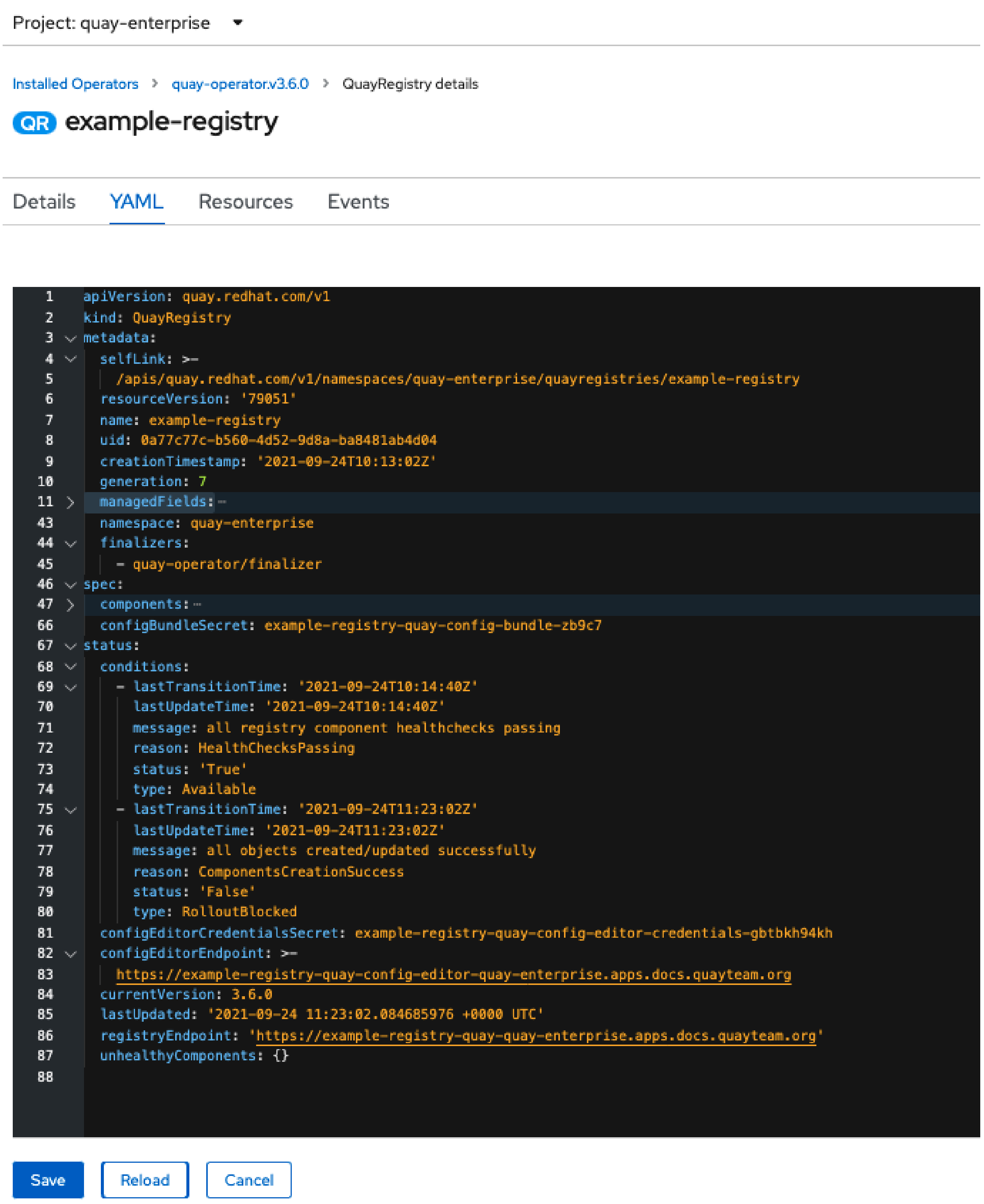
After reconfiguring the Operator, you can track the progress of the redeployment in the YAML tab for the specific instance of QuayRegistry, in this case, **example-registry**:



```
apiVersion: quay.redhat.com/v1
kind: QuayRegistry
metadata:
  selfLink: >-
    /apis/quay.redhat.com/v1/namespaces/quay-enterprise/quayregistries/example-registry
  resourceVersion: '78140'
  name: example-registry
  uid: 0a77c77c-b560-4d52-9d8a-ba8481ab4d04
  creationTimestamp: '2021-09-24T10:13:02Z'
  generation: 7
  managedFields: …
  namespace: quay-enterprise
  finalizers:
    - quay-operator/finalizer
spec:
  components: …
  configBundleSecret: example-registry-quay-config-bundle-zb9c7
status:
  conditions:
    - lastTransitionTime: '2021-09-24T10:14:40Z'
      lastUpdateTime: '2021-09-24T10:14:40Z'
      message: all registry component healthchecks passing
      reason: HealthChecksPassing
      status: 'True'
      type: Available
    - lastTransitionTime: '2021-09-24T11:23:02Z'
      lastUpdateTime: '2021-09-24T11:23:02Z'
      message: all objects created/updated successfully
      reason: ComponentsCreationSuccess
      status: 'False'
      type: RolloutBlocked
  configEditorCredentialsSecret: example-registry-quay-config-editor-credentials-gbtbkh94kh
  configEditorEndpoint: >-
    https://example-registry-quay-config-editor-quay-enterprise.apps.docs.quayteam.org
  currentVersion: 3.6.0
  lastUpdated: '2021-09-24 11:23:02.084685976 +0000 UTC'
```

This object has been updated.
Click reload to see the new version.

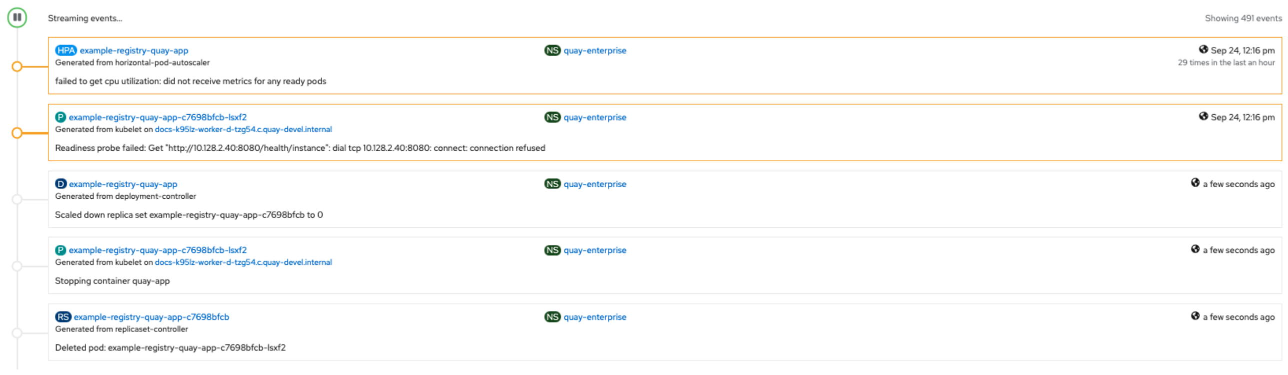
Each time the status changes, you will be prompted to reload the data to see the updated version. Eventually, the Operator will reconcile the changes, and there will be no unhealthy components reported.



```
apiVersion: quay.redhat.com/v1
kind: QuayRegistry
metadata:
  selfLink: >-
    /apis/quay.redhat.com/v1/namespaces/quay-enterprise/quayregistries/example-registry
  resourceVersion: '79051'
  name: example-registry
  uid: 0a77c77c-b560-4d52-9d8a-ba8481ab4d04
  creationTimestamp: '2021-09-24T10:13:02Z'
  generation: 7
  managedFields:…
  namespace: quay-enterprise
  finalizers:
    - quay-operator/finalizer
spec:
  components:…
  configBundleSecret: example-registry-quay-config-bundle-zb9c7
status:
  conditions:
    - lastTransitionTime: '2021-09-24T10:14:40Z'
      lastUpdateTime: '2021-09-24T10:14:40Z'
      message: all registry component healthchecks passing
      reason: HealthChecksPassing
      status: 'True'
      type: Available
    - lastTransitionTime: '2021-09-24T11:23:02Z'
      lastUpdateTime: '2021-09-24T11:23:02Z'
      message: all objects created/updated successfully
      reason: ComponentsCreationSuccess
      status: 'False'
      type: RolloutBlocked
  configEditorCredentialsSecret: example-registry-quay-config-editor-credentials-gbtbkh94kh
  configEditorEndpoint: >-
    https://example-registry-quay-config-editor-quay-enterprise.apps.docs.quayteam.org
  currentVersion: 3.6.0
  lastUpdated: '2021-09-24 11:23:02.084685976 +0000 UTC'
  registryEndpoint: 'https://example-registry-quay-quay-enterprise.apps.docs.quayteam.org'
  unhealthyComponents: {}
```

6.2.2. Events

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



Streaming events... Showing 491 events

HPA example-registry-quay-app NS quay-enterprise Sep 24, 12:16 pm
Generated from horizontal-pod-autoscaler 29 times in the last an hour
failed to get cpu utilization: did not receive metrics for any ready pods

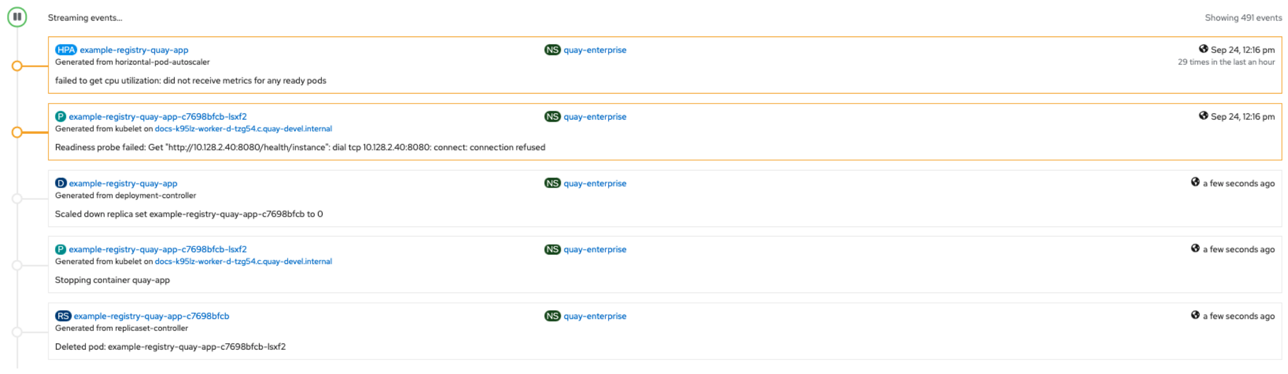
P example-registry-quay-app-c7698bfcb-lsxf2 NS quay-enterprise Sep 24, 12:16 pm
Generated from kubelet on docs-k95lz-worker-d-tzg54.c.quay-devel.internal
Readiness probe failed: Get "http://10.128.2.40:8080/health/instance": dial tcp 10.128.2.40:8080: connect: connection refused

D example-registry-quay-app NS quay-enterprise a few seconds ago
Generated from deployment-controller
Scaled down replica set example-registry-quay-app-c7698bfcb to 0

P example-registry-quay-app-c7698bfcb-lsxf2 NS quay-enterprise a few seconds ago
Generated from kubelet on docs-k95lz-worker-d-tzg54.c.quay-devel.internal
Stopping container quay-app

RS example-registry-quay-app-c7698bfcb NS quay-enterprise a few seconds ago
Generated from replicaset-controller
Deleted pod: example-registry-quay-app-c7698bfcb-lsxf2

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



Streaming events... Showing 491 events

HPA example-registry-quay-app NS quay-enterprise Sep 24, 12:16 pm
Generated from horizontal-pod-autoscaler 29 times in the last an hour
failed to get cpu utilization: did not receive metrics for any ready pods

P example-registry-quay-app-c7698bfcb-lsxf2 NS quay-enterprise Sep 24, 12:16 pm
Generated from kubelet on docs-k95lz-worker-d-tzg54.c.quay-devel.internal
Readiness probe failed: Get "http://10.128.2.40:8080/health/instance": dial tcp 10.128.2.40:8080: connect: connection refused

D example-registry-quay-app NS quay-enterprise a few seconds ago
Generated from deployment-controller
Scaled down replica set example-registry-quay-app-c7698bfcb to 0

P example-registry-quay-app-c7698bfcb-lsxf2 NS quay-enterprise a few seconds ago
Generated from kubelet on docs-k95lz-worker-d-tzg54.c.quay-devel.internal
Stopping container quay-app

RS example-registry-quay-app-c7698bfcb NS quay-enterprise a few seconds ago
Generated from replicaset-controller
Deleted pod: example-registry-quay-app-c7698bfcb-lsxf2

6.3. ACCESSING UPDATED INFORMATION AFTER RECONFIGURATION

6.3.1. 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.

6.3.2. Accessing the updated config.yaml in the UI

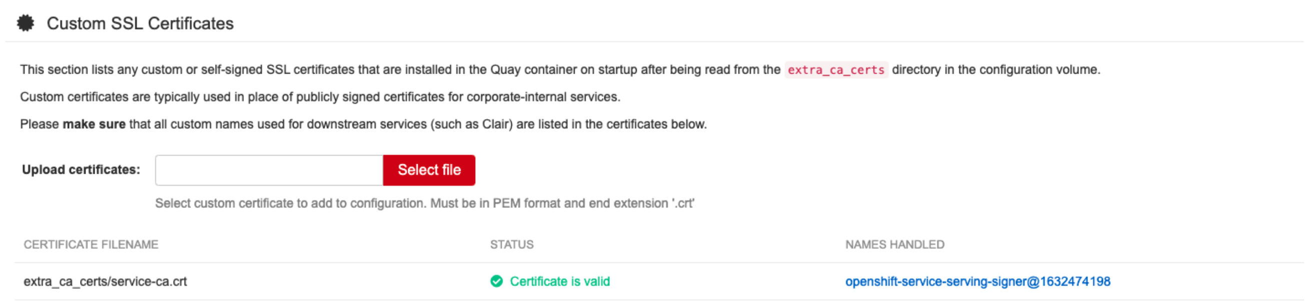
Use the config bundle to access the updated **config.yaml** file.

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
...
```

6.4. CUSTOM SSL CERTIFICATES UI

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



Custom SSL Certificates

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

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

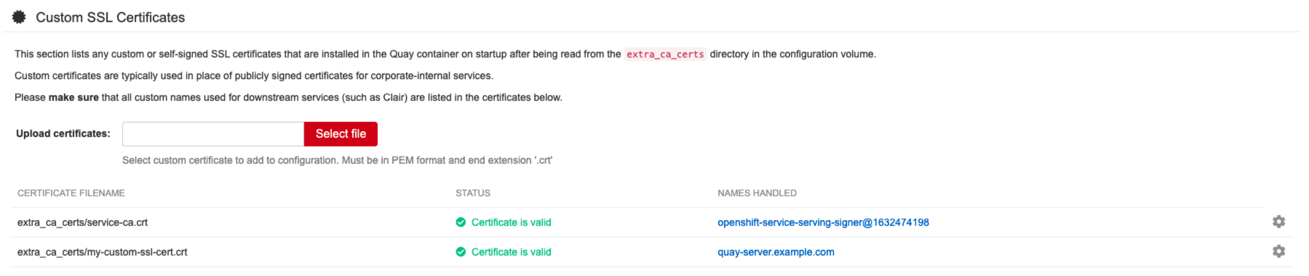
Please **make sure** that all custom names used for downstream services (such as Clair) are listed in the certificates below.

Upload certificates: Select file

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

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

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



Custom SSL Certificates

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

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

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

Upload certificates: Select file

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

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

6.5. EXTERNAL ACCESS TO THE REGISTRY

When running on OpenShift, the **Routes** API is available and will automatically be used as a managed component. After creating the **QuayRegistry**, the external access point can be found in the status block of the **QuayRegistry**:

```
status:
  registryEndpoint: some-quay.my-namespace.apps.mycluster.com
```

CHAPTER 7. QUAY OPERATOR FEATURES

7.1. CONSOLE MONITORING AND ALERTING

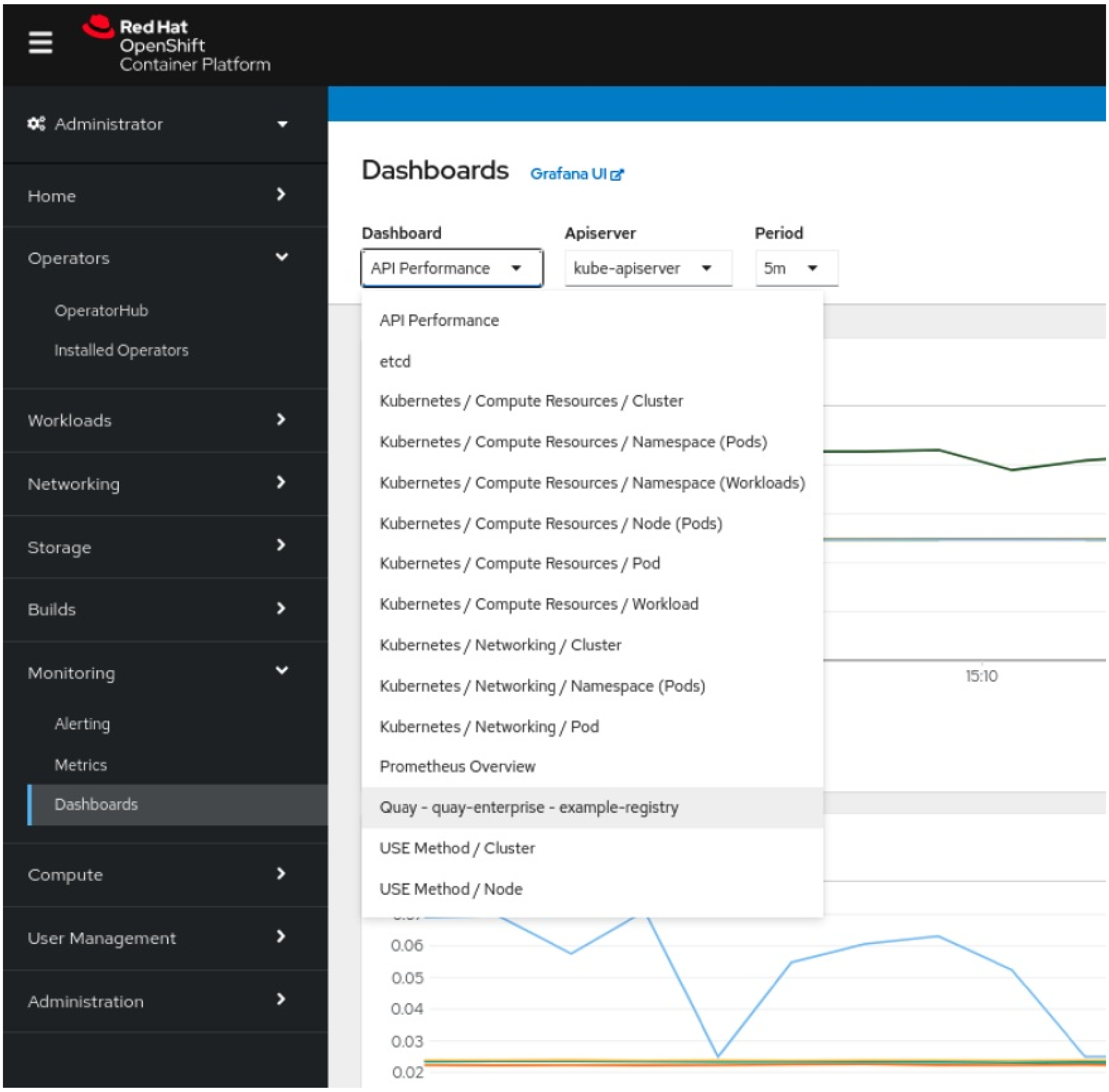
Red Hat Quay provides support for monitoring Quay instances that were deployed using the Operator, from inside the OpenShift console. The new monitoring features include a Grafana dashboard, access to individual metrics, and alerting to notify for frequently restarting Quay pods.

NOTE

To enable the monitoring features, the Operator must be installed in "all namespaces" mode.

7.1.1. Dashboard

In the OpenShift console, navigate to Monitoring → Dashboards and search for the dashboard of your desired Quay registry instance:

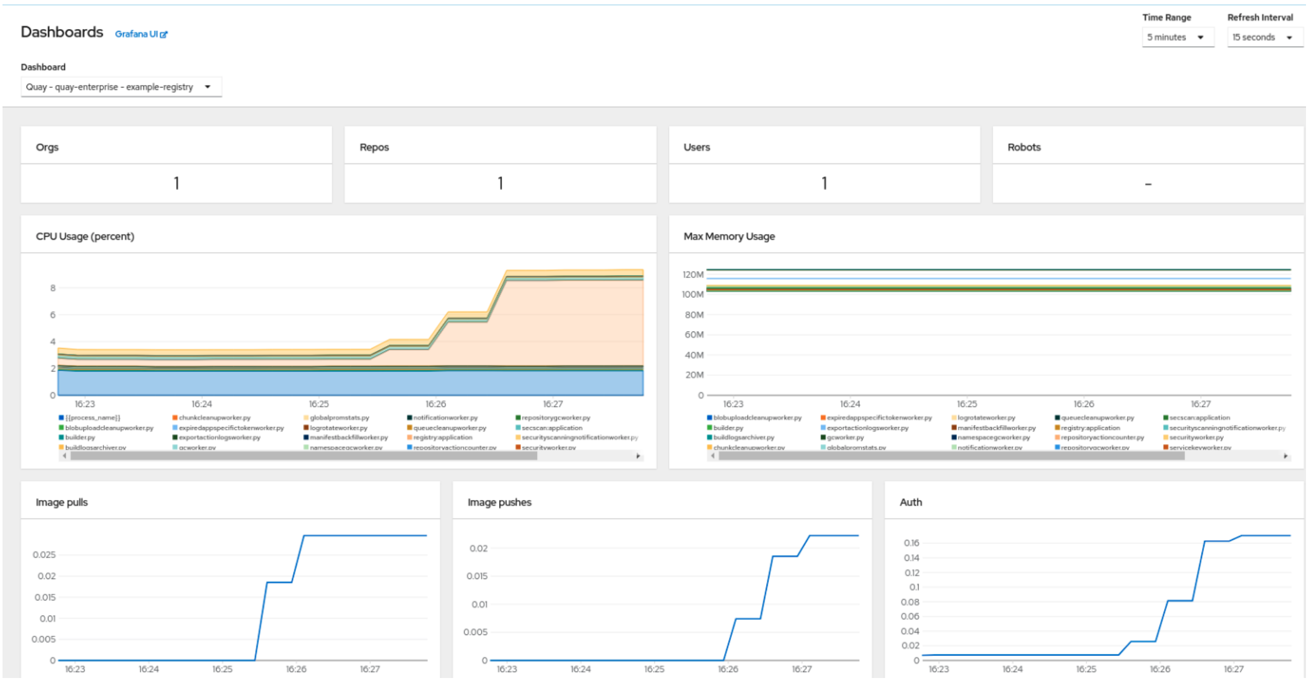


Dashboard: API Performance
Apiserver: kube-apiserver
Period: 5m

API Performance
etcd
Kubernetes / Compute Resources / Cluster
Kubernetes / Compute Resources / Namespace (Pods)
Kubernetes / Compute Resources / Namespace (Workloads)
Kubernetes / Compute Resources / Node (Pods)
Kubernetes / Compute Resources / Pod
Kubernetes / Compute Resources / Workload
Kubernetes / Networking / Cluster
Kubernetes / Networking / Namespace (Pods)
Kubernetes / Networking / Pod
Prometheus Overview
Quay - quay-enterprise - example-registry
USE Method / Cluster
USE Method / Node

The dashboard shows various statistics including:

- The number of Organizations, Repositories, Users and Robot accounts
- CPU Usage and Max Memory Usage
- Rates of Image Pulls and Pushes, and Authentication requests
- API request rate
- Latencies


Dashboards Grafana UI

Time Range: 5 minutes

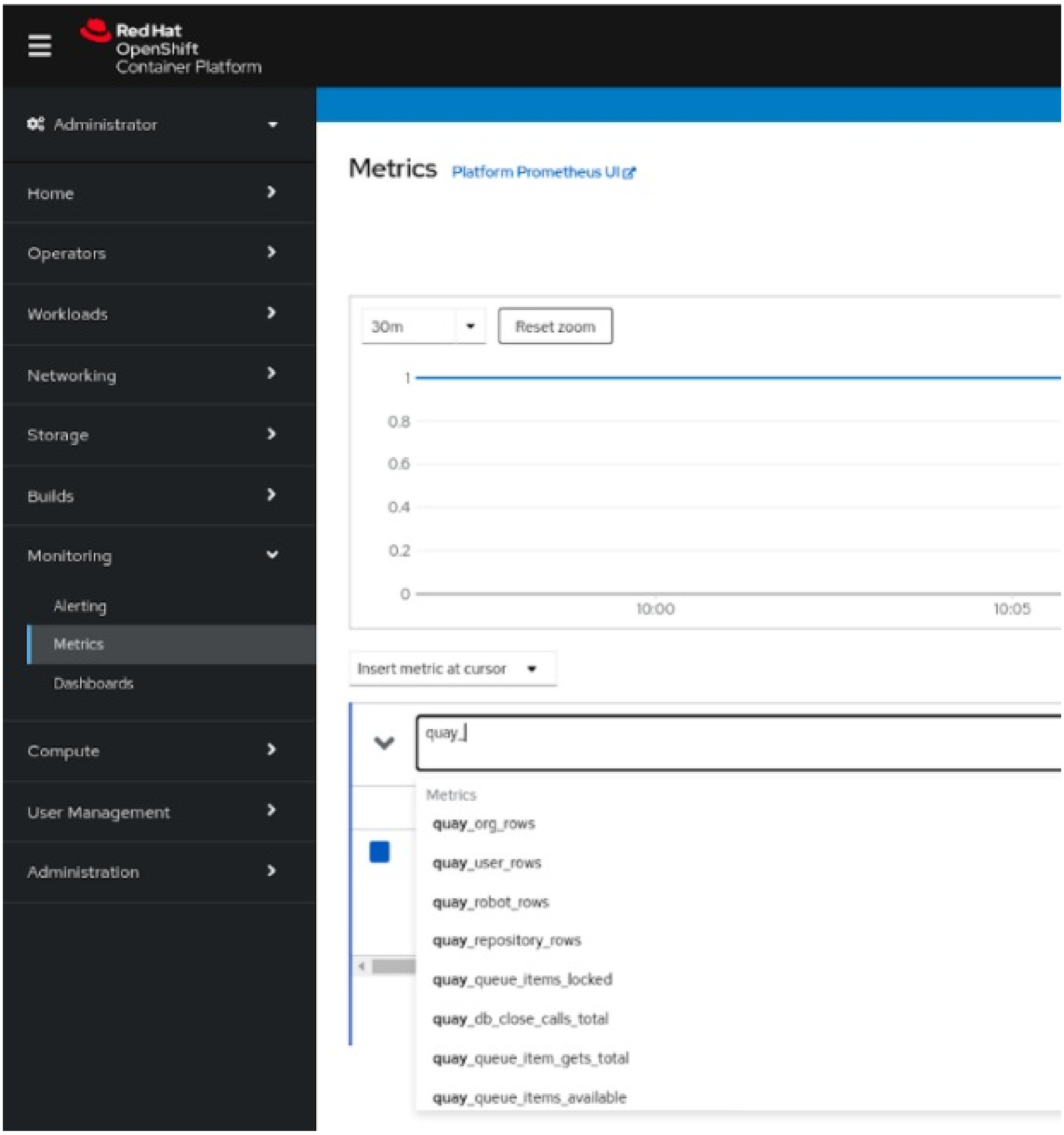
Refresh Interval: 15 seconds

Dashboard: Quay - quay-enterprise - example-registry

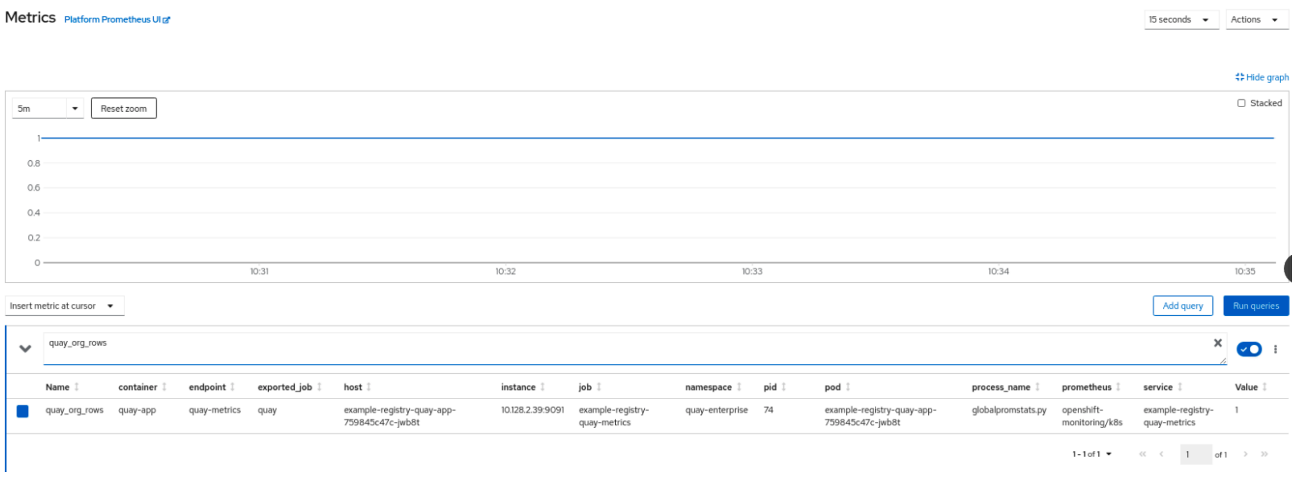
Orgs	Repos	Users	Robots
1	1	1	-

7.1.2. Metrics

You can see the underlying metrics behind the Quay dashboard, by accessing Monitoring → Metrics in the UI. In the Expression field, enter the text **quay_** to see the list of metrics available:



Select a sample metric, for example, **quay_org_rows**:

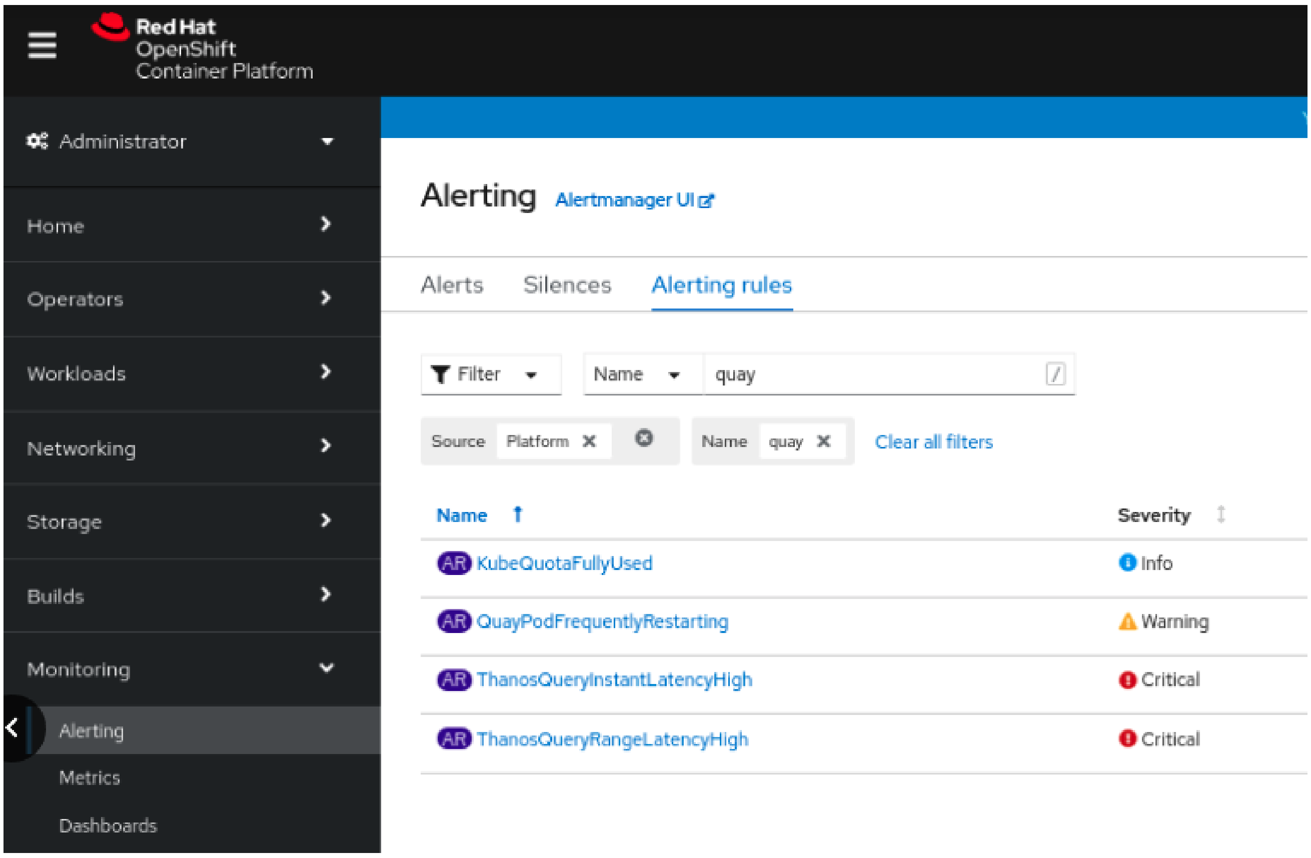


	Name	container	endpoint	exported_job	host	instance	job	namespace	pid	pod	process_name	prometheus	service	Value
🟦	quay_org_rows	quay-app	quay-metrics	quay	example-registry-quay-app-759845c47c-jwb8t	10.128.2.39:9091	example-registry-quay-metrics	quay-enterprise	74	example-registry-quay-app-759845c47c-jwb8t	globalpromstats.py	openshift-monitoring/k8s	example-registry-quay-metrics	1

This metric shows the number of organizations in the registry, and it is directly surfaced in the dashboard as well.

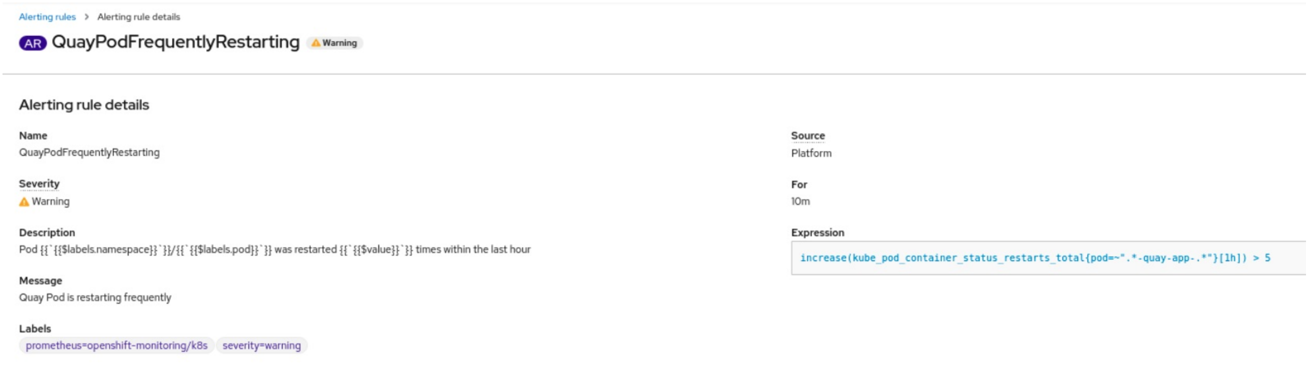
7.1.3. Alerting

An alert is raised if the Quay pods restart too often. The alert can be configured by accessing the Alerting rules tab from Monitoring → Alerting in the consol UI and searching for the Quay-specific alert:



Name	Severity
AR KubeQuotaFullyUsed	Info
AR QuayPodFrequentlyRestarting	Warning
AR ThanosQueryInstantLatencyHigh	Critical
AR ThanosQueryRangeLatencyHigh	Critical

Select the QuayPodFrequentlyRestarting rule detail to configure the alert:


Alerting rules > Alerting rule details
AR QuayPodFrequentlyRestarting Warning

Alerting rule details

Name
QuayPodFrequentlyRestarting

Severity
Warning

Description
Pod {{`{{$labels.namespace}}`}}/{{`{{$labels.pod}}`}} was restarted {{`{{$value}}`}} times within the last hour

Message
Quay Pod is restarting frequently

Labels
prometheus=openshift-monitoring/k8s severity=warning

Source
Platform

For
10m

Expression

```
increase(kube_pod_container_status_restarts_total{pod=~".*-quay-app-.*"}[1h]) > 5
```

7.2. MANUALLY UPDATING THE VULNERABILITY DATABASES FOR CLAIR IN AN AIR-GAPPED OPENSHIFT CLUSTER

Clair utilizes packages called **updaters** that encapsulate the logic of fetching and parsing different vulnerability databases. Clair supports running updaters in a different environment and importing the results. This is aimed at supporting installations that disallow the Clair cluster from talking to the Internet directly.

To manually update the vulnerability databases for Clair in an air-gapped OpenShift cluster, use the following steps:

- Obtain the **clairctl** program
- Retrieve the Clair config
- Use **clairctl** to export the updaters bundle from a Clair instance that has access to the internet
- Update the Clair config in the air-gapped OpenShift cluster to allow access to the Clair database
- Transfer the updaters bundle from the system with internet access, to make it available inside the air-gapped environment
- Use **clairctl** to import the updaters bundle into the Clair instance for the air-gapped OpenShift cluster

7.2.1. Obtaining clairctl

To obtain the **clairctl** program from a Clair deployment in an OpenShift cluster, use the **oc cp** command, for example:

```
$ oc -n quay-enterprise cp example-registry-clair-app-64dd48f866-6ptgw:/usr/bin/clairctl ./clairctl
$ chmod u+x ./clairctl
```

For a standalone Clair deployment, use the **podman cp** command, for example:

```
$ sudo podman cp clairv4:/usr/bin/clairctl ./clairctl
$ chmod u+x ./clairctl
```

7.2.2. Retrieving the Clair config

7.2.2.1. Clair on OpenShift config

To retrieve the configuration file for a Clair instance deployed using the OpenShift Operator, retrieve and decode the config secret using the appropriate namespace, and save it to file, for example:

```
$ kubectl get secret -n quay-enterprise example-registry-clair-config-secret  -o "jsonpath=
{$.data['config\.yaml']}" | base64 -d > clair-config.yaml
```

An excerpt from a Clair configuration file is shown below:

clair-config.yaml

```
http_listen_addr: :8080
introspection_addr: ""
log_level: info
indexer:
    connstring: host=example-registry-clair-postgres port=5432 dbname=postgres user=postgres
password=postgres sslmode=disable
    scanlock_retry: 10
    layer_scan_concurrency: 5
    migrations: true
    scanner:
      package: {}
      dist: {}
      repo: {}
    airgap: false
matcher:
    connstring: host=example-registry-clair-postgres port=5432 dbname=postgres user=postgres
password=postgres sslmode=disable
    max_conn_pool: 100
    indexer_addr: ""
    migrations: true
    period: null
    disable_updaters: false
notifier:
    connstring: host=example-registry-clair-postgres port=5432 dbname=postgres user=postgres
password=postgres sslmode=disable
    migrations: true
    indexer_addr: ""
    matcher_addr: ""
    poll_interval: 5m
    delivery_interval: 1m
    ...
```

7.2.2.2. Standalone Clair config

For standalone Clair deployments, the config file is the one specified in CLAIR_CONF environment variable in the **podman run** command, for example:

```
sudo podman run -d --rm --name clairv4 \
  -p 8081:8081 -p 8089:8089 \
  -e CLAIR_CONF=/clair/config.yaml -e CLAIR_MODE=combo \
  -v /etc/clairv4/config:/clair:Z \
  registry.redhat.io/quay/clair-rhel8:v3.8.0
```

7.2.3. Exporting the updaters bundle

From a Clair instance that has access to the internet, use **clairctl** with the appropriate configuration file to export the updaters bundle:

```
$ ./clairctl --config ./config.yaml export-updaters updates.gz
```

7.2.4. Configuring access to the Clair database in the air-gapped OpenShift cluster

- Use **kubectl** to determine the Clair database service:

  ```
  $ kubectl get svc -n quay-enterprise

  NAME                          TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)
  AGE
  example-registry-clair-app          ClusterIP     172.30.224.93    <none>
  80/TCP,8089/TCP              4d21h
  ```

```
example-registry-clair-postgres      ClusterIP     172.30.246.88    <none>       5432/TCP
4d21h
...
```

- Forward the Clair database port so that it is accessible from the local machine, for example:

```
$ kubectl port-forward -n quay-enterprise service/example-registry-clair-postgres 5432:5432
```

- Update the Clair configuration file:

clair-config.yaml

```
indexer:
    connstring: host=localhost port=5432 dbname=postgres user=postgres
password=postgres sslmode=disable
    scanlock_retry: 10
    layer_scan_concurrency: 5
    migrations: true
    scanner:
    repo:
      rhel-repository-scanner:
        repo2cpe_mapping_file: /data/cpe-map.json
    package:
      rhel_containerscanner:
        name2repos_mapping_file: /data/repo-map.json
```

NOTE

- Replace the value of the **host** in the multiple **connstring** fields with **localhost**.
- As an alternative to using **kubectl port-forward**, you can use **kubefwd** instead. With this method, there is no need to modify the **connstring** field in the Clair configuration file to use **localhost**.

7.2.5. Importing the updaters bundle into the air-gapped environment

After transferring the updaters bundle to the air-gapped environment, use **clairctl** to import the bundle into the Clair database deployed by the OpenShift Operator:

```
$ ./clairctl --config ./clair-config.yaml import-updaters updates.gz
```

7.3. FIPS READINESS AND COMPLIANCE

FIPS (the Federal Information Processing Standard developed by the National Institute of Standards and Technology, NIST) is regarded as the gold standard for securing and encrypting sensitive data, particularly in heavily regulated areas such as banking, healthcare and the public sector. Red Hat Enterprise Linux and Red Hat OpenShift Container Platform support this standard by providing a FIPS mode in which the system would only allow usage of certain, FIPS-validated cryptographic modules, like **openssl**. This ensures FIPS compliance.

Red Hat Quay supports running on FIPS-enabled RHEL and Red Hat OpenShift Container Platform from version 3.5.

CHAPTER 8. ADVANCED CONCEPTS

8.1. DEPLOYING QUAY ON INFRASTRUCTURE NODES

By default, Quay-related pods are placed on arbitrary worker nodes when using the Operator to deploy the registry. The OpenShift Container Platform documentation shows how to use machine sets to configure nodes to only host infrastructure components (see https://docs.openshift.com/container-platform/4.7/machine_management/creating-infrastructure-machinesets.html).

If you are not using OCP MachineSet resources to deploy infra nodes, this section shows you how to manually label and taint nodes for infrastructure purposes.

Once you have configured your infrastructure nodes, either manually or using machine sets, you can then control the placement of Quay pods on these nodes using node selectors and tolerations.

8.1.1. Label and taint nodes for infrastructure use

In the cluster used in this example, there are three master nodes and six worker nodes:

```
$ oc get nodes
NAME                                          STATUS   ROLES    AGE     VERSION
user1-jcnp6-master-0.c.quay-devel.internal         Ready    master   3h30m   v1.20.0+ba45583
user1-jcnp6-master-1.c.quay-devel.internal         Ready    master   3h30m   v1.20.0+ba45583
user1-jcnp6-master-2.c.quay-devel.internal         Ready    master   3h30m   v1.20.0+ba45583
user1-jcnp6-worker-b-65plj.c.quay-devel.internal   Ready    worker   3h21m   v1.20.0+ba45583
user1-jcnp6-worker-b-jr7hc.c.quay-devel.internal   Ready    worker   3h21m   v1.20.0+ba45583
user1-jcnp6-worker-c-jrq4v.c.quay-devel.internal   Ready    worker   3h21m   v1.20.0+ba45583
user1-jcnp6-worker-c-pwxfp.c.quay-devel.internal   Ready    worker   3h21m   v1.20.0+ba45583
user1-jcnp6-worker-d-h5tv2.c.quay-devel.internal   Ready    worker   3h22m   v1.20.0+ba45583
user1-jcnp6-worker-d-m9gg4.c.quay-devel.internal   Ready    worker   3h21m   v1.20.0+ba45583
```

Label the final three worker nodes for infrastructure use:

```
$ oc label node --overwrite user1-jcnp6-worker-c-pwxfp.c.quay-devel.internal node-
role.kubernetes.io/infra=
$ oc label node --overwrite user1-jcnp6-worker-d-h5tv2.c.quay-devel.internal node-
role.kubernetes.io/infra=
$ oc label node --overwrite user1-jcnp6-worker-d-m9gg4.c.quay-devel.internal node-
role.kubernetes.io/infra=
```

Now, when you list the nodes in the cluster, the last 3 worker nodes will have an added role of **infra**:

```
$ oc get nodes
NAME                                          STATUS   ROLES          AGE    VERSION
user1-jcnp6-master-0.c.quay-devel.internal         Ready    master         4h14m  v1.20.0+ba45583
user1-jcnp6-master-1.c.quay-devel.internal         Ready    master         4h15m  v1.20.0+ba45583
user1-jcnp6-master-2.c.quay-devel.internal         Ready    master         4h14m  v1.20.0+ba45583
user1-jcnp6-worker-b-65plj.c.quay-devel.internal   Ready    worker         4h6m   v1.20.0+ba45583
user1-jcnp6-worker-b-jr7hc.c.quay-devel.internal   Ready    worker         4h5m   v1.20.0+ba45583
user1-jcnp6-worker-c-jrq4v.c.quay-devel.internal   Ready    worker         4h5m   v1.20.0+ba45583
user1-jcnp6-worker-c-pwxfp.c.quay-devel.internal   Ready    infra,worker   4h6m   v1.20.0+ba45583
user1-jcnp6-worker-d-h5tv2.c.quay-devel.internal   Ready    infra,worker   4h6m   v1.20.0+ba45583
user1-jcnp6-worker-d-m9gg4.c.quay-devel.internal   Ready    infra,worker   4h6m   v1.20.0+ba45583
```

With an infra node being assigned as a worker, there is a chance that user workloads could get inadvertently assigned to an infra node. To avoid this, you can apply a taint to the infra node and then add tolerations for the pods you want to control.

```
$ oc adm taint nodes user1-jcnp6-worker-c-pwxfp.c.quay-devel.internal node-role.kubernetes.io/infra:NoSchedule
$ oc adm taint nodes user1-jcnp6-worker-d-h5tv2.c.quay-devel.internal node-role.kubernetes.io/infra:NoSchedule
$ oc adm taint nodes user1-jcnp6-worker-d-m9gg4.c.quay-devel.internal node-role.kubernetes.io/infra:NoSchedule
```

8.1.2. Create a Project with node selector and toleration

If you have already deployed Quay using the Quay Operator, remove the installed operator and any specific namespace(s) you created for the deployment.

Create a Project resource, specifying a node selector and toleration as shown in the following example:

quay-registry.yaml

```
kind: Project
apiVersion: project.openshift.io/v1
metadata:
  name: quay-registry
  annotations:
    openshift.io/node-selector: 'node-role.kubernetes.io/infra='
    scheduler.alpha.kubernetes.io/defaultTolerations: >-
      [{"operator": "Exists", "effect": "NoSchedule", "key":
      "node-role.kubernetes.io/infra"}
      ]
```

Use the **oc apply** command to create the project:

```
$ oc apply -f quay-registry.yaml
project.project.openshift.io/quay-registry created
```

Any subsequent resources created in the **quay-registry** namespace should now be scheduled on the dedicated infrastructure nodes.

8.1.3. Install the Quay Operator in the namespace

When installing the Quay Operator, specify the appropriate project namespace explicitly, in this case **quay-registry**. This will result in the operator pod itself landing on one of the three infrastructure nodes:

```
$ oc get pods -n quay-registry -o wide
NAME                              READY   STATUS    RESTARTS   AGE   IP            NODE

quay-operator.v3.4.1-6f6597d8d8-bd4dp   1/1     Running   0          30s   10.131.0.16   user1-jcnp6-worker-d-h5tv2.c.quay-devel.internal
```

8.1.4. Create the registry

Create the registry as explained earlier, and then wait for the deployment to be ready. When you list the Quay pods, you should now see that they have only been scheduled on the three nodes that you have labelled for infrastructure purposes:

```
$ oc get pods -n quay-registry -o wide
NAME                                                  READY   STATUS      RESTARTS   AGE     IP          NODE

example-registry-clair-app-789d6d984d-gpbwd          1/1     Running     1          5m57s   10.130.2.80
user1-jcnp6-worker-d-m9gg4.c.quay-devel.internal
example-registry-clair-postgres-7c8697f5-zkzht       1/1     Running     0          4m53s   10.129.2.19
user1-jcnp6-worker-c-pwxfp.c.quay-devel.internal
example-registry-quay-app-56dd755b6d-glbf7           1/1     Running     1          5m57s   10.129.2.17
user1-jcnp6-worker-c-pwxfp.c.quay-devel.internal
example-registry-quay-config-editor-7bf9bccc7b-dpc6d   1/1     Running     0          5m57s
10.131.0.23   user1-jcnp6-worker-d-h5tv2.c.quay-devel.internal
example-registry-quay-database-8dc7cfd69-dr2cc       1/1     Running     0          5m43s   10.129.2.18
  user1-jcnp6-worker-c-pwxfp.c.quay-devel.internal
example-registry-quay-mirror-78df886bcc-v75p9        1/1     Running     0          5m16s   10.131.0.24
user1-jcnp6-worker-d-h5tv2.c.quay-devel.internal
example-registry-quay-postgres-init-8s8g9            0/1     Completed   0          5m54s   10.130.2.79
user1-jcnp6-worker-d-m9gg4.c.quay-devel.internal
example-registry-quay-redis-5688ddcdb6-ndp4t         1/1     Running     0          5m56s   10.130.2.78
user1-jcnp6-worker-d-m9gg4.c.quay-devel.internal
quay-operator.v3.4.1-6f6597d8d8-bd4dp                1/1     Running     0          22m     10.131.0.16
user1-jcnp6-worker-d-h5tv2.c.quay-devel.internal
```

8.2. ENABLING MONITORING WHEN OPERATOR IS INSTALLED IN A SINGLE NAMESPACE

When Red Hat Quay Operator is installed in a single namespace, the monitoring component is unmanaged. To configure monitoring, you need to enable it for user-defined namespaces in OpenShift Container Platform. For more information, see the OCP documentation for Configuring the monitoring stack and Enabling monitoring for user-defined projects.

The following steps show you how to configure monitoring for Quay, based on the OCP documentation.

8.2.1. Creating a cluster monitoring config map

1. Check whether the **cluster-monitoring-config** ConfigMap object exists:

```
$ oc -n openshift-monitoring get configmap cluster-monitoring-config

Error from server (NotFound): configmaps "cluster-monitoring-config" not found
```

2. If the ConfigMap object does not exist:

   a. Create the following YAML manifest. In this example, the file is called **cluster-monitoring-config.yaml**:

```
$ cat cluster-monitoring-config.yaml

apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
```

b. Create the ConfigMap object:

```
$ oc apply -f cluster-monitoring-config.yaml configmap/cluster-monitoring-config created
```

```
$ oc -n openshift-monitoring get configmap cluster-monitoring-config

NAME                        DATA   AGE
cluster-monitoring-config   1      12s
```

8.2.2. Creating a user-defined workload monitoring config map

1. Check whether the **user-workload-monitoring-config** ConfigMap object exists:

```
$ oc -n openshift-user-workload-monitoring get configmap user-workload-monitoring-config

Error from server (NotFound): configmaps "user-workload-monitoring-config" not found
```

2. If the ConfigMap object does not exist:

   a. Create the following YAML manifest. In this example, the file is called **user-workload-monitoring-config.yaml**:

```
$ cat user-workload-monitoring-config.yaml

apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
```

   b. Create the ConfigMap object:

```
$ oc apply -f user-workload-monitoring-config.yaml

configmap/user-workload-monitoring-config created
```

8.2.3. Enable monitoring for user-defined projects

1. Check whether monitoring for user-defined projects is running:

```
$ oc get pods -n openshift-user-workload-monitoring

No resources found in openshift-user-workload-monitoring namespace.
```

2. Edit the **cluster-monitoring-config** ConfigMap:

```
$ oc -n openshift-monitoring edit configmap cluster-monitoring-config
```

3. Set **enableUserWorkload: true** to enable monitoring for user-defined projects on the cluster:

```
apiVersion: v1
data:
  config.yaml: |
    enableUserWorkload: true
kind: ConfigMap
metadata:
  annotations:
```

4. Save the file to apply the changes and then check that the appropriate pods are running:

```
$ oc get pods -n openshift-user-workload-monitoring

NAME                                   READY   STATUS    RESTARTS   AGE
prometheus-operator-6f96b4b8f8-gq6rl   2/2     Running   0          15s
prometheus-user-workload-0             5/5     Running   1          12s
prometheus-user-workload-1             5/5     Running   1          12s
thanos-ruler-user-workload-0           3/3     Running   0          8s
thanos-ruler-user-workload-1           3/3     Running   0          8s
```

8.2.4. Create a Service object to expose Quay metrics

1. Create a YAML file for the Service object:

```
$ cat quay-service.yaml

apiVersion: v1
kind: Service
metadata:
  annotations:
  labels:
    quay-component: monitoring
    quay-operator/quayregistry: example-registry
  name: example-registry-quay-metrics
  namespace: quay-enterprise
spec:
  ports:
  - name: quay-metrics
    port: 9091
    protocol: TCP
    targetPort: 9091
  selector:
    quay-component: quay-app
    quay-operator/quayregistry: example-registry
  type: ClusterIP
```

2. Create the Service object:

```
$  oc apply -f quay-service.yaml

service/example-registry-quay-metrics created
```

8.2.5. Create a ServiceMonitor object

Configure OpenShift Monitoring to scrape the metrics by creating a ServiceMonitor resource.

1. Create a YAML file for the ServiceMonitor resource:

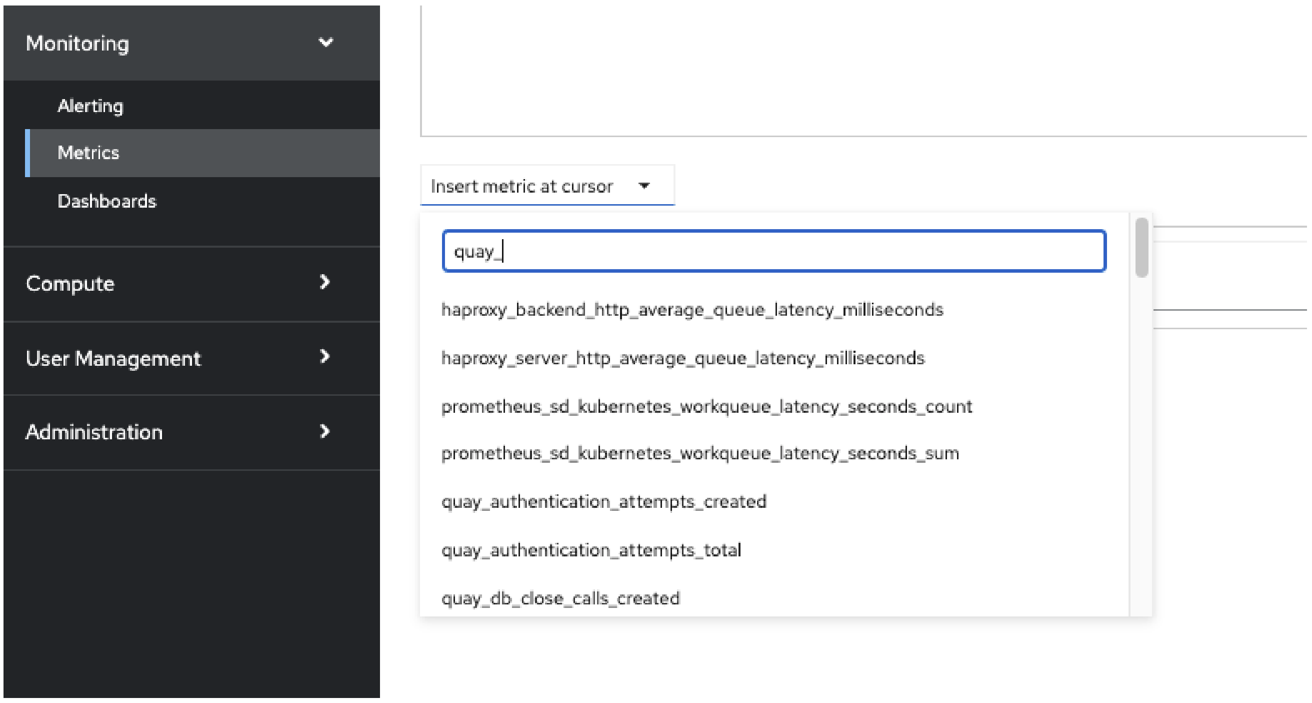
```
$ cat quay-service-monitor.yaml

apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  labels:
    quay-operator/quayregistry: example-registry
  name: example-registry-quay-metrics-monitor
  namespace: quay-enterprise
spec:
  endpoints:
  - port: quay-metrics
  namespaceSelector:
    any: true
  selector:
    matchLabels:
      quay-component: monitoring
```

2. Create the ServiceMonitor:

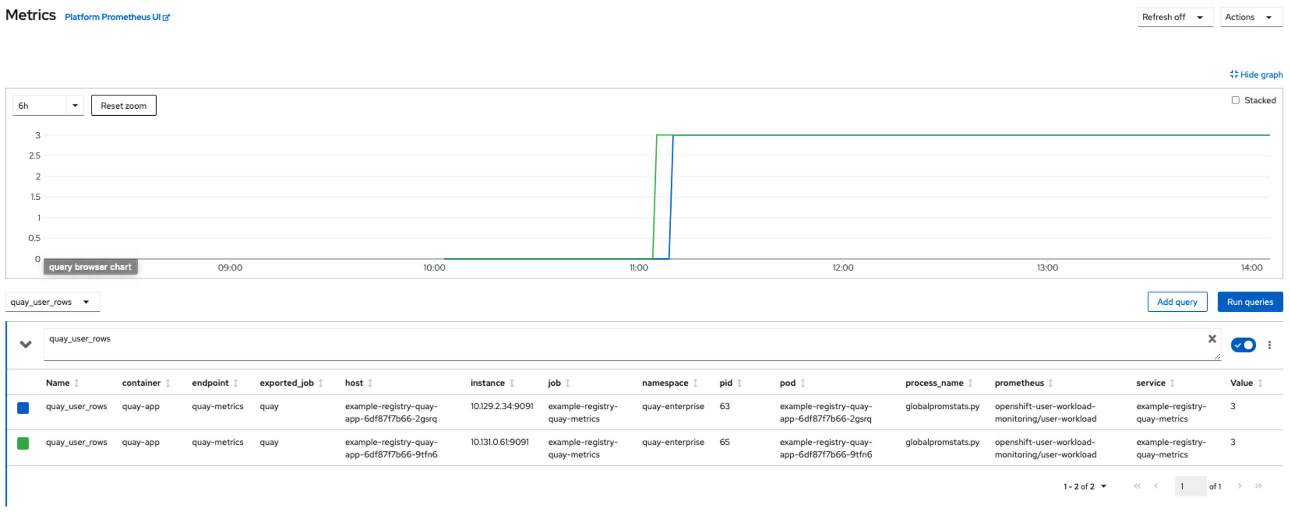
```
$  oc apply -f quay-service-monitor.yaml

servicemonitor.monitoring.coreos.com/example-registry-quay-metrics-monitor created
```

8.2.6. View the metrics in OpenShift

You can access the metrics in the OpenShift console under Monitoring → Metrics. In the Expression field, enter the text **quay_** to see the list of metrics available:


Monitoring
Alerting
Metrics
Dashboards
Compute
User Management
Administration
Insert metric at cursor
quay_
haproxy_backend_http_average_queue_latency_milliseconds
haproxy_server_http_average_queue_latency_milliseconds
prometheus_sd_kubernetes_workqueue_latency_seconds_count
prometheus_sd_kubernetes_workqueue_latency_seconds_sum
quay_authentication_attempts_created
quay_authentication_attempts_total
quay_db_close_calls_created

For example, if you have added users to your registry, select the **quay-users_rows** metric:



quay_user_rows

	Name	container	endpoint	exported_job	host	instance	job	namespace	pid	pod	process_name	prometheus	service	Value
🟦	quay_user_rows	quay-app	quay-metrics	quay	example-registry-quay-app-6df87f7b66-2gsrq	10.129.2.34:9091	example-registry-quay-metrics	quay-enterprise	63	example-registry-quay-app-6df87f7b66-2gsrq	globalpromstats.py	openshift-user-workload-monitoring/user-workload	example-registry-quay-metrics	3
🟩	quay_user_rows	quay-app	quay-metrics	quay	example-registry-quay-app-6df87f7b66-9tfn6	10.131.0.61:9091	example-registry-quay-metrics	quay-enterprise	65	example-registry-quay-app-6df87f7b66-9tfn6	globalpromstats.py	openshift-user-workload-monitoring/user-workload	example-registry-quay-metrics	3

1 - 2 of 2 1 of 1

8.3. RESIZING MANAGED STORAGE

The Quay Operator creates default object storage using the defaults provided by RHOCS when creating a **NooBaa** object (50 Gib). There are two ways to extend this storage; you can resize an existing PVC or add more PVCs to a new storage pool.

8.3.1. Resize Noobaa PVC

1. Log into the OpenShift console and select **Storage** → **Persistent Volume Claims**.
2. Select the **PersistentVolumeClaim** named like **noobaa-default-backing-store-noobaa-pvc-***.
3. From the Action menu, select **Expand PVC**.
4. Enter the new size of the Persistent Volume Claim and select **Expand**.

After a few minutes (depending on the size of the PVC), the expanded size should reflect in the PVC's **Capacity** field.

NOTE

Expanding CSI volumes is a Technology Preview feature only. For more information, see https://access.redhat.com/documentation/en-us/openshift_container_platform/4.6/html/storage/expanding-persistent-volumes.

8.3.2. Add Another Storage Pool

1. Log into the OpenShift console and select **Networking → Routes**. Make sure the **openshift-storage** project is selected.
2. Click on the **Location** field for the **noobaa-mgmt** Route.
3. Log into the Noobaa Management Console.
4. On the main dashboard, under **Storage Resources**, select **Add Storage Resources**.
5. Select **Deploy Kubernetes Pool**
6. Enter a new pool name. Click **Next**.
7. Choose the number of Pods to manage the pool and set the size per node. Click **Next**.
8. Click **Deploy**.

After a few minutes, the additional storage pool will be added to the Noobaa resources and available for use by Red Hat Quay.

8.4. CUSTOMIZING DEFAULT OPERATOR IMAGES

NOTE

Using this mechanism is not supported for production Quay environments and is strongly encouraged only for development/testing purposes. There is no guarantee your deployment will work correctly when using non-default images with the Quay Operator.

In certain circumstances, it may be useful to override the default images used by the Operator. This can be done by setting one or more environment variables in the Quay Operator **ClusterServiceVersion**.

8.4.1. Environment Variables

The following environment variables are used in the Operator to override component images:

Environment Variable	Component
RELATED_IMAGE_COMPONENT_QUAY	**base**
RELATED_IMAGE_COMPONENT_CLAIR	**clair**

RELATED_IMAGE_COMPONENT_POSTGRES	**postgres** and **clair** databases
RELATED_IMAGE_COMPONENT_REDIS	**redis**

NOTE

Override images **must** be referenced by manifest (@sha256:), not by tag (:latest).

8.4.2. Applying Overrides to a Running Operator

When the Quay Operator is installed in a cluster via the Operator Lifecycle Manager (OLM), the managed component container images can be easily overridden by modifying the **ClusterServiceVersion** object, which is OLM's representation of a running Operator in the cluster. Find the Quay Operator's **ClusterServiceVersion** either by using a Kubernetes UI or **kubectl**/**oc**:

```
$ oc get clusterserviceversions -n <your-namespace>
```

Using the UI, **oc edit**, or any other method, modify the Quay **ClusterServiceVersion** to include the environment variables outlined above to point to the override images:

JSONPath: **spec.install.spec.deployments[0].spec.template.spec.containers[0].env**

```
- name: RELATED_IMAGE_COMPONENT_QUAY
  value:
quay.io/projectquay/quay@sha256:c35f5af964431673f4ff5c9e90bdf45f19e38b8742b5903d41c10cc7f63
39a6d
- name: RELATED_IMAGE_COMPONENT_CLAIR
  value:
quay.io/projectquay/clair@sha256:70c99feceb4c0973540d22e740659cd8d616775d3ad1c1698ddf71d
0221f3ce6
- name: RELATED_IMAGE_COMPONENT_POSTGRES
  value: centos/postgresql-10-
centos7@sha256:de1560cb35e5ec643e7b3a772ebaac8e3a7a2a8e8271d9e91ff023539b4dfb33
- name: RELATED_IMAGE_COMPONENT_REDIS
  value: centos/redis-32-
centos7@sha256:06dbb609484330ec6be6090109f1fa16e936afcf975d1cbc5fff3e6c7cae7542
```

Note that this is done at the Operator level, so every QuayRegistry will be deployed using these same overrides.

8.5. AWS S3 CLOUDFRONT

If you use AWS S3 CloudFront for backend registry storage, specify the private key as shown in the following example:

```
$ oc create secret generic --from-file config.yaml=./config_awss3cloudfront.yaml --from-file default-
cloudfront-signing-key.pem=./default-cloudfront-signing-key.pem test-config-bundle
```

8.5.1. Advanced Clair configuration

8.5.1.1. Unmanaged Clair configuration

With Red Hat Quay 3.7, users can run an unmanaged Clair configuration on the Red Hat Quay OpenShift Container Platform Operator. This feature allows users to create an unmanaged Clair database, or run their custom Clair configuration without an unmanaged database.

8.5.1.1.1. Unmanaging a Clair database

An unmanaged Clair database allows the Red Hat Quay Operator to work in a geo-replicated environment, where multiple instances of the Operator must communicate with the same database. An unmanaged Clair database can also be used when a user requires a highly-available (HA) Clair database that exists outside of a cluster.

Procedure

- In the Quay Operator, set the **clairpostgres** component of the QuayRegistry custom resource to unmanaged:

```
apiVersion: quay.redhat.com/v1
kind: QuayRegistry
metadata:
  name: quay370
spec:
  configBundleSecret: config-bundle-secret
  components:
    - kind: objectstorage
      managed: false
    - kind: route
      managed: true
    - kind: tls
      managed: false
    - kind: clairpostgres
      managed: false
```

8.5.1.1.2. Configuring a custom Clair database

The Red Hat Quay Operator for OpenShift Container Platform allows users to provide their own Clair configuration by editing the **configBundleSecret** parameter.

Procedure

1. Create a Quay config bundle secret that includes the **clair-config.yaml**:

```
$ oc create secret generic --from-file config.yaml=./config.yaml --from-file extra_ca_cert_rds-ca-2019-root.pem=./rds-ca-2019-root.pem --from-file clair-config.yaml=./clair-config.yaml --from-file ssl.cert=./ssl.cert --from-file ssl.key=./ssl.key config-bundle-secret
```

Example **clair-config.yaml** configuration:

```
indexer:
    connstring: host=quay-server.example.com port=5432 dbname=quay user=quayrdsdb
password=quayrdsdb sslrootcert=/run/certs/rds-ca-2019-root.pem sslmode=verify-ca
```

```
    layer_scan_concurrency: 6
    migrations: true
    scanlock_retry: 11
log_level: debug
matcher:
    connstring: host=quay-server.example.com port=5432 dbname=quay user=quayrdsdb
password=quayrdsdb sslrootcert=/run/certs/rds-ca-2019-root.pem sslmode=verify-ca
    migrations: true
metrics:
    name: prometheus
notifier:
    connstring: host=quay-server.example.com port=5432 dbname=quay user=quayrdsdb
password=quayrdsdb sslrootcert=/run/certs/rds-ca-2019-root.pem sslmode=verify-ca
    migrations: true
```

NOTE

- The database certificate is mounted under **/run/certs/rds-ca-2019-root.pem** on the Clair application pod in the **clair-config.yaml**. It must be specified when configuring your **clair-config.yaml**.
- An example **clair-config.yaml** can be found at Clair on OpenShift config.

2. Add the **clair-config.yaml** to your bundle secret, named **configBundleSecret**:

```
apiVersion: v1
kind: Secret
metadata:
  name: config-bundle-secret
  namespace: quay-enterprise
data:
  config.yaml: <base64 encoded Quay config>
  clair-config.yaml: <base64 encoded Clair config>
  extra_ca_cert_<name>: <base64 encoded ca cert>
  clair-ssl.crt: >-
  clair-ssl.key: >-
```

NOTE

When updated, the provided **clair-config.yaml** is mounted into the Clair pod. Any fields not provided are automatically populated with defaults using the Clair configuration module.

After proper configuration, the Clair application pod should return to a **Ready** state.

8.5.1.2. Running a custom Clair configuration with a managed database

In some cases, users might want to run a custom Clair configuration with a **managed** database. This is useful in the following scenarios:

- When a user wants to disable an updater.
- When a user is running in an air-gapped environment.

NOTE

- If you are running Quay in an air-gapped environment, the **airgap** parameter of your **clair-config.yaml** must be set to **true**.
- If you are running Quay in an air-gapped environment, you should disable all updaters.

Use the steps in "Configuring a custom Clair database" to configure your database when **clairpostgres** is set to **managed**.

For more information about running Clair in an air-gapped environment, see Configuring access to the Clair database in the air-gapped OpenShift cluster.

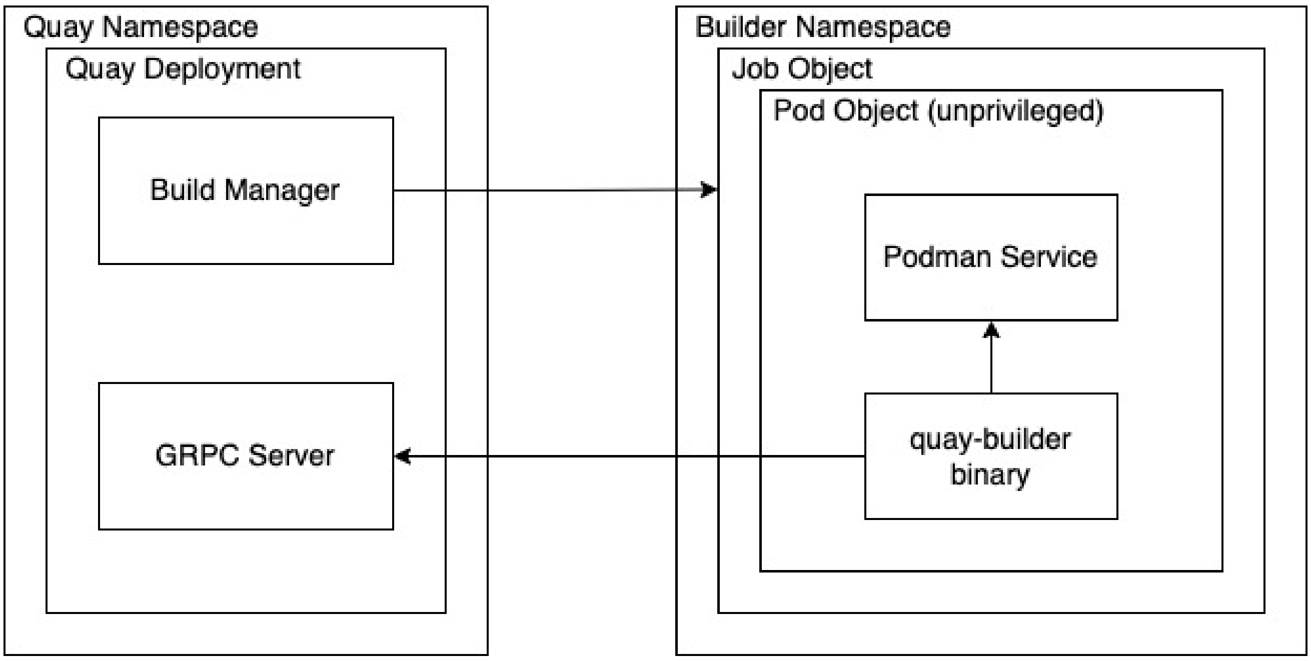
CHAPTER 9. RED HAT QUAY BUILD ENHANCEMENTS

Prior to Red Hat Quay 3.7, Quay ran **podman** commands in virtual machines launched by pods. Running builds on virtual platforms requires enabling nested virtualization, which is not featured in Red Hat Enterprise Linux or OpenShift Container Platform. As a result, builds had to run on bare-metal clusters, which is an inefficient use of resources.

With Red Hat Quay 3.7., the bare-metal constraint required to run builds has been removed by adding an additional build option which does not contain the virtual machine layer. As a result, builds can be run on virtualized platforms. Backwards compatibility to run previous build configurations are also available.

9.1. RED HAT QUAY ENHANCED BUILD ARCHITECTURE

The preceding image shows the expected design flow and architecture of the enhanced build features:



With this enhancement, the build manager first creates the **Job Object**. Then, the **Job Object** then creates a pod using the **quay-builder-image**. The **quay-builder-image** will contain the **quay-builder binary** and the **Podman** service. The created pod runs as **unprivileged**. The **quay-builder binary** then builds the image while communicating status and retrieving build information from the Build Manager.

9.2. RED HAT QUAY BUILD LIMITATIONS

Running builds in Red Hat Quay in an unprivileged context might cause some commands that were working under the previous build strategy to fail. Attempts to change the build strategy could potentially cause performance issues and reliability with the build.

Running builds directly in a container will not have the same isolation as using virtual machines. Changing the build environment might also caused builds that were previously working to fail.

9.3. CREATING A RED HAT QUAY BUILDERS ENVIRONMENT WITH OPENSHIFT CONTAINER PLATFORM

The procedures in this section explain how to create a Red Hat Quay virtual builders environment with OpenShift Container Platform.

9.3.1. OpenShift Container Platform TLS component

The **tls** component allows you to control TLS configuration.

NOTE

Red Hat Quay 3 does not support builders when the TLS component is managed by the Operator.

If you set **tls** to **unmanaged**, you supply your own **ssl.cert** and **ssl.key** files. In this instance, if you want your cluster to support builders, you must add both the Quay route and the builder route name to the SAN list in the cert, or use a wildcard.

To add the builder route, use the following format:

```
[quayregistry-cr-name]-quay-builder-[ocp-namespace].[ocp-domain-name]:443
```

9.3.2. Using OpenShift Container Platform for Red Hat Quay builders

Builders require SSL/TLS certificates. For more information about SSL/TLS certificates, see Adding TLS certificates to the Red Hat Quay container.

If you are using Amazon Web Service (AWS) S3 storage, you must modify your storage bucket in the AWS console, prior to running builders. See "Modifying your AWS S3 storage bucket" in the following section for the required parameters.

9.3.2.1. Preparing OpenShift Container Platform for virtual builders

Use the following procedure to prepare OpenShift Container Platform for Red Hat Quay virtual builders.

NOTE

- This procedure assumes you already have a cluster provisioned and a Quay Operator running.
- This procedure is for setting up a virtual namespace on OpenShift Container Platform.

Procedure

1. Log in to your Red Hat Quay cluster using a cluster administrator account.
2. Create a new project where your virtual builders will be run, for example, **virtual-builders**, by running the following command:

   ```
   $ oc new-project virtual-builders
   ```

3. Create a **ServiceAccount** in the project that will be used to run builds by entering the following command:

   ```
   $ oc create sa -n virtual-builders quay-builder
   ```

4. Provide the created service account with editing permissions so that it can run the build:

```
$ oc adm policy -n virtual-builders add-role-to-user edit system:serviceaccount:virtual-builders:quay-builder
```

5. Grant the Quay builder **anyuid scc** permissions by entering the following command:

```
$ oc adm policy -n virtual-builders add-scc-to-user anyuid -z quay-builder
```

> **NOTE**
>
> This action requires cluster admin privileges. This is required because builders must run as the Podman user for unprivileged or rootless builds to work.

6. Obtain the token for the Quay builder service account.

   a. If using OpenShift Container Platform 4.10 or an earlier version, enter the following command:

   ```
   oc sa get-token -n virtual-builders quay-builder
   ```

   b. If using OpenShift Container Platform 4.11 or later, enter the following command:

   ```
   $ oc create token quay-builder -n virtual-builders
   ```

   Example output

   ```
   eyJhbGciOiJSUzI1NiIsImtpZCI6IldfQUJkaDVmb3ltTHZ0dGZMYjhIWnYxZTQzN2dJVEJxcDJscldSdEUtYWsifQ...
   ```

7. Determine the builder route by entering the following command:

```
$ oc get route -n quay-enterprise
```

Example output

```
NAME                          HOST/PORT                                                          PATH   SERVICES                  PORT   TERMINATION     WILDCARD
...
example-registry-quay-builder   example-registry-quay-builder-quay-enterprise.apps.docs.quayteam.org          example-registry-quay-app    grpc   edge/Redirect   None
...
```

8. Generate a self-signed SSL/TlS certificate with the .crt extension by entering the following command:

```
$ oc extract cm/kube-root-ca.crt -n openshift-apiserver
```

Example output

```
ca.crt
```

9. Rename the **ca.crt** file to **extra_ca_cert_build_cluster.crt** by entering the following command:

```
$ mv ca.crt extra_ca_cert_build_cluster.crt
```

10. Locate the secret for you configuration bundle in the **Console**, and select **Actions → Edit Secret** and add the appropriate builder configuration:

```
FEATURE_USER_INITIALIZE: true
BROWSER_API_CALLS_XHR_ONLY: false
SUPER_USERS:
- <superusername>
FEATURE_USER_CREATION: false
FEATURE_QUOTA_MANAGEMENT: true
FEATURE_BUILD_SUPPORT: True
BUILDMAN_HOSTNAME: <sample_build_route> (1)
BUILD_MANAGER:
  - ephemeral
  - ALLOWED_WORKER_COUNT: 1
    ORCHESTRATOR_PREFIX: buildman/production/
    JOB_REGISTRATION_TIMEOUT: 3600 (2)
    ORCHESTRATOR:
      REDIS_HOST: <sample_redis_hostname> (3)
      REDIS_PASSWORD: ""
      REDIS_SSL: false
      REDIS_SKIP_KEYSPACE_EVENT_SETUP: false
    EXECUTORS:
      - EXECUTOR: kubernetesPodman
        NAME: openshift
        BUILDER_NAMESPACE: <sample_builder_namespace> (4)
        SETUP_TIME: 180
        MINIMUM_RETRY_THRESHOLD:
        BUILDER_CONTAINER_IMAGE: <sample_builder_container_image> (5)
        # Kubernetes resource options
        K8S_API_SERVER: <sample_k8s_api_server> (6)
        K8S_API_TLS_CA: <sample_crt_file> (7)
        VOLUME_SIZE: 8G
        KUBERNETES_DISTRIBUTION: openshift
        CONTAINER_MEMORY_LIMITS: 300m (8)
        CONTAINER_CPU_LIMITS: 1G (9)
        CONTAINER_MEMORY_REQUEST: 300m (10)
        CONTAINER_CPU_REQUEST: 1G (11)
        NODE_SELECTOR_LABEL_KEY: ""
        NODE_SELECTOR_LABEL_VALUE: ""
        SERVICE_ACCOUNT_NAME: <sample_service_account_name>
        SERVICE_ACCOUNT_TOKEN: <sample_account_token> (12)
```

(1) The build route is obtained by running **oc get route -n** with the name of your OpenShift Operator's namespace. A port must be provided at the end of the route, and it should use the following format: **[quayregistry-cr-name]-quay-builder-[ocp-namespace].[ocp-domain-name]:443**.

(2) If the **JOB_REGISTRATION_TIMEOUT** parameter is set too low, you might receive the following error: **failed to register job to build manager: rpc error: code = Unauthenticated desc = Invalid build token: Signature has expired**. It is suggested that

this parameter be set to at least 240.

3 If your Redis host has a password or SSL/TLS certificates, you must update accordingly.

4 Set to match the name of your virtual builders namespace, for example, **virtual-builders**.

5 For early access, the **BUILDER_CONTAINER_IMAGE** is currently **quay.io/projectquay/quay-builder:3.7.0-rc.2**. Note that this might change during the early access window. If this happens, customers are alerted.

6 The **K8S_API_SERVER** is obtained by running **oc cluster-info**.

7 You must manually create and add your custom CA cert, for example, **K8S_API_TLS_CA: /conf/stack/extra_ca_certs/build_cluster.crt**.

8 Defaults to **5120Mi** if left unspecified.

9 For virtual builds, you must ensure that there are enough resources in your cluster. Defaults to **1000m** if left unspecified.

10 Defaults to **3968Mi** if left unspecified.

11 Defaults to **500m** if left unspecified.

12 Obtained when running **oc create sa**.

Sample configuration

```
FEATURE_USER_INITIALIZE: true
BROWSER_API_CALLS_XHR_ONLY: false
SUPER_USERS:
- quayadmin
FEATURE_USER_CREATION: false
FEATURE_QUOTA_MANAGEMENT: true
FEATURE_BUILD_SUPPORT: True
BUILDMAN_HOSTNAME: example-registry-quay-builder-quay-
enterprise.apps.docs.quayteam.org:443
BUILD_MANAGER:
  - ephemeral
  - ALLOWED_WORKER_COUNT: 1
    ORCHESTRATOR_PREFIX: buildman/production/
    JOB_REGISTRATION_TIMEOUT: 3600
    ORCHESTRATOR:
      REDIS_HOST: example-registry-quay-redis
      REDIS_PASSWORD: ""
      REDIS_SSL: false
      REDIS_SKIP_KEYSPACE_EVENT_SETUP: false
    EXECUTORS:
      - EXECUTOR: kubernetesPodman
        NAME: openshift
        BUILDER_NAMESPACE: virtual-builders
        SETUP_TIME: 180
        MINIMUM_RETRY_THRESHOLD:
        BUILDER_CONTAINER_IMAGE: quay.io/projectquay/quay-builder:3.7.0-rc.2
        # Kubernetes resource options
        K8S_API_SERVER: api.docs.quayteam.org:6443
        K8S_API_TLS_CA: /conf/stack/extra_ca_certs/build_cluster.crt
        VOLUME_SIZE: 8G
        KUBERNETES_DISTRIBUTION: openshift
        CONTAINER_MEMORY_LIMITS: 1G
        CONTAINER_CPU_LIMITS: 1080m
        CONTAINER_MEMORY_REQUEST: 1G
        CONTAINER_CPU_REQUEST: 580m
        NODE_SELECTOR_LABEL_KEY: ""
        NODE_SELECTOR_LABEL_VALUE: ""
        SERVICE_ACCOUNT_NAME: quay-builder
        SERVICE_ACCOUNT_TOKEN:
"eyJhbGciOiJSUzI1NiIsImtpZCI6IldfQUJkaDVmb3ltTHZ0dGZMYjhIWnYxZTQzN2dJVEJxcDJs
cldSdEUtYWsifQ"
```

9.3.2.2. Manually adding SSL/TLS certificates

Due to a known issue with the configuration tool, you must manually add your custom SSL/TLS certificates to properly run builders. Use the following procedure to manually add custom SSL/TLS certificates.

For more information creating SSL/TLS certificates, see Adding TLS certificates to the Red Hat Quay container.

9.3.2.2.1. Creating and signing certificates

Use the following procedure to create and sign an SSL/TLS certificate.

Procedure

- Create a certificate authority and sign a certificate. For more information, see Create a Certificate Authority and sign a certificate.

  openssl.cnf

  ```
  [req]
  req_extensions = v3_req
  distinguished_name = req_distinguished_name
  [req_distinguished_name]
  [ v3_req ]
  basicConstraints = CA:FALSE
  keyUsage = nonRepudiation, digitalSignature, keyEncipherment
  subjectAltName = @alt_names
  [alt_names]
  DNS.1 = example-registry-quay-quay-enterprise.apps.docs.quayteam.org (1)
  DNS.2 = example-registry-quay-builder-quay-enterprise.apps.docs.quayteam.org (2)
  ```

  (1) An **alt_name** for the URL of your Red Hat Quay registry must be included.

  (2) An **alt_name** for the **BUILDMAN_HOSTNAME**

  Sample commands

  ```
  $ openssl genrsa -out rootCA.key 2048
  ```

```
$ openssl req -x509 -new -nodes -key rootCA.key -sha256 -days 1024 -out rootCA.pem
$ openssl genrsa -out ssl.key 2048
$ openssl req -new -key ssl.key -out ssl.csr
$ openssl x509 -req -in ssl.csr -CA rootCA.pem -CAkey rootCA.key -CAcreateserial -out
ssl.cert -days 356 -extensions v3_req -extfile openssl.cnf
```

9.3.2.2.2. Setting TLS to unmanaged

Use the following procedure to set **king:tls** to unmanaged.

Procedure

1. In your Red Hat Quay Registry YAML, set **kind: tls** to **managed: false**:

```
  - kind: tls
    managed: false
```

2. On the **Events** page, the change is blocked until you set up the appropriate **config.yaml** file. For example:

```
    - lastTransitionTime: '2022-03-28T12:56:49Z'
      lastUpdateTime: '2022-03-28T12:56:49Z'
      message: >-
        required component `tls` marked as unmanaged, but `configBundleSecret`
        is missing necessary fields
      reason: ConfigInvalid
      status: 'True'
```

9.3.2.2.3. Creating temporary secrets

Use the following procedure to create temporary secrets for the CA certificate.

Procedure

1. Create a secret in your default namespace for the CA certificate:

```
$ oc create secret generic -n quay-enterprise temp-crt --from-file
extra_ca_cert_build_cluster.crt
```

2. Create a secret in your default namespace for the **ssl.key** and **ssl.cert** files:

```
$ oc create secret generic -n quay-enterprise quay-config-ssl --from-file ssl.cert --from-file
ssl.key
```

9.3.2.2.4. Copying secret data to the configuration YAML

Use the following procedure to copy secret data to your **config.yaml** file.

Procedure

1. Locate the new secrets in the console UI at **Workloads → Secrets**.

2. For each secret, locate the YAML view:

```
kind: Secret
apiVersion: v1
metadata:
  name: temp-crt
  namespace: quay-enterprise
  uid: a4818adb-8e21-443a-a8db-f334ace9f6d0
  resourceVersion: '9087855'
  creationTimestamp: '2022-03-28T13:05:30Z'
...
data:
  extra_ca_cert_build_cluster.crt: >-
    LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSURNakNDQWhxZ0F3SUJBZ0l....
type: Opaque
```

```
kind: Secret
apiVersion: v1
metadata:
  name: quay-config-ssl
  namespace: quay-enterprise
  uid: 4f5ae352-17d8-4e2d-89a2-143a3280783c
  resourceVersion: '9090567'
  creationTimestamp: '2022-03-28T13:10:34Z'
...
data:
  ssl.cert: >-
    LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUVaakNDQTA2Z0F3SUJBZ0lVT...
  ssl.key: >-
    LS0tLS1CRUdJTiBSU0EgUFJJVkFURSBLRVktLS0tLQpNSUlFcFFJQkFBS0NBUUVBc...
type: Opaque
```

3. Locate the secret for your Red Hat Quay registry configuration bundle in the UI, or through the command line by running a command like the following:

```
$ oc get quayregistries.quay.redhat.com -o jsonpath="{.items[0].spec.configBundleSecret}
{'\n'}"  -n quay-enterprise
```

4. In the OpenShift Container Platform console, select the YAML tab for your configuration bundle secret, and add the data from the two secrets you created:

```
kind: Secret
apiVersion: v1
metadata:
  name: init-config-bundle-secret
  namespace: quay-enterprise
  uid: 4724aca5-bff0-406a-9162-ccb1972a27c1
  resourceVersion: '4383160'
  creationTimestamp: '2022-03-22T12:35:59Z'
...
data:
  config.yaml: >-
    RkVBVFVSRV9VU0VSX0lOSVRJQUxJWkU6IHRydWUKQlJ...
  extra_ca_cert_build_cluster.crt: >-

LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSURNakNDQWhxZ0F3SUJBZ0ldw....
```

```
  ssl.cert: >-
    LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUVaakNDQTA2Z0F3SUJBZ0lVT...
  ssl.key: >-
    LS0tLS1CRUdJTiBSU0EgUFJJVkFURSBLRVktLS0tLQpNSUlFcFFJQkFBS0NBUUVBc...
type: Opaque
```

5. Click **Save**.

6. Enter the following command to see if your pods are restarting:

```
$ oc get pods -n quay-enterprise
```

Example output

```
NAME                                           READY   STATUS              RESTARTS   AGE
...
example-registry-quay-app-6786987b99-vgg2v              0/1     ContainerCreating   0          2s
example-registry-quay-app-7975d4889f-q7tvl              1/1     Running             0          5d21h
example-registry-quay-app-7975d4889f-zn8bb              1/1     Running             0          5d21h
example-registry-quay-app-upgrade-lswsn                 0/1     Completed           0          6d1h
example-registry-quay-config-editor-77847fc4f5-nsbbv    0/1     ContainerCreating   0          2s
example-registry-quay-config-editor-c6c4d9ccd-2mwg2     1/1     Running             0
5d21h
example-registry-quay-database-66969cd859-n2ssm         1/1     Running             0          6d1h
example-registry-quay-mirror-764d7b68d9-jmlkk           1/1     Terminating         0          5d21h
example-registry-quay-mirror-764d7b68d9-jqzwg           1/1     Terminating         0          5d21h
example-registry-quay-redis-7cc5f6c977-956g8            1/1     Running             0          5d21h
```

7. After your Red Hat Quay registry has reconfigured, enter the following command to check if the Red Hat Quay app pods are running:

```
$ oc get pods -n quay-enterprise
```

Example output

```
example-registry-quay-app-6786987b99-sz6kb              1/1     Running             0          7m45s
example-registry-quay-app-6786987b99-vgg2v              1/1     Running             0          9m1s
example-registry-quay-app-upgrade-lswsn                 0/1     Completed           0          6d1h
example-registry-quay-config-editor-77847fc4f5-nsbbv    1/1     Running             0          9m1s
example-registry-quay-database-66969cd859-n2ssm         1/1     Running             0          6d1h
example-registry-quay-mirror-758fc68ff7-5wxlp           1/1     Running             0          8m29s
example-registry-quay-mirror-758fc68ff7-lbl82           1/1     Running             0          8m29s
example-registry-quay-redis-7cc5f6c977-956g8            1/1     Running             0          5d21h
```

8. In your browser, access the registry endpoint and validate that the certificate has been updated appropriately. For example:

```
Common Name (CN) example-registry-quay-quay-enterprise.apps.docs.quayteam.org
Organisation (O) DOCS
Organisational Unit (OU) QUAY
```

9.3.2.3. Using the UI to create a build trigger

Use the following procedure to use the UI to create a build trigger.

Procedure

1. Log in to your Red Hat Quay repository.
2. Click **Create New Repository** and create a new registry, for example, **testrepo**.
3. On the **Repositories** page, click the **Builds** tab on the navigation pane. Alternatively, use the corresponding URL directly:

   ```
   https://example-registry-quay-quay-
   enterprise.apps.docs.quayteam.org/repository/quayadmin/testrepo?tab=builds
   ```

   IMPORTANT

   In some cases, the builder might have issues resolving hostnames. This issue might be related to the **dnsPolicy** being set to **default** on the job object. Currently, there is no workaround for this issue. It will be resolved in a future version of Red Hat Quay.

4. Click **Create Build Trigger → Custom Git Repository Push**.
5. Enter the HTTPS or SSH style URL used to clone your Git repository, then click **Continue**. For example:

   ```
   https://github.com/gabriel-rh/actions_test.git
   ```

6. Check **Tag manifest with the branch or tag name** and then click **Continue**.
7. Enter the location of the Dockerfile to build when the trigger is invoked, for example, **/Dockerfile** and click **Continue**.
8. Enter the location of the context for the Docker build, for example, /, and click **Continue**.
9. If warranted, create a Robot Account. Otherwise, click **Continue**.
10. Click **Continue** to verify the parameters.
11. On the **Builds** page, click **Options** icon of your Trigger Name, and then click **Run Trigger Now**.
12. Enter a commit SHA from the Git repository and click **Start Build**.
13. You can check the status of your build by clicking the commit in the **Build History** page, or by running **oc get pods -n virtual-builders**. For example:

    ```
    $ oc get pods -n virtual-builders
    ```

    #### Example output

    ```
    NAME                                                    READY   STATUS    RESTARTS   AGE
    f192fe4a-c802-4275-bcce-d2031e635126-9l2b5-25lg2   1/1     Running   0          7s
    ```

```
$ oc get pods -n virtual-builders
```

Example output

```
NAME                                          READY   STATUS        RESTARTS   AGE
f192fe4a-c802-4275-bcce-d2031e635126-9l2b5-25lg2   1/1     Terminating   0          9s
```

```
$ oc get pods -n virtual-builders
```

Example output

```
No resources found in virtual-builders namespace.
```

14. When the build is finished, you can check the status of the tag under **Tags** on the navigation pane.

> **NOTE**
>
> With early access, full build logs and timestamps of builds are currently unavailable.

9.3.2.4. Modifying your AWS S3 storage bucket

If you are using AWS S3 storage, you must change your storage bucket in the AWS console, prior to running builders.

Procedure

1. Log in to your AWS console at s3.console.aws.com.
2. In the search bar, search for **S3** and then click **S3**.
3. Click the name of your bucket, for example, **myawsbucket**.
4. Click the **Permissions** tab.
5. Under **Cross-origin resource sharing (CORS)** include the following parameters:

```
[
    {
        "AllowedHeaders": [
            "Authorization"
        ],
        "AllowedMethods": [
            "GET"
        ],
        "AllowedOrigins": [
            "*"
        ],
        "ExposeHeaders": [],
        "MaxAgeSeconds": 3000
    },
    {
        "AllowedHeaders": [
            "Content-Type",
            "x-amz-acl",
            "origin"
        ],
        "AllowedMethods": [
            "PUT"
        ],
        "AllowedOrigins": [
            "*"
        ],
        "ExposeHeaders": [],
        "MaxAgeSeconds": 3000
    }
]
```

CHAPTER 10. GEO-REPLICATION

Geo-replication allows multiple, geographically distributed Red Hat Quay deployments to work as a single registry from the perspective of a client or user. It significantly improves push and pull performance in a globally-distributed Red Hat Quay setup. Image data is asynchronously replicated in the background with transparent failover / redirect for clients.

With Red Hat Quay 3.7, deployments of Red Hat Quay with geo-replication is supported by standalone and Operator deployments.

10.1. GEO-REPLICATION FEATURES

- When geo-replication is configured, container image pushes will be written to the preferred storage engine for that Red Hat Quay instance (typically the nearest storage backend within the region).
- After the initial push, image data will be replicated in the background to other storage engines.
- The list of replication locations is configurable and those can be different storage backends.
- An image pull will always use the closest available storage engine, to maximize pull performance.
- If replication hasn’t been completed yet, the pull will use the source storage backend instead.

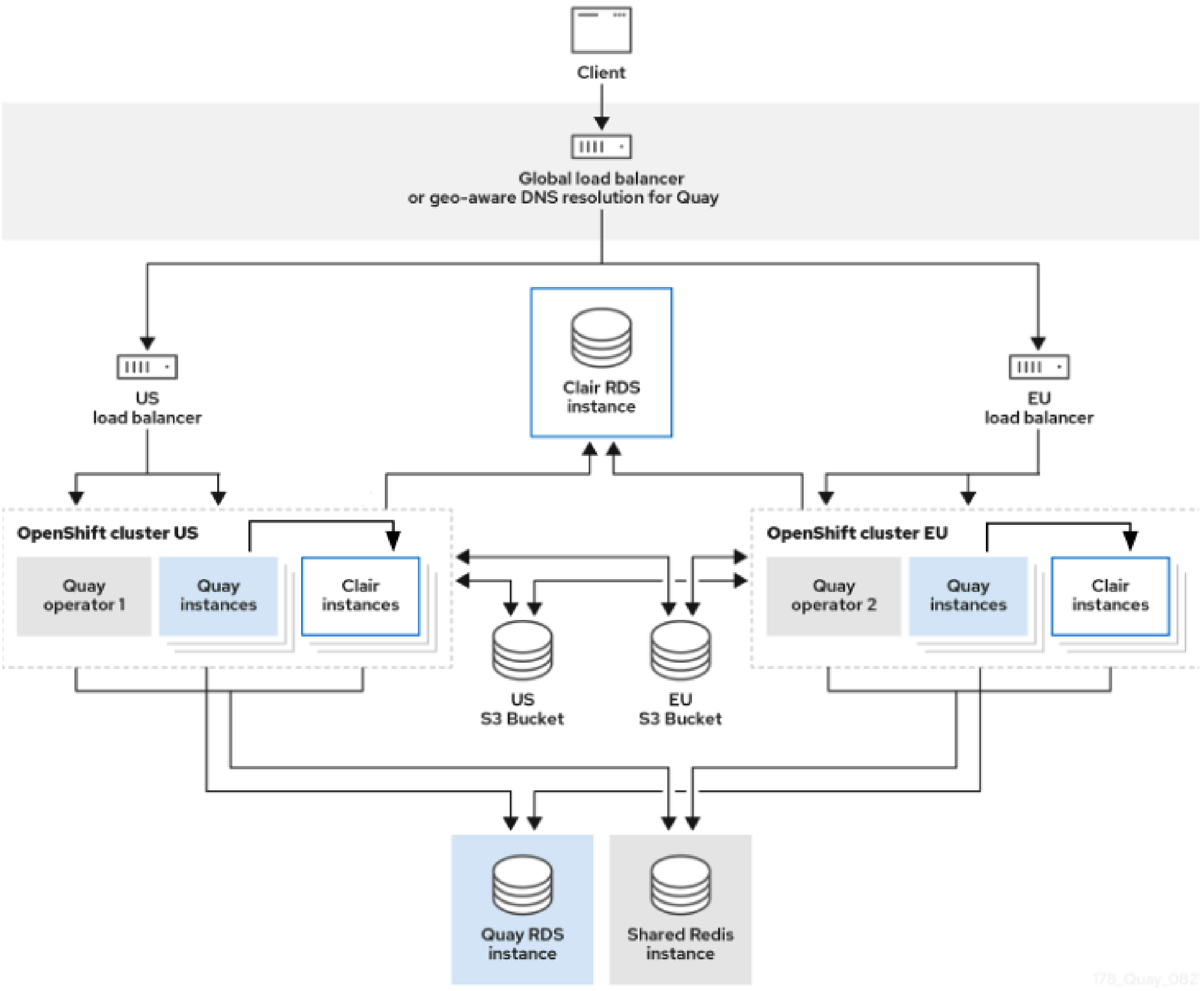
10.2. GEO-REPLICATION REQUIREMENTS AND CONSTRAINTS

- In geo-replicated setups, Red Hat Quay requires that all regions are able to read/write to all other region’s object storage. Object storage must be geographically accessible by all other regions.
- In case of an object storage system failure of one geo-replicating site, that site’s Red Hat Quay deployment must be shut down so that clients are redirected to the remaining site with intact storage systems by a global load balancer. Otherwise, clients will experience pull and push failures.
- Red Hat Quay has no internal awareness of the health or availability of the connected object storage system. If the object storage system of one site becomes unavailable, there will be no automatic redirect to the remaining storage system, or systems, of the remaining site, or sites.
- Geo-replication is asynchronous. The permanent loss of a site incurs the loss of the data that has been saved in that sites' object storage system but has not yet been replicated to the remaining sites at the time of failure.
- A single database, and therefore all metadata and Quay configuration, is shared across all regions.
  Geo-replication does not replicate the database. In the event of an outage, Red Hat Quay with geo-replication enabled will not failover to another database.
- A single Redis cache is shared across the entire Quay setup and needs to accessible by all Quay pods.
- The exact same configuration should be used across all regions, with exception of the storage backend, which can be configured explicitly using the **QUAY_DISTRIBUTED_STORAGE_PREFERENCE** environment variable.

- Geo-Replication requires object storage in each region. It does not work with local storage or NFS.
- Each region must be able to access every storage engine in each region (requires a network path).
- Alternatively, the storage proxy option can be used.
- The entire storage backend (all blobs) is replicated. This is in contrast to repository mirroring, which can be limited to an organization or repository or image.
- All Quay instances must share the same entrypoint, typically via load balancer.
- All Quay instances must have the same set of superusers, as they are defined inside the common configuration file.
- Geo-replication requires your Clair configuration to be set to **unmanaged**. An unmanaged Clair database allows the Red Hat Quay Operator to work in a geo-replicated environment, where multiple instances of the Operator must communicate with the same database. For more information, see Advanced Clair configuration .
- Geo-Replication requires SSL/TSL certificates and keys. For more information, see Using SSL to protect connections to Red Hat Quay.

If the above requirements cannot be met, you should instead use two or more distinct Quay deployments and take advantage of repository mirroring functionality.

10.3. GEO-REPLICATION USING THE RED HAT QUAY OPERATOR



In the example shown above, the Red Hat Quay Operator is deployed in two separate regions, with a common database and a common Redis instance. Localized image storage is provided in each region and image pulls are served from the closest available storage engine. Container image pushes are written to the preferred storage engine for the Quay instance, and will then be replicated, in the background, to the other storage engines.

Because the Operator now manages the Clair security scanner and its database separately, geo-replication setups can be leveraged so that they do not manage the Clair database. Instead, an external shared database would be used. Red Hat Quay and Clair support several providers and vendors of PostgreSQL, which can be found in the Red Hat Quay 3.x test matrix. Additionally, the Operator also supports custom Clair configurations that can be injected into the deployment, which allows users to configure Clair with the connection credentials for the external database.

10.3.1. Setting up geo-replication on Openshift

Procedure

1. Deploy Quay postgres instance:
    a. Login to the database
    b. Create a database for Quay

```
CREATE DATABASE quay;
```

c. Enable pg_trm extension inside the database

```
\c quay;
CREATE EXTENSION IF NOT EXISTS pg_trgm;
```

2. Deploy a Redis instance:

NOTE

- Deploying a Redis instance might be unnecessary if your cloud provider has its own service.
- Deploying a Redis instance is required if you are leveraging Builders.

a. Deploy a VM for Redis

b. Make sure that it is accessible from the clusters where Quay is running

c. Port 6379/TCP must be open

d. Run Redis inside the instance

```
sudo dnf install -y podman
podman run -d --name redis -p 6379:6379 redis
```

3. Create two object storage backends, one for each cluster
Ideally one object storage bucket will be close to the 1st cluster (primary) while the other will run closer to the 2nd cluster (secondary).

4. Deploy the clusters with the same config bundle, using environment variable overrides to select the appropriate storage backend for an individual cluster

5. Configure a load balancer, to provide a single entry point to the clusters

10.3.1.1. Configuration

The **config.yaml** file is shared between clusters, and will contain the details for the common PostgreSQL, Redis and storage backends:

config.yaml

```
SERVER_HOSTNAME: <georep.quayteam.org or any other name> (1)
DB_CONNECTION_ARGS:
  autorollback: true
  threadlocals: true
DB_URI: postgresql://postgres:password@10.19.0.1:5432/quay (2)
BUILDLOGS_REDIS:
  host: 10.19.0.2
  port: 6379
USER_EVENTS_REDIS:
  host: 10.19.0.2
  port: 6379
DISTRIBUTED_STORAGE_CONFIG:
  usstorage:
    - GoogleCloudStorage
    - access_key: GOOGQGPGVMASAAMQABCDEFG
      bucket_name: georep-test-bucket-0
      secret_key: AYWfEaxX/u84XRA2vUX5C987654321
      storage_path: /quaygcp
  eustorage:
    - GoogleCloudStorage
    - access_key: GOOGQGPGVMASAAMQWERTYUIOP
      bucket_name: georep-test-bucket-1
      secret_key: AYWfEaxX/u84XRA2vUX5Cuj12345678
      storage_path: /quaygcp
DISTRIBUTED_STORAGE_DEFAULT_LOCATIONS:
  - usstorage
  - eustorage
DISTRIBUTED_STORAGE_PREFERENCE:
  - usstorage
  - eustorage
FEATURE_STORAGE_REPLICATION: true
```

1 A proper **SERVER_HOSTNAME** must be used for the route and must match the hostname of the global load balancer.

2 To retrieve the configuration file for a Clair instance deployed using the OpenShift Operator, see Retrieving the Clair config.

Create the **configBundleSecret**:

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

In each of the clusters, set the **configBundleSecret** and use the **QUAY_DISTRIBUTED_STORAGE_PREFERENCE** environmental variable override to configure the appropriate storage for that cluster:

NOTE

The **config.yaml** file between both deployments must match. If making a change to one cluster, it must also be changed in the other.

US cluster

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

```
      managed: true
    - kind: tls
      managed: false
    - kind: postgres
      managed: false
    - kind: clairpostgres
      managed: false
    - kind: redis
      managed: false
    - kind: quay
      managed: true
      overrides:
        env:
        - name: QUAY_DISTRIBUTED_STORAGE_PREFERENCE
          value: usstorage
    - kind: mirror
      managed: true
      overrides:
        env:
        - name: QUAY_DISTRIBUTED_STORAGE_PREFERENCE
          value: usstorage
```

+

NOTE

Because TLS is unmanaged, and the route is managed, you must supply the certificates with either with the config tool or directly in the config bundle. For more information, see Configuring TLS and routes.

European cluster

```
apiVersion: quay.redhat.com/v1
kind: QuayRegistry
metadata:
  name: example-registry
  namespace: quay-enterprise
spec:
  configBundleSecret: georep-config-bundle
  components:
    - kind: objectstorage
      managed: false
    - kind: route
      managed: true
    - kind: tls
      managed: false
    - kind: postgres
      managed: false
    - kind: clairpostgres
      managed: false
    - kind: redis
      managed: false
    - kind: quay
      managed: true
      overrides:
```

```
      env:
      - name: QUAY_DISTRIBUTED_STORAGE_PREFERENCE
        value: eustorage
  - kind: mirror
    managed: true
    overrides:
      env:
      - name: QUAY_DISTRIBUTED_STORAGE_PREFERENCE
        value: eustorage
```

+

NOTE

Because TLS is unmanaged, and the route is managed, you must supply the certificates with either with the config tool or directly in the config bundle. For more information, see Configuring TLS and routes.

10.3.2. Mixed storage for geo-replication

Red Hat Quay geo-replication supports the use of different and multiple replication targets, for example, using AWS S3 storage on public cloud and using Ceph storage on-prem. This complicates the key requirement of granting access to all storage backends from all Red Hat Quay pods and cluster nodes. As a result, it is recommended that you:

- Use a VPN to prevent visibility of the internal storage *or*
- Use a token pair that only allows access to the specified bucket used by Quay

This will result in the public cloud instance of Red Hat Quay having access to on-prem storage but the network will be encrypted, protected, and will use ACLs, thereby meeting security requirements.

If you cannot implement these security measures, it may be preferable to deploy two distinct Red Hat Quay registries and to use repository mirroring as an alternative to geo-replication.

CHAPTER 11. BACKING UP AND RESTORING RED HAT QUAY MANAGED BY THE RED HAT QUAY OPERATOR

Use the content within this section to back up and restore Red Hat Quay when managed by the Red Hat Quay Operator on OpenShift Container Platform.

11.1. BACKING UP RED HAT QUAY

This procedure describes how to create a backup of Red Hat Quay deployed on OpenShift Container Platform using the Red Hat Quay Operator

Prerequisites

- A healthy Red Hat Quay deployment on OpenShift Container Platform using the Red Hat Quay Operator (status condition **Available** is set to **true**)
- The components **quay**, **postgres** and **objectstorage** are set to **managed: true**
- If the component **clair** is set to **managed: true** the component **clairpostgres** is also set to **managed: true** (starting with Red Hat Quay Operator v3.7 or later)

> **NOTE**
>
> If your deployment contains partially unmanaged database or storage components and you are using external services for Postgres or S3-compatible object storage to run your Red Hat Quay deployment, you must refer to the service provider or vendor documentation to create a backup of the data. You can refer to the tools described in this guide as a starting point on how to backup your external Postgres database or object storage.

11.1.1. Red Hat Quay configuration backup

1. Backup the **QuayRegistry** custom resource by exporting it:

```
$ oc get quayregistry <quay-registry-name> -n <quay-namespace> -o yaml > quay-registry.yaml
```

2. Edit the resulting **quayregistry.yaml** and remove the status section and the following metadata fields:

```
metadata.creationTimestamp
metadata.finalizers
metadata.generation
metadata.resourceVersion
metadata.uid
```

3. Backup the managed keys secret:

NOTE

If you are running a version older than Red Hat Quay 3.7.0, this step can be skipped. Some secrets are automatically generated while deploying Quay for the first time. These are stored in a secret called **<quay-registry-name>-quay-registry-managed-secret-keys** in the namespace of the **QuayRegistry** resource.

```
$ oc get secret -n <quay-namespace> <quay-registry-name>-quay-registry-managed-secret-
keys -o yaml > managed-secret-keys.yaml
```

4. Edit the the resulting **managed-secret-keys.yaml** file and remove the entry **metadata.ownerReferences**. Your **managed-secret-keys.yaml** file should look similar to the following:

```
apiVersion: v1
kind: Secret
type: Opaque
metadata:
  name: <quayname>-quay-registry-managed-secret-keys
  namespace: <quay-namespace>
data:
  CONFIG_EDITOR_PW: <redacted>
  DATABASE_SECRET_KEY: <redacted>
  DB_ROOT_PW: <redacted>
  DB_URI: <redacted>
  SECRET_KEY: <redacted>
  SECURITY_SCANNER_V4_PSK: <redacted>
```

All information under the **data** property should remain the same.

5. Backup the current Quay configuration:

```
$ oc get secret -n <quay-namespace>  $(oc get quayregistry <quay-registry-name> -n
<quay-namespace>  -o jsonpath='{.spec.configBundleSecret}') -o yaml > config-bundle.yaml
```

6. Backup the **/conf/stack/config.yaml** file mounted inside of the Quay pods:

```
$ oc exec -it quay-pod-name -- cat /conf/stack/config.yaml > quay-config.yaml
```

11.1.2. Scale down your Red Hat Quay deployment

IMPORTANT

This step is needed to create a consistent backup of the state of your Red Hat Quay deployment. Do not omit this step, including in setups where Postgres databases and/or S3-compatible object storage are provided by external services (unmanaged by the Operator).

1. **For Operator version 3.7 and newer:** Scale down the Red Hat Quay deployment by disabling auto scaling and overriding the replica count for Red Hat Quay, mirror workers, and Clair (if managed). Your **QuayRegistry** resource should look similar to the following:

```
apiVersion: quay.redhat.com/v1
kind: QuayRegistry
metadata:
  name: registry
  namespace: ns
spec:
  components:
    …
    - kind: horizontalpodautoscaler
      managed: false (1)
    - kind: quay
      managed: true
      overrides: (2)
        replicas: 0
    - kind: clair
      managed: true
      overrides:
        replicas: 0
    - kind: mirror
      managed: true
      overrides:
        replicas: 0
    …
```

(1) Disable auto scaling of Quay, Clair and Mirroring workers

(2) Set the replica count to 0 for components accessing the database and objectstorage

2. **For Operator version 3.6 and earlier**: Scale down the Red Hat Quay deployment by scaling down the Red Hat Quay Operator first and then the managed Red Hat Quay resources:

```
$ oc scale --replicas=0 deployment $(oc get deployment -n <quay-operator-namespace>|awk '/^quay-operator/ {print $1}') -n <quay-operator-namespace>
$ oc scale --replicas=0 deployment $(oc get deployment -n <quay-namespace>|awk '/quay-app/ {print $1}') -n <quay-namespace>
$ oc scale --replicas=0 deployment $(oc get deployment -n <quay-namespace>|awk '/quay-mirror/ {print $1}') -n <quay-namespace>
$ oc scale --replicas=0 deployment $(oc get deployment -n <quay-namespace>|awk '/clair-app/ {print $1}') -n <quay-namespace>
```

3. Wait for the **registry-quay-app**, **registry-quay-mirror** and **registry-clair-app** pods (depending on which components you set to be managed by the Red Hat Quay Operator) to disappear. You can check their status by running the following command:

```
$ oc get pods -n <quay-namespace>
```

Example output:

```
$ oc get pod

quay-operator.v3.7.1-6f9d859bd-p5ftc              1/1     Running     0              12m
quayregistry-clair-postgres-7487f5bd86-xnxpr       1/1     Running     1 (12m ago)    12m
quayregistry-quay-app-upgrade-xq2v6                0/1     Completed   0              12m
```

```
quayregistry-quay-config-editor-6dfdcfc44f-hlvwm   1/1     Running     0            73s
quayregistry-quay-database-859d5445ff-cqthr        1/1     Running     0            12m
quayregistry-quay-redis-84f888776f-hhgms           1/1     Running     0            12m
```

11.1.3. Red Hat Quay managed database backup

NOTE

If your Red Hat Quay deployment is configured with external (unmanged) Postgres database(s), refer to your vendor's documentation on how to create a consistent backup of these databases.

1. Identify the Quay PostgreSQL pod name:

   ```
   $ oc get pod -l quay-component=postgres -n <quay-namespace> -o
   jsonpath='{.items[0].metadata.name}'
   ```

   Example output:

   ```
   quayregistry-quay-database-59f54bb7-58xs7
   ```

2. Obtain the Quay database name:

   ```
   $ oc -n <quay-namespace> rsh $(oc get pod -l app=quay -o NAME -n <quay-namespace>
   |head -n 1) cat /conf/stack/config.yaml|awk -F"/" '/^DB_URI/ {print $4}'
   quayregistry-quay-database
   ```

3. Download a backup database:

   ```
   $ oc exec quayregistry-quay-database-59f54bb7-58xs7 -- /usr/bin/pg_dump -C quayregistry-
   quay-database  > backup.sql
   ```

11.1.3.1. Red Hat Quay managed object storage backup

The instructions in this section apply to the following configurations:

- Standalone, multi-cloud object gateway configurations
- OpenShift Data Foundations storage requires that the Red Hat Quay Operator provisioned an S3 object storage bucket from, through the ObjectStorageBucketClaim API

NOTE

If your Red Hat Quay deployment is configured with external (unmanged) object storage, refer to your vendor's documentation on how to create a copy of the content of Quay's storage bucket.

1. Decode and export the **AWS_ACCESS_KEY_ID**:

   ```
   $ export AWS_ACCESS_KEY_ID=$(oc get secret -l app=noobaa -n <quay-namespace>  -o
   jsonpath='{.items[0].data.AWS_ACCESS_KEY_ID}' |base64 -d)
   ```

2. Decode and export the **AWS_SECRET_ACCESS_KEY_ID**:

```
$ export AWS_SECRET_ACCESS_KEY=$(oc get secret -l app=noobaa -n <quay-
namespace> -o jsonpath='{.items[0].data.AWS_SECRET_ACCESS_KEY}' |base64 -d)
```

3. Create a new directory and copy all blobs to it:

```
$ mkdir blobs

$ aws s3 sync --no-verify-ssl --endpoint https://$(oc get route s3 -n openshift-storage  -o
jsonpath='{.spec.host}')  s3://$(oc get cm -l app=noobaa -n <quay-namespace> -o
jsonpath='{.items[0].data.BUCKET_NAME}') ./blobs
```

NOTE

You can also use rclone or sc3md instead of the AWS command line utility.

11.1.4. Scale the Red Hat Quay deployment back up

1. **For Operator version 3.7 and newer:**Scale up the Red Hat Quay deployment by re-enabling auto scaling, if desired, and removing the replica overrides for Quay, mirror workers and Clair as applicable. Your **QuayRegistry** resource should look similar to the following:

```
apiVersion: quay.redhat.com/v1
kind: QuayRegistry
metadata:
  name: registry
  namespace: ns
spec:
  components:
    …
    - kind: horizontalpodautoscaler
      managed: true (1)
    - kind: quay (2)
      managed: true
    - kind: clair
      managed: true
    - kind: mirror
      managed: true
    …
```

(1) Re-enables auto scaling of Quay, Clair and Mirroring workers again (if desired)

(2) Replica overrides are removed again to scale the Quay components back up

2. **For Operator version 3.6 and earlier:**Scale up the Red Hat Quay deployment by scaling up the Red Hat Quay Operator again:

```
$ oc scale --replicas=1 deployment $(oc get deployment -n <quay-operator-namespace> |
awk '/^quay-operator/ {print $1}') -n <quay-operator-namespace>
```

3. Check the status of the Red Hat Quay deployment:

```
$ oc wait quayregistry registry --for=condition=Available=true -n <quay-namespace>
```

Example output:

```
apiVersion: quay.redhat.com/v1
kind: QuayRegistry
metadata:
  ...
  name: registry
  namespace: <quay-namespace>
  ...
spec:
  ...
status:
  - lastTransitionTime: '2022-06-20T05:31:17Z'
    lastUpdateTime: '2022-06-20T17:31:13Z'
    message: All components reporting as healthy
    reason: HealthChecksPassing
    status: 'True'
    type: Available
```

11.2. RESTORING RED HAT QUAY

This procedure is used to restore Red Hat Quay when the Red Hat Quay Operator manages the database. It should be performed after a backup of your Red Hat Quay registry has been performed. See Backing up Red Hat Quay for more information.

Prerequisites

- Red Hat Quay is deployed on OpenShift Container Platform using the Red Hat Quay Operator.
- A backup of the Red Hat Quay configuration managed by the Red Hat Quay Operator has been created following the instructions in the Backing up Red Hat Quay section
- Your Red Hat Quay database has been backed up.
- The object storage bucket used by Red Hat Quay has been backed up.
- The components **quay**, **postgres** and **objectstorage** are set to **managed: true**
- If the component **clair** is set to **managed: true**, the component **clairpostgres** is also set to **managed: true** (starting with Red Hat Quay Operator v3.7 or later)
- There is no running Red Hat Quay deployment managed by the Red Hat Quay Operator in the target namespace on your OpenShift Container Platform cluster

NOTE

If your deployment contains partially unmanaged database or storage components and you are using external services for Postgres or S3-compatible object storage to run your Red Hat Quay deployment, you must refer to the service provider or vendor documentation to restore their data from a backup prior to restore Red Hat Quay

11.2.1. Restoring Red Hat Quay and its configuration from a backup

NOTE

These instructions assume you have followed the process in the Backing up Red Hat Quay guide and create the backup files with the same names.

1. Restore the backed up Red Hat Quay configuration and the generated keys from the backup:

```
$ oc create -f ./config-bundle.yaml

$ oc create -f ./managed-secret-keys.yaml
```

IMPORTANT

If you receive the error **Error from server (AlreadyExists): error when creating "./config-bundle.yaml": secrets "config-bundle-secret" already exists**, you must delete your existing resource with **$ oc delete Secret config-bundle-secret -n <quay-namespace>** and recreate it with **$ oc create -f ./config-bundle.yaml**.

2. Restore the **QuayRegistry** custom resource:

```
$ oc create -f ./quay-registry.yaml
```

3. Check the status of the Red Hat Quay deployment and wait for it to be available:

```
$ oc wait quayregistry registry --for=condition=Available=true -n <quay-namespace>
```

11.2.2. Scale down your Red Hat Quay deployment

1. **For Operator version 3.7 and newer:** Scale down the Red Hat Quay deployment by disabling auto scaling and overriding the replica count for Quay, mirror workers and Clair (if managed). Your **QuayRegistry** resource should look similar to the following:

```
apiVersion: quay.redhat.com/v1
kind: QuayRegistry
metadata:
  name: registry
  namespace: ns
spec:
  components:
    …
    - kind: horizontalpodautoscaler
      managed: false (1)
    - kind: quay
      managed: true
      overrides: (2)
        replicas: 0
    - kind: clair
      managed: true
      overrides:
        replicas: 0
    - kind: mirror
      managed: true
      overrides:
        replicas: 0
    …
```

1. Disable auto scaling of Quay, Clair and Mirroring workers
2. Set the replica count to 0 for components accessing the database and objectstorage

2. **For Operator version 3.6 and earlier:**Scale down the Red Hat Quay deployment by scaling down the Red Hat Quay Operator first and then the managed Red Hat Quay resources:

```
$ oc scale --replicas=0 deployment $(oc get deployment -n <quay-operator-namespace>|awk '/^quay-operator/ {print $1}') -n <quay-operator-namespace>

$ oc scale --replicas=0 deployment $(oc get deployment -n <quay-namespace>|awk '/quay-app/ {print $1}') -n <quay-namespace>
$ oc scale --replicas=0 deployment $(oc get deployment -n <quay-namespace>|awk '/quay-mirror/ {print $1}') -n <quay-namespace>
$ oc scale --replicas=0 deployment $(oc get deployment -n <quay-namespace>|awk '/clair-app/ {print $1}') -n <quay-namespace>
```

3. Wait for the **registry-quay-app**, **registry-quay-mirror** and **registry-clair-app** pods (depending on which components you set to be managed by Operator) to disappear. You can check their status by running the following command:

```
$ oc get pods -n <quay-namespace>
```

Example output:

```
registry-quay-config-editor-77847fc4f5-nsbbv   1/1     Running            0          9m1s
registry-quay-database-66969cd859-n2ssm        1/1     Running            0          6d1h
registry-quay-redis-7cc5f6c977-956g8           1/1     Running            0          5d21h
```

11.2.3. Restore your Red Hat Quay database

1. Identify your Quay database pod:

```
$ oc get pod -l quay-component=postgres -n  <quay-namespace> -o jsonpath='{.items[0].metadata.name}'
```

Example output:

```
quayregistry-quay-database-59f54bb7-58xs7
```

2. Upload the backup by copying it from the local environment and into the pod:

```
$ oc cp ./backup.sql -n <quay-namespace> registry-quay-database-66969cd859-n2ssm:/tmp/backup.sql
```

3. Open a remote terminal to the database:

```
$ oc rsh -n <quay-namespace> registry-quay-database-66969cd859-n2ssm
```

4. Enter psql:

```
bash-4.4$ psql
```

5. You can list the database by running the following command:

```
postgres=# \l
```

Example output:

```
                                                  List of databases
           Name            |            Owner            | Encoding |  Collate   |   Ctype    |   Access
privileges
---------------------------+-----------------------------+----------+------------+------------+---------------
--------
postgres                   | postgres                    | UTF8     | en_US.utf8 | en_US.utf8 |
quayregistry-quay-database | quayregistry-quay-database | UTF8     | en_US.utf8 |
en_US.utf8 |
```

6. Drop the database:

```
postgres=# DROP DATABASE "quayregistry-quay-database";
```

Example output:

```
DROP DATABASE
```

7. Exit the postgres CLI to re-enter bash-4.4:

```
\q
```

8. Redirect your PostgreSQL database to your backup database:

```
sh-4.4$ psql < /tmp/backup.sql
```

9. Exit bash:

```
sh-4.4$ exit
```

11.2.4. Restore your Red Hat Quay object storage data

1. Export the **AWS_ACCESS_KEY_ID**:

```
$ export AWS_ACCESS_KEY_ID=$(oc get secret -l app=noobaa -n <quay-namespace>  -o
jsonpath='{.items[0].data.AWS_ACCESS_KEY_ID}' |base64 -d)
```

2. Export the **AWS_SECRET_ACCESS_KEY**:

```
$ export AWS_SECRET_ACCESS_KEY=$(oc get secret -l app=noobaa -n <quay-
namespace> -o jsonpath='{.items[0].data.AWS_SECRET_ACCESS_KEY}' |base64 -d)
```

3. Upload all blobs to the bucket by running the following command:

```
$ aws s3 sync --no-verify-ssl --endpoint https://$(oc get route s3 -n openshift-storage  -o
jsonpath='{.spec.host}') ./blobs  s3://$(oc get cm -l app=noobaa -n <quay-namespace> -o
jsonpath='{.items[0].data.BUCKET_NAME}')
```

NOTE

You can also use rclone or sc3md instead of the AWS command line utility.

11.2.5. Scale up your Red Hat Quay deployment

1. **For Operator version 3.7 and newer:** Scale up the Red Hat Quay deployment by re-enabling auto scaling, if desired, and removing the replica overrides for Quay, mirror workers and Clair as applicable. Your **QuayRegistry** resource should look similar to the following:

```
apiVersion: quay.redhat.com/v1
kind: QuayRegistry
metadata:
  name: registry
  namespace: ns
spec:
  components:
    …
    - kind: horizontalpodautoscaler
      managed: true (1)
    - kind: quay (2)
      managed: true
    - kind: clair
      managed: true
    - kind: mirror
      managed: true
    …
```

(1) Re-enables auto scaling of Red Hat Quay, Clair and mirroring workers again (if desired)

(2) Replica overrides are removed again to scale the Red Hat Quay components back up

2. **For Operator version 3.6 and earlier:** Scale up the Red Hat Quay deployment by scaling up the Red Hat Quay Operator again:

```
$ oc scale --replicas=1 deployment $(oc get deployment -n <quay-operator-namespace> |
awk '/^quay-operator/ {print $1}') -n <quay-operator-namespace>
```

3. Check the status of the Red Hat Quay deployment:

```
$ oc wait quayregistry registry --for=condition=Available=true -n <quay-namespace>
```

Example output:

```
apiVersion: quay.redhat.com/v1
kind: QuayRegistry
metadata:
  ...
  name: registry
  namespace: <quay-namespace>
  ...
spec:
  ...
status:
  - lastTransitionTime: '2022-06-20T05:31:17Z'
    lastUpdateTime: '2022-06-20T17:31:13Z'
    message: All components reporting as healthy
    reason: HealthChecksPassing
    status: 'True'
    type: Available
```

CHAPTER 12. DEPLOYING IPV6 ON THE RED HAT QUAY OPERATOR

Your Red Hat Quay Operator deployment can now be served in locations that only support IPv6, such as Telco and Edge environments.

For a list of known limitations, see IPv6 limitations

12.1. ENABLING THE IPV6 PROTOCOL FAMILY

Use the following procedure to enable IPv6 support on your standalone Red Hat Quay deployment.

Prerequisites

- You have updated Red Hat Quay to 3.8.
- Your host and container software platform (Docker, Podman) must be configured to support IPv6.

Procedure

1. In your deployment's **config.yaml** file, add the **FEATURE_LISTEN_IP_VERSION** parameter and set it to **IPv6**, for example:

```
---
FEATURE_GOOGLE_LOGIN: false
FEATURE_INVITE_ONLY_USER_CREATION: false
FEATURE_LISTEN_IP_VERSION: IPv6
FEATURE_MAILING: false
FEATURE_NONSUPERUSER_TEAM_SYNCING_SETUP: false
---
```

2. Start, or restart, your Red Hat Quay deployment.

3. Check that your deployment is listening to IPv6 by entering the following command:

```
$ curl <quay_endpoint>/health/instance
{"data":{"services":
{"auth":true,"database":true,"disk_space":true,"registry_gunicorn":true,"service_key":true,"web_
gunicorn":true}},"status_code":200}
```

After enabling IPv6 in your deployment's **config.yaml**, all Red Hat Quay features can be used as normal, so long as your environment is configured to use IPv6 and is not hindered by the IPv6 and dual-stack limitations.

WARNING

If your environment is configured to IPv4, but the **FEATURE_LISTEN_IP_VERSION** configuration field is set to **IPv6**, Red Hat Quay will fail to deploy.

12.2. IPV6 LIMITATIONS

- Currently, attempting to configure your Red Hat Quay deployment with the common Azure Blob Storage configuration will not work on IPv6 single stack environments. Because the endpoint of Azure Blob Storage does not support IPv6, there is no workaround in place for this issue.
  For more information, see PROJQUAY-4433.
- Currently, attempting to configure your Red Hat Quay deployment with Amazon S3 CloudFront will not work on IPv6 single stack environments. Because the endpoint of Amazon S3 CloudFront does not support IPv6, there is no workaround in place for this issue.
  For more information, see PROJQUAY-4470.
- Currently, OpenShift Data Foundations (ODF) is unsupported when Red Hat Quay is deployed on IPv6 single stack environments. As a result, ODF cannot be used in IPv6 environments. This limitation is scheduled to be fixed in a future version of OpenShift Data Foundations.
- Currently, dual-stack (IPv4 and IPv6) support does not work on Red Hat Quay OpenShift Container Platform deployments. When Red Hat Quay 3.8 is deployed on OpenShift Container Platform with dual-stack support enabled, the Quay Route generated by the Red Hat Quay Operator only generates an IPv4 address, and not an IPv6 address. As a result, clients with an IPv6 address cannot access the Red Hat Quay application on OpenShift Container Platform. This limitation is scheduled to be fixed in a future version of OpenShift Container Platform.

CHAPTER 13. UPGRADING THE QUAY OPERATOR OVERVIEW

The Quay Operator follows a *synchronized versioning* scheme, which means that each version of the Operator is tied to the version of Quay and the components that it manages. There is no field on the **QuayRegistry** custom resource which sets the version of Quay to deploy; the Operator only knows how to deploy a single version of all components. This scheme was chosen to ensure that all components work well together and to reduce the complexity of the Operator needing to know how to manage the lifecycles of many different versions of Quay on Kubernetes.

13.1. OPERATOR LIFECYCLE MANAGER

The Quay Operator should be installed and upgraded using the Operator Lifecycle Manager (OLM). When creating a **Subscription** with the default **approvalStrategy: Automatic**, OLM will automatically upgrade the Quay Operator whenever a new version becomes available.

WARNING

When the Quay Operator is installed via Operator Lifecycle Manager, it may be configured to support automatic or manual upgrades. This option is shown on the **Operator Hub** page for the Quay Operator during installation. It can also be found in the Quay Operator **Subscription** object via the **approvalStrategy** field. Choosing **Automatic** means that your Quay Operator will automatically be upgraded whenever a new Operator version is released. If this is not desirable, then the **Manual** approval strategy should be selected.

13.2. UPGRADING THE QUAY OPERATOR

The standard approach for upgrading installed Operators on OpenShift is documented at Upgrading installed Operators.

In general, Red Hat Quay supports upgrades from a prior (N-1) minor version only. For example, upgrading directly from Red Hat Quay 3.0.5 to the latest version of 3.5 is not supported. Instead, users would have to upgrade as follows:

1. 3.0.5 → 3.1.3
2. 3.1.3 → 3.2.2
3. 3.2.2 → 3.3.4
4. 3.3.4 → 3.4.z
5. 3.4.z → 3.5.z

This is required to ensure that any necessary database migrations are done correctly and in the right order during the upgrade.

In some cases, Red Hat Quay supports direct, single-step upgrades from prior (N-2, N-3) minor versions. This exception to the normal, prior minor version-only, upgrade simplifies the upgrade procedure for customers on older releases. The following upgrade paths are supported:

1. 3.3.z → 3.6.z
2. 3.4.z → 3.6.z
3. 3.4.z → 3.7.z
4. 3.5.z → 3.7.z
5. 3.7.z → 3.8.z

For users on standalone deployments of Quay wanting to upgrade to 3.8, see the Standalone upgrade guide.

13.2.1. Upgrading Quay

To update Quay from one minor version to the next, for example, 3.4 → 3.5, you need to change the update channel for the Quay Operator.

For **z** stream upgrades, for example, 3.4.2 → 3.4.3, updates are released in the major-minor channel that the user initially selected during install. The procedure to perform a **z** stream upgrade depends on the **approvalStrategy** as outlined above. If the approval strategy is set to **Automatic**, the Quay Operator will upgrade automatically to the newest **z** stream. This results in automatic, rolling Quay updates to newer **z** streams with little to no downtime. Otherwise, the update must be manually approved before installation can begin.

13.2.2. Notes on upgrading directly from 3.3.z or 3.4.z to 3.6

13.2.2.1. Upgrading with edge routing enabled

- Previously, when running a 3.3.z version of Red Hat Quay with edge routing enabled, users were unable to upgrade to 3.4.z versions of Red Hat Quay. This has been resolved with the release of Red Hat Quay 3.6.
- When upgrading from 3.3.z to 3.6, if **tls.termination** is set to **none** in your Red Hat Quay 3.3.z deployment, it will change to HTTPS with TLS edge termination and use the default cluster wildcard certificate. For example:

```
apiVersion: redhatcop.redhat.io/v1alpha1
kind: QuayEcosystem
metadata:
  name: quay33
spec:
  quay:
    imagePullSecretName: redhat-pull-secret
    enableRepoMirroring: true
    image: quay.io/quay/quay:v3.3.4-2
    ...
    externalAccess:
      hostname: quayv33.apps.devcluster.openshift.com
      tls:
        termination: none
    database:
...
```

13.2.2.2. Upgrading with custom TLS certificate/key pairs without Subject Alternative Names

There is an issue for customers using their own TLS certificate/key pairs without Subject Alternative Names (SANs) when upgrading from Red Hat Quay 3.3.4 to Red Hat Quay 3.6 directly. During the upgrade to Red Hat Quay 3.6, the deployment is blocked, with the error message from the Quay Operator pod logs indicating that the Quay TLS certificate must have SANs.

If possible, you should regenerate your TLS certificates with the correct hostname in the SANs. A possible workaround involves defining an environment variable in the **quay-app**, **quay-upgrade** and **quay-config-editor** pods after upgrade to enable CommonName matching:

```
GODEBUG=x509ignoreCN=0
```

The **GODEBUG=x509ignoreCN=0** flag enables the legacy behavior of treating the CommonName field on X.509 certificates as a host name when no SANs are present. However, this workaround is not recommended, as it will not persist across a redeployment.

13.2.2.3. Configuring Clair v4 when upgrading from 3.3.z or 3.4.z to 3.6 using the Quay Operator

To set up Clair v4 on a new Red Hat Quay deployment on OpenShift, it is highly recommended to use the Quay Operator. By default, the Quay Operator will install or upgrade a Clair deployment along with your Red Hat Quay deployment and configure Clair security scanning automatically.

For instructions on setting up Clair v4 on OpenShift, see Setting Up Clair on a Red Hat Quay OpenShift deployment.

13.2.3. Swift configuration when upgrading from 3.3.z to 3.6

When upgrading from Red Hat Quay 3.3.z to 3.6.z, some users might receive the following error: **Switch auth v3 requires tenant_id (string) in os_options**. As a workaround, you can manually update your **DISTRIBUTED_STORAGE_CONFIG** to add the **os_options** and **tenant_id** parameters:

```
DISTRIBUTED_STORAGE_CONFIG:
  brscale:
  - SwiftStorage
  - auth_url: http://****/v3
    auth_version: "3"
    os_options:
      tenant_id: ****
      project_name: ocp-base
      user_domain_name: Default
    storage_path: /datastorage/registry
    swift_container: ocp-svc-quay-ha
    swift_password: *****
    swift_user: *****
```

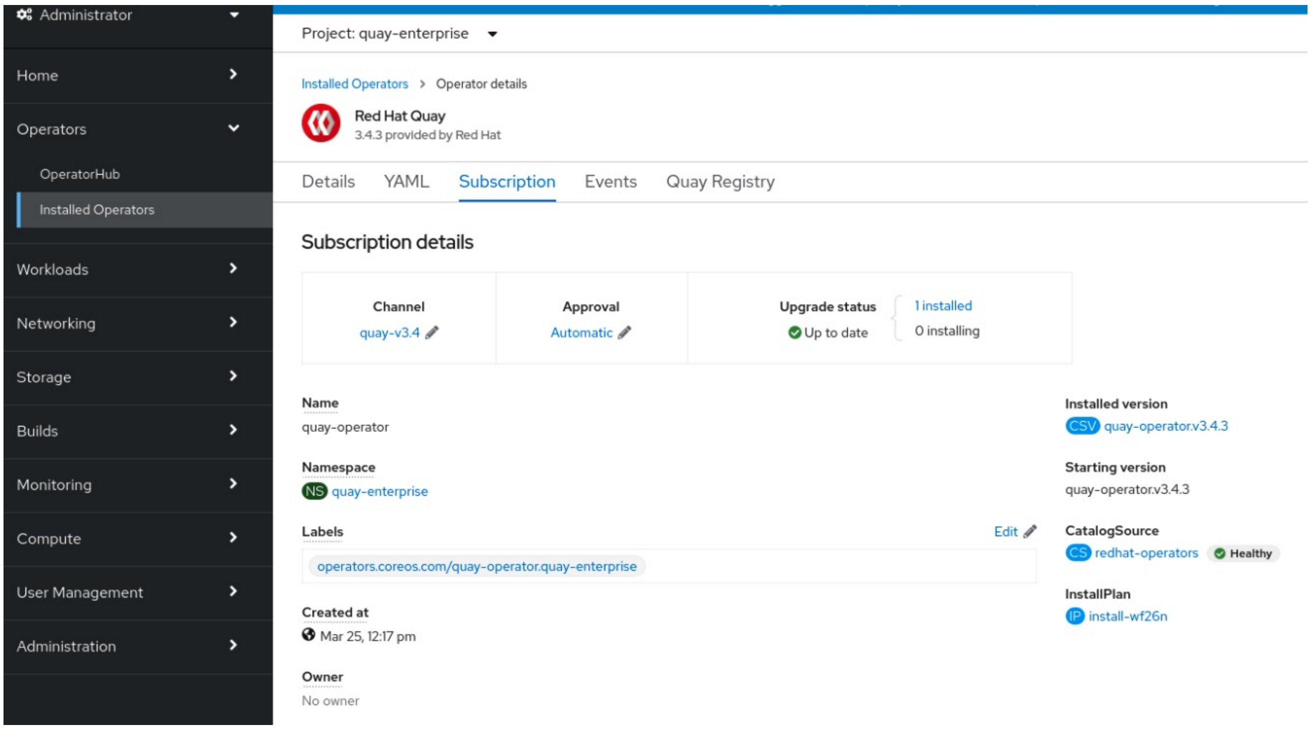
13.2.4. Changing the update channel for an Operator

The subscription of an installed Operator specifies an update channel, which is used to track and receive updates for the Operator. To upgrade the Quay Operator to start tracking and receiving updates from a newer channel, change the update channel in the **Subscription** tab for the installed Quay Operator. For subscriptions with an **Automatic** approval strategy, the upgrade begins automatically and can be monitored on the page that lists the Installed Operators.

13.2.5. Manually approving a pending Operator upgrade

If an installed Operator has the approval strategy in its subscription set to **Manual**, when new updates are released in its current update channel, the update must be manually approved before installation can begin. If the Quay Operator has a pending upgrade, this status will be displayed in the list of Installed Operators. In the **Subscription** tab for the Quay Operator, you can preview the install plan and review the resources that are listed as available for upgrade. If satisfied, click **Approve** and return to the page that lists Installed Operators to monitor the progress of the upgrade.

The following image shows the **Subscription** tab in the UI, including the update **Channel**, the **Approval** strategy, the **Upgrade status** and the **InstallPlan**:



Project: quay-enterprise

Red Hat Quay
3.4.3 provided by Red Hat

Subscription details

Channel	Approval	Upgrade status	
quay-v3.4	Automatic	Up to date	1 installed 0 installing

Name
quay-operator

Namespace
NS quay-enterprise

Labels
operators.coreos.com/quay-operator.quay-enterprise

Created at
Mar 25, 12:17 pm

Owner
No owner

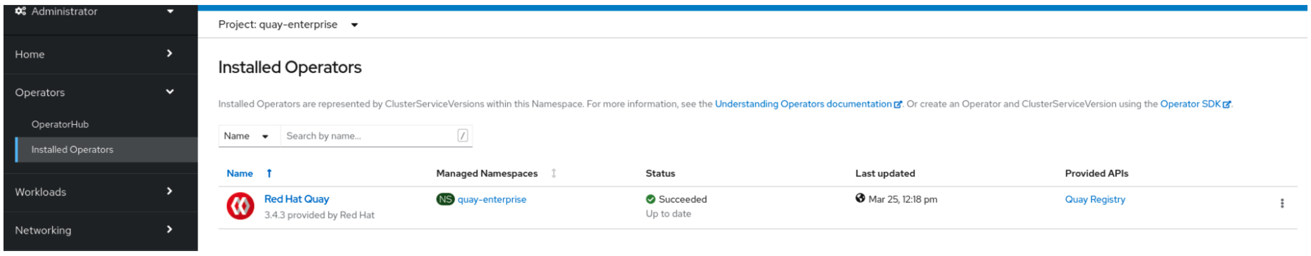
Installed version
CSV quay-operator.v3.4.3

Starting version
quay-operator.v3.4.3

CatalogSource
CS redhat-operators Healthy

InstallPlan
IP install-wf26n

The list of Installed Operators provides a high-level summary of the current Quay installation:



Project: quay-enterprise

Installed Operators

Installed Operators are represented by ClusterServiceVersions within this Namespace. For more information, see the Understanding Operators documentation. Or create an Operator and ClusterServiceVersion using the Operator SDK.

Name	Managed Namespaces	Status	Last updated	Provided APIs
Red Hat Quay 3.4.3 provided by Red Hat	NS quay-enterprise	Succeeded Up to date	Mar 25, 12:18 pm	Quay Registry

13.3. UPGRADING A QUAYREGISTRY

When the Quay Operator starts, it immediately looks for any **QuayRegistries** it can find in the namespace(s) it is configured to watch. When it finds one, the following logic is used:

- If **status.currentVersion** is unset, reconcile as normal.
- If **status.currentVersion** equals the Operator version, reconcile as normal.

- If **status.currentVersion** does not equal the Operator version, check if it can be upgraded. If it can, perform upgrade tasks and set the **status.currentVersion** to the Operator's version once complete. If it cannot be upgraded, return an error and leave the **QuayRegistry** and its deployed Kubernetes objects alone.

13.4. UPGRADING A QUAYECOSYSTEM

Upgrades are supported from previous versions of the Operator which used the **QuayEcosystem** API for a limited set of configurations. To ensure that migrations do not happen unexpectedly, a special label needs to be applied to the **QuayEcosystem** for it to be migrated. A new **QuayRegistry** will be created for the Operator to manage, but the old **QuayEcosystem** will remain until manually deleted to ensure that you can roll back and still access Quay in case anything goes wrong. To migrate an existing **QuayEcosystem** to a new **QuayRegistry**, follow these steps:

1. Add **"quay-operator/migrate": "true"** to the **metadata.labels** of the **QuayEcosystem**.

```
$ oc edit quayecosystem <quayecosystemname>
```

```
metadata:
  labels:
    quay-operator/migrate: "true"
```

2. Wait for a **QuayRegistry** to be created with the same **metadata.name** as your **QuayEcosystem**. The **QuayEcosystem** will be marked with the label **"quay-operator/migration-complete": "true"**.

3. Once the **status.registryEndpoint** of the new **QuayRegistry** is set, access Quay and confirm all data and settings were migrated successfully.

4. When you are confident everything worked correctly, you may delete the **QuayEcosystem** and Kubernetes garbage collection will clean up all old resources.

13.4.1. Reverting QuayEcosystem Upgrade

If something goes wrong during the automatic upgrade from **QuayEcosystem** to **QuayRegistry**, follow these steps to revert back to using the **QuayEcosystem**:

1. Delete the **QuayRegistry** using either the UI or **kubectl**:

```
$ kubectl delete -n <namespace> quayregistry <quayecosystem-name>
```

2. If external access was provided using a **Route**, change the **Route** to point back to the original **Service** using the UI or **kubectl**.

NOTE

If your **QuayEcosystem** was managing the Postgres database, the upgrade process will migrate your data to a new Postgres database managed by the upgraded Operator. Your old database will not be changed or removed but Quay will no longer use it once the migration is complete. If there are issues during the data migration, the upgrade process will exit and it is recommended that you continue with your database as an unmanaged component.

13.4.2. Supported QuayEcosystem Configurations for Upgrades

The Quay Operator will report errors in its logs and in **status.conditions** if migrating a **QuayEcosystem** component fails or is unsupported. All unmanaged components should migrate successfully because no Kubernetes resources need to be adopted and all the necessary values are already provided in Quay's **config.yaml**.

Database

Ephemeral database not supported (**volumeSize** field must be set).

Redis

Nothing special needed.

External Access

Only passthrough **Route** access is supported for automatic migration. Manual migration required for other methods.

- **LoadBalancer** without custom hostname: After the **QuayEcosystem** is marked with label **"quay-operator/migration-complete": "true"**, delete the **metadata.ownerReferences** field from existing **Service** *before* deleting the **QuayEcosystem** to prevent Kubernetes from garbage collecting the **Service** and removing the load balancer. A new **Service** will be created with **metadata.name** format **<QuayEcosystem-name>-quay-app**. Edit the **spec.selector** of the existing **Service** to match the **spec.selector** of the new **Service** so traffic to the old load balancer endpoint will now be directed to the new pods. You are now responsible for the old **Service**; the Quay Operator will not manage it.

- **LoadBalancer**/**NodePort**/**Ingress** with custom hostname: A new **Service** of type **LoadBalancer** will be created with **metadata.name** format **<QuayEcosystem-name>-quay-app**. Change your DNS settings to point to the **status.loadBalancer** endpoint provided by the new **Service**.

Clair

Nothing special needed.

Object Storage

QuayEcosystem did not have a managed object storage component, so object storage will always be marked as unmanaged. Local storage is not supported.

Repository Mirroring

Nothing special needed.

ADDITIONAL RESOURCES

- For more details on the Red Hat Quay Operator, see the upstream quay-operator project.