Red Hat Quay 3.11

Red Hat Quay Operator features

Advanced Red Hat Quay Operator features

Last Updated: 2024-06-04

Red Hat Quay 3.11 Red Hat Quay Operator features

Advanced Red Hat Quay Operator features

Legal Notice

Abstract

Advanced Red Hat Quay Operator features

Table of Contents

CHAPTER 1. FEDERAL INFORMATION PROCESSING STANDARD (FIPS) READINESS AND COMPLIANCE

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

1.1. ENABLING FIPS COMPLIANCE

Use the following procedure to enable FIPS compliance on your Red Hat Quay deployment.

Prerequisite

- If you are running a standalone deployment of Red Hat Quay, your Red Hat Enterprise Linux (RHEL) deployment is version 8 or later and FIPS-enabled.
- If you are deploying Red Hat Quay on OpenShift Container Platform, OpenShift Container Platform is version 4.10 or later.
- Your Red Hat Quay version is 3.5.0 or later.
- If you are using the Red Hat Quay on OpenShift Container Platform on an IBM Power or IBM Z cluster:
  - OpenShift Container Platform version 4.14 or later is required
  - Red Hat Quay version 3.10 or later is required
- You have administrative privileges for your Red Hat Quay deployment.

Procedure

- In your Red Hat Quay **config.yaml** file, set the **FEATURE_FIPS** configuration field to **true**. For example:

  ```
  ---
  FEATURE_FIPS = true
  ---
  ```

  With **FEATURE_FIPS** set to **true**, Red Hat Quay runs using FIPS-compliant hash functions.

CHAPTER 2. CONSOLE MONITORING AND ALERTING

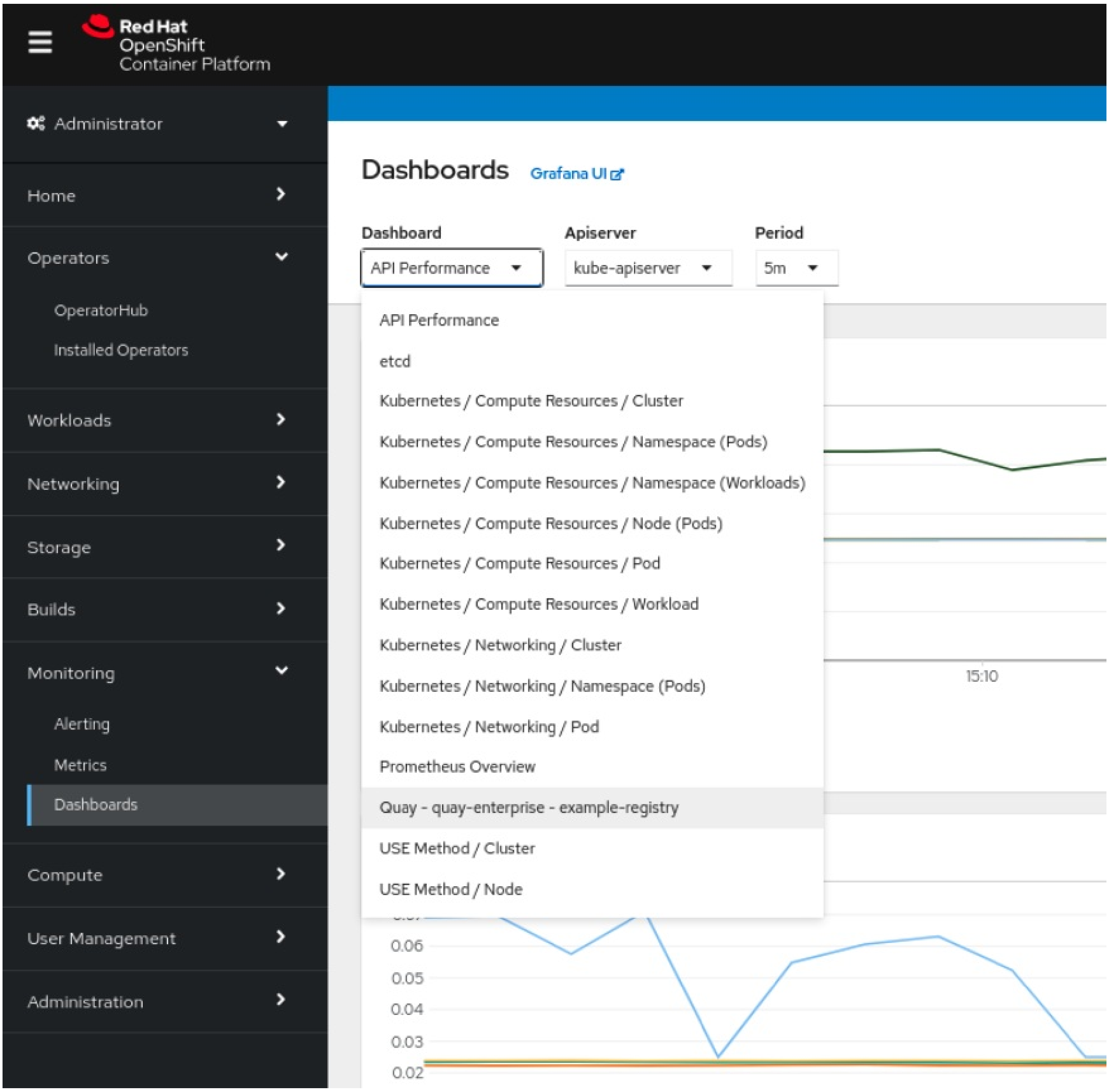
Red Hat Quay provides support for monitoring instances that were deployed by using the Red Hat Quay Operator, from inside the OpenShift Container Platform console. The new monitoring features include a Grafana dashboard, access to individual metrics, and alerting to notify for frequently restarting **Quay**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

NOTE

To enable the monitoring features, the Red Hat Quay Operator must be installed in **All Namespaces** mode.

2.1. DASHBOARD

On the OpenShift Container Platform console, click **Monitoring → Dashboards** and search for the dashboard of your desired Red Hat Quay registry instance:



Dashboard: API Performance

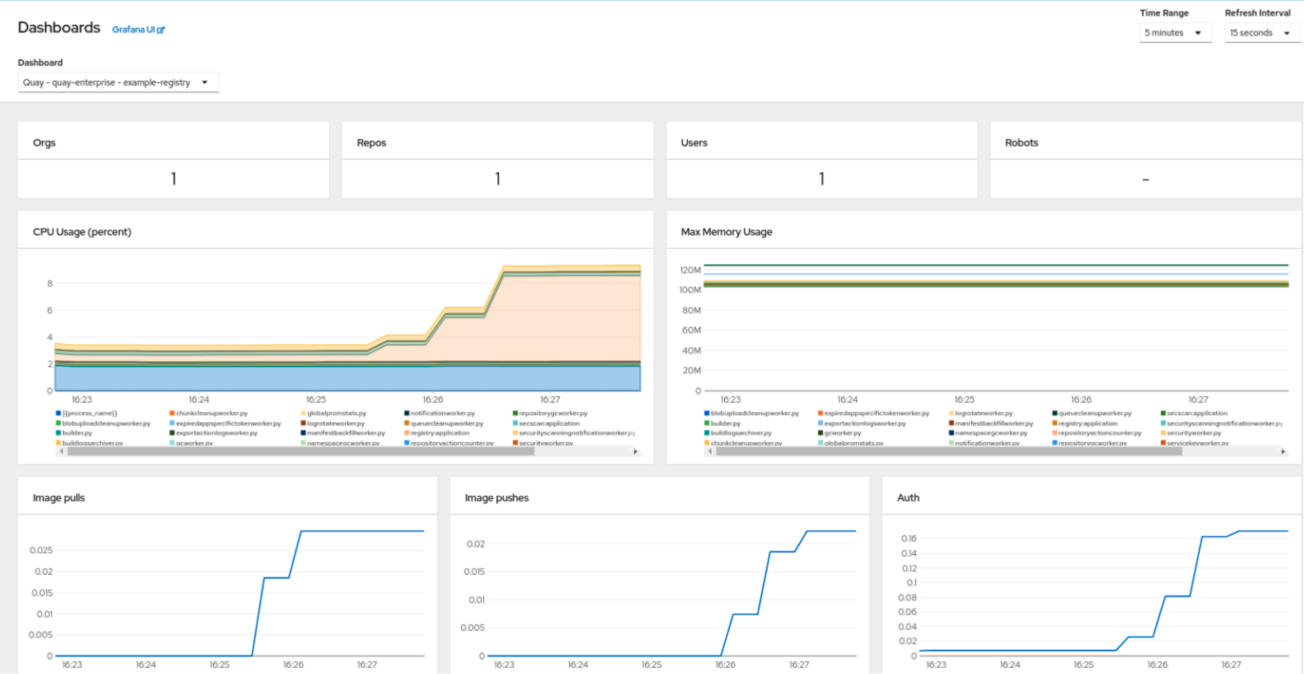
Apiserver: kube-apiserver

Period: 5m

Red Hat
OpenShift
Container Platform
Administrator
Home
Operators
OperatorHub
Installed Operators
Workloads
Networking
Storage
Builds
Monitoring
Alerting
Metrics
Dashboards
Compute
User Management
Administration
Dashboards
Grafana UI
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
15:10
0.06
0.05
0.04
0.03
0.02

The dashboard shows various statistics including the following:

- The number of **Organizations**, **Repositories**, **Users**, and **Robot accounts**
- CPU Usage
- Max memory usage
- Rates of pulls and pushes, and authentication requests
- API request rate
- Latencies


Dashboards

Time Range: 5 minutes

Refresh Interval: 15 seconds

Dashboard: Quay - quay-enterprise - example-registry

Orgs: 1

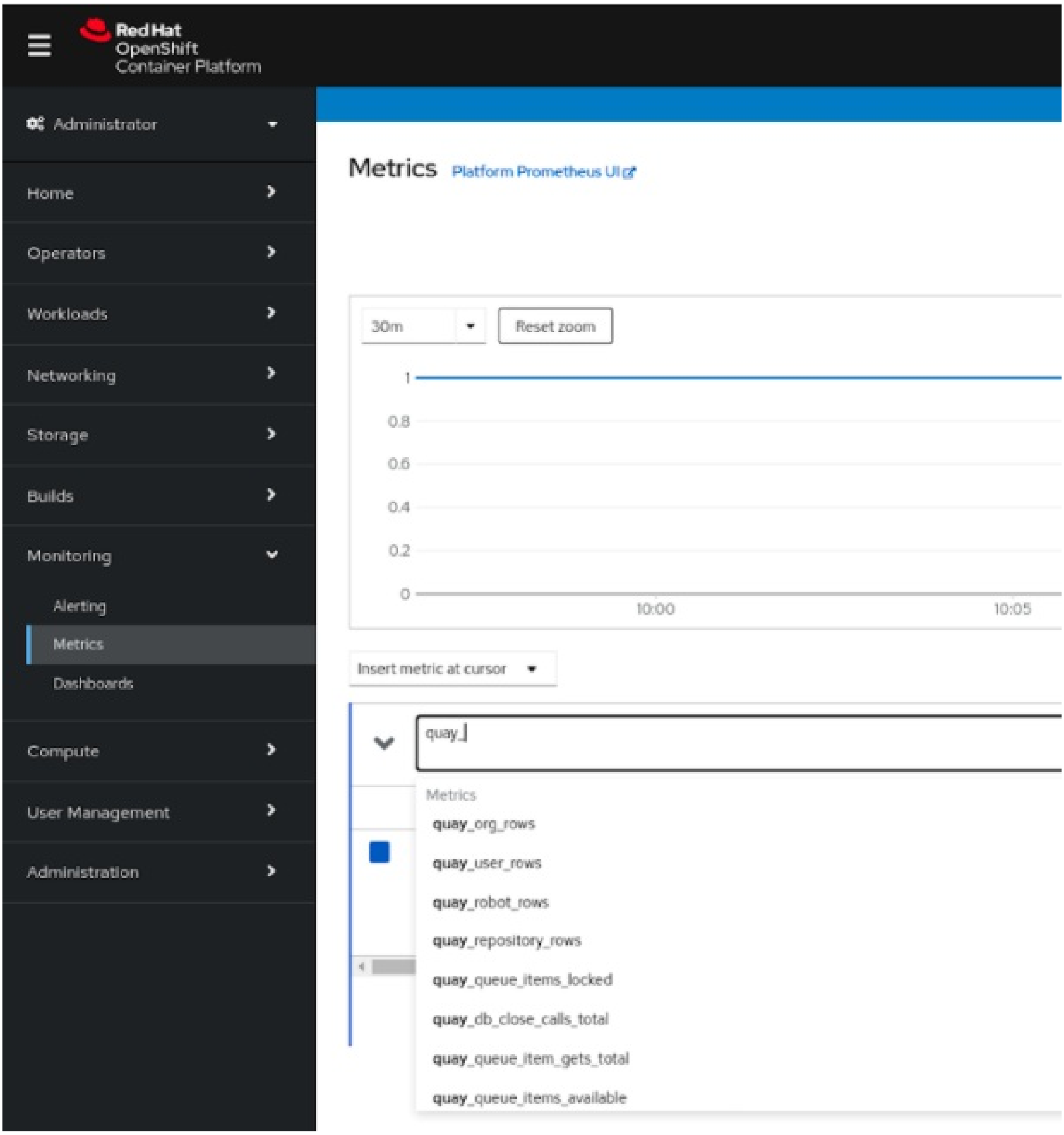
Repos: 1

Users: 1

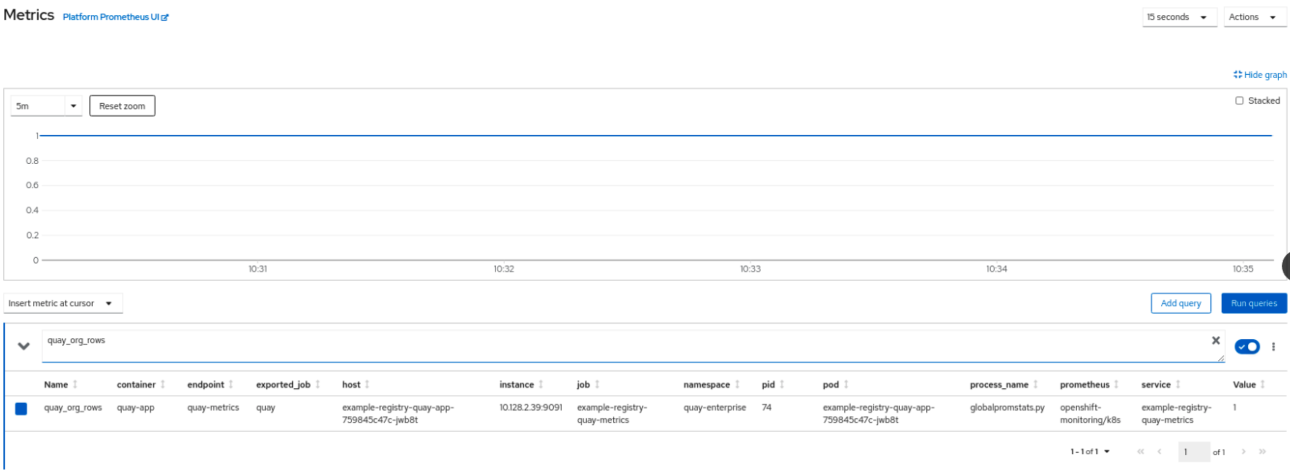
Robots: -

2.2. METRICS

You can see the underlying metrics behind the Red Hat 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**:



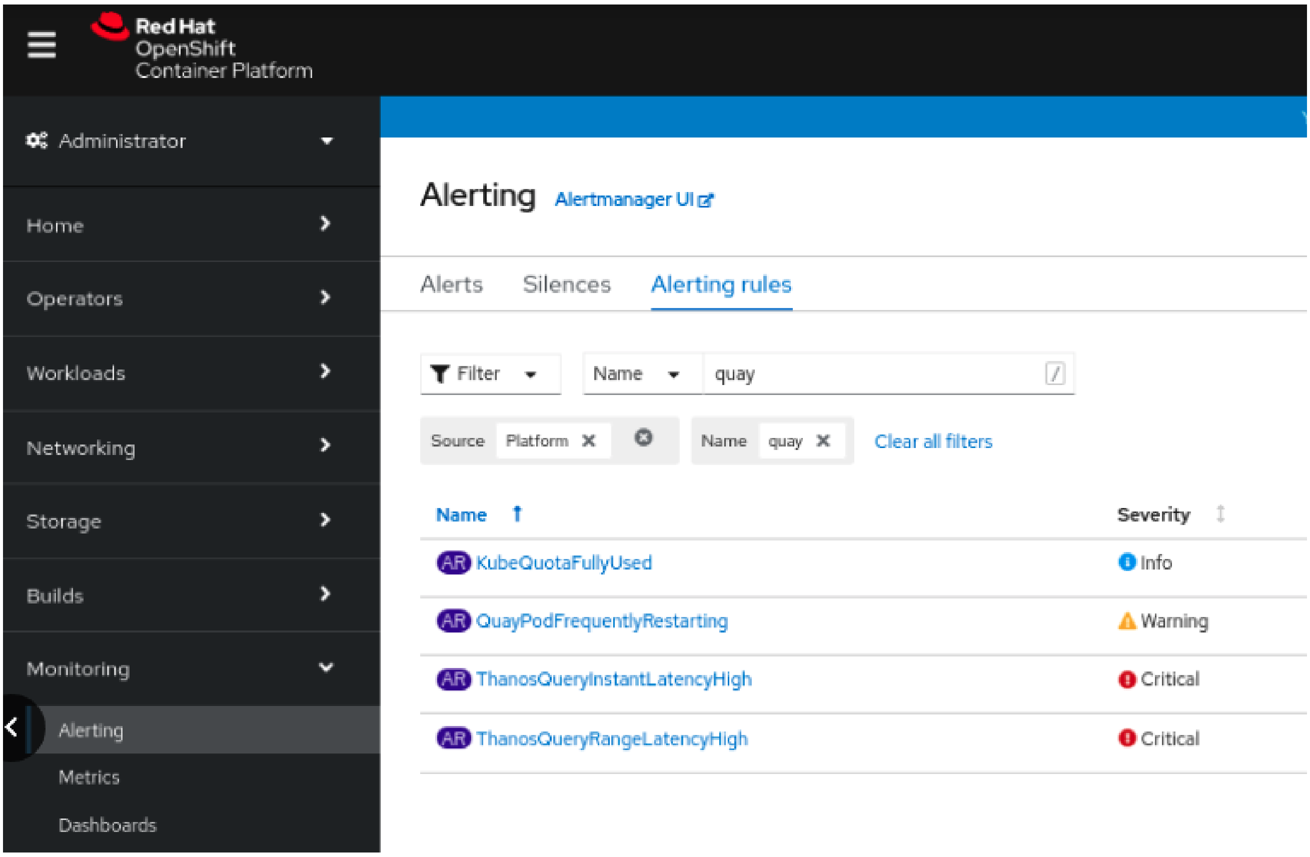
quay_org_rows

	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. It is also directly surfaced in the dashboard.

2.3. 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 console UI and searching for the Quay-specific alert:

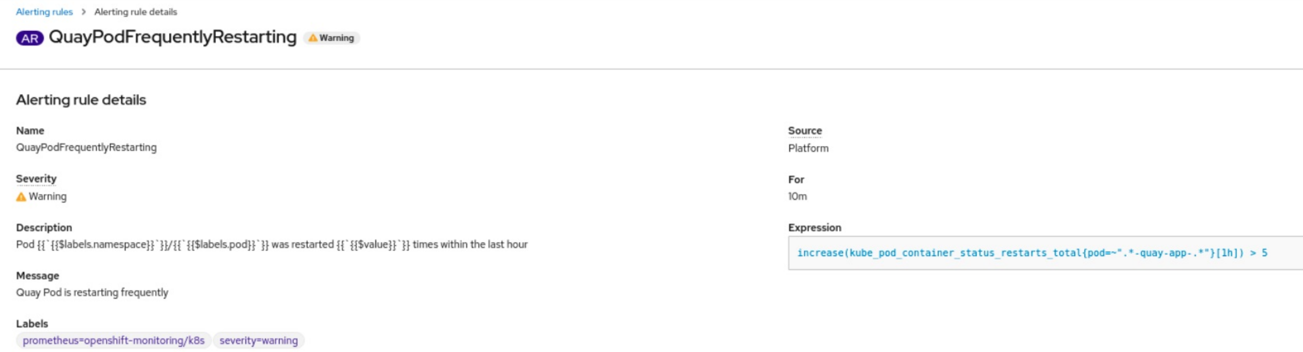


Name: quay

Source: Platform ✕ | Name: quay ✕

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

Select the **QuayPodFrequentlyRestarting** rule detail to configure the alert:



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
```

CHAPTER 3. CLAIR SECURITY SCANNER

3.1. CLAIR VULNERABILITY DATABASES

Clair uses the following vulnerability databases to report for issues in your images:

- Ubuntu Oval database
- Debian Security Tracker
- Red Hat Enterprise Linux (RHEL) Oval database
- SUSE Oval database
- Oracle Oval database
- Alpine SecDB database
- VMware Photon OS database
- Amazon Web Services (AWS) UpdateInfo
- Open Source Vulnerability (OSV) Database

For information about how Clair does security mapping with the different databases, see Claircore Severity Mapping.

3.1.1. Information about Open Source Vulnerability (OSV) database for Clair

Open Source Vulnerability (OSV) is a vulnerability database and monitoring service that focuses on tracking and managing security vulnerabilities in open source software.

OSV provides a comprehensive and up-to-date database of known security vulnerabilities in open source projects. It covers a wide range of open source software, including libraries, frameworks, and other components that are used in software development. For a full list of included ecosystems, see defined ecosystems.

Clair also reports vulnerability and security information for **golang**, **java**, and **ruby** ecosystems through the Open Source Vulnerability (OSV) database.

By leveraging OSV, developers and organizations can proactively monitor and address security vulnerabilities in open source components that they use, which helps to reduce the risk of security breaches and data compromises in projects.

For more information about OSV, see the OSV website.

3.2. CLAIR ON OPENSHIFT CONTAINER PLATFORM

To set up Clair v4 (Clair) on a Red Hat Quay deployment on OpenShift Container Platform, it is recommended to use the Red Hat Quay Operator. By default, the Red Hat Quay Operator installs or upgrades a Clair deployment along with your Red Hat Quay deployment and configure Clair automatically.

3.3. TESTING CLAIR

Use the following procedure to test Clair on either a standalone Red Hat Quay deployment, or on an OpenShift Container Platform Operator-based deployment.

Prerequisites

- You have deployed the Clair container image.

Procedure

1. Pull a sample image by entering the following command:

```
$ podman pull ubuntu:20.04
```

2. Tag the image to your registry by entering the following command:

```
$ sudo podman tag docker.io/library/ubuntu:20.04 <quay-server.example.com>/<user-
name>/ubuntu:20.04
```

3. Push the image to your Red Hat Quay registry by entering the following command:

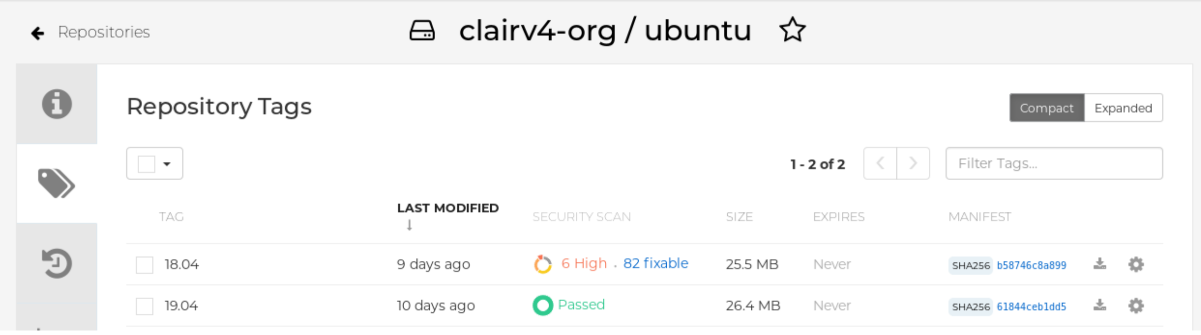
```
$ sudo podman push --tls-verify=false quay-server.example.com/quayadmin/ubuntu:20.04
```

4. Log in to your Red Hat Quay deployment through the UI.

5. Click the repository name, for example, **quayadmin/ubuntu**.

6. In the navigation pane, click **Tags**.

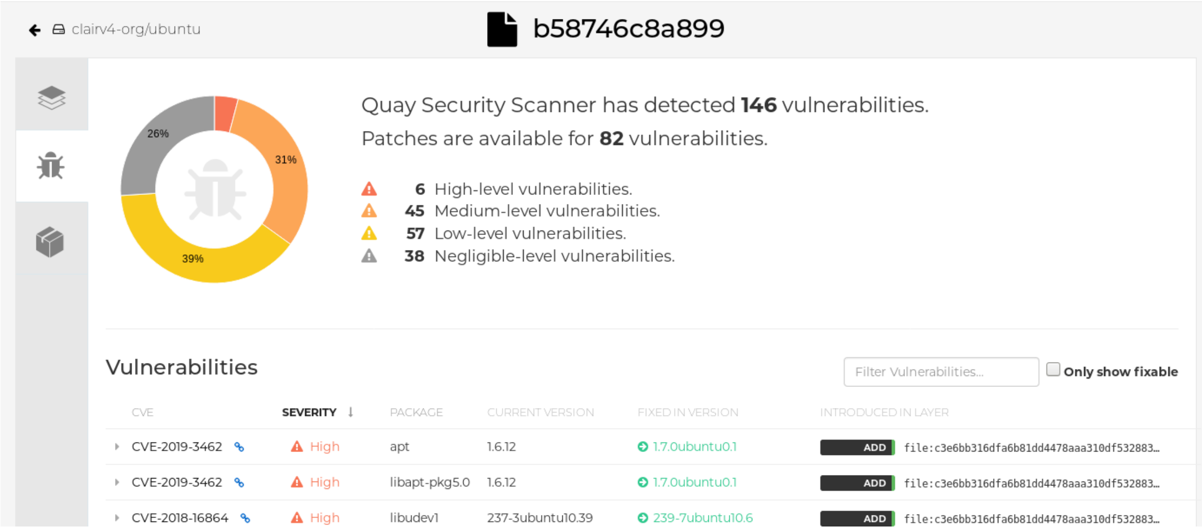
Report summary



TAG	LAST MODIFIED	SECURITY SCAN	SIZE	EXPIRES	MANIFEST
□ 18.04	9 days ago	6 High · 82 fixable	25.5 MB	Never	SHA256 b58746c8a899
□ 19.04	10 days ago	Passed	26.4 MB	Never	SHA256 61844ceb1dd5

7. Click the image report, for example, **45 medium**, to show a more detailed report:

Report details



b58746c8a899

Quay Security Scanner has detected **146** vulnerabilities.
Patches are available for **82** vulnerabilities.

6 High-level vulnerabilities.
45 Medium-level vulnerabilities.
57 Low-level vulnerabilities.
38 Negligible-level vulnerabilities.

Vulnerabilities

Filter Vulnerabilities... □ **Only show fixable**

CVE	SEVERITY ↓	PACKAGE	CURRENT VERSION	FIXED IN VERSION	INTRODUCED IN LAYER
CVE-2019-3462	High	apt	1.6.12	1.7.0ubuntu0.1	ADD file:c3e6bb316dfa6b81dd4478aaa310df532883…
CVE-2019-3462	High	libapt-pkg5.0	1.6.12	1.7.0ubuntu0.1	ADD file:c3e6bb316dfa6b81dd4478aaa310df532883…
CVE-2018-16864	High	libudev1	237-3ubuntu10.39	239-7ubuntu10.6	ADD file:c3e6bb316dfa6b81dd4478aaa310df532883…

NOTE

In some cases, Clair shows duplicate reports on images, for example, **ubi8/nodejs-12** or **ubi8/nodejs-16**. This occurs because vulnerabilities with same name are for different packages. This behavior is expected with Clair vulnerability reporting and will not be addressed as a bug.

3.4. ADVANCED CLAIR CONFIGURATION

Use the procedures in the following sections to configure advanced Clair settings.

3.4.1. Unmanaged Clair configuration

Red Hat Quay users can run an unmanaged Clair configuration with 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.

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.

3.4.1.1. Running a custom Clair configuration with an unmanaged Clair database

Use the following procedure to set your Clair database to unmanaged.

Procedure

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

```
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
```

3.4.1.2. Configuring a custom Clair database with an unmanaged Clair database

Red Hat Quay on OpenShift Container Platform allows users to provide their own Clair database.

Use the following procedure to create a custom Clair database.

NOTE

The following procedure sets up Clair with SSL/TLS certifications. To view a similar procedure that does not set up Clair with SSL/TSL certifications, see "Configuring a custom Clair database with a managed Clair configuration".

Procedure

1. Create a Quay configuration bundle secret that includes the **clair-config.yaml** by entering the following command:

```
$ 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 file

```
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** file to your bundle secret, for example:

```
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>
  ssl.crt: <base64 encoded SSL certificate>
  ssl.key: <base64 encoded SSL private key>
```

NOTE

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

3. You can check the status of your Clair pod by clicking the commit in the **Build History** page, or by running **oc get pods -n <namespace>**. For example:

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

Example output

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

3.4.2. Running a custom Clair configuration with a managed Clair database

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

- When a user wants to disable specific updater resources.
- When a user is running Red Hat Quay in an disconnected environment. For more information about running Clair in a disconnected environment, see Clair in disconnected environments.

NOTE

- If you are running Red Hat Quay in an disconnected environment, the **airgap** parameter of your **clair-config.yaml** must be set to **true**.
- If you are running Red Hat Quay in an disconnected environment, you should disable all updater components.

3.4.2.1. Setting a Clair database to managed

Use the following procedure to set your Clair database to managed.

Procedure

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

```
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: true
```

3.4.2.2. Configuring a custom Clair database with a managed Clair configuration

Red Hat Quay on OpenShift Container Platform allows users to provide their own Clair database.

Use the following procedure to create a custom Clair database.

Procedure

1. Create a Quay configuration bundle secret that includes the **clair-config.yaml** by entering the following command:

```
$ 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 config-bundle-secret
```

Example Clair config.yaml file

```
indexer:
    connstring: host=quay-server.example.com port=5432 dbname=quay user=quayrdsdb password=quayrdsdb sslmode=disable
    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 sslmode=disable
    migrations: true
metrics:
    name: prometheus
notifier:
    connstring: host=quay-server.example.com port=5432 dbname=quay user=quayrdsdb
password=quayrdsdb sslmode=disable
    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** file to your bundle secret, for example:

```
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>
```

NOTE

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

3. You can check the status of your Clair pod by clicking the commit in the **Build History** page, or by running **oc get pods -n <namespace>**. For example:

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

Example output

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

3.4.3. Clair in disconnected environments

NOTE

Currently, deploying Clair in disconnected environments is not supported on IBM Power and IBM Z.

Clair uses a set of components called *updaters* to handle the fetching and parsing of data from various vulnerability databases. Updaters are set up by default to pull vulnerability data directly from the internet and work for immediate use. However, some users might require Red Hat Quay to run in a disconnected environment, or an environment without direct access to the internet. Clair supports disconnected environments by working with different types of update workflows that take network isolation into consideration. This works by using the **clairctl** command line interface tool, which obtains updater data from the internet by using an open host, securely transferring the data to an isolated host, and then important the updater data on the isolated host into Clair.

Use this guide to deploy Clair in a disconnected environment.

IMPORTANT

Due to known issue PROJQUAY-6577, the Red Hat Quay Operator does not properly render customized Clair **config.yaml** files. As a result, the following procedure does not currently work.

Users must create the entire Clair configuration themselves, from the beginning, instead of relying on the Operator to populate the fields. To do this, following the instructions at Procedure to enable Clair scanning of images in disconnected environments .

NOTE

Currently, Clair enrichment data is CVSS data. Enrichment data is currently unsupported in disconnected environments.

For more information about Clair updaters, see "Clair updaters".

3.4.3.1. Setting up Clair in a disconnected OpenShift Container Platform cluster

Use the following procedures to set up an OpenShift Container Platform provisioned Clair pod in a disconnected OpenShift Container Platform cluster.

IMPORTANT

Due to known issue PROJQUAY-6577, the Red Hat Quay Operator does not properly render customized Clair **config.yaml** files. As a result, the following procedure does not currently work.

Users must create the entire Clair configuration themselves, from the beginning, instead of relying on the Operator to populate the fields. To do this, following the instructions at Procedure to enable Clair scanning of images in disconnected environments .

3.4.3.1.1. Installing the clairctl command line utility tool for OpenShift Container Platform deployments

Use the following procedure to install the **clairctl** CLI tool for OpenShift Container Platform deployments.

Procedure

1. Install the **clairctl** program for a Clair deployment in an OpenShift Container Platform cluster by entering the following command:

```
$ oc -n quay-enterprise exec example-registry-clair-app-64dd48f866-6ptgw -- cat
/usr/bin/clairctl > clairctl
```

> **NOTE**
>
> Unofficially, the **clairctl** tool can be downloaded

2. Set the permissions of the **clairctl** file so that it can be executed and run by the user, for example:

```
$ chmod u+x ./clairctl
```

3.4.3.1.2. Retrieving and decoding the Clair configuration secret for Clair deployments on OpenShift Container Platform

Use the following procedure to retrieve and decode the configuration secret for an OpenShift Container Platform provisioned Clair instance on OpenShift Container Platform.

Prerequisites

- You have installed the **clairctl** command line utility tool.

Procedure

1. Enter the following command to retrieve and decode the configuration secret, and then save it to a Clair configuration YAML:

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

2. Update the **clair-config.yaml** file so that the **disable_updaters** and **airgap** parameters are set to **true**, for example:

```
---
indexer:
  airgap: true
---
matcher:
  disable_updaters: true
---
```

3.4.3.1.3. Exporting the updaters bundle from a connected Clair instance

Use the following procedure to export the updaters bundle from a Clair instance that has access to the internet.

Prerequisites

- You have installed the **clairctl** command line utility tool.
- You have retrieved and decoded the Clair configuration secret, and saved it to a Clair **config.yaml** file.
- The **disable_updaters** and **airgap** parameters are set to **true** in your Clair **config.yaml** file.

Procedure

- From a Clair instance that has access to the internet, use the **clairctl** CLI tool with your configuration file to export the updaters bundle. For example:

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

3.4.3.1.4. Configuring access to the Clair database in the disconnected OpenShift Container Platform cluster

Use the following procedure to configure access to the Clair database in your disconnected OpenShift Container Platform cluster.

Prerequisites

- You have installed the **clairctl** command line utility tool.
- You have retrieved and decoded the Clair configuration secret, and saved it to a Clair **config.yaml** file.
- The **disable_updaters** and **airgap** parameters are set to **true** in your Clair **config.yaml** file.
- You have exported the updaters bundle from a Clair instance that has access to the internet.

Procedure

1. Determine your Clair database service by using the **oc** CLI tool, for example:

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

Example output

```
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
...
```

2. Forward the Clair database port so that it is accessible from the local machine. For example:

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

3. Update your Clair **config.yaml** file, for example:

```
indexer:
  connstring: host=localhost port=5432 dbname=postgres user=postgres password=postgres sslmode=disable ①
  scanlock_retry: 10
  layer_scan_concurrency: 5
  migrations: true
  scanner:
    repo:
      rhel-repository-scanner: ②
        repo2cpe_mapping_file: /data/cpe-map.json
    package:
      rhel_containerscanner: ③
        name2repos_mapping_file: /data/repo-map.json
```

① Replace the value of the **host** in the multiple **connstring** fields with **localhost**.

② For more information about the **rhel-repository-scanner** parameter, see "Mapping repositories to Common Product Enumeration information".

③ For more information about the **rhel_containerscanner** parameter, see "Mapping repositories to Common Product Enumeration information".

3.4.3.1.5. Importing the updaters bundle into the disconnected OpenShift Container Platform cluster

Use the following procedure to import the updaters bundle into your disconnected OpenShift Container Platform cluster.

Prerequisites

- You have installed the **clairctl** command line utility tool.
- You have retrieved and decoded the Clair configuration secret, and saved it to a Clair **config.yaml** file.
- The **disable_updaters** and **airgap** parameters are set to **true** in your Clair **config.yaml** file.
- You have exported the updaters bundle from a Clair instance that has access to the internet.
- You have transferred the updaters bundle into your disconnected environment.

Procedure

- Use the **clairctl** CLI tool to import the updaters bundle into the Clair database that is deployed by OpenShift Container Platform. For example:

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

3.4.3.2. Setting up a self-managed deployment of Clair for a disconnected OpenShift Container Platform cluster

Use the following procedures to set up a self-managed deployment of Clair for a disconnected OpenShift Container Platform cluster.

IMPORTANT

Due to known issue PROJQUAY-6577, the Red Hat Quay Operator does not properly render customized Clair **config.yaml** files. As a result, the following procedure does not currently work.

Users must create the entire Clair configuration themselves, from the beginning, instead of relying on the Operator to populate the fields. To do this, following the instructions at Procedure to enable Clair scanning of images in disconnected environments .

3.4.3.2.1. Installing the clairctl command line utility tool for a self-managed Clair deployment on OpenShift Container Platform

Use the following procedure to install the **clairctl** CLI tool for self-managed Clair deployments on OpenShift Container Platform.

Procedure

1. Install the **clairctl** program for a self-managed Clair deployment by using the **podman cp** command, for example:

```
$ sudo podman cp clairv4:/usr/bin/clairctl ./clairctl
```

2. Set the permissions of the **clairctl** file so that it can be executed and run by the user, for example:

```
$ chmod u+x ./clairctl
```

3.4.3.2.2. Deploying a self-managed Clair container for disconnected OpenShift Container Platform clusters

Use the following procedure to deploy a self-managed Clair container for disconnected OpenShift Container Platform clusters.

Prerequisites

- You have installed the **clairctl** command line utility tool.

Procedure

1. Create a folder for your Clair configuration file, for example:

```
$ mkdir /etc/clairv4/config/
```

2. Create a Clair configuration file with the **disable_updaters** parameter set to **true**, for example:

```
---
indexer:
  airgap: true
---
matcher:
  disable_updaters: true
---
```

3. Start Clair by using the container image, mounting in the configuration from the file you created:

```
$ sudo podman run -it --rm --name clairv4 \
-p 8081:8081 -p 8088:8088 \
-e CLAIR_CONF=/clair/config.yaml \
-e CLAIR_MODE=combo \
-v /etc/clairv4/config:/clair:Z \
registry.redhat.io/quay/clair-rhel8:v3.11.1
```

3.4.3.2.3. Exporting the updaters bundle from a connected Clair instance

Use the following procedure to export the updaters bundle from a Clair instance that has access to the internet.

Prerequisites

- You have installed the **clairctl** command line utility tool.
- You have deployed Clair.
- The **disable_updaters** and **airgap** parameters are set to **true** in your Clair **config.yaml** file.

Procedure

- From a Clair instance that has access to the internet, use the **clairctl** CLI tool with your configuration file to export the updaters bundle. For example:

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

3.4.3.2.4. Configuring access to the Clair database in the disconnected OpenShift Container Platform cluster

Use the following procedure to configure access to the Clair database in your disconnected OpenShift Container Platform cluster.

Prerequisites

- You have installed the **clairctl** command line utility tool.
- You have deployed Clair.
- The **disable_updaters** and **airgap** parameters are set to **true** in your Clair **config.yaml** file.
- You have exported the updaters bundle from a Clair instance that has access to the internet.

Procedure

1. Determine your Clair database service by using the **oc** CLI tool, for example:

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

Example output

```
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
...
```

2. Forward the Clair database port so that it is accessible from the local machine. For example:

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

3. Update your Clair **config.yaml** file, for example:

```
indexer:
    connstring: host=localhost port=5432 dbname=postgres user=postgres
password=postgres sslmode=disable (1)
    scanlock_retry: 10
    layer_scan_concurrency: 5
    migrations: true
    scanner:
      repo:
        rhel-repository-scanner: (2)
          repo2cpe_mapping_file: /data/cpe-map.json
      package:
        rhel_containerscanner: (3)
          name2repos_mapping_file: /data/repo-map.json
```

1. Replace the value of the **host** in the multiple **connstring** fields with **localhost**.

2. For more information about the **rhel-repository-scanner** parameter, see "Mapping repositories to Common Product Enumeration information".

3. For more information about the **rhel_containerscanner** parameter, see "Mapping repositories to Common Product Enumeration information".

3.4.3.2.5. Importing the updaters bundle into the disconnected OpenShift Container Platform cluster

Use the following procedure to import the updaters bundle into your disconnected OpenShift Container Platform cluster.

Prerequisites

- You have installed the **clairctl** command line utility tool.
- You have deployed Clair.
- The **disable_updaters** and **airgap** parameters are set to **true** in your Clair **config.yaml** file.
- You have exported the updaters bundle from a Clair instance that has access to the internet.
- You have transferred the updaters bundle into your disconnected environment.

Procedure

- Use the **clairctl** CLI tool to import the updaters bundle into the Clair database that is deployed by OpenShift Container Platform:

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

3.4.4. Mapping repositories to Common Product Enumeration information

NOTE

Currently, mapping repositories to Common Product Enumeration information is not supported on IBM Power and IBM Z.

Clair's Red Hat Enterprise Linux (RHEL) scanner relies on a Common Product Enumeration (CPE) file to map RPM packages to the corresponding security data to produce matching results. These files are owned by product security and updated daily.

The CPE file must be present, or access to the file must be allowed, for the scanner to properly process RPM packages. If the file is not present, RPM packages installed in the container image will not be scanned.

Table 3.1. Clair CPE mapping files

CPE	Link to JSON mapping file
repos2cpe	Red Hat Repository-to-CPE JSON
names2repos	Red Hat Name-to-Repos JSON.

In addition to uploading CVE information to the database for disconnected Clair installations, you must also make the mapping file available locally:

- For standalone Red Hat Quay and Clair deployments, the mapping file must be loaded into the Clair pod.
- For Red Hat Quay on OpenShift Container Platform deployments, you must set the Clair component to **unmanaged**. Then, Clair must be deployed manually, setting the configuration to load a local copy of the mapping file.

3.4.4.1. Mapping repositories to Common Product Enumeration example configuration

Use the **repo2cpe_mapping_file** and **name2repos_mapping_file** fields in your Clair configuration to include the CPE JSON mapping files. For example:

```
indexer:
  scanner:
    repo:
      rhel-repository-scanner:
        repo2cpe_mapping_file: /data/cpe-map.json
```

```
package:
  rhel_containerscanner:
    name2repos_mapping_file: /data/repo-map.json
```

For more information, see How to accurately match OVAL security data to installed RPMs .

CHAPTER 4. DEPLOYING RED HAT QUAY ON INFRASTRUCTURE NODES

By default, **Quay** related pods are placed on arbitrary worker nodes when using the Red Hat Quay Operator to deploy the registry. For more information about how to use machine sets to configure nodes to only host infrastructure components, see Creating infrastructure machine sets.

If you are not using OpenShift Container Platform machine set resources to deploy infra nodes, the section in this document shows you how to manually label and taint nodes for infrastructure purposes. After you have configured your infrastructure nodes either manually or use machines sets, you can control the placement of **Quay** pods on these nodes using node selectors and tolerations.

4.1. LABELING AND TAINTING NODES FOR INFRASTRUCTURE USE

Use the following procedure to label and tain nodes for infrastructure use.

1. Enter the following command to reveal the master and worker nodes. In this example, there are three master nodes and six worker nodes.

```
$ oc get nodes
```

Example output

```
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
```

2. Enter the following commands to label the 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=
```

3. Now, when listing the nodes in the cluster, the last three worker nodes have the **infra** role. For example:

```
$ oc get nodes
```

Example

```
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+ba4558
```

4. When a worker node is assigned the **infra** role, 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 that you want to control. For example:

```
$ 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
```

4.2. CREATING A PROJECT WITH NODE SELECTOR AND TOLERATIONS

Use the following procedure to create a project with node selector and tolerations.

NOTE

The following procedure can also be completed by removing the installed Red Hat Quay Operator and the namespace, or namespaces, used when creating the deployment. Users can then create a new resource with the following annotation.

Procedure

1. Enter the following command to edit the namespace where Red Hat Quay is deployed, and the following annotation:

```
$ oc annotate namespace <namespace> openshift.io/node-selector='node-
role.kubernetes.io/infra='
```

Example output

```
namespace/<namespace> annotated
```

2. Obtain a list of available pods by entering the following command:

```
$ oc get pods -o wide
```

Example output

```
NAME                                          READY   STATUS      RESTARTS        AGE     IP
NODE                                    NOMINATED NODE   READINESS GATES
example-registry-clair-app-5744dd64c9-9d5jt        1/1     Running     0               173m
10.130.4.13   stevsmit-quay-ocp-tes-5gwws-worker-c-6xkn7   <none>           <none>
example-registry-clair-app-5744dd64c9-fg86n        1/1     Running     6 (3h21m ago)   3h24m
10.131.0.91   stevsmit-quay-ocp-tes-5gwws-worker-c-dnhdp   <none>           <none>
example-registry-clair-postgres-845b47cd88-vdchz   1/1     Running     0               3h21m
10.130.4.10   stevsmit-quay-ocp-tes-5gwws-worker-c-6xkn7   <none>           <none>
example-registry-quay-app-64cbc5bcf-8zvgc          1/1     Running     1 (3h24m ago)   3h24m
10.130.2.12   stevsmit-quay-ocp-tes-5gwws-worker-a-tk8dx   <none>           <none>
example-registry-quay-app-64cbc5bcf-pvlz6          1/1     Running     0               3h24m
10.129.4.10   stevsmit-quay-ocp-tes-5gwws-worker-b-fjhz4   <none>           <none>
example-registry-quay-app-upgrade-8gspn            0/1     Completed   0               3h24m
10.130.2.10   stevsmit-quay-ocp-tes-5gwws-worker-a-tk8dx   <none>           <none>
example-registry-quay-database-784d78b6f8-2vkml    1/1     Running     0               3h24m
10.131.4.10   stevsmit-quay-ocp-tes-5gwws-worker-c-2frtg   <none>           <none>
example-registry-quay-mirror-d5874d8dc-fmknp       1/1     Running     0               3h24m
10.129.4.9    stevsmit-quay-ocp-tes-5gwws-worker-b-fjhz4   <none>           <none>
example-registry-quay-mirror-d5874d8dc-t4mff       1/1     Running     0               3h24m
10.129.2.19   stevsmit-quay-ocp-tes-5gwws-worker-a-k7w86   <none>           <none>
example-registry-quay-redis-79848898cb-6qf5x       1/1     Running     0               3h24m
10.130.2.11   stevsmit-quay-ocp-tes-5gwws-worker-a-tk8dx   <none>           <none>
```

3. Enter the following command to delete the available pods:

```
$ oc delete pods --selector quay-operator/quayregistry=example-registry -n quay-enterprise
```

Example output

```
pod "example-registry-clair-app-5744dd64c9-9d5jt" deleted
pod "example-registry-clair-app-5744dd64c9-fg86n" deleted
pod "example-registry-clair-postgres-845b47cd88-vdchz" deleted
pod "example-registry-quay-app-64cbc5bcf-8zvgc" deleted
pod "example-registry-quay-app-64cbc5bcf-pvlz6" deleted
pod "example-registry-quay-app-upgrade-8gspn" deleted
pod "example-registry-quay-database-784d78b6f8-2vkml" deleted
pod "example-registry-quay-mirror-d5874d8dc-fmknp" deleted
pod "example-registry-quay-mirror-d5874d8dc-t4mff" deleted
pod "example-registry-quay-redis-79848898cb-6qf5x" deleted
```

After the pods have been deleted, they automatically cycle back up and should be scheduled on the dedicated infrastructure nodes.

4.3. INSTALLING RED HAT QUAY ON OPENSHIFT CONTAINER PLATFORM ON A SPECIFIC NAMESPACE

Use the following procedure to install Red Hat Quay on OpenShift Container Platform in a specific namespace.

- To install the Red Hat Quay Operator in a specific namespace, you must explicitly specify the appropriate project namespace, as in the following command.
  In the following example, the **quay-registry** namespace is used. This results in the **quay-operator** pod landing on one of the three infrastructure nodes. For example:

  ```
  $ oc get pods -n quay-registry -o wide
  ```

  Example output

  ```
  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
  ```

4.4. CREATING THE RED HAT QUAY REGISTRY

Use the following procedure to create the Red Hat Quay registry.

- Enter the following command to create the Red Hat Quay registry. Then, wait for the deployment to be marked as **ready**. In the following example, you should 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
  ```

  Example output

  ```
  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-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
```

4.5. ENABLING MONITORING WHEN THE RED HAT QUAY OPERATOR IS INSTALLED IN A SINGLE NAMESPACE

NOTE

Currently, enabling monitoring when the Red Hat Quay Operator is installed in a single namespace is not supported on IBM Power and IBM Z.

When the Red Hat Quay Operator is installed in a single namespace, the monitoring component is set to **unmanaged**. To configure monitoring, you must enable it for user-defined namespaces in OpenShift Container Platform.

For more information, see the OpenShift Container Platform documentation for Configuring the monitoring stack and Enabling monitoring for user-defined projects.

The following sections shows you how to enable monitoring for Red Hat Quay based on the OpenShift Container Platform documentation.

4.5.1. Creating a cluster monitoring config map

Use the following procedure check if the **cluster-monitoring-config ConfigMap** object exists.

Procedure

1. Enter the following command to check whether the **cluster-monitoring-config** ConfigMap object exists:

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

Example output

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

2. Optional: If the **ConfigMap** object does not exist, create a YAML manifest. In the following example, the file is called **cluster-monitoring-config.yaml**.

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

3. Optional: If the **ConfigMap** object does not exist, create the **ConfigMap** object:

```
$ oc apply -f cluster-monitoring-config.yaml
```

Example output

```
configmap/cluster-monitoring-config created
```

4. Ensure that the **ConfigMap** object exists by running the following command:

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

Example output

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

4.5.2. Creating a user-defined workload monitoring ConfigMap object

Use the following procedure check if the **user-workload-monitoring-config ConfigMap** object exists.

Procedure

1. Enter the following command to check whether the **user-workload-monitoring-config ConfigMap** object exists:

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

Example output

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

2. If the **ConfigMap** object does not exist, create a YAML manifest. In the following example, the file is called **user-workload-monitoring-config.yaml**.

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

3. Optional: Create the **ConfigMap** object by entering the following command:

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

Example output

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

4.5.3. Enable monitoring for user-defined projects

Use the following procedure to enable monitoring for user-defined projects.

Procedure

1. Enter the following command to check if monitoring for user-defined projects is running:

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

   Example output

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

2. Edit the **cluster-monitoring-config ConfigMap** by entering the following command:

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

3. Set **enableUserWorkload: true** in your **config.yaml** file to enable monitoring for user-defined projects on the cluster:

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

4. Enter the following command to save the file, apply the changes, and ensure that the appropriate pods are running:

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

   Example output

   ```
   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
   ```

4.5.4. Creating a Service object to expose Red Hat Quay metrics

Use the following procedure to create a **Service** object to expose Red Hat Quay metrics.

Procedure

1. Create a YAML file for the Service object:

   ```
   $ cat <<EOF >  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
EOF
```

2. Create the **Service** object by entering the following command:

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

Example output

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

4.5.5. Creating a ServiceMonitor object

Use the following procedure to configure OpenShift Monitoring to scrape the metrics by creating a **ServiceMonitor** resource.

Procedure

1. Create a YAML file for the **ServiceMonitor** resource:

```
$ cat <<EOF >  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
EOF
```

2. Create the **ServiceMonitor** resource by entering the following command:

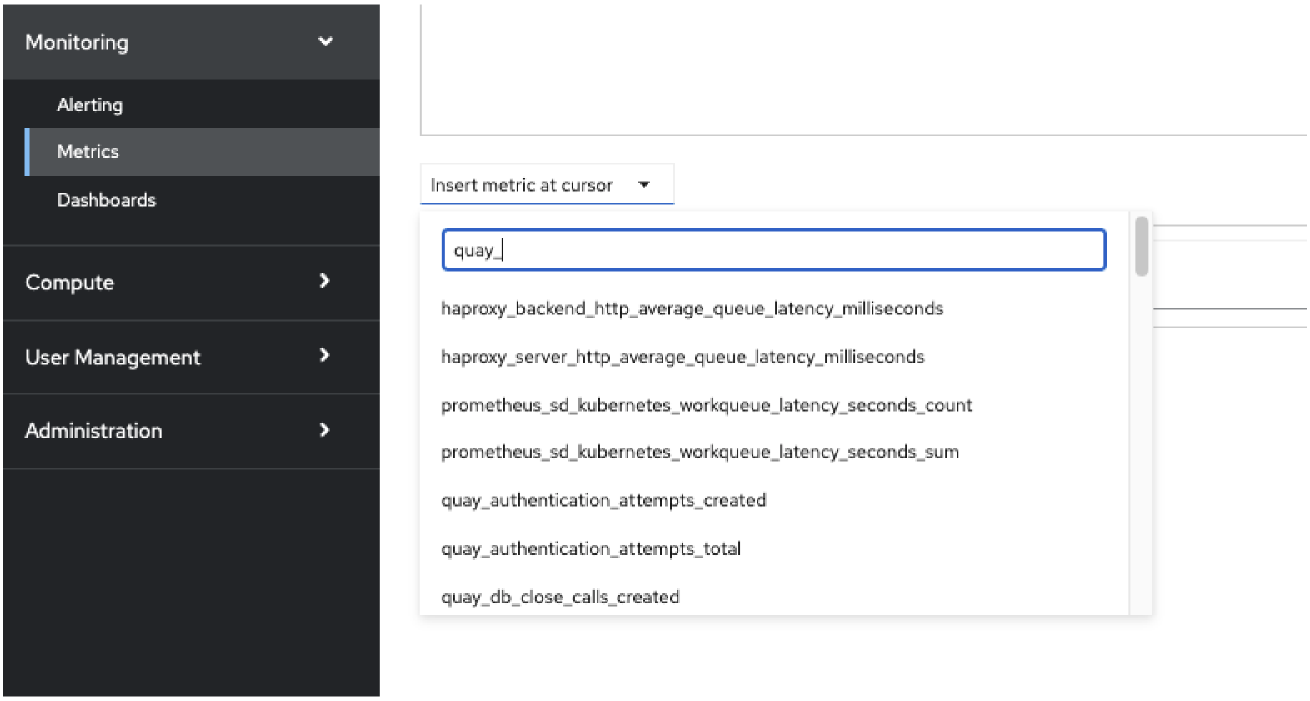
```
$ oc apply -f quay-service-monitor.yaml
```

Example output

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

4.5.6. Viewing metrics in OpenShift Container Platform

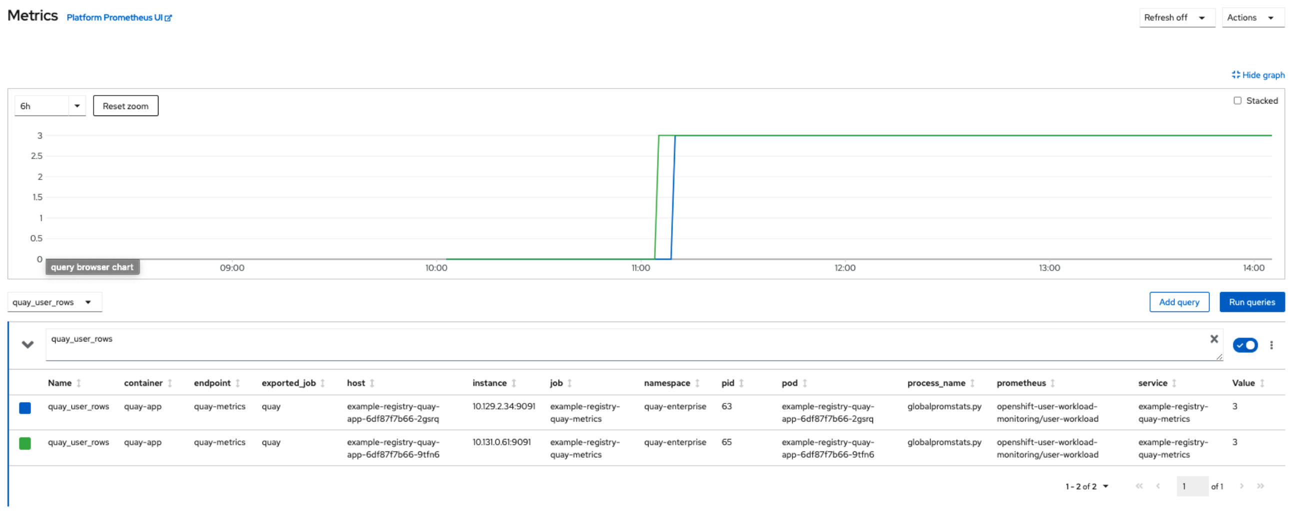
You can access the metrics in the OpenShift Container Platform console under **Monitoring → Metrics**. In the Expression field, enter **quay_** to see the list of metrics available:



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

4.6. RESIZING MANAGED STORAGE

When deploying Red Hat Quay on OpenShift Container Platform, three distinct persistent volume claims (PVCs) are deployed:

- One for the PostgreSQL 13 registry.
- One for the Clair PostgreSQL 13 registry.
- One that uses NooBaa as a backend storage.

NOTE

The connection between Red Hat Quay and NooBaa is done through the S3 API and ObjectBucketClaim API in OpenShift Container Platform. Red Hat Quay leverages that API group to create a bucket in NooBaa, obtain access keys, and automatically set everything up. On the backend, or NooBaa, side, that bucket is creating inside of the backing store. As a result, NooBaa PVCs are not mounted or connected to Red Hat Quay pods.

The default size for the PostgreSQL 13 and Clair PostgreSQL 13 PVCs is set to 50 GiB. You can expand storage for these PVCs on the OpenShift Container Platform console by using the following procedure.

NOTE

The following procedure shares commonality with Expanding Persistent Volume Claims on Red Hat OpenShift Data Foundation.

4.6.1. Resizing PostgreSQL 13 PVCs on Red Hat Quay

Use the following procedure to resize the PostgreSQL 13 and Clair PostgreSQL 13 PVCs.

Prerequisites

- You have cluster admin privileges on OpenShift Container Platform.

Procedure

1. Log into the OpenShift Container Platform console and select **Storage → Persistent Volume Claims**.
2. Select the desired **PersistentVolumeClaim** for either PostgreSQL 13 or Clair PostgreSQL 13, for example, **example-registry-quay-postgres-13**.
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, the expanded size should reflect in the PVC's **Capacity** field.

4.7. CUSTOMIZING DEFAULT OPERATOR IMAGES

NOTE

Currently, customizing default Operator images is not supported on IBM Power and IBM Z.

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

IMPORTANT

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

4.7.1. Environment Variables

The following environment variables are used in the Red Hat Quay 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

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

4.7.2. Applying overrides to a running Operator

When the Red Hat Quay Operator is installed in a cluster through the Operator Lifecycle Manager (OLM), the managed component container images can be easily overridden by modifying the **ClusterServiceVersion** object.

Use the following procedure to apply overrides to a running Red Hat Quay Operator.

Procedure

1. The **ClusterServiceVersion** object is Operator Lifecycle Manager's representation of a running Operator in the cluster. Find the Red Hat Quay Operator's **ClusterServiceVersion** by using a Kubernetes UI or the **kubectl**/**oc** CLI tool. For example:

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

2. Using the UI, **oc edit**, or another method, modify the Red Hat 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:c35f5af964431673f4ff5c9e90bdf45f19e38b8742b5903d41c
10cc7f6339a6d
- name: RELATED_IMAGE_COMPONENT_CLAIR
  value:
quay.io/projectquay/clair@sha256:70c99feceb4c0973540d22e740659cd8d616775d3ad1c169
8ddf71d0221f3ce6
- name: RELATED_IMAGE_COMPONENT_POSTGRES
  value: centos/postgresql-10-
centos7@sha256:de1560cb35e5ec643e7b3a772ebaac8e3a7a2a8e8271d9e91ff023539b4dfb3
3
- name: RELATED_IMAGE_COMPONENT_REDIS
  value: centos/redis-32-
centos7@sha256:06dbb609484330ec6be6090109f1fa16e936afcf975d1cbc5fff3e6c7cae7542
```

NOTE

This is done at the Operator level, so every **QuayRegistry** will be deployed using these same overrides.

4.8. AWS S3 CLOUDFRONT

NOTE

Currently, using AWS S3 CloudFront is not supported on IBM Power and IBM Z.

Use the following procedure if you are using AWS S3 Cloudfront for your backend registry storage.

Procedure

1. Enter the following command to specify the registry key:

```
$ 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
```

CHAPTER 5. RED HAT QUAY BUILD ENHANCEMENTS

Red Hat Quay builds can be run on virtualized platforms. Backwards compatibility to run previous build configurations are also available.

5.1. 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 does not have the same isolation as using virtual machines. Changing the build environment might also caused builds that were previously working to fail.

5.2. 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.

5.2.1. OpenShift Container Platform TLS component

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

NOTE

Red Hat Quay 3.11 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
```

5.2.2. 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.

5.2.2.1. 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
      ```

      NOTE

      When the token expires you will need to request a new token. Optionally, you can also add a custom expiration. For example, specify **--duration 20160m** to retain the token for two weeks.

Example output

```
eyJhbGciOiJSUzI1NiIsImtpZCI6IldfQUJkaDVmb3ltTHZ0dGZMYjhIWnYxZTQzN2dJVEJxc
DJscldSdEUtYWsifQ...
```

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: 0
      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: 0
        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"
```

5.2.2.2. 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.

5.2.2.2.1. 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
  ```

5.2.2.2.2. 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'
```

5.2.2.2.3. 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
```

5.2.2.2.4. 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
```

5.2.2.3. 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.

5.2.2.4. 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
    }
]
```

5.2.2.5. Modifying your Google Cloud Platform object bucket

NOTE

Currently, modifying your Google Cloud Platform object bucket is not supported on IBM Power and IBM Z.

Use the following procedure to configure cross-origin resource sharing (CORS) for virtual builders.

NOTE

Without CORS configuration, uploading a build Dockerfile fails.

Procedure

1. Use the following reference to create a JSON file for your specific CORS needs. For example:

```
$ cat gcp_cors.json
```

Example output

```
[
    {
      "origin": ["*"],
      "method": ["GET"],
      "responseHeader": ["Authorization"],
      "maxAgeSeconds": 3600
    },
    {
      "origin": ["*"],
      "method": ["PUT"],
      "responseHeader": [
              "Content-Type",
              "x-goog-acl",
              "origin"],
      "maxAgeSeconds": 3600
    }
]
```

2. Enter the following command to update your GCP storage bucket:

```
$ gcloud storage buckets update gs://<bucket_name> --cors-file=./gcp_cors.json
```

Example output

```
Updating
  Completed 1
```

3. You can display the updated CORS configuration of your GCP bucket by running the following command:

```
$ gcloud storage buckets describe gs://<bucket_name>  --format="default(cors)"
```

Example output

```
cors:
- maxAgeSeconds: 3600
  method:
  - GET
  origin:
  - '*'
  responseHeader:
```

```
  - Authorization
- maxAgeSeconds: 3600
  method:
  - PUT
  origin:
  - '*'
  responseHeader:
  - Content-Type
  - x-goog-acl
  - origin
```

CHAPTER 6. GEO-REPLICATION

NOTE

Currently, the geo-replication feature is not supported on IBM Power.

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 and redirect for clients.

Deployments of Red Hat Quay with geo-replication is supported on standalone and Operator deployments.

ADDITIONAL RESOURCES

- For more information about the geo-replication feature's architecture, see the architecture guide, which includes technical diagrams and a high-level overview.

6.1. 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. This is 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 has not been completed yet, the pull will use the source storage backend instead.

6.2. GEO-REPLICATION REQUIREMENTS AND CONSTRAINTS

- In geo-replicated setups, Red Hat Quay requires that all regions are able to read and 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. Users must configure a global load balancer (LB) to monitor the health of your distributed system and to route traffic to different sites based on their storage status.
- To check the status of your geo-replication deployment, you must use the **/health/endtoend** checkpoint, which is used for global health monitoring. You must configure the redirect manually using the **/health/endtoend** endpoint. The **/health/instance** end point only checks local instance health.

- 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 Red Hat 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 Red Hat Quay setup and needs to accessible by all Red Hat 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.
- Each region must be able to access every storage engine in each region, which requires a network path.
- Alternatively, the storage proxy option can be used.
- The entire storage backend, for example, all blobs, is replicated. Repository mirroring, by contrast, can be limited to a repository, or an image.
- All Red Hat Quay instances must share the same entrypoint, typically through a load balancer.
- All Red Hat 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 Red Hat Quay Operator must communicate with the same database. For more information, see Advanced Clair configuration .
- Geo-Replication requires SSL/TLS certificates and keys. For more information, see Using SSL/TLS to protect connections to Red Hat Quay.

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

6.2.1. Setting up geo-replication on OpenShift Container Platform

Use the following procedure to set up geo-replication on OpenShift Container Platform.

Procedure

1. Deploy a postgres instance for Red Hat Quay.
2. Login to the database by entering the following command:

```
psql -U <username> -h <hostname> -p <port> -d <database_name>
```

3. Create a database for Red Hat Quay named **quay**. For example:

   ```
   CREATE DATABASE quay;
   ```

4. Enable pg_trm extension inside the database

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

5. 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. Verify that it is accessible from the clusters where Red Hat 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
      ```

6. Create two object storage backends, one for each cluster. Ideally, one object storage bucket will be close to the first, or primary, cluster, and the other will run closer to the second, or secondary, cluster.

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

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

6.2.1.1. Configuring geo-replication for the Red Hat Quay on OpenShift Container Platform

Use the following procedure to configure geo-replication for the Red Hat Quay on OpenShift Container Platform.

Procedure

1. Create a **config.yaml** file that is shared between clusters. This **config.yaml** file contains the details for the common PostgreSQL, Redis and storage backends:

   Geo-replication config.yaml file

   ```
   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
DATABASE_SECRET_KEY: 0ce4f796-c295-415b-bf9d-b315114704b8
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 Container Platform Operator, see Retrieving the Clair config.

2. Create the **configBundleSecret** by entering the following command:

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

3. 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. For example:

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 QuayRegistry example

```
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 SSL/TLS is unmanaged, and the route is managed, you must supply the certificates 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 SSL/TLS is unmanaged, and the route is managed, you must supply the certificates directly in the config bundle. For more information, see Configuring TLS and routes.

6.2.2. 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 premise. 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 the following:

- A VPN to prevent visibility of the internal storage, *or*
- A token pair that only allows access to the specified bucket used by Red Hat Quay

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

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

6.3. UPGRADING A GEO-REPLICATION DEPLOYMENT OF RED HAT QUAY ON OPENSHIFT CONTAINER PLATFORM

Use the following procedure to upgrade your geo-replicated Red Hat Quay on OpenShift Container Platform deployment.

IMPORTANT

- When upgrading geo-replicated Red Hat Quay on OpenShift Container Platform deployment to the next y-stream release (for example, Red Hat Quay 3.7 → Red Hat Quay 3.8), you must stop operations before upgrading.
- There is intermittent downtime down upgrading from one y-stream release to the next.
- It is highly recommended to back up your Red Hat Quay on OpenShift Container Platform deployment before upgrading.

PROCEDURE

This procedure assumes that you are running the Red Hat Quay registry on three or more systems. For this procedure, we will assume three systems named **System A, System B,** and **System C**. **System A** will serve as the primary system in which the Red Hat Quay Operator is deployed.

1. On System B and System C, scale down your Red Hat Quay registry. This is done by disabling auto scaling and overriding the replica county for Red Hat Quay, mirror workers, and Clair if it is managed. Use the following **quayregistry.yaml** file as a reference:

```
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

NOTE

You must keep the Red Hat Quay registry running on System A. Do not update the **quayregistry.yaml** file on System A.

2. Wait for the **registry-quay-app**, **registry-quay-mirror**, and **registry-clair-app** pods to disappear. Enter the following command to check their status:

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

   Example output

   ```
   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-redis-84f888776f-hhgms           1/1     Running     0             12m
   ```

3. On System A, initiate a Red Hat Quay upgrade to the latest y-stream version. This is a manual process. For more information about upgrading installed Operators, see Upgrading installed Operators. For more information about Red Hat Quay upgrade paths, see Upgrading the Red Hat Quay Operator.

4. After the new Red Hat Quay registry is installed, the necessary upgrades on the cluster are automatically completed. Afterwards, new Red Hat Quay pods are started with the latest y-stream version. Additionally, new **Quay** pods are scheduled and started.

5. Confirm that the update has properly worked by navigating to the Red Hat Quay UI:

   a. In the **OpenShift** console, navigate to **Operators** → **Installed Operators**, and click the **Registry Endpoint** link.

      IMPORTANT

      Do not execute the following step until the Red Hat Quay UI is available. Do not upgrade the Red Hat Quay registry on System B and on System C until the UI is available on System A.

6. Confirm that the update has properly worked on System A, initiate the Red Hat Quay upgrade on System B and on System C. The Operator upgrade results in an upgraded Red Hat Quay installation, and the pods are restarted.

   NOTE

   Because the database schema is correct for the new y-stream installation, the new pods on System B and on System C should quickly start.

7. After updating, revert the changes made in step 1 of this procedure by removing **overrides** for the components. For example:

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

```
  namespace: ns
spec:
  components:
    …
    - kind: horizontalpodautoscaler
      managed: true (1)
    - kind: quay
      managed: true
    - kind: clair
      managed: true
    - kind: mirror
      managed: true
    …
```

(1) If the **horizontalpodautoscaler** resource was set to **true** before the upgrade procedure, or if you want Red Hat Quay to scale in case of a resource shortage, set it to **true**.

6.3.1. Removing a geo-replicated site from your Red Hat Quay on OpenShift Container Platform deployment

By using the following procedure, Red Hat Quay administrators can remove sites in a geo-replicated setup.

Prerequisites

- You are logged into OpenShift Container Platform.
- You have configured Red Hat Quay geo-replication with at least two sites, for example, **usstorage** and **eustorage**.
- Each site has its own Organization, Repository, and image tags.

Procedure

1. Sync the blobs between all of your defined sites by running the following command:

```
$ python -m util.backfillreplication
```

WARNING

Prior to removing storage engines from your Red Hat Quay **config.yaml** file, you **must** ensure that all blobs are synced between all defined sites.

When running this command, replication jobs are created which are picked up by the replication worker. If there are blobs that need replicated, the script returns UUIDs of blobs that will be replicated. If you run this command multiple times, and the output from the return script is empty, it does not mean that the replication process is done; it means that there are no more blobs to be queued for replication. Customers should use appropriate judgement before proceeding, as the allotted time replication takes depends on the number of blobs detected.

Alternatively, you could use a third party cloud tool, such as Microsoft Azure, to check the synchronization status.

This step must be completed before proceeding.

2. In your Red Hat Quay **config.yaml** file for site **usstorage**, remove the **DISTRIBUTED_STORAGE_CONFIG** entry for the **eustorage** site.

3. Enter the following command to identify your **Quay** application pods:

```
$ oc get pod -n <quay_namespace>
```

Example output

```
quay390usstorage-quay-app-5779ddc886-2drh2
quay390eustorage-quay-app-66969cd859-n2ssm
```

4. Enter the following command to open an interactive shell session in the **usstorage** pod:

```
$ oc rsh quay390usstorage-quay-app-5779ddc886-2drh2
```

5. Enter the following command to permanently remove the **eustorage** site:

IMPORTANT

The following action cannot be undone. Use with caution.

```
sh-4.4$ python -m util.removelocation eustorage
```

Example output

```
WARNING: This is a destructive operation. Are you sure you want to remove eustorage from
your storage locations? [y/n] y
Deleted placement 30
Deleted placement 31
Deleted placement 32
Deleted placement 33
Deleted location eustorage
```

CHAPTER 7. 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

7.1. OPTIONAL: ENABLING READ-ONLY MODE FOR RED HAT QUAY ON OPENSHIFT CONTAINER PLATFORM

Enabling read-only mode for your Red Hat Quay on OpenShift Container Platform deployment allows you to manage the registry's operations. Administrators can enable read-only mode to restrict write access to the registry, which helps ensure data integrity, mitigate risks during maintenance windows, and provide a safeguard against unintended modifications to registry data. It also helps to ensure that your Red Hat Quay registry remains online and available to serve images to users.

When backing up and restoring, you are required to scale down your Red Hat Quay on OpenShift Container Platform deployment. This results in service unavailability during the backup period which, in some cases, might be unacceptable. Enabling read-only mode ensures service availability during the backup and restore procedure for Red Hat Quay on OpenShift Container Platform deployments.

Prerequisites

- If you are using Red Hat Enterprise Linux (RHEL) 7.x:
  - You have enabled the Red Hat Software Collections List (RHSCL).
  - You have installed Python 3.6.
  - You have downloaded the **virtualenv** package.
  - You have installed the **git** CLI.
- If you are using Red Hat Enterprise Linux (RHEL) 8:
  - You have installed Python 3 on your machine.
  - You have downloaded the **python3-virtualenv** package.
  - You have installed the **git** CLI.
- You have cloned the **https://github.com/quay/quay.git** repository.
- You have installed the **oc** CLI.
- You have access to the cluster with **cluster-admin** privileges.

7.1.1. Creating service keys for Red Hat Quay on OpenShift Container Platform

Red Hat Quay uses service keys to communicate with various components. These keys are used to sign completed requests, such as requesting to scan images, login, storage access, and so on.

Procedure

1. Enter the following command to obtain a list of Red Hat Quay pods:

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

Example output

```
example-registry-clair-app-7dc7ff5844-4skw5          0/1     Error                  0           70d
example-registry-clair-app-7dc7ff5844-nvn4f          1/1     Running                0           31d
example-registry-clair-app-7dc7ff5844-x4smw          0/1     ContainerStatusUnknown   6 (70d
ago)   70d
example-registry-clair-app-7dc7ff5844-xjnvt          1/1     Running                0           60d
example-registry-clair-postgres-547d75759-75c49      1/1     Running                0           70d
example-registry-quay-app-76c8f55467-52wjz           1/1     Running                0           70d
example-registry-quay-app-76c8f55467-hwz4c           1/1     Running                0           70d
example-registry-quay-app-upgrade-57ghs              0/1     Completed              1           70d
example-registry-quay-database-7c55899f89-hmnm6      1/1     Running                0
70d
example-registry-quay-mirror-6cccbd76d-btsnb         1/1     Running                0           70d
example-registry-quay-mirror-6cccbd76d-x8g42         1/1     Running                0           70d
example-registry-quay-redis-85cbdf96bf-4vk5m         1/1     Running                0           70d
```

2. Open a remote shell session to the **Quay** container by entering the following command:

```
$ oc rsh example-registry-quay-app-76c8f55467-52wjz
```

3. Enter the following command to create the necessary service keys:

```
sh-4.4$ python3 tools/generatekeypair.py quay-readonly
```

Example output

```
Writing public key to quay-readonly.jwk
Writing key ID to quay-readonly.kid
Writing private key to quay-readonly.pem
```

7.1.2. Adding keys to the PostgreSQL database

Use the following procedure to add your service keys to the PostgreSQL database.

Prerequistes

- You have created the service keys.

Procedure

1. Enter the following command to enter your Red Hat Quay database environment:

```
$ oc rsh example-registry-quay-app-76c8f55467-52wjz psql -U <database_username> -d
<database_name>
```

2. Display the approval types and associated notes of the **servicekeyapproval** by entering the following command:

```
quay=# select * from servicekeyapproval;
```

Example output

```
 id | approver_id |         approval_type          |       approved_date        | notes
----+-------------+--------------------------------+----------------------------+-------
  1 |             | ServiceKeyApprovalType.AUTOMATIC | 2024-05-07 03:47:48.181347 |
  2 |             | ServiceKeyApprovalType.AUTOMATIC | 2024-05-07 03:47:55.808087 |
  3 |             | ServiceKeyApprovalType.AUTOMATIC | 2024-05-07 03:49:04.27095  |
  4 |             | ServiceKeyApprovalType.AUTOMATIC | 2024-05-07 03:49:05.46235  |
  5 |           1 | ServiceKeyApprovalType.SUPERUSER | 2024-05-07 04:05:10.296796 |
...
```

3. Add the service key to your Red Hat Quay database by entering the following query:

```
quay=# INSERT INTO servicekey
  (name, service, metadata, kid, jwk, created_date, expiration_date)
  VALUES ('quay-readonly',
          'quay',
          '{}',
          '{<contents_of_.kid_file>}',
          '{<contents_of_.jwk_file>}',
          '{<created_date_of_read-only>}',
          '{<expiration_date_of_read-only>}');
```

Example output

```
INSERT 0 1
```

4. Next, add the key approval with the following query:

```
quay=# INSERT INTO servicekeyapproval ('approval_type', 'approved_date', 'notes')
  VALUES ("ServiceKeyApprovalType.SUPERUSER", "CURRENT_DATE",
          {include_notes_here_on_why_this_is_being_added});
```

Example output

```
INSERT 0 1
```

5. Set the **approval_id** field on the created service key row to the **id** field from the created service key approval. You can use the following **SELECT** statements to get the necessary IDs:

```
UPDATE servicekey
SET approval_id = (SELECT id FROM servicekeyapproval WHERE approval_type =
'ServiceKeyApprovalType.SUPERUSER')
WHERE name = 'quay-readonly';
```

```
UPDATE 1
```

7.1.3. Configuring read-only mode Red Hat Quay on OpenShift Container Platform

After the service keys have been created and added to your PostgreSQL database, you must restart the **Quay** container on your OpenShift Container Platform deployment.

IMPORTANT

Deploying Red Hat Quay on OpenShift Container Platform in read-only mode requires you to modify the secrets stored inside of your OpenShift Container Platform cluster. It is highly recommended that you create a backup of the secret prior to making changes to it.

Prerequisites

- You have created the service keys and added them to your PostgreSQL database.

Procedure

1. Enter the following command to read the secret name of your Red Hat Quay on OpenShift Container Platform deployment:

   ```
   $ oc get deployment -o yaml <quay_main_app_deployment_name>
   ```

2. Use the **base64** command to encode the **quay-readonly.kid** and **quay-readonly.pem** files:

   ```
   $ base64 -w0 quay-readonly.kid
   ```

   Example output

   ```
   ZjUyNDFm...
   ```

   ```
   $ base64 -w0 quay-readonly.pem
   ```

   Example output

   ```
   LS0tLS1CRUdJTiBSU0E...
   ```

3. Obtain the current configuration bundle and secret by entering the following command:

   ```
   $ oc get secret quay-config-secret-name -o json | jq '.data."config.yaml"' | cut -d '"' -f2 |
   base64 -d -w0 > config.yaml
   ```

4. Edit the **config.yaml** file and add the following information:

   ```
   # ...
   REGISTRY_STATE: readonly
   INSTANCE_SERVICE_KEY_KID_LOCATION: 'conf/stack/quay-readonly.kid'
   INSTANCE_SERVICE_KEY_LOCATION: 'conf/stack/quay-readonly.pem'
   # ...
   ```

5. Save the file and **base64** encode it by running the following command:

   ```
   $ base64 -w0 quay-config.yaml
   ```

6. Scale down the Red Hat Quay Operator pods to **0**. This ensures that the Operator does not reconcile the secret after editing it.

   ```
   $ oc scale --replicas=0 deployment quay-operator -n openshift-operators
   ```

7. Edit the secret to include the new content:

```
$ oc edit secret quay-config-secret-name -n quay-namespace
```

```
# ...
data:
  "quay-readonly.kid": "ZjUyNDFm..."
  "quay-readonly.pem": "LS0tLS1CRUdJTiBSU0E..."
  "config.yaml": "QUNUSU9OX0xPR19..."
# ...
```

With your Red Hat Quay on OpenShift Container Platform deployment on read-only mode, you can safely manage your registry's operations and perform such actions as backup and restore.

7.1.3.1. Scaling up the Red Hat Quay on OpenShift Container Platform from a read-only deployment

When you no longer want Red Hat Quay on OpenShift Container Platform to be in read-only mode, you can scale the deployment back up and remove the content added from the secret.

Procedure

1. Edit the **config.yaml** file and remove the following information:

```
# ...
REGISTRY_STATE: readonly
INSTANCE_SERVICE_KEY_KID_LOCATION: 'conf/stack/quay-readonly.kid'
INSTANCE_SERVICE_KEY_LOCATION: 'conf/stack/quay-readonly.pem'
# ...
```

2. Scale the Red Hat Quay Operator back up by entering the following command:

```
oc scale --replicas=1 deployment quay-operator -n openshift-operators
```

7.2. BACKING UP RED HAT QUAY

Database backups should be performed regularly using either the supplied tools on the PostgreSQL image or your own backup infrastructure. The Red Hat Quay Operator does not ensure that the PostgreSQL database is backed up.

NOTE

This procedure covers backing up your Red Hat Quay PostgreSQL database. It does not cover backing up the Clair PostgreSQL database. Strictly speaking, backing up the Clair PostgreSQL database is not needed because it can be recreated. If you opt to recreate it from scratch, you will wait for the information to be repopulated after all images inside of your Red Hat Quay deployment are scanned. During this downtime, security reports are unavailable.

If you are considering backing up the Clair PostgreSQL database, you must consider that its size is dependent upon the number of images stored inside of Red Hat Quay. As a result, the database can be extremely large.

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

Prerequisites

- A healthy Red Hat Quay deployment on OpenShift Container Platform using the Red Hat Quay Operator. The 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 v3.7 or later)

> NOTE
>
> If your deployment contains partially unmanaged database or storage components and you are using external services for PostgreSQL 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 PostgreSQL database or object storage.

7.2.1. Red Hat Quay configuration backup

Use the following procedure to back up your Red Hat Quay configuration.

Procedure

1. To back the **QuayRegistry** custom resource by exporting it, enter the following command:

```
$ 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 by entering the following command:

> 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 Red Hat 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 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. Redirect the current **Quay** configuration file by entering the following command:

   ```
   $ 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
   ```

7.2.2. Scaling down your Red Hat Quay deployment

Use the following procedure to 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 PostgreSQL databases and/or S3-compatible object storage are provided by external services (unmanaged by the Red Hat Quay Operator).

Procedure

1. Depending on the version of your Red Hat Quay deployment, scale down your deployment using one of the following options.

   a. **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

b. **For Operator version 3.6 and earlier**. Scale down the Red Hat Quay deployment by scaling down the Red Hat Quay registry 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>
```

2. 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
```

Example output

```
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-database-859d5445ff-cqthr        1/1     Running     0             12m
quayregistry-quay-redis-84f888776f-hhgms           1/1     Running     0             12m
```

7.2.3. Backing up the Red Hat Quay managed database

Use the following procedure to back up the Red Hat Quay managed database.

NOTE

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

Procedure

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
```

7.2.3.1. Backing up the Red Hat Quay managed object storage

Use the following procedure to back up the Red Hat Quay managed object storage. 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.

Procedure

1. Decode and export the **AWS_ACCESS_KEY_ID** by entering the following command:

```
$ 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** by entering the following command:

```
$ 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:

```
$ mkdir blobs
```

NOTE

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

1. Copy all blobs to the directory by entering the following command:

```
$ 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
```

7.2.4. Scale the Red Hat Quay deployment back up

1. Depending on the version of your Red Hat Quay deployment, scale up your deployment using one of the following options.

   a. **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

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

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

2. Check the status of the Red Hat Quay deployment by entering the following command:

```
$ 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
```

7.3. RESTORING RED HAT QUAY

Use the following procedures 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 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 PostgreSQL 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

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

Use the following procedure to restore Red Hat Quay and its configuration files 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.

Procedure

1. Restore the backed up Red Hat Quay configuration by entering the following command:

```
$ oc create -f ./config-bundle.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 generated keys from the backup by entering the following command:

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

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

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

4. 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>
```

7.3.2. Scaling down your Red Hat Quay deployment

Use the following procedure to scale down your Red Hat Quay deployment.

Procedure

1. Depending on the version of your Red Hat Quay deployment, scale down your deployment using one of the following options.

   a. **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

   b. **For Operator version 3.6 and earlier:**Scale down the Red Hat Quay deployment by scaling down the Red Hat Quay registry 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>
```

2. Wait for the **registry-quay-app**, **registry-quay-mirror** and **registry-clair-app** pods (depending on which components you set to be managed by Red Hat Quay 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
   ```

7.3.3. Restoring your Red Hat Quay database

Use the following procedure to restore your Red Hat Quay database.

Procedure

1. Identify your **Quay** database pod by entering the following command:

   ```
   $ 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 by entering the following command:

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

4. Enter psql by running the following command:

   ```
   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 by entering the following command:

```
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 by entering the following command:

```
sh-4.4$ exit
```

7.3.4. Restore your Red Hat Quay object storage data

Use the following procedure to restore your Red Hat Quay object storage data.

Procedure

1. Export the **AWS_ACCESS_KEY_ID** by entering the following command:

```
$ 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** by entering the following command:

```
$ 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.

7.3.5. Scaling up your Red Hat Quay deployment

1. Depending on the version of your Red Hat Quay deployment, scale up your deployment using one of the following options.

   a. **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

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

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

2. 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 8. VOLUME SIZE OVERRIDES

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

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

```
apiVersion: quay.redhat.com/v1
kind: QuayRegistry
metadata:
  name: quay-example
  namespace: quay-enterprise
spec:
  configBundleSecret: config-bundle-secret
  components:
    - kind: objectstorage
      managed: false
    - kind: route
      managed: true
    - kind: tls
      managed: false
    - kind: clair
      managed: true
      overrides:
        volumeSize: 70Gi
    - kind: postgres
      managed: true
      overrides:
        volumeSize: 70Gi
    - kind: clairpostgres
      managed: true
```

NOTE

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

CHAPTER 9. SCANNING POD IMAGES WITH THE CONTAINER SECURITY OPERATOR

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

NOTE

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

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

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

NOTE

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

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

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

NOTE

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

Procedure

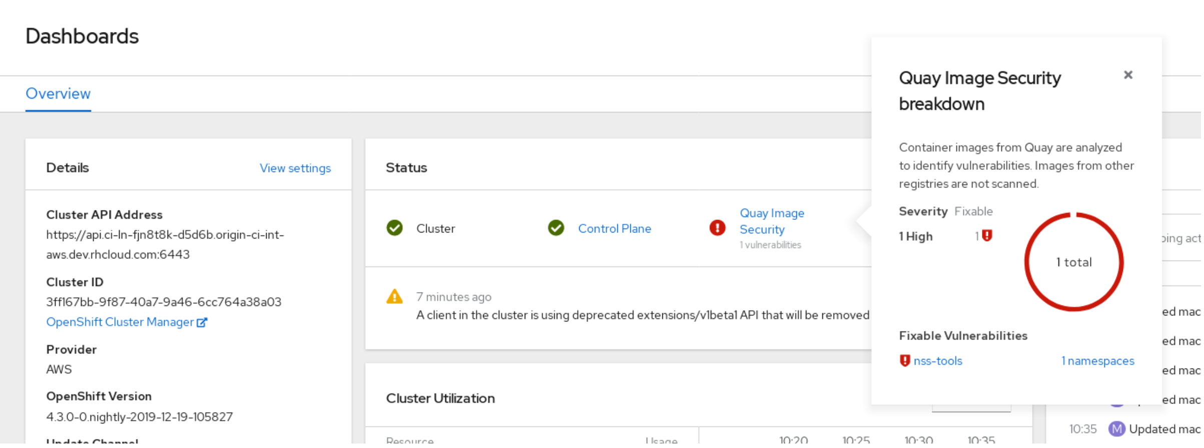
1. On the OpenShift Container Platform console page, select **Operators → OperatorHub** and search for **Container Security Operator**.
2. Select the Container Security Operator, then select **Install** to go to the **Create Operator Subscription** page.
3. Check the settings (all namespaces and automatic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

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

NOTE

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

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



Dashboards

Overview

Details View settings

Cluster API Address
https://api.ci-ln-fjn8t8k-d5d6b.origin-ci-int-aws.dev.rhcloud.com:6443

Cluster ID
3ff167bb-9f87-40a7-9a46-6cc764a38a03
OpenShift Cluster Manager

Provider
AWS

OpenShift Version
4.3.0-0.nightly-2019-12-19-105827

Status

Cluster Control Plane Quay Image Security 1 vulnerabilities

7 minutes ago
A client in the cluster is using deprecated extensions/v1beta1 API that will be removed

Cluster Utilization

Quay Image Security breakdown ×

Container images from Quay are analyzed to identify vulnerabilities. Images from other registries are not scanned.

Severity	Fixable
1 High	1

1 total

Fixable Vulnerabilities

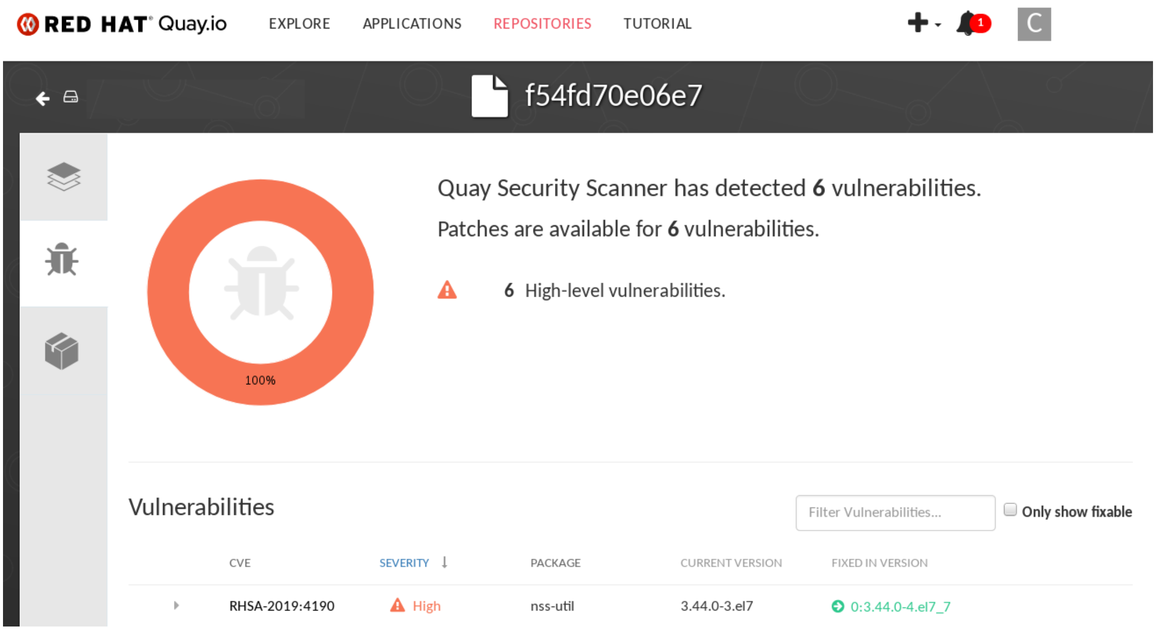
nss-tools 1 namespaces

IMPORTANT

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

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

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



Quay Security Scanner has detected **6** vulnerabilities.

Patches are available for **6** vulnerabilities.

6 High-level vulnerabilities.

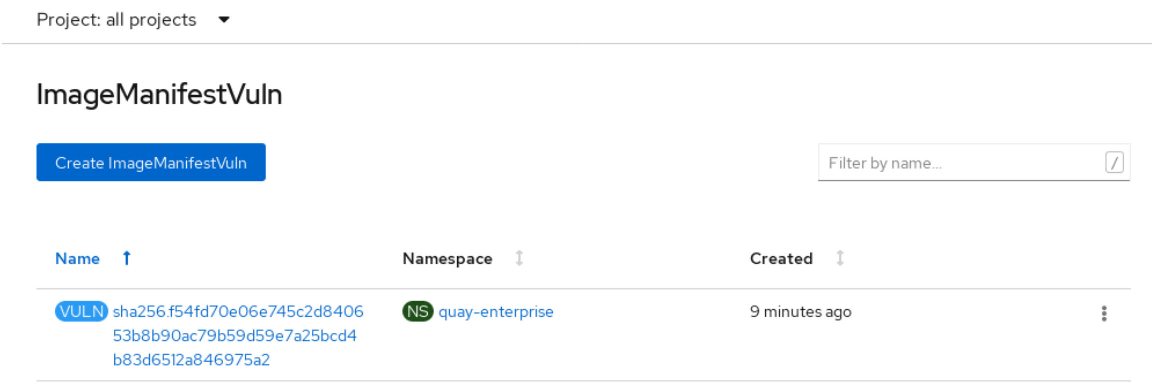
Vulnerabilities

Filter Vulnerabilities...

□ **Only show fixable**

	CVE	SEVERITY	PACKAGE	CURRENT VERSION	FIXED IN VERSION
▸	RHSA-2019:4190	High	nss-util	3.44.0-3.el7	0:3.44.0-4.el7_7

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



ImageManifestVuln

Name	Namespace	Created
VULN sha256.f54fd70e06e745c2d8406 53b8b90ac79b59d59e7a25bcd4 b83d6512a846975a2	NS quay-enterprise	9 minutes ago

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

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

NOTE

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

9.2. QUERY IMAGE VULNERABILITIES FROM THE CLI

Use the following procedure to query image vulnerabilities from the command line interface (CLI).

Procedure

1. Enter the following command to query for detected vulnerabilities:

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

Example output

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

2. Optional. To display details for a particular vulnerability, identify a specific vulnerability and its namespace, and use the **oc describe** command. The following example shows an active container whose image includes an RPM package with a vulnerability:

```
$ oc describe vuln --namespace mynamespace sha256.ac50e3752...
```

Example output

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

CHAPTER 10. CONFIGURING AWS STS FOR RED HAT QUAY

Support for Amazon Web Services (AWS) Security Token Service (STS) is available for standalone Red Hat Quay deployments and Red Hat Quay on OpenShift Container Platform. AWS STS is a web service for requesting temporary, limited-privilege credentials for AWS Identity and Access Management (IAM) users and for users that you authenticate, or *federated users*. This feature is useful for clusters using Amazon S3 as an object storage, allowing Red Hat Quay to use STS protocols to authenticate with Amazon S3, which can enhance the overall security of the cluster and help to ensure that access to sensitive data is properly authenticated and authorized.

Configuring AWS STS is a multi-step process that requires creating an AWS IAM user, creating an S3 role, and configuring your Red Hat Quay **config.yaml** file to include the proper resources.

Use the following procedures to configure AWS STS for Red Hat Quay.

10.1. CREATING AN IAM USER

Use the following procedure to create an IAM user.

Procedure

1. Log in to the Amazon Web Services (AWS) console and navigate to the Identity and Access Management (IAM) console.
2. In the navigation pane, under **Access management** click **Users**.
3. Click **Create User** and enter the following information:
    a. Enter a valid username, for example, **quay-user**.
    b. For **Permissions options**, click **Add user to group**.
4. On the **review and create** page, click **Create user**. You are redirected to the **Users** page.
5. Click the username, for example, **quay-user**.
6. Copy the ARN of the user, for example, **arn:aws:iam::123492922789:user/quay-user**.
7. On the same page, click the **Security credentials** tab.
8. Navigate to **Access keys**.
9. Click **Create access key**.
10. On the **Access key best practices & alternatives**page, click **Command Line Interface (CLI)**, then, check the confirmation box. Then click **Next**.
11. Optional. On the **Set description tag - optional**page, enter a description.
12. Click **Create access key**.
13. Copy and store the access key and the secret access key.

IMPORTANT

This is the only time that the secret access key can be viewed or downloaded. You cannot recover it later. However, you can create a new access key any time.

14. Click **Done**.

10.2. CREATING AN S3 ROLE

Use the following procedure to create an S3 role for AWS STS.

Prerequisites

- You have created an IAM user and stored the access key and the secret access key.

Procedure

1. If you are not already, navigate to the IAM dashboard by clicking **Dashboard**.
2. In the navigation pane, click **Roles** under **Access management**.
3. Click **Create role**.
   - Click **Custom Trust Policy**, which shows an editable JSON policy. By default, it shows the following information:

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Sid": "Statement1",
   "Effect": "Allow",
   "Principal": {},
   "Action": "sts:AssumeRole"
  }
 ]
}
```

4. Under the **Principal** configuration field, add your AWS ARN information. For example:

```
{
    "Version": "2012-10-17",
    "Statement": [
     {
      "Sid": "Statement1",
      "Effect": "Allow",
      "Principal": {
       "AWS": "arn:aws:iam::123492922789:user/quay-user"
      },
      "Action": "sts:AssumeRole"
     }
    ]
}
```

5. Click **Next**.

6. On the **Add permissions** page, type **AmazonS3FullAccess** in the search box. Check the box to add that policy to the S3 role, then click **Next**.

7. On the **Name, review, and create** page, enter the following information:

   a. Enter a role name, for example, **example-role**.

   b. Optional. Add a description.

8. Click the **Create role** button. You are navigated to the **Roles** page. Under **Role name**, the newly created S3 should be available.

10.3. CONFIGURING RED HAT QUAY ON OPENSHIFT CONTAINER PLATFORM TO USE AWS STS

Use the following procedure to edit your Red Hat Quay on OpenShift Container Platform **config.yaml** file to use AWS STS.

NOTE

You can also edit and re-deploy your Red Hat Quay on OpenShift Container Platform **config.yaml** file directly instead of using the OpenShift Container Platform UI.

Prerequisites

- You have configured a Role ARN.
- You have generated a User Access Key.
- You have generated a User Secret Key.

Procedure

1. On the **Home** page of your OpenShift Container Platform deployment, click **Operators → Installed Operators**.

2. Click **Red Hat Quay**.

3. Click **Quay Registry** and then the name of your Red Hat Quay registry.

4. Under **Config Bundle Secret**, click the name of your registry configuration bundle, for example, **quay-registry-config-bundle-qet56**.

5. On the configuration bundle page, click **Actions** to reveal a drop-down menu. Then click **Edit Secret**.

6. Update your the **DISTRIBUTED_STORAGE_CONFIG** fields of your **config.yaml** file with the following information:

```
# ...
DISTRIBUTED_STORAGE_CONFIG:
  default:
   - STSS3Storage
```

```
    - sts_role_arn: <role_arn> ①
      s3_bucket: <s3_bucket_name> ②
      storage_path: <storage_path> ③
      sts_user_access_key: <s3_user_access_key> ④
      sts_user_secret_key: <s3_user_secret_key> ⑤
# ...
```

① The unique Amazon Resource Name (ARN) required when configuring AWS STS

② The name of your s3 bucket.

③ The storage path for data. Usually **/datastorage**.

④ The generated AWS S3 user access key required when configuring AWS STS.

⑤ The generated AWS S3 user secret key required when configuring AWS STS.

7. Click **Save**.

Verification

1. Tag a sample image, for example, **busybox**, that will be pushed to the repository. For example:

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

2. Push the sample image by running the following command:

```
$ podman push <quay-server.example.com>/<organization_name>/busybox:test
```

3. Verify that the push was successful by navigating to the Organization that you pushed the image to in your Red Hat Quay registry → **Tags**.

4. Navigate to the Amazon Web Services (AWS) console and locate your s3 bucket.

5. Click the name of your s3 bucket.

6. On the **Objects** page, click **datastorage/**.

7. On the **datastorage/** page, the following resources should seen:

    - **sha256/**
    - **uploads/**
      These resources indicate that the push was successful, and that AWS STS is properly configured.

CHAPTER 11. INTEGRATING RED HAT QUAY INTO OPENSHIFT CONTAINER PLATFORM WITH THE QUAY BRIDGE OPERATOR

The Quay Bridge Operator duplicates the features of the integrated OpenShift Container Platform registry into the new Red Hat Quay registry. Using the Quay Bridge Operator, you can replace the integrated container registry in OpenShift Container Platform with a Red Hat Quay registry.

The features enabled with the Quay Bridge Operator include:

- Synchronizing OpenShift Container Platform namespaces as Red Hat Quay organizations.
- Creating robot accounts for each default namespace service account.
- Creating secrets for each created robot account, and associating each robot secret to a service account as **Mountable** and **Image Pull Secret**.
- Synchronizing OpenShift Container Platform image streams as Red Hat Quay repositories.
- Automatically rewriting new builds making use of image streams to output to Red Hat Quay.
- Automatically importing an image stream tag after a build completes.

By using the following procedures, you can enable bi-directional communication between your Red Hat Quay and OpenShift Container Platform clusters.

11.1. SETTING UP RED HAT QUAY FOR THE QUAY BRIDGE OPERATOR

In this procedure, you will create a dedicated Red Hat Quay organization, and from a new application created within that organization you will generate an OAuth token to be used with the Quay Bridge Operator in OpenShift Container Platform.

Procedure

1. Log in to Red Hat Quay through the web UI.
2. Select the organization for which the external application will be configured.
3. On the navigation pane, select **Applications**.
4. Select **Create New Application** and enter a name for the new application, for example, **openshift**.
5. On the **OAuth Applications** page, select your application, for example, **openshift**.
6. On the navigation pane, select **Generate Token**.
7. Select the following fields:
   - **Administer Organization**
   - **Administer Repositories**
   - **Create Repositories**

- **View all visible repositories**
- **Read/Write to any accessible repositories**
- **Administer User**
- **Read User Information**

8. Review the assigned permissions.

9. Select **Authorize Application** and then confirm confirm the authorization by selecting **Authorize Application**.

10. Save the generated access token.

IMPORTANT

Red Hat Quay does not offer token management. You cannot list tokens, delete tokens, or modify tokens. The generated access token is only shown once and cannot be re-obtained after closing the page.

11.2. INSTALLING THE QUAY BRIDGE OPERATOR ON OPENSHIFT CONTAINER PLATFORM

In this procedure, you will install the Quay Bridge Operator on OpenShift Container Platform.

Prerequiites

- You have set up Red Hat Quay and obtained an Access Token.
- An OpenShift Container Platform 4.6 or greater environment for which you have cluster administrator permissions.

Procedure

1. Open the **Administrator** perspective of the web console and navigate to **Operators → OperatorHub** on the navigation pane.

2. Search for **Quay Bridge Operator**, click the **Quay Bridge Operator** title, and then click **Install**.

3. Select the version to install, for example, **stable-3.7**, and then click **Install**.

4. Click **View Operator** when the installation finishes to go to the Quay Bridge Operator's **Details** page. Alternatively, you can click **Installed Operators → Red Hat Quay Bridge Operator**to go to the **Details** page.

11.3. CREATING AN OPENSHIFT CONTAINER PLATFORM SECRET FOR THE OAUTH TOKEN

In this procedure, you will add the previously obtained access token to communicate with your Red Hat Quay deployment. The access token will be stored within OpenShift Container Platform as a secret.

Prerequisites

- You have set up Red Hat Quay and obtained an access token.
- You have deployed the Quay Bridge Operator on OpenShift Container Platform.
- An OpenShift Container Platform 4.6 or greater environment for which you have cluster administrator permissions.
- You have installed the OpenShift CLI (oc).

Procedure

- Create a secret that contains the access token in the **openshift-operators** namespace:

```
$ oc create secret -n openshift-operators generic <secret-name> --from-literal=token=
<access_token>
```

11.4. CREATING THE QUAYINTEGRATION CUSTOM RESOURCE

In this procedure, you will create a **QuayIntegration** custom resource, which can be completed from either the web console or from the command line.

Prerequisites

- You have set up Red Hat Quay and obtained an access token.
- You have deployed the Quay Bridge Operator on OpenShift Container Platform.
- An OpenShift Container Platform 4.6 or greater environment for which you have cluster administrator permissions.
- Optional: You have installed the OpenShift CLI (oc).

11.4.1. Optional: Creating the QuayIntegration custom resource using the CLI

Follow this procedure to create the **QuayIntegration** custom resource using the command line.

Procedure

1. Create a **quay-integration.yaml**:

```
$ touch quay-integration.yaml
```

2. Use the following configuration for a minimal deployment of the **QuayIntegration** custom resource:

```
apiVersion: quay.redhat.com/v1
kind: QuayIntegration
metadata:
  name: example-quayintegration
spec:
  clusterID: openshift ❶
  credentialsSecret:
    namespace: openshift-operators
```

```
   name: quay-integration 2
  quayHostname: https://<QUAY_URL>  3
  insecureRegistry: false 4
```

1 The clusterID value should be unique across the entire ecosystem. This value is required and defaults to **openshift**.

2 The **credentialsSecret** property refers to the namespace and name of the secret containing the token that was previously created.

3 Replace the **QUAY_URL** with the hostname of your Red Hat Quay instance.

4 If Red Hat Quay is using self signed certificates, set the property to **insecureRegistry: true**.

For a list of all configuration fields, see "QuayIntegration configuration fields".

1. Create the **QuayIntegration** custom resource:

```
$ oc create -f quay-integration.yaml
```

11.4.2. Optional: Creating the QuayIntegration custom resource using the web console

Follow this procedure to create the **QuayIntegration** custom resource using the web console.

Procedure

1. Open the **Administrator** perspective of the web console and navigate to **Operators → Installed Operators**.

2. Click **Red Hat Quay Bridge Operator**.

3. On the **Details** page of the Quay Bridge Operator, click **Create Instance** on the **Quay Integration** API card.

4. On the **Create QuayIntegration** page, enter the following required information in either **Form view** or **YAML view**:

   - **Name**: The name that will refer to the **QuayIntegration** custom resource object.
   - **Cluster ID**: The ID associated with this cluster. This value should be unique across the entire ecosystem. Defaults to **openshift** if left unspecified.
   - **Credentials secret** Refers to the namespace and name of the secret containing the token that was previously created.
   - **Quay hostname**: The hostname of the Quay registry.

For a list of all configuration fields, see "QuayIntegration configuration fields".

After the **QuayIntegration** custom resource is created, your OpenShift Container Platform cluster will be linked to your Red Hat Quay instance. Organizations within your Red Hat Quay registry should be created for the related namespace for the OpenShift Container Platform environment.

11.5. USING QUAY BRIDGE OPERATOR

Use the following procedure to use the Quay Bridge Operator.

Prerequisites

- You have installed the Red Hat Quay Operator.
- You have logged into OpenShift Container Platform as a cluster administrator.
- You have logged into your Red Hat Quay registry.
- You have installed the Quay Bridge Operator.
- You have configured the **QuayIntegration** custom resource.

Procedure

1. Enter the following command to create a new OpenShift Container Platform project called **e2e-demo**:

   ```
   $ oc new-project e2e-demo
   ```

2. After you have created a new project, a new Organization is created in Red Hat Quay. Navigate to the Red Hat Quay registry and confirm that you have created a new Organization named **openshift_e2e-demo**.

   NOTE

   The **openshift** value of the Organization might different if the clusterID in your **QuayIntegration** resource used a different value.

3. On the Red Hat Quay UI, click the name of the new Organization, for example, **openshift_e2e-demo**.

4. Click **Robot Accounts** in the navigation pane. As part of new project, the following Robot Accounts should have been created:

   - **openshift_e2e-demo+deployer**
   - **openshift_e2e-demo+default**
   - **openshift_e2e-demo+builder**

5. Enter the following command to confirm three secrets containing Docker configuration associated with the applicable Robot Accounts were created:

   ```
   $ oc get secrets builder-quay-openshift deployer-quay-openshift default-quay-openshift
   ```

   Example output

   ```
   stevsmit@stevsmit ocp-quay $ oc get secrets builder-quay-openshift deployer-quay-openshift
   default-quay-openshift
   NAME                    TYPE                              DATA   AGE
   ```

```
builder-quay-openshift    kubernetes.io/dockerconfigjson   1      77m
deployer-quay-openshift   kubernetes.io/dockerconfigjson   1      77m
default-quay-openshift    kubernetes.io/dockerconfigjson   1      77m
```

6. Enter the following command to display detailed information about **builder** ServiceAccount (SA), including its secrets, token expiration, and associated roles and role bindings. This ensures that the project is integrated via the Quay Bridge Operator.

```
$ oc describe sa builder default deployer
```

Example output

```
...
Name:                builder
Namespace:           e2e-demo
Labels:              <none>
Annotations:         <none>
Image pull secrets:  builder-dockercfg-12345
                     builder-quay-openshift
Mountable secrets:   builder-dockercfg-12345
                     builder-quay-openshift
Tokens:              builder-token-12345
Events:              <none>
...
```

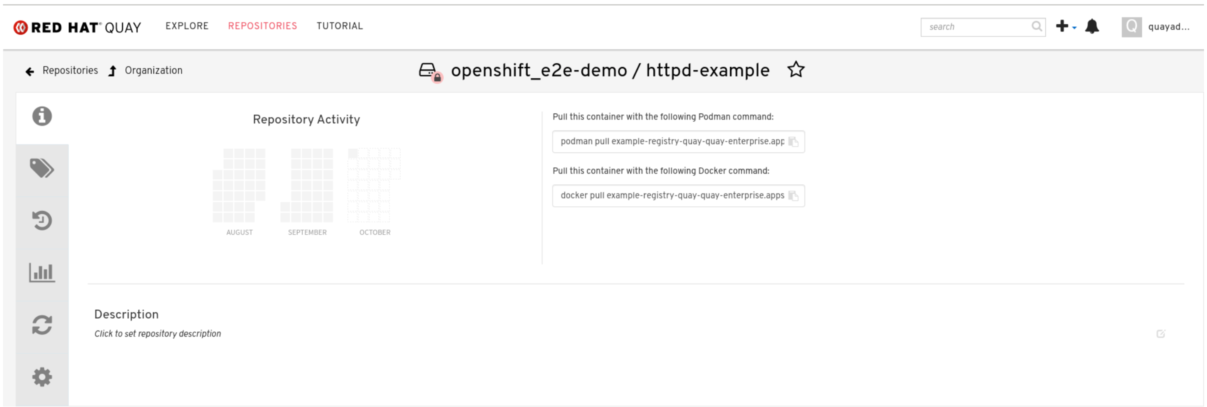
7. Enter the following command to create and deploy a new application called **httpd-template**:

```
$ oc new-app --template=httpd-example
```

Example output

```
--> Deploying template "e2e-demo/httpd-example" to project e2e-demo
...
--> Creating resources ...
    service "httpd-example" created
    route.route.openshift.io "httpd-example" created
    imagestream.image.openshift.io "httpd-example" created
    buildconfig.build.openshift.io "httpd-example" created
    deploymentconfig.apps.openshift.io "httpd-example" created
--> Success
    Access your application via route 'httpd-example-e2e-demo.apps.quay-
ocp.gcp.quaydev.org'
    Build scheduled, use 'oc logs -f buildconfig/httpd-example' to track its progress.
    Run 'oc status' to view your app.
```

After running this command, **BuildConfig**, **ImageStream**, **Service, Route**, and **DeploymentConfig** resources are created. When the **ImageStream** resource is created, an associated repository is created in Red Hat Quay. For example:



Pull this container with the following Podman command:

```
podman pull example-registry-quay-quay-enterprise.app
```

Pull this container with the following Docker command:

```
docker pull example-registry-quay-quay-enterprise.apps
```

8. The **ImageChangeTrigger** for the **BuildConfig** triggers a new Build when the Apache HTTPD image, located in the **openshift** namespace, is resolved. As the new Build is created, the **MutatingWebhookConfiguration** automatically rewriters the output to point at Red Hat Quay. You can confirm that the build is complete by querying the output field of the build by running the following command:

```
$ oc get build httpd-example-1 --template='{{ .spec.output.to.name }}'
```

Example output

```
example-registry-quay-quay-enterprise.apps.quay-ocp.gcp.quaydev.org/openshift_e2e-demo/httpd-example:latest
```

9. On the Red Hat Quay UI, navigate to the **openshift_e2e-demo** Organization and select the **httpd-example** repository.

10. Click **Tags** in the navigation pane and confirm that the **latest** tag has been successfully pushed.

11. Enter the following command to ensure that the latest tag has been resolved:

```
$ oc describe is httpd-example
```

Example output

```
Name:   httpd-example
Namespace:  e2e-demo
Created:  55 minutes ago
Labels:   app=httpd-example
  template=httpd-example
Description:  Keeps track of changes in the application image
Annotations:  openshift.io/generated-by=OpenShiftNewApp
  openshift.io/image.dockerRepositoryCheck=2023-10-02T17:56:45Z
Image Repository: image-registry.openshift-image-registry.svc:5000/e2e-demo/httpd-example
Image Lookup:  local=false
Unique Images:  0
Tags:   1

latest
  tagged from example-registry-quay-quay-enterprise.apps.quay-ocp.gcp.quaydev.org/openshift_e2e-demo/httpd-example:latest
```

12. After the **ImageStream** is resolwillved, a new deployment should have been triggered. Enter the following command to generate a URL output:

```
$ oc get route httpd-example --template='{{ .spec.host }}'
```

Example output

```
httpd-example-e2e-demo.apps.quay-ocp.gcp.quaydev.org
```

13. Navigate to the URL. If a sample webpage appears, the deployment was successful.

14. Enter the following command to delete the resources and clean up your Red Hat Quay repository:

```
$ oc delete project e2e-demo
```

NOTE

The command waits until the project resources have been removed. This can be bypassed by adding the **--wait=false** to the above command

15. After the command completes, navigate to your Red Hat Quay repository and confirm that the **openshift_e2e-demo** Organization is no longer available.

Additional resources

- Best practices dictate that all communication between a client and an image registry be facilitated through secure means. Communication should leverage HTTPS/TLS with a certificate trust between the parties. While Red Hat Quay can be configured to serve an insecure configuration, proper certificates should be utilized on the server and configured on the client. Follow the OpenShift Container Platform documentation for adding and managing certificates at the container runtime level.

CHAPTER 12. DEPLOYING IPV6 ON RED HAT QUAY ON OPENSHIFT CONTAINER PLATFORM

NOTE

Currently, deploying IPv6 on the Red Hat Quay on OpenShift Container Platform is not supported on IBM Power and IBM Z.

Your Red Hat Quay on OpenShift Container Platform 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 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 Microsoft Azure Blob Storage configuration will not work on IPv6 single stack environments. Because the endpoint of Microsoft 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, Red Hat OpenShift Data Foundation is unsupported when Red Hat Quay is deployed on IPv6 single stack environments. As a result, Red Hat OpenShift Data Foundation 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. ADDING CUSTOM SSL/TLS CERTIFICATES WHEN RED HAT QUAY IS DEPLOYED ON KUBERNETES

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

As a temporary workaround, **base64** encoded certificates can be added to the secret *after* Red Hat Quay has been deployed.

Use the following procedure to add custom SSL/TLS certificates when Red Hat Quay is deployed on Kubernetes.

Prerequisites

- Red Hat Quay has been deployed.
- You have a custom **ca.crt** file.

Procedure

1. Base64 encode the contents of an SSL/TLS certificate by entering the following command:

```
$ cat ca.crt | base64 -w 0
```

Example output

```
...c1psWGpqeGlPQmNEWkJPMjJ5d0pDemVnR2QNCnRsbW9JdEF4YnFSdVd3PT0KLS0tLS1
FTkQgQ0VSVElGSUNBVEUtLS0tLQo=
```

2. Enter the following **kubectl** command to edit the **quay-enterprise-config-secret** file:

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

3. Add an entry for the certificate and paste the full **base64** encoded stringer under the entry. For example:

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

4. Use the **kubectl delete** command to remove all Red Hat Quay pods. For example:

```
$ kubectl delete pod quay-operator.v3.7.1-6f9d859bd-p5ftc quayregistry-clair-postgres-
7487f5bd86-xnxpr quayregistry-quay-app-upgrade-xq2v6  quayregistry-quay-database-
859d5445ff-cqthr quayregistry-quay-redis-84f888776f-hhgms
```

Afterwards, the Red Hat Quay deployment automatically schedules replace pods with the new certificate data.

CHAPTER 14. UPGRADING THE RED HAT QUAY OPERATOR OVERVIEW

The Red Hat Quay Operator follows a *synchronized versioning* scheme, which means that each version of the Operator is tied to the version of Red Hat Quay and the components that it manages. There is no field on the **QuayRegistry** custom resource which sets the version of Red Hat Quay to **deploy**; the Operator can only 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 Red Hat Quay on Kubernetes.

14.1. OPERATOR LIFECYCLE MANAGER

The Red Hat 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 Red Hat Quay Operator whenever a new version becomes available.

WARNING

When the Red Hat Quay Operator is installed by Operator Lifecycle Manager, it might be configured to support automatic or manual upgrades. This option is shown on the **OperatorHub** page for the Red Hat Quay Operator during installation. It can also be found in the Red Hat Quay Operator **Subscription** object by the **approvalStrategy** field. Choosing **Automatic** means that your Red Hat 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.

14.2. UPGRADING THE RED HAT QUAY OPERATOR

The standard approach for upgrading installed Operators on OpenShift Container Platform 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 simplifies the upgrade procedure for customers on older releases. The following upgrade paths are supported for Red Hat Quay 3.11:

- 3.9.z → 3.11.z
- 3.10.z → 3.11.z

NOTE

Upgrading from 3.8.z to 3.11 is unsupported. Users must first upgrade to 3.9 or 3.10, and then upgrade to 3.11.

NOTE

The Red Hat Quay Operator can be upgraded from 3.10.X for IBM Power and IBM Z.

For users on standalone deployments of Red Hat Quay wanting to upgrade to 3.11, see the Standalone upgrade guide.

14.2.1. Upgrading Red Hat Quay

To update Red Hat Quay from one minor version to the next, for example, 3.10 → 3.11, you must change the update channel for the Red Hat Quay Operator.

For **z** stream upgrades, for example, 3.10.1 → 3.10.2, 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 Red Hat Quay Operator upgrades automatically to the newest **z** stream. This results in automatic, rolling Red Hat Quay updates to newer **z** streams with little to no downtime. Otherwise, the update must be manually approved before installation can begin.

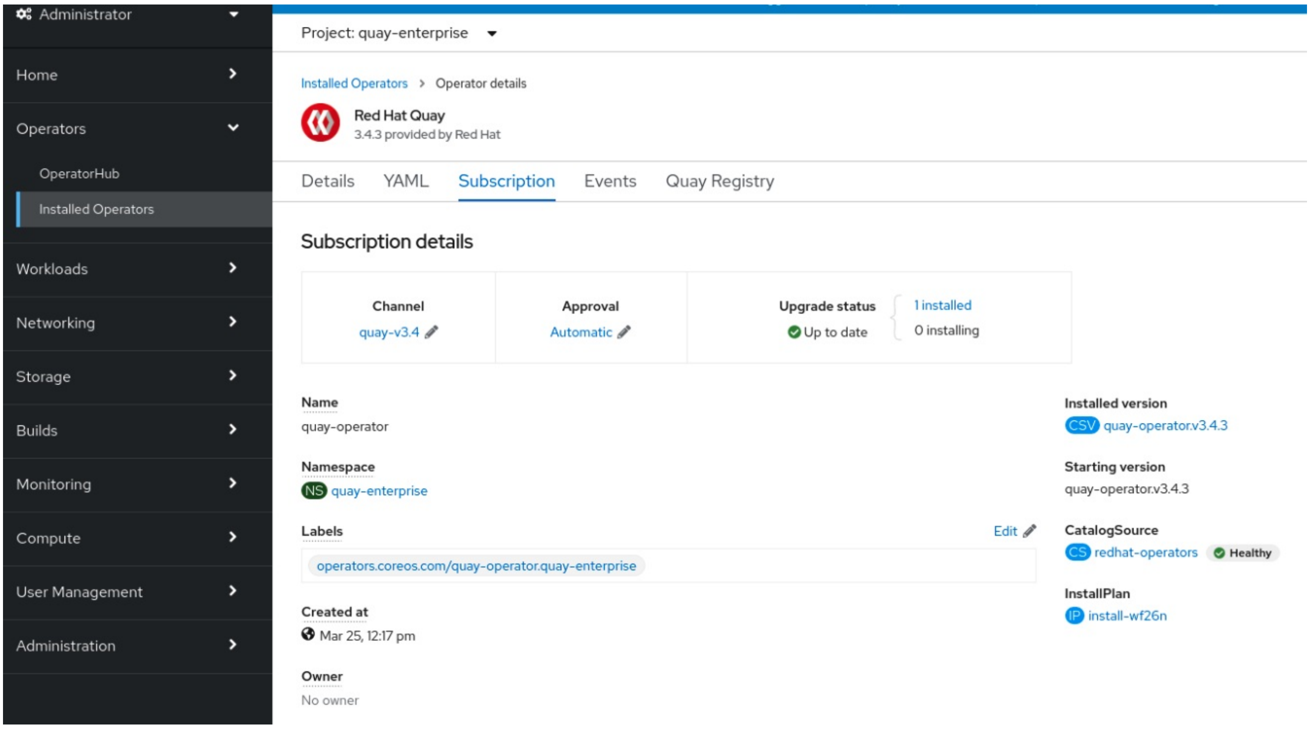
14.2.2. Changing the update channel for the Red Hat Quay Operator

The subscription of an installed Operator specifies an update channel, which is used to track and receive updates for the Operator. To upgrade the Red Hat Quay Operator to start tracking and receiving updates from a newer channel, change the update channel in the **Subscription** tab for the installed Red Hat 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.

14.2.3. 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 Red Hat Quay Operator has a pending upgrade, this status will be displayed in the list of Installed Operators. In the **Subscription** tab for the Red Hat 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**:



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

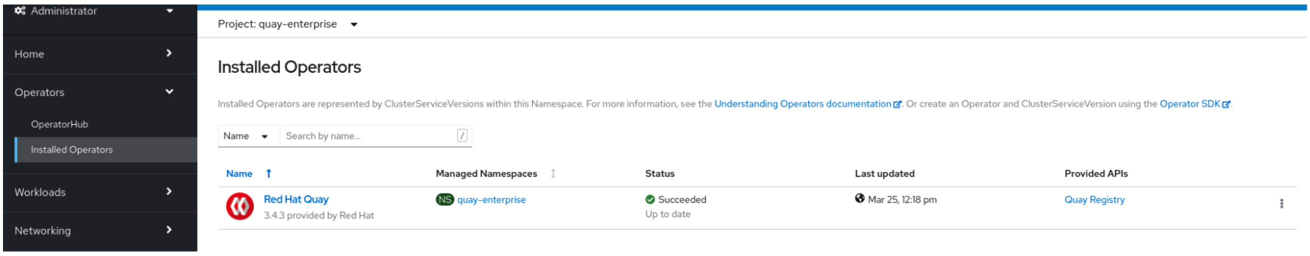
Installed version: CSV quay-operator.v3.4.3

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:



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

14.3. UPGRADING A QUAYREGISTRY RESOURCE

When the Red Hat 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.

14.4. 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**, use the following procedure.

Procedure

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. After the **status.registryEndpoint** of the new **QuayRegistry** is set, access Red Hat Quay and confirm that all data and settings were migrated successfully.

4. If everything works correctly, you can delete the **QuayEcosystem** and Kubernetes garbage collection will clean up all old resources.

14.4.1. 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**:

Procedure

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 PostgreSQL database, the upgrade process will migrate your data to a new PostgreSQL database managed by the upgraded Operator. Your old database will not be changed or removed but Red Hat 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.

14.4.2. Supported QuayEcosystem Configurations for Upgrades

The Red Hat Quay Operator reports 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 Red Hat Quay's **config.yaml** file.

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.