



## Red Hat Quay 3

# Red Hat Quay Operator features

Advanced Red Hat Quay Operator features



## Red Hat Quay 3 Red Hat Quay Operator features

---

Advanced Red Hat Quay Operator features

## Legal Notice

Copyright © 2023 Red Hat, Inc.

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

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

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

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

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

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

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

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

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

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

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

All other trademarks are the property of their respective owners.

## Abstract

Advanced Red Hat Quay Operator features

## Table of Contents

<b>CHAPTER 1. FEDERAL INFORMATION PROCESSING STANDARD (FIPS) READINESS AND COMPLIANCE</b>	<b>5</b>
1.1. ENABLING FIPS COMPLIANCE	5
<b>CHAPTER 2. CONSOLE MONITORING AND ALERTING</b>	<b>6</b>
2.1. DASHBOARD	6
2.2. METRICS	7
2.3. ALERTING	9
<b>CHAPTER 3. CONFIGURING RED HAT QUAY ON OPENSIFT CONTAINER PLATFORM</b>	<b>10</b>
3.1. EDITING THE CONFIG BUNDLE SECRET IN THE OPENSIFT CONTAINER PLATFORM CONSOLE	10
3.2. DETERMINING QUAYREGISTRY ENDPOINTS AND SECRETS	11
3.2.1. Locating the username and password for the config editor tool	12
3.3. DOWNLOADING THE EXISTING CONFIGURATION	13
3.3.1. Using the config editor endpoint to download the existing configuration	13
3.3.2. Using the config bundle secret to download the existing configuration	14
3.4. USING THE CONFIG BUNDLE TO CONFIGURE CUSTOM SSL/TLS CERTS	14
<b>CHAPTER 4. USING THE CONFIG TOOL TO RECONFIGURE RED HAT QUAY ON OPENSIFT CONTAINER PLATFORM</b>	<b>17</b>
4.1. ACCESSING THE CONFIG EDITOR	17
4.1.1. Retrieving the config editor credentials	17
4.1.2. Logging into the config editor	18
4.1.3. Changing configuration	19
4.2. MONITORING RECONFIGURATION IN THE RED HAT QUAY UI	20
4.2.1. QuayRegistry resource	20
4.2.2. Events	22
4.3. ACCESSING UPDATED INFORMATION AFTER RECONFIGURATION	23
4.4. CUSTOM SSL/TLS CERTIFICATES UI	23
4.5. EXTERNAL ACCESS TO THE REGISTRY	24
4.6. QUAYREGISTRY API	24
<b>CHAPTER 5. CLAIR FOR RED HAT QUAY</b>	<b>25</b>
5.1. CLAIR VULNERABILITY DATABASES	25
5.2. CLAIR ON OPENSIFT CONTAINER PLATFORM	25
5.3. TESTING CLAIR	25
5.4. ADVANCED CLAIR CONFIGURATION	26
5.4.1. Unmanaged Clair configuration	27
5.4.1.1. Running a custom Clair configuration with an unmanaged Clair database	27
5.4.1.2. Configuring a custom Clair database with an unmanaged Clair database	27
5.4.2. Running a custom Clair configuration with a managed Clair database	29
5.4.2.1. Setting a Clair database to managed	29
5.4.2.2. Configuring a custom Clair database with a managed Clair configuration	30
5.4.3. Clair in disconnected environments	31
5.4.3.1. Setting up Clair in a disconnected OpenShift Container Platform cluster	31
5.4.3.1.1. Installing the clairctl command line utility tool for OpenShift Container Platform deployments	31
5.4.3.1.2. Retrieving and decoding the Clair configuration secret for Clair deployments on OpenShift Container Platform	32
5.4.3.1.3. Exporting the updaters bundle from a connected Clair instance	32
5.4.3.1.4. Configuring access to the Clair database in the disconnected OpenShift Container Platform cluster	33
5.4.3.1.5. Importing the updaters bundle into the disconnected OpenShift Container Platform cluster	34
5.4.3.2. Setting up a self-managed deployment of Clair for a disconnected OpenShift Container Platform cluster	35

5.4.3.2.1. Installing the clairctl command line utility tool for a self-managed Clair deployment on OpenShift Container Platform	35
5.4.3.2.2. Deploying a self-managed Clair container for disconnected OpenShift Container Platform clusters	35
5.4.3.2.3. Exporting the updaters bundle from a connected Clair instance	36
5.4.3.2.4. Configuring access to the Clair database in the disconnected OpenShift Container Platform cluster	36
5.4.3.2.5. Importing the updaters bundle into the disconnected OpenShift Container Platform cluster	37
5.4.4. Enabling Clair CRDA	38
5.4.5. Mapping repositories to Common Product Enumeration information	38
5.4.5.1. Mapping repositories to Common Product Enumeration example configuration	39
5.5. DEPLOYING RED HAT QUAY ON INFRASTRUCTURE NODES	39
5.5.1. Labeling and tainting nodes for infrastructure use	39
5.5.2. Creating a project with node selector and tolerations	41
5.5.3. Installing the Red Hat Quay Operator in the namespace	41
5.5.4. Creating the Red Hat Quay registry	42
5.6. ENABLING MONITORING WHEN THE RED HAT QUAY OPERATOR IS INSTALLED IN A SINGLE NAMESPACE	42
5.6.1. Creating a cluster monitoring config map	43
5.6.2. Creating a user-defined workload monitoring ConfigMap object	43
5.6.3. Enable monitoring for user-defined projects	44
5.6.4. Creating a Service object to expose Red Hat Quay metrics	45
5.6.5. Creating a ServiceMonitor object	46
5.6.6. Viewing metrics in OpenShift Container Platform	46
5.7. RESIZING MANAGED STORAGE	47
5.7.1. Resizing the NooBaa PVC	47
5.8. CUSTOMIZING DEFAULT OPERATOR IMAGES	48
5.8.1. Environment Variables	48
5.8.2. Applying overrides to a running Operator	48
5.9. AWS S3 CLOUDFRONT	49
<b>CHAPTER 6. RED HAT QUAY BUILD ENHANCEMENTS</b>	<b>50</b>
6.1. RED HAT QUAY BUILD LIMITATIONS	50
6.2. CREATING A RED HAT QUAY BUILDERS ENVIRONMENT WITH OPENSIFT CONTAINER PLATFORM	50
6.2.1. OpenShift Container Platform TLS component	50
6.2.2. Using OpenShift Container Platform for Red Hat Quay builders	50
6.2.2.1. Preparing OpenShift Container Platform for virtual builders	50
6.2.2.2. Manually adding SSL/TLS certificates	54
6.2.2.2.1. Creating and signing certificates	54
6.2.2.2.2. Setting TLS to unmanaged	55
6.2.2.2.3. Creating temporary secrets	56
6.2.2.2.4. Copying secret data to the configuration YAML	56
6.2.2.3. Using the UI to create a build trigger	58
6.2.2.4. Modifying your AWS S3 storage bucket	59
6.2.2.5. Modifying your Google Cloud Platform object bucket	60
<b>CHAPTER 7. GEO-REPLICATION</b>	<b>63</b>
7.1. GEO-REPLICATION FEATURES	63
7.2. GEO-REPLICATION REQUIREMENTS AND CONSTRAINTS	63
7.2.1. Setting up geo-replication on OpenShift Container Platform	64
7.2.1.1. Configuring geo-replication for the Red Hat Quay Operator on OpenShift Container Platform	65
7.2.2. Mixed storage for geo-replication	68
7.3. UPGRADING A GEO-REPLICATION DEPLOYMENT OF THE RED HAT QUAY OPERATOR	68
7.3.1. Removing a geo-replicated site from your Red Hat Quay Operator deployment	70

<b>CHAPTER 8. BACKING UP AND RESTORING RED HAT QUAY MANAGED BY THE RED HAT QUAY OPERATOR</b>	<b>72</b>
8.1. BACKING UP RED HAT QUAY	72
8.1.1. Red Hat Quay configuration backup	72
8.1.2. Scaling down your Red Hat Quay deployment	73
8.1.3. Backing up the Red Hat Quay managed database	75
8.1.3.1. Backing up the Red Hat Quay managed object storage	75
8.1.4. Scale the Red Hat Quay deployment back up	76
8.2. RESTORING RED HAT QUAY	77
8.2.1. Restoring Red Hat Quay and its configuration from a backup	78
8.2.2. Scaling down your Red Hat Quay deployment	79
8.2.3. Restoring your Red Hat Quay database	80
8.2.4. Restore your Red Hat Quay object storage data	81
8.2.5. Scaling up your Red Hat Quay deployment	82
<b>CHAPTER 9. ENABLING OCI SUPPORT WITH THE RED HAT QUAY OPERATOR</b>	<b>84</b>
<b>CHAPTER 10. VOLUME SIZE OVERRIDES</b>	<b>85</b>
<b>CHAPTER 11. SCANNING POD IMAGES WITH THE CONTAINER SECURITY OPERATOR</b>	<b>86</b>
11.1. DOWNLOADING AND RUNNING THE CONTAINER SECURITY OPERATOR IN OPENSIFT CONTAINER PLATFORM	86
11.2. QUERY IMAGE VULNERABILITIES FROM THE CLI	88
<b>CHAPTER 12. DEPLOYING IPV6 ON THE RED HAT QUAY OPERATOR</b>	<b>90</b>
12.1. ENABLING THE IPV6 PROTOCOL FAMILY	90
12.2. IPV6 LIMITATIONS	91
<b>CHAPTER 13. ADDING CUSTOM SSL/TLS CERTIFICATES WHEN RED HAT QUAY IS DEPLOYED ON KUBERNETES</b>	<b>92</b>
<b>CHAPTER 14. UPGRADING THE RED HAT QUAY OPERATOR OVERVIEW</b>	<b>93</b>
14.1. OPERATOR LIFECYCLE MANAGER	93
14.2. UPGRADING THE QUAY OPERATOR	93
14.2.1. Upgrading Quay	94
14.2.2. Updating Red Hat Quay from 3.8 → 3.9	94
14.2.3. Upgrading directly from 3.3.z or 3.4.z to 3.6	96
14.2.3.1. Upgrading with edge routing enabled	96
14.2.3.2. Upgrading with custom SSL/TLS certificate/key pairs without Subject Alternative Names	96
14.2.3.3. Configuring Clair v4 when upgrading from 3.3.z or 3.4.z to 3.6 using the Red Hat Quay Operator	97
14.2.4. Swift configuration when upgrading from 3.3.z to 3.6	97
14.2.5. Changing the update channel for the Red Hat Quay Operator	97
14.2.6. Manually approving a pending Operator upgrade	97
14.3. UPGRADING A QUAYREGISTRY	98
14.4. UPGRADING A QUAYECOSYSTEM	98
14.4.1. Reverting QuayEcosystem Upgrade	99
14.4.2. Supported QuayEcosystem Configurations for Upgrades	99
ADDITIONAL RESOURCES	100





# 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 using the Red Hat Quay Operator, OpenShift Container Platform is version 4.10 or later.
- Your Red Hat Quay version is 3.5.0 or later.
- 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.

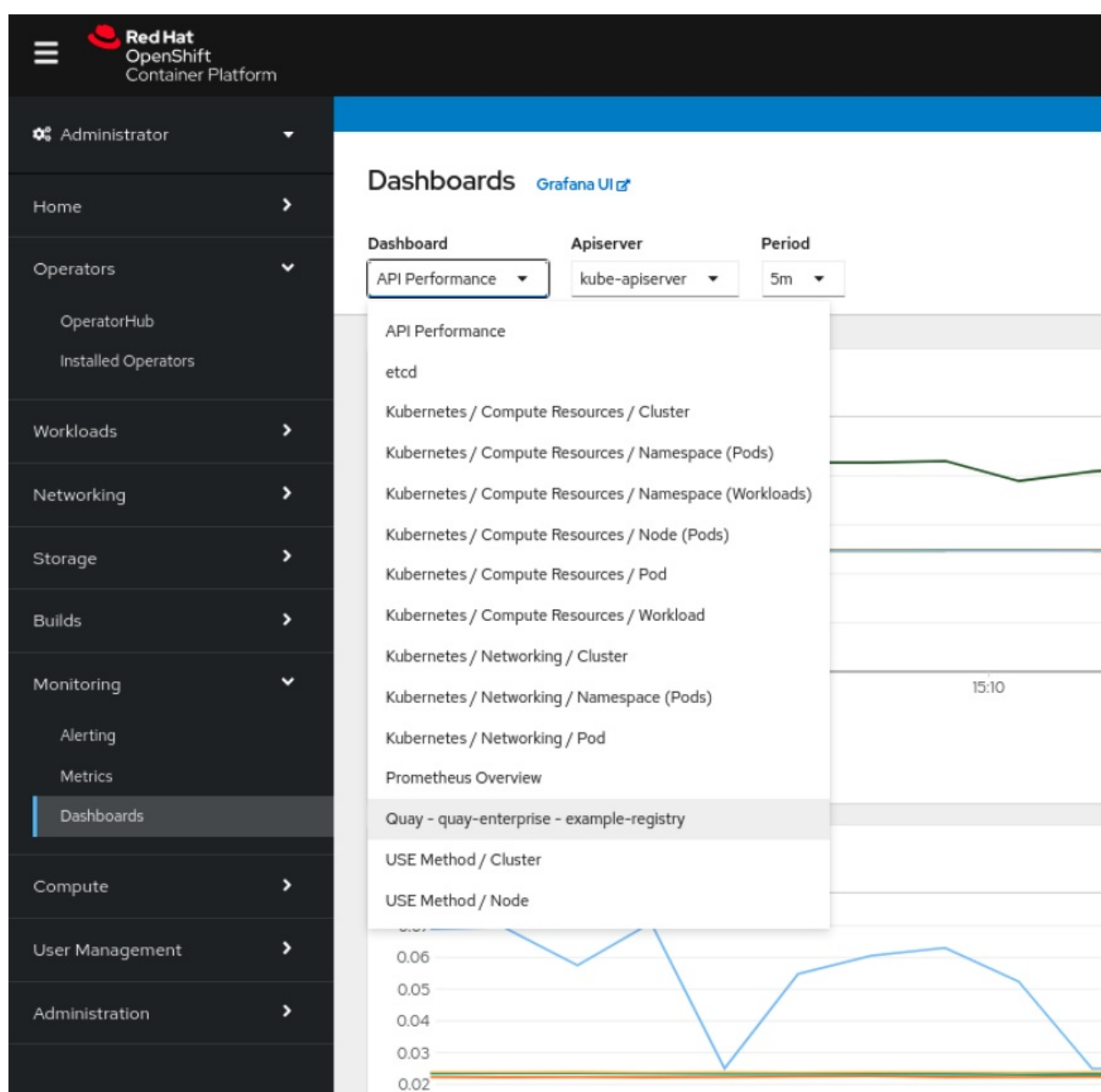


### NOTE

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

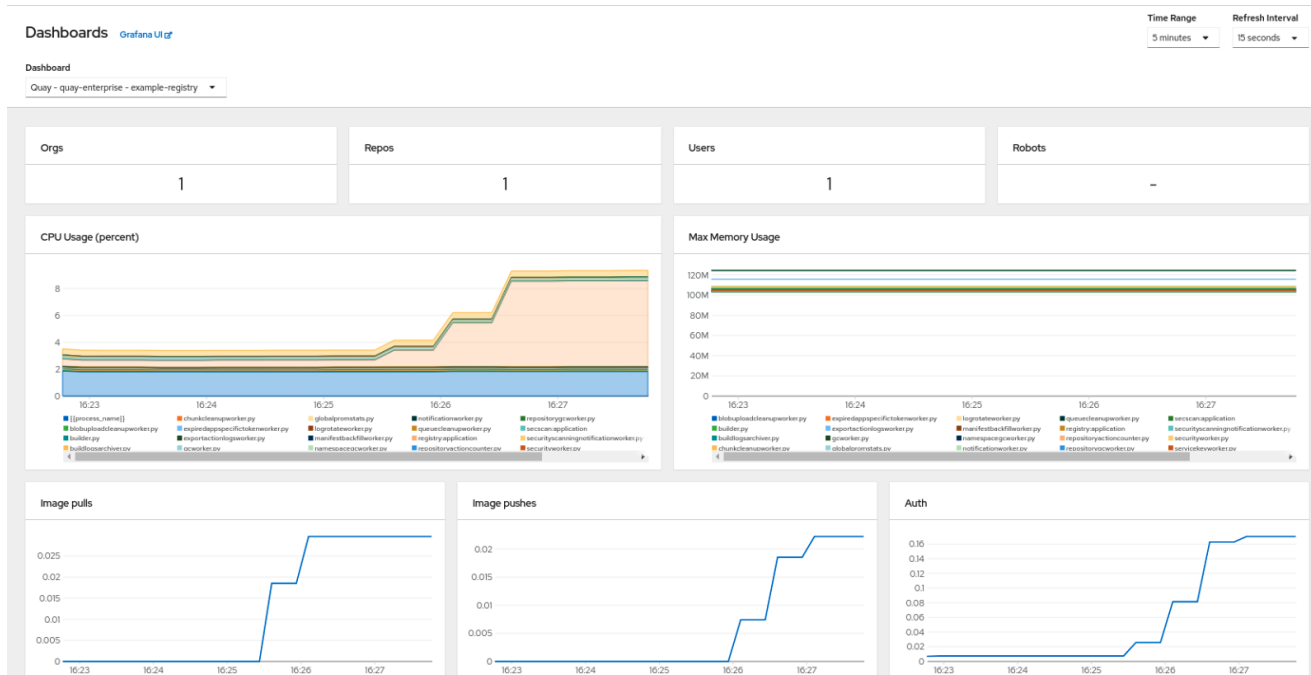
### 2.1. DASHBOARD

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



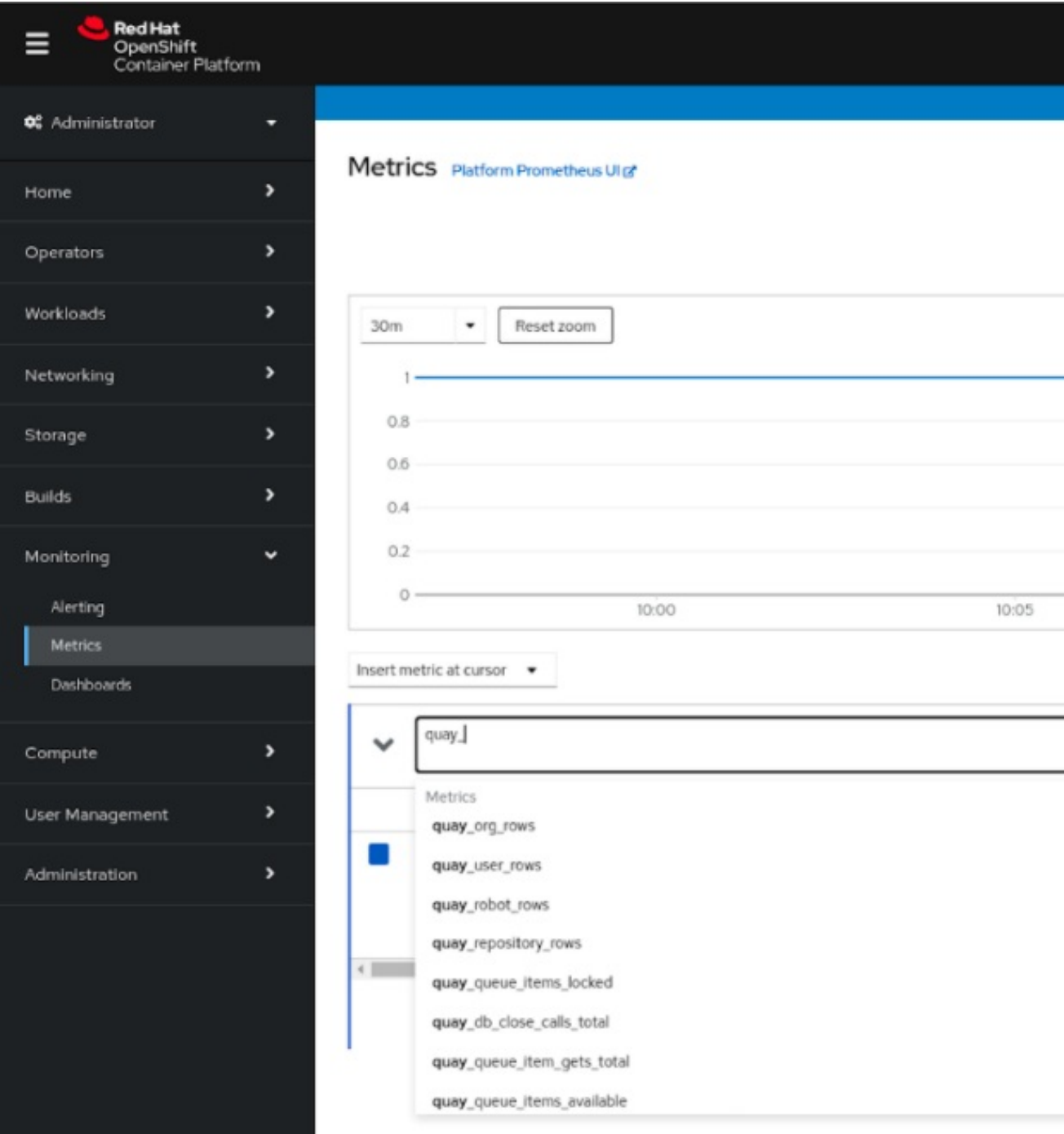
The dashboard shows various statistics including the following:

- 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

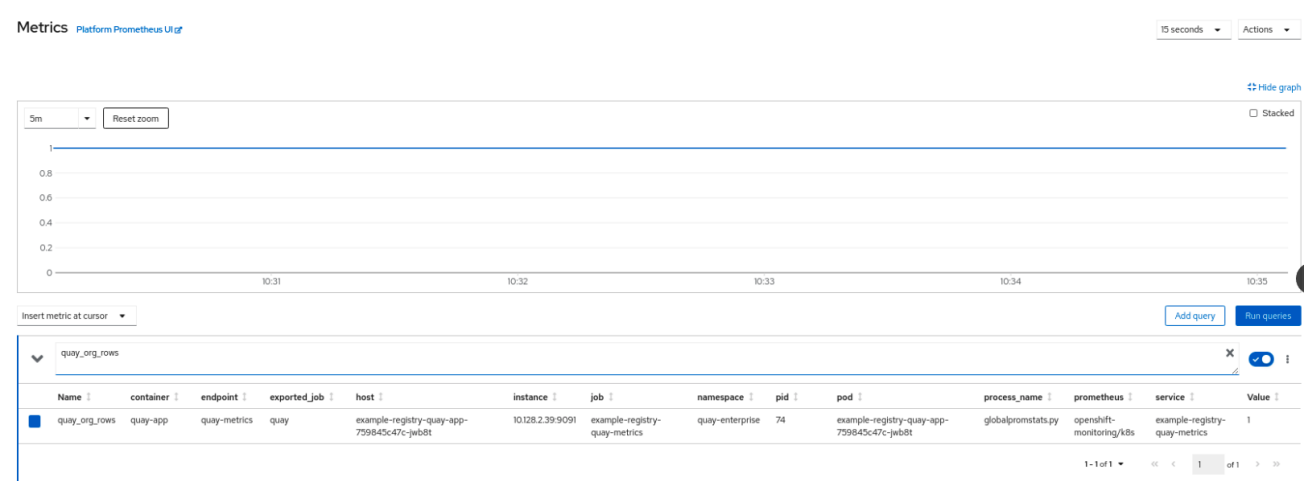


## 2.2. METRICS

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**:



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:

The screenshot shows the Red Hat OpenShift Container Platform console. The left sidebar has a menu with 'Alerting' selected under the 'Monitoring' section. The main content area is titled 'Alerting' and shows the 'Alerting rules' tab. A filter is applied to the 'Name' field with the value 'quay'. Below the filter, there are buttons for 'Source', 'Platform', and 'Name', each with a close icon. A 'Clear all filters' button is also present. The table of alerting rules is as follows:

Name	Severity
KubeQuotaFullyUsed	Info
QuayPodFrequentlyRestarting	Warning
ThanosQueryInstantLatencyHigh	Critical
ThanosQueryRangeLatencyHigh	Critical

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

The screenshot shows the details of the 'QuayPodFrequentlyRestarting' alerting rule. The rule is categorized as 'Warning'. The details are as follows:

Alerting rule details	Source	For	Expression
<p><b>Name</b> QuayPodFrequentlyRestarting</p> <p><b>Severity</b> Warning</p> <p><b>Description</b> Pod {{ {{ \$labels.namespace }} }}/{{ {{ \$labels.pod }} }} was restarted {{ {{ \$value }} }} times within the last hour</p> <p><b>Message</b> Quay Pod is restarting frequently</p> <p><b>Labels</b> prometheus=openshift-monitoring/k8s severity=warning</p>	Platform	10m	<pre>increase(kube_pod_container_status_restarts_total{pod=~".*-quay-app-*"}[1h]) &gt; 5</pre>

## CHAPTER 3. CONFIGURING RED HAT QUAY ON OPENSIFT CONTAINER PLATFORM

After deployment, you can configure the Red Hat Quay application by editing the Red Hat Quay configuration bundle secret **spec.configBundleSecret**. You can also change the managed status of components in the **spec.components** object of the **QuayRegistry** resource.

Alternatively, you can use the config editor UI to configure the Red Hat Quay application. For more information, see [Using the config tool to reconfigure Red Hat Quay on OpenShift Container Platform](#).

### 3.1. EDITING THE CONFIG BUNDLE SECRET IN THE OPENSIFT CONTAINER PLATFORM CONSOLE

Use the following procedure to edit the config bundle secret in the OpenShift Container Platform console.

#### Procedure

1. On the Red Hat Quay Registry overview screen, click the link for the **Config Bundle Secret**

Project: quay-enterprise ▾

Installed Operators > quay-operatorv3.7.0-rc.3 > QuayRegistry details

**QR** example-registry

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

#### Quay Registry overview

<b>Name</b> example-registry	<b>Current Version</b> 3.7.0-rc.3
<b>Namespace</b> quay-enterprise	<b>Config Editor Credentials Secret</b> example-registry-quay-config-editor-credentials-fg2gdtm24
<b>Labels</b> No labels <a href="#">Edit</a>	<b>Registry Endpoint</b> example-registry-quay-quay-enterprise.apps.docs.gcp.quaydev.org
<b>Annotations</b> 0 annotations <a href="#">Edit</a>	<b>Config Editor Endpoint</b> example-registry-quay-config-editor-quay-enterprise.apps.docs.gcp.quaydev.org
<b>Created at</b> 28 Apr 2022, 18:47	
<b>Owner</b> No owner	

---

<b>Config Bundle Secret</b> <a href="#">init-config-bundle-secret</a>	<b>Components</b> <table border="1"> <thead> <tr> <th>Kind</th> </tr> </thead> <tbody> <tr> <td>quay</td> </tr> </tbody> </table>	Kind	quay
Kind			
quay			

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

Project: quay-enterprise ▾

Secrets > Secret details

**init-config-bundle-secret** Add Secret to workload Actions ▾

**Details** **YAML**

**Secret details**

<b>Name</b> init-config-bundle-secret	<b>Type</b> Opaque
<b>Namespace</b> quay-enterprise	
<b>Labels</b> No labels <span>Edit</span>	
<b>Annotations</b> 0 annotations	
<b>Created at</b> 28 Apr 2022, 18:46	
<b>Owner</b> No owner	

Edit labels  
Edit annotations  
Edit Secret  
Delete Secret

3. Modify the configuration and save the changes.

Project: quay-enterprise ▾

## Edit key/value secret

### Secret name \*

init-config-bundle-secret

Unique name of the new secret.

### Key \*

config.yaml

### Value

Browse...

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

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

+ Add key/value

Save

Cancel

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

## 3.2. DETERMINING QUAYREGISTRY ENDPOINTS AND SECRETS

Use the following procedure to find **QuayRegistry** endpoints and secrets.

### Procedure

1. You can examine the **QuayRegistry** resource, using **oc describe quayregistry** or **oc get quayregistry -o yaml**, to find the current endpoints and secrets by entering the following command:

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

## Example output

```

apiVersion: quay.redhat.com/v1
kind: QuayRegistry
metadata:
  ...
  name: example-registry
  namespace: quay-enterprise
  ...
spec:
  components:
    - kind: quay
      managed: true
    ...
    - kind: clairpostgres
      managed: true
  configBundleSecret: init-config-bundle-secret 1
status:
  configEditorCredentialsSecret: example-registry-quay-config-editor-credentials-fg2gdgtm24 2
  configEditorEndpoint: https://example-registry-quay-config-editor-quay-
enterprise.apps.docs.gcp.quaydev.org 3
  currentVersion: 3.7.0
  lastUpdated: 2022-05-11 13:28:38.199476938 +0000 UTC
  registryEndpoint: https://example-registry-quay-quay-enterprise.apps.docs.gcp.quaydev.org 4

```

- 1** The config bundle secret, containing the **config.yaml** file and any SSL/TLS certificates.
- 2** The secret containing the username (typically **quayconfig**) and the password for the config editor tool.
- 3** The URL for the config editor tool, for browser access to the config tool, and for the configuration API.
- 4** The URL for your registry, for browser access to the registry UI, and for the registry API endpoint.

### 3.2.1. Locating the username and password for the config editor tool

Use the following procedure to locate the username and password for the config editor tool.

#### Procedure

1. Enter the following command to retrieve the secret:

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

## Example output

```

apiVersion: v1
data:

```



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

2. Decode the username by entering the following command:

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

#### Example output

```
quayconfig
```

3. Decode the password by entering the following command:

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

#### Example output

```
JFpBEJMCtkPTfjxt
```

## 3.3. DOWNLOADING THE EXISTING CONFIGURATION

The following procedures detail how to download the existing configuration using different strategies.

### 3.3.1. Using the config editor endpoint to download the existing configuration

Use the following procedure to download the existing configuration through the config editor endpoint.

#### Procedure

- Enter the following command, specifying the username and password for the config editor, to download the existing configuration:

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

#### Example output

```
{
  "config.yaml": {
    "ALLOW_PULLS_WITHOUT_STRICT_LOGGING": false,
    "AUTHENTICATION_TYPE": "Database",
    ...
    "USER_RECOVERY_TOKEN_LIFETIME": "30m"
  },
  "certs": {
    "extra_ca_certs/service-ca.crt":
    "LS0tLS1CRUdJTiBDRVJUSUZJQ0FURSB0tLS0tCk1JSURVVENDQWptZ0F3SUJBZ0lJRE9k
    WFhuUXFjMUf3RFFZSkvWklodmNOQVFFTEJRQXdOakUwTURJR0ExVUUKQXd3cmIzQ
    mxibk5vYVdaMEExYTmxjblpwWTJVdGMvYnlkbWx1WnkxemFXZHVaWEpBTVRZek1UYzNPRE
```

```
V3TXpBZQpGdzB5TVRBNU1UWXdOeIF4TkRKYUZ..."
}
```

### 3.3.2. Using the config bundle secret to download the existing configuration

You can use the config bundle secret to download the existing configuration.

#### Procedure

1. Obtain the secret data by entering the following command:

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

#### Example output

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

2. Enter the following command to decode the data:

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

#### Example output

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

## 3.4. USING THE CONFIG BUNDLE TO CONFIGURE CUSTOM SSL/TLS CERTS

You can configure custom SSL/TLS certificates before the initial deployment, or after Red Hat Quay is deployed on OpenShift Container Platform. This is done by creating or updating the config bundle secret.

If you are adding the certificates to an existing deployment, you must include the existing **config.yaml** file in the new config bundle secret, even if you are not making any configuration changes.

Use the following procedure to add custom SSL/TLS certificates.

#### Procedure

1. In your **QuayRegistry** YAML file, set **kind: tls** to **managed:false**, for example:

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

2. Navigate to the **Events** page, which should reveal that the change is blocked until you set up the appropriate config. 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'
```

3. Create the secret using embedded data or by using files.
  - a. Embed the configuration details directly in the **Secret** resource YAML file. For example:

#### custom-ssl-config-bundle.yaml

```
apiVersion: v1
kind: Secret
metadata:
  name: custom-ssl-config-bundle-secret
  namespace: quay-enterprise
data:
  config.yaml: |
    FEATURE_USER_INITIALIZE: true
    BROWSER_API_CALLS_XHR_ONLY: false
    SUPER_USERS:
      - quayadmin
    FEATURE_USER_CREATION: false
    FEATURE_QUOTA_MANAGEMENT: true
    FEATURE_PROXY_CACHE: true
    FEATURE_BUILD_SUPPORT: true
    DEFAULT_SYSTEM_REJECT_QUOTA_BYTES: 102400000
  extra_ca_cert_my-custom-ssl.crt: |
    -----BEGIN CERTIFICATE-----
    MIIDsDCCApigAwIBAgIUCqIzkHjF5i5TXLFy+sepFrZr/UswDQYJKoZIhvcNAQEL
    BQAwbzELMAkGA1UEBhMCSUUxDzANBgNVBAGMBkdBTfDdWTEPMA0GA1UEBwwG
    R0FM
    ....
    -----END CERTIFICATE-----
```

- b. Create the secret from the YAML file:

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

```
..
```

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

- a. Enter the following command to create a generic **Secret** object that contains the **config.yaml** file and a **custom-ssl.crt** file:

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

- b. Create or update the **QuayRegistry** YAML file, referencing the created **Secret**, for example:

#### Example QuayRegistry YAML file

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

- c. Deploy or update the registry using the YAML file by entering the following command:

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


# CHAPTER 4. USING THE CONFIG TOOL TO RECONFIGURE RED HAT QUAY ON OPENSIFT CONTAINER PLATFORM

## 4.1. ACCESSING THE CONFIG EDITOR

In the **Details** section of the **QuayRegistry** object, the endpoint for the config editor is available, along with a link to the **Secret** object that contains the credentials for logging into the config editor. For example:

Project: openshift-operators

[Installed Operators](#) > [quay-operator.v3.5.2](#) > QuayRegistry details

 example

Actions


DetailsYAMLResourcesEvents

Quay Registry overview

Name

example

Namespace

 openshift-operators

Labels

No labels

Annotations

1 annotation

Created at

Jun 24, 5:33 pm


Owner

No owner

Current Version

3.5.2

Config Editor Credentials Secret

 example-quay-config-editor-credentials-9ffggtfc7

Registry Endpoint

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

Config Editor Endpoint

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

### 4.1.1. Retrieving the config editor credentials

Use the following procedure to retrieve the config editor credentials.

#### Procedure

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

17

Project: openshift-operators ▼

---

[Secrets](#) > Secret details

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

Managed by example

---

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

---

### Secret details

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

**Namespace**  
 openshift-operators

**Type**  
Opaque

**Labels** [Edit](#)

quay-operator/quayregistry=example

**Annotations**  
[4 annotations](#)

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

**Owner**  
 example

---

### Data

[Reveal values](#)

**password**

.....

**username**

.....

- In the **Data** section of the **Secret** details page, click **Reveal values** to see the credentials for logging into the config editor. For example:

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

**password**

Zrl1N6tCtZeVww4q

**username**

quayconfig

## 4.1.2. Logging into the config editor

Use the following procedure to log into the config editor.

### Procedure

- Navigate the config editor endpoint. When prompted, enter the username, for example, **quayconfig**, and the password. For example:

Red Hat Quay Setup

⚙️

Custom SSL Certificates

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

Upload certificates:

Select file

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

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

⚙️

Basic Configuration

Registry Title:

Red Hat Quay

Name of registry to be displayed in the Contact Page.

Registry Title Short:

Red Hat Quay

Enterprise Logo URL:

/static/img/RH\_Logo\_Quay\_Black\_UX-horizontal.svg

🖼️

Enter the full URL to your company's logo.

Contact Information:

URL

http://some/url

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

☁️

Server Configuration

Server Hostname:

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

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

TLS:

Red Hat Quay handles TLS

▼

▶ Validate Configuration Changes

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

### 4.1.3. Changing configuration

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

#### Procedure

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

🕒

Time Machine

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

Allowed expiration periods:

4w

Add

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

Default expiration period:

2w

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

Allow users to select expiration:

☒ Enable Expiration Configuration

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

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

19

## Validating configuration

 CONFIGURATION VALIDATED

 Configuration Validated



Continue Editing

Download

Reconfigure Quay

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

## Validating configuration

 CONFIGURATION VALIDATED  
 CONFIG SENT TO OPERATOR

 Configuration Validated

Continue Editing

Download

Reconfigure Quay



## NOTE

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

## 4.2. MONITORING RECONFIGURATION IN THE RED HAT QUAY UI

You can monitor the reconfiguration of Red Hat Quay in real-time.

### 4.2.1. QuayRegistry resource

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



Project: quay-enterprise ▾

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

 example-registry

 Details YAML Resources Events

```

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

```

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

Save

Reload

Cancel

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

Project: quay-enterprise ▾

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

 example-registryDetails YAML Resources Events

```

1  apiVersion: quay.redhat.com/v1
2  kind: QuayRegistry
3  metadata:
4    selfLink: >-
5      /apis/quay.redhat.com/v1/namespaces/quay-enterprise/quayregistries/example-registry
6    resourceVersion: '79051'
7    name: example-registry
8    uid: 0a77c77c-b560-4d52-9d8a-ba8481ab4d04
9    creationTimestamp: '2021-09-24T10:13:02Z'
10   generation: 7
11   managedFields: -
43   namespace: quay-enterprise
44   finalizers:
45     - quay-operator/finalizer
46 spec:
47   components: -
66   configBundleSecret: example-registry-quay-config-bundle-zb9c7
67 status:
68   conditions:
69     - lastTransitionTime: '2021-09-24T10:14:40Z'
70       lastUpdateTime: '2021-09-24T10:14:40Z'
71       message: all registry component healthchecks passing
72       reason: HealthChecksPassing
73       status: 'True'
74       type: Available
75     - lastTransitionTime: '2021-09-24T11:23:02Z'
76       lastUpdateTime: '2021-09-24T11:23:02Z'
77       message: all objects created/updated successfully
78       reason: ComponentsCreationSuccess
79       status: 'False'
80       type: RolloutBlocked
81   configEditorCredentialsSecret: example-registry-quay-config-editor-credentials-gbtbkh94kh
82   configEditorEndpoint: >-
83     https://example-registry-quay-config-editor-quay-enterprise.apps.docs.quayteam.org
84   currentVersion: 3.6.0
85   lastUpdated: '2021-09-24 11:23:02.084685976 +0000 UTC'
86   registryEndpoint: 'https://example-registry-quay-quay-enterprise.apps.docs.quayteam.org'
87   unhealthyComponents: {}
88

```

Save

Reload

Cancel

#### 4.2.2. Events

The **Events** tab for the **QuayRegistry** shows some events related to the redeployment. For example:

Streaming events...

Showing 491 events

example-registry-quay-app

Generated from horizontal-pod-autoscaler

failed to get cpu utilization: did not receive metrics for any ready pods

quay-enterprise

Sep 24, 12:16 pm

29 times in the last an hour

example-registry-quay-app-c7698bfc-lsx2

Generated from kubelet on docs-k95iz-worker-d-tzg54c.quay-devel.internal

Readiness probe failed: Get "http://10.128.2.40:8080/health/instance": dial tcp 10.128.2.40:8080: connect: connection refused

quay-enterprise

Sep 24, 12:16 pm

example-registry-quay-app

Generated from deployment-controller

Scaled down replica set example-registry-quay-app-c7698bfc to 0

quay-enterprise

a few seconds ago

example-registry-quay-app-c7698bfc-lsx2

Generated from kubelet on docs-k95iz-worker-d-tzg54c.quay-devel.internal

Stopping container quay-app

quay-enterprise

a few seconds ago

example-registry-quay-app-c7698bfc

Generated from replicaset-controller

Deleted pod: example-registry-quay-app-c7698bfc-lsx2

quay-enterprise

a few seconds ago

Streaming events, for all resources in the namespace that are affected by the reconfiguration, are available in the OpenShift Container Platform console under **Home → Events**. For example:

II

</

## 4.3. ACCESSING UPDATED INFORMATION AFTER RECONFIGURATION

Use the following procedure to access the updated **config.yaml** file using the Red Hat Quay UI and the config bundle.

### Procedure

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

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

## 4.4. CUSTOM SSL/TLS CERTIFICATES UI

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

#### Custom SSL Certificates

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

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

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

Upload certificates:

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

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

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

#### Custom SSL Certificates

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

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

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

Upload certificates:

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

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

## 4.5. EXTERNAL ACCESS TO THE REGISTRY

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

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

## 4.6. QUAYREGISTRY API

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

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

**QuayRegistry** API fields are outlined in the following sections.

## CHAPTER 5. CLAIR FOR RED HAT QUAY

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

### 5.1. CLAIR VULNERABILITY DATABASES

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

- Ubuntu Oval database
- Debian Oval database
- 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
- Pyup.io (Python) database

For information about how Clair does security mapping with the different databases, see [ClairCore Severity Mapping](#).

### 5.2. CLAIR ON OPENSIFT 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 will install or upgrade a Clair deployment along with your Red Hat Quay deployment and configure Clair automatically.

### 5.3. 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
```

- 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
```

- 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
```

- Log in to your Red Hat Quay deployment through the UI.
- Click the repository name, for example, **quayadmin/ubuntu**.
- 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

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

## Report details

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.

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

## 5.4. ADVANCED CLAIR CONFIGURATION

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

### 5.4.1. 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.

#### 5.4.1.1. 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
```

#### 5.4.1.2. Configuring a custom Clair database with an unmanaged Clair database

The Red Hat Quay Operator for 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
```

### 5.4.2. 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 a disconnected environment. For more information about running Clair in a disconnected environment, see [Configuring access to the Clair database in the air-gapped OpenShift cluster](#).

**NOTE**

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

#### 5.4.2.1. 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

```

#### 5.4.2.2. Configuring a custom Clair database with a managed Clair configuration

The Red Hat Quay Operator for 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.

- 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>
```

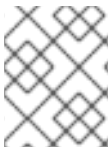
#### Example output

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

### 5.4.3. Clair in disconnected environments

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 importing the updater data on the isolated host into Clair.

Use this guide to deploy Clair in a disconnected environment.



#### 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".

#### 5.4.3.1. 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.

##### 5.4.3.1.1. 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
```

#### 5.4.3.1.2. 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
---
```

#### 5.4.3.1.3. 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
```

#### 5.4.3.1.4. 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 ❶
  scanlock_retry: 10
  layer_scan_concurrency: 5
  migrations: true
  scanner:
    repo:
      rhel-repository-scanner: ❷
      repo2cpe_mapping_file: /data/cpe-map.json
    package:
      rhel_containerscanner: ❸
      name2repos_mapping_file: /data/repo-map.json
```

- ❶ Replace the value of the **host** in the multiple **connstring** fields with **localhost**.
- ❷ For more information about the **rhel-repository-scanner** parameter, see "Mapping repositories to Common Product Enumeration information".
- ❸ For more information about the **rhel\_containerscanner** parameter, see "Mapping repositories to Common Product Enumeration information".

#### 5.4.3.15. 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
```

### 5.4.3.2. 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.

#### 5.4.3.2.1. 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
```

#### 5.4.3.2.2. 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:v3.9.0
```

#### 5.4.3.2.3. 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
```

#### 5.4.3.2.4. 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
...
```

- 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
```

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

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

- ❶ Replace the value of the **host** in the multiple **connstring** fields with **localhost**.
- ❷ For more information about the **rhel-repository-scanner** parameter, see "Mapping repositories to Common Product Enumeration information".
- ❸ For more information about the **rhel\_containerscanner** parameter, see "Mapping repositories to Common Product Enumeration information".

#### 5.4.3.2.5. 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
```

#### 5.4.4. Enabling Clair CRDA

Java scanning depends on a public, Red Hat provided API service called Code Ready Dependency Analytics (CRDA). CRDA is only available with internet access and is not enabled by default.

Use the following procedure to integrate the CRDA service with a custom API key and enable CRDA for Java and Python scanning.

##### Prerequisites

- Red Hat Quay 3.7 or greater

##### Procedure

1. Submit [the API key request form](#) to obtain the Quay-specific CRDA remote matcher.
2. Set the CRDA configuration in your **clair-config.yaml** file:

```
matchers:
  config:
    crda:
      url: https://gw.api.openshift.io/api/v2/
      key: <CRDA_API_KEY> ❶
      source: <QUAY_SERVER_HOSTNAME> ❷
```

- ❶ Insert the Quay-specific CRDA remote matcher from [the API key request form](#) here.
- ❷ The hostname of your Quay server.

#### 5.4.5. Mapping repositories to Common Product Enumeration information

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 5.1. Clair CPE mapping files**

CPE	Link to JSON mapping file
<b>repos2cpe</b>	<a href="#">Red Hat Repository-to-CPE JSON</a>
<b>names2repos</b>	<a href="#">Red Hat Name-to-Repos JSON</a>

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 Operator deployments on OpenShift Container Platform and Clair 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.

#### 5.4.5.1. 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](#) .

## 5.5. 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.

### 5.5.1. Labeling and tainting nodes for infrastructure use

Use the following procedure to label and taint 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+ba45583				

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
```

### 5.5.2. Creating a project with node selector and tolerations

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



#### NOTE

If you have already deployed Red Hat Quay using the Operator, remove the installed Operator and any specific namespaces that you created for the deployment.

#### Procedure

1. Create a project resource, specifying a node selector and toleration. For example:

##### quay-registry.yaml

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

2. Enter the following command to create the project:

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

#### Example output

```
project.project.openshift.io/quay-registry created
```

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

### 5.5.3. Installing the Red Hat Quay Operator in the namespace

Use the following procedure to install the Red Hat Quay Operator in the 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 this example, we are using **quay-registry**. This results in the 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

### 5.5.4. 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-glb7	1/1	Running	1	5m57s		10.129.2.17 user1-jcnp6-worker-c-pwxfp.c.quay-devel.internal
example-registry-quay-config-editor-7bf9bcc7b-dpc6d	1/1	Running	0	5m57s		10.131.0.23 user1-jcnp6-worker-d-h5tv2.c.quay-devel.internal
example-registry-quay-database-8dc7cfd69-dr2cc	1/1	Running	0	5m43s		10.129.2.18 user1-jcnp6-worker-c-pwxfp.c.quay-devel.internal
example-registry-quay-mirror-78df886bcc-v75p9	1/1	Running	0	5m16s		10.131.0.24 user1-jcnp6-worker-d-h5tv2.c.quay-devel.internal
example-registry-quay-postgres-init-8s8g9	0/1	Completed	0	5m54s		10.130.2.79 user1-jcnp6-worker-d-m9gg4.c.quay-devel.internal
example-registry-quay-redis-5688ddcdb6-ndp4t	1/1	Running	0	5m56s		10.130.2.78 user1-jcnp6-worker-d-m9gg4.c.quay-devel.internal
quay-operator.v3.4.1-6f6597d8d8-bd4dp	1/1	Running	0	22m		10.131.0.16 user1-jcnp6-worker-d-h5tv2.c.quay-devel.internal

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

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.

### 5.6.1. 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 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
```

### 5.6.2. 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
```

### 5.6.3. 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

### 5.6.4. 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
```

### 5.6.5. 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:

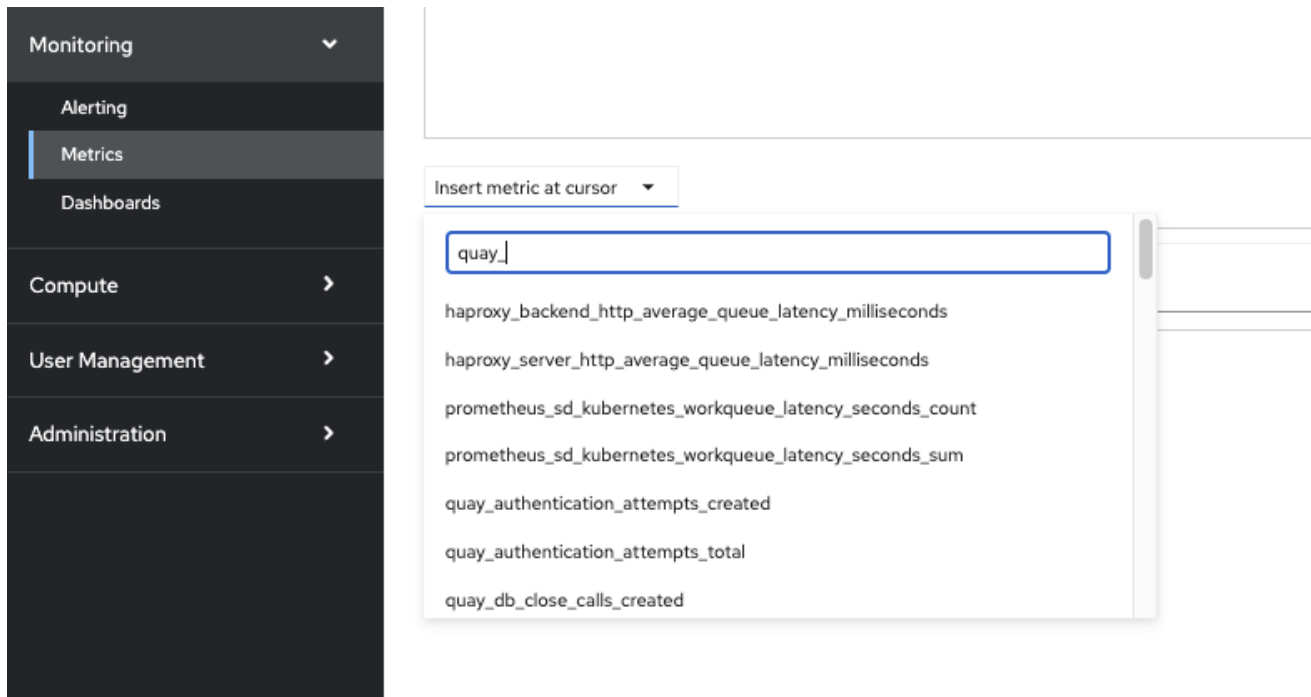
```
$ oc apply -f quay-service-monitor.yaml
```

#### Example output

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

### 5.6.6. 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:



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



## 5.7. RESIZING MANAGED STORAGE

The Red Hat Quay Operator creates default object storage using the defaults provided by Red Hat OpenShift Data Foundation when creating a **NooBaa** object (50 Gib).

There are two ways to extend **NooBaa** object storage:

1. You can resize an existing persistent volume claim (PVC).
2. You can add more PVCs to a new storage pool.

### 5.7.1. Resizing the NooBaa PVC

Use the following procedure to resize the NooBaa PVC.

#### Procedure

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

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

## 5.8. CUSTOMIZING DEFAULT OPERATOR IMAGES

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.

### 5.8.1. Environment Variables

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

Environment Variable	Component
<b>RELATED_IMAGE_COMPONENT_QUAY</b>	<b>base</b>
<b>RELATED_IMAGE_COMPONENT_CLAIR</b>	<b>clair</b>
<b>RELATED_IMAGE_COMPONENT_POSTGRES</b>	<b>postgres</b> and <b>clair</b> databases
<b>RELATED_IMAGE_COMPONENT_REDIS</b>	<b>redis</b>



### NOTE

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

### 5.8.2. 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@sha256:06dbb609484330ec6be6090109f1fa16e936afcf975d1cbc5ff3e6c7cae7542
```



### NOTE

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

## 5.9. AWS S3 CLOUDFRONT

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 6. 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.

### 6.1. 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.

### 6.2. CREATING A RED HAT QUAY BUILDERS ENVIRONMENT WITH OPENSIFT CONTAINER PLATFORM

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

#### 6.2.1. OpenShift Container Platform TLS component

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



#### NOTE

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

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

To add the builder route, use the following format:

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

#### 6.2.2. 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.

##### 6.2.2.1. 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:

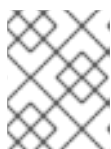
```
$ oc create sa -n virtual-builders quay-builder
```

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
```

**Example output**

```
eyJhbGciOiJSUzI1NiIsImtpZCI6IldfQUJkaDVmb3ltTHZ0dGZMYjhlWnYxZTQzN2dJVEJxc  
DJscldSdEUtYWsisifQ...
```

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/TIS certificate with the .crt extension by entering the following command:

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

#### 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> ❶
BUILD_MANAGER:
- ephemeral
- ALLOWED_WORKER_COUNT: 1
ORCHESTRATOR_PREFIX: buildman/production/
JOB_REGISTRATION_TIMEOUT: 3600 ❷
ORCHESTRATOR:
  REDIS_HOST: <sample_redis_hostname> ❸
  REDIS_PASSWORD: ""
  REDIS_SSL: false
  REDIS_SKIP_KEYSPACE_EVENT_SETUP: false
EXECUTORS:
- EXECUTOR: kubernetesPodman
  NAME: openshift
  BUILDER_NAMESPACE: <sample_builder_namespace> ❹
  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_cert_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:
"eyJhbGciOiJIUzI1NiIsImtpZCI6IldfQUJkaDVmb3ItTHZ0dGZMYjhlWnYxZTQzN2dJVEJxcDJs
cldSdEUtYW5ifQ"

```

#### 6.2.2.2. 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](#).

##### 6.2.2.2.1. 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
```

#### 6.2.2.2.2. Setting TLS to unmanaged

Use the following procedure to set **kind: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'
```

### 6.2.2.2.3. 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
```

### 6.2.2.2.4. 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: >-
    LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSURNakNDQWhxZ0F3SUJBZ0I....
  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: >-
    LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUVaakNDQTA2Z0F3SUJBZ0IVT...
```

```
ssl.key: >-
LS0tLS1CRUdJTiBSU0EgUFJJVkFURSBLRVktLS0tLQpNSUIFcFFJQkFBS0NBUEVBC...
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: >-
    RkVBVFVSRV9VU0VSX0IOSVRJQUxJWkU6IHRydWUKQlJ...
  extra_ca_cert_build_cluster.crt: >-

LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSURNakNDQWhxZ0F3SUJBZ0ldw....
ssl.cert: >-
LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUVaakNDQTA2Z0F3SUJBZ0lVT...
ssl.key: >-
LS0tLS1CRUdJTiBSU0EgUFJJVkFURSBLRVktLS0tLQpNSUIFcFFJQkFBS0NBUEVBC...
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-q7tvI	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-2mwig2	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-jqzwwg	1/1	Terminating	0	5d21h
example-registry-quay-redis-7cc5f6c977-956g8	1/1	Running	0	5d21h

- 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

- 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
```

### 6.2.2.3. Using the UI to create a build trigger

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

#### Procedure

- Log in to your Red Hat Quay repository.
- Click **Create New Repository** and create a new registry, for example, **testrepo**.
- 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.

- 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.

### 6.2.2.4. 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](https://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
  }
]
```

#### 6.2.2.5. Modifying your Google Cloud Platform object bucket

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 7. GEO-REPLICATION

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

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

### 7.1. 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.

### 7.2. 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. 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 site's 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 be 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 endpoint, 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.

### 7.2.1. 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.

### 7.2.1.1. Configuring geo-replication for the Red Hat Quay Operator on OpenShift Container Platform

Use the following procedure to configure geo-replication for the Red Hat Quay Operator.

#### 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> ❶
DB_CONNECTION_ARGS:
  autorollback: true
  threadlocals: true
DB_URI: postgresql://postgres:password@10.19.0.1:5432/quay ❷
BUILDLOGS_REDIS:
  host: 10.19.0.2
  port: 6379
USER_EVENTS_REDIS:
  host: 10.19.0.2
  port: 6379
DISTRIBUTED_STORAGE_CONFIG:
  usstorage:
    - GoogleCloudStorage
    - access_key: GOOGQGPGVMASAAMQABCDEFGG
      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 with either with the config tool or directly in the config bundle. For more information, see [Configuring TLS and routes](#).

## European cluster

```

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

```



## NOTE

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

### 7.2.2. 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:

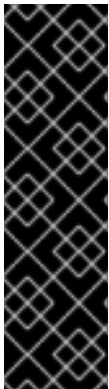
- A VPN to prevent visibility of the internal storage, or
- A token pair that only allows access to the specified bucket used by Red Hat Quay

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.

## 7.3. UPGRADING A GEO-REPLICATION DEPLOYMENT OF THE RED HAT QUAY OPERATOR

Use the following procedure to upgrade your geo-replicated Red Hat Quay Operator.



## IMPORTANT

- When upgrading geo-replicated Red Hat Quay Operator deployments 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 Operator deployment before upgrading.



## PROCEDURE

This procedure assumes that you are running the Red Hat Quay Operator 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 Operator deployment. 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 ❶
    - kind: quay
      managed: true
      overrides: ❷
        replicas: 0
    - kind: clair
      managed: true
      overrides:
        replicas: 0
    - kind: mirror
      managed: true
      overrides:
        replicas: 0
    ...

```

- ❶ Disable auto scaling of Quay, Clair and Mirroring workers
- ❷ Set the replica count to 0 for components accessing the database and objectstorage



#### NOTE

You must keep the Red Hat Quay Operator 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-config-editor-6dfdcfc44f-hlvwm 1/1   Running   0           73s
quayregistry-quay-redis-84f888776f-hhgms    1/1   Running   0           12m

```

3. On System A, initiate a Red Hat Quay Operator 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 Operator 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 Operator on System B and on System C until the UI is available on System A.

6. After confirming that the update has properly worked on System A, initiate the Red Hat Quay Operator 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.

## 7.3.1. Removing a geo-replicated site from your Red Hat Quay Operator 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. Complete this step 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>
```

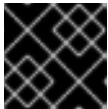
### Example output

```
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.remove_location 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 8. 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

### 8.1. BACKING UP RED HAT QUAY

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

#### Prerequisites

- A healthy Red Hat Quay deployment on OpenShift Container Platform using the Red Hat Quay Operator. 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 Operator 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.

#### 8.1.1. 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:

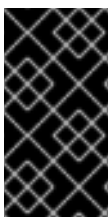
```
$ oc get secret -n <quay-namespace> $(oc get quayregistry <quay_registry_name> -n
<quay_namespace> -o jsonpath='{.spec.configBundleSecret}') -o yaml > config-bundle.yaml
```

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
```

## 8.1.2. 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 ❶
    - kind: quay
      managed: true
      overrides: ❷
        replicas: 0
    - kind: clair
      managed: true
      overrides:
        replicas: 0
    - kind: mirror
      managed: true
      overrides:
        replicas: 0
    ...
```

- ❶ Disable auto scaling of Quay, Clair and Mirroring workers
- ❷ 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 Operator first and then the managed Red Hat Quay resources:

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

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

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

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

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

## 8.1.3. 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
```

### 8.1.3.1. 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 (unmanaged) 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:

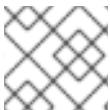
```
$ export AWS_ACCESS_KEY_ID=$(oc get secret -l app=noobaa -n <quay-namespace> -o jsonpath='{.items[0].data.AWS_ACCESS_KEY_ID}' |base64 -d)
```

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
```

## 8.1.4. 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 Operator 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 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
```

## 8.2. 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 Operator v3.7 or later)
- There is no running Red Hat Quay deployment managed by the Red Hat Quay Operator in the target namespace on your OpenShift Container Platform cluster

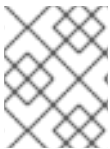


#### NOTE

If your deployment contains partially unmanaged database or storage components and you are using external services for 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

### 8.2.1. 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 "/>**

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>
```

## 8.2.2. 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 ❶
    - kind: quay
      managed: true
      overrides: ❷
        replicas: 0
    - kind: clair
      managed: true
      overrides:
        replicas: 0
    - kind: mirror
      managed: true
      overrides:
        replicas: 0
    ...
```

❶

Disable auto scaling of Quay, Clair and Mirroring workers

❷

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 Operator first and then the managed Red Hat Quay resources:

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

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

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

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

- 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
```

### 8.2.3. Restoring your Red Hat Quay database

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

#### Procedure

- 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
```

- 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
```

- Open a remote terminal to the database by entering the following command:

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

- Enter `psql` by running the following command:

```
bash-4.4$ psql
```

- You can list the database by running the following command:

```
postgres=# \l
```

### Example output

```

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

- Drop the database by entering the following command:

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

### Example output

```
DROP DATABASE
```

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

```
\q
```

- Redirect your PostgreSQL database to your backup database:

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

- Exit bash by entering the following command:

```
sh-4.4$ exit
```

## 8.2.4. Restore your Red Hat Quay object storage data

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

### Procedure

- 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)
```

- 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)
```

- 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.

### 8.2.5. 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 Operator 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 9. ENABLING OCI SUPPORT WITH THE RED HAT QUAY OPERATOR

Use the following procedure to configure Open Container Initiative (OCI) support for Red Hat Quay.

### Procedure

1. Create a **quay-config-bundle** YAML file that includes the following information:

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

2. Enter the following command to create a the **quay-config-bundle** object in the appropriate namespace, passing in the necessary properties to enable OCI support. For example:

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

3. In your **quay-registry.yaml** file, reference the secret for the **spec.configBundleSecret** field. For example:

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

### Additional resources

For more information, see [OCI Support and Red Hat Quay](#)



## CHAPTER 10. 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. This is a known issue and will be fixed in a future version of Red Hat Quay. ([PROJQUAY-4301](#))

## CHAPTER 11. 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 as a Red Hat Quay registry with Clair scanning.
- Exposes vulnerabilities through the **ImageManifestVuln** object in the Kubernetes API.

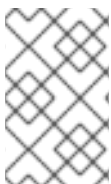


### NOTE

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

### 11.1. DOWNLOADING AND RUNNING THE CONTAINER SECURITY OPERATOR IN OPENSIFT CONTAINER PLATFORM

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



### NOTE

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

#### Procedure

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

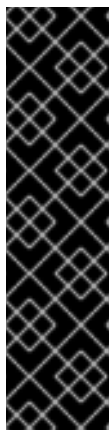
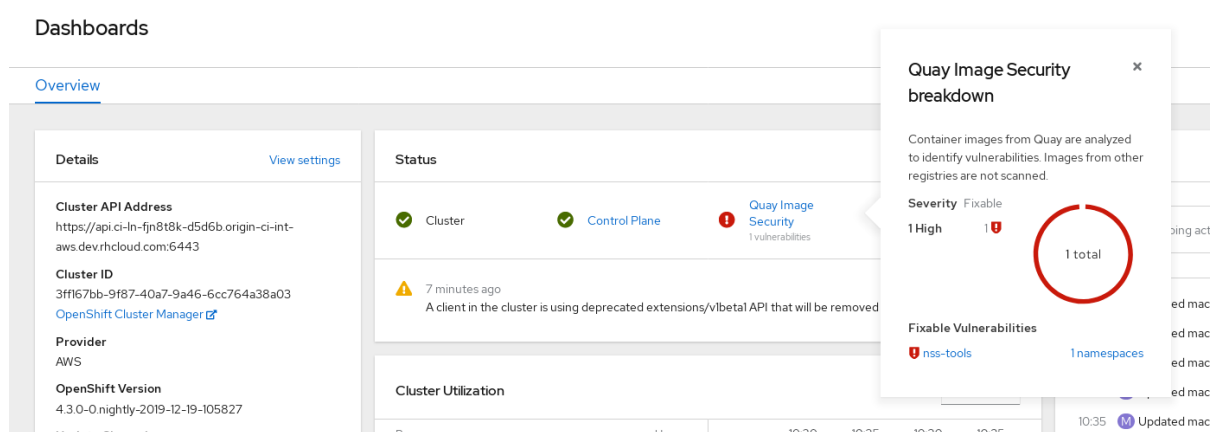
```
$ oc create secret generic container-security-operator-extra-certs --from-file=quay.crt -n openshift-operators
```



## NOTE

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

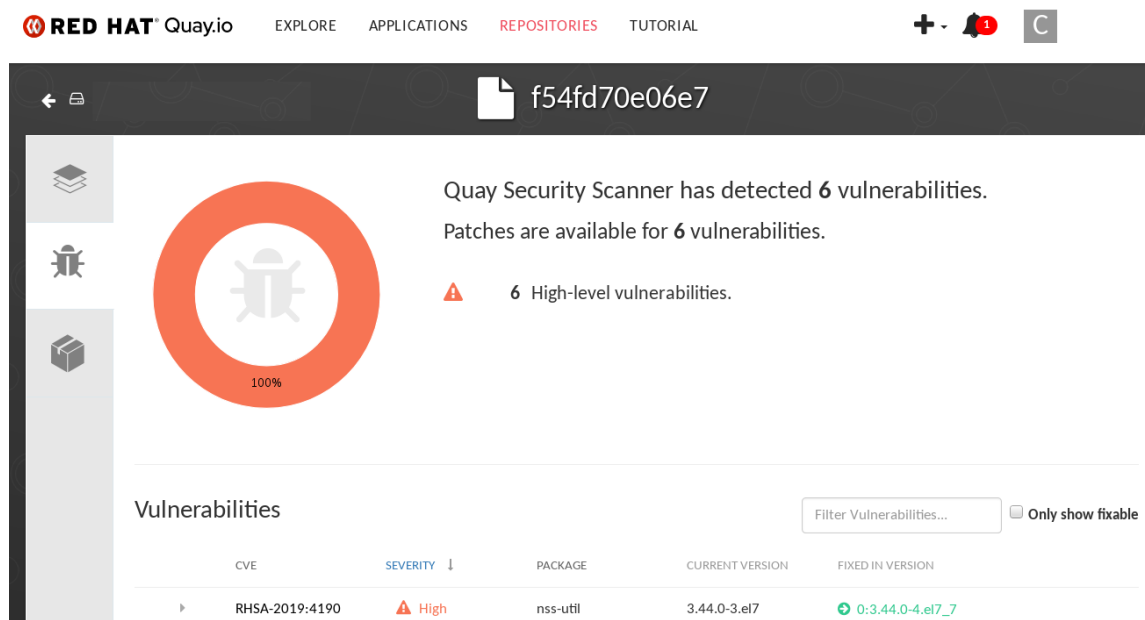
5. Navigate to **Home → Dashboards**. A link to **Image Security** appears under the status section, with a listing of the number of vulnerabilities found so far. Select the link to see a security breakdown, as shown in the following image:



## IMPORTANT

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

6. You can do one of two things at this point to follow up on any detected vulnerabilities:
  - a. Select the link to the vulnerability. You are taken to the container registry, Red Hat Quay or other registry where the container came from, where you can see information about the vulnerability. The following figure shows an example of detected vulnerabilities from a Quay.io registry:



RED HAT Quay.io EXPLORE APPLICATIONS REPOSITORIES TUTORIAL

f54fd70e06e7

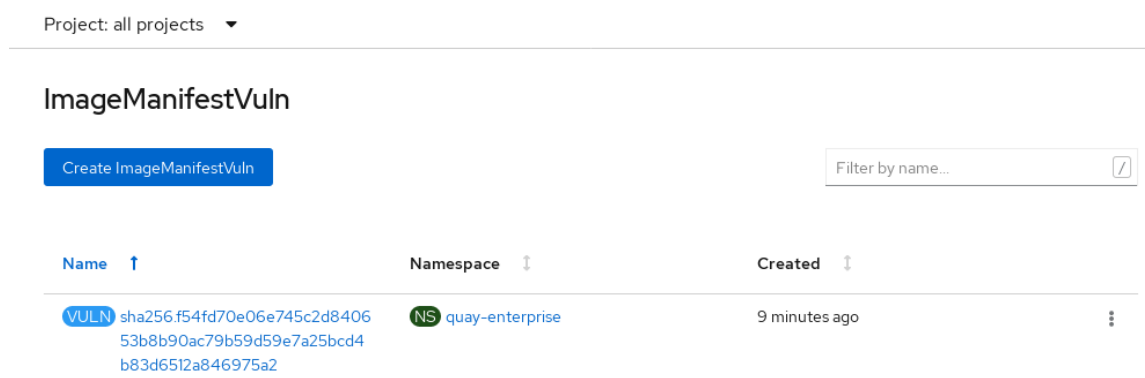
Quay Security Scanner has detected **6** vulnerabilities.  
Patches are available for **6** vulnerabilities.

6 High-level vulnerabilities.

Vulnerabilities

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:



Project: all projects

ImageManifestVuln

Create ImageManifestVuln

Filter by name...

Name	Namespace	Created
VULN sha256:f54fd70e06e745c2d840653b8b90ac79b59d59e7a25bcd4b83d6512a846975a2	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.

## 11.2. QUERY IMAGE VULNERABILITIES FROM THE CLI

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

### Procedure

- 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
```

- 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 12. DEPLOYING IPV6 ON THE RED HAT QUAY OPERATOR

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

For a list of known limitations, see [IPv6 limitations](#)

### 12.1. ENABLING THE IPV6 PROTOCOL FAMILY

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

#### Prerequisites

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

#### Procedure

1. In your deployment's **config.yaml** file, add the **FEATURE\_LISTEN\_IP\_VERSION** parameter and set it to **IPv6**, for example:

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

2. Start, or restart, your Red Hat Quay deployment.
3. Check that your deployment is listening to IPv6 by entering the following command:

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

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

**WARNING**

If your environment is configured to IPv4, but the **FEATURE\_LISTEN\_IP\_VERSION** configuration field is set to **IPv6**, Red Hat Quay will fail to deploy.

## 12.2. IPV6 LIMITATIONS

- Currently, attempting to configure your Red Hat Quay deployment with the common 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

```
...c1psWGpqeGIPQmNEWkJPMjJ5d0pDemVnR2QNCnRsbW9JdEF4YnFSdVd3PT0KLS0tLS1FTkQgQ0VSVEIGSUNBVEUtLS0tLQo=
```

2. Enter the following **kubect**l 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 string under the entry. For example:

```
custom-cert.crt:
c1psWGpqeGIPQmNEWkJPMjJ5d0pDemVnR2QNCnRsbW9JdEF4YnFSdVd3PT0KLS0tLS1FTkQgQ0VSVEIGSUNBVEUtLS0tLQo=
```

4. Use the **kubect**l **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-config-editor-6dfdcfc44f-hlvwm 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 **Operator Hub** 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 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:

1. 3.3.z → 3.6.z
2. 3.4.z → 3.6.z
3. 3.4.z → 3.7.z
4. 3.5.z → 3.7.z
5. 3.7.z → 3.8.z
6. 3.6.z → 3.9.z
7. 3.7.z → 3.9.z
8. 3.8.z → 3.9.z

For users on standalone deployments of Red Hat Quay wanting to upgrade to 3.9, see the [Standalone upgrade](#) guide.

### 14.2.1. Upgrading Quay

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

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

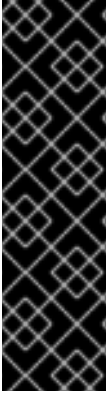
### 14.2.2. Updating Red Hat Quay from 3.8 → 3.9



#### IMPORTANT

If your Red Hat Quay deployment is upgrading from one y-stream to the next, for example, from 3.8.10 → 3.8.11, you must not switch the upgrade channel from **stable-3.8** to **stable-3.9**. Changing the upgrade channel in the middle of a y-stream upgrade will disallow Red Hat Quay from upgrading to 3.9. This is a known issue and will be fixed in a future version of Red Hat Quay.

When updating Red Hat Quay 3.8 → 3.9, the Operator automatically upgrades the existing PostgreSQL databases for Clair and Red Hat Quay from version 10 to version 13.



## IMPORTANT

- This upgrade is irreversible. It is highly recommended that you upgrade to PostgreSQL 13. PostgreSQL 10 had its final release on November 10, 2022 and is no longer supported. For more information, see the [PostgreSQL Versioning Policy](#).
- By default, Red Hat Quay is configured to remove old persistent volume claims (PVCs) from PostgreSQL 10. To disable this setting and backup old PVCs, you must set **POSTGRES\_UPGRADE\_RETAIN\_BACKUP** to **True** in your **quay-operator Subscription** object.

## Prerequisites

- You have installed Red Hat Quay 3.8 on OpenShift Container Platform.
- 100 GB of free, additional storage.  
During the upgrade process, additional persistent volume claims (PVCs) are provisioned to store the migrated data. This helps prevent a destructive operation on user data. The upgrade process rolls out PVCs for 50 GB for both the Red Hat Quay database upgrade, and the Clair database upgrade.

## Procedure

1. Optional. Back up your old PVCs from PostgreSQL 10 by setting **POSTGRES\_UPGRADE\_RETAIN\_BACKUP** to **True** your **quay-operator Subscription** object. For example:

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: quay-operator
  namespace: quay-enterprise
spec:
  channel: stable-3.8
  name: quay-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
config:
  env:
    - name: POSTGRES_UPGRADE_RETAIN_BACKUP
      value: "true"
```

2. In the OpenShift Container Platform Web Console, navigate to **Operators → Installed Operators**.
3. Click on the Red Hat Quay Operator.
4. Navigate to the **Subscription** tab.
5. Under **Subscription details** click **Update channel**.
6. Select **stable-3.9** and save the changes.
7. Check the progress of the new installation under **Upgrade status**. Wait until the upgrade status changes to **1 installed** before proceeding.

8. In your OpenShift Container Platform cluster, navigate to **Workloads → Pods**. Existing pods should be terminated, or in the process of being terminated.
9. Wait for the following pods, which are responsible for upgrading the database and alembic migration of existing data, to spin up: **clair-postgres-upgrade**, **quay-postgres-upgrade**, and **quay-app-upgrade**.
10. After the **clair-postgres-upgrade**, **quay-postgres-upgrade**, and **quay-app-upgrade** pods are marked as **Completed**, the remaining pods for your Red Hat Quay deployment spin up. This takes approximately ten minutes.
11. Verify that the **quay-database** and **clair-postgres** pods now use the **postgresql-13** image.
12. After the **quay-app** pod is marked as **Running**, you can reach your Red Hat Quay registry.

### 14.2.3. Upgrading directly from 3.3.z or 3.4.z to 3.6

The following section provides important information when upgrading from Red Hat Quay 3.3.z or 3.4.z to 3.6.

#### 14.2.3.1. Upgrading with edge routing enabled

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

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

#### 14.2.3.2. Upgrading with custom SSL/TLS certificate/key pairs without Subject Alternative Names

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

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

```
GODEBUG=x509ignoreCN=0
```

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

#### 14.2.3.3. Configuring Clair v4 when upgrading from 3.3.z or 3.4.z to 3.6 using the Red Hat Quay Operator

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

For instructions about setting up Clair v4 in a disconnected OpenShift Container Platform cluster, see [Setting Up Clair on a Red Hat Quay OpenShift deployment](#).

#### 14.2.4. Swift configuration when upgrading from 3.3.z to 3.6

When upgrading from Red Hat Quay 3.3.z to 3.6.z, some users might receive the following error: **Switch auth v3 requires tenant\_id (string) in os\_options**. As a workaround, you can manually update your **DISTRIBUTED\_STORAGE\_CONFIG** to add the **os\_options** and **tenant\_id** parameters:

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

#### 14.2.5. 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.

#### 14.2.6. 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**:

The screenshot shows the 'Subscription' tab for the 'quay-enterprise' project. The left sidebar contains navigation links: Administrator, Home, Operators (selected), OperatorHub, Installed Operators, Workloads, Networking, Storage, Builds, Monitoring, Compute, User Management, and Administration. The main content area displays the 'Subscription details' for the 'Red Hat Quay' operator (version 3.4.3 provided by Red Hat). The details include:

- Channel:** quay-v3.4
- Approval:** Automatic
- Upgrade status:** 1 installed (Up to date), 0 installing
- Name:** quay-operator
- Namespace:** quay-enterprise
- Labels:** operators.coreos.com/quay-operator.quay-enterprise
- Created at:** Mar 25, 12:17 pm
- Owner:** No owner
- Installed version:** quay-operatorv3.4.3
- Starting version:** quay-operatorv3.4.3
- CatalogSource:** redhat-operators (Healthy)
- InstallPlan:** install-wf26n

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

The screenshot shows the 'Installed Operators' page for the 'quay-enterprise' project. The left sidebar is the same as the previous screenshot. The main content area displays a table of installed operators:

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

## 14.3. UPGRADING A QUAYREGISTRY

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

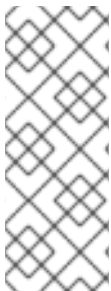
If something goes wrong during the automatic upgrade from **QuayEcosystem** to **QuayRegistry**, follow these steps to revert back to using the **QuayEcosystem**:

### 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.