



Red Hat Quay 3.7

Manage Red Hat Quay

Manage Red Hat Quay

Red Hat Quay 3.7 Manage Red Hat Quay

Manage Red Hat Quay

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[®] is the registered trademark of Linus Torvalds in the United States and other countries.

Java[®] is a registered trademark of Oracle and/or its affiliates.

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

MySQL[®] is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js[®] 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[®] 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

Manage Red Hat Quay

Table of Contents

PREFACE	6
CHAPTER 1. ADVANCED RED HAT QUAY CONFIGURATION	7
1.1. USING RED HAT QUAY CONFIG TOOL TO MODIFY RED HAT QUAY	7
1.1.1. Running the Config Tool from the Red Hat Quay Operator	7
1.1.2. Running the Config Tool from the command line	8
1.1.3. Deploying the config tool using TLS certificates	9
1.2. USING THE API TO MODIFY RED HAT QUAY	9
1.3. EDITING THE CONFIG.YAML FILE TO MODIFY RED HAT QUAY	10
1.3.1. Add name and company to Red Hat Quay sign-in	10
1.3.2. Disable TLS Protocols	10
1.3.3. Rate limit API calls	10
1.3.4. Adjust database connection pooling	10
1.3.4.1. Database connection arguments	11
1.3.4.2. Database SSL configuration	11
1.3.4.2.1. PostgreSQL SSL connection arguments	11
1.3.4.2.2. MySQL SSL connection arguments	12
1.3.4.3. HTTP connection counts	12
1.3.4.4. Dynamic process counts	12
1.3.4.5. Environment variables	13
1.3.4.6. Turning off connection pooling	14
CHAPTER 2. USING THE CONFIGURATION API	15
2.1. RETRIEVING THE DEFAULT CONFIGURATION	15
2.2. RETRIEVING THE CURRENT CONFIGURATION	15
2.3. VALIDATING CONFIGURATION USING THE API	16
2.4. DETERMINING THE REQUIRED FIELDS	17
CHAPTER 3. GETTING RED HAT QUAY RELEASE NOTIFICATIONS	18
CHAPTER 4. USING SSL TO PROTECT CONNECTIONS TO RED HAT QUAY	19
4.1. INTRODUCTION TO USING SSL	19
4.2. CREATE A CERTIFICATE AUTHORITY AND SIGN A CERTIFICATE	19
4.2.1. Create a Certificate Authority	19
4.2.2. Sign a certificate	19
4.3. CONFIGURING SSL USING THE COMMAND LINE	20
4.4. CONFIGURING SSL USING THE UI	21
4.5. TESTING SSL CONFIGURATION USING THE COMMAND LINE	21
4.6. TESTING SSL CONFIGURATION USING THE BROWSER	22
4.7. CONFIGURING PODMAN TO TRUST THE CERTIFICATE AUTHORITY	22
4.8. CONFIGURING THE SYSTEM TO TRUST THE CERTIFICATE AUTHORITY	23
CHAPTER 5. ADDING TLS CERTIFICATES TO THE RED HAT QUAY CONTAINER	25
5.1. ADD TLS CERTIFICATES TO RED HAT QUAY	25
5.2. ADD CERTS WHEN DEPLOYED ON KUBERNETES	25
CHAPTER 6. CONFIGURING ACTION LOG STORAGE FOR ELASTICSEARCH	27
CHAPTER 7. CLAIR SECURITY SCANNING	29
7.1. SETTING UP CLAIR ON A RED HAT QUAY OPENSIFT DEPLOYMENT	29
7.1.1. Deploying Via the Quay Operator	29
7.1.2. Manually Deploying Clair	29
7.2. SETTING UP CLAIR ON A NON-OPENSIFT RED HAT QUAY DEPLOYMENT	34

7.3. ADVANCED CLAIR CONFIGURATION	35
7.3.1. Unmanaged Clair configuration	35
7.3.1.1. Unmanaging a Clair database	35
7.3.1.2. Configuring a custom Clair database	36
7.3.2. Running a custom Clair configuration with a managed database	37
7.4. CLAIR CRDA CONFIGURATION	37
7.4.1. Enabling Clair CRDA	37
7.5. USING CLAIR	38
7.6. CVE RATINGS FROM THE NATIONAL VULNERABILITY DATABASE	38
7.7. CONFIGURING CLAIR FOR DISCONNECTED ENVIRONMENTS	39
7.7.1. Mapping repositories to Common Product Enumeration (CPE) information	40
7.8. CLAIR UPDATER URLS	40
7.9. ADDITIONAL INFORMATION	41
CHAPTER 8. SCANNING POD IMAGES WITH THE CONTAINER SECURITY OPERATOR	42
8.1. DOWNLOADING AND RUNNING THE CONTAINER SECURITY OPERATOR IN OPENSIFT CONTAINER PLATFORM	42
8.2. QUERY IMAGE VULNERABILITIES FROM THE CLI	44
CHAPTER 9. INTEGRATING RED HAT QUAY INTO OPENSIFT CONTAINER PLATFORM WITH THE QUAY BRIDGE OPERATOR	45
9.1. SETTING UP RED HAT QUAY FOR THE QUAY BRIDGE OPERATOR	45
9.2. INSTALLING THE QUAY BRIDGE OPERATOR ON OPENSIFT CONTAINER PLATFORM	46
9.3. CREATING AN OPENSIFT CONTAINER PLATFORM SECRET FOR THE OAUTH TOKEN	46
9.4. CREATING THE QUAYINTEGRATION CUSTOM RESOURCE	47
9.4.1. Optional: Creating the QuayIntegration custom resource using the CLI	47
9.4.2. Optional: Creating the QuayIntegration custom resource using the web console	48
9.5. QUAYINTEGRATION CONFIGURATION FIELDS	49
CHAPTER 10. REPOSITORY MIRRORING	50
10.1. REPOSITORY MIRRORING	50
10.2. REPOSITORY MIRRORING VERSUS GEO-REPLICATION	50
10.3. USING REPOSITORY MIRRORING	51
10.4. MIRRORING CONFIGURATION UI	52
10.5. MIRRORING CONFIGURATION FIELDS	52
10.6. MIRRORING WORKER	53
10.7. CREATING A MIRRORED REPOSITORY	53
10.7.1. Repository mirroring settings	54
10.7.2. Advanced settings	55
10.7.3. Synchronize now	55
10.8. EVENT NOTIFICATIONS FOR MIRRORING	56
10.9. MIRRORING TAG PATTERNS	57
10.9.1. Pattern syntax	57
10.9.2. Example tag patterns	57
10.10. WORKING WITH MIRRORED REPOSITORIES	57
10.11. REPOSITORY MIRRORING RECOMMENDATIONS	59
CHAPTER 11. LDAP AUTHENTICATION SETUP FOR RED HAT QUAY	61
11.1. CONSIDERATIONS PRIOR TO ENABLING LDAP	61
11.1.1. Existing Quay deployments	61
11.1.2. Manual User Creation and LDAP authentication	61
11.2. SET UP LDAP CONFIGURATION	61
11.2.1. Full LDAP URI	61
11.2.2. Team Synchronization	62

11.2.3. Base and Relative Distinguished Names	62
11.2.4. Additional User Filters	63
11.2.5. Administrator DN	64
11.2.6. UID and Mail attributes	64
11.2.7. Validation	65
11.3. COMMON ISSUES	65
11.4. CONFIGURE AN LDAP USER AS SUPERUSER	65
CHAPTER 12. PROMETHEUS AND GRAFANA METRICS UNDER RED HAT QUAY	67
12.1. EXPOSING THE PROMETHEUS ENDPOINT	67
12.1.1. Standalone Red Hat Quay	67
12.1.2. Red Hat Quay Operator	67
12.1.3. Setting up Prometheus to consume metrics	68
12.1.4. DNS configuration under Kubernetes	68
12.1.5. DNS configuration for a manual cluster	68
12.2. INTRODUCTION TO METRICS	68
12.2.1. General registry statistics	68
12.2.2. Queue items	69
12.2.3. Garbage collection metrics	70
12.2.3.1. Multipart uploads metrics	71
12.2.4. Image push / pull metrics	72
12.2.4.1. Image pulls total	72
12.2.4.2. Image bytes pulled	73
12.2.4.3. Image pushes total	73
12.2.4.4. Image bytes pushed	73
12.2.5. Authentication metrics	73
CHAPTER 13. RED HAT QUAY QUOTA MANAGEMENT AND ENFORCEMENT	75
13.1. QUOTA MANAGEMENT CONFIGURATION	75
13.1.1. Default quota	75
13.2. QUOTA MANAGEMENT ARCHITECTURE	76
13.3. ESTABLISHING QUOTA IN RED HAT QUAY UI	76
13.4. ESTABLISHING QUOTA WITH THE RED HAT QUAY API	82
13.4.1. Setting the quota	83
13.4.2. Viewing the quota	83
13.4.3. Modifying the quota	83
13.4.4. Pushing images	84
13.4.4.1. Pushing ubuntu:18.04	84
13.4.4.2. Using the API to view quota usage	84
13.4.4.3. Pushing another image	85
13.4.5. Rejecting pushes using quota limits	86
13.4.5.1. Setting reject and warning limits	86
13.4.5.2. Viewing reject and warning limits	87
13.4.5.3. Pushing an image when the reject limit is exceeded	87
13.4.5.4. Notifications for limits exceeded	88
13.5. QUOTA MANAGEMENT LIMITATIONS	89
CHAPTER 14. GEO-REPLICATION	90
14.1. GEO-REPLICATION FEATURES	90
14.2. GEO-REPLICATION REQUIREMENTS AND CONSTRAINTS	90
14.3. GEO-REPLICATION USING STANDALONE RED HAT QUAY	91
14.3.1. Enable storage replication - standalone Quay	92
14.3.2. Run Red Hat Quay with storage preferences	93
14.4. GEO-REPLICATION USING THE RED HAT QUAY OPERATOR	93

14.4.1. Setting up geo-replication on Openshift	94
14.4.1.1. Configuration	95
14.4.2. Mixed storage for geo-replication	98
CHAPTER 15. BACKING UP AND RESTORING RED HAT QUAY MANAGED BY THE RED HAT QUAY OPERATOR	99
15.1. BACKING UP RED HAT QUAY	99
15.1.1. Red Hat Quay configuration backup	99
15.1.2. Scale down your Red Hat Quay deployment	100
15.1.3. Red Hat Quay managed database backup	102
15.1.3.1. Red Hat Quay managed object storage backup	102
15.1.4. Scale the Red Hat Quay deployment back up	103
15.2. RESTORING RED HAT QUAY	104
15.2.1. Restoring Red Hat Quay and its configuration from a backup	104
15.2.2. Scale down your Red Hat Quay deployment	105
15.2.3. Restore your Red Hat Quay database	106
15.2.4. Restore your Red Hat Quay object storage data	107
15.2.5. Scale up your Red Hat Quay deployment	108
CHAPTER 16. MIGRATING A STANDALONE QUAY DEPLOYMENT TO A RED HAT QUAY OPERATOR MANAGED DEPLOYMENT	110
16.1. BACKING UP A STANDALONE DEPLOYMENT OF RED HAT QUAY	110
16.2. USING BACKED UP STANDALONE CONTENT TO MIGRATE TO OPENSIFT CONTAINER PLATFORM.	111
CHAPTER 17. BACKING UP AND RESTORING RED HAT QUAY ON A STANDALONE DEPLOYMENT	116
17.1. BACKING UP RED HAT QUAY ON STANDALONE DEPLOYMENTS	116
17.2. RESTORING RED HAT QUAY ON STANDALONE DEPLOYMENTS	118
CHAPTER 18. RED HAT QUAY GARBAGE COLLECTION	122
18.1. ABOUT RED HAT QUAY GARBAGE COLLECTION	122
18.2. RED HAT QUAY GARBAGE COLLECTION IN PRACTICE	122
18.2.1. Measuring storage reclamation	123
18.3. GARBAGE COLLECTION CONFIGURATION FIELDS	124
18.4. DISABLING GARBAGE COLLECTION	125
18.5. GARBAGE COLLECTION AND QUOTA MANAGEMENT	125
18.6. GARBAGE COLLECTION IN PRACTICE	125
18.7. RED HAT QUAY GARBAGE COLLECTION METRICS	126
CHAPTER 19. RED HAT QUAY TROUBLESHOOTING	128
CHAPTER 20. SCHEMA FOR RED HAT QUAY CONFIGURATION	129
ADDITIONAL RESOURCES	129

PREFACE

Once you have deployed a Red Hat Quay registry, there are many ways you can further configure and manage that deployment. Topics covered here include:

- Advanced Red Hat Quay configuration
- Setting notifications to alert you of a new Red Hat Quay release
- Securing connections with SSL and TLS certificates
- Directing action logs storage to Elasticsearch
- Configuring image security scanning with Clair
- Scan pod images with the Container Security Operator
- Integrate Red Hat Quay into OpenShift with the Quay Bridge Operator
- Mirroring images with repository mirroring
- Sharing Quay images with a BitTorrent service
- Authenticating users with LDAP
- Enabling Quay for Prometheus and Grafana metrics
- Setting up geo-replication
- Troubleshooting Quay

CHAPTER 1. ADVANCED RED HAT QUAY CONFIGURATION

You can configure your Red Hat Quay after initial deployment using one of the following interfaces:

- The Red Hat Quay Config Tool. With this tool, a web-based interface for configuring the Red Hat Quay cluster is provided when running the **Quay** container in **config** mode. This method is recommended for configuring the Red Hat Quay service.
- Editing the **config.yaml**. The **config.yaml** file contains most configuration information for the Red Hat Quay cluster. Editing the **config.yaml** file directly is possible, but it is only recommended for advanced tuning and performance features that are not available through the Config Tool.
- Red Hat Quay API. Some Red Hat Quay features can be configured through the API.

This content in this section describes how to use each of the aforementioned interfaces and how to configure your deployment with advanced features.

1.1. USING RED HAT QUAY CONFIG TOOL TO MODIFY RED HAT QUAY

The Red Hat Quay Config Tool is made available by running a **Quay** container in **config** mode alongside the regular Red Hat Quay service.

Use the following sections to run the Config Tool from the Red Hat Quay Operator, or to run the Config Tool on host systems from the command line interface (CLI).

1.1.1. Running the Config Tool from the Red Hat Quay Operator

When running the Red Hat Quay Operator on OpenShift Container Platform, the Config Tool is readily available to use. Use the following procedure to access the Red Hat Quay Config Tool.

Prerequisites

1. You have deployed the Red Hat Quay Operator on OpenShift Container Platform.

Procedure.

1. On the OpenShift console, select the Red Hat Quay project, for example, **quay-enterprise**.
2. In the navigation pane, select **Networking** → **Routes**. You should see routes to both the Red Hat Quay application and Config Tool, as shown in the following image:

The screenshot shows the 'Routes' page in the Red Hat Quay console. The left sidebar is open to 'Networking' > 'Routes'. The main area displays a table of routes with columns for Name, Namespace, Status, Location, and Service. Two routes are shown, both with a status of 'Accepted'. The second route, 'example-quayecosystem-quay-config', is highlighted with a black arrow pointing to its 'Location' column, which contains a URL.

3. Select the route to the Config Tool, for example, **example-quayecosystem-quay-config**. The Config Tool UI should open in your browser.
4. Select **Modify configuration for this cluster** to bring up the Config Tool setup, for example:

The screenshot shows the 'Red Hat Quay Setup' page. The page title is 'Red Hat Quay Setup'. Below the title is a progress bar with steps 1-4. Step 1 is 'Almost done! Configure your Redis database and other settings below'. Step 2 is 'Custom SSL Certificates'. The 'Custom SSL Certificates' section contains text about certificates and a table of installed certificates. The table has columns for Certificate Filename, Status, and Names Handled. Two certificates are listed: 'quay.crt' and 'clair.crt', both with a status of 'Certificate is valid'. A 'Save Configuration Changes' button is at the bottom left.

5. Make the desired changes, and then select **Save Configuration Changes**.
6. Make any corrections needed by clicking **Continue Editing**, or, select **Next** to continue.
7. When prompted, select **Download Configuration**. This will download a tarball of your new **config.yaml**, as well as any certificates and keys used with your Red Hat Quay setup. The **config.yaml** can be used to make advanced changes to your configuration or use as a future reference.
8. Select **Go to deployment rollout** → **Populate the configuration to deployments**. Wait for the Red Hat Quay pods to restart for the changes to take effect.

1.1.2. Running the Config Tool from the command line

If you are running Red Hat Quay from a host system, you can use the following procedure to make changes to your configuration after the initial deployment.

1. Prerequisites
 - You have installed either **podman** or **docker**.
2. Start Red Hat Quay in configuration mode.
3. On the first **Quay** node, enter the following command:

```
$ podman run --rm -it --name quay_config -p 8080:8080 \
-v path/to/config-bundle:/conf/stack \
{productrepo}/{quayimage}:{productminv} config <my_secret_password>
```



NOTE

To modify an existing config bundle, you can mount your configuration directory into the **Quay** container.

4. When the Red Hat Quay configuration tool starts, open your browser and navigate to the URL and port used in your configuration file, for example, **quay-server.example.com:8080**.
5. Enter your username and password.
6. Modify your Red Hat Quay cluster as desired.

1.1.3. Deploying the config tool using TLS certificates

You can deploy the config tool with secured TLS certificates by passing environment variables to the runtime variable. This ensures that sensitive data like credentials for the database and storage backend are protected.

The public and private keys must contain valid Subject Alternative Names (SANs) for the route that you deploy the config tool on.

The paths can be specified using **CONFIG_TOOL_PRIVATE_KEY** and **CONFIG_TOOL_PUBLIC_KEY**.

If you are running your deployment from a container, the **CONFIG_TOOL_PRIVATE_KEY** and **CONFIG_TOOL_PUBLIC_KEY** values the locations of the certificates inside of the container. For example:

```
$ podman run --rm -it --name quay_config -p 7070:8080 \
-v ${PRIVATE_KEY_PATH}:/tls/localhost.key \
-v ${PUBLIC_KEY_PATH}:/tls/localhost.crt \
-e CONFIG_TOOL_PRIVATE_KEY=/tls/localhost.key \
-e CONFIG_TOOL_PUBLIC_KEY=/tls/localhost.crt \
-e DEBUGLOG=true \
-ti config-app:dev
```

1.2. USING THE API TO MODIFY RED HAT QUAY

See the [Red Hat Quay API Guide](#) for information on how to access Red Hat Quay API.

1.3. EDITING THE `CONFIG.YAML` FILE TO MODIFY RED HAT QUAY

Some advanced configuration features that are not available through the Config Tool can be implemented by editing the `config.yaml` file directly. Available settings are described in the [Schema for Red Hat Quay configuration](#)

The following examples are settings you can change directly in the `config.yaml` file.

1.3.1. Add name and company to Red Hat Quay sign-in

By setting the following field, users are prompted for their name and company when they first sign in. This is an optional field, but can provide your with extra data about your Red Hat Quay users.

```
---  
FEATURE_USER_METADATA: true  
---
```

1.3.2. Disable TLS Protocols

You can change the `SSL_PROTOCOLS` setting to remove SSL protocols that you do not want to support in your Red Hat Quay instance. For example, to remove TLS v1 support from the default `SSL_PROTOCOLS: ['TLSv1', 'TLSv1.1', 'TLSv1.2']`, change it to the following:

```
---  
SSL_PROTOCOLS : ['TLSv1.1', 'TLSv1.2']  
---
```

1.3.3. Rate limit API calls

Adding the `FEATURE_RATE_LIMITS` parameter to the `config.yaml` file causes `nginx` to limit certain API calls to 30-per-second. If `FEATURE_RATE_LIMITS` is not set, API calls are limited to 300-per-second, effectively making them unlimited.

Rate limiting is important when you must ensure that the available resources are not overwhelmed with traffic.

Some namespaces might require unlimited access, for example, if they are important to CI/CD and take priority. In that scenario, those namespaces might be placed in a list in the `config.yaml` file using the `NON_RATE_LIMITED_NAMESPACES`.

1.3.4. Adjust database connection pooling

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

With the `DB_CONNECTION_POOLING` parameter, each process that interacts with the database will contain a connection pool. These per-process connection pools are configured to maintain a maximum of 20 connections. When under heavy load, it is possible to fill the connection pool for every process within a Red Hat Quay container. Under certain deployments and loads, this might require analysis to ensure that Red Hat Quay does not exceed the database's configured maximum connection count.

Over time, the connection pools will release idle connections. To release all connections immediately, Red Hat Quay must be restarted.

Database connection pooling can be toggled by setting the **DB_CONNECTION_POOLING** to **true** or **false**. For example:

```
---
DB_CONNECTION_POOLING: true
---
```

When **DB_CONNECTION_POOLING** is enabled, you can change the maximum size of the connection pool with the **DB_CONNECTION_ARGS** in your **config.yaml**. For example:

```
---
DB_CONNECTION_ARGS:
  max_connections: 10
---
```

1.3.4.1. Database connection arguments

You can customize your Red Hat Quay database connection settings within the **config.yaml** file. These are dependent on your deployment's database driver, for example, **psycopg2** for Postgres and **pymysql** for MySQL. You can also pass in argument used by Peewee's connection pooling mechanism. For example:

```
---
DB_CONNECTION_ARGS:
  max_connections: n # Max Connection Pool size. (Connection Pooling only)
  timeout: n # Time to hold on to connections. (Connection Pooling only)
  stale_timeout: n # Number of seconds to block when the pool is full. (Connection Pooling only)
---
```

1.3.4.2. Database SSL configuration

Some key-value pairs defined under the **DB_CONNECTION_ARGS** field are generic, while others are specific to the database. In particular, SSL configuration depends on the database that you are deploying.

1.3.4.2.1. PostgreSQL SSL connection arguments

The following YAML shows a sample PostgreSQL SSL configuration:

```
---
DB_CONNECTION_ARGS:
  sslmode: verify-ca
  sslrootcert: /path/to/cacert
---
```

The **sslmode** parameter determines whether, or with, what priority a secure SSL TCP/IP connection will be negotiated with the server. There are six modes for the **sslmode** parameter:

- **disabl:** Only try a non-SSL connection.
- **allow:** Try a non-SSL connection first. Upon failure, try an SSL connection.
- **prefer:** Default. Try an SSL connection first. Upon failure, try a non-SSL connection.

- **require:** Only try an SSL connection. If a root CA file is present, verify the connection in the same way as if **verify-ca** was specified.
- **verify-ca:** Only try an SSL connection, and verify that the server certificate is issued by a trust certificate authority (CA).
- **verify-full:** Only try an SSL connection. Verify that the server certificate is issued by a trust CA, and that the requested server host name matches that in the certificate.

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

1.3.4.2.2. MySQL SSL connection arguments

The following YAML shows a sample MySQL SSL configuration:

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

For more information about the valid connection arguments for MySQL, see [Connecting to the Server Using URI-Like Strings or Key-Value Pairs](#).

1.3.4.3. HTTP connection counts

You can specify the quantity of simultaneous HTTP connections using environment variables. The environment variables can be specified as a whole, or for a specific component. The default for each is 50 parallel connections per process. See the following YAML for example environment variables;

```
---
WORKER_CONNECTION_COUNT_REGISTRY=n
WORKER_CONNECTION_COUNT_WEB=n
WORKER_CONNECTION_COUNT_SECSCAN=n
WORKER_CONNECTION_COUNT=n
---
```



NOTE

Specifying a count for a specific component will override any value set in the **WORKER_CONNECTION_COUNT** configuration field.

1.3.4.4. Dynamic process counts

To estimate the quantity of dynamically sized processes, the following calculation is used by default.



NOTE

Red Hat Quay queries the available CPU count from the entire machine. Any limits applied using kubernetes or other non-virtualized mechanisms will not affect this behavior. Red Hat Quay makes its calculation based on the total number of processors on the Node. The default values listed are simply targets, but shall not exceed the maximum or be lower than the minimum.

Each of the following process quantities can be overridden using the environment variable specified below:

- registry - Provides HTTP endpoints to handle registry action
 - minimum: 8
 - maximum: 64
 - default: $\$CPU_COUNT \times 4$
 - environment variable: `WORKER_COUNT_REGISTRY`
- web - Provides HTTP endpoints for the web-based interface
 - minimum: 2
 - maximum: 32
 - default: $\$CPU_COUNT \times 2$
 - environment_variable: `WORKER_COUNT_WEB`
- secscan - Interacts with Clair
 - minimum: 2
 - maximum: 4
 - default: $\$CPU_COUNT \times 2$
 - environment variable: `WORKER_COUNT_SECSCAN`

1.3.4.5. Environment variables

Red Hat Quay allows overriding default behavior using environment variables. The following table lists and describes each variable and the values they can expect.

Table 1.1. Worker count environment variables

Variable	Description	Values
<code>WORKER_COUNT_REGISTRY</code>	Specifies the number of processes to handle registry requests within the Quay container.	Integer between 8 and 64
<code>WORKER_COUNT_WEB</code>	Specifies the number of processes to handle UI/Web requests within the container.	Integer between 2 and 32
<code>WORKER_COUNT_SECSCAN</code>	Specifies the number of processes to handle Security Scanning (for example, Clair) integration within the container.	Integer between 2 and 4

Variable	Description	Values
DB_CONNECTION_POOLING	Toggle database connection pooling.	"true" or "false"

1.3.4.6. Turning off connection pooling

Red Hat Quay deployments with a large amount of user activity can regularly hit the 2k maximum database connection limit. In these cases, connection pooling, which is enabled by default for Red Hat Quay, can cause database connection count to rise exponentially and require you to turn off connection pooling.

If turning off connection pooling is not enough to prevent hitting the 2k database connection limit, you need to take additional steps to deal with the problem. If this happens, you might need to increase the maximum database connections to better suit your workload.

CHAPTER 2. USING THE CONFIGURATION API

The configuration tool exposes 4 endpoints that can be used to build, validate, bundle and deploy a configuration. The config-tool API is documented at <https://github.com/quay/config-tool/blob/master/pkg/lib/editor/API.md>. In this section, you will see how to use the API to retrieve the current configuration and how to validate any changes you make.

2.1. RETRIEVING THE DEFAULT CONFIGURATION

If you are running the configuration tool for the first time, and do not have an existing configuration, you can retrieve the default configuration. Start the container in config mode:

```
$ sudo podman run --rm -it --name quay_config \
-p 8080:8080 \
registry.redhat.io/quay/quay-rhel8:v3.7.13 config secret
```

Use the **config** endpoint of the configuration API to get the default:

```
$ curl -X GET -u quayconfig:secret http://quay-server:8080/api/v1/config | jq
```

The value returned is the default configuration in JSON format:

```
{
  "config.yaml": {
    "AUTHENTICATION_TYPE": "Database",
    "AVATAR_KIND": "local",
    "DB_CONNECTION_ARGS": {
      "autorollback": true,
      "threadlocals": true
    },
    "DEFAULT_TAG_EXPIRATION": "2w",
    "EXTERNAL_TLS_TERMINATION": false,
    "FEATURE_ACTION_LOG_ROTATION": false,
    "FEATURE_ANONYMOUS_ACCESS": true,
    "FEATURE_APP_SPECIFIC_TOKENS": true,
    ....
  }
}
```

2.2. RETRIEVING THE CURRENT CONFIGURATION

If you have already configured and deployed the Quay registry, stop the container and restart it in configuration mode, loading the existing configuration as a volume:

```
$ sudo podman run --rm -it --name quay_config \
-p 8080:8080 \
-v $QUAY/config:/conf/stack:Z \
registry.redhat.io/quay/quay-rhel8:v3.7.13 config secret
```

Use the **config** endpoint of the API to get the current configuration:

```
$ curl -X GET -u quayconfig:secret http://quay-server:8080/api/v1/config | jq
```

The value returned is the current configuration in JSON format, including database and Redis configuration data:

```
{
  "config.yaml": {
    ...
    "BROWSER_API_CALLS_XHR_ONLY": false,
    "BUILDLOGS_REDIS": {
      "host": "quay-server",
      "password": "strongpassword",
      "port": 6379
    },
    "DATABASE_SECRET_KEY": "4b1c5663-88c6-47ac-b4a8-bb594660f08b",
    "DB_CONNECTION_ARGS": {
      "autorollback": true,
      "threadlocals": true
    },
    "DB_URI": "postgresql://quayuser:quaypass@quay-server:5432/quay",
    "DEFAULT_TAG_EXPIRATION": "2w",
    ...
  }
}
```

2.3. VALIDATING CONFIGURATION USING THE API

You can validate a configuration by posting it to the **config/validate** endpoint:

```
curl -u quayconfig:secret --header 'Content-Type: application/json' --request POST --data '
{
  "config.yaml": {
    ...
    "BROWSER_API_CALLS_XHR_ONLY": false,
    "BUILDLOGS_REDIS": {
      "host": "quay-server",
      "password": "strongpassword",
      "port": 6379
    },
    "DATABASE_SECRET_KEY": "4b1c5663-88c6-47ac-b4a8-bb594660f08b",
    "DB_CONNECTION_ARGS": {
      "autorollback": true,
      "threadlocals": true
    },
    "DB_URI": "postgresql://quayuser:quaypass@quay-server:5432/quay",
    "DEFAULT_TAG_EXPIRATION": "2w",
    ...
  }
}
' http://quay-server:8080/api/v1/config/validate | jq
```

The returned value is an array containing the errors found in the configuration. If the configuration is valid, an empty array `[]` is returned.

2.4. DETERMINING THE REQUIRED FIELDS

You can determine the required fields by posting an empty configuration structure to the **config/validate** endpoint:

```
curl -u quayconfig:secret --header 'Content-Type: application/json' --request POST --data '
{
  "config.yaml": {
  }
} http://quay-server:8080/api/v1/config/validate | jq
```

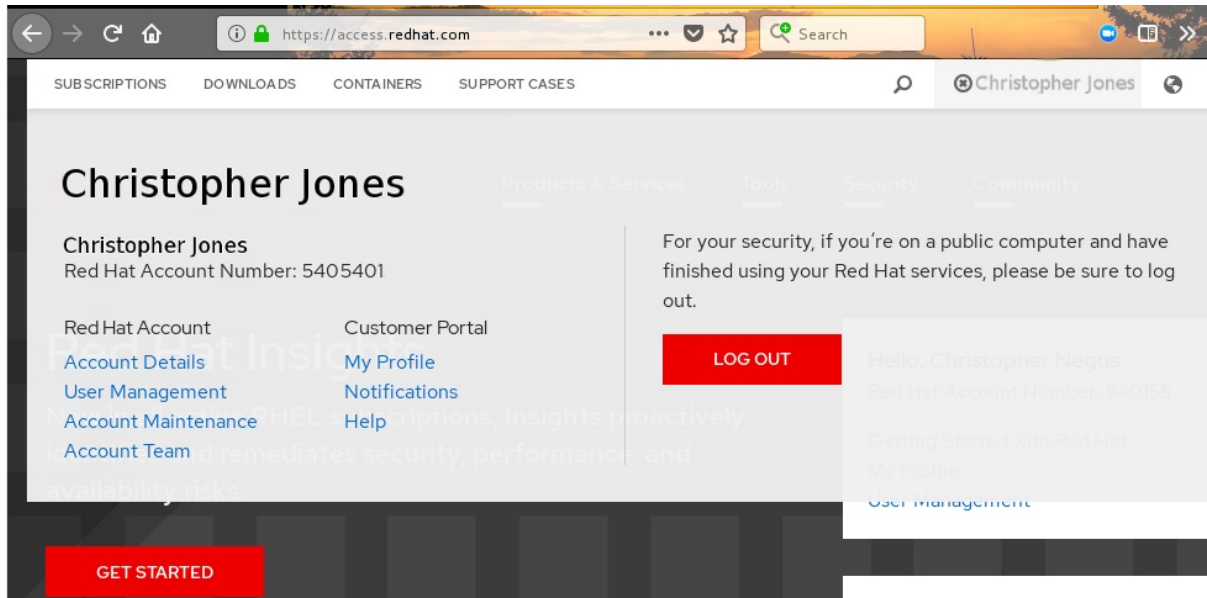
The value returned is an array indicating which fields are required:

```
[
  {
    "FieldGroup": "Database",
    "Tags": [
      "DB_URI"
    ],
    "Message": "DB_URI is required."
  },
  {
    "FieldGroup": "DistributedStorage",
    "Tags": [
      "DISTRIBUTED_STORAGE_CONFIG"
    ],
    "Message": "DISTRIBUTED_STORAGE_CONFIG must contain at least one storage location."
  },
  {
    "FieldGroup": "HostSettings",
    "Tags": [
      "SERVER_HOSTNAME"
    ],
    "Message": "SERVER_HOSTNAME is required"
  },
  {
    "FieldGroup": "HostSettings",
    "Tags": [
      "SERVER_HOSTNAME"
    ],
    "Message": "SERVER_HOSTNAME must be of type Hostname"
  },
  {
    "FieldGroup": "Redis",
    "Tags": [
      "BUILDLOGS_REDIS"
    ],
    "Message": "BUILDLOGS_REDIS is required"
  }
]
```

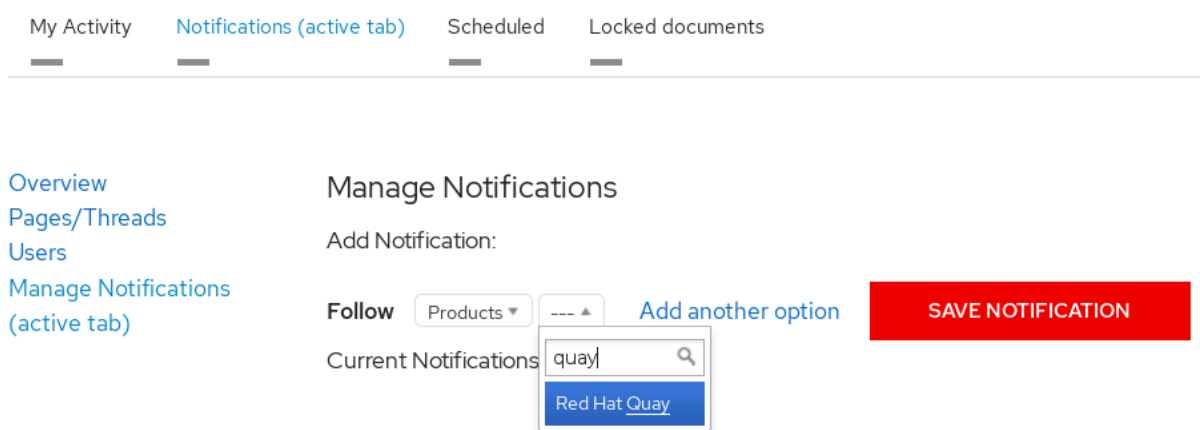
CHAPTER 3. GETTING RED HAT QUAY RELEASE NOTIFICATIONS

To keep up with the latest Red Hat Quay releases and other changes related to Red Hat Quay, you can sign up for update notifications on the [Red Hat Customer Portal](#). After signing up for notifications, you will receive notifications letting you know when there is new a Red Hat Quay version, updated documentation, or other Red Hat Quay news.

1. Log into the [Red Hat Customer Portal](#) with your Red Hat customer account credentials.
2. Select your user name (upper-right corner) to see Red Hat Account and Customer Portal selections:



3. Select Notifications. Your profile activity page appears.
4. Select the Notifications tab.
5. Select Manage Notifications.
6. Select Follow, then choose Products from the drop-down box.
7. From the drop-down box next to the Products, search for and select Red Hat Quay:



8. Select the SAVE NOTIFICATION button. Going forward, you will receive notifications when there are changes to the Red Hat Quay product, such as a new release.

CHAPTER 4. USING SSL TO PROTECT CONNECTIONS TO RED HAT QUAY

4.1. INTRODUCTION TO USING SSL

To configure Red Hat Quay with a [self-signed certificate](#), you need to create a Certificate Authority (CA) and then generate the required key and certificate files.

The following examples assume you have configured the server hostname **quay-server.example.com** using DNS or another naming mechanism, such as adding an entry in your **/etc/hosts** file:

```
$ cat /etc/hosts
...
192.168.1.112 quay-server.example.com
```

4.2. CREATE A CERTIFICATE AUTHORITY AND SIGN A CERTIFICATE

At the end of this procedure, you will have a certificate file and a primary key file named **ssl.cert** and **ssl.key**, respectively.

4.2.1. Create a Certificate Authority

1. Generate the root CA key:

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

2. Generate the root CA cert:

```
$ openssl req -x509 -new -nodes -key rootCA.key -sha256 -days 1024 -out rootCA.pem
```

3. Enter the information that will be incorporated into your certificate request, including the server hostname, for example:

```
Country Name (2 letter code) [XX]:IE
State or Province Name (full name) []:GALWAY
Locality Name (eg, city) [Default City]:GALWAY
Organization Name (eg, company) [Default Company Ltd]:QUAY
Organizational Unit Name (eg, section) []:DOCS
Common Name (eg, your name or your server's hostname) []:quay-server.example.com
```

4.2.2. Sign a certificate

1. Generate the server key:

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

2. Generate a signing request:

```
$ openssl req -new -key ssl.key -out ssl.csr
```

- Enter the information that will be incorporated into your certificate request, including the server hostname, for example:

```
Country Name (2 letter code) [XX]:IE
State or Province Name (full name) []:GALWAY
Locality Name (eg, city) [Default City]:GALWAY
Organization Name (eg, company) [Default Company Ltd]:QUAY
Organizational Unit Name (eg, section) []:DOCS
Common Name (eg, your name or your server's hostname) []:quay-server.example.com
```

- Create a configuration file **openssl.cnf**, specifying the server hostname, for example:

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 = quay-server.example.com
IP.1 = 192.168.1.112
```

- Use the configuration file to generate the certificate **ssl.cert**:

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

4.3. CONFIGURING SSL USING THE COMMAND LINE

Another option when configuring SSL is to use the command line interface.

- Copy the certificate file and primary key file to your configuration directory, ensuring they are named **ssl.cert** and **ssl.key** respectively:

```
$ cp ~/ssl.cert $QUAY/config
$ cp ~/ssl.key $QUAY/config
$ cd $QUAY/config
```

- Edit the **config.yaml** file and specify that you want Quay to handle TLS:

config.yaml

```
...
SERVER_HOSTNAME: quay-server.example.com
...
PREFERRED_URL_SCHEME: https
...
```

- Stop the **Quay** container and restart the registry:


```
$ sudo podman rm -f quay
$ sudo podman run -d --rm -p 80:8080 -p 443:8443 \
--name=quay \
-v $QUAY/config:/conf/stack:Z \
-v $QUAY/storage:/datastorage:Z \
registry.redhat.io/quay/quay-rhel8:v3.7.13
```

4.4. CONFIGURING SSL USING THE UI

This section configures SSL using the Quay UI. To configure SSL using the command line interface, see the following section.

1. Start the **Quay** container in configuration mode:

```
$ sudo podman run --rm -it --name quay_config -p 80:8080 -p 443:8443
registry.redhat.io/quay/quay-rhel8:v3.7.13 config secret
```

2. In the Server Configuration section, select **Red Hat Quay handles TLS** for TLS. Upload the certificate file and private key file created earlier, ensuring that the Server Hostname matches the value used when creating the certs. Validate and download the updated configuration.
3. Stop the **Quay** container and then restart the registry:

```
$ sudo podman rm -f quay
$ sudo podman run -d --rm -p 80:8080 -p 443:8443 \
--name=quay \
-v $QUAY/config:/conf/stack:Z \
-v $QUAY/storage:/datastorage:Z \
registry.redhat.io/quay/quay-rhel8:v3.7.13
```

4.5. TESTING SSL CONFIGURATION USING THE COMMAND LINE

- Use the **podman login** command to attempt to log in to the Quay registry with SSL enabled:

```
$ sudo podman login quay-server.example.com
Username: quayadmin
Password:

Error: error authenticating creds for "quay-server.example.com": error pinging docker registry
quay-server.example.com: Get "https://quay-server.example.com/v2/": x509: certificate
signed by unknown authority
```

- Podman does not trust self-signed certificates. As a workaround, use the **--tls-verify** option:

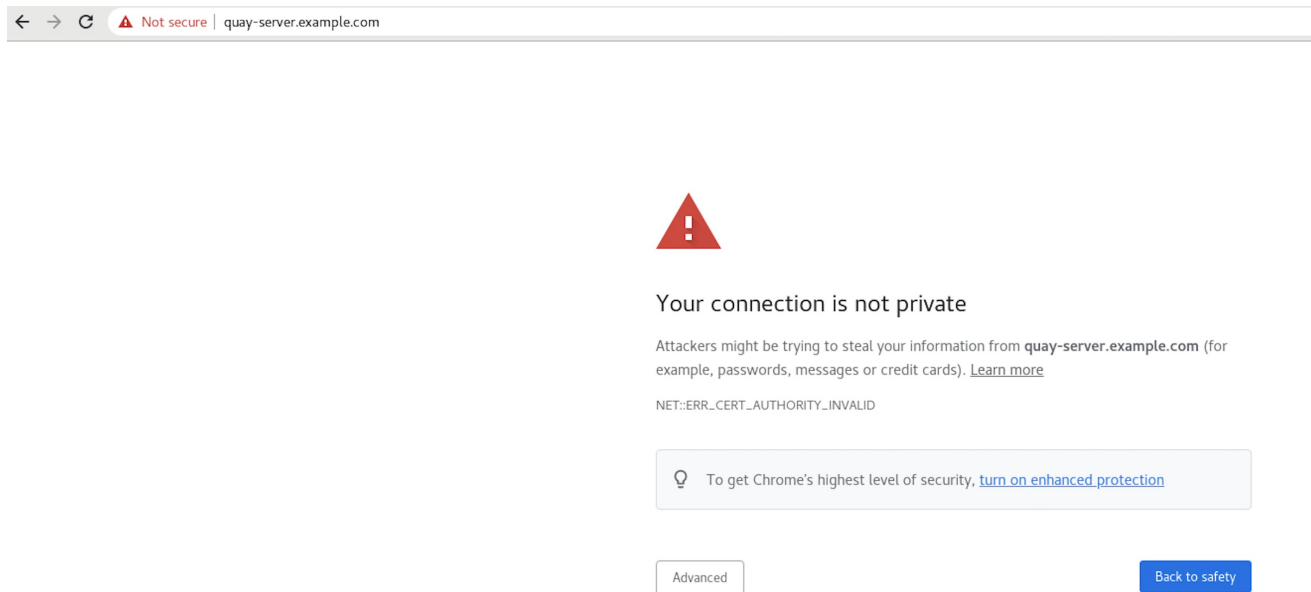
```
$ sudo podman login --tls-verify=false quay-server.example.com
Username: quayadmin
Password:

Login Succeeded!
```

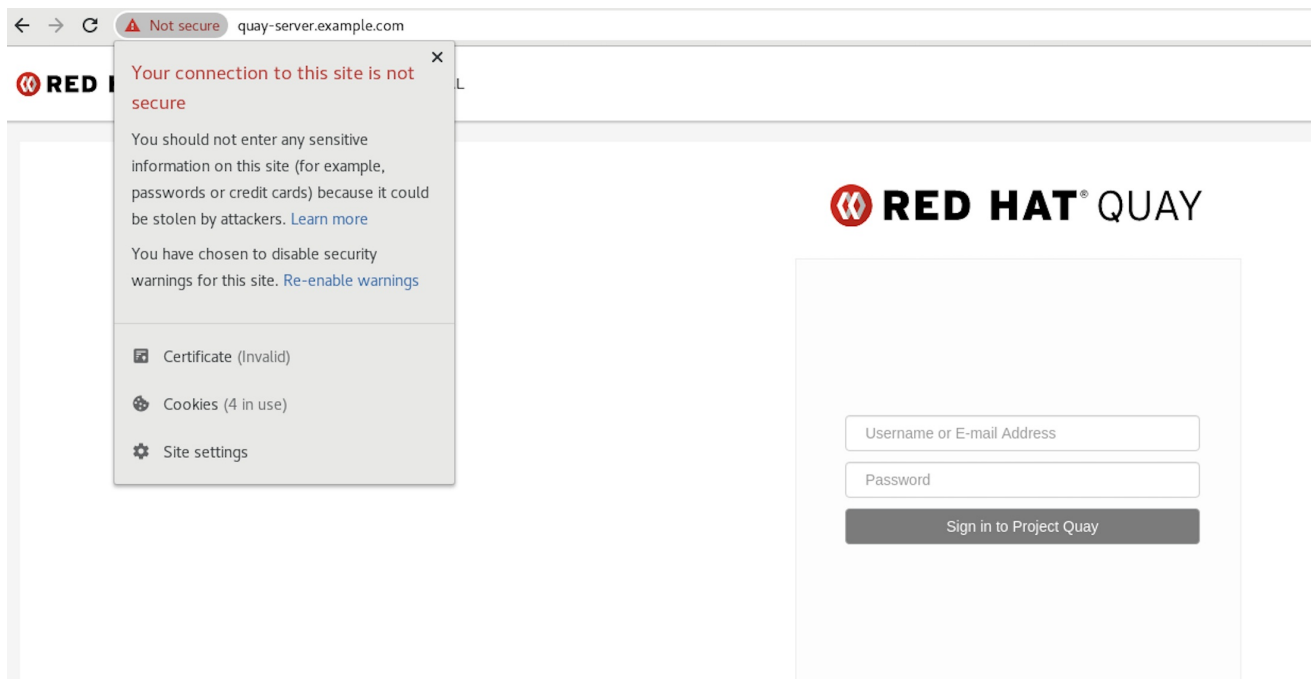
Configuring Podman to trust the root Certificate Authority (CA) is covered in a subsequent section.

4.6. TESTING SSL CONFIGURATION USING THE BROWSER

When you attempt to access the Quay registry, in this case, <https://quay-server.example.com>, the browser warns of the potential risk:



Proceed to the log in screen, and the browser will notify you that the connection is not secure:



Configuring the system to trust the root Certificate Authority (CA) is covered in the subsequent section.

4.7. CONFIGURING PODMAN TO TRUST THE CERTIFICATE AUTHORITY

Podman uses two paths to locate the CA file, namely, `/etc/containers/certs.d/` and `/etc/docker/certs.d/`.

- Copy the root CA file to one of these locations, with the exact path determined by the server hostname, and naming the file **ca.crt**:

```
$ sudo cp rootCA.pem /etc/containers/certs.d/quay-server.example.com/ca.crt
```

- Alternatively, if you are using Docker, you can copy the root CA file to the equivalent Docker directory:

```
$ sudo cp rootCA.pem /etc/docker/certs.d/quay-server.example.com/ca.crt
```

You should no longer need to use the **--tls-verify=false** option when logging in to the registry:

```
$ sudo podman login quay-server.example.com
```

```
Username: quayadmin
```

```
Password:
```

```
Login Succeeded!
```

4.8. CONFIGURING THE SYSTEM TO TRUST THE CERTIFICATE AUTHORITY

1. Copy the root CA file to the consolidated system-wide trust store:

```
$ sudo cp rootCA.pem /etc/pki/ca-trust/source/anchors/
```

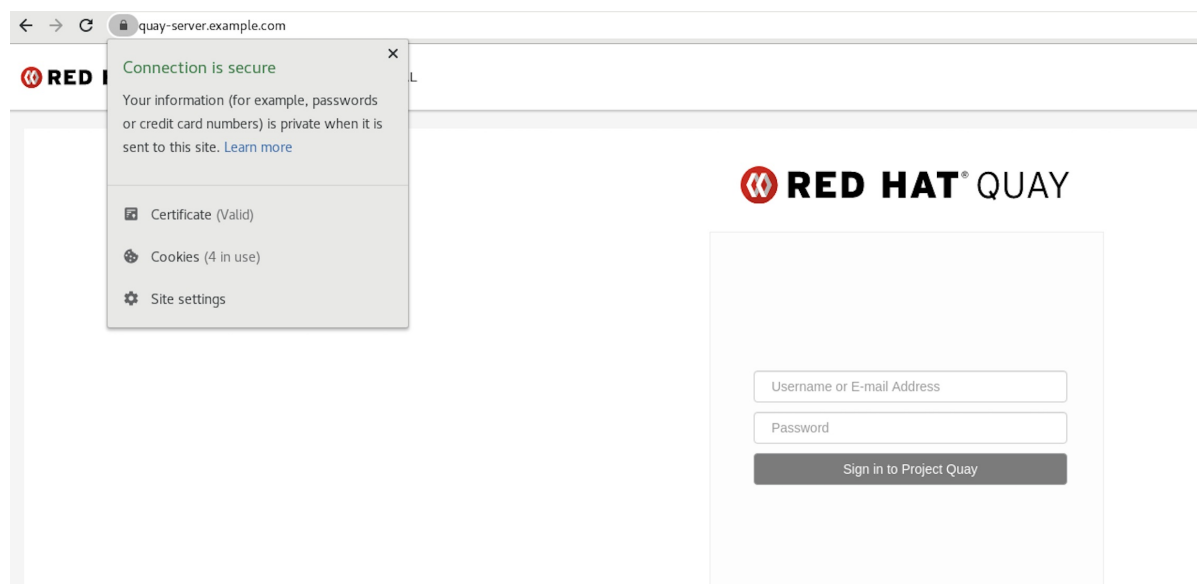
2. Update the system-wide trust store configuration:

```
$ sudo update-ca-trust extract
```

3. You can use the **trust list** command to ensure that the Quay server has been configured:

```
$ trust list | grep quay  
label: quay-server.example.com
```

Now, when you browse to the registry at <https://quay-server.example.com>, the lock icon shows that the connection is secure:



4. To remove the root CA from system-wide trust, delete the file and update the configuration:

```
$ sudo rm /etc/pki/ca-trust/source/anchors/rootCA.pem
$ sudo update-ca-trust extract
$ trust list | grep quay
$
```

More information can be found in the RHEL 8 documentation in the chapter [Using shared system certificates](#).

CHAPTER 5. ADDING TLS CERTIFICATES TO THE RED HAT QUAY CONTAINER

To add custom TLS certificates to Red Hat Quay, create a new directory named **extra_ca_certs/** beneath the Red Hat Quay config directory. Copy any required site-specific TLS certificates to this new directory.

5.1. ADD TLS CERTIFICATES TO RED HAT QUAY

1. View certificate to be added to the container

```
$ cat storage.crt
-----BEGIN CERTIFICATE-----
MIIDTTCCAjWgAwIbAgIJAMVr9ngjJhzbMA0GCSqGSIb3DQEBCwUAMD0xCzAJBgNV
[...]
-----END CERTIFICATE-----
```

2. Create certs directory and copy certificate there

```
$ mkdir -p quay/config/extra_ca_certs
$ cp storage.crt quay/config/extra_ca_certs/
$ tree quay/config/
|— config.yaml
|— extra_ca_certs
|  |— storage.crt
```

3. Obtain the **Quay** container's **CONTAINER ID** with **podman ps**:

```
$ sudo podman ps
CONTAINER ID      IMAGE                                COMMAND                                CREATED
STATUS           PORTS
5a3e82c4a75f     <registry>/<repo>/quay:v3.7.13 "/sbin/my_init"    24 hours ago    Up
18 hours        0.0.0.0:80->80/tcp, 0.0.0.0:443->443/tcp, 443/tcp  grave_keller
```

4. Restart the container with that ID:

```
$ sudo podman restart 5a3e82c4a75f
```

5. Examine the certificate copied into the container namespace:

```
$ sudo podman exec -it 5a3e82c4a75f cat /etc/ssl/certs/storage.pem
-----BEGIN CERTIFICATE-----
MIIDTTCCAjWgAwIbAgIJAMVr9ngjJhzbMA0GCSqGSIb3DQEBCwUAMD0xCzAJBgNV
```

5.2. ADD CERTS WHEN DEPLOYED ON KUBERNETES

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

To get around this error, a base64 encoded certificate can be added to the secret *after* Red Hat Quay has been deployed. Here's how:

1. Begin by base64 encoding the contents of the certificate:

```
$ cat ca.crt
-----BEGIN CERTIFICATE-----
MIIDljCCAn6gAwIBAgIBATANBgkqhkiG9w0BAQsFADA5MRcwFQYDVQQKDA5MQUJlu
TEICQ09SRS5TTzEeMBwGA1UEAwwVQ2VydGlmaWNhdGUgQXV0aG9yaXR5MB4XDTE2
MDExMjA2NTkxMfoXDTM2MDExMjA2NTkxMFowOTEXMBUGA1UECgwOTEFCLkxJQkNP
UkUuU08xHjAcBgNVBAMMFUNlcnRpZmljYXRlIEF1dGhvcml0eTCCASlwDQYJKoZI
[...]
-----END CERTIFICATE-----

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

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

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

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

```
custom-cert.crt:
c1psWGpqeGIPQmNEWkJPMjJ5d0pDemVnR2QNCnRsbW9JdEF4YnFSdVd3PT0KLS0tLS1F
TkQgQ0VSVEIGSUNBVEUtLS0tLQo=
```

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

CHAPTER 6. CONFIGURING ACTION LOG STORAGE FOR ELASTICSEARCH

By default, the past three months of usage logs are stored in the Red Hat Quay database and exposed via the web UI on organization and repository levels. Appropriate administrative privileges are required to see log entries. For deployments with a large amount of logged operations, you can now store the usage logs in Elasticsearch instead of the Red Hat Quay database backend. To do this, you need to provide your own Elasticsearch stack, as it is not included with Red Hat Quay as a customizable component.

Enabling Elasticsearch logging can be done during Red Hat Quay deployment or post-deployment using the Red Hat Quay Config Tool. The resulting configuration is stored in the **config.yaml** file. Once configured, usage log access continues to be provided the same way, via the web UI for repositories and organizations.

Here's how to configure action log storage to change it from the default Red Hat Quay database to use Elasticsearch:

1. Obtain an Elasticsearch account.
2. Open the Red Hat Quay Config Tool (either during or after Red Hat Quay deployment).
3. Scroll to the *Action Log Storage Configuration* setting and select *Elasticsearch* instead of *Database*. The following figure shows the Elasticsearch settings that appear:

Action Log Storage Configuration

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

Action Logs Storage:

Elasticsearch hostname:

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

Elasticsearch access key:

Elasticsearch secret key:

AWS region:

Index prefix:

Logs Producer:

4. Fill in the following information for your Elasticsearch instance:
 - **Elasticsearch hostname:** The hostname or IP address of the system providing the Elasticsearch service.
 - **Elasticsearch port:** The port number providing the Elasticsearch service on the host you just entered. Note that the port must be accessible from all systems running the Red Hat Quay registry. The default is TCP port 9200.

- **Elasticsearch access key.** The access key needed to gain access to the Elastic search service, if required.
- **Elasticsearch secret key.** The secret key needed to gain access to the Elastic search service, if required.
- **AWS region:** If you are running on AWS, set the AWS region (otherwise, leave it blank).
- **Index prefix** Choose a prefix to attach to log entries.
- **Logs Producer:** Choose either Elasticsearch (default) or Kinesis to direct logs to an intermediate Kinesis stream on AWS. You need to set up your own pipeline to send logs from Kinesis to Elasticsearch (for example, Logstash). The following figure shows additional fields you would need to fill in for Kinesis:

The screenshot shows a configuration form with the following fields:

- AWS region:** Text input field containing "The AWS region".
- Index prefix:** Text input field containing "logentry_".
- Logs Producer:** Dropdown menu with "Kinesis" selected.
- Stream name:** Text input field containing "The Kinesis stream name".
- AWS access key:** Text input field containing "The AWS access key".
- AWS secret key:** Text input field containing "The AWS secret key".
- AWS region:** Text input field containing "The AWS region".

At the bottom of the form, there is a yellow banner with a downward arrow icon and the text "9 configuration fields remaining".

5. If you chose Elasticsearch as the Logs Producer, no further configuration is needed. If you chose Kinesis, fill in the following:
 - **Stream name:** The name of the Kinesis stream.
 - **AWS access key.** The name of the AWS access key needed to gain access to the Kinesis stream, if required.
 - **AWS secret key.** The name of the AWS secret key needed to gain access to the Kinesis stream, if required.
 - **AWS region:** The AWS region.
6. When you are done, save the configuration. The Config Tool checks your settings. If there is a problem connecting to the Elasticsearch or Kinesis services, you will see an error and have the opportunity to continue editing. Otherwise, logging will begin to be directed to your Elasticsearch configuration after the cluster restarts with the new configuration.

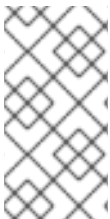
CHAPTER 7. CLAIR SECURITY SCANNING

Clair is a set of micro services that can be used with Red Hat Quay to perform vulnerability scanning of container images associated with a set of Linux operating systems. The micro services design of Clair makes it appropriate to run in a highly scalable configuration, where components can be scaled separately as appropriate for enterprise environments.

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

- Alpine SecDB database
- AWS UpdateInfo
- Debian Oval database
- Oracle Oval database
- RHEL Oval database
- SUSE Oval database
- Ubuntu Oval database
- Pyup.io (python) database

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



NOTE

With the release of Red Hat Quay 3.4, the new Clair V4 (image `registry.redhat.io/quay/clair-rhel8`) fully replaces the prior Clair V2 (image `quay.io/redhat/clair-jwt`). See below for how to run V2 in read-only mode while V4 is updating.

7.1. SETTING UP CLAIR ON A RED HAT QUAY OPENSIFT DEPLOYMENT

7.1.1. Deploying Via the Quay Operator

To set up Clair V4 on a new Red Hat Quay deployment on OpenShift, it is highly recommended to use the Quay Operator. By default, the Quay Operator will install or upgrade a Clair deployment along with your Red Hat Quay deployment and configure Clair security scanning automatically.

7.1.2. Manually Deploying Clair

To configure Clair V4 on an existing Red Hat Quay OpenShift deployment running Clair V2, first ensure Red Hat Quay has been upgraded to at least version 3.4.0. Then use the following steps to manually set up Clair V4 alongside Clair V2.

1. Set your current project to the name of the project in which Red Hat Quay is running. For example:

```
$ oc project quay-enterprise
```

2. Create a Postgres deployment file for Clair v4 (for example, **clairv4-postgres.yaml**) as follows.

clairv4-postgres.yaml

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: clairv4-postgres
  namespace: quay-enterprise
  labels:
    quay-component: clairv4-postgres
spec:
  replicas: 1
  selector:
    matchLabels:
      quay-component: clairv4-postgres
  template:
    metadata:
      labels:
        quay-component: clairv4-postgres
    spec:
      volumes:
        - name: postgres-data
          persistentVolumeClaim:
            claimName: clairv4-postgres
      containers:
        - name: postgres
          image: postgres:11.5
          imagePullPolicy: "IfNotPresent"
          ports:
            - containerPort: 5432
          env:
            - name: POSTGRES_USER
              value: "postgres"
            - name: POSTGRES_DB
              value: "clair"
            - name: POSTGRES_PASSWORD
              value: "postgres"
            - name: PGDATA
              value: "/etc/postgres/data"
          volumeMounts:
            - name: postgres-data
              mountPath: "/etc/postgres"
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: clairv4-postgres
  labels:
    quay-component: clairv4-postgres
spec:
  accessModes:
    - "ReadWriteOnce"
  resources:
    requests:
```

```

    storage: "5Gi"
    volumeName: "clairv4-postgres"
---
apiVersion: v1
kind: Service
metadata:
  name: clairv4-postgres
  labels:
    quay-component: clairv4-postgres
spec:
  type: ClusterIP
  ports:
    - port: 5432
      protocol: TCP
      name: postgres
      targetPort: 5432
  selector:
    quay-component: clairv4-postgres

```

3. Deploy the postgres database as follows:

```
$ oc create -f ./clairv4-postgres.yaml
```

4. Create a Clair **config.yaml** file to use for Clair v4. For example:

config.yaml

```

introspection_addr: :8089
http_listen_addr: :8080
log_level: debug
indexer:
  connstring: host=clairv4-postgres port=5432 dbname=clair user=postgres
  password=postgres sslmode=disable
  scanlock_retry: 10
  layer_scan_concurrency: 5
  migrations: true
matcher:
  connstring: host=clairv4-postgres port=5432 dbname=clair user=postgres
  password=postgres sslmode=disable
  max_conn_pool: 100
  run: ""
  migrations: true
  indexer_addr: clair-indexer
notifier:
  connstring: host=clairv4-postgres port=5432 dbname=clair user=postgres
  password=postgres sslmode=disable
  delivery: 1m
  poll_interval: 5m
  migrations: true
auth:
  psk:
    key: MTU5YzA4Y2ZkNzJoMQ== 1
    iss: ["quay"]
# tracing and metrics
trace:

```

```

name: "jaeger"
probability: 1
jaeger:
  agent_endpoint: "localhost:6831"
  service_name: "clair"
metrics:
  name: "prometheus"

```

- 1 To generate a Clair pre-shared key (PSK), enable **scanning** in the Security Scanner section of the User Interface and click **Generate PSK**.

More information about Clair's configuration format can be found in [upstream Clair documentation](#).

1. Create a secret from the Clair **config.yaml**:

```
$ oc create secret generic clairv4-config-secret --from-file=./config.yaml
```

2. Create the Clair v4 deployment file (for example, **clair-combo.yaml**) and modify it as necessary:

clair-combo.yaml

```

---
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  labels:
    quay-component: clair-combo
  name: clair-combo
spec:
  replicas: 1
  selector:
    matchLabels:
      quay-component: clair-combo
  template:
    metadata:
      labels:
        quay-component: clair-combo
    spec:
      containers:
        - image: registry.redhat.io/quay/clair-rhel8:v3.7.13 1
          imagePullPolicy: IfNotPresent
          name: clair-combo
          env:
            - name: CLAIR_CONF
              value: /clair/config.yaml
            - name: CLAIR_MODE
              value: combo
      ports:
        - containerPort: 8080
          name: clair-http
          protocol: TCP
        - containerPort: 8089
          name: clair-intro

```

```

    protocol: TCP
    volumeMounts:
      - mountPath: /clair/
        name: config
    imagePullSecrets:
      - name: redhat-pull-secret
    restartPolicy: Always
    volumes:
      - name: config
        secret:
          secretName: clairv4-config-secret
  ---
apiVersion: v1
kind: Service
metadata:
  name: clairv4 2
  labels:
    quay-component: clair-combo
spec:
  ports:
    - name: clair-http
      port: 80
      protocol: TCP
      targetPort: 8080
    - name: clair-introspection
      port: 8089
      protocol: TCP
      targetPort: 8089
  selector:
    quay-component: clair-combo
  type: ClusterIP

```

- 1 Change image to latest clair image name and version.
- 2 With the Service set to clairv4, the scanner endpoint for Clair v4 is entered later into the Red Hat Quay config.yaml in the **SECURITY_SCANNER_V4_ENDPOINT** as **http://clairv4**.

3. Create the Clair v4 deployment as follows:

```
$ oc create -f ./clair-combo.yaml
```

4. Modify the **config.yaml** file for your Red Hat Quay deployment to add the following entries at the end:

```

FEATURE_SECURITY_NOTIFICATIONS: true
FEATURE_SECURITY_SCANNER: true
SECURITY_SCANNER_V4_ENDPOINT: http://clairv4 1

```

- 1 Identify the Clair v4 service endpoint

5. Redeploy the modified **config.yaml** to the secret containing that file (for example, **quay-enterprise-config-secret**):

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

- For the new **config.yaml** to take effect, you need to restart the Red Hat Quay pods. Simply deleting the **quay-app** pods causes pods with the updated configuration to be deployed.

At this point, images in any of the organizations identified in the namespace whitelist will be scanned by Clair v4.

7.2. SETTING UP CLAIR ON A NON-OPENSIFT RED HAT QUAY DEPLOYMENT

For Red Hat Quay deployments not running on OpenShift, it is possible to configure Clair security scanning manually. Red Hat Quay deployments already running Clair V2 can use the instructions below to add Clair V4 to their deployment.

- Deploy a (preferably fault-tolerant) Postgres database server. Note that Clair requires the **uuid-oss** extension to be added to its Postgres database. If the user supplied in Clair's **config.yaml** has the necessary privileges to create the extension then it will be added automatically by Clair itself. If not, then the extension must be added before starting Clair. If the extension is not present, the following error will be displayed when Clair attempts to start.

```
ERROR: Please load the "uuid-oss" extension. (SQLSTATE 42501)
```

- Create a Clair config file in a specific folder, for example, **/etc/clairv4/config/config.yaml**).

config.yaml

```
introspection_addr: :8089
http_listen_addr: :8080
log_level: debug
indexer:
  connstring: host=clairv4-postgres port=5432 dbname=clair user=postgres
  password=postgres sslmode=disable
  scanlock_retry: 10
  layer_scan_concurrency: 5
  migrations: true
matcher:
  connstring: host=clairv4-postgres port=5432 dbname=clair user=postgres
  password=postgres sslmode=disable
  max_conn_pool: 100
  run: ""
  migrations: true
  indexer_addr: clair-indexer
notifier:
  connstring: host=clairv4-postgres port=5432 dbname=clair user=postgres
  password=postgres sslmode=disable
  delivery_interval: 1m
  poll_interval: 5m
  migrations: true

# tracing and metrics
trace:
  name: "jaeger"
```

```

probability: 1
jaeger:
  agent_endpoint: "localhost:6831"
  service_name: "clair"
metrics:
  name: "prometheus"

```

More information about Clair's configuration format can be found in [upstream Clair documentation](#).

1. Run Clair via the container image, mounting in the configuration from the file you created.

```

$ podman run -p 8080:8080 -p 8089:8089 -e CLAIR_CONF=/clair/config.yaml -e
CLAIR_MODE=combo -v /etc/clair4/config:/clair -d registry.redhat.io/quay/clair-rhel8:v3.7.13

```

2. Follow the remaining instructions from the previous section for configuring Red Hat Quay to use the new Clair V4 endpoint.

Running multiple Clair containers in this fashion is also possible, but for deployment scenarios beyond a single container the use of a container orchestrator like Kubernetes or OpenShift is strongly recommended.

7.3. ADVANCED CLAIR CONFIGURATION

7.3.1. Unmanaged Clair configuration

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

7.3.1.1. Unmanaging a Clair database

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

Procedure

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

```

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

```

```

managed: false
- kind: clairpostgres
  managed: false

```

7.3.1.2. Configuring a custom Clair database

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

Procedure

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

```

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

```

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

```

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

```



NOTE

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

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

```

apiVersion: v1
kind: Secret
metadata:
  name: config-bundle-secret
  namespace: quay-enterprise
data:
  config.yaml: <base64 encoded Quay config>

```



```
clair-config.yaml: <base64 encoded Clair config>
extra_ca_cert_<name>: <base64 encoded ca cert>
clair-ssl.crt: >-
clair-ssl.key: >-
```



NOTE

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

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

7.3.2. Running a custom Clair configuration with a managed database

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

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



NOTE

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

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

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

7.4. CLAIR CRDA CONFIGURATION

7.4.1. 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.

- 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.

7.5. USING CLAIR

- Log in to your Red Hat Quay cluster and select an organization for which you have configured Clair scanning.
- Select a repository from that organization that holds some images and select Tags from the left navigation. The following figure shows an example of a repository with two images that have been scanned:

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

- If vulnerabilities are found, select to under the Security Scan column for the image to see either all vulnerabilities or those that are fixable. The following figure shows information on all vulnerabilities found:

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...

7.6. CVE RATINGS FROM THE NATIONAL VULNERABILITY DATABASE

With Clair v4.2, enrichment data is now viewable in the Quay UI. Additionally, Clair v4.2 adds CVSS scores from the National Vulnerability Database for detected vulnerabilities.

With this change, if the vulnerability has a CVSS score that is within 2 levels of the distro's score, the Quay UI present's the distro's score by default. For example:

DESCRIPTION

The SUSE coreutils-118n.patch for GNU coreutils allows context-dependent attackers to cause a denial of service (segmentation fault and crash) via a long string to the uniq command, which triggers a stack-based buffer overflow in the alloca function.

▼ CVE-2015-4041 Unknown coreutils 8.30-3 0.0 ADD roots.tar / # buildkit

SEVERITY NOTE

Note that this vulnerability was originally given a CVSSv3 score of 7.8 by NVD but was subsequently reclassified as Unknown by Unknown

VECTORS

Attack Vector	Attack Complexity	Privileges Required	User Interaction	Scope	Confidentiality Impact	Integrity Impact	Availability Impact
● Network	▲ Low	● None	▲ None	● Unchanged	▲ High	▲ High	▲ High
● Adjacent Network	● High	● Low	● Required	● Changed	● Low	● None	● Low
● Local		● High			● None	● None	● None
● Physical							

DESCRIPTION

The keycompare_mb function in sort.c in sort in GNU Coreutils through 8.23 on 64-bit platforms performs a size calculation without considering the number of bytes occupied by multibyte characters, which allows attackers to cause a denial of service (heap-based buffer overflow and application crash) or possibly have unspecified other impact via long UTF-8 strings.

This differs from the previous interface, which would only display the following information:

▼ CVE-2015-4041 Unknown coreutils 8.30-3 0.0 ADD roots.tar / # buildkit

DESCRIPTION

The keycompare_mb function in sort.c in sort in GNU Coreutils through 8.23 on 64-bit platforms performs a size calculation without considering the number of bytes occupied by multibyte characters, which allows attackers to cause a denial of service (heap-based buffer overflow and application crash) or possibly have unspecified other impact via long UTF-8 strings.

7.7. CONFIGURING CLAIR FOR DISCONNECTED ENVIRONMENTS

Clair utilizes a set of components called Updaters to handle the fetching and parsing of data from various vulnerability databases. These Updaters are set up by default to pull vulnerability data directly from the internet and work out of the box. For customers in disconnected environments without direct access to the internet this poses a problem. Clair supports these environments through the ability to work with different types of update workflows that take into account network isolation. Using the **clairctl** command line utility, any process can easily fetch Updater data from the internet via an open host, securely transfer the data to an isolated host, and then import the Updater data on the isolated host into Clair itself.

The steps are as follows.

1. First ensure that your Clair configuration has disabled automated Updaters from running.

config.yaml

```
matcher:
  disable_updaters: true
```

2. Export out the latest Updater data to a local archive. This requires the **clairctl** tool which can be run directly as a binary, or via the Clair container image. Assuming your Clair configuration is in **/etc/clairv4/config/config.yaml**, to run via the container image:

```
$ podman run -it --rm -v /etc/clairv4/config:/etc/clairv4/config:Z -v /path/to/output/directory:/updates:Z --entrypoint /bin/clairctl registry.redhat.io/quay/clair-rhel8:v3.7.13 --config /etc/clairv4/config/config.yaml export-updaters /updates/updaters.gz
```

Note that you need to explicitly reference the Clair configuration. This will create the Updater archive in **/etc/clairv4/updaters/updaters.gz**. If you want to ensure the archive was created without any errors from the source databases, you can supply the **--strict** flag to **clairctl**. The

archive file should be copied over to a volume that is accessible from the disconnected host running Clair. From the disconnected host, use the same procedure now to import the archive into Clair.

```
$ podman run -it --rm -v /etc/clairv4/config:/cfg:Z -v /path/to/output/directory:/updaters:Z --
entrypoint /bin/clairctl registry.redhat.io/quay/clair-rhel8:v3.7.13 --config /cfg/config.yaml
import-updaters /updaters/updaters.gz
```

7.7.1. Mapping repositories to Common Product Enumeration (CPE) information

Clair's RHEL scanner relies on a Common Product Enumeration (CPE) file to properly map RPM packages to the corresponding security data, in order to produce matching results. This file must be present, or access to the file must be allowed, for the scanner to properly process RPMs. If the file is not present, RPMs installed in the container images will not be scanned.

Red Hat publishes the JSON mapping file at <https://www.redhat.com/security/data/metrics/repository-to-cpe.json>.

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

- For standalone Quay and Clair deployments, the mapping file must be loaded into the Clair pod.
- For Operator-based deployments, you must set the Clair component to **unmanaged**. Then deploy Clair manually, setting the configuration to load a local copy of the mapping file.

Use the **repo2cpe_mapping_file** field in the Clair configuration to specify the file:

```
indexer:
  scanner:
    repo:
      rhel-repository-scanner:
        repo2cpe_mapping_file: /path/to/repository-to-cpe.json
```

Further information is available from Red Hat at [How to accurately match OVAL security data to installed RPMs](#).

7.8. CLAIR UPDATER URLS

The following are the HTTP hosts and paths that Clair will attempt to talk to in a default configuration. This list is non-exhaustive, as some servers will issue redirects and some request URLs are constructed dynamically.

- <https://secdb.alpinelinux.org/>
- http://repo.us-west-2.amazonaws.com/2018.03/updates/x86_64/mirror.list
- https://cdn.amazonlinux.com/2/core/latest/x86_64/mirror.list
- <https://www.debian.org/security/oval/>
- <https://linux.oracle.com/security/oval/>
- https://packages.vmware.com/photon/photon_oval_definitions/

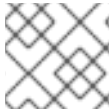
- <https://github.com/pyupio/safety-db/archive/>
- <https://catalog.redhat.com/api/containers/>
- <https://www.redhat.com/security/data/>
- <https://support.novell.com/security/oval/>
- <https://people.canonical.com/~ubuntu-security/oval/>

7.9. ADDITIONAL INFORMATION

For detailed documentation on the internals of Clair, including how the microservices are structured, please see the [Upstream Clair](#) and [ClairCore](#) documentation.

CHAPTER 8. SCANNING POD IMAGES WITH THE CONTAINER SECURITY OPERATOR

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



NOTE

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

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

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



NOTE

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

8.1. DOWNLOADING AND RUNNING THE CONTAINER SECURITY OPERATOR IN OPENSIFT CONTAINER PLATFORM

Use the following procedure to download the Container Security Operator.



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.

1. Go to Operators → OperatorHub (select Security) to see the available **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. Optionally, 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 cert to the CSO (restart the Operator pod for the new certs to take effect):

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

- Open the OpenShift Dashboard (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 figure:

The screenshot shows the OpenShift Dashboard with a modal window titled "Quay Image Security breakdown". The modal contains the following information:

- Quay Image Security breakdown**
- Container images from Quay are analyzed to identify vulnerabilities. Images from other registries are not scanned.
- Severity**: 1 High, 1 Fixable
- 1 total** (indicated by a red circle)
- Fixable Vulnerabilities**: nss-tools, 1 namespaces

The dashboard background shows the "Dashboards" section with "Overview" selected. The "Status" section indicates that the Cluster and Control Plane are healthy, but there is a warning for "Quay Image Security" with 1 vulnerability. A message states: "7 minutes ago: A client in the cluster is using deprecated extensions/v1beta1 API that will be removed".

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

- 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:

The screenshot shows the Red Hat Quay.io interface for a vulnerability scan. The interface displays the following information:

- RED HAT Quay.io** (Logo)
- Navigation: EXPLORE, APPLICATIONS, REPOSITORIES, TUTORIAL
- Image ID: f54fd70e06e7
- Quay Security Scanner has detected 6 vulnerabilities.**
- Patches are available for 6 vulnerabilities.**
- 6 High-level vulnerabilities.**
- Vulnerabilities Table:**

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

- 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

At this point, you know what images are vulnerable, what you need to do to fix those vulnerabilities, and every namespace that the image was run in. So you can:

- Alert anyone running the image that they need to correct the vulnerability
- Stop the images from running (by deleting the deployment or other object that started the pod the image is in)

Note that if you do delete the pod, it may take a few minutes for the vulnerability to reset on the dashboard.

8.2. QUERY IMAGE VULNERABILITIES FROM THE CLI

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

```
$ oc get vuln --all-namespaces
NAMESPACE  NAME  AGE
default    sha256.ca90... 6m56s
skynet     sha256.ca90... 9m37s
```

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

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


CHAPTER 9. INTEGRATING RED HAT QUAY INTO OPENSIFT CONTAINER PLATFORM WITH THE QUAY BRIDGE OPERATOR

Using the Quay Bridge Operator, you can replace the integrated container registry in OpenShift Container Platform with a Red Hat Quay registry. By doing this, your integrated OpenShift Container Platform registry becomes a highly available, enterprise-grade Red Hat Quay registry with enhanced role based access control (RBAC) features.

The primary goal of the Quay Bridge Operator is to duplicate the features of the integrated OpenShift Container Platform registry in the new Red Hat Quay registry. The features enabled with the Quay Bridge Operator include:

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

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

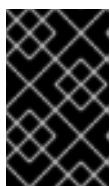
9.1. SETTING UP RED HAT QUAY FOR THE QUAY BRIDGE OPERATOR

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

Procedure

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

- **Create Repositories**
 - **View all visible repositories**
 - **Read/Write to any accessible repositories**
 - **Administer User**
 - **Read User Information**
8. Review the assigned permissions.
 9. Select **Authorize Application** and then confirm confirm the authorization by selecting **Authorize Application**.
 10. Save the generated access token.



IMPORTANT

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

9.2. INSTALLING THE QUAY BRIDGE OPERATOR ON OPENSIFT CONTAINER PLATFORM

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

Prerequisites

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

Procedure

1. Open the **Administrator** perspective of the web console and navigate to **Operators** → **OperatorHub** on the navigation pane.
2. Search for **Quay Bridge Operator**, click the **Quay Bridge Operator** title, and then click **Install**.
3. Select the version to install, for example, **stable-3.7**, and then click **Install**.
4. Click **View Operator** when the installation finishes to go to the Quay Bridge Operator's **Details** page. Alternatively, you can click **Installed Operators** → **Red Hat Quay Bridge Operator** to go to the **Details** page.

9.3. CREATING AN OPENSIFT CONTAINER PLATFORM SECRET FOR THE OAUTH TOKEN

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

Prerequisites

Prerequisites

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

Procedure

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

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

9.4. CREATING THE QUAYINTEGRATION CUSTOM RESOURCE

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

Prerequisites

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

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

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

Procedure

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

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

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

```
apiVersion: quay.redhat.com/v1  
kind: QuayIntegration  
metadata:  
  name: example-quayintegration  
spec:  
  clusterID: openshift 1  
  credentialsSecret:
```

```

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

```

- 1** The clusterID value should be unique across the entire ecosystem. This value is required and defaults to **openshift**.
- 2** The **credentialsSecret** property refers to the namespace and name of the secret containing the token that was previously created.
- 3** Replace the **QUAY_URL** with the hostname of your Red Hat Quay instance.
- 4** If Red Hat Quay is using self signed certificates, set the property to **insecureRegistry: true**.

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

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

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

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

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

Procedure

1. Open the **Administrator** perspective of the web console and navigate to **Operators** → **Installed Operators**.
2. Click **Red Hat Quay Bridge Operator**.
3. On the **Details** page of the Quay Bridge Operator, click **Create Instance** on the **Quay Integration** API card.
4. On the **Create QuayIntegration** page, enter the following required information in either **Form view** or **YAML view**:
 - **Name**: The name that will refer to the **QuayIntegration** custom resource object.
 - **Cluster ID**: The ID associated with this cluster. This value should be unique across the entire ecosystem. Defaults to **openshift** if left unspecified.
 - **Credentials secret**: Refers to the namespace and name of the secret containing the token that was previously created.
 - **Quay hostname**: The hostname of the Quay registry.
For a list of all configuration fields, see "QuayIntegration configuration fields".

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

9.5. QUAYINTEGRATION CONFIGURATION FIELDS

The following configuration fields are available for the QuayIntegration custom resource:

Name	Description	Schema
allowlistNamespaces (Optional)	A list of namespaces to include.	Array
clusterID (Required)	The ID associated with this cluster.	String
credentialsSecret.key (Required)	The secret containing credentials to communicate with the Quay registry.	Object
denylistNamespaces (Optional)	A list of namespaces to exclude.	Array
insecureRegistry (Optional)	Whether to skip TLS verification to the Quay registry	Boolean
quayHostname (Required)	The hostname of the Quay registry.	String
scheduledImageStreamImport (Optional)	Whether to enable image stream importing.	Boolean

CHAPTER 10. REPOSITORY MIRRORING

10.1. REPOSITORY MIRRORING

Red Hat Quay repository mirroring lets you mirror images from external container registries (or another local registry) into your Red Hat Quay cluster. Using repository mirroring, you can synchronize images to Red Hat Quay based on repository names and tags.

From your Red Hat Quay cluster with repository mirroring enabled, you can:

- Choose a repository from an external registry to mirror
- Add credentials to access the external registry
- Identify specific container image repository names and tags to sync
- Set intervals at which a repository is synced
- Check the current state of synchronization

To use the mirroring functionality, you need to:

- Enable repository mirroring in the Red Hat Quay configuration
- Run a repository mirroring worker
- Create mirrored repositories

All repository mirroring configuration can be performed using the configuration tool UI or via the Red Hat Quay API

10.2. REPOSITORY MIRRORING VERSUS GEO-REPLICATION

Red Hat Quay geo-replication mirrors the entire image storage backend data between 2 or more different storage backends while the database is shared (one Red Hat Quay registry with two different blob storage endpoints). The primary use cases for geo-replication are:

- Speeding up access to the binary blobs for geographically dispersed setups
- Guaranteeing that the image content is the same across regions

Repository mirroring synchronizes selected repositories (or subsets of repositories) from one registry to another. The registries are distinct, with each registry having a separate database and separate image storage. The primary use cases for mirroring are:

- Independent registry deployments in different datacenters or regions, where a certain subset of the overall content is supposed to be shared across the datacenters / regions
- Automatic synchronization or mirroring of selected (whitelisted) upstream repositories from external registries into a local Red Hat Quay deployment



NOTE

Repository mirroring and geo-replication can be used simultaneously.

Table 10.1. Red Hat Quay Repository mirroring versus geo-replication

Feature / Capability	Geo-replication	Repository mirroring
What is the feature designed to do?	A shared, global registry	Distinct, different registries
What happens if replication or mirroring hasn't been completed yet?	The remote copy is used (slower)	No image is served
Is access to all storage backends in both regions required?	Yes (all Red Hat Quay nodes)	No (distinct storage)
Can users push images from both sites to the same repository?	Yes	No
Is all registry content and configuration identical across all regions (shared database)	Yes	No
Can users select individual namespaces or repositories to be mirrored?	No	Yes
Can users apply filters to synchronization rules?	No	Yes
Are individual / different RBAC configurations allowed in each region	No	Yes

10.3. USING REPOSITORY MIRRORING

Here are some features and limitations of Red Hat Quay repository mirroring:

- With repository mirroring, you can mirror an entire repository or selectively limit which images are synced. Filters can be based on a comma-separated list of tags, a range of tags, or other means of identifying tags through regular expressions.
- Once a repository is set as mirrored, you cannot manually add other images to that repository.
- Because the mirrored repository is based on the repository and tags you set, it will hold only the content represented by the repo / tag pair. In other words, if you change the tag so that some images in the repository no longer match, those images will be deleted.
- Only the designated robot can push images to a mirrored repository, superseding any role-based access control permissions set on the repository.
- With a mirrored repository, a user can pull images (given read permission) from the repository but can not push images to the repository.

- Changing settings on your mirrored repository can be performed in the Red Hat Quay UI, using the Repositories → Mirrors tab for the mirrored repository you create.
- Images are synced at set intervals, but can also be synced on demand.


10.4. MIRRORING CONFIGURATION UI

1. Start the **Quay** container in configuration mode and select the Enable Repository Mirroring check box. If you want to require HTTPS communications and verify certificates during mirroring, select the HTTPS and cert verification check box.

Repository Mirroring

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

Enable Repository Mirroring

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

Require HTTPS and verify certificates of Quay registry during mirror.

2. Validate and download the **configuration** file, and then restart Quay in registry mode using the updated config file.

10.5. MIRRORING CONFIGURATION FIELDS

Table 10.2. Mirroring configuration

Field	Type	Description
FEATURE_REPO_MIRROR	Boolean	Enable or disable repository mirroring Default: false
REPO_MIRROR_INTERVAL	Number	The number of seconds between checking for repository mirror candidates Default: 30
REPO_MIRROR_SERVER_HOSTNAME	String	Replaces the SERVER_HOSTNAME as the destination for mirroring. Default: None Example: openshift-quay-service

Field	Type	Description
REPO_MIRROR_TLS_VERIFY	Boolean	Require HTTPS and verify certificates of Quay registry during mirror. Default: false
REPO_MIRROR_ROLLBACK	Boolean	When set to true , the repository rolls back after a failed mirror attempt. Default: false

10.6. MIRRORING WORKER

Use the following procedure to start the repository mirroring worker.

Procedure

- If you have not configured TLS communications using a **/root/ca.crt** certificate, enter the following command to start a **Quay** pod with the **repomirror** option:

```
$ sudo podman run -d --name mirroring-worker \
-v $QUAY/config:/conf/stack:Z \
{productrepo}/{quayimage}:{productminv} repomirror
```

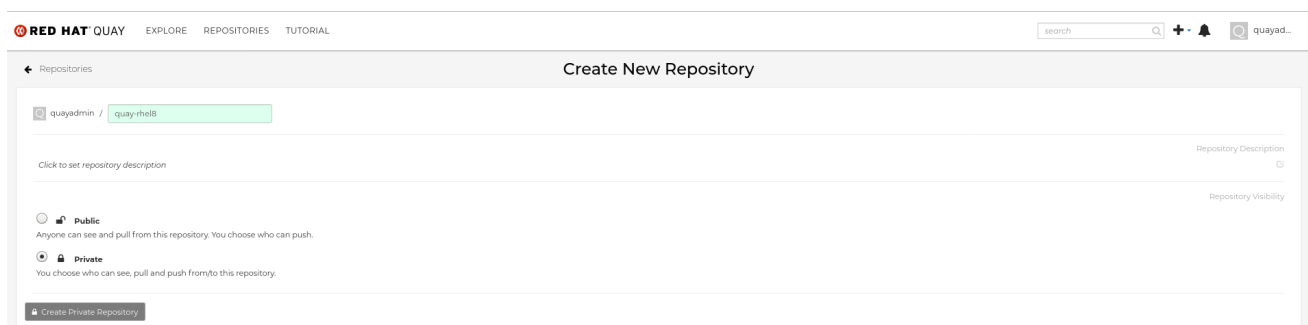
- If you have configured TLS communications using a **/root/ca.crt** certificate, enter the following command to start the repository mirroring worker:

```
$ sudo podman run -d --name mirroring-worker \
-v $QUAY/config:/conf/stack:Z \
-v /root/ca.crt:/etc/pki/ca-trust/source/anchors/ca.crt:Z \
{productrepo}/{quayimage}:{productminv} repomirror
```

10.7. CREATING A MIRRORED REPOSITORY

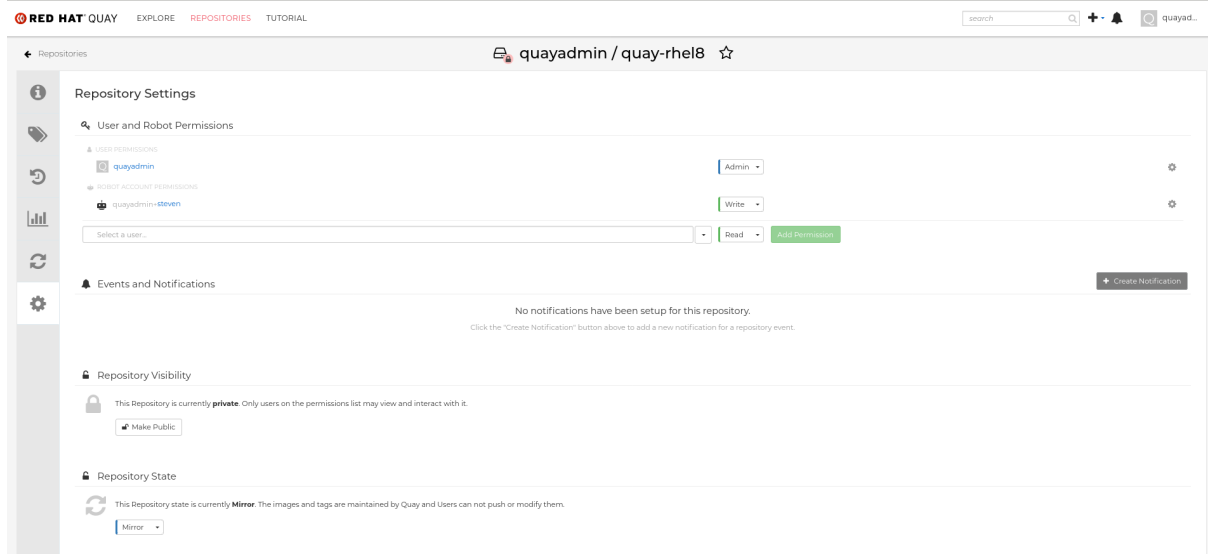
The steps shown in this section assume you already have enabled repository mirroring in the configuration for your Red Hat Quay cluster and that you have a deployed a mirroring worker.

When mirroring a repository from an external container registry, create a new private repository. Typically the same name is used as the target repository, for example, **quay-rhel8**:

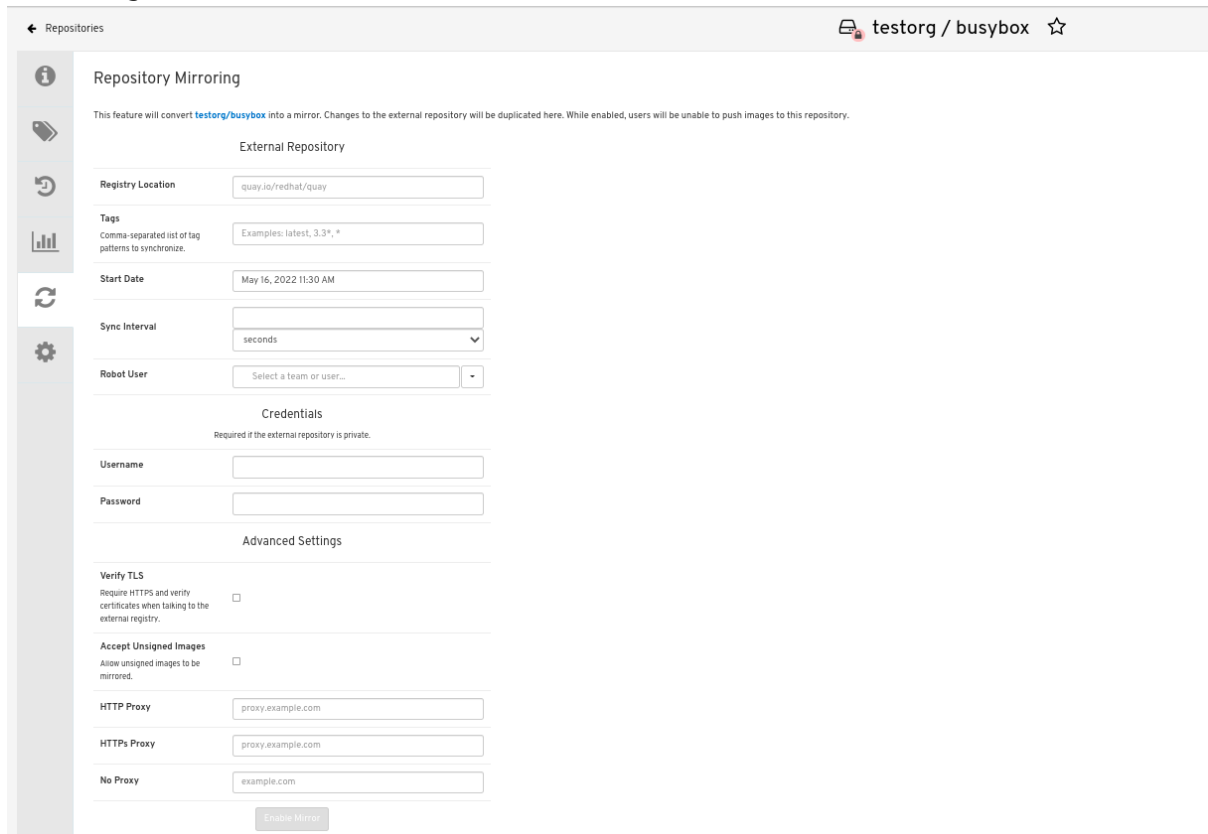


10.7.1. Repository mirroring settings

1. In the Settings tab, set the Repository State to **Mirror**:



2. In the Mirror tab, enter the details for connecting to the external registry, along with the tags, scheduling and access information:



3. Enter the details as required in the following fields:

- **Registry Location:** The external repository you want to mirror, for example, **registry.redhat.io/quay/quay-rhel8**
- **Tags:** This field is required. You may enter a comma-separated list of individual tags or tag patterns. (See *Tag Patterns* section for details.)



NOTE

In order for Quay to get the list of tags in the remote repository, one of the following requirements must be met:

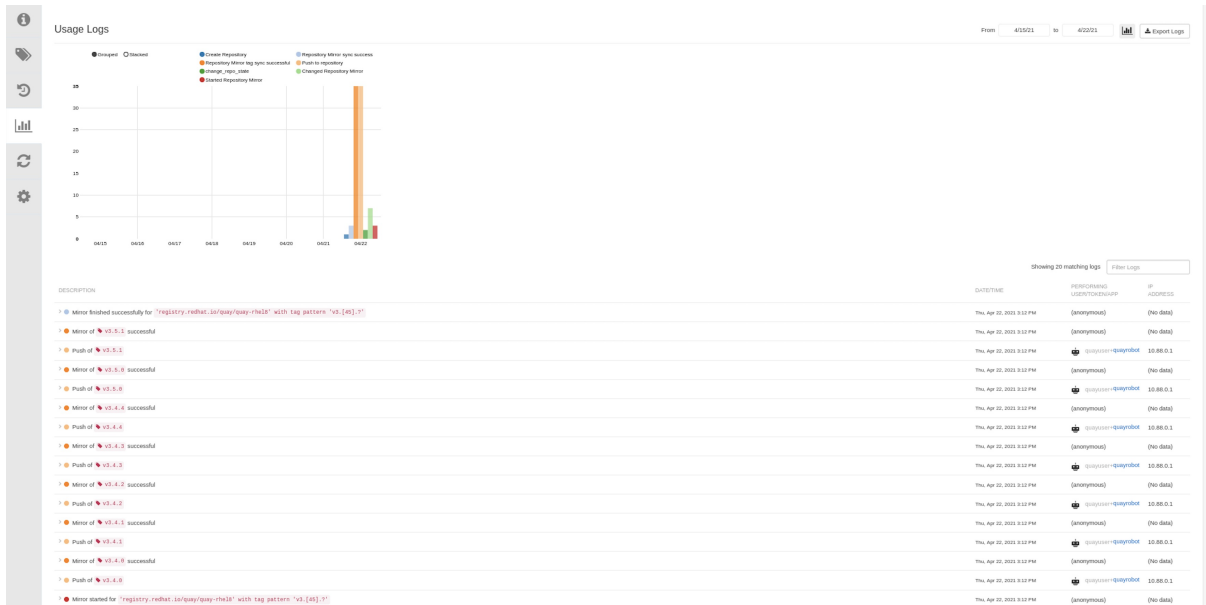
- An image with the "latest" tag must exist in the remote repository *OR*
 - At least one explicit tag, without pattern matching, must exist in the list of tags that you specify
-
- **Start Date:** The date on which mirroring begins. The current date and time is used by default.
 - **Sync Interval:** Defaults to syncing every 24 hours. You can change that based on hours or days.
 - **Robot User:** Create a new robot account or choose an existing robot account to do the mirroring.
 - **Username:** The username for accessing the external registry holding the repository you are mirroring.
 - **Password:** The password associated with the Username. Note that the password cannot include characters that require an escape character (\).

10.7.2. Advanced settings

- In the Advanced Settings section, configure TLS and proxy, if required:
- **Verify TLS:** Check this box if you want to require HTTPS and to verify certificates, when communicating with the target remote registry.
- **HTTP Proxy:** Identify the HTTP proxy server needed to access the remote site, if one is required.
- **HTTPS Proxy:** Identify the HTTPS proxy server needed to access the remote site, if one is required.
- **No Proxy:** List of locations that do not require proxy

10.7.3. Synchronize now

- To perform an immediate mirroring operation, press the Sync Now button on the repository's Mirroring tab. The logs are available on the Usage Logs tab:



When the mirroring is complete, the images will appear in the Tags tab:

The Repository Tags interface shows a list of tags for the repository 'quayuser / quay-rhel8'. The tags listed are v3.5.1, v3.5.0, v3.4.4, v3.4.3, v3.4.2, v3.4.1, and v3.4.0. Each tag entry includes details such as last modified time, size, expiration, and manifest information.

TAG	LAST MODIFIED	SIZE	EXPIRES	MANIFEST
v3.5.1	a minute ago	N/A	Never	SHA256: 3d4d87d375
v3.5.0	a minute ago	N/A	Never	SHA256: 9519f6ae0c
v3.4.4	a minute ago	N/A	Never	SHA256: v41316a29e
v3.4.3	a minute ago	N/A	Never	SHA256: 36a27129f12
v3.4.2	a minute ago	N/A	Never	SHA256: 768b136a07
v3.4.1	a minute ago	N/A	Never	SHA256: 4f85c5340b
v3.4.0	a minute ago	N/A	Never	SHA256: 5e3c488993

Below is an example of a completed Repository Mirroring screen:

The Repository Mirroring configuration screen shows the following settings:

- Enabled:** Checked
- External Repository:** registry.redhat.io/quay/quay-rhel8
- Tags:** v3.[45]*
- Sync Interval:** 1 hour
- Next Sync Date:** May 27, 2022 4:17 PM
- Robot User:** quayadmin-robot
- Advanced Settings:**
 - Credentials:** rhn-support-strobinet
 - Verify TLS:** Unchecked
 - HTTP Proxy:** None
 - HTTPS Proxy:** None
 - No Proxy:** None
- Status:** Scheduled
- Timeout:** None
- Retries Remaining:** 3/3

10.8. EVENT NOTIFICATIONS FOR MIRRORING

There are three notification events for repository mirroring:

- Repository Mirror Started
- Repository Mirror Success
- Repository Mirror Unsuccessful

The events can be configured inside the Settings tab for each repository, and all existing notification methods such as email, slack, Quay UI and webhooks are supported.

10.9. MIRRORING TAG PATTERNS

As noted above, at least one Tag must be explicitly entered (ie. not a tag pattern) or the tag "latest" must exist in the report repository. (The tag "latest" will not be synced unless specified in the tag list.). This is required for Quay to get the list of tags in the remote repository to compare to the specified list to mirror.

10.9.1. Pattern syntax

Pattern	Description
*	Matches all characters
?	Matches any single character
[seq]	Matches any character in seq
[!seq]	Matches any character not in seq

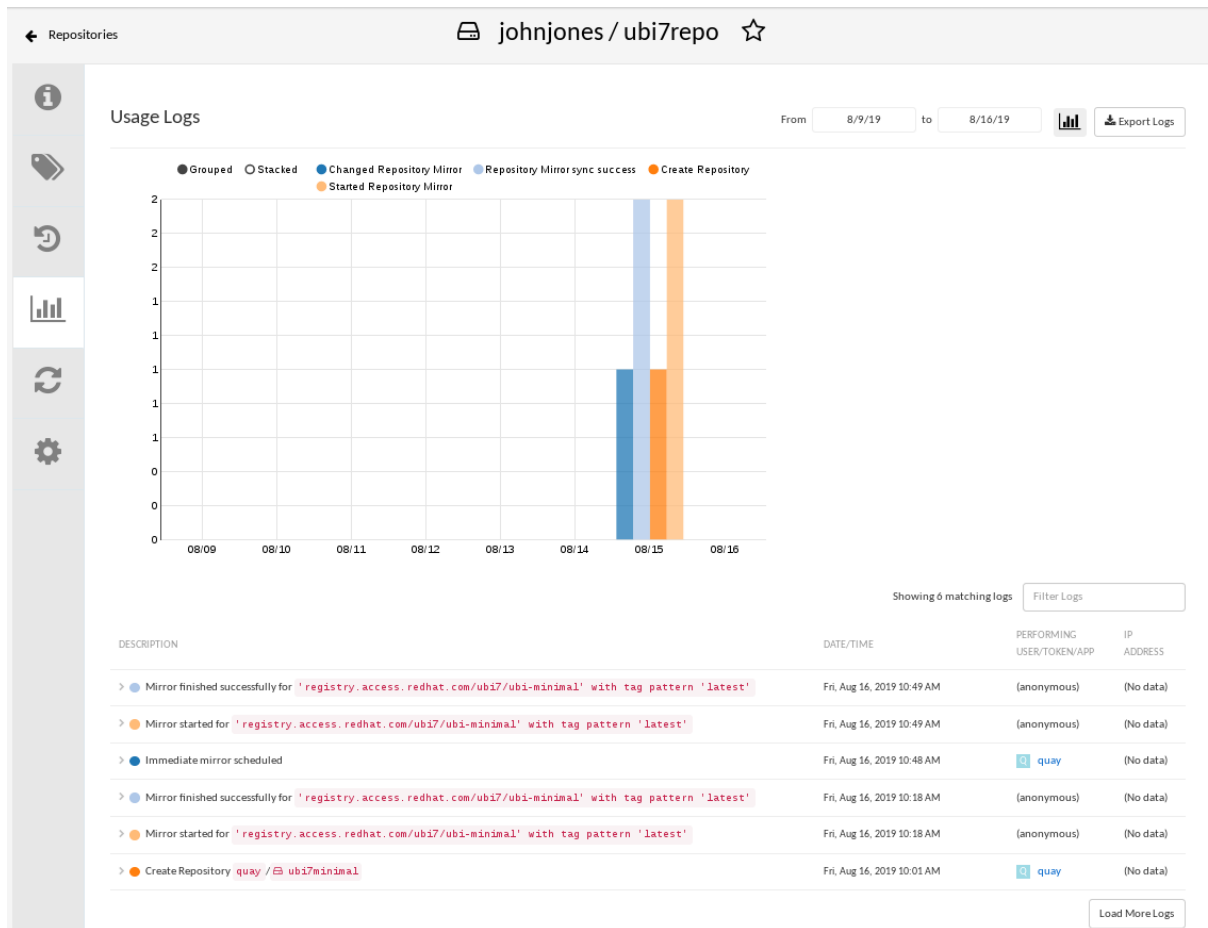
10.9.2. Example tag patterns

Example Pattern	Example Matches
v3*	v3.2, v3.1, v3.2, v3.2-4beta, v3.3
v3.*	v3.1, v3.2, v3.2-4beta
v3.?	v3.1, v3.2, v3.3
v3.[12]	v3.1, v3.2
v3.[12]*	v3.1, v3.2, v3.2-4beta
v3.[!1]*	v3.2, v3.2-4beta, v3.3

10.10. WORKING WITH MIRRORED REPOSITORIES

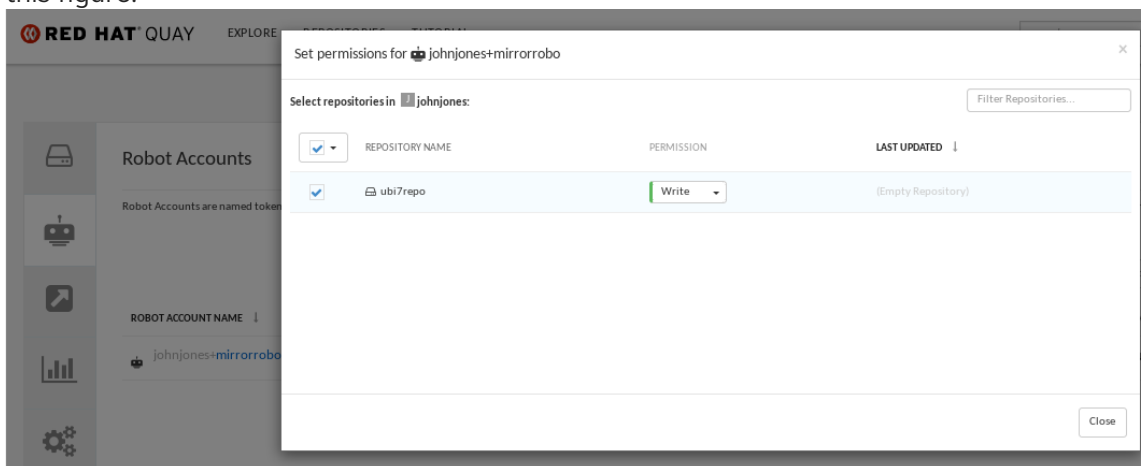
Once you have created a mirrored repository, there are several ways you can work with that repository. Select your mirrored repository from the Repositories page and do any of the following:

- **Enable/disable the repository.** Select the Mirroring button in the left column, then toggle the Enabled check box to enable or disable the repository temporarily.
- **Check mirror logs** To make sure the mirrored repository is working properly, you can check the mirror logs. To do that, select the Usage Logs button in the left column. Here's an example:

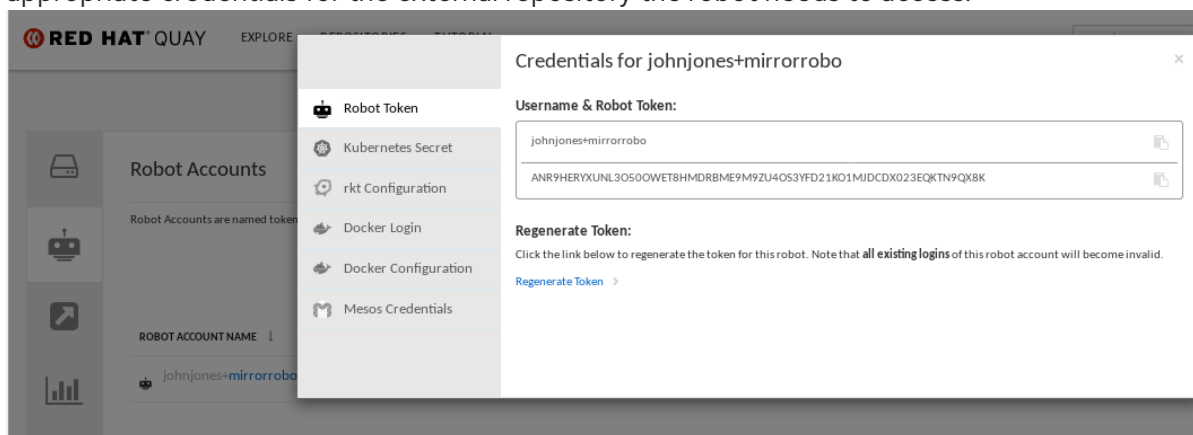


- **Sync mirror now:** To immediately sync the images in your repository, select the Sync Now button.
- **Change credentials:** To change the username and password, select DELETE from the Credentials line. Then select None and add the username and password needed to log into the external registry when prompted.
- **Cancel mirroring:** To stop mirroring, which keeps the current images available but stops new ones from being synced, select the CANCEL button.
- **Set robot permissions:** Red Hat Quay robot accounts are named tokens that hold credentials for accessing external repositories. By assigning credentials to a robot, that robot can be used across multiple mirrored repositories that need to access the same external registry. You can assign an existing robot to a repository by going to Account Settings, then selecting the Robot Accounts icon in the left column. For the robot account, choose the link under the REPOSITORIES column. From the pop-up window, you can:
 - Check which repositories are assigned to that robot.

- Assign read, write or Admin privileges to that robot from the PERMISSION field shown in this figure:



- **Change robot credentials:** Robots can hold credentials such as Kubernetes secrets, Docker login information, and Mesos bundles. To change robot credentials, select the Options gear on the robot's account line on the Robot Accounts window and choose View Credentials. Add the appropriate credentials for the external repository the robot needs to access.



- **Check and change general setting:** Select the Settings button (gear icon) from the left column on the mirrored repository page. On the resulting page, you can change settings associated with the mirrored repository. In particular, you can change User and Robot Permissions, to specify exactly which users and robots can read from or write to the repo.

10.11. REPOSITORY MIRRORING RECOMMENDATIONS

Best practices for repository mirroring include:

- Repository mirroring pods can run on any node. This means you can even run mirroring on nodes where Red Hat Quay is already running.
- Repository mirroring is scheduled in the database and runs in batches. As a result, more workers should mean faster mirroring, since more batches will be processed.
- The optimal number of mirroring pods depends on:
 - The total number of repositories to be mirrored
 - The number of images and tags in the repositories and the frequency of changes
 - Parallel batches

- You should balance your mirroring schedule across all mirrored repositories, so that they do not all start up at the same time.
- For a mid-size deployment, with approximately 1000 users and 1000 repositories, and with roughly 100 mirrored repositories, it is expected that you would use 3-5 mirroring pods, scaling up to 10 pods if required.

CHAPTER 11. LDAP AUTHENTICATION SETUP FOR RED HAT QUAY

The Lightweight Directory Access Protocol (LDAP) is an open, vendor-neutral, industry standard application protocol for accessing and maintaining distributed directory information services over an Internet Protocol (IP) network. Red Hat Quay supports using LDAP as an identity provider.

11.1. CONSIDERATIONS PRIOR TO ENABLING LDAP

11.1.1. Existing Quay deployments

Conflicts between user names can arise when you enable LDAP for an existing Quay deployment that already has users configured. Consider the scenario where a particular user, **alice**, was manually created in Quay prior to enabling LDAP. If the user name **alice** also exists in the LDAP directory, Quay will create a new user **alice-1** when **alice** logs in for the first time using LDAP, and will map the LDAP credentials to this account. This might not be what you want, for consistency reasons, and it is recommended that you remove any potentially conflicting local account names from Quay prior to enabling LDAP.

11.1.2. Manual User Creation and LDAP authentication

When Quay is configured for LDAP, LDAP-authenticated users are automatically created in Quay's database on first log in, if the configuration option **FEATURE_USER_CREATION** is set to **true**. If this option is set to **false**, the automatic user creation for LDAP users will fail and the user is not allowed to log in. In this scenario, the superuser needs to create the desired user account first. Conversely, if **FEATURE_USER_CREATION** is set to **true**, this also means that a user can still create an account from the Quay login screen, even if there is an equivalent user in LDAP.


11.2. SET UP LDAP CONFIGURATION

In the config tool, locate the Authentication section and select "LDAP" from the drop-down menu. Update LDAP configuration fields as required.

Internal Authentication

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

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

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

Authentication:

LDAP

- Here is an example of the resulting entry in the *config.yaml* file:

```
AUTHENTICATION_TYPE: LDAP
```

11.2.1. Full LDAP URI

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

Custom TLS Certificate: Please select a file to upload as **ldap.crt**: No file chosen
 If specified, the certificate (in PEM format) for the LDAP TLS connection.

Allow insecure: **Allow fallback to non-TLS connections**
 If enabled, LDAP will fallback to insecure non-TLS connections if TLS does not succeed.

- The full LDAP URI, including the `ldap://` or `ldaps://` prefix.
- A URI beginning with `ldaps://` will make use of the provided SSL certificate(s) for TLS setup.
- Here is an example of the resulting entry in the `config.yaml` file:

```
LDAP_URI: ldaps://ldap.example.org
```


11.2.2. Team Synchronization

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

- If enabled, organization administrators who are also superusers can set teams to have their membership synchronized with a backing group in LDAP.

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

Resynchronization duration:
 The duration before a team must be re-synchronized. Must be expressed in a duration string form: `30m`, `1h`, `1d`.

Self-service team syncing setup:  If enabled, this feature will allow *any organization administrator* to read the membership of any LDAP group.

Allow non-superusers to enable and manage team syncing
 If enabled, non-superusers will be able to enable and manage team syncing on teams under organizations in which they are administrators.

- The resynchronization duration is the period at which a team must be re-synchronized. Must be expressed in a duration string form: `30m`, `1h`, `1d`.
- Optionally allow non-superusers to enable and manage team syncing under organizations in which they are administrators.
- Here is an example of the resulting entries in the `config.yaml` file:

```
FEATURE_TEAM_SYNCING: true
TEAM_RESYNC_STALE_TIME: 60m
FEATURE_NONSUPERUSER_TEAM_SYNCING_SETUP: true
```

11.2.3. Base and Relative Distinguished Names

Base DN:

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

User Relative DN:

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

Secondary User Relative DNs: [Remove](#)

A list of Distinguished Name path(s) which forms the secondary base path(s) for looking up all user LDAP records, relative to the Base DN defined above. These path(s) will be tried if the user is not found via the primary relative DN.
Example: [ou=employees]

- A Distinguished Name path which forms the base path for looking up all LDAP records. Example: *dc=my,dc=domain,dc=com*
- Optional list of Distinguished Name path(s) which form the secondary base path(s) for looking up all user LDAP records, relative to the Base DN defined above. These path(s) will be tried if the user is not found via the primary relative DN.
- User Relative DN is relative to BaseDN. Example: *ou=NYC* not *ou=NYC,dc=example,dc=org*
- Multiple "Secondary User Relative DNs" may be entered if there are multiple Organizational Units where User objects are located at. Simply type in the Organizational Units and click on Add button to add multiple RDNs. Example: *ou=Users,ou=NYC* and *ou=Users,ou=SFO*
- The "User Relative DN" searches with subtree scope. For example, if your Organization has Organizational Units NYC and SFO under the Users OU (*ou=SFO,ou=Users* and *ou=NYC,ou=Users*), Red Hat Quay can authenticate users from both the NYC and SFO Organizational Units if the User Relative DN is set to Users (*ou=Users*).
- Here is an example of the resulting entries in the *config.yaml* file:

```
LDAP_BASE_DN:
- dc=example
- dc=com
LDAP_USER_RDN:
- ou=users
LDAP_SECONDARY_USER_RDNS:
- ou=bots
- ou=external
```

11.2.4. Additional User Filters

Additional User Filter Expression:

NOTE: This query is added **unescaped** to user lookups, so be VERY careful with the query you specify.

If specified, the additional filter used for all user lookup queries. Note that all Distinguished Names used in the filter must be **full paths**; the **base_dn** is not added automatically here. **Must** be wrapped in parens.

Example: (someOtherField=someOtherValue)

Example: (memberOf=some.full.path.to.a.group)

Example: ((someFirstField=someValue)(someOtherField=someOtherValue))

Example: (&(someFirstField=someValue)(someOtherField=someOtherValue))

- If specified, the additional filter used for all user lookup queries. Note that all Distinguished Names used in the filter must be **full paths**; the Base DN is not added automatically here. **Must** be wrapped in parens. Example: (&(someFirstField=someValue))

(someOtherField=someOtherValue))

- Here is an example of the resulting entry in the *config.yaml* file:


```
LDAP_USER_FILTER: (memberof=cn=developers,ou=groups,dc=example,dc=com)
```

11.2.5. Administrator DN

Administrator DN:

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

Administrator DN Password:

 Note: This will be stored in **plaintext** inside the config.yaml, so setting up a dedicated account or using [a password hash](#) is **highly** recommended.

The password for the Administrator DN.

- The Distinguished Name and password for the administrator account. This account must be able to login and view the records for all user accounts. Example:
uid=admin,ou=employees,dc=my,dc=domain,dc=com
- The password will be stored in **plaintext** inside the config.yaml, so setting up a dedicated account or using a password hash is highly recommended.
- Here is an example of the resulting entries in the *config.yaml* file:

```
LDAP_ADMIN_DN: cn=admin,dc=example,dc=com
LDAP_ADMIN_PASSWD: changeme
```

11.2.6. UID and Mail attributes

UID Attribute:

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

Mail Attribute:

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


- The UID attribute is the name of the property field in LDAP user record to use as the **username**. Typically "uid".
- The Mail attribute is the name of the property field in LDAP user record that stores user e-mail address(es). Typically "mail".
- Either of these may be used during login.
- The logged in username must exist in User Relative DN.
- *sAMAccountName* is the UID attribute for against Microsoft Active Directory setups.
- Here is an example of the resulting entries in the *config.yaml* file:

```
LDAP_UID_ATTR: uid
LDAP_EMAIL_ATTR: mail
```

11.2.7. Validation

Once the configuration is completed, click on "Save Configuration Changes" button to validate the configuration.

Validating configuration



✓ CONFIGURATION VALIDATED

✓ Configuration Validated

Continue Editing

Download

All validation must succeed before proceeding, or additional configuration may be performed by selecting the "Continue Editing" button.

11.3. COMMON ISSUES

Invalid credentials

Administrator DN or Administrator DN Password values are incorrect

Verification of superuser %USERNAME% failed: Username not found The user either does not exist in the remote authentication system OR LDAP auth is misconfigured.

Red Hat Quay can connect to the LDAP server via Username/Password specified in the Administrator DN fields however cannot find the current logged in user with the UID Attribute or Mail Attribute fields in the User Relative DN Path. Either current logged in user does not exist in User Relative DN Path, or Administrator DN user do not have rights to search/read this LDAP path.

11.4. CONFIGURE AN LDAP USER AS SUPERUSER

Once LDAP is configured, you can log in to your Red Hat Quay instance with a valid LDAP username and password. You are prompted to confirm your Red Hat Quay username as shown in the following figure:

Confirm Username

The username `testadmin` was automatically generated to conform to the Docker CLI guidelines for use as a namespace in .

Please confirm the selected username or enter a different username below:

testadmin

Confirm Username

✓ Username valid

To attach superuser privilege to an LDAP user, modify the `config.yaml` file with the username. For example:

SUPER_USERS:

- testadmin

Restart the Red Hat **Quay** container with the updated config.yaml file. The next time you log in, the user will have superuser privileges.

CHAPTER 12. PROMETHEUS AND GRAFANA METRICS UNDER RED HAT QUAY

Red Hat Quay exports a [Prometheus](#)- and Grafana-compatible endpoint on each instance to allow for easy monitoring and alerting.

12.1. EXPOSING THE PROMETHEUS ENDPOINT

12.1.1. Standalone Red Hat Quay

When using **podman run** to start the **Quay** container, expose the metrics port **9091**:

```
$ sudo podman run -d --rm -p 80:8080 -p 443:8443 -p 9091:9091 \
  --name=quay \
  -v $QUAY/config:/conf/stack:Z \
  -v $QUAY/storage:/datastorage:Z \
  registry.redhat.io/quay/quay-rhel8:v3.7.13
```

The metrics will now be available:

```
$ curl quay.example.com:9091/metrics
```

See [Monitoring Quay with Prometheus and Grafana](#) for details on configuring Prometheus and Grafana to monitor Quay repository counts.

12.1.2. Red Hat Quay Operator

Determine the cluster IP for the **quay-metrics** service:

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

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)
example-registry-clair-app	ClusterIP	172.30.61.161	<none>	80/TCP,8089/TCP
example-registry-clair-postgres	ClusterIP	172.30.122.136	<none>	5432/TCP
example-registry-quay-app	ClusterIP	172.30.72.79	<none>	443/TCP,80/TCP,8081/TCP,55443/TCP
example-registry-quay-config-editor	ClusterIP	172.30.185.61	<none>	80/TCP
example-registry-quay-database	ClusterIP	172.30.114.192	<none>	5432/TCP
example-registry-quay-metrics	ClusterIP	172.30.37.76	<none>	9091/TCP
example-registry-quay-redis	ClusterIP	172.30.157.248	<none>	6379/TCP

Connect to your cluster and access the metrics using the cluster IP and port for the **quay-metrics** service:

```
$ oc debug node/master-0
```

```
sh-4.4# curl 172.30.37.76:9091/metrics

# HELP go_gc_duration_seconds A summary of the pause duration of garbage collection cycles.
# TYPE go_gc_duration_seconds summary
go_gc_duration_seconds{quantile="0"} 4.0447e-05
go_gc_duration_seconds{quantile="0.25"} 6.2203e-05
...
```

12.1.3. Setting up Prometheus to consume metrics

Prometheus needs a way to access all Red Hat Quay instances running in a cluster. In the typical setup, this is done by listing all the Red Hat Quay instances in a single named DNS entry, which is then given to Prometheus.

12.1.4. DNS configuration under Kubernetes

A simple [Kubernetes service](#) can be configured to provide the DNS entry for Prometheus.

12.1.5. DNS configuration for a manual cluster

[SkyDNS](#) is a simple solution for managing this DNS record when not using Kubernetes. SkyDNS can run on an [etcd](#) cluster. Entries for each Red Hat Quay instance in the cluster can be added and removed in the etcd store. SkyDNS will regularly read them from there and update the list of Quay instances in the DNS record accordingly.

12.2. INTRODUCTION TO METRICS

Red Hat Quay provides metrics to help monitor the registry, including metrics for general registry usage, uploads, downloads, garbage collection, and authentication.

12.2.1. General registry statistics

General registry statistics can indicate how large the registry has grown.

Metric name	Description
quay_user_rows	Number of users in the database
quay_robot_rows	Number of robot accounts in the database
quay_org_rows	Number of organizations in the database
quay_repository_rows	Number of repositories in the database
quay_security_scanning_unscanned_images_remaining_total	Number of images that are not scanned by the latest security scanner

Sample metrics output

```
# HELP quay_user_rows number of users in the database
```



```

# TYPE quay_user_rows gauge
quay_user_rows{host="example-registry-quay-app-6df87f7b66-9tfn6",instance="",job="quay",pid="65",process_name="globalpromstats.py"} 3

# HELP quay_robot_rows number of robot accounts in the database
# TYPE quay_robot_rows gauge
quay_robot_rows{host="example-registry-quay-app-6df87f7b66-9tfn6",instance="",job="quay",pid="65",process_name="globalpromstats.py"} 2

# HELP quay_org_rows number of organizations in the database
# TYPE quay_org_rows gauge
quay_org_rows{host="example-registry-quay-app-6df87f7b66-9tfn6",instance="",job="quay",pid="65",process_name="globalpromstats.py"} 2

# HELP quay_repository_rows number of repositories in the database
# TYPE quay_repository_rows gauge
quay_repository_rows{host="example-registry-quay-app-6df87f7b66-9tfn6",instance="",job="quay",pid="65",process_name="globalpromstats.py"} 4

# HELP quay_security_scanning_unscanned_images_remaining number of images that are not scanned by the latest security scanner
# TYPE quay_security_scanning_unscanned_images_remaining gauge
quay_security_scanning_unscanned_images_remaining{host="example-registry-quay-app-6df87f7b66-9tfn6",instance="",job="quay",pid="208",process_name="secscan:application"} 5

```

12.2.2. Queue items

The *queue items* metrics provide information on the multiple queues used by Quay for managing work.

Metric name	Description
quay_queue_items_available	Number of items in a specific queue
quay_queue_items_locked	Number of items that are running
quay_queue_items_available_unlocked	Number of items that are waiting to be processed

Metric labels

- **queue_name:** The name of the queue. One of:
 - **exportactionlogs:** Queued requests to export action logs. These logs are then processed and put in storage. A link is then sent to the requester via email.
 - **namespacegc:** Queued namespaces to be garbage collected
 - **notification:** Queue for repository notifications to be sent out
 - **repositorygc:** Queued repositories to be garbage collected
 - **secscanv4:** Notification queue specific for Clair V4
 - **dockerfilebuild:** Queue for Quay docker builds

- **imagestoragereplication**: Queued blob to be replicated across multiple storages
- **chunk_cleanup**: Queued blob segments that needs to be deleted. This is only used by some storage implementations, for example, Swift.

For example, the queue labelled **repositorygc** contains the repositories marked for deletion by the repository garbage collection worker. For metrics with a **queue_name** label of **repositorygc**:

- **quay_queue_items_locked** is the number of repositories currently being deleted.
- **quay_queue_items_available_unlocked** is the number of repositories waiting to get processed by the worker.

Sample metrics output

```
# HELP quay_queue_items_available number of queue items that have not expired
# TYPE quay_queue_items_available gauge
quay_queue_items_available{host="example-registry-quay-app-6df87f7b66-9tfn6",instance="",job="quay",pid="63",process_name="exportactionlogsworker.py",queue_name="exportactionlogs"} 0
...

# HELP quay_queue_items_available_unlocked number of queue items that have not expired and are not locked
# TYPE quay_queue_items_available_unlocked gauge
quay_queue_items_available_unlocked{host="example-registry-quay-app-6df87f7b66-9tfn6",instance="",job="quay",pid="63",process_name="exportactionlogsworker.py",queue_name="exportactionlogs"} 0
...

# HELP quay_queue_items_locked number of queue items that have been acquired
# TYPE quay_queue_items_locked gauge
quay_queue_items_locked{host="example-registry-quay-app-6df87f7b66-9tfn6",instance="",job="quay",pid="63",process_name="exportactionlogsworker.py",queue_name="exportactionlogs"} 0
```

12.2.3. Garbage collection metrics

These metrics show you how many resources have been removed from garbage collection (gc). They show many times the gc workers have run and how many namespaces, repositories, and blobs were removed.

Metric name	Description
quay_gc_iterations_total	Number of iterations by the GCWorker
quay_gc_namespaces_purged_total	Number of namespaces purged by the NamespaceGCWorker
quay_gc_repos_purged_total	Number of repositories purged by the RepositoryGCWorker or NamespaceGCWorker
quay_gc_storage_blobs_deleted_total	Number of storage blobs deleted

Sample metrics output

```
# TYPE quay_gc_iterations_created gauge
quay_gc_iterations_created{host="example-registry-quay-app-6df87f7b66-
9tfn6",instance="",job="quay",pid="208",process_name="secscan:application"}
1.6317823190189714e+09
...

# HELP quay_gc_iterations_total number of iterations by the GCWorker
# TYPE quay_gc_iterations_total counter
quay_gc_iterations_total{host="example-registry-quay-app-6df87f7b66-
9tfn6",instance="",job="quay",pid="208",process_name="secscan:application"} 0
...

# TYPE quay_gc_namespaces_purged_created gauge
quay_gc_namespaces_purged_created{host="example-registry-quay-app-6df87f7b66-
9tfn6",instance="",job="quay",pid="208",process_name="secscan:application"}
1.6317823190189433e+09
...

# HELP quay_gc_namespaces_purged_total number of namespaces purged by the
NamespaceGCWorker
# TYPE quay_gc_namespaces_purged_total counter
quay_gc_namespaces_purged_total{host="example-registry-quay-app-6df87f7b66-
9tfn6",instance="",job="quay",pid="208",process_name="secscan:application"} 0
....

# TYPE quay_gc_repos_purged_created gauge
quay_gc_repos_purged_created{host="example-registry-quay-app-6df87f7b66-
9tfn6",instance="",job="quay",pid="208",process_name="secscan:application"}
1.631782319018925e+09
...

# HELP quay_gc_repos_purged_total number of repositories purged by the RepositoryGCWorker or
NamespaceGCWorker
# TYPE quay_gc_repos_purged_total counter
quay_gc_repos_purged_total{host="example-registry-quay-app-6df87f7b66-
9tfn6",instance="",job="quay",pid="208",process_name="secscan:application"} 0
...

# TYPE quay_gc_storage_blobs_deleted_created gauge
quay_gc_storage_blobs_deleted_created{host="example-registry-quay-app-6df87f7b66-
9tfn6",instance="",job="quay",pid="208",process_name="secscan:application"}
1.6317823190189059e+09
...

# HELP quay_gc_storage_blobs_deleted_total number of storage blobs deleted
# TYPE quay_gc_storage_blobs_deleted_total counter
quay_gc_storage_blobs_deleted_total{host="example-registry-quay-app-6df87f7b66-
9tfn6",instance="",job="quay",pid="208",process_name="secscan:application"} 0
...
```

12.2.3.1. Multipart uploads metrics

The multipart uploads metrics show the number of blobs uploads to storage (S3, Rados, GoogleCloudStorage, RHOCS). These can help identify issues when Quay is unable to correctly upload blobs to storage.

Metric name	Description
quay_multipart_uploads_started_total	Number of multipart uploads to Quay storage that started
quay_multipart_uploads_completed_total	Number of multipart uploads to Quay storage that completed

Sample metrics output

```
# TYPE quay_multipart_uploads_completed_created gauge
quay_multipart_uploads_completed_created{host="example-registry-quay-app-6df87f7b66-9tfn6",instance="",job="quay",pid="208",process_name="secscan:application"}
1.6317823308284895e+09
...

# HELP quay_multipart_uploads_completed_total number of multipart uploads to Quay storage that completed
# TYPE quay_multipart_uploads_completed_total counter
quay_multipart_uploads_completed_total{host="example-registry-quay-app-6df87f7b66-9tfn6",instance="",job="quay",pid="208",process_name="secscan:application"} 0

# TYPE quay_multipart_uploads_started_created gauge
quay_multipart_uploads_started_created{host="example-registry-quay-app-6df87f7b66-9tfn6",instance="",job="quay",pid="208",process_name="secscan:application"}
1.6317823308284352e+09
...

# HELP quay_multipart_uploads_started_total number of multipart uploads to Quay storage that started
# TYPE quay_multipart_uploads_started_total counter
quay_multipart_uploads_started_total{host="example-registry-quay-app-6df87f7b66-9tfn6",instance="",job="quay",pid="208",process_name="secscan:application"} 0
...
```

12.2.4. Image push / pull metrics

A number of metrics are available related to pushing and pulling images.

12.2.4.1. Image pulls total

Metric name	Description
quay_registry_image_pulls_total	The number of images downloaded from the registry.

Metric labels

- **protocol:** the registry protocol used (should always be v2)
- **ref:** ref used to pull - tag, manifest
- **status:** http return code of the request

12.2.4.2. Image bytes pulled

Metric name	Description
quay_registry_image_pulled_estimated_bytes_total	The number of bytes downloaded from the registry

Metric labels

- **protocol:** the registry protocol used (should always be v2)

12.2.4.3. Image pushes total

Metric name	Description
quay_registry_image_pushes_total	The number of images uploaded from the registry.

Metric labels

- **protocol:** the registry protocol used (should always be v2)
- **pstatus:** http return code of the request
- **pmedia_type:** the uploaded manifest type

12.2.4.4. Image bytes pushed

Metric name	Description
quay_registry_image_pushed_bytes_total	The number of bytes uploaded to the registry

Sample metrics output

```
# HELP quay_registry_image_pushed_bytes_total number of bytes pushed to the registry
# TYPE quay_registry_image_pushed_bytes_total counter
quay_registry_image_pushed_bytes_total{host="example-registry-quay-app-6df87f7b66-9tfn6",instance="",job="quay",pid="221",process_name="registry:application"} 0
...
```

12.2.5. Authentication metrics

The authentication metrics provide the number of authentication requests, labeled by type and whether it succeeded or not. For example, this metric could be used to monitor failed basic authentication requests.

Metric name	Description
quay_authentication_attempts_total	Number of authentication attempts across the registry and API

Metric labels

- **auth_kind:** The type of auth used, including:
 - basic
 - oauth
 - credentials
- **success:** true or false

Sample metrics output

```
# TYPE quay_authentication_attempts_created gauge
quay_authentication_attempts_created{auth_kind="basic",host="example-registry-quay-app-6df87f7b66-9tfn6",instance="",job="quay",pid="221",process_name="registry:application",success="True"}
1.6317843039374158e+09
...

# HELP quay_authentication_attempts_total number of authentication attempts across the registry
and API
# TYPE quay_authentication_attempts_total counter
quay_authentication_attempts_total{auth_kind="basic",host="example-registry-quay-app-6df87f7b66-9tfn6",instance="",job="quay",pid="221",process_name="registry:application",success="True"} 2
...
```

CHAPTER 13. RED HAT QUAY QUOTA MANAGEMENT AND ENFORCEMENT

With Red Hat Quay 3.7, users have the ability to report storage consumption and to contain registry growth by establishing configured storage quota limits. On-premise Quay users are now equipped with the following capabilities to manage the capacity limits of their environment:

- **Quota reporting:** With this feature, a superuser can track the storage consumption of all their organizations. Additionally, users can track the storage consumption of their assigned organization.
- **Quota management:** With this feature, a superuser can define soft and hard checks for Red Hat Quay users. Soft checks tell users if the storage consumption of an organization reaches their configured threshold. Hard checks prevent users from pushing to the registry when storage consumption reaches the configured limit.

Together, these features allow service owners of a Quay registry to define service level agreements and support a healthy resource budget.

13.1. QUOTA MANAGEMENT CONFIGURATION

Quota management is now supported under the **FEATURE_QUOTA_MANAGEMENT** property and is turned off by default. To enable quota management, set the feature flag in your **config.yaml** to **true**:

```
FEATURE_QUOTA_MANAGEMENT: true
```



NOTE

In Red Hat Quay 3.7, superuser privileges are required to create, update and delete quotas. While quotas can be set for users as well as organizations, you cannot reconfigure the *user* quota using the Red Hat Quay UI and you must use the API instead.

13.1.1. Default quota

To specify a system-wide default storage quota that is applied to every organization and user, use the **DEFAULT_SYSTEM_REJECT_QUOTA_BYTES** configuration flag.

Table 13.1. Default quota configuration

Field	Type	Description
DEFAULT_SYSTEM_REJECT_QUOTA_BYTES	String	The quota size to apply to all organizations and users. By default, no limit is set.

If you configure a specific quota for an organization or user, and then delete that quota, the system-wide default quota will apply if one has been set. Similarly, if you have configured a specific quota for an organization or user, and then modify the system-wide default quota, the updated system-wide default will override any specific settings.

13.2. QUOTA MANAGEMENT ARCHITECTURE

The **RepositorySize** database table holds the storage consumption, in bytes, of a Red Hat Quay repository within an organization. The sum of all repository sizes for an organization defines the current storage size of a Red Hat Quay organization. When an image push is initialized, the user's organization storage is validated to check if it is beyond the configured quota limits. If an image push exceeds defined quota limitations, a soft or hard check occurs:

- For a soft check, users are notified.
- For a hard check, the push is stopped.

If storage consumption is within configured quota limits, the push is allowed to proceed.

Image manifest deletion follows a similar flow, whereby the links between associated image tags and the manifest are deleted. Additionally, after the image manifest is deleted, the repository size is recalculated and updated in the **RepositorySize** table.

13.3. ESTABLISHING QUOTA IN RED HAT QUAY UI

The following procedure describes how you can report storage consumption and establish storage quota limits.

Prerequisites

- A Red Hat Quay registry.
- A superuser account.
- Enough storage to meet the demands of quota limitations.

Procedure

1. Create a new organization or choose an existing one. Initially, no quota is configured, as can be seen on the **Organization Settings** tab:

Organization Settings

Namespace: testorg
Organization names cannot be changed once set.

Avatar: Avatar is generated based off the organization's name.

Delete organization: [Begin deletion >](#)

Time Machine: 14 days
The amount of time, after a tag is deleted, that the tag is accessible in time machine before being garbage collected.
[Save Expiration Time](#)

Quota Management: No Quota Configured

Proxy Cache:

Remote Registry:
Remote registry that is to be cached. (Eg: For docker hub, docker.io, docker.io/library)

Remote Registry Username:
Username for authenticating into the entered remote registry. For anonymous pulls from the upstream, leave this empty.

Remote Registry Password:
Password for authenticating into the entered remote registry. For anonymous pulls from the

- Log in to the registry as a superuser and navigate to the **Manage Organizations** tab on the **Super User Admin Panel**. Click the **Options** icon of the organization for which you want to create storage quota limits:

Red Hat Quay Management

Organizations

1 - 1 of 1 [Filter Organizations...](#)

NAME ↓	QUOTA CONSUMED
testorg	

- Rename Organization
- ✕ Delete Organization
- ⚡ Take Ownership
- ⚙️ Configure Quota

- Click **Configure Quota** and enter the initial quota, for example, **10 MB**. Then click **Apply** and **Close**:

Manage Organization Quota for testorg ✕

Set storage quota:

KB
✓ MB
GB
TB

Note: No quota policy defined. Users will be able to exceed the storage quota set above.

[Apply](#) [Close](#)

- Check that the quota consumed shows **0 of 10 MB** on the **Manage Organizations** tab of the superuser panel:

Organizations

1 - 1 of 1 < > Filter Organizations...

NAME ↓	QUOTA CONSUMED
testorg	0 of 10 MB

The consumed quota information is also available directly on the Organization page:

Initial consumed quota

T testorg + Create New Repository

Repositories
Total Quota Consumed: (0%) 0 of 10 MB

0 - 0 of 0 < > Filter Repositories...

This namespace doesn't have any viewable repositories.
Either no repositories exist yet or you may not have permission to view any. If you have permission, try [creating a new repository](#).

- To increase the quota to 100MB, navigate to the **Manage Organizations** tab on the superuser panel. Click the **Options** icon and select **Configure Quota**, setting the quota to 100 MB. Click **Apply** and then **Close**:

Manage Organization Quota for testorg ×

Set storage quota:

Quota Policy: **Action** **Quota Threshold** + Add Limit

Note: No quota policy defined. Users will be able to exceed the storage quota set above.

Apply Remove Close

- Push a sample image to the organization from the command line:

Sample commands

```
$ podman pull ubuntu:18.04
```

```
$ podman tag docker.io/library/ubuntu:18.04 example-registry-quay-quay-
enterprise.apps.docs.gcp.quaydev.org/testorg/ubuntu:18.04
```

```
$ podman push --tls-verify=false example-registry-quay-quay-
enterprise.apps.docs.gcp.quaydev.org/testorg/ubuntu:18.04
```

7. On the superuser panel, the quota consumed per organization is displayed:

The screenshot shows the 'Organizations' page. At the top, there is a header with the organization name 'testorg' and a '+ Create New Repository' button. Below this is a 'Repositories' section with a sub-header 'Total Quota Consumed: 27 MB (27%) of 100 MB'. A table lists the repositories with columns: REPOSITORY NAME, LAST MODIFIED, STATE, QUOTA CONSUMED, ACTIVITY, and STAR. One repository is listed: 'testorg / ubuntu' with a last modified time of 'Today at 2:12 PM', state 'Normal', and quota consumed '27 MB (27%)'. The activity column shows a green bar chart, and the star column has a star icon.

8. The Organization page shows the total proportion of the quota used by the image:

Total Quota Consumed for first image

The screenshot shows the 'Repositories' section of the organization page. It displays a table with the following data:

REPOSITORY NAME	LAST MODIFIED	STATE	QUOTA CONSUMED	ACTIVITY	STAR
testorg / ubuntu	Today at 2:12 PM	Normal	27 MB (27%)		☆

9. Pull, tag, and push a second image, for example, **nginx**:

Sample commands

```
$ podman pull nginx
```

```
$ podman tag docker.io/library/nginx example-registry-quay-quay-  
enterprise.apps.docs.gcp.quaydev.org/testorg/nginx
```

```
$ podman push --tls-verify=false example-registry-quay-quay-  
enterprise.apps.docs.gcp.quaydev.org/testorg/nginx
```

10. The Organization page shows the total proportion of the quota used by each repository in that organization:

Total Quota Consumed for each repository

T testorg [+ Create New Repository](#)

Repositories

Total Quota Consumed: 83 MB (83%) of 100 MB

1 - 2 of 2 < >

REPOSITORY NAME	LAST MODIFIED	STATE	QUOTA CONSUMED	ACTIVITY ↓	STAR
T testorg / ubuntu	Today at 2:12 PM	Normal	27 MB (27%)		☆
T testorg / nginx	Today at 2:31 PM	Normal	56 MB (56%)		☆

11. Create *reject* and *warning* limits:

From the superuser panel, navigate to the **Manage Organizations** tab. Click the **Options** icon for the organization and select **Configure Quota**. In the **Quota Policy** section, with the **Action** type set to **Reject**, set the **Quota Threshold** to **80** and click **Add Limit**:

Manage Organization Quota for testorg ×

Set storage quota:

Quota Policy:

Action	Quota Threshold	
Reject ↓	<input type="text" value="80"/>	+ Add Limit

Note: No quota policy defined. Users will be able to exceed the storage quota set above.

12. To create a *warning* limit, select **Warning** as the **Action** type, set the **Quota Threshold** to **70** and click **Add Limit**:

Manage Organization Quota for testorg ×

Set storage quota:

Quota Policy:

Action	Quota Threshold	
Reject ↓	<input type="text" value="80"/>	<input type="button" value="Update"/> <input type="button" value="Remove"/>
Warning ↓	<input type="text" value="70"/>	+ Add Limit

13. Click **Close** on the quota popup. The limits are viewable, but not editable, on the **Settings** tab of the **Organization** page:

The screenshot shows the 'Organization Settings' page for an organization named 'testorg'. The page includes a sidebar with navigation icons and a main content area with the following sections:

- Namespace:** testorg. Note: Organization names cannot be changed once set.
- Avatar:** A blue square with a white 'T'. Note: Avatar is generated based off the organization's name.
- Delete organization:** [Begin deletion >](#)
- Time Machine:** A dropdown menu set to '14 days'. Note: The amount of time, after a tag is deleted, that the tag is accessible in time machine before being garbage collected. A [Save Expiration Time](#) button is below.
- Quota Management:**
 - Set storage quota: 100 MB
 - Quota Policy table:

Action	Quota Threshold
Reject	80
Warning	70

14. Push an image where the reject limit is exceeded:

Because the reject limit (80%) has been set to below the current repository size (~83%), the next push is rejected automatically.

Sample image push

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

```
$ podman tag docker.io/library/ubuntu:20.04 example-registry-quay-quay-
enterprise.apps.docs.gcp.quaydev.org/testorg/ubuntu:20.04
```

```
$ podman push --tls-verify=false example-registry-quay-quay-
enterprise.apps.docs.gcp.quaydev.org/testorg/ubuntu:20.04
```

Sample output when quota exceeded

```
Getting image source signatures
Copying blob d4dfaa212623 [-----] 8.0b / 3.5KiB
Copying blob cba97cc5811c [-----] 8.0b / 15.0KiB
Copying blob 0c78fac124da [-----] 8.0b / 71.8MiB
WARN[0002] failed, retrying in 1s ... (1/3). Error: Error writing blob: Error initiating layer
upload to /v2/testorg/ubuntu/blobs/uploads/ in example-registry-quay-quay-
enterprise.apps.docs.gcp.quaydev.org: denied: Quota has been exceeded on namespace
Getting image source signatures
Copying blob d4dfaa212623 [-----] 8.0b / 3.5KiB
Copying blob cba97cc5811c [-----] 8.0b / 15.0KiB
Copying blob 0c78fac124da [-----] 8.0b / 71.8MiB
WARN[0005] failed, retrying in 1s ... (2/3). Error: Error writing blob: Error initiating layer
upload to /v2/testorg/ubuntu/blobs/uploads/ in example-registry-quay-quay-
enterprise.apps.docs.gcp.quaydev.org: denied: Quota has been exceeded on namespace
Getting image source signatures
Copying blob d4dfaa212623 [-----] 8.0b / 3.5KiB
Copying blob cba97cc5811c [-----] 8.0b / 15.0KiB
```

```

Copying blob 0c78fac124da [-----] 8.0b / 71.8MiB
WARN[0009] failed, retrying in 1s ... (3/3). Error: Error writing blob: Error initiating layer
upload to /v2/testorg/ubuntu/blobs/uploads/ in example-registry-quay-quay-
enterprise.apps.docs.gcp.quaydev.org: denied: Quota has been exceeded on namespace
Getting image source signatures
Copying blob d4dfaa212623 [-----] 8.0b / 3.5KiB
Copying blob cba97cc5811c [-----] 8.0b / 15.0KiB
Copying blob 0c78fac124da [-----] 8.0b / 71.8MiB
Error: Error writing blob: Error initiating layer upload to /v2/testorg/ubuntu/blobs/uploads/ in
example-registry-quay-quay-enterprise.apps.docs.gcp.quaydev.org: denied: Quota has been
exceeded on namespace

```

- When limits are exceeded, notifications are displayed in the UI:

Quota notifications

The screenshot shows the Red Hat Quay interface for the organization 'testorg'. The main content area displays 'Organization Settings' with fields for Namespace (testorg), Avatar (T), Delete organization (Begin deletion), Time Machine (14 days), and Quota Management (Set storage quota: 100 MB). The Quota Policy section shows two rows: one with Action 'Reject' and Quota Threshold '80', and another with Action 'Warning' and Quota Threshold '70'. On the right, a 'Notifications' panel shows three notifications: 'testorg quota has been exceeded', each with a 'Dismiss Notification' link and a timestamp of 'May 5, 2022 4:01:12 PM'.

13.4. ESTABLISHING QUOTA WITH THE RED HAT QUAY API

When an organization is first created, it does not have a quota applied. Use the `/api/v1/organization/{organization}/quota` endpoint:

Sample command

```

$ curl -k -X GET -H "Authorization: Bearer <token>" -H 'Content-Type: application/json'
https://example-registry-quay-quay-
enterprise.apps.docs.gcp.quaydev.org/api/v1/organization/testorg/quota | jq

```

Sample output

```

[]

```

13.4.1. Setting the quota

To set a quota for an organization, POST data to the `/api/v1/organization/{orgname}/quota` endpoint: .Sample command

```
$ curl -k -X POST -H "Authorization: Bearer <token>" -H 'Content-Type: application/json' -d
 '{"limit_bytes": 10485760}' https://example-registry-quay-quay-
 enterprise.apps.docs.quayteam.org/api/v1/organization/testorg/quota | jq
```

Sample output

```
"Created"
```

13.4.2. Viewing the quota

To see the applied quota, **GET** data from the `/api/v1/organization/{orgname}/quota` endpoint:

Sample command

```
$ curl -k -X GET -H "Authorization: Bearer <token>" -H 'Content-Type: application/json'
 https://example-registry-quay-quay-
 enterprise.apps.docs.gcp.quaydev.org/api/v1/organization/testorg/quota | jq
```

Sample output

```
[
  {
    "id": 1,
    "limit_bytes": 10485760,
    "default_config": false,
    "limits": [],
    "default_config_exists": false
  }
]
```

13.4.3. Modifying the quota

To change the existing quota, in this instance from 10 MB to 100 MB, PUT data to the `/api/v1/organization/{orgname}/quota/{quota_id}` endpoint:

Sample command

```
$ curl -k -X PUT -H "Authorization: Bearer <token>" -H 'Content-Type: application/json' -d
 '{"limit_bytes": 104857600}' https://example-registry-quay-quay-
 enterprise.apps.docs.gcp.quaydev.org/api/v1/organization/testorg/quota/1 | jq
```

Sample output

```
{
  "id": 1,
  "limit_bytes": 104857600,
  "default_config": false,
```

```
"limits": [],  
"default_config_exists": false  
}
```

13.4.4. Pushing images

To see the storage consumed, push various images to the organization.

13.4.4.1. Pushing ubuntu:18.04

Push ubuntu:18.04 to the organization from the command line:

Sample commands

```
$ podman pull ubuntu:18.04  
  
$ podman tag docker.io/library/ubuntu:18.04 example-registry-quay-quay-  
enterprise.apps.docs.gcp.quaydev.org/testorg/ubuntu:18.04  
  
$ podman push --tls-verify=false example-registry-quay-quay-  
enterprise.apps.docs.gcp.quaydev.org/testorg/ubuntu:18.04
```

13.4.4.2. Using the API to view quota usage

To view the storage consumed, **GET** data from the `/api/v1/repository` endpoint:

Sample command

```
$ curl -k -X GET -H "Authorization: Bearer <token>" -H 'Content-Type: application/json'  
'https://example-registry-quay-quay-enterprise.apps.docs.gcp.quaydev.org/api/v1/repository?  
last_modified=true&namespace=testorg&popularity=true&public=true&quota=true' | jq
```

Sample output

```
{  
  "repositories": [  
    {  
      "namespace": "testorg",  
      "name": "ubuntu",  
      "description": null,  
      "is_public": false,  
      "kind": "image",  
      "state": "NORMAL",  
      "quota_report": {  
        "quota_bytes": 27959066,  
        "configured_quota": 104857600  
      },  
      "last_modified": 1651225630,  
      "popularity": 0,  
      "is_starred": false  
    }  
  ]  
}
```


13.4.4.3. Pushing another image

1. Pull, tag, and push a second image, for example, **nginx**:

Sample commands

```
$ podman pull nginx

$ podman tag docker.io/library/nginx example-registry-quay-quay-
enterprise.apps.docs.gcp.quaydev.org/testorg/nginx

$ podman push --tls-verify=false example-registry-quay-quay-
enterprise.apps.docs.gcp.quaydev.org/testorg/nginx
```

2. To view the quota report for the repositories in the organization, use the `/api/v1/repository` endpoint:

Sample command

```
$ curl -k -X GET -H "Authorization: Bearer <token>" -H 'Content-Type: application/json'
'https://example-registry-quay-quay-enterprise.apps.docs.gcp.quaydev.org/api/v1/repository?
last_modified=true&namespace=testorg&popularity=true&public=true&quota=true'
```

Sample output

```
{
  "repositories": [
    {
      "namespace": "testorg",
      "name": "ubuntu",
      "description": null,
      "is_public": false,
      "kind": "image",
      "state": "NORMAL",
      "quota_report": {
        "quota_bytes": 27959066,
        "configured_quota": 104857600
      },
      "last_modified": 1651225630,
      "popularity": 0,
      "is_starred": false
    },
    {
      "namespace": "testorg",
      "name": "nginx",
      "description": null,
      "is_public": false,
      "kind": "image",
      "state": "NORMAL",
      "quota_report": {
        "quota_bytes": 59231659,
        "configured_quota": 104857600
      },
      "last_modified": 1651229507,
      "popularity": 0,
    }
  ]
}
```

```

    "is_starred": false
  }
]
}

```

- To view the quota information in the organization details, use the `/api/v1/organization/{orgname}` endpoint:

Sample command

```

$ curl -k -X GET -H "Authorization: Bearer <token>" -H 'Content-Type: application/json'
https://example-registry-quay-quay-
enterprise.apps.docs.gcp.quaydev.org/api/v1/organization/testorg' | jq

```

Sample output

```

{
  "name": "testorg",
  ...
  "quotas": [
    {
      "id": 1,
      "limit_bytes": 104857600,
      "limits": []
    }
  ],
  "quota_report": {
    "quota_bytes": 87190725,
    "configured_quota": 104857600
  }
}

```

13.4.5. Rejecting pushes using quota limits

If an image push exceeds defined quota limitations, a soft or hard check occurs:

- For a soft check, or *warning*, users are notified.
- For a hard check, or *reject*, the push is terminated.

13.4.5.1. Setting reject and warning limits

To set *reject* and *warning* limits, POST data to the `/api/v1/organization/{orgname}/quota/{quota_id}/limit` endpoint:

Sample reject limit command

```

$ curl -k -X POST -H "Authorization: Bearer <token>" -H 'Content-Type: application/json' -d
 '{"type":"Reject","threshold_percent":80}' https://example-registry-quay-quay-
enterprise.apps.docs.gcp.quaydev.org/api/v1/organization/testorg/quota/1/limit

```

Sample warning limit command

```
$ curl -k -X POST -H "Authorization: Bearer <token>" -H 'Content-Type: application/json' -d
 '{"type": "Warning", "threshold_percent": 50}' https://example-registry-quay-quay-
 enterprise.apps.docs.gcp.quaydev.org/api/v1/organization/testorg/quota/1/limit
```

13.4.5.2. Viewing reject and warning limits

To view the *reject* and *warning* limits, use the `/api/v1/organization/{orgname}/quota` endpoint:

View quota limits

```
$ curl -k -X GET -H "Authorization: Bearer <token>" -H 'Content-Type: application/json'
 https://example-registry-quay-quay-
 enterprise.apps.docs.gcp.quaydev.org/api/v1/organization/testorg/quota | jq
```

Sample output for quota limits

```
[
  {
    "id": 1,
    "limit_bytes": 104857600,
    "default_config": false,
    "limits": [
      {
        "id": 2,
        "type": "Warning",
        "limit_percent": 50
      },
      {
        "id": 1,
        "type": "Reject",
        "limit_percent": 80
      }
    ],
    "default_config_exists": false
  }
]
```

13.4.5.3. Pushing an image when the reject limit is exceeded

In this example, the reject limit (80%) has been set to below the current repository size (~83%), so the next push should automatically be rejected.

Push a sample image to the organization from the command line:

Sample image push

```
$ podman pull ubuntu:20.04

$ podman tag docker.io/library/ubuntu:20.04 example-registry-quay-quay-
 enterprise.apps.docs.gcp.quaydev.org/testorg/ubuntu:20.04

$ podman push --tls-verify=false example-registry-quay-quay-
 enterprise.apps.docs.gcp.quaydev.org/testorg/ubuntu:20.04
```

Sample output when quota exceeded

```

Getting image source signatures
Copying blob d4dfaa212623 [-----] 8.0b / 3.5KiB
Copying blob cba97cc5811c [-----] 8.0b / 15.0KiB
Copying blob 0c78fac124da [-----] 8.0b / 71.8MiB
WARN[0002] failed, retrying in 1s ... (1/3). Error: Error writing blob: Error initiating layer upload to
/v2/testorg/ubuntu/blobs/uploads/ in example-registry-quay-quay-
enterprise.apps.docs.gcp.quaydev.org: denied: Quota has been exceeded on namespace
Getting image source signatures
Copying blob d4dfaa212623 [-----] 8.0b / 3.5KiB
Copying blob cba97cc5811c [-----] 8.0b / 15.0KiB
Copying blob 0c78fac124da [-----] 8.0b / 71.8MiB
WARN[0005] failed, retrying in 1s ... (2/3). Error: Error writing blob: Error initiating layer upload to
/v2/testorg/ubuntu/blobs/uploads/ in example-registry-quay-quay-
enterprise.apps.docs.gcp.quaydev.org: denied: Quota has been exceeded on namespace
Getting image source signatures
Copying blob d4dfaa212623 [-----] 8.0b / 3.5KiB
Copying blob cba97cc5811c [-----] 8.0b / 15.0KiB
Copying blob 0c78fac124da [-----] 8.0b / 71.8MiB
WARN[0009] failed, retrying in 1s ... (3/3). Error: Error writing blob: Error initiating layer upload to
/v2/testorg/ubuntu/blobs/uploads/ in example-registry-quay-quay-
enterprise.apps.docs.gcp.quaydev.org: denied: Quota has been exceeded on namespace
Getting image source signatures
Copying blob d4dfaa212623 [-----] 8.0b / 3.5KiB
Copying blob cba97cc5811c [-----] 8.0b / 15.0KiB
Copying blob 0c78fac124da [-----] 8.0b / 71.8MiB
Error: Error writing blob: Error initiating layer upload to /v2/testorg/ubuntu/blobs/uploads/ in example-
registry-quay-quay-enterprise.apps.docs.gcp.quaydev.org: denied: Quota has been exceeded on
namespace

```

13.4.5.4. Notifications for limits exceeded

When limits are exceeded, a notification appears:

Quota notifications

The screenshot shows the Quay web interface for an organization named 'testorg'. The main content area is titled 'Organization Settings' and includes the following sections:

- Namespace:** testorg (Note: Organization names cannot be changed once set.)
- Avatar:** A large 'T' icon (Note: Avatar is generated based off the organization's name.)
- Delete organization:** A button labeled 'Begin deletion >'.
- Time Machine:** A dropdown menu set to '14 days' with a 'Save Expiration Time' button below it. A note states: 'The amount of time, after a tag is deleted, that the tag is accessible in time machine before being garbage collected.'
- Quota Management:**
 - Set storage quota:** A numeric input field with '100' and a unit dropdown menu with 'MB' selected.
 - Quota Policy:** A table with two rows:

Action	Quota Threshold
Reject	80
Warning	70

On the right side, a 'Notifications' panel is open, showing three identical notifications: 'testorg quota has been exceeded' with a 'Dismiss Notification' link and a timestamp of 'May 5, 2022 4:01:12 PM'.

13.5. QUOTA MANAGEMENT LIMITATIONS

Quota management helps organizations to maintain resource consumption. One limitation of quota management is that calculating resource consumption on push results in the calculation becoming part of the push's critical path. Without this, usage data might drift.

The maximum storage quota size is dependent on the selected database:

Table 13.2. Worker count environment variables

Variable	Description
Postgres	8388608 TB
MySQL	8388608 TB
SQL Server	16777216 TB

CHAPTER 14. GEO-REPLICATION

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

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

14.1. GEO-REPLICATION FEATURES

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

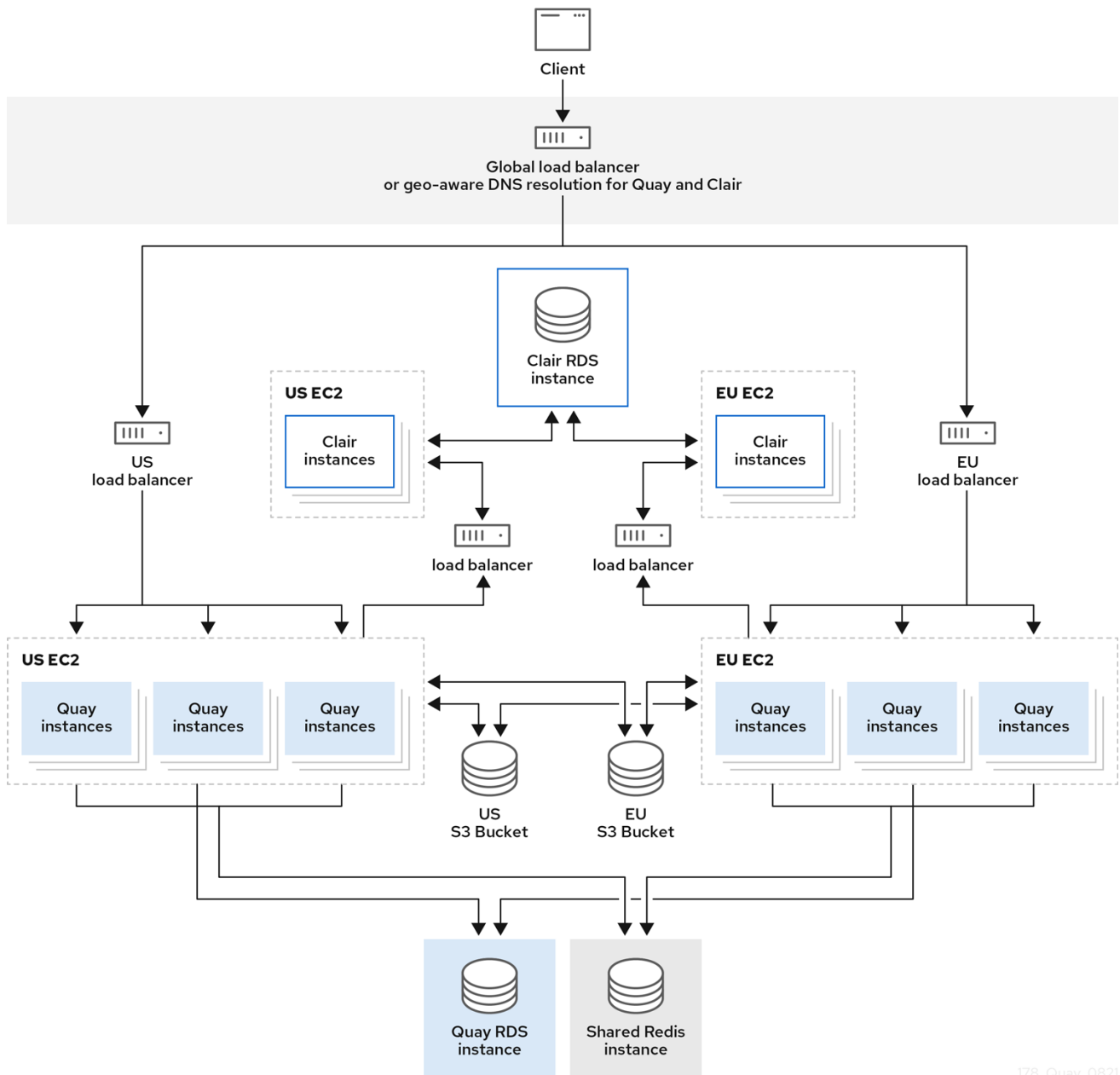
14.2. GEO-REPLICATION REQUIREMENTS AND CONSTRAINTS

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

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

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

14.3. GEO-REPLICATION USING STANDALONE RED HAT QUAY



178_Quay_0821

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



NOTE

In the event that Clair fails in one cluster, for example, the US cluster, US users would not see vulnerability reports in Quay for the second cluster (EU). This is because all Clair instances have the same state. When Clair fails, it is usually because of a problem within the cluster.

14.3.1. Enable storage replication - standalone Quay

1. Scroll down to the section entitled **Registry Storage**.
2. Click **Enable Storage Replication**.

3. Add each of the storage engines to which data will be replicated. All storage engines to be used must be listed.
4. If complete replication of all images to all storage engines is required, under each storage engine configuration click **Replicate to storage engine by default**. This will ensure that all images are replicated to that storage engine. To instead enable per-namespace replication, please contact support.
5. When you are done, click **Save Configuration Changes**. Configuration changes will take effect the next time Red Hat Quay restarts.
6. After adding storage and enabling “Replicate to storage engine by default” for Georeplications, you need to sync existing image data across all storage. To do this, you need to **oc exec** (or **docker/kubect exec**) into the container and run:

```
# scl enable python27 bash
# python -m util.backfillreplication
```

This is a one time operation to sync content after adding new storage.

14.3.2. Run Red Hat Quay with storage preferences

1. Copy the config.yaml to all machines running Red Hat Quay
2. For each machine in each region, add a **QUAY_DISTRIBUTED_STORAGE_PREFERENCE** environment variable with the preferred storage engine for the region in which the machine is running.

For example, for a machine running in Europe with the config directory on the host available from **\$QUAY/config**:

```
$ sudo podman run -d --rm -p 80:8080 -p 443:8443 \
  --name=quay \
  -v $QUAY/config:/conf/stack:Z \
  -e QUAY_DISTRIBUTED_STORAGE_PREFERENCE=europestorage \
  registry.redhat.io/quay/quay-rhel8:v3.7.13
```

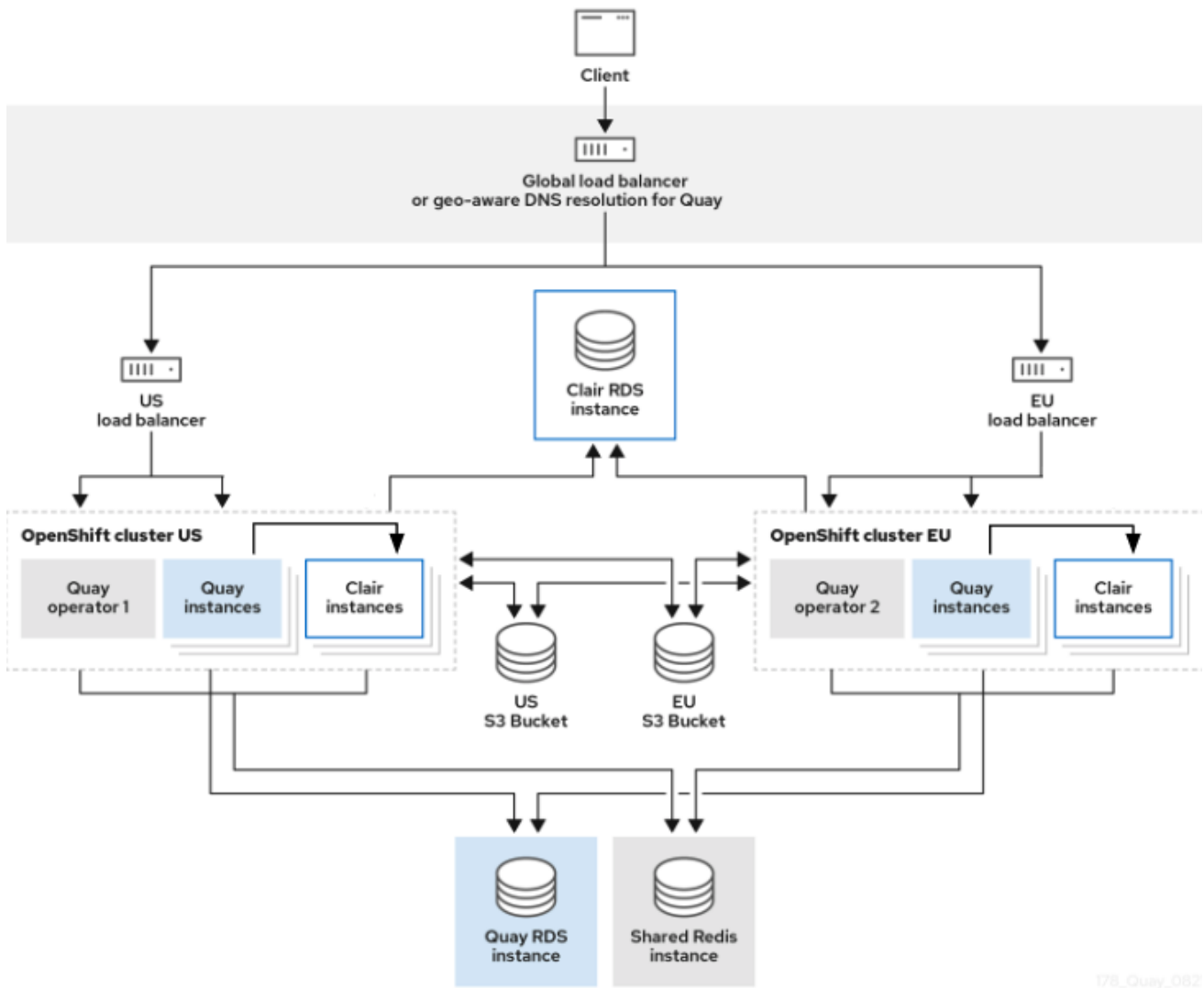


NOTE

The value of the environment variable specified must match the name of a Location ID as defined in the config panel.

3. Restart all Red Hat Quay containers

14.4. GEO-REPLICATION USING THE RED HAT QUAY OPERATOR



178_Quay_062

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

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

14.4.1. Setting up geo-replication on Openshift

Procedure

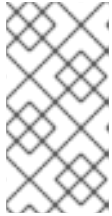
1. Deploy Quay postgres instance:
 - a. Login to the database
 - b. Create a database for Quay

```
CREATE DATABASE quay;
```

- c. Enable `pg_trm` extension inside the database

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

2. Deploy a Redis instance:



NOTE

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

- a. Deploy a VM for Redis
- b. Make sure that it is accessible from the clusters where Quay is running
- c. Port 6379/TCP must be open
- d. Run Redis inside the instance

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

3. Create two object storage backends, one for each cluster
Ideally one object storage bucket will be close to the 1st cluster (primary) while the other will run closer to the 2nd cluster (secondary).
4. Deploy the clusters with the same config bundle, using environment variable overrides to select the appropriate storage backend for an individual cluster
5. Configure a load balancer, to provide a single entry point to the clusters

14.4.1.1. Configuration

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

config.yaml

```
SERVER_HOSTNAME: <georep.quayteam.org or any other name> 1
DB_CONNECTION_ARGS:
  autorollback: true
  threadlocals: true
DB_URI: postgresql://postgres:password@10.19.0.1:5432/quay 2
BUILDLOGS_REDIS:
  host: 10.19.0.2
  port: 6379
USER_EVENTS_REDIS:
  host: 10.19.0.2
```

```

port: 6379
DISTRIBUTED_STORAGE_CONFIG:
  usstorage:
    - GoogleCloudStorage
    - access_key: GOOGQGPVMSAAMQABCDEFG
      bucket_name: georep-test-bucket-0
      secret_key: AYWfEaxX/u84XRA2vUX5C987654321
      storage_path: /quaygcp
  eustorage:
    - GoogleCloudStorage
    - access_key: GOOGQGPVMSAAMQWERTYUIOP
      bucket_name: georep-test-bucket-1
      secret_key: AYWfEaxX/u84XRA2vUX5Cuj12345678
      storage_path: /quaygcp
DISTRIBUTED_STORAGE_DEFAULT_LOCATIONS:
  - usstorage
  - eustorage
DISTRIBUTED_STORAGE_PREFERENCE:
  - usstorage
  - eustorage
FEATURE_STORAGE_REPLICATION: true

```

- 1 A proper **SERVER_HOSTNAME** must be used for the route and must match the hostname of the global load balancer.
- 2 To retrieve the configuration file for a Clair instance deployed using the OpenShift Operator, see [Retrieving the Clair config](#).

Create the **configBundleSecret**:

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

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



NOTE

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

US cluster

```

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

```

```

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

```

+

**NOTE**

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

European cluster

```

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

```

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

+

**NOTE**

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

14.4.2. Mixed storage for geo-replication

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

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

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

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

CHAPTER 15. 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.

15.1. BACKING UP RED HAT QUAY

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

Prerequisites

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



NOTE

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

15.1.1. Red Hat Quay configuration backup

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

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

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

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

3. Backup the managed keys secret:

**NOTE**

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

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

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

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

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

5. Backup the current Quay configuration:

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

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

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

15.1.2. Scale down your Red Hat Quay deployment

**IMPORTANT**

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

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


```

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

```

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

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

```

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

```

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

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

Example output:

```

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

```

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

15.1.3. Red Hat Quay managed database backup



NOTE

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

1. Identify the Quay PostgreSQL pod name:

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

Example output:

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

2. Obtain the Quay database name:

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

3. Download a backup database:

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

15.1.3.1. Red Hat Quay managed object storage backup

The instructions in this section apply to the following configurations:

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



NOTE

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

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

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

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

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

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

```
$ mkdir blobs
```

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



NOTE

You can also use [rclone](#) or [sc3md](#) instead of the AWS command line utility.

15.1.4. Scale the Red Hat Quay deployment back up

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

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

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

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

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

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

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

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

Example output:

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

15.2. RESTORING RED HAT QUAY

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

Prerequisites

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



NOTE

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

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

**NOTE**

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

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

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

**IMPORTANT**

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

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

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

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

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

15.2.2. Scale down your Red Hat Quay deployment

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

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

```
overrides:
  replicas: 0
  ...
```

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

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

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

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

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

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

Example output:

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

15.2.3. Restore your Red Hat Quay database

1. Identify your Quay database pod:

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

Example output:

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

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

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

3. Open a remote terminal to the database:

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

4. Enter `psql`:

```
bash-4.4$ psql
```

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

```
postgres=# \l
```

Example output:

```

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

```

6. Drop the database:

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

Example output:

```
DROP DATABASE
```

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

```
\q
```

8. Redirect your PostgreSQL database to your backup database:

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

9. Exit bash:

```
sh-4.4$ exit
```

15.2.4. Restore your Red Hat Quay object storage data

1. Export the **AWS_ACCESS_KEY_ID**:

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

2. Export the **AWS_SECRET_ACCESS_KEY**:

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

- 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.

15.2.5. Scale up your Red Hat Quay deployment

- 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

- 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>
```

- 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 16. MIGRATING A STANDALONE QUAY DEPLOYMENT TO A RED HAT QUAY OPERATOR MANAGED DEPLOYMENT

The following procedures allow you to back up a standalone Red Hat Quay deployment and migrate it to the Red Hat Quay Operator on OpenShift Container Platform.

16.1. BACKING UP A STANDALONE DEPLOYMENT OF RED HAT QUAY

Procedure

1. Back up the Quay **config.yaml** of your standalone deployment:

```
$ mkdir /tmp/quay-backup
$ cp /path/to/Quay/config/directory/config.yaml /tmp/quay-backup
```

2. Create a backup of the database that your standalone Quay deployment is using:

```
$ pg_dump -h DB_HOST -p 5432 -d QUAY_DATABASE_NAME -U
QUAY_DATABASE_USER -W -O > /tmp/quay-backup/quay-database-backup.sql
```

3. Install the [AWS CLI](#) if you do not have it already.

4. Create an `~/.aws/` directory:

```
$ mkdir ~/.aws/
```

5. Obtain the **access_key** and **secret_key** from the Quay **config.yaml** of your standalone deployment:

```
$ grep -i DISTRIBUTED_STORAGE_CONFIG -A10 /tmp/quay-backup/config.yaml
```

Example output:

```
DISTRIBUTED_STORAGE_CONFIG:
  minio-1:
    - RadosGWStorage
    - access_key: #####
      bucket_name: quay
      hostname: 172.24.10.50
      is_secure: false
      port: "9000"
      secret_key: #####
      storage_path: /datastorage/registry
```

6. Store the **access_key** and **secret_key** from the Quay **config.yaml** file in your `~/.aws` directory:

```
$ touch ~/.aws/credentials
```

7. Optional: Check that your **access_key** and **secret_key** are stored:

```
$ cat > ~/.aws/credentials << EOF
[default]
aws_access_key_id = ACCESS_KEY_FROM_QUAY_CONFIG
aws_secret_access_key = SECRET_KEY_FROM_QUAY_CONFIG
EOF
```

Example output:

```
aws_access_key_id = ACCESS_KEY_FROM_QUAY_CONFIG
aws_secret_access_key = SECRET_KEY_FROM_QUAY_CONFIG
```



NOTE

If the **aws cli** does not automatically collect the **access_key** and **secret_key** from the `~/.aws/credentials` file, you can configure these by running **aws configure** and manually inputting the credentials.

- In your **quay-backup** directory, create a **bucket_backup** directory:

```
$ mkdir /tmp/quay-backup/bucket-backup
```

- Backup all blobs from the S3 storage:

```
$ aws s3 sync --no-verify-ssl --endpoint-url https://PUBLIC_S3_ENDPOINT:PORT
s3://QUAY_BUCKET/ /tmp/quay-backup/bucket-backup/
```



NOTE

The **PUBLIC_S3_ENDPOINT** can be read from the Quay **config.yaml** file under **hostname** in the **DISTRIBUTED_STORAGE_CONFIG**. If the endpoint is insecure, use **http** instead of **https** in the endpoint URL.

Up to this point, you should have a complete backup of all Quay data, blobs, the database, and the **config.yaml** file stored locally. In the following section, you will migrate the standalone deployment backup to Red Hat Quay on OpenShift Container Platform.

16.2. USING BACKED UP STANDALONE CONTENT TO MIGRATE TO OPENSIFT CONTAINER PLATFORM.

Prerequisites

- Your standalone Red Hat Quay data, blobs, database, and **config.yaml** have been backed up.
- Red Hat Quay is deployed on OpenShift Container Platform using the Quay Operator.
- A **QuayRegistry** with all components set to **managed**.



PROCEDURE

The procedure in this documents uses the following namespace: **quay-enterprise**.

1. Scale down the Red Hat Quay Operator:

```
$ oc scale --replicas=0 deployment quay-operator.v3.6.2 -n openshift-operators
```

2. Scale down the application and mirror deployments:

```
$ oc scale --replicas=0 deployment QUAY_MAIN_APP_DEPLOYMENT
QUAY_MIRROR_DEPLOYMENT
```

3. Copy the database SQL backup to the Quay PostgreSQL database instance:

```
$ oc cp /tmp/user/quay-backup/quay-database-backup.sql quay-enterprise/quayregistry-
quay-database-54956cdd54-p7b2w:/var/lib/pgsql/data/userdata
```

4. Obtain the database password from the Operator-created **config.yaml** file:

```
$ oc get deployment quay-quay-app -o json | jq
'.spec.template.spec.volumes[].projected.sources' | grep -i config-secret
```

Example output:

```
"name": "QUAY_CONFIG_SECRET_NAME"
```

```
$ oc get secret quay-quay-config-secret-9t77hb84tb -o json | jq '.data."config.yaml"' | cut -d ""
-f2 | base64 -d -w0 > /tmp/quay-backup/operator-quay-config-yaml-backup.yaml
```

```
cat /tmp/quay-backup/operator-quay-config-yaml-backup.yaml | grep -i DB_URI
```

Example output:

```
postgresql://QUAY_DATABASE_OWNER:PASSWORD@DATABASE_HOST/QUAY_DATAB
ASE_NAME
```

5. Execute a shell inside of the database pod:

```
# oc exec -it quay-postgresql-database-pod -- /bin/bash
```

6. Enter `psql`:

```
bash-4.4$ psql
```

7. Drop the database:

```
postgres=# DROP DATABASE "example-restore-registry-quay-database";
```

Example output:

```
DROP DATABASE
```

8. Create a new database and set the owner as the same name:

```
postgres=# CREATE DATABASE "example-restore-registry-quay-database" OWNER
"example-restore-registry-quay-database";
```

Example output:

```
CREATE DATABASE
```

9. Connect to the database:

```
postgres=# \c "example-restore-registry-quay-database";
```

Example output:

```
You are now connected to database "example-restore-registry-quay-database" as user
"postgres".
```

10. Create a **pg_trgm** extension of your Quay database:

```
example-restore-registry-quay-database=# create extension pg_trgm ;
```

Example output:

```
CREATE EXTENSION
```

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

```
\q
```

12. Set the password for your PostgreSQL deployment:

```
bash-4.4$ psql -h localhost -d "QUAY_DATABASE_NAME" -U QUAY_DATABASE_OWNER
-W < /var/lib/pgsql/data/userdata/quay-database-backup.sql
```

Example output:

```
SET
SET
SET
SET
SET
```

13. Exit bash mode:

```
bash-4.4$ exit
```

14. Create a new configuration bundle for the Red Hat Quay Operator.

```
$ touch config-bundle.yaml
```

15. Create a new configuration bundle for the Red Hat Quay Operator.

15. In your new **config-bundle.yaml**, include all of the information that the registry requires, such as LDAP configuration, keys, and other modifications that your old registry had. Run the following command to move the **secret_key** to your **config-bundle.yaml**:

```
$ cat /tmp/quay-backup/config.yaml | grep SECRET_KEY > /tmp/quay-backup/config-bundle.yaml
```



NOTE

You must manually copy all the LDAP, OIDC and other information and add it to the `/tmp/quay-backup/config-bundle.yaml` file.

16. Create a configuration bundle secret inside of your OpenShift cluster:

```
$ oc create secret generic new-custom-config-bundle --from-file=config.yaml=/tmp/quay-backup/config-bundle.yaml
```

17. Scale up the Quay pods:

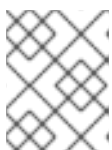
```
$ oc scale --replicas=1 deployment quayregistry-quay-app
deployment.apps/quayregistry-quay-app scaled
```

18. Scale up the mirror pods:

```
$ oc scale --replicas=1 deployment quayregistry-quay-mirror
deployment.apps/quayregistry-quay-mirror scaled
```

19. Patch the **QuayRegistry** CRD so that it contains the reference to the new custom configuration bundle:

```
$ oc patch quayregistry QUAY_REGISTRY_NAME --type=merge -p '{"spec":
{"configBundleSecret":"new-custom-config-bundle"}}'
```



NOTE

If Quay returns a **500** internal server error, you might have to update the **location** of your **DISTRIBUTED_STORAGE_CONFIG** to **default**.

20. Create a new AWS **credentials.yaml** in your `./aws/` directory and include the **access_key** and **secret_key** from the Operator-created **config.yaml** file:

```
$ touch credentials.yaml
```

```
$ grep -i DISTRIBUTED_STORAGE_CONFIG -A10 /tmp/quay-backup/operator-quay-config-yaml-backup.yaml
```

```
$ cat > ~/.aws/credentials << EOF
[default]
aws_access_key_id = ACCESS_KEY_FROM_QUAY_CONFIG
aws_secret_access_key = SECRET_KEY_FROM_QUAY_CONFIG
EOF
```

**NOTE**

If the **aws cli** does not automatically collect the **access_key** and **secret_key** from the `~/.aws/credentials file`, you can configure these by running **aws configure** and manually inputting the credentials.

21. Record the NooBaa's publicly available endpoint:

```
$ oc get route s3 -n openshift-storage -o yaml -o jsonpath="{.spec.host}{'\n'}"
```

22. Sync the backup data to the NooBaa backend storage:

```
$ aws s3 sync --no-verify-ssl --endpoint-url https://NOOBAA_PUBLIC_S3_ROUTE  
/tmp/quay-backup/bucket-backup/* s3://QUAY_DATASTORE_BUCKET_NAME
```

23. Scale the Operator back up to 1 pod:

```
$ oc scale --replicas=1 deployment quay-operator.v3.6.4 -n openshift-operators
```

The Operator will use the custom configuration bundle provided and will reconcile all secrets and deployments. Your new Quay deployment on OpenShift Container Platform should contain all of the information that the old deployment had. All images should be pull-able.

CHAPTER 17. BACKING UP AND RESTORING RED HAT QUAY ON A STANDALONE DEPLOYMENT

Use the content within this section to back up and restore Red Hat Quay in standalone deployments.

17.1. BACKING UP RED HAT QUAY ON STANDALONE DEPLOYMENTS

This procedure describes how to create a backup of Red Hat Quay on standalone deployments.

Procedure

1. Create a temporary backup directory, for example, **quay-backup**:

```
$ mkdir /tmp/quay-backup
```

2. The following example command denotes the local directory that the Red Hat Quay was started in, for example, **/opt/quay-install**:

```
$ podman run --name quay-app \  
-v /opt/quay-install/config:/conf/stack:Z \  
-v /opt/quay-install/storage:/datastorage:Z \  
{productrepo}/{quayimage}:{productminv}
```

Change into the directory that bind-mounts to **/conf/stack** inside of the container, for example, **/opt/quay-install**, by running the following command:

```
$ cd /opt/quay-install
```

3. Compress the contents of your Red Hat Quay deployment into an archive in the **quay-backup** directory by entering the following command:

```
$ tar cvf /tmp/quay-backup/quay-backup.tar.gz *
```

Example output:

```
config.yaml  
config.yaml.bak  
extra_ca_certs/  
extra_ca_certs/ca.crt  
ssl.cert  
ssl.key
```

4. Back up the Quay container service by entering the following command:

```
$ podman inspect quay-app | jq -r '[0].Config.CreateCommand | .[] | paste -s -d ' ' -  
  
/usr/bin/podman run --name quay-app \  
-v /opt/quay-install/config:/conf/stack:Z \  
-v /opt/quay-install/storage:/datastorage:Z \  
{productrepo}/{quayimage}:{productminv}
```


- Redirect the contents of your **conf/stack/config.yaml** file to your temporary **quay-config.yaml** file by entering the following command:

```
$ podman exec -it quay cat /conf/stack/config.yaml > /tmp/quay-backup/quay-config.yaml
```

- Obtain the **DB_URI** located in your temporary **quay-config.yaml** by entering the following command:

```
$ grep DB_URI /tmp/quay-backup/quay-config.yaml
```

Example output:

```
$ postgresql://<username>:test123@172.24.10.50/quay
```

- Extract the PostgreSQL contents to your temporary backup directory in a backup .sql file by entering the following command:

```
$ pg_dump -h 172.24.10.50 -p 5432 -d quay -U <username> -W -O > /tmp/quay-backup/quay-backup.sql
```

- Print the contents of your **DISTRIBUTED_STORAGE_CONFIG** by entering the following command:

```
DISTRIBUTED_STORAGE_CONFIG:
default:
- S3Storage
- s3_bucket: <bucket_name>
  storage_path: /registry
  s3_access_key: <s3_access_key>
  s3_secret_key: <s3_secret_key>
  host: <host_name>
```

- Export the **AWS_ACCESS_KEY_ID** by using the **access_key** credential obtained in Step 7:

```
$ export AWS_ACCESS_KEY_ID=<access_key>
```

- Export the **AWS_SECRET_ACCESS_KEY** by using the **secret_key** obtained in Step 7:

```
$ export AWS_SECRET_ACCESS_KEY=<secret_key>
```

- Sync the **quay** bucket to the **/tmp/quay-backup/blob-backup/** directory from the **hostname** of your **DISTRIBUTED_STORAGE_CONFIG**:

```
$ aws s3 sync s3://<bucket_name> /tmp/quay-backup/blob-backup/ --source-region us-east-2
```

Example output:

```
download:
s3://<user_name>/registry/sha256/9c/9c3181779a868e09698b567a3c42f3744584ddb1398efe2c4ba569a99b823f7a to
registry/sha256/9c/9c3181779a868e09698b567a3c42f3744584ddb1398efe2c4ba569a99b823f7a
```

```
download:  
s3://<user_name>/registry/sha256/e9/e9c5463f15f0fd62df3898b36ace8d15386a6813ffb470f33  
2698ecb34af5b0d to  
registry/sha256/e9/e9c5463f15f0fd62df3898b36ace8d15386a6813ffb470f332698ecb34af5b0d
```

It is recommended that you delete the **quay-config.yaml** file after syncing the **quay** bucket because it contains sensitive information. The **quay-config.yaml** file will not be lost because it is backed up in the **quay-backup.tar.gz** file.

17.2. RESTORING RED HAT QUAY ON STANDALONE DEPLOYMENTS

This procedure describes how to restore Red Hat Quay on standalone deployments.

Prerequisites

- You have backed up your Red Hat Quay deployment.

Procedure

1. Create a new directory that will bind-mount to **/conf/stack** inside of the Red Hat Quay container:

```
$ mkdir /opt/new-quay-install
```

2. Copy the contents of your temporary backup directory created in [Backing up Red Hat Quay on standalone deployments](#) to the **new-quay-install1** directory created in Step 1:

```
$ cp /tmp/quay-backup/quay-backup.tar.gz /opt/new-quay-install/
```

3. Change into the **new-quay-install** directory by entering the following command:

```
$ cd /opt/new-quay-install/
```

4. Extract the contents of your Red Hat Quay directory:

```
$ tar xvf /tmp/quay-backup/quay-backup.tar.gz *
```

Example output:

```
config.yaml  
config.yaml.bak  
extra_ca_certs/  
extra_ca_certs/ca.crt  
ssl.cert  
ssl.key
```

5. Recall the **DB_URI** from your backed-up **config.yaml** file by entering the following command:

```
$ grep DB_URI config.yaml
```

Example output:

```
pgsql://<username>:test123@172.24.10.50/quay
```

- Run the following command to enter the PostgreSQL database server:

```
$ sudo postgres
```

- Enter `psql` and create a new database in 172.24.10.50 to restore the quay databases, for example, **example_restore_registry_quay_database**, by entering the following command:

```
$ psql "host=172.24.10.50 port=5432 dbname=postgres user=<username>
password=test123"
postgres=> CREATE DATABASE example_restore_registry_quay_database;
```

Example output:

```
CREATE DATABASE
```

- Connect to the database by running the following command:

```
postgres=# \c "example-restore-registry-quay-database";
```

Example output:

```
You are now connected to database "example-restore-registry-quay-database" as user
"postgres".
```

- Create a **pg_trgm** extension of your Quay database by running the following command:

```
example_restore_registry_quay_database=> CREATE EXTENSION IF NOT EXISTS
pg_trgm;
```

Example output:

```
CREATE EXTENSION
```

- Exit the postgres CLI by entering the following command:

```
\q
```

- Import the database backup to your new database by running the following command:

```
$ psql "host=172.24.10.50 port=5432 dbname=example_restore_registry_quay_database
user=<username> password=test123" -W < /tmp/quay-backup/quay-backup.sql
```

Example output:

```
SET
SET
SET
SET
SET
```

Update the value of **DB_URI** in your **config.yaml** from **postgresql://<username>:test123@172.24.10.50/quay** to **postgresql://<username>:test123@172.24.10.50/example-restore-registry-quay-database** before restarting the Red Hat Quay deployment.



NOTE

The **DB_URI** format is **DB_URI postgresql://<login_user_name>:<login_user_password>@<postgresql_host>/<quay_database>**. If you are moving from one PostgreSQL server to another PostgreSQL server, update the value of **<login_user_name>**, **<login_user_password>** and **<postgresql_host>** at the same time.

- In the **/opt/new-quay-install** directory, print the contents of your **DISTRIBUTED_STORAGE_CONFIG** bundle:

```
$ cat config.yaml | grep DISTRIBUTED_STORAGE_CONFIG -A10
```

Example output:

```
DISTRIBUTED_STORAGE_CONFIG:
  default:
DISTRIBUTED_STORAGE_CONFIG:
  default:
  - S3Storage
  - s3_bucket: <bucket_name>
    storage_path: /registry
    s3_access_key: <s3_access_key>
    s3_secret_key: <s3_secret_key>
    host: <host_name>
```



NOTE

Your **DISTRIBUTED_STORAGE_CONFIG** in **/opt/new-quay-install** must be updated before restarting your Red Hat Quay deployment.

- Export the **AWS_ACCESS_KEY_ID** by using the **access_key** credential obtained in Step 13:

```
$ export AWS_ACCESS_KEY_ID=<access_key>
```

- Export the **AWS_SECRET_ACCESS_KEY** by using the **secret_key** obtained in Step 13:

```
$ export AWS_SECRET_ACCESS_KEY=<secret_key>
```

- Create a new s3 bucket by entering the following command:

```
$ aws s3 mb s3://<new_bucket_name> --region us-east-2
```

Example output:

```
$ make_bucket: quay
```

16. Upload all blobs to the new s3 bucket by entering the following command:

```
$ aws s3 sync --no-verify-ssl \
--endpoint-url <example_endpoint_url> 1
/tmp/quay-backup/blob-backup/. s3://quay/
```

- 1 The Red Hat Quay registry endpoint must be the same before backup and after restore.

Example output:

```
upload: ../../tmp/quay-backup/blob-
backup/datastorage/registry/sha256/50/505edb46ea5d32b5cbe275eb766d960842a52ee77ac2
25e4dc8abb12f409a30d to
s3://quay/datastorage/registry/sha256/50/505edb46ea5d32b5cbe275eb766d960842a52ee77ac
225e4dc8abb12f409a30d
upload: ../../tmp/quay-backup/blob-
backup/datastorage/registry/sha256/27/27930dc06c2ee27ac6f543ba0e93640dd21eea458eac4
7355e8e5989dea087d0 to
s3://quay/datastorage/registry/sha256/27/27930dc06c2ee27ac6f543ba0e93640dd21eea458ea
c47355e8e5989dea087d0
upload: ../../tmp/quay-backup/blob-
backup/datastorage/registry/sha256/8c/8c7daf5e20eee45ffe4b36761c4bb6729fb3ee60d4f588f
712989939323110ec to
s3://quay/datastorage/registry/sha256/8c/8c7daf5e20eee45ffe4b36761c4bb6729fb3ee60d4f58
8f712989939323110ec
...
```

17. Before restarting your Red Hat Quay deployment, update the storage settings in your config.yaml:

```
DISTRIBUTED_STORAGE_CONFIG:
  default:
DISTRIBUTED_STORAGE_CONFIG:
  default:
  - S3Storage
  - s3_bucket: <new_bucket_name>
    storage_path: /registry
    s3_access_key: <s3_access_key>
    s3_secret_key: <s3_secret_key>
    host: <host_name>
```

CHAPTER 18. RED HAT QUAY GARBAGE COLLECTION

18.1. ABOUT RED HAT QUAY GARBAGE COLLECTION

Red Hat Quay includes automatic and continuous image garbage collection. Garbage collection ensures efficient use of resources for active objects by removing objects that occupy sizeable amounts of disk space, such as dangling or untagged images, repositories, and blobs, including layers and manifests. Garbage collection performed by Red Hat Quay can reduce downtime in your organization's environment.

18.2. RED HAT QUAY GARBAGE COLLECTION IN PRACTICE

Currently, all garbage collection happens discreetly; there are no commands to manually run garbage collection. Red Hat Quay provides metrics that track the status of the different garbage collection workers.

For namespace and repository garbage collection, the progress is tracked based on the size of their respective queues. Namespace and repository garbage collection workers require a global lock to work. As a result, and for performance reasons, only one worker runs at a time.



NOTE

Red Hat Quay shares blobs between namespaces and repositories in order to conserve disk space. For example, if the same image is pushed 10 times, only one copy of that image will be stored.

It is possible that tags can share their layers with different images already stored somewhere in Red Hat Quay. In that case, blobs will stay in storage, because deleting shared blobs would make other images unusable.

Blob expiration is independent of the time machine. If you push a tag to Red Hat Quay and the time machine is set to 0 seconds, and then you delete a tag immediately, garbage collection deletes the tag and everything related to that tag, but will not delete the blob storage until the blob expiration time is reached.

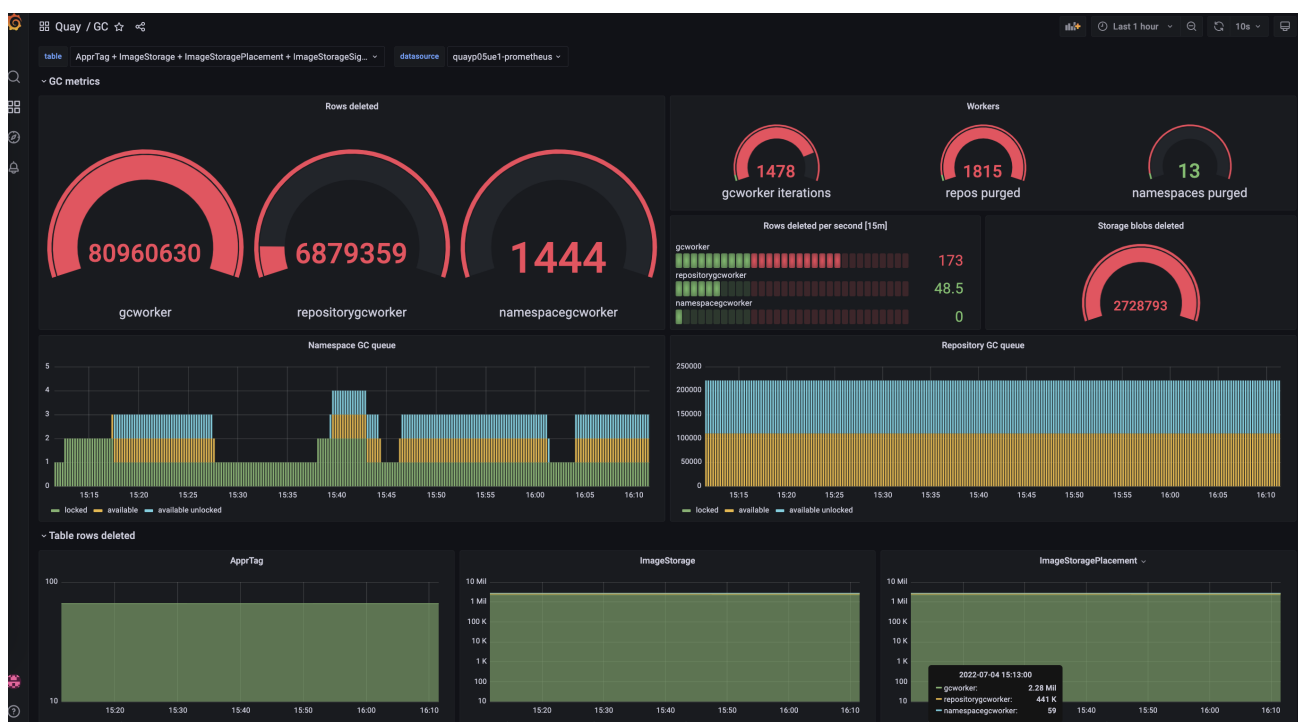
Garbage collecting tagged images works differently than garbage collection on namespaces or repositories. Rather than having a queue of items to work with, the garbage collection workers for tagged images actively search for a repository with inactive or expired tags to clean up. Each instance of garbage collection workers will grab a repository lock, which results in one worker per repository.



NOTE

- In Red Hat Quay, inactive or expired tags are manifests without tags because the last tag was deleted or it expired. The manifest stores information about how the image is composed and stored in the database for each individual tag. When a tag is deleted and the allotted time from **Time Machine** has been met, Red Hat Quay garbage collects the blobs that are not connected to any other manifests in the registry. If a particular blob is connected to a manifest, then it is preserved in storage and only its connection to the manifest that is being deleted is removed.
- Expired images will disappear after the allotted time, but are still stored in Red Hat Quay. The time in which an image is completely deleted, or collected, depends on the **Time Machine** setting of your organization. The default time for garbage collection is 14 days unless otherwise specified. Until that time, tags can be pointed to an expired or deleted images.

For each type of garbage collection, Red Hat Quay provides metrics for the number of rows per table deleted by each garbage collection worker. The following image shows an example of how Red Hat Quay monitors garbage collection with the same metrics:



18.2.1. Measuring storage reclamation

Red Hat Quay does not have a way to track how much space is freed up by garbage collection. Currently, the best indicator of this is by checking how many blobs have been deleted in the provided metrics.



NOTE

The **UploadedBlob** table in the Red Hat Quay metrics tracks the various blobs that are associated with a repository. When a blob is uploaded, it will not be garbage collected before the time designated by the **PUSH_TEMP_TAG_EXPIRATION_SEC** parameter. This is to avoid prematurely deleting blobs that are part of an ongoing push. For example, if garbage collection is set to run often, and a tag is deleted in the span of less than one hour, then it is possible that the associated blobs will not get cleaned up immediately. Instead, and assuming that the time designated by the **PUSH_TEMP_TAG_EXPIRATION_SEC** parameter has passed, the associated blobs will be removed the next time garbage collection runs on that same repository.

18.3. GARBAGE COLLECTION CONFIGURATION FIELDS

The following configuration fields are available to customize what is garbage collected, and the frequency at which garbage collection occurs:

Name	Description	Schema
FEATURE_GARBAGE_COLLECTION	Whether garbage collection is enabled for image tags. Defaults to true .	Boolean
FEATURE_NAMESPACE_GARBAGE_COLLECTION	Whether garbage collection is enabled for namespaces. Defaults to true .	Boolean
FEATURE_REPOSITORY_GARBAGE_COLLECTION	Whether garbage collection is enabled for repositories. Defaults to true .	Boolean
GARBAGE_COLLECTION_FREQUENCY	The frequency, in seconds, at which the garbage collection worker runs. Affects only garbage collection workers. Defaults to 30 seconds.	String
PUSH_TEMP_TAG_EXPIRATION_SEC	The number of seconds that blobs will not be garbage collected after being uploaded. This feature prevents garbage collection from cleaning up blobs that are not referenced yet, but still used as part of an ongoing push.	String

Name	Description	Schema
TAG_EXPIRATION_OPTIONS	List of valid tag expiration values.	String
DEFAULT_TAG_EXPIRATION	Tag expiration time for time machine.	String
CLEAN_BLOB_UPLOAD_FOLDER	Automatically cleans stale blobs left over from an S3 multipart upload. By default, blob files older than two days are cleaned up every hour.	Boolean + Default: true

18.4. DISABLING GARBAGE COLLECTION

The garbage collection features for image tags, namespaces, and repositories are stored in the **config.yaml** file. These features default to **true**.

In rare cases, you might want to disable garbage collection, for example, to control when garbage collection is performed. You can disable garbage collection by setting the **GARBAGE_COLLECTION** features to **false**. When disabled, dangling or untagged images, repositories, namespaces, layers, and manifests are not removed. This might increase the downtime of your environment.



NOTE

There is no command to manually run garbage collection. Instead, you would disable, and then re-enable, the garbage collection feature.

18.5. GARBAGE COLLECTION AND QUOTA MANAGEMENT

Red Hat Quay introduced quota management in 3.7. With quota management, users have the ability to report storage consumption and to contain registry growth by establishing configured storage quota limits.

As of Red Hat Quay 3.7, garbage collection reclaims memory that was allocated to images, repositories, and blobs after deletion. Because the garbage collection feature reclaims memory after deletion, there is a discrepancy between what is stored in an environment's disk space and what quota management is reporting as the total consumption. There is currently no workaround for this issue.

18.6. GARBAGE COLLECTION IN PRACTICE

Use the following procedure to check your Red Hat Quay logs to ensure that garbage collection is working.

Procedure

1. Enter the following command to ensure that garbage collection is properly working:

```
$ sudo podman logs <container_id>
```

Example output:

```
gcworker stdout | 2022-11-14 18:46:52,458 [63] [INFO] [apscheduler.executors.default] Job
"GarbageCollectionWorker._garbage_collection_repos (trigger: interval[0:00:30], next run at: 2022-
11-14 18:47:22 UTC)" executed successfully
```

1. Delete an image tag.
2. Enter the following command to ensure that the tag was deleted:

```
$ podman logs quay-app
```

Example output:

```
unicorn-web stdout | 2022-11-14 19:23:44,574 [233] [INFO] [unicorn.access] 192.168.0.38 - -
[14/Nov/2022:19:23:44 +0000] "DELETE /api/v1/repository/quayadmin/busybox/tag/test HTTP/1.0"
204 0 "http://quay-server.example.com/repository/quayadmin/busybox?tab=tags" "Mozilla/5.0 (X11;
Linux x86_64; rv:102.0) Gecko/20100101 Firefox/102.0"
```

18.7. RED HAT QUAY GARBAGE COLLECTION METRICS

The following metrics show how many resources have been removed by garbage collection. These metrics show how many times the garbage collection workers have run and how many namespaces, repositories, and blobs were removed.

Metric name	Description
quay_gc_iterations_total	Number of iterations by the GCWorker
quay_gc_namespaces_purged_total	Number of namespaces purged by the NamespaceGCWorker
quay_gc_repos_purged_total	Number of repositories purged by the RepositoryGCWorker or NamespaceGCWorker
quay_gc_storage_blobs_deleted_total	Number of storage blobs deleted

Sample metrics output

```
# TYPE quay_gc_iterations_created gauge
quay_gc_iterations_created{host="example-registry-quay-app-6df87f7b66-
9tfn6",instance="",job="quay",pid="208",process_name="secscan:application"}
1.6317823190189714e+09
...

# HELP quay_gc_iterations_total number of iterations by the GCWorker
# TYPE quay_gc_iterations_total counter
quay_gc_iterations_total{host="example-registry-quay-app-6df87f7b66-
```

```
9tfn6",instance="",job="quay",pid="208",process_name="secscan:application"} 0
...

# TYPE quay_gc_namespaces_purged_created gauge
quay_gc_namespaces_purged_created{host="example-registry-quay-app-6df87f7b66-
9tfn6",instance="",job="quay",pid="208",process_name="secscan:application"}
1.6317823190189433e+09
...

# HELP quay_gc_namespaces_purged_total number of namespaces purged by the
NamespaceGCWorker
# TYPE quay_gc_namespaces_purged_total counter
quay_gc_namespaces_purged_total{host="example-registry-quay-app-6df87f7b66-
9tfn6",instance="",job="quay",pid="208",process_name="secscan:application"} 0
....

# TYPE quay_gc_repos_purged_created gauge
quay_gc_repos_purged_created{host="example-registry-quay-app-6df87f7b66-
9tfn6",instance="",job="quay",pid="208",process_name="secscan:application"}
1.631782319018925e+09
...

# HELP quay_gc_repos_purged_total number of repositories purged by the RepositoryGCWorker or
NamespaceGCWorker
# TYPE quay_gc_repos_purged_total counter
quay_gc_repos_purged_total{host="example-registry-quay-app-6df87f7b66-
9tfn6",instance="",job="quay",pid="208",process_name="secscan:application"} 0
...

# TYPE quay_gc_storage_blobs_deleted_created gauge
quay_gc_storage_blobs_deleted_created{host="example-registry-quay-app-6df87f7b66-
9tfn6",instance="",job="quay",pid="208",process_name="secscan:application"}
1.6317823190189059e+09
...

# HELP quay_gc_storage_blobs_deleted_total number of storage blobs deleted
# TYPE quay_gc_storage_blobs_deleted_total counter
quay_gc_storage_blobs_deleted_total{host="example-registry-quay-app-6df87f7b66-
9tfn6",instance="",job="quay",pid="208",process_name="secscan:application"} 0
...
```

CHAPTER 19. RED HAT QUAY TROUBLESHOOTING

Common failure modes and best practices for recovery.

- [I'm receiving HTTP Status Code 429](#)
- [I'm authorized but I'm still getting 403s](#)
- [Base image pull in Dockerfile fails with 403](#)
- [Cannot add a build trigger](#)
- [Build logs are not loading](#)
- [I'm receiving "Cannot locate specified Dockerfile" * Could not reach any registry endpoint](#)
- [Cannot access private repositories using EC2 Container Service](#)
- [Docker is returning an i/o timeout](#)
- [Docker login is failing with an odd error](#)
- [Pulls are failing with an odd error](#)
- [I just pushed but the timestamp is wrong](#)
- [Pulling Private Quay.io images with Marathon/Mesos fails](#)

CHAPTER 20. SCHEMA FOR RED HAT QUAY CONFIGURATION

Most Red Hat Quay configuration information is stored in the **config.yaml** file that is created using the browser-based config tool when Red Hat Quay is first deployed.

The configuration options are described in the Red Hat Quay Configuration Guide.

ADDITIONAL RESOURCES