



Red Hat Quay 3.6

Manage Red Hat Quay

Manage Red Hat Quay

Red Hat Quay 3.6 Manage Red Hat Quay

Manage Red Hat Quay

Legal Notice

Copyright © 2022 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	5
CHAPTER 1. ADVANCED RED HAT QUAY CONFIGURATION	6
1.1. USING RED HAT QUAY CONFIG TOOL TO MODIFY RED HAT QUAY	6
1.1.1. Running the Config Tool from the Red Hat Quay Operator	6
1.1.2. Running the Config Tool from the command line	7
1.2. USING THE API TO MODIFY RED HAT QUAY	8
1.3. EDITING THE CONFIG.YAML FILE TO MODIFY RED HAT QUAY	8
1.3.1. Add name and company to Red Hat Quay sign-in	8
1.3.2. Disable TLS Protocols	8
1.3.3. Rate limit API calls	8
1.3.4. Adjust database connection pooling	8
1.3.4.1. Database connection arguments	9
1.3.4.2. Database SSL configuration	9
1.3.4.2.1. PostgreSQL SSL connection arguments	9
1.3.4.2.2. MySQL SSL connection arguments	10
1.3.4.3. HTTP connection counts	10
1.3.4.4. Dynamic process counts	10
1.3.4.5. Environment variables	11
1.3.4.6. Turning off connection pooling	11
CHAPTER 2. USING THE CONFIGURATION API	12
2.1. RETRIEVING THE DEFAULT CONFIGURATION	12
2.2. RETRIEVING THE CURRENT CONFIGURATION	12
2.3. VALIDATING CONFIGURATION USING THE API	13
2.4. DETERMINING THE REQUIRED FIELDS	14
CHAPTER 3. GETTING RED HAT QUAY RELEASE NOTIFICATIONS	15
CHAPTER 4. USING SSL TO PROTECT CONNECTIONS TO RED HAT QUAY	16
4.1. INTRODUCTION TO USING SSL	16
4.2. CREATE A CERTIFICATE AUTHORITY AND SIGN A CERTIFICATE	16
4.2.1. Create a Certificate Authority	16
4.2.2. Sign a certificate	16
4.3. CONFIGURING SSL USING THE COMMAND LINE	17
4.4. CONFIGURING SSL USING THE UI	18
4.5. TESTING SSL CONFIGURATION USING THE COMMAND LINE	18
4.6. TESTING SSL CONFIGURATION USING THE BROWSER	19
4.7. CONFIGURING PODMAN TO TRUST THE CERTIFICATE AUTHORITY	19
4.8. CONFIGURING THE SYSTEM TO TRUST THE CERTIFICATE AUTHORITY	20
CHAPTER 5. ADDING TLS CERTIFICATES TO THE RED HAT QUAY CONTAINER	22
5.1. ADD TLS CERTIFICATES TO RED HAT QUAY	22
5.2. ADD CERTS WHEN DEPLOYED ON KUBERNETES	22
CHAPTER 6. CONFIGURING ACTION LOG STORAGE FOR ELASTICSEARCH	24
CHAPTER 7. CLAIR SECURITY SCANNING	26
7.1. SETTING UP CLAIR ON A RED HAT QUAY OPENSIFT DEPLOYMENT	26
7.1.1. Deploying Via the Quay Operator	26
7.1.2. Manually Deploying Clair	26
7.2. SETTING UP CLAIR ON A NON-OPENSIFT RED HAT QUAY DEPLOYMENT	31
7.3. USING CLAIR	32

7.4. CVE RATINGS FROM THE NATIONAL VULNERABILITY DATABASE	33
7.5. CONFIGURING CLAIR FOR DISCONNECTED ENVIRONMENTS	33
7.6. CLAIR UPDATER URLS	34
7.7. ADDITIONAL INFORMATION	34
CHAPTER 8. SCAN POD IMAGES WITH THE CONTAINER SECURITY OPERATOR	36
8.1. RUN THE CSO IN OPENSIFT	36
8.2. QUERY IMAGE VULNERABILITIES FROM THE CLI	38
CHAPTER 9. INTEGRATE RED HAT QUAY INTO OPENSIFT WITH THE BRIDGE OPERATOR	39
9.1. RUNNING THE QUAY BRIDGE OPERATOR	39
9.1.1. Prerequisites	39
9.1.2. Setting up and configuring OpenShift and Red Hat Quay	39
9.1.3. Red Hat Quay setup	39
9.1.4. OpenShift Setup	40
9.1.4.1. Deploying the Operator	40
9.1.4.2. Creating an OpenShift secret for the OAuth token	40
9.1.4.3. Create the QuayIntegration Custom Resource	40
CHAPTER 10. REPOSITORY MIRRORING	42
10.1. REPOSITORY MIRRORING	42
10.2. REPOSITORY MIRRORING VERSUS GEO-REPLICATION	42
10.3. USING REPOSITORY MIRRORING	43
10.4. MIRRORING CONFIGURATION UI	44
10.5. MIRRORING CONFIGURATION FIELDS	44
10.6. MIRRORING WORKER	45
10.7. CREATING A MIRRORED REPOSITORY	45
10.7.1. Repository mirroring settings	45
10.7.2. Advanced settings	46
10.7.3. Synchronize now	47
10.8. EVENT NOTIFICATIONS FOR MIRRORING	48
10.9. MIRRORING TAG PATTERNS	48
10.9.1. Pattern syntax	48
10.9.2. Example tag patterns	48
10.10. WORKING WITH MIRRORED REPOSITORIES	49
10.11. REPOSITORY MIRRORING RECOMMENDATIONS	51
CHAPTER 11. BACKING UP AND RESTORING RED HAT QUAY ON AN OPENSIFT CONTAINER PLATFORM DEPLOYMENT	52
11.1. BACKING UP RED HAT QUAY	52
11.2. RESTORING RED HAT QUAY	55
CHAPTER 12. LDAP AUTHENTICATION SETUP FOR RED HAT QUAY	59
12.1. CONSIDERATIONS PRIOR TO ENABLING LDAP	59
12.1.1. Existing Quay deployments	59
12.1.2. Manual User Creation and LDAP authentication	59
12.2. SET UP LDAP CONFIGURATION	59
12.2.1. Full LDAP URI	59
12.2.2. Team Synchronization	60
12.2.3. Base and Relative Distinguished Names	60
12.2.4. Additional User Filters	61
12.2.5. Administrator DN	62
12.2.6. UID and Mail attributes	62
12.2.7. Validation	63

12.3. COMMON ISSUES	63
12.4. CONFIGURE AN LDAP USER AS SUPERUSER	63
CHAPTER 13. PROMETHEUS AND GRAFANA METRICS UNDER RED HAT QUAY	65
13.1. EXPOSING THE PROMETHEUS ENDPOINT	65
13.1.1. Standalone Red Hat Quay	65
13.1.2. Red Hat Quay Operator	65
13.1.3. Setting up Prometheus to consume metrics	66
13.1.4. DNS configuration under Kubernetes	66
13.1.5. DNS configuration for a manual cluster	66
13.2. INTRODUCTION TO METRICS	66
13.2.1. General registry statistics	66
13.2.2. Queue items	67
13.2.3. Garbage collection metrics	68
13.2.3.1. Multipart uploads metrics	69
13.2.4. Image push / pull metrics	70
13.2.4.1. Image pulls total	70
13.2.4.2. Image bytes pulled	71
13.2.4.3. Image pushes total	71
13.2.4.4. Image bytes pushed	71
13.2.5. Authentication metrics	71
CHAPTER 14. GEO-REPLICATION	73
14.1. GEO-REPLICATION FEATURES	73
14.2. GEO-REPLICATION REQUIREMENTS AND CONSTRAINTS	73
14.3. GEO-REPLICATION ARCHITECTURE	74
14.4. ENABLE STORAGE REPLICATION	74
14.4.1. Run Red Hat Quay with storage preferences	75
CHAPTER 15. RED HAT QUAY TROUBLESHOOTING	76
CHAPTER 16. SCHEMA FOR RED HAT QUAY CONFIGURATION	77
ADDITIONAL RESOURCES	77

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 several different interfaces:

- The Red Hat Quay Config Tool: Running the **Quay** container in **config** mode presents a Web-based interface for configuring the Red Hat Quay cluster. This is the recommended method for most configuration of the Red Hat Quay service itself.
- Editing the **config.yaml**: The **config.yaml** file holds most of the configuration information for the Red Hat Quay cluster. Editing that 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 configuration can be done through the API.

While configuration for specific features is covered in separate sections, this section describes how to use each of those interfaces and perform some more advanced configuration.

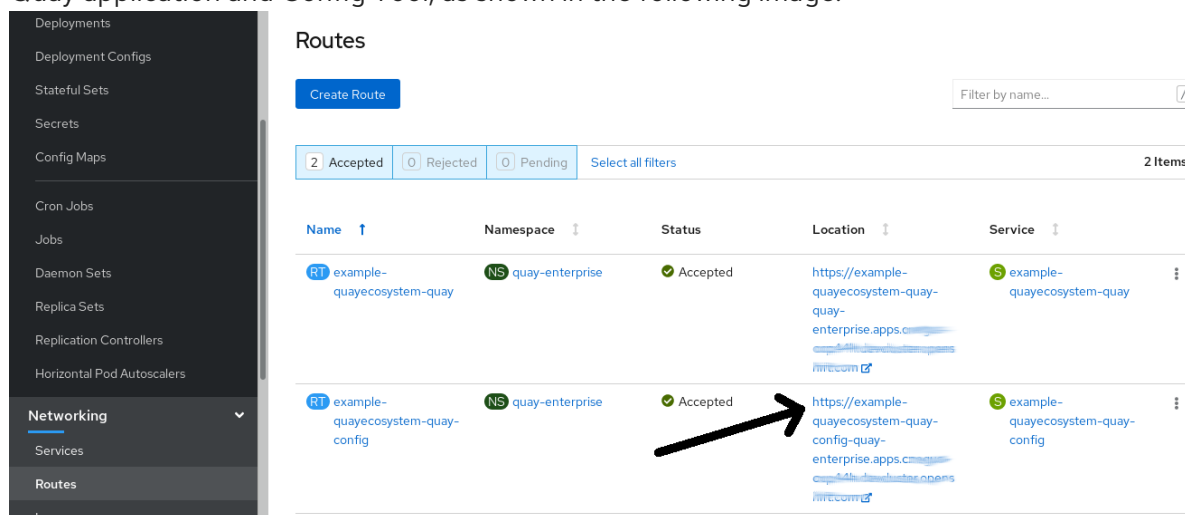
1.1. USING RED HAT QUAY CONFIG TOOL TO MODIFY RED HAT QUAY

The Red Hat Quay Config Tool is made available by running a **Quay** container in **config** mode alongside the regular Red Hat Quay service. Running the Config Tool is different for Red Hat Quay clusters running on OpenShift than it is for those running directly on host systems.

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

If you are running the Red Hat Quay Operator from OpenShift, the Config Tool is probably already available for you to use. To access the Config Tool, do the following:

1. From the OpenShift console, select the project in which Red Hat Quay is running. For example, quay-enterprise.
2. From the left column, select Networking → Routes. You should see routes to both the Red Hat Quay application and Config Tool, as shown in the following image:



3. Select the route to the Config Tool (for example, example-quayecosystem-quay-config) and select it. The Config tool Web UI should open in your browser.
4. Select **Modify configuration for this cluster**. You should see the Config Tool, ready for you to change features of your Red Hat Quay cluster, as shown in the following image:

Red Hat Quay Setup

Almost done!
Configure your Redis database and other settings below

① — 🗄️ — ② — ③ — ④ — 📄

Custom SSL Certificates

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

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

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

Upload certificates: Select file

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

CERTIFICATE FILENAME	STATUS	NAMES HANDLED	⚙️
quay.crt	✔️ Certificate is valid	example-quayecosystem-quay-quay-enterprise.apps.cnegus-ocp44h.devcluster.openshift.com quay-enterprise quay-enterprise@1599743640	⚙️
clair.crt	✔️ Certificate is valid	example-quayecosystem-clair example-quayecosystem-clair.quay-enterprise.svc example-quayecosystem-clair.quay-enterprise.svc.local example-quayecosystem-clair@1599743641	⚙️

Save Configuration Changes

5. When you have made the changes you want, select **Save Configuration Changes**. The Config Tool will validate your changes.
6. Make any corrections as needed by selecting **Continue Editing** or select **Next** to continue on.
7. When prompted, it is recommended that you select **Download Configuration**. That will download a tarball of your new **config.yaml**, as well as any certificates and keys used with your Red Hat Quay setup.
8. Select **Go to deployment rollout**, then **Populate the configuration to deployments**. The Red Hat Quay pods will be restarted and the changes will take effect.

The **config.yaml** file you saved can be used to make advanced changes to your configuration or just kept for future reference.

1.1.2. Running the Config Tool from the command line

If you are running Red Hat Quay directly from a host system, using tools such as the **podman** or **docker** commands, after the initial Red Hat Quay deployment, you can restart the Config Tool to modify your Red Hat Quay cluster. Here's how:

1. **Start quay in config mode** On the first **quay** node run the following, replacing **my-secret-password** with your password. If you would like to modify an existing config bundle, you can simply mount your configuration directory into the **Quay** container as you would in registry mode.

```
# podman run --rm -it --name quay_config -p 8080:8080 \
-v path/to/config-bundle:/conf/stack \
registry.redhat.io/quay/quay-rhel8:v3.6.8 config my-secret-password
```

2. **Open browser:** When the quay configuration tool starts up, open a browser to the URL and port 8080 of the system you are running the configuration tool on (for example <https://myquay.example.com:8080>). You are prompted for a username and password.

At this point, you can begin modifying your Red Hat Quay cluster as described earlier.

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 Red Hat Quay configuration that is not available through the Config Tool can be achieved by editing the `config.yaml` file directly. Available settings are described in the [Schema for Red Hat Quay configuration](#). The following are examples of settings you can change directly in the `config.yaml` file.

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

Setting the following will cause users to be prompted for their name and company when they first sign in. Although this is optional, it can provide you 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 as follows:

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

1.3.3. Rate limit API calls

Adding the `FEATURE_RATE_LIMITS` parameter to the `config.yaml` causes `nginx` to limit certain API calls to 30 per second. If that feature is not set, API calls are limited to 300 per second (effectively unlimited). Rate limiting can be an important feature, if you need to make sure the resources available are not overwhelmed with traffic.

Some namespace may require unlimited access (perhaps they are important to CI/CD and take priority, for example). In this case, those namespace may be placed in a list in `config.yaml` for `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.

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

Overtime, the connection pools will release idle connections. To release all connections immediately, Red Hat Quay requires a restart.

Database connection pooling may be toggled by setting the environment variable **DB_CONNECTION_POOLING={true|false}**

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

```
DB_CONNECTION_ARGS:
  max_connections: 10
```

1.3.4.1. Database connection arguments

You can customize Red Hat Quay database connection settings within the **config.yaml** file. These are entirely dependent upon the underlying database driver, such as **psycopg2** for Postgres and **pymysql** for MySQL. It is also possible to pass in arguments used by Peewee's Connection Pooling mechanism as seen below.

```
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 **DB_CONNECTION_ARGS** are generic while others are database-specific. In particular, SSL configuration depends on the database you are deploying.

1.3.4.2.1. PostgreSQL SSL connection arguments

A sample PostgreSQL SSL configuration is given below:

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

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

- **disable:** only try a non-SSL connection
- **allow:** first try a non-SSL connection; if that fails, try an SSL connection
- **prefer:** (default) first try an SSL connection; if that fails, try a non-SSL connection
- **require:** only try an SSL connection. If a root CA file is present, verify the certificate 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 trusted certificate authority (CA)
- **verify-full:** only try an SSL connection, verify that the server certificate is issued by a trusted CA and that the requested server host name matches that in the certificate

More information on the valid arguments for PostgreSQL is available at <https://www.postgresql.org/docs/current/libpq-connect.html>.

1.3.4.2.2. MySQL SSL connection arguments

A sample MySQL SSL configuration follows:

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

Information on the valid connection arguments for MySQL is available at <https://dev.mysql.com/doc/refman/8.0/en/connecting-using-uri-or-key-value-pairs.html>.

1.3.4.3. HTTP connection counts

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

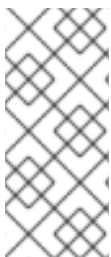
Environment variables:

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

Specifying a count for a specific component will override any value set in `WORKER_CONNECTION_COUNT`.

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 will make 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 x 2
- environment_variable: WORKER_COUNT_WEB
- seccan - Interacts with Clair
 - minimum: 2
 - maximum: 4
 - default: \$CPU_COUNT x 2
 - environment variable: WORKER_COUNT_SECSCAN

1.3.4.5. Environment variables

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

Table 1.1. Worker count environment variables

Variable	Description	Values
WORKER_COUNT_REGISTRY	Specifies the number of processes to handle Registry requests within the Quay container.	Integer between 8 and 64
WORKER_COUNT_WEB	Specifies the number of processes to handle UI/Web requests within the container.	Integer between 2 and 32
WORKER_COUNT_SECSCAN	Specifies the number of processes to handle Security Scanning (e.g. Clair) integration within the container.	Integer between 2 and 4
DB_CONNECTION_POOLING	Toggle database connection pooling. In 3.4, it is disabled by default.	"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 that 2k database connection limit, you need to take additional steps to deal with the problem. In this case 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:v3.6.8 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:v3.6.8 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
```

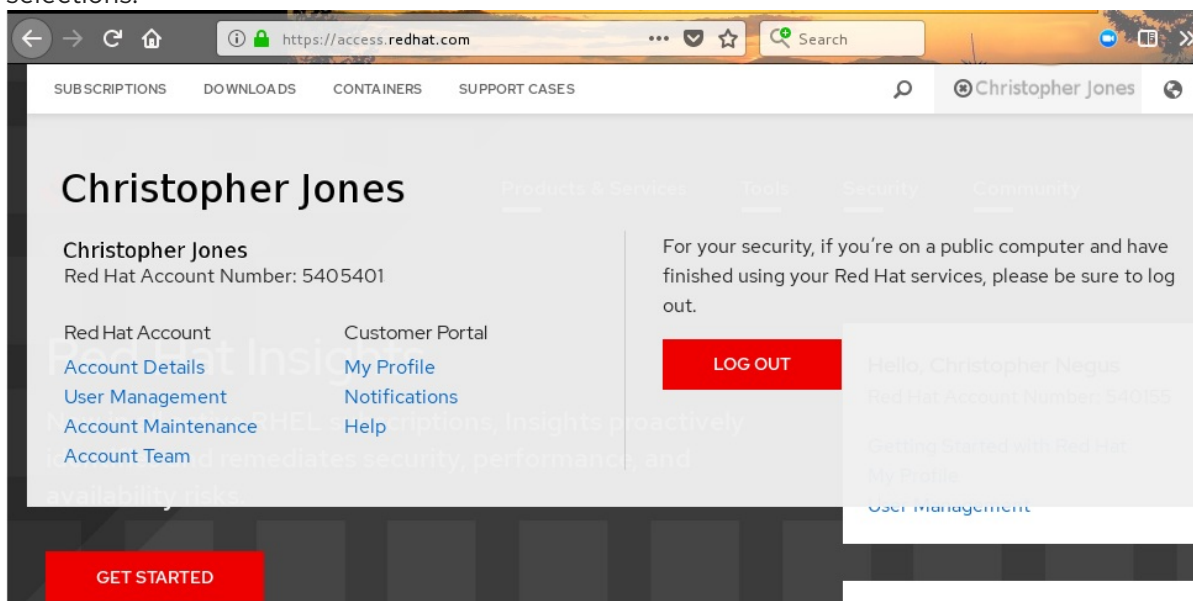
The value returned is an array indicating which fields are required:

```
[
  {
    "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"
  }
]
```

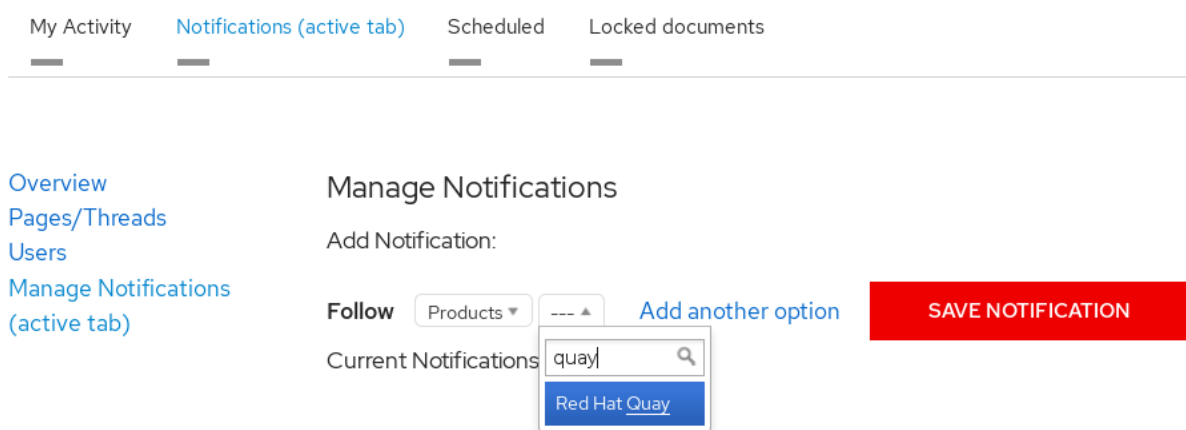
CHAPTER 3. GETTING RED HAT QUAY RELEASE NOTIFICATIONS

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
```

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. 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
```

5. 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.

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

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

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

config.yaml

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

3. Stop the **Quay** container and restart the registry:

```
$ 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:v3.6.8
```

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:v3.6.8 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:v3.6.8
```

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:

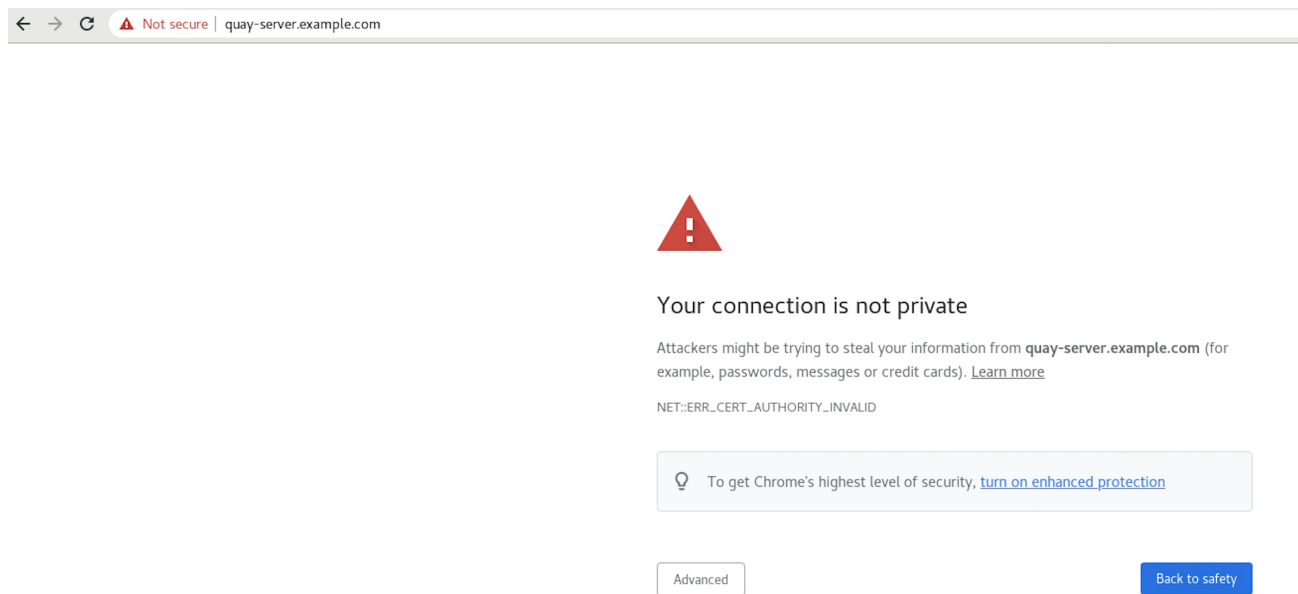
```
$ sudo podman login --tls-verify=false quay-server.example.com
Username: quayadmin
Password:
```

```
Login Succeeded!
```

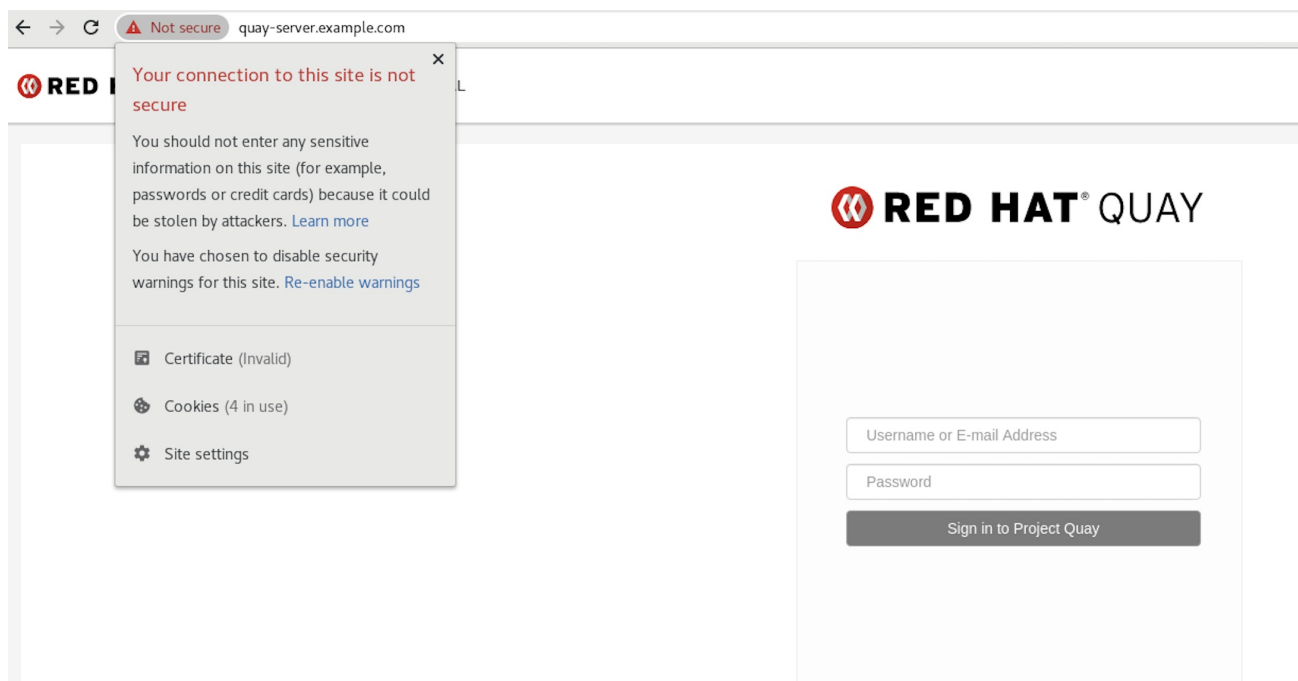
Configuring Podman to trust the root Certificate Authority (CA) is covered in a subsequent section.

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/
```

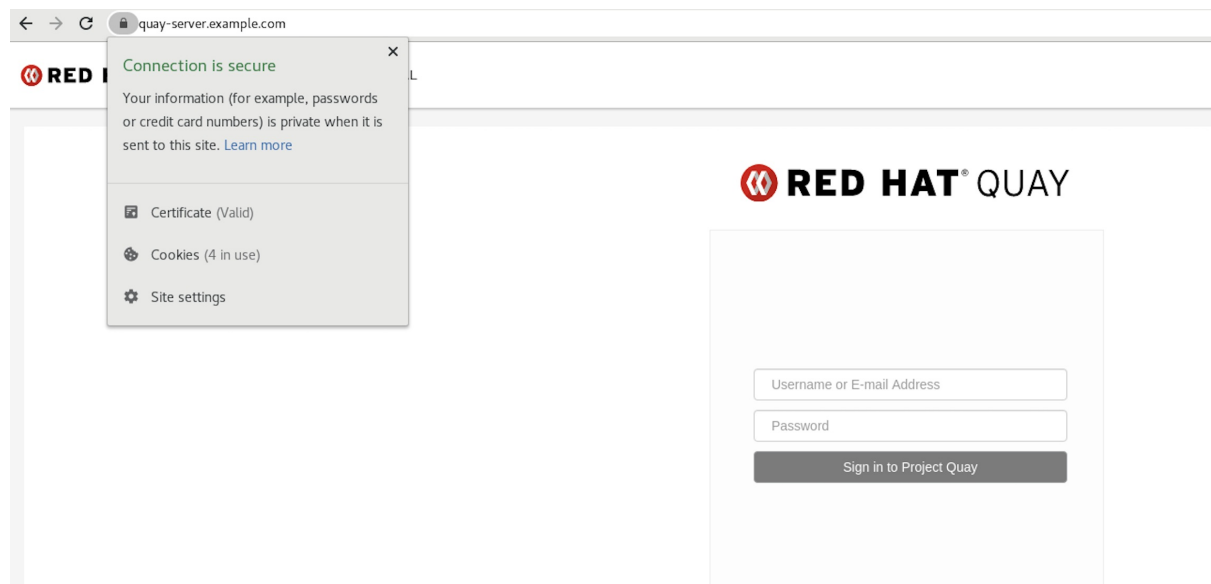
2. Update the system-wide trust store configuration:

```
$ 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:v3.6.8 "/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-----
MIIDljCCAn6gAwIBAgIBATANBgkqhkiG9w00BAQsFADA5MRcwFQYDVQQKDA5MQUlu
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".

A red rectangular box highlights the "Logs Producer" dropdown and the "Stream name", "AWS access key", "AWS secret key", and "AWS region" text input fields. Below the form, a yellow bar with a downward arrow icon contains 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:v3.6.8 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:v3.6.8

```

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

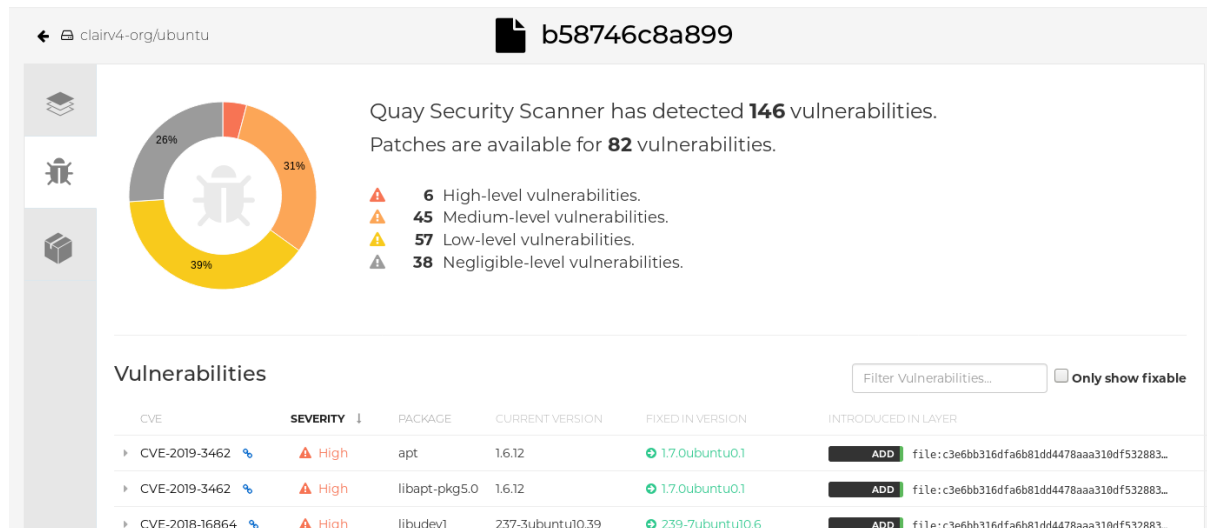
Running multiple Clair containers in this fashion is also possible, but for deployment scenarios beyond a single container the use of a container orchestrator like Kubernetes or OpenShift is strongly recommended.

7.3. USING CLAIR

1. Log in to your Red Hat Quay cluster and select an organization for which you have configured Clair scanning.
2. 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

3. 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:



7.4. 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-110n.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.

7.5. 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
```

- 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:/cfg:Z -v /path/to/output/directory:/updaters:Z --
entrypoint /bin/clairctl registry.redhat.io/quay/clair-rhel8:v3.6.8 --config /cfg/config.yaml
export-updaters /updaters/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:v3.6.8 --config /cfg/config.yaml
import-updaters /updaters/updaters.gz
```

7.6. 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/>

7.7. 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. SCAN POD IMAGES WITH THE CONTAINER SECURITY OPERATOR

Using the [Container Security Operator](#), (CSO) you can scan container images associated with active pods, running on OpenShift (4.2 or later) and other Kubernetes platforms, for known vulnerabilities. The CSO:

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

Using the instructions here, the CSO is installed in the **marketplace-operators** namespace, so it is available to all namespaces on your OpenShift cluster.



NOTE

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

8.1. RUN THE CSO IN OPENSIFT

To start using the CSO in OpenShift, do the following:

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
```

5. 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:

Dashboards

Overview

Details [View settings](#)

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

Cluster ID
3ff167bb-9f87-40a7-9a46-6cc764a38a03
[OpenShift Cluster Manager](#)

Provider
AWS

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

Update Channel

Status

Cluster ✓ Control Plane ✓ Quay Image Security ! 1 vulnerabilities

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

Cluster Utilization

Resource | Usage | 10:20 | 10:25 | 10:30 | 10:35 | 10:35

Quay Image Security breakdown

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

Severity Fixable
1 High 1 !

1 total

Fixable Vulnerabilities
1 nss-tools 1 namespaces

6. 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:

RED HAT Quay.io EXPLORE APPLICATIONS REPOSITORIES TUTORIAL

54fd70e06e7

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

6 High-level vulnerabilities.

Vulnerabilities

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

- 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	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. INTEGRATE RED HAT QUAY INTO OPENSIFT WITH THE BRIDGE OPERATOR

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

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

- Synchronizing OpenShift namespaces as Red Hat Quay organizations.
 - Creating Robot accounts for each default namespace service account
 - Creating Secrets for each created Robot Account (associating each Robot Secret to a Service Account as Mountable and Image Pull Secret)
 - Synchronizing OpenShift ImageStreams as Quay Repositories
- Automatically rewriting new Builds making use of ImageStreams to output to Red Hat Quay
- Automatically importing an ImageStream tag once a build completes

Using this procedure with the Quay Bridge Operator, you enable bi-directional communication between your Red Hat Quay and OpenShift clusters.

9.1. RUNNING THE QUAY BRIDGE OPERATOR

9.1.1. Prerequisites

Before setting up the Bridge Operator, have the following in place:

- An existing Red Hat Quay environment for which you have superuser permissions
- A Red Hat OpenShift Container Platform environment (4.2 or later is recommended) for which you have cluster administrator permissions
- An OpenShift command line tool (**oc** command)

9.1.2. Setting up and configuring OpenShift and Red Hat Quay

Both Red Hat Quay and OpenShift configuration is required:

9.1.3. Red Hat Quay setup

Create a dedicated Red Hat Quay organization, and from a new application you create within that organization, generate an OAuth token to be used with the Quay Bridge Operator in OpenShift

1. Log in to Red Hat Quay as a user with superuser access and select the organization for which the external application will be configured.
2. In the left navigation, select Applications.

3. Select **Create New Application** and entering a name for the new application (for example, **openshift**).
4. With the new application displayed, select it.
5. In the left navigation, select **Generate Token** to create a new OAuth2 token.
6. Select all checkboxes to grant the access needed for the integration.
7. Review the assigned permissions and then select **Authorize Application**, then confirm it.
8. Copy and save the generated Access Token that appears to use in the next section.

9.1.4. OpenShift Setup

Setting up OpenShift for the Quay Bridge Operator requires several steps, including:

9.1.4.1. Deploying the Operator

The fastest method for deploying the operator is to deploy from OperatorHub. From the Administrator perspective in the OpenShift Web Console, navigate to the Operators tab, and then select OperatorHub.

Search for Quay Bridge Operator and then select Install.

Select an Approval Strategy and then select Install which will deploy the operator to the cluster.

9.1.4.2. Creating an OpenShift secret for the OAuth token

The Operator will use the previously obtained Access Token to communicate with Quay. Store this token within OpenShift as a secret.

Execute the following command to create a secret called **quay-integration** in the **openshift-operators** namespace with a key called **token** containing the access token:

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

9.1.4.3. Create the QuayIntegration Custom Resource

Finally, to complete the integration between OpenShift and Quay, a **QuayIntegration** custom resource needs to be created. This can be completed in the Web Console or from the command line.

quay-integration.yaml

```
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 optional 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 QUAY_URL with the hostname of your Red Hat Quay instance.
- 4** If Quay is using self signed certificates, set the property **insecureRegistry: true**.

Create the **QuayIntegration** Custom Resource:

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

At this point a Quay integration resource is created, linking the OpenShift cluster to the Red Hat Quay instance. Organizations within Quay should be created for the related namespaces from the OpenShift environment

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 Quay API

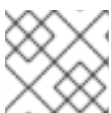
10.2. REPOSITORY MIRRORING VERSUS GEO-REPLICATION

Quay geo-replication mirrors the entire image storage backend data between 2 or more different storage backends while the database is shared (one 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 registry is separate database and 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 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, by default	Yes
Can users apply filters to synchronization rules?	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 not push images to the repository.
- Changing settings on your mirrored repository is done from the Mirrors tab on the Repositories page 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
REPO_MIRROR_TLS_VERIFY	Boolean	Require HTTPS and verify certificates of Quay registry during mirror. Default: false

10.6. MIRRORING WORKER

- To run the repository mirroring worker, start by running a **Quay** pod with the **repomirror** option:

```
$ sudo podman run -d --name mirroring-worker \
-v $QUAY/config:/conf/stack:Z \
registry.redhat.io/quay/quay-rhel8:v3.6.8 repomirror
```

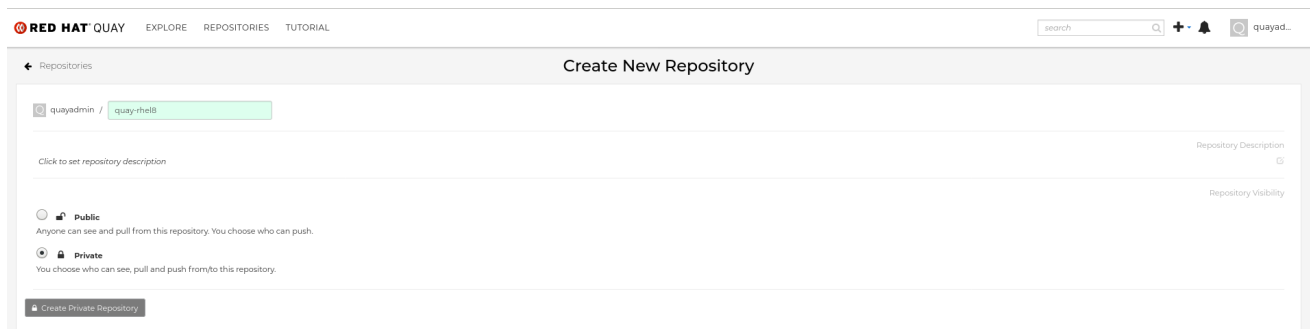
- If you have configured TLS communications using a certificate **/root/ca.crt**, then the following example shows how to start the mirroring worker:

```
$ 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 \
registry.redhat.io/quay/quay-rhel8:v3.6.8 repomirror
```

10.7. CREATING A MIRRORED REPOSITORY

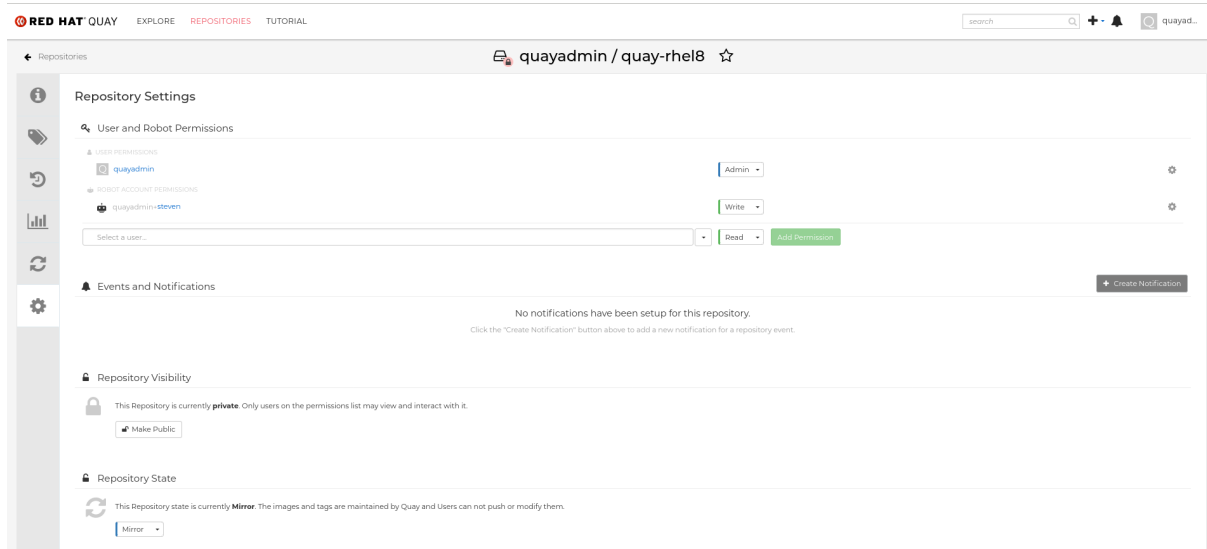
The steps shown in this section assume you already have enabled repository mirroring in the configuration for your Red Hat Quay cluster and that you have a deployed a mirroring worker.

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

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



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

The screenshot shows the 'Repository Mirroring' configuration page in Red Hat Quay. The page is titled 'Repository Mirroring' and includes a warning: 'This feature will convert quayadmin/quay-rhel8 into a mirror. Changes to the external repository will be duplicated here. While enabled, users will be unable to push images to this repository.' The configuration is divided into several sections:

- External Repository:** A section with a warning icon.
- Registry Location:** A text input field containing 'quay.io/redhat/quay'.
- Tags:** A text input field with a warning icon and a note: 'comma-separated list of tag patterns to synchronize'. An example 'latest, 3.2*' is provided.
- Start Date:** A date and time picker set to 'May 27, 2021 4:17 PM'.
- Sync Interval:** A dropdown menu set to 'seconds'.
- Robot User:** A dropdown menu with the option 'Select a user...'.
- Credentials:** A section with a warning icon and a note: 'Required if the external repository is private'. It includes fields for 'Username' and 'Password'.
- Advanced Settings:** A section with a warning icon containing:
 - Verify TLS:** A checkbox labeled 'Require HTTPS and verify certificates when talking to the external registry'.
 - HTTP Proxy:** A text input field containing 'http://example.com'.
 - HTTPs Proxy:** A text input field containing 'https://example.com'.
 - No Proxy:** A text input field containing 'example.com'.

- 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 (\).

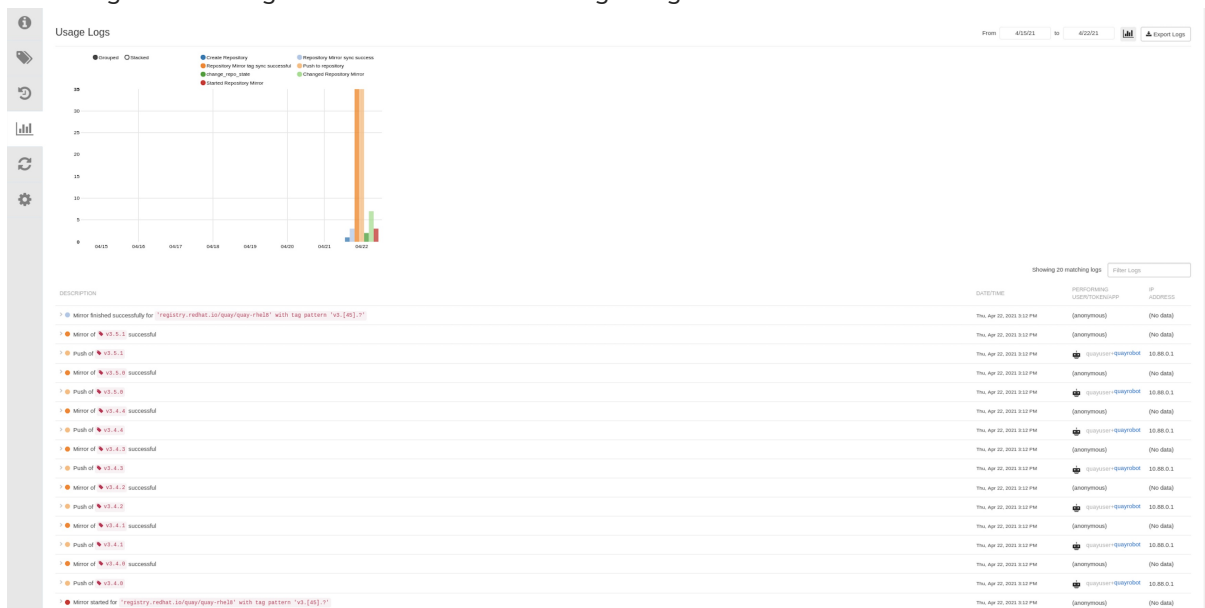
10.7.2. Advanced settings

- 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:

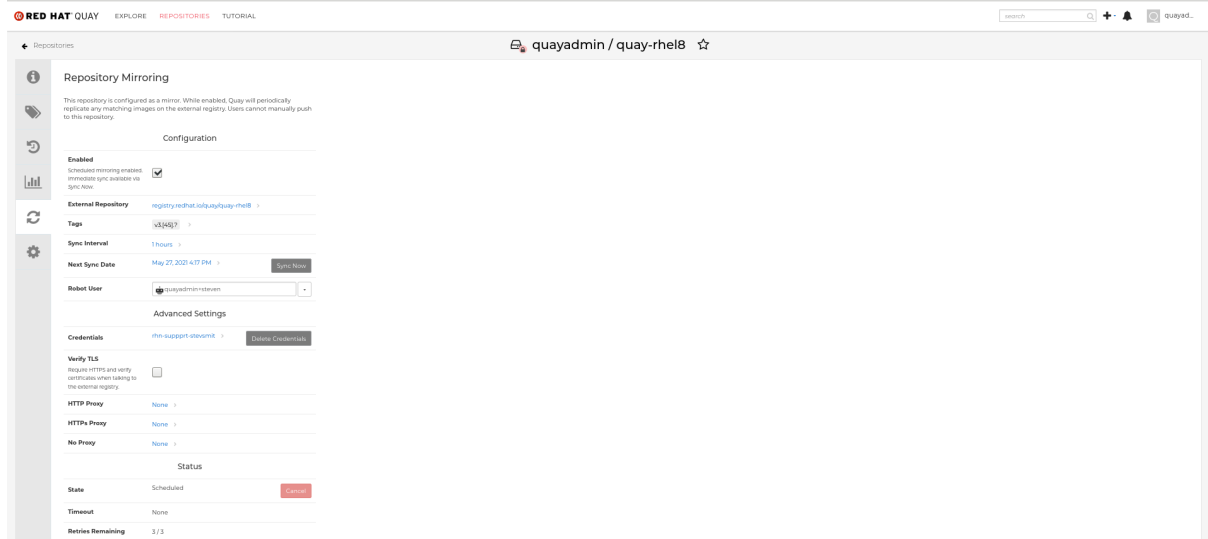


When the mirroring is complete, the images will appear in the Tags tab:

The screenshot shows the 'Repository Tags' interface for the repository 'quayuser / quay-rhel8'. It displays a list of tags with columns for 'TAG', 'LAST MODIFIED', 'SIZE', 'EXPIRES', and 'MANIFEST'. 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, all of which were last modified 'a minute ago' and have a size of 'N/A'.

TAG	LAST MODIFIED	SIZE	EXPIRES	MANIFEST
<input type="checkbox"/> v3.5.1	a minute ago	N/A	Never	SHA256: 3d4d7f2d75
<input type="checkbox"/> v3.5.0	a minute ago	N/A	Never	SHA256: 9559f6e05c
<input type="checkbox"/> v3.4.4	a minute ago	N/A	Never	SHA256: e43236d29e
<input type="checkbox"/> v3.4.3	a minute ago	N/A	Never	SHA256: 36c7110912
<input type="checkbox"/> v3.4.2	a minute ago	N/A	Never	SHA256: 768b18ca07
<input type="checkbox"/> v3.4.1	a minute ago	N/A	Never	SHA256: 6f85c53d0b
<input type="checkbox"/> v3.4.0	a minute ago	N/A	Never	SHA256: 5ec5d88803

Below is an example of a completed Repository Mirroring screen:



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*	v32, 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:

← Repositories johnjones / ubi7repo ☆

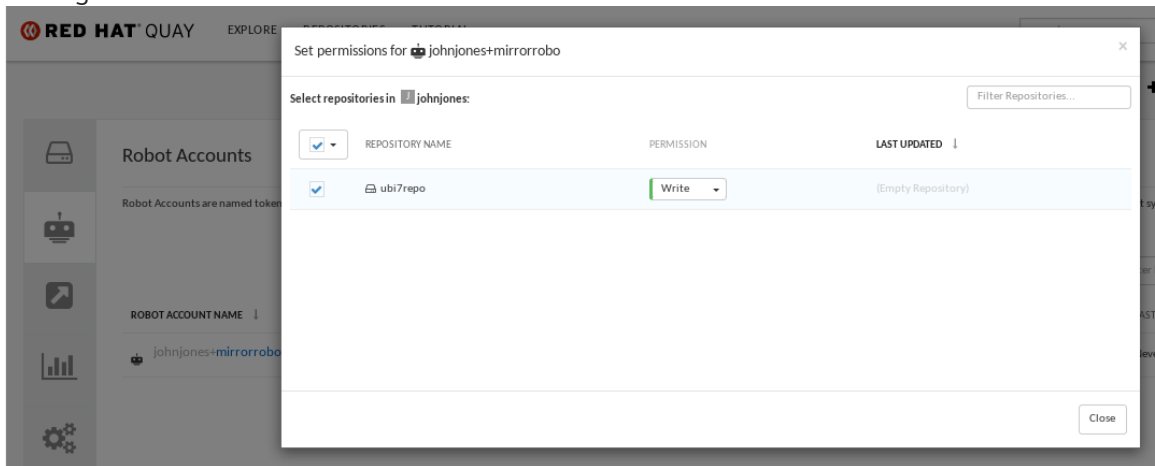
Usage Logs From 8/9/19 to 8/16/19

● Grouped ○ Stacked ● Changed Repository Mirror ● Repository Mirror sync success ● Create Repository ● Started Repository Mirror

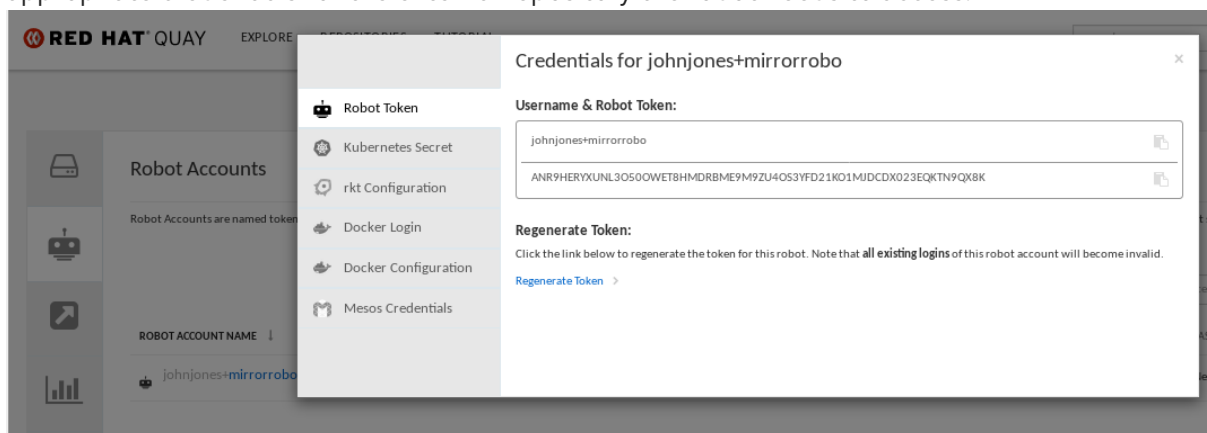
Showing 6 matching logs

DESCRIPTION	DATE/TIME	PERFORMING USER/TOKEN/APP	IP ADDRESS
● Mirror finished successfully for 'registry.access.redhat.com/ubi7/ubi-minimal' with tag pattern 'latest'	Fri, Aug 16, 2019 10:49 AM	(anonymous)	(No data)
● Mirror started for 'registry.access.redhat.com/ubi7/ubi-minimal' with tag pattern 'latest'	Fri, Aug 16, 2019 10:49 AM	(anonymous)	(No data)
● Immediate mirror scheduled	Fri, Aug 16, 2019 10:48 AM	quay	(No data)
● Mirror finished successfully for 'registry.access.redhat.com/ubi7/ubi-minimal' with tag pattern 'latest'	Fri, Aug 16, 2019 10:18 AM	(anonymous)	(No data)
● Mirror started for 'registry.access.redhat.com/ubi7/ubi-minimal' with tag pattern 'latest'	Fri, Aug 16, 2019 10:18 AM	(anonymous)	(No data)
● CreateRepository quay / ubi7minimal	Fri, Aug 16, 2019 10:01 AM	quay	(No data)

- **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

- Repository mirroring pods can run on any node including other nodes where Quay is already running
- Repository mirroring is scheduled in the database and run in batches. As a result, more workers could 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 if required.

CHAPTER 11. BACKING UP AND RESTORING RED HAT QUAY ON AN OPENSIFT CONTAINER PLATFORM DEPLOYMENT

Use the content within this section to back up and restore Red Hat Quay on an OpenShift Container Platform deployment.

11.1. BACKING UP RED HAT QUAY

This procedure is exclusively for OpenShift Container Platform and NooBaa deployments.

Prerequisites

- A Red Hat Quay deployment on OpenShift Container Platform.

Procedure

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 QuayRegistry namespace.

```
$ 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 all owner references. 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
```

7. Scale down the Quay the Quay Operator:

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

8. Scale down the Quay namespace:

```
$ oc scale --replicas=0 deployment $(oc get deployment -n <quay-namespace> -l quay-
component=quay -o jsonpath='{.items[0].metadata.name}') -n <quay-namespace>
```

9. Wait for the **registry-quay-app** pods 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-mirror-758fc68ff7-5wxlp 1/1 Running 0 8m29s
registry-quay-mirror-758fc68ff7-lbl82 1/1 Running 0 8m29s
registry-quay-redis-7cc5f6c977-956g8 1/1 Running 0 5d21h
```

10. 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
```

1. 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
```

2. Download a backup database:

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

3. 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)
```

4. 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)
```

5. 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.

1. Scale up the Quay the Quay Operator:

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

2. Scale up the Quay namespace:

```
$ oc scale --replicas=1 deployment $(oc get deployment -n <quay-namespace> -l quay-
component=quay -o jsonpath='{.items[0].metadata.name}') -n <quay-namespace>
```

3. Check the status of the Operator:

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

Example output:

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

```

name: example-registry
namespace: <quay-namespace>
...
spec:
  components:
  - kind: quay
    managed: true
  ...
  - kind: clairpostgres
    managed: true
  configBundleSecret: init-config-bundle-secret
status:
  configEditorCredentialsSecret: example-registry-quay-config-editor-credentials-fg2gdgtm24
  configEditorEndpoint: https://example-registry-quay-config-editor-quay-
enterprise.apps.docs.gcp.quaydev.org
  currentVersion: 3.7.0
  lastUpdated: 2022-05-11 13:28:38.199476938 +0000 UTC
  registryEndpoint: https://example-registry-quay-quay-enterprise.apps.docs.gcp.quaydev.org
0      5d21h

```

11.2. RESTORING RED HAT QUAY

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

Prerequisites

- Red Hat Quay is deployed on OpenShift Container Platform using the Quay Operator.
- Your Red Hat Quay database has been backed up.

Procedure

1. Restore the backed up Quay configuration and the randomly generated keys:

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

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



NOTE

If you receive the error **Error from server (AlreadyExists): error when creating "/>**

2. Restore the QuayRegistry custom resource:

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

3. Scale down the Quay the Quay Operator:

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

- Scale down the Quay namespace:

```
$ oc scale --replicas=0 deployment $(oc get deployment -n <quay-namespace> -l quay-
component=quay -o jsonpath='{.items[0].metadata.name}') -n <quay-namespace>
```

- 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
```

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

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

- Open a remote terminal to the database:

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

- Enter psql:

```
bash-4.4$ psql
```

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

```
postgres=# \l
```

Example output:

```

                                List of databases
   Name          | Owner          | Encoding | Collate  | Ctype    | Access
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 |
```

- Drop the database:

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

Example output:

```
DROP DATABASE
```

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

```
\q
```

- 12. Redirect your PostgreSQL database to your backup database:

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

- 13. Exit bash:

```
sh-4.4$ exit
```

- 14. 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)
```

- 15. 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)
```

- 16. 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}')
```

- 17. Scale up the Quay the Quay Operator:

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

- 18. Scale up the Quay namespace:

```
$ oc scale --replicas=1 deployment $(oc get deployment -n <quay-namespace> -l quay-component=quay -o jsonpath='{.items[0].metadata.name}') -n <quay-namespace>
```

- 19. Check the status of the Operator and ensure it has come back online:

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

Example output:

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

```
spec:
  components:
  - kind: quay
    managed: true
  ...
  - kind: clairpostgres
    managed: true
  configBundleSecret: init-config-bundle-secret
status:
  configEditorCredentialsSecret: example-registry-quay-config-editor-credentials-fg2gdgtm24
  configEditorEndpoint: https://example-registry-quay-config-editor-quay-
enterprise.apps.docs.gcp.quaydev.org
  currentVersion: 3.7.0
  lastUpdated: 2022-05-11 13:28:38.199476938 +0000 UTC
  registryEndpoint: https://example-registry-quay-quay-enterprise.apps.docs.gcp.quaydev.org
    0      5d21h
```

CHAPTER 12. 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.

12.1. CONSIDERATIONS PRIOR TO ENABLING LDAP

12.1.1. 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.

12.1.2. 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.

12.2. 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
```

12.2.1. 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
```

12.2.2. 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
```

12.2.3. 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:

- `ou=SFO` [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
```

12.2.4. 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)
```

12.2.5. Administrator DN


Administrator DN:

cn=quayenterprise,ou=svc,ou=NYC,dc=example,dc=org

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
```

12.2.6. UID and Mail attributes

UID Attribute:

uid

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

Mail Attribute:

mail

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
```

12.2.7. 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.

12.3. 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.

12.4. 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 13. 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.

13.1. EXPOSING THE PROMETHEUS ENDPOINT

13.1.1. 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:v3.6.8
```

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.

13.1.2. 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
...
```

13.1.3. 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.

13.1.4. DNS configuration under Kubernetes

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

13.1.5. 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.

13.2. 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.

13.2.1. 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

```

13.2.2. 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
```

13.2.3. 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
...
```

13.2.3.1. 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 completed
quay_multipart_uploads_completed_total	Number of multipart uploads to Quay storage that started

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
...
```

13.2.4. Image push / pull metrics

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

13.2.4.1. 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

13.2.4.2. 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)

13.2.4.3. 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

13.2.4.4. 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
...
```

13.2.5. 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 14. GEO-REPLICATION

Geo-replication allows multiple, geographically distributed 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 Quay setup. Image data is asynchronously replicated in the background with transparent failover / redirect for clients.



NOTE

Deploying Red Hat Quay with geo-replication on OpenShift is not supported by the Operator.

14.1. GEO-REPLICATION FEATURES

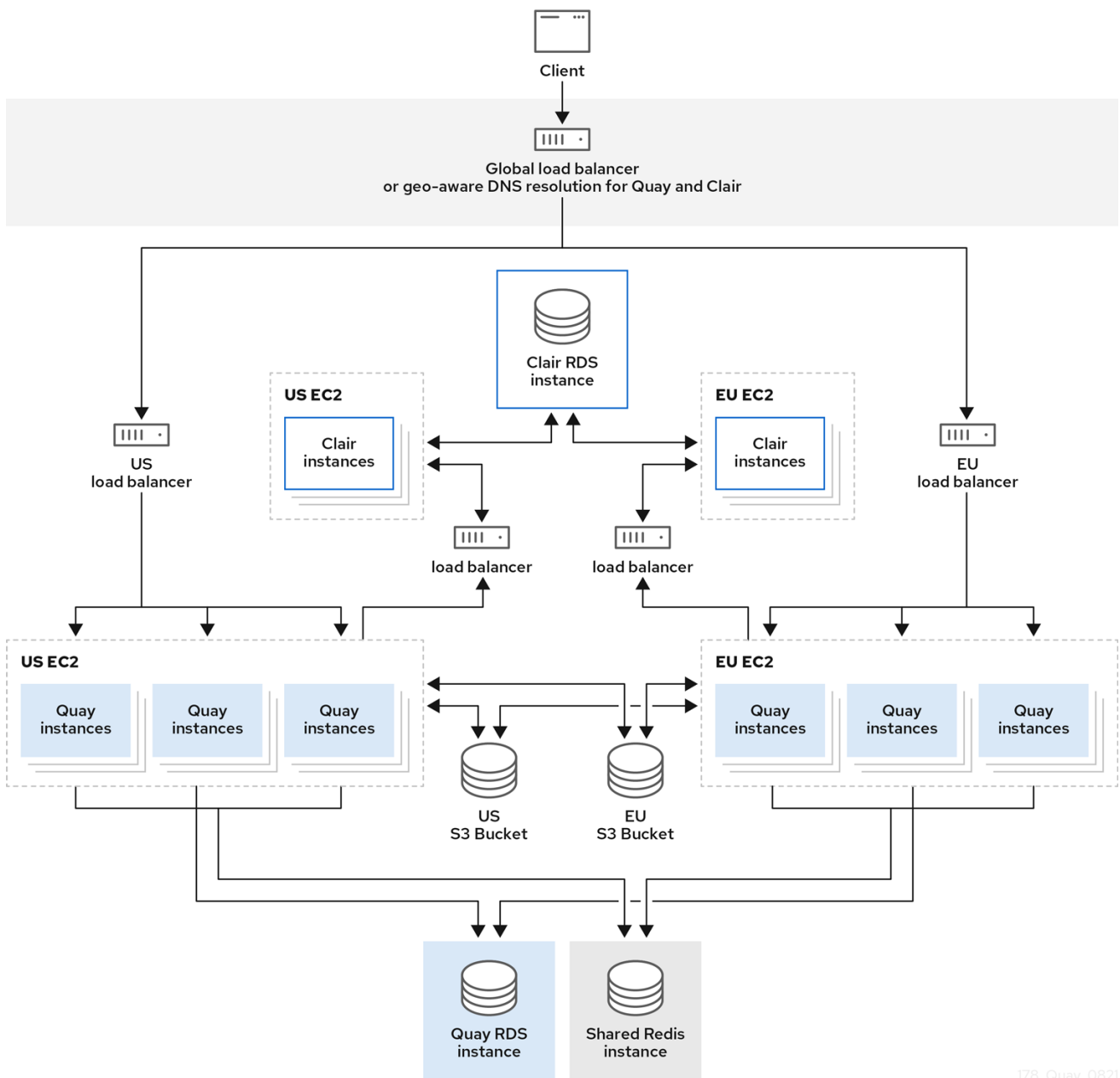
- When geo-replication is configured, container image pushes will be written to the preferred storage engine for that Red Hat Quay instance (typically the nearest storage backend within the region).
- After the initial push, image data will be replicated in the background to other storage engines.
- 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

- A single database, and therefore all metadata and Quay configuration, is shared across all regions.
- 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 (all blobs) is replicated. This is in contrast to repository mirroring, which can be limited to an organization or repository or image.
- All Quay instances must share the same endpoint, typically via load balancer.
- All Quay instances must have the same set of superusers, as they are defined inside the common configuration file.

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 ARCHITECTURE



178_Quay_0821

In the example shown above, Quay is running 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.

14.4. ENABLE STORAGE REPLICATION

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.

14.4.1. 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:v3.6.8
```



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

CHAPTER 15. 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 16. 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