Red Hat Quay 3

Manage Red Hat Quay

Manage Red Hat Quay

Manage Red Hat Quay

Legal Notice

Abstract

Manage Red Hat Quay

Table of Contents

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:

- 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. 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:
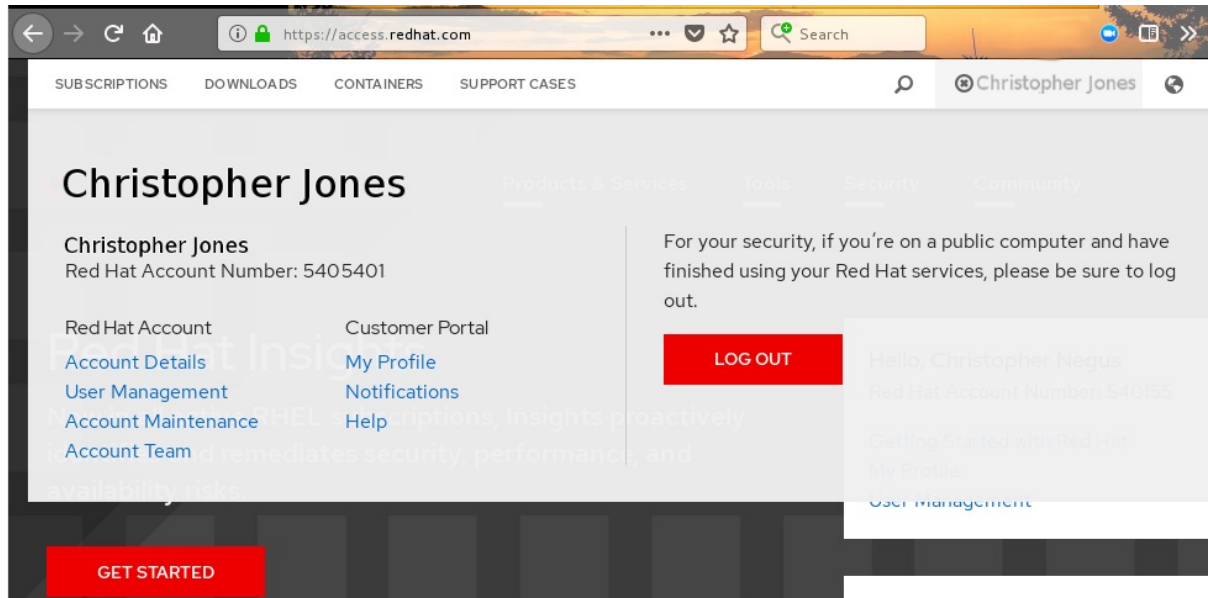
3. Select Notifications. Your profile activity page appears.

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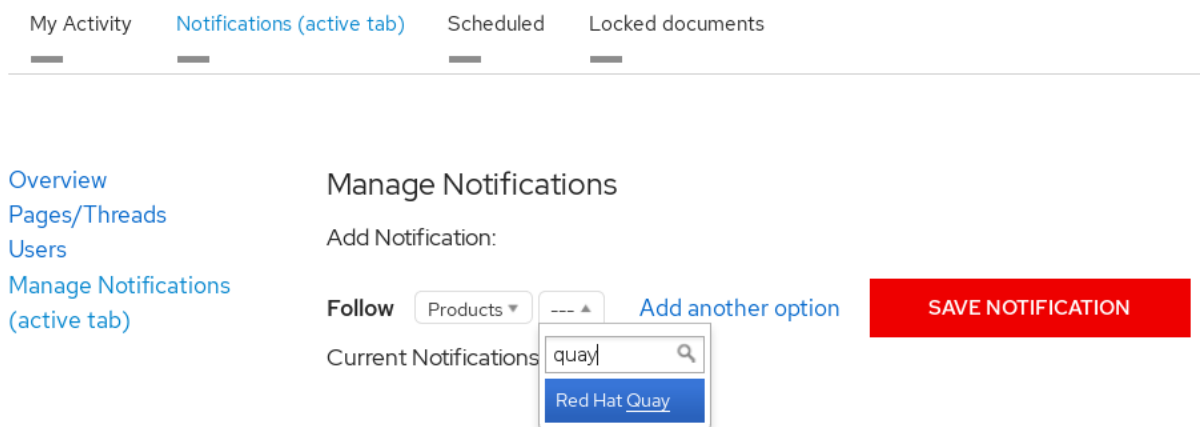
8. Select the SAVE NOTIFICATION button. Going forward, you will receive notifications when there are changes to the Red Hat Quay product, such as a new release.

CHAPTER 2. USING SSL TO PROTECT CONNECTIONS TO RED HAT QUAY

This document assumes you have deployed Red Hat Quay in a single-node or highly available deployment.

To configure Red Hat Quay with a self-signed certificate, you need to create a Certificate Authority (CA), then generate the required key and certificate files. You then enter those files using the Red Hat Quay Config Tool or command line.

2.1. CREATE A CA AND SIGN A CERTIFICATE

1. Create a root CA.

   ```
   $ openssl genrsa -out rootCA.key 2048
   $ openssl req -x509 -new -nodes -key rootCA.key -sha256 -days 1024 -out rootCA.pem
   ```

   The result are rootCA.key and rootCA.pem files in the current directory.

2. **Create certificate and private key.** If you are having Red Hat Quay handle TLS, you need to create a certificate and private key to provide during configuration. You can get those files from a certificate signing authority. Here we show how to create those files using the self-signed certificate authority you just created.
   In this example, you create device.crt and device.key files, which will be uploaded to Red Hat Quay and renamed ssl.cert and ssl.key, respectively.

   Because OpenShift creates long fully qualified domain names, consider using a wildcard to identify the larger domain, instead of the specific route to the Red Hat Quay application. For example, use something like *.apps.openshift.example.com when prompted for the server's hostname:

   ```
   Common Name (eg, your name or your server's hostname) []:*apps.openshift.example.com
   ```

   ```
   $ openssl genrsa -out device.key 2048
   $ openssl req -new -key device.key -out device.csr
   ```

   Then sign the certificate with the root CA created earlier:

   ```
   $ openssl x509 -req -in device.csr -CA rootCA.pem \
        -CAkey rootCA.key -CAcreateserial -out device.crt -days 500 -sha256
   ```

NOTE

Instead generating the *.key and *.crt files as just shown, you could create an **openssl.cnf** file. This lets you add more information to the resulting certificate than you can get by just responding to the prompts in the command for generating the certificate request. In this example of an **openssl.cnf** file, replace **DNS.1** and **IP.1** with the hostname and IP address of the Red Hat Quay server:

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 = reg.example.com
IP.1 = 12.345.678.9
```

You could then generate the key as follows:

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

2.2. CONFIGURE RED HAT QUAY TO USE THE NEW CERTIFICATE

The next step can be accomplished either in the Red Hat Quay screen or from the terminal.

2.2.1. Configure SSL from the Red Hat Quay Setup screen

Start the quay container in config mode, as described in each deployment guide. In the server Configuration section, enable SSL as follows:

1. Set the **Server Hostname** to the appropriate value and check the **Enable SSL** box, then upload the **ssl.key** and **ssl.cert** files (in our example, named **device.key** and **device.crt**, respectively):

☁ Server Configuration

Server Hostname:	reg.example.com
	The HTTP host (and optionally the port number if a non-standard HTTP/HTTPS port) of the location where the registry will be accessible on the network
SSL:	☑ **Enable SSL**
	A valid SSL certificate and private key files are required to use this option.

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

Certificate: /conf/stack/ssl.cert   Select a replacement file:
Choose File ssl.cert
The certificate must be in PEM format.

Private key: /conf/stack/ssl.key   Select a replacement file:
Choose File ssl.key

2. Save the configuration. Red Hat Quay will automatically validate the SSL certificate:

Checking your settings

✔ REDIS
✔ REGISTRY STORAGE
✔ SSL CERTIFICATE AND KEY

✔ Configuration Validated   ⬆ Save Configuration

3. Restart the container

⚠ **Container restart required!**   ⟳ Restart Now
Configuration changes have been made but the container hasn't been restarted yet.

2.2.2. Configure with the command line

By not using the web interface the configuration checking mechanism built into Red Hat Quay is unavailable. It is suggested to use the web interface if possible. For non-OpenShift installations, you can configure SSL from the command-line interface as follows:

1. Copy the **ssl.key** and **ssl.cert** into the specified **config** directory. In this example, the config directory for Red Hat Quay is on a host named reg.example.com in a directory named /mnt/quay/config.
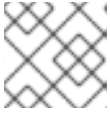
NOTE

The certificate/key files must be named ssl.key and ssl.cert.

```
$ ls
ssl.cert  ssl.key
$ scp ssl.* root@reg.example.com:/mnt/quay/config/
[root@reg.example.com ~]$ ls /mnt/quay/config/
config.yaml  ssl.cert  ssl.key
```

2. Modify the **PREFERRED_URL_SCHEME:** parameter in config.yaml from **http** to **https**

```
PREFERRED_URL_SCHEME: https
```

3. Restart the Red Hat Quay container:

```
$ docker ps
CONTAINER ID  IMAGE                COMMAND             CREATED     STATUS
PORTS             NAMES
eaf45a4aa12d  ...redhat.com/rhscl/redis  "/usr/bin/redis-serve" 22 hours ago  Up 22 hours
0.0.0.0:6379->6379/tcp  dreamy...
cbe7b0fa39d8  quay.io/redhat/quay       "/sbin/my_init"       22 hours ago  Up one hour
80/tcp,443/tcp,443/tcp  ferv...
705fe7311940  mysql:5.7                "/entrypoint.sh mysql"  23 hours ago  Up 22 hours
0.0.0.0:3306->3306/tcp  mysql
$ docker restart cbe7b0fa39d8
```

2.2.3. Test the secure connection

Confirm the configuration by visiting the URL from a browser **https://reg.example.com/**
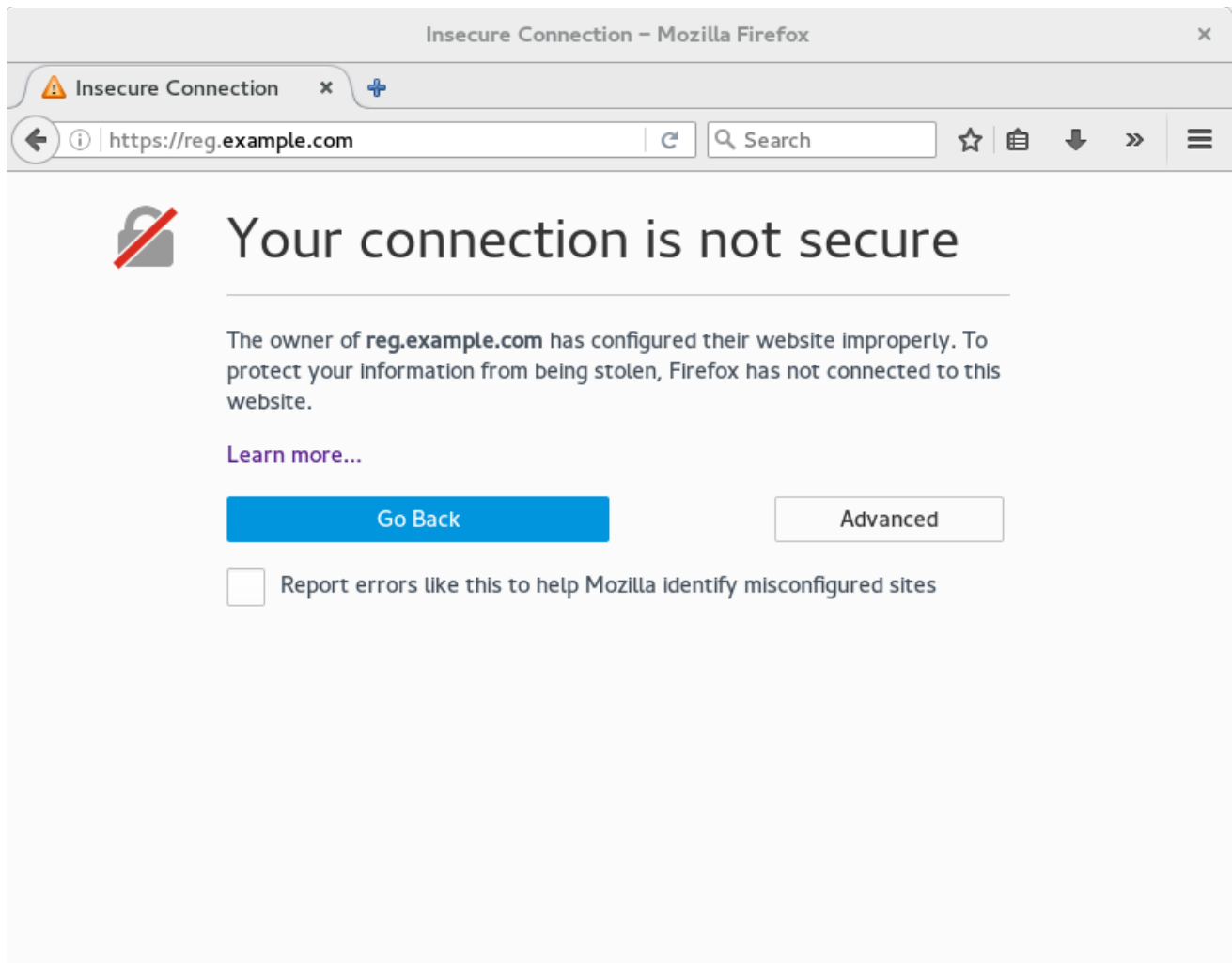
"Your Connection is not secure" means the CA is untrusted but confirms that SSL is functioning properly. To avoid these messages, you need to get a certificate from a trusted certificate authority.

2.3. CONFIGURING DOCKER TO TRUST A CERTIFICATE AUTHORITY

Docker requires that custom certs be installed to **/etc/docker/certs.d/** under a directory with the same name as the hostname private registry. It is also required for the cert to be called **ca.crt**. Here is how to do that:

1. Copy the rootCA file.

   ```
   $ cp tmp/rootCA.pem /etc/docker/certs.d/reg.example.com/ca.crt
   ```

2. After you have copied the rootCA.pem file, **docker login** should authenticate successfully and pushing to the repository should succeed.

   ```
   $ sudo docker push reg.example.com/kbrwn/hello
   The push refers to a repository [reg.example.com/kbrwn/hello]
   5f70bf18a086: Layer already exists
   e493e9cb9dac: Pushed
   1770dbc4af14: Pushed
   a7bb4eb71da7: Pushed
   9fad7adcbd46: Pushed
   2cec07a74a9f: Pushed
   f342e0a3e445: Pushed
   b12f995330bb: Pushed
   2016366cdd69: Pushed
   ```

```
a930437ab3a5: Pushed
15eb0f73cd14: Pushed
latest: digest:
sha256:c24be6d92b0a4e2bb8a8cc7c9bd044278d6abdf31534729b1660a485b1cd315c size:
7864
```

CHAPTER 3. 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.

3.1. 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 **docker ps**:

```
$ docker ps
CONTAINER ID        IMAGE                          COMMAND           CREATED
STATUS              PORTS
5a3e82c4a75f        <registry>/<repo>/quay:v3.3.1 "/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:

```
$ docker restart 5a3e82c4a75f
```

5. Examine the certificate copied into the container namespace:

```
$ docker exec -it 5a3e82c4a75f cat /etc/ssl/certs/storage.pem
-----BEGIN CERTIFICATE-----
MIIDTTCCAjWgAwIBAgIJAMVr9ngjJhzbMA0GCSqGSIb3DQEBCwUAMD0xCzAJBgNV
```

3.2. 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-----
   MIIDljCCAn6gAwIBAgIBATANBgkqhkiG9w0BAQsFADA5MRcwFQYDVQQKDA5MQUIu
   TElCQ09SRS5TTzEeMBwGA1UEAwwVQ2VydGlmaWNhdGUgQXV0aG9yaXR5MB4XDTE2
   MDExMjA2NTkxMFoXDTM2MDExMjA2NTkxMFowOTEXMBUGA1UECgwOTEFCLkxJQkNP
   UkUuU08xHjAcBgNVBAMMFUNlcnRpZmljYXRlIEF1dGhvcml0eTCCASIwDQYJKoZI
   [...]
   -----END CERTIFICATE-----

   $ cat ca.crt | base64 -w 0
   [...]
   c1psWGpqeGlPQmNEWkJPMjJ5d0pDemVnR2QNCnRsbW9JdEF4YnFSdVd3PT0KLS0tLS1F
   TkQgQ0VSVElGSUNBVEUtLS0tLQo=
   ```

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:
   c1psWGpqeGlPQmNEWkJPMjJ5d0pDemVnR2QNCnRsbW9JdEF4YnFSdVd3PT0KLS0tLS1F
   TkQgQ0VSVElGSUNBVEUtLS0tLQo=
   ```

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 4. 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:

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:

AWS region:	The AWS region
Index prefix:	logentry_
Logs Producer:	Kinesis
Stream name:	The Kinesis stream name
AWS access key:	The AWS access key
AWS secret key:	The AWS secret key
AWS region:	The AWS region

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 5. RED HAT QUAY SECURITY SCANNING WITH CLAIR

Red Hat Quay supports scanning container images for known vulnerabilities with a scanning engine such as Clair. This document explains how to configure Clair with Red Hat Quay.

5.1. SET UP CLAIR IN THE RED HAT QUAY CONFIG TOOL

Enabling Clair in Red Hat Quay consists of:

- Starting the Red Hat Quay config tool. See the Red Hat Quay deployment guide for the type of deployment you are doing (OpenShift, Basic, or HA) for how to start the config tool for that environment.

- Enabling security scanning, then generating a private key and PEM file in the config tool

- Including the key and PEM file in the Clair config file

- Start the Clair container

The procedure varies, based on whether you are running Red Hat Quay on OpenShift or directly on a host.

5.1.1. Enabling Clair on a Red Hat Quay OpenShift deployment

To set up Clair on Red Hat Quay in OpenShift, see Add Clair image scanning to Red Hat Quay .

5.1.2. Enabling Clair on a Red Hat Quay Basic or HA deployment

To set up Clair on a Red Hat Quay deployment where the container is running directly on the host system, do the following:

1. **Restart the Red Hat Quay config tool**: Run the quay container again in config mode, open the configuration UI in a browser, then select **Modify an existing configuration**. When prompted, upload the **quay-config.tar.gz** file that was originally created for the deployment.

2. **Enable Security Scanning**: Scroll to the Security Scanner section and select the "Enable Security Scanning" checkbox. From the fields that appear you need to create an authentication key and enter the security scanner endpoint. Here's how:

   - **Generate key**: Click **Create Key**, then from the pop-up window type a name for the Clair private key and an optional expiration date (if blank, the key never expires). Then select Generate Key.

   - **Copy the Clair key and PEM file**: Save the Key ID (to a notepad or similar) and download a copy of the Private Key PEM file (**named security_scanner.pem**) by selecting "Download Private Key" (if you lose the key, you need to generate a new one). You will need the key and PEM file when you start the Clair container later.
   Close the pop-up when you are done. Here is an example of a completed Security Scanner config:

3. **Save the configuration**: Click **Save Configuration Changes** and then select **Download Configuration** to save it to your local system.

4. **Deploy the configuration**: To pick up the changes enabling scanning, as well as other changes you may have made to the configuration, unpack the **quay-config.tar.gz** and copy the resulting files to the config directory. For example:

```
$ tar xvf quay-config.tar.gz
config.yaml  ssl.cert  ssl.key
$ cp config.yaml ssl* /mnt/quay/config
```

Next, start the Clair container and associated database, as described in the following sections.

CHAPTER 6. SETTING UP CLAIR SECURITY SCANNING

Once you have created the necessary key and pem files from the Red Hat Quay config UI, you are ready to start up the Clair container and associated database. Once that is done, you an restart your Red Hat Quay cluster to have those changes take effect.

Procedures for running the Clair container and associated database are different on OpenShift than they are for running those containers directly on a host.

6.1. RUN CLAIR ON A RED HAT QUAY OPENSHIFT DEPLOYMENT

To run the Clair image scanning container and its associated database on an OpenShift environment with your Red Hat Quay cluster, see Add Clair image scanning to Red Hat Quay .

6.2. RUN CLAIR ON A RED HAT QUAY BASIC OR HA DEPLOYMENT

To run Clair and its associated database on non-OpenShift environments (directly on a host), you need to:

- Start up a database

- Configure and start Clair

6.2.1. Get Postgres and Clair

In order to run Clair, a database is required. For production deployments, MySQL is not supported. For production, we recommend you use PostgreSQL or other supported database:

- Running on machines other than those running Red Hat Quay

- Ideally with automatic replication and failover

For testing purposes, a single PostgreSQL instance can be started locally:

1. To start Postgres locally, do the following:

   ```
   # docker run --name postgres -p 5432:5432 -d postgres
   # sleep 5
   # docker run --rm --link postgres:postgres postgres \
      sh -c 'echo "create database clairtest" | psql -h \
      "$POSTGRES_PORT_5432_TCP_ADDR" -p  \
      "$POSTGRES_PORT_5432_TCP_PORT" -U postgres'
   ```

   The configuration string for this test database is:

   ```
   postgresql://postgres@{DOCKER HOST GOES HERE}:5432/clairtest?sslmode=disable
   ```

2. Pull the security-enabled Clair image:

   ```
   docker pull quay.io/redhat/clair-jwt:v3.3.1
   ```

3. Make a configuration directory for Clair

```
# mkdir clair-config
# cd clair-config
```

6.2.2. Configure Clair

Clair can run either as a single instance or in high-availability mode. It is recommended to run more than a single instance of Clair, ideally in an auto-scaling group with automatic healing.

1. Create a **config.yaml** file to be used in the Clair config directory (/**clair**/**config**) from one of the two Clair configuration files shown here.

2. If you are doing a high-availability installation, go through the procedure in Authentication for high-availability scanners to create a Key ID and Private Key (PEM).

3. Save the Private Key (PEM) to a file (such as, $HOME/config/security_scanner.pem).

4. Replace the value of key_id (CLAIR_SERVICE_KEY_ID) with the Key ID you generated and the value of private_key_path with the location of the PEM file (for example, /config/security_scanner.pem).
   For example, those two value might now appear as:

   ```
   key_id: { 4fb9063a7cac00b567ee921065ed16fed7227afd806b4d67cc82de67d8c781b1 }
   private_key_path: /clair/config/security_scanner.pem
   ```

5. Change other values in the configuration file as needed.

6.2.2.1. Clair configuration: High availability

```
clair:
  database:
    type: pgsql
    options:
      # A PostgreSQL Connection string pointing to the Clair Postgres database.
      # Documentation on the format can be found at: http://www.postgresql.org/docs/9.4/static/libpq-connect.html
      source: { POSTGRES_CONNECTION_STRING }
      cachesize: 16384
  api:
    # The port at which Clair will report its health status. For example, if Clair is running at
    # https://clair.mycompany.com, the health will be reported at
    # http://clair.mycompany.com:6061/health.
    healthport: 6061

    port: 6062
    timeout: 900s

    # paginationkey can be any random set of characters. *Must be the same across all Clair instances*.
    paginationkey: "XxoPtCUzrUv4JV5dS+yQ+MdW7yLEJnRMwigVY/bpgtQ="

  updater:
    # interval defines how often Clair will check for updates from its upstream vulnerability databases.
    interval: 6h
  notifier:
```

```
    attempts: 3
    renotifyinterval: 1h
    http:
      # QUAY_ENDPOINT defines the endpoint at which Quay is running.
      # For example: https://myregistry.mycompany.com
      endpoint: { QUAY_ENDPOINT }/secscan/notify
      proxy: http://localhost:6063

jwtproxy:
  signer_proxy:
    enabled: true
    listen_addr: :6063
    ca_key_file: /certificates/mitm.key # Generated internally, do not change.
    ca_crt_file: /certificates/mitm.crt # Generated internally, do not change.
    signer:
      issuer: security_scanner
      expiration_time: 5m
      max_skew: 1m
      nonce_length: 32
      private_key:
        type: preshared
        options:
          # The ID of the service key generated for Clair. The ID is returned when setting up
          # the key in [Quay Setup](security-scanning.md)
          key_id: { CLAIR_SERVICE_KEY_ID }
          private_key_path: /clair/config/security_scanner.pem

  verifier_proxies:
  - enabled: true
    # The port at which Clair will listen.
    listen_addr: :6060

    # If Clair is to be served via TLS, uncomment these lines. See the "Running Clair under TLS"
    # section below for more information.
    # key_file: /clair/config/clair.key
    # crt_file: /clair/config/clair.crt

    verifier:
      # CLAIR_ENDPOINT is the endpoint at which this Clair will be accessible. Note that the port
      # specified here must match the listen_addr port a few lines above this.
      # Example: https://myclair.mycompany.com:6060
      audience: { CLAIR_ENDPOINT }

      upstream: http://localhost:6062
      key_server:
        type: keyregistry
        options:
          # QUAY_ENDPOINT defines the endpoint at which Quay is running.
          # Example: https://myregistry.mycompany.com
          registry: { QUAY_ENDPOINT }/keys/
```

6.2.2.2. Clair configuration: Single instance

```
clair:
  database:
```

```
    type: pgsql
    options:
      # A PostgreSQL Connection string pointing to the Clair Postgres database.
      # Documentation on the format can be found at: http://www.postgresql.org/docs/9.4/static/libpq-
connect.html
      source: { POSTGRES_CONNECTION_STRING }
      cachesize: 16384
  api:
    # The port at which Clair will report its health status. For example, if Clair is running at
    # https://clair.mycompany.com, the health will be reported at
    # http://clair.mycompany.com:6061/health.
    healthport: 6061

    port: 6062
    timeout: 900s

    # paginationkey can be any random set of characters. *Must be the same across all Clair
instances*.
    paginationkey:

  updater:
    # interval defines how often Clair will check for updates from its upstream vulnerability databases.
    interval: 6h
  notifier:
    attempts: 3
    renotifyinterval: 1h
    http:
      # QUAY_ENDPOINT defines the endpoint at which Quay is running.
      # For example: https://myregistry.mycompany.com
      endpoint: { QUAY_ENDPOINT }/secscan/notify
      proxy: http://localhost:6063

jwtproxy:
  signer_proxy:
    enabled: true
    listen_addr: :6063
    ca_key_file: /certificates/mitm.key # Generated internally, do not change.
    ca_crt_file: /certificates/mitm.crt # Generated internally, do not change.
    signer:
      issuer: security_scanner
      expiration_time: 5m
      max_skew: 1m
      nonce_length: 32
      private_key:
        type: autogenerated
        options:
          rotate_every: 12h
          key_folder: /clair/config/
          key_server:
            type: keyregistry
            options:
              # QUAY_ENDPOINT defines the endpoint at which Quay is running.
              # For example: https://myregistry.mycompany.com
              registry: { QUAY_ENDPOINT }/keys/
```

```
    verifier_proxies:
    - enabled: true
      # The port at which Clair will listen.
      listen_addr: :6060

      # If Clair is to be served via TLS, uncomment these lines. See the "Running Clair under TLS"
      # section below for more information.
      # key_file: /clair/config/clair.key
      # crt_file: /clair/config/clair.crt

      verifier:
        # CLAIR_ENDPOINT is the endpoint at which this Clair will be accessible. Note that the port
        # specified here must match the listen_addr port a few lines above this.
        # Example: https://myclair.mycompany.com:6060
        audience: { CLAIR_ENDPOINT }

        upstream: http://localhost:6062
        key_server:
          type: keyregistry
          options:
            # QUAY_ENDPOINT defines the endpoint at which Quay is running.
            # Example: https://myregistry.mycompany.com
            registry: { QUAY_ENDPOINT }/keys/
```

6.2.3. Configuring Clair for TLS

To configure Clair to run with TLS, a few additional steps are required.

6.2.3.1. Using certificates from a public CA

For certificates that come from a public certificate authority, follow these steps:

1. Generate a TLS certificate and key pair for the DNS name at which Clair will be accessed

2. Place these files as **clair.crt** and **clair.key** in your Clair configuration directory

3. Uncomment the **key_file** and **crt_file** lines under **verifier_proxies** in your Clair **config.yaml**

If your certificates use a public CA, you are now ready to run Clair. If you are using your own certificate authority, configure Clair to trust it below.

6.2.3.2. Configuring trust of self-signed SSL

Similar to the process for setting up Docker to trust your self-signed certificates, Clair must also be configured to trust your certificates. Using the same CA certificate bundle used to configure Docker, complete the following steps:

1. Rename the same CA certificate bundle used to set up Quay Registry to **ca.crt**

2. Make sure the **ca.crt** file is mounted inside the Clair container under **/etc/pki/ca-trust/source/anchors/** as in the example below:

```
# docker run --restart=always -p 6060:6060 -p 6061:6061 \
    -v /path/to/clair/config/directory:/clair/config \
    -v /path/to/quay/cert/ca.crt:/etc/pki/ca-trust/source/anchors/ca.crt  \
```

> quay.io/redhat/clair-jwt:v3.3.1

Now Clair will be able to trust the source of your TLS certificates and use them to secure communication between Clair and Quay.

6.2.4. Using Clair data sources

Before scanning container images, Clair tries to figure out the operating system on which the container was built. It does this by looking for specific filenames inside that image (see Table 1). Once Clair knows the operating system, it uses specific security databases to check for vulnerabilities (see Table 2).

Table 6.1. Container files that identify its operating system

Operating system	Files identifying OS type
Redhat/CentOS/Oracle	etc/oracle-release etc/centos-release etc/redhat-release etc/system-release
Alpine	etc/alpine-release
Debian/Ubuntu:	etc/os-release usr/lib/os-release etc/apt/sources.list
Ubuntu	etc/lsb-release

The data sources that Clair uses to scan containers are shown in Table 2.
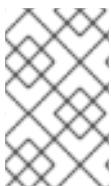
> **NOTE**
>
> You must be sure that Clair has access to all listed data sources by whitelisting access to each data source's location. You might need to add a wild-card character (*) at the end of some URLS that may not be fully complete because they are dynamically built by code.

Table 6.2. Clair data sources and data collected

Data source	Data collected	Whitelist links	Format	License
Debian Security Bug Tracker	Debian 6, 7, 8, unstable namespaces	https://security-tracker.debian.org/tracker/data/json https://security-tracker.debian.org/tracker	dpkg	Debian

Data source	Data collected	Whitelist links	Format	License
Ubuntu CVE Tracker	Ubuntu 12.04, 12.10, 13.04, 14.04, 14.10, 15.04, 15.10, 16.04 namespaces	https://git.launchpad.net/ubuntu-cve-tracker http://people.ubuntu.com/~ubuntu-security/cve/%s	dpkg	GPLv2
Red Hat Security Data	CentOS 5, 6, 7 namespace	https://www.redhat.com/security/data/oval/	rpm	CVRF
Oracle Linux Security Data	Oracle Linux 5, 6, 7 namespaces	https://linux.oracle.com/oval/	rpm	CVRF
Alpine SecDB	Alpine 3.3, 3.4, 3.5 namespaces	https://github.com/alpinelinux/alpine-secdb https://cve.mitre.org/cgi-bin/cvename.cgi?name=	apk	MIT
NIST NVD	Generic vulnerability metadata	https://nvd.nist.gov/feeds/xml/cve/2.0/nvdcve-2.0-%s.xml.gz https://nvd.nist.gov/feeds/xml/cve/2.0/nvdcve-2.0-%s.meta	N/A	Public domain
Amazon Linux Security Advisories	Amazon Linux 2018.03, 2 namespaces	Amazonaws.com mirror list Amazon.com mirror list	rpm	MIT-0

6.2.5. Run Clair

Execute the following command to run Clair:

```
# docker run --restart=always -p 6060:6060 -p 6061:6061 \
    -v /path/to/clair/config/directory:/clair/config \
    quay.io/redhat/clair-jwt:v3.3.1
```

Output similar to the following will be seen on success:

```
2016-05-04 20:01:05,658 CRIT Supervisor running as root (no user in config file)
2016-05-04 20:01:05,662 INFO supervisord started with pid 1
2016-05-04 20:01:06,664 INFO spawned: 'jwtproxy' with pid 8
2016-05-04 20:01:06,666 INFO spawned: 'clair' with pid 9
2016-05-04 20:01:06,669 INFO spawned: 'generate_mitm_ca' with pid 10
time="2016-05-04T20:01:06Z" level=info msg="No claims verifiers specified, upstream should be
configured to verify authorization"
time="2016-05-04T20:01:06Z" level=info msg="Starting reverse proxy (Listening on ':6060')"
2016-05-04 20:01:06.715037 I | pgsql: running database migrations
time="2016-05-04T20:01:06Z" level=error msg="Failed to create forward proxy: open
/certificates/mitm.crt: no such file or directory"
goose: no migrations to run. current version: 20151222113213
2016-05-04 20:01:06.730291 I | pgsql: database migration ran successfully
2016-05-04 20:01:06.730657 I | notifier: notifier service is disabled
2016-05-04 20:01:06.731110 I | api: starting main API on port 6062.
2016-05-04 20:01:06.736558 I | api: starting health API on port 6061.
2016-05-04 20:01:06.736649 I | updater: updater service is disabled.
2016-05-04 20:01:06,740 INFO exited: jwtproxy (exit status 0; not expected)
2016-05-04 20:01:08,004 INFO spawned: 'jwtproxy' with pid 1278
2016-05-04 20:01:08,004 INFO success: clair entered RUNNING state, process has stayed up for >
than 1 seconds (startsecs)
2016-05-04 20:01:08,004 INFO success: generate_mitm_ca entered RUNNING state, process has
stayed up for > than 1 seconds (startsecs)
time="2016-05-04T20:01:08Z" level=info msg="No claims verifiers specified, upstream should be
configured to verify authorization"
time="2016-05-04T20:01:08Z" level=info msg="Starting reverse proxy (Listening on ':6060')"
time="2016-05-04T20:01:08Z" level=info msg="Starting forward proxy (Listening on ':6063')"
2016-05-04 20:01:08,541 INFO exited: generate_mitm_ca (exit status 0; expected)
2016-05-04 20:01:09,543 INFO success: jwtproxy entered RUNNING state, process has stayed up for
> than 1 seconds (startsecs)
```

To verify Clair is running, execute the following command:

```
curl -X GET -I http://path/to/clair/here:6061/health
```

If a **200 OK** code is returned, Clair is running:

```
HTTP/1.1 200 OK
Server: clair
Date: Wed, 04 May 2016 20:02:16 GMT
Content-Length: 0
Content-Type: text/plain; charset=utf-8
```

Once Clair and its associated database are running, you man need to restart your quay application for the changes to take effect.

CHAPTER 7. USING CLAIR V4 SECURITY SCANNING

Clair v4 is the next generation of Clair image scanning available with Red Hat Quay. Clair v4 is currently released as Technology Preview, which means that it is not supported for production use. However, you are encouraged to test Clair v4 as it represents the direction of Clair image scanning development.

To align with the Red Hat Quay release, the current Clair v4 release image is clair:v3.3.1.

7.1. WHAT IS CLAIR V4?

Technically, Clair v4 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 v4 makes it appropriate to run in a highly scalable configuration, where components can be scaled separately as appropriate for enterprise environments.

For the purposes of trying out Clair v4, we recommend running it in combo mode (see clair-combo.yaml) This mode, described here, brings all the microservices together as one process.

All supported security databases for Clair v4 are turned on. These databases include:

- Alpine SecDB database

- AWS UpdateInfo

- Debian Oval database

- Oracle Oval database

- RHEL Oval database

- SUSE Oval database

- Ubuntu Oval database

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

> **WARNING**
>
> Because Clair v4 is Technology Preview, don't expect 100% accurate reporting. Expect that the presentation of vulnerability results will look different going forward than they did with v2. Over time, Clair v4 will produce more results.

The steps for using Clair v4 alongside an existing Red Hat Quay + Clair v2 environment are described next.

7.2. CONFIGURING CLAIR V4

To try out Clair v4, stand up a Red Hat Quay cluster with a running Clair v2 instance. Then use the following procedure to run Clair v4 along side it. Here is how to do that on an OpenShift v4.2 or later cluster on an AWS cloud.

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

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

5. Create a secret from the Clair config.yaml:

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

6. 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: quay.io/redhat/clair:v3.3.1         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.

7. Create the Clair v4 deployment as follows:

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

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

   ```
   ...
   FEATURE_SECURITY_SCANNER: true
   SECURITY_SCANNER_V4_ENDPOINT: http://clairv4 1
   SECURITY_SCANNER_V4_NAMESPACE_WHITELIST: 2
     - "clairv4-org"
     - "foo-org"
   ```

   1   Identify the Clair v4 service endpoint

   2   Replace **clair4-org** and **foo-org** with namespaces (organizations and users) in your Red Hat Quay cluster you want to use Clair v4 scanning

9. 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
```

10. 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.

7.3. USING CLAIR V4

The user interface for viewing vulnerability information gathered by Clair v4 is essentially the same as it was for Clair v2.

1. Log in to your Red Hat Quay cluster and select an organization for which you have configured Clair v4 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:

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:
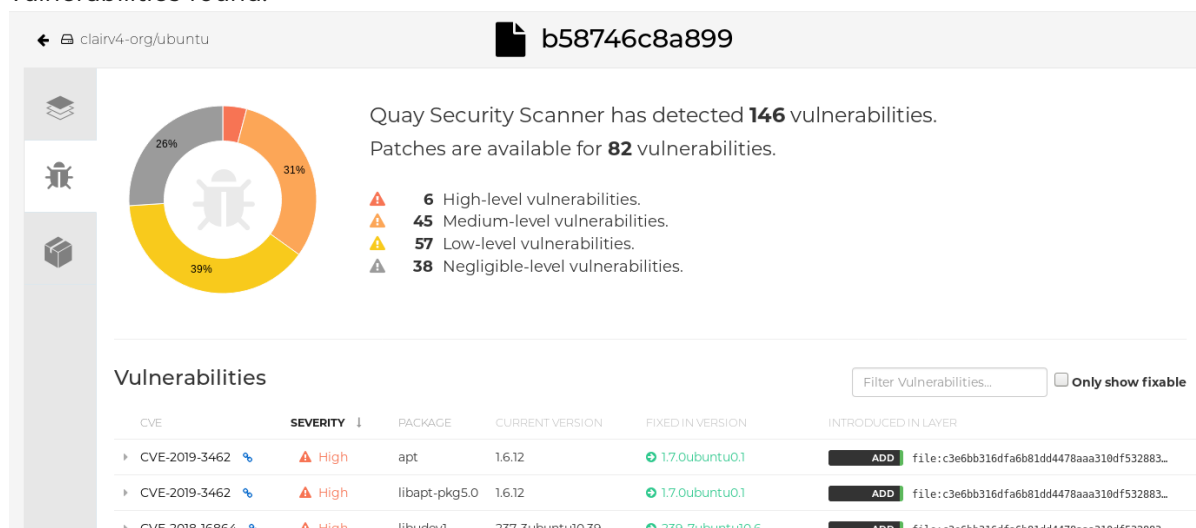
CHAPTER 8. SCAN POD IMAGES WITH THE CONTAINER SECURITY OPERATOR

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 OPENSHIFT

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:

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:

   - 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:

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 OPENSHIFT 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.

> ⚠️ **WARNING**
>
> You cannot have more than one OpenShift Container Platform cluster pointing to the same Red Hat Quay instance from a Quay Bridge Operator. If you did, it would prevent you from creating namespaces of the same name on the two 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:

9.1.2.1. 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.

9.1.2.2. OpenShift Setup

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

- **Creating an OpenShift secret** Using the OAuth token created earlier in Quay, create an OpenShift secret.

- **Adding MutatingWebhookConfiguration support**: To support Red Hat Quay integration with OpenShift, any new Build requests should be intercepted so that the output can be modified to target Red Hat Quay instead of OpenShift's integrated registry.

Support for dynamic interception of API requests that are performed as part of OpenShift's typical build process is facilitated through a MutatingWebhookConfiguration. A MutatingWebhookConfiguration allows for invoking an API running within a project on OpenShift when certain API requests are received.

Kubernetes requires that the webhook endpoint is secured via SSL using a certificate that makes use of the certificate authority for the cluster. Fortunately, OpenShift provides support for generating a certificate signed by the cluster.

1. Using the OpenShift **oc** command line tool, log in to OpenShift as a cluster administrator.

2. Choose an OpenShift namespace to use, such as **openshift-operators** or create a new one.

3. Create an OpenShift secret, replacing <access_token> with the Access Token obtained earlier from Red Hat Quay. For example, this creates a secret with your <access_token> called **quay-integration** with a key called **token**:

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

The result places the newly created private key and certificate within a secret specified. The secret will be mounted into the appropriate located within the operator as declared in the Deployment of the Operator.

4. Create a Service for the Operator's webhook endpoint:

quay-webhook.yaml

```
apiVersion: v1
kind: Service
metadata:
  labels:
    name: quay-bridge-operator
  name: quay-bridge-operator
  namespace: openshift-operators
spec:
  ports:
    - name: https
      port: 443
      protocol: TCP
      targetPort: 8443
  selector:
    name: quay-bridge-operator
  sessionAffinity: None
  type: ClusterIP
```

5. Create the webhook service as follows:

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

6. Download the webhook-create-signed-cert.sh script, so you can use it to generate a certificate signed by a Kubernetes certificate authority.

7. Execute the following command to request the certificate:

```
$ ./webhook-create-signed-cert.sh --namespace openshift-operators \
    --secret quay-bridge-operator-webhook-certs \
    --service quay-bridge-operator
```

8. Execute the following command to retrieve the CA and format the result as a single line so that it can be entered into the MutatingWebhookConfiguration resource:

```
$ oc get configmap -n kube-system \
    extension-apiserver-authentication \
    -o=jsonpath='{.data.client-ca-file}' | base64 | tr -d '\n'
```

9. Replace the ${CA_BUNDLE} variable in the following MutatingWebhookConfiguration YAML:

quay-mutating-webhook.yaml

```
apiVersion: admissionregistration.k8s.io/v1beta1
kind: MutatingWebhookConfiguration
metadata:
  name: quay-bridge-operator
webhooks:
  - name: quayintegration.redhatcop.redhat.io
    clientConfig:
      service:
        namespace: openshift-operators
        name: quay-bridge-operator
```

```
        path: "/admissionwebhook"
       caBundle: "${CA_BUNDLE}"  ❶
     rules:
     - operations:  [ "CREATE" ]
       apiGroups: [ "build.openshift.io" ]
       apiVersions: ["v1" ]
       resources: [ "builds" ]
     failurePolicy: Fail
```

❶ Replace ${CA_BUNDLE} with the output of the previous step. It will appear as one long line that you copy and paste to replace ${CA_BUNDLE}.

10. Create the MutatingWebhookConfiguration as follows:

```
$ oc create -f quay-mutating-webhook.yaml
```

Until the operator is running, new requests for builds will fail since the webserver the MutatingWebhookConfiguration invokes is not available and a proper is response is required in order for the object to be persisted in etcd.

11. Go to the OpenShift console and install the Quay Bridge Operator as follows:

- Select OperatorHub and search for Quay Bridge Operator.

- Select Install

- Choose Installation Mode (all namespaces), Update Channel, and Approval Strategy (Automatic or Manual).

- Select Subscribe

12. Create the custom resource (CR) called **QuayIntegration**. For example:

quay-integration.yaml

```
apiVersion: redhatcop.redhat.io/v1alpha1
kind: QuayIntegration
metadata:
  name: example-quayintegration
spec:
  clusterID: openshift  ❶
  credentialsSecretName: openshift-operators/quay-integration  ❷
  quayHostname: https://<QUAY_URL>  ❸
  whitelistNamespaces:  ❹
   - default
  insecureRegistry: false  ❺
```

❶ The clusterID value should be unique across the entire ecosystem. This value is optional and defaults to openshift.

❷ For credentialsSecretName, replace **openshift-operators/quay-integration** with the name of the namespace and the secret containing the token you created earlier.

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

4 The whitelistNamespaces is optional. If not used, the Bridge Operator will sync all namespaces to Red Hat Quay except the openshift prefixed project. In this example, the white listed namespace (default) will now have an associated Red Hat Quay organization. Use any namespace you like here.

5 If Quay is using self signed certificates, set the property **insecureRegistry: true**.

The result is that organizations within Red Hat Quay should be created for the related namespaces in OpenShift.

13. Create the **QuayIntegration** as follows:

```
$ 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.

The whitelisted namespace you created should now have a Red Hat Quay organization. If you were to use a command such as **oc new-app** to create a new application in that namespace, you would see a new Red Hat Quay repository created for it instead of using the internal registry.

CHAPTER 10. REPOSITORY MIRRORING IN RED HAT QUAY

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

10.1. OVERVIEW OF REPOSITORY MIRRORING

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

- Set intervals at which a repository is synced

- Identify specific container image repository names and tags to sync

- Check the current state of synchronization

With repository mirroring, you mirror a specific subset of content, between two or more distinct registries, to selected datacenters, clusters, or regions. By contrast, Georeplication provides a single, globally distributed Red Hat Quay to serve container images from localized storage. The two approaches to sharing content differ in the following ways:

Table 10.1. Red Hat Quay Repository Mirroring vs. Georeplication

Feature / Capability	Georeplication	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

Here are a few tips for using Red Hat Quay repository mirroring:

- With repository mirroring, you can mirror an entire repository or selectively limit which images are synced based on a comma-separated list of tags, a range of tags, or other means of identifying tags through regular expressions and globs.

- Once set as a mirrored repository, 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 don't match any more, 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 setting on your mirrored repository is done from a 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.

10.2. PREREQUISITES

Before you can use repository mirroring, you must enable repository mirroring from the Red Hat Quay configuration screen and start the repository mirroring worker. Ways of starting up this service are described in the Red Hat Quay deployment guides:

- Deploy Red Hat Quay - Basic

- Deploy Red Hat Quay - High Availability

- Deploy Red Hat Quay on OpenShift

The steps shown in the following section assumes you already have the repository mirroring service running and that you have enabled repository mirroring on your Red Hat Quay cluster.

10.3. CREATE A MIRRORED REPOSITORY

To mirror an external repository from an external container registry, do the following:

1. Log into your Red Hat Quay registry.

2. Create a robot account to pull images for the mirrored repository:

   - Select Account Settings from the drop-down in the upper right corner.

   - Select the Robot Accounts button in the left column.

   - Select Create Robot Account.

   - Add the name and description of the robot account and select Create robot account.

   - Select Close, since the mirrored repository you are adding does not exist yet.

- Select the ROBOT ACCOUNT NAME from the listing.

- When prompted, add the credentials needed by the robot to access the external registry of the repository you want to mirror, then close the Credentials window.

3. Select REPOSITORIES.

4. Select Create New Repository and give it a name.

5. Fill in a repository name, select Public or Private, and select Create Repository.

6. Select the Settings button and change the repository state to MIRROR.

7. Open the new repository and select the Mirroring button in the left column.

8. Fill in the fields to identify the repository you are mirroring in your new repository:

- **Registry URL**: Location of the container registry you want to mirror from.

- **User or Organization**: Typically, the account name associated with the content you are mirroring. For example, with the image registry.example.com/jsmith/myimage:latest, jsmith would be entered here.

- **Repository Name**: The name identifying the name of the set of images. For example, with the image registry.example.com/jsmith/myimage:latest, myimage would be entered here.

- **Sync Interval**: Defaults to syncing every 24 hours. You can change that based on hours or days.

- **Robot User**: Select the robot account you created earlier to do the mirroring.

- **Username**: The username for logging into 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 (\).

- **Start Date**: The date on which mirroring begins. The current date and time used by default.

- **Verify TLS**: Check this box if you want to verify the authenticity of the external registry. Uncheck this box if, for example, you set up Red Hat Quay for testing with a self-signed certificate or no certificate.

- **HTTP Proxy**: Identify the proxy server needed to access the remote site, if one is required.

- **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**
>
> At least one Tag must be explicitly entered (ie. not a tag pattern) *or* the tag "latest" must exist in the remote repository. This is required for Quay to get the list of tags in the remote repository to compare to the specified list to mirror.

Here is an example of a completed Repository Mirroring screen:

9. Select the Enable Mirror button. Here's the resulting Repository Mirroring page:

You can return to this page later to change any of those settings.

10.4. WORKING WITH MIRRORED REPOSITORIES

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

- **Enable/disable the repository**: Select the Mirroring button in the left column, then toggle the Enabled check box to enable or disable the repository temporarily.

- **Check mirror logs**: To make sure the mirrored repository is working properly, you can check the mirror logs. To do that, select the Usage Logs button in the left column. Here's an example:
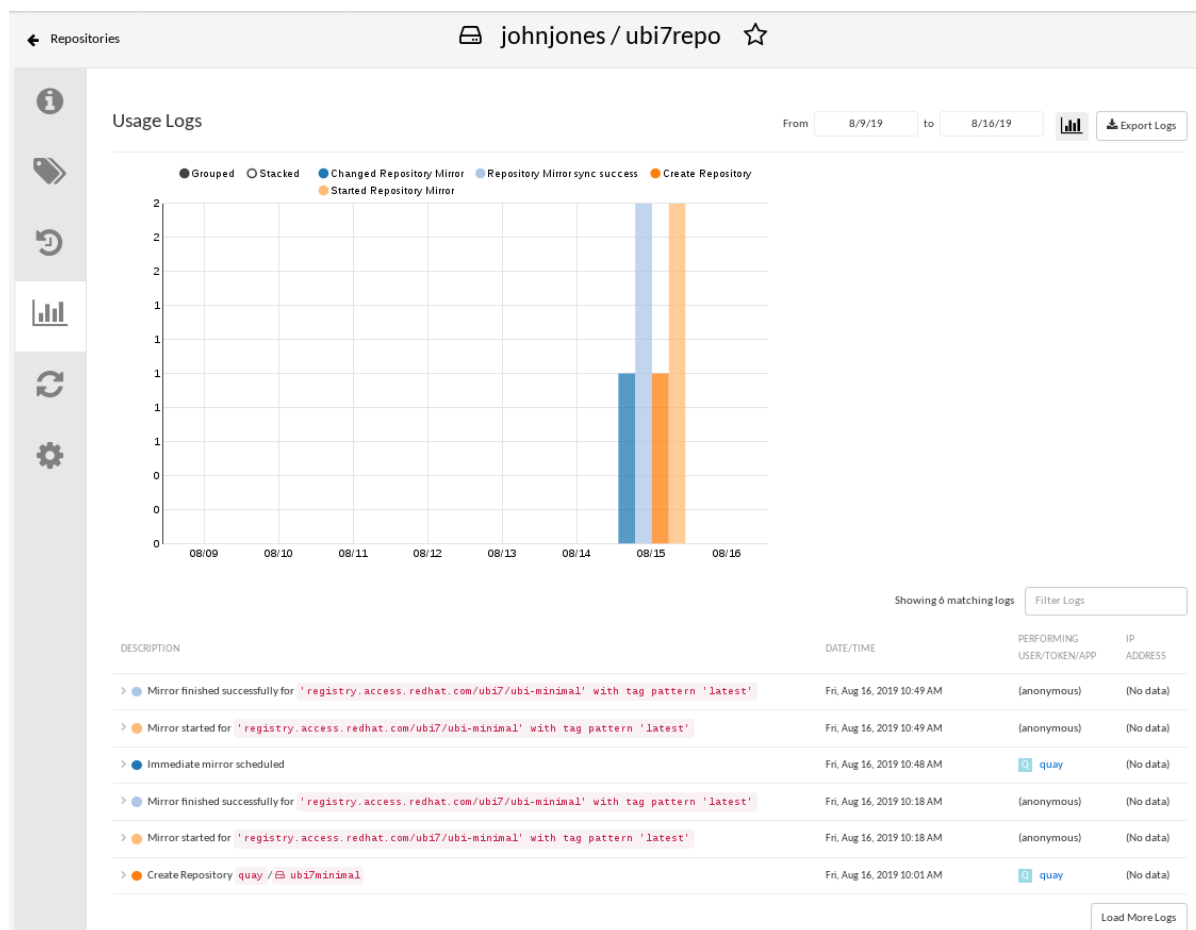
- **Sync mirror now**: To immediately sync the images in your repository, select the Sync Now button.

- **Change credentials**: To change the username and password, select DELETE from the Credentials line. Then select None and add the username and password needed to log into the external registry when prompted.

- **Cancel mirroring**: To stop mirroring, which keeps the current images available but stops new ones from being synced, select the CANCEL button.

- **Set robot permissions**: Red Hat Quay robot accounts are named tokens that hold credentials for accessing external repositories. By assigning credentials to a robot, that robot can be used across multiple mirrored repositories that need to access the same external registry.
  You can assign an existing robot to a repository by going to Account Settings, then selecting the Robot Accounts icon in the left column. For the robot account, choose the link under the REPOSITORIES column. From the pop-up window, you can:

  - Check which repositories are assigned to that robot.

- Assign read, write or Admin privileges to that robot from the PERMISSION field shown in this figure:
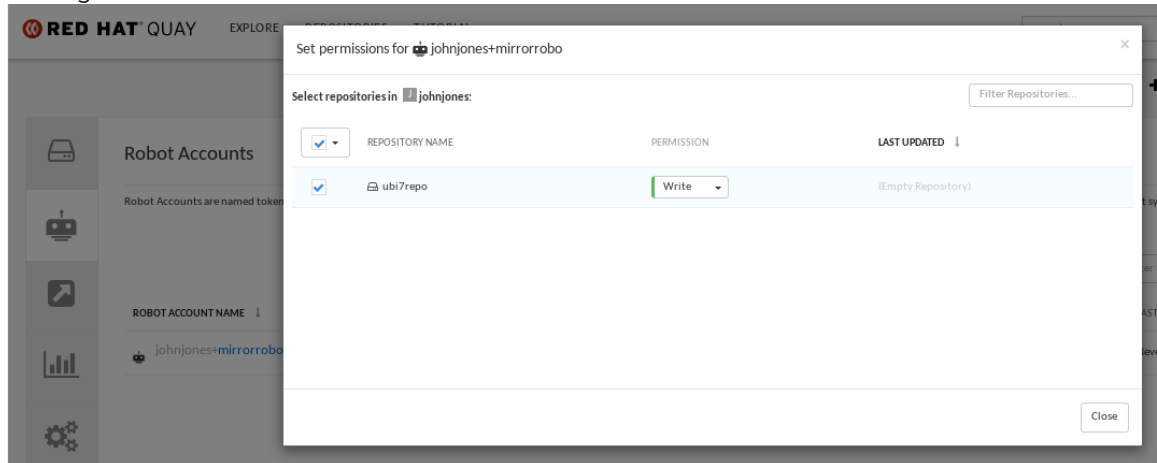
- **Change robot credentials**: Robots can hold credentials such as Kubernetes secrets, Docker login information, and Mesos bundles. To change robot credentials, select the Options gear on the robot's account line on the Robot Accounts window and choose View Credentials. Add the appropriate credentials for the external repository the robot needs to access.
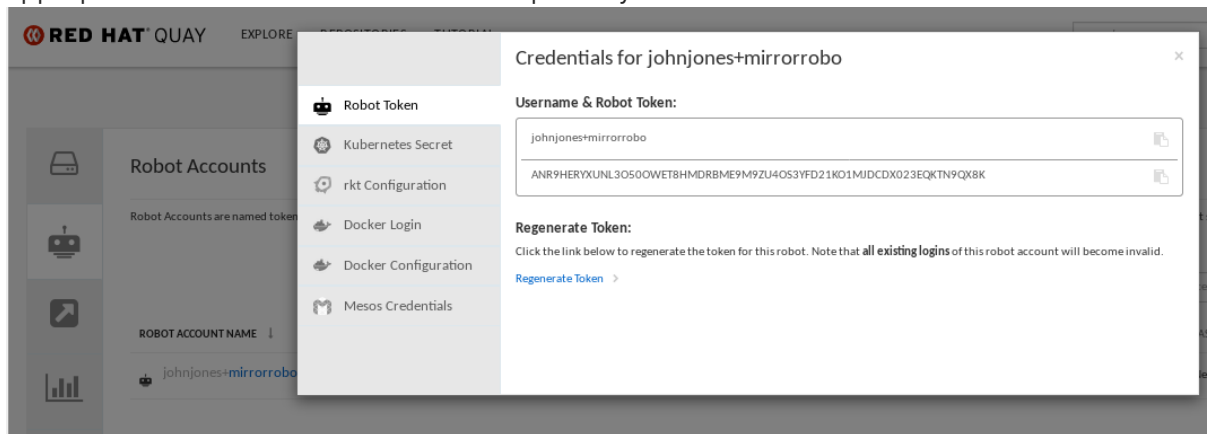
- **Check and change general setting**: Select the Settings button (gear icon) from the left column on the mirrored repository page. On the resulting page, you can change settings associated with the mirrored repository. In particular, you can change User and Robot Permissions, to specify exactly which users and robots can read from or write to the repo.

10.5. 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.

Pattern syntax

Pattern	Description
*	Matches all characters
?	Matches any single character

[seq]	Matches any character in *seq*
[!seq]	Matches any character not in *seq*

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

CHAPTER 11. LDAP AUTHENTICATION SETUP FOR RED HAT QUAY

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

11.1. 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.

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

AUTHENTICATION_TYPE: LDAP

11.1.1. Full LDAP URI

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

- A URI beginning with *ldaps://* will make use of the provided SSL certificate(s) for TLS setup.

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

LDAP_URI: ldaps://ldap.example.org

11.1.2. Team Synchronization

- 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: `60m`

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

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

☐ Allow non-superusers to enable and manage team syncing

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

- The resynchronization duration is the period at which a team must be re-synchronized. Must be expressed in a duration string form: 30m, 1h, 1d.

- Optionally allow non-superusers to enable and manage team syncing under organizations in which they are administrators.

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

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

11.1.3. Base and Relative Distinguished Names

Base DN: `dc=example,dc=org`

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: `ou=NYC`

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

[   ] Add

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

Example: [ou=employees]

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

- Optional list of Distinguished Name path(s) which form the secondary base path(s) for looking up all user LDAP records, relative to the Base DN defined above. These path(s) will be tried if the user is not found via the primary relative DN.

- User Relative DN is relative to BaseDN. Example: *ou=NYC* not *ou=NYC,dc=example,dc=org*

- Multiple "Secondary User Relative DNs" may be entered if there are multiple Organizational Units where User objects are located at. Simply type in the Organizational Units and click on Add button to add multiple RDNs. Example: *ou=Users,ou=NYC and ou=Users,ou=SFO*

- The "User Relative DN" searches with subtree scope. For example, if your Organization has Organizational Units NYC and SFO under the Users OU (*ou=SFO,ou=Users* and *ou=NYC,ou=Users*), Red Hat Quay can authenticate users from both the NYC and *SFO* Organizational Units if the User Relative DN is set to Users (*ou=Users*).

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

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

11.1.4. Additional User Filters

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

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

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

11.1.5. Administrator DN

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

- The password will be stored in **plaintext** inside the config.yaml, so setting up a dedicated account or using a password hash is highly recommended.

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

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

11.1.6. 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
```

11.1.7. Validation

Once the configuration is completed, click on "Save Configuration Changes" button to validate the configuration.
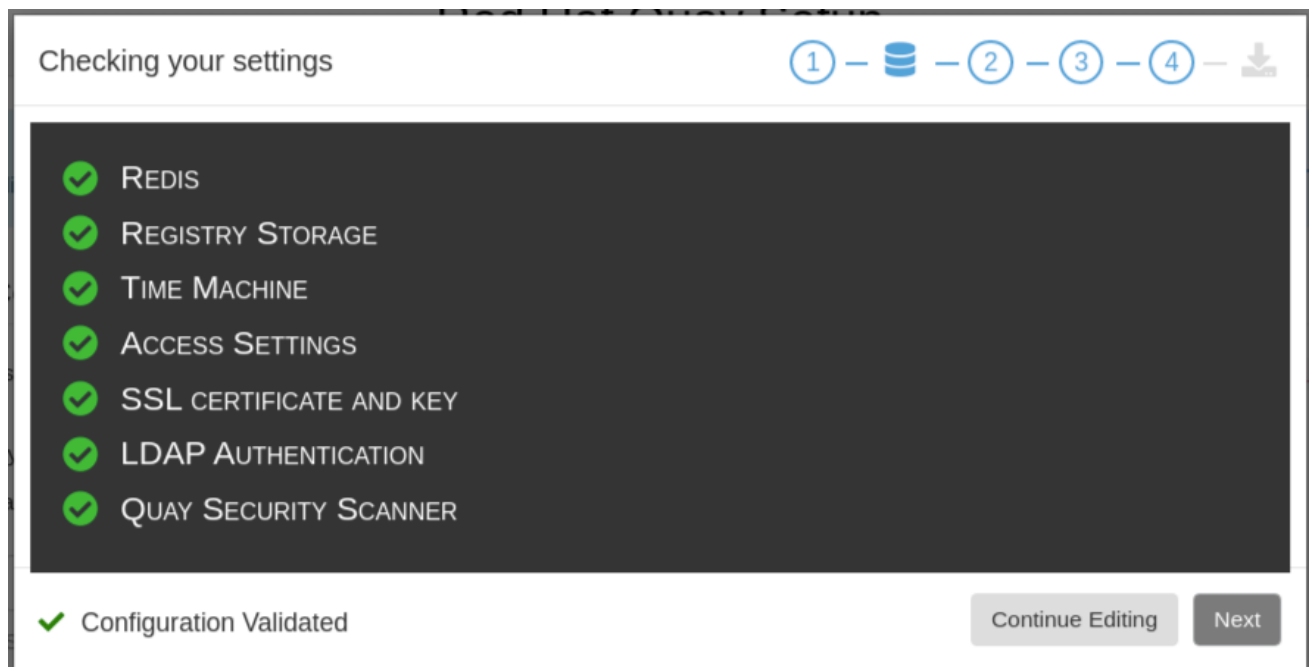
All validation must succeed before proceeding, or additional configuration may be performed by selecting the "Continue Editing" button.

11.2. COMMON ISSUES

Invalid credentials
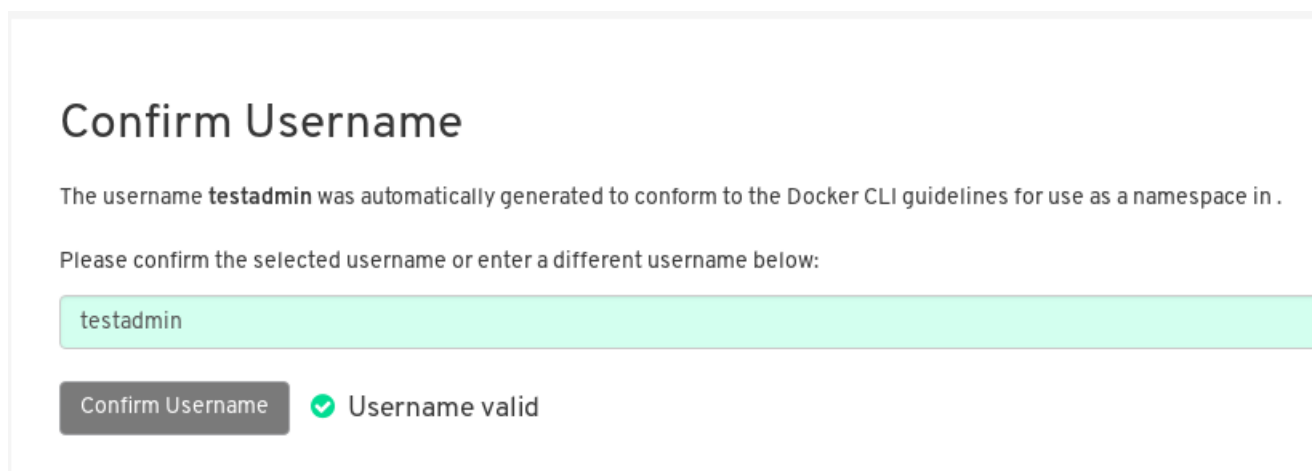
Administrator DN or Administrator DN Password values are incorrect

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

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

11.3. 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:

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

```
SUPER_USERS:
- testadmin
```

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

CHAPTER 12. PROMETHEUS AND GRAFANA METRICS UNDER RED HAT QUAY

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

12.1. EXPOSING THE PROMETHEUS ENDPOINT

The Prometheus- and Grafana-compatible endpoint on the Red Hat Quay instance can be found at port **9092**. See Monitoring Quay with Prometheus and Grafana for details on configuring Prometheus and Grafana to monitor Quay repository counts.

12.1.1. Setting up Prometheus to consume metrics

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

12.1.2. DNS configuration under Kubernetes

A simple Kubernetes service can be configured to provide the DNS entry for Prometheus. Details on running Prometheus under Kubernetes can be found at Prometheus and Kubernetes and Monitoring Kubernetes with Prometheus.

12.1.3. 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.

CHAPTER 13. GEOREPLICATION OF STORAGE IN RED HAT QUAY

Georeplication allows for a single globally-distributed Red Hat Quay to serve container images from localized storage.

When georeplication is configured, container image pushes will be written to the preferred storage engine for that Red Hat Quay instance. After the initial push, image data will be replicated in the background to other storage engines. The list of replication locations is configurable. An image pull will always use the closest available storage engine, to maximize pull performance.

13.1. PREREQUISITES

Georeplication requires that there be a high availability storage engine (S3, GCS, RADOS, Swift) in each geographic region. Further, each region must be able to access **every** storage engine due to replication requirements.
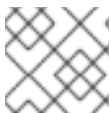
> **NOTE**
>
> Local disk storage is not compatible with georeplication at this time.

13.2. VISIT THE CONFIG TOOL

Open the Red Hat Quay Config Tool to configure storage for georeplication.

13.3. 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/kubectl 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.

13.4. 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 /mnt/quay/config:

   ```
   # docker login quay.io
   Username: yourquayuser
   Password: *****
   # docker run -d -p 443:8443 -p 8080:8080 -v /mnt/quay/config:/conf/stack:Z \
       -e QUAY_DISTRIBUTED_STORAGE_PREFERENCE=europestorage \
       quay.io/redhat/quay:v3.3.1
   ```

> **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 14. 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 15. SCHEMA FOR RED HAT QUAY

> **NOTE**
>
> All fields are optional unless otherwise marked.

- **AUTHENTICATION_TYPE** [string] required: The authentication engine to use for credential authentication.

  - **enum**: Database, LDAP, JWT, Keystone, OIDC.

  - Example: **Database**

- **BUILDLOGS_REDIS** [object] required: Connection information for Redis for build logs caching.

  - **HOST** [string] required: The hostname at which Redis is accessible.

    - Example: **my.redis.cluster**

  - **PASSWORD** [string]: The password to connect to the Redis instance.

    - Example: **mypassword**

  - **PORT** [number]: The port at which Redis is accessible.

    - Example: **1234**

- **DB_URI** [string] required: The URI at which to access the database, including any credentials.

  - **Reference**: https://www.postgresql.org/docs/9.3/static/libpq-connect.html#AEN39495

  - Example: **mysql+pymysql://username:password@dns.of.database/quay**

- **DEFAULT_TAG_EXPIRATION** [string] required: The default, configurable tag expiration time for time machine. Defaults to **2w**.

  - Pattern: **^[0-9]+(w|m|d|h|s)$**

- **DISTRIBUTED_STORAGE_CONFIG** [object] required: Configuration for storage engine(s) to use in Red Hat Quay. Each key is a unique ID for a storage engine, with the value being a tuple of the type and configuration for that engine.

  - Example: **{"local_storage": ["LocalStorage", {"storage_path": "some/path/"}]}**

- **DISTRIBUTED_STORAGE_PREFERENCE** [array] required: The preferred storage engine(s) (by ID in DISTRIBUTED_STORAGE_CONFIG) to use. A preferred engine means it is first checked for pulling and images are pushed to it.

  - **Min Items**: None

    - Example: **[u's3_us_east', u's3_us_west']**

    - **array item** [string]

  - **preferred_url_scheme** [string] required: The URL scheme to use when hitting Red Hat Quay. If Red Hat Quay is behind SSL **at all**, this **must** be **https**.

    - enum: **http, https**

- Example: **https**

- SERVER_HOSTNAME [string] required: The URL at which Red Hat Quay is accessible, without the scheme.

  - Example: **quay.io**

- TAG_EXPIRATION_OPTIONS [array] required: The options that users can select for expiration of tags in their namespace (if enabled).

  - Min Items: None

  - **array item** [string]

  - Pattern: **^[0-9]+(w|m|d|h|s)$**

- USER_EVENTS_REDIS [object] required: Connection information for Redis for user event handling.

  - HOST [string] required: The hostname at which Redis is accessible.

    - Example: **my.redis.cluster**

  - PASSWORD [string]: The password to connect to the Redis instance.

    - Example: **mypassword**

  - PORT [number]: The port at which Redis is accessible.

    - Example: **1234**

- ACTION_LOG_ARCHIVE_LOCATION [string]: If action log archiving is enabled, the storage engine in which to place the archived data.

  - Example: **s3_us_east**

- ACTION_LOG_ARCHIVE_PATH' [string]: If action log archiving is enabled, the path in storage in which to place the archived data.

  - Example: **archives/actionlogs**

- APP_SPECIFIC_TOKEN_EXPIRATION [string, **null**]: The expiration for external app tokens. Defaults to None.

  - Pattern: **^[0-9]+(w|m|d|h|s)$**

- ALLOW_PULLS_WITHOUT_STRICT_LOGGING [boolean]: If true, pulls in which the pull audit log entry cannot be written will still succeed. Useful if the database can fallback into a read-only state and it is desired for pulls to continue during that time. Defaults to False.

  - Example: **True**

- AVATAR_KIND [string]: The types of avatars to display, either generated inline (local) or Gravatar (gravatar)

  - **enum**: local, gravatar

- BITBUCKET_TRIGGER_CONFIG ['object', 'null']: Configuration for using BitBucket for build triggers.

- **consumer_key** [string] required: The registered consumer key(client ID) for this Red Hat Quay instance.

  - **Example**: **0e8dbe15c4c7630b6780**

- **CONSUMER_SECRET** [string] required: The registered consumer secret(client secret) for this Red Hat Quay instance

  - **Example**: e4a58ddd3d7408b7aec109e85564a0d153d3e846

- **BROWSER_API_CALLS_XHR_ONLY** [boolean]: If enabled, only API calls marked as being made by an XHR will be allowed from browsers. Defaults to True.

  - **Example**: False

- **CONTACT_INFO** [array]: If specified, contact information to display on the contact page. If only a single piece of contact information is specified, the contact footer will link directly.

  - **Min Items**: 1

  - **Unique Items**: True

    - **array item 0** [string]: Adds a link to send an e-mail

    - **Pattern**: **^mailto:(.)+$**

    - **Example**: **mailto:support@quay.io**

  - **array item 1** [string]: Adds a link to visit an IRC chat room

    - **Pattern**: **^irc://(.)+$**

    - **Example**: **irc://chat.freenode.net:6665/quay**

  - **array item 2** [string]: Adds a link to call a phone number

    - **Pattern**: **^tel:(.)+$**

    - **Example**: **tel:+1-888-930-3475**

  - **array item 3** [string]: Adds a link to a defined URL

    - **Pattern**: **^http(s)?://(.)+$**

    - **Example**: **https://twitter.com/quayio**

- **BLACKLIST_V2_SPEC** [string]: The Docker CLI versions to which Red Hat Quay will respond that V2 is **unsupported**. Defaults to **<1.6.0**.

  - **Reference**: http://pythonhosted.org/semantic_version/reference.html#semantic_version.Spec

  - **Example**: **<1.8.0**

- **DB_CONNECTION_ARGS** [object]: If specified, connection arguments for the database such as timeouts and SSL.

  - **threadlocals** [boolean] required: Whether to use thread-local connections. Should **ALWAYS** be **true**

- **autorollback** [boolean] required: Whether to use auto-rollback connections. Should ALWAYS be **true**

- **ssl** [object]: SSL connection configuration

  - **ca** [string] required: Absolute container path to the CA certificate to use for SSL connections.

  - Example: **conf/stack/ssl-ca-cert.pem**

- **DEFAULT_NAMESPACE_MAXIMUM_BUILD_COUNT** [number, **null**]: If not None, the default maximum number of builds that can be queued in a namespace.

  - Example: **20**

- **DIRECT_OAUTH_CLIENTID_WHITELIST** [array]: A list of client IDs of **Red Hat Quay-managed** applications that are allowed to perform direct OAuth approval without user approval.

  - **Min Items**: None

  - **Unique Items**: True

  - **Reference**: https://coreos.com/quay-enterprise/docs/latest/direct-oauth.html

    - **array item** [string]

- **DISTRIBUTED_STORAGE_DEFAULT_LOCATIONS** [array]: The list of storage engine(s) (by ID in DISTRIBUTED_STORAGE_CONFIG) whose images should be fully replicated, by default, to all other storage engines.

  - **Min Items**: None

  - Example: **s3_us_east, s3_us_west**

    - **array item** [string]

- **EXTERNAL_TLS_TERMINATION** [boolean]: If TLS is supported, but terminated at a layer before Red Hat Quay, must be true.

  - Example: **True**

- **ENABLE_HEALTH_DEBUG_SECRET** [string, **null**]: If specified, a secret that can be given to health endpoints to see full debug info when not authenticated as a superuser.

  - Example: **somesecrethere**

- **EXPIRED_APP_SPECIFIC_TOKEN_GC** [string, **null**]: Duration of time expired external app tokens will remain before being garbage collected. Defaults to 1d.

  - pattern: **^[0-9]+(w|m|d|h|s)$**

- **FEATURE_ACI_CONVERSION** [boolean]: Whether to enable conversion to ACIs. Defaults to False.

  - Example: **False**

- **FEATURE_ACTION_LOG_ROTATION** [boolean]: Whether or not to rotate old action logs to storage. Defaults to False.

  - Example: **False**

- **FEATURE_ADVERTISE_V2** [boolean]: Whether the v2/ endpoint is visible. Defaults to True.

  - Example: **True**

- **FEATURE_ANONYMOUS_ACCESS** [boolean]: Whether to allow anonymous users to browse and pull public repositories. Defaults to True.

  - Example: **True**

- **FEATURE_APP_REGISTRY** [boolean]: Whether to enable support for App repositories. Defaults to False.

  - Example: **False**

- **FEATURE_APP_SPECIFIC_TOKENS** [boolean]: If enabled, users can create tokens for use by the Docker CLI. Defaults to True.

  - Example: False

- **FEATURE_BITBUCKET_BUILD** [boolean]: Whether to support Bitbucket build triggers. Defaults to False.

  - Example: **False**

- **FEATURE_BUILD_SUPPORT** [boolean]: Whether to support Dockerfile build. Defaults to True.

  - Example: **True**

- **FEATURE_CHANGE_TAG_EXPIRARTION** [boolean]: Whether users and organizations are allowed to change the tag expiration for tags in their namespace. Defaults to True.

  - Example: **False**

- **FEATURE_DIRECT_LOGIN** [boolean]: Whether users can directly login to the UI. Defaults to True.

  - Example: **True**

- **FEATURE_GITHUB_BUILD** [boolean]: Whether to support GitHub build triggers. Defaults to False.

  - Example: **False**

- **FEATURE_GITHUB_LOGIN** [boolean]: Whether GitHub login is supported. Defaults to False.

  - Example: **False**

- **FEATURE_GITLAB_BUILD**[boolean]: Whether to support GitLab build triggers. Defaults to False.

  - Example: **False**

- **FEATURE_GOOGLE_LOGIN** [boolean]: Whether Google login is supported. Defaults to False.

  - Example: **False**

- **FEATURE_INVITE_ONLY_USER_CREATION** [boolean]: Whether users being created must be invited by another user. Defaults to False.

  - Example: False

- ○ Example: **False**

- FEATURE_LIBRARY_SUPPORT [boolean]: Whether to allow for "namespace-less" repositories when pulling and pushing from Docker. Defaults to True.

  - ○ Example: **True**

- FEATURE_MAILING [boolean]: Whether emails are enabled. Defaults to True.

  - ○ Example: **True**

- FEATURE_NONSUPERUSER_TEAM_SYNCING_SETUP [boolean]: If enabled, non-superusers can setup syncing on teams to backing LDAP or Keystone. Defaults To False.

  - ○ Example: **True**

- FEATURE_PARTIAL_USER_AUTOCOMPLETE [boolean]: If set to true, autocompletion will apply to partial usernames. Defaults to True.

  - ○ Example: **True**

- FEATURE_PERMANENT_SESSIONS [boolean]: Whether sessions are permanent. Defaults to True.

  - ○ Example: **True**

- FEATURE_PROXY_STORAGE [boolean]: Whether to proxy all direct download URLs in storage via the registry nginx. Defaults to False.

  - ○ Example: **False**

- FEATURE_PUBLIC_CATALOG [boolean]: If set to true, the **_catalog** endpoint returns public repositories. Otherwise, only private repositories can be returned. Defaults to False.

  - ○ Example: **False**

- FEATURE_READER_BUILD_LOGS [boolean]: If set to true, build logs may be read by those with read access to the repo, rather than only write access or admin access. Defaults to False.

  - ○ Example: False

- FEATURE_RECAPTCHA [boolean]: Whether Recaptcha is necessary for user login and recovery. Defaults to False.

  - ○ Example: **False**

  - ○ **Reference**: https://www.google.com/recaptcha/intro/

- FEATURE_REQUIRE_ENCRYPTED_BASIC_AUTH [boolean]: Whether non-encrypted passwords (as opposed to encrypted tokens) can be used for basic auth. Defaults to False.

  - ○ Example: **False**

- FEATURE_REQUIRE_TEAM_INVITE [boolean]: Whether to require invitations when adding a user to a team. Defaults to True.

  - ○ Example: **True**

- FEATURE_SECURITY_NOTIFICATIONS [boolean]: If the security scanner is enabled, whether to turn on/off security notifications. Defaults to False.

- Example: **False**

- **FEATURE_SECURITY_SCANNER** [boolean]: Whether to turn on/off the security scanner. Defaults to False.

  - **Reference**: https://access.redhat.com/documentation/en-us/red_hat_quay/3.3/html-single/manage_red_hat_quay/#clair-initial-setup

  - Example: **False**

- **FEATURE_STORAGE_REPLICATION** [boolean]: Whether to automatically replicate between storage engines. Defaults to False.

  - Example: **False**

- **FEATURE_SUPER_USERS** [boolean]: Whether superusers are supported. Defaults to True.

  - Example: **True**

- **FEATURE_TEAM_SYNCING** [boolean]: Whether to allow for team membership to be synced from a backing group in the authentication engine (LDAP or Keystone).

  - Example: **True**

- **FEATURE_USER_CREATION** [boolean] :Whether users can be created (by non-superusers). Defaults to True.

  - Example: **True**

- **FEATURE_USER_LOG_ACCESS** [boolean]: If set to true, users will have access to audit logs for their namespace. Defaults to False.

  - Example: **True**

- **FEATURE_USER_METADATA** [boolean]: Whether to collect and support user metadata. Defaults to False.

  - Example: **False**

- **FEATURE_USER_RENAME** [boolean]: If set to true, users can rename their own namespace. Defaults to False.

  - Example: **True**

- **GITHUB_LOGIN_CONFIG** [object, 'null']: Configuration for using GitHub (Enterprise) as an external login provider.

  - **Reference**: https://coreos.com/quay-enterprise/docs/latest/github-auth.html

  - **allowed_organizations** [array]: The names of the GitHub (Enterprise) organizations whitelisted to work with the ORG_RESTRICT option.

    - **Min Items**: None

    - **Unique Items**: True

      - **array item** [string]

- API_ENDPOINT [string]: The endpoint of the GitHub (Enterprise) API to use. Must be overridden for github.com.

  - Example: **https://api.github.com/**

- CLIENT_ID [string] required: The registered client ID for this Red Hat Quay instance; cannot be shared with GITHUB_TRIGGER_CONFIG.

  - Reference: https://coreos.com/quay-enterprise/docs/latest/github-app.html

  - Example: **0e8dbe15c4c7630b6780**

- CLIENT_SECRET [string] required: The registered client secret for this Red Hat Quay instance.

  - Reference: https://coreos.com/quay-enterprise/docs/latest/github-app.html

  - Example: **e4a58ddd3d7408b7aec109e85564a0d153d3e846**

- GITHUB_ENDPOINT [string] required: The endpoint of the GitHub (Enterprise) being hit.

  - Example: **https://github.com/**

- ORG_RESTRICT [boolean]: If true, only users within the organization whitelist can login using this provider.

- Example: **True**

- **GITHUB_TRIGGER_CONFIG** [object, **null**]: Configuration for using GitHub (Enterprise) for build triggers.

  - Reference: https://coreos.com/quay-enterprise/docs/latest/github-build.html

  - API_ENDPOINT [string]: The endpoint of the GitHub (Enterprise) API to use. Must be overridden for github.com.

    - Example: **https://api.github.com/**

  - CLIENT_ID [string] required: The registered client ID for this Red Hat Quay instance; cannot be shared with GITHUB_LOGIN_CONFIG.

    - Reference: https://coreos.com/quay-enterprise/docs/latest/github-app.html

    - Example: **0e8dbe15c4c7630b6780**

  - CLIENT_SECRET [string] required: The registered client secret for this Red Hat Quay instance.

    - Reference: https://coreos.com/quay-enterprise/docs/latest/github-app.html

    - Example: **e4a58ddd3d7408b7aec109e85564a0d153d3e846**

  - GITHUB_ENDPOINT [string] required: The endpoint of the GitHub (Enterprise) being hit.

    - Example: **https://github.com/**

- **GITLAB_TRIGGER_CONFIG** [object]: Configuration for using Gitlab (Enterprise) for external authentication.

  - CLIENT_ID [string] required: The registered client ID for this Red Hat Quay instance.

- Example: **0e8dbe15c4c7630b6780**

  - CLIENT_SECRET [string] required: The registered client secret for this Red Hat Quay instance.

    - Example: **e4a58ddd3d7408b7aec109e85564a0d153d3e846**

    - gitlab_endpoint [string] required: The endpoint at which Gitlab(Enterprise) is running.

      - Example: **https://gitlab.com**

- GOOGLE_LOGIN_CONFIG [object, **null**]: Configuration for using Google for external authentication

  - CLIENT_ID [string] required: The registered client ID for this Red Hat Quay instance.

    - Example: **0e8dbe15c4c7630b6780**

  - CLIENT_SECRET [string] required: The registered client secret for this Red Hat Quay instance.

    - Example: e4a58ddd3d7408b7aec109e85564a0d153d3e846

- HEALTH_CHECKER [string]: The configured health check.

  - Example: **('RDSAwareHealthCheck', {'access_key': 'foo', 'secret_key': 'bar'})**

- LOG_ARCHIVE_LOCATION [string]:If builds are enabled, the storage engine in which to place the archived build logs.

  - Example: **s3_us_east**

- LOG_ARCHIVE_PATH [string]: If builds are enabled, the path in storage in which to place the archived build logs.

  - Example: **archives/buildlogs**

- MAIL_DEFAULT_SENDER [string, **null**]: If specified, the e-mail address used as the **from** when Red Hat Quay sends e-mails. If none, defaults to **support@quay.io**.

  - Example: **support@myco.com**

- MAIL_PASSWORD [string, **null**]: The SMTP password to use when sending e-mails.

  - Example: **mypassword**

- MAIL_PORT [number]: The SMTP port to use. If not specified, defaults to 587.

  - Example: **588**

- MAIL_SERVER [string]: The SMTP server to use for sending e-mails. Only required if FEATURE_MAILING is set to true.

  - Example: **smtp.somedomain.com**

- MAIL_USERNAME [string, 'null']: The SMTP username to use when sending e-mails.

  - Example: **myuser**

- **MAIL_USE_TLS** [boolean]: If specified, whether to use TLS for sending e-mails.

  - Example: **True**

- **MAXIMUM_LAYER_SIZE** [string]: Maximum allowed size of an image layer. Defaults to 20G.

  - Pattern: **^[0-9]+(G|M)$**

  - Example: **100G**

- **PUBLIC_NAMESPACES** [array]: If a namespace is defined in the public namespace list, then it will appear on **all** user's repository list pages, regardless of whether that user is a member of the namespace. Typically, this is used by an enterprise customer in configuring a set of "well-known" namespaces.

  - **Min Items**: None

  - **Unique Items**: True

    - **array item** [string]

- **PROMETHEUS_NAMESPACE** [string]: The prefix applied to all exposed Prometheus metrics. Defaults to **quay**.

  - Example: **myregistry**

- **RECAPTCHA_SITE_KEY** [string]: If recaptcha is enabled, the site key for the Recaptcha service.

- **RECAPTCHA_SECRET_KEY** [string]: 'If recaptcha is enabled, the secret key for the Recaptcha service.

- **REGISTRY_TITLE** [string]: If specified, the long-form title for the registry. Defaults to **Quay Enterprise**.

  - Example: **Corp Container Service**

- **REGISTRY_TITLE_SHORT** [string]: If specified, the short-form title for the registry. Defaults to **Quay Enterprise**.

  - Example: **CCS**

- **SECURITY_SCANNER_ENDPOINT** [string]: The endpoint for the security scanner.

  - Pattern: **^http(s)?://(.)+$**

  - Example: **http://192.168.99.101:6060**

- **SECURITY_SCANNER_INDEXING_INTERVAL** [number]: The number of seconds between indexing intervals in the security scanner. Defaults to 30.

  - Example: **30**

- **SESSION_COOKIE_SECURE** [boolean]: Whether the **secure** property should be set on session cookies. Defaults to False. Recommended to be True for all installations using SSL.

  - Example: True

  - Reference: https://en.wikipedia.org/wiki/Secure_cookies

- **SSL_PROTOCOLS** [array]: If specified, nginx is configured to enabled a list of SSL protocols defined in the list. Removing an SSL protocol from the list disables the protocol during Red Hat Quay startup.

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

- **SUPER_USERS** [array]: Red Hat Quay usernames of those users to be granted superuser privileges.

  - **Min Items**: None

  - **Unique Items**: True

    - **array item** [string]

- **TEAM_RESYNC_STALE_TIME** [string]: If team syncing is enabled for a team, how often to check its membership and resync if necessary (Default: 30m).

  - **Pattern**: **^[0-9]+(w|m|d|h|s)$**

  - **Example**: **2h**

- **USERFILES_LOCATION** [string]: ID of the storage engine in which to place user-uploaded files.

  - **Example**: **s3_us_east**

- **USERFILES_PATH** [string]: Path under storage in which to place user-uploaded files.

  - **Example**: **userfiles**

- **USER_RECOVERY_TOKEN_LIFETIME** [string]: The length of time a token for recovering a user accounts is valid. Defaults to 30m.

  - **Example**: **10m**

  - **Pattern**: **^[0-9]+(w|m|d|h|s)$**

- **V2_PAGINATION_SIZE** [number]: The number of results returned per page in V2 registry APIs.

  - **Example**: **100**

ADDITIONAL RESOURCES