Red Hat Quay 2.9

Manage Red Hat Quay

Manage Red Hat Quay

Last Updated: 2019-08-02

Red Hat Quay 2.9 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:

- Connection security with SSL and TLS certificates
- Image security scanning with Clair
- Sharing Quay images with a BitTorrent service
- Authenticating users with LDAP
- Enabling Quay for Prometheus and Grafana metrics
- Setting up geo-replication
- Troubleshooting Quay
- Upgrading Quay

CHAPTER 1. 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 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 UI or command line.

1.1. 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
```

2. Create an **openssl.cnf** file. Replacing **DNS.1** and **IP.1** with the hostname and IP of the 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
```

3. Create key and certificates. The following set of shell commands invoke the **openssl** utility to create a key for Quay, generate a request for an Authority to sign a new certificate, and finally generate a certificate for Quay signed by the CA created earlier.
Make sure the CA certificate file **rootCA.pem** and the **openssl.cnf** config file are both available.

```
$ openssl genrsa -out ssl.key 2048
$ openssl req -new -key ssl.key -out ssl.csr -subj "/CN=quay-enterprise" -config openssl.cnf
$ openssl x509 -req -in ssl.csr -CA rootCA.pem -CAkey rootCA.key -CAcreateserial -out
ssl.cert -days 356 -extensions v3_req -extfile openssl.cnf
```

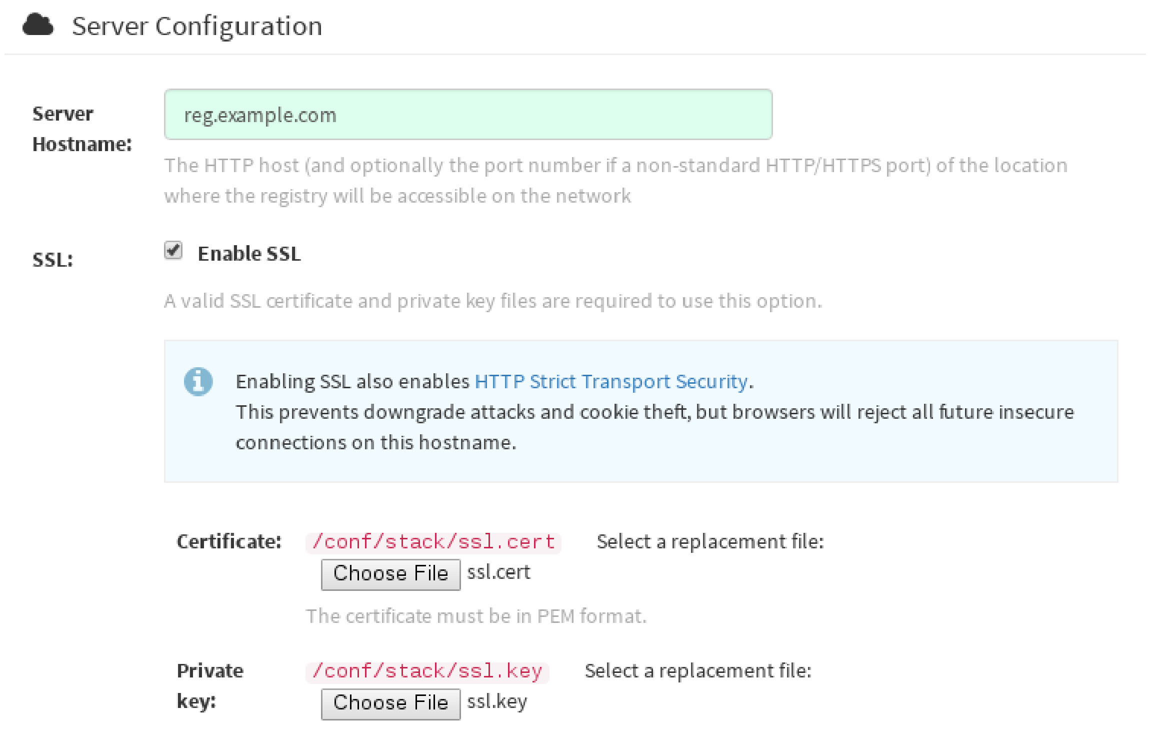
1.2. CONFIGURE QUAY TO USE THE NEW CERTIFICATE

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

1.2.1. Configure with the superuser GUI in Quay

[illegible]

1. Set the **Server Hostname** to the appropriate value and check the **Enable SSL** then upload the **ssl.key** and **ssl.cert** files:



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.

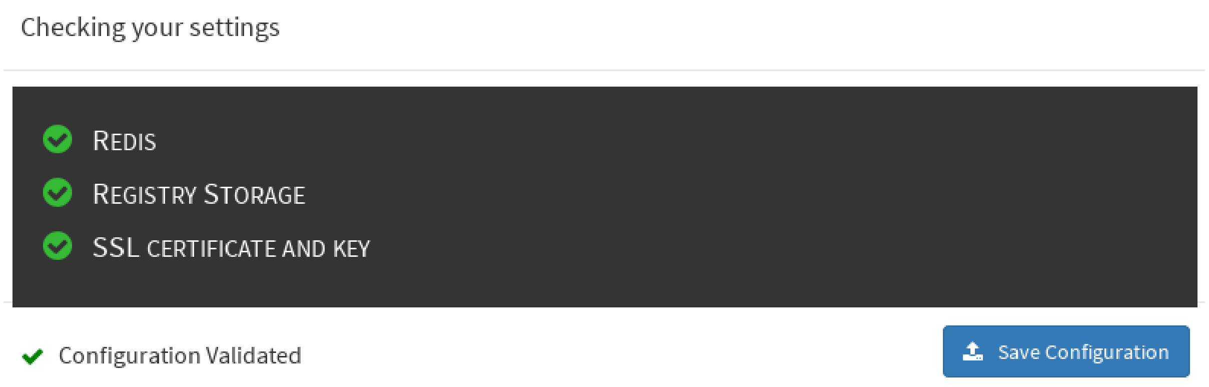
Enabling SSL also enables HTTP Strict Transport Security.
This prevents downgrade attacks and cookie theft, but browsers will reject all future insecure connections on this hostname.

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:

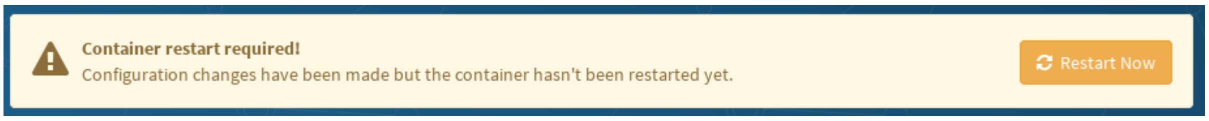


Checking your settings

✓ REDIS
✓ REGISTRY STORAGE
✓ SSL CERTIFICATE AND KEY

✓ Configuration Validated

3. Restart the container



Container restart required!
Configuration changes have been made but the container hasn't been restarted yet.

1.2.2. 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.

1. Copy the **ssl.key** and **ssl.cert** into the specified **config** directory. In this example, the config directory for 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
```

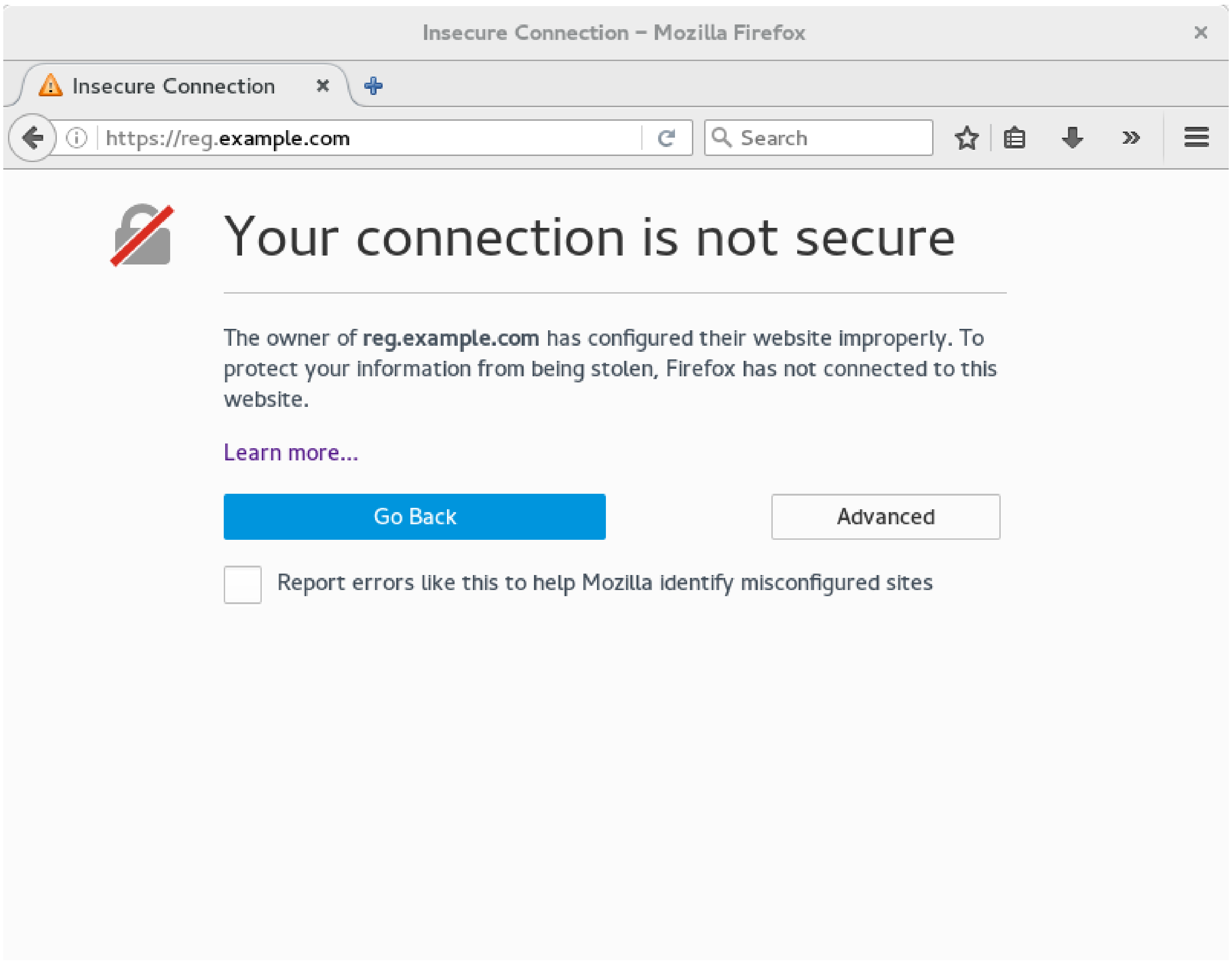
3. Restart the Red Hat Quay container:

```
$ docker ps
CONTAINER ID  IMAGE               COMMAND                CREATED        STATUS        PORTS
NAMES
eaf45a4aa12d  quay.io/quay/redis  "/usr/bin/redis-serve" 22 hours ago  Up 22 hours
0.0.0.0:6379->6379/tcp  dreamy...
cbe7b0fa39d8  quay.io/coreos/quay "/sbin/my_init"        22 hours ago  Up one hour
80/tcp,443/tcp,8443/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
```

1.2.3. Test the secure connection

Confirm the configuration by visiting the URL from a browser **https://reg.example.com/**


Insecure Connection – Mozilla Firefox

https://reg.example.com

Your connection is not secure

The owner of **reg.example.com** has configured their website improperly. To protect your information from being stolen, Firefox has not connected to this website.

Learn more...

Insecure Connection
Search
Go Back
Advanced
Report errors like this to help Mozilla identify misconfigured sites

"Your Connection is not secure" means the CA is untrusted but confirms that SSL is functioning properly. Check Google for how to configure your operating system and web browser to trust your new CA.

1.3. 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
```

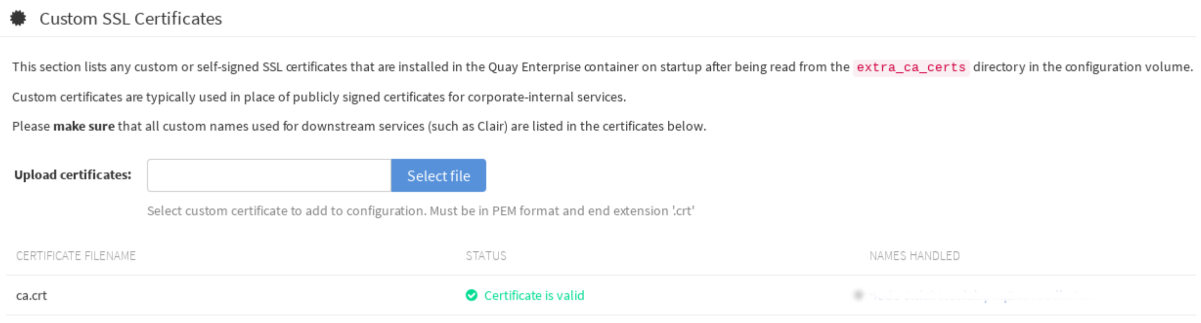
CHAPTER 2. ADDING TLS CERTIFICATES TO THE RED HAT QUAY CONTAINER

To add custom TLS certificates to Red Hat Quay, you can use either the command line interface or the Red Hat Quay user interface. From the command line, you need to create a new directory named **extra_ca_certs**/ beneath the Red Hat Quay config directory and copy any required site-specific TLS certificates to this new directory.

2.1. ADD CUSTOM/SSL CERTIFICATES FROM THE RED HAT QUAY UI

To add custom or self-signed SSL certificates to Red Hat Quay from the web UI, do the following:

1. Navigate to the Red Hat Quay config UI.
2. Scroll to the Custom SSL Certificates section.
3. In the Upload certificates box, select the filename of the certificate. The following figure shows the result of uploading a file named ca.crt.



Custom SSL Certificates

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

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

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

Upload certificates: Select file

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

CERTIFICATE FILENAME	STATUS	NAMES HANDLED
ca.crt	✓ Certificate is valid	[redacted]

2.2. 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       quay.io/coreos/quay:v2.9.5         "/sbin/my_init"         24 hours ago
Up 18 hours        0.0.0.0:80->80/tcp, 0.0.0.0:443->443/tcp, 8443/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
```

2.3. 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 Red Hat Quay config UI.

To get around this error, a base64 encoded certificate can be added to the secret *after* 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=
```

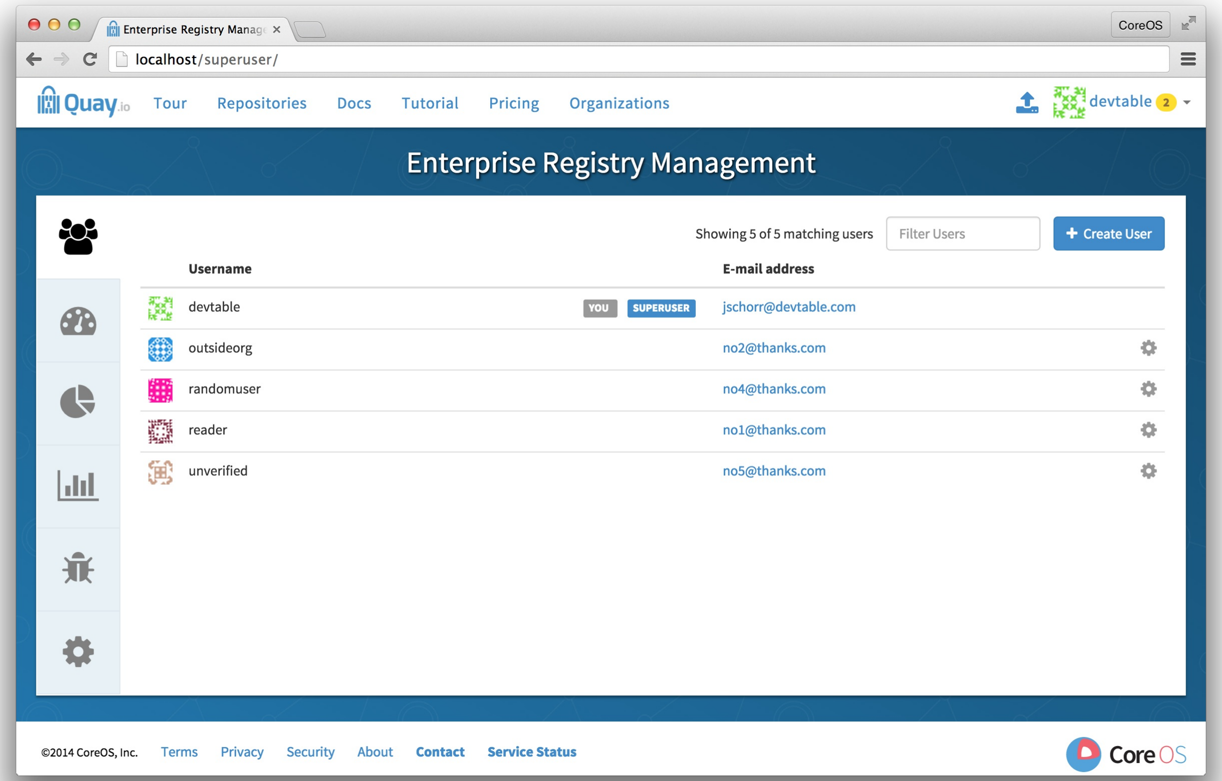
4. Finally, recycle all Red Hat Quay pods. Use **kubectl delete** to remove all Red Hat Quay pods. The Red Hat Quay Deployment will automatically schedule replacement pods with the new certificate data.

CHAPTER 3. 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 Quay.

3.1. VISIT THE MANAGEMENT PANEL

Sign in to a superuser account from the Red Hat Quay login screen. For example, if the host were reg.example.com, you would go to **http://reg.example.com/superuser** to view the management panel:



Showing 5 of 5 matching users

Username		E-mail address
devtable	YOU SUPERUSER	jschorr@devtable.com
outsideorg		no2@thanks.com
randomuser		no4@thanks.com
reader		no1@thanks.com
unverified		no5@thanks.com

3.2. ENABLE SECURITY SCANNING

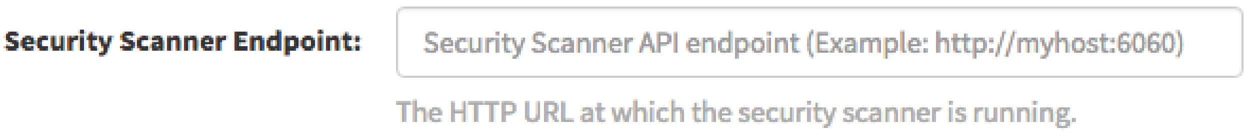
- Click the configuration tab () and scroll down to the section entitled **Security Scanner**.

Enable Security Scanning

- Check the "Enable Security Scanning" box

3.3. ENTER A SECURITY SCANNER

In the "Security Scanner Endpoint" field, enter the HTTP endpoint of a Red Hat Quay-compatible security scanner such as Clair.



Security Scanner Endpoint: Security Scanner API endpoint (Example: http://myhost:6060)

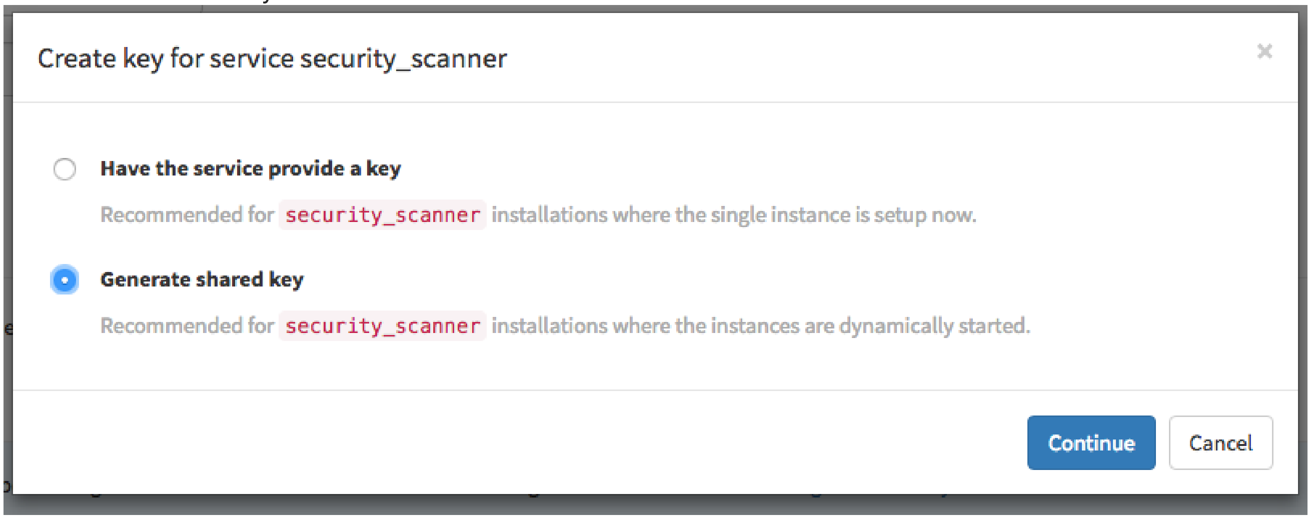
The HTTP URL at which the security scanner is running.

3.4. GENERATE AN AUTH KEY

To connect Red Hat Quay securely to the scanner, click "Create Key >" to create an authentication key between Quay and the Security Scanner.

3.4.1. Authentication for high-availability scanners

If the security scanning engine is running on multiple instances in a high-availability setup, select "Generate shared key":



Create key for service security_scanner

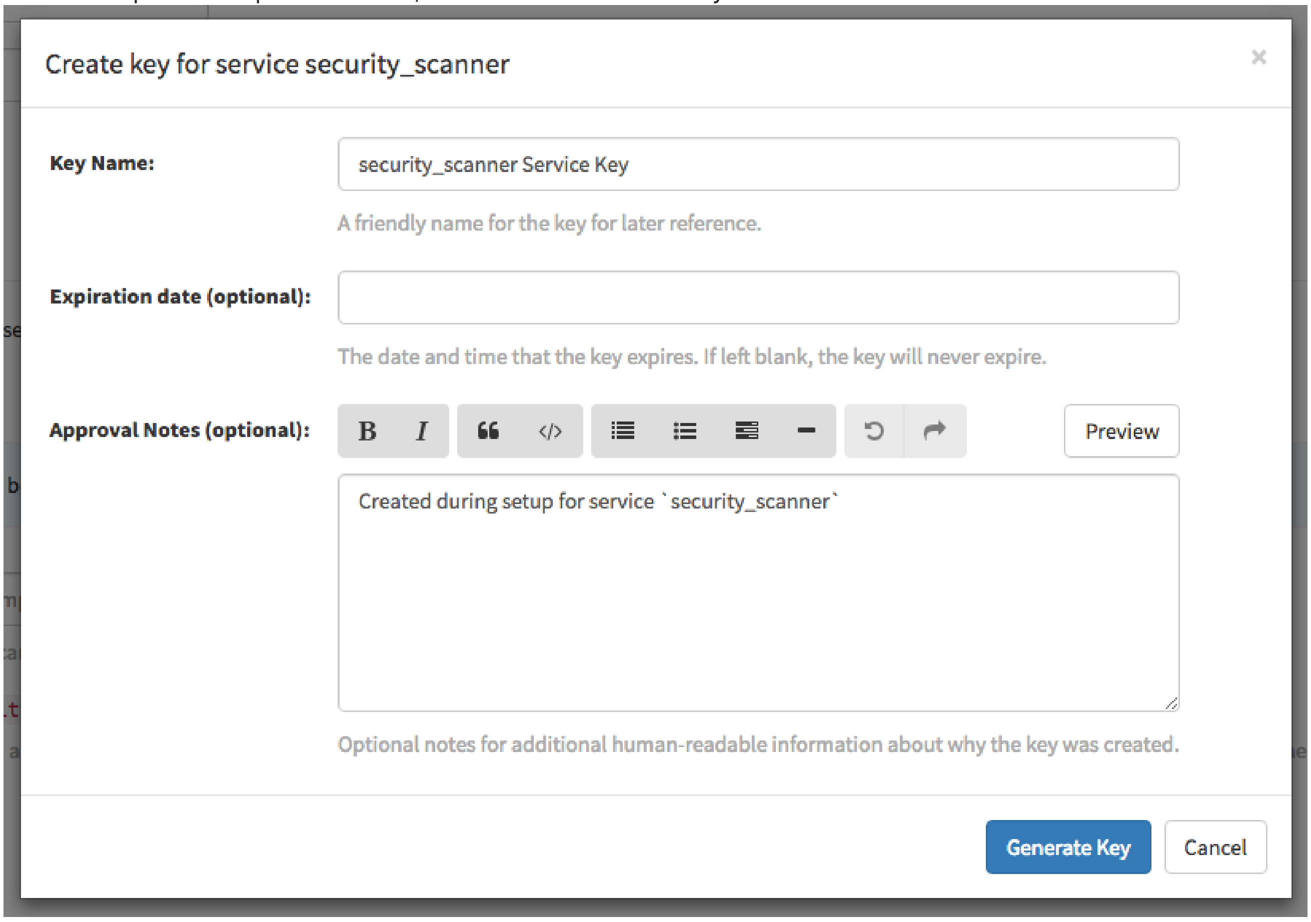
Have the service provide a key

Recommended for `security_scanner` installations where the single instance is setup now.

[✓ Generate shared key]

Recommended for `security_scanner` installations where the instances are dynamically started.

Enter an optional expiration date, and click "Generate Key":



Create key for service security_scanner

Key Name: security_scanner Service Key

A friendly name for the key for later reference.

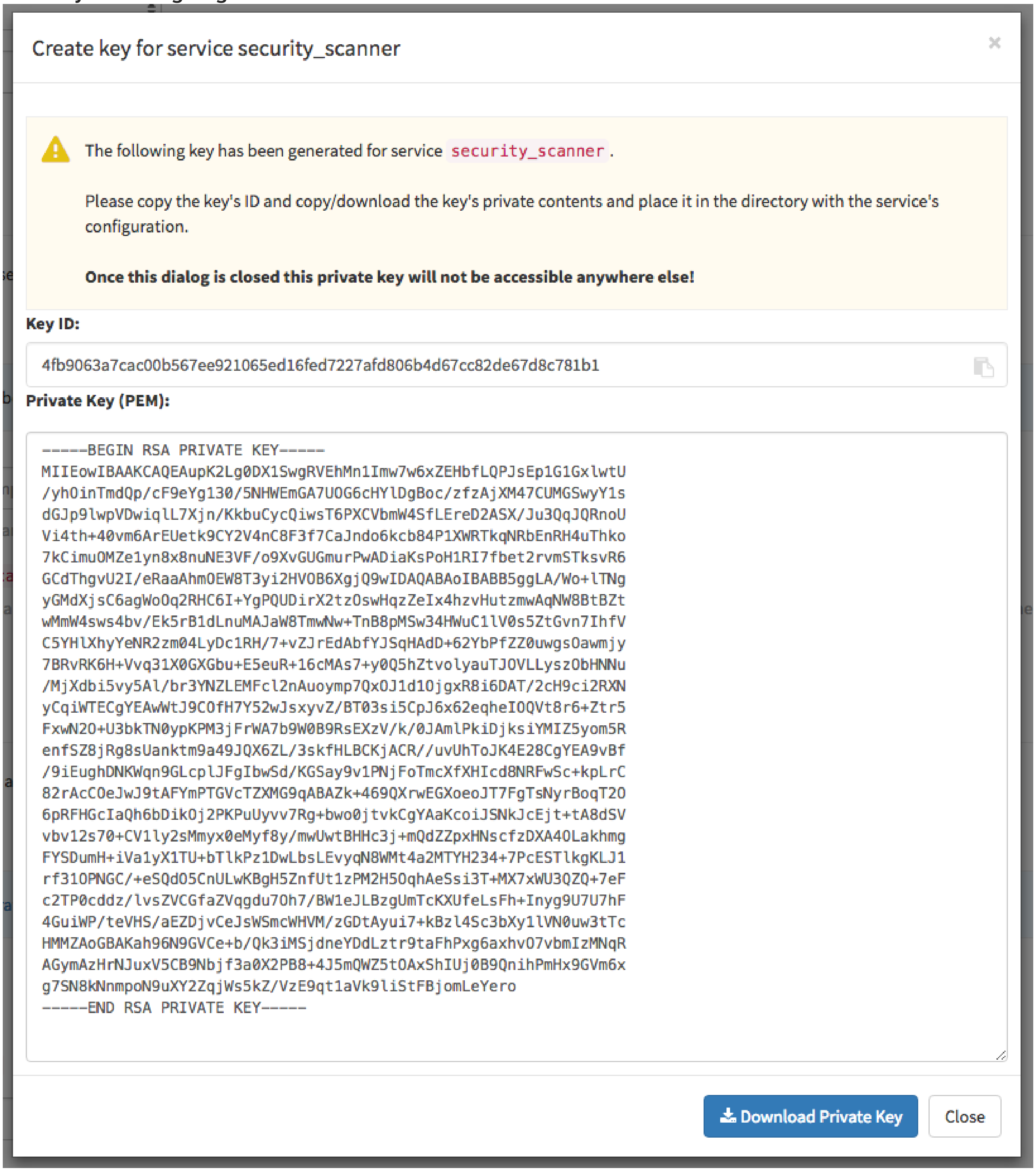
Expiration date (optional):

The date and time that the key expires. If left blank, the key will never expire.

Approval Notes (optional): Created during setup for service `security_scanner`

Optional notes for additional human-readable information about why the key was created.

Save the key ID and download the preshared private key into the configuration directory for the security scanning engine.



Create key for service security_scanner

The following key has been generated for service `security_scanner`.

Please copy the key's ID and copy/download the key's private contents and place it in the directory with the service's configuration.

Once this dialog is closed this private key will not be accessible anywhere else!

Key ID:

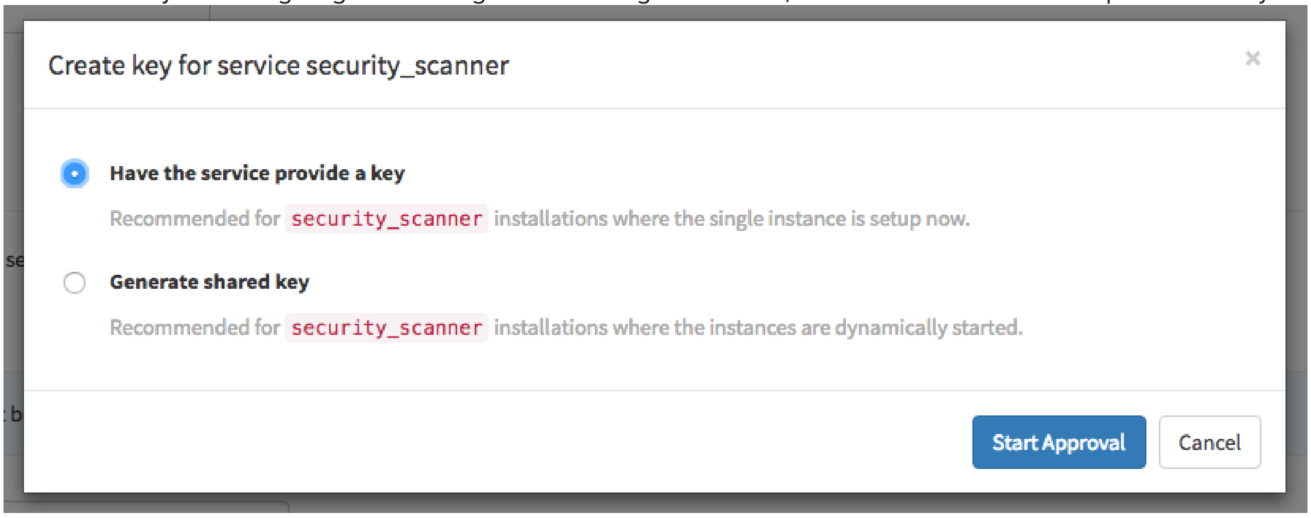
```
4fb9063a7cac00b567ee921065ed16fed7227afd806b4d67cc82de67d8c781b1
```

Private Key (PEM):

```
-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEAupK2Lg0DX1SwgRVEhMn1Imw7w6xZEHbfLQPJsEp1G1GxlwtU
/yhOinTmdQp/cF9eYg130/5NHWEmGA7UOG6cHYlDgBoc/zfzAjXM47CUMGSwyY1s
dGJp9lwpVDwiqlL7Xjn/KkbuCycQiwsT6PXCVbmW4SfLEreD2ASX/Ju3QqJQRnoU
Vi4th+40vm6ArEUetk9CY2V4nC8F3f7CaJndo6kcb84P1XWRTkqNRbEnRH4uThko
7kCimuOMZe1yn8x8nuNE3VF/o9XvGUGmurPwADiaKsPoH1RI7fbet2rvmSTksvR6
GCdThgvU2I/eRaaAhmOEW8T3yi2HVOB6XgjQ9wIDAQABAoIBABB5ggLA/Wo+lTNg
yGMdXjsC6agWoOq2RHC6I+YgPQUDirX2tzOswHqzZeIx4hzvHutzmwAqNW8BtBZt
wMmW4sws4bv/Ek5rB1dLnuMAJaW8TmwNw+TnB8pMSw34HWuC1lV0s5ZtGvn7IhfV
C5YHlXhyYeNR2zm04LyDc1RH/7+vZJrEdAbfYJSqHAdD+62YbPfZZ0uwgsOawmjy
7BRvRK6H+Vvq31X0GXGbu+E5euR+16cMAs7+y0Q5hZtvolyauTJOVLLyszObHNNu
/MjXdbi5vy5Al/br3YNZLEMFcl2nAuoymp7QxOJ1d1OjgxR8i6DAT/2cH9ci2RXN
yCqiWTECgYEAwWtJ9COfH7Y52wJsxyvZ/BT03si5CpJ6x62eqheIOQVt8r6+Ztr5
FxwN2O+U3bkTN0ypKPM3jFrWA7b9W0B9RsEXzV/k/0JAmlPkiDjksiYMIZ5yom5R
enfSZ8jRg8sUanktm9a49JQX6ZL/3skfHLBCKjACR//uvUhToJK4E28CgYEA9vBf
/9iEughDNKWqn9GLcplJFgIbwSd/KGSay9v1PNjFoTmcXfXHIcd8NRFwSc+kpLrC
82rAcCOeJwJ9tAFYmPTGVcTZXMG9qABAZk+469QXrwEGXoeoJT7FgTsNyrBoqT2O
6pRFHGcIaQh6bDikOj2PKPuUyvv7Rg+bwo0jtvkCgYAaKcoiJSNkJcEjt+tA8dSV
vbv12s70+CV1ly2sMmyx0eMyf8y/mwUwtBHHc3j+mQdZZpxHNscfzDXA4OLakhmg
FYSDumH+iVa1yX1TU+bTlkPz1DwLbsLEvyqN8WMt4a2MTYH234+7PcESTlkgKLJ1
rf31OPNGC/+eSQdO5CnULwKBgH5ZnfUt1zPM2H5OqhAeSsi3T+MX7xWU3QZQ+7eF
c2TP0cddz/lvsZVCGfaZVqgdu7Oh7/BW1eJLBzgUmTcKXUfeLsFh+Inyg9U7U7hF
4GuiWP/teVHS/aEZDjvCeJsWSmcWHVM/zGDtAyui7+kBzl4Sc3bXy1lVN0uw3tTc
HMMZAoGBAKah96N9GVCe+b/Qk3iMSjdneYDdLztr9taFhPxg6axhvO7vbmIzMNqR
AGymAzHrNJuxV5CB9Nbjf3a0X2PB8+4J5mQWZ5tOAxShIUj0B9QnihPmHx9GVm6x
g7SN8kNnmpoN9uXY2ZqjWs5kZ/VzE9qt1aVk9liStFBjomLeYero
-----END RSA PRIVATE KEY-----
```

3.4.2. Authentication for single-instance scanners

If the security scanning engine is being run on a single instance, select "Have the service provide a key":


Create key for service security_scanner

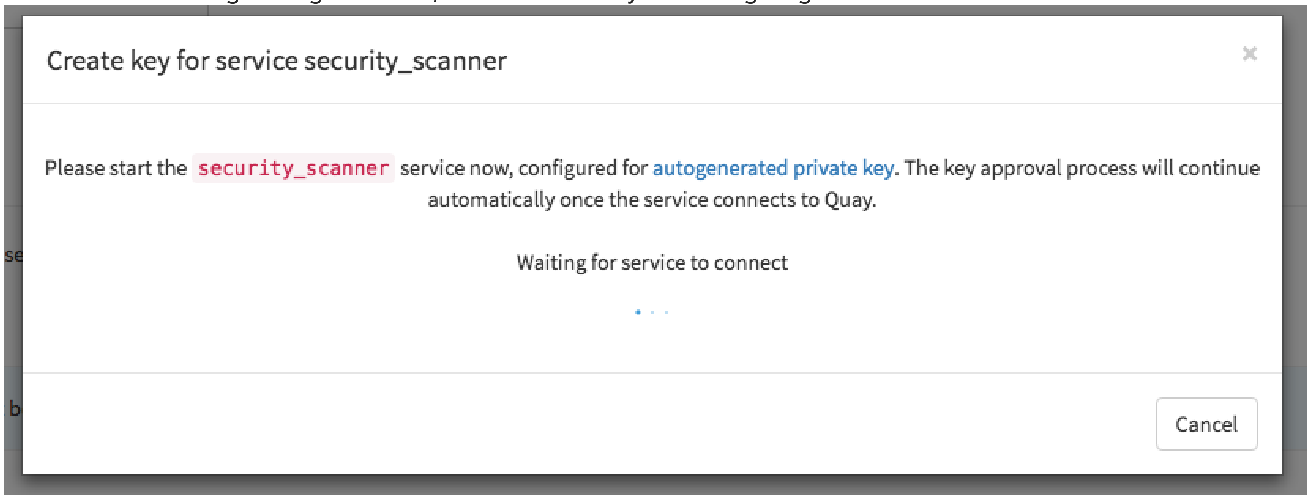
[✓ Have the service provide a key]

Recommended for `security_scanner` installations where the single instance is setup now.

Generate shared key

Recommended for `security_scanner` installations where the instances are dynamically started.

Once the following dialog is visible, run the security scanning engine:


Create key for service security_scanner

Please start the `security_scanner` service now, configured for autogenerated private key. The key approval process will continue automatically once the service connects to Quay.

Waiting for service to connect

When the security scanning engine connects, the key will be automatically approved.

3.5. SAVE CONFIGURATION

- Click "Save Configuration Changes"
- Restart the container (you will be prompted)

CHAPTER 4. SETTING UP CLAIR SECURITY SCANNING

The Clair project is an open source engine that powers Red Hat Quay Security Scanner to detect vulnerabilities in all images within Red Hat Quay, then notify developers as those issues are discovered.

Initial setup includes configuring a Postgres database, downloading the clair image and creating the Clair configuration.

4.1. GET POSTGRES AND CLAIR

In order to run Clair, a Postgres database is required. For production deployments, we recommend a PostgreSQL database running on machines other than those running Red Hat Quay and 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/coreos/clair-jwt:v2.0.7
   ```

3. Make a configuration directory for Clair

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

4.2. 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 in the config directory 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 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: /config/security_scanner.pem
```

4.2.1. 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: /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: /config/clair.key
    # crt_file: /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/
```

4.2.2. 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: /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: /config/clair.key
    # crt_file: /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/
```

4.3. CONFIGURING CLAIR FOR TLS

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

4.3.1. 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.

4.3.2. 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 **/usr/local/share/ca-certificates/** as in the example below:

NOTE

Add **--loglevel=debug** to the **docker run** command line for the clair container to enable debug level logging.

```
# docker run --restart=always -p 6060:6060 -p 6061:6061 \
  -v /path/to/clair/config/directory:/config -v \
  /path/to/quay/cert/ca.crt:/usr/local/share/ca-certificates/ca.crt  \
  quay.io/coreos/clair-jwt:v2.0.7
```

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

4.4. 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 4.1. 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 4.2. 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
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

Data source	Data collected	Whitelist links	Format	License
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

4.5. 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:/config \
   quay.io/coreos/clair-jwt:v2.0.7
```

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
```

4.6. CONTINUE WITH QUAY SETUP

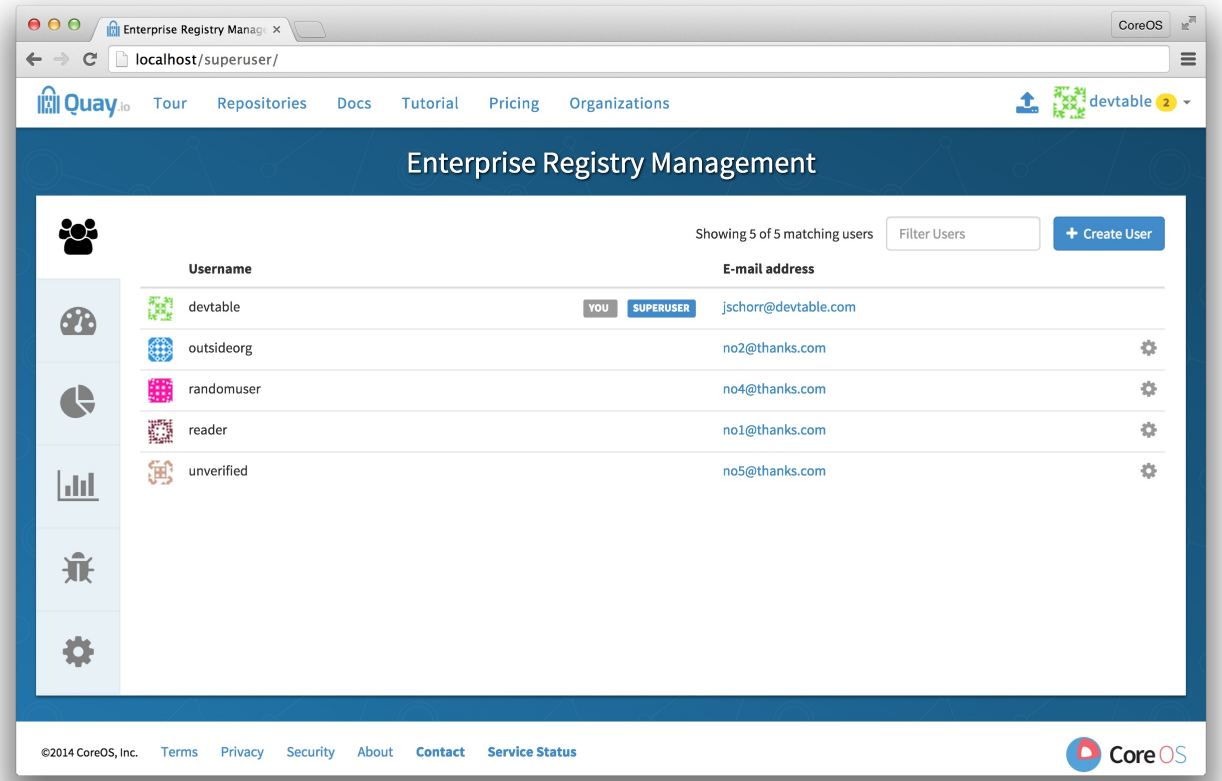
Once Clair setup is complete, continue with Red Hat Quay Security Scanning with Clair .

CHAPTER 5. DISTRIBUTING IMAGES WITH BITTORRENT

Red Hat Quay supports BitTorrent-based distribution of its images to clients via the quayctl tool. BitTorrent-based distribution allows for machines to share image data amongst themselves, resulting in faster downloads and shorter production launch times.

5.1. VISIT THE MANAGEMENT PANEL

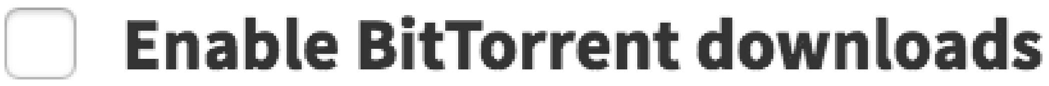
Sign in to a superuser account from the Red Hat Quay login screen. For example, if the host were reg.example.com, you would go to **http://reg.example.com/superuser** to view the management panel:



Username		E-mail address
devtable	YOU SUPERUSER	jschorr@devtable.com
outsideorg		no2@thanks.com
randomuser		no4@thanks.com
reader		no1@thanks.com
unverified		no5@thanks.com

5.2. ENABLE BITTORRENT DISTRIBUTION

- Click the configuration tab and scroll down to the section entitled **BitTorrent-based download**.



☐ Enable BitTorrent downloads

- Check the "Enable BitTorrent downloads" box

5.3. ENTER AN ANNOUNCE URL

In the "Announce URL" field, enter the HTTP endpoint of a JWT-capable BitTorrent tracker's announce URL. This will typically be a URL ending in /**announce**.

5.4. SAVE CONFIGURATION

- Click "Save Configuration Changes"
- Restart the container (you will be prompted)

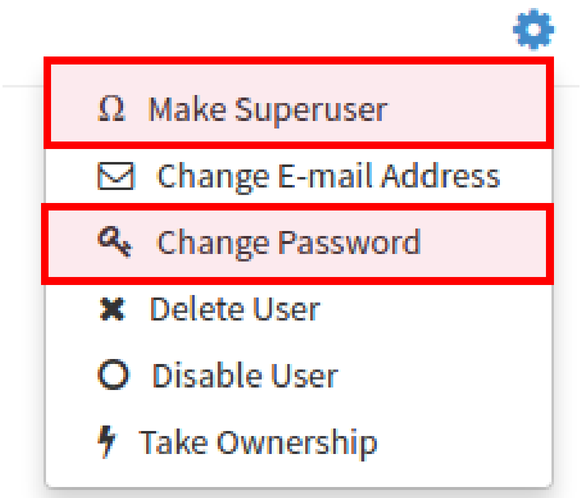
CHAPTER 6. LDAP AUTHENTICATION SETUP FOR RED HAT QUAY

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.

6.1. PREREQUISITES

The Quay LDAP setup workflow requires that the user configuring the LDAP Setup already exist in the LDAP directory. Before attempting the setup, make sure that you are logged in as a superuser that matches user crendentials in LDAP. In order to do so, Navigate to the superuser panel (ex: http(s)://quay.enterprise/superuser) and click on the “Create User” button to create a new User. Make sure to create a user that matches the username/email syntax in LDAP.

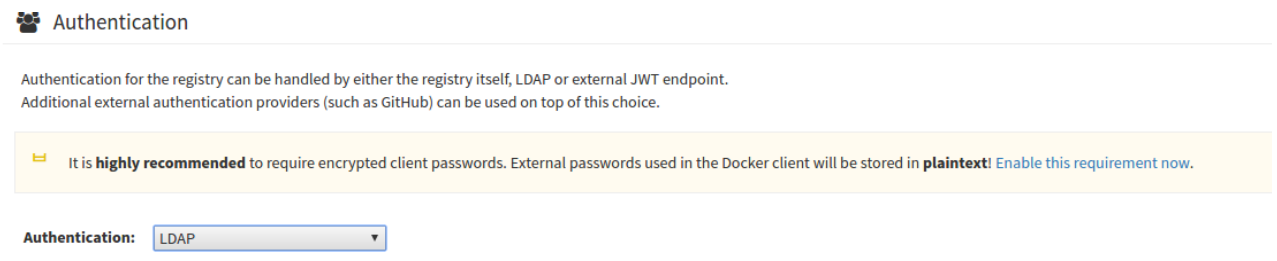
Once the user is created, click on the Settings icon next to the user and choose “Make superuser” option. For ease of troubleshooting, set the User password to the LDAP password.



You will be prompted to restart the container once the new user is created. Restart the Quay container and log in to the superuser panel ***as the user that was just created.***

6.2. SETUP LDAP CONFIGURATION

Navigate to the superuser panel and navigate to settings section. Locate the Authentication section and select “LDAP” from the drop-down menu.

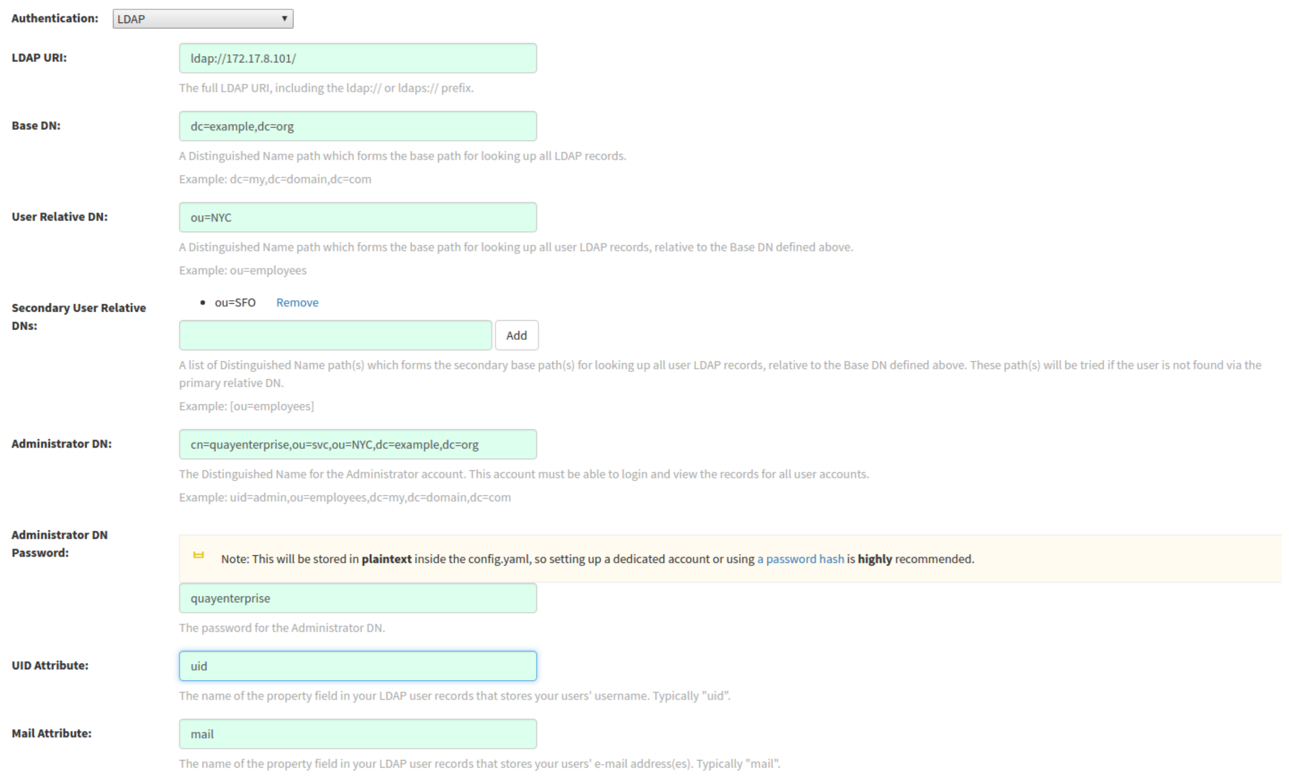

Authentication

Authentication for the registry can be handled by either the registry itself, LDAP or external JWT endpoint.
Additional external authentication providers (such as GitHub) can be used on top of this choice.

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

Authentication: LDAP

Enter LDAP configuration fields as required.



Authentication: LDAP

LDAP URI: ldap://172.17.8.101/

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

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]

Administrator DN: cn=quayenterprise,ou=svc,ou=NYC,dc=example,dc=org

The Distinguished Name for the Administrator account. This account must be able to login and view the records for all user accounts.

Example: uid=admin,ou=employees,dc=my,dc=domain,dc=com

Administrator DN Password:

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

quayenterprise

The password for the Administrator DN.

UID Attribute: uid

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

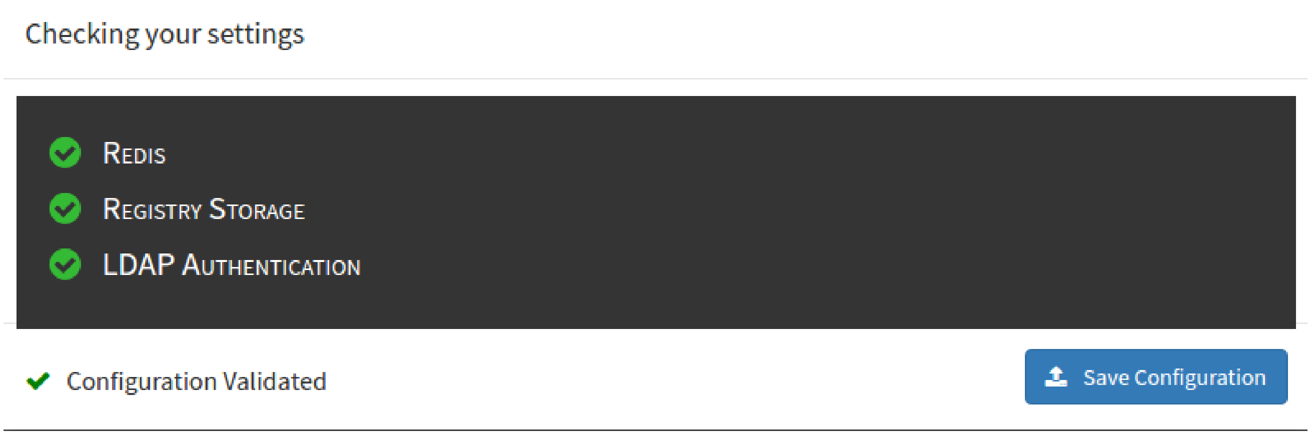
Mail Attribute: mail

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

6.3. TIPS FOR LDAP CONFIGURATION:

- LDAP URI must be in ldap:// or ldaps:// syntax. Typing a URI with ldaps:// prefix will surface the option to provide custom SSL certificate for TLS setup
- User Relative DN is relative to BaseDN (ex: ou=NYC not ou=NYC,dc=example,dc=org)
- Logged in Username must exist in User Relative DN
- You can enter multiple “Secondary User Relative DNs” if there are multiple Organizational Units where User objects are located at. (ex: ou=Users,ou=NYC and ou=Users,ou=SFO). Simply type in the Organizational Units and click on Add button to add multiple RDNs
- sAMAccountName is the UID attribute for against Microsoft Active Directory setups
- Quay searches "User Relative DN" 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**), Quay can authenticate users from both the NYC and SFO Organizational Units if the User Relative DN is set to Users (ou=Users)

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


Checking your settings

✓ Redis
✓ Registry Storage
✓ LDAP Authentication

✓ Configuration Validated

You will be prompted to login with ***LDAP credentials***.

6.4. 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.

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.

CHAPTER 7. 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.

7.1. 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.

7.1.1. 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.

7.1.2. 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.

7.1.3. 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 8. 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.

8.1. PREREQUISITES

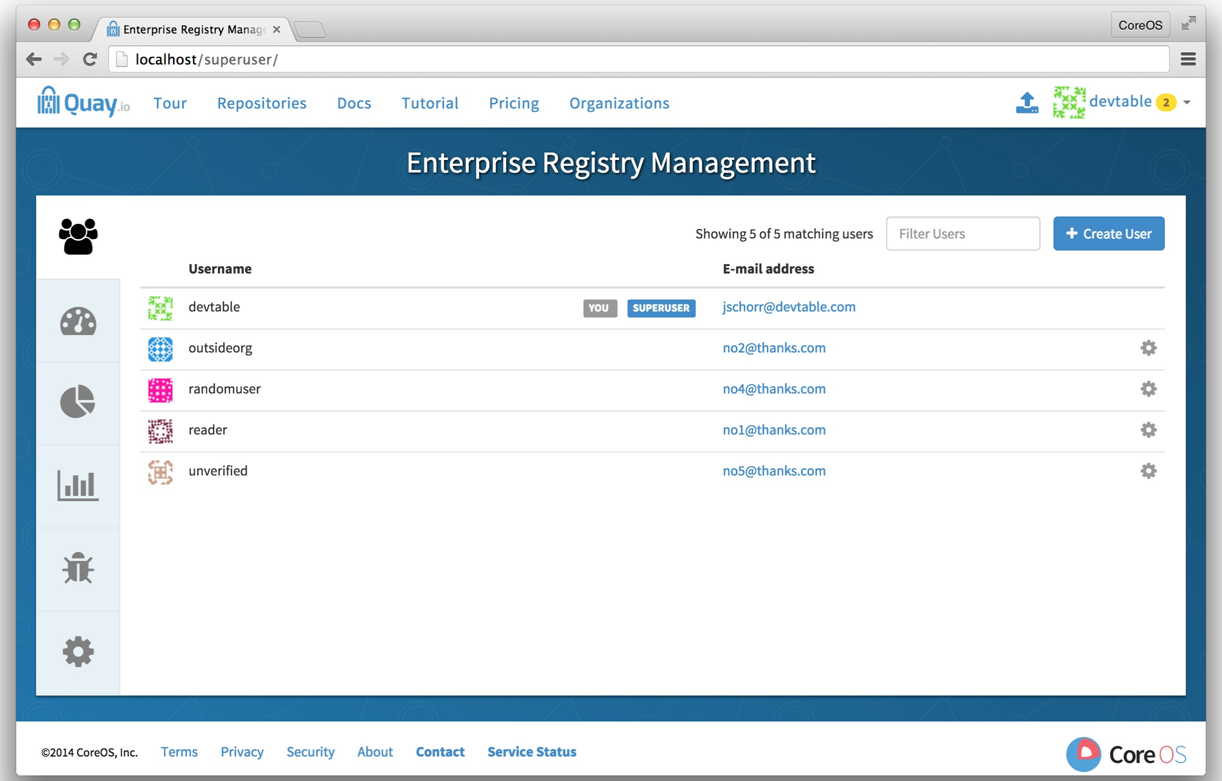
Georeplication requires that there be a high availability storage engine (S3, GCS, RADOS, Swift) in each geographic region. Further, each region must be able to access **every** storage engine due to replication requirements.

NOTE

Local disk storage is not compatible with georeplication at this time.

8.2. VISIT THE MANAGEMENT PANEL

Sign in to a superuser account from the Red Hat Quay login screen. For example, if the host were reg.example.com, you would go to **http://reg.example.com/superuser** to view the management panel:



Showing 5 of 5 matching users

Username		E-mail address
devtable	YOU SUPERUSER	jschorr@devtable.com
outsideorg		no2@thanks.com
randomuser		no4@thanks.com
reader		no1@thanks.com
unverified		no5@thanks.com

8.3. ENABLE STORAGE REPLICATION

1. Click the configuration tab and 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. Click Save to validate.
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:

   ```
   $ venv/bin/python -m util.backfillreplication
   ```

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

8.4. 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 run -d -p 443:443 -p 80:80 -v /mnt/quay/config:/conf/stack \
      -e QUAY_DISTRIBUTED_STORAGE_PREFERENCE=europestorage \
      quay.io/coreos/quay:versiontag
   ```

   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 Quay containers

CHAPTER 9. 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 10. RED HAT QUAY UPGRADE GUIDE

This document describes how to upgrade one or more Quay containers.

10.1. BACKUP THE QUAY DATABASE

The database is the "source of truth" for Quay, and some version upgrades will trigger a schema update and data migration. Such versions are clearly documented in the Red Hat Quay Release Notes .

Backup the database before upgrading Quay. Once the backup completes, use the procedure in this document to stop the running Quay container, start the new container, and check the health of the upgraded Quay service.

10.2. PROVIDE QUAY CREDENTIALS TO THE DOCKER CLIENT

```
# docker login quay.io
```

10.3. PULL THE LATEST QUAY RELEASE FROM THE REPOSITORY.

Check the list of Red Hat Quay releases for the latest version.

```
# docker pull quay.io/coreos/registry:RELEASE_VERSION
```

Replace **RELEASE VERSION** with the desired version of Quay.

10.4. FIND THE RUNNING QUAY CONTAINER ID

```
# docker ps -a
```

The Quay image will be labeled **quay.io/coreos/registry**.

10.5. STOP THE EXISTING QUAY CONTAINER

```
# docker stop QE_CONTAINER_ID
```

10.6. START THE NEW QUAY CONTAINER

```
# docker run --restart=always -p 443:443 -p 80:80 --privileged=true \
   -v /mnt/quay/config:/conf/stack \
   -v /mnt/quay/storage:/datastorage \
   -d quay.io/coreos/registry:RELEASE_VERSION
```

Replace **/local/path/to/config/directory** and **/local/path/to/storage/directory** with the absolute paths to those directories on the host. Replace **RELEASE_VERSION** with the desired Quay version.

Rarely, but occasionally, the new Quay version may perform a database schema upgrade and migration. Versions requiring such database migrations will take potentially much longer to start the first time. These versions are clearly documented in the Red Hat Quay Release Notes , which should be consulted before each Quay upgrade.

10.7. CHECK THE HEALTH OF THE UPGRADED CONTAINER

Visit the /health/endtoend endpoint on the registry hostname and verify that the code is 200 and **is_testing** is false.

10.8. UPGRADE THE REST OF THE CONTAINERS IN THE CLUSTER.

If the upgraded container is healthy, repeat this process for all remaining Quay containers.

CHAPTER 11. UPGRADING QUAY

The full list of Quay versions can be found on the Red Hat Quay Release Notes page.

11.1. SPECIAL NOTE

NOTE

If you are upgrading from a version of Quay older than 2.0.0, you **must** upgrade to Quay 2.0.0 **first**. Please follow the Upgrade to Quay 2.0.0 instructions to upgrade to Quay 2.0.0, and then follow the instructions below to upgrade from 2.0.0 to the latest version you'd like.

11.2. UPGRADING NOTE

NOTE

We **highly** recommend performing upgrades during a scheduled maintenance window, as it will require taking the existing cluster down temporarily. We are working to remove this restriction in a future release.

11.3. THE UPGRADE PROCESS

1. Visit the Red Hat Quay Release Notes page and note the latest version of Quay.
2. Shutdown the Quay cluster: Remove **all** containers from service.
3. On a **single** node, run the newer version of Quay.
4. Quay will perform any necessary database migrations before bringing itself back into service.
5. Watch the logs of the running container to determine when the upgrade has completed:

```
# docker logs -f {containerId}
```

6. Update all other nodes to refer to the new tag and bring them back into service.

CHAPTER 12. UPGRADE TO QUAY 2.0.0

All Quay instances being upgraded from versions < 2.0.0 **must** upgrade to Quay 2.0.0 first before continuing to upgrade. This upgrade has an extra step, documented here.

We **highly** recommend performing this upgrade during a scheduled maintenance window, as it will require taking the existing cluster down temporarily.

12.1. DOWNLOAD QUAY LICENSE

To begin, download your Quay License from your Tectonic Account. Please download or copy this license in **Raw Format** as a file named **license**:

12.2. SHUTDOWN ALL QUAY INSTANCES

Shutdown all running instances of Quay, across all clusters.

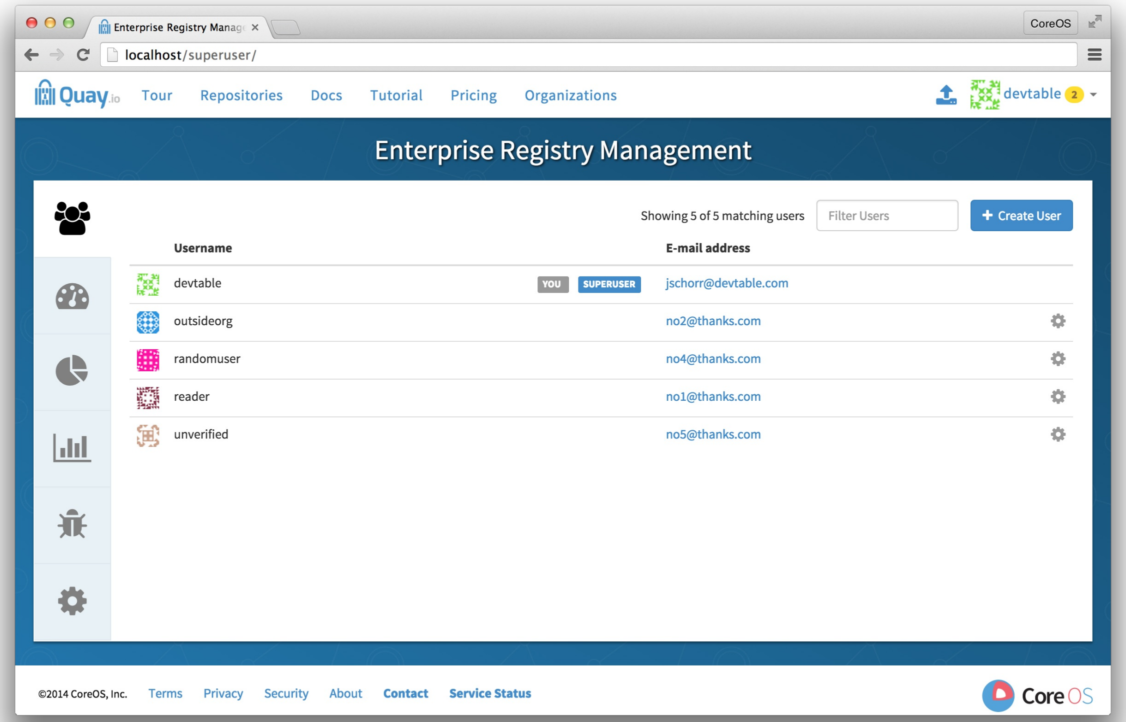
12.3. RUN A SINGLE INSTANCE OF QUAY 2

Run a single instance of Quay 2.0.0 by replacing **quay.io/coreos/registry:{currentVersion}** with **quay.io/coreos/quay:v2.0.0** in your run command, startup script, config or systemd unit.

12.3.1. Add your license to the Quay

Quay setup as a container or under Kubernetes

- Visit the management panel:



Enterprise Registry Management

Showing 5 of 5 matching users

Username		E-mail address
devtable	YOU SUPERUSER	jschorr@devtable.com
outsideorg		no2@thanks.com
randomuser		no4@thanks.com
reader		no1@thanks.com
unverified		no5@thanks.com

Sign in to a super user account from the Red Hat Quay login screen. For example, if the host were reg.example.com, you would go to **http://reg.example.com/superuser** to view the management panel:

- Click the configuration tab
- In the section entitled **License**, paste in the contents of the license downloaded above
- Click **Save Configuration Changes**
- Restart the container (you will be prompted)

12.3.2. Add license via the filesystem

Ensure the Red Hat Quay instance has been shutdown and add the raw format license in **license** file to the directory mapped to **conf/stack**, next to the existing **config.yaml**.

Example:

The **conf/stack** directory is mapped to **quay2/config** in **docker run** command used to bring up Quay:

```
docker run --restart=always -p 443:443 -p 80:80 --privileged=true -v /quay2/config:/conf/stack -v
/quay2/storage:/datastorage -d quay.io/coreos/quay:v2.0.0
```

The **license** file resides in the **quay2/config** directory:

```
$ ls quay2/config/
config.yaml  license

$ cat quay2/license
eyJhbGciOiJSUzI1NiJ9.eyJzY2hlbWFWZXJzaW9uIjoidjIiLCJ2ZXJzaW9uIjoiMSIsImNyZWF0aW9uRGF
ZSI6IjIwMTYtMTAtMjZUMTc6MjM6MjJaIiwiZXhwaXJ
[...]
```

12.4. UPDATE CLUSTER

Update all remaining Quay instances to refer to the new image (**quay.io/coreos/quay:v2.0.0**).

12.5. VERIFY CLUSTER

Verify that your cluster and its license are valid by performing a push or pull. If you receive an HTTP **402**, please make sure your license is properly installed and valid by checking in the management panel (see above for instructions).

If you encounter unusual problems, please contact support.

CHAPTER 13. 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 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 Quay. If Quay is behind SSL **at all**, this **must** be **https**.
    - **enum**: **http, https**

        - Example: **https**

- **SERVER_HOSTNAME** [string] required: The URL at which 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 Quay instance.
  - Example: **0e8dbe15c4c7630b6780**
- **CONSUMER_SECRET** [string] required: The registered consumer secret(client secret) for this Quay instance
  - **Example**: e4a58ddd3d7408b7aec109e85564a0d153d3e846

- **BITTORRENT_ANNOUNCE_URL** [string]: The URL of the announce endpoint on the bittorrent tracker.
  - Pattern: **^http(s)?://(.)+$**
  - Example: **https://localhost:6881/announce**
- **BITTORRENT_PIECE_SIZE** [number]: The bittorent piece size to use. If not specified, defaults to 512 * 1024.
  - Example: **524288**
- **BROWSER_API_CALLS_XHR_ONLY** [boolean]: If enabled, only API calls marked as being made by an XHR will be allowed from browsers. Defaults to True.
  - **Example**: False
- **CONTACT_INFO** [array]: If specified, contact information to display on the contact page. If only a single piece of contact information is specified, the contact footer will link directly.
  - **Min Items**: 1
  - **Unique Items**: True
    - **array item 0** [string]: Adds a link to send an e-mail
    - Pattern: **^mailto:(.)+$**
    - 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 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 **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 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_BITTORRENT** [boolean]: Whether to allow using Bittorrent-based pulls. Defaults to False.
  - **Reference**: https://access.redhat.com/documentation/en-us/red_hat_quay/2.9/html-single/manage_red_hat_quay/#bittorrent-based-distribution
  - 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**
- **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/2.9/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 super users 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-super users). 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 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 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 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 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 Quay instance.
    - Example: **0e8dbe15c4c7630b6780**
  - **CLIENT_SECRET** [string] required: The registered client secret for this 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 Quay instance.
    - Example: **0e8dbe15c4c7630b6780**
  - **CLIENT_SECRET** [string] required: The registered client secret for this 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 Quay sends e-mails. If none, defaults to **support@quay.io**.
  - Example: **support@myco.com**

  - 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)?://(.)+$**
  - **Example**: **http://192.168.99.101:6060**
- **SECURITY_SCANNER_INDEXING_INTERVAL** [number]: The number of seconds between indexing intervals in the security scanner. Defaults to 30.
  - **Example**: **30**
- **SESSION_COOKIE_SECURE** [boolean]: Whether the **secure** property should be set on session cookies. Defaults to False. Recommended to be True for all installations using SSL.
  - **Example**: True
  - **Reference**: https://en.wikipedia.org/wiki/Secure_cookies
- **SUPER_USERS** [array]: Quay usernames of those users to be granted superuser privileges.
  - **Min Items**: None
  - **Unique Items**: True
    - **array item** [string]
- **TEAM_RESYNC_STALE_TIME** [string]: If team syncing is enabled for a team, how often to check its membership and resync if necessary (Default: 30m).
  - **Pattern**: **^[0-9]+(w|m|d|h|s)$**
  - **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