Red Hat Quay 2.9

Manage Red Hat Quay

Manage Red Hat Quay

Last Updated: 2018-09-14

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 Chihaya BitTorrent service
- Authenticating users with LDAP
- Enabling Quay for Prometheus 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 Quay superuser GUI 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.

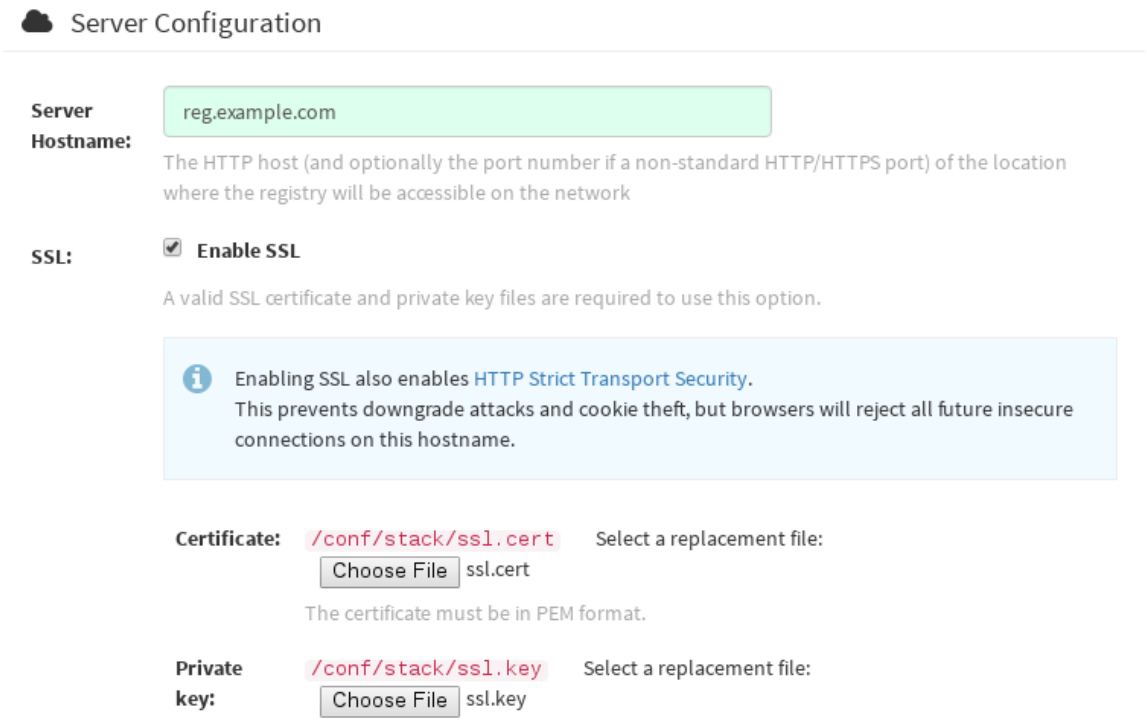
```
$ openssl genrsa -out ssl.key 2048
$ openssl req -new -key ssl.key -out ssl.csr -subj "/CN=quay-
enterprise" -config openssl.cnf
$ openssl x509 -req -in ssl.csr -CA rootCA.pem -CAkey rootCA.key -
CAcreateserial -out ssl.cert -days 356 -extensions v3_req -extfile
openssl.cnf
```

1.2. CONFIGURE QUAY TO USE THE NEW CERTIFICATE

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

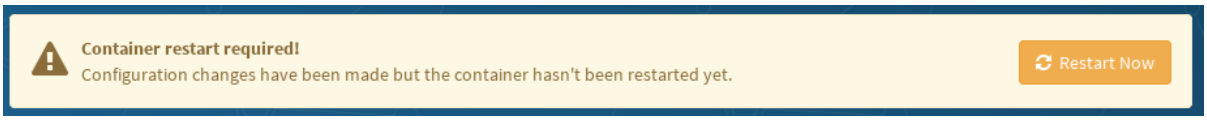
1.2.1. Configure with the superuser GUI in Quay

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



2. Save the configuration. QE will automatically validate the SSL certificate:

3. Restart the container



1.2.2. Configure with the command line

By not using the web interface the configuration checking mechanism built into QE 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.

NOTE

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

```
$ ls
ssl.cert  ssl.key
$ scp ssl.* core@10.7.8.117:/home/core/config/
[core@lan-lab-7 ~]$ ls 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
```

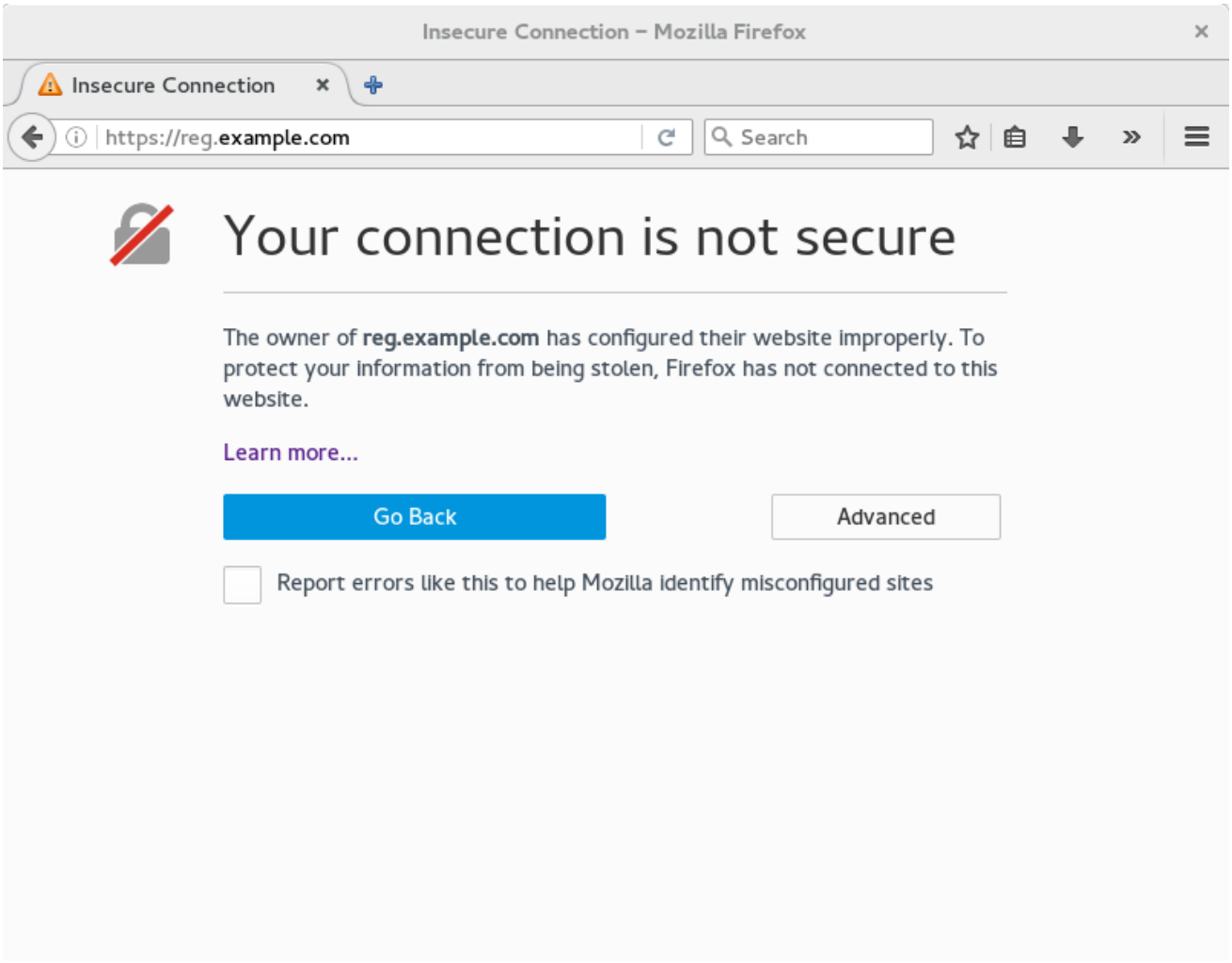
3. Restart the QE container:

```
$ 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/**



"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:c24be6d92b0a4e2bb8a8cc7c9bd044278d6abdf31534729b1660a485b1cd3
15c size: 7864
```

CHAPTER 2. 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.

2.1. ADD TLS CERTIFICATES TO 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:v2.9.3
"/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
```

2.2. 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* 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=
```

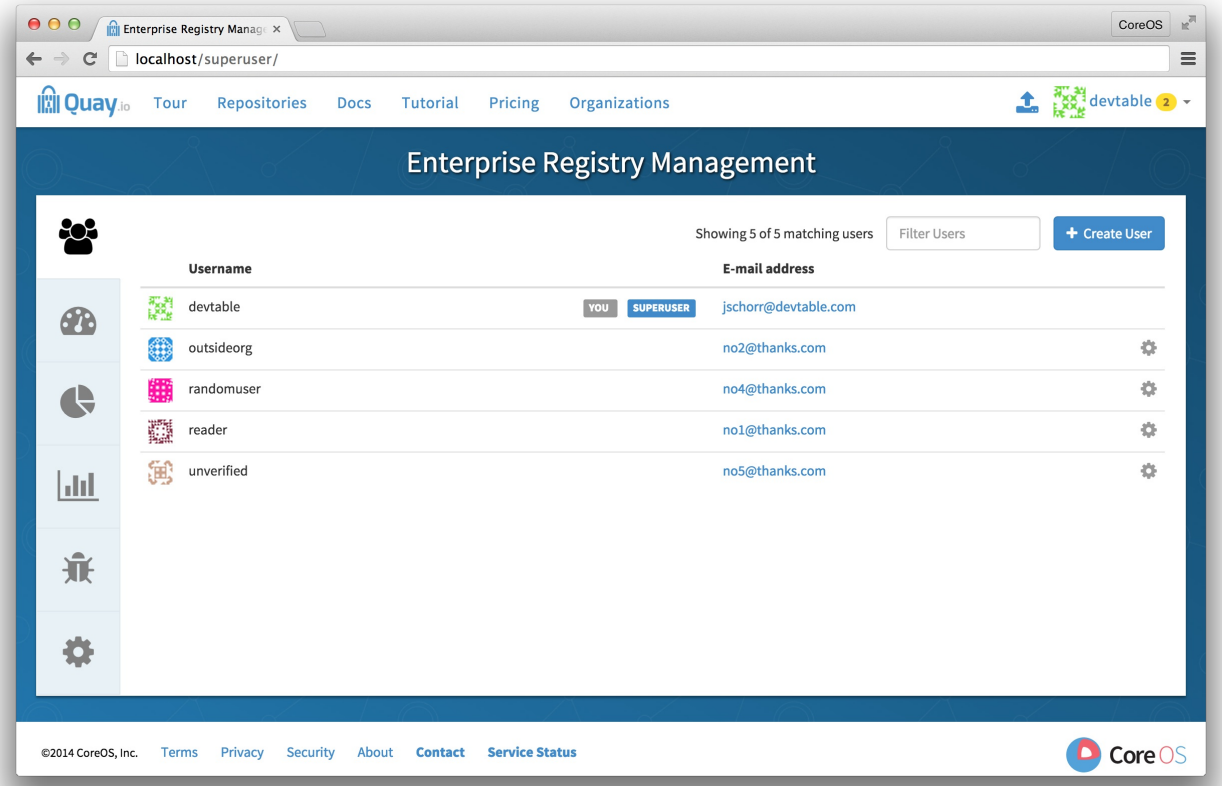
4. Finally, recycle all QE pods. Use **kubectl delete** to remove all QE pods. The QE Deployment will automatically schedule replacement pods with the new certificate data.

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.

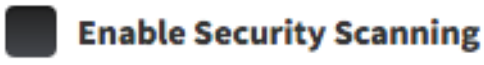
3.1. VISIT THE MANAGEMENT PANEL

Sign in to a super user account and visit `http://yourregister/superuser` to view the management panel:



3.2. ENABLE SECURITY SCANNING

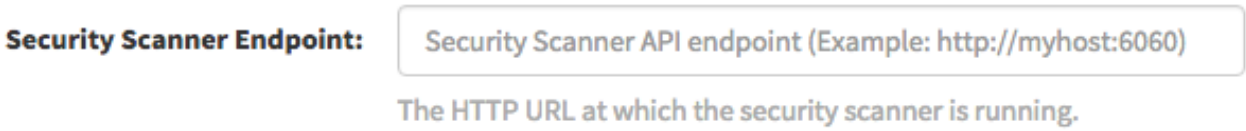
- Click the configuration tab () and scroll down to the section entitled **Security Scanner**.



- 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.

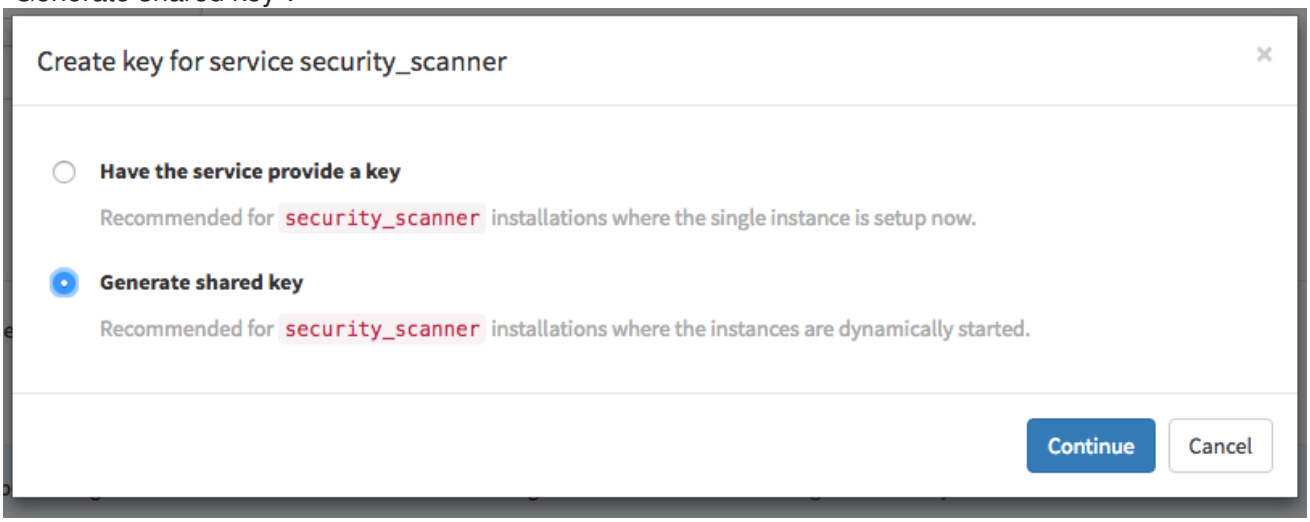


3.4. GENERATE AN AUTH KEY

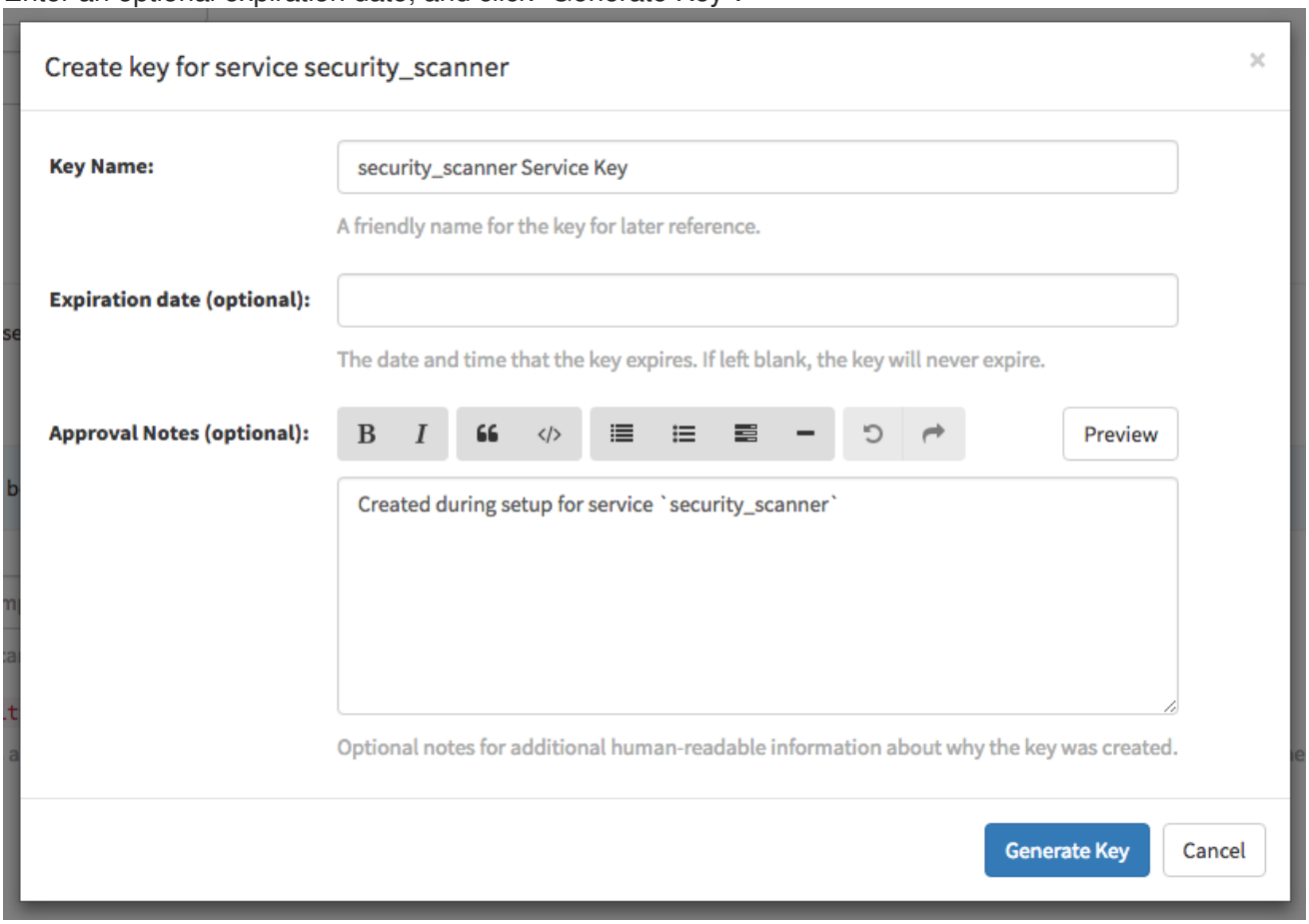
To connect Red Hat Quay securely to the scanner, click "Create Key >" to create an authentication key between Quay and the Security Scanner.

3.4.1. Authentication for high-availability scanners

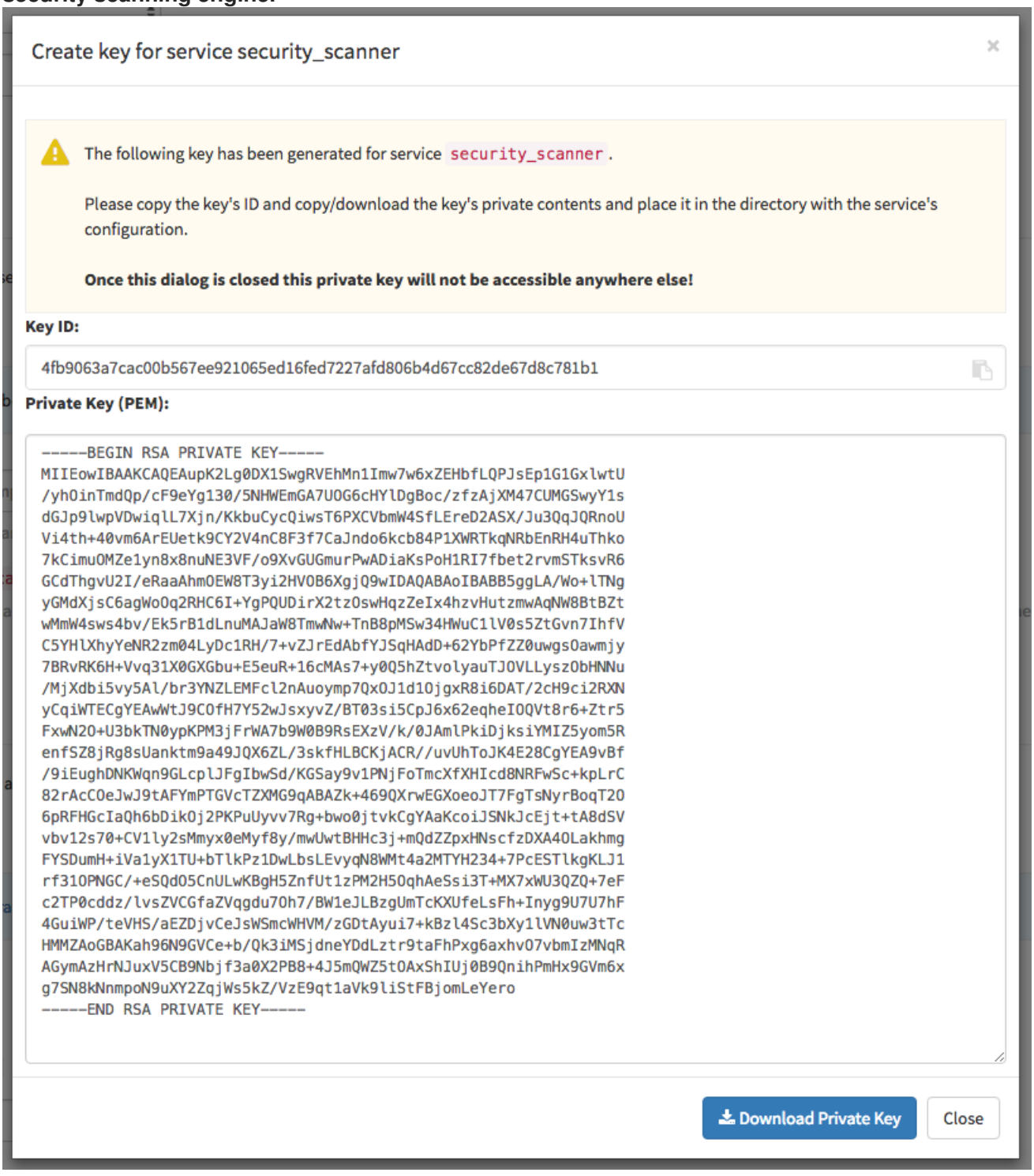
If the security scanning engine is running on multiple instances in a high-availability setup, select "Generate shared key":



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

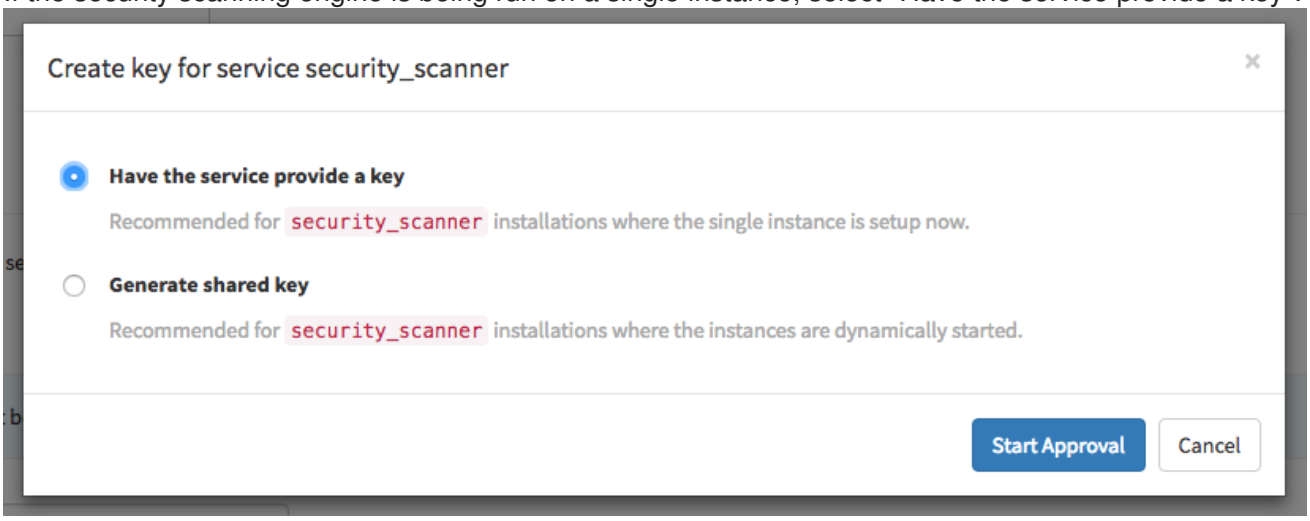


Save the key ID and download the preshared private key into the configuration directory for the security scanning engine.

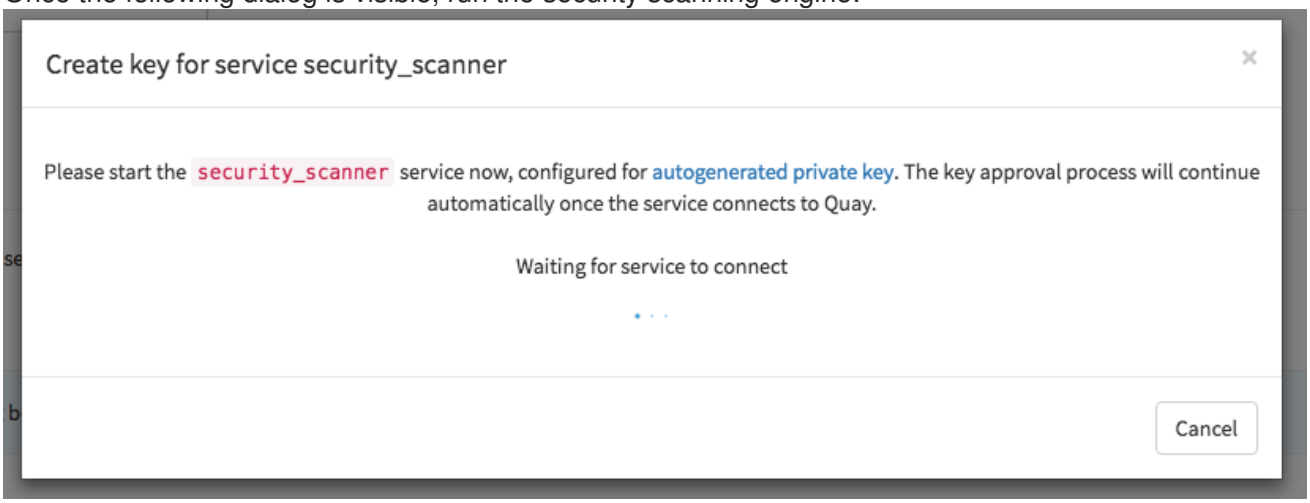


3.4.2. Authentication for single-instance scanners

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



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



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:v2.0.0
```

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.

Create a **`config.yaml`** file in the config directory with the following contents, replacing appearances of \{ VARIABLE } with the appropriate value.

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:

```
# 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:v2.0.0
```

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. 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:v2.0.0
```

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
```

4.5. CONTINUE WITH QUAY SETUP

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

CHAPTER 5. SETTING UP BITTORRENT DISTRIBUTION WITH CHIHAYA

The Chihaya project is an open source BitTorrent tracker that supports JWT-based authorization. It is the preferred tracker for making use of the secure BitTorrent-based distribution feature in Red Hat Quay.

5.1. BASIC BITTORRENT CHIHAYA CONFIGURATION

Copy the following file as **`chihaya.yaml`**, replacing **`{QE LOCATION}`** and **`{TRACKER LOCATION}`** with the reachable endpoint for the Quay instance and the tracker itself, respectively.

```
chihaya:
  announce_interval: 15m
  prometheus_addr: 0.0.0.0:6880

  http:
    addr: 0.0.0.0:6881
    allow_ip_spoofing: true
    real_ip_header: X-Forwarded-For
    read_timeout: 5s
    write_timeout: 5s
    request_timeout: 5s

  storage:
    gc_interval: 14m
    peer_lifetime: 15m
    shards: 16
    max_numwant: 50

  prehooks:
  - name: jwt
    config:
      issuer: '{QE LOCATION}'
      audience: '{TRACKER LOCATION}/announce'
      jwk_set_url: '{QE LOCATION}/keys/services/quay/keys'
      jwk_set_update_interval: 5m
```

5.2. RUNNING THE CHIHAYA SERVICE

Run the following commands to start Chihaya under a Docker container with the specified configuration mounted, making sure to point the **`chihaya.yaml`** to the file created above.

```
$ docker pull quay.io/jzelinskie/chihaya:v2.0.0-rc.1
$ docker run -p 6880-6882:6880-6882 \
  -v $PWD/chihaya.yaml:/etc/chihaya.yaml:ro \
  quay.io/jzelinskie/chihaya:v2.0.0-rc.1
```

5.3. SECURING CHIHAYA

It is recommended to place the tracker behind an SSL-terminating proxy or load balancer of some kind, especially if publicly facing. If setup this way, make sure to update the **`jwtAudience`** value in the configuration to have **`https`** as its prefix, and to refer to the load balancer.

5.4. MAKING CHIHAYA HIGHLY AVAILABLE

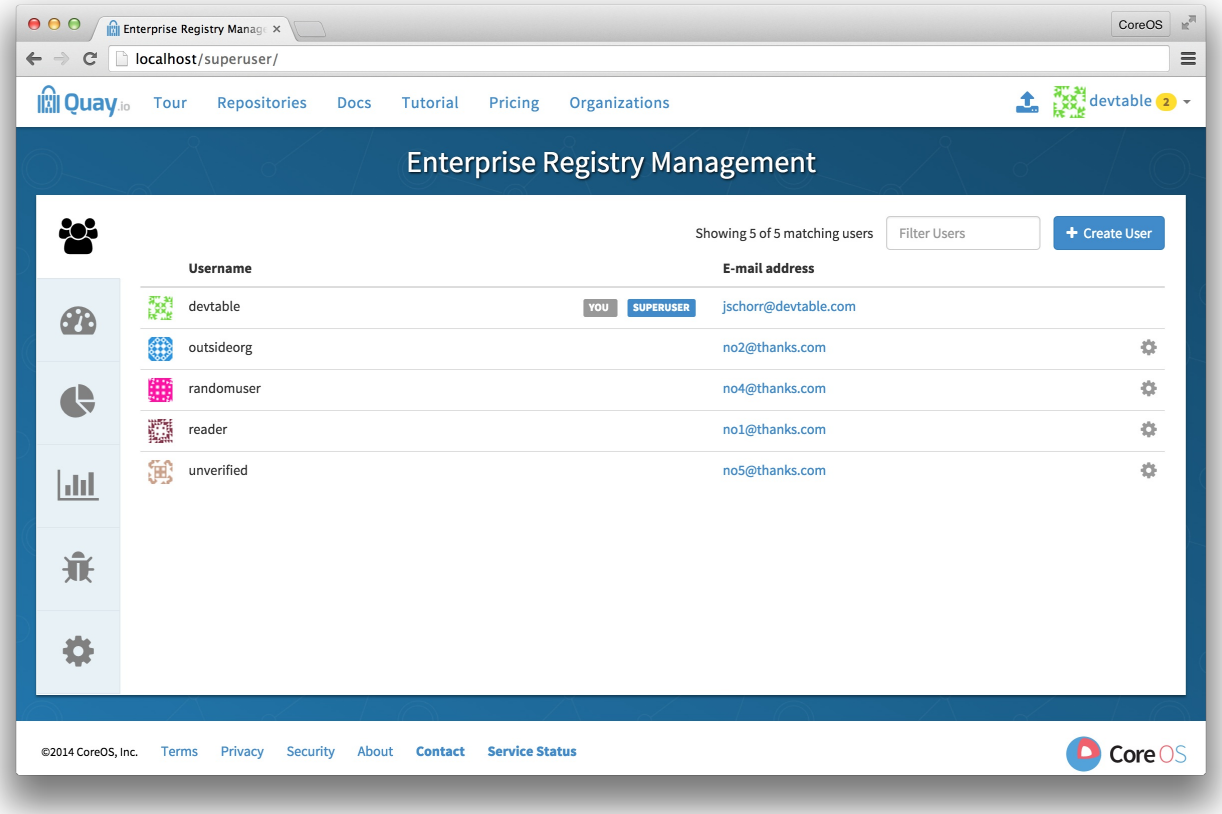
High Availability of the tracker can be handled by running 2 or more instances of the tracker, with one setup as primary and another as secondary, configured with automatic failover. A simple HTTP check can be used to ensure the health of each instance.

CHAPTER 6. 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.

6.1. VISIT THE MANAGEMENT PANEL

Sign in to a super user account and visit `http://yourregister/superuser` to view the management panel:



6.2. ENABLE BITTORRENT DISTRIBUTION

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

- Check the "Enable BitTorrent downloads" box

6.3. ENTER AN ANNOUNCE URL

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

6.4. SAVE CONFIGURATION

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

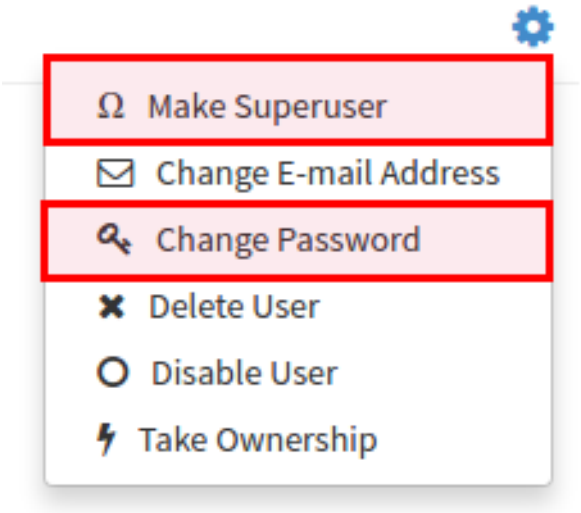
CHAPTER 7. LDAP AUTHENTICATION SETUP FOR RED HAT QUAY

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.

7.1. 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.

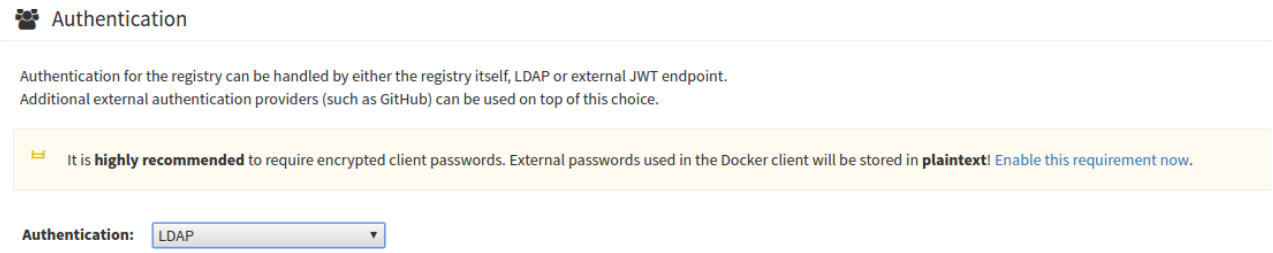
Once the user is created, click on the Settings icon next to the user and choose “Make Superuser” option. For ease of troubleshooting, set the User password to the LDAP password.



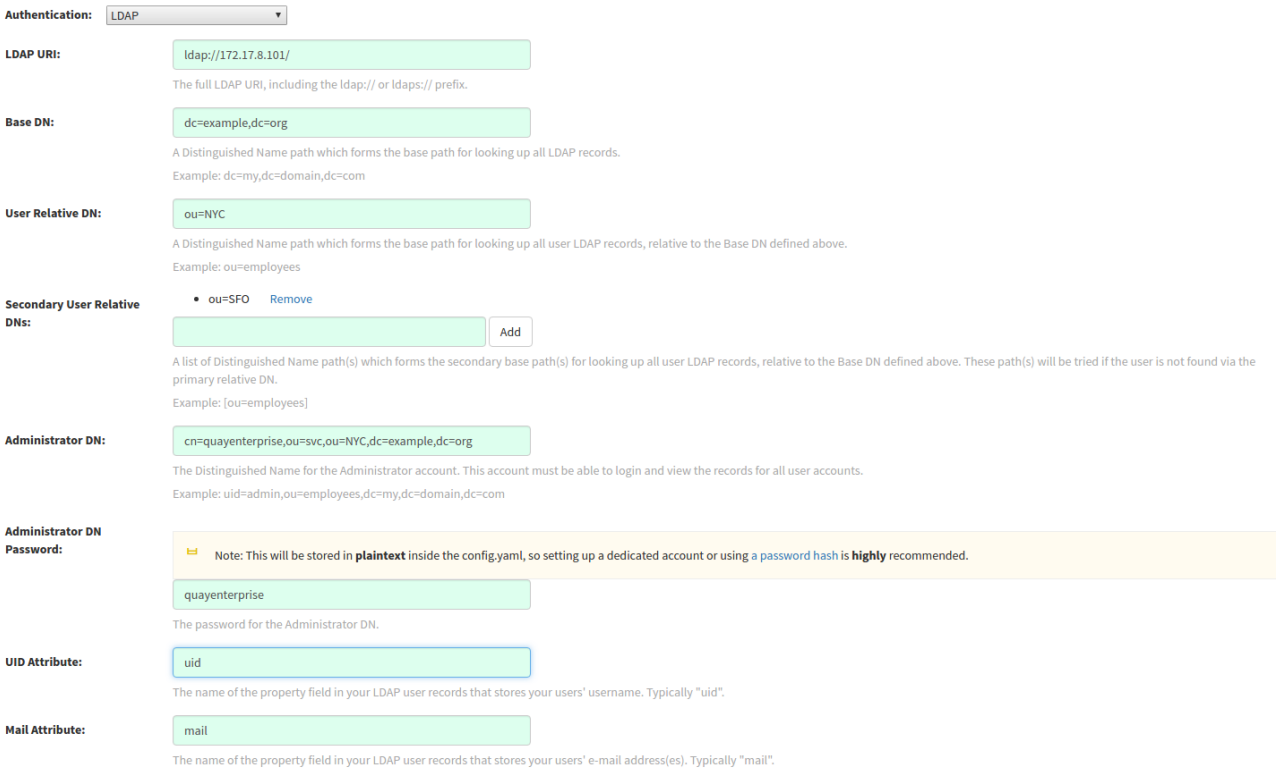
You will be prompted to restart the container once the new user is created. Restart the Quay container and log in to the Superuser panel ***as the user that was just created.***

7.2. 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.



Enter LDAP configuration fields as required.



7.3. 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.

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

7.4. 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 8. PROMETHEUS METRICS UNDER RED HAT QUAY

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

8.1. EXPOSING THE PROMETHEUS ENDPOINT

The Prometheus-compatible endpoint on the Red Hat Quay instance can be found at port **9092**. Simply add **-p 9092:9092** to the **docker run** command (or expose the port via the **Pod** configuration in Kubernetes).

8.1.1. 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.

8.1.2. 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.

8.1.3. 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.

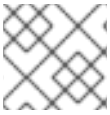
CHAPTER 9. GEOREPLICATION OF STORAGE IN RED HAT QUAY

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 QE 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.

9.1. 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.

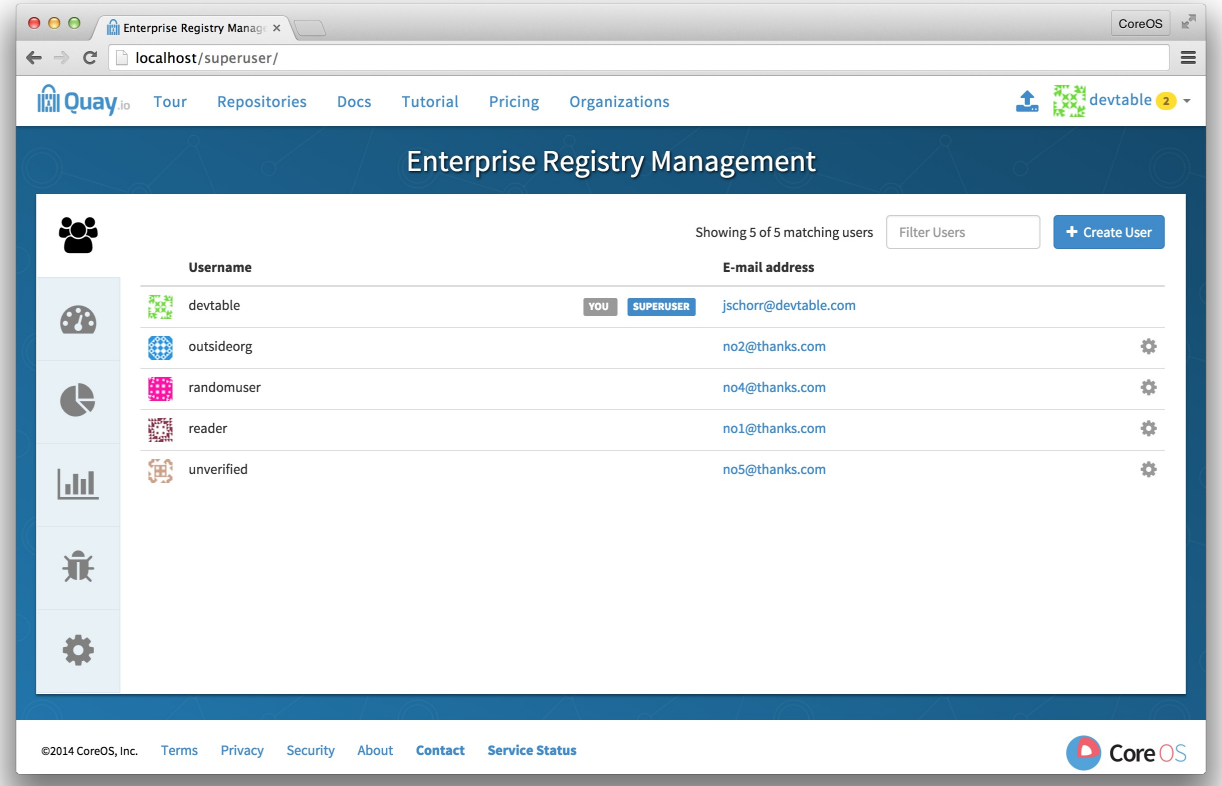


NOTE

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

9.2. VISIT THE MANAGEMENT PANEL

Sign in to a super user account and visit `http://yourregister/superuser` to view the management panel: proc_manage-security-scanning.adoc:



9.3. 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.

9.4. 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:

```
# docker run -d -p 443:443 -p 80:80 -v /conf/stack:/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 10. 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 11. RED HAT QUAY UPGRADE GUIDE

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

11.1. 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 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.

11.2. PROVIDE QUAY CREDENTIALS TO THE DOCKER CLIENT

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

11.3. PULL THE LATEST QUAY RELEASE FROM THE REPOSITORY.

Check the list of Quay releases for the latest version.

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

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

11.4. FIND THE RUNNING QUAY CONTAINER ID

```
# docker ps -a
```

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

11.5. STOP THE EXISTING QUAY CONTAINER

```
# docker stop QE_CONTAINER_ID
```

11.6. START THE NEW QUAY CONTAINER

```
# docker run --restart=always -p 443:443 -p 80:80 --privileged=true \
   -v /local/path/to/config/directory:/conf/stack \
   -v /local/path/to/storage/directory:/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 release notes, which should be consulted before each

Quay upgrade.

11.7. 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.

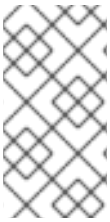
11.8. UPGRADE THE REST OF THE CONTAINERS IN THE CLUSTER.

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

CHAPTER 12. UPGRADING QUAY

The full list of Quay versions can be found on the Quay Releases page.

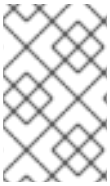
12.1. SPECIAL NOTE



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.

12.2. 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.

12.3. THE UPGRADE PROCESS

1. Visit the Quay Releases 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 13. 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.

13.1. 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`:

13.2. SHUTDOWN ALL QUAY INSTANCES

Shutdown all running instances of Quay, across all clusters.

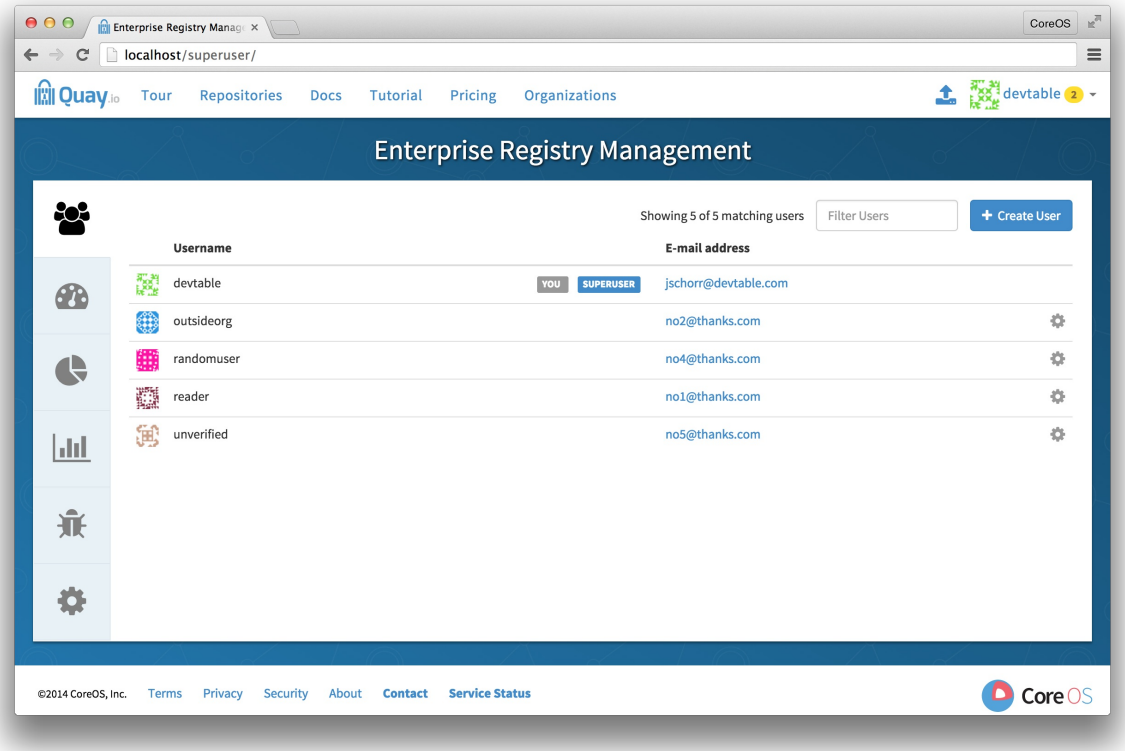
13.3. RUN A SINGLE INSTANCE OF QUAY 2

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.

13.3.1. Add your license to the Quay

Quay setup as a container or under Kubernetes

- Visit the management panel:



Sign in to a super user account and visit `http://yourregister/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)

13.3.2. Add license via the filesystem

Ensure QE 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.eyJzY2hlbWFWZXJzaW9uIjoidjIiLCJ2ZXJzaW9uIjoiMSIsImNyZ
WF0aW9uRGF0ZSI6IjIwMTYtMTAtMjZUMTc6MjM6MjJaIiwiZXhwaXJ
[...]
```

13.4. UPDATE CLUSTER

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

13.5. 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.

ADDITIONAL RESOURCES