



Red Hat Quay 3.2

Deploy Red Hat Quay on OpenShift

Deploy Red Hat Quay on OpenShift

Red Hat Quay 3.2 Deploy Red Hat Quay on OpenShift

Deploy Red Hat Quay on OpenShift

Legal Notice

Copyright © 2020 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

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

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

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

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

Node.js[®] is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack[®] Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

Abstract

Deploy Red Hat Quay on an OpenShift Cluster

Table of Contents

PREFACE	3
CHAPTER 1. OVERVIEW	4
CHAPTER 2. ARCHITECTURE	5
CHAPTER 3. PREREQUISITES FOR RED HAT QUAY ON OPENSIFT	6
CHAPTER 4. SET UP RED HAT QUAY SERVICES	7
4.1. SET UP RED HAT QUAY NAMESPACES AND SECRETS	7
4.2. ADD CLAIR IMAGE SCANNING TO RED HAT QUAY	15
4.3. ADD REPOSITORY MIRRORING RED HAT QUAY	17
CHAPTER 5. STARTING TO USE RED HAT QUAY	19
CHAPTER 6. APPENDIX A: RED HAT QUAY ON OPENSIFT CONFIGURATION FILES	20
6.1. RED HAT QUAY NAMESPACES AND SECRETS	20
6.2. RED HAT QUAY STORAGE	20
6.3. RED HAT QUAY DATABASE	21
6.4. RED HAT QUAY AUTHORIZATION	22
6.5. REDIS DATABASE	23
6.6. RED HAT QUAY CONFIGURATION POD	24
6.7. RED HAT QUAY APPLICATION CONTAINER	25
6.8. CLAIR IMAGE SCANNING	27
6.9. REPOSITORY MIRRORING	31
ADDITIONAL RESOURCES	31

PREFACE

Red Hat Quay is an enterprise-quality container registry. Use Red Hat Quay to build and store container images, then make them available to deploy across your enterprise. Red Hat is working on two approaches to deploying Red Hat Quay on OpenShift:

- **Deploy Red Hat Quay objects individually** The current procedure in this guide provides a set of yaml files that you deploy individually to set up your Red Hat Quay cluster. This procedure is currently fully supported.
- **Deploy Red Hat Quay with an Operator** The Red Hat Quay Setup Operator is being developed to provide a simpler method to deploy and manage a Red Hat Quay cluster. Although currently available as Developer Preview, the setup portion of the [Red Hat Quay Setup Operator](#) procedure is quite solid and represents the future direction of Red Hat Quay deployment on OpenShift. We strongly recommend trying the Red Hat Quay Operator for non-production uses and contributing to the project, if you are so inclined.

CHAPTER 1. OVERVIEW

Features of Red Hat Quay include:

- High availability
- Geo-replication
- Repository mirroring ([Technology Preview](#) in Red Hat Quay v3.1, supported in v3.2)
- Docker v2, schema 2 (multiarch) support
- Continuous integration
- Security scanning with Clair
- Custom log rotation
- Zero downtime garbage collection
- 24/7 support

Red Hat Quay provides support for:

- Multiple authentication and access methods
- Multiple storage backends
- Custom certificates for Quay, Clair, and storage backends
- Application registries
- Different container image types

CHAPTER 2. ARCHITECTURE

Red Hat Quay is made up of several core components.

- **Database:** Used by Red Hat Quay as its primary metadata storage (not for image storage).
- **Redis (key, value store)** Stores live builder logs and the Red Hat Quay tutorial.
- **Quay (container registry):** Runs the quay container as a service, consisting of several components in the pod.
- **Clair:** Scans container images for vulnerabilities and suggests fixes.

For supported deployments, you need to use one of the following types of storage:

- **Public cloud storage:** In public cloud environments, you should use the cloud provider's object storage, such as Amazon S3 (for AWS) or Google Cloud Storage (for Google Cloud).
- **Private cloud storage:** In private clouds, an S3 or Swift compliant Object Store is needed, such as Ceph RADOS, or OpenStack Swift.

Do not use "Locally mounted directory" Storage Engine for any production configurations. Mounted NFS volumes are not supported. Local storage is meant for Red Hat Quay test-only installations.

CHAPTER 3. PREREQUISITES FOR RED HAT QUAY ON OPENSIFT

Here are a few things you need to know before you begin the Red Hat Quay on OpenShift deployment:

- **OpenShift cluster:** You need a privileged account to an OpenShift 3.x or 4.x cluster on which to deploy the Red Hat Quay. That account must have the ability to create namespaces at the cluster scope. To use Red Hat Quay builders, OpenShift 3 is required.
- **Storage:** AWS cloud storage is used as an example in the following procedure. As an alternative, you can create Ceph cloud storage using steps from the [Set up Ceph](#) section of the high availability Red Hat Quay deployment guide. The following is a list of other supported cloud storage:
 - Amazon S3 (see [S3 IAM Bucket Policy](#) for details on configuring an S3 bucket policy for Red Hat Quay)
 - Azure Blob Storage
 - Google Cloud Storage
 - Ceph Object Gateway (RADOS)
 - OpenStack Swift
 - CloudFront + S3
 - NooBaa S3 Storage (See [Configuring Red Hat OpenShift Container Storage for Red Hat Quay](#), currently [Technology Preview](#))
- **Services:** The OpenShift cluster must have enough capacity to run the following containerized services:
 - **Database:** We recommend you use an enterprise-quality database for production use of Red Hat Quay. PostgreSQL is used as an example in this document. Other options include:
 - **Crunchy Data PostgreSQL Operator:** Although not supported directly by Red Hat, the [CrunchDB Operator](#) is available from [Crunchy Data](#) for use with Red Hat Quay. If you take this route, you should have a support contract with Crunchy Data and work directly with them for usage guidance or issues relating to the operator and their database.
 - If your organization already has a high-availability (HA) database, you can use that database with Red Hat Quay. See the [Red Hat Quay Support Policy](#) for details on support for third-party databases and other components.
 - **Key-value database:** Redis is used to serve live builder logs and Red Hat Quay tutorial content to your Red Hat Quay configuration.
 - **Red Hat Quay:** The quay container provides the features to manage the Red Hat Quay registry.

CHAPTER 4. SET UP RED HAT QUAY SERVICES

Deploying Red Hat Quay on OpenShift requires you to create a set of yaml files. Although the **oc** command is used to configure the Red Hat Quay registry here, you could use the OpenShift web UI instead, if you prefer.

Refer to Appendix A for the contents of these yaml files.

Here are a few things to keep in mind:

- Your OpenShift account must have permission to create namespaces at the cluster scope.
- Red Hat Quay runs under its own namespace inside a Kubernetes cluster, so that needs to be created first. You can create it through the **New project** in the OpenShift web console or using `quay-enterprise-namespace.yaml` (as described here).
- You need a working enterprise-quality database. In our example, we illustrate PostgreSQL (version 9.4 or above is required, although we recommend 9.6).
- You can use an existing Redis service (needed for build logs and the Red Hat Quay tutorial) or start one as described in this procedure.

Here are the major steps, detailed below, to complete a Red Hat Quay deployment on OpenShift:

1. Set up the Red Hat Quay namespace and secrets
2. Create the Red Hat Quay database
3. Create Red Hat Quay roles and privileges
4. Create the Redis deployment
5. Prepare to configure Red Hat Quay
6. Start the Red Hat Quay configuration user interface
7. Deploy the Red Hat Quay configuration
8. Add Clair image scanning
9. Add repository mirroring

4.1. SET UP RED HAT QUAY NAMESPACES AND SECRETS

1. **Get Red Hat Quay yaml files** Create a set of yaml files in a directory on your local system from the contents shown in Appendix A. Study each file to determine where you might need to make modifications. You will use **oc create** to create the needed resources from those files.
2. **Log in with oc cli** Login as a user with cluster scope permissions to the OpenShift cluster. For example:

```
$ oc login -u system:admin
```

3. **Create namespace.** Run **oc create quay-enterprise-namespace.yaml** and then make **quay-enterprise** the current project. All objects will be deployed to this namespace/project:

```
$ oc create -f quay-enterprise-namespace.yaml
namespace "quay-enterprise" created
$ oc project quay-enterprise
```

4. **Create the secret for the Red Hat Quay configuration and app** Create the following secrets. During Red Hat Quay configuration, the `config.yaml`, and optionally the `ssl.cert` and `ssl.key`, files are added to the application's secret, so they can be included with the resulting Red Hat Quay application:

```
$ oc create -f quay-enterprise-config-secret.yaml
secret/quay-enterprise-config-secret created
$ oc create secret generic quay-enterprise-secret
```

5. **Create the secret for quay.io** This pull secret provides credentials to pull containers from the Quay.io registry. Refer to [Accessing Red Hat Red Hat Quay](#) to get the credentials you need to add to the `quay-enterprise-redhat-quay-pull-secret.yaml` file, then run **oc create**:

```
$ oc create -f quay-enterprise-redhat-quay-pull-secret.yaml
secret/redhat-quay-pull-secret created
```

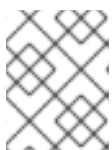
6. **Create the database.** If you are not using your own enterprise-quality database (recommended), this procedure illustrates how to set up a Postgresql database on an OpenShift cluster. This entails creating AWS storage, a postgres deployment, and postgres service, then adding an extension to the database (see the description of **quay-storageclass.yaml** in Appendix A for information on adding encryption to your volumes):

```
$ oc create -f quay-storageclass.yaml
storageclass.storage.k8s.io/quay-storageclass created
$ oc create -f db-pvc.yaml
persistentvolumeclaim/postgres-storage created
$ oc create -f postgres-deployment.yaml
deployment.extensions/postgres-new created
$ oc create -f postgres-service.yaml
service/postgres created
```

```
$ oc get pods -n quay-enterprise
NAME                                READY STATUS RESTARTS AGE
postgres-xxxxxxxx-xxxxx             1/1   Running 0      3m26s
```

Run the following command, replacing the name of the postgres pod with your pod:

```
$ oc exec -it postgres-xxxxxxxx-xxxxx -n quay-enterprise -- /bin/bash -c 'echo "CREATE EXTENSION IF NOT EXISTS pg_trgm" | /opt/rh/rh-postgresql10/root/usr/bin/psql -d quay'
```



NOTE

The **-d database_name** must not be omitted. If it is, the extension will be created on the default PostgreSQL database.

7. **Create a serviceaccount for the database** Create a serviceaccount and grant it anyuid privilege. Running the PostgreSQL deployment under anyuid lets you add persistent storage to the deployment and allow it to store db metadata.

```
# oc create serviceaccount postgres -n quay-enterprise
serviceaccount/postgres created
# oc adm policy add-scc-to-user anyuid -z system:serviceaccount:quay-enterprise:postgres \
scc "anyuid" added to: ["system:serviceaccount:quay-enterprise:system:serviceaccount:quay-
enterprise:postgres"]
```

8. **Create the role and the role binding** Red Hat Quay has native Kubernetes integrations. These integrations require Service Account to have access to the Kubernetes API. When Kubernetes RBAC is enabled, Role Based Access Control policy manifests also have to be deployed. This role will be used to run Red Hat Quay and also to write the config.yaml file that Red Hat Quay creates at the end of the web interface setup:

```
$ oc create -f quay-servicetoken-role-k8s1-6.yaml
$ oc create -f quay-servicetoken-role-binding-k8s1-6.yaml
```

9. **Add privilege:** Make sure that the service account has root privileges, because Red Hat Quay runs strictly under root (this will be changed in the future versions). Throughout this example, the namespace is **quay-enterprise**:

```
$ oc adm policy add-scc-to-user anyuid \
system:serviceaccount:quay-enterprise:default
```

10. **Create Redis deployment** If you haven't already deployed Redis, create a **quay-enterprise-redis.yaml** file and deploy it:

```
$ oc create -f quay-enterprise-redis.yaml
```

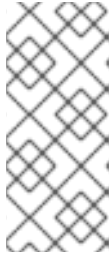
11. **Set up to configure Red Hat Quay** Red Hat Quay V3 added a tool for configuring the Red Hat Quay service before deploying it. Although the config tool is in the same container as the full Red Hat Quay service, it is deployed in a different way, as follows:

```
$ oc create -f quay-enterprise-config.yaml
$ oc create -f quay-enterprise-config-service-clusterip.yaml
$ oc create -f quay-enterprise-config-route.yaml
```

The quay configuration container is now set up to be accessed from port 443 from your Web browser. Before creating the configuration, however, you need to create a route to the permanent Red Hat Quay service. This is because we need the Red Hat Quay service's publicly available FQDN when setting up the application.

12. **Start the Red Hat Quay application** Identify the Red Hat Quay Kubernetes service and create a route for it, then start the Red Hat Quay application as follows:

```
$ oc create -f quay-enterprise-service-clusterip.yaml
service/quay-enterprise-clusterip created
$ oc create -f quay-enterprise-app-route.yaml
route.route.openshift.io/quay-enterprise created
$ oc create -f quay-enterprise-app-rc.yaml
deployment.extensions/quay-enterprise-app created
```

**NOTE**

The creation of the Red Hat Quay application (quay-enterprise-app pod) will not complete until you have finished configuring the application. So don't worry if you see that pod remain in "ContainerCreating" status until the configuration is done. At that point, the new configuration is fed to the application and it will change to the "Running" state.

You will need to know the route to the Red Hat Quay application when you do the configuration step.

- Begin to configure Red Hat Quay.** Open the public route to the Red Hat Quay configuration container in a Web browser. To see the route to the quay configuration service, type the following:

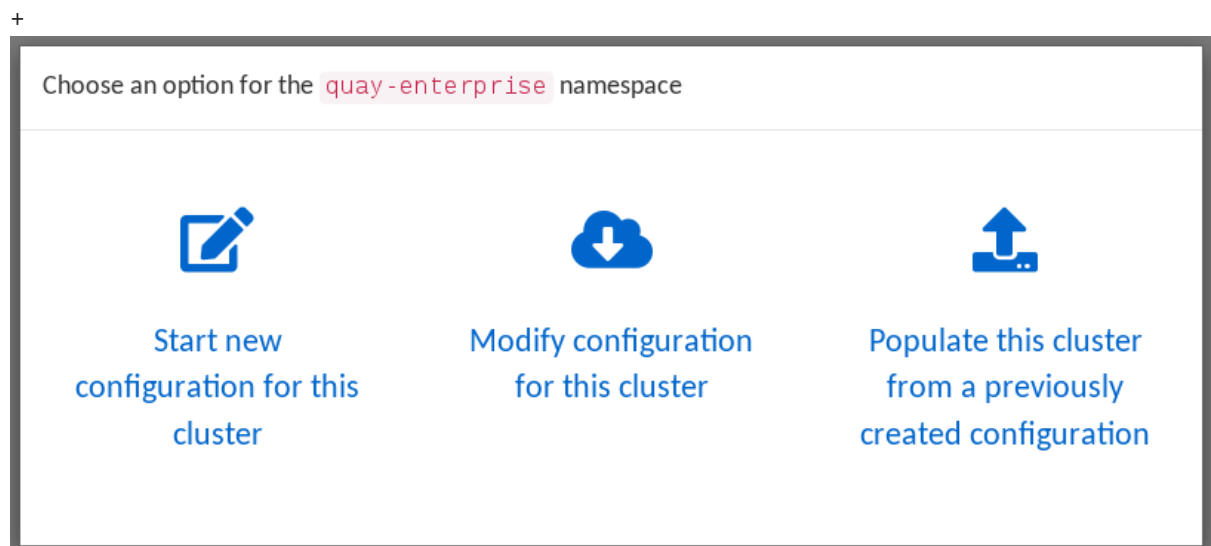
```
$ oc get route -n quay-enterprise quay-enterprise-config
NAME                                HOST/PORT                                PATH
SERVICES                             PORT  TERMINATION  WILDCARD
quay-enterprise-config quay-enterprise-config-quay-enterprise.apps.test.example.com quay-
enterprise-config <all> passthrough None
```

For this example, you would open this URL in your web browser: <https://quay-enterprise-config-quay-enterprise.apps.test.example.com>

- Log in as quayconfig** When prompted, enter the username and password (the password was set as an argument to the quay config container in: **quay-enterprise-config.yaml**):

- User Name: **quayconfig**
- Password: **secret**

You are prompted to select a configuration mode, as shown in the following figure:



- Choose configuration mode:** Select "Start new configuration for this cluster" The result of this selection is the creation of a new configuration file (**config.yaml**) that you will use later for your Red Hat Quay deployment.
- Identify the database:** For the initial setup, add the following information about the type and location of the database to be used by Red Hat Quay:

- **Database Type:** Choose MySQL or PostgreSQL. PostgreSQL is used with the example shown here.
- **Database Server:** Identify the IP address or hostname of the database, along with the port number if it is different from 3306.
- **Username:** Identify a user with full access to the database.
- **Password:** Enter the password you assigned to the selected user.
- **Database Name:** Enter the database name you assigned when you started the database server.
- **SSL Certificate:** For production environments, you should provide an SSL certificate to connect to the database.
To verify the NAME of the service (postgres), type the following:

```
$ oc get services -n quay-enterprise postgres
NAME      TYPE      CLUSTER-IP    EXTERNAL-IP    PORT(S)      AGE
postgres NodePort  172.30.127.41 <none>         5432:32212/TCP 19h
```

The following figure shows an example of the screen for identifying the database used by Red Hat Quay. The example yaml file sets the database server to **postgres**, the user name to **username**, the password to **password**, and the database to **quay**:

The screenshot shows the 'Setup' screen in Red Hat Quay. At the top right, there are four numbered steps (1, 2, 3, 4) and a download icon. Step 1 is highlighted. Below the steps, a message reads: 'Please enter the connection details for your **empty** database. The schema will be created in the following step.'

The form contains the following fields:

- Database Type:** A dropdown menu with 'Postgres' selected.
- Database Server:** A text input field containing 'postgres'. Below it, a note says 'The server (and optionally, custom port) where the database lives'.
- Username:** A text input field containing 'username'. Below it, a note says 'This user must have full access to the database'.
- Password:** A password input field with masked characters (dots).
- Database Name:** A text input field containing 'quay'.
- SSL Certificate:** A 'Browse...' button next to the text 'No file selected.'. Below it, a note says 'Optional SSL certificate (in PEM format) to use to connect to the database'.

At the bottom right of the form, there is a 'Validate Database Settings' button.

17. **Validate database:** Select **Validate Database Settings** and proceed to the next screen.
18. **Create Red Hat Quay superuser** You need to set up an account with superuser privileges to Red Hat Quay, to use for editing Red Hat Quay configuration settings. That information includes a Username, Email address, and Password (entered twice).
The following figure shows an example of the Red Hat Quay Setup screen for setting up a Red Hat Quay superuser account:

Setup

1 — 2 — 3 — 4 —

A superuser is the main administrator of your Red Hat Quay . Only superusers can edit configuration settings.

Username

Minimum 4 characters in length

Email address

Password

Minimum 8 characters in length

Repeat Password

Create Super User

Select **Create Super User**, and proceed to the next screen.

19. **Identify settings:** Go through each of the following settings. The minimum you must enter includes:

- **Server hostname:** The URL to the Red Hat Quay service is required.
- **Redis hostname:** The URL or IP address to the Redis service is required.
Here are all the settings you need to consider:
- **Custom SSL Certificates:** Upload custom or self-signed SSL certificates for use by Red Hat Quay. See [Using SSL to protect connections to Red Hat Quay](#) for details.
Recommended for high availability.



IMPORTANT

Using SSL certificates is recommended for both basic and high availability deployments. If you decide to not use SSL, you must configure your container clients to use your new Red Hat Quay setup as an insecure registry as described in [Test an Insecure Registry](#) .

- **Basic Configuration:** Upload a company logo to rebrand your Red Hat Quay registry.
- **Server Configuration:** Hostname or IP address to reach the Red Hat Quay service, along with TLS indication (recommended for production installations). To get the route to the permanent Red Hat Quay service, type the following:

```
$ oc get route -n quay-enterprise quay-enterprise
NAME          HOST/PORT                                     PATH SERVICES
PORT TERMINATION WILDCARD
quay-enterprise quay-enterprise-quay-enterprise.apps.cnegus-
ocp.devcluster.openshift.com  quay-enterprise-clusterip <all>      None
```

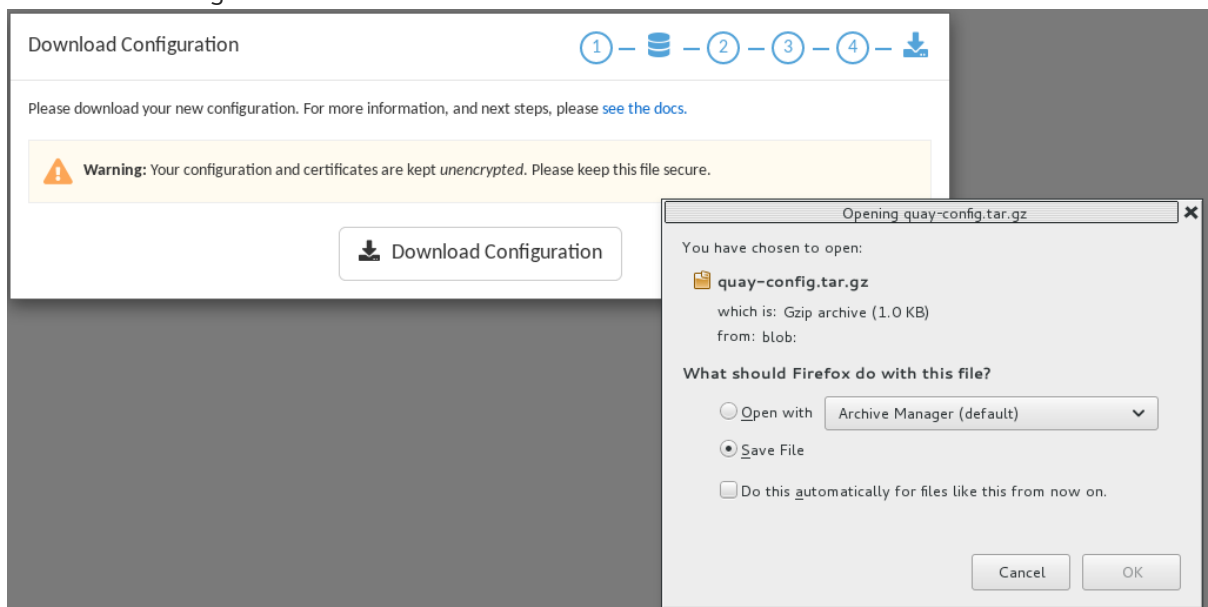

See [Using SSL to protect connections to Red Hat Quay](#) . TLS termination can be done in two different ways:

- On the instance itself, with all TLS traffic governed by the nginx server in the quay container (recommended).
 - On the load balancer. This is not recommended. Access to Red Hat Quay could be lost if the TLS setup is not done correctly on the load balancer.
- **Data Consistency Settings:** Select to relax logging consistency guarantees to improve performance and availability.
 - **Time Machine:** Allow older image tags to remain in the repository for set periods of time and allow users to select their own tag expiration times.
 - **redis:** Identify the hostname or IP address (and optional password) to connect to the redis service used by Red Hat Quay. To find the address of the redis service, type the following:

```
$ oc get services -n quay-enterprise quay-enterprise-redis
NAME          TYPE      CLUSTER-IP  EXTERNAL-IP  PORT(S)  AGE
quay-enterprise-redis ClusterIP  172.30.207.35 <none>      6379/TCP  40m
```

- **Repository Mirroring:** Choose the checkbox to Enable Repository Mirroring. With this enabled, you can create repositories in your Red Hat Quay cluster that mirror selected repositories from remote registries. Before you can enable repository mirroring, start the repository mirroring worker as described later in this procedure.
- **Registry Storage:** Identify the location of storage. A variety of cloud and local storage options are available. Remote storage is required for high availability. Identify the Ceph storage location if you are following the example for Red Hat Quay high availability storage. On OpenShift, the example uses Amazon S3 storage.
- **Action Log Rotation and Archiving:** Select to enable log rotation, which moves logs older than 30 days into storage, then indicate storage area.
- **Security Scanner:** We recommend setting up the Clair security scanner after you have completed the initial Red Hat Quay deployment. Clair setup is described after the end of this procedure.
- **Application Registry:** Enable an additional application registry that includes things like Kubernetes manifests or Helm charts (see the [App Registry specification](#)).
- **BitTorrent-based download:** Allow all registry images to be downloaded using BitTorrent protocol (using the [quayctl](#) tool).
- **rkt Conversion:** Allow **rkt fetch** to be used to fetch images from the Red Hat Quay registry. Public and private GPG2 keys are needed (see [Generating signing keys for ACI conversion](#) for details. This field is deprecated.
- **E-mail:** Enable e-mail to use for notifications and user password resets.
- **Internal Authentication:** Change default authentication for the registry from Local Database to LDAP, Keystone (OpenStack), JWT Custom Authentication, or External Application Token.
- **External Authorization (OAuth):** Enable to allow GitHub or GitHub Enterprise to authenticate to the registry.

- **Google Authentication:** Enable to allow Google to authenticate to the registry.
 - **Access settings:** Basic username/password authentication is enabled by default. Other authentication types that can be enabled include: external application tokens (user-generated tokens used with docker or rkt commands), anonymous access (enable for public access to anyone who can get to the registry), user creation (let users create their own accounts), encrypted client password (require command-line user access to include encrypted passwords), and prefix username autocompletion (disable to require exact username matches on autocompletion).
 - **Registry Protocol Settings:** Leave the **Restrict V1 Push Support** checkbox enabled to restrict access to Docker V1 protocol pushes. Although Red Hat recommends against enabling Docker V1 push protocol, if you do allow it, you must explicitly whitelist the namespaces for which it is enabled.
 - **Dockerfile Build Support** Enable to allow users to submit Dockerfiles to be built and pushed to Red Hat Quay. This is not recommended for multitenant environments.
20. **Save the changes:** Select **Save Configuration Changes**. You are presented with the following Download Configuration screen:



21. **Download configuration:** Select the **Download Configuration** button and save the tarball (**quay-config.tar.gz**) to a local directory. Save this file in case you want to deploy the config files inside manually or just want a record of what you deployed.
22. **Deploy configuration:** Select to rollout the deployment. When prompted, click **Populate configuration to deployments** to deploy the configuration to the Red Hat Quay application. In a few minutes, you should see a green checkmark and the message "Configuration successfully rolled out and deployed!"

**NOTE**

If for some reason the deployment doesn't complete, try deleting the quay-enterprise-app pod. OpenShift should create a new pod and pick up the needed configuration. If that doesn't work, unpack the configuration files (**tar xvf quay-config.tar.gz**) and add them manually to the secret:

```
$ oc create secret generic quay-enterprise-config-secret -n quay-enterprise \
  --from-file=config.yaml=/path/to/config.yaml \
  --from-file=ssl.key=/path/to/ssl.key \
  --from-file=ssl.cert=/path/to/ssl.cert
```

23. **Check pods:** In a couple of minutes (depending on your connection speed), Red Hat Quay should be up and running and the following pods should be visible in the quay-enterprise namespace. You might get a mount error at first, but that should resolve itself:

```
$ oc get pods -n quay-enterprise
NAME                                READY STATUS RESTARTS AGE
postgres-5b4c5d7dd9-f8tqz          1/1 Running 0      46h
quay-enterprise-app-7899c7c77f-jrsrc 1/1 Running 0      45h
quay-enterprise-config-app-86bbbcd446-mwmmg 1/1 Running 0      46h
quay-enterprise-redis-684b9d6f55-tx6w9 1/1 Running 0      46h
```

24. **Get the URL for Red Hat Quay** Type the following to get the hostname of the new Red Hat Quay installation:

```
$ oc get routes -n quay-enterprise quay-enterprise
NAME          HOST/PORT                                PATH SERVICES          PORT
TERMINATION WILDCARD
quay-enterprise quay-enterprise-quay-enterprise.apps.test.example.com  quay-enterprise-
clusterip <all>      None
```

25. **Start using Red Hat Quay.** Open the hostname in a web browser to start using Red Hat Quay.

4.2. ADD CLAIR IMAGE SCANNING TO RED HAT QUAY

Setting up and deploying Clair image scanning for your Red Hat Quay deployment requires the following basic steps:

- Setting up a database for Clair
- Creating authentication keys for Clair
- Deploying Clair

The following procedure assumes you already have a running Red Hat Quay cluster on an OpenShift platform with the Red Hat Quay Setup container running in your browser:

1. **Create the Clair database:** This example configures a postgresql database to use with the Clair image scanner. With the yaml files in the current directory, review those files for possible modifications, then run the following:

```
$ oc create -f postgres-clair-storage.yaml
$ oc create -f postgres-clair-deployment.yaml
$ oc create -f postgres-clair-service.yaml
```

2. **Check Clair database objects:** To view the Clair database objects, type:

```
$ oc get all | grep -i clair
pod/postgres-clair-xxxxxxxx-xxxx 1/1    Running    0          3m45s
deployment.apps/postgres-clair  1/1    1          1          3m45s
service/postgres-clair          NodePort 172.30.193.64 <none> 5432:30680/TCP 159m
replicaset.apps/postgres-clair-xx 1      1          1          3m45s
```

The output shows that the postgres-clair pod is running, postgres-clair was successfully deployed, the postgres-clair service is available on the address and port shown, and 1 replica set of postgres-clair is active.

3. **Open the Red Hat Quay Setup UI** Reload the Red Hat Quay Setup UI and select "Modify configuration for this cluster."
4. **Enable Security Scanning** Scroll to the Security Scanner section and select the "Enable Security Scanning" checkbox. From the fields that appear you need to create an authentication key and enter the security scanner endpoint. Here's how:
 - **Generate key:** Click "Create Key" and then type a name for the Clair private key and an optional expiration date (if blank, the key never expires). Then select Generate Key.
 - **Copy the Clair key and PEM file** Save the Key ID (to a notepad or similar) and download a copy of the Private Key PEM file (named security_scanner.pem) by selecting "Download Private Key" (if you lose this key, you will need to generate a new one).
5. **Modify clair-config.yaml:** Return to the shell and the directory holding your yaml files. Edit the **clair-config.yaml** file and modify the following values:
 - **database.options.source:** Make sure the host, port, dbname, user, password, and ssl mode match those values you set when you create the postgres database for Clair.
 - **key_id:** Search for KEY_ID_HERE in this file and replace it with the contents of the key you generated from the Red Hat Quay Setup screen in the Security Scanner section (security_scanner.pam file).
 - **private_key_path:** Identify the full path to the security_scanner.pem file you saved earlier.
6. **Create the Clair config secret and service** Run the following commands, identifying the paths to your **clair-config.yaml** and **security_scanner.pem** files.

```
$ oc create secret generic clair-scanner-config-secret \
  --from-file=config.yaml=/path/to/clair-config.yaml \
  --from-file=security_scanner.pem=/path/to/security_scanner.pem
$ oc create -f clair-service.yaml
$ oc create -f clair-deployment.yaml
```

7. **Get the clair-service endpoint** In this example, the endpoint of of clair-service would be <http://172.30.133.227:6060>:

```
$ oc get service clair-service
NAME          TYPE          CLUSTER-IP    EXTERNAL-IP  PORT(S)          AGE
clair-service ClusterIP     172.30.133.227 <none>       6060/TCP,6061/TCP 76s
```

8. **Enter Security Scanner Endpoint** Return to the Red Hat Quay Setup screen and fill in the clair-service endpoint. For example, <http://clair-service:6060>
9. **Deploy configuration:** Select to save the configuration, then deploy it when prompted.

A green check mark will appear on the screen when the deployment is done. You can now start using Clair image scanning with Red Hat Quay. For information on the data sources available with the Clair image scanner, see [Using Clair data sources](#).

4.3. ADD REPOSITORY MIRRORING RED HAT QUAY

Enabling repository mirroring allows you to create container image repositories on your Red Hat Quay cluster that exactly match the content of a selected external registry, then sync the contents of those repositories on a regular schedule and on demand.

To add the repository mirroring feature to your Red Hat Quay cluster:

- Run the repository mirroring worker. To do this, you start a quay pod with the **repomirror** option.
- Select "Enable Repository Mirroring in the Red Hat Quay Setup tool.
- Log into your Red Hat Quay Web UI and begin creating mirrored repositories as described in [Repository Mirroring in Red Hat Quay](#).

The following procedure assumes you already have a running Red Hat Quay cluster on an OpenShift platform, with the Red Hat Quay Setup container running in your browser:



NOTE

Instead of running repository mirroring in its own container, you could start the quay application pod with the environment variable **QUAY_OVERRIDE_SERVICES=repomirrorworker=true**. This causes the repomirror worker to run inside the quay application pod instead of as a separate container.

1. **Start the repo mirroring worker.** Start the quay container in **repomirror** mode as follows:

```
$ oc create -f quay-enterprise-mirror.yaml
```


2. **Log into config tool** Log into the Red Hat Quay Setup Web UI (config tool).
3. **Enable repository mirroring** Scroll down the the Repository Mirroring section and select the Enable Repository Mirroring check box, as shown here:

4. **Select HTTPS and cert verification:** If you want to require HTTPS communications and verify certificates during mirroring, select this check box.

Repository Mirroring

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

Enable Repository Mirroring

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

Require HTTPS and verify certificates of Quay registry during mirror.

5. **Save configuration:** Select the Save Configuration Changes button. Repository mirroring should now be enabled on your Red Hat Quay cluster. Refer to [Repository Mirroring in Red Hat Quay](#) for details on setting up your own mirrored container image repositories.

The server hostname you set with the config tools may not represent an endpoint that can be used to copy images to a mirror configured for that server. In that case, you can set a **REPO_MIRROR_SERVER_HOSTNAME** environment variable to identify the server's URL in a way that it can be reached by a skopeo copy command.

CHAPTER 5. STARTING TO USE RED HAT QUAY

With Red Hat Quay now running, you can:

- Select Tutorial from the Quay home page to try the 15-minute tutorial. In the tutorial, you learn to log into Quay, start a container, create images, push repositories, view repositories, and change repository permissions with Quay.
- Refer to the [Use Red Hat Quay](#) for information on working with Red Hat Quay repositories.

CHAPTER 6. APPENDIX A: RED HAT QUAY ON OPENSIFT CONFIGURATION FILES

The following yaml files were created to deploy Red Hat Quay on OpenShift. They are used throughout the deployment procedure in this document. We recommend you copy the files from this document into a directory, review the contents, and make any changes necessary for your deployment.

6.1. RED HAT QUAY NAMESPACES AND SECRETS

quay-enterprise-namespace.yaml

```
apiVersion: v1
kind: Namespace 1
metadata:
  name: quay-enterprise 2
```

- 1** Identifies the Kind as Namespace
- 2** Namespace is set to quay-enterprise throughout the yaml files

quay-enterprise-config-secret.yaml

```
apiVersion: v1
kind: Secret
metadata:
  namespace: quay-enterprise
  name: quay-enterprise-config-secret
```

quay-enterprise-redhat-quay-pull-secret.yaml

```
apiVersion: v1
kind: Secret
metadata:
  namespace: quay-enterprise
  name: redhat-quay-pull-secret
data:
  .dockerconfigjson: <Add credentials> 1
type: kubernetes.io/dockerconfigjson
```

- 1** Change <Add credentials> to include the credentials shown from [Accessing Red Hat Quay](#)

6.2. RED HAT QUAY STORAGE

quay-storageclass.yaml

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: quay-storageclass
```



```
parameters: ❶
  type: gp2
  provisioner: kubernetes.io/aws-ebs
  reclaimPolicy: Delete
```

- ❶ To encrypt the volume, add this to the parameters section (optionally replacing xfs with another filesystem type):

```
  encrypted: "true"
  fsType: xfs (or other fs)
  kmsKeyId:
```

6.3. RED HAT QUAY DATABASE

db-pvc.yaml

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: postgres-storage
  namespace: quay-enterprise
spec:
  accessModes:
    - ReadWriteOnce
  volumeMode: Filesystem
  resources:
    requests:
      storage: 5Gi ❶
  storageClassName: quay-storageclass
```

- ❶ The 5Gi creates 5 gigabytes of storage for use by the Postgres database.

postgres-deployment.yaml

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: postgres
  namespace: quay-enterprise
spec:
  replicas: 1 ❶
  template:
    metadata:
      labels:
        app: postgres
    spec:
      containers:
        - name: postgres
          image: registry.access.redhat.com/rhsc/postgresql-10-rhel7:1-35
          imagePullPolicy: "IfNotPresent"
          ports:
```

```

- containerPort: 5432
env:
- name: POSTGRESQL_USER
  value: "username" 2
- name: POSTGRESQL_DATABASE
  value: "quay"
- name: POSTGRESQL_PASSWORD
  value: "password" 3
volumeMounts:
- mountPath: /var/lib/pgsql/data
  name: postgreddb
serviceAccount: postgres
serviceAccountName: postgres
volumes:
- name: postgreddb
  persistentVolumeClaim:
    claimName: postgres-storage

```

- 1 Only one instance of the postgres database is defined here. Adjust replicas based on demand.
- 2 Replace "username" with a name for your Postgres user
- 3 Replace "password" with a password for your Postgres user

postgres-service.yaml

```

apiVersion: v1
kind: Service
metadata:
  name: postgres
  namespace: quay-enterprise
  labels:
    app: postgres
spec:
  type: NodePort
  ports:
    - port: 5432
  selector:
    app: postgres

```

6.4. RED HAT QUAY AUTHORIZATION

quay-servicetoken-role-k8s1-6.yaml

```

apiVersion: rbac.authorization.k8s.io/v1beta1
kind: Role
metadata:
  name: quay-enterprise-serviceaccount
  namespace: quay-enterprise
rules:
- apiGroups:
  - ""
resources:

```

```

- secrets
verbs:
- get
- put
- patch
- update
- apiGroups:
- ""
resources:
- namespaces
verbs:
- get
- apiGroups:
- extensions
- apps
resources:
- deployments
verbs:
- get
- list
- patch
- update
- watch

```

quay-servicetoken-role-binding-k8s1-6.yaml

```

apiVersion: rbac.authorization.k8s.io/v1beta1
kind: RoleBinding
metadata:
  name: quay-enterprise-secret-writer
  namespace: quay-enterprise
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: quay-enterprise-serviceaccount
subjects:
- kind: ServiceAccount
  name: default

```

6.5. REDIS DATABASE

quay-enterprise-redis.yaml

```

apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  namespace: quay-enterprise
  name: quay-enterprise-redis
  labels:
    quay-enterprise-component: redis
spec:
  replicas: 1 1
  selector:
    matchLabels:

```

```

    quay-enterprise-component: redis
  template:
    metadata:
      namespace: quay-enterprise
      labels:
        quay-enterprise-component: redis
    spec:
      containers:
        - name: redis-master
          image: registry.access.redhat.com/rhsc/redis-32-rhel7
          imagePullPolicy: "IfNotPresent"
          ports:
            - containerPort: 6379
---
apiVersion: v1
kind: Service
metadata:
  namespace: quay-enterprise
  name: quay-enterprise-redis
  labels:
    quay-enterprise-component: redis
spec:
  ports:
    - port: 6379
  selector:
    quay-enterprise-component: redis

```

- 1** Only one instance of the redis database is defined here. Adjust replicas based on demand.

6.6. RED HAT QUAY CONFIGURATION POD

quay-enterprise-config.yaml

```

apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  namespace: quay-enterprise
  name: quay-enterprise-config-app
  labels:
    quay-enterprise-component: config-app
spec:
  replicas: 1
  selector:
    matchLabels:
      quay-enterprise-component: config-app
  template:
    metadata:
      namespace: quay-enterprise
      labels:
        quay-enterprise-component: config-app
    spec:
      containers:
        - name: quay-enterprise-config-app
          image: quay.io/redhat/quay:v3.2.2

```

```

ports:
- containerPort: 8443
command: ["/quay-registry/quay-entrypoint.sh"]
args: ["config", "secret"]
imagePullSecrets:
- name: redhat-quay-pull-secret

```

quay-enterprise-config-service-clusterip.yaml

```

apiVersion: v1
kind: Service
metadata:
  namespace: quay-enterprise
  name: quay-enterprise-config
spec:
  type: ClusterIP
  ports:
  - protocol: TCP
    name: https
    port: 443
    targetPort: 8443
  selector:
    quay-enterprise-component: config-app

```

quay-enterprise-config-route.yaml

```

apiVersion: v1
kind: Route
metadata:
  name: quay-enterprise-config
  namespace: quay-enterprise
spec:
  to:
    kind: Service
    name: quay-enterprise-config
  tls:
    termination: passthrough

```

6.7. RED HAT QUAY APPLICATION CONTAINER

quay-enterprise-service-clusterip.yaml

```

apiVersion: v1
kind: Service
metadata:
  namespace: quay-enterprise
  name: quay-enterprise-clusterip
spec:
  type: ClusterIP
  ports:
  - protocol: TCP
    name: https
    port: 443

```

```
targetPort: 8443
selector:
  quay-enterprise-component: app
```

quay-enterprise-app-route.yaml

```
apiVersion: v1
kind: Route
metadata:
  name: quay-enterprise
  namespace: quay-enterprise
spec:
  to:
    kind: Service
    name: quay-enterprise-clusterip
  tls:
    termination: passthrough
```

quay-enterprise-app-rc.yaml

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  namespace: quay-enterprise
  name: quay-enterprise-app
  labels:
    quay-enterprise-component: app
spec:
  replicas: 1 1
  selector:
    matchLabels:
      quay-enterprise-component: app
  template:
    metadata:
      namespace: quay-enterprise
      labels:
        quay-enterprise-component: app
    spec:
      volumes:
        - name: configvolume
          secret:
            secretName: quay-enterprise-secret
      containers:
        - name: quay-enterprise-app
          image: quay.io/redhat/quay:v3.2.2
          ports:
            - containerPort: 8443
          volumeMounts:
            - name: configvolume
              readOnly: false
              mountPath: /conf/stack
      imagePullSecrets:
        - name: redhat-quay-pull-secret
```

- 1 Only one instance of the quay container is defined here. Adjust replicas based on demand.

6.8. CLAIR IMAGE SCANNING

postgres-clair-storage.yaml

```

apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: postgres-clair-storage
  namespace: quay-enterprise
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 5Gi
  storageClassName: quay-storageclass

```

postgres-clair-deployment.yaml

```

apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  labels:
    app: postgres-clair
  name: postgres-clair
  namespace: quay-enterprise
spec:
  replicas: 1
  selector:
    matchLabels:
      app: postgres-clair
  template:
    metadata:
      labels:
        app: postgres-clair
    spec:
      containers:
      - env:
        - name: POSTGRESQL_USER
          value: clair 1
        - name: POSTGRESQL_DATABASE
          value: clair 2
        - name: POSTGRESQL_PASSWORD
          value: test123 3
        image: registry.access.redhat.com/rhsc/postgresql-10-rhel7:1-35
        imagePullPolicy: IfNotPresent
        name: postgres-clair
        ports:
        - containerPort: 5432
          protocol: TCP
      volumeMounts:

```

```

- mountPath: /var/lib/pgsql/data
  name: postgreddb
  serviceAccount: postgres
  serviceAccountName: postgres
volumes:
- name: postgreddb
  persistentVolumeClaim:
    claimName: postgres-clair-storage

```

- 1 Set the username for the Clair postgres database (clair by default)
- 2 Set the name of the Clair postgres database
- 3 Set the password for the Clair postgres user

postgres-clair-service.yaml

```

apiVersion: v1
kind: Service
metadata:
  labels:
    app: postgres-clair
  name: postgres-clair
  namespace: quay-enterprise
spec:
  ports:
  - nodePort: 30680
    port: 5432
    protocol: TCP
    targetPort: 5432
  selector:
    app: postgres-clair
  type: NodePort

```

clair-config.yaml

```

clair:
  database:
    type: pgsq
    options:
      source: host=postgres-clair port=5432 dbname=clair user=clair password=test123
  sslmode=disable 1
    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 Enterprise is running.
```

```
# For example: https://myregistry.mycompany.com
```

```
endpoint: http://quay-enterprise-clusterip/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_cert_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 Enterprise Setup](security-scanning.md)
```

```
key_id: cd40f1c6a63f574c68ce882258925374882fac2b2f535ae5f8157c429e0c4b2e 2
```

```
private_key_path: /clair/config/security_scanner.pem
```

```
verifier_proxies:
```

```
- enabled: true
```

```
# The port at which Clair will listen.
```

```
listen_addr: :6060
```

```
# If Clair is to be served via TLS, uncomment these lines. See the "Running Clair under TLS"
```

```
# section below for more information.
```

```
# key_file: /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: http://clair-service:6060
```

```
upstream: http://localhost:6062
```

```
key_server:
```

```
type: keyregistry
```

```
options:
```

```
# QUAY_ENDPOINT defines the endpoint at which Quay Enterprise is running.
```

```
# Example: https://myregistry.mycompany.com
```

```
registry: http://quay-enterprise-clusterip/keys/
```

- 1 Check that the database options match those set earlier in `postgres-clair-deployment.yaml`.
- 2 Insert the Key ID matches the value from the key generated from the Red Hat Quay Setup screen.

`clair-service.yaml`

```
apiVersion: v1
kind: Service
metadata:
  name: clair-service
  namespace: quay-enterprise
spec:
  ports:
    - name: clair-api
      port: 6060
      protocol: TCP
      targetPort: 6060
    - name: clair-health
      port: 6061
      protocol: TCP
      targetPort: 6061
  selector:
    quay-enterprise-component: clair-scanner
  type: ClusterIP
```

`clair-deployment.yaml`

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  labels:
    quay-enterprise-component: clair-scanner
  name: clair-scanner
  namespace: quay-enterprise
spec:
  replicas: 1
  selector:
    matchLabels:
      quay-enterprise-component: clair-scanner
  template:
    metadata:
      labels:
        quay-enterprise-component: clair-scanner
    namespace: quay-enterprise
    spec:
      containers:
        - image: quay.io/redhat/clair-jwt:v3.2.2
          imagePullPolicy: IfNotPresent
          name: clair-scanner
          ports:
            - containerPort: 6060
              name: clair-api
              protocol: TCP
            - containerPort: 6061
```

```

    name: clair-health
    protocol: TCP
    volumeMounts:
    - mountPath: /clair/config
      name: configvolume
    imagePullSecrets:
    - name: redhat-quay-pull-secret
    restartPolicy: Always
    volumes:
    - name: configvolume
      secret:
        secretName: clair-scanner-config-secret

```

6.9. REPOSITORY MIRRORING

quay-enterprise-mirror.yaml

```

apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  namespace: quay-enterprise
  name: quay-enterprise-mirror
  labels:
    quay-enterprise-component: mirror-app
spec:
  replicas: 1
  selector:
    matchLabels:
      quay-enterprise-component: mirror-app
  template:
    metadata:
      namespace: quay-enterprise
      labels:
        quay-enterprise-component: mirror-app
    spec:
      containers:
      - name: quay-enterprise-mirror-app
        image: quay.io/redhat/quay:v3.2.2
        ports:
        - containerPort: 8443
        command: ["/quay-registry/quay-entypoint.sh"]
        args: ["repomirror"]
      imagePullSecrets:
      - name: redhat-quay-pull-secret

```

ADDITIONAL RESOURCES