Red Hat Quay 3.2

Deploy Red Hat Quay on OpenShift with Quay Setup Operator

Deploy Red Hat Quay on OpenShift with Quay Setup Operator

Last Updated: 2020-04-30

Red Hat Quay 3.2 Deploy Red Hat Quay on OpenShift with Quay Setup Operator

Deploy Red Hat Quay on OpenShift with Quay Setup Operator

Abstract

Deploy Red Hat Quay on an OpenShift Cluster with the Red Hat Quay Operator

Table of Contents

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 currently supports two approaches to deploying Red Hat Quay on OpenShift:

- **Deploy Red Hat Quay with the Red Hat Quay Setup Operator**: The Red Hat Quay Setup Operator provides a simple method to deploy and manage a Red Hat Quay cluster. This is the now preferred procedure for deploying Red Hat Quay on OpenShift and is covered in this guide.
- **Deploy Red Hat Quay objects individually**: This procedure provides a set of yaml files that you deploy individually to set up your Red Hat Quay cluster. This procedure is expected to eventually be deprecated.

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 OPENSHIFT

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. DEPLOYING RED HAT QUAY

This procedure:

- Installs the Red Hat Quay Setup Operator on OpenShift from the OperatorHub
- Deploys a Red Hat Quay cluster on OpenShift with the Setup Operator

You have the option of changing dozens of settings before deploying the Red Hat Quay Setup Operator. The Operator automates the entire start-up process, by-passing the Red Hat Quay config tool. You can choose to skip the Operator's automated configuration and use the config tool directly.

Prerequisites

- An OpenShift 3.x or 4.x cluster
- Cluster-scope admin privilege to the OpenShift cluster

Procedure

You have two choices for deploying the Red Hat Quay Operator:

- **Advanced Setup**: Go through the **Customizing your Red Hat Quay cluster** section and change any setting you desire before running this procedure.
- **Standard Setup**: Just step through the procedure as is to use all the default setting.

4.1. INSTALL THE RED HAT QUAY SETUP OPERATOR

1. From the OpenShift console, select Operators → OperatorHub, then select the Red Hat Quay Operator.
2. Select Install. The Operator Subscription page appears.
3. Choose the following then select Subscribe:
   - Installation Mode: Select a specific namespace to install to
   - Update Channel: Choose the update channel (only one may be available)
   - Approval Strategy: Choose to approve automatic or manual updates

4.2. DEPLOY A RED HAT QUAY ECOSYSTEM

1. See the Accessing Red Hat Quay article for information on getting credentials needed to obtain the quay container from Quay.io. Then put those credentials in a file. In this example, we create a config.json in the local directory.
2. Create a secret that includes your credentials, as follows:

```
$ oc create secret generic redhat-pull-secret \
    --from-file=".dockerconfigjson=config.json" --type='kubernetes.io/dockerconfigjson'
```

3. Create a custom resource file (in this example, named **quayecosystem_cr.yaml**) or copy one from the quay-operator examples page. This example uses default settings:

```
apiVersion: redhatcop.redhat.io/v1alpha1
kind: QuayEcosystem
metadata:
  name: example-quayecosystem
spec:
  quay:
    imagePullSecretName: redhat-pull-secret
```

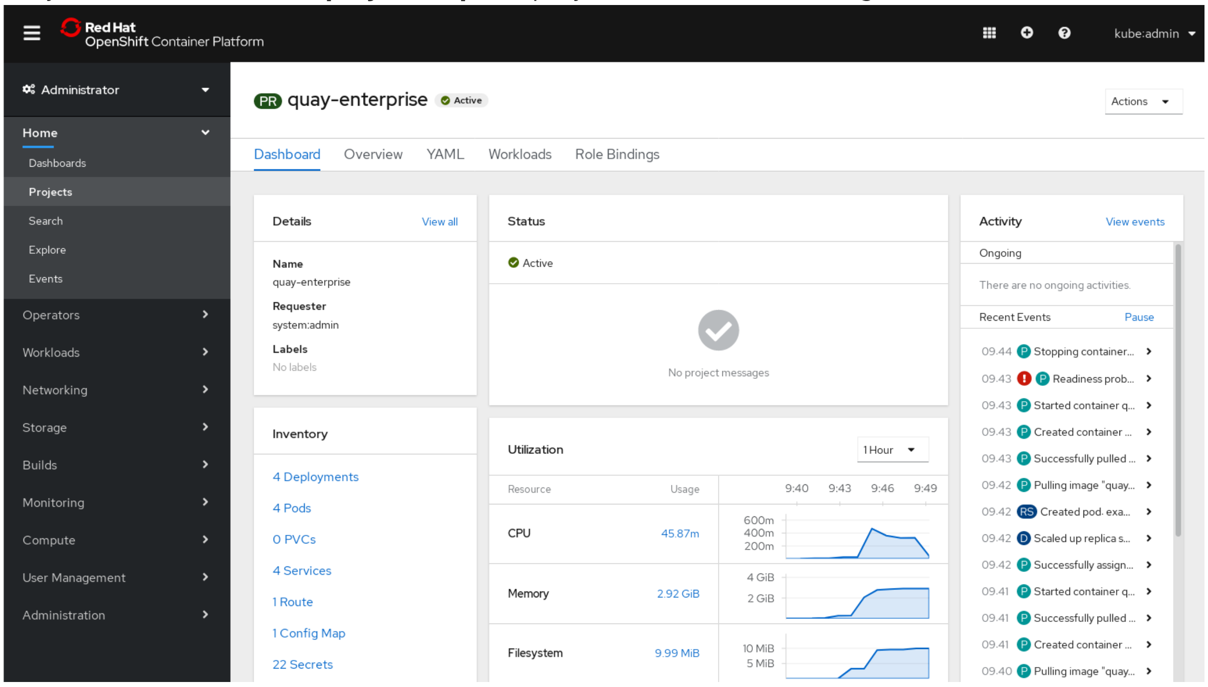
4. Go through the **Customizing your Red Hat Quay cluster** section to choose configuration settings you want to change.

5. Deploy the Quay ecosystem from your custom resource file as follows:

```
$ oc create -f quayecosystem_cr.yaml
```

   Deploying the custom resource automatically creates and configures the Red Hat Quay cluster, which includes the Red Hat Quay, PostgreSQL, and Redis services.

6. To check the status of your Red Hat Quay cluster, log in to your OpenShift web console, select Projects, then select the **quay-enterprise** project to see the following:



Name
quay-enterprise
Requester
system:admin
Labels
No labels

Active

No project messages

There are no ongoing activities.

4 Deployments
4 Pods
0 PVCs
4 Services
1 Route
1 Config Map
22 Secrets

09:44 P Stopping container...
09:43 P Readiness prob...
09:43 P Started container q...
09:43 P Created container ...
09:43 P Successfully pulled ...
09:42 P Pulling image "quay...
09:42 RS Created pod: exa...
09:42 D Scaled up replica s...
09:42 P Successfully assign...
09:41 P Started container q...
09:41 P Successfully pulled ...
09:41 P Created container ...
09:40 P Pulling image "quay...

If Red Hat Quay is running, here is how to get started using your Red Hat Quay configuration:

- Get the route to your new Red Hat Quay cluster as follows:

```
$ oc get route
NAME                          HOST/PORT
        PATH                  SERVICES PORT TERMINATION          WILDCARD
example-quayecosystem-quay example-quayecosystem-quay-default.example.com
        example-quayecosystem-quay        8443 passthrough/Redirect  None
```

- Using that route, log in with the superuser credentials (Username: **quay** and Password: **password** or change credentials as described in the next section)

Additional resources

- For more details on the Red Hat Quay Setup Operator, see the upstream quay-operator project.

CHAPTER 5. CUSTOMIZING YOUR RED HAT QUAY CLUSTER

Although you can run a default Red Hat Quay setup by simply creating a secret and the **QuayEcosystem** custom resource, the following sections describe how you can modify the default setup. Some of those modifications must be made when you deploy the QuayEcosystem, while others can be done after the cluster is running.

5.1. CHANGING YOUR RED HAT QUAY CREDENTIALS

The Red Hat Quay Setup Operator sets up default administrative credentials. Review the default superuser and configuration credentials and change as needed.

5.1.1. Red Hat Quay superuser credentials

The Red Hat Quay superuser credentials let you manage the users, projects and other components of your Red Hat Quay deployment. Here's how superuser credentials are set by default:

- Username: **quay**
- Password: **password**
- Email: **quay@redhat.com**

To change the superuser credentials, create a new secret:

```
$ oc create secret generic <secret_name> \
  --from-literal=superuser-username=<username> \
  --from-literal=superuser-password=<password> \
  --from-literal=superuser-email=<email>
```

The superuser password must be at least 8 characters.

5.1.2. Red Hat Quay configuration credentials

A dedicated Red Hat Quay deployment runs to manage Red Hat Quay configuration settings. Using the route to that configuration, you log in with the following credentials:

- Username: **quayconfig**
- Password: **quay**

You cannot change the username, but you can change the password as follows:

```
$ oc create secret generic quay-config-app \
    --from-literal=config-app-password=<password>
```

5.2. PROVIDING PERSISTENT STORAGE USING POSTGRESQL DATABASE

The PostgreSQL relational database is used by default as the persistent store for Red Hat Quay. PostgreSQL can either be deployed by the Operator within the namespace or leverage an existing instance. The determination of whether to provision an instance or not within the current namespace depends on whether the server property within the **QuayEcosystem** is defined.

The following options are a portion of the available options to configure the PostgreSQL database:

Property	Description
image	Location of the database image
volumeSize	Size of the volume in Kubernetes capacity units

NOTE

It is important to note that persistent storage for the database will only be provisioned if the **volumeSize** property is specified when provisioned by the operator.

Define the values as shown below:

```
apiVersion: redhatcop.redhat.io/v1alpha1
kind: QuayEcosystem
metadata:
  name: example-quayecosystem
spec:
  quay:
    imagePullSecretName: redhat-pull-secret
    database:
      volumeSize: 10Gi
```

5.3. SPECIFYING DATABASE CREDENTIALS

The credentials for accessing the server can be specified through a Secret or when being provisioned by the operator, leverage the following default values:

- **Username**: **quay**
- **Password**: **quay**
- **Root Password**: **quayAdmin**
- **Database Name**: **quay**

To define alternate values, create a secret as shown below:

```
oc create secret generic <secret_name> \
    --from-literal=database-username=<username> \
    --from-literal=database-password=<password> \
    --from-literal=database-root-password=<root-password> \
    --from-literal=database-name=<database-name>
```

Reference the name of the secret in the **QuayEcosystem** custom resource as shown below:

```
apiVersion: redhatcop.redhat.io/v1alpha1
kind: QuayEcosystem
metadata:
  name: example-quayecosystem
```

```
spec:
  quay:
    imagePullSecretName: redhat-pull-secret
    database:
      credentialsSecretName: <secret_name>
```

5.3.1. Using an existing PostgreSQL database instance

Instead of having the operator deploy an instance of PostgreSQL in the project, an existing instance can be leveraged by specifying the location in the server field along with the credentials for access as described in the previous section. The following is an example of how to specify connecting to a remote PostgreSQL instance:

```
apiVersion: redhatcop.redhat.io/v1alpha1
kind: QuayEcosystem
metadata:
  name: example-quayecosystem
spec:
  quay:
    imagePullSecretName: redhat-pull-secret
    database:
      credentialsSecretName: <secret_name>
      server: postgresql.databases.example.com
```

5.4. CHOOSING A REGISTRY STORAGE BACKEND

Red Hat Quay supports multiple backends for the purpose of image storage and consist of a variety of local and cloud storage options. The following sections provide an overview how to configure the Red Hat Quay Setup Operator to make use of these backends.

5.4.1. Overview of storage backends

Storage for Red Hat Quay can be configured using the **registryBackend** field within the quay property in the **QuayEcosystem** resource which contain an array of backends. The ability to define multiple backends enables replication and high availability of images.

```
apiVersion: redhatcop.redhat.io/v1alpha1
kind: QuayEcosystem
metadaLocalStorageta:
  name: example-quayecosystem
spec:
  quay:
    registryBackends:
      - name: backend1
        s3:
        ...
```

The definition of a **registryBackend** is an optional field, and if omitted, **LocalStorage** will be configured (ephemeral, through the use of a **PersistentVolume**, can be enabled if desired).

5.4.2. Sensitive storage values

In many cases, access to storage requires the use of sensitive values. Each backend that requires such configuration can be included in a Secret and defined within the **credentialsSecretName** property of the backend.

Instead of declaring the registry backend properties within the specific backend, the values can be added to a secret as shown below:

```
oc create secret generic s3-credentials \
  --from-literal=accessKey=<accessKey> \
  --from-literal=secretKey=<secretKey>
```

With the values now present in the secret, the properties explicitly declared in the backend can be removed.

Specific details on the types of properties supported for each backend are found in the registry backend details below.

5.4.3. Storage replication

Support is available to replicate the registry storage to multiple backends. To activate storage replication, set the **enableStorageReplication** property to the value of **true**. Individual registry backends can also be configured to be replicated by default by setting the **replicateByDefault** property to the value of true. A full configuration demonstrating the replication options available is shown below:

```
apiVersion: redhatcop.redhat.io/v1alpha1
kind: QuayEcosystem
metadata:
  name: example-quayecosystem
spec:
  quay:
    enableStorageReplication: true
    registryBackends:
      - name: azure-ussouthcentral
        credentialsSecretName: azure-ussouthcentral-registry
        replicateByDefault: true
        azure:
          containerName: quay
      - name: azure-seasia
        credentialsSecretName: azure-seasia-registry
        replicateByDefault: true
        azure:
          containerName: quay
```

NOTE

Support for replicated storage is not available for the local registry backend and will result in an error during the verification phase.

5.4.4. Registry storage backend types

One or more of the following registry storage backends can be defined to specify the underlying storage for the Red Hat Quay registry:

5.4.4.1. Local Storage

The following is an example for configuring the registry to make use of **local** storage:

```
apiVersion: redhatcop.redhat.io/v1alpha1
kind: QuayEcosystem
metadata:
  name: example-quayecosystem
spec:
  quay:
    registryBackends:
      - name: local
        local:
          storagePath: /opt/quayregistry
```

The following is a comprehensive list of properties for the **local** registry backend:

Property	Description	Credential Secret Supported	Required
storagePath	Storage Directory	No	No

5.4.4.2. Configuring persistent local storage

By default, Red Hat Quay uses an ephemeral volume for local storage. In order to avoid data loss, persistent storage is required. To enable the use of a **PersistentVolume** to store images, specify the **registryStorage** parameter underneath the quay property.

The following example will cause a **PersistentVolumeClaim** to be created within the project requesting storage of 10Gi and an access mode of **ReadWriteOnce**. The default value is **ReadWriteMany**.

```
apiVersion: redhatcop.redhat.io/v1alpha1
kind: QuayEcosystem
metadata:
  name: example-quayecosystem
spec:
  quay:
    imagePullSecretName: redhat-pull-secret
    registryStorage:
      persistentVolumeAccessModes:
        - ReadWriteOnce
      persistentVolumeSize: 10Gi
```

A Storage Class can also be provided using the **persistentVolumeStorageClassName** property.

5.4.4.3. Amazon Web Services (S3)

The following is an example for configuring the registry to make use of S3 storage on Amazon Web Services.

```
apiVersion: redhatcop.redhat.io/v1alpha1
kind: QuayEcosystem
metadata:
  name: example-quayecosystem
spec:
  quay:
    registryBackends:
      - name: s3
        s3:
          accessKey: <accessKey>
          bucketName: <bucketName>
          secretKey: <secretKey
          host: <host>
```

The following is a comprehensive list of properties for the **s3** registry backend:

Property	Description	Credential Secret Supported	Required
storagePath	Storage Directory	No	No
bucketName	S3 Bucket	No	Yes
accessKey	AWS Access Key	Yes	Yes
secretKey	AWS Secret Key	Yes	Yes
host	S3 Host	No	No
port	S3 Port	No	No

5.4.4.4. Microsoft Azure storage

The following is an example for configuring the registry to make use of Blob storage on the Microsoft Azure platform.

```
apiVersion: redhatcop.redhat.io/v1alpha1
kind: QuayEcosystem
metadata:
  name: example-quayecosystem
spec:
  quay:
    registryBackends:
      - name: azure
        azure:
          containerName: <containerName>
          accountName: <accountName>
          accountKey: <accountKey>
```

The following is a comprehensive list of properties for the **azure** registry backend:

Property	Description	Credential Secret Suppazureorted	Required
storagePath	Storage Directory	No	No

containerName	Azure Storage Container	No	Yes
accountName	Azure Account Name	No	Yes
accountKey	Azure Account Key	No	Yes
sas_token	Azure SAS Token	No	No

5.4.4.5. Google Cloud storage

The following is an example for configuring the registry to make use of Blob storage on the Google Cloud Platform.

```
apiVersion: redhatcop.redhat.io/v1alpha1azure
kind: QuayEcosystem
metadata:
  name: example-quayecosystem
spec:
  quay:
    registryBackends:
      - name: googleCloud
        googleCloud:
        accessKey: <accessKey>
        secretKey: <secretKey>
        bucketName: <bucketName>
```

The following is a comprehensive list of properties for the **googlecloud** registry backend:

Property	Description	Credential Secret Supported	Required
storagePath	Storage Directory	No	No
accessKey	Cloud Access Key	Yes	Yes
secretKey	Cloud Secret Key	Yes	Yes
bucketName	GCS Bucket	No	Yes

5.4.4.6. NooBaa (RHOCS) storage

The following is an example for configuring the registry to make use of NooBaa (Red Hat OpenShift Container Storage) storage.

```
apiVersion: redhatcop.redhat.io/v1alpha1
kind: QuayEcosystem
metadata:
  name: example-quayecosystem
```

```
spec:
  quay:
    registryBackends:
      - name: rhocs
        rhocs:
          hostname: <hostname>
          secure: <secure>
          accessKey: <accessKey>
          secretKey: <secretKey>
          bucketName: <bucketName>
```

The following is a comprehensive list of properties for the **rhocs** registry backend:

Property	Description	Credential Secret Supported	Required
storagePath	Storage Directory	No	No
hostname	NooBaa Server Hostname	No	Yes
port	Custom Port	No	No
secure	Is Secure	No	No
secretKey	Secret Key	Yes	Yes
bucketName	Bucket Name	No	Yes

5.4.4.7. RADOS storage

The following is an example for configuring the registry to make use of RADOS storage.

```
apiVersion: redhatcop.redhat.io/v1alpha1
kind: QuayEcosystem
metadata:
  name: example-quayecosystem
spec:
  quay:
    registryBackends:
      - name: rados
        rhocs:
          hostname: <hostname>
          secure: <is_secure>
          accessKey: <accessKey>
          secretKey: <secretKey>
          bucketName: <bucketName>
```

The following is a comprehensive list of properties for the **rados** registry backend:

Property	Description	Credential Secret Supported	Required
storagePath	Storage Directory	No	No
hostname	Rados Server Hostname	No	Yes
port	Custom Port	No	No
secure	Is Secure	No	No
accessKey	Access Key	Yes	Yes
secretKey	Secret Key	Yes	Yes
bucketName	Bucket Name	No	Yes

5.4.4.8. Swift (OpenStack) storage

The following is an example for configuring the registry to make use of Swift storage.

```
apiVersion: redhatcop.redhat.io/v1alpha1
kind: QuayEcosystem
metadata:
  name: example-quayecosystem
spec:
  quay:
    registryBackends:
      - name: swift
        rhocs:
          authVersion: <authVersion>
          authURL: <authURL>
          container: <container>
          user: <user>
          password: <password>
          caCertPath: <caCertPath>
          osOptions:
            object_storage_url: <object_storage_url>
            user_domain_name: <user_domain_name>
            project_id: <project_id>
```

The following is a comprehensive list of properties for the **swift** registry backend:

Property	Description	Credential Secret Supported	Required
storagePath	Storage Directory	No	No

authVersion	Swift Auth Version	No	Yes
authURL	Swift Auth URL	No	Yes
container	Swift Container Name	No	Yes
user	Username	Yes	Yes
password	Key/Password	Yes	Yes
caCertPath	CA Cert Filename	No	No
tempURLKey	Temp URL Key	No	No
osOptions	OS Options	No	No

5.4.4.9. CloudFront (S3) storage

The following is an example for configuring the registry to make use of S3 storage on Amazon Web Services.

```
apiVersion: redhatcop.redhat.io/v1alpha1
kind: QuayEcosystem
metadata:
  name: example-quayecosystem
spec:
  quay:
    registryBackends:
      - name: s3
        s3:
          accessKey: <accessKey>
          bucketName: <bucketName>
          secretKey: <secretKey>
          host: <host>
          distributionDomain: <distributionDomain>
          key_ID: <key_ID>
          privateKeyFilename: <privateKeyFilename>
```

The following is a comprehensive list of properties for the **cloudfrontS3** registry backend:

Property	Description	Credential Secret Supported	Required
storagePath	Storage Directory	No	No
bucketName	S3 Bucket	No	Yes
accessKey	AWS Access Key	Yes	Yes

secretKey	AWS Secret Key	Yes	Yes
host	S3 Host	No	No
port	S3 Port	No	No
distributionDomain	CloudFront Distribution Domain Name	No	Yes
keyID	CloudFront Key ID	No	Yes
privateKeyFilename	CloudFront Private Key	No	Yes

5.5. INJECTING CONFIGURATION FILES

Files related to the configuration of Red Hat Quay are located in the **/conf/stack** directory. There are situations for which additional user-defined configuration files need to be added to this directory (such as certificates and private keys). For Red Hat Quay deployments not managed by the Operator, these files are managed by the Red Hat Quay config tool.

The Red Hat Quay Setup Operator supports the injection of these assets within the **configFiles** property in the **quay** property of the **QuayEcosystem** object where one or more assets can be specified.

Two types of configuration files can be specified by the type property:

- **config**: Configuration files that will be added to the **/conf/stack** directory
- **extraCaCerts**: Certificates to be trusted by the quay container

Configuration files are stored as values within Secrets. The first step is to create a secret containing these files. The following command illustrates how a private key can be added:

```
$ oc create secret generic quayconfigfile --from-file=<path_to_file>
```

With the secret created, the secret containing the configuration file can be referenced in the **QuayEcosystem** object as shown below:

```
apiVersion: redhatcop.redhat.io/v1alpha1
kind: QuayEcosystem
metadata:
  name: example-quayecosystem
spec:
  quay:
    configFiles:
      - secretName: quayconfigfile
```

By default, the **config** type is assumed. If the contents of the secret contains certificates that should be added to the **extra_ca_certs** directory, specify the type as **extraCaCert** as shown below:

```
apiVersion: redhatcop.redhat.io/v1alpha1
kind: QuayEcosystem
```

```
metadata:
  name: example-quayecosystem
spec:
  quay:
    configFiles:
      - secretName: quayconfigfile
        type: extraCaCert
```

Individual keys within a secret can be referenced to fine tune the resources that are added to the configuration using the **files** property as shown below:

```
apiVersion: redhatcop.redhat.io/v1alpha1
kind: QuayEcosystem
metadata:
  name: example-quayecosystem
spec:
  quay:
    configFiles:
      - secretName: quayconfigfile
        files:
          - key: myprivatekey.pem
            filename: cloudfront.pemQuay
          - key: myExtraCaCert.crt
            type: extraCaCert
```

The example above assumes that two files have been added to a secret called **quayconfigfile**. The file **myprivatekey.pem** that was added to the secret will be mounted in the quay pod at the path **/conf/stack/cloudfront.pem** since it is a config file type and specifies a custom filename that should be projected into the pod. The **myExtraCaCert.crt** file will be added to the Quay pod within at the path **/conf/stack/extra_certs/myExtraCert.crt**

NOTE

The **type** property within **files** property overrides the value in the **configFiles** property.

5.6. SKIPPING AUTOMATED SETUP

The operator by default is configured to complete the automated setup process for Red Hat Quay. This can be bypassed by setting the **skipSetup** field to **true** as shown below:

```
apiVersion: redhatcop.redhat.io/v1alpha1
kind: QuayEcosystem
metadata:
  name: example-quayecosystem
spec:
  quay:
    imagePullSecretName: redhat-pull-secret
    skipSetup: true
```

5.7. PROVIDING SSL CERTIFICATES

Red Hat Quay, as a secure registry, makes use of SSL certificates to secure communication between the various components within the ecosystem. Transport to the Quay user interface and container registry is secured via SSL certificates. These certificates are generated at startup with the OpenShift route being configured with a TLS termination type of **Passthrough**.

5.7.1. User provided certificates

SSL certificates can be provided and used instead of having the operator generate certificates. Certificates can be provided in a secret which is then referenced in the **QuayEcosystem** custom resource.

The secret containing custom certificates must define the following keys:

- **ssl.cert**: All of the certificates (root, intermediate, certificate) concatinated into a single file
- **ssl.key**: Private key as for the SSL certificate

Create a secret containing the certificate and private key:

```
oc create secret generic custom-quay-ssl \
  --from-file=ssl.key=<ssl_private_key> \
  --from-file=ssl.cert=<ssl_certificate>
```

The secret containing the certificates are referenced using the **sslCertificatesSecretName** property as shown below:

```
apiVersion: redhatcop.redhat.io/v1alpha1
kind: QuayEcosystem
metadata:
  name: example-quayecosystem
spec:
  quay:
    imagePullSecretName: redhat-pull-secret
    sslCertificatesSecretName: custom-quay-ssl
```

5.8. SPECIFYING THE RED HAT QUAY ROUTE

Red Hat Quay makes use of an OpenShift route to enable ingress. The hostname for this route is automatically generated as per the configuration of the OpenShift cluster. Alternatively, the hostname for this route can be explicitly specified using the **hostname** property under the **quay** field as shown below:

```
apiVersion: redhatcop.redhat.io/v1alpha1
kind: QuayEcosystem
metadata:
  name: example-quayecosystem
spec:
  quay:
    hostname: example-quayecosystem-quay-quay-enterprise.apps.openshift.example.com
    imagePullSecretName: redhat-pull-secret
```

5.9. SPECIFYING A RED HAT QUAY CONFIGURATION ROUTE

During the development process, you may want to test the provisioning and setup of Red Hat Quay. By default, the Operator will use the internal service to communicate with the configuration pod. However, when running external to the cluster, you will need to specify the ingress location for which the setup process can use.

Specify the **configHostname** as shown below:

```
apiVersion: redhatcop.redhat.io/v1alpha1
kind: QuayEcosystem
metadata:
  name: example-quayecosystem
spec:
  quay:
    configHostname: example-quayecosystem-quay-config-quay-
enterprise.apps.openshift.example.com
    imagePullSecretName: redhat-pull-secret
```

CHAPTER 6. CONFIGURATION DEPLOYMENT AFTER INITIAL SETUP

In order to conserve resources, the configuration deployment of Red Hat Quay is removed after the initial setup. In certain cases, there may be a need to further configure the Red Hat Quay environment. To specify that the configuration deployment should be retained, the **keepConfigDeployment** property within the **quay** object can be set as **true** as shown below:

```
apiVersion: redhatcop.redhat.io/v1alpha1
kind: QuayEcosystem
metadata:
  name: example-quayecosystem
spec:
  quay:
    imagePullSecretName: redhat-pull-secret
    keepConfigDeployment: true
```

6.1. SETTING REDIS PASSWORD

By default, the operator managed Redis instance is deployed without a password. A password can be specified by creating a secret containing the password in the key **password**. The following command can be used to create the secret:

```
$ oc create secret generic <secret_name> \
   --from-literal=password=<password>
```

The secret can then be specified within the **redis** section using the **credentialsSecretName** as shown below:

```
apiVersion: redhatcop.redhat.io/v1alpha1
kind: QuayEcosystem
metadata:
  name: example-quayecosystem
spec:
  redis:
    credentialsSecretName: <secret_name>
    imagePullSecretName: redhat-pull-secret
```

6.2. ENABLING CLAIR IMAGE SCANNING

Clair is a vulnerability assessment tool for application containers. Support is available to automatically provision and configure both Clair and its integration with Red Hat Quay. A property called **clair** can be specified in the **QuayEcosystem** object along with **enabled: true** within this field in order to deploy Clair. An example is shown below:

```
apiVersion: redhatcop.redhat.io/v1alpha1
kind: QuayEcosystem
metadata:
  name: example-quayecosystem
spec:
  quay:
    imagePullSecretName: redhat-pull-secret
  clair:
    enabled: true
    imagePullSecretName: redhat-pull-secret
```

6.2.1. Clair update interval

Clair routinely queries CVE databases in order to build its own internal database. By default, this value is set at 500m. You can modify the time interval between checks by setting the **updateInterval** property as shown below:

```
apiVersion: redhatcop.redhat.io/v1alpha1
kind: QuayEcosystem
metadata:
  name: example-quayecosystem
spec:
  quay:
    imagePullSecretName: redhat-pull-secret
  clair:
    enabled: true
    imagePullSecretName: redhat-pull-secret
    updateInterval: "60m"
```

The above configuration would have Clair update every 60 minutes.

6.3. SETTING COMMON ATTRIBUTES

Each of the following components expose a set of similar properties that can be specified in order to customize the runtime execution:

- Red Hat Quay
- Red Hat Quay Configuration
- Red Hat Quay PostgreSQL
- Redis
- Clair
- Clair PostgreSQL

6.3.1. Image pull secret

As referenced in prior sections, an Image Pull Secret can specify the name of the secret containing credentials to an image from a protected registry using the property **imagePullSecret**.

6.3.2. Image

There may be a desire to make use of an alternate image or source location for each of the components in the Quay ecosystem. The most common use case is to make use of an image registry that contains all of the needed images instead of being sourced from the public internet. Each component has a property called image where the location of the related image can be referenced from.

The following is an example of how a customized image location can be specified:

```
apiVersion: redhatcop.redhat.io/v1alpha1
kind: QuayEcosystem
metadata:
  name: example-quayecosystem
spec:
  quay:
    image: myregistry.example.com/quay/quay:v3.1.0
```

6.3.3. Compute resources

Compute Resources such as memory and CPU can be specified in the same form as any other value in a **PodTemplate**. CPU and Memory values for **requests** and **limits** can be specified under a property called **resources**.

NOTE

In the case of the **QuayConfiguration** deployment, **configResources** is the property which should be referenced underneath the **quay** property.

The following is an example of how compute resources can be specified:

```
apiVersion: redhatcop.redhat.io/v1alpha1
kind: QuayEcosystem
metadata:
  name: example-quayecosystem
spec:
  quay:
    imagePullSecretName: redhat-pull-secret
    resources:
      requests:
        memory: 512Mi
```

6.3.4. Probes

Readiness and Liveness Probes can be specified in the same form as any other value in a **PodTemplate**.

The following is how a **readinessProbe** and **livenessProbe** can be specified:

```
apiVersion: redhatcop.redhat.io/v1alpha1
kind: QuayEcosystem
metadata:
  name: example-quayecosystem
spec:
  quay:
    imagePullSecretName: redhat-pull-secret
    livenessProbe:
      initialDelaySeconds: 120
      httpGet:
        path: /health/instance
        port: 8443
        scheme: HTTPS
    readinessProbe:
      initialDelaySeconds: 10
      httpGet:
        path: /health/instance
        port: 8443
        scheme: HTTPS
```

NOTE

If a value for either property is not specified, an opinionated default value is applied.

6.3.5. Node Selector

Components of the **QuayEcosystem** may need to be deployed to only a subset of available nodes in a Kubernetes cluster. This functionality can be set on each of the resources using the **nodeSelector** property as shown below:

```
apiVersion: redhatcop.redhat.io/v1alpha1
kind: QuayEcosystem
metadata:
  name: example-quayecosystem
spec:
  quay:
    imagePullSecretName: redhat-pull-secret
    nodeSelector:
      node-role.kubernetes.io/infra: true
```

6.3.6. Deployment strategy

Each of the core components consist of Kubernetes **Deployments**. This resource supports the method in which new versions are released. This operator supports making use of the **RollingUpdate** and **Recreate** strategies. Either value can be defined by using the **deploymentStrategy** property on the desired resource as shown below:

```
apiVersion: redhatcop.redhat.io/v1alpha1
kind: QuayEcosystem
metadata:
  name: example-quayecosystem
spec:
  quay:
    imagePullSecretName: redhat-pull-secret
    deploymentStrategy: RollingUpdate
```

NOTE

The absence of a defined value will make use of the **RollingUpdate** strategy.

6.3.7. Environment Variables

In addition to environment variables that are automatically configured by the operator, users can define their own set of environment variables in order to customize the managed resources. Each core component includes a property called envVars where environment variables can be defined. An example is shown below:

```
apiVersion: redhatcop.redhat.io/v1alpha1
kind: QuayEcosystem
metadata:
  name: example-quayecosystem
spec:
  quay:
    imagePullSecretName: redhat-pull-secret
    envVars:
      - name: FOO
        value: bar
```

NOTE

Environment variables for the Quay configuration pod can be managed by specifying the **configEnvVars** property on the **quay** resource.

WARNING

User defined environment variables are given precedence over those managed by the operator. Undesirable results may occur if conflicting keys are used.

CHAPTER 7. TROUBLESHOOTING

To resolve issues running, configuring and utilizing the operator, the following steps may be utilized:

7.1. ERRORS DURING INITIAL SETUP

The **QuayEcosystem** custom resource will attempt to provide the progress of the status of the deployment and configuration of Red Hat Quay. Additional information related to any errors in the setup process can be found by viewing the log messages of the **config** pod as shown below:

```
$ oc logs $(oc get pods -l=quay-enterprise-component=config -o name)
```

From the OpenShift console, you can follow the Pods and Deployments that are created for your Red Hat Quay cluster.

CHAPTER 8. LOCAL DEVELOPMENT

Execute the following steps to develop the functionality locally. It is recommended that development be done using a cluster with cluster-admin permissions.

Clone the repository, then resolve all dependencies using **go mod**:

```
$ export GO111MODULE=on
$ go mod vendor
```

Using the operator-sdk, run the operator locally:

```
$ operator-sdk up local --namespace=quay-enterprise
```

CHAPTER 9. UPGRADING RED HAT QUAY AND CLAIR

Before upgrading to a new version of Red Hat Quay or Clair, refer to the Upgrade Red Hat Quay guide for details. The instructions here tell you how to change the quay and clair-jwt containers, but do not provide the full upgrade instructions.

At the point in the upgrade instructions where you are ready to identify the new quay and clair-jwt containers, here is what you do:

```
$ oc edit quayecosystem/quayecosystem
```

Find and update the following entries:

```
image: quay.io/redhat/clair-jwt:vX.X.X
image: quay.io/redhat/quay:vX.X.X
```

Once saved, the operator will automatically apply the upgrade.

NOTE

If you used a different name than **QuayEcosystem** for the custom resource to deploy your Quay ecosystem, you will have to replace the name to fit the proper value.

CHAPTER 10. 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.

ADDITIONAL RESOURCES