Deploying the Red Hat Quay Operator on OpenShift Container Platform
Abstract

Deploy the Red Hat Quay Operator on an OpenShift Container Platform cluster
# Table of Contents

PREFACE .................................................................................................................. 4

CHAPTER 1. INTRODUCTION TO THE RED HAT QUAY OPERATOR ......................................... 5
1.1. RED HAT QUAY OPERATOR COMPONENTS .................................................................. 5
1.2. USING MANAGED COMPONENTS ............................................................................. 6
1.3. USING UNMANAGED COMPONENTS FOR DEPENDENCIES ....................................... 6
1.4. CONFIG BUNDLE SECRET ....................................................................................... 7
1.5. PREREQUISITES FOR RED HAT QUAY ON OPENSHIFT CONTAINER PLATFORM ............ 7
1.5.1. OpenShift cluster .............................................................................................. 7
1.5.2. Resource Requirements ................................................................................... 7
1.5.3. Object Storage ................................................................................................. 8

CHAPTER 2. INSTALLING THE RED HAT QUAY OPERATOR FROM THE OPERATORHUB ................. 9

CHAPTER 3. CONFIGURING RED HAT QUAY BEFORE DEPLOYMENT ........................................ 10
3.1. PRE-CONFIGURING RED HAT QUAY FOR AUTOMATION ........................................... 10
3.1.1. Allowing the API to create the first user ............................................................. 10
3.1.2. Enabling general API access ........................................................................... 10
3.1.3. Adding a superuser ........................................................................................ 11
3.1.4. Restricting user creation ............................................................................... 11
3.1.5. Enabling new functionality in Red Hat Quay 3.8 .............................................. 11
3.1.6. Enabling new functionality in Red Hat Quay 3.7 .............................................. 11
3.1.7. Suggested configuration for automation .......................................................... 12
3.2. CONFIGURING OBJECT STORAGE ......................................................................... 12
3.2.1. Using unmanaged storage ............................................................................ 12
3.2.1.1. AWS S3 storage ....................................................................................... 12
3.2.1.2. Google Cloud storage ............................................................................. 12
3.2.1.3. Microsoft Azure storage ......................................................................... 13
3.2.1.4. Ceph/RadosGW Storage ........................................................................ 13
3.2.1.5. Swift storage ............................................................................................ 14
3.2.1.6. NooBaa unmanaged storage ................................................................... 14
3.2.2. Using an unmanaged NooBaa instance ........................................................... 14
3.2.3. Managed storage ........................................................................................... 15
3.2.3.1. Leveraging the Multicloud Object Gateway Component in the Red Hat OpenShift Data Foundation Operator for Red Hat Quay .......................................................... 16
3.2.3.1.1. Installing the Local Storage Operator on OpenShift Container Platform ................................................. 16
3.2.3.1.2. Installing Red Hat OpenShift Data Foundation on OpenShift Container Platform ........................................... 17
3.2.3.1.3. Creating a standalone Multicloud Object Gateway using the OpenShift Container Platform UI .................. 18
3.2.3.1.4. Create A standalone Multicloud Object Gateway using the CLI ................. 20

CHAPTER 4. CONFIGURING TRAFFIC INGRESS ........................................................................... 23
4.1. CONFIGURING SSL/TLS AND ROUTES .................................................................. 23
4.1.1. Creating the config bundle secret with the SSL/TLS cert and key pair ................ 23

CHAPTER 5. CONFIGURING THE DATABASE ............................................................................ 25
5.1. USING AN EXISTING POSTGRESQL DATABASE ......................................................... 25
5.1.1. Database configuration .................................................................................... 25
5.1.1.1. Database URI .......................................................................................... 25
5.1.2. Database connection arguments ..................................................................... 26
5.1.2.1. PostgreSQL SSL/TLS connection arguments ........................................... 26
5.1.2.2. MySQL SSL/TLS connection arguments ............................................... 27
5.1.2.3. Using the managed PostgreSQL database .............................................. 27
5.1.2.1. PostgreSQL database recommendations .................................................. 28
5.2. CONFIGURING EXTERNAL REDIS
   5.2.1. Using an unmanaged Redis database 28
   5.2.2. Using unmanaged Horizontal Pod Autoscalers 29
     5.2.2.1. Disabling the Horizontal Pod Autoscaler 29
   5.2.3. Disabling the Route component 30
   5.2.4. Disabling the monitoring component 31
   5.2.5. Disabling the mirroring component 31

CHAPTER 6. DEPLOYING THE RED HAT QUAY OPERATOR ........................................ 32
   6.1. DEPLOYING RED HAT QUAY FROM THE COMMAND LINE 32
     6.1.1. Viewing created components using the command line 34
     6.1.2. Horizontal Pod Autoscaling 35
       6.1.2.1. Using the API to create the first user 35
     6.1.3. Monitoring and debugging the deployment process 37
   6.2. DEPLOYING RED HAT QUAY FROM THE OPENSSHIFT CONTAINER PLATFORM CONSOLE 40
     6.2.1. Using the Red Hat Quay UI to create the first user 40

CHAPTER 7. VIEWING THE STATUS OF THE QUAYREGISTRY OBJECT ......................... 43
   7.1. VIEWING THE REGISTRY ENDPOINT 43
   7.2. VIEWING THE CONFIG EDITOR ENDPOINT 43
   7.3. VIEWING THE CONFIG EDITOR CREDENTIALS SECRET 43
   7.4. VIEWING THE VERSION OF RED HAT QUAY IN USE 43
   7.5. VIEWING THE CONDITIONS OF YOUR RED HAT QUAY DEPLOYMENT 43

CHAPTER 8. UPDATING RED HAT QUAY AND THE RED HAT QUAY AND CLAIR POSTGRESQL DATABASES ON OPENSSHIFT CONTAINER PLATFORM ................................................. 44
   8.1. UPDATING RED HAT QUAY AND THE RED HAT QUAY AND CLAIR POSTGRESQL DATABASES USING THE OPENSIFT CONTAINER PLATFORM WEB CONSOLE 44
   8.2. UPDATING RED HAT QUAY AND THE RED HAT QUAY AND CLAIR POSTGRESQL DATABASES USING THE CLI 46
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.

The Red Hat Quay Operator provides a simple method to deploy and manage Red Hat Quay on an OpenShift cluster.

With the release of Red Hat Quay 3.4.0, the Red Hat Quay Operator was re-written to offer an enhanced experience and to add more support for Day 2 operations. As a result, the Red Hat Quay Operator is now simpler to use and is more opinionated. The key difference from versions prior to Red Hat Quay 3.4.0 include the following:

- The `QuayEcosystem` custom resource has been replaced with the `QuayRegistry` custom resource.

- The default installation options produces a fully supported Red Hat Quay environment, with all managed dependencies (database, caches, object storage, and so on) supported for production use.

  **NOTE**

  Some components might not be highly available.

- A new validation library for Red Hat Quay’s configuration, which is shared by the Red Hat Quay application and config tool for consistency.

- Object storage can now be managed by the Red Hat Quay Operator using the `ObjectBucketClaim` Kubernetes API

  **NOTE**

  Red Hat OpenShift Data Foundation can be used to provide a supported implementation of this API on OpenShift Container Platform.

- Customization of the container images used by deployed pods for testing and development scenarios.
Use the content in this chapter to execute the following:

- Install the Red Hat Quay Operator
- Configure managed, or unmanaged, object storage
- Configure unmanaged components, such as the database, Redis, routes, TLS, and so on.
- Deploy the Red Hat Quay registry on OpenShift Container Platform using the Red Hat Quay Operator
- Use advanced features supported by the Red Hat Quay Operator
- Upgrade the registry by upgrading the Red Hat Quay Operator

1.1. RED HAT QUAY OPERATOR COMPONENTS

Red Hat Quay has a significant number of dependencies. These include a database, object storage, Redis, and others. The Red Hat Quay Operator manages an opinionated deployment of Red Hat Quay and its dependencies on Kubernetes. These dependencies are treated as components and are configured through the QuayRegistry API.

In the QuayRegistry custom resource, the spec.components field configures components. Each component contains two fields: kind (the name of the component), and managed (a boolean that addresses whether the component lifecycle is handled by the Red Hat Quay Operator). By default, all components are managed and auto-filled upon reconciliation for visibility:

```yaml
spec:
  components:
    - kind: quay
      managed: true
    - kind: postgres
      managed: true
    - kind: clair
      managed: true
    - kind: redis
      managed: true
    - kind: horizontalpodautoscaler
      managed: true
    - kind: objectstorage
      managed: true
    - kind: route
      managed: true
    - kind: mirror
      managed: true
    - kind: monitoring
      managed: true
    - kind: tls
      managed: true
    - kind: clairpostgres
      managed: true
```

1.2. USING MANAGED COMPONENTS

Unless your QuayRegistry custom resource specifies otherwise, the Red Hat Quay Operator uses defaults for the following managed components:

- **quay**: Holds overrides for the Red Hat Quay deployment. For example, environment variables and number of replicas. This component is new as of Red Hat Quay 3.7 and cannot be set to unmanaged.

- **postgres**: For storing the registry metadata. As of Red Hat Quay 3.9, uses a version of PostgreSQL 13 from Software Collections.

  **NOTE**
  
  When upgrading from Red Hat Quay 3.8 → 3.9, the Operator automatically handles upgrading PostgreSQL 10 to PostgreSQL 13. This upgrade is required. PostgreSQL 10 had its final release on November 10, 2022 and is no longer supported.

- **clair**: Provides image vulnerability scanning.

- **redis**: Stores live builder logs and the Red Hat Quay tutorial. Also includes the locking mechanism that is required for garbage collection.

- **horizontalpodautoscaler**: Adjusts the number of Quay pods depending on memory/cpu consumption.

- **objectstorage**: For storing image layer blobs, utilizes the ObjectBucketClaim Kubernetes API which is provided by Noobaa or RHOCs.

- **route**: Provides an external entrypoint to the Red Hat Quay registry from outside of OpenShift Container Platform.

- **mirror**: Configures repository mirror workers to support optional repository mirroring.

- **monitoring**: Features include a Grafana dashboard, access to individual metrics, and alerting to notify for frequently restarting Quay pods.

- **tls**: Configures whether Red Hat Quay or OpenShift Container Platform handles SSL/TLS.

- **clairpostgres**: Configures a managed Clair database. This is a separate database than the PostgreSQL database used to deploy Red Hat Quay.

The Red Hat Quay Operator handles any required configuration and installation work needed for Red Hat Quay to use the managed components. If the opinionated deployment performed by the Red Hat Quay Operator is unsuitable for your environment, you can provide the Red Hat Quay Operator with unmanaged resources (overrides) as described in the following sections.

1.3. USING UNMANAGED COMPONENTS FOR DEPENDENCIES

If you have existing components such as PostgreSQL, Redis, or object storage that you want to use with Red Hat Quay, you first configure them within the Red Hat Quay configuration bundle (config.yaml). Then, they must be referenced in your QuayRegistry bundle as a Kubernetes Secret while indicating which components are unmanaged.
The Red Hat Quay config editor can also be used to create or modify an existing config bundle and simplifies the process of updating the Kubernetes Secret, especially for multiple changes. When Red Hat Quay’s configuration is changed by the config editor and sent to the Red Hat Quay Operator, the deployment is updated to reflect the new configuration.

If you are using an unmanaged PostgreSQL database, and the version is PostgreSQL 10, it is highly recommended that you upgrade to PostgreSQL 13. PostgreSQL 10 had its final release on November 10, 2022 and is no longer supported. For more information, see the PostgreSQL Versioning Policy.

See the following sections for configuring unmanaged components:

- Using an existing PostgreSQL database
- Using unmanaged Horizontal Pod Autoscalers
- Using unmanaged storage
- Using an unmanaged NooBaa instance
- Using an unmanaged Redis database
- Disabling the route component
- Disabling the monitoring component
- Disabling the mirroring component

1.4. CONFIG BUNDLE SECRET

The spec.configBundleSecret field is a reference to the metadata.name of a Secret in the same namespace as the QuayRegistry. This Secret must contain a config.yaml key/value pair. This config.yaml file is a Red Hat Quay config.yaml file. This field is optional, and is auto-filled by the Red Hat Quay Operator if not provided. If provided, it serves as the base set of config fields which are later merged with other fields from any managed components to form a final output Secret, which is then mounted into the Red Hat Quay application pods.

1.5. PREREQUISITES FOR RED HAT QUAY ON OPENSHIFT CONTAINER PLATFORM

Before you begin the deployment of Red Hat Quay Operator on OpenShift Container Platform, you should consider the following.

1.5.1. OpenShift cluster

You need a privileged account to an OpenShift Container Platform 4.5 or later cluster on which to deploy the Red Hat Quay Operator. That account must have the ability to create namespaces at the cluster scope.

1.5.2. Resource Requirements
Each Red Hat Quay application pod has the following resource requirements:

- 8 Gi of memory
- 2000 millicores of CPU.

The Red Hat Quay Operator creates at least one application pod per Red Hat Quay deployment it manages. Ensure your OpenShift Container Platform cluster has sufficient compute resources for these requirements.

1.5.3. Object Storage

By default, the Red Hat Quay Operator uses the **ObjectBucketClaim** Kubernetes API to provision object storage. Consuming this API decouples the Red Hat Quay Operator from any vendor-specific implementation. Red Hat OpenShift Data Foundation provides this API through its NooBaa component, which will be used in this example.

Red Hat Quay can be manually configured to use any of the following supported cloud storage options:

- Amazon S3 (see [S3 IAM Bucket Policy](#) for details on configuring an S3 bucket policy for Red Hat Quay)
- MicroShift Azure Blob Storage
- Google Cloud Storage
- Ceph Object Gateway (RADOS)
- OpenStack Swift
- CloudFront + S3
CHAPTER 2. INSTALLING THE RED HAT QUAY OPERATOR FROM THE OPERATORHUB

Use the following procedure to install the Red Hat Quay Operator from the OpenShift Container Platform OperatorHub.

Procedure


2. In the search box, type Red Hat Quay and select the official Red Hat Quay Operator provided by Red Hat. This directs you to the Installation page, which outlines the features, prerequisites, and deployment information.

3. Select Install. This directs you to the Operator Installation page.

4. The following choices are available for customizing the installation:
   a. Update Channel: Choose the update channel, for example, stable-3.7 for the latest release.
   b. Installation Mode: Choose All namespaces on the cluster if you want the Red Hat Quay Operator to be available cluster-wide. Choose A specific namespace on the cluster if you want it deployed only within a single namespace. It is recommended that you install the Red Hat Quay Operator cluster-wide. If you choose a single namespace, the monitoring component will not be available by default.
      • Approval Strategy: Choose to approve either automatic or manual updates. Automatic update strategy is recommended.

5. Select Install.
CHAPTER 3. CONFIGURING RED HAT QUAY BEFORE DEPLOYMENT

The Red Hat Quay Operator can manage all of the Red Hat Quay components when deployed on OpenShift Container Platform. This is the default configuration, however, you can manage one or more components externally when you want more control over the set up.

Use the following pattern to configure unmanaged Red Hat Quay components.

Procedure

1. Create a `config.yaml` configuration file with the appropriate settings.
2. Create a `Secret` using the configuration file by entering the following command:
   ```bash
   $ oc create secret generic --from-file config.yaml=./config.yaml config-bundle-secret
   ```
3. Create a `quayregistry.yaml` file, identifying the unmanaged components and also referencing the created `Secret`, for example:

   **Example QuayRegistry YAML file**

   ```yaml
   apiVersion: quay.redhat.com/v1
   kind: QuayRegistry
   metadata:
     name: example-registry
     namespace: quay-enterprise
   spec:
     configBundleSecret: config-bundle-secret
     components:
       - kind: objectstorage
         managed: false
   
   $ oc create -n quay-enterprise -f quayregistry.yaml
   ```

3.1. PRE-CONFIGURING RED HAT QUAY FOR AUTOMATION

Red Hat Quay supports several configuration options that enable automation. Users can configure these options before deployment to reduce the need for interaction with the user interface.

3.1.1. Allowing the API to create the first user

To create the first user, users need to set the FEATURE_USER_INITIALIZE parameter to true and call the `/api/v1/user/initialize` API. Unlike all other registry API calls that require an OAuth token generated by an OAuth application in an existing organization, the API endpoint does not require authentication.

Users can use the API to create a user such as `quayadmin` after deploying Red Hat Quay, provided no other users have been created. For more information, see Using the API to create the first user.

3.1.2. Enabling general API access
Users should set the `BROWSER_API_CALLS_XHR_ONLY` config option to false to allow general access to the Red Hat Quay registry API.

### 3.1.3. Adding a superuser

After deploying Red Hat Quay, users can create a user and give the first user administrator privileges with full permissions. Users can configure full permissions in advance by using the `SUPER_USER` configuration object. For example:

```yaml
...  
SERVER_HOSTNAME: quay-server.example.com  
SETUP_COMPLETE: true  
SUPER_USERS:  
  - quayadmin  
...  
```

### 3.1.4. Restricting user creation

After you have configured a superuser, you can restrict the ability to create new users to the superuser group by setting the `FEATURE_USER_CREATION` to `false`. For example:

```yaml
...  
FEATURE_USER_INITIALIZE: true  
BROWSER_API_CALLS_XHR_ONLY: false  
SUPER_USERS:  
  - quayadmin  
FEATURE_USER_CREATION: false  
...  
```

### 3.1.5. Enabling new functionality in Red Hat Quay 3.8

To use new Red Hat Quay 3.8 functions, enable some or all of the following features:

```yaml
...  
FEATURE_UI_V2: true  
FEATURE_LISTEN_IP_VERSION:  
FEATURE_SUPERUSERS_FULL_ACCESS: true  
GLOBAL_READONLY_SUPER_USERS:  
  -  
FEATURE_RESTRICTED_USERS: true  
RESTRICTED_USERS_WHITELIST:  
  -  
...  
```

### 3.1.6. Enabling new functionality in Red Hat Quay 3.7

To use new Red Hat Quay 3.7 functions, enable some or all of the following features:

```yaml
...  
FEATURE_QUOTA_MANAGEMENT: true  
FEATURE_BUILD_SUPPORT: true  
FEATURE_PROXY_CACHE: true  
...  
```
3.1.7. Suggested configuration for automation

The following `config.yaml` parameters are suggested for automation:

```
... FEATURE_STORAGE_REPLICATION: true
DEFAULT_SYSTEM_REJECT_QUOTA_BYTES: 102400000
...
```

### 3.2. CONFIGURING OBJECT STORAGE

You need to configure object storage before installing Red Hat Quay, irrespective of whether you are allowing the Red Hat Quay Operator to manage the storage or managing it yourself.

If you want the Red Hat Quay Operator to be responsible for managing storage, see the section on Managed storage for information on installing and configuring NooBaa and the Red Hat OpenShift Data Foundations Operator.

If you are using a separate storage solution, set `objectstorage` as `unmanaged` when configuring the Operator. See the following section, Unmanaged storage, for details of configuring existing storage.

#### 3.2.1. Using unmanaged storage

This section provides configuration examples for unmanaged storage for your convenience. Refer to the Red Hat Quay configuration guide for complete instructions on how to set up object storage.

##### 3.2.1.1. AWS S3 storage

Use the following example when configuring AWS S3 storage for your Red Hat Quay deployment.

```
DISTRIBUTED_STORAGE_CONFIG:
  s3Storage:
    - S3Storage
      - host: s3.us-east-2.amazonaws.com
        s3_access_key: ABCDEFGHIJKLMNOP
        s3_secret_key: OL3ABCDEFGHIJKLMNOP
        s3_bucket: quay_bucket
        storage_path: /datastorage/registry
  DISTRIBUTED_STORAGE_DEFAULT_LOCATIONS: []
  DISTRIBUTED_STORAGE_PREFERENCE:
    - s3Storage
```

##### 3.2.1.2. Google Cloud storage

Use the following example when configuring Google Cloud storage for your Red Hat Quay deployment.
3.2.1.3. Microsoft Azure storage

Use the following example when configuring Microsoft Azure storage for your Red Hat Quay deployment.

DISTRIBUTED_STORAGE_CONFIG:
  googleCloudStorage:
    - GoogleCloudStorage
    - access_key: GOOGQIMFB3ABCDEFGHIJKLMNOP
      bucket_name: quay-bucket
      secret_key: FhDAYe2HeuAKfZCAGyOioNaaRABCDEFGHJKLMN
      storage_path: /datastorage/registry
  azureStorage:
    - AzureStorage
      azure_account_name: azure_account_name_here
      azure_container: azure_container_here
      storage_path: /datastorage/registry
      azure_account_key: azure_account_key_here
      sas_token: some/path/
      endpoint_url: https://[account-name].blob.core.usgovcloudapi.net
  radosGWStorage: #storage config name
    - RadosGWStorage #actual driver
    - access_key: access_key_here #parameters
      secret_key: secret_key_here
      bucket_name: bucket_name_here
      hostname: hostname_here
      is_secure: 'true'
      port: '443'
      storage_path: /datastorage/registry

DISTRIBUTED_STORAGE_DEFAULT_LOCATIONS: []
DISTRIBUTED_STORAGE_PREFERENCE:
  - googleCloudStorage

3.2.1.3. Microsoft Azure storage

Use the following example when configuring Microsoft Azure storage for your Red Hat Quay deployment.

DISTRIBUTED_STORAGE_CONFIG:
  azureStorage:
    - AzureStorage
      azure_account_name: azure_account_name_here
      azure_container: azure_container_here
      storage_path: /datastorage/registry
      azure_account_key: azure_account_key_here
      sas_token: some/path/
      endpoint_url: https://[account-name].blob.core.usgovcloudapi.net

DISTRIBUTED_STORAGE_DEFAULT_LOCATIONS: []
DISTRIBUTED_STORAGE_PREFERENCE:
  - azureStorage

The endpoint_url parameter for Microsoft Azure storage is optional and can be used with Microsoft Azure Government (MAG) endpoints. If left blank, the endpoint_url will connect to the normal Microsoft Azure region.

As of Red Hat Quay 3.7, you must use the Primary endpoint of your MAG Blob service. Using the Secondary endpoint of your MAG Blob service will result in the following error: AuthenticationErrorDetail:Cannot find the claimed account when trying to GetProperties for the account whusc8-secondary.

3.2.1.4. Ceph/RadosGW Storage

Use the following example when configuring Ceph/RadosGW storage for your Red Hat Quay deployment.

DISTRIBUTED_STORAGE_CONFIG:
  radosGWStorage:
    - RadosGWStorage
      access_key: access_key_here
      secret_key: secret_key_here
      bucket_name: bucket_name_here
      hostname: hostname_here
      is_secure: 'true'
      port: '443'
      storage_path: /datastorage/registry
3.2.1.5. Swift storage

Use the following example when configuring Swift storage for your Red Hat Quay deployment.

DISTRIBUTED_STORAGE_CONFIG:
swiftStorage:
  - SwiftStorage
    - swift_user: swift_user_here
    - swift_password: swift_password_here
    - swift_container: swift_container_here
    - auth_url: https://example.org/swift/v1/quay
    - auth_version: 1
    - ca_cert_path: /conf/stack/swift.cert
    - storage_path: /datastorage/registry

3.2.1.6. NooBaa unmanaged storage

Use the following procedure to deploy NooBaa as your unmanaged storage configuration.

Procedure

1. Create a NooBaa Object Bucket Claim in the Red Hat Quay console by navigating to Storage → Object Bucket Claims.

2. Retrieve the Object Bucket Claim Data details, including the Access Key, Bucket Name, Endpoint (hostname), and Secret Key.

3. Create a config.yaml configuration file that uses the information for the Object Bucket Claim:

DISTRIBUTED_STORAGE_CONFIG:
default:
  - RHOCSStorage
    - access_key: WmrXtISGk8B3nABCDEFGH
    - bucket_name: my-noobaa-bucket-claim-8b844191-dc6c-444e-9ea4-87ece0abcdef
    - hostname: s3.openshift-storage.svc.cluster.local
    - is_secure: true
    - port: "443"
    - secret_key: X9P5SDGJtmSuHFCMSLMMbdNCMfUABCDEFGH+C5QD
    - storage_path: /datastorage/registry

For more information about configuring an Object Bucket Claim, see Object Bucket Claim.

3.2.2. Using an unmanaged NooBaa instance
Use the following procedure to use an unmanaged NooBaa instance for your Red Hat Quay deployment.

**Procedure**

1. Create a NooBaa Object Bucket Claim in the console at Storage → Object Bucket Claims.

2. Retrieve the Object Bucket Claim Data details including the **Access Key**, **Bucket Name**, **Endpoint (hostname)**, and **Secret Key**.

3. Create a `config.yaml` configuration file using the information for the Object Bucket Claim. For example:

   ```yaml
   DISTRIBUTED_STORAGE_CONFIG:
   default:
   - RHOCSStorage
     access_key: WmrXtSGk8B3nABCDEFGHI
     bucket_name: my-noobaa-bucket-claim-8b844191-dc6c-444e-9ea4-87e0abe5def
     hostname: s3.openshift-storage.svc.cluster.local
     is_secure: true
     port: "443"
     secret_key: X9P5SDGJtmSuHFCMSLmbdNCMiUABCDEFGH+5QD
     storage_path: /datastorage/registry
   DISTRIBUTED_STORAGE_DEFAULT_LOCATIONS: []
   DISTRIBUTED_STORAGE_PREFERENCE:
   - default
   
   3.2.3. Managed storage

   If you want the Red Hat Quay Operator to manage object storage for Red Hat Quay, your cluster needs to be capable of providing object storage through the **ObjectBucketClaim** API. Using the Red Hat OpenShift Data Foundation Operator, there are two supported options available:

   - A standalone instance of the Multi-Cloud Object Gateway backed by a local Kubernetes **PersistentVolume** storage
     - Not highly available
     - Included in the Red Hat Quay subscription
     - Does not require a separate subscription for Red Hat OpenShift Data Foundation

   - A production deployment of Red Hat OpenShift Data Foundation with scale-out Object Service and Ceph
     - Highly available
     - Requires a separate subscription for Red Hat OpenShift Data Foundation

   To use the standalone instance option, continue reading below. For production deployment of Red Hat OpenShift Data Foundation, please refer to the [official documentation](#).
NOTE

Object storage disk space is allocated automatically by the Red Hat Quay Operator with 50 GiB. This number represents a usable amount of storage for most small to medium Red Hat Quay installations but might not be sufficient for your use cases. Resizing the Red Hat OpenShift Data Foundation volume is currently not handled by the Red Hat Quay Operator. See the section below about resizing managed storage for more details.

3.2.3.1. Leveraging the Multicloud Object Gateway Component in the Red Hat OpenShift Data Foundation Operator for Red Hat Quay

As part of a Red Hat Quay subscription, users are entitled to use the Multicloud Object Gateway component of the Red Hat OpenShift Data Foundation Operator (formerly known as OpenShift Container Storage Operator). This gateway component allows you to provide an S3-compatible object storage interface to Red Hat Quay backed by Kubernetes PersistentVolume-based block storage. The usage is limited to a Red Hat Quay deployment managed by the Operator and to the exact specifications of the multicloud Object Gateway instance as documented below.

Since Red Hat Quay does not support local filesystem storage, users can leverage the gateway in combination with Kubernetes PersistentVolume storage instead, to provide a supported deployment. A PersistentVolume is directly mounted on the gateway instance as a backing store for object storage and any block-based StorageClass is supported.

By the nature of PersistentVolume, this is not a scale-out, highly available solution and does not replace a scale-out storage system like Red Hat OpenShift Data Foundation. Only a single instance of the gateway is running. If the pod running the gateway becomes unavailable due to rescheduling, updates or unplanned downtime, this will cause temporary degradation of the connected Red Hat Quay instances.

Using the following procedures, you will install the Local Storage Operator, Red Hat OpenShift Data Foundation, and create a standalone Multicloud Object Gateway to deploy Red Hat Quay on OpenShift Container Platform.

NOTE

The following documentation shares commonality with the official Red Hat OpenShift Data Foundation documentation.

3.2.3.1.1. Installing the Local Storage Operator on OpenShift Container Platform

Use the following procedure to install the Local Storage Operator from the Operator Hub before creating Red Hat OpenShift Data Foundation clusters on local storage devices.

1. Log in to the OpenShift Web Console.
2. Click Operators → OperatorHub.
3. Type local storage into the search box to find the Local Storage Operator from the list of Operators. Click Local Storage.
4. Click Install.
5. Set the following options on the Install Operator page:
   - For Update channel, select stable.
   - For Installation mode, select A specific namespace on the cluster
For Installed Namespace, select **Operator recommended namespace openshift-local-storage**.

- For Update approval, select **Automatic**.

6. Click **Install**.

3.2.3.1.2. Installing Red Hat OpenShift Data Foundation on OpenShift Container Platform

Use the following procedure to install Red Hat OpenShift Data Foundation on OpenShift Container Platform.

**Prerequisites**

- Access to an OpenShift Container Platform cluster using an account with **cluster-admin** and Operator installation permissions.

- You must have at least three worker nodes in the OpenShift Container Platform cluster.

- For additional resource requirements, see the Planning your deployment guide.

**Procedure**

1. Log in to the **OpenShift Web Console**.

2. Click **Operators → OperatorHub**.

3. Type **OpenShift Data Foundation** in the search box. Click **OpenShift Data Foundation**.

4. Click **Install**.

5. Set the following options on the Install Operator page:

   - For Update channel, select the most recent stable version.

   - For Installation mode, select **A specific namespace on the cluster**

   - For Installed Namespace, select **Operator recommended Namespace: openshift-storage**.

   - For Update approval, select **Automatic** or **Manual**.

     If you select **Automatic** updates, then the Operator Lifecycle Manager (OLM) automatically upgrades the running instance of your Operator without any intervention.

     If you select **Manual** updates, then the OLM creates an update request. As a cluster administrator, you must then manually approve that update request to update the Operator to a newer version.

   - For Console plugin, select **Enable**.

6. Click **Install**.

   After the Operator is installed, a pop-up with a message, **Web console update is available** appears on the user interface. Click **Refresh web console** from this pop-up for the console changes to reflect.

7. Continue to the following section, "Creating a standalone Multicloud Object Gateway", to leverage the Multicloud Object Gateway Component for Red Hat Quay.
3.2.3.1.3. Creating a standalone Multicloud Object Gateway using the OpenShift Container Platform UI

Use the following procedure to create a standalone Multicloud Object Gateway.

**Prerequisites**

- You have installed the Local Storage Operator.
- You have installed the Red Hat OpenShift Data Foundation Operator.

**Procedure**

1. In the **OpenShift Web Console**, click **Operators → Installed Operators** to view all installed Operators. Ensure that the namespace is **openshift-storage**.

2. Click **Create StorageSystem**.

3. On the **Backing storage** page, select the following:
   a. Select **Multicloud Object Gateway** for **Deployment type**.
   b. Select the **Create a new StorageClass using the local storage devices** option.
   c. Click **Next**.

   **NOTE**
   You are prompted to install the Local Storage Operator if it is not already installed. Click **Install**, and follow the procedure as described in "Installing the Local Storage Operator on OpenShift Container Platform".

4. On the **Create local volume set** page, provide the following information:
   a. Enter a name for the **LocalVolumeSet** and the **StorageClass**. By default, the local volume set name appears for the storage class name. You can change the name.
   b. Choose one of the following:
      - **Disk on all nodes**
        Uses the available disks that match the selected filters on all the nodes.
      - **Disk on selected nodes**
        Uses the available disks that match the selected filters only on the selected nodes.
   c. From the available list of **Disk Type**, select **SSD/NVMe**.
   d. Expand the **Advanced** section and set the following options:

<table>
<thead>
<tr>
<th>Volume Mode</th>
<th>Filesystem is selected by default. Always ensure that Filesystem is selected for Volume Mode.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device Type</td>
<td>Select one or more device type from the dropdown list.</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>--------------------------------------------------------</td>
</tr>
<tr>
<td>Disk Size</td>
<td>Set a minimum size of 100GB for the device and maximum available size of the device that needs to be included.</td>
</tr>
<tr>
<td>Maximum Disks Limit</td>
<td>This indicates the maximum number of PVs that can be created on a node. If this field is left empty, then PVs are created for all the available disks on the matching nodes.</td>
</tr>
</tbody>
</table>

e. Click **Next**
   A pop-up to confirm the creation of **LocalVolumeSet** is displayed.

f. Click **Yes** to continue.

5. In the **Capacity and nodes** page, configure the following:
   a. **Available raw capacity** is populated with the capacity value based on all the attached disks associated with the storage class. This takes some time to show up. The **Selected nodes** list shows the nodes based on the storage class.
   b. Click **Next** to continue.

6. Optional. Select the **Connect to an external key management service** checkbox. This is optional for cluster-wide encryption.
   a. From the **Key Management Service Provider** drop-down list, either select **Vault** or **Thales CipherTrust Manager (using KMIP)**. If you selected **Vault**, go to the next step. If you selected **Thales CipherTrust Manager (using KMIP)**, go to step iii.
   b. Select an **Authentication Method**.
      Using Token Authentication method
      - Enter a unique **Connection Name**, host **Address** of the Vault server (‘https://<hostname or ip>’), **Port** number and **Token**.
      - Expand **Advanced Settings** to enter additional settings and certificate details based on your **Vault** configuration:
        - Enter the Key Value secret path in **Backend Path** that is dedicated and unique to OpenShift Data Foundation.
        - Optional: Enter **TLS Server Name** and **Vault Enterprise Namespace**
        - Upload the respective PEM encoded certificate file to provide the **CA Certificate**, **Client Certificate**, and **Client Private Key**.
        - Click **Save** and skip to step iv.
      Using Kubernetes authentication method
      - Enter a unique Vault **Connection Name**, host **Address** of the Vault server (‘https://<hostname or ip>’), **Port** number and **Role** name.
- Expand **Advanced Settings** to enter additional settings and certificate details based on your Vault configuration:
  - Enter the Key Value secret path in **Backend Path** that is dedicated and unique to Red Hat OpenShift Data Foundation.
  - Optional: Enter **TLS Server Name** and **Authentication Path** if applicable.
  - Upload the respective PEM encoded certificate file to provide the **CA Certificate**, **Client Certificate**, and **Client Private Key**.
  - Click **Save** and skip to step iv.

c. To use **Thales CipherTrust Manager (using KMIP)** as the KMS provider, follow the steps below:
   i. Enter a unique **Connection Name** for the Key Management service within the project.
   ii. In the **Address** and **Port** sections, enter the IP of Thales CipherTrust Manager and the port where the KMIP interface is enabled. For example:
      - **Address**: 123.34.3.2
      - **Port**: 5696
   iii. Upload the **Client Certificate**, **CA certificate**, and **Client Private Key**.
   iv. If StorageClass encryption is enabled, enter the Unique Identifier to be used for encryption and decryption generated above.
   v. The **TLS Server** field is optional and used when there is no DNS entry for the KMIP endpoint. For example, `kmip_all_<port>.ciphertrustmanager.local`.

d. Select a **Network**.

e. Click **Next**.

7. In the **Review and create** page, review the configuration details. To modify any configuration settings, click **Back**.

8. Click **Create StorageSystem**.

### 3.2.3.1.4. Create A standalone Multicloud Object Gateway using the CLI

Use the following procedure to install the Red Hat OpenShift Data Foundation (formerly known as OpenShift Container Storage) Operator and configure a single instance Multi-Cloud Gateway service.

**NOTE**

The following configuration cannot be run in parallel on a cluster with Red Hat OpenShift Data Foundation installed.

**Procedure**

1. On the **OpenShift Web Console**, and then select **Operators → OperatorHub**.
2. Search for **Red Hat OpenShift Data Foundation** and then select **Install**.
3. Accept all default options, and then select **Install**.

4. Confirm that the Operator has installed by viewing the **Status** column, which should be marked as **Succeeded**.

   ![WARNING]
   
   **WARNING**
   
   When the installation of the Red Hat OpenShift Data Foundation Operator is finished, you are prompted to create a storage system. Do not follow this instruction. Instead, create NooBaa object storage as outlined the following steps.

5. On your machine, create a file named **noobaa.yaml** with the following information:

   ```yaml
   apiVersion: noobaa.io/v1alpha1
   kind: NooBaa
   metadata:
     name: noobaa
     namespace: openshift-storage
   spec:
     dbResources:
       requests:
         cpu: '0.1'
         memory: 1Gi
     dbType: postgres
     coreResources:
       requests:
         cpu: '0.1'
         memory: 1Gi
   
   This creates a single instance deployment of the **Multi-cloud Object Gateway**.

6. Apply the configuration with the following command:

   ```bash
   $ oc create -n openshift-storage -f noobaa.yaml
   
   **Example output**
   
   noobaa.noobaa.io/noobaa created
   
   7. After a few minutes, the **Multi-cloud Object Gateway** should finish provisioning. You can enter the following command to check its status:

   ```bash
   $ oc get -n openshift-storage noobaas noobaa -w
   
   **Example output**
   
<table>
<thead>
<tr>
<th>NAME</th>
<th>MGMT-ENDPOINTS</th>
<th>S3-ENDPOINTS</th>
<th>IMAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>PHASE</td>
<td>AGE</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
8. Configure a backing store for the gateway by creating the following YAML file, named `noobaa-pv-backing-store.yaml`:

```yaml
apiVersion: noobaa.io/v1alpha1
display: BackingStore
displayProperties:
  - noobaa.io/finalizer
labels:
  app: noobaa
default: noobaa-pv-backing-store
kind: PersistentVolume
metadata:
  name: noobaa-pv-backing-store
namespace: openshift-storage
spec:
  pvPool:
    numVolumes: 1
    resources:
      requests:
        storage: 50Gi
    type: pv-pool
  storageClass: STORAGE-CLASS-NAME
```

1. The overall capacity of the object storage service. Adjust as needed.

2. The `StorageClass` to use for the `PersistentVolumes` requested. Delete this property to use the cluster default.

9. Enter the following command to apply the configuration:

```
$ oc create -f noobaa-pv-backing-store.yaml
```

**Example output**

```
backingstore.noobaa.io/noobaa-pv-backing-store created
```

This creates the backing store configuration for the gateway. All images in Red Hat Quay will be stored as objects through the gateway in a `PersistentVolume` created by the above configuration.

10. Run the following command to make the `PersistentVolume` backing store the default for all `ObjectBucketClaims` issued by the Red Hat Quay Operator:

```
$ oc patch bucketclass noobaa-default-bucket-class --patch '{"spec": {"placementPolicy": {"tiers": [{"backingStores": [{"noobaa-pv-backing-store"]}]}}}}' --type merge -n openshift-storage
```
CHAPTER 4. CONFIGURING TRAFFIC INGRESS

4.1. CONFIGURING SSL/TLS AND ROUTES

Support for OpenShift Container Platform Edge-Termination Routes has been added by way of a new managed component, `tls`. This separates the `route` component from SSL/TLS and allows users to configure both separately.

**EXTERNAL_TLS_TERMINATION**: `true` is the opinionated setting.

- Managed `tls` means that the default cluster wildcard certificate is used.
- Unmanaged `tls` means that the user provided key and certificate pair is be injected into the `Route`.

The `ssl.cert` and `ssl.key` are now moved to a separate, persistent secret, which ensures that the key and certificate pair are not re-generated upon every reconcile. The key and certificate pair are now formatted as `edge` routes and mounted to the same directory in the Quay container.

Multiple permutations are possible when configuring SSL/TLS and Routes, but the following rules apply:

- If SSL/TLS is **managed**, then your route must also be **managed**
- If SSL/TLS is **unmanaged**, then you must supply certificates, either with the config tool or directly in the config bundle

The following table describes the valid options:

<table>
<thead>
<tr>
<th>Option</th>
<th>Route</th>
<th>TLS</th>
<th>Certs provided</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>My own load balancer handles TLS</td>
<td>Managed</td>
<td>Managed</td>
<td>No</td>
<td>Edge Route with default wildcard cert</td>
</tr>
<tr>
<td>Red Hat Quay handles TLS</td>
<td>Managed</td>
<td>Unmanaged</td>
<td>Yes</td>
<td>Passthrough route with certs mounted inside the pod</td>
</tr>
<tr>
<td>Red Hat Quay handles TLS</td>
<td>Unmanaged</td>
<td>Unmanaged</td>
<td>Yes</td>
<td>Certificates are set inside the quay pod but route must be created manually</td>
</tr>
</tbody>
</table>

**NOTE**

Red Hat Quay 3.7 does not support builders when TLS is managed by the Operator.

4.1.1. Creating the config bundle secret with the SSL/TLS cert and key pair

Use the following procedure to create a config bundle secret that includes your own SSL/TLS certificate and key pair.
Procedure

- Enter the following command to create config bundle secret that includes your own SSL/TLS certificate and key pair:

  ```bash
  $ oc create secret generic --from-file config.yaml=./config.yaml --from-file ssl.cert=./ssl.cert --from-file ssl.key=./ssl.key config-bundle-secret
  ```
CHAPTER 5. CONFIGURING THE DATABASE

5.1. USING AN EXISTING POSTGRESQL DATABASE

If you are using an externally managed PostgreSQL database, you must manually enable the `pg_trgm` extension for a successful deployment.

Use the following procedure to deploy an existing PostgreSQL database.

Procedure

1. Create a `config.yaml` file with the necessary database fields. For example:

   **Example config.yaml file:**
   ```yaml
   DB_URI: postgresql://test-quay-database:postgres@test-quay-database:5432/test-quay-database
   ```

2. Create a Secret using the configuration file:

   ```bash
   $ kubectl create secret generic --from-file config.yaml=./config.yaml config-bundle-secret
   ```

3. Create a QuayRegistry YAML file which marks the `postgres` component as `unmanaged` and references the created Secret. For example:

   **Example quayregistry.yaml file**
   ```yaml
   apiVersion: quay.redhat.com/v1
   kind: QuayRegistry
   metadata:
     name: example-registry
     namespace: quay-enterprise
   spec:
     configBundleSecret: config-bundle-secret
     components:
     - kind: postgres
       managed: false
   ```

4. Deploy the registry as detailed in the following sections.

5.1.1. Database configuration

This section describes the database configuration fields available for Red Hat Quay deployments.

5.1.1.1. Database URI

With Red Hat Quay, connection to the database is configured by using the required `DB_URI` field.

The following table describes the `DB_URI` configuration field:

**Table 5.1. Database URI**
5.1.1.2. Database connection arguments

Optional connection arguments are configured by the `DB_CONNECTION_ARGS` parameter. Some of the key-value pairs defined under `DB_CONNECTION_ARGS` are generic, while others are database specific.

The following table describes database connection arguments:

**Table 5.2. Database connection arguments**

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>DB_CONNECTION_ARGS</code></td>
<td>Object</td>
<td>Optional connection arguments for the database, such as timeouts and SSL/TLS.</td>
</tr>
<tr>
<td>.autrollback</td>
<td>Boolean</td>
<td>Whether to use thread-local connections. Should always be <code>true</code>.</td>
</tr>
<tr>
<td>.threadlocals</td>
<td>Boolean</td>
<td>Whether to use auto-rollback connections. Should always be <code>true</code>.</td>
</tr>
</tbody>
</table>

5.1.1.2.1. PostgreSQL SSL/TLS connection arguments

With SSL/TLS, configuration depends on the database you are deploying. The following example shows a PostgreSQL SSL/TLS configuration:

```yaml
DB_CONNECTION_ARGS:
  sslmode: verify-ca
  sslrootcert: /path/to/cacert
```

The `sslmode` option determines whether, or with, what priority a secure SSL/TLS TCP/IP connection will be negotiated with the server. There are six modes:

**Table 5.3. SSL/TLS options**

<table>
<thead>
<tr>
<th>SSL/TLS Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sslmode</code></td>
<td>Determines whether, or with, what priority a secure SSL/TLS TCP/IP connection will be negotiated with the server. There are six modes:</td>
</tr>
<tr>
<td>Mode</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>disable</td>
<td>Your configuration only tries non-SSL/TLS connections.</td>
</tr>
<tr>
<td>allow</td>
<td>Your configuration first tries a non-SSL/TLS connection. Upon failure, tries an SSL/TLS connection.</td>
</tr>
<tr>
<td>prefer</td>
<td>(Default) Your configuration first tries an SSL/TLS connection. Upon failure, tries a non-SSL/TLS connection.</td>
</tr>
<tr>
<td>require</td>
<td>Your configuration only tries an SSL/TLS connection. If a root CA file is present, it verifies the certificate in the same way as if verify-ca was specified.</td>
</tr>
<tr>
<td>verify-ca</td>
<td>Your configuration only tries an SSL/TLS connection, and verifies that the server certificate is issued by a trusted certificate authority (CA).</td>
</tr>
<tr>
<td>verify-full</td>
<td>Only tries an SSL/TLS connection, and verifies that the server certificate is issued by a trusted CA and that the requested server hostname matches that in the certificate.</td>
</tr>
</tbody>
</table>

For more information on the valid arguments for PostgreSQL, see Database Connection Control Functions.

5.1.2. Using the managed PostgreSQL database

With Red Hat Quay 3.9, if your database is managed by the Red Hat Quay Operator, updating from Red Hat Quay 3.8 → 3.9 automatically handles upgrading PostgreSQL 10 to PostgreSQL 13.
IMPORTANT

- Users with a managed database are required to upgrade their PostgreSQL database from 10 → 13.
- If your Red Hat Quay and Clair databases are managed by the Operator, the database upgrades for each component must succeed for the 3.9.0 upgrade to be successful. If either of the database upgrades fail, the entire Red Hat Quay version upgrade fails. This behavior is expected.

If you do not want the Red Hat Quay Operator to upgrade your PostgreSQL deployment from PostgreSQL 10 → 13, you must set the PostgreSQL parameter to managed: false in your quayregistry.yaml file. For more information about setting your database to unmanaged, see Using an existing Postgres database.

IMPORTANT

- It is highly recommended that you upgrade to PostgreSQL 13. PostgreSQL 10 had its final release on November 10, 2022 and is no longer supported. For more information, see the PostgreSQL Versioning Policy.

If you want your PostgreSQL database to match the same version as your Red Hat Enterprise Linux (RHEL) system, see Migrating to a RHEL 8 version of PostgreSQL for RHEL 8 or Migrating to a RHEL 9 version of PostgreSQL for RHEL 9.

For more information about the Red Hat Quay 3.8 → 3.9 procedure, see “Updating Red Hat Quay and the Red Hat Quay and Clair PostgreSQL databases on OpenShift Container Platform”.

5.1.2.1. PostgreSQL database recommendations

The Red Hat Quay team recommends the following for managing your PostgreSQL database.

- Database backups should be performed regularly using either the supplied tools on the PostgreSQL image or your own backup infrastructure. The Red Hat Quay Operator does not currently ensure that the PostgreSQL database is backed up.
- Restoring the PostgreSQL database from a backup must be done using PostgreSQL tools and procedures. Be aware that your Quay pods should not be running while the database restore is in progress.
- Database disk space is allocated automatically by the Red Hat Quay Operator with 50 GiB. This number represents a usable amount of storage for most small to medium Red Hat Quay installations but might not be sufficient for your use cases. Resizing the database volume is currently not handled by the Red Hat Quay Operator.

5.2. CONFIGURING EXTERNAL REDIS

Use the content in this section to set up an external Redis deployment.

5.2.1. Using an unmanaged Redis database

Use the following procedure to set up an external Redis database.

Procedure
1. Create a `config.yaml` file using the following Redis fields:

```yaml
BUILDLOGS_REDIS:
  host: quay-server.example.com
  port: 6379
  ssl: false

USER_EVENTS_REDIS:
  host: quay-server.example.com
  port: 6379
  ssl: false
```

2. Enter the following command to create a secret using the configuration file:

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

3. Create a `quayregistry.yaml` file that sets the Redis component to **unmanaged** and references the created secret:

```yaml
apiVersion: quay.redhat.com/v1
kind: QuayRegistry
metadata:
  name: example-registry
  namespace: quay-enterprise
spec:
  configBundleSecret: config-bundle-secret
  components:
    - kind: redis
      managed: false
```

4. Deploy the Red Hat Quay registry.

**Additional resources**

**Redis configuration fields**

**5.2.2. Using unmanaged Horizontal Pod Autoscalers**

Horizontal Pod Autoscalers (HPAs) are now included with the **Clair**, **Quay**, and **Mirror** pods, so that they now automatically scale during load spikes.

As HPA is configured by default to be managed, the number of **Clair**, **Quay**, and **Mirror** pods is set to two. This facilitates the avoidance of downtime when updating or reconfiguring Red Hat Quay by the Operator or during rescheduling events.

**5.2.2.1. Disabling the Horizontal Pod Autoscaler**

To disable autoscaling or create your own **HorizontalPodAutoscaler**, specify the component as **unmanaged** in the **QuayRegistry** instance. For example:

```yaml
apiVersion: quay.redhat.com/v1
kind: QuayRegistry
metadata:
  name: example-registry
```
5.2.3. Disabling the Route component

Use the following procedure to prevent the Red Hat Quay Operator from creating a route.

**Procedure**

1. Set the component as `managed: false` in the `quayregistry.yaml` file:

```yaml
apiVersion: quay.redhat.com/v1
kind: QuayRegistry
metadata:
  namespace: quay-enterprise
spec:
  components:
    - kind: horizontalpodautoscaler
      managed: false
```

2. Edit the `config.yaml` file to specify that Red Hat Quay handles SSL/TLS. For example:

```yaml
EXTERNAL_TLS_TERMINATION: false
SERVER_HOSTNAME: example-registry-quay-quay-enterprise.apps.user1.example.com
PREFERRED_URL_SCHEME: https
```

If you do not configure the unmanaged route correctly, the following error is returned:

```
{
  "kind": "QuayRegistry",
  "namespace": "quay-enterprise",
  "name": "example-registry",
  "uid": "d5879ba5-cc92-406c-ba62-8b19cf56d4aa",
  "apiVersion": "quay.redhat.com/v1",
  "resourceVersion": "2418527"
},
"reason": "ConfigInvalid",
"message": "required component `route` marked as unmanaged, but `configBundleSecret` is missing necessary fields"
}
```
NOTE

Disabling the default route means you are now responsible for creating a Route, Service, or Ingress in order to access the Red Hat Quay instance. Additionally, whatever DNS you use must match the SERVER_HOSTNAME in the Red Hat Quay config.

5.2.4. Disabling the monitoring component

If you install the Red Hat Quay Operator in a single namespace, the monitoring component is automatically set to managed: false. Use the following reference to explicitly disable monitoring.

Unmanaged monitoring

```yaml
apiVersion: quay.redhat.com/v1
kind: QuayRegistry
metadata:
  name: example-registry
  namespace: quay-enterprise
spec:
  components:
  - kind: monitoring
    managed: false
```

To enable monitoring in this scenario, see Enabling monitoring when the Red Hat Quay Operator is installed in a single namespace.

5.2.5. Disabling the mirroring component

To disable mirroring explicitly, use the following YAML configuration:

Unmanaged mirroring example YAML configuration

```yaml
apiVersion: quay.redhat.com/v1
kind: QuayRegistry
metadata:
  name: example-registry
  namespace: quay-enterprise
spec:
  components:
  - kind: mirroring
    managed: false
```
CHAPTER 6. DEPLOYING THE RED HAT QUAY OPERATOR

The Red Hat Quay Operator can be deployed from the command line or from the OpenShift Container Platform console, however the steps are fundamentally the same.

6.1. DEPLOYING RED HAT QUAY FROM THE COMMAND LINE

Use the following procedure to deploy Red Hat Quay from using the command-line interface (CLI).

Prerequisites

- You have logged into OpenShift Container Platform using the CLI.

Procedure

1. Create a namespace, for example, quay-enterprise, by entering the following command:

   ```
   $ oc new-project quay-enterprise
   ```

2. Optional. If you want to pre-configure any aspects of your Red Hat Quay deployment, create a Secret for the config bundle:

   ```
   $ oc create secret generic quay-enterprise-config-bundle --from-file=config-bundle.tar.gz=/path/to/config-bundle.tar.gz
   ```

3. Create a QuayRegistry custom resource in a file called quayregistry.yaml

   a. For a minimal deployment, using all the defaults:

      ```
      quayregistry.yaml:
      ```
      ```
      apiVersion: quay.redhat.com/v1
      kind: QuayRegistry
      metadata:
        name: example-registry
        namespace: quay-enterprise
      ```

   b. Optional. If you want to have some components unmanaged, add this information in the spec field. A minimal deployment might look like the following example:

      ```
      Example quayregistry.yaml with unmanaged components
      ```
      ```
      apiVersion: quay.redhat.com/v1
      kind: QuayRegistry
      metadata:
        name: example-registry
        namespace: quay-enterprise
      spec:
        components:
          - kind: clair
            managed: false
          - kind: horizontalpodautoscaler
            managed: false
      ```
c. Optional. If you have created a config bundle, for example, `init-config-bundle-secret`, reference it in the `quayregistry.yaml` file:

**Example quayregistry.yaml with a config bundle**

```yaml
apiVersion: quay.redhat.com/v1
kind: QuayRegistry
metadata:
  name: example-registry
  namespace: quay-enterprise
spec:
  configBundleSecret: init-config-bundle-secret
```

d. Optional. If you have a proxy configured, you can add the information using overrides for Red Hat Quay, Clair, and mirroring:

**Example quayregistry.yaml with proxy configured**

```yaml
kind: QuayRegistry
metadata:
  name: quay37
spec:
  configBundleSecret: config-bundle-secret
  components:
    - kind: objectstorage
      managed: false
    - kind: route
      managed: true
    - kind: mirror
      managed: true
  overrides:
    env:
      - name: DEBUGLOG
        value: "true"
      - name: HTTP_PROXY
        value: quayproxy.qe.devcluster.openshift.com:3128
      - name: HTTPS_PROXY
        value: quayproxy.qe.devcluster.openshift.com:3128
      - name: NO_PROXY
        value: svc.cluster.local,localhost,quay370.apps.quayperf370.perfscale.devcluster.openshift.com
    - kind: tls
      managed: false
    - kind: clair
      managed: true
    overrides:
      env:
        - name: HTTP_PROXY
          value: quayproxy.qe.devcluster.openshift.com:3128
        - name: HTTPS_PROXY
          value: quayproxy.qe.devcluster.openshift.com:3128
```
4. Create the **QuayRegistry** in specified namespace:

   ```bash
   $ oc create -n quay-enterprise -f quayregistry.yaml
   ```

5. Enter the following command to see when the **status.registryEndpoint** is populated:

   ```bash
   $ oc get quayregistry -n quay-enterprise example-registry -o jsonpath="
   {.status.registryEndpoint}" -w
   ```

**Additional resources**

- For more information about how to track the progress of your Red Hat Quay deployment, see [Monitoring and debugging the deployment process](#).

### 6.1.1. Viewing created components using the command line

Use the following procedure to view deployed Red Hat Quay components.

**Prerequisites**

- You have deployed the Red Hat Quay Operator on `{ocp}`.

**Procedure**

1. Enter the following command to view the deployed components:

   ```bash
   $ oc get pods -n quay-enterprise
   ```

**Example output**

<table>
<thead>
<tr>
<th>NAME</th>
<th>READY</th>
<th>STATUS</th>
<th>RESTARTS</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>example-registry-clair-app-5ffc9f77d6-jwr9sb</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>3m42s</td>
</tr>
<tr>
<td>example-registry-clair-app-5ffc9f77d6-wgp7dd</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>3m41s</td>
</tr>
<tr>
<td>example-registry-clair-postgres-54956d6d9c-rgs8l</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>3m5s</td>
</tr>
<tr>
<td>example-registry-quay-app-79c6b86c7b-8qnr2</td>
<td>1/1</td>
<td>Running</td>
<td>4</td>
<td>3m42s</td>
</tr>
</tbody>
</table>
### 6.1.2. Horizontal Pod Autoscaling

A default deployment shows the following running pods:

- Two pods for the Red Hat Quay application itself (`example-registry-quay-app-*`)
- One Redis pod for Red Hat Quay logging (`example-registry-quay-redis-*`)
- One database pod for PostgreSQL used by Red Hat Quay for metadata storage (`example-registry-quay-database-*`)
- One pod for the Red Hat Quay config editor (`example-registry-quay-config-editor-*`)
- Two Quay mirroring pods (`example-registry-quay-mirror-*`)
- Two pods for the Clair application (`example-registry-clair-app-*`)
- One PostgreSQL pod for Clair (`example-registry-clair-postgres-*`)

Horizontal Pod Autoscaling is configured by default to be managed, and the number of pods for Quay, Clair and repository mirroring is set to two. This facilitates the avoidance of downtime when updating or reconfiguring Red Hat Quay through the Red Hat Quay Operator or during rescheduling events. You can enter the following command to view information about HPA objects:

```bash
$ oc get hpa -n quay-enterprise
```

**Example output**

<table>
<thead>
<tr>
<th>NAME</th>
<th>REFERENCE</th>
<th>TARGETS</th>
<th>MINPODS</th>
<th>MAXPODS</th>
</tr>
</thead>
<tbody>
<tr>
<td>example-registry-clair-app</td>
<td>Deployment/example-registry-clair-app</td>
<td>16%/90%, 0%/90%</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>example-registry-quay-app</td>
<td>Deployment/example-registry-quay-app</td>
<td>31%/90%, 1%/90%</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>example-registry-quay-mirror</td>
<td>Deployment/example-registry-quay-mirror</td>
<td>27%/90%, 0%/90%</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

### 6.1.2.1. Using the API to create the first user

Use the following procedure to create the first user in your Red Hat Quay organization.

**Prerequisites**

- The config option `FEATURE_USER_INITIALIZE` must be set to `true`. 
No users can already exist in the database.

**PROCEDURE**

This procedure requests an OAuth token by specifying "access_token": true.

1. As the root user, install **python39** by entering the following command:

   ```
   $ sudo yum install python39
   ```

2. Upgrade the **pip** package manager for Python 3.9:

   ```
   $ python3.9 -m pip install --upgrade pip
   ```

3. Use the **pip** package manager to install the **bcrypt** package:

   ```
   $ pip install bcrypt
   ```

4. Generate a secure, hashed password using the **bcrypt** package in Python 3.9 by entering the following command:

   ```
   $ python3.9 -c 'import bcrypt; print(bcrypt.hashpw(b"subquay12345", bcrypt.gensalt(12)).decode("utf-8"))'
   ```

5. Open your Red Hat Quay configuration file and update the following configuration fields:

   ```
   FEATURE_USER_INITIALIZE: true
   SUPER_USERS:
   - quayadmin
   ```

6. Stop the Red Hat Quay service by entering the following command:

   ```
   $ sudo podman stop quay
   ```

7. Start the Red Hat Quay service by entering the following command:

   ```
   $ sudo podman run -d -p 80:8080 -p 443:8443 --name=quay -v $QUAY/config:/conf/stack:Z
     -v $QUAY/storage:/datastorage:Z {productrepo}/{quayimage}:{productminv}
   ```

8. Run the following **CURL** command to generate a new user with a username, password, email, and access token:

   ```
   $ curl -X POST -k http://quay-server.example.com/api/v1/user/initialize --header 'Content-Type: application/json' --data '{ "username": "quayadmin", "password": "quaypass12345", "email": "quayadmin@example.com", "access_token": true}'
   ```

If successful, the command returns an object with the username, email, and encrypted password. For example:

```json
{
    "access_token": "6B4QTRSTSD1HMIG915VPX7BMEZBV89GPNY2FC2ED",
    "email": "quayadmin@example.com",
    "encrypted_password": "1nZMLH57RIE5UGdL/yYpDOHlqiNCgimb6W9kfF8MjZ1xrfDpRyRs9NUuNuAitW",
    "username": "quayadmin"
} #
```
If a user already exists in the database, an error is returned:

```json
{"message": "Cannot initialize user in a non-empty database"}
```

If your password is not at least eight characters or contains whitespace, an error is returned:

```json
{"message": "Failed to initialize user: Invalid password, password must be at least 8 characters and contain no whitespace."}
```

9. Log in to your Red Hat Quay deployment by entering the following command:

```bash
$ sudo podman login -u quayadmin -p quaypass12345 http://quay-server.example.com --tls-verify=false
```

**Example output**

```
Login Succeeded!
```

**Additional resources**

For more information on pre-configuring your Red Hat Quay deployment, see the section *Pre-configuring Red Hat Quay for automation*

### 6.1.3. Monitoring and debugging the deployment process

Users can now troubleshoot problems during the deployment phase. The status in the `QuayRegistry` object can help you monitor the health of the components during the deployment and help you debug any problems that may arise.

**Procedure**

1. Enter the following command to check the status of your deployment:

```bash
$ oc get quayregistry -n quay-enterprise -o yaml
```

**Example output**

```
apiVersion: v1
items:
- apiVersion: quay.redhat.com/v1
  kind: QuayRegistry
  metadata:
    creationTimestamp: "2021-09-14T10:51:22Z"
    generation: 3
    name: example-registry
    namespace: quay-enterprise
    resourceVersion: "50147"
    selfLink: /apis/quay.redhat.com/v1/namespaces/quay-enterprise/quayregistries/example-registry
```
Use the `oc get pods` command to view the current state of the deployed components:

```bash
$ oc get pods -n quay-enterprise
```

### Example output

<table>
<thead>
<tr>
<th>NAME</th>
<th>READY</th>
<th>STATUS</th>
<th>RESTARTS</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>example-registry-clair-app-86554c6b49-ds7bl</td>
<td>0/1</td>
<td>ContainerCreating</td>
<td>0</td>
<td>2s</td>
</tr>
<tr>
<td>example-registry-clair-app-86554c6b49-hxp5s</td>
<td>0/1</td>
<td>Running</td>
<td>1</td>
<td>17s</td>
</tr>
<tr>
<td>example-registry-clair-postgres-68d8857899-lbc5n</td>
<td>0/1</td>
<td>ContainerCreating</td>
<td>0</td>
<td>17s</td>
</tr>
<tr>
<td>example-registry-quay-app-upgrade-h2v7h</td>
<td>0/1</td>
<td>ContainerCreating</td>
<td>0</td>
<td>9s</td>
</tr>
<tr>
<td>example-registry-quay-config-editor-5f646cbb7-lbcn</td>
<td>0/1</td>
<td>ContainerCreating</td>
<td>0</td>
<td>17s</td>
</tr>
<tr>
<td>example-registry-quay-database-66f495c9bc-wqsf</td>
<td>0/1</td>
<td>ContainerCreating</td>
<td>0</td>
<td>17s</td>
</tr>
<tr>
<td>example-registry-quay-mirror-854c88457b-d845g</td>
<td>0/1</td>
<td>Init:0/1</td>
<td>0</td>
<td>2s</td>
</tr>
<tr>
<td>example-registry-quay-mirror-854c88457b-fghxv</td>
<td>0/1</td>
<td>Init:0/1</td>
<td>0</td>
<td>17s</td>
</tr>
<tr>
<td>example-registry-quay-postgres-init-bktdd</td>
<td>0/1</td>
<td>Terminating</td>
<td>0</td>
<td>17s</td>
</tr>
<tr>
<td>example-registry-quay-redis-f9b9d44bf-4htpz</td>
<td>0/1</td>
<td>ContainerCreating</td>
<td>0</td>
<td>17s</td>
</tr>
</tbody>
</table>

3. While the deployment is in progress, the **QuayRegistry** object will show the current status. In this instance, database migrations are taking place, and other components are waiting until completion:

```json
status:
  conditions:
  - lastTransitionTime: "2021-09-14T10:52:04Z"
    lastUpdateTime: "2021-09-14T10:52:04Z"
    message: all objects created/updated successfully
    reason: ComponentsCreationSuccess
```
When the deployment process finishes successfully, the status in the QuayRegistry object shows no unhealthy components:

```yaml
status: "False"
type: RolloutBlocked
- lastTransitionTime: "2021-09-14T10:52:05Z"
  lastUpdateTime: "2021-09-14T10:52:05Z"
  message: running database migrations
  reason: MigrationsInProgress
  status: "False"
type: Available
  configEditorCredentialsSecret: example-registry-quay-config-editor-credentials-btbkcg8dc9
  configEditorEndpoint: https://example-registry-quay-config-editor-quay-enterprise.apps.docs.quayteam.org
  lastUpdated: 2021-09-14 10:52:05.371425635 +0000 UTC
  unhealthyComponents:
    clair:
      - lastTransitionTime: "2021-09-14T10:51:32Z"
        lastUpdateTime: "2021-09-14T10:51:32Z"
        message: 'Deployment example-registry-clair-postgres: Deployment does not have minimum availability.'
        reason: MinimumReplicasUnavailable
        status: "False"
type: Available
      - lastTransitionTime: "2021-09-14T10:51:32Z"
        lastUpdateTime: "2021-09-14T10:51:32Z"
        message: 'Deployment example-registry-clair-app: Deployment does not have minimum availability.'
        reason: MinimumReplicasUnavailable
        status: "False"
type: Available
    mirror:
      - lastTransitionTime: "2021-09-14T10:51:32Z"
        lastUpdateTime: "2021-09-14T10:51:32Z"
        message: 'Deployment example-registry-quay-mirror: Deployment does not have minimum availability.'
        reason: MinimumReplicasUnavailable
        status: "False"
type: Available
```

4. When the deployment process finishes successfully, the status in the QuayRegistry object shows no unhealthy components:

```yaml
status:
  conditions:
    - lastTransitionTime: "2021-09-14T10:52:36Z"
      lastUpdateTime: "2021-09-14T10:52:36Z"
      message: all registry component healthchecks passing
      reason: HealthChecksPassing
      status: "True"
type: Available
    - lastTransitionTime: "2021-09-14T10:52:46Z"
      lastUpdateTime: "2021-09-14T10:52:46Z"
      message: all objects created/updated successfully
      reason: ComponentsCreationSuccess
      status: "False"
type: RolloutBlocked
  configEditorCredentialsSecret: example-registry-quay-config-editor-credentials-hg7gg7h57m
```
6.2. DEPLOYING RED HAT QUAY FROM THE OPENS SHIFT CONTAINER PLATFORM CONSOLE

1. Create a namespace, for example, quay-enterprise.

2. Select Operators → Installed Operators, then select the Quay Operator to navigate to the Operator detail view.

3. Click ‘Create Instance’ on the ‘Quay Registry’ tile under ‘Provided APIs’.

4. Optionally change the ‘Name’ of the QuayRegistry. This will affect the hostname of the registry. All other fields have been populated with defaults.

5. Click ‘Create’ to submit the QuayRegistry to be deployed by the Quay Operator.

6. You should be redirected to the QuayRegistry list view. Click on the QuayRegistry you just created to see the details view.

7. Once the ‘Registry Endpoint’ has a value, click it to access your new Quay registry via the UI. You can now select ‘Create Account’ to create a user and sign in.

6.2.1. Using the Red Hat Quay UI to create the first user

Use the following procedure to create the first user by the Red Hat Quay UI.

**NOTE**

This procedure assumes that the FEATURE_USER_CREATION config option has not been set to false. If it is false, the Create Account functionality on the UI will be disabled, and you will have to use the API to create the first user.

**Procedure**

1. In the OpenShift Container Platform console, navigate to Operators → Installed Operators, with the appropriate namespace / project.

2. Click on the newly installed Quay Registry object to view the details. For example:
3. After the **Registry Endpoint** has a value, navigate to this URL in your browser.

4. Select **Create Account** in the Red Hat Quay registry UI to create a user. For example:

![Create Account](image)

5. Enter the details for **Username**, **Password**, **Email**, and then click **Create Account**. For example:
After creating the first user, you are automatically logged in to the Red Hat Quay registry. For example:

![Create new account form](image)

After creating the first user, you are automatically logged in to the Red Hat Quay registry.
CHAPTER 7. VIEWING THE STATUS OF THE QUAYREGISTRY OBJECT

Lifecycle observability for a given Red Hat Quay deployment is reported in the status section of the corresponding QuayRegistry object. The Red Hat Quay Operator constantly updates this section, and this should be the first place to look for any problems or state changes in Red Hat Quay or its managed dependencies.

7.1. VIEWING THE REGISTRY ENDPOINT

Once Red Hat Quay is ready to be used, the status.registryEndpoint field will be populated with the publicly available hostname of the registry.

7.2. VIEWING THE CONFIG EDITOR ENDPOINT

Access Red Hat Quay’s UI-based config editor using status.configEditorEndpoint.

7.3. VIEWING THE CONFIG EDITOR CREDENTIALS SECRET

The username and password for the config editor UI will be stored in a Secret in the same namespace as the QuayRegistry referenced by status.configEditorCredentialsSecret.

7.4. VIEWING THE VERSION OF RED HAT QUAY IN USE

The current version of Red Hat Quay that is running will be reported in status.currentVersion.

7.5. VIEWING THE CONDITIONS OF YOUR RED HAT QUAY DEPLOYMENT

Certain conditions will be reported in status.conditions.
CHAPTER 8. UPDATING RED HAT QUAY AND THE RED HAT QUAY AND CLAIR POSTGRESQL DATABASES ON OPENSIFT CONTAINER PLATFORM

IMPORTANT

If your Red Hat Quay deployment is upgrading from one y-stream to the next, for example, from 3.8.10 → 3.8.11, you must not switch the upgrade channel from stable-3.8 to stable-3.9. Changing the upgrade channel in the middle of a y-stream upgrade will disallow Red Hat Quay from upgrading to 3.9. This is a known issue and will be fixed in a future version of Red Hat Quay.

When updating Red Hat Quay 3.8 → 3.9, the Operator automatically upgrades the existing PostgreSQL databases for Clair and Red Hat Quay from version 10 to version 13.

IMPORTANT

- Users with a managed database are required to upgrade their PostgreSQL database from 10 → 13.

- If your Red Hat Quay and Clair databases are managed by the Operator, the database upgrades for each component must succeed for the 3.9.0 upgrade to be successful. If either of the database upgrades fail, the entire Red Hat Quay version upgrade fails. This behavior is expected.

You can update Red Hat Quay and the Red Hat Quay and Clair PostgreSQL databases on OpenShift Container Platform by using the Web Console UI, or by using the CLI.

8.1. UPDATING RED HAT QUAY AND THE RED HAT QUAY AND CLAIR POSTGRESQL DATABASES USING THE OPENSIFT CONTAINER PLATFORM WEB CONSOLE

Use the following procedure to update Red Hat Quay and the Red Hat Quay and Clair PostgreSQL databases using the OpenShift Container Platform web console.

IMPORTANT

- This upgrade is irreversible. It is highly recommended that you upgrade to PostgreSQL 13. PostgreSQL 10 had its final release on November 10, 2022 and is no longer supported. For more information, see the PostgreSQL Versioning Policy.

- If your Red Hat Quay and Clair databases are managed by the Operator, the database upgrades for each component must succeed for the 3.9.0 upgrade to be successful. If either of the database upgrades fail, the entire Red Hat Quay version upgrade fails. This behavior is expected.

- By default, Red Hat Quay is configured to remove old persistent volume claims (PVCs) from PostgreSQL 10. To disable this setting and backup old PVCs, you must set POSTGRES_UPGRADE_RETAIN_BACKUP to True in your quay-operator Subscription object.
Prerequisites

- You have installed Red Hat Quay 3.6, 3.7, or 3.8 on OpenShift Container Platform.

- 100 GB of free, additional storage. During the upgrade process, additional persistent volume claims (PVCs) are provisioned to store the migrated data. This helps prevent a destructive operation on user data. The upgrade process rolls out PVCs for 50 GB for both the Red Hat Quay database upgrade, and the Clair database upgrade.

Procedure

1. Optional. Back up your old PVCs from PostgreSQL 10 by setting `POSTGRES_UPGRADE_RETAIN_BACKUP` to `True` your `quay-operator Subscription` object. For example:

   ```yaml
   apiVersion: operators.coreos.com/v1alpha1
   kind: Subscription
   metadata:
     name: quay-operator
     namespace: quay-enterprise
   spec:
     channel: stable-3.8
     name: quay-operator
     source: redhat-operators
     sourceNamespace: openshift-marketplace
     config:
       env:
         - name: POSTGRES_UPGRADE_RETAIN_BACKUP
           value: "true"
   ```

2. In the OpenShift Container Platform Web Console, navigate to `Operators → Installed Operators`.

3. Click on the Red Hat Quay Operator.

4. Navigate to the `Subscription` tab.

5. Under `Subscription details` click `Update channel`.

6. Select `stable-3.9` and save the changes.

7. Check the progress of the new installation under `Upgrade status`. Wait until the upgrade status changes to `1 installed` before proceeding.

8. In your OpenShift Container Platform cluster, navigate to `Workloads → Pods`. Existing pods should be terminated, or in the process of being terminated.

9. Wait for the following pods, which are responsible for upgrading the database and alembic migration of existing data, to spin up: `clair-postgres-upgrade`, `quay-postgres-upgrade`, and `quay-app-upgrade`.

10. After the `clair-postgres-upgrade`, `quay-postgres-upgrade`, and `quay-app-upgrade` pods are marked as `Completed`, the remaining pods for your Red Hat Quay deployment spin up. This takes approximately ten minutes.
11. Verify that the quay-database and clair-postgres pods now use the postgresql-13 image.

12. After the quay-app pod is marked as Running, you can reach your Red Hat Quay registry.

8.2. UPDATING RED HAT QUAY AND THE RED HAT QUAY AND CLAIR POSTGRESQL DATABASES USING THE CLI

Use the following procedure to update Red Hat Quay and the Red Hat Quay and Clair PostgreSQL databases using the command-line interface (CLI).

**IMPORTANT**

- This upgrade is irreversible. It is highly recommended that you upgrade to PostgreSQL 13. PostgreSQL 10 had its final release on November 10, 2022 and is no longer supported. For more information, see the PostgreSQL Versioning Policy.

- By default, Red Hat Quay is configured to remove old persistent volume claims (PVCs) from PostgreSQL 10. To disable this setting and backup old PVCs, you must set POSTGRES_UPGRADE_RETAIN_BACKUP to True in your quay-operator Subscription object.

Prerequisites

- You have installed Red Hat Quay 3.6, 3.7, or 3.8 on OpenShift Container Platform.

- 100 GB of free, additional storage. During the upgrade process, additional persistent volume claims (PVCs) are provisioned to store the migrated data. This helps prevent a destructive operation on user data. The upgrade process rolls out PVCs for 50 GB for both the Red Hat Quay database upgrade, and the Clair database upgrade.

Procedure

1. Retrieve your quay-operator configuration file by entering the following oc get command:

   ```
   $ oc get subscription quay-operator -n quay-enterprise -o yaml > quay-operator.yaml
   ```

2. Retrieve the latest version of the Red Hat Quay Operator and its channel by entering the following command:

   ```
   oc get packagemanifests quay-operator \
   -o jsonpath="{range .status.channels[*]}{@.currentCSV} {@.name}{"\n"}{end}" \
   | awk '{print "STARTING_CSV=" $1 " CHANNEL=" $2 }' \
   | sort -nr \
   | head -1
   ```

   **Example output**

   ```
   STARTING_CSV=quay-operator.v3.9.0 CHANNEL=stable-3.9
   ```

3. Using the output from the previous command, update your Subscription custom resource for the Red Hat Quay Operator and save it as quay-operator.yaml. For example:
Specify the value you obtained in the previous step for the `spec.channel` parameter.

Optional. Back up your old PVCs from PostgreSQL 10 by setting `POSTGRES_UPGRADE_RETAIN_BACKUP` to `True` your `quay-operator Subscription` object.

4. Enter the following command to apply the configuration:

```bash
$ oc apply -f quay-operator.yaml
```

Example output

```
subscription.operators.coreos.com/quay-operator created
```

Next steps

- Red Hat Quay features