Red Hat

Red Hat Quay 3.11

Deploying the Red Hat Quay Operator on OpenShift Container Platform

Deploying the Red Hat Quay Operator on OpenShift Container Platform

Last Updated: 2024-04-11

Red Hat Quay 3.11 Deploying the Red Hat Quay Operator on OpenShift Container Platform

Deploying the Red Hat Quay Operator on OpenShift Container Platform

Legal Notice

Copyright © 2024 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at
http://creativecommons.org/licenses/by-sa/3.0/
. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

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

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

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

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

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

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

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

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

All other trademarks are the property of their respective owners.

Abstract

Deploy the Red Hat Quay Operator on an OpenShift Container Platform cluster

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.

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, such as 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.
- 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.

CHAPTER 1. INTRODUCTION TO THE RED HAT QUAY OPERATOR

Use the content in this chapter to execute the following:

- Install Red Hat Quay on OpenShift Container Platform using 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 Red Hat Quay
- Upgrade the Red Hat Quay registry by using the Red Hat Quay Operator

1.1. RED HAT QUAY OPERATOR COMPONENTS

Red Hat Quay has many dependencies. These dependencies 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:

Example **QuayRegistry** resource

```
apiVersion: quay.redhat.com/v1
kind: QuayRegistry
metadata:
  name: example-registry
  namespace: quay-enterprise
  spec:
    configBundleSecret: config-bundle-secret
    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 deployment of Red Hat Quay on OpenShift Container Platform, 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 Red Hat OpenShift Data Foundation.
- **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 notifications 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, or 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, or the **config.yaml** file. Then, they must be referenced in your **QuayRegistry** bundle as a Kubernetes **Secret** while indicating which components are unmanaged.

NOTE

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** resource. This **Secret** must contain a **config.yaml** key/value pair.

The **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

Consider the following prerequisites prior to deploying Red Hat Quay on OpenShift Container Platform using the Red Hat Quay Operator.

1.5.1. OpenShift Container Platform cluster

To deploy the Red Hat Quay Operator, you must have an OpenShift Container Platform 4.5 or later cluster and access to an administrative account. The administrative 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 is used as an example throughout this documentation.

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)
- Microsoft Azure Blob Storage
- Google Cloud Storage
- Ceph Object Gateway (RADOS)
- OpenStack Swift
- CloudFront + S3

1.5.4. StorageClass

When deploying **Quay** and **Clair** PostgreSQL databases using the Red Hat Quay Operator, a default **StorageClass** is configured in your cluster.

The default **StorageClass** used by the Red Hat Quay Operator provisions the Persistent Volume Claims required by the **Quay** and **Clair** databases. These PVCs are used to store data persistently, ensuring that your Red Hat Quay registry and Clair vulnerability scanner remain available and maintain their state across restarts or failures.

Before proceeding with the installation, verify that a default **StorageClass** is configured in your cluster to ensure seamless provisioning of storage for **Quay** and **Clair** components.

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

1. Using the OpenShift Container Platform console, select **Operators → OperatorHub**.
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-3.11** for the latest release.
    b. **Installation Mode:**
        i. Choose **All namespaces on the cluster** if you want the Red Hat Quay Operator to be available cluster-wide. 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.
        ii. Choose **A specific namespace on the cluster** if you want it deployed only within a single namespace.
            - **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. Use the following reference for a minimal configuration:

```
$ touch config.yaml
```

```
AUTHENTICATION_TYPE: Database
BUILDLOGS_REDIS:
  host: <quay-server.example.com>
  password: <strongpassword>
  port: 6379
  ssl: false
DATABASE_SECRET_KEY: <0ce4f796-c295-415b-bf9d-b315114704b8>
DB_URI: <postgresql://quayuser:quaypass@quay-server.example.com:5432/quay>
DEFAULT_TAG_EXPIRATION: 2w
DISTRIBUTED_STORAGE_CONFIG:
  default:
    - LocalStorage
    - storage_path: /datastorage/registry
DISTRIBUTED_STORAGE_DEFAULT_LOCATIONS: []
DISTRIBUTED_STORAGE_PREFERENCE:
  - default
PREFERRED_URL_SCHEME: http
SECRET_KEY: <e8f9fe68-1f84-48a8-a05f-02d72e6eccba>
SERVER_HOSTNAME: <quay-server.example.com>
SETUP_COMPLETE: true
TAG_EXPIRATION_OPTIONS:
  - 0s
  - 1d
  - 1w
  - 2w
  - 4w
USER_EVENTS_REDIS:
  host: <quay-server.example.com>
  port: 6379
  ssl: false
```

2. Create a **Secret** using the configuration file by entering the following command:

```
$ 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

```
apiVersion: quay.redhat.com/v1
kind: QuayRegistry
metadata:
  name: example-registry
  namespace: quay-enterprise
spec:
  configBundleSecret: <config_bundle_secret>
  components:
    - kind: objectstorage
      managed: false
# ...
```

4. Enter the following command to deploy the registry by using the **quayregistry.yaml** file:

```
$ 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** configuration 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:

```
# ...
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:

```
# ...
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 3.11

To use new Red Hat Quay 3.11 functions, enable some or all of the following features:

```
# ...
FEATURE_UI_V2: true
FEATURE_UI_V2_REPO_SETTINGS: true
FEATURE_AUTO_PRUNE: true
ROBOTS_DISALLOW: false
# ...
```

3.1.6. Suggested configuration for automation

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

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

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: ABCDEFGHIJKLMN
      s3_secret_key: OL3ABCDEFGHIJKLMN
      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.

```
DISTRIBUTED_STORAGE_CONFIG:
    googleCloudStorage:
        - GoogleCloudStorage
        - access_key: GOOGQIMFB3ABCDEFGHIJKLMN
          bucket_name: quay-bucket
          secret_key: FhDAYe2HeuAKfvZCAGyOioNaaRABCDEFGHIJKLMN
          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 1
DISTRIBUTED_STORAGE_DEFAULT_LOCATIONS: []
DISTRIBUTED_STORAGE_PREFERENCE:
    - azureStorage
```

1 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: #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: #must contain name of the storage config
    - radosGWStorage
```

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
DISTRIBUTED_STORAGE_DEFAULT_LOCATIONS: []
DISTRIBUTED_STORAGE_PREFERENCE:
    - swiftStorage
```

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: WmrXtSGk8B3nABCDEFGH
      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: X9P5SDGJtmSuHFCMSLMbdNCMfUABCDEFGH+C5QD
      storage_path: /datastorage/registry
DISTRIBUTED_STORAGE_DEFAULT_LOCATIONS: []
DISTRIBUTED_STORAGE_PREFERENCE:
  - default
```

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:

```
DISTRIBUTED_STORAGE_CONFIG:
  default:
    - RHOCSStorage
    - access_key: WmrXtSGk8B3nABCDEFGH
      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: X9P5SDGJtmSuHFCMSLMbdNCMfUABCDEFGH+C5QD
      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 **OperatorHub** 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:

Volume Mode	Filesystem is selected by default. Always ensure that Filesystem is selected for Volume Mode.
Device Type	Select one or more device type from the dropdown list.
Disk Size	Set a minimum size of 100GB for the device and maximum available size of the device that needs to be included.
Maximum Disks Limit	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.

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

   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:

```
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:

```
$ 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:

```
$ oc get -n openshift-storage noobaas noobaa -w
```

Example output

```
NAME    MGMT-ENDPOINTS            S3-ENDPOINTS             IMAGE
PHASE   AGE
noobaa  [https://10.0.32.3:30318]  [https://10.0.32.3:31958]  registry.redhat.io/ocs4/mcg-
core-
rhel8@sha256:56624aa7dd4ca178c1887343c7445a9425a841600b1309f6deace37ce6b8678d
Ready   3d18h
```

8. Configure a backing store for the gateway by creating the following YAML file, named **noobaa-pv-backing-store.yaml**:

```
apiVersion: noobaa.io/v1alpha1
kind: BackingStore
metadata:
  finalizers:
  - noobaa.io/finalizer
  labels:
    app: noobaa
  name: noobaa-pv-backing-store
  namespace: openshift-storage
spec:
  pvPool:
    numVolumes: 1
    resources:
      requests:
        storage: 50Gi 1
    storageClass: STORAGE-CLASS-NAME 2
  type: pv-pool
```

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 have 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.

NOTE

- 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 regenerated 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 directly in the config bundle.

The following table describes the valid options:

Table 4.1. Valid configuration options for TLS and routes

Option	Route	TLS	Certs provided	Result
My own load balancer handles TLS	Managed	Managed	No	Edge route with default wildcard cert
Red Hat Quay handles TLS	Managed	Unmanaged	Yes	Passthrough route with certs mounted inside the pod
Red Hat Quay handles TLS	Unmanaged	Unmanaged	Yes	Certificates are set inside of the **quay** pod, but the route must be created manually

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:

```
$ 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 RESOURCES FOR MANAGED COMPONENTS ON OPENSHIFT CONTAINER PLATFORM

You can manually adjust the resources on Red Hat Quay on OpenShift Container Platform for the following components that have running pods:

- **quay**
- **clair**
- **mirroring**
- **clairpostgres**
- **postgres**

This feature allows users to run smaller test clusters, or to request more resources upfront in order to avoid partially degraded **Quay** pods. Limitations and requests can be set in accordance with Kubernetes resource units.

The following components should not be set lower than their minimum requirements. This can cause issues with your deployment and, in some cases, result in failure of the pod's deployment.

- **quay**: Minimum of 6 GB, 2vCPUs
- **clair**: Recommended of 2 GB memory, 2 vCPUs
- **clairpostgres**: Minimum of 200 MB

You can configure resource requests on the OpenShift Container Platform UI, or by directly by updating the **QuayRegistry** YAML.

IMPORTANT

The default values set for these components are the suggested values. Setting resource requests too high or too low might lead to inefficient resource utilization, or performance degradation, respectively.

5.1. CONFIGURING RESOURCE REQUESTS BY USING THE OPENSHIFT CONTAINER PLATFORM UI

Use the following procedure to configure resources by using the OpenShift Container Platform UI.

Procedure

1. On the OpenShift Container Platform developer console, click **Operators → Installed Operators → Red Hat Quay**.
2. Click **QuayRegistry**.
3. Click the name of your registry, for example, **example-registry**.
4. Click **YAML**.

5. In the **spec.components** field, you can override the resource of the **quay**, **clair**, **mirroring clairpostgres**, and **postgres** resources by setting values for the **.overrides.resources.limits** and the **overrides.resources.requests** fields. For example:

```
spec:
  components:
    - kind: clair
      managed: true
      overrides:
        resources:
          limits:
            cpu: "5"      # Limiting to 5 CPU (equivalent to 5000m or 5000 millicpu)
            memory: "18Gi"  # Limiting to 18 Gibibytes of memory
          requests:
            cpu: "4"      # Requesting 4 CPU
            memory: "4Gi"   # Requesting 4 Gibibytes of memory
    - kind: postgres
      managed: true
      overrides:
        resources:
          limits: {} (1)
          requests:
            cpu: "700m"   # Requesting 700 millicpu or 0.7 CPU
            memory: "4Gi"   # Requesting 4 Gibibytes of memory
    - kind: mirror
      managed: true
      overrides:
        resources:
          limits: (2)
          requests:
            cpu: "800m"   # Requesting 800 millicpu or 0.8 CPU
            memory: "1Gi"   # Requesting 1 Gibibyte of memory
    - kind: quay
      managed: true
      overrides:
        resources:
          limits:
            cpu: "4"     # Limiting to 4 CPU
            memory: "10Gi"  # Limiting to 10 Gibibytes of memory
          requests:
            cpu: "4Gi"   # Requesting 4 CPU
            memory: "10Gi"  # Requesting 10 Gibi of memory
    - kind: clairpostgres
      managed: true
      overrides:
        resources:
          limits:
            cpu: "800m"   # Limiting to 800 millicpu or 0.8 CPU
            memory: "3Gi"   # Limiting to 3 Gibibytes of memory
          requests: {}
```

1. Setting the **limits** or **requests** fields to **{}** uses the default values for these resources.

2. Leaving the **limits** or **requests** field empty puts no limitations on these resources.

6. Optional. After you have overridden the default values, you can use null, denoted by **{}**, to reset back to the default value allocated to the component. For example:

```
# ...
  - kind: clairpostgres
    managed: true
    overrides:
      resources:
        limits: {}
        requests: {}
# ...
```

5.2. CONFIGURING RESOURCE REQUESTS BY EDITING THE QUAYREGISTRY YAML

You can re-configure Red Hat Quay to configure resource requests after you have already deployed a registry. This can be done by editing the **QuayRegistry** YAML file directly and then re-deploying the registry.

Procedure

1. Optional: If you do not have a local copy of the **QuayRegistry** YAML file, enter the following command to obtain it:

```
$ oc get quayregistry <registry_name> -n <namespace> -o yaml > quayregistry.yaml
```

2. Open the **quayregistry.yaml** created from Step 1 of this procedure and make the desired changes. For example:

```
  - kind: quay
    managed: true
    overrides:
      resources:
        limits: {}
        requests:
          cpu: "0.7"   # Requesting 0.7 CPU (equivalent to 500m or 500 millicpu)
          memory: "512Mi"   # Requesting 512 Mebibytes of memory
```

3. Save the changes.

4. Apply the Red Hat Quay registry using the updated configurations by running the following command:

```
$ oc replace -f quayregistry.yaml
```

Example output

```
quayregistry.quay.redhat.com/example-registry replaced
```

CHAPTER 6. CONFIGURING THE DATABASE

6.1. 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:

   ```
   DB_URI: postgresql://test-quay-database:postgres@test-quay-database:5432/test-quay-database
   ```

2. Create a **Secret** using the configuration file:

   ```
   $ 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

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

Next steps

- Continue to the following sections to deploy the registry.

6.1.1. Database configuration

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

6.1.1.1. 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 6.1. Database URI

Field	Type	Description
DB_URI (Required)	String	The URI for accessing the database, including any credentials. Example **DB_URI** field: **postgresql://quayuser:quaypass@quay-server.example.com:5432/quay**

6.1.1.2. 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 6.2. Database connection arguments

Field	Type	Description
DB_CONNECTION_ARGS	Object	Optional connection arguments for the database, such as timeouts and SSL/TLS.
.autorollback	Boolean	Whether to use thread-local connections. Should always be **true**
.threadlocals	Boolean	Whether to use auto-rollback connections. Should always be **true**

6.1.1.2.1. 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:

```
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 6.3. SSL/TLS options

Mode	Description
disable	Your configuration only tries non-SSL/TLS connections.
allow	Your configuration first tries a non-SSL/TLS connection. Upon failure, tries an SSL/TLS connection.
prefer (Default)	Your configuration first tries an SSL/TLS connection. Upon failure, tries a non-SSL/TLS connection.
require	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.
verify-ca	Your configuration only tries an SSL/TLS connection, and verifies that the server certificate is issued by a trusted certificate authority (CA).
verify-full	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.

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

6.1.1.2.2. MySQL SSL/TLS connection arguments

The following example shows a sample MySQL SSL/TLS configuration:

```
DB_CONNECTION_ARGS:
  ssl:
    ca: /path/to/cacert
```

Information on the valid connection arguments for MySQL is available at Connecting to the Server Using URI-Like Strings or Key-Value Pairs.

6.1.2. 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 Upgrading the Red Hat Quay Operator overview.

6.1.2.1. 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.

6.2. CONFIGURING EXTERNAL REDIS

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

6.2.1. 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:

```
# ...
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:

```
$ 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:

```
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

6.2.2. 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 through the Operator or during rescheduling events.

6.2.2.1. Disabling the Horizontal Pod Autoscaler

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

```
apiVersion: quay.redhat.com/v1
```

```
kind: QuayRegistry
metadata:
  name: example-registry
  namespace: quay-enterprise
spec:
  components:
    - kind: horizontalpodautoscaler
      managed: false
# ...
```

6.2.3. 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:

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

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

   ```
   # ...
   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.

6.2.4. 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

```
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.

6.2.5. Disabling the mirroring component

To disable mirroring, use the following YAML configuration:

Unmanaged mirroring example YAML configuration

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

CHAPTER 7. DEPLOYING RED HAT QUAY USING THE OPERATOR

Red Hat Quay on OpenShift Container Platform can be deployed using command-line interface or from the OpenShift Container Platform console. The steps are fundamentally the same.

7.1. 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
    - kind: mirror
      managed: false
    - kind: monitoring
      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

```
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

```
  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
            - name: NO_PROXY
              value:
svc.cluster.local,localhost,quay370.apps.quayperf370.perfscale.devcluster.openshift.com
      - kind: quay
        managed: true
        overrides:
          env:
            - name: DEBUGLOG
              value: "true"
            - name: NO_PROXY
              value:
svc.cluster.local,localhost,quay370.apps.quayperf370.perfscale.devcluster.openshift.com
            - 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 the specified namespace by entering the following command:

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

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

   ```
   $ 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 .

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

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

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

```
$ sudo podman stop quay
```

3. 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}
```

4. 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:

```
{"access_token":"6B4QTRSTSD1HMIG915VPX7BMEZBVB9GPNY2FC2ED",
"email":"quayadmin@example.com","encrypted_password":"1nZMLH57RIE5UGdL/yYpDOHL
qiNCgimb6W9kfF8MjZ1xrfDpRyRs9NUnUuNuAitW","username":"quayadmin"} #
gitleaks:allow
```

If a user already exists in the database, an error is returned:

```
{"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:

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

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

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

Example output

```
Login Succeeded!
```

7.1.2. Viewing created components using the command line

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

Prerequisites

- You have deployed Red Hat Quay on OpenShift Container Platform.

Procedure

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

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

Example output

```
NAME                                                    READY   STATUS      RESTARTS   AGE
example-registry-clair-app-5ffc9f77d6-jwr9s             1/1     Running     0          3m42s
example-registry-clair-app-5ffc9f77d6-wgp7d             1/1     Running     0          3m41s
example-registry-clair-postgres-54956d6d9c-rgs8l        1/1     Running     0          3m5s
example-registry-quay-app-79c6b86c7b-8qnr2              1/1     Running     4          3m42s
example-registry-quay-app-79c6b86c7b-xk85f              1/1     Running     4          3m41s
example-registry-quay-app-upgrade-5kl5r                 0/1     Completed   4          3m50s
example-registry-quay-database-b466fc4d7-tfrnx          1/1     Running     2          3m42s
example-registry-quay-mirror-6d9bd78756-6lj6p           1/1     Running     0          2m58s
example-registry-quay-mirror-6d9bd78756-bv6gq           1/1     Running     0          2m58s
example-registry-quay-postgres-init-dzbmx               0/1     Completed   0          3m43s
example-registry-quay-redis-8bd67b647-skgqx             1/1     Running     0          3m42s
```

7.1.3. 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-***)
- 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 PPod 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:

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

Example output

```
NAME                  REFERENCE                         TARGETS      MINPODS   MAXPODS
REPLICAS   AGE
example-registry-clair-app    Deployment/example-registry-clair-app    16%/90%, 0%/90%   2
10      2      13d
example-registry-quay-app     Deployment/example-registry-quay-app     31%/90%, 1%/90%   2
20      2      13d
example-registry-quay-mirror  Deployment/example-registry-quay-mirror  27%/90%, 0%/90%   2
20      2      13d
```

Additional resources

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

7.1.4. 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 an help you debug any problems that may arise.

Procedure

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

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

Example output

Immediately after deployment, the **QuayRegistry** object will show the basic configuration:

```
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
    uid: e3fc82ba-e716-4646-bb0f-63c26d05e00e
  spec:
    components:
    - 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
    configBundleSecret: example-registry-config-bundle-kt55s
```

```
kind: List
metadata:
  resourceVersion: ""
  selfLink: ""
```

2. Use the **oc get pods** command to view the current state of the deployed components:

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

Example output

```
NAME                                                  READY   STATUS              RESTARTS   AGE
example-registry-clair-app-86554c6b49-ds7bl           0/1     ContainerCreating   0          2s
example-registry-clair-app-86554c6b49-hxp5s           0/1     Running             1          17s
example-registry-clair-postgres-68d8857899-lbc5n      0/1     ContainerCreating   0          17s
example-registry-quay-app-upgrade-h2v7h               0/1     ContainerCreating   0          9s
example-registry-quay-database-66f495c9bc-wqsjf       0/1     ContainerCreating   0          17s
example-registry-quay-mirror-854c88457b-d845g         0/1     Init:0/1            0          2s
example-registry-quay-mirror-854c88457b-fghxv         0/1     Init:0/1            0          17s
example-registry-quay-postgres-init-bktdt             0/1     Terminating         0          17s
example-registry-quay-redis-f9b9d44bf-4htpz           0/1     ContainerCreating   0          17s
```

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:

```
  status:
    conditions:
    - lastTransitionTime: "2021-09-14T10:52:04Z"
      lastUpdateTime: "2021-09-14T10:52:04Z"
      message: all objects created/updated successfully
      reason: ComponentsCreationSuccess
      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
    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:

```
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
  currentVersion: {producty}
  lastUpdated: 2021-09-14 10:52:46.104181633 +0000 UTC
  registryEndpoint: https://example-registry-quay-quay-enterprise.apps.docs.quayteam.org
  unhealthyComponents: {}
```

7.2. DEPLOYING RED HAT QUAY FROM THE OPENSHIFT 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.

7.2.1. 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.

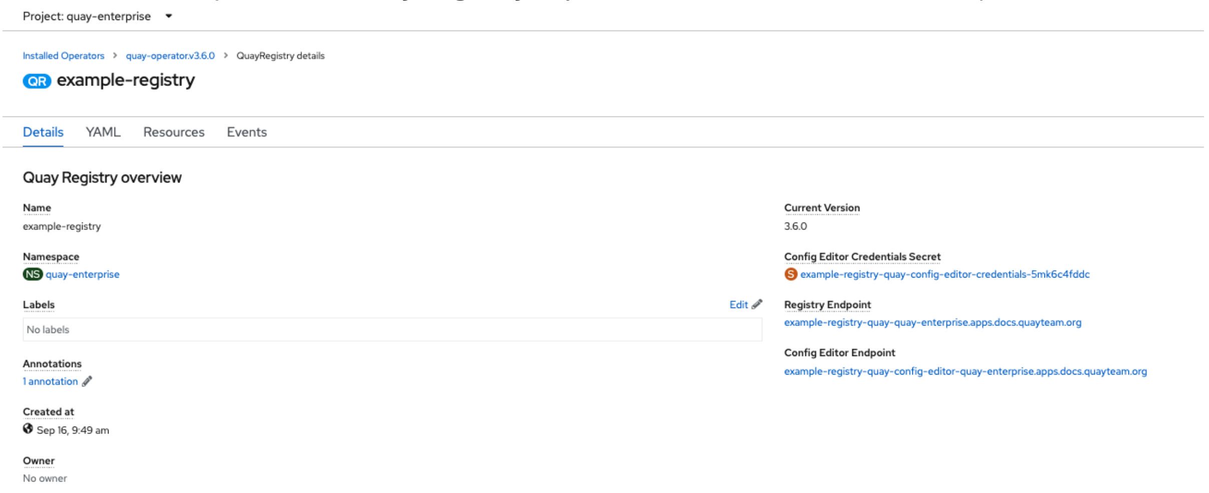
NOTE

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 **QuayRegistry** object to view the details. For example:



Quay Registry overview

Name
example-registry

Namespace
NS quay-enterprise

Labels
No labels

Annotations
1 annotation

Created at
Sep 16, 9:49 am

Owner
No owner

Current Version
3.6.0

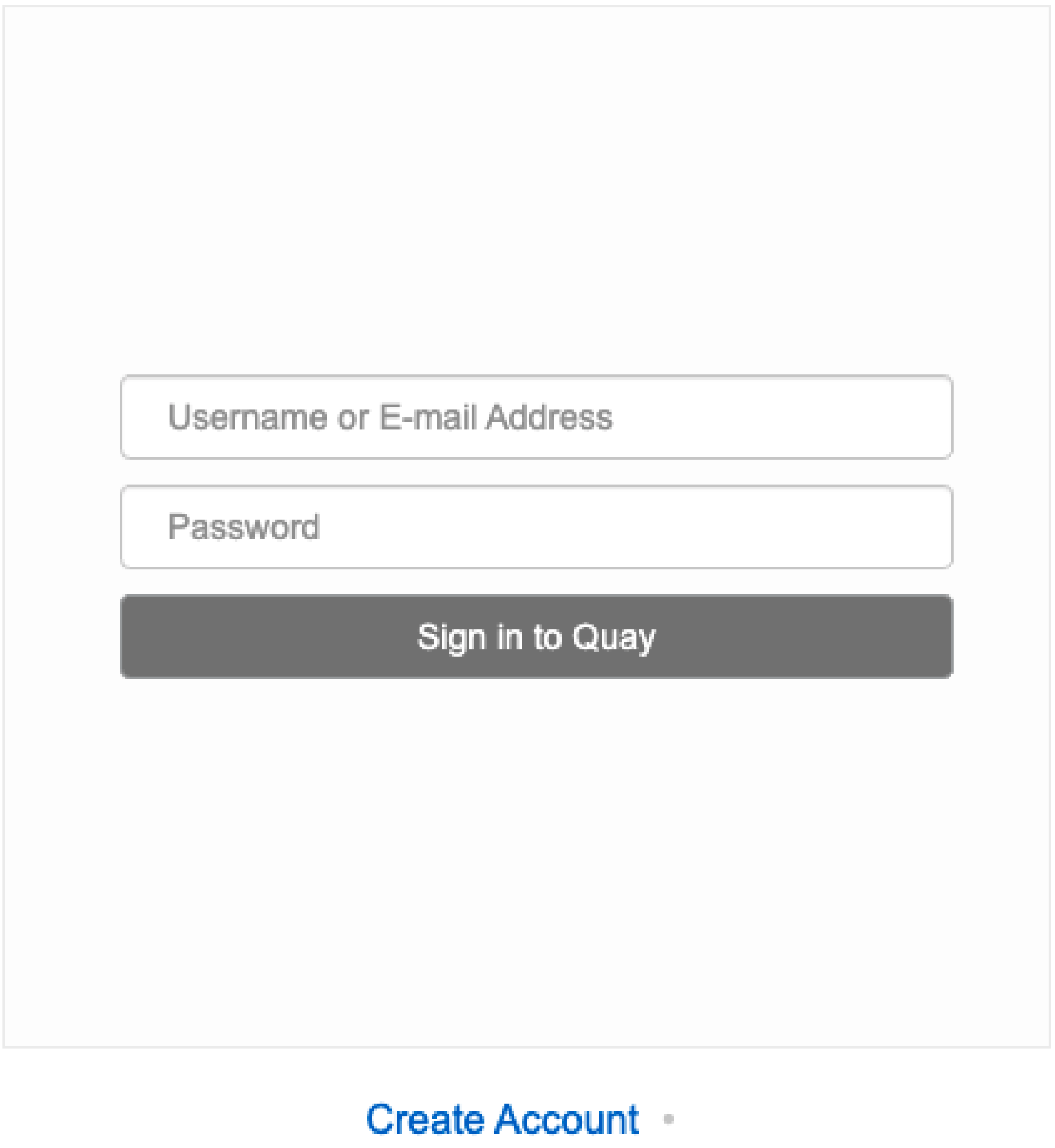
Config Editor Credentials Secret
S example-registry-quay-config-editor-credentials-5mk6c4fddc

Registry Endpoint
example-registry-quay-quay-enterprise.apps.docs.quayteam.org

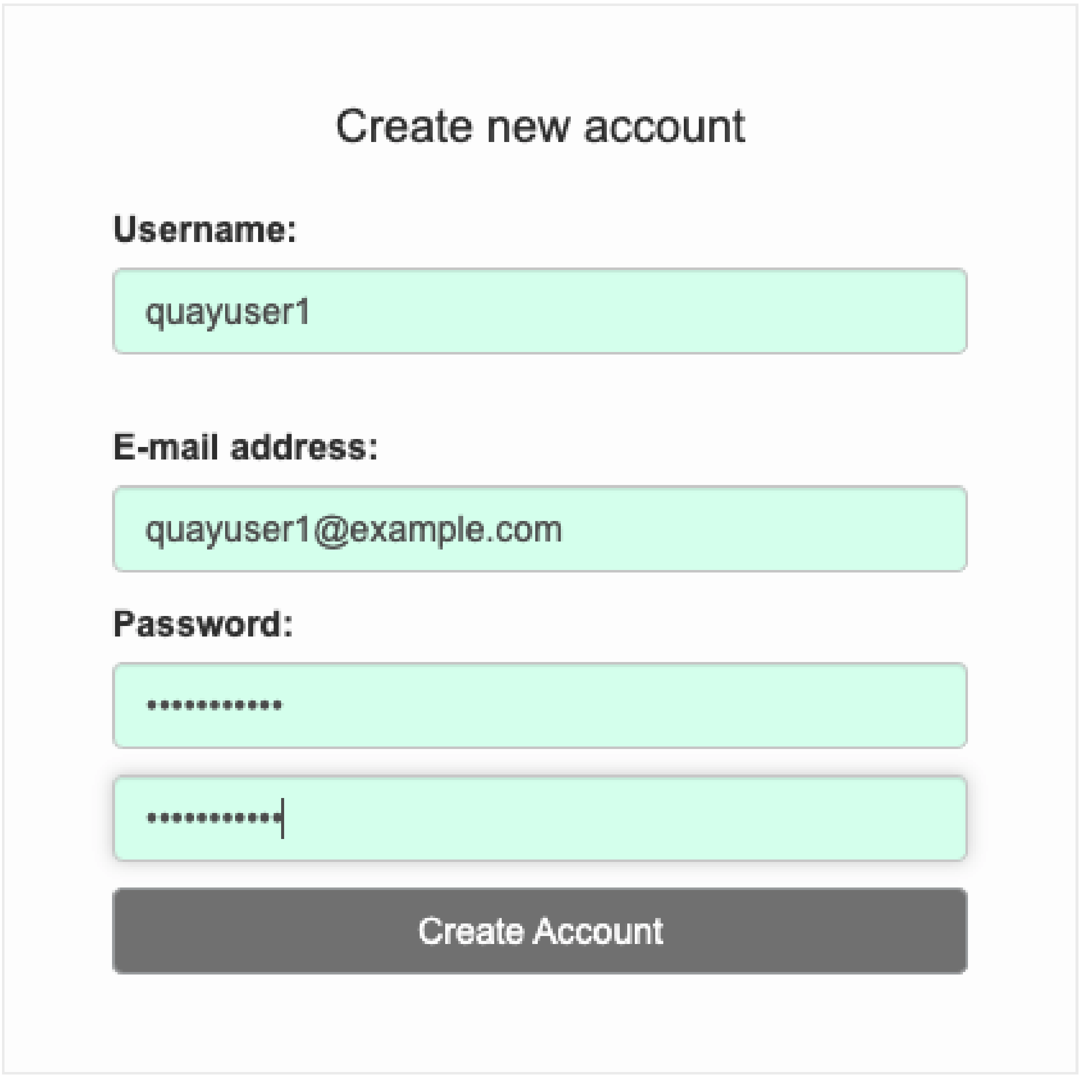
Config Editor Endpoint
example-registry-quay-config-editor-quay-enterprise.apps.docs.quayteam.org

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:



5. Enter the details for **Username**, **Password**, **Email**, and then click **Create Account**. For example:



Create new account

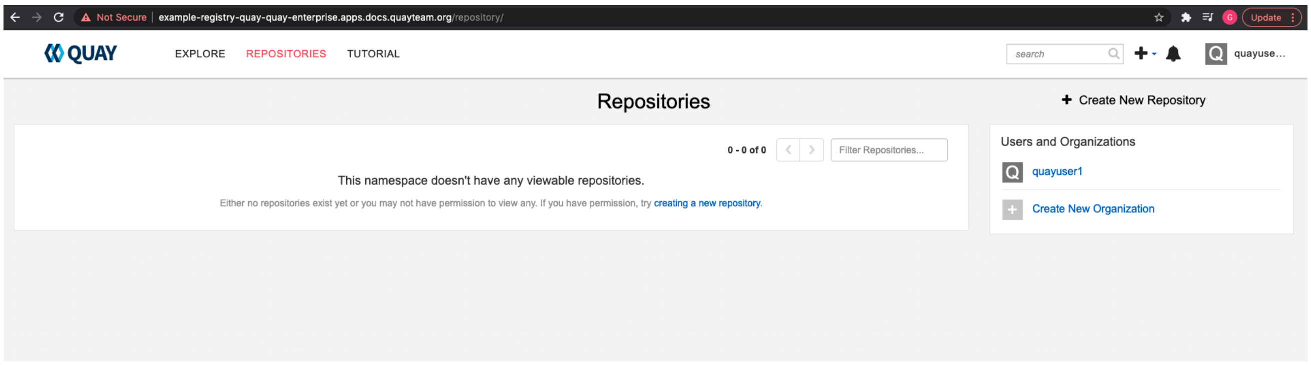
Username: quayuser1

E-mail address: quayuser1@example.com

Password: ●●●●●●●●●●●

●●●●●●●●●●●

After creating the first user, you are automatically logged in to the Red Hat Quay registry. For example:



Repositories

0 - 0 of 0

This namespace doesn't have any viewable repositories.

Either no repositories exist yet or you may not have permission to view any. If you have permission, try creating a new repository.

Users and Organizations

quayuser1

CHAPTER 8. 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.

8.1. 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.

8.2. 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**.

8.3. VIEWING THE CONDITIONS OF YOUR RED HAT QUAY DEPLOYMENT

Certain conditions will be reported in **status.conditions**.

CHAPTER 9. CUSTOMIZING RED HAT QUAY ON OPENSHIFT CONTAINER PLATFORM

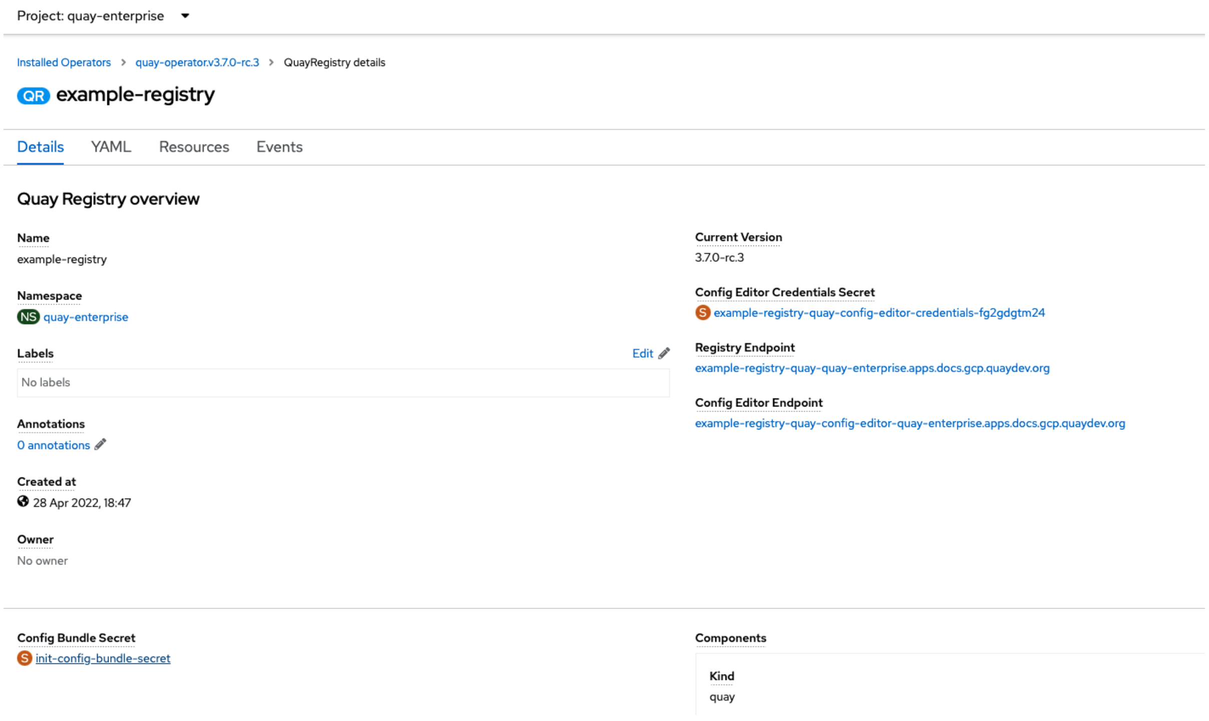
After deployment, you can customize the Red Hat Quay application by editing the Red Hat Quay configuration bundle secret **spec.configBundleSecret**. You can also change the managed status of components and configure resource requests for some components in the **spec.components** object of the **QuayRegistry** resource.

9.1. EDITING THE CONFIG BUNDLE SECRET IN THE OPENSHIFT CONTAINER PLATFORM CONSOLE

Use the following procedure to edit the config bundle secret in the OpenShift Container Platform console.

Procedure

1. On the Red Hat Quay Registry overview screen, click the link for the **Config Bundle Secret**



Project: quay-enterprise

Installed Operators > quay-operator.v3.7.0-rc.3 > QuayRegistry details

QR example-registry

Details YAML Resources Events

Quay Registry overview

Name
example-registry

Namespace
NS quay-enterprise

Labels Edit
No labels

Annotations
0 annotations

Created at
28 Apr 2022, 18:47

Owner
No owner

Current Version
3.7.0-rc.3

Config Editor Credentials Secret
S example-registry-quay-config-editor-credentials-fg2gdgtm24

Registry Endpoint
example-registry-quay-quay-enterprise.apps.docs.gcp.quaydev.org

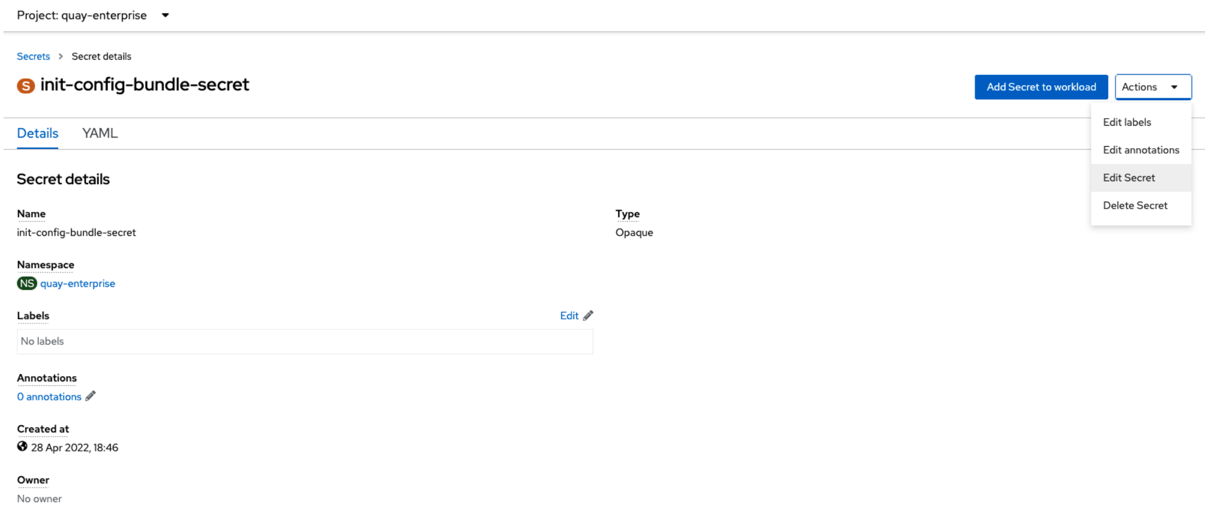
Config Editor Endpoint
example-registry-quay-config-editor-quay-enterprise.apps.docs.gcp.quaydev.org

Config Bundle Secret
S init-config-bundle-secret

Components

Kind
quay

2. To edit the secret, click **Actions → Edit Secret**.



Project: quay-enterprise

Secrets > Secret details

S init-config-bundle-secret

Details YAML

Secret details

Name
init-config-bundle-secret

Type
Opaque

Namespace
NS quay-enterprise

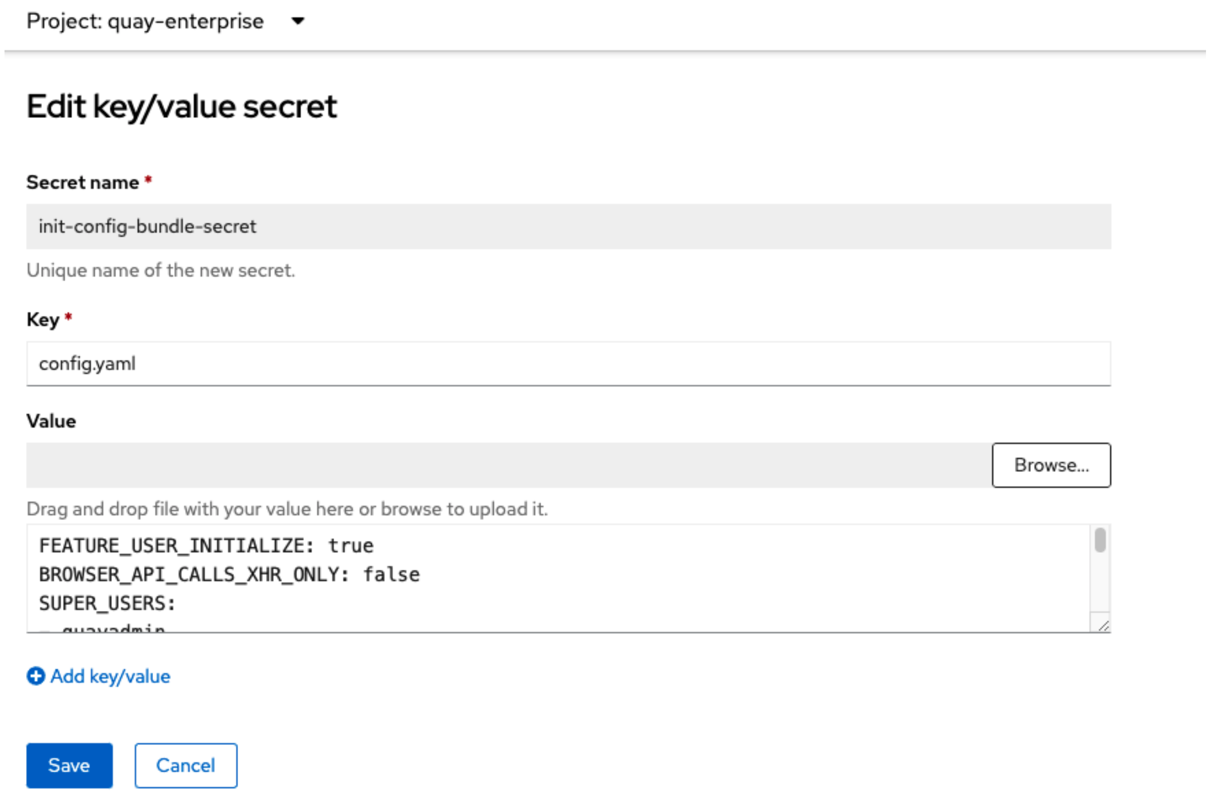
Labels Edit
No labels

Annotations
0 annotations

Created at
28 Apr 2022, 18:46

Owner
No owner

3. Modify the configuration and save the changes.



init-config-bundle-secret

config.yaml

```
FEATURE_USER_INITIALIZE: true
BROWSER_API_CALLS_XHR_ONLY: false
SUPER_USERS:
```

4. Monitor the deployment to ensure successful completion and that the configuration changes have taken effect.

9.2. DETERMINING QUAYREGISTRY ENDPOINTS AND SECRETS

Use the following procedure to find **QuayRegistry** endpoints and secrets.

Procedure

1. You can examine the **QuayRegistry** resource, using **oc describe quayregistry** or **oc get quayregistry -o yaml**, to find the current endpoints and secrets by entering the following command:

```
$ oc get quayregistry example-registry -n quay-enterprise -o yaml
```

Example output

```
apiVersion: quay.redhat.com/v1
kind: QuayRegistry
metadata:
  ...
  name: example-registry
  namespace: quay-enterprise
  ...
spec:
  components:
  - kind: quay
    managed: true
  ...
  - kind: clairpostgres
    managed: true
  configBundleSecret: init-config-bundle-secret (1)
status:
  currentVersion: 3.7.0
  lastUpdated: 2022-05-11 13:28:38.199476938 +0000 UTC
  registryEndpoint: https://example-registry-quay-quay-enterprise.apps.docs.gcp.quaydev.org
(2)
```

(1) The config bundle secret, containing the **config.yaml** file and any SSL/TLS certificates.

(2) The URL for your registry, for browser access to the registry UI, and for the registry API endpoint.

9.3. DOWNLOADING THE EXISTING CONFIGURATION

The following procedures detail how to download the existing configuration using different strategies.

9.3.1. Using the config bundle secret to download the existing configuration

You can use the config bundle secret to download the existing configuration.

Procedure

1. Obtain the secret data by entering the following command:

```
$ oc get secret -n quay-enterprise init-config-bundle-secret -o jsonpath='{.data}'
```

Example output

```
{
    "config.yaml": "RkVBVFVSRV9VU0 ... MDAwMAo="
}
```

2. Enter the following command to decode the data:

```
$ echo 'RkVBVFVSRV9VU0 ... MDAwMAo=' | base64 --decode
```

Example output

```
FEATURE_USER_INITIALIZE: true
BROWSER_API_CALLS_XHR_ONLY: false
SUPER_USERS:
- quayadmin
FEATURE_USER_CREATION: false
FEATURE_QUOTA_MANAGEMENT: true
FEATURE_PROXY_CACHE: true
FEATURE_BUILD_SUPPORT: true
DEFAULT_SYSTEM_REJECT_QUOTA_BYTES: 102400000
```

9.4. USING THE CONFIG BUNDLE TO CONFIGURE CUSTOM SSL/TLS CERTS

You can configure custom SSL/TLS certificates before the initial deployment, or after Red Hat Quay is deployed on OpenShift Container Platform. This is done by creating or updating the config bundle secret.

If you are adding the certificates to an existing deployment, you must include the existing **config.yaml** file in the new config bundle secret, even if you are not making any configuration changes.

Use the following procedure to add custom SSL/TLS certificates.

Procedure

1. In your **QuayRegistry** YAML file, set **kind: tls** to **managed:false**, for example:

```
- kind: tls
  managed: false
```

2. Navigate to the **Events** page, which should reveal that the change is blocked until you set up the appropriate config. For example:

```
- lastTransitionTime: '2022-03-28T12:56:49Z'
  lastUpdateTime: '2022-03-28T12:56:49Z'
  message: >-
    required component `tls` marked as unmanaged, but `configBundleSecret`
    is missing necessary fields
  reason: ConfigInvalid
  status: 'True'
```

3. Create the secret using embedded data or by using files.

   a. Embed the configuration details directly in the **Secret** resource YAML file. For example:

   custom-ssl-config-bundle.yaml

```
apiVersion: v1
kind: Secret
metadata:
  name: custom-ssl-config-bundle-secret
  namespace: quay-enterprise
data:
  config.yaml: |
    FEATURE_USER_INITIALIZE: true
    BROWSER_API_CALLS_XHR_ONLY: false
    SUPER_USERS:
    - quayadmin
    FEATURE_USER_CREATION: false
    FEATURE_QUOTA_MANAGEMENT: true
    FEATURE_PROXY_CACHE: true
    FEATURE_BUILD_SUPPORT: true
    DEFAULT_SYSTEM_REJECT_QUOTA_BYTES: 102400000
  extra_ca_cert_my-custom-ssl.crt: |
    -----BEGIN CERTIFICATE-----
    MIIDsDCCApigAwIBAgIUCqlzkHjF5i5TXLFy+sepFrZr/UswDQYJKoZIhvcNAQEL
```

```
BQAwbzELMAkGA1UEBhMCSUUxDzANBgNVBAgMBkdBTFdBWTEPMA0GA1UEBwwG
R0FM
    ....
    -----END CERTIFICATE-----
```

b. Create the secret from the YAML file:

```
$ oc create  -f custom-ssl-config-bundle.yaml
```

..

4. Alternatively, you can create files containing the desired information, and then create the secret from those files.

   a. Enter the following command to create a generic **Secret** object that contains the **config.yaml** file and a **custom-ssl.crt** file:

   ```
   $ oc create secret generic custom-ssl-config-bundle-secret \
     --from-file=config.yaml \
     --from-file=extra_ca_cert_my-custom-ssl.crt=my-custom-ssl.crt
   ```

   b. Create or update the **QuayRegistry** YAML file, referencing the created **Secret**, for example:

   Example QuayRegistry YAML file

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

   c. Deploy or update the registry using the YAML file by entering the following command:

   ```
   $ oc apply -f quayregistry.yaml
   ```

Next steps

- Red Hat Quay features