



# Red Hat Quay 3.13

## Securing Red Hat Quay

Securing Red Hat Quay





## 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<sup>®</sup> is the registered trademark of Linus Torvalds in the United States and other countries.

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

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

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

Node.js<sup>®</sup> 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<sup>®</sup> 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

Securing Red Hat Quay: SSL/TLS, Certificates, and Encryption

---

## Table of Contents

<b>PREFACE</b> .....	<b>3</b>
<b>CHAPTER 1. SSL AND TLS FOR RED HAT QUAY</b> .....	<b>4</b>
1.1. CREATING A CERTIFICATE AUTHORITY	4
1.2. CONFIGURING SSL/TLS FOR STANDALONE RED HAT QUAY DEPLOYMENTS	5
1.2.1. Configuring custom SSL/TLS certificates by using the command line interface	5
1.2.2. Configuring Podman to trust the Certificate Authority	6
1.2.3. Configuring the system to trust the certificate authority	7
1.3. CONFIGURING CUSTOM SSL/TLS CERTIFICATES FOR RED HAT QUAY ON OPENSIFT CONTAINER PLATFORM	8
1.3.1. Creating a custom SSL/TLS configBundleSecret resource	8
<b>CHAPTER 2. CERTIFICATE-BASED AUTHENTICATION BETWEEN RED HAT QUAY AND SQL</b> .....	<b>12</b>
2.1. CONFIGURING CERTIFICATE-BASED AUTHENTICATION WITH SQL	12
<b>CHAPTER 3. ADDING ADDITIONAL CERTIFICATE AUTHORITIES FOR RED HAT QUAY</b> .....	<b>15</b>
3.1. ADDING ADDITIONAL CERTIFICATE AUTHORITIES TO THE RED HAT QUAY CONTAINER	15
3.2. ADDING ADDITIONAL CERTIFICATE AUTHORITIES TO RED HAT QUAY ON OPENSIFT CONTAINER PLATFORM	16
3.2.1. Downloading the existing configuration	16
3.2.2. Adding additional Certificate Authorities to Red Hat Quay on OpenShift Container Platform	17
3.3. ADDING CUSTOM SSL/TLS CERTIFICATES WHEN RED HAT QUAY IS DEPLOYED ON KUBERNETES	19



## PREFACE

Red Hat Quay offers administrators the ability to secure communication and trusted access to their repositories through the use of Transport Layer Security (TLS), certificate management, and encryption techniques. Properly configuring SSL/TLS and implementing custom certificates can help safeguard data, secure external connections, and maintain trust between Red Hat Quay and the integrated services of your choosing.

The following topics are covered:

- Configuring custom SSL/TLS certificates for standalone Red Hat Quay deployments
- Configuring custom SSL/TLS certificates for Red Hat Quay on OpenShift Container Platform
- Adding additional Certificate Authorities to the Red Hat Quay container
- Adding additional Certificate Authorities to Red Hat Quay on OpenShift Container Platform

# CHAPTER 1. SSL AND TLS FOR RED HAT QUAY

The Secure Sockets Layer (SSL) protocol was originally developed by Netscape Corporation to provide a mechanism for secure communication over the Internet. Subsequently, the protocol was adopted by the Internet Engineering Task Force (IETF) and renamed to Transport Layer Security (TLS).

TLS (Transport Layer Security) is a cryptographic protocol used to secure network communications. When hardening system security settings by configuring preferred key-exchange protocols, authentication methods, and encryption algorithms, it is necessary to bear in mind that the broader the range of supported clients, the lower the resulting security. Conversely, strict security settings lead to limited compatibility with clients, which can result in some users being locked out of the system. Be sure to target the strictest available configuration and only relax it when it is required for compatibility reasons.

Red Hat Quay can be configured to use SSL/TLS certificates to ensure secure communication between clients and the Red Hat Quay server. This configuration involves the use of valid SSL/TLS certificates, which can be obtained from a trusted Certificate Authority (CA) or generated as self-signed certificates for internal use.

## 1.1. CREATING A CERTIFICATE AUTHORITY

Use the following procedure to set up your own CA and use it to issue a server certificate for your domain. This allows you to secure communications with SSL/TLS using your own certificates.

### Procedure

1. Generate the root CA key by entering the following command:

```
$ openssl genrsa -out rootCA.key 2048
```

2. Generate the root CA certificate by entering the following command:

```
$ openssl req -x509 -new -nodes -key rootCA.key -sha256 -days 1024 -out rootCA.pem
```

3. Enter the information that will be incorporated into your certificate request, including the server hostname, for example:

```
Country Name (2 letter code) [XX]:IE
State or Province Name (full name) []:GALWAY
Locality Name (eg, city) [Default City]:GALWAY
Organization Name (eg, company) [Default Company Ltd]:QUAY
Organizational Unit Name (eg, section) []:DOCS
Common Name (eg, your name or your server's hostname) []:quay-server.example.com
```

4. Generate the server key by entering the following command:

```
$ openssl genrsa -out ssl.key 2048
```

5. Generate a signing request by entering the following command:

```
$ openssl req -new -key ssl.key -out ssl.csr
```



6. Enter the information that will be incorporated into your certificate request, including the server hostname, for example:

```
Country Name (2 letter code) [XX]:IE
State or Province Name (full name) []:GALWAY
Locality Name (eg, city) [Default City]:GALWAY
Organization Name (eg, company) [Default Company Ltd]:QUAY
Organizational Unit Name (eg, section) []:DOCS
Common Name (eg, your name or your server's hostname) []:quay-server.example.com
Email Address []:
```

7. Create a configuration file **openssl.cnf**, specifying the server hostname, for example:

#### Example openssl.cnf file

```
[req]
req_extensions = v3_req
distinguished_name = req_distinguished_name
[req_distinguished_name]
[ v3_req ]
basicConstraints = CA:FALSE
keyUsage = nonRepudiation, digitalSignature, keyEncipherment
subjectAltName = @alt_names
[alt_names]
DNS.1 = <quay-server.example.com>
IP.1 = 192.168.1.112
```

8. Use the configuration file to generate the certificate **ssl.cert**:

```
$ openssl x509 -req -in ssl.csr -CA rootCA.pem -CAkey rootCA.key -CAcreateserial -out
ssl.cert -days 356 -extensions v3_req -extfile openssl.cnf
```

9. Confirm your created certificates and files by entering the following command:

```
$ ls /path/to/certificates
```

#### Example output

```
rootCA.key ssl-bundle.cert ssl.key custom-ssl-config-bundle-secret.yaml rootCA.pem ssl.cert
openssl.cnf rootCA.srl ssl.csr
```

## 1.2. CONFIGURING SSL/TLS FOR STANDALONE RED HAT QUAY DEPLOYMENTS

For standalone Red Hat Quay deployments, SSL/TLS certificates must be configured by using the command-line interface and by updating your **config.yaml** file manually.

### 1.2.1. Configuring custom SSL/TLS certificates by using the command line interface

SSL/TLS must be configured by using the command-line interface (CLI) and updating your **config.yaml** file manually.

## Prerequisites

- You have created a certificate authority and signed the certificate.

## Procedure

1. Copy the certificate file and primary key file to your configuration directory, ensuring they are named **ssl.cert** and **ssl.key** respectively:

```
cp ~/ssl.cert ~/ssl.key /path/to/configuration_directory
```

2. Navigate to the configuration directory by entering the following command:

```
$ cd /path/to/configuration_directory
```

3. Edit the **config.yaml** file and specify that you want Red Hat Quay to handle SSL/TLS:

### Example config.yaml file

```
# ...
SERVER_HOSTNAME: <quay-server.example.com>
...
PREFERRED_URL_SCHEME: https
# ...
```

4. Optional: Append the contents of the **rootCA.pem** file to the end of the **ssl.cert** file by entering the following command:

```
$ cat rootCA.pem >> ssl.cert
```

5. Stop the **Quay** container by entering the following command:

```
$ sudo podman stop <quay_container_name>
```

6. Restart the registry by entering the following command:

```
$ sudo podman run -d --rm -p 80:8080 -p 443:8443 \
  --name=quay \
  -v $QUAY/config:/conf/stack:Z \
  -v $QUAY/storage:/datastorage:Z \
  registry.redhat.io/quay/quay-rhel8:v3.13.1
```

## 1.2.2. Configuring Podman to trust the Certificate Authority

Podman uses two paths to locate the Certificate Authority (CA) file: **/etc/containers/certs.d/** and **/etc/docker/certs.d/**. Use the following procedure to configure Podman to trust the CA.

## Procedure

1. Copy the root CA file to one of **/etc/containers/certs.d/** or **/etc/docker/certs.d/**. Use the exact path determined by the server hostname, and name the file **ca.crt**:

```
$ sudo cp rootCA.pem /etc/containers/certs.d/quay-server.example.com/ca.crt
```

- 2. Verify that you no longer need to use the **--tls-verify=false** option when logging in to your Red Hat Quay registry:

```
$ sudo podman login quay-server.example.com
```

### Example output

```
Login Succeeded!
```

## 1.2.3. Configuring the system to trust the certificate authority

Use the following procedure to configure your system to trust the certificate authority.

### Procedure

1. Enter the following command to copy the **rootCA.pem** file to the consolidated system-wide trust store:

```
$ sudo cp rootCA.pem /etc/pki/ca-trust/source/anchors/
```

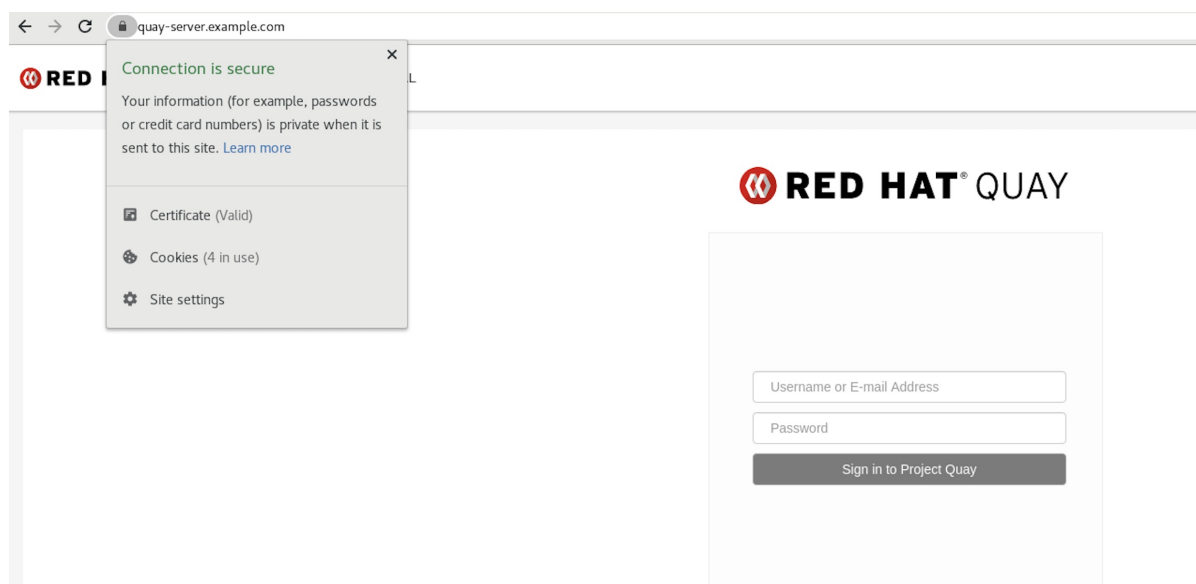
2. Enter the following command to update the system-wide trust store configuration:

```
$ sudo update-ca-trust extract
```

3. Optional. You can use the **trust list** command to ensure that the **Quay** server has been configured:

```
$ trust list | grep quay
label: quay-server.example.com
```

Now, when you browse to the registry at <https://quay-server.example.com>, the lock icon shows that the connection is secure:



4. To remove the **rootCA.pem** file from system-wide trust, delete the file and update the configuration:

```
$ sudo rm /etc/pki/ca-trust/source/anchors/rootCA.pem
```

```
$ sudo update-ca-trust extract
```

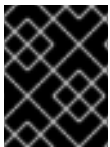
```
$ trust list | grep quay
```

More information can be found in the RHEL 9 documentation in the chapter [Using shared system certificates](#).

## 1.3. CONFIGURING CUSTOM SSL/TLS CERTIFICATES FOR RED HAT QUAY ON OPENSIFT CONTAINER PLATFORM

When Red Hat Quay is deployed on OpenShift Container Platform, the **tls** component of the **QuayRegistry** custom resource definition (CRD) is set to **managed** by default. As a result, OpenShift Container Platform's Certificate Authority is used to create HTTPS endpoints and to rotate SSL/TLS certificates.

You can configure custom SSL/TLS certificates before or after the initial deployment of Red Hat Quay on OpenShift Container Platform. This process involves creating or updating the **configBundleSecret** resource within the **QuayRegistry** YAML file to integrate your custom certificates and setting the **tls** component to **unmanaged**.



### IMPORTANT

When configuring custom SSL/TLS certificates for Red Hat Quay, administrators are responsible for certificate rotation.

The following procedures enable you to apply custom SSL/TLS certificates to ensure secure communication and meet specific security requirements for your Red Hat Quay on OpenShift Container Platform deployment. These steps assumed you have already created a Certificate Authority (CA) bundle or an **ssl.key**, and an **ssl.cert**. The procedure then shows you how to integrate those files into your Red Hat Quay on OpenShift Container Platform deployment, which ensures that your registry operates with the specified security settings and conforms to your organization's SSL/TLS policies.



### NOTE

- The following procedure is used for securing Red Hat Quay with an HTTPS certificate. Note that this differs from managing Certificate Authority Trust Bundles. CA Trust Bundles are used by system processes within the **Quay** container to verify certificates against trusted CAs, and ensure that services like LDAP, storage backend, and OIDC connections are trusted.
- 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.

### 1.3.1. Creating a custom SSL/TLS configBundleSecret resource

After creating your custom SSL/TLS certificates, you can create a custom **configBundleSecret** resource for Red Hat Quay on OpenShift Container Platform, which allows you to upload **ssl.cert** and **ssl.key** files.

## Prerequisites

- You have base64 decoded the original config bundle into a **config.yaml** file. For more information, see [Downloading the existing configuration](#).
- You have generated custom SSL certificates and keys.

## Procedure

1. Create a new YAML file, for example, **custom-ssl-config-bundle-secret.yaml**:

```
$ touch custom-ssl-config-bundle-secret.yaml
```

2. Create the **custom-ssl-config-bundle-secret** resource.
  - a. Create the resource by entering the following command:

```
$ oc -n <namespace> create secret generic custom-ssl-config-bundle-secret \
  --from-file=config.yaml=</path/to/config.yaml> \ 1
  --from-file=ssl.cert=</path/to/ssl.cert> \ 2
  --from-file=extra_ca_cert_<name-of-certificate>.cert=ca-certificate-bundle.crt
  \ 3
  --from-file=ssl.key=</path/to/ssl.key> \ 4
  --dry-run=client -o yaml > custom-ssl-config-bundle-secret.yaml
```

**1** Where **<config.yaml>** is your **base64 decoded config.yaml** file.

**2** Where **<ssl.cert>** is your **ssl.cert** file.

**3** Optional. The **--from-file=extra\_ca\_cert\_<name-of-certificate>.cert=ca-certificate-bundle.crt** field allows Red Hat Quay to recognize custom Certificate Authority (CA) files. If you are using LDAP, OIDC, or another service that uses custom CAs, you must add them via the **extra\_ca\_cert** path. For more information, see "Adding additional Certificate Authorities to Red Hat Quay on OpenShift Container Platform."

**4** Where **<ssl.key>** is your **ssl.key** file.

3. Optional. You can check the content of the **custom-ssl-config-bundle-secret.yaml** file by entering the following command:

```
$ cat custom-ssl-config-bundle-secret.yaml
```

## Example output

```
apiVersion: v1
data:
  config.yaml:
    QUxMT1dfUFVMTFNFV0IUSE9VVF9TVFJJQ1RfTE9HR0IORzogZmFsc2UKQVVUSEVOVEID
    QVRJT05fVFIQRTOGRGF0YWJhc2UKREVGQVVMVF9UQUdfRvhQSVJBVEIPTjogMncKREI
    TVFJJQIVURURfU1R...
  ssl.cert:
    LS0tLS1CRUdJTiBDRVJUSUZJQ0FURSB0tLS0tCk1JSUVYakNDQTBhZ0F3SUJBZ0lVTUFBR
    k1YVWIWVHN0MGxNTWI3U1I0eFV5eTJjd0RRWUplb1pJaHZjTkFRRUwKQIFBd2dZZ3hDek
    FKQmdOVkKJBWVR...
```

```

extra_ca_cert_<name-of-
certificate>:LS0tLS1CRUdJTiBDRVJUSUZJQ0FURSU0tLS0tCk1JSUVYakNDQTBhZ0F3SUJB
Z0lVTUFBRk1YVWlWVHN0MGxNTWI3U1I0eFV5eTJjd0RRWUplb1pJaHZjTkFRRUwKQlFBd
2dZZ3hDe...
ssl.key:
LS0tLS1CRUdJTiBQUklWQVRFIEtFWS0tLS0tCk1JSUV2UUlCQURBTkNa3Foa2lHOXcwQk
FRRUZXBQVNDQktjd2dnU2pBZ0VBQW9JQkFRQ2c0VWxzOVV1SVJPY1oKcFhpZk9MVEEdqa
S9neUxQMlpiMXQ...
kind: Secret
metadata:
  creationTimestamp: null
  name: custom-ssl-config-bundle-secret
  namespace: <namespace>

```

4. Create the **configBundleSecret** resource by entering the following command:

```
$ oc create -n <namespace> -f custom-ssl-config-bundle-secret.yaml
```

#### Example output

```
secret/custom-ssl-config-bundle-secret created
```

5. Update the **QuayRegistry** YAML file to reference the **custom-ssl-config-bundle-secret** object by entering the following command:

```
$ oc patch quayregistry <registry_name> -n <namespace> --type=merge -p '{"spec":
{"configBundleSecret":"custom-ssl-config-bundle-secret"}}'
```

#### Example output

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

6. Set the **tls** component of the **QuayRegistry** YAML to **false** by entering the following command:

```
$ oc patch quayregistry <registry_name> -n <namespace> --type=merge -p '{"spec":
{"components":[{"kind":"tls","managed":false}]}'
```

#### Example output

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

7. Ensure that your **QuayRegistry** YAML file has been updated to use the custom SSL **configBundleSecret** resource, and that your **tls** resource is set to **false** by entering the following command:

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

#### Example output

```
# ...
configBundleSecret: custom-ssl-config-bundle-secret
# ...
```

```
spec:  
  components:  
    - kind: tls  
      managed: false  
# ...
```

## Verification

- Confirm a TLS connection to the server and port by entering the following command:

```
$ openssl s_client -connect <quay-server.example.com>:443
```

## Example output

```
# ...  
SSL-Session:  
  Protocol : TLSv1.3  
  Cipher   : TLS_AES_256_GCM_SHA384  
  Session-ID:  
0E995850DC3A8EB1A838E2FF06CE56DBA81BD8443E7FA05895FBD6FBDE9FE737  
  Session-ID-ctx:  
  Resumption PSK:  
1EA68F33C65A0F0FA2655BF9C1FE906152C6E3FEEE3AEB6B1B99BA7C41F06077989352  
C58E07CD2FBDC363FA8A542975  
  PSK identity: None  
  PSK identity hint: None  
  SRP username: None  
  TLS session ticket lifetime hint: 7200 (seconds)  
  
# ...
```

## CHAPTER 2. CERTIFICATE-BASED AUTHENTICATION BETWEEN RED HAT QUAY AND SQL

Red Hat Quay administrators can configure certificate-based authentication between Red Hat Quay and SQL (PostgreSQL and GCP CloudSQL) by supplying their own SSL/TLS certificates for client-side authentication. This provides enhanced security and allows for easier automation for your Red Hat Quay registry.

The following sections shows you how to configure certificate-based authentication between Red Hat Quay and PostgreSQL, and Red Hat Quay and CloudSQL.

### 2.1. CONFIGURING CERTIFICATE-BASED AUTHENTICATION WITH SQL

The following procedure demonstrates how to connect Red Hat Quay with an SQL database using secure client-side certificates. This method ensures both connectivity and authentication through Certificate Trust Verification, as it verifies the SQL server's certificate against a trusted Certificate Authority (CA). This enhances the security of the connection between Red Hat Quay and your SQL server while simplifying automation for your deployment. Although the example uses Google Cloud Platform's CloudSQL, the procedure also applies to PostgreSQL and other supported databases.

#### Prerequisites

- You have generated custom Certificate Authorities (CAs) and your SSL/TLS certificates and keys are available in **PEM** format that will be used to generate an SSL connection with your CloudSQL database. For more information, see [SSL and TLS for Red Hat Quay](#).
- You have **base64 decoded** the original config bundle into a **config.yaml** file. For more information, see [Downloading the existing configuration](#).
- You are using an externally managed PostgreSQL or CloudSQL database. For more information, see [Using and existing PostgreSQL database](#) with the **DB\_URI** variable set.
- Your externally managed PostgreSQL or CloudSQL database is configured for SSL/TLS.
- The **postgres** component of your **QuayRegistry** CRD is set to **managed: false**, and your CloudSQL database is set with the **DB\_URI** configuration variable. The following procedure uses **postgresql://<cloudsql\_username>:<dbpassword>@<database\_host>:<port>/<database\_name>**.

#### Procedure

1. After you have generated the CAs and SSL/TLS certificates and keys for your CloudSQL database and ensured that they are in **.pem** format, test the SSL connection to your CloudSQL server:
  - a. Initiate a connection to your CloudSQL server by entering the following command:

```
$ psql "sslmode=verify-ca sslrootcert=<ssl_server_certificate_authority>.pem sslcert=<ssl_client_certificate>.pem sslkey=<ssl_client_key>.pem hostaddr=<database_host> port=<5432> user=<cloudsql_username> dbname=<cloudsql_database_name>"
```

2. In your Red Hat Quay directory, create a new YAML file, for example, **quay-config-bundle.yaml**, by running the following command:



```
$ touch quay-config-bundle.yaml
```

3. Create a **postgresql-client-certs** resource by entering the following command:

```
$ oc -n <quay_namespace> create secret generic postgresql-client-certs \
--from-file config.yaml=<path/to/config.yaml> 1
--from-file=tls.crt=<path/to/ssl_client_certificate.pem> 2
--from-file=tls.key=<path/to/ssl_client_key.pem> 3
--from-file=ca.crt=<path/to/ssl_server_certificate.pem> 4
```

- 1 Where `<config.yaml>` is your **base64 decoded config.yaml** file.
- 2 Where **ssl\_client\_certificate.pem** is your SSL certificate in **.pem** format.
- 3 Where **ssl\_client\_key.pem** is your SSL key in **.pem** format.
- 4 Where **ssl\_server\_certificate.pem** is your SSL root CA in **.pem** format.

4. Edit your `quay-config-bundle.yaml` file to include the following database connection settings:



### IMPORTANT

- The information included in the **DB\_CONNECTION\_ARGS** variable, for example, **sslmode**, **sslrootcert**, **sslcrt**, and **sslkey** must match the information appended to the **DB\_URI** variable. Failure to match might result in a failed connection.
- You cannot specify custom filenames or paths. Certificate file paths for **sslrootcert**, **sslcrt**, and **sslkey** are hardcoded defaults and mounted into the **Quay** pod from the Kubernetes secret. You must adhere to the following naming conventions or it will result in a failed connection.

```
DB_CONNECTION_ARGS:
  autorollback: true
  sslmode: verify-ca 1
  sslrootcert: /.postgresql/root.crt 2
  sslcrt: /.postgresql/postgresql.crt 3
  sslkey: /.postgresql/postgresql.key 4
  threadlocals: true 5
DB_URI: postgresql://<dbusername>:<dbpassword>@<database_host>:
<port>/<database_name>?sslmode=verify-
full&sslrootcert=/.postgresql/root.crt&sslcrt=/.postgresql/postgresql.crt&sslkey=/.postgresql/postgresql.key 6
```

- 1 Using **verify-ca** ensures that the database connection uses SSL/TLS and verifies the server certificate against a trusted CA. This can work with both trusted CA and self-signed CA certificates. However, this mode does not verify the hostname of the server. For full hostname and certificate verification, use **verify-full**. For more information about the configuration options available, see [PostgreSQL SSL/TLS connection arguments](#).

2

The **root.crt** file contains the root certificate used to verify the SSL/TLS connection with your CloudSQL database. This file is mounted in the **Quay** pod from the Kubernetes

- 3 The **postgresql.crt** file contains the client certificate used to authenticate the connection to your CloudSQL database. This file is mounted in the **Quay** pod from the Kubernetes secret.
- 4 The **postgresql.key** file contains the private key associated with the client certificate. This file is mounted in the **Quay** pod from the Kubernetes secret.
- 5 Enables auto-rollback for connections.
- 6 The URI that accesses your CloudSQL database. Must be appended with the **sslmode** type, your **root.crt**, **postgresql.crt**, and **postgresql.key** files. The SSL/TLS information included in **DB\_URI** must match the information provided in **DB\_CONNECTION\_ARGS**. If you are using CloudSQL, you must include your database username and password in this variable.

5. Create the **configBundleSecret** resource by entering the following command:

```
$ oc create -n <namespace> -f quay-config-bundle.yaml
```

#### Example output

```
secret/quay-config-bundle created
```

6. Update the **QuayRegistry** YAML file to reference the **quay-config-bundle** object by entering the following command:

```
$ oc patch quayregistry <registry_name> -n <namespace> --type=merge -p '{"spec": {"configBundleSecret": "quay-config-bundle"}}'
```

#### Example output

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

7. Ensure that your **QuayRegistry** YAML file has been updated to use the extra CA certificate **configBundleSecret** resource by entering the following command:

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

#### Example output

```
# ...
  configBundleSecret: quay-config-bundle
# ...
```

## CHAPTER 3. ADDING ADDITIONAL CERTIFICATE AUTHORITIES FOR RED HAT QUAY

Certificate Authorities (CAs) are used by Red Hat Quay to verify SSL/TLS connections with external services, like ODIC providers, LDAP providers, storage providers, and so on.

The following sections provide information about uploading additional CAs to Red Hat Quay depending on your deployment type.

### 3.1. ADDING ADDITIONAL CERTIFICATE AUTHORITIES TO THE RED HAT QUAY CONTAINER

The **extra\_ca\_certs** directory is the directory where additional Certificate Authorities (CAs) can be stored to extend the set of trusted certificates. These certificates are used by Red Hat Quay to verify SSL/TLS connections with external services. When deploying Red Hat Quay, you can place the necessary CAs in this directory to ensure that connections to services like LDAP, OIDC, and storage systems are properly secured and validated.

For standalone Red Hat Quay deployments, you must create this directory and copy the additional CA certificates into that directory.

#### Prerequisites

- You have a CA for the desired service.

#### Procedure

1. View the certificate to be added to the container by entering the following command:

```
$ cat storage.crt
```

#### Example output

```
-----BEGIN CERTIFICATE-----  
MIIDTTCCAjWgAwIBAgIJAMVr9ngjJhzbMA0GCSqGSIb3DQEBCwUAMD0xCzAJBgNV...  
-----END CERTIFICATE-----
```

2. Create the **extra\_ca\_certs** in the **/config** folder of your Red Hat Quay directory by entering the following command:

```
$ mkdir -p /path/to/quay_config_folder/extra_ca_certs
```

3. Copy the CA file to the **extra\_ca\_certs** folder. For example:

```
$ cp storage.crt /path/to/quay_config_folder/extra_ca_certs/
```

4. Ensure that the **storage.crt** file exists within the **extra\_ca\_certs** folder by entering the following command:

```
$ tree /path/to/quay_config_folder/extra_ca_certs
```

#### Example output

```
| /path/to/quay_config_folder/extra_ca_certs
| |----- storage.crt-----
```

- Obtain the **CONTAINER ID** of your **Quay** consider by entering the following command:

```
| $ podman ps
```

#### Example output

```
| CONTAINER ID      IMAGE                                COMMAND                                CREATED
| STATUS           PORTS                                COMMAND                                CREATED
| 5a3e82c4a75f     <registry>/<repo>/quay:{productminv} "/sbin/my_init"          24 hours ago
| Up 18 hours      0.0.0.0:80->80/tcp, 0.0.0.0:443->443/tcp, 443/tcp  grave_keller
```

- Restart the container by entering the following command

```
| $ podman restart 5a3e82c4a75f
```

- Confirm that the certificate was copied into the container namespace by running the following command:

```
| $ podman exec -it 5a3e82c4a75f cat /etc/ssl/certs/storage.pem
```

#### Example output

```
| -----BEGIN CERTIFICATE-----
| MIIDTTCCAjWgAwIBAgIJAMVr9ngjJhzbMA0GCSqGSIb3DQEBCwUAMD0xCzAJBgNV...
| -----END CERTIFICATE-----
```

## 3.2. ADDING ADDITIONAL CERTIFICATE AUTHORITIES TO RED HAT QUAY ON OPENSIFT CONTAINER PLATFORM

On Red Hat Quay on OpenShift Container Platform, the **extra\_ca\_certs** configuration field is used to populate additional Certificate Authorities (CAs) into the CA directory, which then adds the CAs into the system trust bundle. These certificates are used by Red Hat Quay to verify SSL/TLS connections with external services like LDAP, OIDC, and storage systems.

When deploying or redeploying Red Hat Quay on OpenShift Container Platform, you can add one, or multiple, CAs into the CA directory to ensure that external services are properly secured and validated. On Red Hat Quay on OpenShift Container Platform deployments, you must manually add the **extra\_ca\_certs** configuration field to your **config.yaml** file and re-upload the **config.yaml** to OpenShift Container Platform.

The following procedures show you how to download your existing configuration file, add additional CAs to your Red Hat Quay on OpenShift Container Platform deployment, and then re-upload the configuration file.

### 3.2.1. Downloading the existing configuration

The following procedure shows you how to download the existing configuration by locating the **Config Bundle Secret**.

## Procedure

1. Describe the **QuayRegistry** resource by entering the following command:

```
$ oc describe quayregistry -n <quay_namespace>

# ...
Config Bundle Secret: example-registry-config-bundle-v123x
# ...
```

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

```
$ oc get secret -n <quay_namespace> <example-registry-config-bundle-v123x> -o
jsonpath='{.data}'
```

### Example output

```
{
  "config.yaml": "RkVBFVSRV9VU0 ... MDAwMAo="
}
```

3. Decode the data by entering the following command:

```
$ echo 'RkVBFVSRV9VU0 ... 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
```

4. Optional. You can export the data into a YAML file into the current directory by passing in the **>> config.yaml** flag. For example:

```
$ echo 'RkVBFVSRV9VU0 ... MDAwMAo=' | base64 --decode >> config.yaml
```

## 3.2.2. Adding additional Certificate Authorities to Red Hat Quay on OpenShift Container Platform

The following example shows you how to add additional Certificate Authorities to your Red Hat Quay on OpenShift Container Platform deployment.

### Prerequisites

- You have base64 decoded the original config bundle into a **config.yaml** file. For more information, see [Downloading the existing configuration](#).

- You have a Certificate Authority (CA) file or files.

## Procedure

- Create a new YAML file, for example, **extra-ca-certificate-config-bundle-secret.yaml**:

```
$ touch extra-ca-certificate-config-bundle-secret.yaml
```

- Create the **extra-ca-certificate-config-bundle-secret** resource.

- Create the resource by entering the following command:

```
$ oc -n <namespace> create secret generic extra-ca-certificate-config-bundle-secret \
  --from-file=config.yaml=</path/to/config.yaml> \ 1
  --from-file=extra_ca_cert_<name-of-certificate-one>=</path/to/certificate_one> \ 2
  --from-file=extra_ca_cert_<name-of-certificate-two>=</path/to/certificate_two> \ 3
  --from-file=extra_ca_cert_<name-of-certificate-three>=</path/to/certificate_three> \ 4
  --dry-run=client -o yaml > extra-ca-certificate-config-bundle-secret.yaml
```

- Where **<config.yaml>** is your **base64 decoded config.yaml** file.
- The extra CA file to be added to into the system trust bundle.
- Optional. A second CA file to be added into the system trust bundle.
- Optional. A third CA file to be added into the system trust bundle.

- Optional. You can check the content of the **extra-ca-certificate-config-bundle-secret.yaml** file by entering the following command:

```
$ cat extra-ca-certificate-config-bundle-secret.yaml
```

## Example output

```
apiVersion: v1
data:
  config.yaml:
  QUxMT1dfUFVMTFNfV0IUSE9VVF9TVFJJQ1RfTE9HR0IORzogZmFsc2UKQVVUSEVOVEID
  QVRJT05fVFIQRToGRGF0YWJhc2UKREVGVQVVMVF9UQUdfRVhQSVJBVEIPTjogMncKUFJ
  FRkVSU...
  extra_ca_cert_certificate-one:
  LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUQyVENDQXNHZ0F3SUJBZ0lVS2xO
  ai90VUJBZHBkNURjYkdRQUo4anRuKzd3d0RRWUpLb1pJaHZjTkFRRUwKQIFBd2ZERUxN
  QWtHQ...
  extra_ca_cert_certificate-three:
  LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUQ0ekNDQXN1Z0F3SUJBZ0lVQmJpT
  XNUeExjM0s4ODNWby9GTThsWXIOS2IFd0RRWUpLb1pJaHZjTkFRRUwKQIFBd2ZERUxN
  QWtHQ...
  extra_ca_cert_certificate-two:
  LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUQ0ekNDQXN1Z0F3SUJBZ0lVVFVPT
  XZ2YVdFOFRYV3djYTNoWIBCTnV2QjYwd0RRWUpLb1pJaHZjTkFRRUwKQIFBd2ZERUxN
  QWtHQ...
kind: Secret
```

```

metadata:
  creationTimestamp: null
  name: custom-ssl-config-bundle-secret
  namespace: <namespace>

```

4. Create the **configBundleSecret** resource by entering the following command:

```
$ oc create -n <namespace> -f extra-ca-certificate-config-bundle-secret.yaml
```

#### Example output

```
secret/extra-ca-certificate-config-bundle-secret created
```

5. Update the **QuayRegistry** YAML file to reference the **extra-ca-certificate-config-bundle-secret** object by entering the following command:

```
$ oc patch quayregistry <registry_name> -n <namespace> --type=merge -p '{"spec": {"configBundleSecret": "extra-ca-certificate-config-bundle-secret"}}'
```

#### Example output

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

6. Ensure that your **QuayRegistry** YAML file has been updated to use the extra CA certificate **configBundleSecret** resource by entering the following command:

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

#### Example output

```

# ...
configBundleSecret: extra-ca-certificate-config-bundle-secret
# ...

```

## 3.3. ADDING CUSTOM SSL/TLS CERTIFICATES WHEN RED HAT QUAY IS DEPLOYED ON KUBERNETES

When deployed on Kubernetes, Red Hat Quay mounts in a secret as a volume to store config assets. Currently, this breaks the upload certificate function of the superuser panel.

As a temporary workaround, **base64** encoded certificates can be added to the secret *after* Red Hat Quay has been deployed.

Use the following procedure to add custom SSL/TLS certificates when Red Hat Quay is deployed on Kubernetes.

### Prerequisites

- Red Hat Quay has been deployed.
- You have a custom **ca.crt** file.

## Procedure

1. Base64 encode the contents of an SSL/TLS certificate by entering the following command:

```
$ cat ca.crt | base64 -w 0
```

### Example output

```
...c1psWGpqeGIPQmNEWkJPMjJ5d0pDemVnR2QNCnRsbW9JdEF4YnFSdVd3PT0KLS0tLS1FTkQgQ0VSVEIGSUNBVEUtLS0tLQo=
```

2. Enter the following **kubectl** command to edit the **quay-enterprise-config-secret** file:

```
$ kubectl --namespace quay-enterprise edit secret/quay-enterprise-config-secret
```

3. Add an entry for the certificate and paste the full **base64** encoded stringer under the entry. For example:

```
custom-cert.crt:  
c1psWGpqeGIPQmNEWkJPMjJ5d0pDemVnR2QNCnRsbW9JdEF4YnFSdVd3PT0KLS0tLS1FTkQgQ0VSVEIGSUNBVEUtLS0tLQo=
```

4. Use the **kubectl delete** command to remove all Red Hat Quay pods. For example:

```
$ kubectl delete pod quay-operator.v3.7.1-6f9d859bd-p5ftc quayregistry-clair-postgres-7487f5bd86-xnxpr quayregistry-quay-app-upgrade-xq2v6 quayregistry-quay-database-859d5445ff-cqthr quayregistry-quay-redis-84f888776f-hhgms
```

Afterwards, the Red Hat Quay deployment automatically schedules replace pods with the new certificate data.