



Red Hat Quay 3

Manage Red Hat Quay

Manage Red Hat Quay

Red Hat Quay 3 Manage Red Hat Quay

Manage 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[®] 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

Manage Red Hat Quay

Table of Contents

PREFACE	7
CHAPTER 1. ADVANCED RED HAT QUAY CONFIGURATION	8
1.1. USING THE API TO MODIFY RED HAT QUAY	8
1.2. EDITING THE CONFIG.YAML FILE TO MODIFY RED HAT QUAY	8
1.2.1. Adding name and company to Red Hat Quay sign-in	8
CHAPTER 2. USING THE CONFIGURATION API	10
2.1. RETRIEVING THE DEFAULT CONFIGURATION	10
2.2. RETRIEVING THE CURRENT CONFIGURATION	10
2.3. VALIDATING CONFIGURATION USING THE API	11
2.4. DETERMINING THE REQUIRED FIELDS	12
CHAPTER 3. GETTING RED HAT QUAY RELEASE NOTIFICATIONS	13
CHAPTER 4. USING SSL TO PROTECT CONNECTIONS TO RED HAT QUAY	14
4.1. USING SSL/TLS	14
4.2. CREATING A CERTIFICATE AUTHORITY	14
4.2.1. Signing the certificate	14
4.3. CONFIGURING SSL/TLS USING THE COMMAND LINE INTERFACE	15
4.4. CONFIGURING SSL/TLS USING THE RED HAT QUAY UI	16
4.5. TESTING THE SSL/TLS CONFIGURATION USING THE CLI	17
4.6. TESTING THE SSL/TLS CONFIGURATION USING A BROWSER	17
4.7. CONFIGURING PODMAN TO TRUST THE CERTIFICATE AUTHORITY	18
4.8. CONFIGURING THE SYSTEM TO TRUST THE CERTIFICATE AUTHORITY	19
CHAPTER 5. ADDING TLS CERTIFICATES TO THE RED HAT QUAY CONTAINER	21
5.1. ADD TLS CERTIFICATES TO RED HAT QUAY	21
5.2. ADDING CUSTOM SSL/TLS CERTIFICATES WHEN RED HAT QUAY IS DEPLOYED ON KUBERNETES	21
CHAPTER 6. CONFIGURING ACTION LOG STORAGE FOR ELASTICSEARCH AND SPLUNK	23
6.1. CONFIGURING ACTION LOG STORAGE FOR ELASTICSEARCH	23
6.2. CONFIGURING ACTION LOG STORAGE FOR SPLUNK	24
6.2.1. Installing and creating a username for Splunk	24
6.2.2. Generating a Splunk token	25
6.2.2.1. Generating a Splunk token using the Splunk UI	25
6.2.2.2. Generating a Splunk token using the CLI	25
6.2.3. Configuring Red Hat Quay to use Splunk	26
6.2.4. Creating an action log	27
6.3. UNDERSTANDING USAGE LOGS	28
6.3.1. Viewing database logs	29
6.3.2. Log entry kind_ids	30
CHAPTER 7. CLAIR SECURITY SCANNER	36
7.1. CLAIR VULNERABILITY DATABASES	36
7.1.1. Information about Open Source Vulnerability (OSV) database for Clair	36
7.2. SETTING UP CLAIR ON STANDALONE RED HAT QUAY DEPLOYMENTS	36
7.3. CLAIR ON OPENSIFT CONTAINER PLATFORM	39
7.4. TESTING CLAIR	39
CHAPTER 8. REPOSITORY MIRRORING	41
8.1. REPOSITORY MIRRORING	41
8.2. REPOSITORY MIRRORING COMPARED TO GEO-REPLICATION	41

8.3. USING REPOSITORY MIRRORING	42
8.4. MIRRORING CONFIGURATION UI	43
8.5. MIRRORING CONFIGURATION FIELDS	43
8.6. MIRRORING WORKER	44
8.7. CREATING A MIRRORED REPOSITORY	44
8.7.1. Repository mirroring settings	45
8.7.2. Advanced settings	46
8.7.3. Synchronize now	47
8.8. EVENT NOTIFICATIONS FOR MIRRORING	48
8.9. MIRRORING TAG PATTERNS	48
8.9.1. Pattern syntax	48
8.9.2. Example tag patterns	48
8.10. WORKING WITH MIRRORED REPOSITORIES	49
8.11. REPOSITORY MIRRORING RECOMMENDATIONS	50
CHAPTER 9. IPV6 AND DUAL-STACK DEPLOYMENTS	52
9.1. ENABLING THE IPV6 PROTOCOL FAMILY	52
9.2. ENABLING THE DUAL-STACK PROTOCOL FAMILY	53
9.3. IPV6 AND DUAL-STACK LIMITATIONS	54
CHAPTER 10. LDAP AUTHENTICATION SETUP FOR RED HAT QUAY	55
10.1. CONSIDERATIONS WHEN ENABLING LDAP	55
Existing Red Hat Quay deployments	55
Manual User Creation and LDAP authentication	55
10.2. CONFIGURING LDAP FOR RED HAT QUAY	55
10.3. ENABLING THE LDAP_RESTRICTED_USER_FILTER CONFIGURATION FIELD	56
10.4. ENABLING THE LDAP_SUPERUSER_FILTER CONFIGURATION FIELD	57
10.5. COMMON LDAP CONFIGURATION ISSUES	58
10.6. LDAP CONFIGURATION FIELDS	59
CHAPTER 11. CONFIGURING OIDC FOR RED HAT QUAY	60
11.1. CONFIGURING MICROSOFT ENTRA ID OIDC ON A STANDALONE DEPLOYMENT OF RED HAT QUAY	60
11.2. CONFIGURING RED HAT SINGLE SIGN-ON FOR RED HAT QUAY	61
11.2.1. Configuring the Red Hat Single Sign-On Operator for use with the Red Hat Quay Operator	61
11.2.1.1. Configuring the Red Hat Quay Operator to use Red Hat Single Sign-On	62
11.3. TEAM SYNCHRONIZATION FOR RED HAT QUAY OIDC DEPLOYMENTS	64
11.3.1. Enabling synchronization for Red Hat Quay OIDC deployments	64
11.3.2. Setting up your Red Hat Quay deployment for team synchronization	65
CHAPTER 12. CONFIGURING AWS STS FOR RED HAT QUAY	67
12.1. CREATING AN IAM USER	67
12.2. CREATING AN S3 ROLE	68
12.3. CONFIGURING RED HAT QUAY TO USE AWS STS	69
CHAPTER 13. PROMETHEUS AND GRAFANA METRICS UNDER RED HAT QUAY	71
13.1. EXPOSING THE PROMETHEUS ENDPOINT	71
13.1.1. Standalone Red Hat Quay	71
13.1.2. Red Hat Quay Operator	71
13.1.3. Setting up Prometheus to consume metrics	72
13.1.4. DNS configuration under Kubernetes	72
13.1.5. DNS configuration for a manual cluster	72
13.2. INTRODUCTION TO METRICS	72
13.2.1. General registry statistics	72

13.2.2. Queue items	73
13.2.3. Garbage collection metrics	74
13.2.3.1. Multipart uploads metrics	75
13.2.4. Image push / pull metrics	76
13.2.4.1. Image pulls total	76
13.2.4.2. Image bytes pulled	77
13.2.4.3. Image pushes total	77
13.2.4.4. Image bytes pushed	77
13.2.5. Authentication metrics	77
CHAPTER 14. RED HAT QUAY QUOTA MANAGEMENT AND ENFORCEMENT OVERVIEW	79
14.1. QUOTA MANAGEMENT LIMITATIONS	79
14.2. QUOTA MANAGEMENT FOR RED HAT QUAY 3.9	79
14.2.1. Option A: Configuring quota management for Red Hat Quay 3.9 by adjusting the QUOTA_TOTAL_DELAY feature flag	80
14.2.2. Option B: Configuring quota management for Red Hat Quay 3.9 by setting QUOTA_TOTAL_DELAY_SECONDS to 0	80
14.3. TESTING QUOTA MANAGEMENT FOR RED HAT QUAY 3.9	81
14.4. SETTING DEFAULT QUOTA	83
14.5. ESTABLISHING QUOTA IN RED HAT QUAY UI	83
14.6. ESTABLISHING QUOTA WITH THE RED HAT QUAY API	89
14.6.1. Setting the quota	90
14.6.2. Viewing the quota	90
14.6.3. Modifying the quota	90
14.6.4. Pushing images	91
14.6.4.1. Pushing ubuntu:18.04	91
14.6.4.2. Using the API to view quota usage	91
14.6.4.3. Pushing another image	92
14.6.5. Rejecting pushes using quota limits	93
14.6.5.1. Setting reject and warning limits	93
14.6.5.2. Viewing reject and warning limits	94
14.6.5.3. Pushing an image when the reject limit is exceeded	94
14.6.5.4. Notifications for limits exceeded	95
14.7. CALCULATING THE TOTAL REGISTRY SIZE IN RED HAT QUAY 3.9	96
14.8. PERMANENTLY DELETING AN IMAGE TAG	97
14.8.1. Permanently deleting an image tag using the Red Hat Quay v2 UI	97
14.8.2. Permanently deleting an image tag using the Red Hat Quay legacy UI	97
CHAPTER 15. RED HAT QUAY AUTO-PRUNING OVERVIEW	99
15.1. PREREQUISITES AND LIMITATIONS FOR AUTO-PRUNING	99
15.2. MANAGING AUTO-PRUNING POLICIES USING THE RED HAT QUAY UI	99
15.2.1. Configuring the Red Hat Quay auto-pruning feature	99
15.2.2. Creating an auto-prune policy for a namespace using the Red Hat Quay v2 UI	100
15.2.3. Creating an auto-prune policy for a namespace using the Red Hat Quay API	101
15.2.4. Creating an auto-prune policy for a namespace for the current user using the API	102
15.2.5. Creating an auto-prune policy for a repository using the Red Hat Quay v2 UI	103
15.2.6. Creating an auto-prune policy for a repository using the Red Hat Quay API	105
15.2.7. Creating an auto-prune policy on a repository for a user with the API	106
CHAPTER 16. GEO-REPLICATION	108
16.1. GEO-REPLICATION FEATURES	108
16.2. GEO-REPLICATION REQUIREMENTS AND CONSTRAINTS	108
16.2.1. Enabling storage replication for standalone Red Hat Quay	109
16.2.2. Run Red Hat Quay with storage preferences	110

16.2.3. Removing a geo-replicated site from your standalone Red Hat Quay deployment	111
16.2.4. Setting up geo-replication on OpenShift Container Platform	113
16.2.4.1. Configuring geo-replication for the Red Hat Quay on OpenShift Container Platform	114
16.2.5. Removing a geo-replicated site from your Red Hat Quay on OpenShift Container Platform deployment	116
16.3. MIXED STORAGE FOR GEO-REPLICATION	118
CHAPTER 17. BACKING UP AND RESTORING RED HAT QUAY ON A STANDALONE DEPLOYMENT	119
17.1. OPTIONAL: ENABLING READ-ONLY MODE FOR RED HAT QUAY	119
17.1.1. Creating service keys for standalone Red Hat Quay	119
17.1.2. Adding keys to the PostgreSQL database	120
17.1.3. Configuring read-only mode for standalone Red Hat Quay	122
17.1.4. Updating read-only expiration time	123
17.2. BACKING UP RED HAT QUAY ON STANDALONE DEPLOYMENTS	123
17.3. RESTORING RED HAT QUAY ON STANDALONE DEPLOYMENTS	125
CHAPTER 18. MIGRATING A STANDALONE RED HAT QUAY DEPLOYMENT TO A RED HAT QUAY OPERATOR DEPLOYMENT	130
18.1. BACKING UP A STANDALONE DEPLOYMENT OF RED HAT QUAY	130
18.2. USING BACKED UP STANDALONE CONTENT TO MIGRATE TO OPENSIFT CONTAINER PLATFORM.	131
CHAPTER 19. CONFIGURING ARTIFACT TYPES	136
19.1. CONFIGURING OCI ARTIFACT TYPES	137
19.2. CONFIGURING ADDITIONAL ARTIFACT TYPES	137
19.3. CONFIGURING UNKNOWN MEDIA TYPES	138
CHAPTER 20. RED HAT QUAY GARBAGE COLLECTION	139
20.1. RED HAT QUAY GARBAGE COLLECTION IN PRACTICE	139
20.1.1. Measuring storage reclamation	140
20.2. GARBAGE COLLECTION CONFIGURATION FIELDS	140
20.3. DISABLING GARBAGE COLLECTION	142
20.4. GARBAGE COLLECTION AND QUOTA MANAGEMENT	143
20.5. GARBAGE COLLECTION IN PRACTICE	143
20.6. RED HAT QUAY GARBAGE COLLECTION METRICS	143
CHAPTER 21. USING THE V2 UI	146
21.1. V2 USER INTERFACE CONFIGURATION	146
21.1.1. Creating a new organization using the v2 UI	146
21.1.2. Deleting an organization using the v2 UI	146
21.1.3. Creating a new repository using the v2 UI	147
21.1.4. Deleting a repository using the v2 UI	147
21.1.5. Pushing an image to the v2 UI	148
21.1.6. Deleting an image using the v2 UI	148
21.1.7. Creating a new team using the Red Hat Quay v2 UI	149
21.1.8. Creating a robot account using the v2 UI	150
21.1.8.1. Bulk managing robot account repository access using the Red Hat Quay v2 UI	151
21.1.9. Creating default permissions using the Red Hat Quay v2 UI	152
21.1.10. Organization settings for the v2 UI	152
21.1.11. Viewing image tag information using the v2 UI	153
21.1.12. Adjusting repository settings using the v2 UI	153
21.2. VIEWING RED HAT QUAY TAG HISTORY	154
21.3. ADDING AND MANAGING LABELS ON THE RED HAT QUAY V2 UI	155
21.4. SETTING TAG EXPIRATIONS ON THE RED HAT QUAY V2 UI	155
21.5. SELECTING COLOR THEME PREFERENCE ON THE RED HAT QUAY V2 UI	156

21.6. VIEWING USAGE LOGS ON THE RED HAT QUAY V2 UI	156
21.7. ENABLING THE LEGACY UI	157
CHAPTER 22. PERFORMING HEALTH CHECKS ON RED HAT QUAY DEPLOYMENTS	158
22.1. RED HAT QUAY HEALTH CHECK ENDPOINTS	158
22.2. NAVIGATING TO A RED HAT QUAY HEALTH CHECK ENDPOINT	159
CHAPTER 23. BRANDING A RED HAT QUAY DEPLOYMENT ON THE LEGACY UI	160
CHAPTER 24. SCHEMA FOR RED HAT QUAY CONFIGURATION	161

PREFACE

Once you have deployed a Red Hat Quay registry, there are many ways you can further configure and manage that deployment. Topics covered here include:

- Advanced Red Hat Quay configuration
- Setting notifications to alert you of a new Red Hat Quay release
- Securing connections with SSL/TLS certificates
- Directing action logs storage to Elasticsearch
- Configuring image security scanning with Clair
- Scan pod images with the Container Security Operator
- Integrate Red Hat Quay into OpenShift Container Platform with the Quay Bridge Operator
- Mirroring images with repository mirroring
- Sharing Red Hat Quay images with a BitTorrent service
- Authenticating users with LDAP
- Enabling Quay for Prometheus and Grafana metrics
- Setting up geo-replication
- Troubleshooting Red Hat Quay

For a complete list of Red Hat Quay configuration fields, see the [Configure Red Hat Quay](#) page.

CHAPTER 1. ADVANCED RED HAT QUAY CONFIGURATION

You can configure your Red Hat Quay after initial deployment using one of the following methods:

- **Editing the `config.yaml` file.** The `config.yaml` file contains most configuration information for the Red Hat Quay cluster. Editing the `config.yaml` file directly is the primary method for advanced tuning and enabling specific features.
- **Using the Red Hat Quay API** Some Red Hat Quay features can be configured through the API.

This content in this section describes how to use each of the aforementioned interfaces and how to configure your deployment with advanced features.

1.1. USING THE API TO MODIFY RED HAT QUAY

See the [Red Hat Quay API Guide](#) for information on how to access Red Hat Quay API.

1.2. EDITING THE CONFIG.YAML FILE TO MODIFY RED HAT QUAY

Advanced features can be implemented by editing the `config.yaml` file directly. All configuration fields for Red Hat Quay features and settings are available in the [Red Hat Quay configuration guide](#).

The following example is one setting that you can change directly in the `config.yaml` file. Use this example as a reference when editing your `config.yaml` file for other features and settings.

1.2.1. Adding name and company to Red Hat Quay sign-in

By setting the `FEATURE_USER_METADATA` field to `true`, users are prompted for their name and company when they first sign in. This is an optional field, but can provide you with extra data about your Red Hat Quay users.

Use the following procedure to add a name and a company to the Red Hat Quay sign-in page.

Procedure

1. Add, or set, the `FEATURE_USER_METADATA` configuration field to `true` in your `config.yaml` file. For example:

```
# ...  
FEATURE_USER_METADATA: true  
# ...
```

1. Redeploy Red Hat Quay.
2. Now, when prompted to log in, users are requested to enter the following information:

Tell us a bit more about yourself

This information will be displayed in your user profile.

Given Name**Family Name****Company****Location**

CHAPTER 2. USING THE CONFIGURATION API

The configuration tool exposes 4 endpoints that can be used to build, validate, bundle and deploy a configuration. The config-tool API is documented at <https://github.com/quay/config-tool/blob/master/pkg/lib/editor/API.md>. In this section, you will see how to use the API to retrieve the current configuration and how to validate any changes you make.

2.1. RETRIEVING THE DEFAULT CONFIGURATION

If you are running the configuration tool for the first time, and do not have an existing configuration, you can retrieve the default configuration. Start the container in config mode:

```
$ sudo podman run --rm -it --name quay_config \  
-p 8080:8080 \  
registry.redhat.io/quay/quay-rhel8:v3.11.1 config secret
```

Use the **config** endpoint of the configuration API to get the default:

```
$ curl -X GET -u quayconfig:secret http://quay-server:8080/api/v1/config | jq
```

The value returned is the default configuration in JSON format:

```
{  
  "config.yaml": {  
    "AUTHENTICATION_TYPE": "Database",  
    "AVATAR_KIND": "local",  
    "DB_CONNECTION_ARGS": {  
      "autorollback": true,  
      "threadlocals": true  
    },  
    "DEFAULT_TAG_EXPIRATION": "2w",  
    "EXTERNAL_TLS_TERMINATION": false,  
    "FEATURE_ACTION_LOG_ROTATION": false,  
    "FEATURE_ANONYMOUS_ACCESS": true,  
    "FEATURE_APP_SPECIFIC_TOKENS": true,  
    ....  
  }  
}
```

2.2. RETRIEVING THE CURRENT CONFIGURATION

If you have already configured and deployed the Quay registry, stop the container and restart it in configuration mode, loading the existing configuration as a volume:

```
$ sudo podman run --rm -it --name quay_config \  
-p 8080:8080 \  
-v $QUAY/config:/conf/stack:Z \  
registry.redhat.io/quay/quay-rhel8:v3.11.1 config secret
```

Use the **config** endpoint of the API to get the current configuration:

```
$ curl -X GET -u quayconfig:secret http://quay-server:8080/api/v1/config | jq
```

The value returned is the current configuration in JSON format, including database and Redis configuration data:

```
{
  "config.yaml": {
    ...
    "BROWSER_API_CALLS_XHR_ONLY": false,
    "BUILDLOGS_REDIS": {
      "host": "quay-server",
      "password": "strongpassword",
      "port": 6379
    },
    "DATABASE_SECRET_KEY": "4b1c5663-88c6-47ac-b4a8-bb594660f08b",
    "DB_CONNECTION_ARGS": {
      "autorollback": true,
      "threadlocals": true
    },
    "DB_URI": "postgresql://quayuser:quaypass@quay-server:5432/quay",
    "DEFAULT_TAG_EXPIRATION": "2w",
    ...
  }
}
```

2.3. VALIDATING CONFIGURATION USING THE API

You can validate a configuration by posting it to the **config/validate** endpoint:

```
curl -u quayconfig:secret --header 'Content-Type: application/json' --request POST --data '
{
  "config.yaml": {
    ...
    "BROWSER_API_CALLS_XHR_ONLY": false,
    "BUILDLOGS_REDIS": {
      "host": "quay-server",
      "password": "strongpassword",
      "port": 6379
    },
    "DATABASE_SECRET_KEY": "4b1c5663-88c6-47ac-b4a8-bb594660f08b",
    "DB_CONNECTION_ARGS": {
      "autorollback": true,
      "threadlocals": true
    },
    "DB_URI": "postgresql://quayuser:quaypass@quay-server:5432/quay",
    "DEFAULT_TAG_EXPIRATION": "2w",
    ...
  }
}
' http://quay-server:8080/api/v1/config/validate | jq
```

The returned value is an array containing the errors found in the configuration. If the configuration is valid, an empty array [] is returned.

2.4. DETERMINING THE REQUIRED FIELDS

You can determine the required fields by posting an empty configuration structure to the **config/validate** endpoint:

```
curl -u quayconfig:secret --header 'Content-Type: application/json' --request POST --data '{
  "config.yaml": {
  }
}' http://quay-server:8080/api/v1/config/validate | jq
```

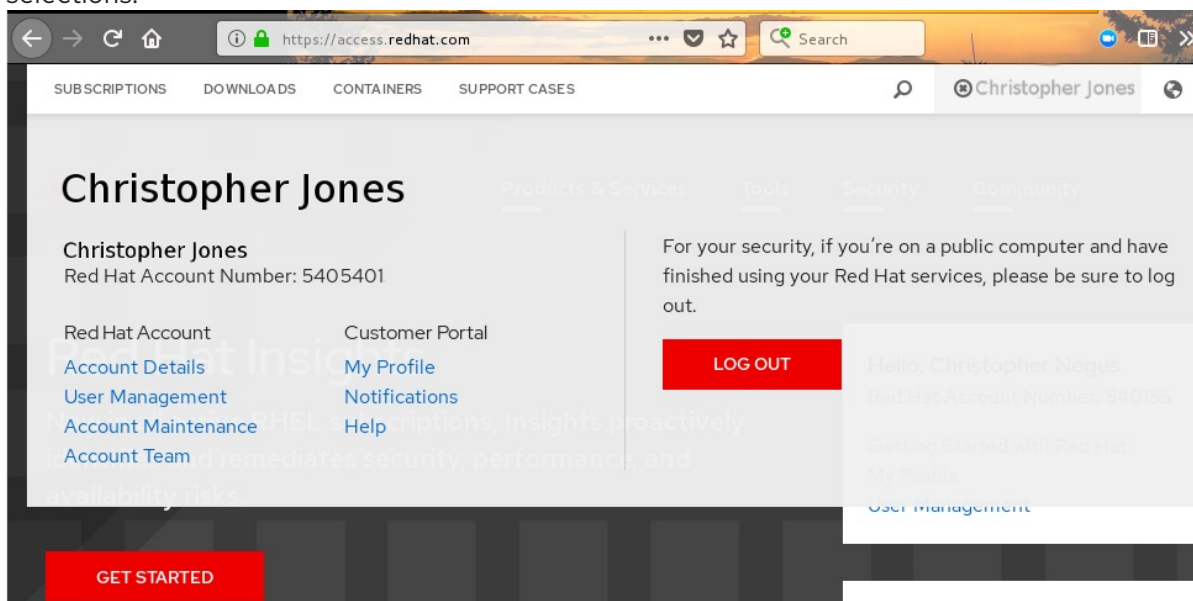
The value returned is an array indicating which fields are required:

```
[
  {
    "FieldGroup": "Database",
    "Tags": [
      "DB_URI"
    ],
    "Message": "DB_URI is required."
  },
  {
    "FieldGroup": "DistributedStorage",
    "Tags": [
      "DISTRIBUTED_STORAGE_CONFIG"
    ],
    "Message": "DISTRIBUTED_STORAGE_CONFIG must contain at least one storage location."
  },
  {
    "FieldGroup": "HostSettings",
    "Tags": [
      "SERVER_HOSTNAME"
    ],
    "Message": "SERVER_HOSTNAME is required"
  },
  {
    "FieldGroup": "HostSettings",
    "Tags": [
      "SERVER_HOSTNAME"
    ],
    "Message": "SERVER_HOSTNAME must be of type Hostname"
  },
  {
    "FieldGroup": "Redis",
    "Tags": [
      "BUILDLOGS_REDIS"
    ],
    "Message": "BUILDLOGS_REDIS is required"
  }
]
```

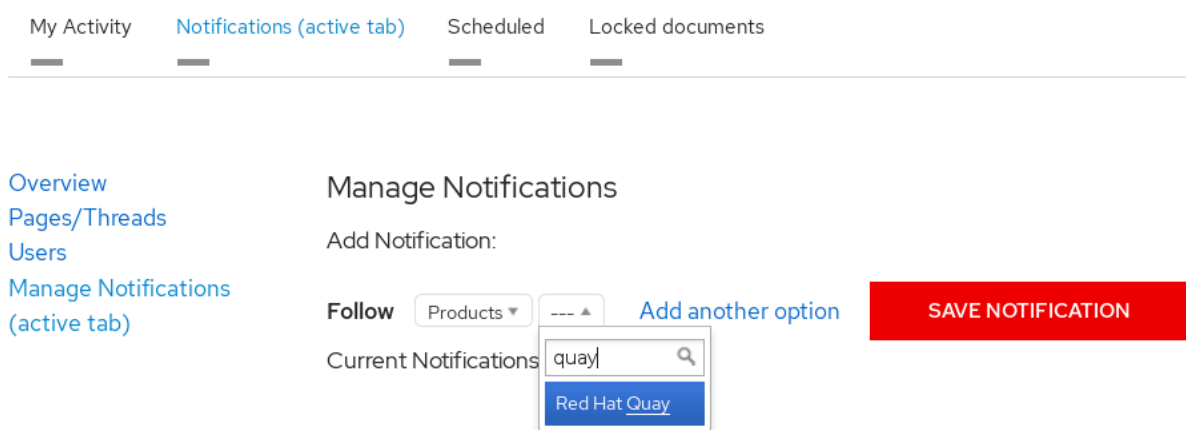

CHAPTER 3. GETTING RED HAT QUAY RELEASE NOTIFICATIONS

To keep up with the latest Red Hat Quay releases and other changes related to Red Hat Quay, you can sign up for update notifications on the [Red Hat Customer Portal](#). After signing up for notifications, you will receive notifications letting you know when there is new a Red Hat Quay version, updated documentation, or other Red Hat Quay news.

1. Log into the [Red Hat Customer Portal](#) with your Red Hat customer account credentials.
2. Select your user name (upper-right corner) to see Red Hat Account and Customer Portal selections:



3. Select Notifications. Your profile activity page appears.
4. Select the Notifications tab.
5. Select Manage Notifications.
6. Select Follow, then choose Products from the drop-down box.
7. From the drop-down box next to the Products, search for and select Red Hat Quay:



8. Select the SAVE NOTIFICATION button. Going forward, you will receive notifications when there are changes to the Red Hat Quay product, such as a new release.

CHAPTER 4. USING SSL TO PROTECT CONNECTIONS TO RED HAT QUAY

4.1. USING SSL/TLS

To configure Red Hat Quay with a self-signed certificate, you must create a Certificate Authority (CA) and a primary key file named **ssl.cert** and **ssl.key**.



NOTE

The following examples assume that you have configured the server hostname **quay-server.example.com** using DNS or another naming mechanism, such as adding an entry in your **/etc/hosts** file. For more information, see "Configuring port mapping for Red Hat Quay".

4.2. CREATING A CERTIFICATE AUTHORITY

Use the following procedure to create a Certificate Authority (CA).

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
```

4.2.1. Signing the certificate

Use the following procedure to sign the certificate.

Procedure

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

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

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

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

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

openssl.cnf

```
[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
```

5. 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
```

4.3. CONFIGURING SSL/TLS USING THE COMMAND LINE INTERFACE

Use the following procedure to configure SSL/TLS using the CLI.

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 $QUAY/config
```

2. Change into the **\$QUAY/config** directory by entering the following command:

```
$ cd $QUAY/config
```

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

config.yaml

```
...  
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
```

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:v3.11.1
```

4.4. CONFIGURING SSL/TLS USING THE RED HAT QUAY UI

Use the following procedure to configure SSL/TLS using the Red Hat Quay UI.

To configure SSL/TLS using the command line interface, see "Configuring SSL/TLS using the command line interface".

Prerequisites

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

Procedure

1. Start the **Quay** container in configuration mode:

```
$ sudo podman run --rm -it --name quay_config -p 80:8080 -p 443:8443  
registry.redhat.io/quay/quay-rhel8:v3.11.1 config secret
```

2. In the **Server Configuration** section, select **Red Hat Quay handles TLS** for SSL/TLS. Upload the certificate file and private key file created earlier, ensuring that the **Server Hostname** matches the value used when the certificates were created.
3. Validate and download the updated configuration.
4. Stop the **Quay** container and then restart the registry by entering the following command:

```
$ sudo podman rm -f quay
$ 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:v3.11.1
```

4.5. TESTING THE SSL/TLS CONFIGURATION USING THE CLI

Use the following procedure to test your SSL/TLS configuration using the CLI.

Procedure

- Enter the following command to attempt to log in to the Red Hat Quay registry with SSL/TLS enabled:

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

Example output

```
Error: error authenticating creds for "quay-server.example.com": error pinging docker registry
quay-server.example.com: Get "https://quay-server.example.com/v2/": x509: certificate
signed by unknown authority
```

1. Because Podman does not trust self-signed certificates, you must use the **--tls-verify=false** option:

```
$ sudo podman login --tls-verify=false quay-server.example.com
```

Example output

```
Login Succeeded!
```

In a subsequent section, you will configure Podman to trust the root Certificate Authority.

4.6. TESTING THE SSL/TLS CONFIGURATION USING A BROWSER

Use the following procedure to test your SSL/TLS configuration using a browser.

Procedure

1. Navigate to your Red Hat Quay registry endpoint, for example, <https://quay-server.example.com>. If configured correctly, the browser warns of the potential risk:

← → ↻ ⚠ Not secure | quay-server.example.com



Your connection is not private

Attackers might be trying to steal your information from [quay-server.example.com](#) (for example, passwords, messages or credit cards). [Learn more](#)

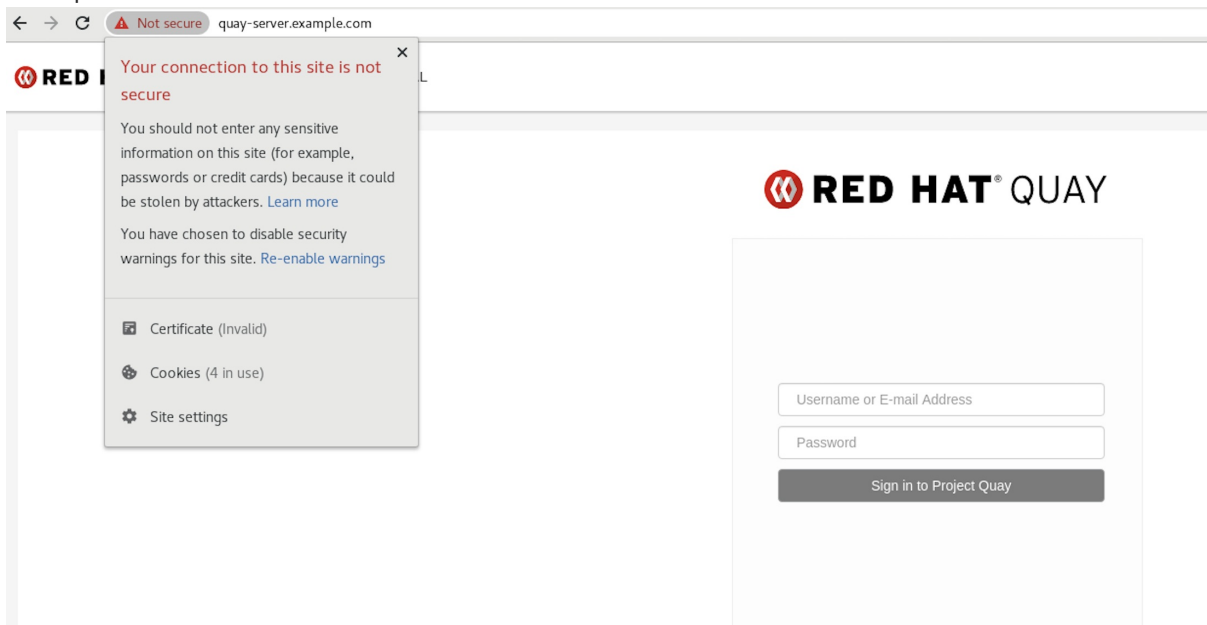
NET::ERR_CERT_AUTHORITY_INVALID

💡 To get Chrome's highest level of security, [turn on enhanced protection](#)

Advanced

Back to safety

- Proceed to the log in screen. The browser notifies you that the connection is not secure. For example:



In the following section, you will configure Podman to trust the root Certificate Authority.

4.7. 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

- 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
```

- 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!
```

4.8. CONFIGURING THE SYSTEM TO TRUST THE CERTIFICATE AUTHORITY

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

Procedure

- 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/
```

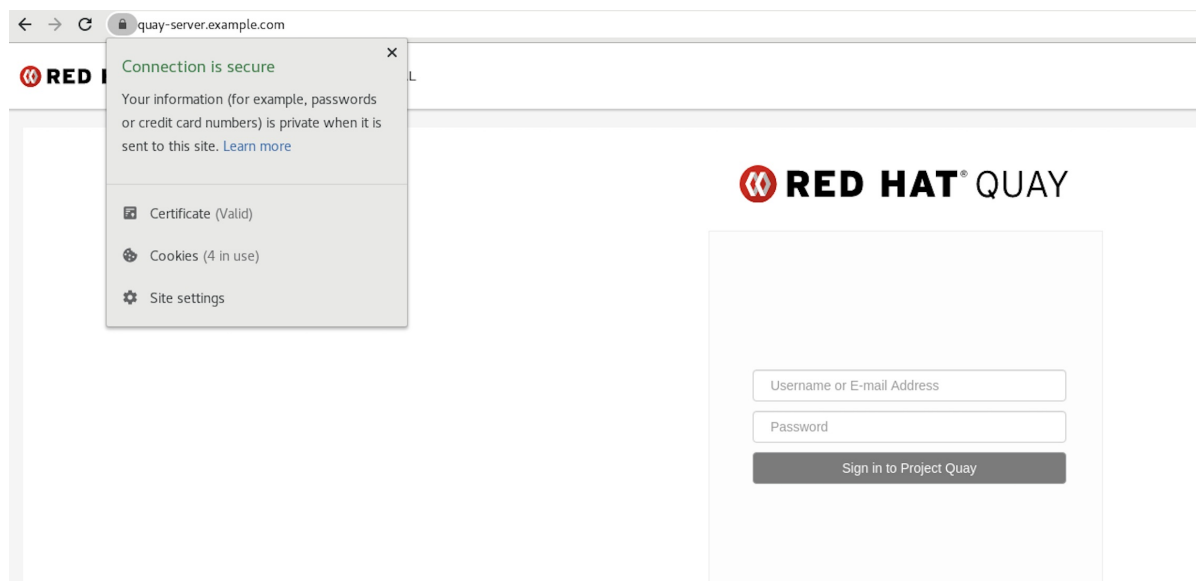
- Enter the following command to update the system-wide trust store configuration:

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

- 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:



- 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](#).

CHAPTER 5. ADDING TLS CERTIFICATES TO THE RED HAT QUAY CONTAINER

To add custom TLS certificates to Red Hat Quay, create a new directory named **extra_ca_certs/** beneath the Red Hat Quay config directory. Copy any required site-specific TLS certificates to this new directory.

5.1. ADD TLS CERTIFICATES TO RED HAT QUAY

1. View certificate to be added to the container

```
$ cat storage.crt
-----BEGIN CERTIFICATE-----
MIIDTTCCAjWgAwIbAgIJAMVr9ngjJhzbMA0GCSqGSIb3DQEBCwUAMD0xCzAJBgNV
[...]
-----END CERTIFICATE-----
```

2. Create certs directory and copy certificate there

```
$ mkdir -p quay/config/extra_ca_certs
$ cp storage.crt quay/config/extra_ca_certs/
$ tree quay/config/
|— config.yaml
|— extra_ca_certs
|  |— storage.crt
```

3. Obtain the **Quay** container's **CONTAINER ID** with **podman ps**:

```
$ sudo podman ps
CONTAINER ID      IMAGE                                COMMAND                                CREATED
STATUS           PORTS                                GRAVEYARD                                UP
5a3e82c4a75f    <registry>/<repo>/quay:v3.11.1 "/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
```

4. Restart the container with that ID:

```
$ sudo podman restart 5a3e82c4a75f
```

5. Examine the certificate copied into the container namespace:

```
$ sudo podman exec -it 5a3e82c4a75f cat /etc/ssl/certs/storage.pem
-----BEGIN CERTIFICATE-----
MIIDTTCCAjWgAwIbAgIJAMVr9ngjJhzbMA0GCSqGSIb3DQEBCwUAMD0xCzAJBgNV
```

5.2. 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 string 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.

CHAPTER 6. CONFIGURING ACTION LOG STORAGE FOR ELASTICSEARCH AND SPLUNK

By default, usage logs are stored in the Red Hat Quay database and exposed through the web UI on organization and repository levels. Appropriate administrative privileges are required to see log entries. For deployments with a large amount of logged operations, you can store the usage logs in Elasticsearch and Splunk instead of the Red Hat Quay database backend.

6.1. CONFIGURING ACTION LOG STORAGE FOR ELASTICSEARCH



NOTE

To configure action log storage for Elasticsearch, you must provide your own Elasticsearch stack; it is not included with Red Hat Quay as a customizable component.

Enabling Elasticsearch logging can be done during Red Hat Quay deployment or post-deployment by updating your **config.yaml** file. When configured, usage log access continues to be provided through the web UI for repositories and organizations.

Use the following procedure to configure action log storage for Elasticsearch:

Procedure

1. Obtain an Elasticsearch account.
2. Update your Red Hat Quay **config.yaml** file to include the following information:

```
# ...
LOGS_MODEL: elasticsearch 1
LOGS_MODEL_CONFIG:
  producer: elasticsearch 2
  elasticsearch_config:
    host: http://<host.elasticsearch.example>:<port> 3
    port: 9200 4
    access_key: <access_key> 5
    secret_key: <secret_key> 6
    use_ssl: True 7
    index_prefix: <logentry> 8
    aws_region: <us-east-1> 9
# ...
```

- 1 The method for handling log data.
- 2 Choose either Elasticsearch or Kinesis to direct logs to an intermediate Kinesis stream on AWS. You need to set up your own pipeline to send logs from Kinesis to Elasticsearch, for example, Logstash.
- 3 The hostname or IP address of the system providing the Elasticsearch service.
- 4 The port number providing the Elasticsearch service on the host you just entered. Note that the port must be accessible from all systems running the Red Hat Quay registry. The default is TCP port **9200**.

- 5 The access key needed to gain access to the Elasticsearch service, if required.
 - 6 The secret key needed to gain access to the Elasticsearch service, if required.
 - 7 Whether to use SSL/TLS for Elasticsearch. Defaults to **True**.
 - 8 Choose a prefix to attach to log entries.
 - 9 If you are running on AWS, set the AWS region (otherwise, leave it blank).
3. Optional. If you are using Kinesis as your logs producer, you must include the following fields in your **config.yaml** file:

```
kinesis_stream_config:  
  stream_name: <kinesis_stream_name> 1  
  access_key: <aws_access_key> 2  
  secret_key: <aws_secret_key> 3  
  aws_region: <aws_region> 4
```

- 1 The name of the Kinesis stream.
 - 2 The name of the AWS access key needed to gain access to the Kinesis stream, if required.
 - 3 The name of the AWS secret key needed to gain access to the Kinesis stream, if required.
 - 4 The AWS region.
4. Save your **config.yaml** file and restart your Red Hat Quay deployment.

6.2. CONFIGURING ACTION LOG STORAGE FOR SPLUNK

[Splunk](#) is an alternative to Elasticsearch that can provide log analyses for your Red Hat Quay data.

Enabling Splunk logging can be done during Red Hat Quay deployment or post-deployment using the configuration tool. The resulting configuration is stored in the **config.yaml** file. When configured, usage log access continues to be provided through the Splunk web UI for repositories and organizations.

Use the following procedures to enable Splunk for your Red Hat Quay deployment.

6.2.1. Installing and creating a username for Splunk

Use the following procedure to install and create Splunk credentials.

Procedure

1. Create a Splunk account by navigating to [Splunk](#) and entering the required credentials.
2. Navigate to the [Splunk Enterprise Free Trial](#) page, select your platform and installation package, and then click **Download Now**.
3. Install the Splunk software on your machine. When prompted, create a username, for example, **splunk_admin** and password.

4. After creating a username and password, a localhost URL will be provided for your Splunk deployment, for example, http://<sample_url>.remote.csb:8000/. Open the URL in your preferred browser.
5. Log in with the username and password you created during installation. You are directed to the Splunk UI.

6.2.2. Generating a Splunk token

Use one of the following procedures to create a bearer token for Splunk.

6.2.2.1. Generating a Splunk token using the Splunk UI

Use the following procedure to create a bearer token for Splunk using the Splunk UI.

Prerequisites

- You have installed Splunk and created a username.

Procedure

1. On the Splunk UI, navigate to **Settings** → **Tokens**.
2. Click **Enable Token Authentication**.
3. Ensure that **Token Authentication** is enabled by clicking **Token Settings** and selecting **Token Authentication** if necessary.
4. Optional: Set the expiration time for your token. This defaults at 30 days.
5. Click **Save**.
6. Click **New Token**.
7. Enter information for **User** and **Audience**.
8. Optional: Set the **Expiration** and **Not Before** information.
9. Click **Create**. Your token appears in the **Token** box. Copy the token immediately.



IMPORTANT

If you close out of the box before copying the token, you must create a new token. The token in its entirety is not available after closing the **New Token** window.

6.2.2.2. Generating a Splunk token using the CLI

Use the following procedure to create a bearer token for Splunk using the CLI.

Prerequisites

- You have installed Splunk and created a username.

Procedure

1. In your CLI, enter the following **CURL** command to enable token authentication, passing in your Splunk username and password:

```
$ curl -k -u <username>:<password> -X POST <scheme>://<host>:
<port>/services/admin/token-auth/tokens_auth -d disabled=false
```

2. Create a token by entering the following **CURL** command, passing in your Splunk username and password.

```
$ curl -k -u <username>:<password> -X POST <scheme>://<host>:
<port>/services/authorization/tokens?output_mode=json --data name=<username> --data
audience=Users --data-urlencode expires_on=+30d
```

3. Save the generated bearer token.

6.2.3. Configuring Red Hat Quay to use Splunk

Use the following procedure to configure Red Hat Quay to use Splunk.

Prerequisites

- You have installed Splunk and created a username.
- You have generated a Splunk bearer token.

Procedure

1. Open your Red Hat Quay **config.yaml** file and add the following configuration fields:

```
# ...
LOGS_MODEL: splunk
LOGS_MODEL_CONFIG:
  producer: splunk
  splunk_config:
    host: http://<user_name>.remote.csb 1
    port: 8089 2
    bearer_token: <bearer_token> 3
    url_scheme: <http/https> 4
    verify_ssl: False 5
    index_prefix: <splunk_log_index_name> 6
    ssl_ca_path: <location_to_ssl-ca-cert.pem> 7
# ...
```

- 1 String. The Splunk cluster endpoint.
- 2 Integer. The Splunk management cluster endpoint port. Differs from the Splunk GUI hosted port. Can be found on the Splunk UI under **Settings** → **Server Settings** → **General Settings**.
- 3 String. The generated bearer token for Splunk.
- 4 String. The URL scheme for access the Splunk service. If Splunk is configured to use TLS/SSL, this must be **https**.

- 5 Boolean. Whether to enable TLS/SSL. Defaults to **true**.
 - 6 String. The Splunk index prefix. Can be a new, or used, index. Can be created from the Splunk UI.
 - 7 String. The relative container path to a single **.pem** file containing a certificate authority (CA) for TLS/SSL validation.
2. If you are configuring **ssl_ca_path**, you must configure the SSL/TLS certificate so that Red Hat Quay will trust it.
 - a. If you are using a standalone deployment of Red Hat Quay, SSL/TLS certificates can be provided by placing the certificate file inside of the **extra_ca_certs** directory, or inside of the relative container path and specified by **ssl_ca_path**.
 - b. If you are using the Red Hat Quay Operator, create a config bundle secret, including the certificate authority (CA) of the Splunk server. For example:

```
$ oc create secret generic --from-file config.yaml=./config_390.yaml --from-file
extra_ca_cert_splunkserver.crt=./splunkserver.crt config-bundle-secret
```

Specify the **conf/stack/extra_ca_certs/splunkserver.crt** file in your **config.yaml**. For example:

```
# ...
LOGS_MODEL: splunk
LOGS_MODEL_CONFIG:
  producer: splunk
  splunk_config:
    host: ec2-12-345-67-891.us-east-2.compute.amazonaws.com
    port: 8089
    bearer_token: eyJra
    url_scheme: https
    verify_ssl: true
    index_prefix: quay123456
    ssl_ca_path: conf/stack/splunkserver.crt
# ...
```

6.2.4. Creating an action log

Use the following procedure to create a user account that can forward action logs to Splunk.



IMPORTANT

You must use the Splunk UI to view Red Hat Quay action logs. At this time, viewing Splunk action logs on the Red Hat Quay **Usage Logs** page is unsupported, and returns the following message: **Method not implemented. Splunk does not support log lookups.**

Prerequisites

- You have installed Splunk and created a username.
- You have generated a Splunk bearer token.

- You have configured your Red Hat Quay **config.yaml** file to enable Splunk.

Procedure

1. Log in to your Red Hat Quay deployment.
2. Click on the name of the organization that you will use to create an action log for Splunk.
3. In the navigation pane, click **Robot Accounts** → **Create Robot Account**
4. When prompted, enter a name for the robot account, for example **spunkrobotaccount**, then click **Create robot account**
5. On your browser, open the Splunk UI.
6. Click **Search and Reporting**.
7. In the search bar, enter the name of your index, for example, **<splunk_log_index_name>** and press **Enter**.
The search results populate on the Splunk UI. Logs are forwarded in JSON format. A response might look similar to the following:

```
{
  "log_data": {
    "kind": "authentication", 1
    "account": "quayuser123", 2
    "performer": "John Doe", 3
    "repository": "projectQuay", 4
    "ip": "192.168.1.100", 5
    "metadata_json": {...}, 6
    "datetime": "2024-02-06T12:30:45Z" 7
  }
}
```

- 1 Specifies the type of log event. In this example, **authentication** indicates that the log entry relates to an authentication event.
- 2 The user account involved in the event.
- 3 The individual who performed the action.
- 4 The repository associated with the event.
- 5 The IP address from which the action was performed.
- 6 Might contain additional metadata related to the event.
- 7 The timestamp of when the event occurred.

6.3. UNDERSTANDING USAGE LOGS

By default, usage logs are stored in the Red Hat Quay database. They are exposed through the web UI, on the organization and repository levels, and in the **Superuser Admin Panel**.

Database logs capture a wide ranges of events in Red Hat Quay, such as the changing of account plans, user actions, and general operations. Log entries include information such as the action performed (**kind_id**), the user who performed the action (**account_id** or **performer_id**), the timestamp (**datetime**), and other relevant data associated with the action (**metadata_json**).

6.3.1. Viewing database logs

The following procedure shows you how to view repository logs that are stored in a PostgreSQL database.

Prerequisites

- You have administrative privileges.
- You have installed the **psql** CLI tool.

Procedure

1. Enter the following command to log in to your Red Hat Quay PostgreSQL database:

```
$ psql -h <quay-server.example.com> -p 5432 -U <user_name> -d <database_name>
```

Example output

```
psql (16.1, server 13.7)
Type "help" for help.
```

2. Optional. Enter the following command to display the tables list of your PostgreSQL database:

```
quay=> \dt
```

Example output

```

          List of relations
 Schema |      Name      | Type | Owner
-----+-----+-----+-----
 public | logentry       | table | quayuser
 public | logentry2      | table | quayuser
 public | logentry3      | table | quayuser
 public | logentrykind   | table | quayuser
 ...
```

3. You can enter the following command to return a list of **repository_ids** that are required to return log information:

```
quay=> SELECT id, name FROM repository;
```

Example output

```

 id |      name
-----+-----
  3 | new_repository_name
```

```
6 | api-repo
7 | busybox
...
```

4. Enter the following command to use the **logentry3** relation to show log information about one of your repositories:

```
SELECT * FROM logentry3 WHERE repository_id = <repository_id>;
```

Example output

```
id | kind_id | account_id | performer_id | repository_id | datetime | ip | metadata_json
59 | 14 | 2 | 1 | 6 | 2024-05-13 15:51:01.897189 | 192.168.1.130 | {"repo": "api-repo",
"namespace": "test-org"}
```

In the above example, the following information is returned:

```
{
  "log_data": {
    "id": 59 1
    "kind_id": "14", 2
    "account_id": "2", 3
    "performer_id": "1", 4
    "repository_id": "6", 5
    "ip": "192.168.1.100", 6
    "metadata_json": {"repo": "api-repo", "namespace": "test-org"} 7
    "datetime": "2024-05-13 15:51:01.897189" 8
  }
}
```

- 1** The unique identifier for the log entry.
- 2** The action that was done. In this example, it was **14**. The key, or table, in the following section shows you that this **kind_id** is related to the creation of a repository.
- 3** The account that performed the action.
- 4** The performer of the action.
- 5** The repository that the action was done on. In this example, **6** correlates to the **api-repo** that was discovered in Step 3.
- 6** The IP address where the action was performed.
- 7** Metadata information, including the name of the repository and its namespace.
- 8** The time when the action was performed.

6.3.2. Log entry kind_ids

The following table represents the **kind_ids** associated with Red Hat Quay actions.

kind_id	Action	Description
1	account_change_cc	Change of credit card information.
2	account_change_password	Change of account password.
3	account_change_plan	Change of account plan.
4	account_convert	Account conversion.
5	add_repo_accesstoken	Adding an access token to a repository.
6	add_repo_notification	Adding a notification to a repository.
7	add_repo_permission	Adding permissions to a repository.
8	add_repo_webhook	Adding a webhook to a repository.
9	build_dockerfile	Building a Dockerfile.
10	change_repo_permission	Changing permissions of a repository.
11	change_repo_visibility	Changing the visibility of a repository.
12	create_application	Creating an application.
13	create_prototype_permission	Creating permissions for a prototype.
14	create_repo	Creating a repository.
15	create_robot	Creating a robot (service account or bot).
16	create_tag	Creating a tag.
17	delete_application	Deleting an application.
18	delete_prototype_permission	Deleting permissions for a prototype.
19	delete_repo	Deleting a repository.
20	delete_repo_accesstoken	Deleting an access token from a repository.
21	delete_repo_notification	Deleting a notification from a repository.
22	delete_repo_permission	Deleting permissions from a repository.
23	delete_repo_trigger	Deleting a repository trigger.

kind_id	Action	Description
24	delete_repo_webhook	Deleting a webhook from a repository.
25	delete_robot	Deleting a robot.
26	delete_tag	Deleting a tag.
27	manifest_label_add	Adding a label to a manifest.
28	manifest_label_delete	Deleting a label from a manifest.
29	modify_prototype_permission	Modifying permissions for a prototype.
30	move_tag	Moving a tag.
31	org_add_team_member	Adding a member to a team.
32	org_create_team	Creating a team within an organization.
33	org_delete_team	Deleting a team within an organization.
34	org_delete_team_member_invite	Deleting a team member invitation.
35	org_invite_team_member	Inviting a member to a team in an organization.
36	org_remove_team_member	Removing a member from a team.
37	org_set_team_description	Setting the description of a team.
38	org_set_team_role	Setting the role of a team.
39	org_team_member_invite_accepted	Acceptance of a team member invitation.
40	org_team_member_invite_declined	Declining of a team member invitation.
41	pull_repo	Pull from a repository.
42	push_repo	Push to a repository.
43	regenerate_robot_token	Regenerating a robot token.
44	repo_verb	Generic repository action (specifics might be defined elsewhere).

kind_id	Action	Description
45	reset_application_client_secret	Resetting the client secret of an application.
46	revert_tag	Reverting a tag.
47	service_key_approve	Approving a service key.
48	service_key_create	Creating a service key.
49	service_key_delete	Deleting a service key.
50	service_key_extend	Extending a service key.
51	service_key_modify	Modifying a service key.
52	service_key_rotate	Rotating a service key.
53	setup_repo_trigger	Setting up a repository trigger.
54	set_repo_description	Setting the description of a repository.
55	take_ownership	Taking ownership of a resource.
56	update_application	Updating an application.
57	change_repo_trust	Changing the trust level of a repository.
58	reset_repo_notification	Resetting repository notifications.
59	change_tag_expiration	Changing the expiration date of a tag.
60	create_app_specific_token	Creating an application-specific token.
61	revoke_app_specific_token	Revoking an application-specific token.
62	toggle_repo_trigger	Toggling a repository trigger on or off.
63	repo_mirror_enabled	Enabling repository mirroring.
64	repo_mirror_disabled	Disabling repository mirroring.
65	repo_mirror_config_changed	Changing the configuration of repository mirroring.
66	repo_mirror_sync_started	Starting a repository mirror sync.

kind_id	Action	Description
67	repo_mirror_sync_failed	Repository mirror sync failed.
68	repo_mirror_sync_success	Repository mirror sync succeeded.
69	repo_mirror_sync_now_requested	Immediate repository mirror sync requested.
70	repo_mirror_sync_tag_success	Repository mirror tag sync succeeded.
71	repo_mirror_sync_tag_failed	Repository mirror tag sync failed.
72	repo_mirror_sync_test_success	Repository mirror sync test succeeded.
73	repo_mirror_sync_test_failed	Repository mirror sync test failed.
74	repo_mirror_sync_test_started	Repository mirror sync test started.
75	change_repo_state	Changing the state of a repository.
76	create_proxy_cache_config	Creating proxy cache configuration.
77	delete_proxy_cache_config	Deleting proxy cache configuration.
78	start_build_trigger	Starting a build trigger.
79	cancel_build	Cancelling a build.
80	org_create	Creating an organization.
81	org_delete	Deleting an organization.
82	org_change_email	Changing organization email.
83	org_change_invoicing	Changing organization invoicing.
84	org_change_tag_expiration	Changing organization tag expiration.
85	org_change_name	Changing organization name.
86	user_create	Creating a user.
87	user_delete	Deleting a user.
88	user_disable	Disabling a user.

kind_id	Action	Description
89	user_enable	Enabling a user.
90	user_change_email	Changing user email.
91	user_change_password	Changing user password.
92	user_change_name	Changing user name.
93	user_change_invoicing	Changing user invoicing.
94	user_change_tag_expiration	Changing user tag expiration.
95	user_change_metadata	Changing user metadata.
96	user_generate_client_key	Generating a client key for a user.
97	login_success	Successful login.
98	logout_success	Successful logout.
99	permanently_delete_tag	Permanently deleting a tag.
100	autoprunes_tag_delete	Auto-pruning tag deletion.
101	create_namespace_autoprune_policy	Creating namespace auto-prune policy.
102	update_namespace_autoprune_policy	Updating namespace auto-prune policy.
103	delete_namespace_autoprune_policy	Deleting namespace auto-prune policy.
104	login_failure	Failed login attempt.

CHAPTER 7. CLAIR SECURITY SCANNER

7.1. CLAIR VULNERABILITY DATABASES

Clair uses the following vulnerability databases to report for issues in your images:

- Ubuntu Oval database
- Debian Security Tracker
- Red Hat Enterprise Linux (RHEL) Oval database
- SUSE Oval database
- Oracle Oval database
- Alpine SecDB database
- VMware Photon OS database
- Amazon Web Services (AWS) UpdateInfo
- [Open Source Vulnerability \(OSV\) Database](#)

For information about how Clair does security mapping with the different databases, see [Claircore Severity Mapping](#).

7.1.1. Information about Open Source Vulnerability (OSV) database for Clair

Open Source Vulnerability (OSV) is a vulnerability database and monitoring service that focuses on tracking and managing security vulnerabilities in open source software.

OSV provides a comprehensive and up-to-date database of known security vulnerabilities in open source projects. It covers a wide range of open source software, including libraries, frameworks, and other components that are used in software development. For a full list of included ecosystems, see [defined ecosystems](#).

Clair also reports vulnerability and security information for **golang**, **java**, and **ruby** ecosystems through the Open Source Vulnerability (OSV) database.

By leveraging OSV, developers and organizations can proactively monitor and address security vulnerabilities in open source components that they use, which helps to reduce the risk of security breaches and data compromises in projects.

For more information about OSV, see [the OSV website](#).

7.2. SETTING UP CLAIR ON STANDALONE RED HAT QUAY DEPLOYMENTS

For standalone Red Hat Quay deployments, you can set up Clair manually.

Procedure

1. In your Red Hat Quay installation directory, create a new directory for the Clair database data:
 -


```
$ mkdir /home/<user-name>/quay-poc/postgres-clairv4
```

- Set the appropriate permissions for the **postgres-clairv4** file by entering the following command:

```
$ setfacl -m u:26:-wx /home/<user-name>/quay-poc/postgres-clairv4
```

- Deploy a Clair PostgreSQL database by entering the following command:

```
$ sudo podman run -d --name postgresql-clairv4 \
  -e POSTGRES_USER=clairuser \
  -e POSTGRES_PASSWORD=clairpass \
  -e POSTGRES_DATABASE=clair \
  -e POSTGRES_ADMIN_PASSWORD=adminpass \
  -p 5433:5432 \
  -v /home/<user-name>/quay-poc/postgres-clairv4:/var/lib/pgsql/data:Z \
  registry.redhat.io/rhel8/postgresql-13:1-109
```

- Install the PostgreSQL **uuid-osp** module for your Clair deployment:

```
$ podman exec -it postgresql-clairv4 /bin/bash -c 'echo "CREATE EXTENSION IF NOT EXISTS \"uuid-osp\"\" | psql -d clair -U postgres'
```

Example output

```
CREATE EXTENSION
```



NOTE

Clair requires the **uuid-osp** extension to be added to its PostgreSQL database. For users with proper privileges, creating the extension will automatically be added by Clair. If users do not have the proper privileges, the extension must be added before start Clair.

If the extension is not present, the following error will be displayed when Clair attempts to start: **ERROR: Please load the "uuid-osp" extension. (SQLSTATE 42501).**

- Stop the **Quay** container if it is running and restart it in configuration mode, loading the existing configuration as a volume:

```
$ sudo podman run --rm -it --name quay_config \
  -p 80:8080 -p 443:8443 \
  -v $QUAY/config:/conf/stack:Z \
  {productrepo}/{quayimage}:{productminv} config secret
```

- Log in to the configuration tool and click **Enable Security Scanning** in the **Security Scanner** section of the UI.
- Set the HTTP endpoint for Clair using a port that is not already in use on the **quay-server** system, for example, **8081**.
- Create a pre-shared key (PSK) using the **Generate PSK** button.

Security Scanner UI

Security Scanner

If enabled, all images pushed to Quay will be scanned via the external security scanning service, with vulnerability information available in the UI and API, as well as async notification support.

Enable Security Scanning

i A scanner compliant with the Quay Security Scanning API must be running to use this feature. Documentation on running Clair can be found at [Running Clair Security Scanner](#).

Security Scanner Endpoint:
The HTTP URL at which the security scanner is running.

Security Scanner PSK:
Clair Pre-Shared Key. Make sure to include this value in your Clair config.

- Validate and download the **config.yaml** file for Red Hat Quay, and then stop the **Quay** container that is running the configuration editor.
- Extract the new configuration bundle into your Red Hat Quay installation directory, for example:

```
$ tar xvf quay-config.tar.gz -d /home/<user-name>/quay-poc/
```

- Create a folder for your Clair configuration file, for example:

```
$ mkdir /etc/opt/clairv4/config/
```

- Change into the Clair configuration folder:

```
$ cd /etc/opt/clairv4/config/
```

- Create a Clair configuration file, for example:

```
http_listen_addr: :8081
introspection_addr: :8088
log_level: debug
indexer:
  connstring: host=quay-server.example.com port=5433 dbname=clair user=clairuser
  password=clairpass sslmode=disable
  scanlock_retry: 10
  layer_scan_concurrency: 5
  migrations: true
matcher:
  connstring: host=quay-server.example.com port=5433 dbname=clair user=clairuser
  password=clairpass sslmode=disable
  max_conn_pool: 100
  migrations: true
  indexer_addr: clair-indexer
notifier:
  connstring: host=quay-server.example.com port=5433 dbname=clair user=clairuser
  password=clairpass sslmode=disable
  delivery_interval: 1m
  poll_interval: 5m
  migrations: true
auth:
  psk:
    key: "MTU5YzA4Y2ZkNzJoMQ=="
    iss: ["quay"]
```

```
# tracing and metrics
trace:
  name: "jaeger"
  probability: 1
  jaeger:
    agent:
      endpoint: "localhost:6831"
      service_name: "clair"
metrics:
  name: "prometheus"
```

For more information about Clair's configuration format, see [Clair configuration reference](#).

14. Start Clair by using the container image, mounting in the configuration from the file you created:

```
$ sudo podman run -d --name clairv4 \
-p 8081:8081 -p 8088:8088 \
-e CLAIR_CONF=/clair/config.yaml \
-e CLAIR_MODE=combo \
-v /etc/opt/clairv4/config:/clair:Z \
registry.redhat.io/quay/clair-rhel8:v3.11.1
```



NOTE

Running multiple Clair containers is also possible, but for deployment scenarios beyond a single container the use of a container orchestrator like Kubernetes or OpenShift Container Platform is strongly recommended.

7.3. CLAIR ON OPENSIFT CONTAINER PLATFORM

To set up Clair v4 (Clair) on a Red Hat Quay deployment on OpenShift Container Platform, it is recommended to use the Red Hat Quay Operator. By default, the Red Hat Quay Operator installs or upgrades a Clair deployment along with your Red Hat Quay deployment and configure Clair automatically.

7.4. TESTING CLAIR

Use the following procedure to test Clair on either a standalone Red Hat Quay deployment, or on an OpenShift Container Platform Operator-based deployment.

Prerequisites

- You have deployed the Clair container image.

Procedure

1. Pull a sample image by entering the following command:

```
$ podman pull ubuntu:20.04
```

2. Tag the image to your registry by entering the following command:

```
$ sudo podman tag docker.io/library/ubuntu:20.04 <quay-server.example.com>/<user-name>/ubuntu:20.04
```

3. Push the image to your Red Hat Quay registry by entering the following command:

```
$ sudo podman push --tls-verify=false quay-server.example.com/quayadmin/ubuntu:20.04
```

4. Log in to your Red Hat Quay deployment through the UI.
5. Click the repository name, for example, **quayadmin/ubuntu**.
6. In the navigation pane, click **Tags**.

Report summary

Repository Tags

TAG	LAST MODIFIED	SECURITY SCAN	SIZE	EXPIRES	MANIFEST
18.04	9 days ago	6 High · 82 fixable	25.5 MB	Never	SHA256 b58746c8a899
19.04	10 days ago	Passed	26.4 MB	Never	SHA256 61844ceb1dd5

7. Click the image report, for example, **45 medium**, to show a more detailed report:

Report details

Quay Security Scanner has detected **146** vulnerabilities.
Patches are available for **82** vulnerabilities.

- 6 High-level vulnerabilities.
- 45 Medium-level vulnerabilities.
- 57 Low-level vulnerabilities.
- 38 Negligible-level vulnerabilities.

Vulnerabilities

CVE	SEVERITY	PACKAGE	CURRENT VERSION	FIXED IN VERSION	INTRODUCED IN LAYER
CVE-2019-3462	High	apt	1.6.12	1.7.0ubuntu0.1	file:c3e6bb316dfa6b81dd4478aaa310df532883...
CVE-2019-3462	High	libapt-pkg5.0	1.6.12	1.7.0ubuntu0.1	file:c3e6bb316dfa6b81dd4478aaa310df532883...
CVE-2018-16864	High	libudev1	237-3ubuntu10.39	239-7ubuntu10.6	file:c3e6bb316dfa6b81dd4478aaa310df532883...



NOTE

In some cases, Clair shows duplicate reports on images, for example, **ubi8/nodejs-12** or **ubi8/nodejs-16**. This occurs because vulnerabilities with same name are for different packages. This behavior is expected with Clair vulnerability reporting and will not be addressed as a bug.

CHAPTER 8. REPOSITORY MIRRORING

8.1. REPOSITORY MIRRORING

Red Hat Quay repository mirroring lets you mirror images from external container registries, or another local registry, into your Red Hat Quay cluster. Using repository mirroring, you can synchronize images to Red Hat Quay based on repository names and tags.

From your Red Hat Quay cluster with repository mirroring enabled, you can perform the following:

- Choose a repository from an external registry to mirror
- Add credentials to access the external registry
- Identify specific container image repository names and tags to sync
- Set intervals at which a repository is synced
- Check the current state of synchronization

To use the mirroring functionality, you need to perform the following actions:

- Enable repository mirroring in the Red Hat Quay configuration file
- Run a repository mirroring worker
- Create mirrored repositories

All repository mirroring configurations can be performed using the configuration tool UI or by the Red Hat Quay API.

8.2. REPOSITORY MIRRORING COMPARED TO GEO-REPLICATION

Red Hat Quay geo-replication mirrors the entire image storage backend data between 2 or more different storage backends while the database is shared, for example, one Red Hat Quay registry with two different blob storage endpoints. The primary use cases for geo-replication include the following:

- Speeding up access to the binary blobs for geographically dispersed setups
- Guaranteeing that the image content is the same across regions

Repository mirroring synchronizes selected repositories, or subsets of repositories, from one registry to another. The registries are distinct, with each registry having a separate database and separate image storage.

The primary use cases for mirroring include the following:

- Independent registry deployments in different data centers or regions, where a certain subset of the overall content is supposed to be shared across the data centers and regions
- Automatic synchronization or mirroring of selected (allowlisted) upstream repositories from external registries into a local Red Hat Quay deployment

**NOTE**

Repository mirroring and geo-replication can be used simultaneously.

Table 8.1. Red Hat Quay Repository mirroring and geo-replication comparison

Feature / Capability	Geo-replication	Repository mirroring
What is the feature designed to do?	A shared, global registry	Distinct, different registries
What happens if replication or mirroring has not been completed yet?	The remote copy is used (slower)	No image is served
Is access to all storage backends in both regions required?	Yes (all Red Hat Quay nodes)	No (distinct storage)
Can users push images from both sites to the same repository?	Yes	No
Is all registry content and configuration identical across all regions (shared database)?	Yes	No
Can users select individual namespaces or repositories to be mirrored?	No	Yes
Can users apply filters to synchronization rules?	No	Yes
Are individual / different role-base access control configurations allowed in each region	No	Yes

8.3. USING REPOSITORY MIRRORING

The following list shows features and limitations of Red Hat Quay repository mirroring:

- With repository mirroring, you can mirror an entire repository or selectively limit which images are synced. Filters can be based on a comma-separated list of tags, a range of tags, or other means of identifying tags through Unix shell-style wildcards. For more information, see the documentation for [wildcards](#).
- When a repository is set as mirrored, you cannot manually add other images to that repository.
- Because the mirrored repository is based on the repository and tags you set, it will hold only the content represented by the repository and tag pair. For example if you change the tag so that some images in the repository no longer match, those images will be deleted.

- Only the designated robot can push images to a mirrored repository, superseding any role-based access control permissions set on the repository.
- Mirroring can be configured to rollback on failure, *or* to run on a best-effort basis.
- With a mirrored repository, a user with *read* permissions can pull images from the repository but cannot push images to the repository.
- Changing settings on your mirrored repository can be performed in the Red Hat Quay user interface, using the **Repositories** → **Mirrors** tab for the mirrored repository you create.
- Images are synced at set intervals, but can also be synced on demand.


8.4. MIRRORING CONFIGURATION UI

1. Start the **Quay** container in configuration mode and select the Enable Repository Mirroring check box. If you want to require HTTPS communications and verify certificates during mirroring, select the HTTPS and cert verification check box.

 Repository Mirroring

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

Enable Repository Mirroring

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

Require HTTPS and verify certificates of Quay registry during mirror.

2. Validate and download the **configuration** file, and then restart Quay in registry mode using the updated config file.

8.5. MIRRORING CONFIGURATION FIELDS

Table 8.2. Mirroring configuration

Field	Type	Description
FEATURE_REPO_MIRROR	Boolean	Enable or disable repository mirroring Default: false
REPO_MIRROR_INTERVAL	Number	The number of seconds between checking for repository mirror candidates Default: 30

Field	Type	Description
REPO_MIRROR_SERVER_HOSTNAME	String	Replaces the SERVER_HOSTNAME as the destination for mirroring. Default: None Example: openshift-quay-service
REPO_MIRROR_TLS_VERIFY	Boolean	Require HTTPS and verify certificates of Quay registry during mirror. Default: false
REPO_MIRROR_ROLLBACK	Boolean	When set to true , the repository rolls back after a failed mirror attempt. Default: false

8.6. MIRRORING WORKER

Use the following procedure to start the repository mirroring worker.

Procedure

- If you have not configured TLS communications using a **/root/ca.crt** certificate, enter the following command to start a **Quay** pod with the **repomirror** option:

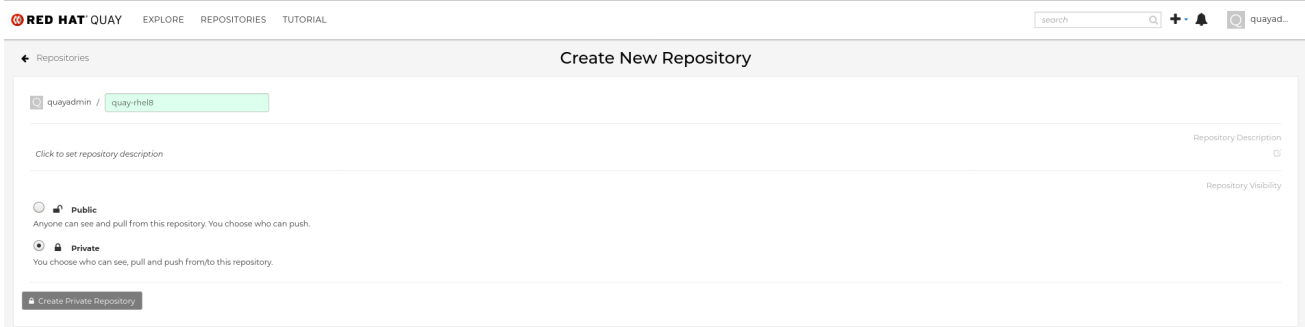
```
$ sudo podman run -d --name mirroring-worker \
-v $QUAY/config:/conf/stack:Z \
registry.redhat.io/quay/quay-rhel8:v3.11.1 repomirror
```

- If you have configured TLS communications using a **/root/ca.crt** certificate, enter the following command to start the repository mirroring worker:

```
$ sudo podman run -d --name mirroring-worker \
-v $QUAY/config:/conf/stack:Z \
-v /root/ca.crt:/etc/pki/ca-trust/source/anchors/ca.crt:Z \
registry.redhat.io/quay/quay-rhel8:v3.11.1 repomirror
```

8.7. CREATING A MIRRORED REPOSITORY

When mirroring a repository from an external container registry, you must create a new private repository. Typically, the same name is used as the target repository, for example, **quay-rhel8**.



8.7.1. Repository mirroring settings

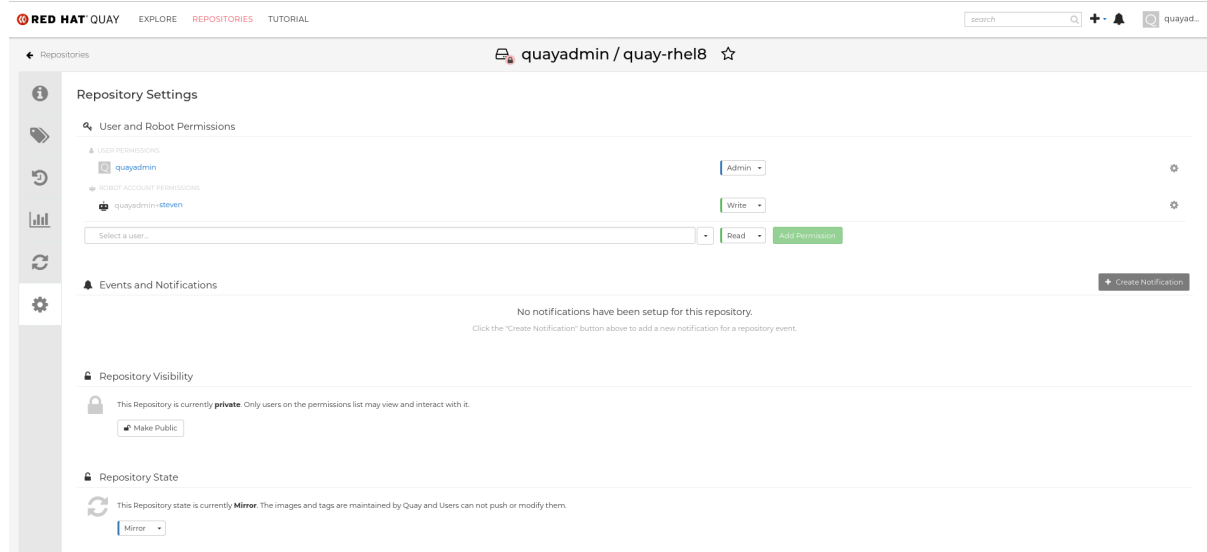
Use the following procedure to adjust the settings of your mirrored repository.

Prerequisites

- You have enabled repository mirroring in your Red Hat Quay configuration file.
- You have deployed a mirroring worker.

Procedure

1. In the Settings tab, set the Repository State to **Mirror**:



2. In the Mirror tab, enter the details for connecting to the external registry, along with the tags, scheduling and access information:

← Repositories testorg / busybox ☆

Repository Mirroring

This feature will convert `testorg/busybox` into a mirror. Changes to the external repository will be duplicated here. While enabled, users will be unable to push images to this repository.

External Repository

Registry Location:

Tags:

Start Date:

Sync Interval:

Robot User:

Credentials

Required if the external repository is private.

Username:

Password:

Advanced Settings

Verify TLS: Require HTTPS and verify certificates when talking to the external registry.

Accept Unsigned Images: Allow unsigned images to be mirrored.

HTTP Proxy:

HTTPs Proxy:

No Proxy:

3. Enter the details as required in the following fields:

- **Registry Location:** The external repository you want to mirror, for example, **registry.redhat.io/quay/quay-rhel8**
- **Tags:** This field is required. You may enter a comma-separated list of individual tags or tag patterns. (See *Tag Patterns* section for details.)
- **Start Date:** The date on which mirroring begins. The current date and time is used by default.
- **Sync Interval:** Defaults to syncing every 24 hours. You can change that based on hours or days.
- **Robot User:** Create a new robot account or choose an existing robot account to do the mirroring.
- **Username:** The username for accessing the external registry holding the repository you are mirroring.
- **Password:** The password associated with the Username. Note that the password cannot include characters that require an escape character (`\`).

8.7.2. Advanced settings

In the **Advanced Settings** section, you can configure SSL/TLS and proxy with the following options:

- **Verify TLS:** Select this option if you want to require HTTPS and to verify certificates when communicating with the target remote registry.
- **Accept Unsigned Images:** Selecting this option allows unsigned images to be mirrored.

- **HTTP Proxy:** Select this option if you want to require HTTPS and to verify certificates when communicating with the target remote registry.
- **HTTPS PROXY:** Identify the HTTPS proxy server needed to access the remote site, if a proxy server is needed.
- **No Proxy:** List of locations that do not require proxy.

8.7.3. Synchronize now

Use the following procedure to initiate the mirroring operation.

Procedure

- To perform an immediate mirroring operation, press the Sync Now button on the repository's Mirroring tab. The logs are available on the Usage Logs tab:

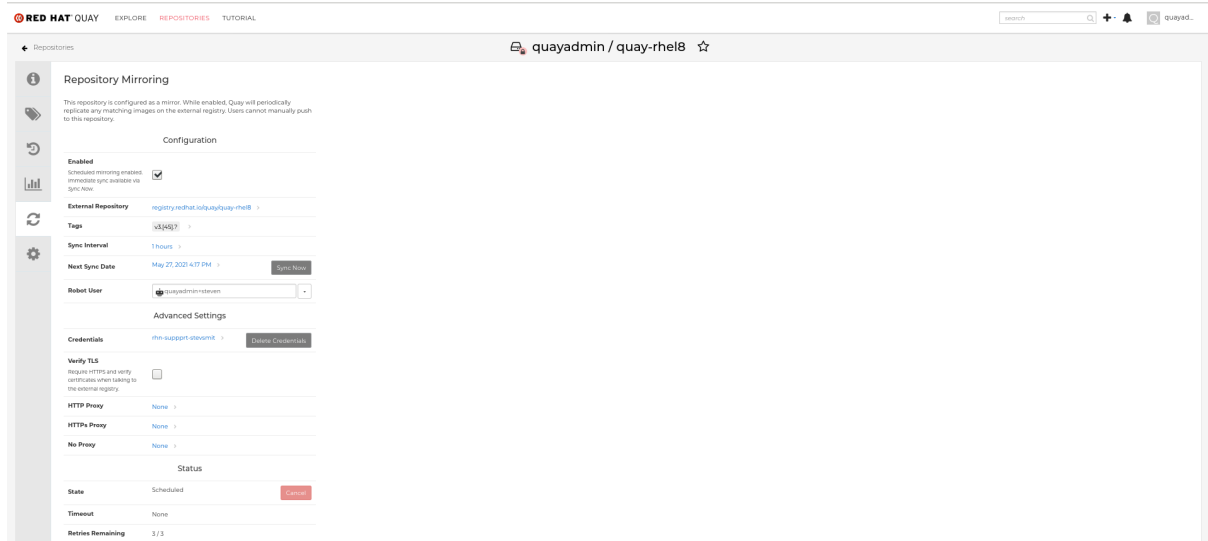


When the mirroring is complete, the images will appear in the Tags tab:

The screenshot shows the 'Repository Tags' interface for the repository 'quayuser / quay-rhel8'. It displays a list of tags with their respective details.

TAG	LAST MODIFIED	SIZE	EXPIRES	MANIFEST
<input type="checkbox"/> v3.5.1	a minute ago	N/A	Never	sha256:3d4e87d3275
<input type="checkbox"/> v3.5.0	a minute ago	N/A	Never	sha256:9559fae0c
<input type="checkbox"/> v3.4.4	a minute ago	N/A	Never	sha256:e5135a29e
<input type="checkbox"/> v3.4.3	a minute ago	N/A	Never	sha256:392f1331f2
<input type="checkbox"/> v3.4.2	a minute ago	N/A	Never	sha256:75833ba07
<input type="checkbox"/> v3.4.1	a minute ago	N/A	Never	sha256:4f85c5280a
<input type="checkbox"/> v3.4.0	a minute ago	N/A	Never	sha256:5e12d8880

Below is an example of a completed Repository Mirroring screen:



8.8. EVENT NOTIFICATIONS FOR MIRRORING

There are three notification events for repository mirroring:

- Repository Mirror Started
- Repository Mirror Success
- Repository Mirror Unsuccessful

The events can be configured inside of the **Settings** tab for each repository, and all existing notification methods such as email, Slack, Quay UI, and webhooks are supported.

8.9. MIRRORING TAG PATTERNS

At least one tag must be entered. The following table references possible image tag patterns.

8.9.1. Pattern syntax

Pattern	Description
*	Matches all characters
?	Matches any single character
[seq]	Matches any character in <i>seq</i>
[!seq]	Matches any character not in <i>seq</i>

8.9.2. Example tag patterns

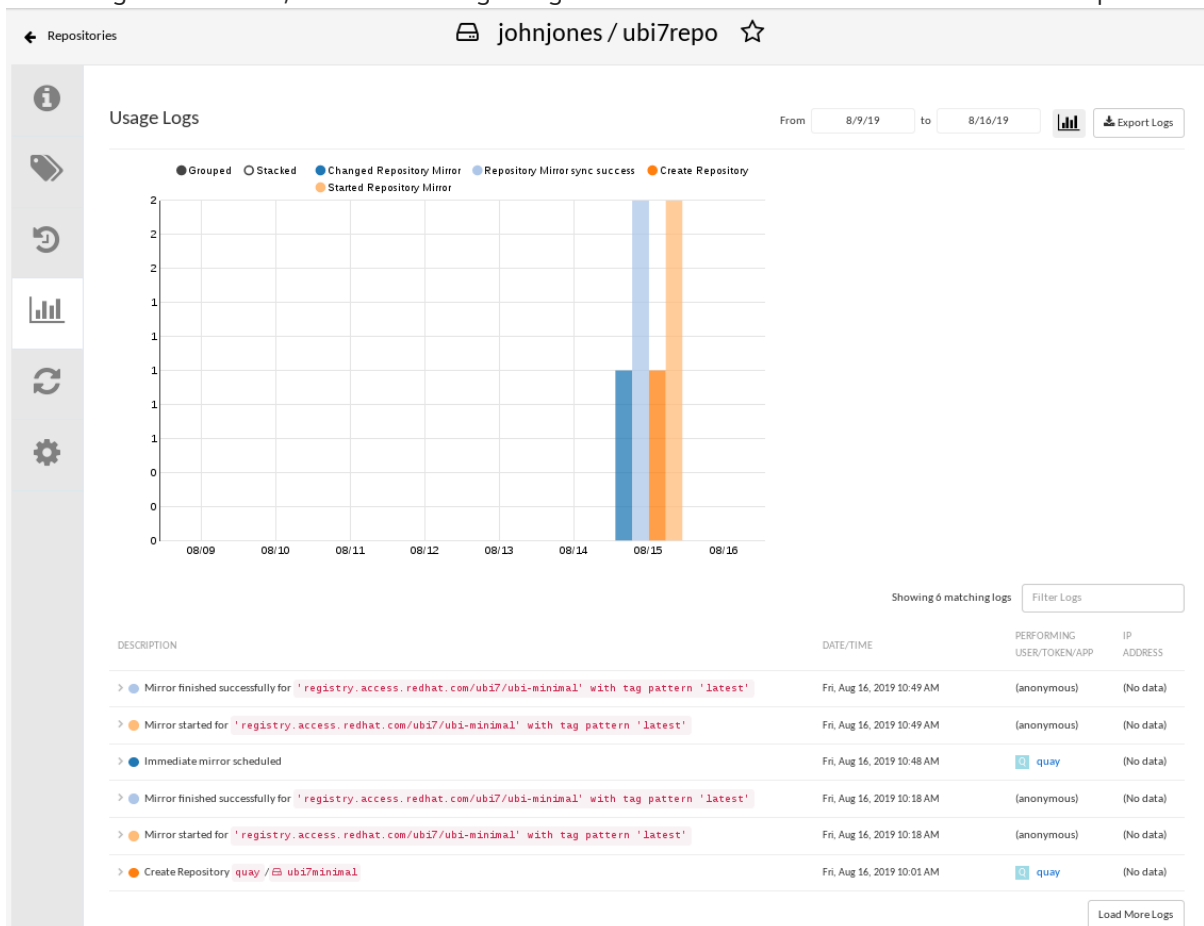
Example Pattern	Example Matches
v3*	v32, v3.1, v3.2, v3.2-4beta, v3.3

v3.*	v3.1, v3.2, v3.2-4beta
v3.?	v3.1, v3.2, v3.3
v3.[12]	v3.1, v3.2
v3.[12]*	v3.1, v3.2, v3.2-4beta
v3.[!1]*	v3.2, v3.2-4beta, v3.3

8.10. WORKING WITH MIRRORED REPOSITORIES

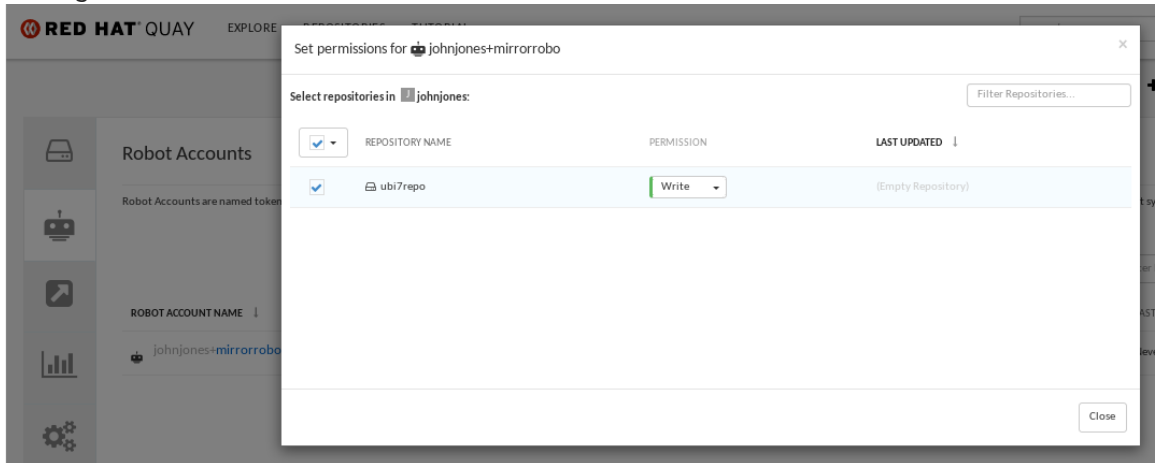
Once you have created a mirrored repository, there are several ways you can work with that repository. Select your mirrored repository from the Repositories page and do any of the following:

- **Enable/disable the repository.** Select the Mirroring button in the left column, then toggle the Enabled check box to enable or disable the repository temporarily.
- **Check mirror logs.** To make sure the mirrored repository is working properly, you can check the mirror logs. To do that, select the Usage Logs button in the left column. Here's an example:

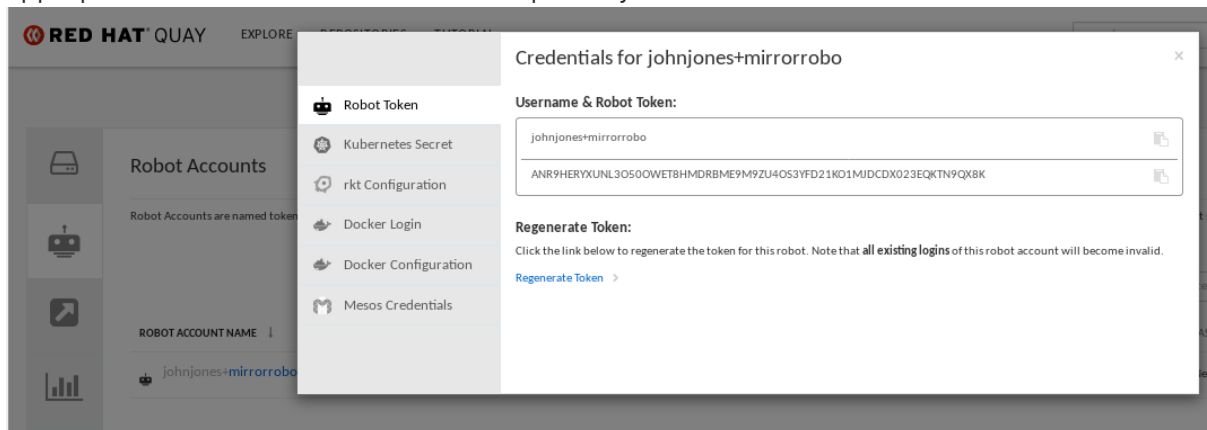


- **Sync mirror now.** To immediately sync the images in your repository, select the Sync Now button.
- **Change credentials:** To change the username and password, select DELETE from the Credentials line. Then select None and add the username and password needed to log into the external registry when prompted.

- **Cancel mirroring:** To stop mirroring, which keeps the current images available but stops new ones from being synced, select the CANCEL button.
- **Set robot permissions:** Red Hat Quay robot accounts are named tokens that hold credentials for accessing external repositories. By assigning credentials to a robot, that robot can be used across multiple mirrored repositories that need to access the same external registry. You can assign an existing robot to a repository by going to Account Settings, then selecting the Robot Accounts icon in the left column. For the robot account, choose the link under the REPOSITORIES column. From the pop-up window, you can:
 - Check which repositories are assigned to that robot.
 - Assign read, write or Admin privileges to that robot from the PERMISSION field shown in this figure:



- **Change robot credentials:** Robots can hold credentials such as Kubernetes secrets, Docker login information, and Mesos bundles. To change robot credentials, select the Options gear on the robot's account line on the Robot Accounts window and choose View Credentials. Add the appropriate credentials for the external repository the robot needs to access.



- **Check and change general setting:** Select the Settings button (gear icon) from the left column on the mirrored repository page. On the resulting page, you can change settings associated with the mirrored repository. In particular, you can change User and Robot Permissions, to specify exactly which users and robots can read from or write to the repo.

8.11. REPOSITORY MIRRORING RECOMMENDATIONS

Best practices for repository mirroring include the following:

- Repository mirroring pods can run on any node. This means that you can run mirroring on nodes where Red Hat Quay is already running.
- Repository mirroring is scheduled in the database and runs in batches. As a result, repository workers check each repository mirror configuration file and reads when the next sync needs to be. More mirror workers means more repositories can be mirrored at the same time. For example, running 10 mirror workers means that a user can run 10 mirroring operators in parallel. If a user only has 2 workers with 10 mirror configurations, only 2 operators can be performed.
- The optimal number of mirroring pods depends on the following conditions:
 - The total number of repositories to be mirrored
 - The number of images and tags in the repositories and the frequency of changes
 - Parallel batching
For example, if a user is mirroring a repository that has 100 tags, the mirror will be completed by one worker. Users must consider how many repositories one wants to mirror in parallel, and base the number of workers around that.

Multiple tags in the same repository cannot be mirrored in parallel.

CHAPTER 9. IPV6 AND DUAL-STACK DEPLOYMENTS

Your standalone Red Hat Quay deployment can now be served in locations that only support IPv6, such as Telco and Edge environments. Support is also offered for dual-stack networking so your Red Hat Quay deployment can listen on IPv4 and IPv6 simultaneously.

For a list of known limitations, see [IPv6 limitations](#)

9.1. ENABLING THE IPV6 PROTOCOL FAMILY

Use the following procedure to enable IPv6 support on your standalone Red Hat Quay deployment.

Prerequisites

- You have updated Red Hat Quay to 3.8.
- Your host and container software platform (Docker, Podman) must be configured to support IPv6.

Procedure

1. In your deployment's **config.yaml** file, add the **FEATURE_LISTEN_IP_VERSION** parameter and set it to **IPv6**, for example:

```
---
FEATURE_GOOGLE_LOGIN: false
FEATURE_INVITE_ONLY_USER_CREATION: false
FEATURE_LISTEN_IP_VERSION: IPv6
FEATURE_MAILING: false
FEATURE_NONSUPERUSER_TEAM_SYNCING_SETUP: false
---
```

2. Start, or restart, your Red Hat Quay deployment.
3. Check that your deployment is listening to IPv6 by entering the following command:

```
$ curl <quay_endpoint>/health/instance
{"data":{"services":
{"auth":true,"database":true,"disk_space":true,"registry_gunicorn":true,"service_key":true,"web_
gunicorn":true}}, "status_code":200}
```

After enabling IPv6 in your deployment's **config.yaml**, all Red Hat Quay features can be used as normal, so long as your environment is configured to use IPv6 and is not hindered by the `ipv6-limitations[current limitations]`.

**WARNING**

If your environment is configured to IPv4, but the **FEATURE_LISTEN_IP_VERSION** configuration field is set to **IPv6**, Red Hat Quay will fail to deploy.

9.2. ENABLING THE DUAL-STACK PROTOCOL FAMILY

Use the following procedure to enable dual-stack (IPv4 and IPv6) support on your standalone Red Hat Quay deployment.

Prerequisites

- You have updated Red Hat Quay to 3.8.
- Your host and container software platform (Docker, Podman) must be configured to support IPv6.

Procedure

1. In your deployment's **config.yaml** file, add the **FEATURE_LISTEN_IP_VERSION** parameter and set it to **dual-stack**, for example:

```
---
FEATURE_GOOGLE_LOGIN: false
FEATURE_INVITE_ONLY_USER_CREATION: false
FEATURE_LISTEN_IP_VERSION: dual-stack
FEATURE_MAILING: false
FEATURE_NONSUPERUSER_TEAM_SYNCING_SETUP: false
---
```

2. Start, or restart, your Red Hat Quay deployment.
3. Check that your deployment is listening to both channels by entering the following command:
 - a. For IPv4, enter the following command:

```
$ curl --ipv4 <quay_endpoint>
{"data":{"services":
{"auth":true,"database":true,"disk_space":true,"registry_gunicorn":true,"service_key":true,"
web_gunicorn":true}},"status_code":200}
```

- b. For IPv6, enter the following command:

```
$ curl --ipv6 <quay_endpoint>
{"data":{"services":
{"auth":true,"database":true,"disk_space":true,"registry_gunicorn":true,"service_key":true,"
web_gunicorn":true}},"status_code":200}
```

After enabling dual-stack in your deployment's **config.yaml**, all Red Hat Quay features can be used as normal, so long as your environment is configured for dual-stack.

9.3. IPV6 AND DUA-STACK LIMITATIONS

- Currently, attempting to configure your Red Hat Quay deployment with the common Azure Blob Storage configuration will not work on IPv6 single stack environments. Because the endpoint of Azure Blob Storage does not support IPv6, there is no workaround in place for this issue.
For more information, see [PROJQUAY-4433](#).
- Currently, attempting to configure your Red Hat Quay deployment with Amazon S3 CloudFront will not work on IPv6 single stack environments. Because the endpoint of Amazon S3 CloudFront does not support IPv6, there is no workaround in place for this issue.
For more information, see [PROJQUAY-4470](#).

CHAPTER 10. LDAP AUTHENTICATION SETUP FOR RED HAT QUAY

Lightweight Directory Access Protocol (LDAP) is an open, vendor-neutral, industry standard application protocol for accessing and maintaining distributed directory information services over an Internet Protocol (IP) network. Red Hat Quay supports using LDAP as an identity provider.

10.1. CONSIDERATIONS WHEN ENABLING LDAP

Prior to enabling LDAP for your Red Hat Quay deployment, you should consider the following.

Existing Red Hat Quay deployments

Conflicts between usernames can arise when you enable LDAP for an existing Red Hat Quay deployment that already has users configured. For example, one user, **alice**, was manually created in Red Hat Quay prior to enabling LDAP. If the username **alice** also exists in the LDAP directory, Red Hat Quay automatically creates a new user, **alice-1**, when **alice** logs in for the first time using LDAP. Red Hat Quay then automatically maps the LDAP credentials to the **alice** account. For consistency reasons, this might be erroneous for your Red Hat Quay deployment. It is recommended that you remove any potentially conflicting local account names from Red Hat Quay prior to enabling LDAP.

Manual User Creation and LDAP authentication

When Red Hat Quay is configured for LDAP, LDAP-authenticated users are automatically created in Red Hat Quay's database on first log in, if the configuration option **FEATURE_USER_CREATION** is set to **true**. If this option is set to **false**, the automatic user creation for LDAP users fails, and the user is not allowed to log in. In this scenario, the superuser needs to create the desired user account first. Conversely, if **FEATURE_USER_CREATION** is set to **true**, this also means that a user can still create an account from the Red Hat Quay login screen, even if there is an equivalent user in LDAP.

10.2. CONFIGURING LDAP FOR RED HAT QUAY

You can configure LDAP for Red Hat Quay by updating your **config.yaml** file directly and restarting your deployment. Use the following procedure as a reference when configuring LDAP for Red Hat Quay.

1. Update your **config.yaml** file directly to include the following relevant information:

```
# ...
AUTHENTICATION_TYPE: LDAP 1
# ...
LDAP_ADMIN_DN: uid=<name>,ou=Users,o=<organization_id>,dc=
<example_domain_component>,dc=com 2
LDAP_ADMIN_PASSWD: ABC123 3
LDAP_ALLOW_INSECURE_FALLBACK: false 4
LDAP_BASE_DN: 5
  - o=<organization_id>
  - dc=<example_domain_component>
  - dc=com
LDAP_EMAIL_ATTR: mail 6
LDAP_UID_ATTR: uid 7
LDAP_URI: ldap://<example_url>.com 8
LDAP_USER_FILTER: (memberof=cn=developers,ou=Users,dc=<domain_name>,dc=com)
9
LDAP_USER_RDN: 10
  - ou=<example_organization_unit>
```

```

- o=<organization_id>
- dc=<example_domain_component>
- dc=com
# ...

```

- 1 Required. Must be set to **LDAP**.
- 2 Required. The admin DN for LDAP authentication.
- 3 Required. The admin password for LDAP authentication.
- 4 Required. Whether to allow SSL/TLS insecure fallback for LDAP authentication.
- 5 Required. The base DN for LDAP authentication.
- 6 Required. The email attribute for LDAP authentication.
- 7 Required. The UID attribute for LDAP authentication.
- 8 Required. The LDAP URI.
- 9 Required. The user filter for LDAP authentication.
- 10 Required. The user RDN for LDAP authentication.

2. After you have added all required LDAP fields, save the changes and restart your Red Hat Quay deployment.

10.3. ENABLING THE LDAP_RESTRICTED_USER_FILTER CONFIGURATION FIELD

The **LDAP_RESTRICTED_USER_FILTER** configuration field is a subset of the **LDAP_USER_FILTER** configuration field. When configured, this option allows Red Hat Quay administrators the ability to configure LDAP users as restricted users when Red Hat Quay uses LDAP as its authentication provider.

Use the following procedure to enable LDAP restricted users on your Red Hat Quay deployment.

Prerequisites

- Your Red Hat Quay deployment uses LDAP as its authentication provider.
- You have configured the **LDAP_USER_FILTER** field in your **config.yaml** file.

Procedure

1. In your deployment's **config.yaml** file, add the **LDAP_RESTRICTED_USER_FILTER** parameter and specify the group of restricted users, for example, **members**:

```

# ...
AUTHENTICATION_TYPE: LDAP
# ...
LDAP_ADMIN_DN: uid=<name>,ou=Users,o=<organization_id>,dc=
<example_domain_component>,dc=com
LDAP_ADMIN_PASSWD: ABC123

```

```

LDAP_ALLOW_INSECURE_FALLBACK: false
LDAP_BASE_DN:
  - o=<organization_id>
  - dc=<example_domain_component>
  - dc=com
LDAP_EMAIL_ATTR: mail
LDAP_UID_ATTR: uid
LDAP_URI: ldap://<example_url>.com
LDAP_USER_FILTER: (memberof=cn=developers,ou=Users,o=
<example_organization_unit>,dc=<example_domain_component>,dc=com)
LDAP_RESTRICTED_USER_FILTER: (<filterField>=<value>) ❶
LDAP_USER_RDN:
  - ou=<example_organization_unit>
  - o=<organization_id>
  - dc=<example_domain_component>
  - dc=com
# ...

```

❶ Configures specified users as restricted users.

2. Start, or restart, your Red Hat Quay deployment.

After enabling the **LDAP_RESTRICTED_USER_FILTER** feature, your LDAP Red Hat Quay users are restricted from reading and writing content, and creating organizations.

10.4. ENABLING THE LDAP_SUPERUSER_FILTER CONFIGURATION FIELD

With the **LDAP_SUPERUSER_FILTER** field configured, Red Hat Quay administrators can configure Lightweight Directory Access Protocol (LDAP) users as superusers if Red Hat Quay uses LDAP as its authentication provider.

Use the following procedure to enable LDAP superusers on your Red Hat Quay deployment.

Prerequisites

- Your Red Hat Quay deployment uses LDAP as its authentication provider.
- You have configured the **LDAP_USER_FILTER** field in your **config.yaml** file.

Procedure

1. In your deployment's **config.yaml** file, add the **LDAP_SUPERUSER_FILTER** parameter and add the group of users you want configured as super users, for example, **root**:

```

# ...
AUTHENTICATION_TYPE: LDAP
# ...
LDAP_ADMIN_DN: uid=<name>,ou=Users,o=<organization_id>,dc=
<example_domain_component>,dc=com
LDAP_ADMIN_PASSWD: ABC123
LDAP_ALLOW_INSECURE_FALLBACK: false
LDAP_BASE_DN:
  - o=<organization_id>

```

```

- dc=<example_domain_component>
- dc=com
LDAP_EMAIL_ATTR: mail
LDAP_UID_ATTR: uid
LDAP_URI: ldap://<example_url>.com
LDAP_USER_FILTER: (memberof=cn=developers,ou=Users,o=
<example_organization_unit>,dc=<example_domain_component>,dc=com)
LDAP_SUPERUSER_FILTER: (<filterField>=<value>) 1
LDAP_USER_RDN:
- ou=<example_organization_unit>
- o=<organization_id>
- dc=<example_domain_component>
- dc=com
# ...

```

1 Configures specified users as superusers.

2. Start, or restart, your Red Hat Quay deployment.

After enabling the **LDAP_SUPERUSER_FILTER** feature, your LDAP Red Hat Quay users have superuser privileges. The following options are available to superusers:

- Manage users
- Manage organizations
- Manage service keys
- View the change log
- Query the usage logs
- Create globally visible user messages

10.5. COMMON LDAP CONFIGURATION ISSUES

The following errors might be returned with an invalid configuration.

- **Invalid credentials.** If you receive this error, the Administrator DN or Administrator DN password values are incorrect. Ensure that you are providing accurate Administrator DN and password values.
- ***Verification of superuser %USERNAME% failed** This error is returned for the following reasons:
 - The username has not been found.
 - The user does not exist in the remote authentication system.
 - LDAP authorization is configured improperly.
- **Cannot find the current logged in user** When configuring LDAP for Red Hat Quay, there may be situations where the LDAP connection is established successfully using the username and password provided in the **Administrator DN** fields. However, if the current logged-in user cannot be found within the specified **User Relative DN** path using the **UID Attribute** or **Mail Attribute** fields, there are typically two potential reasons for this:

- The current logged in user does not exist in the **User Relative DN** path.
- The **Administrator DN** does not have rights to search or read the specified LDAP path. To fix this issue, ensure that the logged in user is included in the **User Relative DN** path, or provide the correct permissions to the **Administrator DN** account.

10.6. LDAP CONFIGURATION FIELDS

For a full list of LDAP configuration fields, see [LDAP configuration fields](#)

CHAPTER 11. CONFIGURING OIDC FOR RED HAT QUAY

Configuring OpenID Connect (OIDC) for Red Hat Quay can provide several benefits to your deployment. For example, OIDC allows users to authenticate to Red Hat Quay using their existing credentials from an OIDC provider, such as [Red Hat Single Sign-On](#), Google, Github, Microsoft, or others. Other benefits of OIDC include centralized user management, enhanced security, and single sign-on (SSO). Overall, OIDC configuration can simplify user authentication and management, enhance security, and provide a seamless user experience for Red Hat Quay users.

The following procedures show you how to configure Microsoft Entra ID on a standalone deployment of Red Hat Quay, and how to configure Red Hat Single Sign-On on an Operator-based deployment of Red Hat Quay. These procedures are interchangeable depending on your deployment type.



NOTE

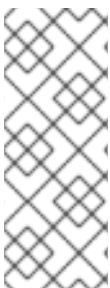
By following these procedures, you will be able to add any OIDC provider to Red Hat Quay, regardless of which identity provider you choose to use.

11.1. CONFIGURING MICROSOFT ENTRA ID OIDC ON A STANDALONE DEPLOYMENT OF RED HAT QUAY

By integrating Microsoft Entra ID authentication with Red Hat Quay, your organization can take advantage of the centralized user management and security features offered by Microsoft Entra ID. Some features include the ability to manage user access to Red Hat Quay repositories based on their Microsoft Entra ID roles and permissions, and the ability to enable multi-factor authentication and other security features provided by Microsoft Entra ID.

Azure Active Directory (Microsoft Entra ID) authentication for Red Hat Quay allows users to authenticate and access Red Hat Quay using their Microsoft Entra ID credentials.

Use the following procedure to configure Microsoft Entra ID by updating the Red Hat Quay **config.yaml** file directly.



PROCEDURE

- Using the following procedure, you can add any ODIC provider to Red Hat Quay, regardless of which identity provider is being added.
- If your system has a firewall in use, or proxy enabled, you must whitelist all Azure API endpoints for each Oauth application that is created. Otherwise, the following error is returned: **x509: certificate signed by unknown authority**.

1. Use the following reference and update your **config.yaml** file with your desired OIDC provider's credentials:

```
# ...
AZURE_LOGIN_CONFIG: 1
  CLIENT_ID: <client_id> 2
  CLIENT_SECRET: <client_secret> 3
  OIDC_SERVER: <oidc_server_address_> 4
  SERVICE_NAME: Microsoft Entra ID 5
  VERIFIED_EMAIL_CLAIM_NAME: <verified_email> 6
# ...
```


- 1 The parent key that holds the OIDC configuration settings. In this example, the parent key used is **AZURE_LOGIN_CONFIG**, however, the string **AZURE** can be replaced with any arbitrary string based on your specific needs, for example **ABC123**. However, the following strings are not accepted: **GOOGLE**, **GITHUB**. These strings are reserved for their respective identity platforms and require a specific **config.yaml** entry contingent upon when platform you are using.
 - 2 The client ID of the application that is being registered with the identity provider.
 - 3 The client secret of the application that is being registered with the identity provider.
 - 4 The address of the OIDC server that is being used for authentication. In this example, you must use **sts.windows.net** as the issuer identifier. Using <https://login.microsoftonline.com> results in the following error: **Could not create provider for AzureAD. Error: oidc: issuer did not match the issuer returned by provider, expected "https://login.microsoftonline.com/73f2e714-xxxx-xxxx-xxxx-dffe1df8a5d5" got "https://sts.windows.net/73f2e714-xxxx-xxxx-xxxx-dffe1df8a5d5/"**.
 - 5 The name of the service that is being authenticated.
 - 6 The name of the claim that is used to verify the email address of the user.
2. Proper configuration of Microsoft Entra ID results three redirects with the following format:
 - https://QUAY_HOSTNAME/oauth2/<name_of_service>/callback
 - https://QUAY_HOSTNAME/oauth2/<name_of_service>/callback/attach
 - https://QUAY_HOSTNAME/oauth2/<name_of_service>/callback/cli
 3. Restart your Red Hat Quay deployment.

11.2. CONFIGURING RED HAT SINGLE SIGN-ON FOR RED HAT QUAY

Based on the Keycloak project, Red Hat Single Sign-On (RH-SSO) is an open source identity and access management (IAM) solution provided by Red Hat. RH-SSO allows organizations to manage user identities, secure applications, and enforce access control policies across their systems and applications. It also provides a unified authentication and authorization framework, which allows users to log in one time and gain access to multiple applications and resources without needing to re-authenticate. For more information, see [Red Hat Single Sign-On](#).

By configuring Red Hat Single Sign-On on Red Hat Quay, you can create a seamless authentication integration between Red Hat Quay and other application platforms like OpenShift Container Platform.

11.2.1. Configuring the Red Hat Single Sign-On Operator for use with the Red Hat Quay Operator

Use the following procedure to configure Red Hat Single Sign-On for the Red Hat Quay Operator on OpenShift Container Platform.

Prerequisites

- You have set up the Red Hat Single Sign-On Operator. For more information, see [Red Hat Single Sign-On Operator](#).

- You have configured [SSL/TLS for your Red Hat Quay on OpenShift Container Platform deployment](#) and for Red Hat Single Sign-On.
- You have generated a single Certificate Authority (CA) and uploaded it to your Red Hat Single Sign-On Operator and to your Red Hat Quay configuration.

Procedure

1. Navigate to the Red Hat Single Sign-On **Admin Console**.
 - a. On the OpenShift Container Platform **Web Console**, navigate to **Network** → **Route**.
 - b. Select the **Red Hat Single Sign-On** project from the drop-down list.
 - c. Find the Red Hat Single Sign-On **Admin Console** in the **Routes** table.
2. Select the Realm that you will use to configure Red Hat Quay.
3. Click **Clients** under the **Configure** section of the navigation panel, and then click the **Create** button to add a new OIDC for Red Hat Quay.
4. Enter the following information.
 - **Client ID:** `quay-enterprise`
 - **Client Protocol:** `openid-connect`
 - **Root URL:** `https://<quay_endpoint>/`
5. Click **Save**. This results in a redirect to the **Clients** setting panel.
6. Navigate to **Access Type** and select **Confidential**.
7. Navigate to **Valid Redirect URIs**. You must provide three redirect URIs. The value should be the fully qualified domain name of the Red Hat Quay registry appended with `/oauth2/redhatsso/callback`. For example:
 - `https://<quay_endpoint>/oauth2/redhatsso/callback`
 - `https://<quay_endpoint>/oauth2/redhatsso/callback/attach`
 - `https://<quay_endpoint>/oauth2/redhatsso/callback/cli`
8. Click **Save** and navigate to the new **Credentials** setting.
9. Copy the value of the Secret.

11.2.1.1. Configuring the Red Hat Quay Operator to use Red Hat Single Sign-On

Use the following procedure to configure Red Hat Single Sign-On with the Red Hat Quay Operator.

Prerequisites

- You have set up the Red Hat Single Sign-On Operator. For more information, see [Red Hat Single Sign-On Operator](#).

- You have configured [SSL/TLS for your Red Hat Quay on OpenShift Container Platform deployment](#) and for Red Hat Single Sign-On.
- You have generated a single Certificate Authority (CA) and uploaded it to your Red Hat Single Sign-On Operator and to your Red Hat Quay configuration.

Procedure

1. Edit your Red Hat Quay **config.yaml** file by navigating to **Operators** → **Installed Operators** → **Red Hat Quay** → **Quay Registry** → **Config Bundle Secret**. Then, click **Actions** → **Edit Secret**. Alternatively, you can update the **config.yaml** file locally.
2. Add the following information to your Red Hat Quay on OpenShift Container Platform **config.yaml** file:

```
# ...
RHSSO_LOGIN_CONFIG: 1
  CLIENT_ID: <client_id> 2
  CLIENT_SECRET: <client_secret> 3
  OIDC_SERVER: <oidc_server_url> 4
  SERVICE_NAME: <service_name> 5
  SERVICE_ICON: <service_icon> 6
  VERIFIED_EMAIL_CLAIM_NAME: <example_email_address> 7
  PREFERRED_USERNAME_CLAIM_NAME: <preferred_username> 8
  LOGIN_SCOPES: 9
    - 'openid'
# ...
```

- 1 The parent key that holds the OIDC configuration settings. In this example, the parent key used is **AZURE_LOGIN_CONFIG**, however, the string **AZURE** can be replaced with any arbitrary string based on your specific needs, for example **ABC123**. However, the following strings are not accepted: **GOOGLE**, **GITHUB**. These strings are reserved for their respective identity platforms and require a specific **config.yaml** entry contingent upon when platform you are using.
- 2 The client ID of the application that is being registered with the identity provider, for example, **quay-enterprise**.
- 3 The Client Secret from the **Credentials** tab of the **quay-enterprise** OIDC client settings.
- 4 The fully qualified domain name (FQDN) of the Red Hat Single Sign-On instance, appended with **/auth/realms/** and the Realm name. You must include the forward slash at the end, for example, **https://sso-redhat.example.com//auth/realms/<keycloak_realm_name>/**.
- 5 The name that is displayed on the Red Hat Quay login page, for example, **Red hat Single Sign On**.
- 6 Changes the icon on the login screen. For example, **/static/img/RedHat.svg**.
- 7 The name of the claim that is used to verify the email address of the user.
- 8 The name of the claim that is used to verify the email address of the user.
- 9 The scopes to send to the OIDC provider when performing the login flow, for example,

- Restart your Red Hat Quay on OpenShift Container Platform deployment with Red Hat Single Sign-On enabled.

11.3. TEAM SYNCHRONIZATION FOR RED HAT QUAY OIDC DEPLOYMENTS

Administrators can leverage an OpenID Connect (OIDC) identity provider that supports group or team synchronization to apply repository permissions to sets of users in Red Hat Quay. This allows administrators to avoid having to manually create and sync group definitions between Red Hat Quay and the OIDC group.

11.3.1. Enabling synchronization for Red Hat Quay OIDC deployments

Use the following procedure to enable team synchronization when your Red Hat Quay deployment uses an OIDC authenticator.



IMPORTANT

The following procedure does not use a specific OIDC provider. Instead, it provides a general outline of how best to approach team synchronization between an OIDC provider and Red Hat Quay. Any OIDC provider can be used to enable team synchronization, however, setup might vary depending on your provider.

Procedure

- Update your **config.yaml** file with the following information:

```
AUTHENTICATION_TYPE: OIDC
# ...
OIDC_LOGIN_CONFIG:
  CLIENT_ID: 1
  CLIENT_SECRET: 2
  OIDC_SERVER: 3
  SERVICE_NAME: 4
  PREFERRED_GROUP_CLAIM_NAME: 5
  LOGIN_SCOPES: ['openid', '<example_scope>'] 6
  OIDC_DISABLE_USER_ENDPOINT: false 7
# ...
FEATURE_TEAM_SYNCING: true 8
FEATURE_NONSUPERUSER_TEAM_SYNCING_SETUP: true 9
FEATURE_UI_V2: true
# ...
```

- Required. The registered OIDC client ID for this Red Hat Quay instance.
- Required. The registered OIDC client secret for this Red Hat Quay instance.
- Required. The address of the OIDC server that is being used for authentication. This URL should be such that a **GET** request to **<OIDC_SERVER>/well-known/openid-configuration** returns the provider's configuration information. This configuration information is essential for the relying party (RP) to interact securely with the OpenID Connect provider and obtain necessary details for authentication and authorization processes.

- 4 Required. The name of the service that is being authenticated.
- 5 Required. The key name within the OIDC token payload that holds information about the user's group memberships. This field allows the authentication system to extract group membership information from the OIDC token so that it can be used with Red Hat Quay.
- 6 Required. Adds additional scopes that Red Hat Quay uses to communicate with the OIDC provider. Must include **'openid'**. Additional scopes are optional.
- 7 Whether to allow or disable the **/userinfo** endpoint. If using Azure Entra ID, set this field to **true**. Defaults to **false**.
- 8 Required. Whether to allow for team membership to be synced from a backing group in the authentication engine.
- 9 Optional. If enabled, non-superusers can setup team synchronization.

2. Restart your Red Hat Quay registry.

11.3.2. Setting up your Red Hat Quay deployment for team synchronization

1. Log in to your Red Hat Quay registry via your OIDC provider.
2. On the Red Hat Quay v2 UI dashboard, click **Create Organization**.
3. Enter an Organization name, for example, **test-org**.
4. Click the name of the Organization.
5. In the navigation pane, click **Teams and membership**.
6. Click **Create new team** and enter a name, for example, **testteam**.
7. On the **Create team** pop-up:
 - a. Optional. Add this team to a repository.
 - b. Add a team member, for example, **user1**, by typing in the user's account name.
 - c. Add a robot account to this team. This page provides the option to create a robot account.
8. Click **Next**.
9. On the **Review and Finish** page, review the information that you have provided and click **Review and Finish**.
10. To enable team synchronization for your Red Hat Quay OIDC deployment, click **Enable Directory Sync** on the **Teams and membership** page.
11. You are prompted to enter the group Object ID if your OIDC authenticator is Azure Entra ID, or the group name if using a different provider. Note the message in the popup:



WARNING

Please note that once team syncing is enabled, the membership of users who are already part of the team will be revoked. OIDC group will be the single source of truth. This is a non-reversible action. Team's user membership from within Quay will be ready-only.

12. Click **Enable Sync**.
13. You are returned to the **Teams and membership** page. Note that users of this team are removed and are re-added upon logging back in. At this stage, only the robot account is still part of the team.
A banner at the top of the page confirms that the team is synced:

This team is synchronized with a group in OIDC and its user membership is therefore read-only.

By clicking the **Directory Synchronization Config** accordion, the OIDC group that your deployment is synchronized with appears.

14. Log out of your Red Hat Quay registry and continue on to the verification steps.

Verification

Use the following verification procedure to ensure that **user1** appears as a member of the team.

1. Log back in to your Red Hat Quay registry.
2. Click **Organizations** → **test-org** → **test-team Teams and memberships** **user1** now appears as a team member for this team.

Verification

Use the following procedure to remove **user1** from a group via your OIDC provider, and subsequently remove them from the team on Red Hat Quay.

1. Navigate to your OIDC provider's administration console.
2. Navigate to the **Users** page of your OIDC provider. The name of this page varies depending on your provider.
3. Click the name of the user associated with Red Hat Quay, for example, **user1**.
4. Remove the user from group in the configured identity provider.
5. Remove, or unassign, the access permissions from the user.
6. Log in to your Red Hat Quay registry.
7. Click **Organizations** → **test-org** → **test-team Teams and memberships** **user1** has been removed from this team.

CHAPTER 12. CONFIGURING AWS STS FOR RED HAT QUAY

Support for Amazon Web Services (AWS) Security Token Service (STS) is available for standalone Red Hat Quay deployments and Red Hat Quay on OpenShift Container Platform. AWS STS is a web service for requesting temporary, limited-privilege credentials for AWS Identity and Access Management (IAM) users and for users that you authenticate, or *federated users*. This feature is useful for clusters using Amazon S3 as an object storage, allowing Red Hat Quay to use STS protocols to authenticate with Amazon S3, which can enhance the overall security of the cluster and help to ensure that access to sensitive data is properly authenticated and authorized.

Configuring AWS STS is a multi-step process that requires creating an AWS IAM user, creating an S3 role, and configuring your Red Hat Quay **config.yaml** file to include the proper resources.

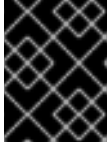
Use the following procedures to configure AWS STS for Red Hat Quay.

12.1. CREATING AN IAM USER

Use the following procedure to create an IAM user.

Procedure

1. Log in to the Amazon Web Services (AWS) console and navigate to the Identity and Access Management (IAM) console.
2. In the navigation pane, under **Access management** click **Users**.
3. Click **Create User** and enter the following information:
 - a. Enter a valid username, for example, **quay-user**.
 - b. For **Permissions options**, click **Add user to group**.
4. On the **review and create** page, click **Create user**. You are redirected to the **Users** page.
5. Click the username, for example, **quay-user**.
6. Copy the ARN of the user, for example, **arn:aws:iam::123492922789:user/quay-user**.
7. On the same page, click the **Security credentials** tab.
8. Navigate to **Access keys**.
9. Click **Create access key**.
10. On the **Access key best practices & alternatives** page, click **Command Line Interface (CLI)**, then, check the confirmation box. Then click **Next**.
11. Optional. On the **Set description tag - optional** page, enter a description.
12. Click **Create access key**.
13. Copy and store the access key and the secret access key.



IMPORTANT

This is the only time that the secret access key can be viewed or downloaded. You cannot recover it later. However, you can create a new access key any time.

14. Click **Done**.

12.2. CREATING AN S3 ROLE

Use the following procedure to create an S3 role for AWS STS.

Prerequisites

- You have created an IAM user and stored the access key and the secret access key.

Procedure

1. If you are not already, navigate to the IAM dashboard by clicking **Dashboard**.
2. In the navigation pane, click **Roles** under **Access management**.
3. Click **Create role**.
 - Click **Custom Trust Policy**, which shows an editable JSON policy. By default, it shows the following information:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Statement1",
      "Effect": "Allow",
      "Principal": {},
      "Action": "sts:AssumeRole"
    }
  ]
}
```

4. Under the **Principal** configuration field, add your AWS ARN information. For example:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Statement1",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::123492922789:user/quay-user"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```


5. Click **Next**.
6. On the **Add permissions** page, type **AmazonS3FullAccess** in the search box. Check the box to add that policy to the S3 role, then click **Next**.
7. On the **Name, review, and create** page, enter the following information:
 - a. Enter a role name, for example, **example-role**.
 - b. Optional. Add a description.
8. Click the **Create role** button. You are navigated to the **Roles** page. Under **Role name**, the newly created S3 should be available.

12.3. CONFIGURING RED HAT QUAY TO USE AWS STS

Use the following procedure to edit your Red Hat Quay **config.yaml** file to use AWS STS.

Procedure

1. Update your **config.yaml** file for Red Hat Quay to include the following information:

```
# ...
DISTRIBUTED_STORAGE_CONFIG:
  default:
    - STSS3Storage
    - sts_role_arn: <role_arn> 1
      s3_bucket: <s3_bucket_name> 2
      storage_path: <storage_path> 3
      sts_user_access_key: <s3_user_access_key> 4
      sts_user_secret_key: <s3_user_secret_key> 5
# ...
```

- 1 The unique Amazon Resource Name (ARN) required when configuring AWS STS
- 2 The name of your s3 bucket.
- 3 The storage path for data. Usually **/datastorage**.
- 4 The generated AWS S3 user access key required when configuring AWS STS.
- 5 The generated AWS S3 user secret key required when configuring AWS STS.

2. Restart your Red Hat Quay deployment.

Verification

1. Tag a sample image, for example, **busybox**, that will be pushed to the repository. For example:

```
$ podman tag docker.io/library/busybox <quay-
server.example.com>/<organization_name>/busybox:test
```

2. Push the sample image by running the following command:

■

█ \$ podman push <quay-server.example.com>/<organization_name>/busybox:test

3. Verify that the push was successful by navigating to the Organization that you pushed the image to in your Red Hat Quay registry → **Tags**.
4. Navigate to the Amazon Web Services (AWS) console and locate your s3 bucket.
5. Click the name of your s3 bucket.
6. On the **Objects** page, click **datastorage/**.
7. On the **datastorage/** page, the following resources should be seen:
 - **sha256/**
 - **uploads/**
These resources indicate that the push was successful, and that AWS STS is properly configured.

CHAPTER 13. PROMETHEUS AND GRAFANA METRICS UNDER RED HAT QUAY

Red Hat Quay exports a [Prometheus](#)- and Grafana-compatible endpoint on each instance to allow for easy monitoring and alerting.

13.1. EXPOSING THE PROMETHEUS ENDPOINT

13.1.1. Standalone Red Hat Quay

When using **podman run** to start the **Quay** container, expose the metrics port **9091**:

```
$ sudo podman run -d --rm -p 80:8080 -p 443:8443 -p 9091:9091 \
  --name=quay \
  -v $QUAY/config:/conf/stack:Z \
  -v $QUAY/storage:/datastorage:Z \
  registry.redhat.io/quay/quay-rhel8:v3.11.1
```

The metrics will now be available:

```
$ curl quay.example.com:9091/metrics
```

See [Monitoring Quay with Prometheus and Grafana](#) for details on configuring Prometheus and Grafana to monitor Quay repository counts.

13.1.2. Red Hat Quay Operator

Determine the cluster IP for the **quay-metrics** service:

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

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)
example-registry-clair-app	ClusterIP	172.30.61.161	<none>	80/TCP,8089/TCP
example-registry-clair-postgres	ClusterIP	172.30.122.136	<none>	5432/TCP
example-registry-quay-app	ClusterIP	172.30.72.79	<none>	443/TCP,80/TCP,8081/TCP,55443/TCP
example-registry-quay-config-editor	ClusterIP	172.30.185.61	<none>	80/TCP
example-registry-quay-database	ClusterIP	172.30.114.192	<none>	5432/TCP
example-registry-quay-metrics	ClusterIP	172.30.37.76	<none>	9091/TCP
example-registry-quay-redis	ClusterIP	172.30.157.248	<none>	6379/TCP

Connect to your cluster and access the metrics using the cluster IP and port for the **quay-metrics** service:

```
$ oc debug node/master-0
```

```
sh-4.4# curl 172.30.37.76:9091/metrics

# HELP go_gc_duration_seconds A summary of the pause duration of garbage collection cycles.
# TYPE go_gc_duration_seconds summary
go_gc_duration_seconds{quantile="0"} 4.0447e-05
go_gc_duration_seconds{quantile="0.25"} 6.2203e-05
...
```

13.1.3. Setting up Prometheus to consume metrics

Prometheus needs a way to access all Red Hat Quay instances running in a cluster. In the typical setup, this is done by listing all the Red Hat Quay instances in a single named DNS entry, which is then given to Prometheus.

13.1.4. DNS configuration under Kubernetes

A simple [Kubernetes service](#) can be configured to provide the DNS entry for Prometheus.

13.1.5. DNS configuration for a manual cluster

[SkyDNS](#) is a simple solution for managing this DNS record when not using Kubernetes. SkyDNS can run on an [etcd](#) cluster. Entries for each Red Hat Quay instance in the cluster can be added and removed in the etcd store. SkyDNS will regularly read them from there and update the list of Quay instances in the DNS record accordingly.

13.2. INTRODUCTION TO METRICS

Red Hat Quay provides metrics to help monitor the registry, including metrics for general registry usage, uploads, downloads, garbage collection, and authentication.

13.2.1. General registry statistics

General registry statistics can indicate how large the registry has grown.

Metric name	Description
quay_user_rows	Number of users in the database
quay_robot_rows	Number of robot accounts in the database
quay_org_rows	Number of organizations in the database
quay_repository_rows	Number of repositories in the database
quay_security_scanning_unscanned_images_remainin g_total	Number of images that are not scanned by the latest security scanner

Sample metrics output

```
# HELP quay_user_rows number of users in the database
```

```

# TYPE quay_user_rows gauge
quay_user_rows{host="example-registry-quay-app-6df87f7b66-9tfn6",instance="",job="quay",pid="65",process_name="globalpromstats.py"} 3

# HELP quay_robot_rows number of robot accounts in the database
# TYPE quay_robot_rows gauge
quay_robot_rows{host="example-registry-quay-app-6df87f7b66-9tfn6",instance="",job="quay",pid="65",process_name="globalpromstats.py"} 2

# HELP quay_org_rows number of organizations in the database
# TYPE quay_org_rows gauge
quay_org_rows{host="example-registry-quay-app-6df87f7b66-9tfn6",instance="",job="quay",pid="65",process_name="globalpromstats.py"} 2

# HELP quay_repository_rows number of repositories in the database
# TYPE quay_repository_rows gauge
quay_repository_rows{host="example-registry-quay-app-6df87f7b66-9tfn6",instance="",job="quay",pid="65",process_name="globalpromstats.py"} 4

# HELP quay_security_scanning_unscanned_images_remaining number of images that are not scanned by the latest security scanner
# TYPE quay_security_scanning_unscanned_images_remaining gauge
quay_security_scanning_unscanned_images_remaining{host="example-registry-quay-app-6df87f7b66-9tfn6",instance="",job="quay",pid="208",process_name="secscan:application"} 5

```

13.2.2. Queue items

The *queue items* metrics provide information on the multiple queues used by Quay for managing work.

Metric name	Description
quay_queue_items_available	Number of items in a specific queue
quay_queue_items_locked	Number of items that are running
quay_queue_items_available_unlocked	Number of items that are waiting to be processed

Metric labels

- **queue_name:** The name of the queue. One of:
 - **exportactionlogs:** Queued requests to export action logs. These logs are then processed and put in storage. A link is then sent to the requester via email.
 - **namespacegc:** Queued namespaces to be garbage collected
 - **notification:** Queue for repository notifications to be sent out
 - **repositorygc:** Queued repositories to be garbage collected
 - **secscanv4:** Notification queue specific for Clair V4
 - **dockerfilebuild:** Queue for Quay docker builds

- **imagestoragereplication**: Queued blob to be replicated across multiple storages
- **chunk_cleanup**: Queued blob segments that needs to be deleted. This is only used by some storage implementations, for example, Swift.

For example, the queue labelled **repositorygc** contains the repositories marked for deletion by the repository garbage collection worker. For metrics with a **queue_name** label of **repositorygc**:

- **quay_queue_items_locked** is the number of repositories currently being deleted.
- **quay_queue_items_available_unlocked** is the number of repositories waiting to get processed by the worker.

Sample metrics output

```
# HELP quay_queue_items_available number of queue items that have not expired
# TYPE quay_queue_items_available gauge
quay_queue_items_available{host="example-registry-quay-app-6df87f7b66-9tfn6",instance="",job="quay",pid="63",process_name="exportactionlogsworker.py",queue_name="exportactionlogs"} 0
...

# HELP quay_queue_items_available_unlocked number of queue items that have not expired and are not locked
# TYPE quay_queue_items_available_unlocked gauge
quay_queue_items_available_unlocked{host="example-registry-quay-app-6df87f7b66-9tfn6",instance="",job="quay",pid="63",process_name="exportactionlogsworker.py",queue_name="exportactionlogs"} 0
...

# HELP quay_queue_items_locked number of queue items that have been acquired
# TYPE quay_queue_items_locked gauge
quay_queue_items_locked{host="example-registry-quay-app-6df87f7b66-9tfn6",instance="",job="quay",pid="63",process_name="exportactionlogsworker.py",queue_name="exportactionlogs"} 0
```

13.2.3. Garbage collection metrics

These metrics show you how many resources have been removed from garbage collection (gc). They show many times the gc workers have run and how many namespaces, repositories, and blobs were removed.

Metric name	Description
quay_gc_iterations_total	Number of iterations by the GCWorker
quay_gc_namespaces_purged_total	Number of namespaces purged by the NamespaceGCWorker
quay_gc_repos_purged_total	Number of repositories purged by the RepositoryGCWorker or NamespaceGCWorker
quay_gc_storage_blobs_deleted_total	Number of storage blobs deleted

Sample metrics output

```
# TYPE quay_gc_iterations_created gauge
quay_gc_iterations_created{host="example-registry-quay-app-6df87f7b66-
9tfn6",instance="",job="quay",pid="208",process_name="secscan:application"}
1.6317823190189714e+09
...

# HELP quay_gc_iterations_total number of iterations by the GCWorker
# TYPE quay_gc_iterations_total counter
quay_gc_iterations_total{host="example-registry-quay-app-6df87f7b66-
9tfn6",instance="",job="quay",pid="208",process_name="secscan:application"} 0
...

# TYPE quay_gc_namespaces_purged_created gauge
quay_gc_namespaces_purged_created{host="example-registry-quay-app-6df87f7b66-
9tfn6",instance="",job="quay",pid="208",process_name="secscan:application"}
1.6317823190189433e+09
...

# HELP quay_gc_namespaces_purged_total number of namespaces purged by the
NamespaceGCWorker
# TYPE quay_gc_namespaces_purged_total counter
quay_gc_namespaces_purged_total{host="example-registry-quay-app-6df87f7b66-
9tfn6",instance="",job="quay",pid="208",process_name="secscan:application"} 0
....

# TYPE quay_gc_repos_purged_created gauge
quay_gc_repos_purged_created{host="example-registry-quay-app-6df87f7b66-
9tfn6",instance="",job="quay",pid="208",process_name="secscan:application"}
1.631782319018925e+09
...

# HELP quay_gc_repos_purged_total number of repositories purged by the RepositoryGCWorker or
NamespaceGCWorker
# TYPE quay_gc_repos_purged_total counter
quay_gc_repos_purged_total{host="example-registry-quay-app-6df87f7b66-
9tfn6",instance="",job="quay",pid="208",process_name="secscan:application"} 0
...

# TYPE quay_gc_storage_blobs_deleted_created gauge
quay_gc_storage_blobs_deleted_created{host="example-registry-quay-app-6df87f7b66-
9tfn6",instance="",job="quay",pid="208",process_name="secscan:application"}
1.6317823190189059e+09
...

# HELP quay_gc_storage_blobs_deleted_total number of storage blobs deleted
# TYPE quay_gc_storage_blobs_deleted_total counter
quay_gc_storage_blobs_deleted_total{host="example-registry-quay-app-6df87f7b66-
9tfn6",instance="",job="quay",pid="208",process_name="secscan:application"} 0
...
```

13.2.3.1. Multipart uploads metrics

The multipart uploads metrics show the number of blobs uploads to storage (S3, Rados, GoogleCloudStorage, RHOCS). These can help identify issues when Quay is unable to correctly upload blobs to storage.

Metric name	Description
quay_multipart_uploads_started_total	Number of multipart uploads to Quay storage that started
quay_multipart_uploads_completed_total	Number of multipart uploads to Quay storage that completed

Sample metrics output

```
# TYPE quay_multipart_uploads_completed_created gauge
quay_multipart_uploads_completed_created{host="example-registry-quay-app-6df87f7b66-9tfn6",instance="",job="quay",pid="208",process_name="secscan:application"}
1.6317823308284895e+09
...

# HELP quay_multipart_uploads_completed_total number of multipart uploads to Quay storage that completed
# TYPE quay_multipart_uploads_completed_total counter
quay_multipart_uploads_completed_total{host="example-registry-quay-app-6df87f7b66-9tfn6",instance="",job="quay",pid="208",process_name="secscan:application"} 0

# TYPE quay_multipart_uploads_started_created gauge
quay_multipart_uploads_started_created{host="example-registry-quay-app-6df87f7b66-9tfn6",instance="",job="quay",pid="208",process_name="secscan:application"}
1.6317823308284352e+09
...

# HELP quay_multipart_uploads_started_total number of multipart uploads to Quay storage that started
# TYPE quay_multipart_uploads_started_total counter
quay_multipart_uploads_started_total{host="example-registry-quay-app-6df87f7b66-9tfn6",instance="",job="quay",pid="208",process_name="secscan:application"} 0
...
```

13.2.4. Image push / pull metrics

A number of metrics are available related to pushing and pulling images.

13.2.4.1. Image pulls total

Metric name	Description
quay_registry_image_pulls_total	The number of images downloaded from the registry.

Metric labels

- **protocol:** the registry protocol used (should always be v2)
- **ref:** ref used to pull - tag, manifest
- **status:** http return code of the request

13.2.4.2. Image bytes pulled

Metric name	Description
quay_registry_image_pulled_estimated_bytes_total	The number of bytes downloaded from the registry

Metric labels

- **protocol:** the registry protocol used (should always be v2)

13.2.4.3. Image pushes total

Metric name	Description
quay_registry_image_pushes_total	The number of images uploaded from the registry.

Metric labels

- **protocol:** the registry protocol used (should always be v2)
- **pstatus:** http return code of the request
- **pmedia_type:** the uploaded manifest type

13.2.4.4. Image bytes pushed

Metric name	Description
quay_registry_image_pushed_bytes_total	The number of bytes uploaded to the registry

Sample metrics output

```
# HELP quay_registry_image_pushed_bytes_total number of bytes pushed to the registry
# TYPE quay_registry_image_pushed_bytes_total counter
quay_registry_image_pushed_bytes_total{host="example-registry-quay-app-6df87f7b66-9tfn6",instance="",job="quay",pid="221",process_name="registry:application"} 0
...
```

13.2.5. Authentication metrics

The authentication metrics provide the number of authentication requests, labeled by type and whether it succeeded or not. For example, this metric could be used to monitor failed basic authentication requests.

Metric name	Description
quay_authentication_attempts_total	Number of authentication attempts across the registry and API

Metric labels

- **auth_kind:** The type of auth used, including:
 - basic
 - oauth
 - credentials
- **success:** true or false

Sample metrics output

```
# TYPE quay_authentication_attempts_created gauge
quay_authentication_attempts_created{auth_kind="basic",host="example-registry-quay-app-6df87f7b66-9tfn6",instance="",job="quay",pid="221",process_name="registry:application",success="True"}
1.6317843039374158e+09
...

# HELP quay_authentication_attempts_total number of authentication attempts across the registry
and API
# TYPE quay_authentication_attempts_total counter
quay_authentication_attempts_total{auth_kind="basic",host="example-registry-quay-app-6df87f7b66-9tfn6",instance="",job="quay",pid="221",process_name="registry:application",success="True"} 2
...
```

CHAPTER 14. RED HAT QUAY QUOTA MANAGEMENT AND ENFORCEMENT OVERVIEW

With Red Hat Quay, users have the ability to report storage consumption and to contain registry growth by establishing configured storage quota limits. On-premise Red Hat Quay users are now equipped with the following capabilities to manage the capacity limits of their environment:

- **Quota reporting:** With this feature, a superuser can track the storage consumption of all their organizations. Additionally, users can track the storage consumption of their assigned organization.
- **Quota management:** With this feature, a superuser can define soft and hard checks for Red Hat Quay users. Soft checks tell users if the storage consumption of an organization reaches their configured threshold. Hard checks prevent users from pushing to the registry when storage consumption reaches the configured limit.

Together, these features allow service owners of a Red Hat Quay registry to define service level agreements and support a healthy resource budget.

14.1. QUOTA MANAGEMENT LIMITATIONS

Quota management helps organizations to maintain resource consumption. One limitation of quota management is that calculating resource consumption on push results in the calculation becoming part of the push's critical path. Without this, usage data might drift.

The maximum storage quota size is dependent on the selected database:

Table 14.1. Worker count environment variables

Variable	Description
Postgres	8388608 TB
MySQL	8388608 TB
SQL Server	16777216 TB

14.2. QUOTA MANAGEMENT FOR RED HAT QUAY 3.9

If you are upgrading to Red Hat Quay 3.9, you must reconfigure the quota management feature. This is because with Red Hat Quay 3.9, calculation is done differently. As a result, totals prior to Red Hat Quay 3.9 are no longer valid. There are two methods for configuring quota management in Red Hat Quay 3.9, which are detailed in the following sections.



NOTE

- This is a one time calculation that must be done after you have upgraded to Red Hat Quay 3.9.
- Superuser privileges are required to create, update and delete quotas. While quotas can be set for users as well as organizations, you cannot reconfigure the *user* quota using the Red Hat Quay UI and you must use the API instead.

14.2.1. Option A: Configuring quota management for Red Hat Quay 3.9 by adjusting the `QUOTA_TOTAL_DELAY` feature flag

Use the following procedure to recalculate Red Hat Quay 3.9 quota management by adjusting the `QUOTA_TOTAL_DELAY` feature flag.



NOTE

With this recalculation option, the totals appear as **0.00 KB** until the allotted time designated for `QUOTA_TOTAL_DELAY`.

Prerequisites

- You have upgraded to Red Hat Quay 3.9.
- You are logged into Red Hat Quay 3.9 as a superuser.

Procedure

1. Deploy Red Hat Quay 3.9 with the following `config.yaml` settings:

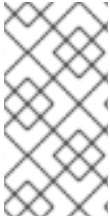
```
FEATURE_QUOTA_MANAGEMENT: true
FEATURE_GARBAGE_COLLECTION: true
PERMANENTLY_DELETE_TAGS: true
QUOTA_TOTAL_DELAY_SECONDS: 1800 1
RESET_CHILD_MANIFEST_EXPIRATION: true
```

- 1** The `QUOTA_TOTAL_DELAY_SECONDS` flag defaults to **1800** seconds, or 30 minutes. This allows Red Hat Quay 3.9 to successfully deploy before the quota management feature begins calculating storage consumption for every blob that has been pushed. Setting this flag to a lower number might result in miscalculation; it **must** be set to a number that is greater than the time it takes your Red Hat Quay deployment to start. **1800** is the recommended setting, however larger deployments that take longer than 30 minutes to start might require a longer duration than **1800**.

2. Navigate to the Red Hat Quay UI and click the name of your Organization.
3. The **Total Quota Consumed** should read **0.00 KB**. Additionally, the **Backfill Queued** indicator should be present.
4. After the allotted time, for example, 30 minutes, refresh your Red Hat Quay deployment page and return to your Organization. Now, the **Total Quota Consumed** should be present.

14.2.2. Option B: Configuring quota management for Red Hat Quay 3.9 by setting `QUOTA_TOTAL_DELAY_SECONDS` to 0

Use the following procedure to recalculate Red Hat Quay 3.9 quota management by setting `QUOTA_TOTAL_DELAY_SECONDS` to 0.



NOTE

Using this option prevents the possibility of miscalculations, however is more time intensive. Use the following procedure for when your Red Hat Quay deployment swaps the **FEATURE_QUOTA_MANAGEMENT** parameter from **false** to **true**. Most users will find xref:

Prerequisites

- You have upgraded to Red Hat Quay 3.9.
- You are logged into Red Hat Quay 3.9 as a superuser.

Procedure

1. Deploy Red Hat Quay 3.9 with the following **config.yaml** settings:

```
FEATURE_GARBAGE_COLLECTION: true
FEATURE_QUOTA_MANAGEMENT: true
QUOTA_BACKFILL: false
QUOTA_TOTAL_DELAY_SECONDS: 0
PERMANENTLY_DELETE_TAGS: true
RESET_CHILD_MANIFEST_EXPIRATION: true
```

2. Navigate to the Red Hat Quay UI and click the name of your Organization.
3. The **Total Quota Consumed** should read **0.00 KB**.
4. Redeploy Red Hat Quay and set the **QUOTA_BACKFILL** flag set to **true**. For example:

```
QUOTA_BACKFILL: true
```



NOTE

If you choose to disable quota management after it has calculated totals, Red Hat Quay marks those totals as stale. If you re-enable the quota management feature again in the future, those namespaces and repositories are recalculated by the backfill worker.

14.3. TESTING QUOTA MANAGEMENT FOR RED HAT QUAY 3.9

With quota management configured for Red Hat Quay 3.9, duplicative images are now only counted once towards the repository total.

Use the following procedure to test that a duplicative image is only counted once toward the repository total.

Prerequisites

- You have configured quota management for Red Hat Quay 3.9.

Procedure

1. Pull a sample image, for example, **ubuntu:18.04**, by entering the following command:

```
$ podman pull ubuntu:18.04
```

- Tag the same image twice by entering the following command:

```
$ podman tag docker.io/library/ubuntu:18.04 quay-server.example.com/quota-test/ubuntu:tag1
```

```
$ podman tag docker.io/library/ubuntu:18.04 quay-server.example.com/quota-test/ubuntu:tag2
```

- Push the sample image to your organization by entering the following commands:

```
$ podman push --tls-verify=false quay-server.example.com/quota-test/ubuntu:tag1
```

```
$ podman push --tls-verify=false quay-server.example.com/quota-test/ubuntu:tag2
```

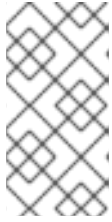
- On the Red Hat Quay UI, navigate to **Organization** and click the **Repository Name**, for example, **quota-test/ubuntu**. Then, click **Tags**. There should be two repository tags, **tag1** and **tag2**, each with the same manifest. For example:

TAG	LAST MODIFIED	SIZE	EXPIRES	MANIFEST
tag2	6 minutes ago	26.6 MB	Never	SHA256: 3fcb093159b8
tag1	7 minutes ago	26.6 MB	Never	SHA256: 3fcb093159b8

However, by clicking on the **Organization** link, we can see that the **Total Quota Consumed** is **27.94 MB**, meaning that the Ubuntu image has only been accounted for once:

REPOSITORY NAME	LAST MODIFIED	QUOTA CONSUMED	ACTIVITY	STAR
quota-test / ubuntu	Today at 3:51 PM	27.94 MB		

If you delete one of the Ubuntu tags, the **Total Quota Consumed** remains the same.

**NOTE**

If you have configured the Red Hat Quay time machine to be longer than **0** seconds, subtraction will not happen until those tags pass the time machine window. If you want to expedite permanent deletion, see Permanently deleting an image tag in Red Hat Quay 3.9.

14.4. SETTING DEFAULT QUOTA

To specify a system-wide default storage quota that is applied to every organization and user, you can use the **DEFAULT_SYSTEM_REJECT_QUOTA_BYTES** configuration flag.

If you configure a specific quota for an organization or user, and then delete that quota, the system-wide default quota will apply if one has been set. Similarly, if you have configured a specific quota for an organization or user, and then modify the system-wide default quota, the updated system-wide default will override any specific settings.

For more information about the **DEFAULT_SYSTEM_REJECT_QUOTA_BYTES** flag,

see link:

14.5. ESTABLISHING QUOTA IN RED HAT QUAY UI

The following procedure describes how you can report storage consumption and establish storage quota limits.

Prerequisites

- A Red Hat Quay registry.
- A superuser account.
- Enough storage to meet the demands of quota limitations.

Procedure

1. Create a new organization or choose an existing one. Initially, no quota is configured, as can be seen on the **Organization Settings** tab:

Organization Settings

Namespace: testorg
Organization names cannot be changed once set.

Avatar:
Avatar is generated based off the organization's name.

Delete organization: [Begin deletion >](#)

Time Machine: 14 days
The amount of time, after a tag is deleted, that the tag is accessible in time machine before being garbage collected.
[Save Expiration Time](#)

Quota Management: [No Quota Configured](#)

Proxy Cache:

Remote Registry:
Remote registry that is to be cached. (Eg: For docker hub, docker.io, docker.io/library)

Remote Registry Username:
Username for authenticating into the entered remote registry. For anonymous pulls from the upstream, leave this empty.

Remote Registry Password:
Password for authenticating into the entered remote registry. For anonymous pulls from the

- Log in to the registry as a superuser and navigate to the **Manage Organizations** tab on the **Super User Admin Panel**. Click the **Options** icon of the organization for which you want to create storage quota limits:

Red Hat Quay Management

Organizations

1 - 1 of 1 [Filter Organizations...](#)

NAME ↓	QUOTA CONSUMED
testorg	

- Rename Organization
- ✕ Delete Organization
- ⚡ Take Ownership
- ⚙️ Configure Quota

- Click **Configure Quota** and enter the initial quota, for example, **10 MB**. Then click **Apply** and **Close**:

Manage Organization Quota for testorg ✕

Set storage quota:

KB
 ✓ MB
 GB
 TB

Note: No quota policy defined. Users will be able to exceed the storage quota set above.

[Apply](#) [Close](#)

- Check that the quota consumed shows **0 of 10 MB** on the **Manage Organizations** tab of the superuser panel:

Organizations

1 - 1 of 1

NAME ↓	QUOTA CONSUMED
testorg	0 of 10 MB

The consumed quota information is also available directly on the Organization page:

Initial consumed quota

testorg + Create New Repository

Repositories

Total Quota Consumed: (0%) 0 of 10 MB

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](#).

- To increase the quota to 100MB, navigate to the **Manage Organizations** tab on the superuser panel. Click the **Options** icon and select **Configure Quota**, setting the quota to 100 MB. Click **Apply** and then **Close**:

Manage Organization Quota for testorg ×

Set storage quota:

Quota Policy: **Action** **Quota Threshold**

Note: No quota policy defined. Users will be able to exceed the storage quota set above.

- Pull a sample image by entering the following command:

```
$ podman pull ubuntu:18.04
```

- Tag the sample image by entering the following command:

```
$ podman tag docker.io/library/ubuntu:18.04 example-registry-quay-quay-
enterprise.apps.docs.gcp.quaydev.org/testorg/ubuntu:18.04
```

- Push the sample image to the organization by entering the following command:

```
$ podman push --tls-verify=false example-registry-quay-quay-
enterprise.apps.docs.gcp.quaydev.org/testorg/ubuntu:18.04
```

9. On the superuser panel, the quota consumed per organization is displayed:

Organizations

1 - 1 of 1

NAME ↓	QUOTA CONSUMED
testorg	27 MB (26.66%) of 100 MB

10. The Organization page shows the total proportion of the quota used by the image:

Total Quota Consumed for first image

testorg [+ Create New Repository](#)

Repositories
Total Quota Consumed: 27 MB (27%) of 100 MB

1 - 1 of 1

REPOSITORY NAME	LAST MODIFIED	STATE	QUOTA CONSUMED	ACTIVITY ↓	STAR
testorg / ubuntu	Today at 2:12 PM	Normal	27 MB (27%)		

11. Pull a second sample image by entering the following command:

```
$ podman pull nginx
```

12. Tag the second image by entering the following command:

```
$ podman tag docker.io/library/nginx example-registry-quay-quay-
enterprise.apps.docs.gcp.quaydev.org/testorg/nginx
```

13. Push the second image to the organization by entering the following command:

```
$ podman push --tls-verify=false example-registry-quay-quay-
enterprise.apps.docs.gcp.quaydev.org/testorg/nginx
```

14. The Organization page shows the total proportion of the quota used by each repository in that organization:

Total Quota Consumed for each repository

testorg + Create New Repository

Repositories
Total Quota Consumed: 83 MB (83%) of 100 MB

1 - 2 of 2 Filter Repositories...

REPOSITORY NAME	LAST MODIFIED	STATE	QUOTA CONSUMED	ACTIVITY ↓	STAR
testorg / ubuntu	Today at 2:12 PM	Normal	27 MB (27%)		☆
testorg / nginx	Today at 2:31 PM	Normal	56 MB (56%)		☆

15. Create *reject* and *warning* limits:

From the superuser panel, navigate to the **Manage Organizations** tab. Click the **Options** icon for the organization and select **Configure Quota**. In the **Quota Policy** section, with the **Action** type set to **Reject**, set the **Quota Threshold** to **80** and click **Add Limit**:

Manage Organization Quota for testorg ×

Set storage quota:

Quota Policy:

Action **Quota Threshold** + Add Limit

Note: No quota policy defined. Users will be able to exceed the storage quota set above.

Apply Remove Close

16. To create a *warning* limit, select **Warning** as the **Action** type, set the **Quota Threshold** to **70** and click **Add Limit**:

Manage Organization Quota for testorg ×

Set storage quota:

Quota Policy:








Action **Quota Threshold** Update Remove

Action **Quota Threshold** + Add Limit

Apply Remove Close


17. Click **Close** on the quota popup. The limits are viewable, but not editable, on the **Settings** tab of the **Organization** page:

T testorg
+ Create New Repository

-  Organization Settings
- 
- 
- 
- 
- 
- 

Organization Settings

Namespace: testorg
Organization names cannot be changed once set.

Avatar: 
Avatar is generated based off the organization's name.

Delete organization: [Begin deletion >](#)

Time Machine: 14 days
The amount of time, after a tag is deleted, that the tag is accessible in time machine before being garbage collected.
[Save Expiration Time](#)

Quota Management: Set storage quota:

Quota Policy:	Action	Quota Threshold
	<input type="text" value="Reject"/>	<input type="text" value="80"/>
	<input type="text" value="Warning"/>	<input type="text" value="70"/>

18. Push an image where the reject limit is exceeded:

Because the reject limit (80%) has been set to below the current repository size (~83%), the next pushed image is rejected automatically.

Sample image push

```
$ podman pull ubuntu:20.04
```

```
$ podman tag docker.io/library/ubuntu:20.04 example-registry-quay-quay-
enterprise.apps.docs.gcp.quaydev.org/testorg/ubuntu:20.04
```

```
$ podman push --tls-verify=false example-registry-quay-quay-
enterprise.apps.docs.gcp.quaydev.org/testorg/ubuntu:20.04
```

Sample output when quota exceeded

```
Getting image source signatures
Copying blob d4dfaa212623 [-----] 8.0b / 3.5KiB
Copying blob cba97cc5811c [-----] 8.0b / 15.0KiB
Copying blob 0c78fac124da [-----] 8.0b / 71.8MiB
WARN[0002] failed, retrying in 1s ... (1/3). Error: Error writing blob: Error initiating layer
upload to /v2/testorg/ubuntu/blobs/uploads/ in example-registry-quay-quay-
enterprise.apps.docs.gcp.quaydev.org: denied: Quota has been exceeded on namespace
Getting image source signatures
Copying blob d4dfaa212623 [-----] 8.0b / 3.5KiB
Copying blob cba97cc5811c [-----] 8.0b / 15.0KiB
Copying blob 0c78fac124da [-----] 8.0b / 71.8MiB
WARN[0005] failed, retrying in 1s ... (2/3). Error: Error writing blob: Error initiating layer
upload to /v2/testorg/ubuntu/blobs/uploads/ in example-registry-quay-quay-
enterprise.apps.docs.gcp.quaydev.org: denied: Quota has been exceeded on namespace
Getting image source signatures
Copying blob d4dfaa212623 [-----] 8.0b / 3.5KiB
Copying blob cba97cc5811c [-----] 8.0b / 15.0KiB
```

```

Copying blob 0c78fac124da [-----] 8.0b / 71.8MiB
WARN[0009] failed, retrying in 1s ... (3/3). Error: Error writing blob: Error initiating layer
upload to /v2/testorg/ubuntu/blobs/uploads/ in example-registry-quay-quay-
enterprise.apps.docs.gcp.quaydev.org: denied: Quota has been exceeded on namespace
Getting image source signatures
Copying blob d4dfaa212623 [-----] 8.0b / 3.5KiB
Copying blob cba97cc5811c [-----] 8.0b / 15.0KiB
Copying blob 0c78fac124da [-----] 8.0b / 71.8MiB
Error: Error writing blob: Error initiating layer upload to /v2/testorg/ubuntu/blobs/uploads/ in
example-registry-quay-quay-enterprise.apps.docs.gcp.quaydev.org: denied: Quota has been
exceeded on namespace

```

19. When limits are exceeded, notifications are displayed in the UI:

Quota notifications

The screenshot shows the Red Hat Quay interface for the organization 'testorg'. The main content area displays 'Organization Settings' with fields for Namespace (testorg), Avatar (T), Delete organization (Begin deletion), Time Machine (14 days), and Quota Management (Set storage quota: 100 MB). The Quota Policy section shows two rows: Action 'Reject' with Quota Threshold '80', and Action 'Warning' with Quota Threshold '70'. On the right, a 'Notifications' panel shows three notifications: 'testorg quota has been exceeded', each with a 'Dismiss Notification' link and a timestamp of 'May 5, 2022 4:01:12 PM'.

14.6. ESTABLISHING QUOTA WITH THE RED HAT QUAY API

When an organization is first created, it does not have a quota applied. Use the `/api/v1/organization/{organization}/quota` endpoint:

Sample command

```

$ curl -k -X GET -H "Authorization: Bearer <token>" -H 'Content-Type: application/json'
https://example-registry-quay-quay-
enterprise.apps.docs.gcp.quaydev.org/api/v1/organization/testorg/quota | jq

```

Sample output

```

[]

```

14.6.1. Setting the quota

To set a quota for an organization, POST data to the `/api/v1/organization/{orgname}/quota` endpoint: .Sample command

```
$ curl -k -X POST -H "Authorization: Bearer <token>" -H 'Content-Type: application/json' -d '{"limit_bytes": 10485760}' https://example-registry-quay-quay-enterprise.apps.docs.quayteam.org/api/v1/organization/testorg/quota | jq
```

Sample output

```
"Created"
```

14.6.2. Viewing the quota

To see the applied quota, **GET** data from the `/api/v1/organization/{orgname}/quota` endpoint:

Sample command

```
$ curl -k -X GET -H "Authorization: Bearer <token>" -H 'Content-Type: application/json' https://example-registry-quay-quay-enterprise.apps.docs.gcp.quaydev.org/api/v1/organization/testorg/quota | jq
```

Sample output

```
[
  {
    "id": 1,
    "limit_bytes": 10485760,
    "default_config": false,
    "limits": [],
    "default_config_exists": false
  }
]
```

14.6.3. Modifying the quota

To change the existing quota, in this instance from 10 MB to 100 MB, PUT data to the `/api/v1/organization/{orgname}/quota/{quota_id}` endpoint:

Sample command

```
$ curl -k -X PUT -H "Authorization: Bearer <token>" -H 'Content-Type: application/json' -d '{"limit_bytes": 104857600}' https://example-registry-quay-quay-enterprise.apps.docs.gcp.quaydev.org/api/v1/organization/testorg/quota/1 | jq
```

Sample output

```
{
  "id": 1,
  "limit_bytes": 104857600,
  "default_config": false,

```

```
"limits": [],
"default_config_exists": false
}
```

14.6.4. Pushing images

To see the storage consumed, push various images to the organization.

14.6.4.1. Pushing ubuntu:18.04

Push ubuntu:18.04 to the organization from the command line:

Sample commands

```
$ podman pull ubuntu:18.04

$ podman tag docker.io/library/ubuntu:18.04 example-registry-quay-quay-
enterprise.apps.docs.gcp.quaydev.org/testorg/ubuntu:18.04

$ podman push --tls-verify=false example-registry-quay-quay-
enterprise.apps.docs.gcp.quaydev.org/testorg/ubuntu:18.04
```

14.6.4.2. Using the API to view quota usage

To view the storage consumed, **GET** data from the `/api/v1/repository` endpoint:

Sample command

```
$ curl -k -X GET -H "Authorization: Bearer <token>" -H 'Content-Type: application/json'
'https://example-registry-quay-quay-enterprise.apps.docs.gcp.quaydev.org/api/v1/repository?
last_modified=true&namespace=testorg&popularity=true&public=true' | jq
```

Sample output

```
{
  "repositories": [
    {
      "namespace": "testorg",
      "name": "ubuntu",
      "description": null,
      "is_public": false,
      "kind": "image",
      "state": "NORMAL",
      "quota_report": {
        "quota_bytes": 27959066,
        "configured_quota": 104857600
      },
      "last_modified": 1651225630,
      "popularity": 0,
      "is_starred": false
    }
  ]
}
```

14.6.4.3. Pushing another image

1. Pull, tag, and push a second image, for example, **nginx**:

Sample commands

```
$ podman pull nginx

$ podman tag docker.io/library/nginx example-registry-quay-quay-
enterprise.apps.docs.gcp.quaydev.org/testorg/nginx

$ podman push --tls-verify=false example-registry-quay-quay-
enterprise.apps.docs.gcp.quaydev.org/testorg/nginx
```

2. To view the quota report for the repositories in the organization, use the `/api/v1/repository` endpoint:

Sample command

```
$ curl -k -X GET -H "Authorization: Bearer <token>" -H 'Content-Type: application/json'
'https://example-registry-quay-quay-enterprise.apps.docs.gcp.quaydev.org/api/v1/repository?
last_modified=true&namespace=testorg&popularity=true&public=true'
```

Sample output

```
{
  "repositories": [
    {
      "namespace": "testorg",
      "name": "ubuntu",
      "description": null,
      "is_public": false,
      "kind": "image",
      "state": "NORMAL",
      "quota_report": {
        "quota_bytes": 27959066,
        "configured_quota": 104857600
      },
      "last_modified": 1651225630,
      "popularity": 0,
      "is_starred": false
    },
    {
      "namespace": "testorg",
      "name": "nginx",
      "description": null,
      "is_public": false,
      "kind": "image",
      "state": "NORMAL",
      "quota_report": {
        "quota_bytes": 59231659,
        "configured_quota": 104857600
      },
      "last_modified": 1651229507,
      "popularity": 0,
    }
  ]
}
```



```

    "is_starred": false
  }
]
}

```

- To view the quota information in the organization details, use the `/api/v1/organization/{orgname}` endpoint:

Sample command

```

$ curl -k -X GET -H "Authorization: Bearer <token>" -H 'Content-Type: application/json'
'https://example-registry-quay-quay-enterprise.apps.docs.gcp.quaydev.org/api/v1/organization/testorg' | jq

```

Sample output

```

{
  "name": "testorg",
  ...
  "quotas": [
    {
      "id": 1,
      "limit_bytes": 104857600,
      "limits": []
    }
  ],
  "quota_report": {
    "quota_bytes": 87190725,
    "configured_quota": 104857600
  }
}

```

14.6.5. Rejecting pushes using quota limits

If an image push exceeds defined quota limitations, a soft or hard check occurs:

- For a soft check, or *warning*, users are notified.
- For a hard check, or *reject*, the push is terminated.

14.6.5.1. Setting reject and warning limits

To set *reject* and *warning* limits, POST data to the `/api/v1/organization/{orgname}/quota/{quota_id}/limit` endpoint:

Sample reject limit command

```

$ curl -k -X POST -H "Authorization: Bearer <token>" -H 'Content-Type: application/json' -d
'{"type":"Reject","threshold_percent":80}' https://example-registry-quay-quay-
enterprise.apps.docs.gcp.quaydev.org/api/v1/organization/testorg/quota/1/limit

```

Sample warning limit command

```
$ curl -k -X POST -H "Authorization: Bearer <token>" -H 'Content-Type: application/json' -d
{"type": "Warning", "threshold_percent": 50} https://example-registry-quay-quay-
enterprise.apps.docs.gcp.quaydev.org/api/v1/organization/testorg/quota/1/limit
```

14.6.5.2. Viewing reject and warning limits

To view the *reject* and *warning* limits, use the `/api/v1/organization/{orgname}/quota` endpoint:

View quota limits

```
$ curl -k -X GET -H "Authorization: Bearer <token>" -H 'Content-Type: application/json'
https://example-registry-quay-quay-
enterprise.apps.docs.gcp.quaydev.org/api/v1/organization/testorg/quota | jq
```

Sample output for quota limits

```
[
  {
    "id": 1,
    "limit_bytes": 104857600,
    "default_config": false,
    "limits": [
      {
        "id": 2,
        "type": "Warning",
        "limit_percent": 50
      },
      {
        "id": 1,
        "type": "Reject",
        "limit_percent": 80
      }
    ],
    "default_config_exists": false
  }
]
```

14.6.5.3. Pushing an image when the reject limit is exceeded

In this example, the reject limit (80%) has been set to below the current repository size (~83%), so the next push should automatically be rejected.

Push a sample image to the organization from the command line:

Sample image push

```
$ podman pull ubuntu:20.04

$ podman tag docker.io/library/ubuntu:20.04 example-registry-quay-quay-
enterprise.apps.docs.gcp.quaydev.org/testorg/ubuntu:20.04

$ podman push --tls-verify=false example-registry-quay-quay-
enterprise.apps.docs.gcp.quaydev.org/testorg/ubuntu:20.04
```

Sample output when quota exceeded

```

Getting image source signatures
Copying blob d4dfaa212623 [-----] 8.0b / 3.5KiB
Copying blob cba97cc5811c [-----] 8.0b / 15.0KiB
Copying blob 0c78fac124da [-----] 8.0b / 71.8MiB
WARN[0002] failed, retrying in 1s ... (1/3). Error: Error writing blob: Error initiating layer upload to
/v2/testorg/ubuntu/blobs/uploads/ in example-registry-quay-quay-
enterprise.apps.docs.gcp.quaydev.org: denied: Quota has been exceeded on namespace
Getting image source signatures
Copying blob d4dfaa212623 [-----] 8.0b / 3.5KiB
Copying blob cba97cc5811c [-----] 8.0b / 15.0KiB
Copying blob 0c78fac124da [-----] 8.0b / 71.8MiB
WARN[0005] failed, retrying in 1s ... (2/3). Error: Error writing blob: Error initiating layer upload to
/v2/testorg/ubuntu/blobs/uploads/ in example-registry-quay-quay-
enterprise.apps.docs.gcp.quaydev.org: denied: Quota has been exceeded on namespace
Getting image source signatures
Copying blob d4dfaa212623 [-----] 8.0b / 3.5KiB
Copying blob cba97cc5811c [-----] 8.0b / 15.0KiB
Copying blob 0c78fac124da [-----] 8.0b / 71.8MiB
WARN[0009] failed, retrying in 1s ... (3/3). Error: Error writing blob: Error initiating layer upload to
/v2/testorg/ubuntu/blobs/uploads/ in example-registry-quay-quay-
enterprise.apps.docs.gcp.quaydev.org: denied: Quota has been exceeded on namespace
Getting image source signatures
Copying blob d4dfaa212623 [-----] 8.0b / 3.5KiB
Copying blob cba97cc5811c [-----] 8.0b / 15.0KiB
Copying blob 0c78fac124da [-----] 8.0b / 71.8MiB
Error: Error writing blob: Error initiating layer upload to /v2/testorg/ubuntu/blobs/uploads/ in example-
registry-quay-quay-enterprise.apps.docs.gcp.quaydev.org: denied: Quota has been exceeded on
namespace

```

14.6.5.4. Notifications for limits exceeded

When limits are exceeded, a notification appears:

Quota notifications

The screenshot displays the Red Hat Quay interface for the 'testorg' organization. The main content area is titled 'Organization Settings' and includes the following sections:

- Namespace:** testorg (Note: Organization names cannot be changed once set.)
- Avatar:** A large 'T' icon (Note: Avatar is generated based off the organization's name.)
- Delete organization:** A button labeled 'Begin deletion >'.
- Time Machine:** A dropdown menu set to '14 days'. Below it, a note states: 'The amount of time, after a tag is deleted, that the tag is accessible in time machine before being garbage collected.' A 'Save Expiration Time' button is present.
- Quota Management:** A section for 'Set storage quota' with a text input '100' and a dropdown menu 'MB'. Below this is a 'Quota Policy' table:

Quota Policy	Action	Quota Threshold
	Reject	80
	Warning	70

On the right side, a 'Notifications' panel shows three identical notifications: 'testorg quota has been exceeded', each with a 'Dismiss Notification' link and a timestamp of 'May 5, 2022 4:01:12 PM'.

14.7. CALCULATING THE TOTAL REGISTRY SIZE IN RED HAT QUAY 3.9

Use the following procedure to queue a registry total calculation.



NOTE

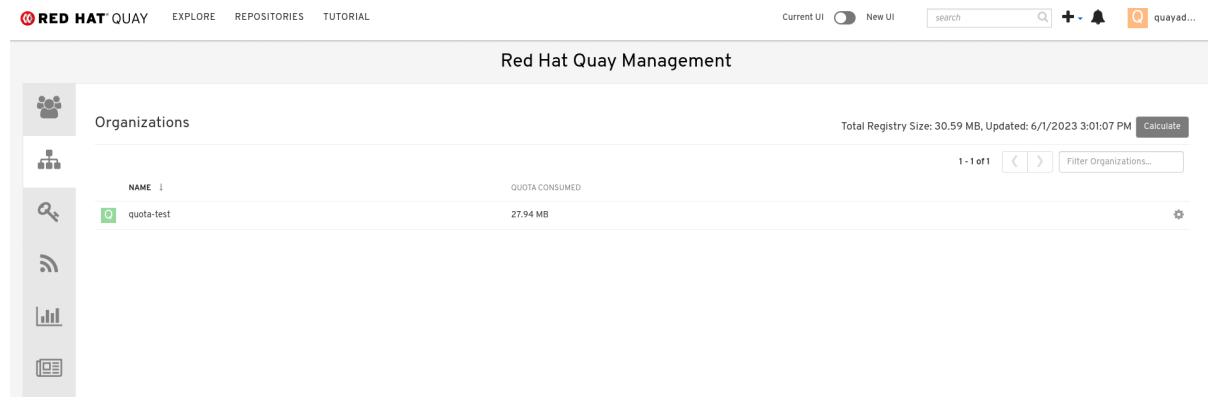
This feature is done on-demand, and calculating a registry total is database intensive. Use with caution.

Prerequisites

- You have upgraded to Red Hat Quay 3.9.
- You are logged in as a Red Hat Quay superuser.

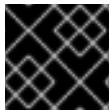
Procedure

1. On the Red Hat Quay UI, click your username → **Super User Admin Panel**.
2. In the navigation pane, click **Manage Organizations**.
3. Click **Calculate**, next to **Total Registry Size: 0.00 KB, Updated: Never , Calculation required**. Then, click **Ok**.
4. After a few minutes, depending on the size of your registry, refresh the page. Now, the Total Registry Size should be calculated. For example:



14.8. PERMANENTLY DELETING AN IMAGE TAG

In some cases, users might want to delete an image tag outside of the time machine window. Use the following procedure to manually delete an image tag permanently.



IMPORTANT

The results of the following procedure cannot be undone. Use with caution.

14.8.1. Permanently deleting an image tag using the Red Hat Quay v2 UI

Use the following procedure to permanently delete an image tag using the Red Hat Quay v2 UI.

Prerequisites

- You have set **FEATURE_UI_V2** to **true** in your **config.yaml** file.

Procedure

- Ensure that the **PERMANENTLY_DELETE_TAGS** and **RESET_CHILD_MANIFEST_EXPIRATION** parameters are set to **true** in your **config.yaml** file. For example:

```
PERMANENTLY_DELETE_TAGS: true
RESET_CHILD_MANIFEST_EXPIRATION: true
```

- In the navigation pane, click **Repositories**.
- Click the name of the repository, for example, **quayadmin/busybox**.
- Check the box of the image tag that will be deleted, for example, **test**.
- Click **Actions** → **Permanently Delete**.



IMPORTANT

This action is permanent and cannot be undone.

14.8.2. Permanently deleting an image tag using the Red Hat Quay legacy UI

Use the following procedure to permanently delete an image tag using the Red Hat Quay legacy UI.

Procedure

1. Ensure that the **PERMANENTLY_DELETE_TAGS** and **RESET_CHILD_MANIFEST_EXPIRATION** parameters are set to **true** in your **config.yaml** file. For example:

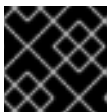
```
PERMANENTLY_DELETE_TAGS: true
RESET_CHILD_MANIFEST_EXPIRATION: true
```

2. On the Red Hat Quay UI, click **Repositories** and the name of the repository that contains the image tag you will delete, for example, **quayadmin/busybox**.
3. In the navigation pane, click **Tags**.
4. Check the box of the name of the tag you want to delete, for example, **test**.
5. Click the **Actions** drop down menu and select **Delete Tags → Delete Tag**.
6. Click **Tag History** in the navigation pane.
7. On the name of the tag that was just deleted, for example, **test**, click **Delete test** under the **Permanently Delete** category. For example:

Permanently delete image tag

The screenshot displays the Red Hat Quay interface for the repository **quayadmin/busybox**. The **Tag History** section is active, showing a table of tag changes. The table has the following structure:

TAG CHANGE	DATE/TIME	REVERT	PERMANENTLY DELETE
test was deleted	Mon, May 22, 2023 4:44 PM	Restore to SHA256: c189f9c1315	Delete test SHA256: c189f9c1315
test was created pointing to SHA256: c189f9c1315	Mon, May 22, 2023 4:39 PM		



IMPORTANT

This action is permanent and cannot be undone.

CHAPTER 15. RED HAT QUAY AUTO-PRUNING OVERVIEW

Red Hat Quay administrators can set up auto-pruning policies on namespaces and repositories. This feature allows for image tags to be automatically deleted within a namespace or a repository based on specified criteria, which allows Red Hat Quay organization owners to stay below the storage quota by automatically pruning content.

Currently, two policies have been added:

- **Prune images by the number of tags** For this policy, when the actual number of tags exceeds the desired number of tags, the oldest tags are deleted by their creation date until the desired number of tags is achieved.
- **Prune image tags by creation date** For this policy, any tags with a creation date older than the given time span, for example, 10 days, are deleted.

After tags are automatically pruned, they go into the Red Hat Quay time machine, or the amount of time, after a tag is deleted, that the tag is accessible before being garbage collected. The expiration time of an image tag is dependent on your organization's settings. For more information, see [Red Hat Quay garbage collection](#).

Users can only configure one policy per namespace or repository; this can be done through the Red Hat Quay v2 UI. Policies can also be set by using the API endpoints through the command-line interface (CLI).

15.1. PREREQUISITES AND LIMITATIONS FOR AUTO-PRUNING

The following prerequisites and limitations apply to the auto-pruning feature:

- This feature is not available when using the Red Hat Quay legacy UI. You must use the v2 UI to create, view, or modify auto-pruning policies.
- Auto-pruning is only supported in databases that support the **FOR UPDATE SKIP LOCKED** SQL command.

15.2. MANAGING AUTO-PRUNING POLICIES USING THE RED HAT QUAY UI

Auto-pruning policies are created using the Red Hat Quay v2 UI. This can be done after you have configured your Red Hat Quay **config.yaml** file to enable the auto-pruning feature and the v2 UI.



NOTE

This feature is not available when using the Red Hat Quay legacy UI.

15.2.1. Configuring the Red Hat Quay auto-pruning feature

Use the following procedure to configure your Red Hat Quay **config.yaml** file to enable the auto-pruning feature.

Prerequisites

- You have set **FEATURE_UI_V2** to **true** in your **config.yaml** file.

Procedure

- In your Red Hat Quay **config.yaml** file, add, and set, the **FEATURE_AUTO_PRUNE** environment variable to **True**. For example:

```
# ...  
FEATURE_AUTO_PRUNE: true  
# ...
```

15.2.2. Creating an auto-prune policy for a namespace using the Red Hat Quay v2 UI

Use the following procedure to create an auto-prune policy for a namespace using the Red Hat Quay v2 UI.

Prerequisites

- You have enabled the **FEATURE_AUTO_PRUNE** feature.

Procedure

1. Tag four sample images, for example, **busybox**, that will be pushed to the repository with auto-pruning enabled. For example:

```
$ podman tag docker.io/library/busybox <quay-server.example.com>/<quayadmin>/busybox:test
```

```
$ podman tag docker.io/library/busybox <quay-server.example.com>/<quayadmin>/busybox:test2
```

```
$ podman tag docker.io/library/busybox <quay-server.example.com>/<quayadmin>/busybox:test3
```

```
$ podman tag docker.io/library/busybox <quay-server.example.com>/<quayadmin>/busybox:test4
```

2. Push the four sample images, for example, **busybox**, to the repository with auto-pruning enabled by entering the following commands:

```
$ podman push <quay-server.example.com>/quayadmin/busybox:test
```

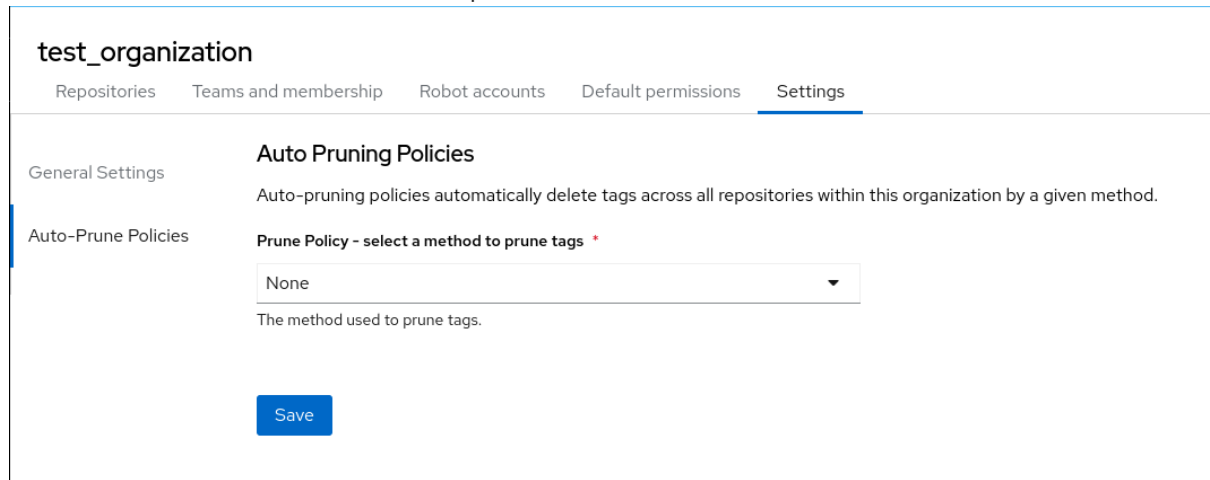
```
$ podman push <quay-server.example.com>/<quayadmin>/busybox:test2
```

```
$ podman push <quay-server.example.com>/<quayadmin>/busybox:test3
```

```
$ podman push <quay-server.example.com>/<quayadmin>/busybox:test4
```

3. Check that there are four tags in your repository.
4. On the Red Hat Quay v2 UI, click **Organizations** in the navigation pane.

5. Select the name of an organization that you will apply the auto-pruning feature to, for example, **test_organization**.
6. Click **Settings**.
7. Click **Auto-Prune Policies**. For example:



8. Click the drop down menu and select the desired policy, for example, **By number of tags**.
9. Select the desired number of tags to keep. By default, this is set at **20** tags. For this example, the number of tags to keep is set at **3**.
10. Click **Save**. A notification that your auto-prune policy has been updated appears.

Verification

- Navigate to the **Tags** page of your Organization's repository. With this example, tags are marked for deletion starting from the tag's oldest creation date. After a few minutes, the auto-pruner worker removes tags that no longer fit within the established criteria. In this example, it removes the **busybox:test** tag, and keeps the **busybox:test2**, **busybox:test3**, and **busybox:test4** tag. After tags are automatically pruned, they go into the Red Hat Quay time machine, or the amount of time after a tag is deleted that the tag is accessible before being garbage collected. The expiration time of an image tag is dependent on your organization's settings. For more information, see [Red Hat Quay garbage collection](#).

15.2.3. Creating an auto-prune policy for a namespace using the Red Hat Quay API

You can use Red Hat Quay API endpoints to manage auto-pruning policies for a namespace.

Prerequisites

- You have set **BROWSER_API_CALLS_XHR_ONLY: false** in your **config.yaml** file.
- You have created an OAuth access token.
- You have logged into Red Hat Quay.

Procedure

1. Enter the following **POST** command create a new policy that limits the number of tags allowed in an organization:

```
$ curl -X POST -H "Authorization: Bearer <access_token>" -H "Content-Type: application/json" -d '{"method": "number_of_tags", "value": 10}' http://<quay-server.example.com>/api/v1/organization/<organization_name>/autoprunepolicy/
```

Alternatively, you can set tags to expire for a specified time after their creation date:

```
$ curl -X POST -H "Authorization: Bearer <access_token>" -H "Content-Type: application/json" -d '{"method": "creation_date", "value": "7d"}' http://<quay-server.example.com>/api/v1/organization/<organization_name>/autoprunepolicy/
```

Example output

```
{"uuid": "73d64f05-d587-42d9-af6d-e726a4a80d6e"}
```

Attempting to create multiple policies returns the following error:

```
{"detail": "Policy for this namespace already exists, delete existing to create new policy", "error_message": "Policy for this namespace already exists, delete existing to create new policy", "error_type": "invalid_request", "title": "invalid_request", "type": "http://<quay-server.example.com>/api/v1/error/invalid_request", "status": 400}
```

2. Check your auto-prune policy by entering the following command:

```
$ curl -X GET -H "Authorization: Bearer <access_token>" http://<quay-server.example.com>/api/v1/organization/<organization_name>/autoprunepolicy/
```

Example output

```
{"policies": [{"uuid": "73d64f05-d587-42d9-af6d-e726a4a80d6e", "method": "creation_date", "value": "7d"}]}
```

3. You can delete the auto-prune policy by entering the following command. Note that deleting the policy requires the UUID.

```
$ curl -X DELETE -H "Authorization: Bearer <access_token>" http://<quay-server.example.com>/api/v1/organization/<organization_name>/autoprunepolicy/73d64f05-d587-42d9-af6d-e726a4a80d6e
```

15.2.4. Creating an auto-prune policy for a namespace for the current user using the API

You can use Red Hat Quay API endpoints to manage auto-pruning policies for your account.



NOTE

The use of **/user/** in the following commands represents the user that is currently logged into Red Hat Quay.

Prerequisites

- You have set **BROWSER_API_CALLS_XHR_ONLY: false** in your **config.yaml** file.
- You have created an OAuth access token.
- You have logged into Red Hat Quay.

Procedure

1. Enter the following **POST** command create a new policy that limits the number of tags for the current user:

```
$ curl -X POST -H "Authorization: Bearer <access_token>" -H "Content-Type: application/json" -d '{"method": "number_of_tags", "value": 10}' http://<quay-server.example.com>/api/v1/<user>/autoprunepolicy/
```

Example output

```
{"uuid": "8c03f995-ca6f-4928-b98d-d75ed8c14859"}
```

2. Check your auto-prune policy by entering the following command:

```
$ curl -X GET -H "Authorization: Bearer <access_token>" http://<quay-server.example.com>/api/v1/<user>/autoprunepolicy/8c03f995-ca6f-4928-b98d-d75ed8c14859
```

Alternatively, you can include the UUID:

```
$ curl -X GET -H "Authorization: Bearer <access_token>" http://<quay-server.example.com>/api/v1/<user>/autoprunepolicy/
```

Example output

```
{"policies": [{"uuid": "8c03f995-ca6f-4928-b98d-d75ed8c14859", "method": "number_of_tags", "value": 10}]}
```

3. You can delete the auto-prune policy by entering the following command. Note that deleting the policy requires the UUID.

```
$ curl -X DELETE -H "Authorization: Bearer <access_token>" http://<quay-server.example.com>/api/v1/<user>/autoprunepolicy/8c03f995-ca6f-4928-b98d-d75ed8c14859
```

Example output

```
{"uuid": "8c03f995-ca6f-4928-b98d-d75ed8c14859"}
```

15.2.5. Creating an auto-prune policy for a repository using the Red Hat Quay v2 UI

Use the following procedure to create an auto-prune policy for a repository using the Red Hat Quay v2 UI.

Prerequisites

- You have enabled the **FEATURE_AUTO_PRUNE** feature.

Procedure

1. Tag four sample images, for example, **busybox**, that will be pushed to the repository with auto-pruning enabled. For example:

```
$ podman tag docker.io/library/busybox <quay-server.example.com>/<organization_name>/<repository_name>:test
```

```
$ podman tag docker.io/library/busybox <quay-server.example.com>/<organization_name>/<repository_name>:test2
```

```
$ podman tag docker.io/library/busybox <quay-server.example.com>/<organization_name>/<repository_name>:test3
```

```
$ podman tag docker.io/library/busybox <quay-server.example.com>/<organization_name>/<repository_name>:test4
```

2. Push the four sample images, for example, **busybox**, to the repository with auto-pruning enabled by entering the following commands:

```
$ podman push <quay-server.example.com>/<organization_name>/<repository_name>:test
```

```
$ podman push <quay-server.example.com>/<organization_name>/<repository_name>:test2
```

```
$ podman push <quay-server.example.com>/<organization_name>/<repository_name>:test3
```

```
$ podman push <quay-server.example.com>/<organization_name>/<repository_name>:test4
```

3. Check that there are four tags in your repository.
4. On the Red Hat Quay v2 UI, click **Repository** in the navigation pane.
5. Select the name of an organization that you will apply the auto-pruning feature to, for example, **<organization_name>/<repository_name>**.
6. Click **Settings**.
7. Click **Repository Auto-Prune Policies**.
8. Click the drop down menu and select the desired policy, for example, **By number of tags**.
9. Select the desired number of tags to keep. By default, this is set at **20** tags. For this example, the number of tags to keep is set at **3**.
10. Click **Save**. A notification that your auto-prune policy has been updated appears.

Verification

- Navigate to the **Tags** page of your Organization's repository. With this example, tags are marked for deletion starting from the tag's oldest creation date. After a few minutes, the auto-pruner

worker removes tags that no longer fit within the established criteria. In this example, it removes the **busybox:test** tag, and keeps the **busybox:test2**, **busybox:test3**, and **busybox:test4** tag. After tags are automatically pruned, they go into the Red Hat Quay time machine, or the amount of time after a tag is deleted that the tag is accessible before being garbage collected. The expiration time of an image tag is dependent on your organization's settings. For more information, see [Red Hat Quay garbage collection](#).

15.2.6. Creating an auto-prune policy for a repository using the Red Hat Quay API

You can use Red Hat Quay API endpoints to manage auto-pruning policies for an repository.

Prerequisites

- You have set **BROWSER_API_CALLS_XHR_ONLY: false** in your **config.yaml** file.
- You have created an OAuth access token.
- You have logged into Red Hat Quay.

Procedure

1. Enter the following **POST** command create a new policy that limits the number of tags allowed in an organization:

```
$ curl -X POST -H "Authorization: Bearer <access_token>" -H "Content-Type: application/json" -d '{"method": "number_of_tags", "value": 2}' http://<quay-server.example.com>/api/v1/repository/<organization_name>/<repository_name>/autoprunepolicy/
```

Alternatively, you can set tags to expire for a specified time after their creation date:

```
$ curl -X POST -H "Authorization: Bearer <access_token>" -H "Content-Type: application/json" -d '{"method": "creation_date", "value": "7d"}' http://<quay-server.example.com>/api/v1/repository/<organization_name>/<repository_name>/autoprunepolicy/
```

Example output

```
{"uuid": "ce2bdcc0-ced2-4a1a-ac36-78a9c1bed8c7"}
```

Attempting to create multiple policies returns the following error:

```
{"detail": "Policy for this namespace already exists, delete existing to create new policy", "error_message": "Policy for this namespace already exists, delete existing to create new policy", "error_type": "invalid_request", "title": "invalid_request", "type": "http://quay-server.example.com/api/v1/error/invalid_request", "status": 400}
```

2. Check your auto-prune policy by entering the following command:

```
$ curl -X GET -H "Authorization: Bearer <access_token>" http://<quay-server.example.com>/api/v1/repository/<organization_name>/<repository_name>/autoprunepolicy/
```

Alternatively, you can include the UUID:

```
$ curl -X GET -H "Authorization: Bearer <access_token>" http://<quay-server.example.com>/api/v1/repository/<organization_name>/<repository_name>/autoprunepolicy/ce2bdcc0-ced2-4a1a-ac36-78a9c1bed8c7
```

Example output

```
{"policies": [{"uuid": "ce2bdcc0-ced2-4a1a-ac36-78a9c1bed8c7", "method": "number_of_tags", "value": 10}]}
```

3. You can delete the auto-prune policy by entering the following command. Note that deleting the policy requires the UUID.

```
$ curl -X DELETE -H "Authorization: Bearer <access_token>" http://<quay-server.example.com>/api/v1/repository/<organization_name>/<repository_name>/autoprunepolicy/ce2bdcc0-ced2-4a1a-ac36-78a9c1bed8c7
```

Example output

```
{"uuid": "ce2bdcc0-ced2-4a1a-ac36-78a9c1bed8c7"}
```

15.2.7. Creating an auto-prune policy on a repository for a user with the API

You can use Red Hat Quay API endpoints to manage auto-pruning policies on a repository for user accounts that are not your own, so long as you have **admin** privileges on the repository.

Prerequisites

- You have set **BROWSER_API_CALLS_XHR_ONLY: false** in your **config.yaml** file.
- You have created an OAuth access token.
- You have logged into Red Hat Quay.
- You have **admin** privileges on the repository that you are creating the policy for.

Procedure

1. Enter the following **POST** command create a new policy that limits the number of tags for the current user:

```
$ curl -X POST -H "Authorization: Bearer <access_token>" -H "Content-Type: application/json" -d '{"method": "number_of_tags", "value": 2}' http://<quay-server.example.com>/api/v1/repository/<user_account>/<user_repository>/autoprunepolicy/
```

Example output

```
{"uuid": "7726f79c-cbc7-490e-98dd-becdc6fefce7"}
```

2. Check your auto-prune policy by entering the following command:

```
$ curl -X GET -H "Authorization: Bearer <access_token>" http://<quay-server.example.com>/api/v1/repository/<user_account>/<user_repository>/autoprunepolicy/
```

Alternatively, you can include the UUID:

```
$ curl -X GET -H "Authorization: Bearer <access_token>" http://<quay-server.example.com>/api/v1/repository/<user_account>/<user_repository>/autoprunepolicy/7726f79c-cbc7-490e-98dd-becdc6fefce7
```

Example output

```
{"policies": [{"uuid": "7726f79c-cbc7-490e-98dd-becdc6fefce7", "method": "number_of_tags", "value": 2}]}
```

3. You can delete the auto-prune policy by entering the following command. Note that deleting the policy requires the UUID.

```
$ curl -X DELETE -H "Authorization: Bearer <access_token>" http://<quay-server.example.com>/api/v1/user/autoprunepolicy/7726f79c-cbc7-490e-98dd-becdc6fefce7
```

Example output

```
{"uuid": "7726f79c-cbc7-490e-98dd-becdc6fefce7"}
```

CHAPTER 16. GEO-REPLICATION



NOTE

Currently, the geo-replication feature is not supported on IBM Power.

Geo-replication allows multiple, geographically distributed Red Hat Quay deployments to work as a single registry from the perspective of a client or user. It significantly improves push and pull performance in a globally-distributed Red Hat Quay setup. Image data is asynchronously replicated in the background with transparent failover and redirect for clients.

Deployments of Red Hat Quay with geo-replication is supported on standalone and Operator deployments.

16.1. GEO-REPLICATION FEATURES

- When geo-replication is configured, container image pushes will be written to the preferred storage engine for that Red Hat Quay instance. This is typically the nearest storage backend within the region.
- After the initial push, image data will be replicated in the background to other storage engines.
- The list of replication locations is configurable and those can be different storage backends.
- An image pull will always use the closest available storage engine, to maximize pull performance.
- If replication has not been completed yet, the pull will use the source storage backend instead.

16.2. GEO-REPLICATION REQUIREMENTS AND CONSTRAINTS

- In geo-replicated setups, Red Hat Quay requires that all regions are able to read and write to all other region's object storage. Object storage must be geographically accessible by all other regions.
- In case of an object storage system failure of one geo-replicating site, that site's Red Hat Quay deployment must be shut down so that clients are redirected to the remaining site with intact storage systems by a global load balancer. Otherwise, clients will experience pull and push failures.
- Red Hat Quay has no internal awareness of the health or availability of the connected object storage system. Users must configure a global load balancer (LB) to monitor the health of your distributed system and to route traffic to different sites based on their storage status.
- To check the status of your geo-replication deployment, you must use the **/health/endoend** checkpoint, which is used for global health monitoring. You must configure the redirect manually using the **/health/endoend** endpoint. The **/health/instance** end point only checks local instance health.
- If the object storage system of one site becomes unavailable, there will be no automatic redirect to the remaining storage system, or systems, of the remaining site, or sites.
- Geo-replication is asynchronous. The permanent loss of a site incurs the loss of the data that has been saved in that sites' object storage system but has not yet been replicated to the remaining sites at the time of failure.

- A single database, and therefore all metadata and Red Hat Quay configuration, is shared across all regions.
Geo-replication does not replicate the database. In the event of an outage, Red Hat Quay with geo-replication enabled will not failover to another database.
- A single Redis cache is shared across the entire Red Hat Quay setup and needs to be accessible by all Red Hat Quay pods.
- The exact same configuration should be used across all regions, with exception of the storage backend, which can be configured explicitly using the **QUAY_DISTIBUTED_STORAGE_PREFERENCE** environment variable.
- Geo-replication requires object storage in each region. It does not work with local storage.
- Each region must be able to access every storage engine in each region, which requires a network path.
- Alternatively, the storage proxy option can be used.
- The entire storage backend, for example, all blobs, is replicated. Repository mirroring, by contrast, can be limited to a repository, or an image.
- All Red Hat Quay instances must share the same endpoint, typically through a load balancer.
- All Red Hat Quay instances must have the same set of superusers, as they are defined inside the common configuration file.
- Geo-replication requires your Clair configuration to be set to **unmanaged**. An unmanaged Clair database allows the Red Hat Quay Operator to work in a geo-replicated environment, where multiple instances of the Red Hat Quay Operator must communicate with the same database. For more information, see [Advanced Clair configuration](#).
- Geo-Replication requires SSL/TLS certificates and keys. For more information, see [Using SSL/TLS to protect connections to Red Hat Quay](#).

If the above requirements cannot be met, you should instead use two or more distinct Red Hat Quay deployments and take advantage of repository mirroring functions.

16.2.1. Enabling storage replication for standalone Red Hat Quay

Use the following procedure to enable storage replication on Red Hat Quay.

Procedure

1. Update your **config.yaml** file to include the storage engines to which data will be replicated. You must list all storage engines to be used:

```
# ...
FEATURE_STORAGE_REPLICATION: true
# ...
DISTRIBUTED_STORAGE_CONFIG:
  usstorage:
    - RHOCSStorage
    - access_key: <access_key>
      bucket_name: <example_bucket>
      hostname: my.noobaa.hostname
```

```

    is_secure: false
    port: "443"
    secret_key: <secret_key>
    storage_path: /datastorage/registry
  eustorage:
    - S3Storage
    - host: s3.amazonaws.com
      port: "443"
      s3_access_key: <access_key>
      s3_bucket: <example bucket>
      s3_secret_key: <secret_key>
      storage_path: /datastorage/registry
  DISTRIBUTED_STORAGE_DEFAULT_LOCATIONS: []
  DISTRIBUTED_STORAGE_PREFERENCE:
    - usstorage
    - eustorage
  # ...

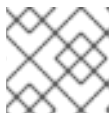
```

- Optional. If complete replication of all images to all storage engines is required, you can replicate images to the storage engine by manually setting the **DISTRIBUTED_STORAGE_DEFAULT_LOCATIONS** field. This ensures that all images are replicated to that storage engine. For example:

```

# ...
DISTRIBUTED_STORAGE_DEFAULT_LOCATIONS:
  - usstorage
  - eustorage
# ...

```

**NOTE**

To enable per-namespace replication, contact Red Hat Quay support.

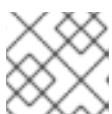
- After adding storage and enabling **Replicate to storage engine by default** for geo-replication, you must sync existing image data across all storage. To do this, you must execute into the container by running the following command:

```
$ podman exec -it <container_id>
```

- To sync the content after adding new storage, enter the following commands:

```
# scl enable python27 bash
```

```
# python -m util.backfillreplication
```

**NOTE**

This is a one time operation to sync content after adding new storage.

16.2.2. Run Red Hat Quay with storage preferences

- Copy the config.yaml to all machines running Red Hat Quay

- For each machine in each region, add a **QUAY_DISTRIBUTED_STORAGE_PREFERENCE** environment variable with the preferred storage engine for the region in which the machine is running.

For example, for a machine running in Europe with the config directory on the host available from **\$QUAY/config**:

```
$ sudo podman run -d --rm -p 80:8080 -p 443:8443 \
  --name=quay \
  -v $QUAY/config:/conf/stack:Z \
  -e QUAY_DISTRIBUTED_STORAGE_PREFERENCE=europestorage \
  registry.redhat.io/quay/quay-rhel8:v3.11.1
```



NOTE

The value of the environment variable specified must match the name of a Location ID as defined in the config panel.

- Restart all Red Hat Quay containers

16.2.3. Removing a geo-replicated site from your standalone Red Hat Quay deployment

By using the following procedure, Red Hat Quay administrators can remove sites in a geo-replicated setup.

Prerequisites

- You have configured Red Hat Quay geo-replication with at least two sites, for example, **usstorage** and **eustorage**.
- Each site has its own Organization, Repository, and image tags.

Procedure

- Sync the blobs between all of your defined sites by running the following command:

```
$ python -m util.backfillreplication
```



WARNING

Prior to removing storage engines from your Red Hat Quay **config.yaml** file, you **must** ensure that all blobs are synced between all defined sites. Complete this step before proceeding.

- In your Red Hat Quay **config.yaml** file for site **usstorage**, remove the **DISTRIBUTED_STORAGE_CONFIG** entry for the **eustorage** site.
- Enter the following command to obtain a list of running containers:

```
$ podman ps
```

Example output

```
CONTAINER ID IMAGE COMMAND NAMES
CREATED STATUS PORTS
92c5321cde38 registry.redhat.io/rhel8/redis-5:1 run-redis 11
days ago Up 11 days ago 0.0.0.0:6379->6379/tcp redis
4e6d1ecd3811 registry.redhat.io/rhel8/postgresql-13:1-109 run-postgresql
33 seconds ago Up 34 seconds ago 0.0.0.0:5432->5432/tcp postgresql-quay
d2eadac74fda registry-proxy.engineering.redhat.com/rh-osbs/quay-quay-rhel8:v3.9.0-131
registry 4 seconds ago Up 4 seconds ago 0.0.0.0:80->8080/tcp, 0.0.0.0:443->8443/tcp
quay
```

4. Enter the following command to execute a shell inside of the PostgreSQL container:

```
$ podman exec -it postgresql-quay -- /bin/bash
```

5. Enter `psql` by running the following command:

```
bash-4.4$ psql
```

6. Enter the following command to reveal a list of sites in your geo-replicated deployment:

```
quay=# select * from imagestoragelocation;
```

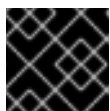
Example output

```
id | name
----+-----
 1 | usstorage
 2 | eustorage
```

7. Enter the following command to exit the postgres CLI to re-enter `bash-4.4`:

```
\q
```

8. Enter the following command to permanently remove the **eustorage** site:



IMPORTANT

The following action cannot be undone. Use with caution.

```
bash-4.4$ python -m util.remove_location eustorage
```

Example output

```
WARNING: This is a destructive operation. Are you sure you want to remove eustorage from
your storage locations? [y/n] y
Deleted placement 30
Deleted placement 31
```

Deleted placement 32
 Deleted placement 33
 Deleted location eustorage

16.2.4. Setting up geo-replication on OpenShift Container Platform

Use the following procedure to set up geo-replication on OpenShift Container Platform.

Procedure

1. Deploy a postgres instance for Red Hat Quay.
2. Login to the database by entering the following command:

```
psql -U <username> -h <hostname> -p <port> -d <database_name>
```

3. Create a database for Red Hat Quay named **quay**. For example:

```
CREATE DATABASE quay;
```

4. Enable pg_trm extension inside the database

```
\c quay;  
CREATE EXTENSION IF NOT EXISTS pg_trgm;
```

5. Deploy a Redis instance:



NOTE

- Deploying a Redis instance might be unnecessary if your cloud provider has its own service.
- Deploying a Redis instance is required if you are leveraging Builders.

- a. Deploy a VM for Redis
- b. Verify that it is accessible from the clusters where Red Hat Quay is running
- c. Port 6379/TCP must be open
- d. Run Redis inside the instance

```
sudo dnf install -y podman  
podman run -d --name redis -p 6379:6379 redis
```

6. Create two object storage backends, one for each cluster. Ideally, one object storage bucket will be close to the first, or primary, cluster, and the other will run closer to the second, or secondary, cluster.
7. Deploy the clusters with the same config bundle, using environment variable overrides to select the appropriate storage backend for an individual cluster.
8. Configure a load balancer to provide a single entry point to the clusters.

16.2.4.1. Configuring geo-replication for the Red Hat Quay on OpenShift Container Platform

Use the following procedure to configure geo-replication for the Red Hat Quay on OpenShift Container Platform.

Procedure

1. Create a **config.yaml** file that is shared between clusters. This **config.yaml** file contains the details for the common PostgreSQL, Redis and storage backends:

Geo-replication config.yaml file

```
SERVER_HOSTNAME: <georep.quayteam.org or any other name> 1
DB_CONNECTION_ARGS:
  autorollback: true
  threadlocals: true
DB_URI: postgresql://postgres:password@10.19.0.1:5432/quay 2
BUILDLOGS_REDIS:
  host: 10.19.0.2
  port: 6379
USER_EVENTS_REDIS:
  host: 10.19.0.2
  port: 6379
DATABASE_SECRET_KEY: 0ce4f796-c295-415b-bf9d-b315114704b8
DISTRIBUTED_STORAGE_CONFIG:
  usstorage:
    - GoogleCloudStorage
    - access_key: GOOGQGPVMSAAMQABCDEFG
      bucket_name: georep-test-bucket-0
      secret_key: AYWfEaxX/u84XRA2vUX5C987654321
      storage_path: /quaygcp
  eustorage:
    - GoogleCloudStorage
    - access_key: GOOGQGPVMSAAMQWERTYUIOP
      bucket_name: georep-test-bucket-1
      secret_key: AYWfEaxX/u84XRA2vUX5Cuj12345678
      storage_path: /quaygcp
DISTRIBUTED_STORAGE_DEFAULT_LOCATIONS:
  - usstorage
  - eustorage
DISTRIBUTED_STORAGE_PREFERENCE:
  - usstorage
  - eustorage
FEATURE_STORAGE_REPLICATION: true
```

- 1 A proper **SERVER_HOSTNAME** must be used for the route and must match the hostname of the global load balancer.
- 2 To retrieve the configuration file for a Clair instance deployed using the OpenShift Container Platform Operator, see [Retrieving the Clair config](#).

2. Create the **configBundleSecret** by entering the following command:

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

- In each of the clusters, set the **configBundleSecret** and use the **QUAY_DISTRIBUTED_STORAGE_PREFERENCE** environmental variable override to configure the appropriate storage for that cluster. For example:



NOTE

The **config.yaml** file between both deployments must match. If making a change to one cluster, it must also be changed in the other.

US cluster QuayRegistry example

```
apiVersion: quay.redhat.com/v1
kind: QuayRegistry
metadata:
  name: example-registry
  namespace: quay-enterprise
spec:
  configBundleSecret: georep-config-bundle
  components:
    - kind: objectstorage
      managed: false
    - kind: route
      managed: true
    - kind: tls
      managed: false
    - kind: postgres
      managed: false
    - kind: clairpostgres
      managed: false
    - kind: redis
      managed: false
    - kind: quay
      managed: true
  overrides:
    env:
      - name: QUAY_DISTRIBUTED_STORAGE_PREFERENCE
        value: usstorage
    - kind: mirror
      managed: true
  overrides:
    env:
      - name: QUAY_DISTRIBUTED_STORAGE_PREFERENCE
        value: usstorage
```



NOTE

Because SSL/TLS is unmanaged, and the route is managed, you must supply the certificates directly in the config bundle. For more information, see [Configuring TLS and routes](#).

European cluster

```

apiVersion: quay.redhat.com/v1
kind: QuayRegistry
metadata:
  name: example-registry
  namespace: quay-enterprise
spec:
  configBundleSecret: georep-config-bundle
  components:
    - kind: objectstorage
      managed: false
    - kind: route
      managed: true
    - kind: tls
      managed: false
    - kind: postgres
      managed: false
    - kind: clairpostgres
      managed: false
    - kind: redis
      managed: false
    - kind: quay
      managed: true
  overrides:
    env:
      - name: QUAY_DISTRIBUTED_STORAGE_PREFERENCE
        value: eustorage
    - kind: mirror
      managed: true
  overrides:
    env:
      - name: QUAY_DISTRIBUTED_STORAGE_PREFERENCE
        value: eustorage

```



NOTE

Because SSL/TLS is unmanaged, and the route is managed, you must supply the certificates directly in the config bundle. For more information, see [Configuring TLS and routes](#).

16.2.5. Removing a geo-replicated site from your Red Hat Quay on OpenShift Container Platform deployment

By using the following procedure, Red Hat Quay administrators can remove sites in a geo-replicated setup.

Prerequisites

- You are logged into OpenShift Container Platform.
- You have configured Red Hat Quay geo-replication with at least two sites, for example, **usstorage** and **eustorage**.
- Each site has its own Organization, Repository, and image tags.

Procedure

Procedure

1. Sync the blobs between all of your defined sites by running the following command:

```
$ python -m util.backfillreplication
```



WARNING

Prior to removing storage engines from your Red Hat Quay **config.yaml** file, you **must** ensure that all blobs are synced between all defined sites.

When running this command, replication jobs are created which are picked up by the replication worker. If there are blobs that need replicated, the script returns UUIDs of blobs that will be replicated. If you run this command multiple times, and the output from the return script is empty, it does not mean that the replication process is done; it means that there are no more blobs to be queued for replication. Customers should use appropriate judgement before proceeding, as the allotted time replication takes depends on the number of blobs detected.

Alternatively, you could use a third party cloud tool, such as Microsoft Azure, to check the synchronization status.

This step must be completed before proceeding.

2. In your Red Hat Quay **config.yaml** file for site **usstorage**, remove the **DISTRIBUTED_STORAGE_CONFIG** entry for the **eustorage** site.
3. Enter the following command to identify your **Quay** application pods:

```
$ oc get pod -n <quay_namespace>
```

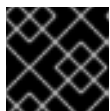
Example output

```
quay390usstorage-quay-app-5779ddc886-2drh2
quay390eustorage-quay-app-66969cd859-n2ssm
```

4. Enter the following command to open an interactive shell session in the **usstorage** pod:

```
$ oc rsh quay390usstorage-quay-app-5779ddc886-2drh2
```

5. Enter the following command to permanently remove the **eustorage** site:



IMPORTANT

The following action cannot be undone. Use with caution.

```
sh-4.4$ python -m util.removelocation eustorage
```

Example output

```
WARNING: This is a destructive operation. Are you sure you want to remove eustorage from
your storage locations? [y/n] y
Deleted placement 30
Deleted placement 31
Deleted placement 32
Deleted placement 33
Deleted location eustorage
```

16.3. MIXED STORAGE FOR GEO-REPLICATION

Red Hat Quay geo-replication supports the use of different and multiple replication targets, for example, using AWS S3 storage on public cloud and using Ceph storage on premise. This complicates the key requirement of granting access to all storage backends from all Red Hat Quay pods and cluster nodes. As a result, it is recommended that you use the following:

- A VPN to prevent visibility of the internal storage, *or*
- A token pair that only allows access to the specified bucket used by Red Hat Quay

This results in the public cloud instance of Red Hat Quay having access to on-premise storage, but the network will be encrypted, protected, and will use ACLs, thereby meeting security requirements.

If you cannot implement these security measures, it might be preferable to deploy two distinct Red Hat Quay registries and to use repository mirroring as an alternative to geo-replication.

CHAPTER 17. BACKING UP AND RESTORING RED HAT QUAY ON A STANDALONE DEPLOYMENT

Use the content within this section to back up and restore Red Hat Quay in standalone deployments.

17.1. OPTIONAL: ENABLING READ-ONLY MODE FOR RED HAT QUAY

Enabling read-only mode for your Red Hat Quay deployment allows you to manage the registry's operations. Red Hat Quay administrators can enable read-only mode to restrict write access to the registry, which helps ensure data integrity, mitigate risks during maintenance windows, and provide a safeguard against unintended modifications to registry data. It also helps to ensure that your Red Hat Quay registry remains online and available to serve images to users.

Prerequisites

- If you are using Red Hat Enterprise Linux (RHEL) 7.x:
 - You have enabled the Red Hat Software Collections List (RHSC).
 - You have installed Python 3.6.
 - You have downloaded the **virtualenv** package.
 - You have installed the **git** CLI.
- If you are using Red Hat Enterprise Linux (RHEL) 8:
 - You have installed Python 3 on your machine.
 - You have downloaded the **python3-virtualenv** package.
 - You have installed the **git** CLI.
- You have cloned the <https://github.com/quay/quay.git> repository.

17.1.1. Creating service keys for standalone Red Hat Quay

Red Hat Quay uses service keys to communicate with various components. These keys are used to sign completed requests, such as requesting to scan images, login, storage access, and so on.

Procedure

1. If your Red Hat Quay registry is readily available, you can generate service keys inside of the **Quay** registry container.
 - a. Enter the following command to generate a key pair inside of the **Quay** container:

```
$ podman exec quay python3 tools/generatekeypair.py quay-readonly
```
2. If your Red Hat Quay is not readily available, you must generate your service keys inside of a virtual environment.
 - a. Change into the directory of your Red Hat Quay deployment and create a virtual environment inside of that directory:

```
$ cd <$QUAY>/quay && virtualenv -v venv
```

- b. Activate the virtual environment by entering the following command:

```
$ source venv/bin/activate
```

- c. Optional. Install the **pip** CLI tool if you do not have it installed:

```
$ venv/bin/pip install --upgrade pip
```

- d. In your Red Hat Quay directory, create a **requirements-generatekeys.txt** file with the following content:

```
$ cat << EOF > requirements-generatekeys.txt
cryptography==3.4.7
pyparser==2.19
pypcryptodome==3.9.4
pypcryptodomex==3.9.4
pyjwt==1.4.2
PyJWT==1.7.1
Authlib==1.0.0a2
EOF
```

- e. Enter the following command to install the Python dependencies defined in the **requirements-generatekeys.txt** file:

```
$ venv/bin/pip install -r requirements-generatekeys.txt
```

- f. Enter the following command to create the necessary service keys:

```
$ PYTHONPATH=. venv/bin/python /<path_to_cloned_repo>/tools/generatekeypair.py
quay-readonly
```

Example output

```
Writing public key to quay-readonly.jwk
Writing key ID to quay-readonly.kid
Writing private key to quay-readonly.pem
```

- g. Enter the following command to deactivate the virtual environment:

```
$ deactivate
```

17.1.2. Adding keys to the PostgreSQL database

Use the following procedure to add your service keys to the PostgreSQL database.

Prerequisites

- You have created the service keys.

Procedure

1. Enter the following command to enter your Red Hat Quay database environment:

```
$ podman exec -it postgresql-quay psql -U postgres -d quay
```

2. Display the approval types and associated notes of the **servicekeyapproval** by entering the following command:

```
quay=# select * from servicekeyapproval;
```

Example output

```
id | approver_id | approval_type | approved_date | notes
-----+-----+-----+-----+-----
 1 |              | ServiceKeyApprovalType.AUTOMATIC | 2024-05-07 03:47:48.181347 |
 2 |              | ServiceKeyApprovalType.AUTOMATIC | 2024-05-07 03:47:55.808087 |
 3 |              | ServiceKeyApprovalType.AUTOMATIC | 2024-05-07 03:49:04.27095 |
 4 |              | ServiceKeyApprovalType.AUTOMATIC | 2024-05-07 03:49:05.46235 |
 5 |            1 | ServiceKeyApprovalType.SUPERUSER | 2024-05-07 04:05:10.296796 |
...
```

3. Add the service key to your Red Hat Quay database by entering the following query:

```
quay=# INSERT INTO servicekey
(name, service, metadata, kid, jwk, created_date, expiration_date)
VALUES ('quay-readonly',
       'quay',
       '{}',
       '{<contents_of_.kid_file>}',
       '{<contents_of_.jwk_file>}',
       '{<created_date_of_read-only>}',
       '{<expiration_date_of_read-only>}');
```

Example output

```
INSERT 0 1
```

4. Next, add the key approval with the following query:

```
quay=# INSERT INTO servicekeyapproval ('approval_type', 'approved_date', 'notes')
VALUES ("ServiceKeyApprovalType.SUPERUSER", "CURRENT_DATE",
       {include_notes_here_on_why_this_is_being_added});
```

Example output

```
INSERT 0 1
```

5. Set the **approval_id** field on the created service key row to the **id** field from the created service key approval. You can use the following **SELECT** statements to get the necessary IDs:

```
UPDATE servicekey
SET approval_id = (SELECT id FROM servicekeyapproval WHERE approval_type =
```

```
'ServiceKeyApprovalType.SUPERUSER')
WHERE name = 'quay-readonly';
```

```
UPDATE 1
```

17.1.3. Configuring read-only mode for standalone Red Hat Quay

After the service keys have been created and added to your PostgreSQL database, you must restart the **Quay** container on your standalone deployment.

Prerequisites

- You have created the service keys and added them to your PostgreSQL database.

Procedure

- Shutdown all Red Hat Quay instances on all virtual machines. For example:

```
$ podman stop <quay_container_name_on_virtual_machine_a>
```

```
$ podman stop <quay_container_name_on_virtual_machine_b>
```

- Enter the following command to copy the contents of the **quay-readonly.kid** file and the **quay-readonly.pem** file to the directory that holds your Red Hat Quay configuration bundle:

```
$ cp quay-readonly.kid quay-readonly.pem $Quay/config
```

- Enter the following command to set file permissions on all files in your configuration bundle folder:

```
$ setfacl -m user:1001:rw $Quay/config/*
```

- Modify your Red Hat Quay **config.yaml** file and add the following information:

```
# ...
REGISTRY_STATE: readonly
INSTANCE_SERVICE_KEY_KID_LOCATION: 'conf/stack/quay-readonly.kid'
INSTANCE_SERVICE_KEY_LOCATION: 'conf/stack/quay-readonly.pem'
# ...
```

- Distribute the new configuration bundle to all Red Hat Quay instances.
- Start Red Hat Quay by entering the following command:

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

- After starting Red Hat Quay, a banner inside in your instance informs users that Red Hat Quay is running in read-only mode. Pushes should be rejected and a 405 error should be logged. You can test this by running the following command:

```
$ podman push <quay-server.example.com>/quayadmin/busybox:test
```

Example output

```
613be09ab3c0: Preparing
denied: System is currently read-only. Pulls will succeed but all write operations are currently
suspended.
```

With your Red Hat Quay deployment on read-only mode, you can safely manage your registry's operations and perform such actions as backup and restore.

- Optional. After you are finished with read-only mode, you can return to normal operations by removing the following information from your **config.yaml** file. Then, restart your Red Hat Quay deployment:

```
# ...
REGISTRY_STATE: readonly
INSTANCE_SERVICE_KEY_KID_LOCATION: 'conf/stack/quay-readonly.kid'
INSTANCE_SERVICE_KEY_LOCATION: 'conf/stack/quay-readonly.pem'
# ...
```

```
$ podman restart <container_id>
```

17.1.4. Updating read-only expiration time

The Red Hat Quay read-only key has an expiration date, and when that date passes the key is deactivated. Before the key expires, its expiration time can be updated in the database. To update the key, connect your Red Hat Quay production database using the methods described earlier and issue the following query:

```
quay=# UPDATE servicekey SET expiration_date = 'new-date' WHERE id = servicekey_id;
```

The list of service key IDs can be obtained by running the following query:

```
SELECT id, name, expiration_date FROM servicekey;
```

17.2. BACKING UP RED HAT QUAY ON STANDALONE DEPLOYMENTS

This procedure describes how to create a backup of Red Hat Quay on standalone deployments.

Procedure

- Create a temporary backup directory, for example, **quay-backup**:

```
$ mkdir /tmp/quay-backup
```

- The following example command denotes the local directory that the Red Hat Quay was started in, for example, **/opt/quay-install**:

```
$ podman run --name quay-app \
-v /opt/quay-install/config:/conf/stack:Z \
-v /opt/quay-install/storage:/datastorage:Z \
registry.redhat.io/quay/quay-rhel8:v3.11.1
```

Change into the directory that bind-mounts to **/conf/stack** inside of the container, for example, **/opt/quay-install**, by running the following command:

```
$ cd /opt/quay-install
```

3. Compress the contents of your Red Hat Quay deployment into an archive in the **quay-backup** directory by entering the following command:

```
$ tar cvf /tmp/quay-backup/quay-backup.tar.gz *
```

Example output:

```
config.yaml
config.yaml.bak
extra_ca_certs/
extra_ca_certs/ca.crt
ssl.cert
ssl.key
```

4. Back up the Quay container service by entering the following command:

```
$ podman inspect quay-app | jq -r '[0].Config.CreateCommand | .[]' | paste -s -d ' ' -

/usr/bin/podman run --name quay-app \
-v /opt/quay-install/config:/conf/stack:Z \
-v /opt/quay-install/storage:/datastorage:Z \
registry.redhat.io/quay/quay-rhel8:v3.11.1
```

5. Redirect the contents of your **conf/stack/config.yaml** file to your temporary **quay-config.yaml** file by entering the following command:

```
$ podman exec -it quay cat /conf/stack/config.yaml > /tmp/quay-backup/quay-config.yaml
```

6. Obtain the **DB_URI** located in your temporary **quay-config.yaml** by entering the following command:

```
$ grep DB_URI /tmp/quay-backup/quay-config.yaml
```

Example output:

```
$ postgresql://<username>:test123@172.24.10.50/quay
```

7. Extract the PostgreSQL contents to your temporary backup directory in a backup .sql file by entering the following command:

```
$ pg_dump -h 172.24.10.50 -p 5432 -d quay -U <username> -W -O > /tmp/quay-backup/quay-backup.sql
```


- Print the contents of your **DISTRIBUTED_STORAGE_CONFIG** by entering the following command:

```
DISTRIBUTED_STORAGE_CONFIG:
default:
- S3Storage
- s3_bucket: <bucket_name>
  storage_path: /registry
  s3_access_key: <s3_access_key>
  s3_secret_key: <s3_secret_key>
  host: <host_name>
```

- Export the **AWS_ACCESS_KEY_ID** by using the **access_key** credential obtained in Step 7:

```
$ export AWS_ACCESS_KEY_ID=<access_key>
```

- Export the **AWS_SECRET_ACCESS_KEY** by using the **secret_key** obtained in Step 7:

```
$ export AWS_SECRET_ACCESS_KEY=<secret_key>
```

- Sync the **quay** bucket to the **/tmp/quay-backup/blob-backup/** directory from the **hostname** of your **DISTRIBUTED_STORAGE_CONFIG**:

```
$ aws s3 sync s3://<bucket_name> /tmp/quay-backup/blob-backup/ --source-region us-east-2
```

Example output:

```
download:
s3://<user_name>/registry/sha256/9c/9c3181779a868e09698b567a3c42f3744584ddb1398efe2c4ba569a99b823f7a to
registry/sha256/9c/9c3181779a868e09698b567a3c42f3744584ddb1398efe2c4ba569a99b823f7a
download:
s3://<user_name>/registry/sha256/e9/e9c5463f15f0fd62df3898b36ace8d15386a6813ffb470f332698ecb34af5b0d to
registry/sha256/e9/e9c5463f15f0fd62df3898b36ace8d15386a6813ffb470f332698ecb34af5b0d
```

It is recommended that you delete the **quay-config.yaml** file after syncing the **quay** bucket because it contains sensitive information. The **quay-config.yaml** file will not be lost because it is backed up in the **quay-backup.tar.gz** file.

17.3. RESTORING RED HAT QUAY ON STANDALONE DEPLOYMENTS

This procedure describes how to restore Red Hat Quay on standalone deployments.

Prerequisites

- You have backed up your Red Hat Quay deployment.

Procedure

1. Create a new directory that will bind-mount to **/conf/stack** inside of the Red Hat Quay container:

```
$ mkdir /opt/new-quay-install
```

2. Copy the contents of your temporary backup directory created in [Backing up Red Hat Quay on standalone deployments](#) to the **new-quay-install1** directory created in Step 1:

```
$ cp /tmp/quay-backup/quay-backup.tar.gz /opt/new-quay-install/
```

3. Change into the **new-quay-install** directory by entering the following command:

```
$ cd /opt/new-quay-install/
```

4. Extract the contents of your Red Hat Quay directory:

```
$ tar xvf /tmp/quay-backup/quay-backup.tar.gz *
```

Example output:

```
config.yaml
config.yaml.bak
extra_ca_certs/
extra_ca_certs/ca.crt
ssl.cert
ssl.key
```

5. Recall the **DB_URI** from your backed-up **config.yaml** file by entering the following command:

```
$ grep DB_URI config.yaml
```

Example output:

```
postgresql://<username>:test123@172.24.10.50/quay
```

6. Run the following command to enter the PostgreSQL database server:

```
$ sudo postgres
```

7. Enter psql and create a new database in 172.24.10.50 to restore the quay databases, for example, **example_restore_registry_quay_database**, by entering the following command:

```
$ psql "host=172.24.10.50 port=5432 dbname=postgres user=<username>
password=test123"
postgres=> CREATE DATABASE example_restore_registry_quay_database;
```

Example output:

```
CREATE DATABASE
```

8. Connect to the database by running the following command:

```
postgres=# \c "example-restore-registry-quay-database";
```

Example output:

```
You are now connected to database "example-restore-registry-quay-database" as user
"postgres".
```

9. Create a **pg_trgm** extension of your Quay database by running the following command:

```
example_restore_registry_quay_database=> CREATE EXTENSION IF NOT EXISTS
pg_trgm;
```

Example output:

```
CREATE EXTENSION
```

10. Exit the postgres CLI by entering the following command:

```
\q
```

11. Import the database backup to your new database by running the following command:

```
$ psql "host=172.24.10.50 port=5432 dbname=example_restore_registry_quay_database
user=<username> password=test123" -W < /tmp/quay-backup/quay-backup.sql
```

Example output:

```
SET
SET
SET
SET
SET
```

Update the value of **DB_URI** in your **config.yaml** from **postgresql://<username>:test123@172.24.10.50/quay** to **postgresql://<username>:test123@172.24.10.50/example-restore-registry-quay-database** before restarting the Red Hat Quay deployment.



NOTE

The **DB_URI** format is **DB_URI postgresql://<login_user_name>:<login_user_password>@<postgresql_host>/<quay_database>**. If you are moving from one PostgreSQL server to another PostgreSQL server, update the value of **<login_user_name>**, **<login_user_password>** and **<postgresql_host>** at the same time.

12. In the **/opt/new-quay-install** directory, print the contents of your **DISTRIBUTED_STORAGE_CONFIG** bundle:

```
$ cat config.yaml | grep DISTRIBUTED_STORAGE_CONFIG -A10
```

Example output:

```
DISTRIBUTED_STORAGE_CONFIG:
  default:
DISTRIBUTED_STORAGE_CONFIG:
  default:
  - S3Storage
  - s3_bucket: <bucket_name>
    storage_path: /registry
    s3_access_key: <s3_access_key>
    s3_secret_key: <s3_secret_key>
    host: <host_name>
```



NOTE

Your **DISTRIBUTED_STORAGE_CONFIG** in **/opt/new-quay-install** must be updated before restarting your Red Hat Quay deployment.

13. Export the **AWS_ACCESS_KEY_ID** by using the **access_key** credential obtained in Step 13:

```
$ export AWS_ACCESS_KEY_ID=<access_key>
```

14. Export the **AWS_SECRET_ACCESS_KEY** by using the **secret_key** obtained in Step 13:

```
$ export AWS_SECRET_ACCESS_KEY=<secret_key>
```

15. Create a new s3 bucket by entering the following command:

```
$ aws s3 mb s3://<new_bucket_name> --region us-east-2
```

Example output:

```
$ make_bucket: quay
```

16. Upload all blobs to the new s3 bucket by entering the following command:

```
$ aws s3 sync --no-verify-ssl \
  --endpoint-url <example_endpoint_url> 1
  /tmp/quay-backup/blob-backup/. s3://quay/
```

- 1** The Red Hat Quay registry endpoint must be the same before backup and after restore.

Example output:

```
upload: ../../tmp/quay-backup/blob-
backup/datastorage/registry/sha256/50/505edb46ea5d32b5cbe275eb766d960842a52ee77ac2
25e4dc8abb12f409a30d to
s3://quay/datastorage/registry/sha256/50/505edb46ea5d32b5cbe275eb766d960842a52ee77ac
225e4dc8abb12f409a30d
upload: ../../tmp/quay-backup/blob-
backup/datastorage/registry/sha256/27/27930dc06c2ee27ac6f543ba0e93640dd21eea458eac4
7355e8e5989dea087d0 to
s3://quay/datastorage/registry/sha256/27/27930dc06c2ee27ac6f543ba0e93640dd21eea458ea
c47355e8e5989dea087d0
```

```
upload: ../../tmp/quay-backup/blob-  
backup/datastorage/registry/sha256/8c/8c7daf5e20eee45ffe4b36761c4bb6729fb3ee60d4f588f  
712989939323110ec to  
s3://quay/datastorage/registry/sha256/8c/8c7daf5e20eee45ffe4b36761c4bb6729fb3ee60d4f58  
8f712989939323110ec  
...
```

17. Before restarting your Red Hat Quay deployment, update the storage settings in your config.yaml:

```
DISTRIBUTED_STORAGE_CONFIG:  
  default:  
DISTRIBUTED_STORAGE_CONFIG:  
  default:  
  - S3Storage  
  - s3_bucket: <new_bucket_name>  
    storage_path: /registry  
    s3_access_key: <s3_access_key>  
    s3_secret_key: <s3_secret_key>  
    host: <host_name>
```

CHAPTER 18. MIGRATING A STANDALONE RED HAT QUAY DEPLOYMENT TO A RED HAT QUAY OPERATOR DEPLOYMENT

The following procedures allow you to back up a standalone Red Hat Quay deployment and migrate it to the Red Hat Quay Operator on OpenShift Container Platform.

18.1. BACKING UP A STANDALONE DEPLOYMENT OF RED HAT QUAY

Procedure

1. Back up the **config.yaml** of your standalone Red Hat Quay deployment:

```
$ mkdir /tmp/quay-backup
$ cp /path/to/Quay/config/directory/config.yaml /tmp/quay-backup
```

2. Create a backup of the database that your standalone Red Hat Quay deployment is using:

```
$ pg_dump -h DB_HOST -p 5432 -d QUAY_DATABASE_NAME -U
QUAY_DATABASE_USER -W -O > /tmp/quay-backup/quay-database-backup.sql
```

3. Install the [AWS CLI](#) if you do not have it already.

4. Create an `~/.aws/` directory:

```
$ mkdir ~/.aws/
```

5. Obtain the **access_key** and **secret_key** from the **config.yaml** of your standalone deployment:

```
$ grep -i DISTRIBUTED_STORAGE_CONFIG -A10 /tmp/quay-backup/config.yaml
```

Example output:

```
DISTRIBUTED_STORAGE_CONFIG:
  minio-1:
    - RadosGWStorage
    - access_key: #####
      bucket_name: quay
      hostname: 172.24.10.50
      is_secure: false
      port: "9000"
      secret_key: #####
      storage_path: /datastorage/registry
```

6. Store the **access_key** and **secret_key** from the **config.yaml** file in your `~/.aws` directory:

```
$ touch ~/.aws/credentials
```

7. Optional: Check that your **access_key** and **secret_key** are stored:

```
$ cat > ~/.aws/credentials << EOF
```

```
[default]
aws_access_key_id = ACCESS_KEY_FROM_QUAY_CONFIG
aws_secret_access_key = SECRET_KEY_FROM_QUAY_CONFIG
EOF
```

Example output:

```
aws_access_key_id = ACCESS_KEY_FROM_QUAY_CONFIG
aws_secret_access_key = SECRET_KEY_FROM_QUAY_CONFIG
```



NOTE

If the **aws cli** does not automatically collect the **access_key** and **secret_key** from the `~/.aws/credentials` file, you can, you can configure these by running **aws configure** and manually inputting the credentials.

- In your **quay-backup** directory, create a **bucket_backup** directory:

```
$ mkdir /tmp/quay-backup/bucket-backup
```

- Backup all blobs from the S3 storage:

```
$ aws s3 sync --no-verify-ssl --endpoint-url https://PUBLIC_S3_ENDPOINT:PORT
s3://QUAY_BUCKET/ /tmp/quay-backup/bucket-backup/
```



NOTE

The **PUBLIC_S3_ENDPOINT** can be read from the Red Hat Quay **config.yaml** file under **hostname** in the **DISTRIBUTED_STORAGE_CONFIG**. If the endpoint is insecure, use **http** instead of **https** in the endpoint URL.

Up to this point, you should have a complete backup of all Red Hat Quay data, blobs, the database, and the **config.yaml** file stored locally. In the following section, you will migrate the standalone deployment backup to Red Hat Quay on OpenShift Container Platform.

18.2. USING BACKED UP STANDALONE CONTENT TO MIGRATE TO OPENSIFT CONTAINER PLATFORM.

Prerequisites

- Your standalone Red Hat Quay data, blobs, database, and **config.yaml** have been backed up.
- Red Hat Quay is deployed on OpenShift Container Platform using the Red Hat Quay Operator.
- A **QuayRegistry** with all components set to **managed**.



PROCEDURE

The procedure in this documents uses the following namespace: **quay-enterprise**.

- Scale down the Red Hat Quay Operator:

```
$ oc scale --replicas=0 deployment quay-operator.v3.6.2 -n openshift-operators
```

- Scale down the application and mirror deployments:

```
$ oc scale --replicas=0 deployment QUAY_MAIN_APP_DEPLOYMENT
QUAY_MIRROR_DEPLOYMENT
```

- Copy the database SQL backup to the **Quay** PostgreSQL database instance:

```
$ oc cp /tmp/user/quay-backup/quay-database-backup.sql quay-enterprise/quayregistry-
quay-database-54956cdd54-p7b2w:/var/lib/pgsql/data/userdata
```

- Obtain the database password from the Operator-created **config.yaml** file:

```
$ oc get deployment quay-quay-app -o json | jq
'.spec.template.spec.volumes[].projected.sources' | grep -i config-secret
```

Example output:

```
"name": "QUAY_CONFIG_SECRET_NAME"
```

```
$ oc get secret quay-quay-config-secret-9t77hb84tb -o json | jq '.data."config.yaml"' | cut -d ""
-f2 | base64 -d -w0 > /tmp/quay-backup/operator-quay-config-yaml-backup.yaml
```

```
cat /tmp/quay-backup/operator-quay-config-yaml-backup.yaml | grep -i DB_URI
```

Example output:

```
postgresql://QUAY_DATABASE_OWNER:PASSWORD@DATABASE_HOST/QUAY_DATAB
ASE_NAME
```

- Execute a shell inside of the database pod:

```
# oc exec -it quay-postgresql-database-pod -- /bin/bash
```

- Enter `psql`:

```
bash-4.4$ psql
```

- Drop the database:

```
postgres=# DROP DATABASE "example-restore-registry-quay-database";
```

Example output:

```
DROP DATABASE
```

- Create a new database and set the owner as the same name:

```
postgres=# CREATE DATABASE "example-restore-registry-quay-database" OWNER
"example-restore-registry-quay-database";
```


-

Example output:

```
CREATE DATABASE
```

9. Connect to the database:

```
postgres=# \c "example-restore-registry-quay-database";
```

Example output:

```
You are now connected to database "example-restore-registry-quay-database" as user
"postgres".
```

10. Create a **pg_trgm** extension of your **Quay** database:

```
example-restore-registry-quay-database=# create extension pg_trgm ;
```

Example output:

```
CREATE EXTENSION
```

11. Exit the postgres CLI to re-enter bash-4.4:

```
\q
```

12. Set the password for your PostgreSQL deployment:

```
bash-4.4$ psql -h localhost -d "QUAY_DATABASE_NAME" -U QUAY_DATABASE_OWNER
-W < /var/lib/pgsql/data/userdata/quay-database-backup.sql
```

Example output:

```
SET
SET
SET
SET
SET
```

13. Exit bash mode:

```
bash-4.4$ exit
```

14. Create a new configuration bundle for the Red Hat Quay Operator.

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

15. In your new **config-bundle.yaml**, include all of the information that the registry requires, such as LDAP configuration, keys, and other modifications that your old registry had. Run the following command to move the **secret_key** to your **config-bundle.yaml**:

```
$ cat /tmp/quay-backup/config.yaml | grep SECRET_KEY > /tmp/quay-backup/config-bundle.yaml
```



NOTE

You must manually copy all the LDAP, OIDC and other information and add it to the `/tmp/quay-backup/config-bundle.yaml` file.

16. Create a configuration bundle secret inside of your OpenShift cluster:

```
$ oc create secret generic new-custom-config-bundle --from-file=config.yaml=/tmp/quay-backup/config-bundle.yaml
```

17. Scale up the **Quay** pods:

```
$ oc scale --replicas=1 deployment quayregistry-quay-app deployment.apps/quayregistry-quay-app scaled
```

18. Scale up the mirror pods:

```
$ oc scale --replicas=1 deployment quayregistry-quay-mirror deployment.apps/quayregistry-quay-mirror scaled
```

19. Patch the **QuayRegistry** CRD so that it contains the reference to the new custom configuration bundle:

```
$ oc patch quayregistry QUAY_REGISTRY_NAME --type=merge -p '{"spec": {"configBundleSecret": "new-custom-config-bundle"}}'
```



NOTE

If Red Hat Quay returns a **500** internal server error, you might have to update the **location** of your **DISTRIBUTED_STORAGE_CONFIG** to **default**.

20. Create a new AWS **credentials.yaml** in your `./aws/` directory and include the **access_key** and **secret_key** from the Operator-created **config.yaml** file:

```
$ touch credentials.yaml
```

```
$ grep -i DISTRIBUTED_STORAGE_CONFIG -A10 /tmp/quay-backup/operator-quay-config-yaml-backup.yaml
```

```
$ cat > ~/.aws/credentials << EOF
[default]
aws_access_key_id = ACCESS_KEY_FROM_QUAY_CONFIG
aws_secret_access_key = SECRET_KEY_FROM_QUAY_CONFIG
EOF
```

**NOTE**

If the **aws cli** does not automatically collect the **access_key** and **secret_key** from the `~/.aws/credentials` file, you can configure these by running **aws configure** and manually inputting the credentials.

21. Record the NooBaa's publicly available endpoint:

```
$ oc get route s3 -n openshift-storage -o yaml -o jsonpath="{.spec.host}{'\n'}"
```

22. Sync the backup data to the NooBaa backend storage:

```
$ aws s3 sync --no-verify-ssl --endpoint-url https://NOOBAA_PUBLIC_S3_ROUTE  
/tmp/quay-backup/bucket-backup/* s3://QUAY_DATASTORE_BUCKET_NAME
```

23. Scale the Operator back up to 1 pod:

```
$ oc scale --replicas=1 deployment quay-operator.v3.6.4 -n openshift-operators
```

The Operator uses the custom configuration bundle provided and reconciles all secrets and deployments. Your new Red Hat Quay deployment on OpenShift Container Platform should contain all of the information that the old deployment had. You should be able to pull all images.

CHAPTER 19. CONFIGURING ARTIFACT TYPES

As a Red Hat Quay administrator, you can configure Open Container Initiative (OCI) artifact types and other experimental artifact types through the **FEATURE_GENERAL_OCI_SUPPORT**, **ALLOWED_OCI_ARTIFACT_TYPES**, and **IGNORE_UNKNOWN_MEDIATYPES** configuration fields.

The following Open Container Initiative (OCI) artifact types are built into Red Hat Quay by default and are enabled through the **FEATURE_GENERAL_OCI_SUPPORT** configuration field:

Field	Media Type	Supported content types
Helm	<code>application/vnd.cncf.helm.config.v1+json</code>	<code>application/tar+gzip</code> , <code>application/vnd.cncf.helm.chart.content.v1.tar+gzip</code>
Cosign	<code>application/vnd.oci.image.config.v1+json</code>	<code>application/vnd.dev.cosign.unsigned.v1+json</code> , <code>application/vnd.dsse.envelope.v1+json</code>
SPDX	<code>application/vnd.oci.image.config.v1+json</code>	<code>text/spdx</code> , <code>text/spdx+xml</code> , <code>text/spdx+json</code>
Syft	<code>application/vnd.oci.image.config.v1+json</code>	<code>application/vnd.syft+json</code>
CycloneDX	<code>application/vnd.oci.image.config.v1+json</code>	<code>application/vnd.cyclonedx</code> , <code>application/vnd.cyclonedx+xml</code> , <code>application/vnd.cyclonedx+json</code>
In-toto	<code>application/vnd.oci.image.config.v1+json</code>	<code>application/vnd.in-toto+json</code>
Unknown	<code>application/vnd.cncf.openpolicyagent.policy.layer.v1+rego</code>	<code>application/vnd.cncf.openpolicyagent.policy.layer.v1+rego</code> , <code>application/vnd.cncf.openpolicyagent.data.layer.v1+json</code>

Additionally, Red Hat Quay uses the *ZStandard*, or *zstd*, to reduce the size of container images or other related artifacts. Zstd helps optimize storage and improve transfer speeds when working with container images.

Use the following procedures to configure support for the default and experimental OCI media types.

19.1. CONFIGURING OCI ARTIFACT TYPES

Use the following procedure to configure artifact types that are embedded in Red Hat Quay by default.

Prerequisites

- You have Red Hat Quay administrator privileges.

Procedure

- In your Red Hat Quay **config.yaml** file, enable support for general OCI support by setting the **FEATURE_GENERAL_OCI_SUPPORT** field to **true**. For example:

```
FEATURE_GENERAL_OCI_SUPPORT: true
```

With **FEATURE_GENERAL_OCI_SUPPORT** set to true, Red Hat Quay users can now push and pull charts of the default artifact types to their Red Hat Quay deployment.

19.2. CONFIGURING ADDITIONAL ARTIFACT TYPES

Use the following procedure to configure additional, and specific, artifact types for your Red Hat Quay deployment.



NOTE

Using the **ALLOWED_OCI_ARTIFACT_TYPES** configuration field, you can restrict which artifact types are accepted by your Red Hat Quay registry. If you want your Red Hat Quay deployment to accept all artifact types, see "Configuring unknown media types".

Prerequisites

- You have Red Hat Quay administrator privileges.

Procedure

- Add the **ALLOWED_OCI_ARTIFACT_TYPES** configuration field, along with the configuration and layer types:

```
FEATURE_GENERAL_OCI_SUPPORT: true
ALLOWED_OCI_ARTIFACT_TYPES:
  <oci config type 1>:
  - <oci layer type 1>
  - <oci layer type 2>

  <oci config type 2>:
  - <oci layer type 3>
  - <oci layer type 4>
```

For example, you can add Singularity Image Format (SIF) support by adding the following to your **config.yaml** file:

ALLOWED_OCI_ARTIFACT_TYPES:

```
application/vnd.oci.image.config.v1+json:  
- application/vnd.dev.cosign.simplesigning.v1+json  
application/vnd.cncf.helm.config.v1+json:  
- application/tar+gzip  
application/vnd.sylabs.sif.config.v1+json:  
- application/vnd.sylabs.sif.layer.v1+tar
```

**NOTE**

When adding OCI artifact types that are not configured by default, Red Hat Quay administrators will also need to manually add support for Cosign and Helm if desired.

Now, users can tag SIF images for their Red Hat Quay registry.

19.3. CONFIGURING UNKNOWN MEDIA TYPES

Use the following procedure to enable all artifact types for your Red Hat Quay deployment.

**NOTE**

With this field enabled, your Red Hat Quay deployment accepts all artifact types.

Prerequisites

- You have Red Hat Quay administrator privileges.

Procedure

1. Add the **IGNORE_UNKNOWN_MEDIATYPES** configuration field to your Red Hat Quay **config.yaml** file:

```
IGNORE_UNKNOWN_MEDIATYPES: true
```

With this field enabled, your Red Hat Quay deployment accepts unknown and unrecognized artifact types.

CHAPTER 20. RED HAT QUAY GARBAGE COLLECTION

Red Hat Quay includes automatic and continuous image garbage collection. Garbage collection ensures efficient use of resources for active objects by removing objects that occupy sizeable amounts of disk space, such as dangling or untagged images, repositories, and blobs, including layers and manifests. Garbage collection performed by Red Hat Quay can reduce downtime in your organization's environment.

20.1. RED HAT QUAY GARBAGE COLLECTION IN PRACTICE

Currently, all garbage collection happens discreetly, and there are no commands to manually run garbage collection. Red Hat Quay provides metrics that track the status of the different garbage collection workers.

For namespace and repository garbage collection, the progress is tracked based on the size of their respective queues. Namespace and repository garbage collection workers require a global lock to work. As a result, and for performance reasons, only one worker runs at a time.



NOTE

Red Hat Quay shares blobs between namespaces and repositories in order to conserve disk space. For example, if the same image is pushed 10 times, only one copy of that image will be stored.

It is possible that tags can share their layers with different images already stored somewhere in Red Hat Quay. In that case, blobs will stay in storage, because deleting shared blobs would make other images unusable.

Blob expiration is independent of the time machine. If you push a tag to Red Hat Quay and the time machine is set to 0 seconds, and then you delete a tag immediately, garbage collection deletes the tag and everything related to that tag, but will not delete the blob storage until the blob expiration time is reached.

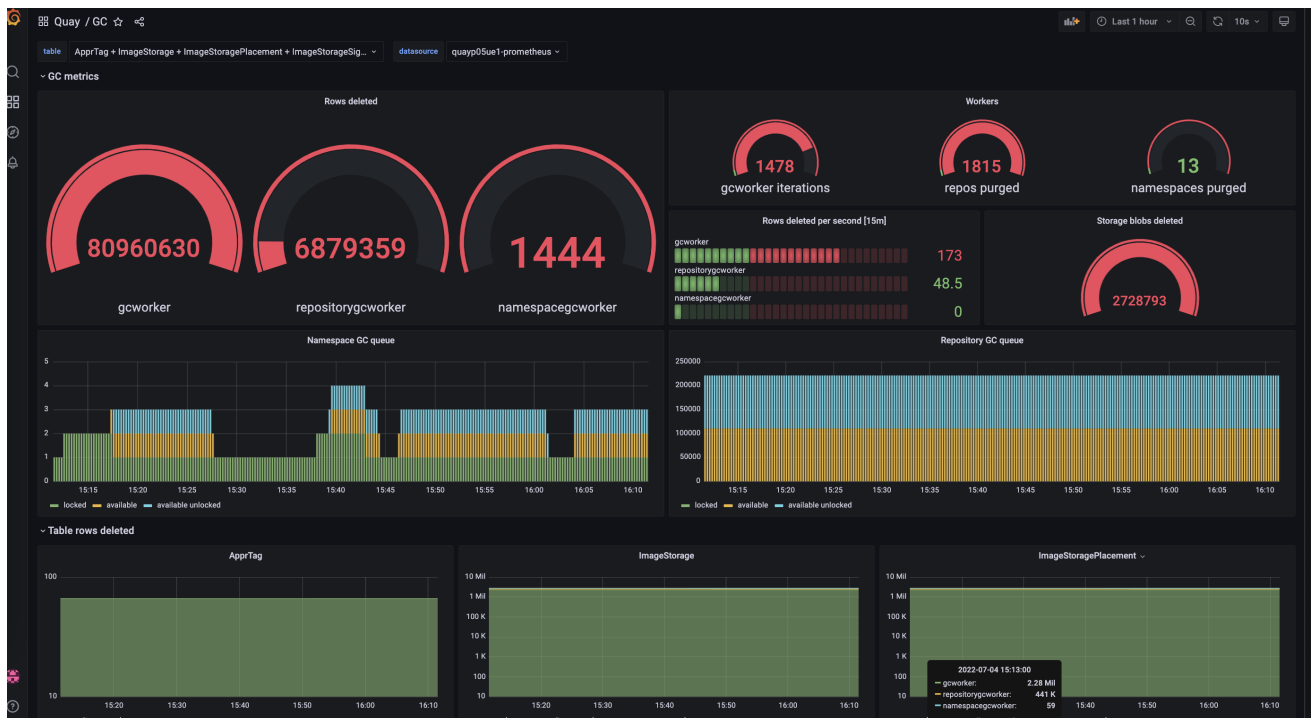
Garbage collecting tagged images works differently than garbage collection on namespaces or repositories. Rather than having a queue of items to work with, the garbage collection workers for tagged images actively search for a repository with inactive or expired tags to clean up. Each instance of garbage collection workers will grab a repository lock, which results in one worker per repository.



NOTE

- In Red Hat Quay, inactive or expired tags are manifests without tags because the last tag was deleted or it expired. The manifest stores information about how the image is composed and stored in the database for each individual tag. When a tag is deleted and the allotted time from **Time Machine** has been met, Red Hat Quay garbage collects the blobs that are not connected to any other manifests in the registry. If a particular blob is connected to a manifest, then it is preserved in storage and only its connection to the manifest that is being deleted is removed.
- Expired images will disappear after the allotted time, but are still stored in Red Hat Quay. The time in which an image is completely deleted, or collected, depends on the **Time Machine** setting of your organization. The default time for garbage collection is 14 days unless otherwise specified. Until that time, tags can be pointed to an expired or deleted images.

For each type of garbage collection, Red Hat Quay provides metrics for the number of rows per table deleted by each garbage collection worker. The following image shows an example of how Red Hat Quay monitors garbage collection with the same metrics:



20.1.1. Measuring storage reclamation

Red Hat Quay does not have a way to track how much space is freed up by garbage collection. Currently, the best indicator of this is by checking how many blobs have been deleted in the provided metrics.



NOTE

The **UploadedBlob** table in the Red Hat Quay metrics tracks the various blobs that are associated with a repository. When a blob is uploaded, it will not be garbage collected before the time designated by the **PUSH_TEMP_TAG_EXPIRATION_SEC** parameter. This is to avoid prematurely deleting blobs that are part of an ongoing push. For example, if garbage collection is set to run often, and a tag is deleted in the span of less than one hour, then it is possible that the associated blobs will not get cleaned up immediately. Instead, and assuming that the time designated by the **PUSH_TEMP_TAG_EXPIRATION_SEC** parameter has passed, the associated blobs will be removed the next time garbage collection is triggered to run by another expired tag on the same repository.

20.2. GARBAGE COLLECTION CONFIGURATION FIELDS

The following configuration fields are available to customize what is garbage collected, and the frequency at which garbage collection occurs:

Name	Description	Schema
------	-------------	--------

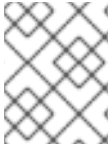
Name	Description	Schema
FEATURE_GARBAGE_COLLECTION	Whether garbage collection is enabled for image tags. Defaults to true .	Boolean
FEATURE_NAMESPACE_GARBAGE_COLLECTION	Whether garbage collection is enabled for namespaces. Defaults to true .	Boolean
FEATURE_REPOSITORY_GARBAGE_COLLECTION	Whether garbage collection is enabled for repositories. Defaults to true .	Boolean
GARBAGE_COLLECTION_FREQUENCY	The frequency, in seconds, at which the garbage collection worker runs. Affects only garbage collection workers. Defaults to 30 seconds.	String

Name	Description	Schema
<code>PUSH_TEMP_TAG_EXPIRATION_SEC</code>	The number of seconds that blobs will not be garbage collected after being uploaded. This feature prevents garbage collection from cleaning up blobs that are not referenced yet, but still used as part of an ongoing push.	String
<code>TAG_EXPIRATION_OPTIONS</code>	List of valid tag expiration values.	String
<code>DEFAULT_TAG_EXPIRATION</code>	Tag expiration time for time machine.	String
<code>CLEAN_BLOB_UPLOAD_FOLDER</code>	Automatically cleans stale blobs left over from an S3 multipart upload. By default, blob files older than two days are cleaned up every hour.	Boolean + Default: true

20.3. DISABLING GARBAGE COLLECTION

The garbage collection features for image tags, namespaces, and repositories are stored in the **config.yaml** file. These features default to **true**.

In rare cases, you might want to disable garbage collection, for example, to control when garbage collection is performed. You can disable garbage collection by setting the **GARBAGE_COLLECTION** features to **false**. When disabled, dangling or untagged images, repositories, namespaces, layers, and manifests are not removed. This might increase the downtime of your environment.



NOTE

There is no command to manually run garbage collection. Instead, you would disable, and then re-enable, the garbage collection feature.

20.4. GARBAGE COLLECTION AND QUOTA MANAGEMENT

Red Hat Quay introduced quota management in 3.7. With quota management, users have the ability to report storage consumption and to contain registry growth by establishing configured storage quota limits.

As of Red Hat Quay 3.7, garbage collection reclaims memory that was allocated to images, repositories, and blobs after deletion. Because the garbage collection feature reclaims memory after deletion, there is a discrepancy between what is stored in an environment's disk space and what quota management is reporting as the total consumption. There is currently no workaround for this issue.

20.5. GARBAGE COLLECTION IN PRACTICE

Use the following procedure to check your Red Hat Quay logs to ensure that garbage collection is working.

Procedure

1. Enter the following command to ensure that garbage collection is properly working:

```
$ sudo podman logs <container_id>
```

Example output:

```
gcworker stdout | 2022-11-14 18:46:52,458 [63] [INFO] [apscheduler.executors.default] Job
"GarbageCollectionWorker._garbage_collection_repos (trigger: interval[0:00:30], next run at:
2022-11-14 18:47:22 UTC)" executed successfully
```

2. Delete an image tag.
3. Enter the following command to ensure that the tag was deleted:

```
$ podman logs quay-app
```

Example output:

```
unicorn-web stdout | 2022-11-14 19:23:44,574 [233] [INFO] [unicorn.access] 192.168.0.38
- - [14/Nov/2022:19:23:44 +0000] "DELETE /api/v1/repository/quayadmin/busybox/tag/test
HTTP/1.0" 204 0 "http://quay-server.example.com/repository/quayadmin/busybox?tab=tags"
"Mozilla/5.0 (X11; Linux x86_64; rv:102.0) Gecko/20100101 Firefox/102.0"
```

20.6. RED HAT QUAY GARBAGE COLLECTION METRICS

The following metrics show how many resources have been removed by garbage collection. These metrics show how many times the garbage collection workers have run and how many namespaces, repositories, and blobs were removed.

Metric name	Description
quay_gc_iterations_total	Number of iterations by the GCWorker
quay_gc_namespaces_purged_total	Number of namespaces purged by the NamespaceGCWorker
quay_gc_repos_purged_total	Number of repositories purged by the RepositoryGCWorker or NamespaceGCWorker
quay_gc_storage_blobs_deleted_total	Number of storage blobs deleted

Sample metrics output

```
# TYPE quay_gc_iterations_created gauge
quay_gc_iterations_created{host="example-registry-quay-app-6df87f7b66-9tfn6",instance="",job="quay",pid="208",process_name="secscan:application"}
1.6317823190189714e+09
...

# HELP quay_gc_iterations_total number of iterations by the GCWorker
# TYPE quay_gc_iterations_total counter
quay_gc_iterations_total{host="example-registry-quay-app-6df87f7b66-9tfn6",instance="",job="quay",pid="208",process_name="secscan:application"} 0
...

# TYPE quay_gc_namespaces_purged_created gauge
quay_gc_namespaces_purged_created{host="example-registry-quay-app-6df87f7b66-9tfn6",instance="",job="quay",pid="208",process_name="secscan:application"}
1.6317823190189433e+09
...

# HELP quay_gc_namespaces_purged_total number of namespaces purged by the
NamespaceGCWorker
# TYPE quay_gc_namespaces_purged_total counter
quay_gc_namespaces_purged_total{host="example-registry-quay-app-6df87f7b66-9tfn6",instance="",job="quay",pid="208",process_name="secscan:application"} 0
....

# TYPE quay_gc_repos_purged_created gauge
quay_gc_repos_purged_created{host="example-registry-quay-app-6df87f7b66-9tfn6",instance="",job="quay",pid="208",process_name="secscan:application"}
1.631782319018925e+09
...

# HELP quay_gc_repos_purged_total number of repositories purged by the RepositoryGCWorker or
NamespaceGCWorker
# TYPE quay_gc_repos_purged_total counter
quay_gc_repos_purged_total{host="example-registry-quay-app-6df87f7b66-9tfn6",instance="",job="quay",pid="208",process_name="secscan:application"} 0
...

# TYPE quay_gc_storage_blobs_deleted_created gauge
```

```
quay_gc_storage_blobs_deleted_created{host="example-registry-quay-app-6df87f7b66-9tfn6",instance="",job="quay",pid="208",process_name="secscan:application"}
1.6317823190189059e+09
...

# HELP quay_gc_storage_blobs_deleted_total number of storage blobs deleted
# TYPE quay_gc_storage_blobs_deleted_total counter
quay_gc_storage_blobs_deleted_total{host="example-registry-quay-app-6df87f7b66-9tfn6",instance="",job="quay",pid="208",process_name="secscan:application"} 0
...
```

CHAPTER 21. USING THE V2 UI

21.1. V2 USER INTERFACE CONFIGURATION

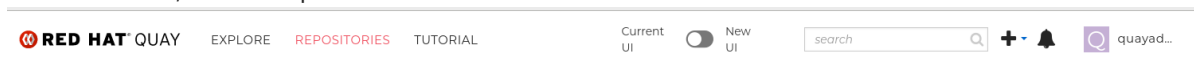


IMPORTANT

- This UI is currently in beta and subject to change. In its current state, users can only create, view, and delete organizations, repositories, and image tags.
- When using the old UI, timed-out sessions would require that the user input their password again in the pop-up window. With the new UI, users are returned to the main page and required to input their username and password credentials. This is a known issue and will be fixed in a future version of the new UI.
- There is a discrepancy in how image manifest sizes are reported between the legacy UI and the new UI. In the legacy UI, image manifests were reported in mebibytes. The v2 UI uses the standard definition of megabyte (MB) to report image manifest sizes.

Procedure

1. Log in to your deployment.
2. In the navigation pane of your deployment, you are given the option to toggle between **Current UI** and **New UI**. Click the toggle button to set it to new UI, and then click **Use Beta Environment**, for example:



21.1.1. Creating a new organization using the v2 UI

Prerequisites

- You have toggled your deployment to use the v2 UI.

Use the following procedure to create an organization using the v2 UI.

Procedure

1. Click **Organization** in the navigation pane.
2. Click **Create Organization**.
3. Enter an **Organization Name**, for example, **testorg**.
4. Click **Create**.

Now, your example organization should populate under the **Organizations** page.

21.1.2. Deleting an organization using the v2 UI

Use the following procedure to delete an organization using the v2 UI.

Procedure

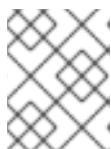
1. On the **Organizations** page, select the name of the organization you want to delete, for example, **testorg**.
2. Click the **More Actions** drop down menu.
3. Click **Delete**.



NOTE

On the **Delete** page, there is a **Search** input box. With this box, users can search for specific organizations to ensure that they are properly scheduled for deletion. For example, if a user is deleting 10 organizations and they want to ensure that a specific organization was deleted, they can use the **Search** input box to confirm said organization is marked for deletion.

4. Confirm that you want to permanently delete the organization by typing **confirm** in the box.
5. Click **Delete**.
After deletion, you are returned to the **Organizations** page.



NOTE

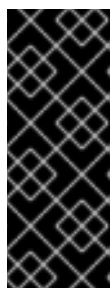
You can delete more than one organization at a time by selecting multiple organizations, and then clicking **More Actions** → **Delete**.

21.1.3. Creating a new repository using the v2 UI

Use the following procedure to create a repository using the v2 UI.

Procedure

1. Click **Repositories** on the navigation pane.
2. Click **Create Repository**.
3. Select a namespace, for example, **quayadmin**, and then enter a **Repository name**, for example, **testrepo**.



IMPORTANT

Do not use the following words in your repository name: *** build * trigger * tag**

When these words are used for repository names, users are unable access the repository, and are unable to permanently delete the repository. Attempting to delete these repositories returns the following error: **Failed to delete repository <repository_name>, HTTP404 - Not Found.**

4. Click **Create**.
Now, your example repository should populate under the **Repositories** page.

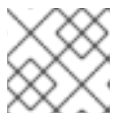
21.1.4. Deleting a repository using the v2 UI

Prerequisites

- You have created a repository.

Procedure

1. On the **Repositories** page of the v2 UI, click the name of the image you want to delete, for example, **quay/admin/busybox**.
2. Click the **More Actions** drop-down menu.
3. Click **Delete**.



NOTE

If desired, you could click **Make Public** or **Make Private**.

4. Type **confirm** in the box, and then click **Delete**.
5. After deletion, you are returned to the **Repositories** page.

21.1.5. Pushing an image to the v2 UI

Use the following procedure to push an image to the v2 UI.

Procedure

1. Pull a sample image from an external registry:

```
$ podman pull busybox
```

2. Tag the image:

```
$ podman tag docker.io/library/busybox quay-server.example.com/quayadmin/busybox:test
```

3. Push the image to your registry:

```
$ podman push quay-server.example.com/quayadmin/busybox:test
```

4. Navigate to the **Repositories** page on the v2 UI and ensure that your image has been properly pushed.
5. You can check the security details by selecting your image tag, and then navigating to the **Security Report** page.

21.1.6. Deleting an image using the v2 UI

Use the following procedure to delete an image using the v2 UI.

Prerequisites

- You have pushed an image to your registry.

Procedure

1. On the **Repositories** page of the v2 UI, click the name of the image you want to delete, for example, **quay/admin/busybox**.
2. Click the **More Actions** drop-down menu.
3. Click **Delete**.



NOTE

If desired, you could click **Make Public** or **Make Private**.

4. Type **confirm** in the box, and then click **Delete**.
5. After deletion, you are returned to the **Repositories** page.

21.1.7. Creating a new team using the Red Hat Quay v2 UI

Use the following procedure to create a new team using the Red Hat Quay v2 UI.

Prerequisites

- You have created an organization with a repository.

Procedure

1. On the Red Hat Quay v2 UI, click the name of an organization.
2. On your organization's page, click **Teams and membership**.
3. Click the **Create new team** box.
4. In the **Create team** popup window, provide a name for your new team.
5. Optional. Provide a description for your new team.
6. Click **Proceed**. A new popup window appears.
7. Optional. Add this team to a repository, and set the permissions to one of **Read**, **Write**, **Admin**, or **None**.
8. Optional. Add a team member or robot account. To add a team member, enter the name of their Red Hat Quay account.
9. Review and finish the information, then click **Review and Finish**. The new team appears under the **Teams and membership page**. From here, you can click the kebab menu, and select one of the following options:
 - **Manage Team Members**. On this page, you can view all members, team members, robot accounts, or users who have been invited. You can also add a new team member by clicking **Add new member**.
 - **Set repository permissions**. On this page, you can set the repository permissions to one of **Read**, **Write**, **Admin**, or **None**.

- **Delete.** This popup windows allows you to delete the team by clicking **Delete**.
10. Optional. You can click the one of the following options to reveal more information about teams, members, and collaborators:
 - **Team View.** This menu shows all team names, the number of members, the number of repositories, and the role for each team.
 - **Members View.** This menu shows all usernames of team members, the teams that they are part of, the repository permissions of the user.
 - **Collaborators View.** This menu shows repository collaborators. Collaborators are users that do not belong to any team in the organization, but who have direct permissions on one or more repositories belonging to the organization.

21.1.8. Creating a robot account using the v2 UI

Use the following procedure to create a robot account using the v2 UI.

Procedure

1. On the v2 UI, click **Organizations**.
2. Click the name of the organization that you will create the robot account for, for example, **test-org**.
3. Click the **Robot accounts** tab → **Create robot account**
4. In the **Provide a name for your robot account** box, enter a name, for example, **robot1**.
5. Optional. The following options are available if desired:
 - a. Add the robot to a team.
 - b. Add the robot to a repository.
 - c. Adjust the robot's permissions.
6. On the **Review and finish** page, review the information you have provided, then click **Review and finish**. The following alert appears: **Successfully created robot account with robot name: <organization_name> + <robot_name>**.
Alternatively, if you tried to create a robot account with the same name as another robot account, you might receive the following error message: **Error creating robot account**.
7. Optional. You can click **Expand** or **Collapse** to reveal descriptive information about the robot account.
8. Optional. You can change permissions of the robot account by clicking the kebab menu → **Set repository permissions**. The following message appears: **Successfully updated repository permission**.
9. Optional. To delete your robot account, check the box of the robot account and click the trash can icon. A popup box appears. Type **confirm** in the text box, then, click **Delete**. Alternatively, you can click the kebab menu → **Delete**. The following message appears: **Successfully deleted robot account**.

21.1.8.1. Bulk managing robot account repository access using the Red Hat Quay v2 UI

Use the following procedure to manage, in bulk, robot account repository access using the Red Hat Quay v2 UI.

Prerequisites

- You have created a robot account.
- You have created multiple repositories under a single organization.

Procedure

1. On the Red Hat Quay v2 UI landing page, click **Organizations** in the navigation pane.
2. On the **Organizations** page, select the name of the organization that has multiple repositories. The number of repositories under a single organization can be found under the **Repo Count** column.
3. On your organization's page, click **Robot accounts**.
4. For the robot account that will be added to multiple repositories, click the kebab icon → **Set repository permissions**.
5. On the **Set repository permissions** page, check the boxes of the repositories that the robot account will be added to. For example:

Set repository permissions ✕

Provide a name for your robot account: *

Description

Add to repository (optional)

2 selected ▾

All
Selected
⋮

1 - 3 of 3 ▾
⏪ <
1 of 1
> ⏩

Repository	Permissions	Last Updated
<input checked="" type="checkbox"/> test_repository	Read ▾	Never
<input type="checkbox"/> test_repository_2	None ▾	Never
<input checked="" type="checkbox"/> test_repository_3	Read ▾	Never

1 - 3 of 3 ▾
⏪ <
1 of 1
> ⏩

Save
Cancel

6. Set the permissions for the robot account, for example, **None**, **Read**, **Write**, **Admin**.

7. Click **save**. An alert that says **Success alert: Successfully updated repository permission** appears on the **Set repository permissions** page, confirming the changes.
8. Return to the **Organizations → Robot accounts** page. Now, the **Repositories** column of your robot account shows the number of repositories that the robot account has been added to.

21.1.9. Creating default permissions using the Red Hat Quay v2 UI

Default permissions defines permissions that should be granted automatically to a repository when it is created, in addition to the default of the repository's creator. Permissions are assigned based on the user who created the repository.

Use the following procedure to create default permissions using the Red Hat Quay v2 UI.

Procedure

1. Click the name of an organization.
2. Click **Default permissions**.
3. Click **create default permissions** A toggle drawer appears.
4. Select either **Anyone** or **Specific user** to create a default permission when a repository is created.
 - a. If selecting **Anyone**, the following information must be provided:
 - **Applied to**. Search, invite, or add a user/robot/team.
 - **Permission**. Set the permission to one of **Read**, **Write**, or **Admin**.
 - b. If selecting **Specific user**, the following information must be provided:
 - **Repository creator**. Provide either a user or robot account.
 - **Applied to**. Provide a username, robot account, or team name.
 - **Permission**. Set the permission to one of **Read**, **Write**, or **Admin**.
5. Click **Create default permission** A confirmation box appears, returning the following alert: **Successfully created default permission for creator**

21.1.10. Organization settings for the v2 UI

Use the following procedure to alter your organization settings using the v2 UI.

Procedure

1. On the v2 UI, click **Organizations**.
2. Click the name of the organization that you will create the robot account for, for example, **test-org**.
3. Click the **Settings** tab.
4. Optional. Enter the email address associated with the organization.

5. Optional. Set the allotted time for the **Time Machine** feature to one of the following:
 - 1 week
 - 1 month
 - 1 year
 - Never
6. Click **Save**.

21.1.11. Viewing image tag information using the v2 UI

Use the following procedure to view image tag information using the v2 UI.

Procedure

1. On the v2 UI, click **Repositories**.
2. Click the name of a repository, for example, **quayadmin/busybox**.
3. Click the name of the tag, for example, **test**. You are taken to the **Details** page of the tag. The page reveals the following information:
 - Name
 - Repository
 - Digest
 - Vulnerabilities
 - Creation
 - Modified
 - Size
 - Labels
 - How to fetch the image tag
4. Optional. Click **Security Report** to view the tag's vulnerabilities. You can expand an advisory column to open up CVE data.
5. Optional. Click **Packages** to view the tag's packages.
6. Click the name of the repository, for example, **busybox**, to return to the **Tags** page.
7. Optional. Hover over the **Pull** icon to reveal the ways to fetch the tag.
8. Check the box of the tag, or multiple tags, click the **Actions** drop down menu, and then **Delete** to delete the tag. Confirm deletion by clicking **Delete** in the popup box.

21.1.12. Adjusting repository settings using the v2 UI

Use the following procedure to adjust various settings for a repository using the v2 UI.

Procedure

1. On the v2 UI, click **Repositories**.
2. Click the name of a repository, for example, **quayadmin/busybox**.
3. Click the **Settings** tab.
4. Optional. Click **User and robot permissions**. You can adjust the settings for a user or robot account by clicking the dropdown menu option under **Permissions**. You can change the settings to **Read**, **Write**, or **Admin**.
5. Optional. Click **Events and notifications**. You can create an event and notification by clicking **Create Notification**. The following event options are available:
 - Push to Repository
 - Package Vulnerability Found
 - Image build failed
 - Image build queued
 - Image build started
 - Image build success
 - Image build cancelledThen, issue a notification. The following options are available:
 - Email Notification
 - Flowdock Team Notification
 - HipChat Room Notification
 - Slack Notification
 - Webhook POSTAfter selecting an event option and the method of notification, include a **Room ID #**, a **Room Notification Token**, then, click **Submit**.
6. Optional. Click **Repository visibility**. You can make the repository private, or public, by clicking **Make Public**.
7. Optional. Click **Delete repository**. You can delete the repository by clicking **Delete Repository**.

21.2. VIEWING RED HAT QUAY TAG HISTORY

Use the following procedure to view tag history on the Red Hat Quay v2 UI.

Procedure

1. On the Red Hat Quay v2 UI dashboard, click **Repositories** in the navigation pane.

2. Click the name of a repository that has image tags.
3. Click **Tag History**. On this page, you can perform the following actions:
 - Search by tag name
 - Select a date range
 - View tag changes
 - View tag modification dates and the time at which they were changed

21.3. ADDING AND MANAGING LABELS ON THE RED HAT QUAY V2 UI

Red Hat Quay administrators can add and manage labels for tags by using the following procedure.

Procedure

1. On the Red Hat Quay v2 UI dashboard, click **Repositories** in the navigation pane.
2. Click the name of a repository that has image tags.
3. Click the menu kebab for an image and select **Edit labels**.
4. In the **Edit labels** window, click **Add new label**.
5. Enter a label for the image tag using the **key=value** format, for example, **com.example.release-date=2023-11-14**.



NOTE

The following error is returned when failing to use the **key=value** format: **Invalid label format, must be key value separated by =.**

6. Click the whitespace of the box to add the label.
7. Optional. Add a second label.
8. Click **Save labels** to save the label to the image tag. The following notification is returned: **Created labels successfully**.
9. Optional. Click the same image tag's menu kebab → **Edit labels** → **X** on the label to remove it; alternatively, you can edit the text. Click **Save labels**. The label is now removed or edited.

21.4. SETTING TAG EXPIRATIONS ON THE RED HAT QUAY V2 UI

Red Hat Quay administrators can set expiration dates for certain tags in a repository. This helps automate the cleanup of older or unused tags, helping to reduce storage space.

Procedure

1. On the Red Hat Quay v2 UI dashboard, click **Repositories** in the navigation pane.
2. Click the name of a repository that has image tags.

3. Click the menu kebab for an image and select **Change expiration**.
4. Optional. Alternatively, you can bulk add expiration dates by clicking the box of multiple tags, and then select **Actions** → **Set expiration**.
5. In the **Change Tags Expiration** window, set an expiration date, specifying the day of the week, month, day of the month, and year. For example, **Wednesday, November 15, 2023**. Alternatively, you can click the calendar button and manually select the date.
6. Set the time, for example, **2:30 PM**.
7. Click **Change Expiration** to confirm the date and time. The following notification is returned: **Successfully set expiration for tag test to Nov 15, 2023, 2:26 PM**.
8. On the Red Hat Quay v2 UI **Tags** page, you can see when the tag is set to expire. For example:

repository > test > robotest

robotest

Tags Tag History

1 - 4 of 4

Tag	Security	Size	Last Modified	Expires	Manifest	Pull
<input type="checkbox"/> test3	Unable to get security details	2.27 MB	Nov 13, 2023, 2:03 PM	▲ 9 hours	sha256:72d85b66ec75	
<input type="checkbox"/> test2	Unable to get security details	2.27 MB	Nov 13, 2023, 2:01 PM	▲ 9 hours	sha256:72d85b66ec75	
<input type="checkbox"/> test1	Unable to get security details	2.27 MB	Nov 13, 2023, 1:52 PM	▲ 9 hours	sha256:72d85b66ec75	
<input type="checkbox"/> test	Unable to get security details	2.27 MB	Nov 13, 2023, 1:44 PM	▲ 3 days	sha256:72d85b66ec75	

1 - 4 of 4

21.5. SELECTING COLOR THEME PREFERENCE ON THE RED HAT QUAY V2 UI

Users can switch between light and dark modes when using the v2 UI. This feature also includes an automatic mode selection, which chooses between light or dark modes depending on the user's browser preference.

Use the following procedure to switch between automatic, light, and dark modes.

Procedure

1. Log in to your Red Hat Quay repository.
2. In the navigation pane, click your username, for example, **quayadmin**.
3. Under **Appearance**, select between **Light theme**, **Dark theme**, and **Device-based theme**. Device based theme sets the mode depending on your browser's color preference.

21.6. VIEWING USAGE LOGS ON THE RED HAT QUAY V2 UI

Red Hat Quay logs can provide valuable information about the way that your Red Hat Quay registry is being used. Logs can be viewed by Organization, repository, or namespace on the v2 UI by using the following procedure.

Procedure

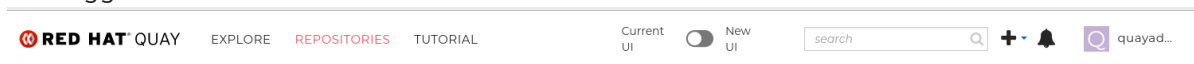
1. Log in to your Red Hat Quay registry.
2. Navigate to an Organization, repository, or namespace for which you are an administrator of.
3. Click **Logs**.



4. Optional. Set the date range for viewing log entries by adding dates to the **From** and **To** boxes.
5. Optional. Export the logs by clicking **Export**. You must enter an email address or a valid callback URL that starts with **http://** or **https://**. This process can take an hour depending on how many logs there are.

21.7. ENABLING THE LEGACY UI

1. In the navigation pane, you are given the option to toggle between **Current UI** and **New UI**. Click the toggle button to set it to **Current UI**.

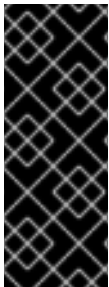


CHAPTER 22. PERFORMING HEALTH CHECKS ON RED HAT QUAY DEPLOYMENTS

Health check mechanisms are designed to assess the health and functionality of a system, service, or component. Health checks help ensure that everything is working correctly, and can be used to identify potential issues before they become critical problems. By monitoring the health of a system, Red Hat Quay administrators can address abnormalities or potential failures for things like geo-replication deployments, Operator deployments, standalone Red Hat Quay deployments, object storage issues, and so on. Performing health checks can also help reduce the likelihood of encountering troubleshooting scenarios.

Health check mechanisms can play a role in diagnosing issues by providing valuable information about the system's current state. By comparing health check results with expected benchmarks or predefined thresholds, deviations or anomalies can be identified quicker.

22.1. RED HAT QUAY HEALTH CHECK ENDPOINTS



IMPORTANT

Links contained herein to any external website(s) are provided for convenience only. Red Hat has not reviewed the links and is not responsible for the content or its availability. The inclusion of any link to an external website does not imply endorsement by Red Hat of the website or its entities, products, or services. You agree that Red Hat is not responsible or liable for any loss or expenses that may result due to your use of (or reliance on) the external site or content.

Red Hat Quay has several health check endpoints. The following table shows you the health check, a description, an endpoint, and an example output.

Table 22.1. Health check endpoints

Health check	Description	Endpoint	Example output
instance	The instance endpoint acquires the entire status of the specific Red Hat Quay instance. Returns a dict with key-value pairs for the following: auth , database , disk_space , registry_gunicorn , service_key , and web_gunicorn . Returns a number indicating the health check response of either 200 , which indicates that the instance is healthy, or 503 , which indicates an issue with your deployment.	https://{quay-ip-endpoint}/health/instance or https://{quay-ip-endpoint}/health	<pre>{"data":{"services":{"auth":true,"database":true,"disk_space":true,"registry_gunicorn":true,"service_key":true,"web_gunicorn":true}}, "status_code":200}</pre>

Health check	Description	Endpoint	Example output
endtoend	The endtoend endpoint conducts checks on all services of your Red Hat Quay instance. Returns a dict with key-value pairs for the following: auth , database , redis , storage . Returns a number indicating the health check response of either 200 , which indicates that the instance is healthy, or 503 , which indicates an issue with your deployment.	https://{quay-ip-endpoint}/health/endtoend	<pre>{"data":{"services":{"auth":true,"database":true,"redis":true,"storage":true}},"status_code":200}</pre>
warning	The warning endpoint conducts a check on the warnings. Returns a dict with key-value pairs for the following: disk_space_warning . Returns a number indicating the health check response of either 200 , which indicates that the instance is healthy, or 503 , which indicates an issue with your deployment.	https://{quay-ip-endpoint}/health/warning	<pre>{"data":{"services":{"disk_space_warning":true}},"status_code":503}</pre>

22.2. NAVIGATING TO A RED HAT QUAY HEALTH CHECK ENDPOINT

Use the following procedure to navigate to the **instance** endpoint. This procedure can be repeated for **endtoend** and **warning** endpoints.

Procedure

1. On your web browser, navigate to <https://{quay-ip-endpoint}/health/instance>.
2. You are taken to the health instance page, which returns information like the following:

```
{
  "data": {
    "services": {
      "auth": true,
      "database": true,
      "disk_space": true,
      "registry_gunicorn": true,
      "service_key": true,
      "web_gunicorn": true
    }
  },
  "status_code": 200
}
```

For Red Hat Quay, "**status_code**": **200** means that the instance is health. Conversely, if you receive "**status_code**": **503**, there is an issue with your deployment.

CHAPTER 23. BRANDING A RED HAT QUAY DEPLOYMENT ON THE LEGACY UI

You can brand the UI of your Red Hat Quay deployment by changing the registry title, logo, footer image, and by directing users to a website embedded in the footer image.

Procedure

1. Update your Red Hat Quay **config.yaml** file to add the following parameters:

```
BRANDING:  
  logo: 1  
  footer_img: 2  
  footer_url: 3  
---  
REGISTRY_TITLE: 4  
REGISTRY_TITLE_SHORT: 5
```

- 1 The URL of the image that will appear at the top of your Red Hat Quay deployment.
 - 2 The URL of the image that will appear at the bottom of your Red Hat Quay deployment.
 - 3 The URL of the website that users will be directed to when clicking the footer image.
 - 4 The long-form title for the registry. This is displayed in frontend of your Red Hat Quay deployment, for example, at the sign in page of your organization.
 - 5 The short-form title for the registry. The title is displayed on various pages of your organization, for example, as the title of the tutorial on your organization's **Tutorial** page.
2. Restart your Red Hat Quay deployment. After restarting, your Red Hat Quay deployment is updated with a new logo, footer image, and footer image URL.

CHAPTER 24. SCHEMA FOR RED HAT QUAY CONFIGURATION

Most Red Hat Quay configuration information is stored in the **config.yaml** file. All configuration options are described in the [Red Hat Quay Configuration Guide](#).