Red Hat Quay 3

Use Red Hat Quay

Use Red Hat Quay

Last Updated: 2024-07-23

Red Hat Quay 3 Use Red Hat Quay

Use Red Hat Quay

Legal Notice

Abstract

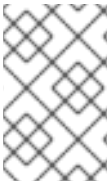
Learn to use Red Hat Quay

Table of Contents

PREFACE

Red Hat Quay container image registries serve as centralized hubs for storing container images. Users of Red Hat Quay can create repositories to effectively manage images and grant specific read (pull) and write (push) permissions to the repositories as deemed necessary. Administrative privileges expand these capabilities, allowing users to perform a broader set of tasks, like the ability to add users and control default settings.

This guide offers an overview of Red Hat Quay's users and organizations, its tenancy model, and basic operations like creating and deleting users, organizations, and repositories, handling access, and interacting with tags. It includes both UI and API operations.



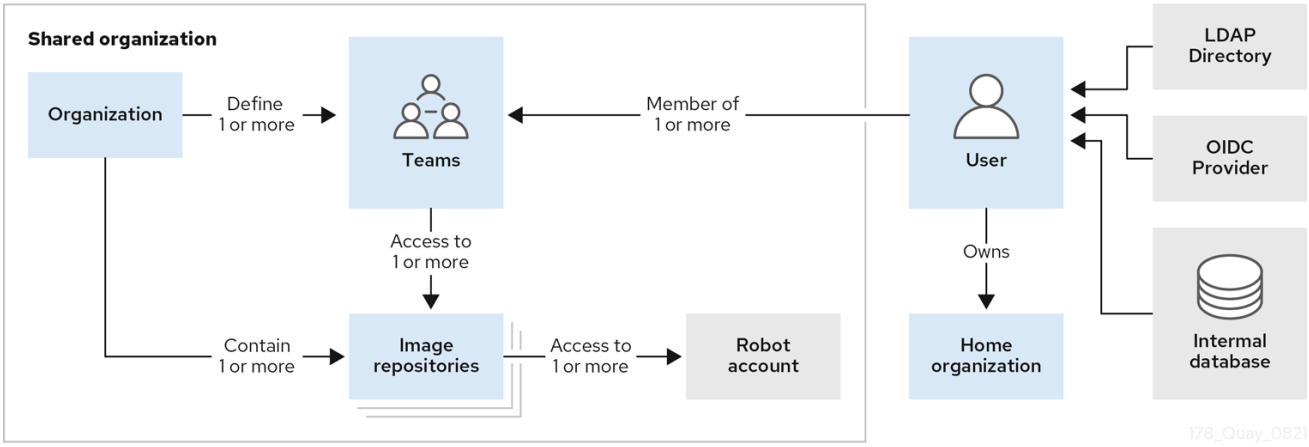
NOTE

The following API endpoints are linked to their associated entry in the Red Hat Quay API guide. The Red Hat Quay API guide provides more information about each endpoint, such as response codes and optional query parameters.

CHAPTER 1. RED HAT QUAY TENANCY MODEL

Before creating repositories to contain your container images in Red Hat Quay, you should consider how these repositories will be structured. With Red Hat Quay, each repository requires a connection with either an *Organization* or a *User*. This affiliation defines ownership and access control for the repositories.

1.1. TENANCY MODEL



- **Organizations** provide a way of sharing repositories under a common namespace that does not belong to a single user. Instead, these repositories belong to several users in a shared setting, such as a company.

- **Teams** provide a way for an Organization to delegate permissions. Permissions can be set at the global level (for example, across all repositories), or on specific repositories. They can also be set for specific sets, or groups, of users.

- **Users** can log in to a registry through the web UI or a by using a client like Podman and using their respective login commands, for example, **$ podman login**. Each user automatically gets a user namespace, for example, **<quay-server.example.com>/<user>/<username>**, or **quay.io/<username>** if you are using Quay.io.

- **Superusers** have enhanced access and privileges through the **Super User Admin Panel** in the user interface. Superuser API calls are also available, which are not visible or accessible to normal users.

- **Robot accounts** provide automated access to repositories for non-human users like pipeline tools. Robot accounts are similar to OpenShift Container Platform **Service Accounts**. Permissions can be granted to a robot account in a repository by adding that account like you would another user or team.

CHAPTER 2. RED HAT QUAY USER ACCOUNTS OVERVIEW

A *user account* represents an individual with authenticated access to the platform's features and functionalities. User accounts provide the capability to create and manage repositories, upload and retrieve container images, and control access permissions for these resources. This account is pivotal for organizing and overseeing container image management within Red Hat Quay.

You can create and delete new users on the zRed Hat Quay UI or by using the Red Hat Quay API.

2.1. CREATING A USER ACCOUNT BY USING THE UI

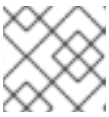
Use the following procedure to create a new user for your Red Hat Quay repository using the UI.

Prerequisites

- You are logged into your Red Hat Quay deployment as a superuser.

Procedure

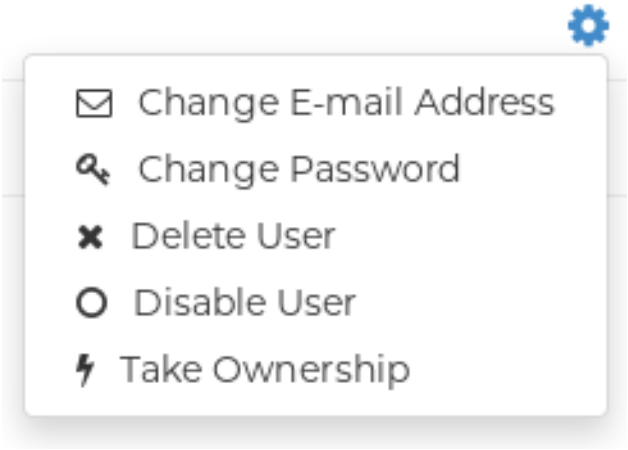
1. Log in to your Red Hat Quay repository as the superuser.
2. In the navigation pane, select your account name, and then click **Super User Admin Panel**.
3. Click the **Users** icon in the column.
4. Click the **Create User** button.
5. Enter the new user's Username and Email address, and then click the **Create User** button.
6. You are redirected to the **Users** page, where there is now another Red Hat Quay user.



NOTE

You might need to refresh the **Users** page to show the additional user.

7. On the **Users** page, click the **Options** cogwheel associated with the new user. A drop-down menu appears, as shown in the following figure:



8. Click **Change Password**.
9. Add the new password, and then click **Change User Password**.

The new user can now use that username and password to log in using the web UI or through their preferred container client, like Podman.

2.2. CREATING A USER ACCOUNT BY USING THE RED HAT QUAY API

Use the following procedure to create a new user for your Red Hat Quay repository by using the API.

Prerequisites

- You are logged into your Red Hat Quay deployment as a superuser.
- You have Created an OAuth access token.
- You have set **BROWSER_API_CALLS_XHR_ONLY: false** in your **config.yaml** file.

Procedure

1. Enter the following command to create a new user using the **POST /api/v1/superuser/users/** endpoint:

```
$ curl -X POST -H "Authorization: Bearer <bearer_token>" -H "Content-Type:
application/json" -d '{
  "username": "newuser",
  "email": "newuser@example.com"
}' "https://<quay-server.example.com>/api/v1/superuser/users/"
```

Example output

```
{"username": "newuser", "email": "newuser@example.com", "password":
"IJWZ8TIY301KPFOW3WEUJEVZ3JR11CY1", "encrypted_password":
"9Q36xF54YEOLjetayC0NBaIKgcFFmIHsS3xTZDLzZSrhTBkxUc9FDwUKfnxLWhco6oBJV1N
DBjoBcDGmsZMYPt1dSA4yWpPe/JKY9pnDcsw="}
```

2. Navigate to your Red Hat Quay registry endpoint, for example, **quay-server.example.com** and login with the username and password generated from the API call. In this scenario, the username is **newuser** and the password is **IJWZ8TIY301KPFOW3WEUJEVZ3JR11CY1**. Alternatively, you can log in to the registry with the CLI. For example:

```
$ podman login <quay-server.example.com>
```

Example output

```
username: newuser
password: IJWZ8TIY301KPFOW3WEUJEVZ3JR11CY1
```

3. Optional. You can obtain a list of all users, including superusers, by using the **GET /api/v1/superuser/users/** endpoint:

```
$ curl -X GET -H "Authorization: Bearer <bearer_token>" "https://<quay-
server.example.com>/api/v1/superuser/users/"
```

Example output

{"users": [{"kind": "user", "name": "quayadmin", "username": "quayadmin", "email": "quay@quay.com", "verified": true, "avatar": {"name": "quayadmin", "hash": "b28d563a6dc76b4431fc7b0524bbff6b810387dac86d9303874871839859c7cc", "color": "#17becf", "kind": "user"}, "super_user": true, "enabled": true}, {"kind": "user", "name": "newuser", "username": "newuser", "email": "newuser@example.com", "verified": true, "avatar": {"name": "newuser", "hash": "f338a2c83bfdde84abe2d3348994d70c34185a234cfbf32f9e323e3578e7e771", "color": "#9edae5", "kind": "user"}, "super_user": false, "enabled": true}]}

2.3. DELETING A USER BY USING THE UI

Use the following procedure to delete a user from your Red Hat Quay repository using the UI. Note that after deleting the user, any repositories that the user had in their private account become unavailable.

NOTE

In some cases, when accessing the **Users** tab in the **Superuser Admin Panel** of the Red Hat Quay UI, you might encounter a situation where no users are listed. Instead, a message appears, indicating that Red Hat Quay is configured to use external authentication, and users can only be created in that system.

This error occurs for one of two reasons:

- The web UI times out when loading users. When this happens, users are not accessible to perform any operations on.
- On LDAP authentication. When a userID is changed but the associated email is not. Currently, Red Hat Quay does not allow the creation of a new user with an old email address.

When this happens, you must delete the user using the Red Hat Quay API.

Prerequisites

- You are logged into your Red Hat Quay deployment as a superuser.

Procedure

1. Log in to your Red Hat Quay repository as the superuser.
2. In the navigation pane, select your account name, and then click **Super User Admin Panel**.
3. Click the **Users** icon in the navigation pane.
4. Click the **Options** cogwheel beside the user to be deleted.
5. Click **Delete User**, and then confirm deletion by clicking **Delete User**.

2.4. DELETING A USER BY USING THE RED HAT QUAY API

Use the following procedure to delete a user from Red Hat Quay using the API.

IMPORTANT

After deleting the user, any repositories that this user had in his private account become unavailable.

Prerequisites

- You are logged into your Red Hat Quay deployment as a superuser.
- You have Created an OAuth access token.
- You have set **BROWSER_API_CALLS_XHR_ONLY: false** in your **config.yaml** file.

Procedure

1. Enter the following **DELETE /api/v1/superuser/users/{username}** command to delete a user from the command line:

```
$ curl -X DELETE -H "Authorization: Bearer <insert token here>" https://<quay-server.example.com>/api/v1/superuser/users/<username>
```

2. The CLI does not return information when deleting a user from the CLI. To confirm deletion, you can check the Red Hat Quay UI by navigating to **Superuser Admin Panel → Users**, or by entering the following **GET /api/v1/superuser/users/** command. You can then check to see if they are present.

```
$ curl -X GET -H "Authorization: Bearer <bearer_token>" "https://<quay-server.example.com>/api/v1/superuser/users/"
```

CHAPTER 3. RED HAT QUAY ORGANIZATIONS OVERVIEW

In = Red Hat Quay an organization is a grouping of users, repositories, and teams. It provides a means to organize and manage access control and permissions within the registry. With organizations, administrators can assign roles and permissions to users and teams. Other useful information about organizations includes the following:

- You cannot have an organization embedded within another organization. To subdivide an organization, you use teams.
- Organizations cannot contain users directly. You must first add a team, and then add one or more users to each team.

NOTE

Individual users can be added to specific repositories inside of an organization. Consequently, those users are not members of any team on the **Repository Settings** page. The **Collaborators View** on the **Teams and Memberships** page shows users who have direct access to specific repositories within the organization without needing to be part of that organization specifically.

- Teams can be set up in organizations as just members who use the repositories and associated images, or as administrators with special privileges for managing the Organization.

Users can create their own organization to share repositories of container images. This can be done through the Red Hat Quay UI, or by the Red Hat Quay API if you have an OAuth token.

3.1. CREATING AN ORGANIZATION BY USING THE UI

Use the following procedure to create a new organization by using the UI.

Procedure

1. Log in to your Red Hat Quay registry.
2. Click **Organization** in the navigation pane.
3. Click **Create Organization**.
4. Enter an **Organization Name**, for example, **testorg**.
5. Enter an **Organization Email**.
6. Click **Create**.

Now, your example organization should populate under the **Organizations** page.

3.2. CREATING AN ORGANIZATION BY USING THE RED HAT QUAY API

Use the following procedure to create a new organization using the Red Hat Quay API.

Prerequisites

- You have Created an OAuth access token.

- You have set **BROWSER_API_CALLS_XHR_ONLY: false** in your **config.yaml** file.

Procedure

1. Enter the following command to create a new organization using the **POST /api/v1/organization/** endpoint:

```
$ curl -X POST    -H "Authorization: Bearer <bearer_token>" -H "Content-Type:
application/json"    -d '{
    "name": "<new_organization_name>"
  }'   "https://<quay-server.example.com>/api/v1/organization/"
```

Example output

```
"Created"
```

3.3. ORGANIZATION SETTINGS

With = Red Hat Quay, some basic organization settings can be adjusted by using the UI. This includes adjusting general settings, such as the e-mail address associated with the organization, and *time machine* settings, which allows administrators to adjust when a tag is garbage collected after it is permanently deleted.

Use the following procedure to alter your organization settings by 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:
   - **A few seconds**
   - **A day**
   - **7 days**
   - **14 days**
   - **A month**
6. Click **Save**.

3.4. DELETING AN ORGANIZATION BY USING THE 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**.

3.5. DELETING AN ORGANIZATION BY USING THE RED HAT QUAY API

Use the following procedure to delete an organization using the Red Hat Quay API.

Prerequisites

- You have Created an OAuth access token.
- You have set **BROWSER_API_CALLS_XHR_ONLY: false** in your **config.yaml** file.

Procedure

1. Enter the following command to delete an organization using the **DELETE /api/v1/organization/{orgname}** endpoint:

```
$ curl -X DELETE \
  -H "Authorization: Bearer lfV4lVf9qRsyoFnrgEno1umlOrsdp8lPyMnfUDYY" \
  "https://<quay-server.example.com>/api/v1/organization/<organization_name>"
```

2. The CLI does not return information when deleting an organization from the CLI. To confirm deletion, you can check the Red Hat Quay UI, or you can enter the **GET /api/v1/organization/{orgname}** command to see if details are returned for the deleted organization:

```
$ curl -X GET \
  -H "Authorization: Bearer lfV4lVf9qRsyoFnrgEno1umlOrsdp8lPyMnfUDYY" \
  "<quay-server.example.com>/api/v1/organization/<organization_name>"
```

Example output

```
{"detail": "Not Found", "error_message": "Not Found", "error_type": "not_found", "title":
"not_found", "type": "http://<quay-server.example.com>/api/v1/error/not_found", "status":
404}
```

CHAPTER 4. RED HAT QUAY REPOSITORY OVERVIEW

A repository provides a central location for storing a related set of container images. These images can be used to build applications along with their dependencies in a standardized format.

Repositories are organized by namespaces. Each namespace can have multiple repositories. For example, you might have a namespace for your personal projects, one for your company, or one for a specific team within your organization.

Red Hat Quay provides users with access controls for their repositories. Users can make a repository public, meaning that anyone can pull, or download, the images from it, or users can make it private, restricting access to authorized users or teams.

There are three ways to create a repository in Red Hat Quay: by pushing an image with the relevant **podman** command, by using the Red Hat Quay UI, or by using the Red Hat Quay API. Similarly, repositories can be deleted by using the UI or the proper API endpoint.

4.1. CREATING A REPOSITORY BY USING THE UI

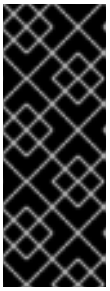
Use the following procedure to create a repository using the Red Hat Quay UI.

Procedure

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.
5. Optional. Click **Settings → Repository visibility → Make private** to set the repository to private.

4.2. CREATING A REPOSITORY BY USING PODMAN

With the proper credentials, you can *push* an image to a repository using Podman that does not yet exist in your Red Hat Quay instance. Pushing an image refers to the process of uploading a container image from your local system or development environment to a container registry like Red Hat Quay. After

pushing an image to your registry, a repository is created.

Use the following procedure to create an image repository by pushing an image.

Prerequisites

- You have download and installed the **podman** CLI.
- You have logged into your registry.
- You have pulled an image, for example, busybox.

Procedure

1. Pull a sample page from an example registry. For example:

```
$ sudo podman pull busybox
```

Example output

```
Trying to pull docker.io/library/busybox...
Getting image source signatures
Copying blob 4c892f00285e done
Copying config 22667f5368 done
Writing manifest to image destination
Storing signatures
22667f53682a2920948d19c7133ab1c9c3f745805c14125859d20cede07f11f9
```

2. Tag the image on your local system with the new repository and image name. For example:

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

3. Push the image to the registry. Following this step, you can use your browser to see the tagged image in your repository.

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

Example output

```
Getting image source signatures
Copying blob 6b245f040973 done
Copying config 22667f5368 done
Writing manifest to image destination
Storing signatures
```

4.3. CREATING A REPOSITORY BY USING THE API

Use the following procedure to create an image repository using the Red Hat Quay API.

Prerequisites

- You have Created an OAuth access token.
- You have set **BROWSER_API_CALLS_XHR_ONLY: false** in your **config.yaml** file.

Procedure

1. Enter the following command to create a repository using the **POST /api/v1/repository** endpoint:

```
$ curl -X POST \
  -H "Authorization: Bearer <bearer_token>" \
  -H "Content-Type: application/json" \
  -d '{
    "repository": "<new_repository_name>",
    "visibility": "<public>",
    "description": "<This is a description of the new repository>."
  }' \
  "https://quay-server.example.com/api/v1/repository"
```

Example output

```
{"namespace": "quayadmin", "name": "<new_repository_name>", "kind": "image"}
```

4.4. DELETING A REPOSITORY BY USING THE UI

You can delete a repository directly on the UI.

Prerequisites

- You have created a repository.

Procedure

1. On the **Repositories** page of the v2 UI, check the box of the repository that you want to delete, for example, **quayadmin/busybox**.
2. Click the **Actions** drop-down menu.
3. Click **Delete**.
4. Type **confirm** in the box, and then click **Delete**.
   After deletion, you are returned to the **Repositories** page.

4.5. DELETING A REPOSITORY BY USING THE RED HAT QUAY API

Use the following procedure to delete a repository using the Red Hat Quay API.

Prerequisites

- You have Created an OAuth access token.
- You have set **BROWSER_API_CALLS_XHR_ONLY: false** in your **config.yaml** file.

Procedure

1. Enter the following command to delete a repository using the **DELETE /api/v1/repository/{repository}** endpoint:

```
$ curl -X DELETE   -H "Authorization: Bearer <bearer_token>" "quay-server.example.com/api/v1/repository/<namespace>/<repository_name>"
```

2. The CLI does not return information when deleting a repository from the CLI. To confirm deletion, you can check the Red Hat Quay UI, or you can enter the following **GET /api/v1/repository/{repository}** command to see if details are returned for the deleted repository:

```
$ curl -X GET -H "Authorization: Bearer <bearer_token>" "quay-server.example.com/api/v1/repository/<namespace>/<repository_name>"
```

Example output

```
{"detail": "Not Found", "error_message": "Not Found", "error_type": "not_found", "title": "not_found", "type": "http://quay-server.example.com/api/v1/error/not_found", "status": 404}
```

CHAPTER 5. RED HAT QUAY ROBOT ACCOUNT OVERVIEW

Robot Accounts are used to set up automated access to the repositories in your Red Hat Quay registry. They are similar to OpenShift Container Platform service accounts.

Setting up a Robot Account results in the following:

- Credentials are generated that are associated with the Robot Account.
- Repositories and images that the Robot Account can push and pull images from are identified.
- Generated credentials can be copied and pasted to use with different container clients, such as Docker, Podman, Kubernetes, Mesos, and so on, to access each defined repository.

Each Robot Account is limited to a single user namespace or Organization. For example, the Robot Account could provide access to all repositories for the user **quayadmin**. However, it cannot provide access to repositories that are not in the user's list of repositories.

Robot Accounts can be created using the Red Hat Quay UI, or through the CLI using the Red Hat Quay API.

5.1. CREATING A ROBOT ACCOUNT BY USING THE 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**. The name of your Robot Account becomes a combination of your username plus the name of the robot, for example, **quayadmin+robot1**
5. Optional. The following options are available if desired:
   a. Add the robot account to a team.
   b. Add the robot account to a repository.
   c. Adjust the robot account'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. You can click the name of your robot account to obtain the following information:

    - **Robot Account**: Select this obtain the robot account token. You can regenerate the token by clicking **Regenerate token now**.
    - **Kubernetes Secret**: Select this to download credentials in the form of a Kubernetes pull secret YAML file.
    - **Podman**: Select this to copy a full **podman login** command line that includes the credentials.
    - **Docker Configuration**: Select this to copy a full **docker login** command line that includes the credentials.

5.2. CREATING A ROBOT ACCOUNT BY USING THE RED HAT QUAY API

Use the following procedure to create a robot account using the Red Hat Quay API.

Prerequisites

- You have Created an OAuth access token.
- You have set **BROWSER_API_CALLS_XHR_ONLY: false** in your **config.yaml** file.

Procedure

- Enter the following command to create a new robot account for an organization using the **PUT /api/v1/organization/{orgname}/robots/{robot_shortname}** endpoint:

```
$ curl -X PUT  -H "Authorization: Bearer <bearer_token>" "https://<quay-server.example.com>/api/v1/organization/<organization_name>/robots/<robot_name>"
```

Example output

```
{"name": "orgname+robot-name", "created": "Fri, 10 May 2024 15:11:00 -0000", "last_accessed": null, "description": "", "token": "WB4FUG4PP2278KK579EN4NDP150CPYOG6DN42MP6JF8IAJ4PON4RC7DIOH5UEFBP", "unstructured_metadata": null}
```

- Enter the following command to create a new robot account for the current user with the **PUT /api/v1/user/robots/{robot_shortname}** endpoint:

```
$ curl -X PUT  -H "Authorization: Bearer <bearer_token>" "https://<quay-server.example.com>/api/v1/user/robots/<robot_name>"
```

Example output

```
{"name": "quayadmin+robot-name", "created": "Fri, 10 May 2024 15:24:57 -0000", "last_accessed": null, "description": "", "token":
```

"MXFE7NSOWPN33O7UC3THY0BN03DW940CMWTLRBE2EPTI8JPX0B0CWIIDGTI4YTJ6"
, "unstructured_metadata": null}

5.3. BULK MANAGING ROBOT ACCOUNT REPOSITORY ACCESS

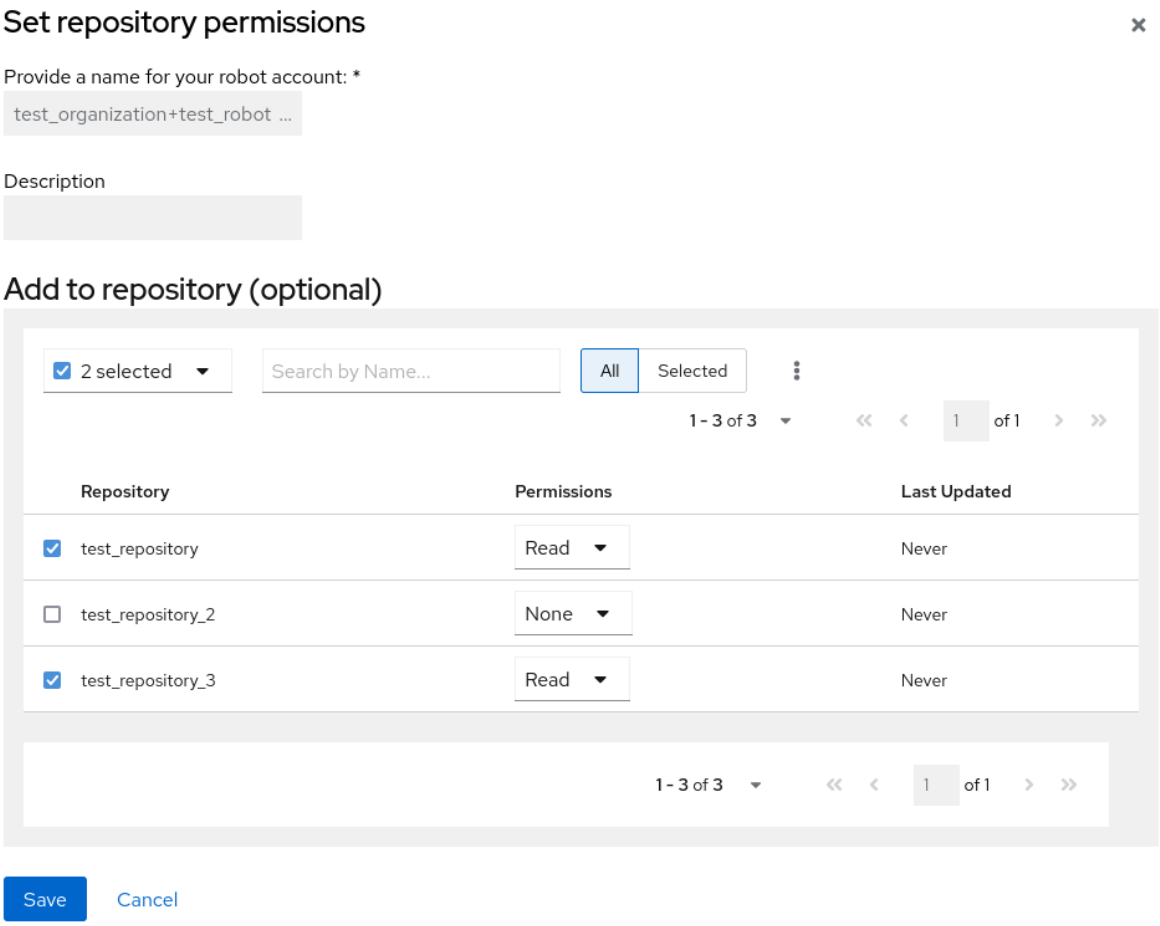
Use the following procedure to manage, in bulk, robot account repository access by using the Red Hat Quay v2 UI.

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:



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.

5.4. DISABLING ROBOT ACCOUNTS BY USING THE UI

Red Hat Quay administrators can manage robot accounts by disallowing users to create new robot accounts.

IMPORTANT

Robot accounts are mandatory for repository mirroring. Setting the **ROBOTS_DISALLOW** configuration field to **true** breaks mirroring configurations. Users mirroring repositories should not set **ROBOTS_DISALLOW** to **true** in their **config.yaml** file. This is a known issue and will be fixed in a future release of Red Hat Quay.

Use the following procedure to disable robot account creation.

Prerequisites

- You have created multiple robot accounts.

Procedure

1. Update your **config.yaml** field to add the **ROBOTS_DISALLOW** variable, for example:

```
ROBOTS_DISALLOW: true
```

2. Restart your Red Hat Quay deployment.

Verification: Creating a new robot account

1. Navigate to your Red Hat Quay repository.
2. Click the name of a repository.
3. In the navigation pane, click **Robot Accounts**.
4. Click **Create Robot Account**.
5. Enter a name for the robot account, for example, **<organization-name/username>+<robot-name>**.
6. Click **Create robot account** to confirm creation. The following message appears: **Cannot create robot account. Robot accounts have been disabled. Please contact your administrator.**

Verification: Logging into a robot account

1. On the command-line interface (CLI), attempt to log in as one of the robot accounts by entering the following command:

```
$ podman login -u="<organization-name/username>+<robot-name>" -
p="KETJ6VN0WT8YLLNXUJJ4454ZI6TZJ98NV41OE02PC2IQXVXRFQ1EJ36V12345678"
<quay-server.example.com>
```

The following error message is returned:

```
Error: logging into "<quay-server.example.com>": invalid username/password
```

2. You can pass in the **log-level=debug** flag to confirm that robot accounts have been deactivated:

```
$ podman login -u="<organization-name/username>+<robot-name>" -
p="KETJ6VN0WT8YLLNXUJJ4454ZI6TZJ98NV41OE02PC2IQXVXRFQ1EJ36V12345678" -
-log-level=debug <quay-server.example.com>
```

```
...
DEBU[0000] error logging into "quay-server.example.com": unable to retrieve auth token:
invalid username/password: unauthorized: Robot accounts have been disabled. Please
contact your administrator.
```

5.5. REGENERATING A ROBOT ACCOUNT TOKEN BY USING THE RED HAT QUAY API

Use the following procedure to regenerate a robot account token using the Red Hat Quay API.

Prerequisites

- You have Created an OAuth access token.
- You have set **BROWSER_API_CALLS_XHR_ONLY: false** in your **config.yaml** file.

Procedure

- Enter the following command to regenerate a robot account token for an organization using the **POST /api/v1/organization/{orgname}/robots/{robot_shortname}/regenerate** endpoint:

```
$ curl -X POST \
  -H "Authorization: Bearer <bearer_token>" \
  "quay-
server.example.com/api/v1/organization/<orgname>/robots/<robot_shortname>/regenerate"
```

Example output

```
{"name": "test-org+test", "created": "Fri, 10 May 2024 17:46:02 -0000", "last_accessed": null,
"description": "", "token":
"MXZ9DATUWRD8WCMT8AZIPYE0IEZHJJ1B8P8ZEIXC0W552DUMMTNJJH02HFGXTOV
G"}
```

- Enter the following command to regenerate a robot account token for the current user with the **POST /api/v1/user/robots/{robot_shortname}/regenerate** endpoint:

```
$ curl -X POST \
  -H "Authorization: Bearer <bearer_token>" \
  "quay-server.example.com/api/v1/user/robots/<robot_shortname>/regenerate"
```

Example output

```
{"name": "quayadmin+test", "created": "Fri, 10 May 2024 14:12:11 -0000", "last_accessed":
null, "description": "", "token":
"CWLBVAODE61IXNDJ40GERFOZPB3ARZDRCP4X70ID1NB28AI0OOJBTR9S4M0ACYMD
"}
```

5.6. DELETING A ROBOT ACCOUNT BY USING THE UI

Use the following procedure to delete a robot account using the Red Hat Quay UI.

Procedure

1. Log into your Red Hat Quay registry:
2. Click the name of the Organization that has the robot account.
3. Click **Robot accounts**.
4. Check the box of the robot account to be deleted.
5. Click the kebab menu.
6. Click **Delete**.

7. Type **confirm** into the textbox, then click **Delete**.

5.7. DELETING A ROBOT ACCOUNT BY USING THE RED HAT QUAY API

Use the following procedure to delete a robot account using the Red Hat Quay API.

Prerequisites

- You have Created an OAuth access token.
- You have set **BROWSER_API_CALLS_XHR_ONLY: false** in your **config.yaml** file.

Procedure

1. Enter the following command to delete a robot account for an organization using the **DELETE /api/v1/organization/{orgname}/robots/{robot_shortname}** endpoint:

```
curl -X DELETE \
  -H "Authorization: Bearer lfV4lVf9qRsyoFnrgEno1umlOrsdp8lPyMnfUDYY" \
  "quay-
server.example.com/api/v1/organization/<organization_name>/robots/<robot_shortname>"
```

2. The CLI does not return information when deleting a robot account with the API. To confirm deletion, you can check the Red Hat Quay UI, or you can enter the following **GET /api/v1/organization/{orgname}/robots** command to see if details are returned for the robot account:

```
$ curl -X GET   -H "Authorization: Bearer <bearer_token>"   "https://<quay-
server.example.com>/api/v1/organization/<organization_name>/robots"
```

Example output

```
{"robots": []}
```

- Enter the following command to delete a robot account for the current user with the **DELETE /api/v1/user/robots/{robot_shortname}** endpoint:

```
curl -X DELETE \
  -H "Authorization: Bearer lfV4lVf9qRsyoFnrgEno1umlOrsdp8lPyMnfUDYY" \
  "quay-server.example.com/api/v1/user/robots/<robot_shortname>"
```

3. The CLI does not return information when deleting a robot account for the current user with the API. To confirm deletion, you can check the Red Hat Quay UI, or you can enter the following **GET /api/v1/user/robots/{robot_shortname}** command to see if details are returned for the robot account:

```
$ curl -X GET \
  -H "Authorization: Bearer lfV4lVf9qRsyoFnrgEno1umlOrsdp8lPyMnfUDYY" \
  "quay-server.example.com/api/v1/user/robots/<robot_shortname>"
```

Example output

```
{"message":"Could not find robot with specified username"}
```

CHAPTER 6. ACCESS MANAGEMENT FOR RED HAT QUAY

As a Red Hat Quay user, you can create your own repositories and make them accessible to other users that are part of your instance. Alternatively, you can create an organization and associate a set of repositories directly to that organization, referred to as an *organization repository*.

Organization repositories differ from basic repositories in that the organization is intended to set up shared repositories through groups of users. In Red Hat Quay, groups of users can be either *Teams*, or sets of users with the same permissions, or *individual users*. You can also allow access to user repositories and organization repositories by creating credentials associated with Robot Accounts. Robot Accounts make it easy for a variety of container clients, such as Docker or Podman, to access your repositories without requiring that the client have a Red Hat Quay user account.

6.1. RED HAT QUAY TEAMS OVERVIEW

In Red Hat Quay a *team* is a group of users with shared permissions, allowing for efficient management and collaboration on projects. Teams can help streamline access control and project management within organizations and repositories. They can be assigned designated permissions and help ensure that members have the appropriate level of access to their repositories based on their roles and responsibilities.

6.1.1. Creating a team by using the UI

When you create a team for your organization you can select the team name, choose which repositories to make available to the team, and decide the level of access to the team.

Use the following procedure to create a team for your organization repository.

Prerequisites

- You have created an organization.

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 the following:
    - **None**. Team members have no permission to the repository.
    - **Read**. Team members can view and pull from the repository.
    - **Write**. Team members can read (pull) from and write (push) to the repository.

- **Admin**. Full access to pull from, and push to, the repository, plus the ability to do administrative tasks associated with the repository.

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**.

6.1.2. Creating a team by using the API

When you create a team for your organization with the API you can select the team name, choose which repositories to make available to the team, and decide the level of access to the team.

Use the following procedure to create a team for your organization repository.

Prerequisites

- You have created an organization.
- You have Created an OAuth access token.
- You have set **BROWSER_API_CALLS_XHR_ONLY: false** in your **config.yaml** file.

Procedure

1. Enter the following command to create a team for your organization:

```
$ ----------------------------
$ curl -k -X PUT -H 'Accept: application/json' -H 'Content-Type: application/json' -H
"Authorization: Bearer <bearer_token>"  --data '{"role": "creator"}' https://<quay-
server.example.com>/api/v1/organization/<organization_name>/team/<team_name>
```

Example output

```
{"name": "example_team", "description": "", "can_view": true, "role": "creator", "avatar":
{"name": "example_team", "hash":
"dec209fd7312a2284b689d4db3135e2846f27e0f40fa126776a0ce17366bc989", "color":
"#e7ba52", "kind": "team"}, "new_team": true}
```

6.1.3. Managing a team by using the UI

After you have created a team, you can use the UI to manage team members, set repository permissions, delete the team, or view more general information about the team.

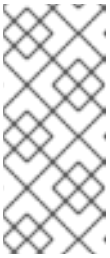
6.1.3.1. Adding users to a team by using the UI

With administrative privileges to an Organization, you can add users and robot accounts to a team. When you add a user, Red Hat Quay sends an email to that user. The user remains pending until they accept the invitation.

Use the following procedure to add users or robot accounts to a team.

Procedure

1. On the Red Hat Quay landing page, click the name of your Organization.
2. In the navigation pane, click **Teams and Membership**.
3. Select the menu kebab of the team that you want to add users or robot accounts to. Then, click **Manage team members**.
4. Click **Add new member**.
5. In the textbox, enter information for one of the following:
   - A username from an account on the registry.
   - The email address for a user account on the registry.
   - The name of a robot account. The name must be in the form of <organization_name>+<robot_name>.



NOTE

Robot Accounts are immediately added to the team. For user accounts, an invitation to join is mailed to the user. Until the user accepts that invitation, the user remains in the **INVITED TO JOIN** state. After the user accepts the email invitation to join the team, they move from the **INVITED TO JOIN** list to the **MEMBERS** list for the Organization.

6. Click **Add member**.

6.1.3.2. Setting a team role by using the UI

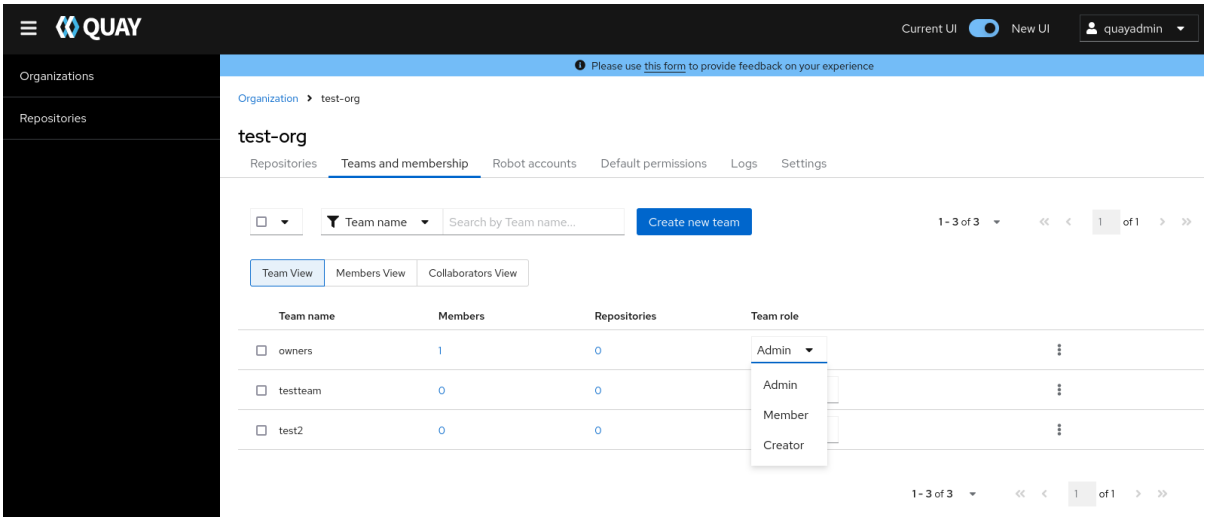
After you have created a team, you can set the role of that team within the Organization.

Prerequisites

- You have created a team.

Procedure

1. On the Red Hat Quay landing page, click the name of your Organization.
2. In the navigation pane, click **Teams and Membership**.
3. Select the **TEAM ROLE** drop-down menu, as shown in the following figure:



4. For the selected team, choose one of the following roles:

    - **Admin**. Full administrative access to the organization, including the ability to create teams, add members, and set permissions.
    - **Member**. Inherits all permissions set for the team.
    - **Creator**. All member permissions, plus the ability to create new repositories.

6.1.3.2.1. Managing team members and repository permissions

Use the following procedure to manage team members and set repository permissions.

- On the **Teams and membership** page of your organization, you can also manage team members and set repository permissions.
    - 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 the following:
        - **None**. Team members have no permission to the repository.
        - **Read**. Team members can view and pull from the repository.
        - **Write**. Team members can read (pull) from and write (push) to the repository.
        - **Admin**. Full access to pull from, and push to, the repository, plus the ability to do administrative tasks associated with the repository.
    - **Delete**. This popup windows allows you to delete the team by clicking **Delete**.

6.1.3.2.2. Viewing additional information about a team

Use the following procedure to view general information about the team.

Procedure

- On the **Teams and membership** page of your organization, 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.

6.1.4. Managing a team by using the Red Hat Quay API

After you have created a team, you can use the API to obtain information about team permissions or team members, add, update, or delete team members (including by email), or delete an organization team.

The following procedures show you how to how to manage a team using the Red Hat Quay API.

6.1.4.1. Managing team members and repository permissions by using the API

Use the following procedures to add a member to a team (by direct invite or by email), or to remove a member from a team.

Prerequisites

- You have Created an OAuth access token.
- You have set **BROWSER_API_CALLS_XHR_ONLY: false** in your **config.yaml** file.

Procedure

- Enter the **PUT /api/v1/organization/{orgname}/team/{teamname}/members/{membername}** command to add or invite a member to an existing team:

```
$ curl -X PUT \
  -H "Authorization: Bearer <your_access_token>" \
  "quay-
server.example.com/api/v1/organization/<organization_name>/team/<team_name>/members/<
member_name>"
```

Example output

```
{"name": "testuser", "kind": "user", "is_robot": false, "avatar": {"name": "testuser", "hash":
"d51d17303dc3271ac3266fb332d7df919bab882bbfc7199d2017a4daac8979f0", "color":
"#5254a3", "kind": "user"}, "invited": false}
```

- Enter the **DELETE /api/v1/organization/{orgname}/team/{teamname}/members/{membername}** command to remove a member of a team:

```
$ curl -X DELETE \
```

```
  -H "Authorization: Bearer <your_access_token>" \
  "quay-
server.example.com/api/v1/organization/<organization_name>/team/<team_name>/members/<
member_name>"
```

This command does not an output in the CLI. To ensure that a member has been deleted, you can enter the **GET /api/v1/organization/{orgname}/team/{teamname}/members** command and ensure that the member is not returned in the output.

```
$ curl -X GET \
  -H "Authorization: Bearer <your_access_token>" \
  "quay-
server.example.com/api/v1/organization/<organization_name>/team/<team_name>/members"
```

Example output

```
{"name": "owners", "members": [{"name": "quayadmin", "kind": "user", "is_robot": false,
"avatar": {"name": "quayadmin", "hash":
"b28d563a6dc76b4431fc7b0524bbff6b810387dac86d9303874871839859c7cc", "color":
"#17becf", "kind": "user"}, "invited": false}, {"name": "test-org+test", "kind": "user", "is_robot":
true, "avatar": {"name": "test-org+test", "hash":
"aa85264436fe9839e7160bf349100a9b71403a5e9ec684d5b5e9571f6c821370", "color":
"#8c564b", "kind": "robot"}, "invited": false}], "can_edit": true}
```

- You can enter the **PUT /api/v1/organization/{orgname}/team/{teamname}/invite/{email}** command to invite a user, by email address, to an existing team:

```
$ curl -X PUT \
  -H "Authorization: Bearer <your_access_token>" \
  "quay-
server.example.com/api/v1/organization/<organization_name>/team/<team_name>/invite/<ema
l>"
```

- You can enter the **DELETE /api/v1/organization/{orgname}/team/{teamname}/invite/{email}** command to delete the invite of an email address to join a team. For example:

```
$ curl -X DELETE \
  -H "Authorization: Bearer <your_access_token>" \
  "quay-
server.example.com/api/v1/organization/<organization_name>/team/<team_name>/invite/<ema
l>"
```

6.1.4.2. Setting the role of a team within an organization by using the API

Use the following procedure to view and set the role a team within an organization using the API.

Prerequisites

- You have Created an OAuth access token.
- You have set **BROWSER_API_CALLS_XHR_ONLY: false** in your **config.yaml** file.

Procedure

1. Enter the following command to return a list of repository permissions for the organization's team. Note that your team must have been added to a repository for this command to return information.

```
$ curl -X GET \
  -H "Authorization: Bearer <your_access_token>" \
  "quay-
server.example.com/api/v1/organization/<organization_name>/team/<team_name>/permissions
"
```

Example output

```
{"permissions": [{"repository": {"name": "api-repo", "is_public": true}, "role": "admin"}]}
```

2. You can create or update a team within an organization to have a specified role of **admin**, **member**, or **creator**. For example:

```
$ curl -X PUT \
  -H "Authorization: Bearer <your_access_token>" \
  -H "Content-Type: application/json" \
  -d '{
    "role": "<role>"
  }' \
  "quay-server.example.com/api/v1/organization/<organization_name>/team/<team_name>"
```

Example output

```
{"name": "testteam", "description": "", "can_view": true, "role": "creator", "avatar": {"name":
"testteam", "hash":
"827f8c5762148d7e85402495b126e0a18b9b168170416ed04b49aae551099dc8", "color":
"#ff7f0e", "kind": "team"}, "new_team": false}
```

6.1.4.3. Deleting a team within an organization by using the API

Use the following procedure to delete a team within an organization by using the API.

Prerequisites

- You have Created an OAuth access token.
- You have set **BROWSER_API_CALLS_XHR_ONLY: false** in your **config.yaml** file.

Procedure

- You can delete a team within an organization by entering the **DELETE /api/v1/organization/{orgname}/team/{teamname}** command:

```
$ curl -X DELETE \
  -H "Authorization: Bearer <your_access_token>" \
  "quay-server.example.com/api/v1/organization/<organization_name>/team/<team_name>"
```

This command does not return output in the CLI.

6.2. CREATING AND MANAGING DEFAULT PERMISSIONS BY USING THE UI

Default permissions define 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**

6.3. CREATING AND MANAGING DEFAULT PERMISSIONS BY USING THE API

Use the following procedures to manage default permissions using the Red Hat Quay API.

Prerequisites

- You have Created an OAuth access token.
- You have set **BROWSER_API_CALLS_XHR_ONLY: false** in your **config.yaml** file.

Procedure

1. Enter the following command to create a default permission with the **POST /api/v1/organization/{orgname}/prototypes** endpoint:

```
$ curl -X POST    -H "Authorization: Bearer <bearer_token>"    -H "Content-Type:
application/json"    --data '{
```

```
    "role": "<admin_read_or_write>",
    "delegate": {
      "name": "<username>",
      "kind": "user"
    },
    "activating_user": {
      "name": "<robot_name>"
    }
  }'   https://<quay-server.example.com>/api/v1/organization/<organization_name>/prototypes
```

Example output

```
{"activating_user": {"name": "test-org+test", "is_robot": true, "kind": "user", "is_org_member": true, "avatar": {"name": "test-org+test", "hash": "aa85264436fe9839e7160bf349100a9b71403a5e9ec684d5b5e9571f6c821370", "color": "#8c564b", "kind": "robot"}}, "delegate": {"name": "testuser", "is_robot": false, "kind": "user", "is_org_member": false, "avatar": {"name": "testuser", "hash": "f660ab912ec121d1b1e928a0bb4bc61b15f5ad44d5efdc4e1c92a25e99b8e44a", "color": "#6b6ecf", "kind": "user"}}, "role": "admin", "id": "977dc2bc-bc75-411d-82b3-604e5b79a493"}
```

2. Enter the following command to update a default permission using the **PUT /api/v1/organization/{orgname}/prototypes/{prototypeid}** endpoint, for example, if you want to change the permission type. You must include the ID that was returned when you created the policy.

```
$ curl -X PUT \
  -H "Authorization: Bearer <bearer_token>" \
  -H "Content-Type: application/json" \
  --data '{
    "role": "write"
  }' \
  https://<quay-server.example.com>/api/v1/organization/<organization_name>/prototypes/<prototypeid>
```

Example output

```
{"activating_user": {"name": "test-org+test", "is_robot": true, "kind": "user", "is_org_member": true, "avatar": {"name": "test-org+test", "hash": "aa85264436fe9839e7160bf349100a9b71403a5e9ec684d5b5e9571f6c821370", "color": "#8c564b", "kind": "robot"}}, "delegate": {"name": "testuser", "is_robot": false, "kind": "user", "is_org_member": false, "avatar": {"name": "testuser", "hash": "f660ab912ec121d1b1e928a0bb4bc61b15f5ad44d5efdc4e1c92a25e99b8e44a", "color": "#6b6ecf", "kind": "user"}}, "role": "write", "id": "977dc2bc-bc75-411d-82b3-604e5b79a493"}
```

3. You can delete the permission by entering the **DELETE /api/v1/organization/{orgname}/prototypes/{prototypeid}** command:

```
curl -X DELETE \
  -H "Authorization: Bearer <bearer_token>" \
  -H "Accept: application/json" \
  https://<quay-server.example.com>/api/v1/organization/<organization_name>/prototypes/<prototype_id>
```

This command does not return an output. Instead, you can obtain a list of all permissions by entering the **GET /api/v1/organization/{orgname}/prototypes** command:

```
$ curl -X GET \
  -H "Authorization: Bearer <bearer_token>" \
  -H "Accept: application/json" \
  https://<quay-server.example.com>/api/v1/organization/<organization_name>/prototypes
```

Example output

```
{"prototypes": []}
```

6.4. ADJUSTING ACCESS SETTINGS FOR A REPOSITORY BY USING THE UI

Use the following procedure to adjust access settings for a user or robot account for a repository using the v2 UI.

Prerequisites

- You have created a user account or robot account.

Procedure

1. Log into Red Hat Quay.
2. On the v2 UI, click **Repositories**.
3. Click the name of a repository, for example, **quayadmin/busybox**.
4. Click the **Settings** tab.
5. 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**.
   - **Read**. The User or Robot Account can view and pull from the repository.
   - **Write**. The User or Robot Account can read (pull) from and write (push) to the repository.
   - **Admin**. The User or Robot account has access to pull from, and push to, the repository, plus the ability to do administrative tasks associated with the repository.

6.5. ADJUSTING ACCESS SETTINGS FOR A REPOSITORY BY USING THE API

Use the following procedure to adjust access settings for a user or robot account for a repository by using the API.

Prerequisites

- You have created a user account or robot account.

- You have Created an OAuth access token.
- You have set **BROWSER_API_CALLS_XHR_ONLY: false** in your **config.yaml** file.

Procedure

1. Enter the following **PUT /api/v1/repository/{repository}/permissions/user/{username}** command to change the permissions of a user:

```
$ curl -X PUT \
  -H "Authorization: Bearer <bearer_token>" \
  -H "Content-Type: application/json" \
  -d '{"role": "admin"}' \
  https://<quay-server.example.com>/api/v1/repository/<namespace>/<repository>/permissions/user/<username>
```

Example output

```
{"role": "admin", "name": "quayadmin+test", "is_robot": true, "avatar": {"name": "quayadmin+test", "hash": "ca9afae0a9d3ca322fc8a7a866e8476dd6c98de543decd186ae090e420a88feb", "color": "#8c564b", "kind": "robot"}}
```

2. To delete the current permission, you can enter the **DELETE /api/v1/repository/{repository}/permissions/user/{username}** command:

```
$ curl -X DELETE \
  -H "Authorization: Bearer <bearer_token>" \
  -H "Accept: application/json" \
  https://<quay-server.example.com>/api/v1/repository/<namespace>/<repository>/permissions/user/<username>
```

This command does not return any output in the CLI. Instead, you can check that the permissions were deleted by entering the **GET /api/v1/repository/{repository}/permissions/user/** command:

```
$ curl -X GET \
  -H "Authorization: Bearer <bearer_token>" \
  -H "Accept: application/json" \
  https://<quay-server.example.com>/api/v1/repository/<namespace>/<repository>/permissions/user/<username>/
```

Example output

```
{"message":"User does not have permission for repo."}
```

CHAPTER 7. IMAGE TAGS OVERVIEW

An *image tag* refers to a label or identifier assigned to a specific version or variant of a container image. Container images are typically composed of multiple layers that represent different parts of the image. Image tags are used to differentiate between different versions of an image or to provide additional information about the image.

Image tags have the following benefits:

- **Versioning and Releases**: Image tags allow you to denote different versions or releases of an application or software. For example, you might have an image tagged as *v1.0* to represent the initial release and *v1.1* for an updated version. This helps in maintaining a clear record of image versions.
- **Rollbacks and Testing**: If you encounter issues with a new image version, you can easily revert to a previous version by specifying its tag. This is helpful during debugging and testing phases.
- **Development Environments**: Image tags are beneficial when working with different environments. You might use a *dev* tag for a development version, *qa* for quality assurance testing, and *prod* for production, each with their respective features and configurations.
- **Continuous Integration/Continuous Deployment (CI/CD)**: CI/CD pipelines often utilize image tags to automate the deployment process. New code changes can trigger the creation of a new image with a specific tag, enabling seamless updates.
- **Feature Branches**: When multiple developers are working on different features or bug fixes, they can create distinct image tags for their changes. This helps in isolating and testing individual features.
- **Customization**: You can use image tags to customize images with different configurations, dependencies, or optimizations, while keeping track of each variant.
- **Security and Patching**: When security vulnerabilities are discovered, you can create patched versions of images with updated tags, ensuring that your systems are using the latest secure versions.
- **Dockerfile Changes**: If you modify the Dockerfile or build process, you can use image tags to differentiate between images built from the previous and updated Dockerfiles.

Overall, image tags provide a structured way to manage and organize container images, enabling efficient development, deployment, and maintenance workflows.

7.1. VIEWING IMAGE TAG INFORMATION BY USING THE UI

Use the following procedure to view image tag information using the v2 UI.

Prerequisites

- You have pushed an image tag to a repository.

Procedure

1. On the v2 UI, click **Repositories**.
2. Click the name of a repository.

3. Click the name of a tag. You are taken to the **Details** page of that tag. The page reveals the following information:
    - Name
    - Repository
    - Digest
    - Vulnerabilities
    - Creation
    - Modified
    - Size
    - Labels
    - How to fetch the image tag
4. Click **Security Report** to view the tag's vulnerabilities. You can expand an advisory column to open up CVE data.
5. Click **Packages** to view the tag's packages.
6. Click the name of the repository to return to the **Tags** page.

7.2. VIEWING IMAGE TAG INFORMATION BY USING THE API

Use the following procedure to view image tag information by using the API

Prerequisites

- You have pushed an image tag to a Red Hat Quay repository.
- You have Created an OAuth access token.
- You have set **BROWSER_API_CALLS_XHR_ONLY: false** in your **config.yaml** file.

Procedure

1. To obtain tag information, you must use the **GET /api/v1/repository/{repository}** API endpoint and pass in the **includeTags** parameter. For example:

```
$ curl -X GET \
  -H "Authorization: Bearer <bearer_token>" \
  -H "Accept: application/json" \
  https://<quay-server.example.com>/api/v1/repository/<namespace>/<repository_name>?
includeTags=true
```

Example output

```
{"namespace": "quayadmin", "name": "busybox", "kind": "image", "description": null,
"is_public": false, "is_organization": false, "is_starred": false, "status_token": "d8f5e074-690a-
46d7-83c8-8d4e3d3d0715", "trust_enabled": false, "tag_expiration_s": 1209600,
```

"is_free_account": true, "state": "NORMAL", "tags": {"example": {"name": "example", "size": 2275314, "last_modified": "Tue, 14 May 2024 14:48:51 -0000", "manifest_digest": "sha256:57583a1b9c0a7509d3417387b4f43acf80d08cdcf5266ac87987be3f8f919d5d"}, "test": {"name": "test", "size": 2275314, "last_modified": "Tue, 14 May 2024 14:04:48 -0000", "manifest_digest": "sha256:57583a1b9c0a7509d3417387b4f43acf80d08cdcf5266ac87987be3f8f919d5d"}}, "can_write": true, "can_admin": true}

2. Alternatively, you can use the **GET /api/v1/repository/{repository}/tag/** endpoint. For example:

```
$ curl -X GET \
  -H "Authorization: Bearer <bearer_token>" \
  -H "Accept: application/json" \
  https://<quay-
server.example.com>/api/v1/repository/<namespace>/<repository_name>/tag/
```

Example output

{"tags": [{"name": "test-two", "reversion": true, "start_ts": 1718737153, "manifest_digest": "sha256:57583a1b9c0a7509d3417387b4f43acf80d08cdcf5266ac87987be3f8f919d5d", "is_manifest_list": false, "size": 2275314, "last_modified": "Tue, 18 Jun 2024 18:59:13 -0000"}, {"name": "test-two", "reversion": false, "start_ts": 1718737029, "end_ts": 1718737153, "manifest_digest": "sha256:0cd3dd6236e246b349e63f76ce5f150e7cd5dbf2f2f1f88dbd734430418dbaea", "is_manifest_list": false, "size": 2275317, "last_modified": "Tue, 18 Jun 2024 18:57:09 -0000", "expiration": "Tue, 18 Jun 2024 18:59:13 -0000"}, {"name": "test-two", "reversion": false, "start_ts": 1718737018, "end_ts": 1718737029, "manifest_digest": "sha256:0cd3dd6236e246b349e63f76ce5f150e7cd5dbf2f2f1f88dbd734430418dbaea", "is_manifest_list": false, "size": 2275317, "last_modified": "Tue, 18 Jun 2024 18:56:58 -0000", "expiration": "Tue, 18 Jun 2024 18:57:09 -0000"}, {"name": "sample_tag", "reversion": false, "start_ts": 1718736147, "manifest_digest": "sha256:57583a1b9c0a7509d3417387b4f43acf80d08cdcf5266ac87987be3f8f919d5d", "is_manifest_list": false, "size": 2275314, "last_modified": "Tue, 18 Jun 2024 18:42:27 -0000"}, {"name": "test-two", "reversion": false, "start_ts": 1717680780, "end_ts": 1718737018, "manifest_digest": "sha256:57583a1b9c0a7509d3417387b4f43acf80d08cdcf5266ac87987be3f8f919d5d", "is_manifest_list": false, "size": 2275314, "last_modified": "Thu, 06 Jun 2024 13:33:00 -0000", "expiration": "Tue, 18 Jun 2024 18:56:58 -0000"}, {"name": "tag-test", "reversion": false, "start_ts": 1717680378, "manifest_digest": "sha256:57583a1b9c0a7509d3417387b4f43acf80d08cdcf5266ac87987be3f8f919d5d", "is_manifest_list": false, "size": 2275314, "last_modified": "Thu, 06 Jun 2024 13:26:18 -0000"}, {"name": "example", "reversion": false, "start_ts": 1715698131, "manifest_digest": "sha256:57583a1b9c0a7509d3417387b4f43acf80d08cdcf5266ac87987be3f8f919d5d", "is_manifest_list": false, "size": 2275314, "last_modified": "Tue, 14 May 2024 14:48:51 -0000"}], "page": 1, "has_additional": false}

7.3. ADDING A NEW IMAGE TAG TO AN IMAGE BY USING THE UI

You can add a new tag to an image in Red Hat Quay.

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, then click **Add new tag**.
4. Enter a name for the tag, then, click **Create tag**.
   The new tag is now listed on the **Repository Tags** page.

7.4. ADDING A NEW TAG TO AN IMAGE TAG TO AN IMAGE BY USING THE API

You can add a new tag, or restore an old one, to an image by using the API.

Prerequisites

- You have Created an OAuth access token.
- You have set **BROWSER_API_CALLS_XHR_ONLY: false** in your **config.yaml** file.

Procedure

1. You can change which image a tag points to or create a new tag by using the **PUT /api/v1/repository/{repository}/tag/{tag}** command:

```
$ curl -X PUT \
  -H "Authorization: Bearer <bearer_token>" \
  -H "Content-Type: application/json" \
  --data '{
    "manifest_digest": "<manifest_digest>"
  }' \
  https://<quay-
server.example.com>/api/v1/repository/<namespace>/<repository_name>/tag/<tag>
```

Example output

```
"Updated"
```

2. You can restore a repository tag to its previous image by using the **POST /api/v1/repository/{repository}/tag/{tag}/restore** command. For example:

```
$ curl -X POST \
  -H "Authorization: Bearer <bearer_token>" \
  -H "Content-Type: application/json" \
  --data '{
    "manifest_digest": <manifest_digest>
  }' \
  quay-server.example.com/api/v1/repository/quayadmin/busybox/tag/test/restore
```

Example output

```
{}
```

3. To see a list of tags after creating a new tag you can use the **GET /api/v1/repository/{repository}/tag/** command. For example:

```
$ curl -X GET \
  -H "Authorization: Bearer <bearer_token>" \
  -H "Accept: application/json" \
   https://<quay-
server.example.com>/api/v1/repository/<namespace>/<repository_name>/tag
```

Example output


```
{"tags": [{"name": "test", "reversion": false, "start_ts": 1716324069, "manifest_digest":
"sha256:57583a1b9c0a7509d3417387b4f43acf80d08cdcf5266ac87987be3f8f919d5d",
"is_manifest_list": false, "size": 2275314, "last_modified": "Tue, 21 May 2024 20:41:09 -
0000"}, {"name": "example", "reversion": false, "start_ts": 1715698131, "manifest_digest":
"sha256:57583a1b9c0a7509d3417387b4f43acf80d08cdcf5266ac87987be3f8f919d5d",
"is_manifest_list": false, "size": 2275314, "last_modified": "Tue, 14 May 2024 14:48:51 -
0000"}, {"name": "example", "reversion": false, "start_ts": 1715697708, "end_ts":
1715698131, "manifest_digest":
"sha256:57583a1b9c0a7509d3417387b4f43acf80d08cdcf5266ac87987be3f8f919d5d",
"is_manifest_list": false, "size": 2275314, "last_modified": "Tue, 14 May 2024 14:41:48 -
0000", "expiration": "Tue, 14 May 2024 14:48:51 -0000"}, {"name": "test", "reversion": false,
"start_ts": 1715695488, "end_ts": 1716324069, "manifest_digest":
"sha256:57583a1b9c0a7509d3417387b4f43acf80d08cdcf5266ac87987be3f8f919d5d",
"is_manifest_list": false, "size": 2275314, "last_modified": "Tue, 14 May 2024 14:04:48 -
0000", "expiration": "Tue, 21 May 2024 20:41:09 -0000"}, {"name": "test", "reversion": false,
"start_ts": 1715631517, "end_ts": 1715695488, "manifest_digest":
"sha256:57583a1b9c0a7509d3417387b4f43acf80d08cdcf5266ac87987be3f8f919d5d",
"is_manifest_list": false, "size": 2275314, "last_modified": "Mon, 13 May 2024 20:18:37 -
0000", "expiration": "Tue, 14 May 2024 14:04:48 -0000"}], "page": 1, "has_additional": false}
```


7.5. ADDING AND MANAGING LABELS BY USING THE UI

Administrators can add and manage labels for tags by using the following procedure.

Procedure

1. On the 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.

7.6. ADDING AND MANAGING LABELS BY USING THE API

Red Hat Quay administrators can add and manage labels for tags with the API by using the following procedure.

Prerequisites

- You have Created an OAuth access token.
- You have set **BROWSER_API_CALLS_XHR_ONLY: false** in your **config.yaml** file.

Procedure

1. Use the **GET /api/v1/repository/{repository}/manifest/{manifestref}** command to retrieve the details of a specific manifest in a repository:

```
$ curl -X GET \
  -H "Authorization: Bearer <bearer_token>" \
  -H "Accept: application/json" \
  https://<quay-server.example.com>/api/v1/repository/<repository>/manifest/<manifestref>
```

2. Use the **GET /api/v1/repository/{repository}/manifest/{manifestref}/labels** command to retrieve a list of labels for a specific manifest:

```
$ curl -X GET \
  -H "Authorization: Bearer <bearer_token>" \
  -H "Accept: application/json" \
  https://<quay-
server.example.com>/api/v1/repository/<repository>/manifest/<manifestref>/labels
```

Example output

```
{"labels": [{"id": "e9f717d2-c1dd-4626-802d-733a029d17ad", "key":
"org.opencontainers.image.url", "value": "https://github.com/docker-library/busybox",
"source_type": "manifest", "media_type": "text/plain"}, {"id": "2d34ec64-4051-43ad-ae06-
d5f81003576a", "key": "org.opencontainers.image.version", "value": "1.36.1-glibc",
"source_type": "manifest", "media_type": "text/plain"}]}
```

3. Use the **GET /api/v1/repository/{repository}/manifest/{manifestref}/labels/{labelid}** command to obtain information about a specific manifest:

```
$ curl -X GET \
  -H "Authorization: Bearer <bearer_token>" \
  -H "Accept: application/json" \
  https://<quay-
server.example.com>/api/v1/repository/<repository>/manifest/<manifestref>/labels/<label_id>
```

Example output

```
{"id": "e9f717d2-c1dd-4626-802d-733a029d17ad", "key": "org.opencontainers.image.url",
"value": "https://github.com/docker-library/busybox", "source_type": "manifest", "media_type":
"text/plain"}
```

4. You can add an additional label to a manifest in a given repository with the **POST /api/v1/repository/{repository}/manifest/{manifestref}/labels** command. For example:

```
$ curl -X POST \
  -H "Authorization: Bearer <bearer_token>" \
  -H "Content-Type: application/json" \
  --data '{
    "key": "<key>",
    "value": "<value>",
    "media_type": "<media_type>"
  }' \
  https://<quay-
server.example.com>/api/v1/repository/<repository>/manifest/<manifestref>/labels
```

Example output

```
{"label": {"id": "346593fd-18c8-49db-854f-4cb1fb76ff9c", "key": "example-key", "value":
"example-value", "source_type": "api", "media_type": "text/plain"}}
```

5. You can delete a label using the https://docs.redhat.com/en/documentation/red_hat_quay/3/html-single/red_hat_quay_api_guide/index#deletemanifestlabel**DELETE /api/v1/repository/{repository}/manifest/{manifestref}/labels/{labelid}**] command:

```
$ curl -X DELETE \
  -H "Authorization: Bearer <bearer_token>" \
  https://<quay-
server.example.com>/api/v1/repository/<repository>/manifest/<manifestref>/labels/<labelid>
```

This command does not return output in the CLI. You can use one of the commands above to ensure that it was successfully removed.

7.7. SETTING TAG EXPIRATIONS

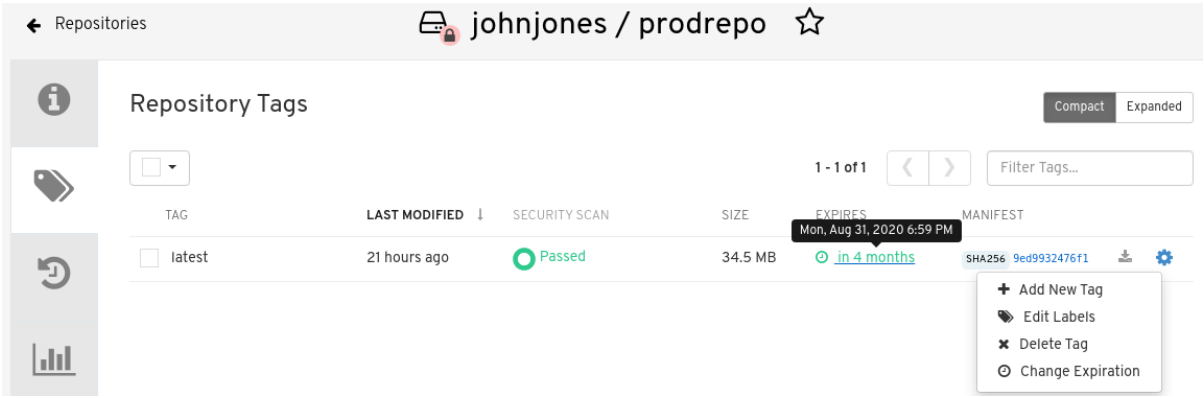
Image tags can be set to expire from a Red Hat Quay repository at a chosen date and time using the *tag expiration* feature. This feature includes the following characteristics:

- When an image tag expires, it is deleted from the repository. If it is the last tag for a specific image, the image is also set to be deleted.
- Expiration is set on a per-tag basis. It is not set for a repository as a whole.
- After a tag is expired or deleted, it is not immediately removed from the registry. This is contingent upon the allotted time designed in the *time machine* feature, which defines when the tag is permanently deleted, or garbage collected. By default, this value is set at *14 days*, however the administrator can adjust this time to one of multiple options. Up until the point that garbage collection occurs, tags changes can be reverted.

The Red Hat Quay superuser has no special privilege related to deleting expired images from user repositories. There is no central mechanism for the superuser to gather information and act on user repositories. It is up to the owners of each repository to manage expiration and the deletion of their images.

Tag expiration can be set up in one of two ways:

- By setting the **quay.expires-after=** label in the Dockerfile when the image is created. This sets a time to expire from when the image is built.
- By selecting an expiration date on the Red Hat Quay UI. For example:

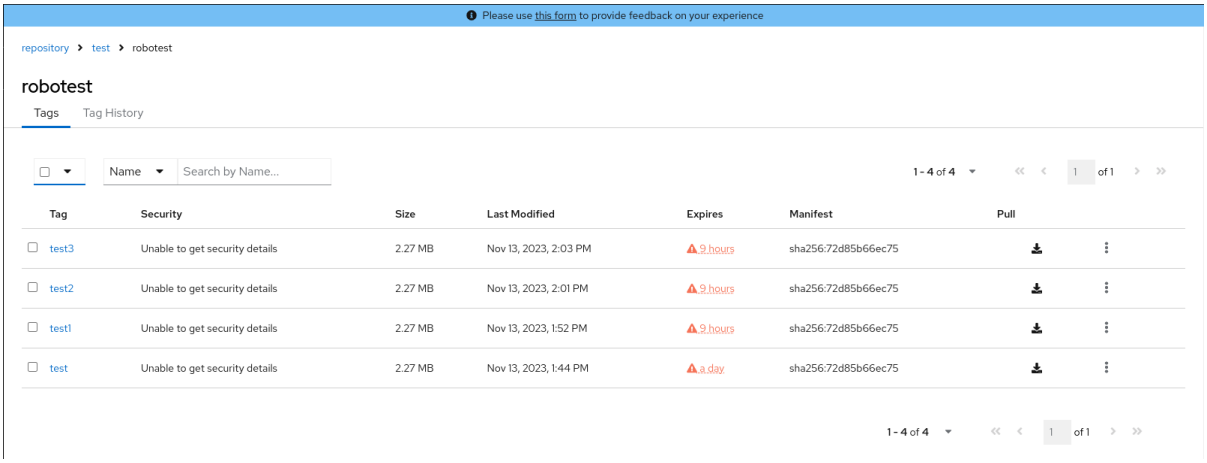


Setting tag expirations can help automate the cleanup of older or unused tags, helping to reduce storage space.

7.7.1. Setting tag expiration from a repository

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:



7.7.2. Setting tag expiration from a Dockerfile

You can add a label, for example, **quay.expires-after=20h** to an image tag by using the **docker label** command to cause the tag to automatically expire after the time that is indicated. The following values for hours, days, or weeks are accepted:

- **1h**
- **2d**
- **3w**

Expiration begins from the time that the image is pushed to the registry.

Procedure

- Enter the following **docker label** command to add a label to the desired image tag. The label should be in the format **quay.expires-after=20h** to indicate that the tag should expire after 20 hours. Replace 20h with the desired expiration time. For example:

```
$ docker label quay.expires-after=20h quay-server.example.com/quayadmin/<image>:<tag>
```

7.7.3. Setting tag expirations by using the API

Image tags can be set to expire by using the API.

Prerequisites

- You have Created an OAuth access token.
- You have set **BROWSER_API_CALLS_XHR_ONLY: false** in your **config.yaml** file.

Procedure

- You can set when an image a tag expires by using the **PUT /api/v1/repository/{repository}/tag/{tag}** command and passing in the expiration field:

```
$ curl -X PUT \
  -H "Authorization: Bearer <bearer_token>" \
  -H "Content-Type: application/json" \
  --data '{
```

```
    "manifest_digest": "<manifest_digest>"
  }' \
  https://<quay-
server.example.com>/api/v1/repository/<namespace>/<repository_name>/tag/<tag>
```

Example output

```
"Updated"
```

7.8. FETCHING AN IMAGE BY TAG OR DIGEST

Red Hat Quay offers multiple ways of pulling images using Docker and Podman clients.

Procedure

1. Navigate to the **Tags** page of a repository.
2. Under **Manifest**, click the **Fetch Tag** icon.
3. When the popup box appears, users are presented with the following options:
   - Podman Pull (by tag)
   - Docker Pull (by tag)
   - Podman Pull (by digest)
   - Docker Pull (by digest)
     Selecting any one of the four options returns a command for the respective client that allows users to pull the image.
4. Click **Copy Command** to copy the command, which can be used on the command-line interface (CLI). For example:

```
$ podman pull quay-server.example.com/quayadmin/busybox:test2
```

7.9. VIEWING RED HAT QUAY TAG HISTORY BY USING THE UI

Red Hat Quay offers a comprehensive history of images and their respective image tags.

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

7.10. VIEWING RED HAT QUAY TAG HISTORY BY USING THE API

Red Hat Quay offers a comprehensive history of images and their respective image tags.

Prerequisites

- You have Created an OAuth access token.
- You have set **BROWSER_API_CALLS_XHR_ONLY: false** in your **config.yaml** file.

Procedure

1. Enter the following command to view tag history by using the **GET /api/v1/repository/{repository}/tag/** command and passing in one of the following queries:
   - **onlyActiveTags=<true/false>**: Filters to only include active tags.
   - **page=<number>**: Specifies the page number of results to retrieve.
   - **limit=<number>**: Limits the number of results per page.
   - **specificTag=<tag_name>**: Filters the tags to include only the tag with the specified name.

     ```
     $ curl -X GET \
       -H "Authorization: Bearer <bearer_token>" \
       -H "Accept: application/json" \
       "https://<quay-server.example.com>/api/v1/repository/<namespace>/<repository>/tag/?onlyActiveTags=true&page=1&limit=10"
     ```

     Example output

     ```
     {"tags": [{"name": "test-two", "reversion": false, "start_ts": 1717680780, "manifest_digest": "sha256:57583a1b9c0a7509d3417387b4f43acf80d08cdcf5266ac87987be3f8f919d5d", "is_manifest_list": false, "size": 2275314, "last_modified": "Thu, 06 Jun 2024 13:33:00 -0000"}, {"name": "tag-test", "reversion": false, "start_ts": 1717680378, "manifest_digest": "sha256:57583a1b9c0a7509d3417387b4f43acf80d08cdcf5266ac87987be3f8f919d5d", "is_manifest_list": false, "size": 2275314, "last_modified": "Thu, 06 Jun 2024 13:26:18 -0000"}, {"name": "example", "reversion": false, "start_ts": 1715698131, "manifest_digest": "sha256:57583a1b9c0a7509d3417387b4f43acf80d08cdcf5266ac87987be3f8f919d5d", "is_manifest_list": false, "size": 2275314, "last_modified": "Tue, 14 May 2024 14:48:51 -0000"}], "page": 1, "has_additional": false}
     ```

2. By using the **specificTag=<tag_name>** query, you can filter results for a specific tag. For example:

   ```
   $ curl -X GET   -H "Authorization: Bearer lfV4lVf9qRsyoFnrgEno1umIOrsdp8lPyMnfUDYY" -H "Accept: application/json"   "quay-server.example.com/api/v1/repository/quayadmin/busybox/tag/?onlyActiveTags=true&page=1&limit=20&specificTag=test-two"
   ```

   Example output

{"tags": [{"name": "test-two", "reversion": true, "start_ts": 1718737153, "manifest_digest": "sha256:57583a1b9c0a7509d3417387b4f43acf80d08cdcf5266ac87987be3f8f919d5d", "is_manifest_list": false, "size": 2275314, "last_modified": "Tue, 18 Jun 2024 18:59:13 -0000"}], "page": 1, "has_additional": false}

7.11. DELETING AN IMAGE TAG

Deleting an image tag removes that specific version of the image from the registry.

To delete an image tag, use the following procedure.

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.

NOTE

Deleting an image tag can be reverted based on the amount of time allotted assigned to the *time machine* feature. For more information, see "Reverting tag changes".

7.12. DELETING AN IMAGE BY USING THE API

You can delete an old image tag by using the API.

Prerequisites

- You have Created an OAuth access token.
- You have set **BROWSER_API_CALLS_XHR_ONLY: false** in your **config.yaml** file.

Procedure

1. You can delete an image tag by using the **DELETE /api/v1/repository/{repository}/tag/{tag}** command:

```
$ curl -X DELETE \
  -H "Authorization: Bearer <bearer_token>" \
  https://<quay-
server.example.com>/api/v1/repository/<namespace>/<repository_name>/tag/<tag>
```

-

This command does not return output in the CLI. Continue on to the next step to return a list of tags.

2. To see a list of tags after deleting a tag, you can use the **GET /api/v1/repository/{repository}/tag/** command. For example:

```
$ curl -X GET \
  -H "Authorization: Bearer <bearer_token>" \
  -H "Accept: application/json" \
   https://<quay-
server.example.com>/api/v1/repository/<namespace>/<repository_name>/tag
```

Example output

```
{"tags": [{"name": "test", "reversion": false, "start_ts": 1716324069, "manifest_digest":
"sha256:57583a1b9c0a7509d3417387b4f43acf80d08cdcf5266ac87987be3f8f919d5d",
"is_manifest_list": false, "size": 2275314, "last_modified": "Tue, 21 May 2024 20:41:09 -
0000"}, {"name": "example", "reversion": false, "start_ts": 1715698131, "manifest_digest":
"sha256:57583a1b9c0a7509d3417387b4f43acf80d08cdcf5266ac87987be3f8f919d5d",
"is_manifest_list": false, "size": 2275314, "last_modified": "Tue, 14 May 2024 14:48:51 -
0000"}, {"name": "example", "reversion": false, "start_ts": 1715697708, "end_ts":
1715698131, "manifest_digest":
"sha256:57583a1b9c0a7509d3417387b4f43acf80d08cdcf5266ac87987be3f8f919d5d",
"is_manifest_list": false, "size": 2275314, "last_modified": "Tue, 14 May 2024 14:41:48 -
0000", "expiration": "Tue, 14 May 2024 14:48:51 -0000"}, {"name": "test", "reversion": false,
"start_ts": 1715695488, "end_ts": 1716324069, "manifest_digest":
"sha256:57583a1b9c0a7509d3417387b4f43acf80d08cdcf5266ac87987be3f8f919d5d",
"is_manifest_list": false, "size": 2275314, "last_modified": "Tue, 14 May 2024 14:04:48 -
0000", "expiration": "Tue, 21 May 2024 20:41:09 -0000"}, {"name": "test", "reversion": false,
"start_ts": 1715631517, "end_ts": 1715695488, "manifest_digest":
"sha256:57583a1b9c0a7509d3417387b4f43acf80d08cdcf5266ac87987be3f8f919d5d",
"is_manifest_list": false, "size": 2275314, "last_modified": "Mon, 13 May 2024 20:18:37 -
0000", "expiration": "Tue, 14 May 2024 14:04:48 -0000"}], "page": 1, "has_additional": false}
```

7.13. REVERTING TAG CHANGES BY USING THE UI

Red Hat Quay offers a comprehensive *time machine* feature that allows older images tags to remain in the repository for set periods of time so that they can revert changes made to tags. This feature allows users to revert tag changes, like tag deletions.

Procedure

1. On the **Repositories** page of the v2 UI, click the name of the image you want to revert.
2. Click the **Tag History** tab.
3. Find the point in the timeline at which image tags were changed or removed. Next, click the option under **Revert** to restore a tag to its image.

7.14. REVERTING TAG CHANGES BY USING THE API

Red Hat Quay offers a comprehensive *time machine* feature that allows older images tags to remain in the repository for set periods of time so that they can revert changes made to tags. This feature allows users to revert tag changes, like tag deletions.

Prerequisites

- You have Created an OAuth access token.
- You have set **BROWSER_API_CALLS_XHR_ONLY: false** in your **config.yaml** file.

Procedure

1. You can restore a repository tag to its previous image by using the **POST /api/v1/repository/{repository}/tag/{tag}/restore** command. For example:

```
$ curl -X POST \
  -H "Authorization: Bearer <bearer_token>" \
  -H "Content-Type: application/json" \
  --data '{
    "manifest_digest": <manifest_digest>
  }' \
  quay-server.example.com/api/v1/repository/quayadmin/busybox/tag/test/restore
```

Example output

```
{}
```

2. To see a list of tags after restoring an old tag you can use the **GET /api/v1/repository/{repository}/tag/** command. For example:

```
$ curl -X GET \
  -H "Authorization: Bearer <bearer_token>" \
  -H "Accept: application/json" \
   https://<quay-
server.example.com>/api/v1/repository/<namespace>/<repository_name>/tag
```

Example output

```
{"tags": [{"name": "test", "reversion": false, "start_ts": 1716324069, "manifest_digest":
"sha256:57583a1b9c0a7509d3417387b4f43acf80d08cdcf5266ac87987be3f8f919d5d",
"is_manifest_list": false, "size": 2275314, "last_modified": "Tue, 21 May 2024 20:41:09 -
0000"}, {"name": "example", "reversion": false, "start_ts": 1715698131, "manifest_digest":
"sha256:57583a1b9c0a7509d3417387b4f43acf80d08cdcf5266ac87987be3f8f919d5d",
"is_manifest_list": false, "size": 2275314, "last_modified": "Tue, 14 May 2024 14:48:51 -
0000"}, {"name": "example", "reversion": false, "start_ts": 1715697708, "end_ts":
1715698131, "manifest_digest":
"sha256:57583a1b9c0a7509d3417387b4f43acf80d08cdcf5266ac87987be3f8f919d5d",
"is_manifest_list": false, "size": 2275314, "last_modified": "Tue, 14 May 2024 14:41:48 -
0000", "expiration": "Tue, 14 May 2024 14:48:51 -0000"}, {"name": "test", "reversion": false,
"start_ts": 1715695488, "end_ts": 1716324069, "manifest_digest":
"sha256:57583a1b9c0a7509d3417387b4f43acf80d08cdcf5266ac87987be3f8f919d5d",
"is_manifest_list": false, "size": 2275314, "last_modified": "Tue, 14 May 2024 14:04:48 -
0000", "expiration": "Tue, 21 May 2024 20:41:09 -0000"}, {"name": "test", "reversion": false,
"start_ts": 1715631517, "end_ts": 1715695488, "manifest_digest":
```

"sha256:57583a1b9c0a7509d3417387b4f43acf80d08cdcf5266ac87987be3f8f919d5d", "is_manifest_list": false, "size": 2275314, "last_modified": "Mon, 13 May 2024 20:18:37 -0000", "expiration": "Tue, 14 May 2024 14:04:48 -0000"}], "page": 1, "has_additional": false}

CHAPTER 8. VIEWING AND EXPORTING LOGS

Activity logs are gathered for all repositories and namespace in Red Hat Quay.

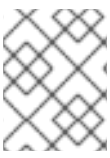
Viewing usage logs of Red Hat Quay. can provide valuable insights and benefits for both operational and security purposes. Usage logs might reveal the following information:

- **Resource Planning**: Usage logs can provide data on the number of image pulls, pushes, and overall traffic to your registry.
- **User Activity**: Logs can help you track user activity, showing which users are accessing and interacting with images in the registry. This can be useful for auditing, understanding user behavior, and managing access controls.
- **Usage Patterns**: By studying usage patterns, you can gain insights into which images are popular, which versions are frequently used, and which images are rarely accessed. This information can help prioritize image maintenance and cleanup efforts.
- **Security Auditing**: Usage logs enable you to track who is accessing images and when. This is crucial for security auditing, compliance, and investigating any unauthorized or suspicious activity.
- **Image Lifecycle Management**: Logs can reveal which images are being pulled, pushed, and deleted. This information is essential for managing image lifecycles, including deprecating old images and ensuring that only authorized images are used.
- **Compliance and Regulatory Requirements**: Many industries have compliance requirements that mandate tracking and auditing of access to sensitive resources. Usage logs can help you demonstrate compliance with such regulations.
- **Identifying Abnormal Behavior**: Unusual or abnormal patterns in usage logs can indicate potential security breaches or malicious activity. Monitoring these logs can help you detect and respond to security incidents more effectively.
- **Trend Analysis**: Over time, usage logs can provide trends and insights into how your registry is being used. This can help you make informed decisions about resource allocation, access controls, and image management strategies.

There are multiple ways of accessing log files:

- Viewing logs through the web UI.
- Exporting logs so that they can be saved externally.
- Accessing log entries using the API.

To access logs, you must have administrative privileges for the selected repository or namespace.



NOTE

A maximum of 100 log results are available at a time via the API. To gather more results that that, you must use the log exporter feature described in this chapter.

8.1. VIEWING USAGE LOGS

Logs can provide valuable information about the way that your 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.

8.2. VIEWING USAGE LOGS BY USING THE API

Logs can be viewed by Organization or repository by using the API. They can also be aggregated (grouped), or listed with more detailed. Logs can also be viewed by user, a specific date range, or by page.

8.2.1. Viewing aggregated logs

Aggregated logs can be viewed by Organization, repository, a specific user, or the current user. You can also pass in optional commands like **performer**, **starttime/endtime**, and **next_page** to filter results.

Prerequisites

- You have Created an OAuth access token.
- You have set **BROWSER_API_CALLS_XHR_ONLY: false** in your **config.yaml** file.

Procedure

1. Use the **GET /api/v1/user/aggregatelogs** API endpoint to return the aggregated (or grouped) logs for the current user:

```
$ curl -X GET \
```

```
  -H "Authorization: Bearer <bearer_token>" \
  -H "Accept: application/json" \
  "https://<quay-server.example.com>/api/v1/user/aggregatelogs"
```

Example output

```
{"aggregated": [{"kind": "create_tag", "count": 1, "datetime": "Tue, 18 Jun 2024 00:00:00 -0000"}, {"kind": "manifest_label_add", "count": 1, "datetime": "Tue, 18 Jun 2024 00:00:00 -0000"}, {"kind": "push_repo", "count": 2, "datetime": "Tue, 18 Jun 2024 00:00:00 -0000"}, {"kind": "revert_tag", "count": 1, "datetime": "Tue, 18 Jun 2024 00:00:00 -0000"}]}
```

You can also pass in the **performer** and **starttime/endtime** queries to obtain aggregated logs for a specific user between a specific time period. For example:

```
$ curl -X GET \
  -H "Authorization: Bearer <bearer_token>" \
  -H "Accept: application/json" \
  "<quay-server.example.com>/api/v1/user/aggregatelogs?performer=
<username>&starttime=<MM/DD/YYYY>&endtime=<MM/DD/YYYY>"
```

2. Aggregated logs can also be viewed by Organization by using the **GET /api/v1/organization/{orgname}/aggregatelogs**. For example:

```
$ curl -X GET \
  -H "Authorization: Bearer <bearer_token>" \
  -H "Accept: application/json" \
  "<quay-server.example.com>/api/v1/organization/{orgname}/aggregatelogs"
```

3. Aggregated logs can also be viewed by repository by using the **GET /api/v1/repository/{repository}/aggregatelogs** command. The following example includes the **starttime/endtime** fields:

```
$ curl -X GET \
  -H "Authorization: Bearer <bearer_token>" \
  -H "Accept: application/json" \
  "<quay-
server.example.com>/api/v1/repository/<repository_name>/<namespace>/aggregatelogs?
starttime=2024-01-01&endtime=2024-06-18""
```

8.2.2. Viewing detailed logs

Detailed logs can be viewed by Organization, repository, a specific user, or the current user. You can also pass in optional fields like **performer**, **starttime/endtime**, and **next_page** to filter results.

Procedure

1. Use the **GET /api/v1/user/logs** API endpoint to return a list of log entries for a user. For example:

```
$ curl -X GET   -H "Authorization: Bearer lfV4lVf9qRsyoFnrgEno1umIOrsdp8lPyMnfUDYY"
-H "Accept: application/json"   "quay-server.example.com/api/v1/user/logs"
```

You can also pass in the **performer** and **startime/endtime** queries to obtain logs for a specific user between a specific time period. For example:

```
$ curl -X GET    -H "Authorization: Bearer lfV4lVf9qRsyoFnrgEno1umIOrsdp8lPyMnfUDYY"
-H "Accept: application/json"    "http://quay-server.example.com/api/v1/user/logs?
performer=quayuser&starttime=01/01/2024&endtime=06/18/2024"
```

Example output

```
---
{"start_time": "Mon, 01 Jan 2024 00:00:00 -0000", "end_time": "Wed, 19 Jun 2024 00:00:00 -
0000", "logs": [{"kind": "revert_tag", "metadata": {"username": "quayuser", "repo": "busybox",
"tag": "test-two", "manifest_digest":
"sha256:57583a1b9c0a7509d3417387b4f43acf80d08cdcf5266ac87987be3f8f919d5d"}, "ip":
"192.168.1.131", "datetime": "Tue, 18 Jun 2024 18:59:13 -0000", "performer": {"kind": "user",
"name": "quayuser", "is_robot": false, "avatar": {"name": "quayuser", "hash":
"b28d563a6dc76b4431fc7b0524bbff6b810387dac86d9303874871839859c7cc", "color":
"#17becf", "kind": "user"}}}, {"kind": "push_repo", "metadata": {"repo": "busybox",
"namespace": "quayuser", "user-agent": "containers/5.30.1 (github.com/containers/image)",
"tag": "test-two", "username": "quayuser", }
---
```

2. Use the **GET /api/v1/organization/{orgname}/logs** endpoint to return logs for a specified organization:

```
$ curl -X GET \
  -H "Authorization: Bearer <bearer_token>" \
  -H "Accept: application/json" \
  "http://<quay-server.example.com>/api/v1/organization/{orgname}/logs"
```

3. Use the **GET /api/v1/repository/{repository}/logs** endpoint to return logs for a specified repository:

```
$ curl -X GET \
  -H "Authorization: Bearer <bearer_token>" \
  -H "Accept: application/json" \
  "http://<quay-server.example.com>/api/v1/repository/{repository}/logs"
```

8.3. EXPORTING REPOSITORY LOGS BY USING THE UI

You can obtain a larger number of log files and save them outside of the Red Hat Quay database by using the **Export Logs** feature. This feature has the following benefits and constraints:

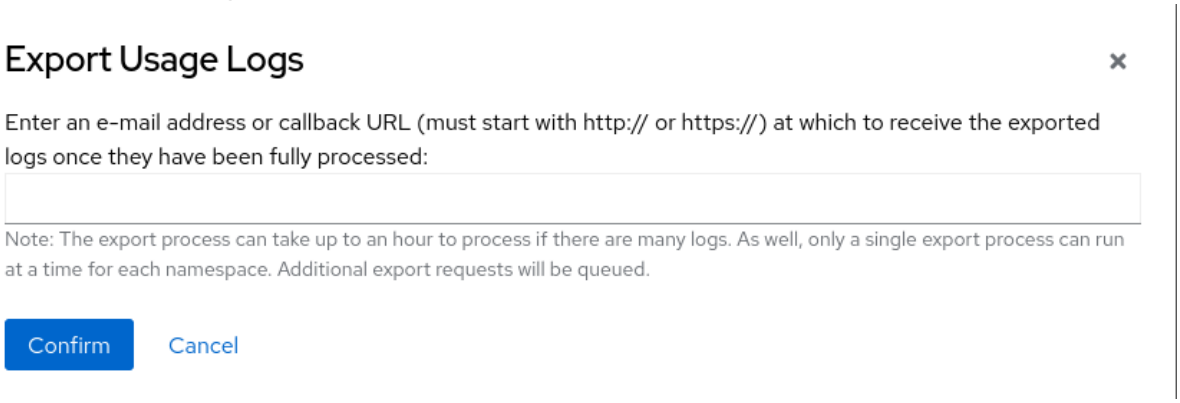
- You can choose a range of dates for the logs you want to gather from a repository.
- You can request that the logs be sent to you by an email attachment or directed to a callback URL.
- To export logs, you must be an administrator of the repository or namespace.
- 30 days worth of logs are retained for all users.
- Export logs only gathers log data that was previously produced. It does not stream logging data.

- Your Red Hat Quay instance must be configured for external storage for this feature. Local storage does not work for exporting logs.
- When logs are gathered and made available to you, you should immediately copy that data if you want to save it. By default, the data expires after one hour.

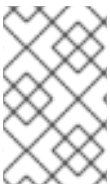
Use the following procedure to export logs.

Procedure

1. Select a repository for which you have administrator privileges.
2. Click the **Logs** tab.
3. Optional. If you want to specify specific dates, enter the range in the **From** and **to** boxes.
4. Click the **Export Logs** button. An Export Usage Logs pop-up appears, as shown



5. Enter an email address or callback URL to receive the exported log. For the callback URL, you can use a URL to a specified domain, for example, <webhook.site>.
6. Select **Confirm** to start the process for gather the selected log entries. Depending on the amount of logging data being gathered, this can take anywhere from a few minutes to several hours to complete.
7. When the log export is completed, the one of following two events happens:
   - An email is received, alerting you to the available of your requested exported log entries.
   - A successful status of your log export request from the webhook URL is returned. Additionally, a link to the exported data is made available for you to delete to download the logs.



NOTE

The URL points to a location in your Red Hat Quay external storage and is set to expire within one hour. Make sure that you copy the exported logs before the expiration time if you intend to keep your logs.

8.4. EXPORTING LOGS BY USING THE API

Detailed logs can be exported to a callback URL or to an email address.

Prerequisites

- You have Created an OAuth access token.
- You have set **BROWSER_API_CALLS_XHR_ONLY: false** in your **config.yaml** file.

Procedure

1. Use the **POST /api/v1/user/exportlogs** endpoint to export logs for the current user:

```
$ curl -X POST \
  -H "Authorization: Bearer <bearer_token>" \
  -H "Content-Type: application/json" \
  -H "Accept: application/json" \
  -d '{
        "starttime": "<MM/DD/YYYY>",
        "endtime": "<MM/DD/YYYY>",
        "callback_email": "your.email@example.com"
      }' \
  "http://<quay-server.example.com>/api/v1/user/exportlogs"
```

Example output

```
{"export_id": "6a0b9ea9-444c-4a19-9db8-113201c38cd4"}
```

2. Use the **POST /api/v1/organization/{orgname}/exportlogs** endpoint to export logs for an Organization:

```
$ curl -X POST \
  -H "Authorization: Bearer <bearer_token>" \
  -H "Content-Type: application/json" \
  -H "Accept: application/json" \
  -d '{
        "starttime": "<MM/DD/YYYY>",
        "endtime": "<MM/DD/YYYY>",
        "callback_email": "org.logs@example.com"
      }' \
  "http://<quay-server.example.com>/api/v1/organization/{orgname}/exportlogs"
```

3. Use the **POST /api/v1/repository/{repository}/exportlogs** endpoint to export logs for a repository:

```
$ curl -X POST \
  -H "Authorization: Bearer <bearer_token>" \
  -H "Content-Type: application/json" \
  -H "Accept: application/json" \
  -d '{
        "starttime": "2024-01-01",
        "endtime": "2024-06-18",
        "callback_url": "http://your-callback-url.example.com"
      }' \
  "http://<quay-server.example.com>/api/v1/repository/{repository}/exportlogs"
```

CHAPTER 9. CLAIR SECURITY SCANS

Clair security scanner is not enabled for Red Hat Quay by default. To enable Clair, see Clair on Red Hat Quay.

Clair security scans can be viewed on the UI, or by the API.

Procedure

1. Navigate to a repository and click **Tags** in the navigation pane. This page shows the results of the security scan.
2. To reveal more information about multi-architecture images, click **See Child Manifests** to see the list of manifests in extended view.
3. Click a relevant link under **See Child Manifests**, for example, **1 Unknown** to be redirected to the **Security Scanner** page.
4. The **Security Scanner** page provides information for the tag, such as which CVEs the image is susceptible to, and what remediation options you might have available.

NOTE

Image scanning only lists vulnerabilities found by Clair security scanner. What users do about the vulnerabilities are uncovered is up to said user. Red Hat Quay superusers do not act on found vulnerabilities.

9.1. VIEWING CLAIR SECURITY SCANS BY USING THE UI

You can view Clair security scans on the UI.

Procedure

1. Navigate to a repository and click **Tags** in the navigation pane. This page shows the results of the security scan.
2. To reveal more information about multi-architecture images, click **See Child Manifests** to see the list of manifests in extended view.
3. Click a relevant link under **See Child Manifests**, for example, **1 Unknown** to be redirected to the **Security Scanner** page.
4. The **Security Scanner** page provides information for the tag, such as which CVEs the image is susceptible to, and what remediation options you might have available.

NOTE

Image scanning only lists vulnerabilities found by Clair security scanner. What users do about the vulnerabilities are uncovered is up to said user. Red Hat Quay superusers do not act on found vulnerabilities.

9.2. VIEW CLAIR SECURITY SCANS BY USING THE UI

You can view Clair security scans by using the API.

Procedure

- Use the **GET /api/v1/repository/{repository}/manifest/{manifestref}/security** endpoint to retrieve security information about a specific manifest in a repository. For example:

```
$ curl -X GET \
  -H "Authorization: Bearer <bearer_token>" \
  -H "Accept: application/json" \
  "https://quay-
server.example.com/api/v1/repository/<namespace>/<repository>/manifest/<manifest_digest>/s
ecurity?vulnerabilities=<true_or_false>"
```

Example output

```
{"status": "queued", "data": null}
```

CHAPTER 10. NOTIFICATIONS OVERVIEW

Red Hat Quay supports adding *notifications* to a repository for various events that occur in the repository's lifecycle.

10.1. NOTIFICATION ACTIONS

Notifications are added to the **Events and Notifications** section of the **Repository Settings** page. They are also added to the **Notifications** window, which can be found by clicking the *bell* icon in the navigation pane of Red Hat Quay.

Red Hat Quay notifications can be setup to be sent to a *User*, *Team*, or the *organization* as a whole.

E-mail notifications
E-mails are sent to specified addresses that describe the specified event. E-mail addresses must be verified on a *per-repository* basis.

Webhook POST notifications
An HTTP **POST** call is made to the specified URL with the event's data. For more information about event data, see "Repository events description".

When the URL is HTTPS, the call has an SSL client certificate set from Red Hat Quay. Verification of this certificate proves that the call originated from Red Hat Quay. Responses with the status code in the **2xx** range are considered successful. Responses with any other status code are considered failures and result in a retry of the webhook notification.

Flowdock notifications
Posts a message to Flowdock.

Hipchat notifications
Posts a message to HipChat.

Slack notifications
Posts a message to Slack.

10.2. CREATING NOTIFICATIONS BY USING THE UI

Use the following procedure to add notifications.

Prerequisites

- You have created a repository.
- You have administrative privileges for the repository.

Procedure

1. Navigate to a repository on Red Hat Quay.
2. In the navigation pane, click **Settings**.
3. In the **Events and Notifications** category, click **Create Notification** to add a new notification for a repository event. The **Create notification** popup box appears.

4. On the **Create repository** popup box, click the **When this event occurs** box to select an event. You can select a notification for the following types of events:
    - Push to Repository
    - Image build failed
    - Image build queued
    - Image build started
    - Image build success
    - Image build cancelled
    - Image expiry trigger
5. After you have selected the event type, select the notification method. The following methods are supported:
    - Quay Notification
    - E-mail Notification
    - Webhook POST
    - Flowdock Team Notification
    - HipChat Room Notification
    - Slack Notification
      Depending on the method that you choose, you must include additional information. For example, if you select **E-mail**, you are required to include an e-mail address and an optional notification title.
6. After selecting an event and notification method, click **Create Notification**.

10.2.1. Creating an image expiration notification

Image expiration event triggers can be configured to notify users through email, Slack, webhooks, and so on, and can be configured at the repository level. Triggers can be set for images expiring in any amount of days, and can work in conjunction with the auto-pruning feature.

Image expiration notifications can be set by using the Red Hat Quay v2 UI or by using the **createRepoNotification** API endpoint.

Prerequisites

- **FEATURE_GARBAGE_COLLECTION: true** is set in your **config.yaml** file.
- Optional. **FEATURE_AUTO_PRUNE: true** is set in your **config.yaml** file.

Procedure

1. On the Red Hat Quay v2 UI, click **Repositories**.
2. Select the name of a repository.

3. Click **Settings** → **Events and notifications**.
4. Click **Create notification**. The **Create notification** popup box appears.
5. Click the **Select event...** box, then click **Image expiry trigger**.
6. In the **When the image is due to expiry in days** box, enter the number of days before the image's expiration when you want to receive an alert. For example, use **1** for 1 day.
7. In the **Select method...** box, click one of the following:
   - E-mail
   - Webhook POST
   - Flowdock Team Notification
   - HipChat Room Notification
   - Slack Notification
8. Depending on which method you chose, include the necessary data. For example, if you chose **Webhook POST**, include the **Webhook URL**.
9. Optional. Provide a **POST JSON body template**.
10. Optional. Provide a **Title** for your notification.
11. Click **Submit**. You are returned to the **Events and notifications** page, and the notification now appears.
12. Optional. You can set the **NOTIFICATION_TASK_RUN_MINIMUM_INTERVAL_MINUTES** variable in your config.yaml file. with this field set, if there are any expiring images notifications will be sent automatically. By default, this is set to **300**, or 5 hours, however it can be adjusted as warranted.

```
NOTIFICATION_TASK_RUN_MINIMUM_INTERVAL_MINUTES: 300 (1)
```

(1) By default, this field is set to **300**, or 5 hours.

Verification

1. Click the menu kebab → **Test Notification**. The following message is returned:

```
Test Notification Queued
A test version of this notification has been queued and should appear shortly
```

2. Depending on which method you chose, check your e-mail, webhook address, Slack channel, and so on. The information sent should look similar to the following example:

```
{
  "repository": "sample_org/busybox",
  "namespace": "sample_org",
  "name": "busybox",
  "docker_url": "quay-server.example.com/sample_org/busybox",
```

```
  "homepage": "http://quay-server.example.com/repository/sample_org/busybox",
  "tags": [
    "latest",
    "v1"
  ],
  "expiring_in": "1 days"
}
```

10.3. CREATING NOTIFICATIONS BY USING THE API

Use the following procedure to add notifications.

Prerequisites

- You have created a repository.
- You have administrative privileges for the repository.
- You have Created an OAuth access token.
- You have set **BROWSER_API_CALLS_XHR_ONLY: false** in your **config.yaml** file.

Procedure

1. Enter the following **POST /api/v1/repository/{repository}/notification** command to create a notification on your repository:

```
$ curl -X POST \
  -H "Authorization: Bearer <bearer_token>" \
  -H "Content-Type: application/json" \
  --data '{
    "event": "<event>",
    "method": "<method>",
    "config": {
      "<config_key>": "<config_value>"
    },
    "eventConfig": {
      "<eventConfig_key>": "<eventConfig_value>"
    }
  }' \
  https://<quay-
server.example.com>/api/v1/repository/<namespace>/<repository_name>/notification/
```

This command does not return output in the CLI. Instead, you can enter the following **GET /api/v1/repository/{repository}/notification/{uuid}** command to obtain information about the repository notification:

```
{"uuid": "240662ea-597b-499d-98bb-2b57e73408d6", "title": null, "event": "repo_push",
"method": "quay_notification", "config": {"target": {"name": "quayadmin", "kind": "user",
"is_robot": false, "avatar": {"name": "quayadmin", "hash":
"b28d563a6dc76b4431fc7b0524bbff6b810387dac86d9303874871839859c7cc", "color":
"#17becf", "kind": "user"}}}, "event_config": {}, "number_of_failures": 0}
```

2. You can test your repository notification by entering the following **POST /api/v1/repository/{repository}/notification/{uuid}/test** command:

```
$ curl -X POST \
  -H "Authorization: Bearer <bearer_token>" \
  https://<quay-server.example.com>/api/v1/repository/<repository>/notification/<uuid>/test
```

Example output

```
{}
```

3. You can reset repository notification failures to 0 by entering the following **POST /api/v1/repository/{repository}/notification/{uuid}** command:

```
$ curl -X POST \
  -H "Authorization: Bearer <bearer_token>" \
  https://<quay-server.example.com>/api/v1/repository/<repository>/notification/<uuid>
```

4. Enter the following **DELETE /api/v1/repository/{repository}/notification/{uuid}** command to delete a repository notification:

```
$ curl -X DELETE \
  -H "Authorization: Bearer <bearer_token>" \
  https://<quay-
server.example.com>/api/v1/repository/<namespace>/<repository_name>/notification/<uuid>
```

This command does not return output in the CLI. Instead, you can enter the following **GET /api/v1/repository/{repository}/notification/{uuid}** command to retrieve a list of all notifications:

```
$ curl -X GET  -H "Authorization: Bearer <bearer_token>"   -H "Accept: application/json"
https://<quay-
server.example.com>/api/v1/repository/<namespace>/<repository_name>/notification/
```

Example output

```
{"notifications": []}
```

10.4. REPOSITORY EVENTS DESCRIPTION

The following sections detail repository events.

Repository Push

A successful push of one or more images was made to the repository:

```
{
  "name": "repository",
  "repository": "dgangaia/test",
  "namespace": "dgangaia",
  "docker_url": "quay.io/dgangaia/test",
  "homepage": "https://quay.io/repository/dgangaia/repository",
  "updated_tags": [
```

```
    "latest"
  ]
}
```

Dockerfile Build Queued

The following example is a response from a Dockerfile Build that has been queued into the Build system.

NOTE

Responses can differ based on the use of optional attributes.

```
{
  "build_id": "296ec063-5f86-4706-a469-f0a400bf9df2",
  "trigger_kind": "github",                                          //Optional
  "name": "test",
  "repository": "dgangaia/test",
  "namespace": "dgangaia",
  "docker_url": "quay.io/dgangaia/test",
  "trigger_id": "38b6e180-9521-4ff7-9844-acf371340b9e",                    //Optional
  "docker_tags": [
    "master",
    "latest"
  ],
  "repo": "test",
  "trigger_metadata": {
    "default_branch": "master",
    "commit": "b7f7d2b948aacbe844ee465122a85a9368b2b735",
    "ref": "refs/heads/master",
    "git_url": "git@github.com:dgangaia/test.git",
    "commit_info": {                                                 //Optional
      "url": "https://github.com/dgangaia/test/commit/b7f7d2b948aacbe844ee465122a85a9368b2b735",
      "date": "2019-03-06T12:48:24+11:00",
      "message": "adding 5",
      "author": {                                                    //Optional
        "username": "dgangaia",
        "url": "https://github.com/dgangaia",                              //Optional
        "avatar_url": "https://avatars1.githubusercontent.com/u/43594254?v=4"    //Optional
      },
      "committer": {
        "username": "web-flow",
        "url": "https://github.com/web-flow",
        "avatar_url": "https://avatars3.githubusercontent.com/u/19864447?v=4"
      }
    }
  },
  "is_manual": false,
  "manual_user": null,
  "homepage": "https://quay.io/repository/dgangaia/test/build/296ec063-5f86-4706-a469-
f0a400bf9df2"
}
```

Dockerfile Build started

The following example is a response from a Dockerfile Build that has been queued into the Build system.

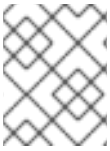
NOTE

Responses can differ based on the use of optional attributes.

```
{
  "build_id": "a8cc247a-a662-4fee-8dcb-7d7e822b71ba",
  "trigger_kind": "github",                                          //Optional
  "name": "test",
  "repository": "dgangaia/test",
  "namespace": "dgangaia",
  "docker_url": "quay.io/dgangaia/test",
  "trigger_id": "38b6e180-9521-4ff7-9844-acf371340b9e",                  //Optional
  "docker_tags": [
    "master",
    "latest"
  ],
  "build_name": "50bc599",
  "trigger_metadata": {                                              //Optional
    "commit": "50bc5996d4587fd4b2d8edc4af652d4cec293c42",
    "ref": "refs/heads/master",
    "default_branch": "master",
    "git_url": "git@github.com:dgangaia/test.git",
    "commit_info": {                                                 //Optional
      "url": "https://github.com/dgangaia/test/commit/50bc5996d4587fd4b2d8edc4af652d4cec293c42",
      "date": "2019-03-06T14:10:14+11:00",
      "message": "test build",
      "committer": {                                               //Optional
        "username": "web-flow",
        "url": "https://github.com/web-flow",                          //Optional
        "avatar_url": "https://avatars3.githubusercontent.com/u/19864447?v=4"   //Optional
      },
      "author": {                                                  //Optional
        "username": "dgangaia",
        "url": "https://github.com/dgangaia",                          //Optional
        "avatar_url": "https://avatars1.githubusercontent.com/u/43594254?v=4"   //Optional
      }
    }
  },
  "homepage": "https://quay.io/repository/dgangaia/test/build/a8cc247a-a662-4fee-8dcb-
7d7e822b71ba"
}
```

Dockerfile Build successfully completed

The following example is a response from a Dockerfile Build that has been successfully completed by the Build system.



NOTE

This event occurs simultaneously with a *Repository Push* event for the built image or images.

```
{
  "build_id": "296ec063-5f86-4706-a469-f0a400bf9df2",
  "trigger_kind": "github",                                          //Optional
```

```
  "name": "test",
  "repository": "dgangaia/test",
  "namespace": "dgangaia",
  "docker_url": "quay.io/dgangaia/test",
  "trigger_id": "38b6e180-9521-4ff7-9844-acf371340b9e",                    //Optional
  "docker_tags": [
    "master",
    "latest"
  ],
  "build_name": "b7f7d2b",
  "image_id": "sha256:0339f178f26ae24930e9ad32751d6839015109eabdf1c25b3b0f2abf8934f6cb",
  "trigger_metadata": {
    "commit": "b7f7d2b948aacbe844ee465122a85a9368b2b735",
    "ref": "refs/heads/master",
    "default_branch": "master",
    "git_url": "git@github.com:dgangaia/test.git",
    "commit_info": {                                                       //Optional
      "url": "https://github.com/dgangaia/test/commit/b7f7d2b948aacbe844ee465122a85a9368b2b735",
      "date": "2019-03-06T12:48:24+11:00",
      "message": "adding 5",
      "committer": {                                                       //Optional
        "username": "web-flow",
        "url": "https://github.com/web-flow",                              //Optional
        "avatar_url": "https://avatars3.githubusercontent.com/u/19864447?v=4"
//Optional
      },
      "author": {                                                          //Optional
        "username": "dgangaia",
        "url": "https://github.com/dgangaia",                              //Optional
        "avatar_url": "https://avatars1.githubusercontent.com/u/43594254?v=4"     //Optional
      }
    }
  },
  "homepage": "https://quay.io/repository/dgangaia/test/build/296ec063-5f86-4706-a469-
f0a400bf9df2",
  "manifest_digests": [

"quay.io/dgangaia/test@sha256:2a7af5265344cc3704d5d47c4604b1efcbd227a7a6a6ff73d6e4e08a27f
d7d99",

"quay.io/dgangaia/test@sha256:569e7db1a867069835e8e97d50c96eccafde65f08ea3e0d5debaf16e25
45d9d1"
  ]
}
```

Dockerfile Build failed

The following example is a response from a Dockerfile Build that has failed.

```
{
  "build_id": "5346a21d-3434-4764-85be-5be1296f293c",
  "trigger_kind": "github",                                                //Optional
  "name": "test",
  "repository": "dgangaia/test",
  "docker_url": "quay.io/dgangaia/test",
  "error_message": "Could not find or parse Dockerfile: unknown instruction: GIT",
  "namespace": "dgangaia",
```

```
  "trigger_id": "38b6e180-9521-4ff7-9844-acf371340b9e",                    //Optional
  "docker_tags": [
    "master",
    "latest"
  ],
  "build_name": "6ae9a86",
  "trigger_metadata": {                                                    //Optional
    "commit": "6ae9a86930fc73dd07b02e4c5bf63ee60be180ad",
    "ref": "refs/heads/master",
    "default_branch": "master",
    "git_url": "git@github.com:dgangaia/test.git",
    "commit_info": {                                                       //Optional
      "url": "https://github.com/dgangaia/test/commit/6ae9a86930fc73dd07b02e4c5bf63ee60be180ad",
      "date": "2019-03-06T14:18:16+11:00",
      "message": "failed build test",
      "committer": {                                                       //Optional
        "username": "web-flow",
        "url": "https://github.com/web-flow",                              //Optional
        "avatar_url": "https://avatars3.githubusercontent.com/u/19864447?v=4"     //Optional
      },
      "author": {                                                          //Optional
        "username": "dgangaia",
        "url": "https://github.com/dgangaia",                              //Optional
        "avatar_url": "https://avatars1.githubusercontent.com/u/43594254?v=4"     //Optional
      }
    }
  },
  "homepage": "https://quay.io/repository/dgangaia/test/build/5346a21d-3434-4764-85be-
5be1296f293c"
}
```

Dockerfile Build cancelled

The following example is a response from a Dockerfile Build that has been cancelled.

```
{
  "build_id": "cbd534c5-f1c0-4816-b4e3-55446b851e70",
  "trigger_kind": "github",
  "name": "test",
  "repository": "dgangaia/test",
  "namespace": "dgangaia",
  "docker_url": "quay.io/dgangaia/test",
  "trigger_id": "38b6e180-9521-4ff7-9844-acf371340b9e",
  "docker_tags": [
    "master",
    "latest"
  ],
  "build_name": "cbce83c",
  "trigger_metadata": {
    "commit": "cbce83c04bfb59734fc42a83aab738704ba7ec41",
    "ref": "refs/heads/master",
    "default_branch": "master",
    "git_url": "git@github.com:dgangaia/test.git",
    "commit_info": {
      "url": "https://github.com/dgangaia/test/commit/cbce83c04bfb59734fc42a83aab738704ba7ec41",
      "date": "2019-03-06T14:27:53+11:00",
      "message": "testing cancel build",
```

```
      "committer": {
        "username": "web-flow",
        "url": "https://github.com/web-flow",
        "avatar_url": "https://avatars3.githubusercontent.com/u/19864447?v=4"
      },
      "author": {
        "username": "dgangaia",
        "url": "https://github.com/dgangaia",
        "avatar_url": "https://avatars1.githubusercontent.com/u/43594254?v=4"
      }
    }
  },
  "homepage": "https://quay.io/repository/dgangaia/test/build/cbd534c5-f1c0-4816-b4e3-
55446b851e70"
}
```

Vulnerability detected

The following example is a response from a Dockerfile Build has detected a vulnerability in the repository.

```
{
  "repository": "dgangaia/repository",
  "namespace": "dgangaia",
  "name": "repository",
  "docker_url": "quay.io/dgangaia/repository",
  "homepage": "https://quay.io/repository/dgangaia/repository",

  "tags": ["latest", "othertag"],

  "vulnerability": {
    "id": "CVE-1234-5678",
    "description": "This is a bad vulnerability",
    "link": "http://url/to/vuln/info",
    "priority": "Critical",
    "has_fix": true
  }
}
```

CHAPTER 11. AUTOMATICALLY BUILDING DOCKERFILES WITH BUILD WORKERS

Red Hat Quay supports building Dockerfiles using a set of worker nodes on OpenShift Container Platform or Kubernetes. Build triggers, such as GitHub webhooks, can be configured to automatically build new versions of your repositories when new code is committed.

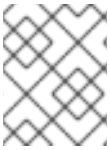
This document shows you how to enable Builds with your Red Hat Quay installation, and set up one more more OpenShift Container Platform or Kubernetes clusters to accept Builds from Red Hat Quay.

11.1. SETTING UP RED HAT QUAY BUILDERS WITH OPENSHIFT CONTAINER PLATFORM

You must pre-configure Red Hat Quay Builders prior to using it with OpenShift Container Platform.

11.1.1. Configuring the OpenShift Container Platform TLS component

The **tls** component allows you to control TLS configuration.



NOTE

Red Hat Quay does not support Builders when the TLS component is managed by the Red Hat Quay Operator.

If you set **tls** to **unmanaged**, you supply your own **ssl.cert** and **ssl.key** files. In this instance, if you want your cluster to support Builders, you must add both the **Quay** route and the Builder route name to the SAN list in the certificate; alternatively you can use a wildcard.

To add the builder route, use the following format:

```
[quayregistry-cr-name]-quay-builder-[ocp-namespace].[ocp-domain-name]
```

11.1.2. Preparing OpenShift Container Platform for Red Hat Quay Builders

Prepare Red Hat Quay Builders for OpenShift Container Platform by using the following procedure.

Prerequisites

- You have configured the OpenShift Container Platform TLS component.

Procedure

1. Enter the following command to create a project where Builds will be run, for example, **builder**:

```
$ oc new-project builder
```

2. Create a new **ServiceAccount** in the the **builder** namespace by entering the following command:

```
$ oc create sa -n builder quay-builder
```

3. Enter the following command to grant a user the **edit** role within the **builder** namespace:

```
$ oc policy add-role-to-user -n builder edit system:serviceaccount:builder:quay-builder
```

4. Enter the following command to retrieve a token associated with the **quay-builder** service account in the **builder** namespace. This token is used to authenticate and interact with the OpenShift Container Platform cluster's API server.

```
$ oc sa get-token -n builder quay-builder
```

5. Identify the URL for the OpenShift Container Platform cluster's API server. This can be found in the OpenShift Container Platform Web Console.

6. Identify a worker node label to be used when schedule Build **jobs**. Because Build pods need to run on bare metal worker nodes, typically these are identified with specific labels.
Check with your cluster administrator to determine exactly which node label should be used.

7. Optional. If the cluster is using a self-signed certificate, you must get the Kube API Server's certificate authority (CA) to add to Red Hat Quay's extra certificates.

   a. Enter the following command to obtain the name of the secret containing the CA:

```
$ oc get sa openshift-apiserver-sa --namespace=openshift-apiserver -o json | jq
'.secrets[] | select(.name | contains("openshift-apiserver-sa-token"))'.name
```

   b. Obtain the **ca.crt** key value from the secret in the OpenShift Container Platform Web Console. The value begins with **"-----BEGIN CERTIFICATE-----"`**.

   c. Import the CA to Red Hat Quay. Ensure that the name of this file matches **K8S_API_TLS_CA**.

8. Create the following **SecurityContextConstraints** resource for the **ServiceAccount**:

```
apiVersion: security.openshift.io/v1
kind: SecurityContextConstraints
metadata:
  name: quay-builder
priority: null
readOnlyRootFilesystem: false
requiredDropCapabilities: null
runAsUser:
  type: RunAsAny
seLinuxContext:
  type: RunAsAny
seccompProfiles:
- '*'
supplementalGroups:
  type: RunAsAny
volumes:
- '*'
allowHostDirVolumePlugin: true
allowHostIPC: true
allowHostNetwork: true
allowHostPID: true
allowHostPorts: true
```

```
allowPrivilegeEscalation: true
allowPrivilegedContainer: true
allowedCapabilities:
- '*'
allowedUnsafeSysctls:
- '*'
defaultAddCapabilities: null
fsGroup:
  type: RunAsAny
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: quay-builder-scc
  namespace: builder
rules:
- apiGroups:
  - security.openshift.io
  resourceNames:
  - quay-builder
  resources:
  - securitycontextconstraints
  verbs:
  - use
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: quay-builder-scc
  namespace: builder
subjects:
- kind: ServiceAccount
  name: quay-builder
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: quay-builder-scc
```

11.1.3. Configuring Red Hat Quay Builders

Use the following procedure to enable Red Hat Quay Builders.

Procedure

1. Ensure that your Red Hat Quay **config.yaml** file has Builds enabled, for example:

```
FEATURE_BUILD_SUPPORT: True
```

2. Add the following information to your Red Hat Quay **config.yaml** file, replacing each value with information that is relevant to your specific installation:

```
BUILD_MANAGER:
- ephemeral
- ALLOWED_WORKER_COUNT: 1
  ORCHESTRATOR_PREFIX: buildman/production/
```

```
  ORCHESTRATOR:
    REDIS_HOST: quay-redis-host
    REDIS_PASSWORD: quay-redis-password
    REDIS_SSL: true
    REDIS_SKIP_KEYSPACE_EVENT_SETUP: false
  EXECUTORS:
  - EXECUTOR: kubernetes
    BUILDER_NAMESPACE: builder
    K8S_API_SERVER: api.openshift.somehost.org:6443
    K8S_API_TLS_CA: /conf/stack/extra_ca_certs/build_cluster.crt
    VOLUME_SIZE: 8G
    KUBERNETES_DISTRIBUTION: openshift
    CONTAINER_MEMORY_LIMITS: 5120Mi
    CONTAINER_CPU_LIMITS: 1000m
    CONTAINER_MEMORY_REQUEST: 3968Mi
    CONTAINER_CPU_REQUEST: 500m
    NODE_SELECTOR_LABEL_KEY: beta.kubernetes.io/instance-type
    NODE_SELECTOR_LABEL_VALUE: n1-standard-4
    CONTAINER_RUNTIME: podman
    SERVICE_ACCOUNT_NAME: *****
    SERVICE_ACCOUNT_TOKEN: *****
    QUAY_USERNAME: quay-username
    QUAY_PASSWORD: quay-password
    WORKER_IMAGE: <registry>/quay-quay-builder
    WORKER_TAG: some_tag
    BUILDER_VM_CONTAINER_IMAGE: <registry>/quay-quay-builder-qemu-rhcos:v3.4.0
    SETUP_TIME: 180
    MINIMUM_RETRY_THRESHOLD: 0
    SSH_AUTHORIZED_KEYS:
    - ssh-rsa 12345 someuser@email.com
    - ssh-rsa 67890 someuser2@email.com
```

For more information about each configuration field, see

11.2. OPENSHIFT CONTAINER PLATFORM *ROUTES* LIMITATIONS

The following limitations apply when you are using the Red Hat Quay Operator on OpenShift Container Platform with a managed **route** component:

- Currently, OpenShift Container Platform *Routes* are only able to serve traffic to a single port. Additional steps are required to set up Red Hat Quay Builds.
- Ensure that your **kubectl** or **oc** CLI tool is configured to work with the cluster where the Red Hat Quay Operator is installed and that your **QuayRegistry** exists; the **QuayRegistry** does not have to be on the same bare metal cluster where *Builders* run.
- Ensure that HTTP/2 ingress is enabled on the OpenShift cluster by following these steps.
- The Red Hat Quay Operator creates a **Route** resource that directs gRPC traffic to the Build manager server running inside of the existing **Quay** pod, or pods. If you want to use a custom hostname, or a subdomain like **<builder-registry.example.com>**, ensure that you create a CNAME record with your DNS provider that points to the **status.ingress[0].host** of the create **Route** resource. For example:

```
$ kubectl get -n <namespace> route <quayregistry-name>-quay-builder -o jsonpath=
{.status.ingress[0].host}
```

- Using the OpenShift Container Platform UI or CLI, update the **Secret** referenced by **spec.configBundleSecret** of the **QuayRegistry** with the Build cluster CA certificate. Name the key **extra_ca_cert_build_cluster.cert**. Update the **config.yaml** file entry with the correct values referenced in the Builder configuration that you created when you configured Red Hat Quay Builders, and add the **BUILDMAN_HOSTNAME** CONFIGURATION FIELD:

```
BUILDMAN_HOSTNAME: <build-manager-hostname> 1
BUILD_MANAGER:
- ephemeral
- ALLOWED_WORKER_COUNT: 1
  ORCHESTRATOR_PREFIX: buildman/production/
  JOB_REGISTRATION_TIMEOUT: 600
  ORCHESTRATOR:
    REDIS_HOST: <quay_redis_host
    REDIS_PASSWORD: <quay_redis_password>
    REDIS_SSL: true
    REDIS_SKIP_KEYSPACE_EVENT_SETUP: false
  EXECUTORS:
  - EXECUTOR: kubernetes
    BUILDER_NAMESPACE: builder
    ...
```

1 The externally accessible server hostname which the build jobs use to communicate back to the Build manager. Default is the same as **SERVER_HOSTNAME**. For OpenShift **Route**, it is either **status.ingress[0].host** or the CNAME entry if using a custom hostname. **BUILDMAN_HOSTNAME** must include the port number, for example, **somehost:443** for an OpenShift Container Platform Route, as the gRPC client used to communicate with the build manager does not infer any port if omitted.

11.3. TROUBLESHOOTING BUILDS

The *Builder* instances started by the Build manager are ephemeral. This means that they will either get shut down by Red Hat Quay on timeouts or failure, or garbage collected by the control plane (EC2/K8s). In order to obtain the Build logs, you must do so while the Builds are running.

11.3.1. DEBUG config flag

The **DEBUG** flag can be set to **true** in order to prevent the Builder instances from getting cleaned up after completion or failure. For example:

```
EXECUTORS:
  - EXECUTOR: ec2
    DEBUG: true
    ...
  - EXECUTOR: kubernetes
    DEBUG: true
    ...
```

When set to **true**, the debug feature prevents the Build nodes from shutting down after the **quay-builder** service is done or fails. It also prevents the Build manager from cleaning up the instances by terminating EC2 instances or deleting Kubernetes jobs. This allows debugging Builder node issues.

Debugging should not be set in a production cycle. The lifetime service still exists; for example, the instance still shuts down after approximately two hours. When this happens, EC2 instances are terminated, and Kubernetes jobs are completed.

Enabling debug also affects the **ALLOWED_WORKER_COUNT**, because the unterminated instances and jobs still count toward the total number of running workers. As a result, the existing Builder workers must be manually deleted if **ALLOWED_WORKER_COUNT** is reached to be able to schedule new Builds.

Setting DEBUG will also affect **ALLOWED_WORKER_COUNT**, as the unterminated instances/jobs will still count towards the total number of running workers. This means the existing builder workers will need to manually be deleted if **ALLOWED_WORKER_COUNT** is reached to be able to schedule new Builds.

11.3.2. Troubleshooting OpenShift Container Platform and Kubernetes Builds

Use the following procedure to troubleshooting OpenShift Container Platform Kubernetes Builds.

Procedure

1. Create a port forwarding tunnel between your local machine and a pod running with either an OpenShift Container Platform cluster or a Kubernetes cluster by entering the following command:

```
$ oc port-forward <builder_pod> 9999:2222
```

2. Establish an SSH connection to the remote host using a specified SSH key and port, for example:

```
$ ssh -i /path/to/ssh/key/set/in/ssh_authorized_keys -p 9999 core@localhost
```

3. Obtain the **quay-builder** service logs by entering the following commands:

```
$ systemctl status quay-builder
```

```
$ journalctl -f -u quay-builder
```

11.4. SETTING UP GITHUB BUILDS

If your organization plans to have Builds be conducted by pushes to Github, or Github Enterprise, continue with *Creating an OAuth application in GitHub* .

CHAPTER 12. BUILDING CONTAINER IMAGES

Building container images involves creating a blueprint for a containerized application. Blueprints rely on base images from other public repositories that define how the application should be installed and configured.

Red Hat Quay supports the ability to build Docker and Podman container images. This functionality is valuable for developers and organizations who rely on container and container orchestration.

12.1. BUILD CONTEXTS

When building an image with Docker or Podman, a directory is specified to become the *build context*. This is true for both manual Builds and Build triggers, because the Build that is created by Red Hat Quay is not different than running **docker build** or **podman build** on your local machine.

Red Hat Quay Build contexts are always specified in the *subdirectory* from the Build setup, and fallback to the root of the Build source if a directory is not specified.

When a build is triggered, Red Hat Quay Build workers clone the Git repository to the worker machine, and then enter the Build context before conducting a Build.

For Builds based on **.tar** archives, Build workers extract the archive and enter the Build context. For example:

Extracted Build archive

```
example
├── .git
├── Dockerfile
├── file
└── subdir
    └── Dockerfile
```

Imagine that the *Extracted Build archive* is the directory structure got a Github repository called **example.** If no subdirectory is specified in the Build trigger setup, or when manually starting the Build, the Build operates in the example directory.

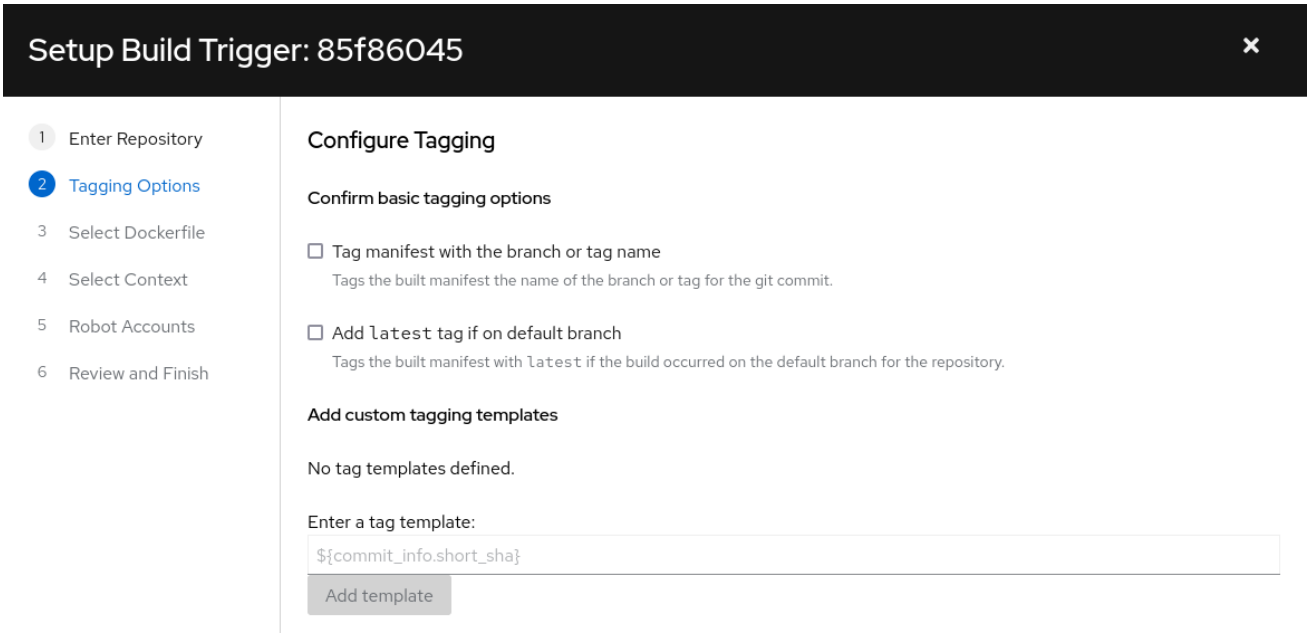
If a subdirectory is specified in the Build trigger setup, for example, **subdir**, only the Dockerfile within it is visible to the Build. This means that you cannot use the **ADD** command in the Dockerfile to add **file**, because it is outside of the Build context.

Unlike Docker Hub, the Dockerfile is part of the Build context on Red Hat Quay. As a result, it must not appear in the **.dockerignore** file.

12.2. TAG NAMING FOR BUILD TRIGGERS

Custom tags are available for use in Red Hat Quay.

One option is to include any string of characters assigned as a tag for each built image. Alternatively, you can use the following tag templates on the **Configure Tagging** section of the build trigger to tag images with information from each commit:



- **${commit}**: Full SHA of the issued commit
- **${parsed_ref.branch}**: Branch information (if available)
- **${parsed_ref.tag}**: Tag information (if available)
- **${parsed_ref.remote}**: The remote name
- **${commit_info.date}**: Date when the commit was issued
- **${commit_info.author.username}**: Username of the author of the commit
- **${commit_info.short_sha}**: First 7 characters of the commit SHA
- **${committer.properties.username}**: Username of the committer

This list is not complete, but does contain the most useful options for tagging purposes. You can find the complete tag template schema on this page.

For more information, see Set up custom tag templates in build triggers for Red Hat Quay and Quay.io

12.3. SKIPPING A SOURCE CONTROL-TRIGGERED BUILD

To specify that a commit should be ignored by the Red Hat Quay build system, add the text **[skip build]** or **[build skip]** anywhere in your commit message.

12.4. VIEWING AND MANAGING BUILDS

Repository Builds can be viewed and managed on the Red Hat Quay UI.

Procedure

1. Navigate to a Red Hat Quay repository using the UI.
2. In the navigation pane, select **Builds**.

12.5. CREATING A NEW BUILD

Red Hat Quay can create new Builds so long as **FEATURE_BUILD_SUPPORT** is set to to **true** in their **config.yaml** file.

Prerequisites

- You have navigated to the **Builds** page of your repository.
- **FEATURE_BUILD_SUPPORT** is set to to **true** in your **config.yaml** file.

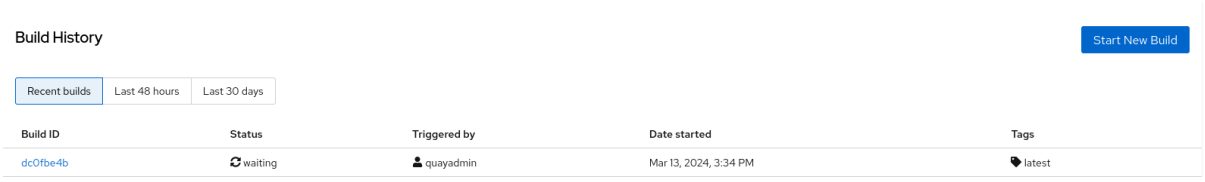
Procedure

1. On the **Builds** page, click **Start New Build**.
2. When prompted, click **Upload Dockerfile** to upload a Dockerfile or an archive that contains a Dockerfile at the root directory.
3. Click **Start Build**.

NOTE

- Currently, users cannot specify the Docker build context when manually starting a build.
- Currently, BitBucket is unsupported on the Red Hat Quay v2 UI.

4. You are redirected to the Build, which can be viewed in real-time. Wait for the Dockerfile Build to be completed and pushed.
5. Optional. you can click **Download Logs** to download the logs, or **Copy Logs** to copy the logs.
6. Click the back button to return to the **Repository Builds** page, where you can view the Build History.

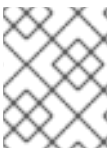


12.6. BUILD TRIGGERS

Build triggers invoke builds whenever the triggered condition is met, for example, a source control push, creating a webhook call, and so on.

12.6.1. Creating a Build trigger

Use the following procedure to create a Build trigger using a custom Git repository.



NOTE

The following procedure assumes that you have not included Github credentials in your **config.yaml** file.

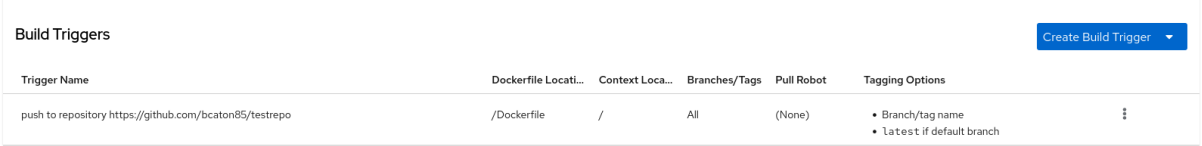
Prerequisites

- You have navigated to the **Builds** page of your repository.

Procedure

1. On the **Builds** page, click **Create Build Trigger**.
2. Select the desired platform, for example, Github, BitBucket, Gitlab, or use a custom Git repository. For this example, we are using a custom Git repository from Github.
3. Enter a custom Git repository name, for example, **git@github.com:<username>/<repo>.git**. Then, click **Next**.
4. When prompted, configure the tagging options by selecting one of, or both of, the following options:
   - **Tag manifest with the branch or tag name** When selecting this option, the built manifest the name of the branch or tag for the git commit are tagged.
   - **Add latest tag if on default branch** When selecting this option, the built manifest with latest if the build occurred on the default branch for the repository are tagged.
     Optionally, you can add a custom tagging template. There are multiple tag templates that you can enter here, including using short SHA IDs, timestamps, author names, committer, and branch names from the commit as tags. For more information, see "Tag naming for Build triggers".

     After you have configured tagging, click **Next**.
5. When prompted, select the location of the Dockerfile to be built when the trigger is invoked. If the Dockerfile is located at the root of the git repository and named Dockerfile, enter **/Dockerfile** as the Dockerfile path. Then, click **Next**.
6. When prompted, select the context for the Docker build. If the Dockerfile is located at the root of the Git repository, enter / as the build context directory. Then, click **Next**.
7. Optional. Choose an optional robot account. This allows you to pull a private base image during the build process. If you know that a private base image is not used, you can skip this step.
8. Click **Next**. Check for any verification warnings. If necessary, fix the issues before clicking **Finish**.
9. You are alerted that the trigger has been successfully activated. Note that using this trigger requires the following actions:
   - You must give the following public key read access to the git repository.
   - You must set your repository to **POST** to the following URL to trigger a build.
     Save the SSH Public Key, then click **Return to <organization_name>/<repository_name>**. You are redirected to the **Builds** page of your repository.
10. On the **Builds** page, you now have a Build trigger. For example:



12.6.2. Manually triggering a Build

Builds can be triggered manually by using the following procedure.

Procedure

1. On the **Builds** page, **Start new build**.
2. When prompted, select **Invoke Build Trigger**.
3. Click **Run Trigger Now** to manually start the process.
   After the build starts, you can see the Build ID on the **Repository Builds** page.

12.7. SETTING UP A CUSTOM GIT TRIGGER

A *custom Git trigger* is a generic way for any Git server to act as a Build trigger. It relies solely on SSH keys and webhook endpoints. Everything else is left for the user to implement.

12.7.1. Creating a trigger

Creating a custom Git trigger is similar to the creation of any other trigger, with the exception of the following:

- Red Hat Quay cannot automatically detect the proper Robot Account to use with the trigger. This must be done manually during the creation process.
- There are extra steps after the creation of the trigger that must be done. These steps are detailed in the following sections.

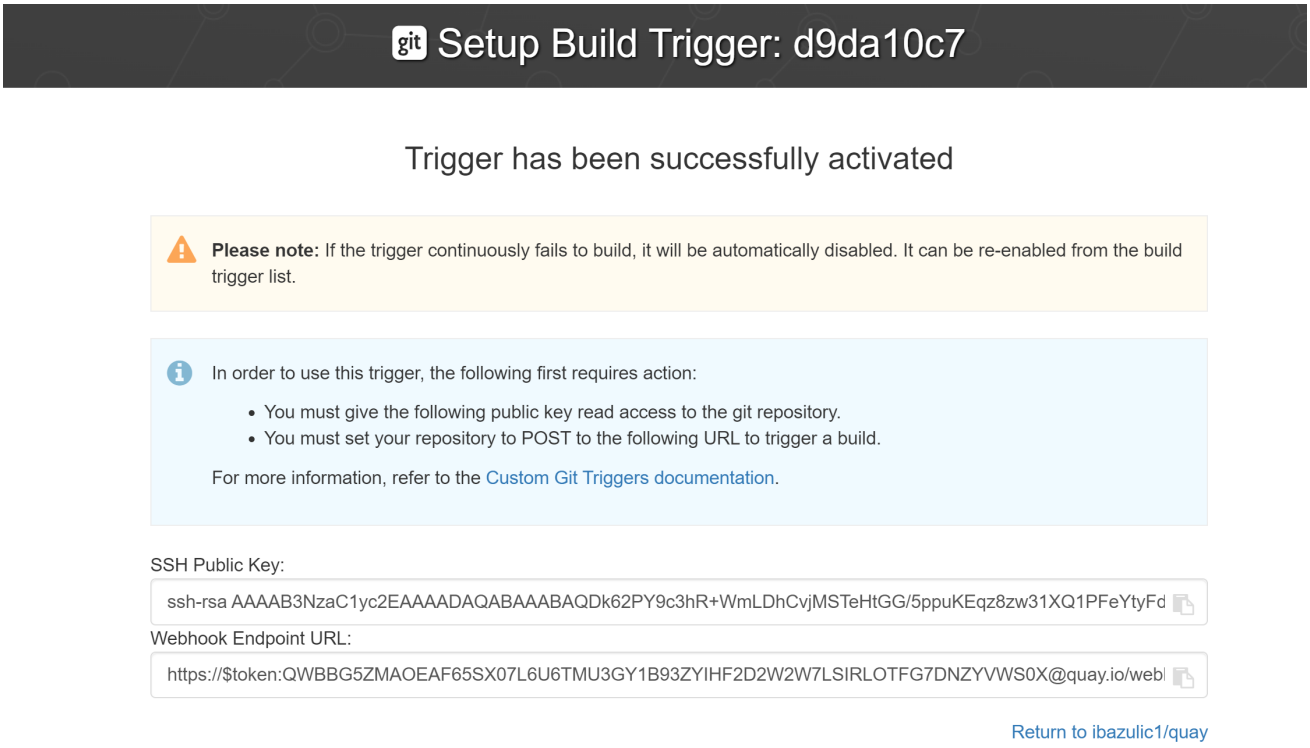
12.7.2. Custom trigger creation setup

When creating a custom Git trigger, two additional steps are required:

1. You must provide read access to the SSH public key that is generated when creating the trigger.
2. You must setup a webhook that POSTs to the Red Hat Quay endpoint to trigger the build.

The key and the URL are available by selecting **View Credentials** from the **Settings**, or *gear* icon.

View and modify tags from your repository



12.7.2.1. SSH public key access

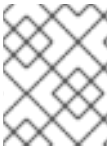
Depending on the Git server configuration, there are multiple ways to install the SSH public key that Red Hat Quay generates for a custom Git trigger.

For example, Git documentation describes a small server setup in which adding the key to **$HOME/.ssh/authorize_keys** would provide access for Builders to clone the repository. For any git repository management software that is not officially supported, there is usually a location to input the key often labeled as **Deploy Keys**.

12.7.2.2. Webhook

To automatically trigger a build, one must **POST** a **.json** payload to the webhook URL using the following format.

This can be accomplished in various ways depending on the server setup, but for most cases can be done with a **post-receive** Git Hook.



NOTE

This request requires a **Content-Type** header containing **application/json** in order to be valid.

Example webhook

```
{
  "commit": "1c002dd",                          // required
  "ref": "refs/heads/master",                   // required
  "default_branch": "master",                   // required
  "commit_info": {                              // optional
    "url": "gitsoftware.com/repository/commits/1234567", // required
    "message": "initial commit",                // required
```

```
    "date": "timestamp",                          // required
    "author": {                                   // optional
      "username": "user",                         // required
      "avatar_url": "gravatar.com/user.png",      // required
      "url": "gitsoftware.com/users/user"         // required
    },
    "committer": {                                // optional
      "username": "user",                         // required
      "avatar_url": "gravatar.com/user.png",      // required
      "url": "gitsoftware.com/users/user"         // required
    }
  }
}
```

CHAPTER 13. CREATING AN OAUTH APPLICATION IN GITHUB

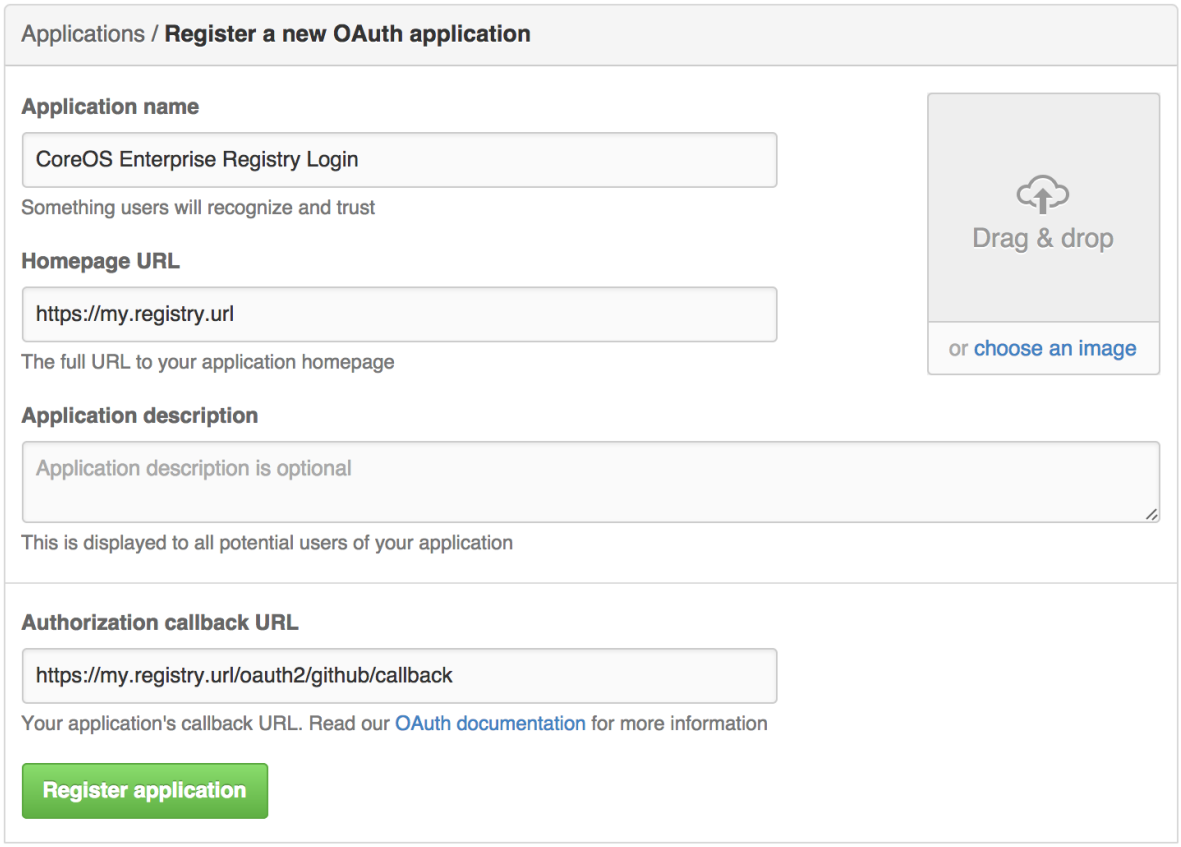
You can authorize your Red Hat Quay registry to access a GitHub account and its repositories by registering it as a GitHub OAuth application.

13.1. CREATE NEW GITHUB APPLICATION

Use the following procedure to create an OAuth application in Github.

Procedure

1. Log into Github Enterprise.
2. In the navigation pane, select your username → **Your organizations**.
3. In the navigation pane, select **Applications**.
4. Click Register New Application. The **Register a new OAuth application** configuration screen is displayed, for example:



5. Enter a name for the application in the **Application name** textbox.
6. In the **Homepage URL** textbox, enter your Red Hat Quay URL.

NOTE

If you are using public GitHub, the Homepage URL entered must be accessible by your users. It can still be an internal URL.

7. In the **Authorization callback URL**, enter https://<RED_HAT_QUAY_URL>/oauth2/github/callback.

8. Click **Register application** to save your settings.

9. When the new application's summary is shown, record the Client ID and the Client Secret shown for the new application.

CHAPTER 14. RED HAT QUAY QUOTA MANAGEMENT AND ENFORCEMENT OVERVIEW

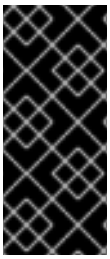
With Red Hat Quay, users have the ability to report storage consumption and to contain registry growth by establishing configured storage quota limits. On-premise Red Hat Quay users are now equipped with the following capabilities to manage the capacity limits of their environment:

- **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 ARCHITECTURE

With the quota management feature enabled, individual blob sizes are summed at the repository and namespace level. For example, if two tags in the same repository reference the same blob, the size of that blob is only counted once towards the repository total. Additionally, manifest list totals are counted toward the repository total.



IMPORTANT

Because manifest list totals are counted toward the repository total, the total quota consumed when upgrading from a previous version of Red Hat Quay might be reportedly differently in Red Hat Quay 3.9. In some cases, the new total might go over a repository's previously-set limit. Red Hat Quay administrators might have to adjust the allotted quota of a repository to account for these changes.

The quota management feature works by calculating the size of existing repositories and namespace with a backfill worker, and then adding or subtracting from the total for every image that is pushed or garbage collected afterwords. Additionally, the subtraction from the total happens when the manifest is garbage collected.

NOTE

Because subtraction occurs from the total when the manifest is garbage collected, there is a delay in the size calculation until it is able to be garbage collected. For more information about garbage collection, see Red Hat Quay garbage collection .

The following database tables hold the quota repository size, quota namespace size, and quota registry size, in bytes, of a Red Hat Quay repository within an organization:

- **QuotaRepositorySize**
- **QuotaNameSpaceSize**
- **QuotaRegistrySize**

The organization size is calculated by the backfill worker to ensure that it is not duplicated. When an image push is initialized, the user's organization storage is validated to check if it is beyond the configured quota limits. If an image push exceeds defined quota limitations, a soft or hard check occurs:

- For a soft check, users are notified.
- For a hard check, the push is stopped.

If storage consumption is within configured quota limits, the push is allowed to proceed.

Image manifest deletion follows a similar flow, whereby the links between associated image tags and the manifest are deleted. Additionally, after the image manifest is deleted, the repository size is recalculated and updated in the **QuotaRepositorySize**, **QuotaNameSpaceSize**, and **QuotaRegistrySize** tables.

14.2. 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

14.3. QUOTA MANAGEMENT CONFIGURATION FIELDS

Table 14.2. Quota management configuration

Field	Type	Description
FEATURE_QUOTA_MANAGEMENT	Boolean	Enables configuration, caching, and validation for quota management feature. **Default:** `False`
DEFAULT_SYSTEM_REJECT_QUOTA_BYTES	String	Enables system default quota reject byte allowance for all organizations. By default, no limit is set.

Field	Type	Description
QUOTA_BACKFILL	Boolean	Enables the quota backfill worker to calculate the size of pre-existing blobs. Default: **True**
QUOTA_TOTAL_DELAY_SECONDS	String	The time delay for starting the quota backfill. Rolling deployments can cause incorrect totals. This field **must** be set to a time longer than it takes for the rolling deployment to complete. Default: **1800**
PERMANENTLY_DELETE_TAGS	Boolean	Enables functionality related to the removal of tags from the time machine window. Default: **False**
RESET_CHILD_MANIFEST_EXPIRATION	Boolean	Resets the expirations of temporary tags targeting the child manifests. With this feature set to **True**, child manifests are immediately garbage collected. Default: **False**

14.3.1. Example quota management configuration

The following YAML is the suggested configuration when enabling quota management.

Quota management YAML configuration

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

14.4. 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

```
[]
```

14.4.1. 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"
```

14.4.2. 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
  }
]
```

14.4.3. 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
}
```

14.4.4. Pushing images

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

14.4.4.1. 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
```

14.4.4.2. 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
    }
  ]
}
```

14.4.4.3. 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
    }
  ]
}
```


3. 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
  }
}
```


14.4.5. 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.

14.4.5.1. 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
```

14.4.5.2. 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
  }
]
```

14.4.5.3. 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
```

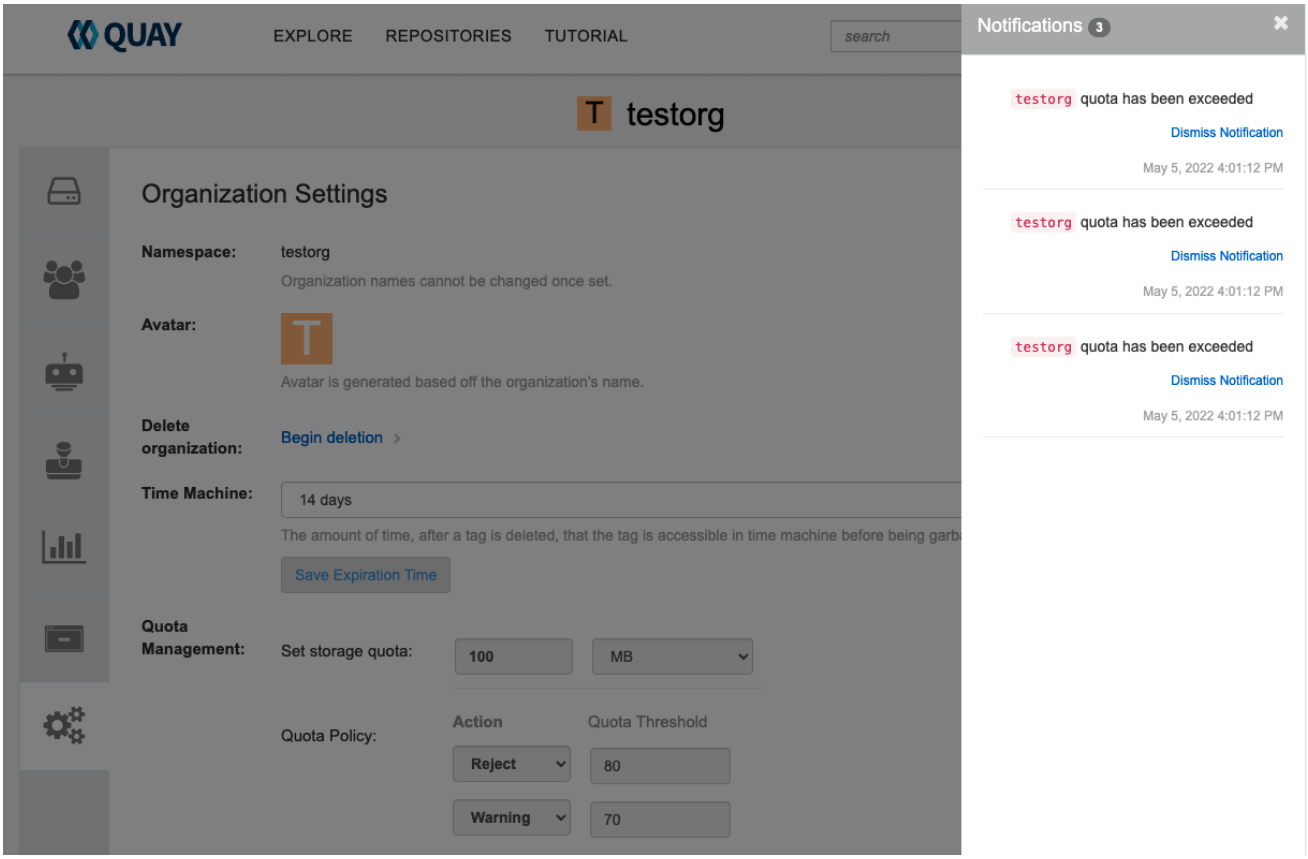
Sample output when quota exceeded

```
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
```

14.4.5.4. Notifications for limits exceeded

When limits are exceeded, a notification appears:

Quota notifications


QUAY
EXPLORE
REPOSITORIES
TUTORIAL
search
T testorg
Organization Settings
Namespace:
testorg
Organization names cannot be changed once set.
Avatar:
T
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 garb
Save Expiration Time
Quota Management:
Set storage quota:
100
MB
Quota Policy:
Action
Quota Threshold
Reject
80
Warning
70
Notifications 3
testorg quota has been exceeded
Dismiss Notification
May 5, 2022 4:01:12 PM
testorg quota has been exceeded
Dismiss Notification
May 5, 2022 4:01:12 PM
testorg quota has been exceeded
Dismiss Notification
May 5, 2022 4:01:12 PM

CHAPTER 15. RED HAT QUAY AS A PROXY CACHE FOR UPSTREAM REGISTRIES

With the growing popularity of container development, customers increasingly rely on container images from upstream registries like Docker or Google Cloud Platform to get services up and running. Today, registries have rate limitations and throttling on the number of times users can pull from these registries.

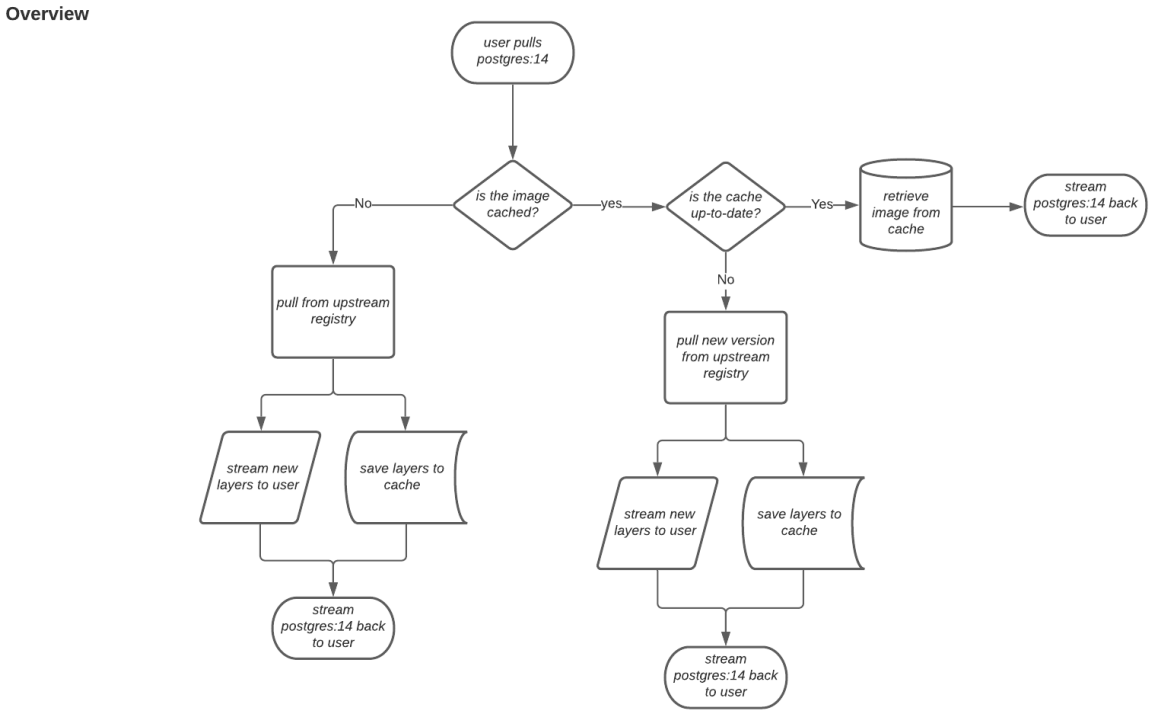
With this feature, Red Hat Quay will act as a proxy cache to circumvent pull-rate limitations from upstream registries. Adding a cache feature also accelerates pull performance, because images are pulled from the cache rather than upstream dependencies. Cached images are only updated when the upstream image digest differs from the cached image, reducing rate limitations and potential throttling.

With Red Hat Quay cache proxy, the following features are available:

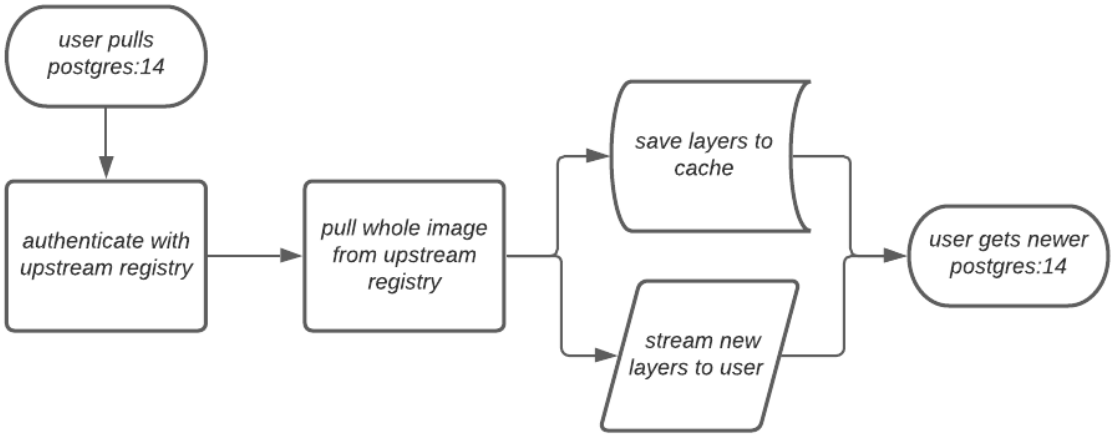
- Specific organizations can be defined as a cache for upstream registries.
- Configuration of a Quay organization that acts as a cache for a specific upstream registry. This repository can be defined by using the Quay UI, and offers the following configurations:
  - Upstream registry credentials for private repositories or increased rate limiting.
  - Expiration timer to avoid surpassing cache organization size.
- Global on/off configurable via the configuration application.
- Caching of entire upstream registries or just a single namespace, for example, all of **docker.io** or just **docker.io/library**.
- Logging of all cache pulls.
- Cached images scannability by Clair.

15.1. PROXY CACHE ARCHITECTURE

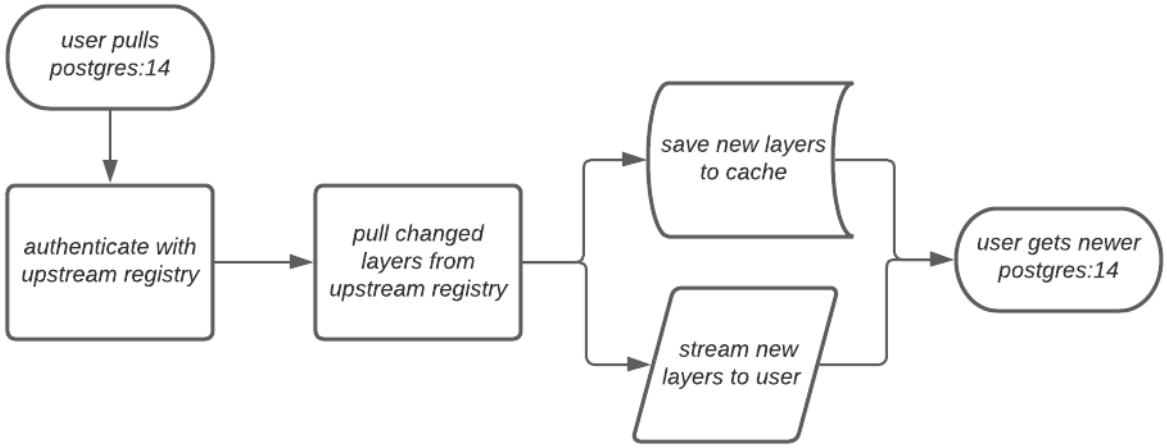
The following image shows the expected design flow and architecture of the proxy cache feature.



When a user pulls an image, for example, **postgres:14**, from an upstream repository on Red Hat Quay, the repository checks to see if an image is present. If the image does not exist, a fresh pull is initiated. After being pulled, the image layers are saved to cache and server to the user in parallel. The following image depicts an architectural overview of this scenario:

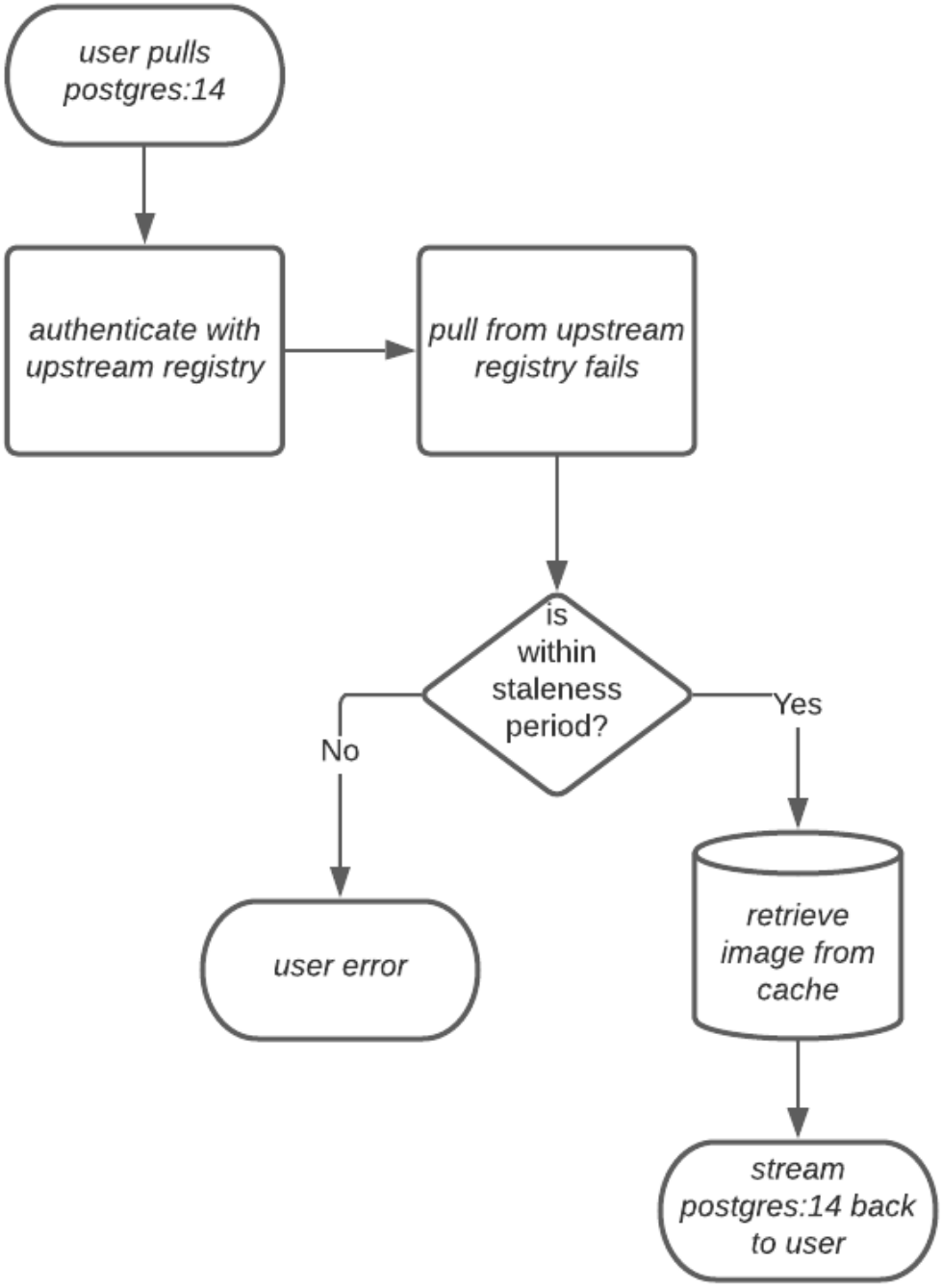


If the image in the cache exists, users can rely on Quay's cache to stay up-to-date with the upstream source so that newer images from the cache are automatically pulled. This happens when tags of the original image have been overwritten in the upstream registry. The following image depicts an architectural overview of what happens when the upstream image and cached version of the image are different:



If the upstream image and cached version are the same, no layers are pulled and the cached image is delivered to the user.

In some cases, users initiate pulls when the upstream registry is down. If this happens with the configured staleness period, the image stored in cache is delivered. If the pull happens after the configured staleness period, the error is propagated to the user. The following image depicts an architectural overview when a pull happens after the configured staleness period:



Quay administrators can leverage the configurable size limit of an organization to limit cache size so that backend storage consumption remains predictable. This is achieved by discarding images from the cache according to the frequency in which an image is used. The following image depicts an architectural overview of this scenario:

15.2. PROXY CACHE LIMITATIONS

Proxy caching with Red Hat Quay has the following limitations:

- Your proxy cache must have a size limit of greater than, or equal to, the image you want to cache. For example, if your proxy cache organization has a maximum size of 500 MB, and the image a user wants to pull is 700 MB, the image will be cached and will overflow beyond the configured limit.
- Cached images must have the same properties that images on a Quay repository must have.

15.3. USING RED HAT QUAY TO PROXY A REMOTE REGISTRY

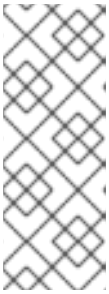
The following procedure describes how you can use Red Hat Quay to proxy a remote registry. This procedure is set up to proxy quay.io, which allows users to use **podman** to pull any public image from any namespace on quay.io.

Prerequisites

- **FEATURE_PROXY_CACHE** in your config.yaml is set to **true**.
- Assigned the **Member** team role. For more information about team roles, see Users and organizations in Red Hat Quay.

Procedure

1. In your Quay organization on the UI, for example, **cache-quayio**, click **Organization Settings** on the left hand pane.
2. Optional: Click **Add Storage Quota** to configure quota management for your organization. For more information about quota management, see Quota Management.



NOTE

In some cases, pulling images with Podman might return the following error when quota limit is reached during a pull: **unable to pull image: Error parsing image configuration: Error fetching blob: invalid status code from registry 403 (Forbidden)**. Error **403** is inaccurate, and occurs because Podman hides the correct API error: **Quota has been exceeded on namespace**. This known issue will be fixed in a future Podman update.

3. In **Remote Registry** enter the name of the remote registry to be cached, for example, **quay.io**, and click **Save**.

NOTE

By adding a namespace to the **Remote Registry**, for example, **quay.io/<namespace>**, users in your organization will only be able to proxy from that namespace.

4. Optional: Add a **Remote Registry Username** and **Remote Registry Password**.

NOTE

If you do not set a **Remote Registry Username** and **Remote Registry Password**, you cannot add one without removing the proxy cache and creating a new registry.

5. Optional: Set a time in the **Expiration** field.

NOTE

- The default tag **Expiration** field for cached images in a proxy organization is set to 86400 seconds. In the proxy organization, the tag expiration is refreshed to the value set in the UI's **Expiration** field every time the tag is pulled. This feature is different than Quay's default individual tag expiration feature. In a proxy organization, it is possible to override the individual tag feature. When this happens, the individual tag's expiration is reset according to the **Expiration** field of the proxy organization.
- Expired images will disappear after the allotted time, but are still stored in 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.

6. Click **Save**.
7. On the CLI, pull a public image from the registry, for example, quay.io, acting as a proxy cache:

```
$ podman pull <registry_url>/<organization_name>/<quayio_namespace>/<image_name>
```

IMPORTANT

If your organization is set up to pull from a single namespace in the remote registry, the remote registry namespace must be omitted from the URL. For example, **podman pull <registry_url>/<organization_name>/<image_name>**.

15.3.1. Leveraging storage quota limits in proxy organizations

With Red Hat Quay 3.8, the proxy cache feature has been enhanced with an auto-pruning feature for tagged images. The auto-pruning of image tags is only available when a proxied namespace has quota limitations configured. Currently, if an image size is greater than quota for an organization, the image is skipped from being uploaded until an administrator creates the necessary space. Now, when an image is pushed that exceeds the allotted space, the auto-pruning enhancement marks the least recently used tags for deletion. As a result, the new image tag is stored, while the least used image tag is marked for deletion.

IMPORTANT

- As part of the auto-pruning feature, the tags that are marked for deletion are eventually garbage collected by the garbage collector (gc) worker process. As a result, the quota size restriction is not fully enforced during this period.
- Currently, the namespace quota size computation does not take into account the size for manifest child. This is a known issue and will be fixed in a future version of Red Hat Quay.

15.3.1.1. Testing the storage quota limits feature in proxy organizations

Use the following procedure to test the auto-pruning feature of an organization with proxy cache and storage quota limitations enabled.

Prerequisites

- Your organization is configured to serve as a proxy organization. The following example proxies from quay.io.
- **FEATURE_PROXY_CACHE** is set to **true** in your **config.yaml** file.
- **FEATURE_QUOTA_MANAGEMENT** is set to **true** in your **config.yaml** file.
- Your organization is configured with a quota limit, for example, **150 MB**.

Procedure

1. Pull an image to your repository from your proxy organization, for example:

```
$ podman pull quay-server.example.com/proxytest/projectquay/quay:3.7.9
```

2. Depending on the space left in your repository, you might need to pull additional images from your proxy organization, for example:

```
$ podman pull quay-server.example.com/proxytest/projectquay/quay:3.6.2
```

3. In the Red Hat Quay registry UI, click the name of your repository.
   - Click **Tags** in the navigation pane and ensure that **quay:3.7.9** and **quay:3.6.2** are tagged.
4. Pull the last image that will result in your repository exceeding the allotted quota, for example:

```
$ podman pull quay-server.example.com/proxytest/projectquay/quay:3.5.1
```

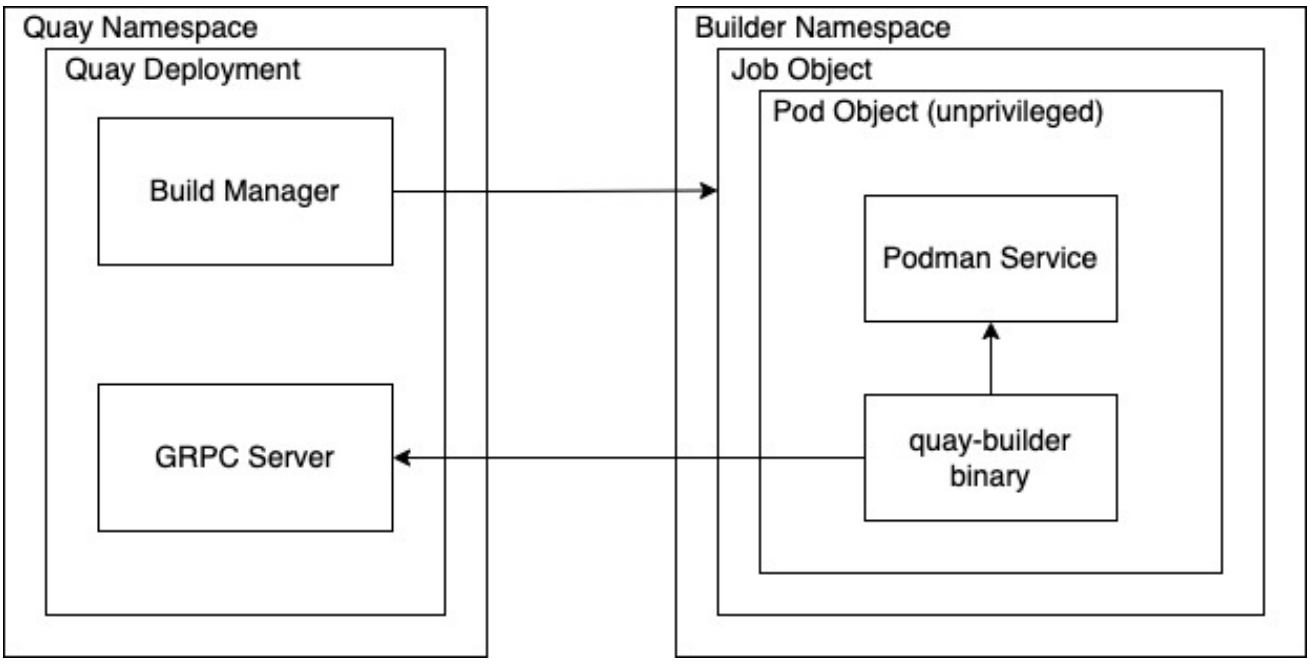
5. Refresh the **Tags** page of your Red Hat Quay registry. The first image that you pushed, for example, **quay:3.7.9** should have been auto-pruned. The **Tags** page should now show **quay:3.6.2** and **quay:3.5.1**.

CHAPTER 16. RED HAT QUAY BUILD ENHANCEMENTS

Red Hat Quay builds can be run on virtualized platforms. Backwards compatibility to run previous build configurations are also available.

16.1. RED HAT QUAY ENHANCED BUILD ARCHITECTURE

The following image shows the expected design flow and architecture of the enhanced build features:



With this enhancement, the build manager first creates the **Job Object**. Then, the **Job Object** then creates a pod using the **quay-builder-image**. The **quay-builder-image** will contain the **quay-builder binary** and the **Podman** service. The created pod runs as **unprivileged**. The **quay-builder binary** then builds the image while communicating status and retrieving build information from the Build Manager.

16.2. RED HAT QUAY BUILD LIMITATIONS

Running builds in Red Hat Quay in an unprivileged context might cause some commands that were working under the previous build strategy to fail. Attempts to change the build strategy could potentially cause performance issues and reliability with the build.

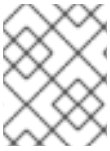
Running builds directly in a container does not have the same isolation as using virtual machines. Changing the build environment might also caused builds that were previously working to fail.

16.3. CREATING A RED HAT QUAY BUILDERS ENVIRONMENT WITH OPENSHIFT CONTAINER PLATFORM

The procedures in this section explain how to create a Red Hat Quay virtual builders environment with OpenShift Container Platform.

16.3.1. OpenShift Container Platform TLS component

The **tls** component allows you to control TLS configuration.



NOTE

Red Hat Quay 3 does not support builders when the TLS component is managed by the Operator.

If you set **tls** to **unmanaged**, you supply your own **ssl.cert** and **ssl.key** files. In this instance, if you want your cluster to support builders, you must add both the Quay route and the builder route name to the SAN list in the cert, or use a wildcard.

To add the builder route, use the following format:

```
[quayregistry-cr-name]-quay-builder-[ocp-namespace].[ocp-domain-name]:443
```

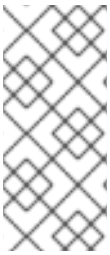
16.3.2. Using OpenShift Container Platform for Red Hat Quay builders

Builders require SSL/TLS certificates. For more information about SSL/TLS certificates, see Using SSL/TLS certificates.

If you are using Amazon Web Service (AWS) S3 storage, you must modify your storage bucket in the AWS console, prior to running builders. See "Modifying your AWS S3 storage bucket" in the following section for the required parameters.

16.3.2.1. Preparing OpenShift Container Platform for virtual builders

Use the following procedure to prepare OpenShift Container Platform for Red Hat Quay virtual builders.



NOTE

- This procedure assumes you already have a cluster provisioned and a Quay Operator running.
- This procedure is for setting up a virtual namespace on OpenShift Container Platform.

Procedure

1. Log in to your Red Hat Quay cluster using a cluster administrator account.
2. Create a new project where your virtual builders will be run, for example, **virtual-builders**, by running the following command:

```
$ oc new-project virtual-builders
```

3. Create a **ServiceAccount** in the project that will be used to run builds by entering the following command:

```
$ oc create sa -n virtual-builders quay-builder
```

4. Provide the created service account with editing permissions so that it can run the build:

```
$ oc adm policy -n virtual-builders add-role-to-user edit system:serviceaccount:virtual-builders:quay-builder
```

5. Grant the Quay builder **anyuid scc** permissions by entering the following command:

```
$ oc adm policy -n virtual-builders add-scc-to-user anyuid -z quay-builder
```

NOTE

This action requires cluster admin privileges. This is required because builders must run as the Podman user for unprivileged or rootless builds to work.

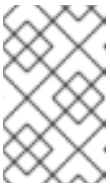
6. Obtain the token for the Quay builder service account.

   a. If using OpenShift Container Platform 4.10 or an earlier version, enter the following command:

```
oc sa get-token -n virtual-builders quay-builder
```

   b. If using OpenShift Container Platform 4.11 or later, enter the following command:

```
$ oc create token quay-builder -n virtual-builders
```



NOTE

When the token expires you will need to request a new token. Optionally, you can also add a custom expiration. For example, specify **--duration 20160m** to retain the token for two weeks.

Example output

```
eyJhbGciOiJSUzI1NiIsImtpZCI6IldfQUJkaDVmb3ltTHZ0dGZMYjhIWnYxZTQzN2dJVEJxc
DJscldSdEUtYWsifQ...
```

7. Determine the builder route by entering the following command:

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

Example output

```
NAME                          HOST/PORT                                                    PATH
SERVICES                        PORT   TERMINATION     WILDCARD
...
example-registry-quay-builder         example-registry-quay-builder-quay-
enterprise.apps.docs.quayteam.org                example-registry-quay-app            grpc
edge/Redirect   None
...
```

8. Generate a self-signed SSL/TLS certificate with the .crt extension by entering the following command:

```
$ oc extract cm/kube-root-ca.crt -n openshift-apiserver
```

Example output

```
ca.crt
```

9. Rename the **ca.crt** file to **extra_ca_cert_build_cluster.crt** by entering the following command:

```
$ mv ca.crt extra_ca_cert_build_cluster.crt
```

10. Locate the secret for you configuration bundle in the **Console**, and select **Actions → Edit Secret** and add the appropriate builder configuration:

```
FEATURE_USER_INITIALIZE: true
BROWSER_API_CALLS_XHR_ONLY: false
SUPER_USERS:
- <superusername>
FEATURE_USER_CREATION: false
FEATURE_QUOTA_MANAGEMENT: true
FEATURE_BUILD_SUPPORT: True
BUILDMAN_HOSTNAME: <sample_build_route> 1
BUILD_MANAGER:
 - ephemeral
 - ALLOWED_WORKER_COUNT: 1
  ORCHESTRATOR_PREFIX: buildman/production/
  JOB_REGISTRATION_TIMEOUT: 3600 2
  ORCHESTRATOR:
   REDIS_HOST: <sample_redis_hostname> 3
   REDIS_PASSWORD: ""
   REDIS_SSL: false
   REDIS_SKIP_KEYSPACE_EVENT_SETUP: false
  EXECUTORS:
   - EXECUTOR: kubernetesPodman
     NAME: openshift
     BUILDER_NAMESPACE: <sample_builder_namespace> 4
     SETUP_TIME: 180
     MINIMUM_RETRY_THRESHOLD: 0
     BUILDER_CONTAINER_IMAGE: <sample_builder_container_image> 5
     # Kubernetes resource options
     K8S_API_SERVER: <sample_k8s_api_server> 6
     K8S_API_TLS_CA: <sample_crt_file> 7
     VOLUME_SIZE: 8G
     KUBERNETES_DISTRIBUTION: openshift
     CONTAINER_MEMORY_LIMITS: 300m 8
     CONTAINER_CPU_LIMITS: 1G 9
     CONTAINER_MEMORY_REQUEST: 300m 10
     CONTAINER_CPU_REQUEST: 1G 11
     NODE_SELECTOR_LABEL_KEY: ""
     NODE_SELECTOR_LABEL_VALUE: ""
     SERVICE_ACCOUNT_NAME: <sample_service_account_name>
     SERVICE_ACCOUNT_TOKEN: <sample_account_token> 12
```

1. The build route is obtained by running **oc get route -n** with the name of your OpenShift Operator's namespace. A port must be provided at the end of the route, and it should use the following format: **[quayregistry-cr-name]-quay-builder-[ocp-namespace].[ocp-domain-name]:443**.

2. If the **JOB_REGISTRATION_TIMEOUT** parameter is set too low, you might receive the following error: **failed to register job to build manager: rpc error: code = Unauthenticated desc = Invalid build token: Signature has expired**. It is suggested that this parameter be set to at least 240.
3. If your Redis host has a password or SSL/TLS certificates, you must update accordingly.
4. Set to match the name of your virtual builders namespace, for example, **virtual-builders**.
5. For early access, the **BUILDER_CONTAINER_IMAGE** is currently **quay.io/projectquay/quay-builder:3.7.0-rc.2**. Note that this might change during the early access window. If this happens, customers are alerted.
6. The **K8S_API_SERVER** is obtained by running **oc cluster-info**.
7. You must manually create and add your custom CA cert, for example, **K8S_API_TLS_CA: /conf/stack/extra_ca_certs/build_cluster.crt**.
8. Defaults to **5120Mi** if left unspecified.
9. For virtual builds, you must ensure that there are enough resources in your cluster. Defaults to **1000m** if left unspecified.
10. Defaults to **3968Mi** if left unspecified.
11. Defaults to **500m** if left unspecified.
12. Obtained when running **oc create sa**.

Sample configuration

```
FEATURE_USER_INITIALIZE: true
BROWSER_API_CALLS_XHR_ONLY: false
SUPER_USERS:
- quayadmin
FEATURE_USER_CREATION: false
FEATURE_QUOTA_MANAGEMENT: true
FEATURE_BUILD_SUPPORT: True
BUILDMAN_HOSTNAME: example-registry-quay-builder-quay-
enterprise.apps.docs.quayteam.org:443
BUILD_MANAGER:
  - ephemeral
  - ALLOWED_WORKER_COUNT: 1
    ORCHESTRATOR_PREFIX: buildman/production/
    JOB_REGISTRATION_TIMEOUT: 3600
    ORCHESTRATOR:
      REDIS_HOST: example-registry-quay-redis
      REDIS_PASSWORD: ""
      REDIS_SSL: false
      REDIS_SKIP_KEYSPACE_EVENT_SETUP: false
    EXECUTORS:
      - EXECUTOR: kubernetesPodman
        NAME: openshift
        BUILDER_NAMESPACE: virtual-builders
        SETUP_TIME: 180
        MINIMUM_RETRY_THRESHOLD: 0
```

```
    BUILDER_CONTAINER_IMAGE: quay.io/projectquay/quay-builder:3.7.0-rc.2
    # Kubernetes resource options
    K8S_API_SERVER: api.docs.quayteam.org:6443
    K8S_API_TLS_CA: /conf/stack/extra_ca_certs/build_cluster.crt
    VOLUME_SIZE: 8G
    KUBERNETES_DISTRIBUTION: openshift
    CONTAINER_MEMORY_LIMITS: 1G
    CONTAINER_CPU_LIMITS: 1080m
    CONTAINER_MEMORY_REQUEST: 1G
    CONTAINER_CPU_REQUEST: 580m
    NODE_SELECTOR_LABEL_KEY: ""
    NODE_SELECTOR_LABEL_VALUE: ""
    SERVICE_ACCOUNT_NAME: quay-builder
    SERVICE_ACCOUNT_TOKEN:
"eyJhbGciOiJSUzI1NiIsImtpZCI6IldfQUJkaDVmb3ltTHZ0dGZMYjhIWnYxZTQzN2dJVEJxcDJs
cldSdEUtYWsifQ"
```

16.3.2.2. Manually adding SSL/TLS certificates

Due to a known issue with the configuration tool, you must manually add your custom SSL/TLS certificates to properly run builders. Use the following procedure to manually add custom SSL/TLS certificates.

For more information about SSL/TLS certificates, see Using SSL/TLS certificates.

16.3.2.2.1. Creating and signing certificates

Use the following procedure to create and sign an SSL/TLS certificate.

Procedure

- Create a certificate authority and sign a certificate. For more information, see Using SSL/TLS certificates.

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 = example-registry-quay-quay-enterprise.apps.docs.quayteam.org (1)
DNS.2 = example-registry-quay-builder-quay-enterprise.apps.docs.quayteam.org (2)
```

1. An **alt_name** for the URL of your Red Hat Quay registry must be included.
2. An **alt_name** for the **BUILDMAN_HOSTNAME**

Sample commands

-

```
$ openssl genrsa -out rootCA.key 2048
$ openssl req -x509 -new -nodes -key rootCA.key -sha256 -days 1024 -out rootCA.pem
$ openssl genrsa -out ssl.key 2048
$ openssl req -new -key ssl.key -out ssl.csr
$ openssl x509 -req -in ssl.csr -CA rootCA.pem -CAkey rootCA.key -CAcreateserial -out
ssl.cert -days 356 -extensions v3_req -extfile openssl.cnf
```

16.3.2.2.2. Setting TLS to unmanaged

Use the following procedure to set **king:tls** to unmanaged.

Procedure

1. In your Red Hat Quay Registry YAML, set **kind: tls** to **managed: false**:

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

2. On the **Events** page, the change is blocked until you set up the appropriate **config.yaml** file. For example:

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

16.3.2.2.3. Creating temporary secrets

Use the following procedure to create temporary secrets for the CA certificate.

Procedure

1. Create a secret in your default namespace for the CA certificate:

```
$ oc create secret generic -n quay-enterprise temp-crt --from-file
extra_ca_cert_build_cluster.crt
```

2. Create a secret in your default namespace for the **ssl.key** and **ssl.cert** files:

```
$ oc create secret generic -n quay-enterprise quay-config-ssl --from-file ssl.cert --from-file
ssl.key
```

16.3.2.2.4. Copying secret data to the configuration YAML

Use the following procedure to copy secret data to your **config.yaml** file.

Procedure

1. Locate the new secrets in the console UI at **Workloads → Secrets**.

2. For each secret, locate the YAML view:

```
kind: Secret
apiVersion: v1
metadata:
  name: temp-crt
  namespace: quay-enterprise
  uid: a4818adb-8e21-443a-a8db-f334ace9f6d0
  resourceVersion: '9087855'
  creationTimestamp: '2022-03-28T13:05:30Z'
...
data:
  extra_ca_cert_build_cluster.crt: >-
    LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSURNakNDQWhxZ0F3SUJBZ0l....
type: Opaque
```

```
kind: Secret
apiVersion: v1
metadata:
  name: quay-config-ssl
  namespace: quay-enterprise
  uid: 4f5ae352-17d8-4e2d-89a2-143a3280783c
  resourceVersion: '9090567'
  creationTimestamp: '2022-03-28T13:10:34Z'
...
data:
  ssl.cert: >-
    LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUVaakNDQTA2Z0F3SUJBZ0lVT...
  ssl.key: >-
    LS0tLS1CRUdJTiBSU0EgUFJJVkFURSBLRVktLS0tLQpNSUlFcFFJQkFBS0NBUUVBc...
type: Opaque
```

3. Locate the secret for your Red Hat Quay registry configuration bundle in the UI, or through the command line by running a command like the following:

```
$ oc get quayregistries.quay.redhat.com -o jsonpath="{.items[0].spec.configBundleSecret}
{'\n'}"  -n quay-enterprise
```

4. In the OpenShift Container Platform console, select the YAML tab for your configuration bundle secret, and add the data from the two secrets you created:

```
kind: Secret
apiVersion: v1
metadata:
  name: init-config-bundle-secret
  namespace: quay-enterprise
  uid: 4724aca5-bff0-406a-9162-ccb1972a27c1
  resourceVersion: '4383160'
  creationTimestamp: '2022-03-22T12:35:59Z'
...
data:
  config.yaml: >-
    RkVBVFVSRV9VU0VSX0lOSVRJQUxJWkU6IHRydWUKQlJ...
  extra_ca_cert_build_cluster.crt: >-
```

```
LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSURNakNDQWhxZ0F3SUJBZ0ldw....
  ssl.cert: >-
    LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUVaakNDQTA2Z0F3SUJBZ0lVT...
  ssl.key: >-
    LS0tLS1CRUdJTiBSU0EgUFJJVkFURSBLRVktLS0tLQpNSUlFcFFJQkFBS0NBUUVBc...
type: Opaque
```

5. Click **Save**.

6. Enter the following command to see if your pods are restarting:

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

Example output

```
NAME                                             READY   STATUS              RESTARTS   AGE
...
example-registry-quay-app-6786987b99-vgg2v       0/1     ContainerCreating   0          2s
example-registry-quay-app-7975d4889f-q7tvl       1/1     Running             0          5d21h
example-registry-quay-app-7975d4889f-zn8bb       1/1     Running             0          5d21h
example-registry-quay-app-upgrade-lswsn          0/1     Completed           0          6d1h
example-registry-quay-config-editor-77847fc4f5-nsbbv   0/1     ContainerCreating   0          2s
example-registry-quay-config-editor-c6c4d9ccd-2mwg2    1/1     Running             0
5d21h
example-registry-quay-database-66969cd859-n2ssm  1/1     Running             0          6d1h
example-registry-quay-mirror-764d7b68d9-jmlkk    1/1     Terminating         0          5d21h
example-registry-quay-mirror-764d7b68d9-jqzwg    1/1     Terminating         0          5d21h
example-registry-quay-redis-7cc5f6c977-956g8     1/1     Running             0          5d21h
```

7. After your Red Hat Quay registry has reconfigured, enter the following command to check if the Red Hat Quay app pods are running:

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

Example output

```
example-registry-quay-app-6786987b99-sz6kb       1/1     Running             0          7m45s
example-registry-quay-app-6786987b99-vgg2v       1/1     Running             0          9m1s
example-registry-quay-app-upgrade-lswsn          0/1     Completed           0          6d1h
example-registry-quay-config-editor-77847fc4f5-nsbbv   1/1     Running             0          9m1s
example-registry-quay-database-66969cd859-n2ssm  1/1     Running             0          6d1h
example-registry-quay-mirror-758fc68ff7-5wxlp    1/1     Running             0          8m29s
example-registry-quay-mirror-758fc68ff7-lbl82    1/1     Running             0          8m29s
example-registry-quay-redis-7cc5f6c977-956g8     1/1     Running             0          5d21h
```

8. In your browser, access the registry endpoint and validate that the certificate has been updated appropriately. For example:

```
Common Name (CN) example-registry-quay-quay-enterprise.apps.docs.quayteam.org
Organisation (O) DOCS
Organisational Unit (OU) QUAY
```

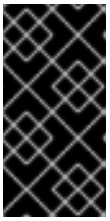
16.3.2.3. Using the UI to create a build trigger

Use the following procedure to use the UI to create a build trigger.

Procedure

1. Log in to your Red Hat Quay repository.
2. Click **Create New Repository** and create a new registry, for example, **testrepo**.
3. On the **Repositories** page, click the **Builds** tab on the navigation pane. Alternatively, use the corresponding URL directly:

```
https://example-registry-quay-quay-enterprise.apps.docs.quayteam.org/repository/quayadmin/testrepo?tab=builds
```



IMPORTANT

In some cases, the builder might have issues resolving hostnames. This issue might be related to the **dnsPolicy** being set to **default** on the job object. Currently, there is no workaround for this issue. It will be resolved in a future version of Red Hat Quay.

4. Click **Create Build Trigger** → **Custom Git Repository Push**.
5. Enter the HTTPS or SSH style URL used to clone your Git repository, then click **Continue**. For example:

```
https://github.com/gabriel-rh/actions_test.git
```

6. Check **Tag manifest with the branch or tag name** and then click **Continue**.
7. Enter the location of the Dockerfile to build when the trigger is invoked, for example, **/Dockerfile** and click **Continue**.
8. Enter the location of the context for the Docker build, for example, /, and click **Continue**.
9. If warranted, create a Robot Account. Otherwise, click **Continue**.
10. Click **Continue** to verify the parameters.
11. On the **Builds** page, click **Options** icon of your Trigger Name, and then click **Run Trigger Now**.
12. Enter a commit SHA from the Git repository and click **Start Build**.
13. You can check the status of your build by clicking the commit in the **Build History** page, or by running **oc get pods -n virtual-builders**. For example:

```
$ oc get pods -n virtual-builders
```

Example output

```
NAME                                     READY   STATUS    RESTARTS   AGE
f192fe4a-c802-4275-bcce-d2031e635126-9l2b5-25lg2   1/1     Running   0          7s
```

```
$ oc get pods -n virtual-builders
```

Example output

```
NAME                                     READY   STATUS        RESTARTS   AGE
f192fe4a-c802-4275-bcce-d2031e635126-9l2b5-25lg2   1/1     Terminating   0          9s
```

```
$ oc get pods -n virtual-builders
```

Example output

```
No resources found in virtual-builders namespace.
```

14. When the build is finished, you can check the status of the tag under **Tags** on the navigation pane.

NOTE

With early access, full build logs and timestamps of builds are currently unavailable.

16.3.2.4. Modifying your AWS S3 storage bucket

If you are using AWS S3 storage, you must change your storage bucket in the AWS console, prior to running builders.

Procedure

1. Log in to your AWS console at s3.console.aws.com.
2. In the search bar, search for **S3** and then click **S3**.
3. Click the name of your bucket, for example, **myawsbucket**.
4. Click the **Permissions** tab.
5. Under **Cross-origin resource sharing (CORS)** include the following parameters:

```
[
    {
        "AllowedHeaders": [
            "Authorization"
        ],
        "AllowedMethods": [
            "GET"
        ],
        "AllowedOrigins": [
            "*"
        ],
        "ExposeHeaders": [],
        "MaxAgeSeconds": 3000
    },
    {
```

```
        "AllowedHeaders": [
            "Content-Type",
            "x-amz-acl",
            "origin"
        ],
        "AllowedMethods": [
            "PUT"
        ],
        "AllowedOrigins": [
            "*"
        ],
        "ExposeHeaders": [],
        "MaxAgeSeconds": 3000
    }
]
```

16.3.2.5. Modifying your Google Cloud Platform object bucket

NOTE

Currently, modifying your Google Cloud Platform object bucket is not supported on IBM Power and IBM Z.

Use the following procedure to configure cross-origin resource sharing (CORS) for virtual builders.

NOTE

Without CORS configuration, uploading a build Dockerfile fails.

Procedure

1. Use the following reference to create a JSON file for your specific CORS needs. For example:

```
$ cat gcp_cors.json
```

Example output

```
[
   {
     "origin": ["*"],
     "method": ["GET"],
     "responseHeader": ["Authorization"],
     "maxAgeSeconds": 3600
   },
   {
     "origin": ["*"],
     "method": ["PUT"],
     "responseHeader": [
            "Content-Type",
            "x-goog-acl",
            "origin"],
```

```
    "maxAgeSeconds": 3600
  }
]
```

2. Enter the following command to update your GCP storage bucket:

```
$ gcloud storage buckets update gs://<bucket_name> --cors-file=./gcp_cors.json
```

Example output

```
Updating
  Completed 1
```

3. You can display the updated CORS configuration of your GCP bucket by running the following command:

```
$ gcloud storage buckets describe gs://<bucket_name>  --format="default(cors)"
```

Example output

```
cors:
- maxAgeSeconds: 3600
  method:
  - GET
  origin:
  - '*'
  responseHeader:
  - Authorization
- maxAgeSeconds: 3600
  method:
  - PUT
  origin:
  - '*'
  responseHeader:
  - Content-Type
  - x-goog-acl
  - origin
```

CHAPTER 17. USING THE RED HAT QUAY API

Red Hat Quay provides a full OAuth 2, RESTful API. [OAuth 2] RESTful API provides the following benefits:

- It is available from endpoint **/api/v1** endpoint of your Red Hat Quay host. For example, **https://<quay-server.example.com>/api/v1**.
- It allows users to connect to endpoints through their browser to **GET**, **POST**, **DELETE**, and **PUT** Red Hat Quay settings by enabling the Swagger UI.
- It can be accessed by applications that make API calls and use OAuth tokens.
- It sends and receives data as JSON.

The following section describes how to access the Red Hat Quay API so that it can be used with your deployment.

17.1. ACCESSING THE QUAY API FROM QUAY.IO

If you don't have your own Red Hat Quay cluster running yet, you can explore the Red Hat Quay API available from Quay.io from your web browser:

```
https://docs.quay.io/api/swagger/
```

The API Explorer that appears shows Quay.io API endpoints. You will not see superuser API endpoints or endpoints for Red Hat Quay features that are not enabled on Quay.io (such as Repository Mirroring).

From API Explorer, you can get, and sometimes change, information on:

- Billing, subscriptions, and plans
- Repository builds and build triggers
- Error messages and global messages
- Repository images, manifests, permissions, notifications, vulnerabilities, and image signing
- Usage logs
- Organizations, members and OAuth applications
- User and robot accounts
- and more...

Select to open an endpoint to view the Model Schema for each part of the endpoint. Open an endpoint, enter any required parameters (such as a repository name or image), then select the **Try it out!** button to query or change settings associated with a Quay.io endpoint.

17.2. CREATING A V1 OAUTH ACCESS TOKEN

OAuth access tokens are credentials that allow you to access protected resources in a secure manner. With Red Hat Quay, you must create an OAuth access token before you can access the API endpoints of your organization.

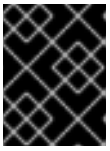
Use the following procedure to create an OAuth access token.

Prerequisites

- You have logged in to Red Hat Quay as an administrator.

Procedure

1. On the main page, select an Organization.
2. In the navigation pane, select **Applications**.
3. Click **Create New Application** and provide a new application name, then press **Enter**.
4. On the **OAuth Applications** page, select the name of your application.
5. Optional. Enter the following information:
    a. **Application Name**
    b. **Homepage URL**
    c. **Description**
    d. **Avatar E-mail**
    e. **Redirect/Callback URL prefix**
6. In the navigation pane, select **Generate Token**.
7. Check the boxes for the following options:
    a. **Administer Organization**
    b. **Administer Repositories**
    c. **Create Repositories**
    d. **View all visible repositories**
    e. **Read/Write to any accessible repositories**
    f. **Super User Access**
    g. **Administer User**
    h. **Read User Information**
8. Click **Generate Access Token**. You are redirected to a new page.
9. Review the permissions that you are allowing, then click **Authorize Application**. Confirm your decision by clicking **Authorize Application**.
10. You are redirected to the **Access Token** page. Copy and save the access token.



IMPORTANT

This is the only opportunity to copy and save the access token. It cannot be reobtained after leaving this page.

17.3. CREATING AN OCI REFERRERS OAUTH ACCESS TOKEN

In some cases, you might want to create an OCI referrers OAuth access token. This token is used to list OCI referrers of a manifest under a repository.

Procedure

1. Update your **config.yaml** file to include the **FEATURE_REFERRERS_API: true** field. For example:

```
# ...
FEATURE_REFERRERS_API: true
# ...
```

2. Enter the following command to Base64 encode your credentials:

```
$ echo -n '<username>:<password>' | base64
```

Example output

```
abcdeWFkbWluOjE5ODIraWROZXQxIQ==
```

3. Enter the following command to use the base64 encoded string and modify the URL endpoint to your Red Hat Quay server:

```
$ curl --location '<quay-server.example.com>/v2/auth?service=<quay-
server.example.com>&scope=repository:quay/listocireferrs:pull,push' --header 'Authorization:
Basic <base64_username:password_encode_token>' -k | jq
```

Example output

```
{
  "token":
"eyJhbGciOiJSUzI1NiIsImtpZCI6IjI5RWNtWmdiZ0l6czBBZW16emhTMHM1R0g2RDJnV2JGU
TdUNGZYand4MlUiLCJ0eXAiOiJKV1QifQ.eyJpc3MiOiJxdWF5IiwiYXVkIjoicXVheS1zZXJ2ZXIu
ZXhhbXBsZS5jb20iLCJuYmYiOjE3MjEzMzAzNDYsImlhdCI6MTcyMTMzMDM0NiwiZXhwIjoxNz
IxMzMzOTQ2LCJzdWIiOiJxdWF5YWRtaW4iLCJhY2Nlc3MiOlt7InR5cGUiOiJyZXBvc2l0b3J5Iiw
ibmFtZSI6InF1YXkvbGlzdG9jaXJlZmVycnMiLCJhY3Rpb25zIjpbXX1dLCJjb250ZXh0Ijp7InZlcn
Npb24iOjIsImVudGl0eV9raW5kIjoidXNlciIsImVudGl0eV9yZWZlcmVuY2UiOiJkZjI1M2QyNC0zZ
WUwLTRkODItOTcxYi1hZGYxMWYyNzBlM2IiLCJraW5kIjoidXNlciIsInVzZXIiOiJxdWF5YWRta
W4iLCJjb20uYXBvc3RpbGxlLnJvb3RzIjp7InF1YXkvbGlzdG9jaXJlZmVycnMiOiIkZGlzYWJsZW
QifSwiY29tLmFwb3N0aWxsZS5yb290IjoiJGRpc2FibGVkIn19.sBR765ea-
E41b2SfiIS36qoOmIZ6DEn9hvsCq3cszn6umlnKiBkc1jq6O1KlxtIhPdf8m8-
xtLMJakxkKST4mJg5CHR5WG2AVExuT6nCHg9KuzOZTkafMJeUzC4lxRsrdgKXyGUaYONO
ALf6bW_IebSIOOVt55m83-KVz5NMHSov9VmQlPCfGnWS3pq3bG-
nUaLhGRuSKc1EoGgnKlULNr9gAgzwBmB7-
MGioP7NL5_IQtrbjFyBdckQuJcpcwNK78gb8MQIwI-
```

```
e6WMvBT94pQkdD6bibo6zpFayFKSc6PsoO4Z4PjiON6vnD4kqEpX6rw5Yj7unv4RKjA_iHG-
BoQ"
}
```

17.4. REASSIGNING AN OAUTH ACCESS TOKEN

Organization administrators can assign OAuth API tokens to be created by other user's with specific permissions. This allows the audit logs to be reflected accurately when the token is used by a user that has no organization administrative permissions to create an OAuth API token.

NOTE

The following procedure only works on the current Red Hat Quay UI. It is not currently implemented in the Red Hat Quay v2 UI.

Prerequisites

- You are logged in as a user with organization administrative privileges, which allows you to assign an OAuth API token.

NOTE

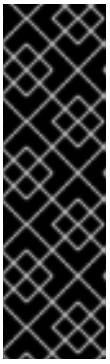
OAuth API tokens are used for authentication and not authorization. For example, the user that you are assigning the OAuth token to must have the **Admin** team role to use administrative API endpoints. For more information, see Managing access to repositories.

Procedure

1. Optional. If not already, update your Red Hat Quay **config.yaml** file to include the **FEATURE_ASSIGN_OAUTH_TOKEN: true** field:

```
# ...
FEATURE_ASSIGN_OAUTH_TOKEN: true
# ...
```

2. Optional. Restart your Red Hat Quay registry.
3. Log in to your Red Hat Quay registry as an organization administrator.
4. Click the name of the organization in which you created the OAuth token for.
5. In the navigation pane, click **Applications**.
6. Click the proper application name.
7. In the navigation pane, click **Generate Token**.
8. Click **Assign another user** and enter the name of the user that will take over the OAuth token.
9. Check the boxes for the desired permissions that you want the new user to have. For example, if you only want the new user to be able to create repositories, click **Create Repositories**.



IMPORTANT

Permission control is defined by the team role within an organization and must be configured regardless of the options selected here. For example, the user that you are assigning the OAuth token to must have the **Admin** team role to use administrative API endpoints.

Solely checking the **Super User Access** box does not actually grant the user this permission. Superusers must be configured via the **config.yaml** file *and* the box must be checked here.

10. Click **Assign token**. A popup box appears that confirms authorization with the following message and shows you the approved permissions:

> This will prompt user <username> to generate a token with the following permissions:
> repo:create

11. Click **Assign token** in the popup box. You are redirected to a new page that displays the following message:

> Token assigned successfully

Verification

1. After reassigning an OAuth token, the assigned user must accept the token to receive the bearer token, which is required to use API endpoints. Request that the assigned user logs into the Red Hat Quay registry.
2. After they have logged in, they must click their username under **Users and Organizations**.
3. In the navigation pane, they must click **External Logins And Applications**.
4. Under **Authorized Applications**, they must confirm the application by clicking **Authorize Application**. They are directed to a new page where they must reconfirm by clicking **Authorize Application**.
5. They are redirected to a new page that reveals their bearer token. They must save this bearer token, as it cannot be viewed again.

17.5. ACCESSING YOUR QUAY API FROM A WEB BROWSER

By enabling Swagger, you can access the API for your own Red Hat Quay instance through a web browser. This URL exposes the Red Hat Quay API explorer via the Swagger UI and this URL:

> https://<yourquayhost>/api/v1/discovery.

That way of accessing the API does not include superuser endpoints that are available on Red Hat Quay installations. Here is an example of accessing a Red Hat Quay API interface running on the local system by running the swagger-ui container image:

```
# export SERVER_HOSTNAME=<yourhostname>
# sudo podman run -p 8888:8080 -e API_URL=https://$SERVER_HOSTNAME:8443/api/v1/discovery
docker.io/swaggerapi/swagger-ui
```

With the swagger-ui container running, open your web browser to localhost port 8888 to view API endpoints via the swagger-ui container.

To avoid errors in the log such as "API calls must be invoked with an X-Requested-With header if called from a browser," add the following line to the **config.yaml** on all nodes in the cluster and restart Red Hat Quay:

```
BROWSER_API_CALLS_XHR_ONLY: false
```

17.6. ACCESSING THE RED HAT QUAY API FROM THE COMMAND LINE

You can use the **curl** command to GET, PUT, POST, or DELETE settings via the API for your Red Hat Quay cluster. Replace **<token>** with the OAuth access token you created earlier to get or change settings in the following examples.

CHAPTER 18. OPEN CONTAINER INITIATIVE SUPPORT

Container registries were originally designed to support container images in the Docker image format. To promote the use of additional runtimes apart from Docker, the Open Container Initiative (OCI) was created to provide a standardization surrounding container runtimes and image formats. Most container registries support the OCI standardization as it is based on the Docker image manifest V2, Schema 2 format.

In addition to container images, a variety of artifacts have emerged that support not just individual applications, but also the Kubernetes platform as a whole. These range from Open Policy Agent (OPA) policies for security and governance to Helm charts and Operators that aid in application deployment.

Red Hat Quay is a private container registry that not only stores container images, but also supports an entire ecosystem of tooling to aid in the management of containers. Red Hat Quay strives to be as compatible as possible with the OCI 1.1 *Image and Distribution specifications*, and supports common media types like *Helm charts* (as long as they pushed with a version of Helm that supports OCI) and a variety of arbitrary media types within the manifest or layer components of container images. Support for OCI media types differs from previous iterations of Red Hat Quay, when the registry was more strict about accepted media types. Because Red Hat Quay now works with a wider array of media types, including those that were previously outside the scope of its support, it is now more versatile accommodating not only standard container image formats but also emerging or unconventional types.

In addition to its expanded support for novel media types, Red Hat Quay ensures compatibility with Docker images, including V2_2 and V2_1 formats. This compatibility with Docker V2_2 and V2_1 images demonstrates Red Hat Quay's' commitment to providing a seamless experience for Docker users. Moreover, Red Hat Quay continues to extend its support for Docker V1 pulls, catering to users who might still rely on this earlier version of Docker images.

Support for OCI artifacts are enabled by default. The following examples show you how to use some some media types, which can be used as examples for using other OCI media types.

18.1. HELM AND OCI PREREQUISITES

Helm simplifies how applications are packaged and deployed. Helm uses a packaging format called *Charts* which contain the Kubernetes resources representing an application. Red Hat Quay supports Helm charts so long as they are a version supported by OCI.

Use the following procedures to pre-configure your system to use Helm and other OCI media types.

The most recent version of Helm can be downloaded from the Helm releases page. After you have downloaded Helm, you must enable your system to trust SSL/TLS certificates used by Red Hat Quay.

18.1.1. Enabling your system to trust SSL/TLS certificates used by Red Hat Quay

Communication between the Helm client and Red Hat Quay is facilitated over HTTPS. As of Helm 3.5, support is only available for registries communicating over HTTPS with trusted certificates. In addition, the operating system must trust the certificates exposed by the registry. You must ensure that your operating system has been configured to trust the certificates used by Red Hat Quay. Use the following procedure to enable your system to trust the custom certificates.

Procedure

1. Enter the following command to copy the **rootCA.pem** file to the **/etc/pki/ca-trust/source/anchors/** folder:

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

2. Enter the following command to update the CA trust store:

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

18.2. USING HELM CHARTS

Use the following example to download and push an etherpad chart from the Red Hat Community of Practice (CoP) repository.

Prerequisites

- You have logged into Red Hat Quay.

Procedure

1. Add a chart repository by entering the following command:

```
$ helm repo add redhat-cop https://redhat-cop.github.io/helm-charts
```

2. Enter the following command to update the information of available charts locally from the chart repository:

```
$ helm repo update
```

3. Enter the following command to pull a chart from a repository:

```
$ helm pull redhat-cop/etherpad --version=0.0.4 --untar
```

4. Enter the following command to package the chart into a chart archive:

```
$ helm package ./etherpad
```

Example output

```
Successfully packaged chart and saved it to: /home/user/linux-amd64/etherpad-0.0.4.tgz
```

5. Log in to Red Hat Quay using **helm registry login**:

```
$ helm registry login quay370.apps.quayperf370.perfscale.devcluster.openshift.com
```

6. Push the chart to your repository using the **helm push** command:

```
$ helm push etherpad-0.0.4.tgz
oci://quay370.apps.quayperf370.perfscale.devcluster.openshift.com
```

Example output:

```
Pushed: quay370.apps.quayperf370.perfscale.devcluster.openshift.com/etherpad:0.0.4
Digest: sha256:a6667ff2a0e2bd7aa4813db9ac854b5124ff1c458d170b70c2d2375325f2451b
```

7. Ensure that the push worked by deleting the local copy, and then pulling the chart from the repository:

```
$ rm -rf etherpad-0.0.4.tgz
```

```
$ helm pull oci://quay370.apps.quayperf370.perfscale.devcluster.openshift.com/etherpad --version 0.0.4
```

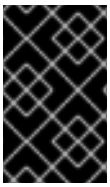
Example output:

```
Pulled: quay370.apps.quayperf370.perfscale.devcluster.openshift.com/etherpad:0.0.4
Digest: sha256:4f627399685880daf30cf77b6026dc129034d68c7676c7e07020b70cf7130902
```

18.3. ANNOTATION PARSING

Some OCI media types do not utilize labels and, as such, critical information such as expiration timestamps are not included. Red Hat Quay supports metadata passed through annotations to accommodate OCI media types that do not include these labels for metadata transmission. Tools such as ORAS (OCI Registry as Storage) can now be used to embed information with artifact types to help ensure that images operate properly, for example, to expire.

The following procedure uses ORAS to add an expiration date to an OCI media artifact.



IMPORTANT

If you pushed an image with **podman push**, and then add an annotation with **oras**, the MIME type is changed. Consequently, you will not be able to pull the same image with **podman pull** because Podman does not recognize that MIME type.

Prerequisites

- You have downloaded the **oras** CLI. For more information, see Installation.
- You have pushed an OCI media artifact to your Red Hat Quay repository.

Procedure

- By default, some OCI media types, like **application/vnd.oci.image.manifest.v1+json**, do not use certain labels, like expiration timestamps. You can use a CLI tool like ORAS (**oras**) to add annotations to OCI media types. For example:

```
$ oras push --annotation "quay.expires-after=2d" \ 1
--annotation "expiration = 2d" \ 2
quay.io/<organization_name>/<repository>/<image_name>:<tag>
```

1. Set the expiration time for 2 days, indicated by **2d**.
2. Adds the expiration label.

Example output

```
✓ Exists    application/vnd.oci.empty.v1+json
```

```
2/2  B 100.00%     0s
  └─ sha256:44136fa355b3678a1146ad16f7e8649e94fb4fc21fe77e8310c060f61caaff8a
✓ Uploaded  application/vnd.oci.image.manifest.v1+json
561/561  B 100.00%  511ms
  └─ sha256:9b4f2d43b62534423894d077f0ff0e9e496540ec8b52b568ea8b757fc9e7996b
Pushed [registry] quay.io/stevsmit/testorg3/oci-image:v1
ArtifactType: application/vnd.unknown.artifact.v1
Digest: sha256:9b4f2d43b62534423894d077f0ff0e9e496540ec8b52b568ea8b757fc9e7996b
```

Verification

1. Pull the image with **oras**. For example:

```
$ oras pull quay.io/<organization_name>/<repository>/<image_name>:<tag>
```

2. Inspect the changes using **oras**. For example:

```
$ oras manifest fetch quay.io/<organization_name>/<repository>/<image_name>:<tag>
```

Example output

```
{"schemaVersion":2,"mediaType":"application/vnd.oci.image.manifest.v1+json","artifactType":"a
pplication/vnd.unknown.artifact.v1","config":
{"mediaType":"application/vnd.oci.empty.v1+json","digest":"sha256:44136fa355b3678a1146ad1
6f7e8649e94fb4fc21fe77e8310c060f61caaff8a","size":2,"data":"e30="},"layers":
[{"mediaType":"application/vnd.oci.empty.v1+json","digest":"sha256:44136fa355b3678a1146ad
16f7e8649e94fb4fc21fe77e8310c060f61caaff8a","size":2,"data":"e30="}],"annotations":
{"org.opencontainers.image.created":"2024-07-11T15:22:42Z","version ":" 8.11"}}
```

18.4. ATTACHING REFERRERS TO AN IMAGE TAG

The following procedure shows you how to attach referrers to an image tag using different schemas supported by the OCI distribution spec 1.1 using the **oras** CLI. This is useful for attaching and managing additional metadata like referrers to container images.

Prerequisites

- You have downloaded the **oras** CLI. For more information, see Installation.
- You have access to an OCI media artifact.

Procedure

1. Tag an OCI media artifact by entering the following command:

```
$ podman tag <myartifact_image> <quay-
server.example.com>/<organization_name>/<repository>/<image_name>:<tag>
```

2. Push the artifact to your Red Hat Quay registry. For example:

```
$ podman push <myartifact_image> <quay-
server.example.com>/<organization_name>/<repository>/<image_name>:<tag>
```

3. Enter the following command to attach a manifest using the OCI 1.1 referrers **API** schema with **oras**:

```
$ oras attach --artifact-type <MIME_type> --distribution-spec v1.1-referrers-api
<myartifact_image> \
<quay-server.example.com>/<organization_name>/<repository>/<image_name>:<tag> \
<example_file>.txt
```

Example output

```
-spec v1.1-referrers-api quay.io/testorg3/myartifact-image:v1.0 hi.txt
✓ Exists    hi.txt                                                  3/3  B 100.00%     0s
  └─ sha256:98ea6e4f216f2fb4b69fff9b3a44842c38686ca685f3f55dc48c5d3fb1107be4
✓ Exists    application/vnd.oci.empty.v1+json                       2/2  B 100.00%     0s
  └─ sha256:44136fa355b3678a1146ad16f7e8649e94fb4fc21fe77e8310c060f61caaff8a
✓ Uploaded  application/vnd.oci.image.manifest.v1+json             723/723  B 100.00%
677ms
  └─ sha256:31c38e6adcc59a3cfbd2ef971792aaf124cbde8118e25133e9f9c9c4cd1d00c6
Attached to [registry] quay.io/testorg3/myartifact-
image@sha256:db440c57edfad40c682f9186ab1c1075707ce7a6fdda24a89cb8c10eaad424da

Digest: sha256:31c38e6adcc59a3cfbd2ef971792aaf124cbde8118e25133e9f9c9c4cd1d00c6
```

4. Enter the following command to attach a manifest using the OCI 1.1 referrers **tag** schema:

```
$ oras attach --artifact-type <MIME_type> --distribution-spec v1.1-referrers-tag \
<myartifact_image> <quay-
server.example.com>/<organization_name>/<repository>/<image_name>:<tag> \
<example_file>.txt
```

Example output

```
✓ Exists    hi.txt                                                3/3  B 100.00%     0s
  └─ sha256:98ea6e4f216f2fb4b69fff9b3a44842c38686ca685f3f55dc48c5d3fb1107be4
✓ Exists    application/vnd.oci.empty.v1+json                     2/2  B 100.00%     0s
  └─ sha256:44136fa355b3678a1146ad16f7e8649e94fb4fc21fe77e8310c060f61caaff8a
✓ Uploaded  application/vnd.oci.image.manifest.v1+json           723/723  B 100.00%
465ms
  └─ sha256:2d4b54201c8b134711ab051389f5ba24c75c2e6b0f0ff157fce8ffdfe104f383
Attached to [registry] quay.io/testorg3/myartifact-
image@sha256:db440c57edfad40c682f9186ab1c1075707ce7a6fdda24a89cb8c10eaad424da

Digest: sha256:2d4b54201c8b134711ab051389f5ba24c75c2e6b0f0ff157fce8ffdfe104f383
```

5. Enter the following command to discoverer referrers of the artifact using the **tag** schema:

```
$ oras discover --insecure --distribution-spec v1.1-referrers-tag \
<quay-server.example.com>/<organization_name>/<repository>/<image_name>:<tag>
```

Example output

```
quay.io/testorg3/myartifact-
image@sha256:db440c57edfad40c682f9186ab1c1075707ce7a6fdda24a89cb8c10eaad424da
```

```
└── doc/example
    └── sha256:2d4b54201c8b134711ab051389f5ba24c75c2e6b0f0ff157fce8ffdfe104f383
```

6. Enter the following command to discoverer referrers of the artifact using the **API** schema:

```
$ oras discover --distribution-spec v1.1-referrers-api \
<quay-server.example.com>/<organization_name>/<repository>/<image_name>:<tag>
```

Example output

```
Discovered 3 artifacts referencing v1.0
Digest:
sha256:db440c57edfad40c682f9186ab1c1075707ce7a6fdda24a89cb8c10eaad424da

Artifact Type   Digest
                sha256:2d4b54201c8b134711ab051389f5ba24c75c2e6b0f0ff157fce8ffdfe104f383

sha256:22b7e167793808f83db66f7d35fbe0088b34560f34f8ead36019a4cc48fd346b

sha256:bb2b7e7c3a58fd9ba60349473b3a746f9fe78995a88cb329fc2fd1fd892ea4e4
```

7. Optional. You can also discover referrers by using the **/v2/<organization_name>/<repository_name>/referrers/<sha256_digest>** endpoint. For this to work, you must generate a v2 API token and set **FEATURE_REFERRERS_API: true** in your **config.yaml** file.

   a. Update your **config.yaml** file to include the **FEATURE_REFERRERS_API** field. For example:

```
# ...
FEATURE_REFERRERS_API: true
# ...
```

   b. Enter the following command to Base64 encode your credentials:

```
$ echo -n '<username>:<password>' | base64
```

Example output

```
abcdeWFkbWluOjE5ODIraWROZXQxIQ==
```

   c. Enter the following command to use the base64 encoded token and modify the URL endpoint to your Red Hat Quay server:

```
$ curl --location '<quay-server.example.com>/v2/auth?service=<quay-
server.example.com>&scope=repository:quay/listocireferrs:pull,push' --header
'Authorization: Basic <base64_username:password_encode_token>' -k | jq
```

Example output

```
{
  "token":
```

"eyJhbGciOiJSUzI1NiIsImtpZCI6Ijl5RWNtWmdiZ0l6czBBZW16emhTMHM1R0g2RDJnV2J
GUTdUNGZYand4MlUiLCJ0eXAiOiJKV1QifQ.eyJpc3MiOiJxdWF5IiwiYXVkIjoicXVheS1zZ
XJ2ZXIuZXhhbXBsZS5jb20iLCJuYmYiOjE3MjEzMzAzNDYsImlhdCI6MTcyMTMzMDM0Ni
wiZXhwIjoxNzIxMzMzOTQ2LCJzdWIiOiJxdWF5YWRtaW4iLCJhY2Nlc3MiOlt7InR5cGUiOi
JyZXBvc2l0b3J5IiwibmFtZSI6InF1YXkvbGlzdG9jaXJlZmVycmMiLCJhY3Rpb25zIjpbXX1dL
CJjb250ZXh0Ijp7InZlcnNpb24iOjIsImVudGl0eV9raW5kIjoidXNlciIsImVudGl0eV9yZWZlcmV
uY2UiOiJkZjI1M2QyNC0zZWUwLTRkODItOTcxYi1hZGYxMWYyNzBlM2IiLCJraW5kIjoidX
NlciIsInVzZXIiOiJxdWF5YWRtaW4iLCJjb20uYXBvc3RpbGxlLnJvb3RzIjp7InF1YXkvbGlzdG
9jaXJlZmVycmMiOiIkZGlzYWJsZWQifSwiY29tLmFwb3N0aWxsZS5yb290IjoiJGRpc2FibGV
kIn19.sBR765ea-
E41b2SfiIS36qoOmIZ6DEn9hvsCq3cszn6umlnKiBkc1jq6O1KlxtIhPdf8m8-
xtLMJakxkKST4mJg5CHR5WG2AVExuT6nCHg9KuzOZTkafMJeUzC4lxRsrdgKXyGUaYO
NOALf6bW_IebSIOOVt55m83-KVz5NMHSov9VmQlPCfGnWS3pq3bG-
nUaLhGRuSKc1EoGgnKlULNr9gAgzwBmB7-
MGioP7NL5_IQtrbjFyBdckQuJcpcwNK78gb8MQIwI-
e6WMvBT94pQkdD6bibo6zpFayFKSc6PsoO4Z4PjiON6vnD4kqEpX6rw5Yj7unv4RKjA_iH
G-BoQ"
}

8. Enter the following command, using the v2 API token, to list OCI referrers of a manifest under a repository:

```
$ GET https://<quay-
server.example.com>/v2/<organization_name>/<repository_name>/referrers/sha256:0de63ba2
d98ab328218a1b6373def69ec0d0e7535866f50589111285f2bf3fb8
--header 'Authorization: Bearer <v2_bearer_token> -k | jq
```

Example output

```
{
  "schemaVersion": 2,
  "mediaType": "application/vnd.oci.image.index.v1+json",
  "manifests": [
    {
      "mediaType": "application/vnd.oci.image.manifest.v1+json",
      "digest":
"sha256:2d4b54201c8b134711ab051389f5ba24c75c2e6b0f0ff157fce8ffdfe104f383",
      "size": 793
    },
  ]
}
```