Red Hat Quay 3.12

About Quay IO

About Quay IO

Last Updated: 2024-11-05

Red Hat Quay 3.12 About Quay IO

About Quay IO

Legal Notice

Abstract

About Quay IO

Table of Contents

PREFACE

This comprehensive guide provides users with the knowledge and tools needed to make the most of our robust and feature-rich container registry service, Quay.io.

CHAPTER 1. QUAY.IO OVERVIEW

Quay.io is a registry for storing, building, and distributing container images and other OCI artifacts. Quay.io has gained widespread popularity among developers, organizations, and enterprises for establishing itself as one of the leading platforms in the containerization ecosystem. It offers both free and paid tiers to cater to various user needs.

At its core, Quay.io serves as a centralized repository for storing, managing, and distributing container images. Quay.io is flexible, easy to use, and offers an intuitive web interface that allows users to quickly upload and manage their container images. Developers can create private repositories, ensuring sensitive or proprietary code remains secure within their organization. Additionally, users can set up access controls and manage team collaboration, enabling seamless sharing of container images among designated team members.

Quay.io addresses container security concerns through its integrated image scanner, Clair. The service automatically scans container images for known vulnerabilities and security issues, providing developers with valuable insights into potential risks and suggesting remediation steps.

Quay.io excels in automation and supports integration with popular Continuous Integration/Continuous Deployment (CI/CD) tools and platforms, enabling seamless automation of the container build and deployment processes. As a result, developers can streamline their workflows, significantly reducing manual intervention and improving overall development efficiency.

Quay.io caters to the needs of both large and small-scale deployments. Its architecture and support for high availability ensures that organizations can rely on it for mission-critical applications. The platform can handle significant container image traffic and offers efficient replication and distribution mechanisms to deliver container images to various geographical locations.

Quay.io has established itself as an active hub for container enthusiasts. Developers can discover a vast collection of pre-built, public container images shared by other users, making it easier to find useful tools, applications, and services for their projects. This open sharing ecosystem fosters collaboration and accelerates software development within the container community.

As containerization continues to gain momentum in the software development landscape, Quay.io remains at the forefront, continually improving and expanding its services. The platform's commitment to security, ease of use, automation, and community engagement has solidified its position as a preferred container registry service for both individual developers and large organizations alike.

CHAPTER 2. QUAY.IO SUPPORT

Technical support is a crucial aspect of the Quay.io container registry service, providing assistance not only in managing container images but also ensuring the functionality and availability of the hosted platform.

To help users with functionality-related issues, Red Hat offers Quay.io customers access to several resources. The Red Hat Knowledgebase contains valuable content to maximize the potential of Red Hat's products and technologies. Users can find articles, product documentation, and videos that outline best practices for installing, configuring, and utilizing Red Hat products. It also serves as a hub for solutions to known issues, providing concise root cause descriptions and remedial steps.

Additionally, Quay.io customers can count on the technical support team to address questions, troubleshoot problems, and provide solutions for an optimized experience with the platform. Whether it involves understanding specific features, customizing configurations, or resolving container image build issues, the support team is dedicated to guiding users through each step with clarity and expertise.

For incidents related to service disruptions or performance issues not listed on the Quay.io status page, which includes availability and functionality concerns, paying customers can raise a technical support ticket using the Red Hat Customer Portal . A service incident is defined as an unplanned interruption of service or reduction in service quality, affecting multiple users of the platform.

With this comprehensive technical support system in place, Quay.io ensures that users can confidently manage their container images, optimize their platform experience, and overcome any challenges that might arise.

Additional resources

Current Red Hat Quay and Quay.io users can find more information about troubleshooting and support in the Red Hat Quay Troubleshooting guide .

CHAPTER 3. QUAY.IO USER INTERFACE OVERVIEW

The user interface (UI) of Quay.io is a fundamental component that serves as the user's gateway to managing and interacting with container images within the platform's ecosystem. Quay.io's UI is designed to provide an intuitive and user-friendly interface, making it easy for users of all skill levels to navigate and harness Quay.io's features and functionalities.

This documentation section aims to introduce users to the key elements and functionalities of Quay.io's UI. It will cover essential aspects such as the UI's layout, navigation, and key features, providing a solid foundation for users to explore and make the most of Quay.io's container registry service.

Throughout this documentation, step-by-step instructions, visual aids, and practical examples are provided on the following topics:

- Exploring applications and repositories
- Using the Quay.io tutorial
- Pricing and Quay.io plans
- Signing in and using Quay.io features

Collectively, this document ensures that users can quickly grasp the UI's nuances and successfully navigate their containerization journey with Quay.io.

3.1. QUAY.IO LANDING PAGE

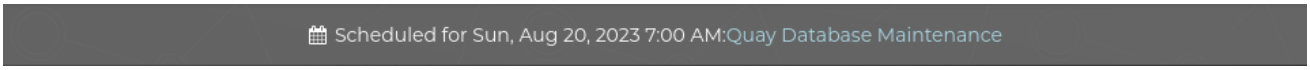
The Quay.io landing page serves as the central hub for users to access the container registry services offered. This page provides essential information and links to guide users in securely storing, building, and deploying container images effortlessly.

The landing page of Quay.io includes links to the following resources:

- Explore. On this page, you can search the Quay.io database for various applications and repositories.
- Tutorial. On this page, you can take a step-by-step walkthrough that shows you how to use Quay.io.
- Pricing. On this page, you can learn about the various pricing tiers offered for Quay.io. There are also various FAQs addressed on this page.
- Sign in. By clicking this link, you are re-directed to sign into your Quay.io repository.

.

The landing page also includes information about scheduled maintenance. During scheduled maintenance, Quay.io is operational in read-only mode, and pulls function as normal. Pushes and builds are non-operational during scheduled maintenance. You can subscribe to updates regarding Quay.io maintenance by navigating to Quay.io **Status** page and clicking **Subscribe To Updates**.



The landing page also includes links to the following resources:

- Documentation. This page provides documentation for using Quay.io.
- Terms. This page provides legal information about Red Hat Online Services.
- Privacy. This page provides information about Red Hat's Privacy Statement.
- Security. this page provides information about Quay.io security, including SSL/TLS, encryption, passwords, access controls, firewalls, and data resilience.
- About. This page includes information about packages and projects used and a brief history of the product.
- Contact. This page includes information about support and contacting the Red Hat Support Team.
- All Systems Operational. This page includes information the status of Quay.io and a brief history of maintenance.
- Cookies. By clicking this link, a popup box appears that allows you to set your cookie preferences.

.

You can also find information about Trying Red Hat Quay on premise or Trying Red Hat Quay on the cloud, which redirects you to the **Pricing** page. Each option offers a free trial.

3.1.1. Creating a Quay.io account

New users of Quay.io are required to both Register for a Red Hat account and create a Quay.io username. These accounts are correlated, with two distinct differences:

- The Quay.io account can be used to push and pull container images or Open Container Initiative images to Quay.io to store images.
- The Red Hat account provides users access to the Quay.io user interface. For paying customers, this account can also be used to access images from the Red Hat Ecosystem Catalog, which can be pushed to their Quay.io repository.

Users must first register for a Red Hat account, and then create a Quay.io account. Users need both accounts to properly use all features of Quay.io.

3.1.1.1. Registering for a Red Hat Account

Use the following procedure to register for a Red Hat account for Quay.io.

Procedure

1. Navigate to the Red Hat Customer Portal .
2. In navigation pane, click **Log In**.
3. When navigated to the log in page, click **Register for a Red Hat Account**

4. Enter a Red Hat login ID.
5. Enter a password.
6. Enter the following personal information:
    - **First name**
    - **Last name**
    - **Email address**
    - **Phone number**
7. Enter the following contact information that is relative to your country or region. For example:
    - **Country/region**
    - **Address**
    - **Postal code**
    - **City**
    - **County**
8. Select and agree to Red Hat's terms and conditions.
9. Click **Create my account**
10. Navigate to Quay.io and log in.

3.1.1.2. Creating a Quay.io user account

Use the following procedure to create a Quay.io user account.

Prerequisites

- You have created a Red Hat account.

Procedure

1. If required, resolve the captcha by clicking **I am not a robot** and confirming. You are redirected to a **Confirm Username** page.
2. On the **Confirm Username** page, enter a username. By default, a username is generated. If the same username already exists, a number is added at the end to make it unique. This username is be used as a namespace in the Quay Container Registry.
3. After deciding on a username, click **Confirm Username**. You are redirected to the Quay.io **Repositories** page, which serves as a dedicated hub where users can access and manage their repositories with ease. From this page, users can efficiently organize, navigate, and interact with their container images and related resources.

3.1.1.3. Quay.io Single Sign On support

Red Hat Single Sign On (SSO) can be used with Quay.io. Use the following procedure to set up Red Hat SSO with Quay.io. For most users, these accounts are already linked. However, for some legacy Quay.io users, this procedure might be required.

Prerequisites

- You have created a Quay.io account.

Procedure

1. Navigate to to the Quay.io **Recovery** page.
2. Enter your username and password, then click **Sign in to Quay Container Registry**
3. In the navigation pane, click your username → **Account Settings**.
4. In the navigation pane, click **External Logins and Applications**.
5. Click **Attach to Red Hat**
6. If you are already signed into Red Hat SSO, your account is automatically linked. Otherwise, you are prompted to sign into Red Hat SSO by entering your Red Hat login or email, and the password. Alternatively, you might need to create a new account first.
   After signing into Red Hat SSO, you can choose to authenticate against Quay.io using your Red Hat account from the login page.

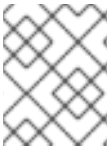
Additional resources

- For more information, see Quay.io Now Supports Red Hat Single Sign On .

3.1.2. Exploring Quay.io

The Quay.io **Explore** page is a valuable hub that allows users to delve into a vast collection of container images, applications, and repositories shared by the Quay.io community. With its intuitive and user-friendly design, the **Explore** page offers a powerful search function, enabling users to effortlessly discover containerized applications and resources.

3.1.3. Trying Quay.io (deprecated)



NOTE

The Red Hat Quay tutorial is currently deprecated and will be removed when the v2 UI goes generally available (GA).

The Quay.io **Tutorial** page offers users and introduction to the Quay.io container registry service. By clicking **Continue Tutorial** users learn how to perform the following features on Quay.io:

- Logging into Quay Container Registry from the Docker CLI
- Starting a container
- Creating images from a container
- Pushing a repository to Quay Container Registry

- Viewing a repository
- Setting up build triggers
- Changing a repository's permissions

3.1.4. Information about Quay.io pricing

In addition to a free tier, Quay.io also offers several paid plans that have enhanced benefits.

The Quay.io **Pricing** page offers information about Quay.io plans and the associated prices of each plan. The cost of each tier can be found on the **Pricing** page. All Quay.io plans include the following benefits:

- Continuous integration
- Public repositories
- Robot accounts
- Teams
- SSL/TLS encryption
- Logging and auditing
- Invoice history

Quay.io subscriptions are handled by the Stripe payment processing platform. A valid credit card is required to sign up for Quay.io.

To sign up for Quay.io, use the following procedure.

Procedure

1. Navigate to the Quay.io **Pricing** page.
2. Decide on a plan, for example, **Small**, and click **Buy Now**. You are redirected to the **Create New Organization** page. Enter the following information:
   - **Organization Name**
   - **Organization Email**
   - Optional. You can select a different plan if you want a plan larger, than, for example, **Small**.
3. Resolve that captcha, and select **Create Organization**.
4. You are redirected to Stripe. Enter the following information:
   - **Card information**, including **MM/YY** and the **CVC**
   - **Name on card**
   - **Country or region**
   - **ZIP** (if applicable)

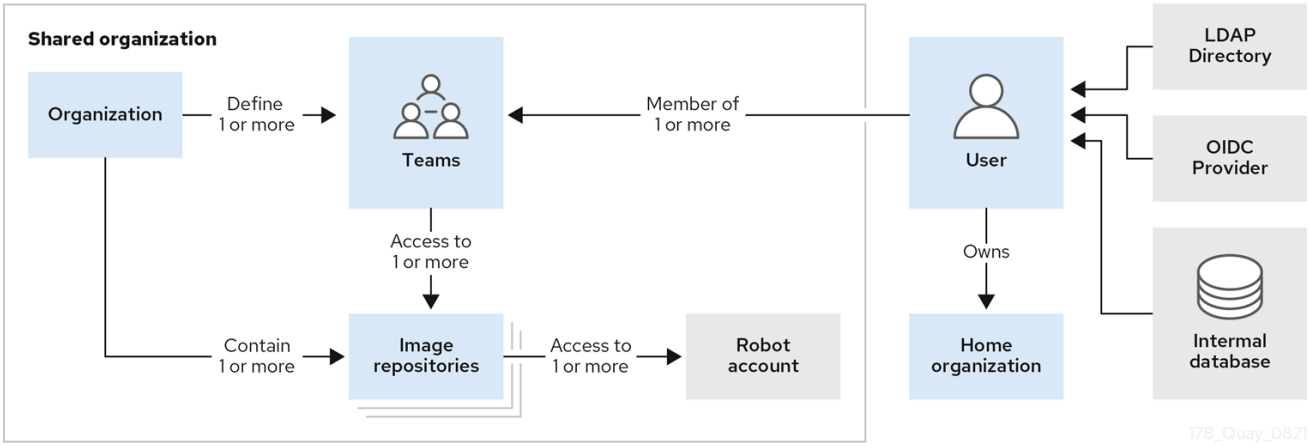
- Check the box if you want your information to be saved.
- **Phone Number**

5. Click **Subscribe** after all boxes have been filled.

CHAPTER 4. RED HAT QUAY TENANCY MODEL

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

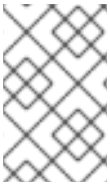
4.1. TENANCY MODEL



- **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.
- **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.

4.2. LOGGING INTO QUAY

A user account for Quay.io represents an individual with authenticated access to the platform's features and functionalities. Through this account, you gain 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 your container image management within Quay.io.



NOTE

Not all features on Quay.io require that users be logged in. For example, you can anonymously pull an image from Quay.io without being logged in, so long as the image you are pulling comes from a public repository.

Users have two options for logging into Quay.io:

- By logging in through Quay.io.
  This option provides users with the legacy UI, as well as an option to use the beta UI environment, which adheres to PatternFly UI principles.

- By logging in through the Red Hat Hybrid Cloud Console .
  This option uses Red Hat SSO for authentication, and is a public managed service offering by Red Hat. This option *always* requires users to login. Like other managed services, Quay on the Red Hat Hybrid Cloud Console enhances the user experience by adhering to PatternFly UI principles.

Differences between using Quay.io directly and Quay on the Red Hat Hybrid Cloud Console are negligible, including for users on the free tier. Whether you are using Quay.io directly, on the Hybrid Cloud Console, features that require login, such as pushing to a repository, use your Quay.io username specifications.

4.2.1. Logging into Quay.io

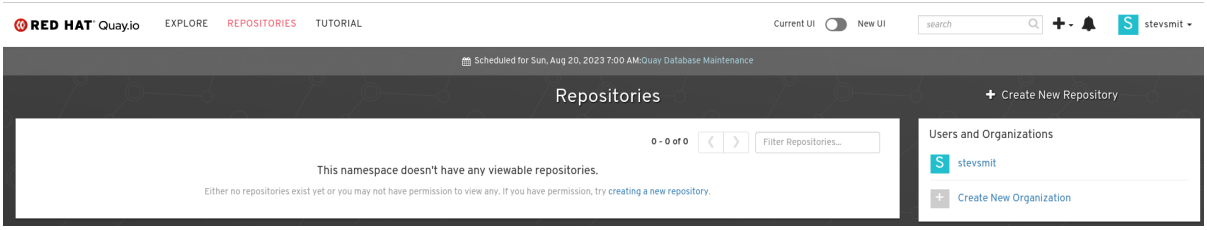
Use the following procedure to log into Quay.io.

Prerequisites

- You have created a Red Hat account and a Quay.io account. For more information, see "Creating a Quay.io account".

Procedure

1. Navigate to Quay.io.
2. In the navigation pane, select **Sign In** and log in using your Red Hat credentials.
3. If it is your first time logging in, you must confirm the automatically-generated username. Click **Confirm Username** to log in.
   You are redirected to the Quay.io repository landing page.



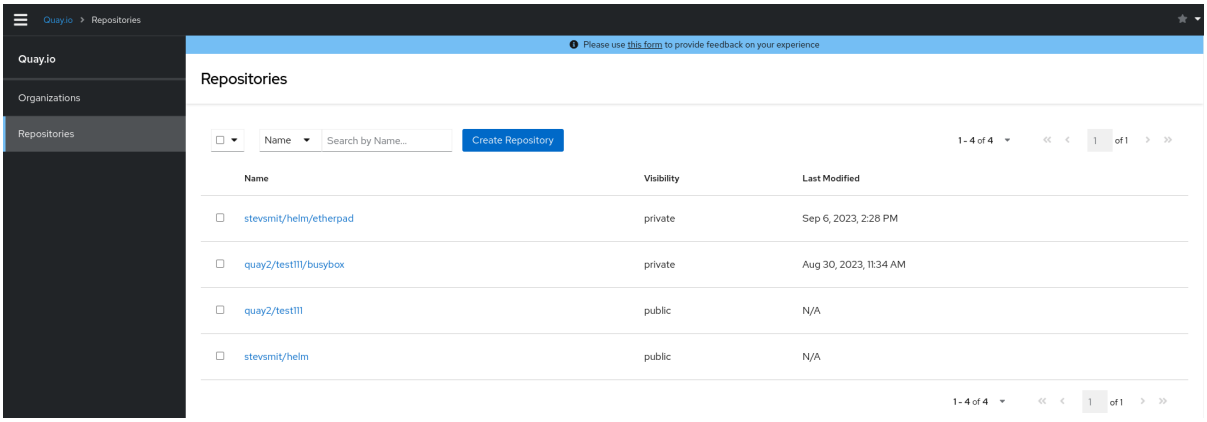
4.2.2. Logging into Quay through the Hybrid Cloud Console

Prerequisites

- You have created a Red Hat account and a Quay.io account. For more information, see "Creating a Quay.io account".

Procedure

1. Navigate to the Quay on the Red Hat Hybrid Cloud Console and log in using your Red Hat account. You are redirected to the Quay repository landing page:


Quay.io > Repositories
Quay.io
Organizations
Repositories
Please use this form to provide feedback on your experience
Repositories
Name
Search by Name...
Create Repository
1 - 4 of 4
1 of 1
Name
Visibility
Last Modified
stevsmit/helm/etherpad
private
Sep 6, 2023, 2:28 PM
quay2/test111/busybox
private
Aug 30, 2023, 11:34 AM
quay2/test111
public
N/A
stevsmit/helm
public
N/A
1 - 4 of 4
1 of 1

CHAPTER 5. QUAY.IO ORGANIZATIONS OVERVIEW

In = Quay.io 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 Quay.io UI.

5.1. 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.

5.2. ORGANIZATION SETTINGS

With = Quay.io, 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

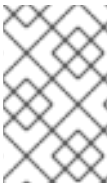
1. On the v2 UI, click **Organizations**.
2. Click the name of the organization that you will create the robot account for, for example, **test-org**.
3. Click the **Settings** tab.
4. Optional. Enter the email address associated with the organization.
5. Optional. Set the allotted time for the **Time Machine** feature to one of the following:
   - **A few seconds**
   - **A day**
   - **7 days**
   - **14 days**
   - **A month**
6. Click **Save**.

5.3. 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.

With a paid plan, Quay.io 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.

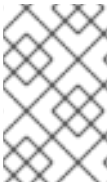


NOTE

The free tier of Quay.io does not allow for private repositories. You must upgrade to a paid tier of Quay.io to create a private repository. For more information, see "Information about Quay.io pricing".

There are two ways to create a repository in Quay.io: by pushing an image with the relevant **podman** command, or by using the Quay.io UI. You can also use the UI to delete a repository.

If you push an image through the command-line interface (CLI) without first creating a repository on the UI, the created repository is set to **Private**, regardless of the plan you have.



NOTE

It is recommended that you create a repository on the Quay.io UI before pushing an image. Quay.io checks the plan status and does not allow creation of a private repository if a plan is not active.

5.3.1. Creating a repository by using the UI

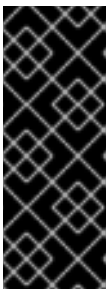
Use the following procedure to create a repository using the Quay.io UI.

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.

5.3.2. 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 Quay.io 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 Quay.io. After pushing an image to your registry, a repository is created.

If you push an image through the command-line interface (CLI) without first creating a repository on the UI, the created repository is set to **Private**, regardless of the plan you have.

NOTE

It is recommended that you create a repository on the Quay.io UI before pushing an image. Quay.io checks the plan status and does not allow creation of a private repository if a plan is not active.

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:

```
$ 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:

```
$ podman tag docker.io/library/busybox quay.io/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.

```
$ podman push --tls-verify=false quay.io/quayadmin/busybox:test
```

Example output

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

5.4. ACCESS MANAGEMENT FOR RED HAT QUAY

As a Quay.io 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 Quay.io, 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 Quay.io user account.

5.5. USER SETTINGS

The **User Settings** page provides users a way to set their email address, password, account type, set up desktop notifications, select an avatar, delete an account, adjust the *time machine* setting, and view billing information.

5.5.1. Navigating to the User Settings page

Use the following procedure to navigate to the **User Settings** page.

Procedure

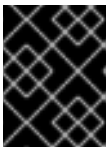
1. On Quay.io, click your username in the header.
2. Select **Account Settings**. You are redirected to the **User Settings** page.

5.5.2. Adjusting user settings

Use the following procedure to adjust user settings.

Procedure

- To change your email address, select the current email address for **Email Address**. In the pop-up window, enter a new email address, then, click **Change Email**. A verification email will be sent before the change is applied.
- To change your password, click **Change password**. Enter the new password in both boxes, then click **Change Password**.
- Change the account type by clicking **Individual Account**, or the option next to **Account Type**. In some cases, you might have to leave an organization prior to changing the account type.
- Adjust your desktop notifications by clicking the option next to **Desktop Notifications**. Users can either enable, or disable, this feature.
- You can delete an account by clicking **Begin deletion**. You cannot delete an account if you have an active plan, or if you are a member of an organization where you are the only administrator. You must confirm deletion by entering the namespace.



IMPORTANT

Deleting an account is not reversible and will delete all of the account's data including repositories, created build triggers, and notifications.

- You can set the *time machine* feature by clicking the drop-box next to **Time Machine**. This feature dictates the amount of time after a tag is deleted that the tag is accessible in time machine before being garbage collected. After selecting a time, click **Save Expiration Time**.

5.5.3. Billing information

You can view billing information on the **User Settings**. In this section, the following information is available:

- **Current Plan**. This section denotes the current plan Quay.io plan that you are signed up for. It also shows the amount of private repositories you have.

- **Invoices**. If you are on a paid plan, you can click **View Invoices** to view a list of invoices.
- **Receipts**. If you are on a paid plan, you can select whether to have receipts for payment emailed to you, another user, or to opt out of receipts altogether.

CHAPTER 6. 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.

6.1. 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.

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

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

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.

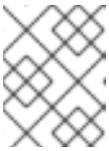
6.3. ADDING AND MANAGING LABELS BY USING THE UI

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.

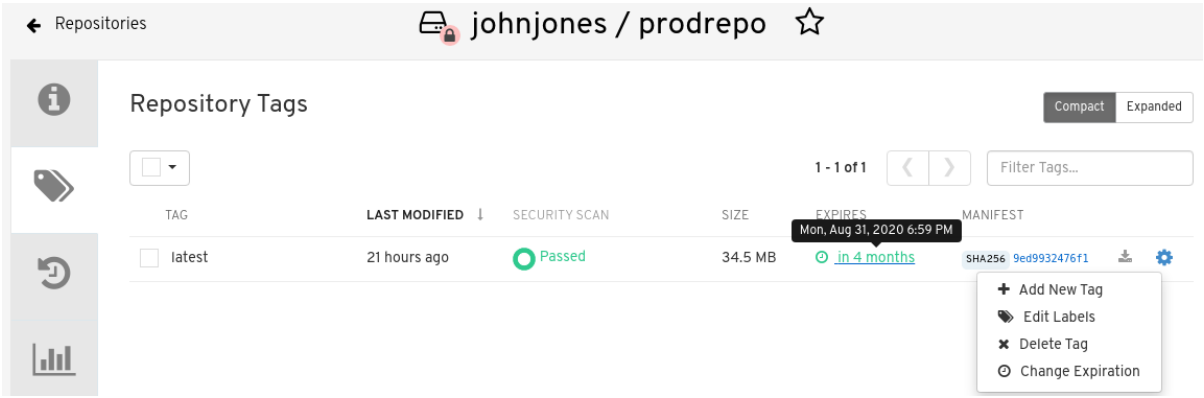
6.4. SETTING TAG EXPIRATIONS

Image tags can be set to expire from a Quay.io 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.

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 Quay.io UI. For example:

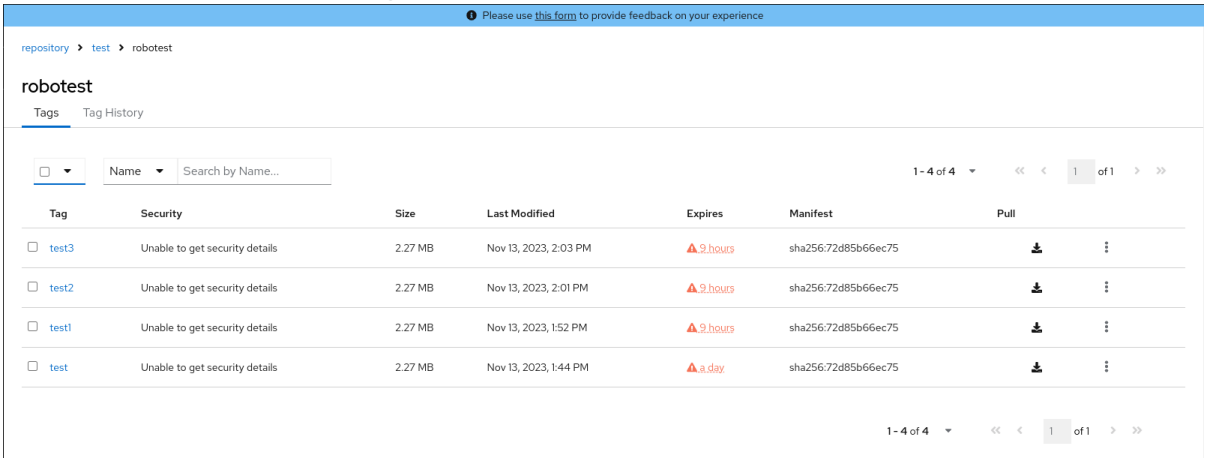


Setting tag expirations can help automate the cleanup of older or unused tags, helping to reduce storage space.

6.4.1. 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:



6.4.2. 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>
```

6.5. FETCHING AN IMAGE BY TAG OR DIGEST

Quay.io 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.io/quayadmin/busybox:test2
```

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

Quay.io 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

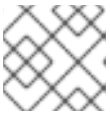
6.7. DELETING AN IMAGE TAG

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".

6.8. REVERTING TAG CHANGES BY USING THE UI

Quay.io 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.

CHAPTER 7. VIEWING AND EXPORTING LOGS

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

Viewing usage logs of Quay.io. 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.

7.1. 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.

7.2. EXPORTING REPOSITORY LOGS BY USING THE UI

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

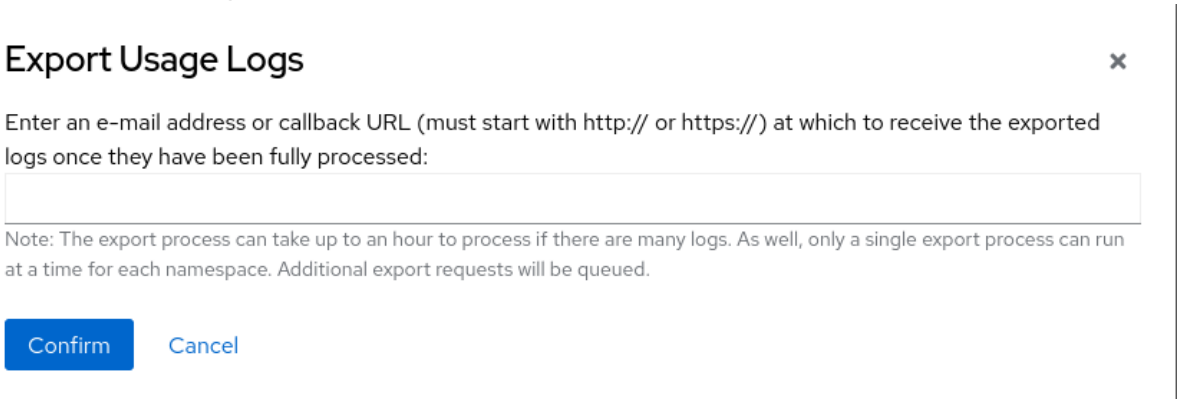
- You can choose a range of dates for the logs you want to gather from a repository.
- You can request that the logs be sent to you by an email attachment or directed to a callback URL.
- To export logs, you must be an administrator of the repository or namespace.
- 30 days worth of logs are retained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

CHAPTER 8. CLAIR SECURITY SCANNER

Clair v4 (Clair) is an open source application that leverages static code analyses for parsing image content and reporting vulnerabilities affecting the content. Clair is packaged with Quay.io, is automatically enabled, and is managed by the Red Hat Quay development team.

For Quay.io users, images are automatically indexed after they are pushed to your repository. Reports are then fetched from Clair, which matches images against its CVE's database to report security information. This process happens automatically on Quay.io, and manual recans are not required.

8.1. ABOUT CLAIR

Clair uses Common Vulnerability Scoring System (CVSS) data from the National Vulnerability Database (NVD) to enrich vulnerability data, which is a United States government repository of security-related information, including known vulnerabilities and security issues in various software components and systems. Using scores from the NVD provides Clair the following benefits:

- **Data synchronization**. Clair can periodically synchronize its vulnerability database with the NVD. This ensures that it has the latest vulnerability data.
- **Matching and enrichment**. Clair compares the metadata and identifiers of vulnerabilities it discovers in container images with the data from the NVD. This process involves matching the unique identifiers, such as Common Vulnerabilities and Exposures (CVE) IDs, to the entries in the NVD. When a match is found, Clair can enrich its vulnerability information with additional details from NVD, such as severity scores, descriptions, and references.
- **Severity Scores**. The NVD assigns severity scores to vulnerabilities, such as the Common Vulnerability Scoring System (CVSS) score, to indicate the potential impact and risk associated with each vulnerability. By incorporating NVD's severity scores, Clair can provide more context on the seriousness of the vulnerabilities it detects.

If Clair finds vulnerabilities from NVD, a detailed and standardized assessment of the severity and potential impact of vulnerabilities detected within container images is reported to users on the UI. CVSS enrichment data provides Clair the following benefits:

- **Vulnerability prioritization**. By utilizing CVSS scores, users can prioritize vulnerabilities based on their severity, helping them address the most critical issues first.
- **Assess Risk**. CVSS scores can help Clair users understand the potential risk a vulnerability poses to their containerized applications.
- **Communicate Severity**. CVSS scores provide Clair users a standardized way to communicate the severity of vulnerabilities across teams and organizations.
- **Inform Remediation Strategies**. CVSS enrichment data can guide Quay.io users in developing appropriate remediation strategies.
- **Compliance and Reporting**. Integrating CVSS data into reports generated by Clair can help organizations demonstrate their commitment to addressing security vulnerabilities and complying with industry standards and regulations.

8.1.1. Clair vulnerability databases

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

- Ubuntu Oval database

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

8.1.2. Clair supported dependencies

Clair supports identifying and managing the following dependencies:

- Java
- Golang
- Python
- Ruby

This means that it can analyze and report on the third-party libraries and packages that a project in these languages relies on to work correctly.

When an image that contains packages from a language unsupported by Clair is pushed to your repository, a vulnerability scan cannot be performed on those packages. Users do not receive an analysis or security report for unsupported dependencies or packages. As a result, the following consequences should be considered:

- **Security risks**. Dependencies or packages that are not scanned for vulnerability might pose security risks to your organization.
- **Compliance issues**. If your organization has specific security or compliance requirements, unscanned, or partially scanned, container images might result in non-compliance with certain regulations.

NOTE

Scanned images are indexed, and a vulnerability report is created, but it might omit data from certain unsupported languages. For example, if your container image contains a Lua application, feedback from Clair is not provided because Clair does not detect it. It can detect other languages used in the container image, and shows detected CVEs for those languages. As a result, Clair images are *fully scanned* based on what it supported by Clair.

8.2. 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.

8.3. CLAIR SEVERITY MAPPING

Clair offers a comprehensive approach to vulnerability assessment and management. One of its essential features is the normalization of security databases' severity strings. This process streamlines the assessment of vulnerability severities by mapping them to a predefined set of values. Through this mapping, clients can efficiently react to vulnerability severities without the need to decipher the intricacies of each security database's unique severity strings. These mapped severity strings align with those found within the respective security databases, ensuring consistency and accuracy in vulnerability assessment.

8.3.1. Clair severity strings

Clair alerts users with the following severity strings:

- Unknown
- Negligible
- Low
- Medium
- High
- Critical

These severity strings are similar to the strings found within the relevant security database.

Alpine mapping
Alpine SecDB database does not provide severity information. All vulnerability severities will be Unknown.

Alpine Severity	Clair Severity
*	Unknown

AWS mapping
AWS UpdateInfo database provides severity information.

AWS Severity	Clair Severity
low	Low
medium	Medium
important	High
critical	Critical

Debian mapping
Debian Oval database provides severity information.

Debian Severity	Clair Severity
*	Unknown
Unimportant	Low
Low	Medium
Medium	High
High	Critical

Oracle mapping
Oracle Oval database provides severity information.

Oracle Severity	Clair Severity
N/A	Unknown
LOW	Low
MODERATE	Medium
IMPORTANT	High
CRITICAL	Critical

RHEL mapping
RHEL Oval database provides severity information.

RHEL Severity	Clair Severity
None	Unknown
Low	Low
Moderate	Medium
Important	High
Critical	Critical

SUSE mapping
SUSE Oval database provides severity information.

Severity	Clair Severity
None	Unknown
Low	Low
Moderate	Medium
Important	High
Critical	Critical

Ubuntu mapping
Ubuntu Oval database provides severity information.

Severity	Clair Severity
Untriaged	Unknown
Negligible	Negligible
Low	Low
Medium	Medium
High	High
Critical	Critical

OSV mapping

Table 8.1. CVSS v3

Table 8.1. CVSSv3

Base Score	Clair Severity
0.0	Negligible
0.1-3.9	Low
4.0-6.9	Medium
7.0-8.9	High
9.0-10.0	Critical

Table 8.2. CVSSv2

Base Score	Clair Severity
0.0-3.9	Low
4.0-6.9	Medium
7.0-10	High

CHAPTER 9. BUILDING CONTAINER IMAGES

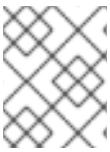
Building container images involves creating a blueprint for a containerized application. Blueprints rely on base images from other public repositories that define how the application should be installed and configured.

NOTE

Because blueprints rely on images from other public repositories, they might be subject to rate limiting. Consequently, your build *could* fail.

Quay.io 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.

On Quay.io, this feature works the same across both free, and paid, tier plans.



NOTE

Quay.io limits the number of simultaneous builds that a single user can submit at one time.

9.1. 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 Quay.io is not different than running **docker build** or **podman build** on your local machine.

Quay.io 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, Quay.io 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.

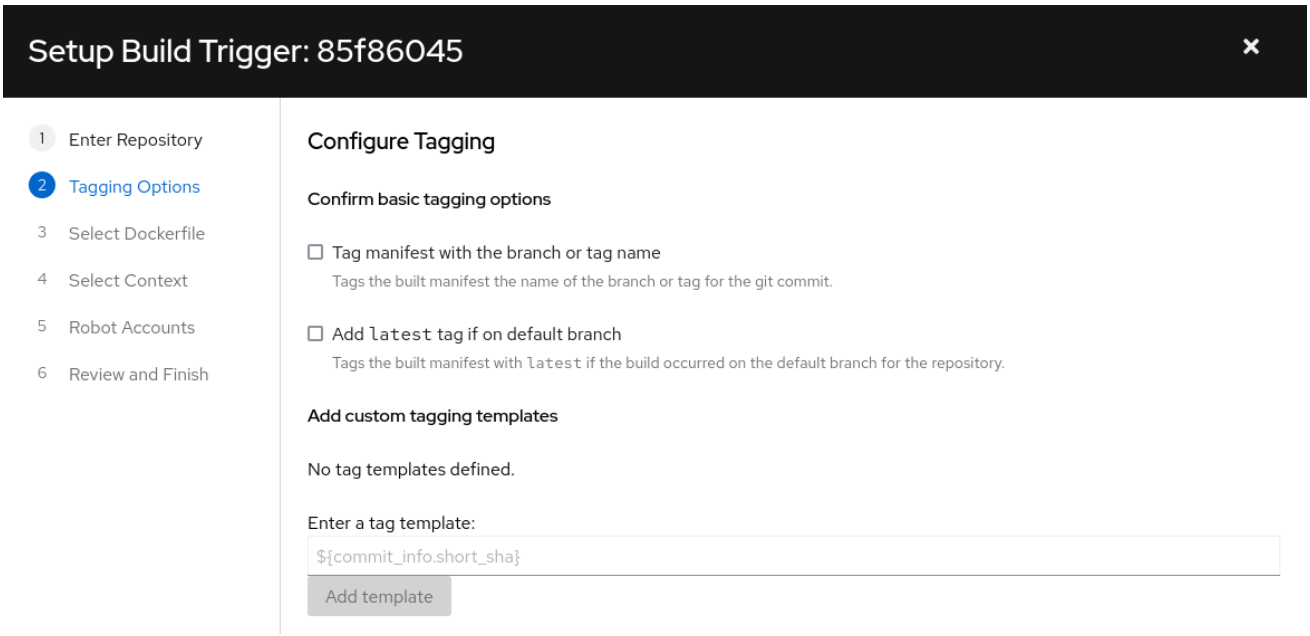
If a subdirectory is specified in the Build trigger setup, for example, **subdir**, only the Dockerfile within it is visible to the Build. This means that you cannot use the **ADD** command in the Dockerfile to add **file**, because it is outside of the Build context.

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

9.2. TAG NAMING FOR BUILD TRIGGERS

Custom tags are available for use in Quay.io.

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

9.3. SKIPPING A SOURCE CONTROL-TRIGGERED BUILD

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

9.4. VIEWING AND MANAGING BUILDS

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

Procedure

1. Navigate to Quay.io and select a repository.
2. In the navigation pane, select **Builds**.

9.5. CREATING A NEW BUILD

By default, Quay.io users can create new Builds out-of-the-box.

Prerequisites

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

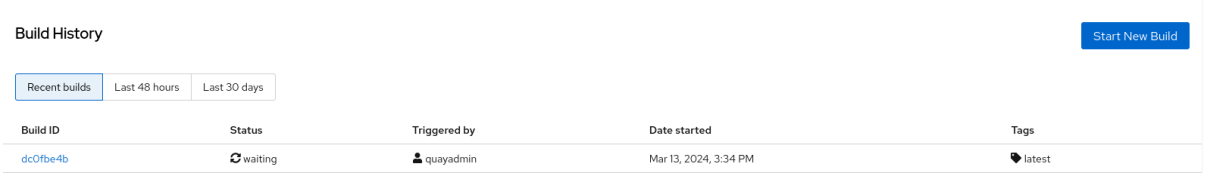
Procedure

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.

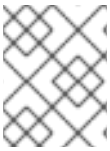


9.6. BUILD TRIGGERS

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

9.6.1. 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

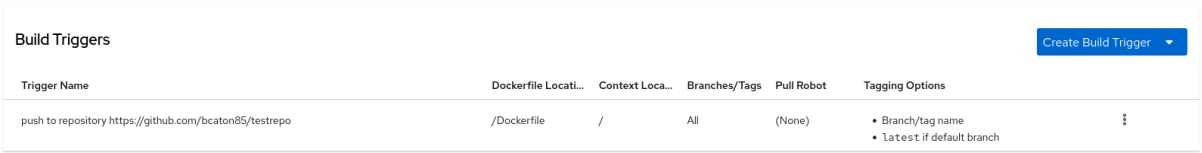
- You have navigated to the **Builds** page of your repository.

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:



9.6.2. 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.

9.7. 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.

9.7.1. Creating a trigger

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

- Quay.io 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.

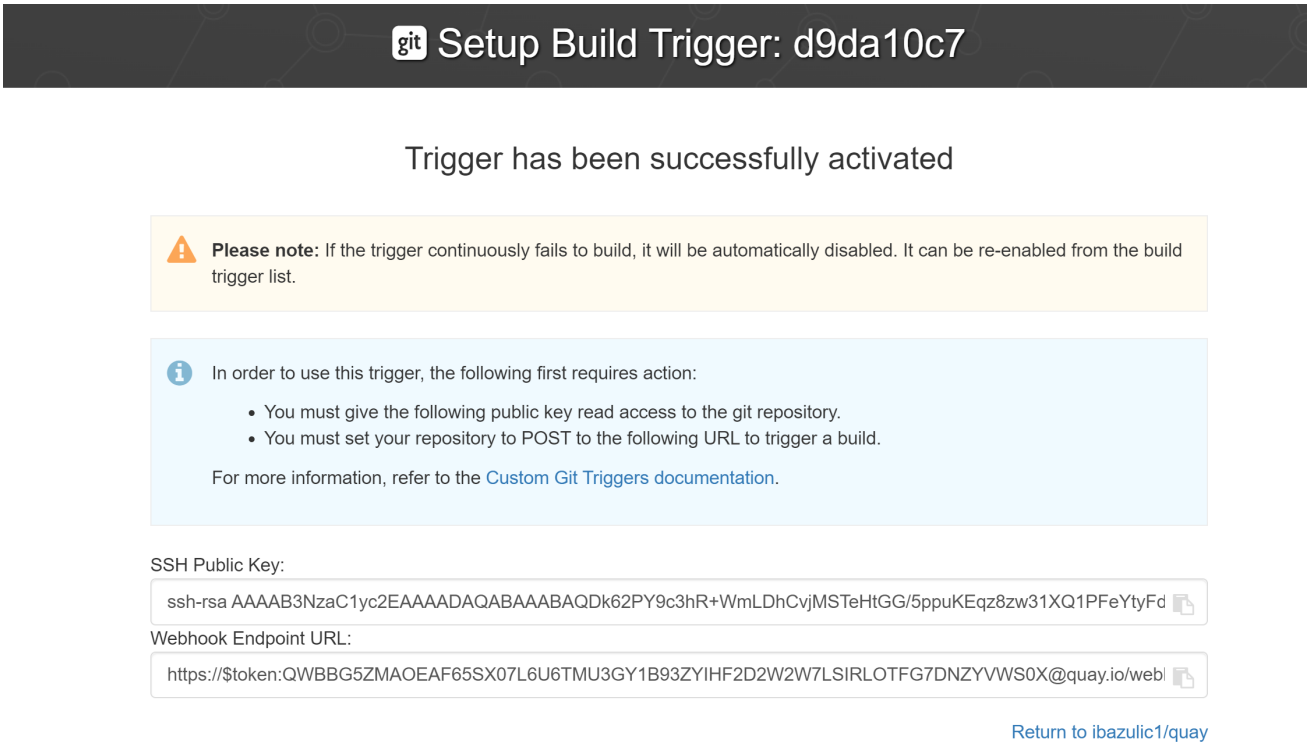
9.7.2. Custom trigger creation setup

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 Quay.io 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



9.7.2.1. SSH public key access

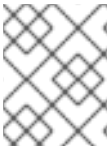
Depending on the Git server configuration, there are multiple ways to install the SSH public key that Quay.io generates for a custom Git trigger.

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**.

9.7.2.2. 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 10. NOTIFICATIONS OVERVIEW

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

CHAPTER 11. 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.

Quay.io 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. Quay.io 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 Quay.io, when the registry was more strict about accepted media types. Because Quay.io 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, Quay.io ensures compatibility with Docker images, including V2_2 and V2_1 formats. This compatibility with Docker V2_2 and V2_1 images demonstrates Quay.io's commitment to providing a seamless experience for Docker users. Moreover, Quay.io 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.

11.1. 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. Quay.io 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.

11.2. 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 Quay.io.

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 Quay.io using **helm registry login**:

```
$ helm registry login quay.io
```

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

```
helm push etherpad-0.0.4.tgz oci://quay.io/<organization_name>/helm
```

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://quay.io/<organization_name>/helm/etherpad --version 0.0.4
```

Example output:

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

11.3. COSIGN OCI SUPPORT

Cosign is a tool that can be used to sign and verify container images. It uses the **ECDSA-P256** signature algorithm and Red Hat's Simple Signing payload format to create public keys that are stored in PKIX files. Private keys are stored as encrypted PEM files.

Cosign currently supports the following:

- Hardware and KMS Signing
- Bring-your-own PKI
- OIDC PKI
- Built-in binary transparency and timestamping service

Use the following procedure to directly install Cosign.

Prerequisites

- You have installed Go version 1.16 or later.

Procedure

1. Enter the following **go** command to directly install Cosign:

```
$ go install github.com/sigstore/cosign/cmd/cosign@v1.0.0
```

Example output

```
go: downloading github.com/sigstore/cosign v1.0.0
go: downloading github.com/peterbourgon/ff/v3 v3.1.0
```

2. Generate a key-value pair for Cosign by entering the following command:

```
$ cosign generate-key-pair
```

Example output

```
Enter password for private key:
Enter again:
Private key written to cosign.key
Public key written to cosign.pub
```

3. Sign the key-value pair by entering the following command:

```
$ cosign sign -key cosign.key quay.io/user1/busybox:test
```

Example output

```
Enter password for private key:
Pushing signature to: quay-server.example.com/user1/busybox:sha256-
ff13b8f6f289b92ec2913fa57c5dd0a874c3a7f8f149aabee50e3d01546473e3.sig
```

If you experience the **error: signing quay-server.example.com/user1/busybox:test: getting remote image: GET https://quay-server.example.com/v2/user1/busybox/manifests/test: UNAUTHORIZED: access to the requested resource is not authorized; map[]** error, which occurs because Cosign relies on **~./docker/config.json** for authorization, you might need to execute the following command:

```
$ podman login --authfile ~/.docker/config.json quay.io
```

Example output

```
Username:
Password:
Login Succeeded!
```

4. Enter the following command to see the updated authorization configuration:

```
$ cat ~/.docker/config.json
{
 "auths": {
  "quay-server.example.com": {
   "auth": "cXVheWFkbWluOnBhc3N3b3Jk"
  }
 }
```

11.4. INSTALLING AND USING COSIGN

Use the following procedure to directly install Cosign.

Prerequisites

- You have installed Go version 1.16 or later.
- You have set **FEATURE_GENERAL_OCI_SUPPORT** to **true** in your **config.yaml** file.

Procedure

1. Enter the following **go** command to directly install Cosign:

```
$ go install github.com/sigstore/cosign/cmd/cosign@v1.0.0
```

Example output

```
go: downloading github.com/sigstore/cosign v1.0.0
go: downloading github.com/peterbourgon/ff/v3 v3.1.0
```

2. Generate a key-value pair for Cosign by entering the following command:

```
$ cosign generate-key-pair
```

Example output

```
Enter password for private key:
Enter again:
Private key written to cosign.key
Public key written to cosign.pub
```

3. Sign the key-value pair by entering the following command:

```
-
```

```
$ cosign sign -key cosign.key quay.io/user1/busybox:test
```

Example output

```
Enter password for private key:
Pushing signature to: quay-server.example.com/user1/busybox:sha256-
ff13b8f6f289b92ec2913fa57c5dd0a874c3a7f8f149aabee50e3d01546473e3.sig
```

If you experience the **error: signing quay-server.example.com/user1/busybox:test: getting remote image: GET https://quay-server.example.com/v2/user1/busybox/manifests/test: UNAUTHORIZED: access to the requested resource is not authorized; map[]** error, which occurs because Cosign relies on **~./docker/config.json** for authorization, you might need to execute the following command:

```
$ podman login --authfile ~/.docker/config.json quay.io
```

Example output

```
Username:
Password:
Login Succeeded!
```

4. Enter the following command to see the updated authorization configuration:

```
$ cat ~/.docker/config.json
{
 "auths": {
  "quay-server.example.com": {
   "auth": "cXVheWFkbWluOnBhc3N3b3Jk"
  }
 }
```