

Red Hat Quay 3.4

Deploy Red Hat Quay for proof-of-concept (non-production) purposes

Deploy Red Hat Quay

Last Updated: 2021-11-12

Red Hat Quay 3.4 Deploy Red Hat Quay for proof-of-concept (nonproduction) purposes

Deploy Red Hat Quay

Legal Notice

Copyright © 2021 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

http://creativecommons.org/licenses/by-sa/3.0/

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

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

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

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

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

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

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

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

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

All other trademarks are the property of their respective owners.

Abstract

Get started with Red Hat Quay

Table of Contents

PREFACE	. 4
CHAPTER 1. OVERVIEW	. 5 5
1.1.1. Internal components	5
1.1.2. External components	5
CHAPTER 2. GETTING STARTED WITH RED HAT QUAY	. 7
2.1. PREREQUISITES	7
2.1.1. Using podman	7
2.2. CONFIGURING THE RHEL SERVER	7
2.2.1. Install and register Red Hat Enterprise Linux server	7
2.2.2. Installing podman	8
2.2.3. Registry authentication	8
2.2.4. Firewall configuration	8
2.2.5. IP addressing and naming services	8
2.3. CONFIGURING THE DATABASE	9
2.3.1. Setting up Postgres	9
2.4. CONFIGURING REDIS	10
2.4.1. Setting up Redis	10
2.5. CONFIGURING RED HAT QUAY	10
2.5.1. Red Hat Quay setup	10
2.5.I.I. Basic configuration	10
2.5.1.2. Server configuration	11
2.5.1.3. Database	11
2.5.1.4. Reals	11
	11
2.61 Prorequisites	11
2.6.2. Prepare config folder	11
2.6.3. Prepare local storage for image data	12
2.6.4. Deploy the Red Hat Quay registry	12
2.7 USING RED HAT QUAY	12
271 Push and pull images	12
	12
CHAPTER 3. ADVANCED RED HAT QUAY DEPLOYMENT	14
3.1. QUAY SUPERUSER	14
3.1.1. Adding a superuser to Quay using the UI	14
3.1.2. Editing the config.yaml file to add a superuser	14
3.1.3. Accessing the superuser admin panel	15
3.2. DEPLOYING CLAIR V4	15
3.2.1. Deploying a separate database for Clair	15
3.2.2. Quay configuration for Clair	10
3.2.3. Clair configuration	1/
3.2.4. Ruining Clair	10
3.2.5. Using Clair security scanning	10
3.31 Using systemd unit files with Podman	19 10
3.3.2 Starting stopping and checking the status of services	2∩
3.3.3 Testing restart after reboot	20 2∩
3.3.4 Configuring Quay's dependency on Clair	20
o.o. i. configuring addy 5 dependency on oldin	ا ک

CHAPTER 4. NEXT STEPS	23

PREFACE

Red Hat Quay is an enterprise-quality registry for building, securing and serving container images. This procedure describes how to deploy Red Hat Quay for proof-of-concept (non-production) purposes.

CHAPTER 1. OVERVIEW

Features of Red Hat Quay include:

- High availability
- Geo-replication
- Repository mirroring
- Docker v2, schema 2 (multiarch) support
- Continuous integration
- Security scanning with Clair
- Custom log rotation
- Zero downtime garbage collection
- 24/7 support

Red Hat Quay provides support for:

- Multiple authentication and access methods
- Multiple storage backends
- Custom certificates for Quay, Clair, and storage backends
- Application registries
- Different container image types

1.1. ARCHITECTURE

Red Hat Quay consists of a number of core components, both internal and external.

1.1.1. Internal components

- Quay (container registry): Runs the quay container as a service, consisting of several components in the pod.
- Clair: Scans container images for vulnerabilities and suggests fixes.

1.1.2. External components

- Database: Used by Red Hat Quay as its primary metadata storage (not for image storage).
- Redis (key-value store): Stores live builder logs and the Red Hat Quay tutorial.
- **Cloud storage**: For supported deployments, you need to use one of the following types of storage:
 - **Public cloud storage**: In public cloud environments, you should use the cloud provider's object storage, such as Amazon S3 (for AWS) or Google Cloud Storage (for Google Cloud).

• **Private cloud storage**: In private clouds, an S3 or Swift compliant Object Store is needed, such as Ceph RADOS, or OpenStack Swift.



WARNING

Do not use "Locally mounted directory" Storage Engine for any production configurations. Mounted NFS volumes are not supported. Local storage is meant for Red Hat Quay test-only installations.

CHAPTER 2. GETTING STARTED WITH RED HAT QUAY

The Red Hat Quay registry can be deployed for non-production purposes on a single machine (either physical or virtual) with the following specifications.

2.1. PREREQUISITES

- Red Hat Enterprise Linux (RHEL) Obtain the latest Red Hat Enterprise Linux 8 server media from the Downloads page and follow the installation instructions available in the Product Documentation for Red Hat Enterprise Linux 8.
- Valid Red Hat Subscription Configure a valid Red Hat Enterprise Linux 8 server subscription.
- CPUs: Two or more virtual CPUs
- RAM: 4GB or more
- **Disk space**: The required disk space depends on the storage needs for the registry. Approximately 30GB of disk space should be enough for a test system, broken down as follows:
 - At least 10GB of disk space for the operating system (Red Hat Enterprise Linux Server).
 - At least 10GB of disk space for docker storage (to run 3 containers)
 - At least 10GB of disk space for Quay local storage (CEPH or other local storage might require more memory)

More information on sizing can be found at Quay 3.x Sizing Guidlines.

2.1.1. Using podman

This document uses **podman** for creating and deploying containers. If you do not have **podman** installed on your system, you should be able to use the equivalent **docker** commands. For more information on podman and related technologies, see Building, running, and managing Linux containers on Red Hat Enterprise Linux 8.

2.2. CONFIGURING THE RHEL SERVER

2.2.1. Install and register Red Hat Enterprise Linux server

Install the latest RHEL 8 server. You can do a minimal install (shell access only) or Server plus GUI (if you want a desktop). Register and subscribe your RHEL server system as described in How to register and subscribe a system.... The following commands register your system and list available subscriptions. Choose an available RHEL server subscription, attach to its pool ID and upgrade to the latest software:

- # subscription-manager register --username=<user_name> --password=<password>
- # subscription-manager refresh
- # subscription-manager list --available
- # subscription-manager attach --pool=<pool_id>
- # yum update -y

+

2.2.2. Installing podman

Install podman, if it is not already present on your system:

\$ sudo yum install -y podman

Alternatively, you can install the **container-tools** module, which pulls in the full set of container software packages:

\$ sudo yum module install -y container-tools

2.2.3. Registry authentication

Set up authentication to **registry.redhat.io**, so that you can pull the quay container, as described in Red Hat Container Registry Authentication. Note that this differs from earlier Red Hat Quay releases where the images were hosted on quay.io.

For example, you can log in to the registry:

\$ sudo podman login registry.redhat.io
Username: <username>
Password: <password>

2.2.4. Firewall configuration

If you have a firewall running on your system, to access the Red Hat Quay config tool (port 8443) and application (ports 8080 and 443) outside of the local system, run the following commands (add -- **zone=<yourzone>** for each command to open ports on a particular zone):

firewall-cmd --permanent --add-port=8443/tcp
firewall-cmd --permanent --add-port=8080/tcp
firewall-cmd --permanent --add-port=443/tcp
firewall-cmd --reload

2.2.5. IP addressing and naming services

There are a number of ways to configure the component containers in Red Hat Quay so that they can talk to each other:

• Using the IP addresses for the containers: You can determine the IP address for containers with **podman inspect** and then use these values in the configuration tool when specifying the connection strings, for example:

\$ sudo podman inspect -f "{{.NetworkSettings.IPAddress}}" postgresql-quay

This approach is susceptible to host restarts, as the IP addresses for the containers will change after a reboot.

- Using a naming service: If you want your deployment to survive container restarts, which typically result in changed IP addresses, you can implement a naming service. For example, the dnsname plugin is used to allow containers to resolve each other by name.
- Using the host network: You can use the podman run command with the --net=host option

and then use container ports on the host when specifying the addresses in the configuration. This option is susceptible to port conflicts when two containers want to use the same port, and as a result it is not recommended.

• **Configuring port mapping:** You can use port mappings to expose ports on the host and then use these ports in combination with the host IP address or host name.

This document uses port mapping in the subsequent examples, and assumes a static IP address for your host system. In this example, **quay-server** has the IP address **192.168.1.112**.

\$ cat /etc/hosts

192.168.1.112 quay-server

Component	Port mapping	Address
Quay	-р 8080:8080	http://quay- server:8080
Postgres for Quay	-р 5432:5432	quay-server:5432
Redis	-р 6379:6379	quay-server:6379
Postgres for Clair V4	-р 5433:5432	quay-server:5433
Clair V4	-p 8081:8080	http://quay-server:8081

2.3. CONFIGURING THE DATABASE

Quay requires a database for storing metadata and Postgres is recommended, especially for highly available configurations. Alternatively, you can use MySQL with a similar approach to configuration as described below for Postgres.

2.3.1. Setting up Postgres

In this proof-of-concept scenario, you will use a directory on the local file system to persist database data.

• In the installation folder, denoted here by the variable \$QUAY, create a directory for the database data and set the permissions appropriately:

\$ mkdir -p \$QUAY/postgres-quay \$ setfacl -m u:26:-wx \$QUAY/postgres-quay

• Use podman to run the Postgres container, specifying the username, password, database name and port, together with the volume definition for database data:

\$ sudo podman run -d --rm --name postgresql-quay \
 -e POSTGRESQL_USER=quayuser \
 -e POSTGRESQL_PASSWORD=quaypass \
 -e POSTGRESQL_DATABASE=quay \

-e POSTGRESQL_ADMIN_PASSWORD=adminpass \
-p 5432:5432 \
-v \$QUAY/postgres-quay:/var/lib/pgsql/data:Z \
registry.redhat.io/rhel8/postgresql-10:1

• Ensure that the Postgres **pg_trgm** module is installed, as it is required by Quay:

\$ sudo podman exec -it postgresql-quay /bin/bash -c 'echo "CREATE EXTENSION IF NOT EXISTS pg_trgm" | psql -d quay -U postgres'

2.4. CONFIGURING REDIS

Redis ia a key-value store, used by Quay for live builder logs and the Red Hat Quay tutorial.

2.4.1. Setting up Redis

Use podman to run the Redis container, specifying the port and password:

```
$ sudo podman run -d --rm --name redis \
-p 6379:6379 \
-e REDIS_PASSWORD=strongpassword \
registry.redhat.io/rhel8/redis-5:1
```

2.5. CONFIGURING RED HAT QUAY

Before running the Red Hat Quay service, you need to generate a configuration file containing details of all the components, including registry settings, and database and Redis connection parameters. To generate the configuration file, you run the quay container in **config** mode, specifying a password (in this instance, **secret**) for the **quayconfig** user:

\$ sudo podman run --rm -it --name quay_config -p 8080:8080 registry.redhat.io/quay/quay-rhel8:v3.4.7 config secret

Use your browser to access the user interface for the configuration tool at **http://quay-server:8080** (assuming you have configured the **quay-server** hostname in your **hosts** file). Login with the username **quayconfig** and password **secret** (or whatever value you specified in the podman run command above).

2.5.1. Red Hat Quay setup

In the configuration editor, you enter details for the following:

- Basic configuration
- Server configuration
- Database
- Redis

2.5.1.1. Basic configuration

In the basic configuration setting, complete the registry title and the registry short title fields (or you can use the default values, if they are specified).

2.5.1.2. Server configuration

Specify the HTTP host and port, for the location where the registry will be accessible on the network, in this instance, **quay-server:8080**.

2.5.1.3. Database

In the database section, specify connection details for the database that Red Hat Quay uses to store metadata. If you have followed the instructions in this document for deploying a proof-of-concept system, the following values would be entered:

- Database Type: Postgres
- Database Server: quay-server:5432
- Username: quayuser
- **Password:** quaypass
- Database Name: quay

2.5.1.4. Redis

The Redis key-value store is used to store real-time events and build logs. If you have followed the instructions in this document for deploying a proof-of-concept system, the following values would be specified:

- Redis Hostname: quay-server
- Redis port: 6379 (default)

2.5.2. Validate and download configuration

When all required fields have been set, validate your settings by choosing the Validate Configuration Changes button. If any errors are reported, continue editing your configuration until all required fields are valid and Red Hat Quay can connect to your database and Redis servers.

Once your configuration is valid, download the configuration file and then stop the quay container that is running the configuration editor.

2.6. DEPLOYING RED HAT QUAY

2.6.1. Prerequisites

- Your Quay database and Redis servers are running.
- You have generated a valid configuration bundle.
- You have stopped the Quay container that you used to run the configuration editor.

2.6.2. Prepare config folder

Unpack the configuration bundle so that Quay can use it, for example:

- \$ mkdir \$QUAY/config \$ cp ~/Downloads/quay-config.tar.gz \$QUAY/config \$ cd \$QUAY/config \$ tar wit quay config tar az
- \$ tar xvf quay-config.tar.gz

2.6.3. Prepare local storage for image data

In this proof-of-concept deployment, use the local file system to store the registry images:

\$ mkdir \$QUAY/storage \$ setfacl -m u:1001:-wx \$QUAY/storage

2.6.4. Deploy the Red Hat Quay registry

Use podman to run the quay container, specifying the appropriate volumes for your configuration data and local storage for image data:

```
$ sudo podman run -d --rm -p 8080:8080 \
--name=quay \
-v $QUAY/config:/conf/stack:Z \
-v $QUAY/storage:/datastorage:Z \
registry.redhat.io/quay/quay-rhel8:v3.4.7
```

2.7. USING RED HAT QUAY

Use your browser to access the user interface for the Red Hat Quay registry at **quay-server:8080** (assuming you have configured the **quay-server** hostname in your **hosts** file). Select 'Create User' and add a user, for example, **quayadmin** with a password **password**.

You can now use the user interface to create new organizations and repositories, and to search and browse existing repositories. Alternatively, you can use the command line interface to interact with the registry and to pull and push images.

From the command line, log in to the registry:

\$ sudo podman logintls-verify=false quay-server:8080
Username: quayadmin
Password:
Login Succeeded!

2.7.1. Push and pull images

To test pushing and pulling images from the Red Hat Quay registry, first pull a sample image from an external registry:

\$ sudo podman pull busybox 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

Use the **podman images** command to see the local copy:

\$ sudo podman imagesREPOSITORYTAGIMAGE IDCREATEDSIZEdocker.io/library/busyboxlatest22667f53682a14 hours ago1.45 MB...

Tag this image, in preparation for pushing it to the Red Hat Quay registry:

\$ sudo podman tag docker.io/library/busybox quay-server:8080/quayadmin/busybox:test

Now push the image to the Red Hat Quay registry:

\$ sudo podman push --tls-verify=false quay-server:8080/quayadmin/busybox:test Getting image source signatures Copying blob 6b245f040973 done Copying config 22667f5368 done Writing manifest to image destination Storing signatures

At this point, you can use your browser to see the tagged image in your repository. To test access to the image from the command line, first delete the local copy of the image:

\$ sudo podman rmi quay-server:8080/quayadmin/busybox:test Untagged: quay-server:8080/quayadmin/busybox:test

Now pull the image again, this time from your Red Hat Quay registry:

\$ sudo podman pull --tls-verify=false quay-server:8080/quayadmin/busybox:test Trying to pull quay-server:8080/quayadmin/busybox:test... Getting image source signatures Copying blob 6ef22a7134ba [------] 0.0b / 0.0b Copying config 22667f5368 done Writing manifest to image destination Storing signatures 22667f53682a2920948d19c7133ab1c9c3f745805c14125859d20cede07f11f9

CHAPTER 3. ADVANCED RED HAT QUAY DEPLOYMENT

3.1. QUAY SUPERUSER

A **superuser** is a Quay user account that has extended privileges, including the ability to:

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

3.1.1. Adding a superuser to Quay using the UI

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

```
$ sudo podman run --rm -it --name quay_config \
    -p 8080:8080 \
    -v $QUAY/config:/conf/stack:Z \
    registry.redhat.io/quay/quay-rhel8:v3.4.7 config secret
```

In the Access Settings section of the UI, enter the name of the user (in this instance, **quayadmin**) in the Super Users field and press Add.

O Access Settings					
Various settings around access and	I authentication to the registry.				
Basic Credentials Login:	Login to User Interface via credentials is enabled (requires at least one OIDC provider to disable) If enabled, users will be able to login to the user Interface via their username and password credentials. If disabled, users will only be able to login to the user Interface via one of the configured External Authentication providers.				
External Application tokens	Allow external application tokens				
	If enabled, users will be able to generate external application tokens for use on the Docker and rkt CLL Note that these tokens will not be required unless "App Token" is chosen as the Internal Authentication method above.				
External application token expiration	The expiration time for user generated external application tokens. If none, tokens will never expire,				
Anonymous Access:	C Enable Anonymous Access				
	If enabled, public repositories and search can be accessed by anyone that can reach the registry, even if they are not authenticated. Disable to only allow authenticated users to view and pull "public" resources.				
User Creation:	Z Enable Non-Superuser User Creation				
	If enabled, user accounts can be created by anyone (unless restricted below to invited users). Users can always be created in the users panel in this superuser tool, even if this feature is disabled. If disabled, users can ONLY be created in the superuser tool or via team sync.				
Encrypted Client Password:	Require Encrypted Client Passwords				
	If enabled, users will not be able to login from the Docker command line with a non-encrypted password and must generate an encrypted password to use.				
Prefix username	Allow prefix username autocompletion				
autocompletion:	If disabled, autocompletion for users will only match on exact usernames.				
Super Users:	No Super Users defined				
	suayadmid Add				
	Users included in this list will be given elevated access to Quay				

Validate and download the configuration bundle and then terminate the Quay container that is running in config mode. Extract the **config.yaml** file to the configuration directory and restart the Quay container in registry mode.

3.1.2. Editing the config.yaml file to add a superuser

You can also add a superuser by editing the **config.yaml** file directly. The list of superuser accounts is stored as an array in the field **SUPER_USERS**:

\$QUAY/config/config.yaml

```
SERVER_HOSTNAME: quay-server:8080
SETUP_COMPLETE: true
SUPER_USERS:
- quayadmin
...
```

3.1.3. Accessing the superuser admin panel

To access the Super User Admin Panel, click on the current user's name or avatar in the top right-hand corner of the UI. If the user has been added as a superuser, an extra item is presented in the drop-down list called Super User Admin Panel.

🔞 RED HAT" QUAY	EXPLORE	REPOSITORIES	TUTORIAL			search Q 🕂 -	duayad
				Repositories		+ Create New Ro	Account Settings Super User Admin Panel
				1-1of1 < > Filter	r Repositories	Users and Organizations	Sign out all sessions
REPOSITORY NAME			LAST MODIFIED	ACTIVITY 1	STAR	Q quayadmin	
Q quayadmin / ubuntu			02/23/2021	all		+ Create New Organization	

3.2. DEPLOYING CLAIR V4

Clair is an application for parsing image contents and reporting vulnerabilities affecting the contents. This is performed via static analysis and not at runtime. Clair's analysis is broken into three distinct parts:

- Indexing: Indexing starts with submitting a Manifest to Clair. On receipt, Clair will fetch layers, scan their contents, and return an intermediate representation called an IndexReport. Manifests are Clair's representation of a container image. Clair leverages the fact that OCI Manifests and Layers are content-addressed to reduce duplicated work. Once a Manifest is indexed, the IndexReport is persisted for later retrieval.
- **Matching:** Matching takes an IndexReport and correlates vulnerabilities affecting the manifest that the report represents. Clair is continually ingesting new security data and a request to the matcher will always provide you with the most up to date vulnerability analysis of an IndexReport.
- **Notifications:** Clair implements a notification service. When new vulnerabilities are discovered, the notifier service will determine if these vulnerabilities affect any indexed Manifests. The notifier will then take action according to its configuration.

3.2.1. Deploying a separate database for Clair

Clair requires a Postgres database. You can share a common database between Quay and Clair if Quay is also using Postgres, but in this example a separate, Clair-specific database is deployed.

In this proof-of-concept scenario, you will use a directory on the local file system to persist database data.

• In the installation folder, denoted here by the variable \$QUAY, create a directory for the Clair database data and set the permissions appropriately:

\$ mkdir -p \$QUAY/postgres-clairv4 \$ setfacl -m u:26:-wx \$QUAY/postgres-clairv4

 Use podman to run the Postgres container, specifying the username, password, database name and port, together with the volume definition for database data. As the standard Postgres port, 5432, is already in use by the Quay deployment, expose a different port, in this instance 5433:

\$ sudo podman run -d --rm --name postgresql-clairv4 \
 -e POSTGRESQL_USER=clairuser \
 -e POSTGRESQL_PASSWORD=clairpass \
 -e POSTGRESQL_DATABASE=clair \
 -e POSTGRESQL_ADMIN_PASSWORD=adminpass \
 -p 5433:5432 \
 -v \$QUAY/postgres-clairv4:/var/lib/pgsql/data:Z \
 registry.redhat.io/rhel8/postgresql-10:1

• Ensure that the Postgres **uuid-ossp** module is installed, as it is required by Clair:

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

3.2.2. Quay configuration for Clair

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

\$ sudo podman run --rm -it --name quay_config \
 -p 8080:8080 \
 -v \$QUAY/config:/conf/stack:Z \
 registry.redhat.io/quay/quay-rhel8:v3.4.7 config secret

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

- Security Scanner Endpoint: http://quay-server:8081
- Security Scanner PSK: MTU5YzA4Y2ZkNzJoMQ==

The UI for setting the scanner data is shown in the following image:

Security Scanner UI

🟦 Security Scanner						
If enabled, all images pushed to Quay will be scanned via the external security scanning service, with vulnerability information available in the UI and API, as well as async notification support.						
Enable Security Scanning						
i A scanner compliant with	A scanner compliant with the Quay Security Scanning API must be running to use this feature. Documentation on running Clair can be found at Running Clair Security Scanner.					
Security Scanner Endpoint:	http://quay-server:8081					
	The HTTP URL at which the security scanner is running.					
Security Scanner PSK:	MTU5YzA4Y2ZkNzJoMQ==	Generate PSK				
	Clair Pre-Shared Key. Make sure to include this value in your Clair conf	íg.				

Validate and download the configuration and then stop the Quay container that is running the configuration editor. Extract the configuration bundle as before into the **\$QUAY/config** directory.

\$ cp ~/Downloads/quay-config.tar.gz \$QUAY/config

- \$ cd \$QUAY/config
- \$ tar xvf quay-config.tar.gz

The Quay configuration file has been updated to contain the fields for the security scanner:

\$QUAY/config/config.yaml

FEATURE_SECURITY_NOTIFICATIONS: false FEATURE_SECURITY_SCANNER: true ... SECURITY_SCANNER_INDEXING_INTERVAL: 30 SECURITY_SCANNER_V4_ENDPOINT: http://quay-server:8081 SECURITY_SCANNER_V4_PSK: MTU5YzA4Y2ZkNzJoMQ== SERVER_HOSTNAME: quay-server:8080 ...

3.2.3. Clair configuration

Detailed information on Clair configuration is available at https://github.com/quay/clair/blob/main/Documentation/reference/config.md. The following example provides a minimal configuration for use in a proof of concept deployment:

/etc/clairv4/config/config.yaml

http_listen_addr: :8081 introspection_addr: :8089 log level: debug indexer: connstring: host=quay-server port=5433 dbname=clair user=clairuser password=clairpass sslmode=disable scanlock retry: 10 layer scan concurrency: 5 migrations: true matcher: connstring: host=quay-server port=5433 dbname=clair user=clairuser password=clairpass sslmode=disable max_conn_pool: 100 run: "" migrations: true indexer addr: clair-indexer notifier: connstring: host=quay-server port=5433 dbname=clair user=clairuser password=clairpass sslmode=disable delivery_interval: 1m poll interval: 5m migrations: true auth: psk: key: "MTU5YzA4Y2ZkNzJoMQ=="

```
iss: ["quay"]
# tracing and metrics
trace:
name: "jaeger"
probability: 1
jaeger:
   agent_endpoint: "localhost:6831"
   service_name: "clair"
metrics:
   name: "prometheus"
```

- http_listen_addr is set to the port of the Clair HTTP endpoint that you specified in the Quay configuration tool, in this case :8081.
- The Clair pre-shared key (PSK) that you generated in the Quay configuration tool is used for authentication, with the issuer, specified in the **iss** field, set to **quay**.

3.2.4. Running Clair

Use the **podman run** command to run the Clair container, exposing the HTTP endpoint port that you specified in the configuration tool, in this case **8081**:

```
sudo podman run -d --rm --name clairv4 \
-p 8081:8081 -p 8089:8089 \
-e CLAIR_CONF=/clair/config.yaml -e CLAIR_MODE=combo \
-v /etc/clairv4/config:/clair:Z \
registry.redhat.io/quay/clair-rhel8:v3.4.7
```

Now restart the Quay container, using the updated configuration file containing the scanner settings:

```
$ sudo podman run -d --rm -p 8080:8080 \
--name=quay \
-v $QUAY/config:/conf/stack:Z \
-v $QUAY/storage:/datastorage:Z \
registry.redhat.io/quay/quay-rhel8:v3.4.7
```

3.2.5. Using Clair security scanning

From the command line, log in to the registry:

```
$ sudo podman login --tls-verify=false quay-server:8080
Username: quayadmin
Password:
Login Succeeded!
```

Pull, tag and push a sample image to the registry:

```
$ sudo podman pull ubuntu:20.04
$ sudo podman tag docker.io/library/ubuntu:20.04 quay-server:8080/quayadmin/ubuntu:20.04
$ sudo podman push --tls-verify=false quay-server:8080/quayadmin/ubuntu:20.04
```

The results from the security scanning can be seen in the Quay UI, as shown in the following images:

Scanning summary

🔞 RED H	AT QUAY EXPLORE REPOSITORIES TUTORIAL		search	Q +- 🜲 Q quayad
← Repos	tories	🗠 quayadmin / ubuntu 🖒		
0	Repository Tags			Compact Expanded
۲			1 - 1 of 1	< > Filter Tags
	TAG	LAST MODIFIED 1 SECURITY SCAN	SIZE EXPIRES	MANIFEST
Ð	20.04	a minute ago	28.6 MB Never	SHA256 bZcSc6eebc86 🚨 👷
•				

Scanning details

🔞 RED H	IAT QUAY EXPLORE REPOSITORIES TU	ITORIAL				search Q +- 🖡 Q quayad
🗲 🖯 quay	yadmin/ubuntu		b2c50	c6eebc06		
۲ بَرَ ا	22% 5% Quay Sect A 2 Me A 27 Lo A 8 Neg	urity Scanner has de dium-level vulnerabilities. v-level vulnerabilities. gligible-level vulnerabilities	etected 37 vulnerabil	ities.		
	Vulnerabilities					Filter Vulnerabilities
	CVE	SEVERITY ↓	PACKAGE	CURRENT VERSION	FIXED IN VERSION	INTRODUCED IN LAYER
	▹ CVE-2018-20839 on Ubuntu 20.04 (focal) - medium. %	A Medium	libsystemd0	245.4-4ubuntu3.4	(None)	ADD file:2a90223d9f00d3le31eff6b207c57af4b7d
	> CVE-2018-20839 on Ubuntu 20.04 (focal) - medium. %	A Medium	libudev1	245.4-4ubuntu3.4	(None)	ADD file:2a90223d9f00d31e31eff6b207c57af4b7d
	VE-2019-18276 on Ubuntu 20.04 (focal) - low. %	A Low	bash	5.0-6ubuntu1.1	(None)	ADD file:2a90223d9f00d31e31eff6b207c57af4b7d
	VE-2017-18018 on Ubuntu 20.04 (focal) - low. %	A Low	coreutils	8.30-3ubuntu2	(None)	ADD file:2a90223d9f00d31e31eff6b207c57af4b7d
	VE-2016-2781 on Ubuntu 20.04 (focal) - Iow. %	A Low	coreutils	8.30-3ubuntu2	(None)	ADD file:2a90223d9f00d31e31eff6b207c57af4b7d
	VE-2019-13050 on Ubuntu 20.04 (focal) - low. %	A Low	gpgv	2.2.19-3ubuntu2	(None)	ADD file:2a90223d9f00d31e31eff6b207c57af4b7d
	VE-2019-25013 on Ubuntu 20.04 (focal) - low. %	A Low	libc-bin	2.31-0ubuntu9.1	(None)	ADD file:2a90223d9f00d31e31eff6b207c57af4b7d
	VE-2020-27618 on Ubuntu 20.04 (focal) - low. %	A Low	libc-bin	2.31-0ubuntu9.1	(None)	ADD file:2a90223d9f00d31e31eff6b207c57af4b7d

3.3. RESTARTING CONTAINERS

Because the **--restart** option is not fully supported by podman, you can configure **podman** as a systemd service, as described in Porting containers to systemd using Podman

3.3.1. Using systemd unit files with Podman

By default, Podman generates a unit file for existing containers or pods. You can generate more portable systemd unit files using the **podman generate systemd --new** command. The **--new** flag instructs Podman to generate unit files that create, start and remove containers.

• Create the systemd unit files from a running Red Hat Quay registry as follows:

\$ sudo podman generate systemd --new --files --name redis
\$ sudo podman generate systemd --new --files --name postgresql-quay
\$ sudo podman generate systemd --new --files --name quay
\$ sudo podman generate systemd --new --files --name postgresql-clairv4
\$ sudo podman generate systemd --new --files --name clairv4

• Copy the unit files to /usr/lib/systemd/system for installing them as a root user:

\$ sudo cp -Z container-redis.service /usr/lib/systemd/system
\$ sudo cp -Z container-postgresql-quay.service /usr/lib/systemd/system
\$ sudo cp -Z container-quay.service /usr/lib/systemd/system
\$ sudo cp -Z container-postgresql-clairv4.service /usr/lib/systemd/system

- \$ sudo cp -Z container-clairv4.service /usr/lib/systemd/system
- Reload systemd manager configuration:

\$ sudo systemctl daemon-reload

• Enable the services and start them at boot time:

\$ sudo systemctl enable --now container-redis.service
\$ sudo systemctl enable --now container-postgresql-quay.service
\$ sudo systemctl enable --now container-quay.service
\$ sudo systemctl enable --now container-postgresql-clairv4.service
\$ sudo systemctl enable --now container-clairv4.service

3.3.2. Starting, stopping and checking the status of services

• Check the status of the Quay components:

\$ sudo systemctl status container-redis.service
\$ sudo systemctl status container-postgresql-quay.service
\$ sudo systemctl status container-quay.service
\$ sudo systemctl status container-postgresql-clairv4.service
\$ sudo systemctl status container-clairv4.service

• To stop the Quay component services:

\$ sudo systemctl stop container-redis.service
\$ sudo systemctl stop container-postgresql-quay.service
\$ sudo systemctl stop container-quay.service
\$ sudo systemctl stop container-postgresql-clairv4.service
\$ sudo systemctl stop container-clairv4.service

• To start the Quay component services:

\$ sudo systemctl start container-redis.service
\$ sudo systemctl start container-postgresql-quay.service
\$ sudo systemctl start container-quay.service
\$ sudo systemctl start container-postgresql-clairv4.service
\$ sudo systemctl start container-clairv4.service

3.3.3. Testing restart after reboot

Once you have the services configured and enabled, reboot the system. When the system has restarted, use **podman ps** to check that all the containers for the Quay components have been restarted:

\$ sudo podman ps -a CONTAINER ID IMAGE PORTS NAMES

COMMAND	CREATED	STATUS

4e87c7889246 registry.redhat.io/rhel8/postgresql-10:1 run-postgresql 19 seconds ago Up 18 seconds ago 0.0.0.0:5432->5432/tcp postgresql-quay b8fbac1920d4 registry.redhat.io/rhel8/redis-5:1 run-redis 19 seconds ago Up 18 seconds ago 0.0.0.0:6379->6379/tcp redis d959d5bf7a24 registry.redhat.io/rhel8/postgresql-10:1 run-postgresql 18 seconds ago Up 18 seconds ago 0.0.0.0:5433->5432/tcp postgresql-10:1 run-postgresql 18 seconds ago Up 18 seconds ago 0.0.0.0:5433->5432/tcp postgresql-clairv4 e75ff8651dbd registry.redhat.io/quay/clair-rhel8:v3.4.0 18 seconds ago Up 17 seconds ago 0.0.0.0:8081->8080/tcp clairv4

In this instance, the Quay container itself has failed to start up. This is due to the fact that, when security scanning is enabled in Quay, it tries to connect to Clair on startup. However, Clair has not finished initializing and cannot accept connections and, as a result, Quay terminates immediately. To overcome this issue, you need to configure the Quay service to have a dependency on the Clair service, as shown in the following section.

3.3.4. Configuring Quay's dependency on Clair

In the **systemd** service file for Quay, set up a dependency on the Clair service in the **[Unit]** section by setting **After=container-clairv4.service**. To give the Clair container time to initialize, add a delay in the **[Service]** section, for example **RestartSec=30**. Here is an example of the modified Quay file, after configuring the dependency on Clair:

/usr/lib/systemd/system/container-quay.service

container-quay.service # autogenerated by Podman 2.0.5 # Tue Feb 16 17:02:26 GMT 2021 [Unit] Description=Podman container-quay.service Documentation=man:podman-generate-systemd(1) Wants=network.target After=container-clairv4.service [Service] Environment=PODMAN SYSTEMD UNIT=%n Restart=on-failure RestartSec=30 ExecStartPre=/bin/rm -f %t/container-quay.pid %t/container-quay.ctr-id ExecStart=/usr/bin/podman run --conmon-pidfile %t/container-quay.pid --cidfile %t/containerquay.ctr-id --cgroups=no-conmon -d --rm -p 8080:8080 --name=quay -v /home/user1/quay/config:/conf/stack:Z -v /home/user1/quay/storage:/datastorage:Z registry.redhat.io/quay/quay-rhel8:v3.4.0 ExecStop=/usr/bin/podman stop --ignore --cidfile %t/container-quay.ctr-id -t 10 ExecStopPost=/usr/bin/podman rm --ignore -f --cidfile %t/container-quay.ctr-id PIDFile=%t/container-quay.pid KillMode=none Type=forking [Install]

WantedBy=multi-user.target default.target

Once you have updated the Quay service configuration, reboot the server and immediately run **podman ps**:

\$ sudo podman ps -a CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES 4e87c7889246 registry.redhat.io/rhel8/postgresql-10:1 run-postgresql 29 seconds ago Up 28 seconds ago 0.0.0.0:5432->5432/tcp postgresgl-quay b8fbac1920d4 registry.redhat.io/rhel8/redis-5:1 run-redis 29 seconds ago Up 28 seconds ago 0.0.0.0:6379->6379/tcp redis d959d5bf7a24 registry.redhat.io/rhel8/postgresql-10:1 run-postgresql 28 seconds ago Up 28 seconds ago 0.0.0.0:5433->5432/tcp postgresgl-clairv4 e75ff8651dbd registry.redhat.io/quay/clair-rhel8:v3.4.0 28 seconds ago Up 27 seconds ago 0.0.0.0:8081->8080/tcp clairv4

Initially, the Quay container will not be available, but once the **RestartSec** delay has expired, it should start up:

\$ sudo podman ps -a CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES 4e87c7889246 registry.redhat.io/rhel8/postgresql-10:1 run-postgresql 35 seconds ago Up 34 seconds ago 0.0.0.0:5432->5432/tcp postgresgl-quay ab9f0e6ad7c3 registry.redhat.io/quay/quay-rhel8:v3.4.0 registry 3 seconds ago Up 2 seconds ago 0.0.0.0:8080->8080/tcp quay b8fbac1920d4 registry.redhat.io/rhel8/redis-5:1 run-redis 35 seconds ago Up 34 seconds ago 0.0.0.0:6379->6379/tcp redis d959d5bf7a24 registry.redhat.io/rhel8/postgresgl-10:1 run-postgresgl 34 seconds ago Up 34 seconds ago 0.0.0.0:5433->5432/tcp postgresgl-clairv4 e75ff8651dbd registry.redhat.io/guay/clair-rhel8:v3.4.0 34 seconds ago Up 33 seconds ago 0.0.0.0:8081->8080/tcp clairv4

The **CREATED** field for the quay container shows the 30 second difference in creation time, as configured in the service definition.

Log in to the Red Hat Quay registry at **quay-server:8080** to check that everything has restarted correctly.

CHAPTER 4. NEXT STEPS

This document shows how to configure and deploy a proof-of-concept version of Red Hat Quay. For more information on deploying to a production environment, see the guide "Deploy Red Hat Quay - High Availability".

The "Use Red Hat Quay" guide shows you how to:

- Add users and repositories
- Use tags
- Automatically build Dockerfiles with build workers
- Set up build triggers
- Add notifications for repository events

The "Manage Red Hat Quay" guide shows you how to:

- Use SSL and TLS
- Enable security scanning with Clair
- Use repository mirroring
- Configure LDAP authentication
- Use georeplication of storage