



# Red Hat Training and Certification

Managing and Building Container  
Images and Containers

Travis Michette

Version 1.2

# Table of Contents

Introduction .....	1
Lab Machine Requirements .....	1
Using the Lab Guide .....	2
1. RHEL 8 Changes .....	3
1.1. Setup Exercise .....	3
1.2. TMUX Usage .....	6
1.3. SystemD Overview .....	9
1.3.1. Understanding SystemD Unit Files and Creating a Service .....	9
1.4. FirewallD .....	14
1.4.1. FirewallD Service Definitions .....	14
1.4.2. The <b>firewall-cmd</b> Utility .....	15
1.4.3. FirewallD Files and Locations .....	16
1.4.4. Defining a Custom Service File .....	17
1.4.5. FirewallD Configuration Files .....	18
1.5. Cockpit .....	21
1.5.1. Installing Additional Cockpit Packages .....	21
1.5.1.1. Cockpit to Manage Multiple Systems .....	22
2. Managing Containers with the New Runtime .....	26
2.1. Deploying Containers with the New Container Runtime .....	26
2.1.1. The Podman Container Engine .....	26
2.2. Podman Configuration Files .....	28
2.2.1. Root-Based Podman .....	28
2.2.2. Rootless Podman .....	29
2.3. Container Image Storage .....	31
2.3.1. Root-Based Podman .....	31
2.3.2. Rootless Podman .....	31
2.4. Container Networking .....	32
2.4.1. Root-Based Podman .....	32
2.4.2. Rootless Podman .....	34
2.5. Managing Containers using the Red Hat Web Console .....	34
2.5.1. Installing Cockpit Podman Plugins .....	35
2.5.2. Using Cockpit to Manage Containers .....	37
2.5.2.1. Using Cockpit to Manage Containers (Rootless) .....	49
3. Podman Pods and Container/Pod Resource Files .....	59
3.1. Using and Leveraging Pods with Podman .....	59
3.1.1. Creating a Pod and Adding a Container .....	61

3.1.2. Cleaning up Pods and Containers .....	62
3.2. Creating a multi-container Pod for Wordpress .....	64
3.3. Podman Image and Container Pruning .....	69
3.3.1. Using Podman Prune Commands .....	69
3.3.1.1. Cleaning Up Pods .....	69
3.3.1.2. Cleaning Up Containers .....	70
3.3.1.3. Cleaning Up Images .....	71
3.4. Using Podman to Create YAML Resource Files .....	74
3.4.1. Creating a Deployment File from a Container .....	75
3.4.2. Creating a Deployment file from a Pod .....	79
4. Container Images .....	88
4.1. Introduction to Buildah .....	88
4.1.1. Buildah Commands and Overview .....	88
4.2. Building Containers with Buildah .....	92
4.2.1. Building images as the <b>root</b> user .....	92
4.2.2. Building an Image Using Buildah Rootless .....	98
4.3. Managing Images and System Storage .....	101
4.3.1. Pushing Images to an Image Registry with Buildah .....	101
4.3.2. Deleting Images from the local image registry with Buildah .....	102
5. Managing Containers as System Services .....	105
5.1. Starting Containers Automatically with the Server .....	105
5.2. Running <b>Systemd</b> Services as a Regular User .....	105
5.2.1. Creating and Managing Systemd User Services .....	105
5.2.2. Managing Containers Using <b>Systemd</b> Services .....	106
5.2.2.1. Creating a Dedicated User Account to Run Containers .....	106
5.2.2.2. Creating the Systemd Unit File .....	106
5.2.2.3. Starting and Stopping Containers Using Systemd .....	107
5.2.2.4. Configuring Containers to Start When the Host Machine Starts .....	107
5.2.3. Managing Containers Running as Root with Systemd .....	107
5.2.4. Orchestrating Containers at Scale .....	108
5.3. DEMO - Managing Containers as Services .....	108
6. Container Management with Ansible .....	115
6.1. Ansible Refresher .....	115
6.1.1. Ansible Basics .....	115
6.1.1.1. Ansible Concepts and Architecture .....	115
6.1.1.2. Building an Ansible Inventory .....	116
6.1.1.2.1. Static Inventory .....	116
6.1.1.3. Ansible Configuration Files .....	117

6.2. Ansible Podman Collection .....	118
6.2.1. Obtaining Podman Collections .....	118
6.2.2. Installing and Using the <b>containers.podman</b> Collection .....	119
6.3. Building Container Images Using Ansible .....	122
6.4. Deploying Podman Containers Using Ansible .....	126
6.4.1. Analyzing Running Containers with Ansible .....	131
6.4.2. Controlling Running Containers with Ansible .....	134
7. Quay Image Registry .....	139
7.1. Installing the Quay Image Registry Manually .....	140
7.1.1. Preparing the Server for Quay and Running Supporting Containers .....	141
7.1.2. Preparing Quay and Clair Containers .....	150
7.1.2.1. Starting Quay and the Mirroring Containers .....	154
7.2. Installing the Quay Image Registry with Ansible .....	156
7.2.1. Deploying Quay with Ansible .....	157
7.3. Setting up the Quay Web Console and Testing Quay .....	161
7.3.1. Configuring the Quay Super User .....	161
7.3.2. Testing Quay and ClairV4 Image Scanning .....	166
7.3.3. Testing Repository Mirroring .....	169
7.3.4. Using the Quay Image Registry .....	174
7.3.4.1. Configuring Quay as the <b>Superuser</b> .....	174
7.3.4.2. Creating Organizations .....	177
7.3.4.3. Testing Organizations .....	182
7.4. Inspecting Images with Skopeo on Remote Registries .....	188
7.4.1. Using Skopeo to Inspect Images and Copy Images .....	188
Appendix A: Running Containers .....	195
A.1. Introducing Containers .....	195
A.1.1. Introducing Container Technology .....	195
A.1.1.1. Comparing Containers to Virtual Machines .....	195
A.1.1.2. Exploring the Implementation of Containers .....	196
A.1.1.3. Planning for Containers .....	196
A.1.1.4. Running Containers from Container Images .....	196
A.1.1.5. Designing Container-based Architectures .....	196
A.1.2. Managing Containers with Podman .....	196
A.1.2.1. Running Rootless Containers .....	197
A.1.3. Managing Containers at Scale .....	197
A.2. Running a Basic Container .....	197
A.2.1. Installing Container Management Tools .....	197
A.2.2. Selecting Container Images and Registries .....	197

A.2.2.1. Container Naming Conventions .....	198
A.2.3. Running Containers .....	198
A.2.4. Analyzing Container Isolation .....	200
A.2.5. DEMO - Running a Basic Container .....	200
A.3. Finding and Managing Container Images .....	203
A.3.1. Configuring Container Registries .....	204
A.3.1.1. Registry Security .....	204
A.3.2. Finding Container Images .....	205
A.3.2.1. Using the Red Hat Container Catalog .....	205
A.3.3. Inspecting Container Images .....	206
A.3.4. Removing Local Container Images .....	206
A.3.5. DEMO - Finding and Managing Container Images .....	207
A.4. Performing Advanced Container Management .....	210
A.4.1. Administering Containers with Podman .....	210
A.4.2. Configuring Containers .....	210
A.4.2.1. Mapping Container Host Ports to the Container .....	210
A.4.2.2. Passing Environment Variables to Configure a Container .....	211
A.4.3. Managing Containers .....	212
A.4.4. Running Commands in a Container .....	213
A.4.5. DEMO - Performing Advanced Container Management .....	213
A.5. Attaching Persistent Storage to a Container .....	217
A.5.1. Preparing Permanent Storage Locations .....	217
A.5.2. Providing Persistent Storage from the Container Host .....	217
A.5.2.1. Preparing the Host Directory .....	217
A.5.2.2. Mounting a Volume .....	217
A.5.3. DEMO - Attaching Persistent Storage to a Container .....	218
Appendix B: System Security Policy and Compliance .....	219
B.1. Customizing SCAP Content .....	219
B.2. Running a SCAP Scan with Custom Content .....	231
B.3. Creating an Ansible Remediation Playbook Based on SCAP Scan Results .....	241

# Introduction

This guide will cover additional and supplemental materials for the DO180 custom container course. As part of this guide, students will be learning the following concepts:

## Course Objectives

- Exploring RHEL 8.x Differences
- Exploring SystemD and Creating SystemD Services
- Exploring FirewallD and FirewallD Custom Services
- Exploring Cockpit and the Red Hat Web Console
- Rootless Podman
- Leveraging Podman's Pod Capabilities
- Managing Containers with the Red Hat Web Console (Cockpit)
- Running Containers as a Service (SystemD)
- Building Container Images with Buildah (from scratch)
- Building Container Images from Containerfile/Dockerfile Files
- Installing/Configuring/Using Quay Container Registry
- Using ClairV4 and Quay Mirroring with Quay
- Exploring Skopeo to Interact with Container Images

## Courses for Reference

- DO180
- RH134
- RH354

This course will use a DO180 course for the hands-on lab environment. The environment has been modified to have an additional machine to perform custom exercises and have the Quay registry installed locally.

# Lab Machine Requirements

In addition to the DO180 lab environment, a new VM has been added to that environment. This machine can be setup and configured locally with the following requirements:

### *VM Requirements*

- RHEL 8.4+
- 6 vCPU (8 vCPU Recommended)
- 12GB RAM (16GB Recommended)

- 60GB Storage (Image and Database storage)

## Using the Lab Guide

This lab guide and contents within the guide are meant to supplement the course and materials delivered as part of a custom DO180 delivery. It is advisable to download the RH354 course manual and the RH134 course manual prior to the class delivery. The DO180 course guide can be downloaded as part of the course.

### *Course Materials*

All course materials and lab materials for the custom portion of the course can be found here: [https://github.com/tmichett/OCP\\_Demos](https://github.com/tmichett/OCP_Demos)

The lab guide for this course can be downloaded from here: [https://github.com/tmichett/OCP\\_Demos/blob/main/Containers/Containers.pdf](https://github.com/tmichett/OCP_Demos/blob/main/Containers/Containers.pdf)



# 1. RHEL 8 Changes

RHEL 8.x brought several significant changes and expanded upon other changes that were introduced as part of the **SystemD** switch in RHEL 7.x. This course will highlight some of the most significant changes and enhancements to RHEL 8.x.

## RHEL 8.x Enhancements

- TMUX
- SystemD
- FirewallD

Before beginning the course, there is a quick set of steps that will be completed as an exercise so that the environment can be setup and prepared. The **Workstation** machine will be our GUI machine used throughout the week and is running RHEL 8.2. We will also be using a custom RHEL 8.4 system called **server**.

## 1.1. Setup Exercise

Before you begin using the customized DO180 Workshop Lab Environment a couple things must be accomplished.

### *Before you Begin*

1. Prepare Workstation
2. SSH Keys must be Generated and Distributed for the root user on **server**
3. Github Projects Must be Downloaded
4. Ansible Playbooks Must be Executed

There will be two Github projects used in this course for the custom exercises and demonstrations.

- **OCP Demos:** [https://github.com/tmichett/OCP\\_Demos](https://github.com/tmichett/OCP_Demos) - The **/Containers** directory contains most of the resources being used for the course.
- **Quay\_Lab\_PoC:** [https://github.com/tmichett/quay\\_lab\\_poc](https://github.com/tmichett/quay_lab_poc) - This repository contains playbooks that will setup and deploy Quay locally on the **server** system.

The lab environment will need **server** configured to be our **quay.local** system and will need to have access to all packages and keys.

### Example 1. Lab Installation and Setup Exercise

#### 1. Create Directories and Download Github Repos on **workstation**

##### *Listing 1. Creating Directory and Switch to Directory*

```
[student@workstation ~]$ mkdir github ; cd github
```

##### *Listing 2. Cloning Repositories - Cloning quay\_lab\_poc on workstation*

```
[student@workstation github]$ git clone https://github.com/tmichett/quay_lab_poc.git
Cloning into 'quay_lab_poc'...
remote: Enumerating objects: 266, done.
remote: Counting objects: 100% (266/266), done.
remote: Compressing objects: 100% (167/167), done.
remote: Total 266 (delta 67), reused 253 (delta 54), pack-reused 0
Receiving objects: 100% (266/266), 5.90 MiB | 22.90 MiB/s, done.
Resolving deltas: 100% (67/67), done
```

##### *Listing 3. Cloning Repositories - Cloning OCP\_Demos on workstation*

```
[student@workstation github]$ git clone https://github.com/tmichett/OCP_Demos.git
Cloning into 'OCP_Demos'...
remote: Enumerating objects: 763, done.
remote: Counting objects: 100% (763/763), done.
remote: Compressing objects: 100% (527/527), done.
remote: Total 763 (delta 329), reused 595 (delta 181), pack-reused 0
Receiving objects: 100% (763/763), 37.16 MiB | 29.22 MiB/s, done.
Resolving deltas: 100% (329/329), done.
```

#### 2. Switch to the **OCP\_Demos/Containers/labs/server\_prep** Folder on **workstation**

```
[student@workstation github]$ cd OCP_Demos/Containers/labs/server_prep
```

#### 3. Execute the **Setup\_Workstation.yml** playbook on **workstation**

```
[student@workstation server_prep]$ ansible-playbook Setup_Workstation.yml
```

#### 4. Create SSH keys on **workstation**

```
[student@workstation server_prep]$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/student/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/student/.ssh/id_rsa.
Your public key has been saved in /home/student/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:/FXYb+9b7dGD9CJpwTdsE0MqJkitLBuHYJDofp4Jz3w
student@workstation.lab.example.com
The key's randomart image is:
+---[RSA 3072]----+
|+. . . .
|oo . . . oo |
|o . o... o ..oo |
| . + + .o o ...o |
| . = S o.B o |
| o o . . * =.+|
| B o .+ o +=|
| B E . . .o+|
| . .+|
+---[SHA256]----+
```

## 5. Copy SSH key for the **root** user to **server** from **workstation**

```
[student@workstation server_prep]$ ssh-copy-id root@server
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed:
"/home/student/.ssh/id_rsa.pub"
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter
out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted
now it is to install the new keys
root@server's password:
```

Number of key(s) added: 1

Now try logging into the machine, with: "ssh 'root@server'"  
and check to make sure that only the key(s) you wanted were added.

## 6. Execute the **Setup\_Server.yml** playbook on **workstation**

```
[student@workstation server_prep]$ ansible-playbook Setup_Server.yml
```

## 7. Install the **Container Tools** package on **server**

```
[root@server ~]# yum module install container-tools
```

#### 8. Install **cockpit** on server

```
[root@server ~]# yum install cockpit* ①
```

① The " is added to install all Cockpit packages and plugins. This will allow the \*cockpit-podman to already be installed and ready on the system.

## 1.2. TMUX Usage

As part of the upgrade process and package replacement process in RHEL8, several packages have not only been deprecated, they've been completely removed. The **screen** package is one package that is no longer available for installation. Instead, a new terminal program **tmux** has been introduced to provide the **screen** functionality as well as other enhancements.

*TMUX Usage*

TMUX References



Red Hat Learning Community: <https://learn.redhat.com/t5/Platform-Linux/Using-tmux-to-execute-commands-on-servers-in-parallel/m-p/2200>

Tactical TMUX: <https://danielmiessler.com/study/tmux/>

In the example below, we will explore the **screen** functionalities of TMUX with respect to attaching and detaching of sessions.

## Example 2. LAB: Installing TMUX

### 1. Install tmux with YUM

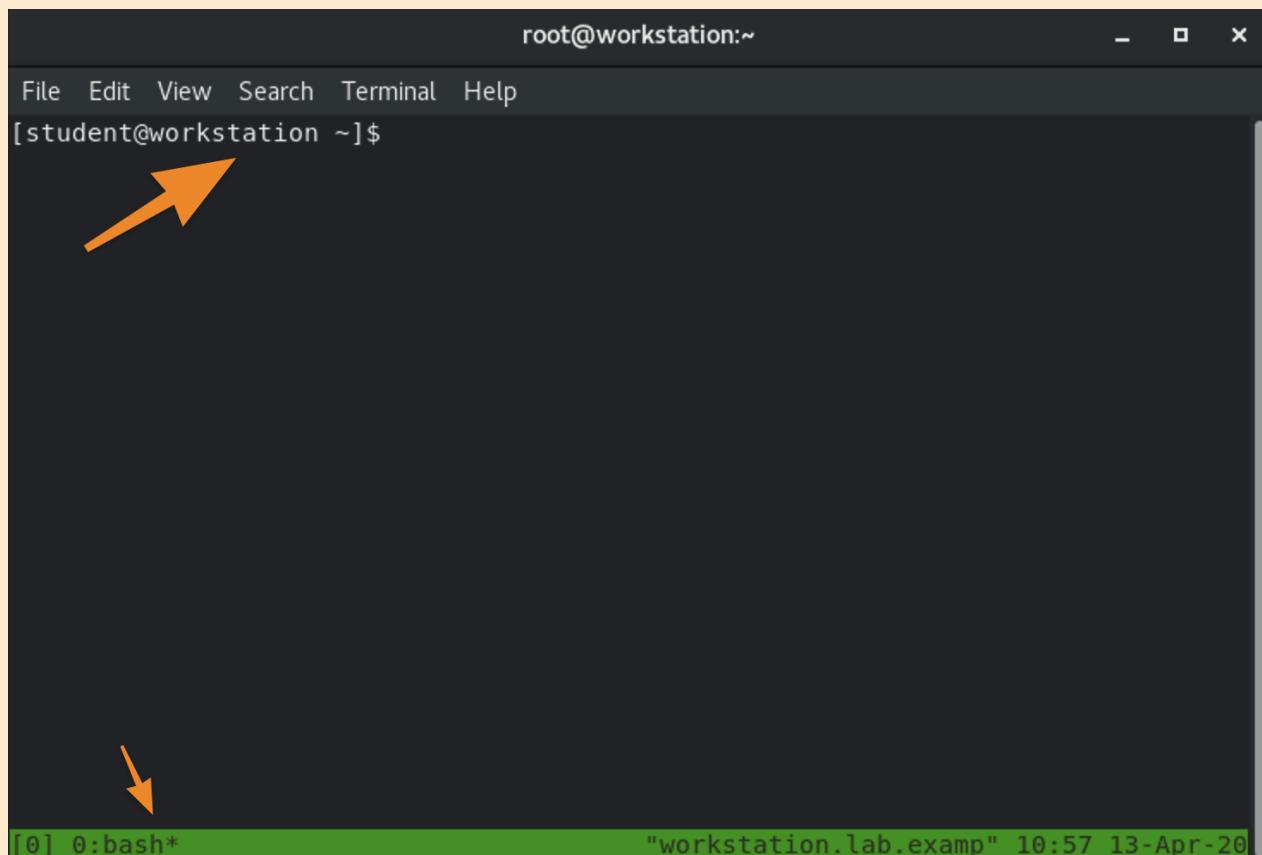
*Listing 4. Installation of TMUX*

```
[root@workstation ~]# yum install tmux
Red Hat Enterprise Linux 8.0 AppStream (dvd)      21 MB/s | 5.3 MB   00:00
Red Hat Enterprise Linux 8.0 BaseOS (dvd)        23 MB/s | 2.2 MB   00:00
Last metadata expiration check: 0:00:01 ago on Mon 13 Apr 2020 10:55:47 AM EDT.
Package tmux-2.7-1.el8.x86_64 is already installed.
Dependencies resolved.
Nothing to do.
Complete!
```

### 2. Launch tmux

*Listing 5. Using tmux*

```
[student@workstation ~]$ tmux
```



*Figure 1. tmux Terminal Window*

### 3. Placing tmux Window in Background (CTRL+B+D)

***Listing 6. Detaching a tmux Session***

```
[student@workstation ~]$ tmux
[detached (from session 0)]
```

**4. Re-attaching to tmux*****Listing 7. Source Description***

```
[student@workstation ~]$ tmux list-sessions
0: 1 windows (created Mon Apr 13 11:01:53 2020) [80x23]

[student@workstation ~]$ tmux attach-session -t 0
```

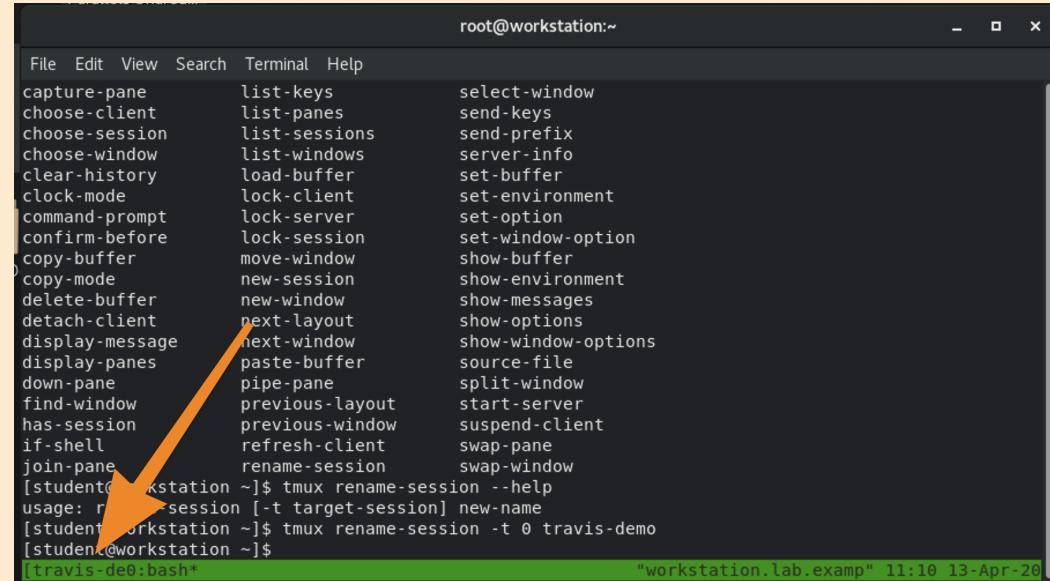
***Naming or Renaming Sessions***

It is possible that once you are in a **tmux** session to rename the session.

***Listing 8. Renaming a TMUX Session***

```
[student@workstation ~]$ tmux rename-session --help
usage: rename-session [-t target-session] new-name

[student@workstation ~]$ tmux rename-session -t 0 travis-demo
```

```
root@workstation:~-
File Edit View Search Terminal Help
capture-pane      list-keys          select-window
choose-client     list-panes         send-keys
choose-session    list-sessions     send-prefix
choose-window     list-windows      server-info
clear-history    load-buffer       set-buffer
clock-mode        lock-client      set-environment
command-prompt   lock-server      set-option
confirm-before   move-window      set-window-option
copy-buffer       new-session       show-buffer
copy-mode         new-window       show-environment
delete-buffer    new-window       show-messages
detach-client    next-layout      show-options
display-message  next-window      show-window-options
display-panes    paste-buffer     source-file
down-pane        pipe-pane        split-window
find-window      previous-layout  start-server
has-session       previous-window suspend-client
if-shell          refresh-client  swap-pane
join-pane         rename-session  swap-window
[student@workstation ~]$ tmux rename-session --help
usage: rename-session [-t target-session] new-name
[student@workstation ~]$ tmux rename-session -t 0 travis-demo
[student@workstation ~]$ "workstation.lab.example" 11:10 13-Apr-20
```

***Figure 2. Renamed tmux Session***

**5. Rename the tmux Session and place in background with (**CTRL+B, then D**)**

```
[student@workstation ~]$ tmux rename-session -t 0 travis-demo
```

```
[student@workstation ~]$ tmux attach-session  
[detached (from session travis-demo)] ①
```

① Verifying TMUX Sessions Exited

6. Create a new session and place in background with (***CTRL+B, then D***)

```
[student@workstation ~]$ tmux new-session
```



If you receive a warning/error message "**sessions should be nested with care, unset \$TMUX to force**", that means the current TMUX Session has not been placed in the background. It is important not to nest TMUX sessions as this can cause confusion.

7. List Sessions

```
[student@workstation ~]$ tmux list-sessions  
1: 1 windows (created Tue Nov  9 08:09:40 2021) [86x18]  
travis-demo: 1 windows (created Tue Nov  9 08:04:27 2021) [86x18]
```

8. Reattach to the **travis-demo** session

```
[student@workstation ~]$ tmux attach-session -t travis-demo
```

## 1.3. SystemD Overview

### 1.3.1. Understanding SystemD Unit Files and Creating a Service

Starting in RHEL 7, **SystemD** replaced the older style Linux SystemV (sysvinit daemon). This change continued through with RHEL 8 and more systemd tools replaced the traditional legacy tools. This small section will provide references and an overview of how **systemd** can be used to replace the older *init* scripts that were placed in **/etc/init.d** or some of the other directories.

### *SystemD References*

Linus Torvalds and others on Linux's systemd: <https://www.zdnet.com/article/linus-torvalds-and-others-on-linuxs-systemd/>

SysVinit Vs systemd Cheatsheet: <https://www.2daygeek.com/sysvinit-vs-systemd-cheatsheet-systemctl-command-usage/>



### Example 3. LAB: Creating a SystemD Script and Service

#### 1. Create the Init Script

*Listing 9. Init Script to Run at Startup*

```
[root@server ~]# vim /usr/bin/systemd_test_service.sh
#!/usr/bin/bash

DATE=`date '+%Y-%m-%d %H:%M:%S'`
echo "This is a sample service started at ${DATE} for the SystemD Custom course."
| systemd-cat -p info

while :
do
echo "Looping...";
sleep 30;
done
```

#### 2. Make script executable

*Listing 10. Running chmod on script*

```
[root@server ~]# chmod +x /usr/bin/systemd_test_service.sh
```

#### 3. Edit SystemD Service (Unit File)

*Listing 11. Editing the .service File*

```
[root@server ~]# vim /etc/systemd/system/systemd_test_service.service

[Unit]
Description=Systemd-Test-Service example systemd service.

[Service]
Type=simple
ExecStart=/bin/bash /usr/bin/systemd_test_service.sh

[Install]
WantedBy=multi-user.target
```

#### 4. Fix Permissions on .service File

*Listing 12. Running chmod on .service File*

```
[root@server ~]# chmod 644 /etc/systemd/system/systemd_test_service.service
```

*SystemD Services*

Once a script has been created and made executable and a service file has properly been created and placed in **/etc/systemd/system/** directory it is possible to use the **systemctl** command to interact with the service file and make it active on boot.

*Default SystemD Directories*

It is important to note that there are several default SystemD directories. When defining your own files, the proper location is to place them in **/etc/systemd**. There is more information available in other Red Hat courses, specifically the RH442 Performance Tuning course.

Default Location: **/lib/systemd/**

## 5. Starting and Enabling a Custom Service

*Listing 13. Controlling a Custom Service with systemctl*

```
[root@server ~]# systemctl enable systemd_test_service.service --now
Created symlink /etc/systemd/system/multi-
user.target.wants/systemd_test_service.service →
/etc/systemd/system/systemd_test_service.service.
```

## 6. Checking Status of a Custom Service

***Listing 14. Using systemctl to Check Service Status***

```
[root@server ~]# systemctl status systemd_test_service.service
● systemd_test_service.service - Systemd-Test-Service example systemd service.
   Loaded: loaded (/etc/systemd/system/systemd_test_service.service; enabled;
   vendor >
     Active: active (running) since Mon 2021-10-25 10:07:42 EDT; 24s ago
       Main PID: 1533 (bash)
          Tasks: 2 (limit: 152500)
         Memory: 612.0K
            CGrou: /system.slice/systemd_test_service.service
                      └─1533 /bin/bash /usr/bin/systemd_test_service.sh
                          └─1537 sleep 30

Oct 25 10:07:42 server.lab.example.com systemd[1]: Started Systemd-Test-Service
examp>
Oct 25 10:07:42 server.lab.example.com bash[1533]: Looping...
```

**7. Checking /var/log/messages for Custom Service*****Listing 15. Seaching Log file for Custom Service***

```
[root@server ~]# grep -i "systemd custom" /var/log/messages
Oct 25 10:07:42 server journal[1536]: This is a sample service started at 2021-10-
25 10:07:42 for the SystemD Custom course.
```

### *SystemD References*

Creating a Service at Boot: <https://www.linode.com/docs/quick-answers/linux/start-service-at-boot/>

Overview of SystemD for RHEL7: <https://access.redhat.com/articles/754933>

Converting traditional sysV init scripts to Red Hat Enterprise Linux 7 systemd unit files: <https://www.redhat.com/en/blog/converting-traditional-sysv-init-scripts-red-hat-enterprise-linux-7-systemd-unit-files>

 Creating and Modifying SystemD Unit Files: [https://access.redhat.com/documentation/en-us/red\\_hat\\_enterprise\\_linux/7/html/system\\_administrators\\_guide/sect-managing\\_services\\_with\\_systemd-unit\\_files](https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/system_administrators_guide/sect-managing_services_with_systemd-unit_files)

Creating a Linux service with systemd: <https://medium.com/@benmorel/creating-a-linux-service-with-systemd-611b5c8b91d6>

How to create systemd service unit in Linux: <https://linuxconfig.org/how-to-create-systemd-service-unit-in-linux>

### *References*

 **Systemd Timers for Scheduling Tasks:** <https://fedoramagazine.org/systemd-timers-for-scheduling-tasks/>

## 1.4. FirewallD

Beginning in RHEL 8.0, Red Hat moved away from **iptables** as the back-end firewall implementation and instead moved to NFTables. However, the introduction of FirewallD and the **firewall-cmd** management commands implemented in RHEL 7.x have evolved and leveraging FirewallD is still the preferred firewall management solution in RHEL 8.x.

### 1.4.1. FirewallD Service Definitions

FirewallD was introduced in RHEL7 as part of the SystemD transition and a new way to manage firewalls without using the underlying firewall implementation (**iptables**). FirewallD with **firewall-cmd** continues to be used in RHEL8 as the preferred method of managing and maintaining firewall rules.

#### *FirewallD Resources*

<https://firewalld.org/>

<https://www.liquidweb.com/kb/an-introduction-to-firewalld/>

<https://cheatography.com/mikael-leberre/cheat-sheets/firewall-cmd/>

## 1.4.2. The firewall-cmd Utility

The **firewall-cmd** utility is the primary method to manage and interact with firewall rules on RHEL7/8 systems. The **firewall-cmd** utility supports BASH completion and allows firewall rules to be added based on defined services or by specifying ports/protocols.

*Listing 16. Allowing HTTP through the Firewall by Port/Protocol*

```
[root@server ~]# firewall-cmd --add-port=80/tcp
success

[root@server ~]# firewall-cmd --list-all
public (active)
  target: default
  icmp-block-inversion: no
  interfaces: enp1s0
  sources:
  services: cockpit dhcpcv6-client ssh
  ports: 80/tcp
  protocols:
  masquerade: no
  forward-ports:
  source-ports:
  icmp-blocks:
  rich rules:

[root@server ~]# firewall-cmd --remove-port=80/tcp
success
```

**Listing 17. Allowing HTTP through the Firewall by Service**

```
[root@server ~]# firewall-cmd --add-service=
Display all 154 possibilities? (y or n)

[root@server ~]# firewall-cmd --add-service=http
success

[root@server ~]# firewall-cmd --list-all
public (active)
  target: default
  icmp-block-inversion: no
  interfaces: enp1s0
  sources:
  services: cockpit dhcpcv6-client http ssh
  ports:
  protocols:
  masquerade: no
  forward-ports:
  source-ports:
  icmp-blocks:
  rich rules:
```

**firewall-cmd Usage Warning**

The **firewall-cmd** utility can be used to make changes to the running firewall as shown in the examples above. This does not make changes to the firewall config file. In order to make the changes to the configuration file, it is necessary to use the **--permanent** options to have the changes written to a file.



When using **--permanent**, and you are not making changes to the current firewall runtime, it is also necessary to use: **firewall-cmd --reload** to reload or load new firewall rules from the firewall configuration file.

**Important Header**

It is important to note that presently the Red Hat Web Console (cockpit) only supports management of **firewalld** using defined services.

### 1.4.3. FirewallD Files and Locations

FirewallD has a few locations for files both for configuration and usage. As with most configuration files on a Linux system, those rely in **/etc/**. The user configurable files for FirewallD also reside in **/etc/**. Default configuration files for services, zones, and other FirewallD functionality resides in **/usr/lib/firewalld**. This location contains all defined services files and default configuration files for FirewallD and used by the **firewall-cmd** utility.

*Listing 18. FirewallD Configuration Files*

```
[root@server ~]# tree /etc/firewalld/
/etc/firewalld/
├── firewalld.conf
├── helpers
├── icmptypes
├── ipsets
├── lockdown-whitelist.xml
├── services
└── zones
    ├── public.xml
    └── public.xml.old
```

5 directories, 4 files

*Listing 19. FirewallD Default Configuration Files*

```
[root@server ~]# tree /usr/lib/firewalld/
/usr/lib/firewalld/
├── helpers
│   ├── amanda.xml
│   ├── ftp.xml
│   ├── h323.xml
│   ├── irc.xml
│   ├── netbios-ns.xml
│   ├── pptp.xml
│   └── proto-gre.xml
... output omitted ...
└── zones
    ├── block.xml
    ├── dmz.xml
    ├── drop.xml
    ├── external.xml
    ├── home.xml
    ├── internal.xml
    ├── public.xml
    ├── trusted.xml
    └── work.xml
```

5 directories, 222 files

#### 1.4.4. Defining a Custom Service File

It is possible to define custom **firewalld** service files. These files can be used for your environments for custom applications or custom firewall rules. These service files can also be checked into a version

control system such as **git**.



#### *Creating a Custom Service File using Existing File as Base*

It is easiest to take an existing service file, copy it to the **/etc/firewalld/services** directory, rename and edit the file.

### 1. Copy existing FirewallD service file to **/etc/firewalld/services**

#### *Listing 20. Using SSH Service File as a Starting Point*

```
[root@server ~]# cp /usr/lib/firewalld/services/ssh.xml  
/etc/firewalld/services/custom_ssh.xml
```

### 2. Edit the file and specify the new options

#### *Listing 21. Edit the Custom SSH Service Definition*

```
[root@server ~]# vim /etc/firewalld/services/custom_ssh.xml  
<?xml version="1.0" encoding="utf-8"?>  
<service>  
  <short>SSH_Custom</short>  
  <description>This is a custom SSH service definition for paranoid people that don't  
  want to run SSH on the default port of 22. This will allow SSH to run on the port 8022  
  to meet our defined security guidance.</description>  
  <port protocol="tcp" port="8022"/>  
</service>
```

### 3. Listing FirewallD Services to Verify New Service

#### *Listing 22. Getting Listing of FirewallD Services*

```
[root@server ~]# firewall-cmd --get-services | grep custom_ssh
```



#### *FirewallD Delays in Discovering a Service*

Depending on system speed and refreshing of FirewallD daemon, the new service might not be picked up immediately. It will be discovered or if you are in a hurry, you can run **firewall-cmd --reload** command to immediately have the service discovered and available.

## 1.4.5. FirewallD Configuration Files

As stated above, the main FirewallD configuration files are located in **/etc/firewalld**. To specifically change or view the configuration file on the system, you generally want to look at the firewalls in the **Firewall Zone**. This is found by opening the corresponding zone file in **/etc/firewalld/zones** directory.

## 1. Viewing Firewall configuration for the Public Zone.

*Listing 23. FirewallD Configuration for Public*

```
[root@server ~]# cat /etc/firewalld/zones/public.xml
<?xml version="1.0" encoding="utf-8"?>
<zone>
  <short>Public</short>
  <description>For use in public areas. You do not trust the other computers on
networks to not harm your computer. Only selected incoming connections are
accepted.</description>
  <service name="ssh"/>
  <service name="dhcpcv6-client"/>
  <service name="cockpit"/>
</zone>
```

## 2. Adding a custom service

*Listing 24. Adding our new Service*

```
[root@server ~]# firewall-cmd --add-service=custom_ssh --permanent
success
```

## 3. Verifying Firewall configuration for the Public Zone.

*Listing 25. FirewallD Configuration for Public with Custom Service*

```
[root@server ~]# cat /etc/firewalld/zones/public.xml
<?xml version="1.0" encoding="utf-8"?>
<zone>
  <short>Public</short>
  <description>For use in public areas. You do not trust the other computers on
networks to not harm your computer. Only selected incoming connections are
accepted.</description>
  <service name="ssh"/>
  <service name="dhcpcv6-client"/>
  <service name="cockpit"/>
  <service name="custom_ssh"/>
</zone>
```

#### Example 4. LAB: Creating a Quay Firewall Service

In preparation of the Quay exercises, this lab will be used to create a custom service file for use with FirewallD.

*Table 1. Network Ports for Firewalls and Port Mapping for Quay*

Service Name	Network Port
Quay HTTPS	8443/TCP
Quay HTTP	8080/TCP
Quay HTTP	80/TCP
Quay HTTPS	443/TCP
Postgresql Quay	5432/TCP
Postgresql ClairV4	5433/TCP
Redis for Quay	6379
ClairV4 HTTP	8081/TCP

1. Copy the `ssh.xml` to `quay_lab.xml` as a template.

```
[student@workstation ~]$ cp /usr/lib/firewalld/services/ssh.xml ~/quay_lab.xml
```

2. Modify and edit the `quay_lab.xml` with the needed firewall ports.

```
[student@workstation ~]$ vim quay_lab.xml
<service>
  <short>QUAY-LAB</short>
  <description>This service has been created for the Quay Lab PoC. It will be used for deploying Quay, Quay Mirroring, and the ClairV4 image scanning service. It is also designed to support the Postgres and Redis supporting services.</description>
  <port protocol="tcp" port="80"/>
  <port protocol="tcp" port="8080"/>
  <port protocol="tcp" port="8443"/>
  <port protocol="tcp" port="443"/>
  <port protocol="tcp" port="5432"/>
  <port protocol="tcp" port="5433"/>
  <port protocol="tcp" port="6379"/>
  <port protocol="tcp" port="8081"/>
</service>
```

3. Copy the file to the **server** for later use

```
[student@workstation ~]$ scp quay_lab.xml root@server:/etc/firewalld/services/
quay_lab.xml                                         100%  627    247.8KB/s  00:00
```

## 1.5. Cockpit

Another change with RHEL 8.x was with the graphical server and the desktop manager. The X11 project and XWindows has been replaced largely with Wayland/Gnome3 as the graphical rendering environment of choice. These changes introduced some dependencies on the graphical management of several system services and components. The Wayland change no longer supports the X11 forwarding, so it was necessary to build tools or extend existing tools to be managed differently. The **cockpit** project was utilized and implemented in RHEL 8 as the new Red Hat Web Management Console which brought in numerous plugins allowing these services to be managed graphically within a standard web browser.

*Red Hat Web Management Console*



Leverage portions of the RH354 text around Cockpit and the Red Hat Web Management Console before looking at the additional Cockpit packages.

### 1.5.1. Installing Additional Cockpit Packages

*Listing 26. Install all Base Cockpit Packages*

```
[root@server ~]# yum install cockpit*
```

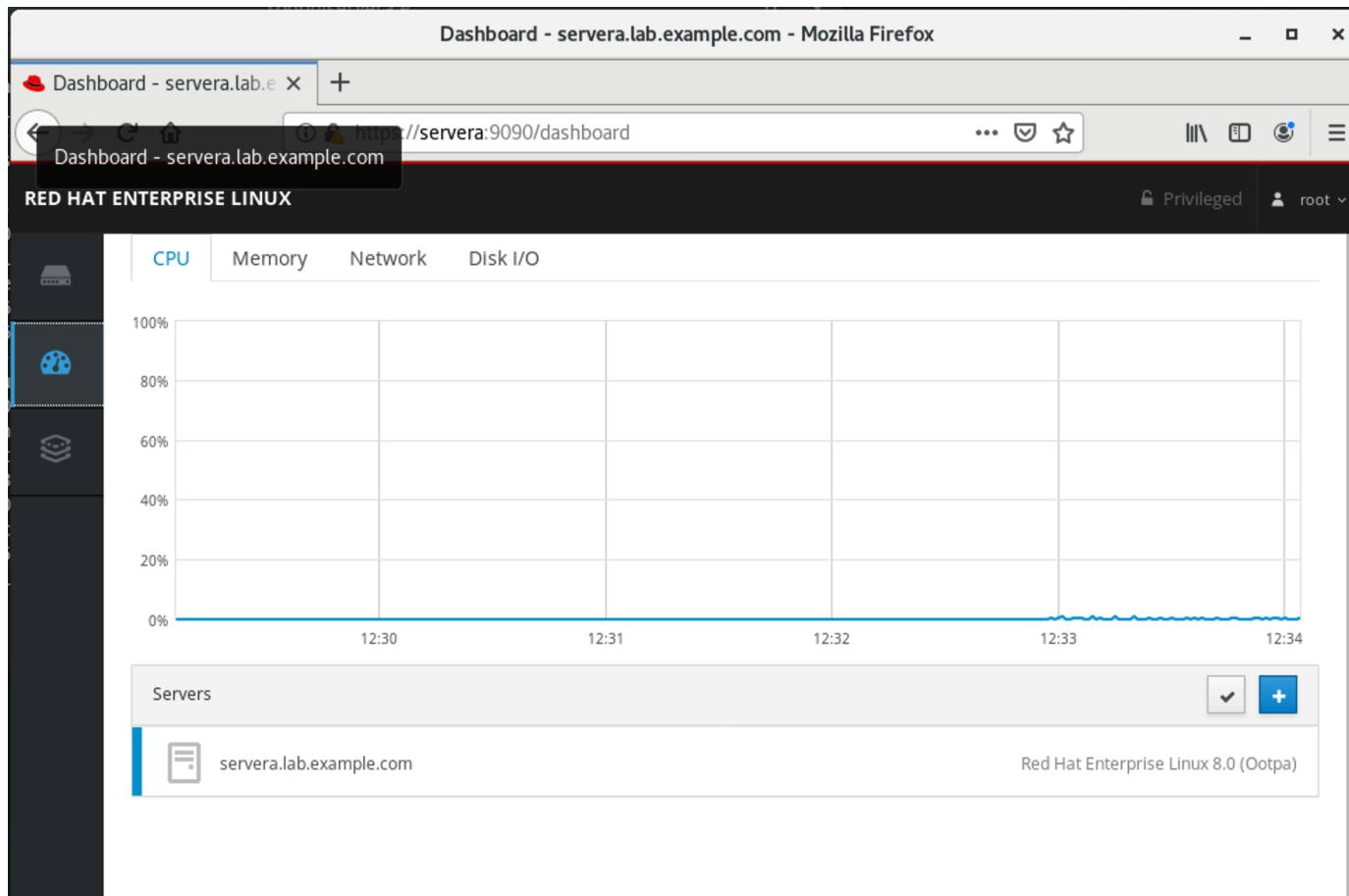


Figure 3. Cockpit Dashboard

#### Cockpit Plugin and Package Availability

Using the installation method above will install all Cockpit packages and plugins that are available in currently subscribed channels. However, not all components will work until additional back-end components are installed. For example, the **Composer** cockpit plugins need composer and other items installed in order to be able to be fully utilized. This installation gives the **Cockpit Dashboard Plugin** which allows connecting to multiple Web Consoles.



##### 1.5.1.1. Cockpit to Manage Multiple Systems

It is possible to use a single **cockpit** Interface to manage multiple servers.

**Example 5. LAB: Setting up Cockpit to control multiple systems**

1. Ensure cockpit socket/service is running and configured on all systems.

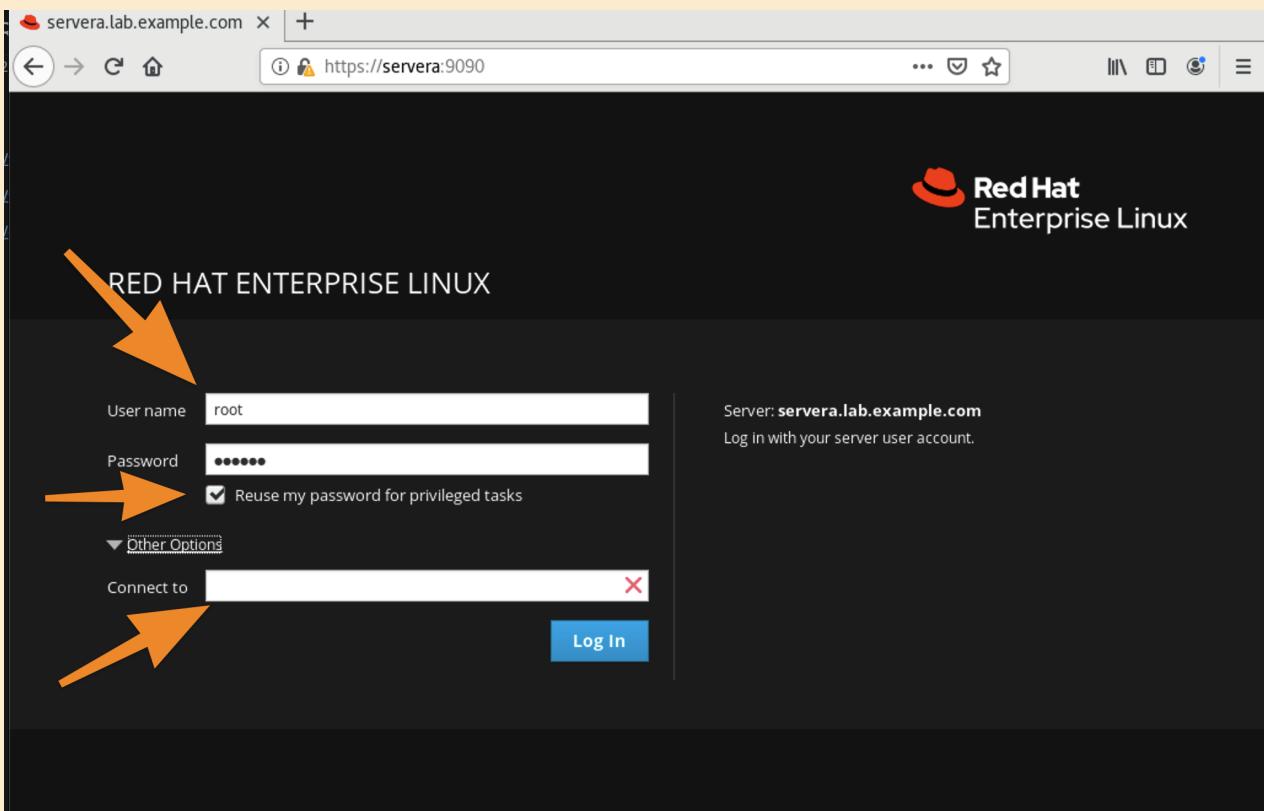
**Listing 27. Test and Enable Cockpit (Both workstation and server)**

```
[root@workstation ~]# systemctl enable cockpit.socket --now
Created symlink /etc/systemd/system/sockets.target.wants/cockpit.socket →
/usr/lib/systemd/system/cockpit.socket.

[student@workstation ~]$ ssh root@server
Activate the web console with: systemctl enable --now cockpit.socket

[root@server ~]# systemctl enable --now cockpit.socket
Created symlink /etc/systemd/system/sockets.target.wants/cockpit.socket →
/usr/lib/systemd/system/cockpit.socket.
```

2. Connect to the Cockpit Web Console

**Figure 4. Red Hat Web Console (cockpit)**

3. Add another Web Console from the Cockpit Dashboard

- a. Navigate to the Dashboard
- b. Click the "+" and complete the information
  - **workstation.lab.example.com**

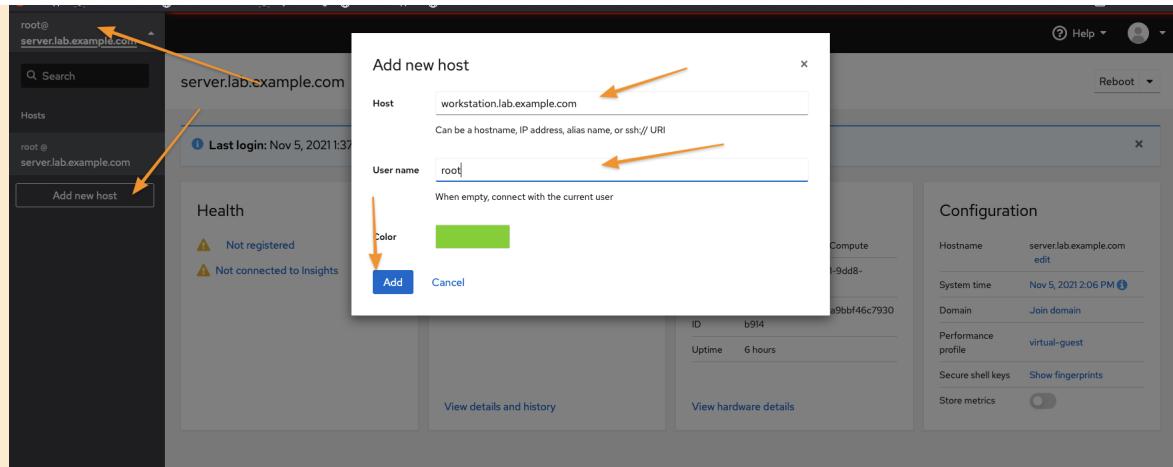


Figure 5. Adding Additional machines

#### 4. Verify the System Fingerprint and click Connect

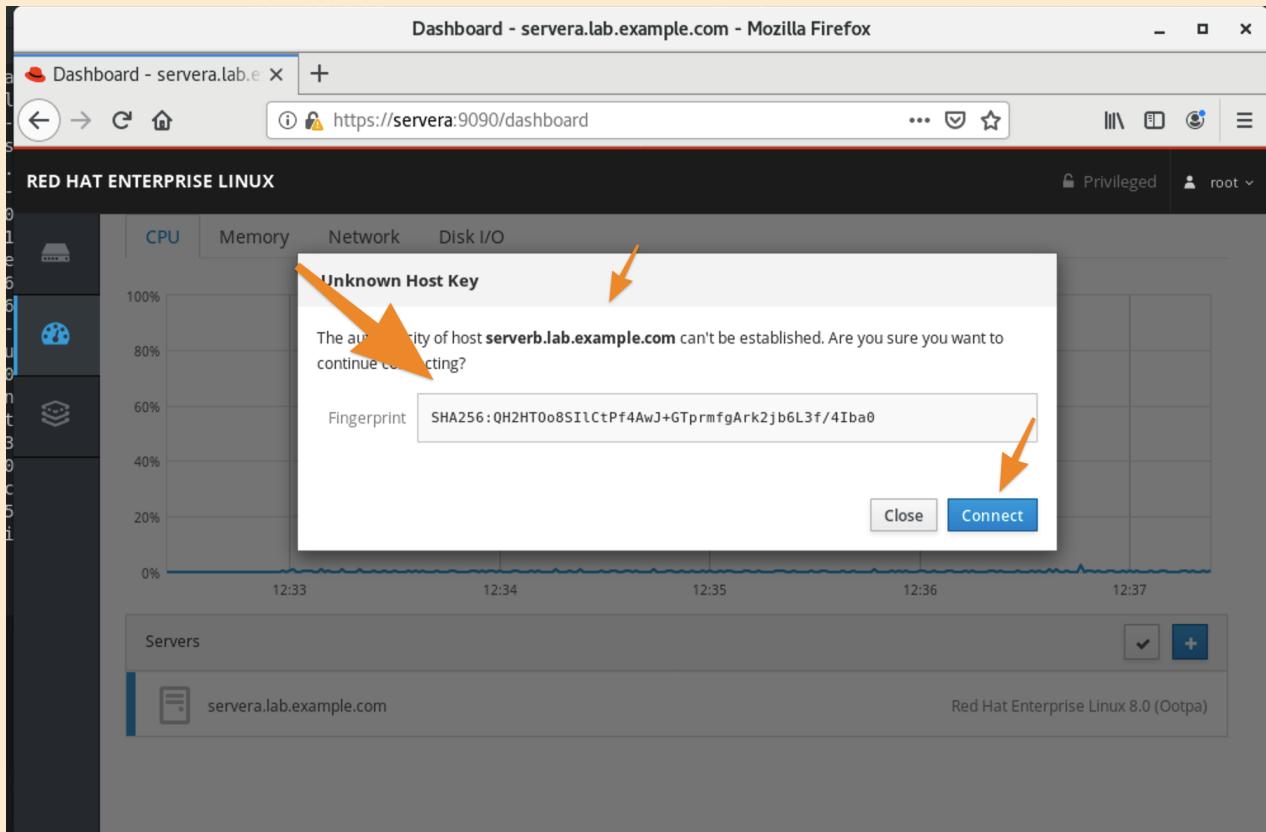


Figure 6. Fingerprint Verification

#### 5. It is now possible to switch systems from the System Drop-down menu

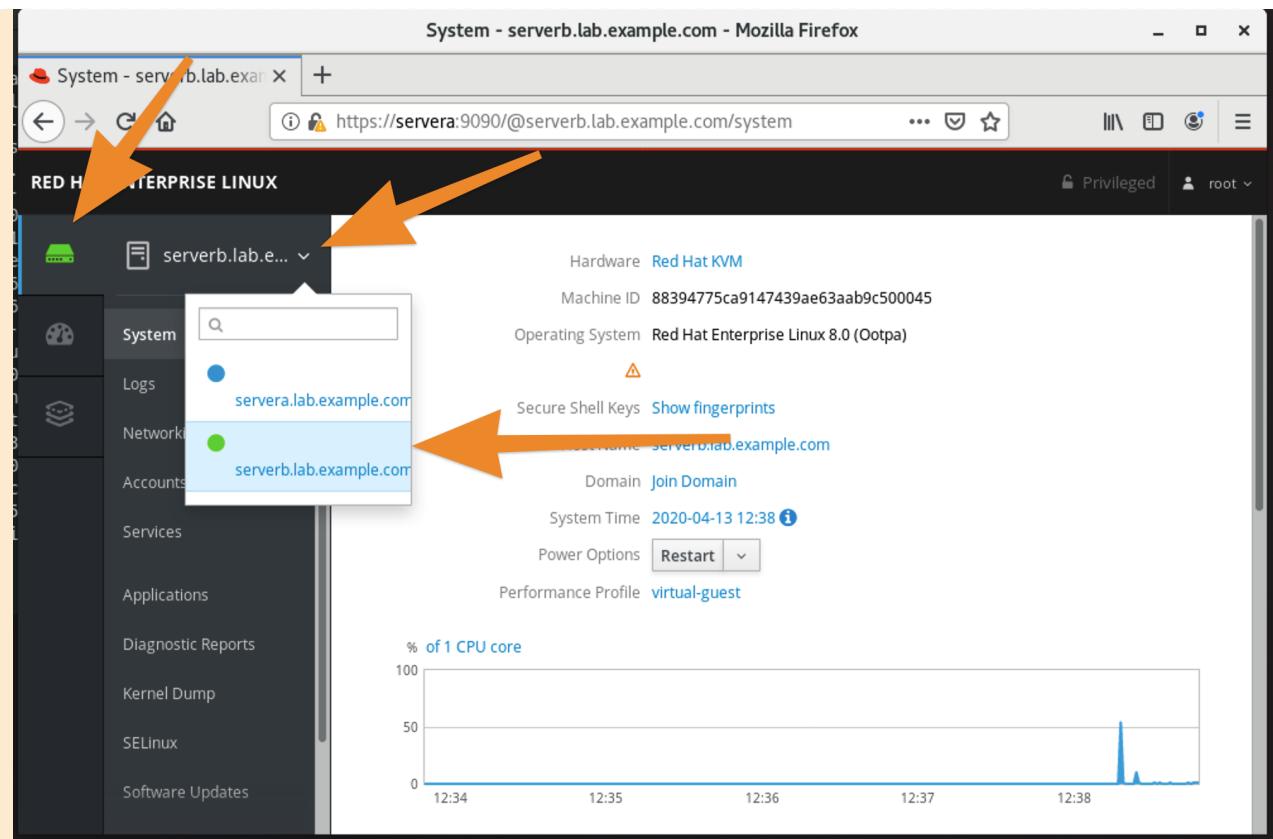


Figure 7. Switching Systems

## 2. Managing Containers with the New Runtime

### 2.1. Deploying Containers with the New Container Runtime

#### 2.1.1. The Podman Container Engine

RHEL8 includes the **container-tools** package module. New engine is **podman** replaces **docker** and **moby**. It also contains new tools **buildah** to build container images and **skopeo** to manage images on registries like **runc**. The new toolset allows building/running containers without daemons.

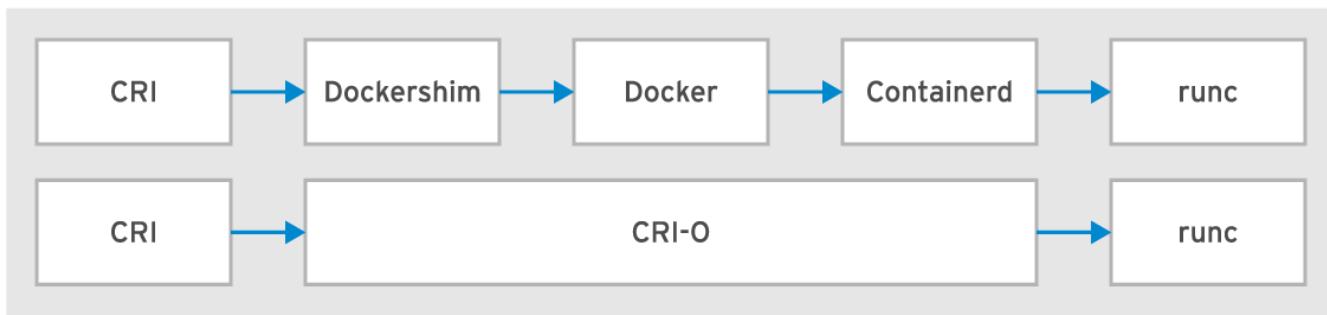


Figure 8. Docker to RHEL8 Container Runtime

#### Container Runtime Toolset

- Docker replaced with new container runtime
- New toolset supports OCI and reuse of third-party images
- Integrates with **audit** of Docker client-server model
- **container-tools** module provides new container runtime tools and engine.

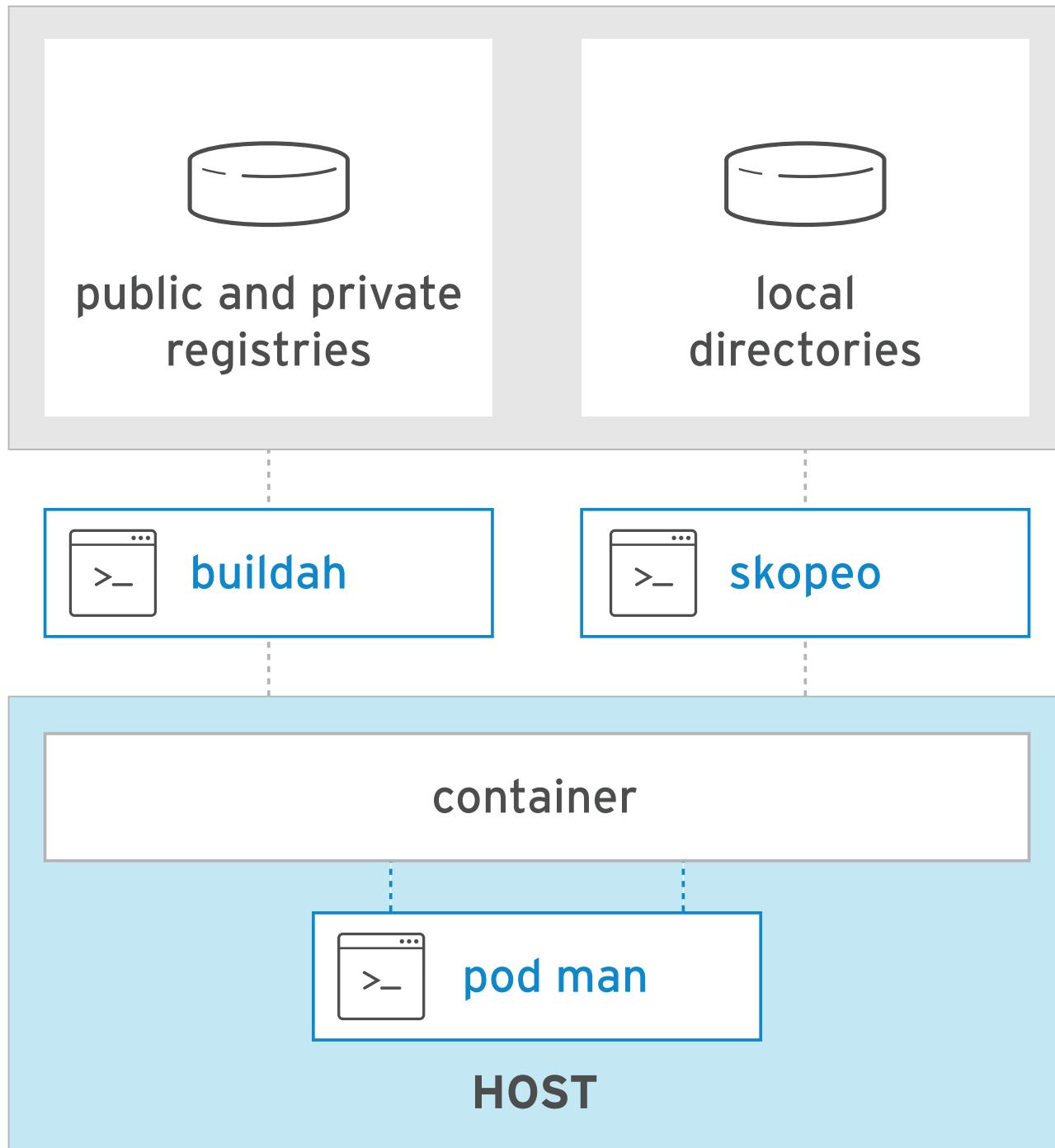


Figure 9. New Container Runtime

### Describing new Container Runtime Tool

- The **podman** engine is daemonless and supporting container execution.
- **podman** syntax is similar to the docker command, supporting **Dockerfile** use
- **Buildah** builds container images, from scratch or a Dockerfile.
- Copy and inspect container images in both local and remote container registries with **Skopeo**

- **Skopeo** supports Docker and private registries, the Atomic registry, and local directories, including those which use OCI



RHEL8 includes **Pacemaker** containers with **podman** as a tech preview. Pacemaker supports execution of the container across multiple hosts.

#### *Listing 28. Installation of Container Tools*

```
[student@workstation ~]$ sudo yum module install container-tools
```

## 2.2. Podman Configuration Files

Depending on whether **podman** is being run as a privileged user or as a **rootless** user, the configuration file locations and directives will be different.

Podman uses the **registry.conf** file on the system to retrieve information for container registries. The file is located at **/etc/containers/registries.conf**. The list of registries Podman uses are configured in the **[registries.search]** section and the **[registries.insecure]** section of the configuration file.

### *Rootless Podman Configuration*



The system-wide **registries.conf** file located at **/etc/containers/registries.conf** can be overridden by a user's configuration file located in **\$HOME/.config/containers** directory.

### *Container Image Registries and Searching*



While it is best practice to specify a fully qualified image, it is possible to provide just an image name on the command line. If an image name is specified, it will search the registries in the listed order to determine which image to select.

### 2.2.1. Root-Based Podman

When running Podman as a privileged user, the default configurations are used. Generally these files are not modified, however, it is a common practice to modify the **registries.conf** file to add additional container image registries as well as alter the registry search order.

#### *Default Configuration Files*

- **/etc/containers/storage.conf** - Contains information about the default storage for Podman containers when using **podman** in a non-rootless fashion.
- **/etc/containers/registries.conf** - Contains information about the registries used for **podman** when doing **podman search** and also for pulling/running of container images.
- **/etc/cni/net.d/87-podman-bridge.conflist** - Contains information related to the CNI (container network interface) and IP configuration for the container networks.

## 2.2.2. Rootless Podman

When running **podman** in **rootless** mode, all configuration files will be based on user configurations and are located within the user's home directory.

### User-Based Podman Configuration Files

- **/home/student/.config/containers/storage.conf**: Specifies storage configuration for Podman in **rootless** mode.
- **/home/student/.config/containers/registries.conf**: Specifies registry configuration for Podman in **rootless** mode.

#### *Podman User Configuration Files*

If the configuration files don't exist in the user's home directory, **podman** will default back up to **/etc** for the configuration files and if they aren't there, it will fall back to the **/usr/share/containers** directory.

#### *Listing 29. User Registries*



```
[student@servera ~]$ cat .config/containers/registries.conf
unqualified-search-registries = ['registry.lab.example.com']

[[registry]]
location = "registry.lab.example.com"
insecure = true
blocked = false
```

### *Podman Configuration File Precedence*

**podman** has multiple configuration file locations and the order of precedence depends on the if the files exist. Traditionally in a **rootless** implementation, the configuration files will be in the user's `$HOME/.config/containers` directory. In the case of networking, there is no CNI equivalent for **rootless** Podman as **rootless** containers rely on **SLIRP** for networking.

#### **Containers.conf**

1. `$HOME/.config/containers/containers.conf`
2. `/etc/containers/containers.conf`
3. `/usr/share/containers/containers.conf`



#### *Storage Configurations*

1. `$HOME/.config/containers/storage.conf`
2. `/etc/containers/storage.conf`

#### *Registry Configurations*

1. `HOME/.config/containers/registries.conf`
2. `/etc/containers/registries.d/`
3. `/etc/containers/registries.conf`

Precedence of **1** is the highest and will override all others.

#### *References*



- **Podman Project - Config Files and Tutorial:** [https://github.com/containers/podman/blob/main/docs/tutorials/rootless\\_tutorial.md](https://github.com/containers/podman/blob/main/docs/tutorials/rootless_tutorial.md)
- **Why can't I use sudo with rootless Podman?:** <https://www.redhat.com/sysadmin/sudo-rootless-podman>
- **What happens behind the scenes of a rootless Podman container?:** <https://www.redhat.com/sysadmin/behind-scenes-podman>
- **Rootless containers using Podman:** <https://www.redhat.com/sysadmin/rootless-containers-podman>
- **Container video series: Rootless containers, process separation, and OpenSCAP:** <https://www.redhat.com/sysadmin/container-video-series> (Really good by Brian Smith - Former RH TAM, now works in the RHEL Platform BU)
- **Why can't rootless Podman pull my image?:** <https://www.redhat.com/sysadmin/rootless-podman>

## 2.3. Container Image Storage

Containers use ephemeral storage for running containers which is an overlay filesystem with a new read/write (RW) layer added to the original container image. This ephemeral storage is removed once the container is removed from the system. Container images on the other-hand are stored locally on the system in the container image storage registry. It is important to know where the image storage is located for both container images and ephemeral storage for running containers so that it can be monitored for pro-active system administration and cleanup. The storage location for both container images and ephemeral storage is generally defined in the **storage.conf** file.

### 2.3.1. Root-Based Podman

A default installation of Red Hat Container Tools (podman) will create a configuration file for storage located **/etc/containers/storage.conf**. These are the most likely settings to be used when running Podman as a **root** user.

*Listing 30. Default Storage Location*

```
[storage]

# Default Storage Driver
driver = "overlay"

# Temporary storage location
runroot = "/var/run/containers/storage"

# Primary Read/Write location of container storage
graphroot = "/var/lib/containers/storage"
```

There are plenty of options regarding containers and image storage, but the primary locations to monitor are **/var/run/containers/storage** and **/var/lib/containers/storage** as these will be the most used locations by **podman**.

### 2.3.2. Rootless Podman

Podman uses the storage configuration file located **\$HOME/.config/containers/storage.conf**. These settings are used because the **overlay** filesystem must use a user-space filesystem overlay so it uses **FUSEFS** storage drives for the user-space filesystems.

***Listing 31. Default Storage Location for Rootless Podman***

```
[storage]
  driver = "overlay"
  runroot = "/run/user/1000"
  graphroot = "/home/student/.local/share/containers/storage" ①
[storage.options]

... OUTPUT OMITTED ...

mount_program = "/usr/bin/fuse-overlayfs" ②
```

① Image storage location

② FUSEFS Overlay Filesystem Driver

There are plenty of options regarding containers and image storage, but the primary locations to monitor are `/var/run/containers/storage` and `/var/lib/containers/storage` as these will be the most used locations by `podman`.

## 2.4. Container Networking

Podman is meant to provide all management of containers and the container runtime. Podman is capable of managing the container network (SDN) for root-based Podman containers. The CNI controls the specifications for networking and how the SDN is defined on the system. There is no SDN available for **Rootless** containers as `podman` implements networking for **Rootless** containers using **SLIRP**.

### 2.4.1. Root-Based Podman

Podman root-level containers can leverage CNI. The containers SDN network is defined in the `/etc/cni/net.d/87-podman-bridge.conf` configuration file. Containers can communicate to each other within the SDN on the same system.

**Listing 32. /etc/cni/net.d/87-podman-bridge.conf**

```
{  
    "cniVersion": "0.4.0",  
    "name": "podman",  
    "plugins": [  
        {  
            "type": "bridge", ①  
            "bridge": "cni-podman0", ②  
            "isGateway": true,  
            "ipMasq": true,  
            "ipam": {  
                "type": "host-local",  
                "routes": [  
                    {  
                        "dst": "0.0.0.0/0"  
                    }  
                ],  
                "ranges": [ ③  
                    [  
                        {  
                            "subnet": "10.88.0.0/16",  
                            "gateway": "10.88.0.1"  
                        }  
                    ]  
                ]  
            }  
        },  
        {  
            "type": "portmap",  
            "capabilities": {  
                "portMappings": true  
            }  
        },  
        {  
            "type": "firewall"  
        }  
    ]  
}
```

① Defines network type as a **Bridge**

② Defines the network name for the bridge as **cni-podman0** for the container SDN

③ Defines the network IP Address range for the container SDN

## 2.4.2. Rootless Podman

For **rootless** podman the networking for containers leverages **SLIRP**. This is provided by the **slirp4netns** package and provides user-mode networking and namespaces for the networks. Containers running as a **Rootless** container will not receive an IP address from the CNI SDN and cannot communicate with each other or the outside except by using and leveraging **port forwarding**.



*Red Hat Container Catalog*

**Red Hat Container Catalog:** <https://catalog.redhat.com/software/containers/search>

## 2.5. Managing Containers using the Red Hat Web Console

The Red Hat Web Management Console (Cockpit) can be used to manage container images as well as running containers. However, in order to manage some of the aspects around containers, certain configurations must already be in place.

Cockpit can almost manage the full container lifecycle. using the **Podman containers** plugin. Container images can be downloaded, managed, deleted. Containers can be launched from container images, stopped, restarted. Containers can even be analyzed providing runtime information, container logs, and even console access. Existing containers can also be turned into new container images using the **Commit** functions.

### Container Runtime Details

The **Details** tab can provide container runtime information. This is similar to running the **podman inspect** without providing all the details.

ID	4c76fe54f5353858fbb5e22928bbd440911769b503daecb29a95e4494b63ee05
Created	Today at 1:10 PM
Image	quay.io/redhattraining/httpd-parent:latest
Command	/bin/sh -c "/usr/sbin/httpd -DFOREGROUND"
State	Up since Today at 1:10 PM
Ports	0.0.0.0:80 → 80/tcp
IP address	10.88.0.49
IP prefix length	16
Gateway	10.88.0.1
MAC address	0e:ae:e5:84:d4:fd

Figure 10. Container Runtime Information

### Container Logs

The **Logs** tab is capable of sharing the container logs. This is similar to running the **podman logs** command.

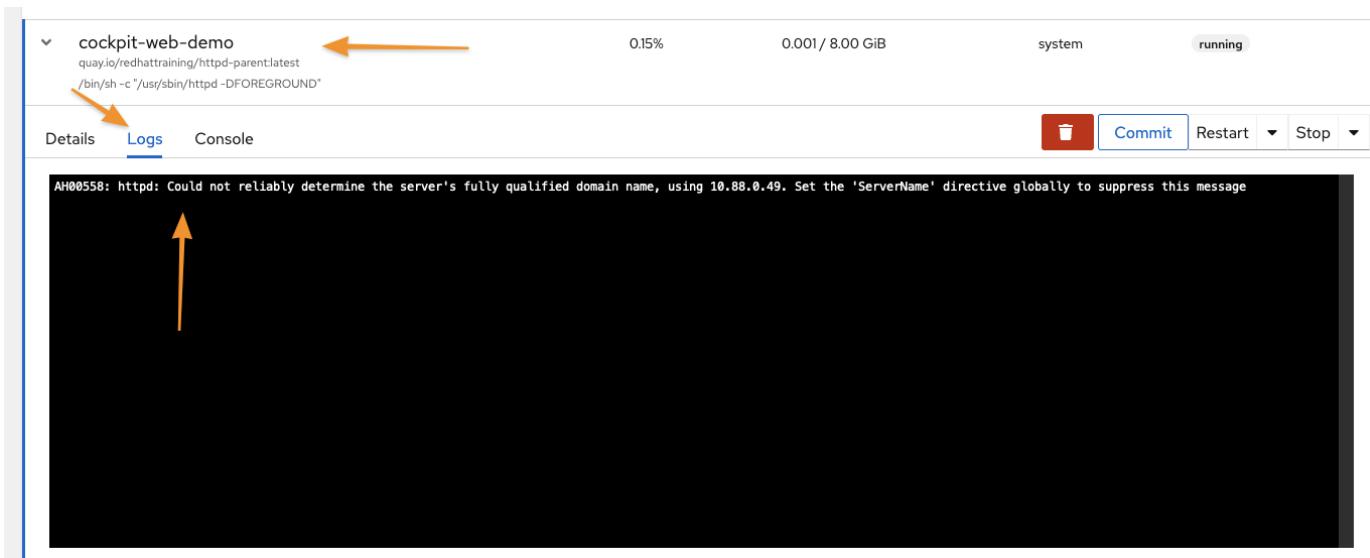


Figure 11. Container Logs

#### Container Console

Cockpit also allows accessing a console directly inside the running container. Accessing the console via cockpit is the same as running the **podman exec -it <Container> /bin/bash** command.

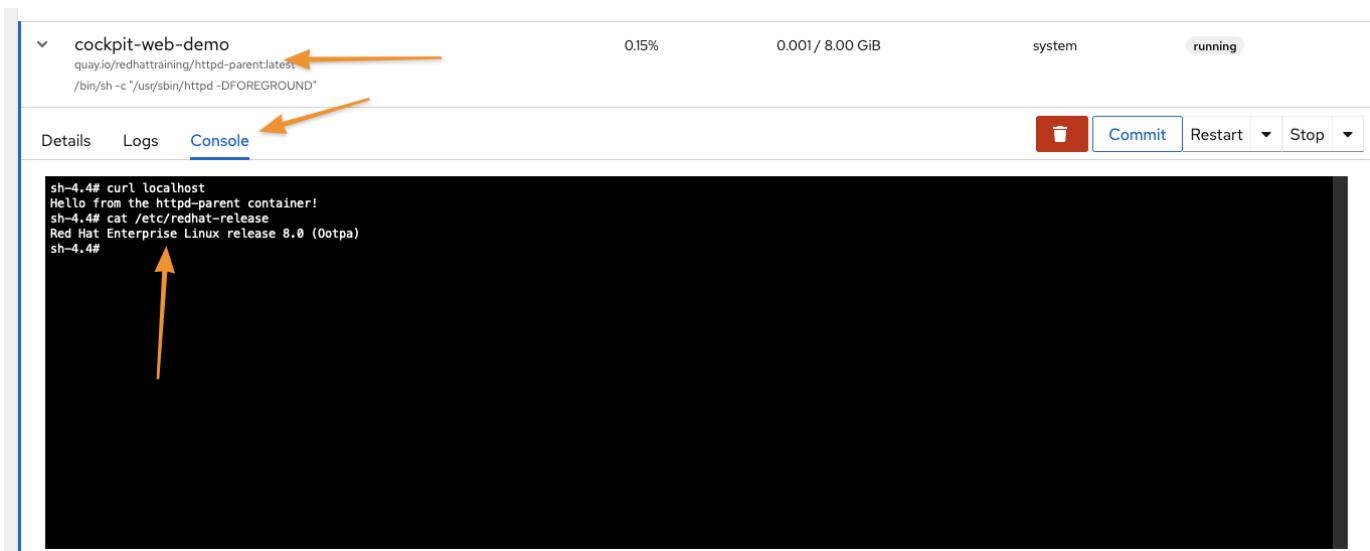


Figure 12. Container Console

### 2.5.1. Installing Cockpit Podman Plugins

In order to utilize Cockpit to manage containers with Podman there must be two things that are in place.

#### Cockpit podman Requirements

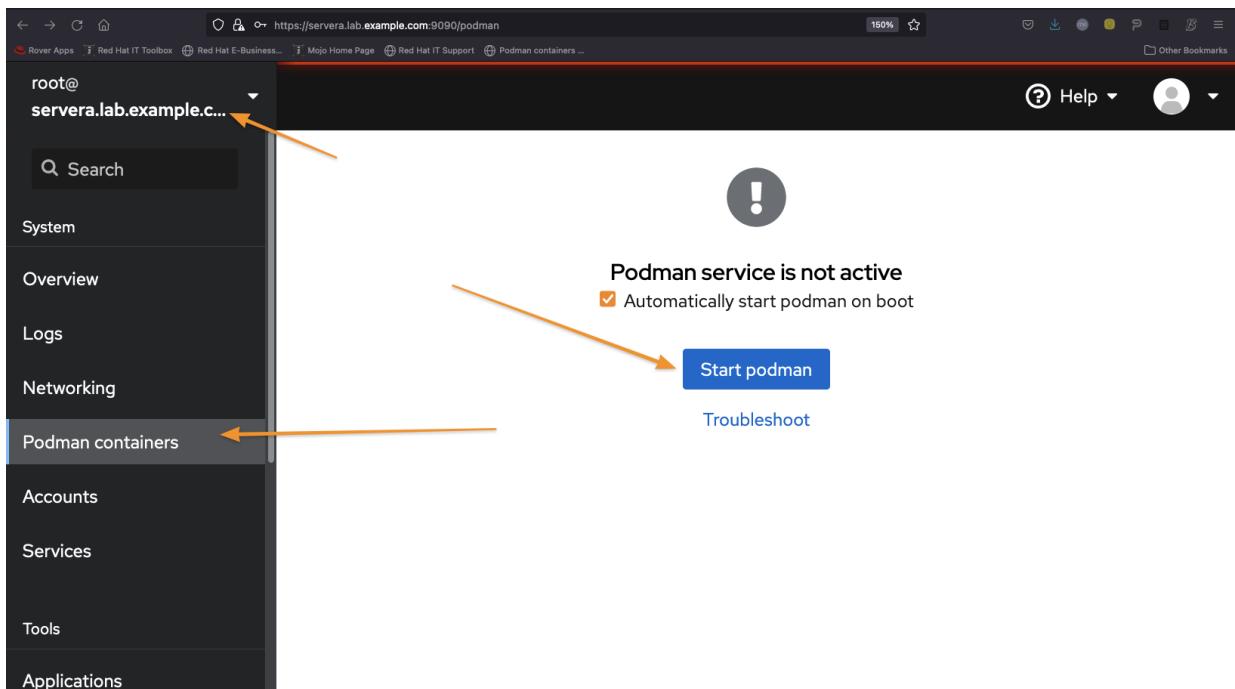
- Podman cockpit plugins
- Podman service started and running on the system
  1. Install **cockpit-podman** Modules

***Listing 33. Installation of Cockpit podman Plugins***

```
[root@servera ~]# yum install cockpit-podman  
... OUTPUT OMITTED ...  
  
Complete!  
[root@servera ~]#
```

**2. Enable podman Service in Cockpit**

- a. Sign into Cockpit
- b. Click "Podman containers"
- c. Click "Start podman"



*Figure 13. Cockpit - Podman Containers (Setup)*

The screenshot shows the Cockpit web console interface. On the left, a sidebar menu includes options like System, Overview, Logs, Networking, Podman containers (which is highlighted with a blue background), Accounts, Services, Tools, and Applications. The main content area is titled 'Containers'. It has a search bar at the top labeled 'Type to filter...'. Below the title, there's a 'Show' dropdown set to 'Only running' and a table with columns: Container, CPU, Memory, Owner, and State. Three containers are listed:

Container	CPU	Memory	Owner	State
clairv4 registry.redhat.io/quay/clair-rhel8:v3.6.0-70	0.38%	0.001 / 8.00 GiB	system	running
postgresql-clairv4 registry.redhat.io/rhel8/postgresql-10:1 run-postgresql	0.29%	0.001 / 8.00 GiB	system	running
postgresql-quay registry.redhat.io/rhel8/postgresql-10:1 run-postgresql	0.05%	0.001 / 8.00 GiB	system	running

Figure 14. Cockpit - Podman Containers (In-Use)

*Ensure Cockpit is both Installed and Enabled*

*Listing 34. Installing Cockpit*



```
[root@servera ~]# yum install cockpit
```

```
... OUTPUT OMITTED ...
```

```
subscription-manager-cockpit-1.28.13-2.el8.noarch
tracer-common-0.7.5-2.el8.noarch
```

Complete!

*Listing 35. Enabling Cockpit Socket*

```
[root@servera ~]# systemctl enable cockpit.socket --now
Created symlink
/etc/systemd/system/sockets.target.wants/cockpit.socket →
/usr/lib/systemd/system/cockpit.socket
```

## 2.5.2. Using Cockpit to Manage Containers

Once installed and enabled, the **Podman containers** plugin can begin managing containers and images. There are a few things to remember and consider as the **plugin** can't do everything.

*Plugin Considerations and Limitations*

- **Registries:** Image registries must be setup and specified ahead of time in the `/etc/containers/registries.conf` file.
  - Image registries must be in the configuration file in order for images to be downloaded and used. If the configuration file is updated after the **podman** service has been started, the service must be "restarted" in order for Cockpit to see the changes.
  - There are some difficulties with providing authentication information to registries as there isn't currently an interface to login to the registry within Cockpit.
- **Storage Volumes:** It might be necessary to have volumes and permissions setup to properly mount/map the volumes into the running containers

#### *Registry Credentials*

In order for Podman to authenticate, you must provide credentials. There needs to be a config file and you may also need to login to **podman** from the command line.

#### *Listing 36. .docker/config.json Config File*

```
[student@servera ~]$ cat .docker/config.json
{
    "ServerURL": "https://registry.redhat.io/v1",
    "Username": "RHNID",
    "Secret": "RHNPassword"
}
```



#### *Listing 37. CLI Login*

```
[student@servera ~]$ podman login registry.redhat.io
Authenticating with existing credentials...
Existing credentials are valid. Already logged in to registry.redhat.io
```

**Example 6. LAB: Container Image Management with Cockpit**

1. Navigate to the **Images** section and select **Get new image**

The screenshot shows the Cockpit web interface. On the left, a sidebar menu lists various system sections: System, Overview, Logs, Networking, Podman containers (which is currently selected and highlighted in blue), Accounts, Services, Tools, and Applications. A search bar labeled "Search" is at the top of the sidebar. The main content area is titled "Images". It features a search bar at the top with the placeholder "Type to filter...". Below it is a table with columns: Name, Created, Size, and Owner. The table lists several container images from the Red Hat registry. An orange arrow points from the "Podman containers" menu item in the sidebar to the "Images" table. Another orange arrow points from the "Get new image" button in the top right corner of the table area.

Name	Created	Size	Owner
registry.redhat.io/quay/clair-rhel8:v3.5.1	04/19/2021	264 MiB	system
registry.redhat.io/quay/clair-rhel8:v3.6.0-70	10/12/2021	148 MiB	system
registry.redhat.io/quay/quay-rhel8:v3.5.1	04/19/2021	1.24 GiB	system
registry.redhat.io/quay/quay-rhel8:v3.6.0-62	10/12/2021	928 MiB	system
registry.redhat.io/rhel8/postgresql-10:1	09/15/2021	453 MiB	system
registry.redhat.io/rhel8/redis-5:1	09/15/2021	300 MiB	system

Figure 15. Cockpit - Podman Container Images

The screenshot shows a modal dialog titled "Search for an image". It has three input fields: "Search for" (with placeholder "Search by name or desc..."), "in" (with dropdown menu showing "All registries"), and "Tag" (with input field containing "latest"). Below these fields is a message: "No images found" followed by the sub-instruction "Please start typing to look for images.". At the bottom of the dialog are two buttons: "Download" and "Cancel". In the background, the main Cockpit interface is visible, showing a table of images with one entry: "registry.redhat.io/rhel8/postgresql-10:1" (Created: 09/15/2021, Size: 453 MiB, Owner: system).

Figure 16. Search for Images

2. Select the registry to search, tags, and image name/description, then click "Download"
- a. Search for the HTTPD-Parent image in the **quay.io** registry.

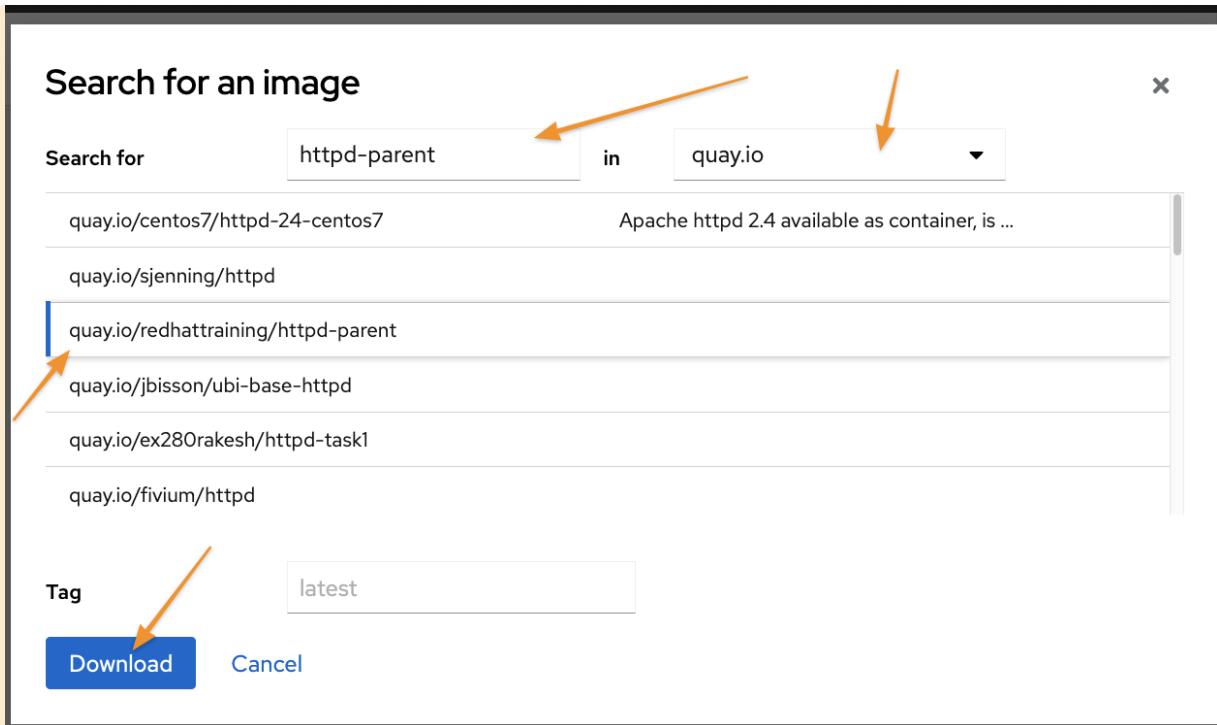


Figure 17. HTTPD 2.4 Parent Image from Red Hat Training - Quay.io

#### Adding the quay.io Registry to /etc/containers/registries.conf

Update the **registries.conf** file and then restart the **podman** service.

#### Listing 38. Edit /etc/containers/registries.conf

```
# To ensure compatibility with docker we've included docker.io
# in the default search list. However Red Hat
# does not curate, patch or maintain container images from the
# docker.io registry.
[registries.search]
registries = ['registry.access.redhat.com',
'registry.redhat.io', 'docker.io', 'quay.io']①
```



① Adding quay.io

#### Listing 39. Restart podman

```
[root@servera ~]# systemctl restart podman.service
```

### Registry Authentication

It is important to keep in mind, images requiring authentication to a registry will provide errors in Cockpit as you will be accessing the registry as an unauthenticated user.

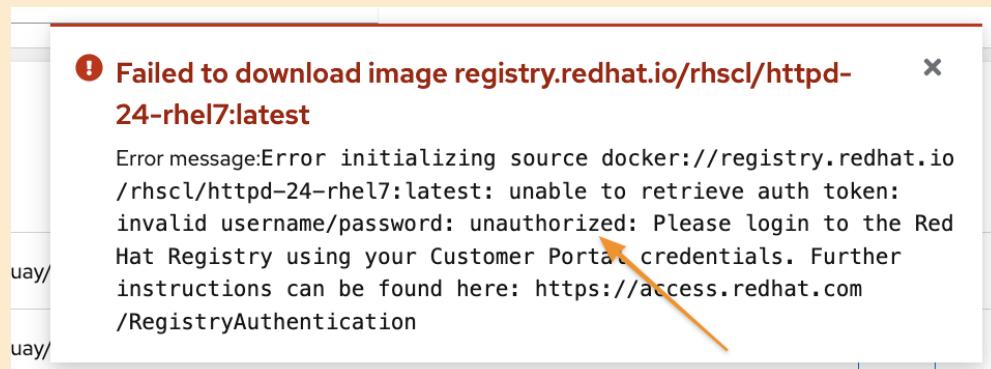


Figure 18. Registry Authentication Issues

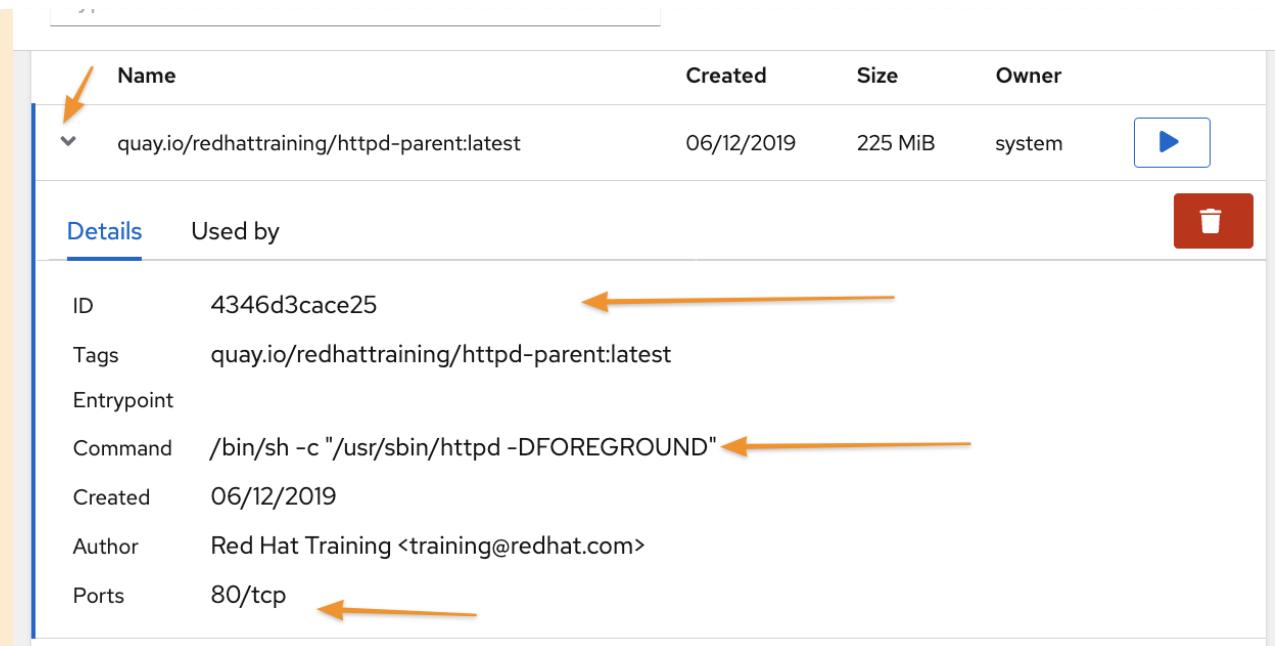
Based on timeline and platform difficulties, images downloaded in the Red Hat Web Management Console will be coming from public (unauthenticated) registries.

3. Confirm image was downloaded.

Images					<a href="#">+ Get new image</a>
	Name	Created	Size	Owner	
>	quay.io/redhattraining/httpd-parent:latest	06/12/2019	225 MiB	system	<a href="#">▶</a>
>	registry.redhat.io/quay/clair-rhel8:v3.5.1	04/19/2021	264 MiB	system	<a href="#">▶</a>
>	registry.redhat.io/quay/clair-rhel8:v3.6.0-70	10/12/2021	148 MiB	system	<a href="#">▶</a>
>	registry.redhat.io/quay/quay-rhel8:v3.5.1	04/19/2021	1.24 GiB	system	<a href="#">▶</a>
>	registry.redhat.io/quay/quay-rhel8:v3.6.0-62	10/12/2021	928 MiB	system	<a href="#">▶</a>
>	registry.redhat.io/rhel8/postgresql-10:1	09/15/2021	453 MiB	system	<a href="#">▶</a>

Figure 19. Images in Local Registry

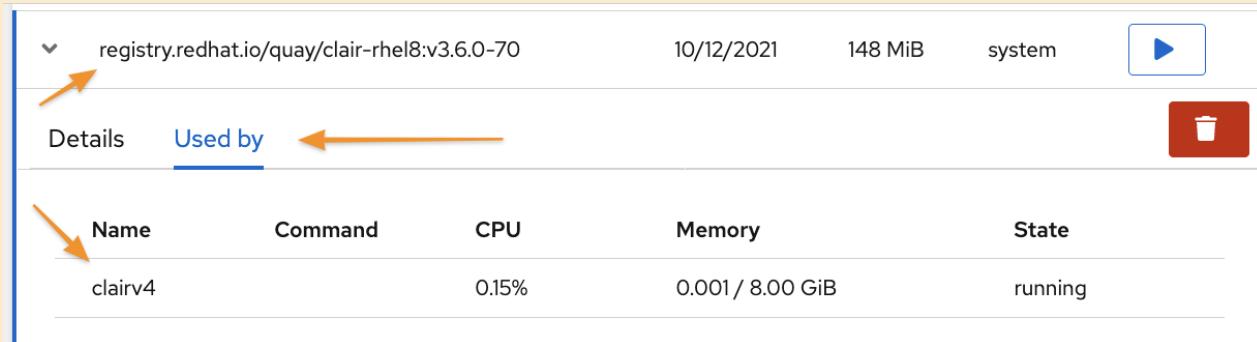
4. Explore image details by clicking the > and expanding the available information.



The screenshot shows a table of container images. The first row has a dropdown arrow next to 'Name' and a play button icon. The second row shows the image details for 'quay.io/redhattraining/httpd-parent:latest'. The 'Details' tab is selected, showing fields like ID, Tags, Entrypoint, Command, Created, Author, and Ports. Orange arrows point from the labels 'ID', 'Command', and 'Ports' to their corresponding values in the table.

Name	Created	Size	Owner
quay.io/redhattraining/httpd-parent:latest	06/12/2019	225 MiB	system
<b>Details</b> Used by			
ID	4346d3cace25		
Tags	quay.io/redhattraining/httpd-parent:latest		
Entrypoint			
Command	/bin/sh -c "/usr/sbin/httpd -DFOREGROUND"		
Created	06/12/2019		
Author	Red Hat Training <training@redhat.com>		
Ports	80/tcp		

Figure 20. HTTPD-Parent Image Details - Image Information



The screenshot shows a table of containers. The first row has a dropdown arrow next to 'Name' and a play button icon. The second row shows the container details for 'clairv4'. The 'Used by' tab is selected, showing the container name, command, CPU usage, memory usage, and state. Orange arrows point from the labels 'Name', 'Command', and 'CPU' to their corresponding values in the table.

Name	Command	CPU	Memory	State
clairv4		0.15%	0.001 / 8.00 GiB	running

Figure 21. Clair Image Details - Containers Using Image

### Container Management with Cockpit

It is possible to manage both **rootless** and **root-based** containers in Cockpit depending on the user you are using to access the management console. In order to create and launch new containers from Cockpit, the image must already be downloaded and existing in the local image registry.

**Example 7. LAB: Container Creation**

1. Select the image to be used to launch the container by clicking the **Play** button.

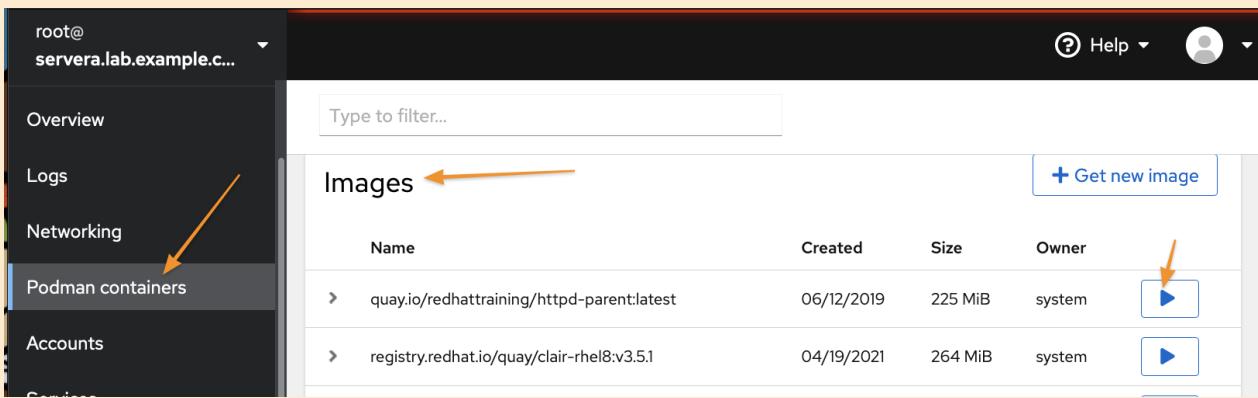


Figure 22. HTTPD-Parent Image Selection

2. Enter details you wish to use for your image and click "Run"
- a. Container Name
  - b. Port mapping
  - c. Volume mapping

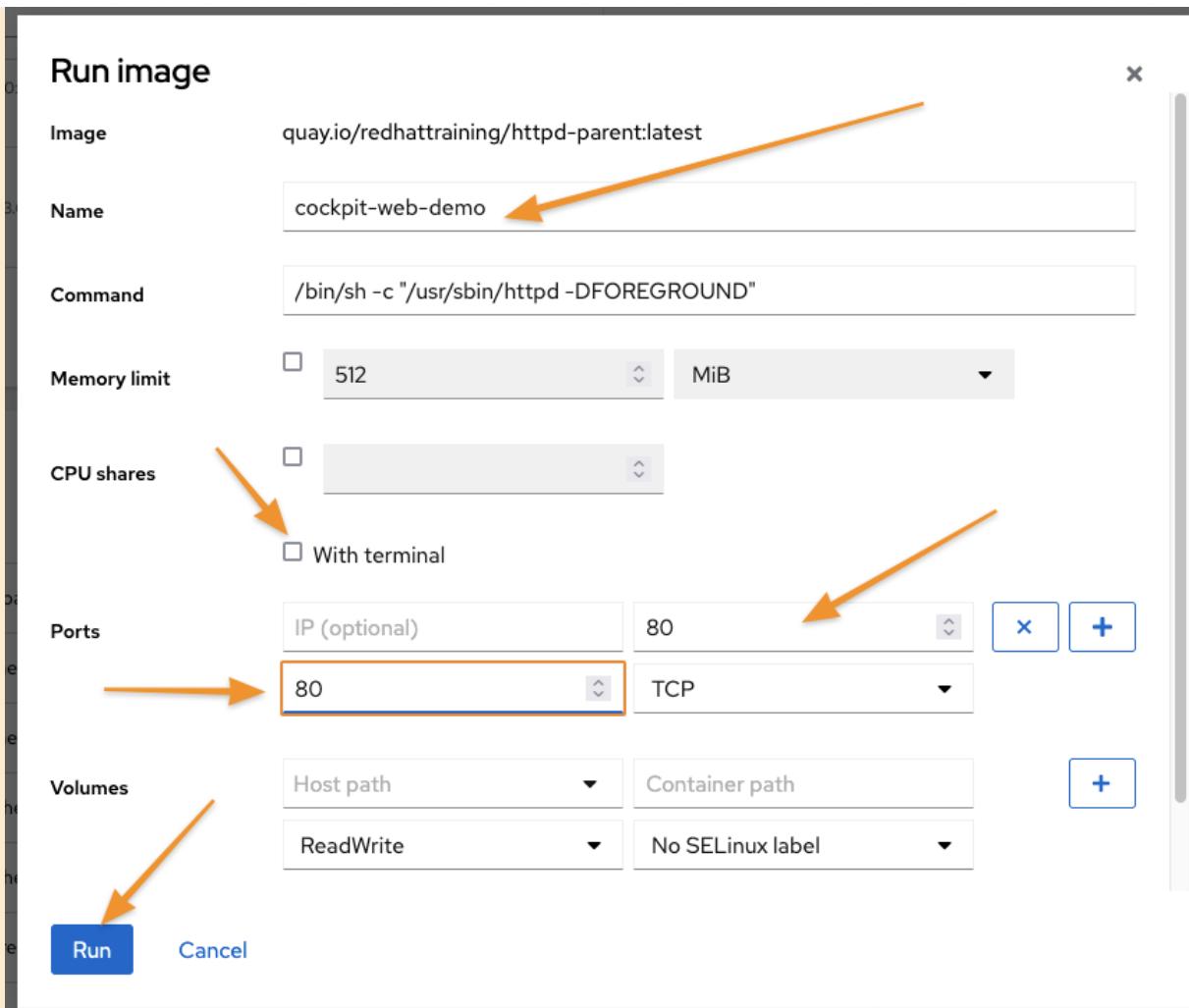


Figure 23. Launching a Demo Container

**NOTE Regarding With terminal**

For several containers, you should **uncheck** the ***With terminal*** option as this could cause the container to exit unexpectedly.

### 3. Verify container was launched

Container	CPU	Memory	Owner	State
clairv4 registry.redhat.io/quay/clair-rhel8:v3.6.0-70	0.15%	0.001 / 8.00 GiB	system	running
cockpit-web-demo quay.io/redhattraining/httpd-parent:latest /bin/sh -c "/usr/sbin/httpd -DFOREGROUND"	0.14%	0.001 / 8.00 GiB	system	running
postgresql-clairv4 registry.redhat.io/quay/postgresql-101 run-postgresql	0.07%	0.001 / 8.00 GiB	system	running

Figure 24. Running Container Verification

cockpit-web-demo  
quay.io/redhattraining/httpd-parent:latest  
/bin/sh -c "/usr/sbin/httpd -DFOREGROUND"

0.13% 0.001 / 8.00 GiB system running

**Details** Logs Console Commit Restart Stop

ID: d22db15153025d5e5966f8651755f3b5b2fa7120376ba6a8d9f0da6d1c8b9f77  
Created: Today at 11:46 AM  
Image: quay.io/redhattraining/httpd-parent:latest  
Command: /bin/sh -c "/usr/sbin/httpd -DFOREGROUND"  
State: Up since Today at 11:46 AM  
Ports: 0.0.0.0:80 → 80/tcp  
IP address: 10.88.0.40  
IP prefix length: 16  
Gateway: 10.88.0.1  
MAC address: 5a:fb:08:cb:23:71

Figure 25. Running Container Details

#### 4. Allow connection to container externally through Firewall

- Click **Networking**
- Click **Edit rules and zones** for the Firewall

Logs  
Networking  
Podman containers  
Accounts  
Services  
Interfaces

11:47 11:48 11:49 11:50 11:51 11:47 11:48 11:49 11:50 11:51

Networking Firewall Edit rules and zones

1 active zone

Add bond Add team Add bridge Add VLAN

Figure 26. Using Cockpit to Manage Firewalls

#### 5. Click **Add Services** and either select an existing service or create custom ports, then click the **Add Services** button on the **Add services to public zone**.

- Verify the service was added

root@server.lab.example.com... Networking > Firewall

Firewall

public zone Interfaces enp1s0

Service: ssh, dhcpv6-client, cockpit, Additional ports

Add services to public zone

Services Custom ports

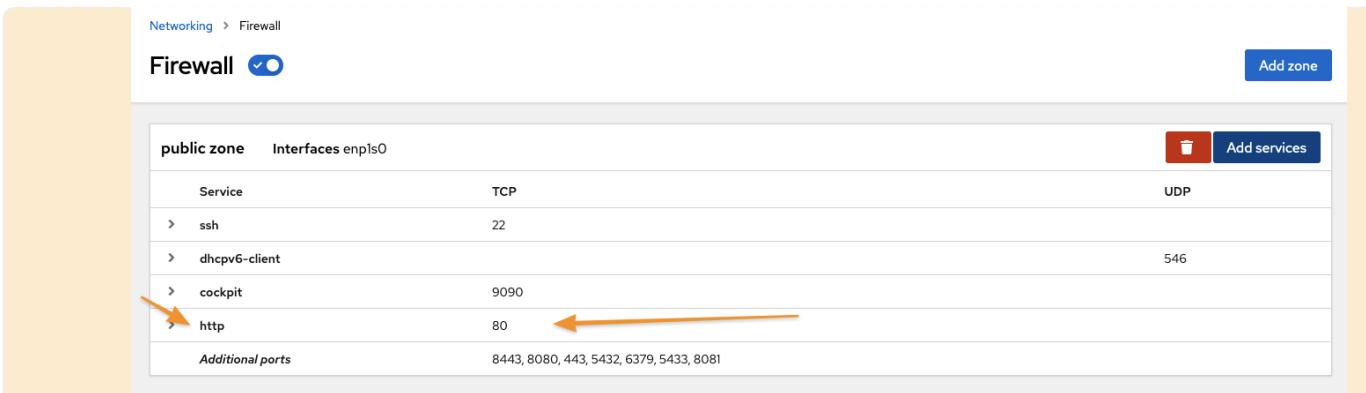
Filter services: http

http: TCP-80

Add services Cancel

UDP 546

Figure 27. Add the HTTP Service



The screenshot shows the Cockpit Firewall configuration. Under the 'public zone' tab, there is a table of services. The 'http' service is highlighted with an orange arrow pointing to its port number, 80. The table also lists other services like ssh, dhcpcv6-client, cockpit, and additional ports.

Service	TCP	UDP
ssh	22	
dhcpcv6-client		546
cockpit	9090	
http	80	
Additional ports	8443, 8080, 443, 5432, 6379, 5433, 8081	

Figure 28. HTTP Service Available

- Verify the container is running and hosting the website via the console and through a web browser.

*Listing 40. Local Container Verification*

```
sh-4.4# curl localhost
Hello from the httpd-parent container!
sh-4.4#
```



The screenshot shows the Cockpit interface with the 'Podman containers' menu item selected. A container named 'cockpit-web-demo' is listed, and its console output is shown in a terminal window. The output displays the message 'Hello from the httpd-parent container!'. Arrows point from the 'Console' tab in the container card to the terminal window and the output text.

Figure 29. Container Console in Cockpit



The screenshot shows a Firefox browser window with the URL 'servera' in the address bar. The page content displays the message 'Hello from the httpd-parent container!', which corresponds to the output seen in the Cockpit console.

Figure 30. Application Verification in Firefox

**Example 8. LAB: Saving a Container Image from a Running Container**

1. Select the container to use as a base for the image

- a. Click **Commit**

The screenshot shows a list of containers. The first container is 'cockpit-web-demo' with the following details:

- ID: 4c76fe54f535385fb5e22928bbd44091l769b503daecb29a95e4494b63ee05
- Created: Today at 1:10 PM
- Image: quay.io/redhattraining/httpd-parent:latest
- Command: /bin/sh -c "/usr/sbin/httpd -DFOREGROUND"
- State: Up since Today at 1:10 PM
- Ports: 0.0.0.0:80 → 80/tcp
- IP address: 10.88.0.49
- IP prefix length: 16
- Gateway: 10.88.0.1
- MAC address: 0e:ae:e5:84:d4:fd

At the top right of the container card, there is a 'Commit' button, which is highlighted with a blue box and an arrow pointing to it.

Figure 31. Creating an new image from a container.

2. Select and complete the **Commit image** form

- a. Specify **Image name**
  - b. Other fields are optional

The screenshot shows the 'Commit image' dialog box. The fields are filled as follows:

- Container name: cockpit-web-demo
- Format: oci (radio button selected)
- Image name: cockpit-image-test
- Tag: (empty)
- Author: (empty)
- Message: (empty)
- Command: (empty)
- Checkboxes:
  - Pause the container
  - Set container on build variables

At the bottom left is a 'Commit' button, which is highlighted with a blue box and an arrow pointing to it. To its right is a 'Cancel' button.

Figure 32. Committing image

*Images from a Running Container*

It is possible to create images from running container. If the container is running, it will be paused briefly while the image is created.

### 3. Verify the image was created

Images					<a href="#">+ Get new image</a>
Name	Created	Size	Owner		
> localhost/cockpit-image-test:latest	Today at 1:32 PM	225 MiB	system		
> quay.io/redhattraining/httpd-parent:latest	06/12/2019	225 MiB	system		

Figure 33. Image Verification

*Image Tags*

If there is no tag specified when the image is being committed, **podman** will automatically add the **latest** tag to the image.

### Example 9. LAB: Container Cleanup

1. Select the container to stop and remove.
  - a. If you just want to stop a running container, click **Stop**
  - b. To stop and remove the container, click the **Trashcan**

The screenshot shows the Cockpit web console interface. In the center, there's a detailed view of a container named 'cockpit-web-demo'. At the top, there's a summary bar with the container name, image, resource usage (0.15% CPU, 0.001 / 8.00 GiB memory), and state ('running'). Below this is a navigation bar with 'Details', 'Logs', and 'Console' tabs. The 'Details' tab is selected. Underneath, a table lists various container properties like ID, Created, Image, Command, State, Ports, IP address, IP prefix length, Gateway, and MAC address. In the top right corner of the main content area, there are several buttons: 'Commit', 'Restart', 'Stop', and a trashcan icon. Orange arrows highlight the 'Stop' button and the trashcan icon.

Figure 34. Container Cleanup

#### *Deleting a Running Container*

Running containers must be stopped before they can be deleted or removed. If a container is running, and you click the **Trashcan** it will stop and delete the container. This is similar to performing a **podman rm -f <Container>** as it will force stop and then remove the container.



The screenshot shows the Cockpit web console with a modal dialog box overlaid. The dialog box contains the text 'Please confirm forced deletion of cockpit-web-demo' and 'Container is currently running.' It has two buttons: 'Force delete' (highlighted with an orange arrow) and 'Cancel'. In the background, the container details for 'cockpit-web-demo' are visible, including its ID, creation time, image, command, state, ports, IP address, IP prefix length, gateway, and MAC address. The trashcan icon in the top right corner of the main content area is also highlighted with an orange arrow.

Figure 35. Force Removing a Running Container

#### 2.5.2.1. Using Cockpit to Manage Containers (Rootless)

Just as Podman can manage containers as a "rootless" user, it is also possible to manage containers within Cockpit as a "rootless" user. All the same principles apply in Cockpit as they do with **podman**.

Additionally, Cockpit has multiple options if the user happens to be in the **sudoers** file.

**Example 10. LAB: Exploring Cockpit Containers as the *student* User**

1. Login to Cockpit as the Student user

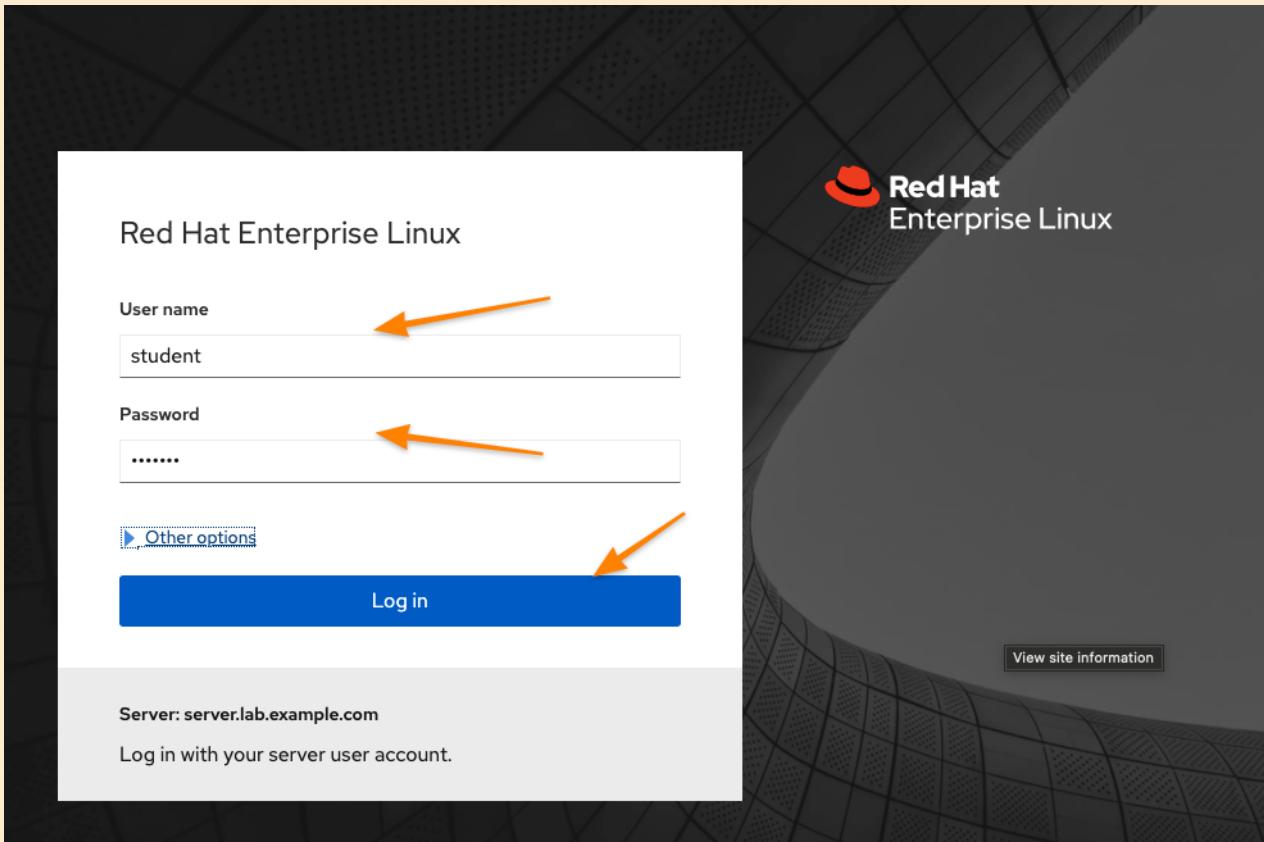


Figure 36. Cockpit Student Login

2. Navigate to the **Podman containers** section

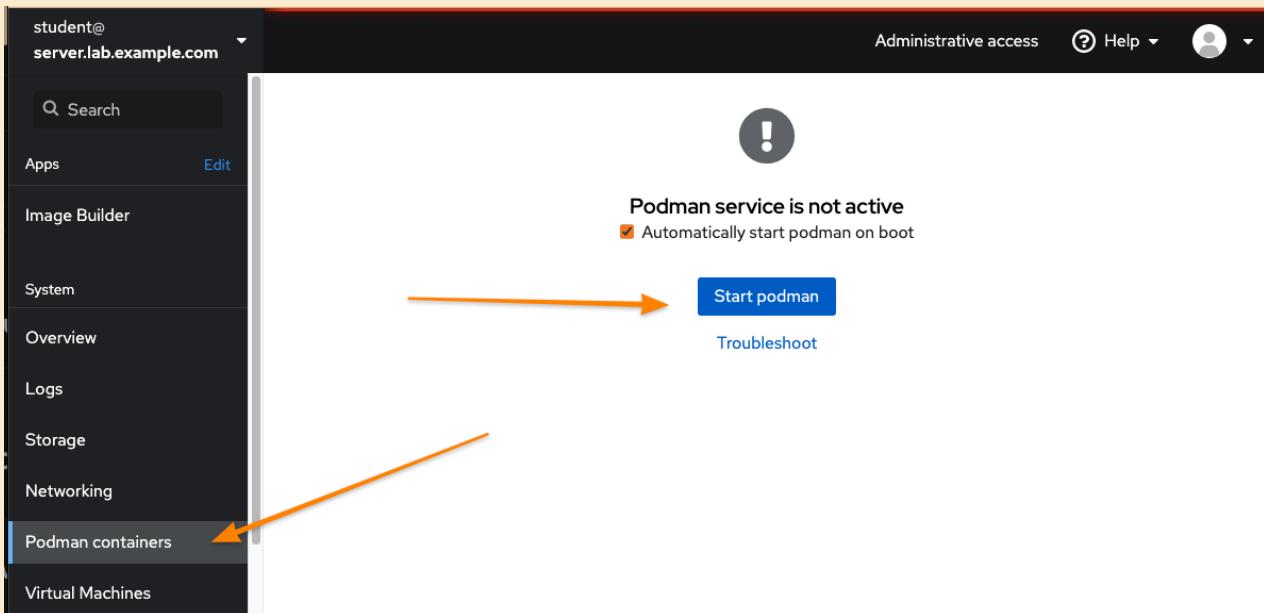
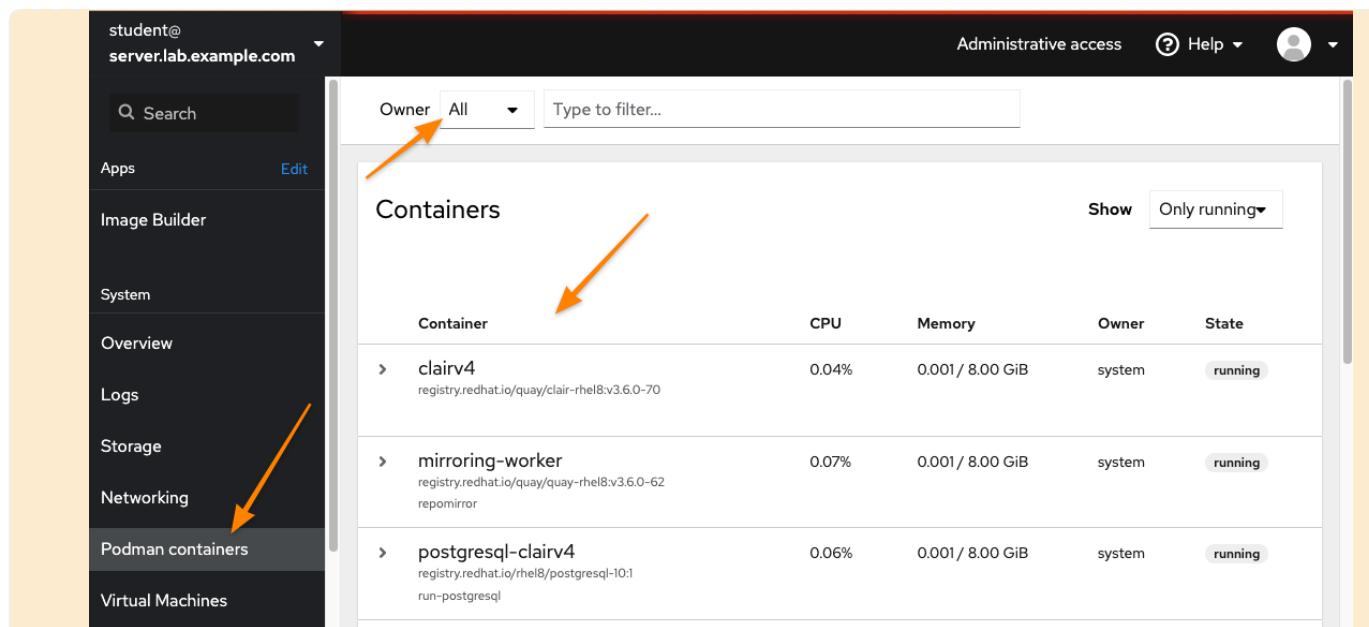


Figure 37. Podman containers plugin

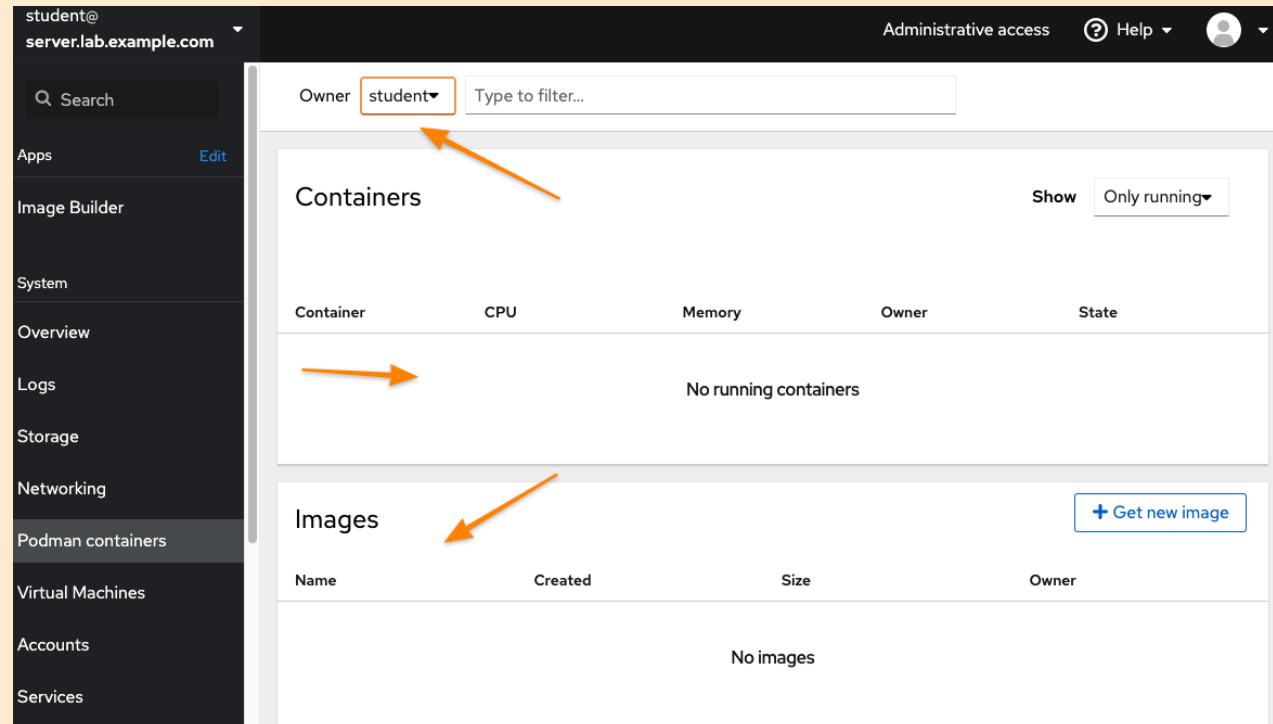
3. Click **Start podman**



Containers

Container	CPU	Memory	Owner	State
clairv4 registry.redhat.io/quay/clair-rhel8:v3.6.0-70	0.04%	0.001 / 8.00 GiB	system	running
mirroring-worker registry.redhat.io/quay/quay-rhel8:v3.6.0-62 repomirror	0.07%	0.001 / 8.00 GiB	system	running
postgresql-clairv4 registry.redhat.io/rhel8/postgresql-10:1 run-postgresql	0.06%	0.001 / 8.00 GiB	system	running

Figure 38. Podman containers - Owner All View



Containers

Container	CPU	Memory	Owner	State
No running containers				

Images

Name	Created	Size	Owner
No images			

Figure 39. Podman containers - Owner student View

The screenshot shows the Red Hat Web Console interface for managing containers. On the left, a sidebar lists various system sections: Apps, Image Builder, System, Overview, Logs, Storage, Networking, Podman containers (which is currently selected), and Virtual Machines. The main area is titled 'Containers' and displays three running containers:

Container	CPU	Memory	Owner	State
clairy4 registry.redhat.io/quay/clair-rhel8:v3.6.0-70	0.07%	0.001 / 8.00 GiB	system	running
mirroring-worker registry.redhat.io/quay/quay-rhel8:v3.6.0-62 repomirror	0.07%	0.001 / 8.00 GiB	system	running
postgresql-clairy4 registry.redhat.io/rhel8/postgresql-10:1 run-postgresql	0.04%	0.001 / 8.00 GiB	system	running

At the top of the container list, there are two filters: 'Owner' (set to 'System') and 'Type to filter...'. To the right, there is a 'Show' button followed by a dropdown menu set to 'Only running'.

Figure 40. Podman containers - Owner System View



The **student** user is able to become root as it exists in the **sudoers** file. Based on Cockpit's abilities, the **student** user can see both containers and images owned by **student**, but it can also see containers and images owned by **system**.

Launching and managing containers is also performed the same way as the student user.

**Example 11. LAB: Working with Containers in Cockpit as student.**

1. Download a container image.
  - a. Registry: **registry.access.redhat.com**
  - b. Image: **ubi7**

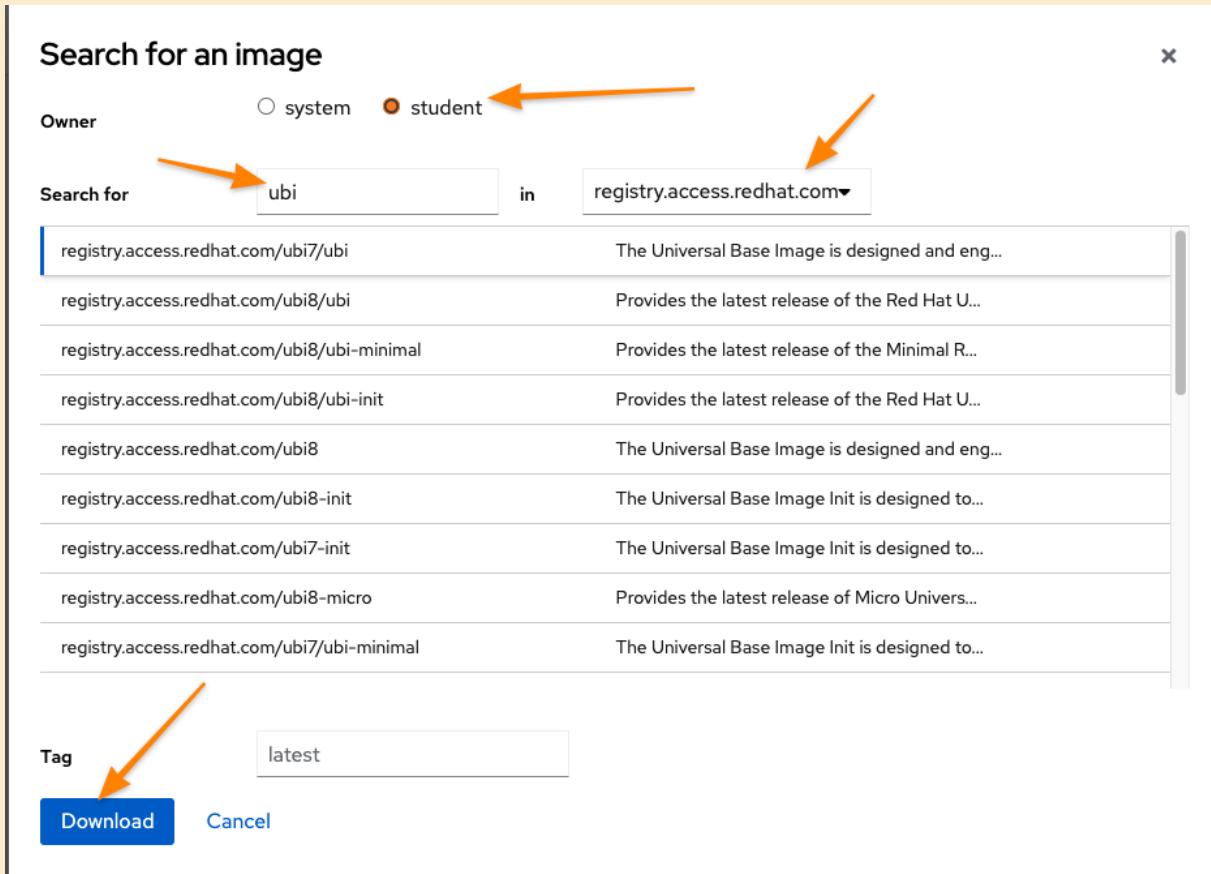


Figure 41. UBI7 Image Download

2. Launch a Container with the UBI7 Image
  - a. Container Name: ubi7-test
  - b. Image: **registry.access.redhat.com/ubi7/ubi:latest**

The screenshot shows the Red Hat Web Console interface. On the left, a sidebar menu includes options like Apps, Image Builder, System, Overview, Logs, Storage, Networking, Podman containers (which is highlighted with an orange arrow), Virtual Machines, and Accounts. The main content area is titled 'Containers' and shows a table with columns: Container, CPU, Memory, Owner, and State. A message at the top says 'No running containers'. Below it is another section titled 'Images' with a table. The table has columns: Name, Created, Size, and Owner. It lists one item: 'registry.access.redhat.com/ubi7/ubi:latest' (Created: 10/05/2021, Size: 206 MiB, Owner: student). A blue button labeled '+ Get new image' is visible above the table. An orange arrow points from the 'Podman containers' menu item to the 'Images' section.

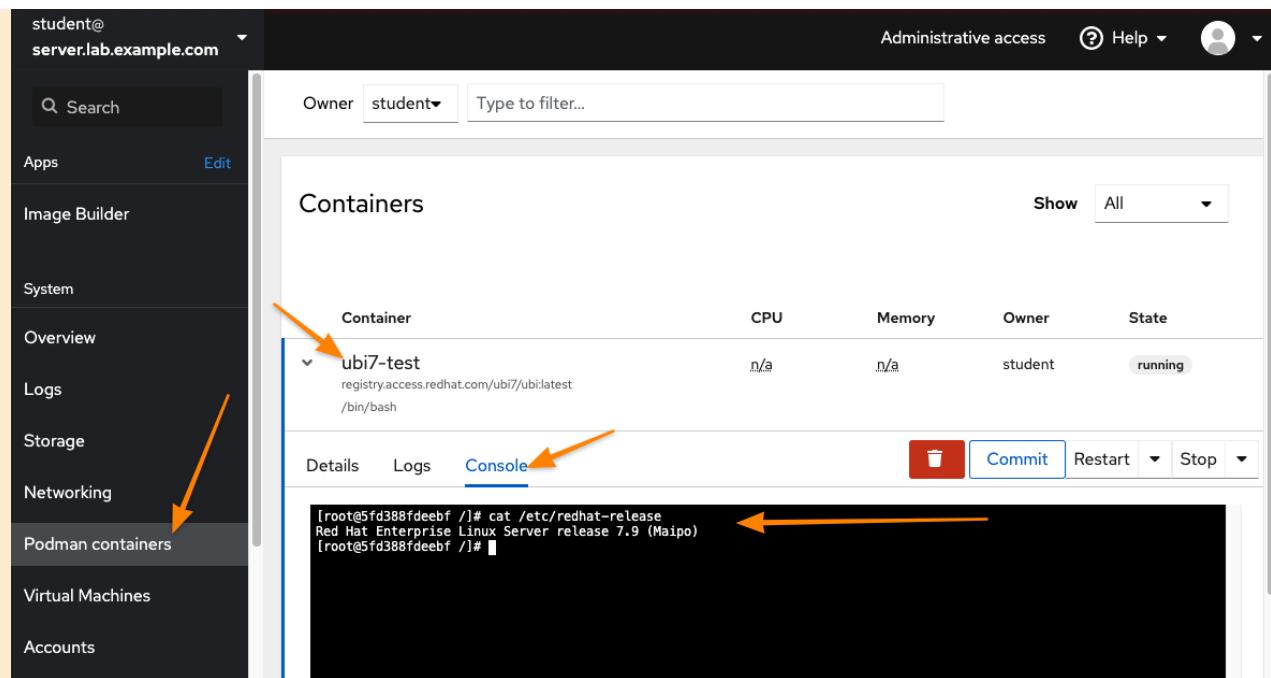
Figure 42. UBI7 Image - Local Repository

The screenshot shows the 'Run image' dialog box. It has the following fields:

- Image:** registry.access.redhat.com/ubi7/ubi:latest
- Name:** ubi7-test (highlighted with an orange arrow)
- Command:** /bin/bash
- Memory limit:** 512 MiB (unchecked)
- Ports:** IP (optional) and Host port (optional) both have '+' buttons. Under 'Container port', there is a dropdown set to TCP and a '+' button.
- Volumes:** Host path and Container path both have dropdown menus and '+' buttons. Under 'ReadWrite', there is a dropdown set to No SELinux label and a '+' button.

At the bottom are 'Run' and 'Cancel' buttons. Orange arrows point from the sidebar's 'Podman containers' link to the 'Images' section, from the 'Name' field to the 'With terminal' checkbox, and from the 'Volumes' section to the 'Run' button.

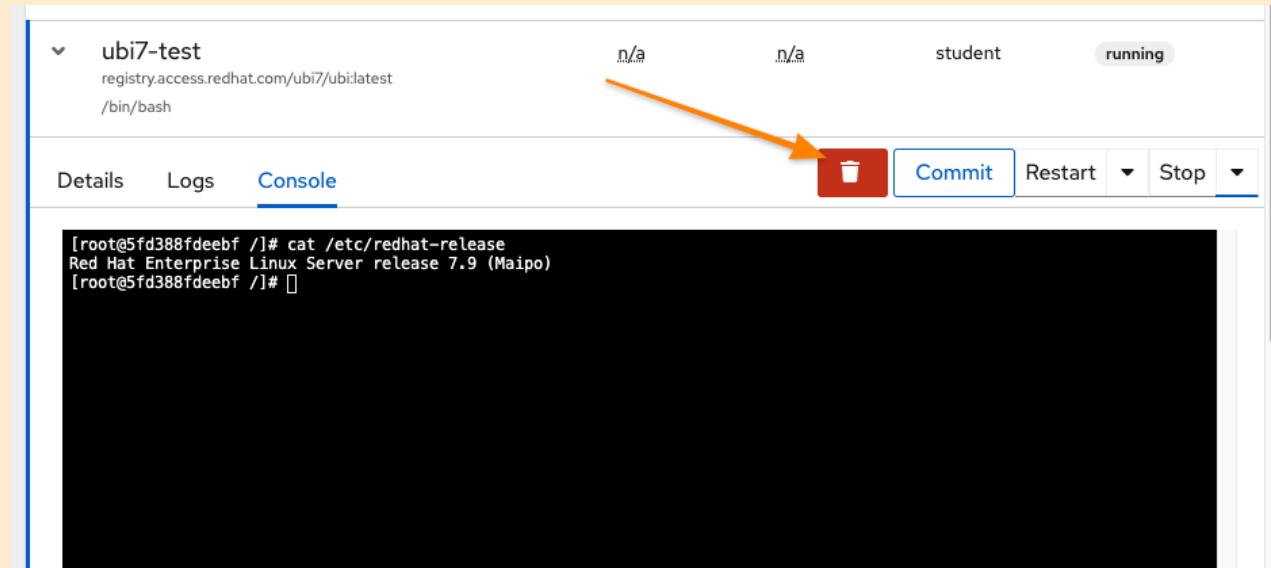
### 3. Explore the ubi7-test Container



The screenshot shows the Red Hat Container Management interface. On the left, a sidebar lists various management options: Apps, Image Builder, System, Overview, Logs, Storage, Networking, Podman containers (which is selected and highlighted in blue), Virtual Machines, and Accounts. The main panel is titled "Containers" and shows a table with one row for "ubi7-test". The table columns are Container, CPU, Memory, Owner, and State. The "ubi7-test" row has values: n/a, n/a, student, and running. Below the table, there are tabs for Details, Logs, and Console, with the Console tab currently active. The console window displays a terminal session with the following text:  
[root@5fd388fdeebf /]# cat /etc/redhat-release  
Red Hat Enterprise Linux Server release 7.9 (Maipo)  
[root@5fd388fdeebf /]#  
An orange arrow points from the sidebar's "Podman containers" option to the "Container" table. Another orange arrow points from the "Console" tab to the terminal window.

Figure 43. UBI7 Container - Interactive Console

#### 4. Cleanup running Container (*force delete*)



This screenshot shows the same interface as Figure 43, but the "ubi7-test" container is now listed as "deleted" in the table. The table columns are Container, CPU, Memory, Owner, and State. The "ubi7-test" row has values: n/a, n/a, student, and deleted. The "Console" tab is still active, showing the same terminal session as before. An orange arrow points from the "Deleted" status in the table to the red trash can icon in the action bar below the table.

Figure 44. UBI7 Container - Deletion

#### 5. Cleanup unneeded images

Name	Created	Size	Owner
registry.access.redhat.com/ubi7/ubi:latest	10/05/2021	206 MiB	student

**Details**    Used by

ID: 6f683d6ef7a8  
Tags: registry.access.redhat.com/ubi7/ubi:latest  
Entrypoint:  
Command: /bin/bash  
Created: 10/05/2021  
Author: Red Hat, Inc.  
Ports:

Figure 45. UBI7 Image - Deletion

6. Verify both the container and image have been deleted

Owner: student▼ Type to filter...

Containers

Show: All ▾

Container	CPU	Memory	Owner	State
No containers				

Images

+ Get new image

Name	Created	Size	Owner
No images			

Figure 46. Cleanup Successful

*References*

**Oracle Linux: Use Cockpit to Manage Podman Containers:** <https://docs.oracle.com/en/operating-systems/oracle-linux/8/tutorial-cockpit-podman/#Before-You-Begin>

**Managing your Podman containers with Cockpit on Fedora:** <https://www.tutorialworks.com/podman-monitoring-cockpit-fedora/>

# 3. Podman Pods and Container/Pod Resource Files

Podman has replaced Docker/Moby in RHEL 8 and will be the container management application moving forward. Podman (short for Pod Manager) is a daemon-less way to run containers and pods as a standard non-privileged user or as a privileged or "root" user. Podman also lends itself nicely to Kubernetes since it manages both containers and pods. Remember that the smallest item that Kubernetes/OpenShift manages is a pod and that pods consist of one or more containers.

## 3.1. Using and Leveraging Pods with Podman

Generally the concept of pod is associated with orchestration tools such as Red Hat OpenShift and Kubernetes. A pod is a collection of one or more containers and will always include an **Infrastructure** or **Infra** container. The **Infra** container is meant to serve as a placeholder for the namespace allowing the pod to remain while allowing containers to be stopped and started while the pod stays running.

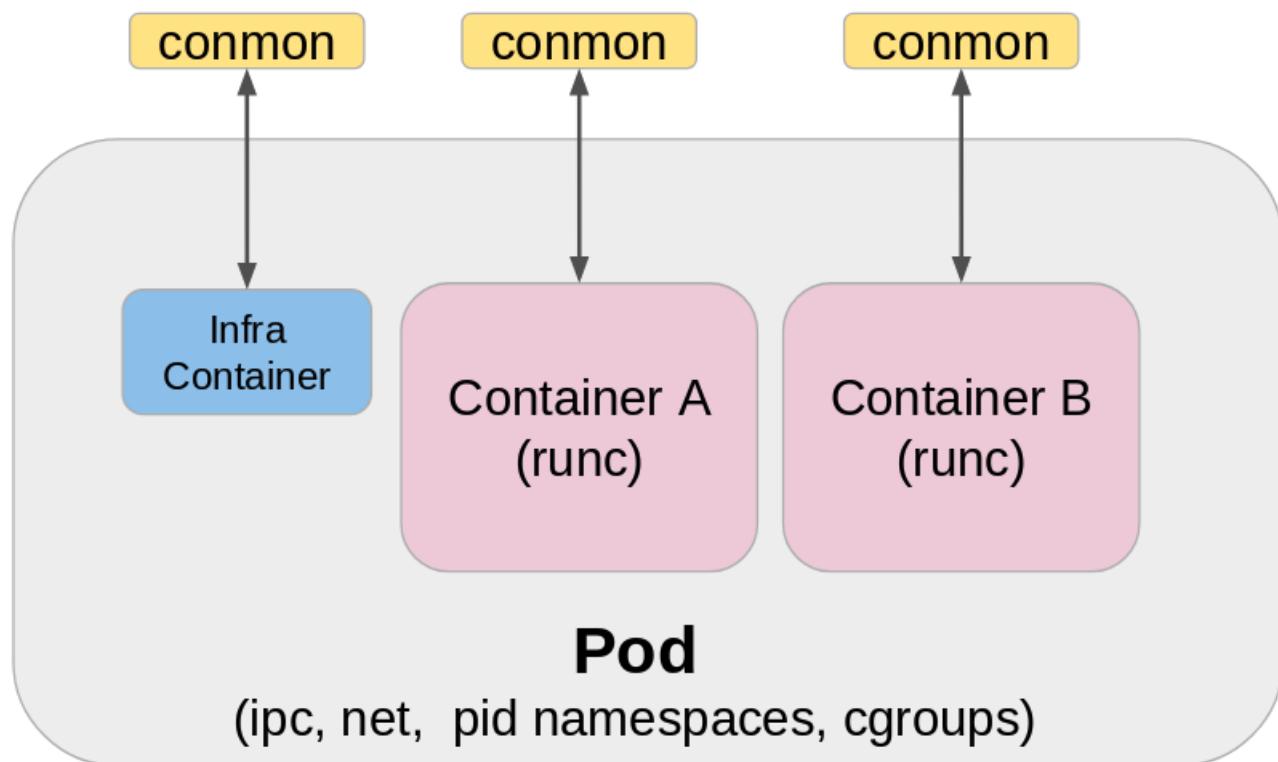


Figure 47. Example of a Pod

When using **pods** port bindings and network port mapping is done at the **pod** level, therefore this is assigned to the **infra** container. In order to change and modify ports on the container, it is necessary to delete the **pod** as once resources have been assigned, they can't be changed.

Podman is capable of managing pods with the **podman pod** command. It is possible to get assistance

and help from **podman** directly by using the **podman pod --help** command.

*Listing 41. podman Pod Management*

```
[root@server ~]# podman pod --help
Manage pods
```

Description:

Pods are a group of one or more containers sharing the same network, pid and ipc namespaces.

Usage:

```
podman pod [command]
```

Available Commands:

create	Create a new empty pod
exists	Check if a pod exists in local storage
inspect	Displays a pod configuration
kill	Send the specified signal or SIGKILL to containers in pod
pause	Pause one or more pods
prune	Remove all stopped pods and their containers
ps	List pods
restart	Restart one or more pods
rm	Remove one or more pods
start	Start one or more pods
stats	Display a live stream of resource usage statistics for the containers in one or more pods
stop	Stop one or more pods
top	Display the running processes of containers in a pod
unpause	Unpause one or more pods

Additional more detailed help is available by using the **podman pod <command> --help** for command specific help.

**Listing 42. podman pod stop Help**

```
[root@server ~]# podman pod stop --help  
Stop one or more pods
```

**Description:**

The pod name or ID can be used.

This command will stop all running containers in each of the specified pods.

**Usage:**

```
podman pod stop [options] POD [POD...]
```

**Examples:**

```
podman pod stop mywebserverpod  
podman pod stop --latest  
podman pod stop --time 0 490eb 3557fb
```

**Options:**

-a, --all	Stop all running pods
-i, --ignore	Ignore errors when a specified pod is missing
-l, --latest	Act on the latest container podman is aware of
	Not supported with the "--remote" flag
--pod-id-file stringArray	Write the pod ID to the file
-t, --time uint	Seconds to wait for pod stop before killing the
container (default 10)	

### 3.1.1. Creating a Pod and Adding a Container

Podman pods are created just like containers. It is possible to skip the **podman pod create** command to create the pod and then subsequently run the pod with the **podman pod run** command. Instead, you can directly use the **podman pod run** command to create and run the pod. When creating a pod with a container as a single command, you MUST use the **new:<name>** directives to name a new Pod letting **podman** know you aren't attempting to reuse an existing Pod.

***Listing 43. Creating a Container and Running in a Pod (Multiple Commands)***

```
[root@server ~]# podman pod create --name rht-demo -p 8080:80 ①
e18e307bab06dd45d3cc5a90472ea075979aacb3adfc2071dfd68dca284b03a3

[root@server ~]# podman run -d --pod rht-demo --name webserver
quay.io/redhattraining/httpd-parent ②
deeb7e789eafa99f45e6774f6ec0f124b1abbf9d211946c2fa2360e515db1c96

[root@server ~]# curl localhost:8080
Hello from the httpd-parent container!

[root@server ~]# podman pod ps
POD ID      NAME      STATUS     CREATED      INFRA ID      # OF CONTAINERS
e18e307bab06  rht-demo  Running   3 minutes ago  3113f296524c  2

[root@server ~]# podman ps
CONTAINER ID  IMAGE                                     COMMAND          CREATED
STATUS        PORTS           NAMES
3113f296524c  registry.access.redhat.com/ubi8/pause:latest    3
minutes ago   Up About a minute ago  0.0.0.0:8080->80/tcp  e18e307bab06-infra ③
deeb7e789eaf  quay.io/redhattraining/httpd-parent       /bin/sh -c /usr/s...  About a
minute ago   Up About a minute ago  0.0.0.0:8080->80/tcp  webserver ④
```

① Create the Pod and port forwards for the Pod

② Create a container to run in the Pod. The port forward comes from the **pod** definition.

③ **Infra** Container for the Pod

④ Actually running webserver container. **NOTE: port 8080 forwards to container port 80.**

***Listing 44. Creating a Container and Running in a Pod (Single Command)***

```
[root@server ~]# podman run -d --pod new:rht-demo -p 80:80 quay.io/redhattraining/httpd-
parent
17ac5ef7876afdc52a356f14284411617d9e2c7d8ab58ea5545cec61ab7547

[root@server ~]# curl localhost
Hello from the httpd-parent container!
```

### 3.1.2. Cleaning up Pods and Containers

It is necessary to do cleanup and maintenance on Pods and Containers as this doesn't occur automatically with Podman as it does with Openshift/Kubernetes. Just as with managing regular containers, managing pods requires stopping and deleting Pods before those pods can be removed.

It is possible to delete and remove the pod and running containers in a single instance. In order to

delete and remove pods, it is necessary to know the names and status of pods which can be found with the **podman pod ps** command.

*Listing 45. Podman Pod Cleanup*

```
[root@server ~]# podman pod ps ①
POD ID      NAME      STATUS   CREATED      INFRA ID      # OF CONTAINERS
e18e307bab06  rht-demo  Running  12 minutes ago  3113f296524c  2

[root@server ~]# podman ps ②
CONTAINER ID  IMAGE                                     COMMAND          CREATED
STATUS        PORTS          NAMES
3113f296524c  registry.access.redhat.com/ubi8/pause:latest    15
minutes ago  Up 13 minutes ago  0.0.0.0:8080->80/tcp  e18e307bab06-infra
deeb7e789eaf  quay.io/redhattraining/httpd-parent        /bin/sh -c /usr/s...  13
minutes ago  Up 13 minutes ago  0.0.0.0:8080->80/tcp  webserver
```

① Running Pods

② Running Containers

In order to delete a pod and the containers running within the pod, you must use the **--force** option.

*Listing 46. Source Description*

```
[root@server ~]# podman pod rm rht-demo --force ①
e18e307bab06dd45d3cc5a90472ea075979aacb3adfc2071dfd68dca284b03a3

[root@server ~]# podman pod ps ②
POD ID      NAME      STATUS   CREATED      INFRA ID      # OF CONTAINERS

[root@server ~]# podman ps -a ③
CONTAINER ID  IMAGE      COMMAND      CREATED      STATUS      PORTS      NAMES
```

① Stopping all containers and removing the Pod

② Verifying pod has been removed

③ Verifying there are no running containers and all containers have been removed.

**podman pod rm Errors**

You cannot remove a pod that has running or paused containers.

***Listing 47. podman pod rm Error***

```
[root@server ~]# podman pod rm rht-demo
Error: pod
e18e307bab06dd45d3cc5a90472ea075979aacb3adfc2071dfd68dca284b03a3 has
containers that are not ready to be removed: cannot remove container
3113f296524c48a2eabcd6f114f64c213e8844ecd2eb754557683dc7c205203 as it is
running - running or paused containers cannot be removed without force:
container state improper
```

## 3.2. Creating a multi-container Pod for Wordpress

**Example 12. LAB: Creating a multi-container Pod for Wordpress**

In this exercise, you will be creating a pod to run the Wordpress application. This pod will contain multiple containers, namely Wordpress and MySQL/MariaDB.

1. Change to the **github/OCP\_Demos/Containers/labs/Pods** on Workstation

```
[student@workstation ~]$ cd github/OCP_Demos/Containers/labs/Pods/
```

2. Execute the **Deploy\_Exercise\_Files** Playbook

```
[student@workstation Pods]$ ansible-playbook Deploy_Exercise_Files.yml
```

3. SSH to the **server** as root

```
[student@workstation ~]$ ssh root@server
```

4. Create the pod for running the Wordpress web application

```
[root@server ~]# podman pod create --name wordpress_pod_demo -p 80:80  
5cbe751d69a9fdefb5225180c4897c95f60245eaf049570c2980bfd74e2d
```

5. Verify the pod has been created

*Listing 48. Verification of Pod creation*

```
[root@server ~]# podman pod ps --ctr-names ①  
POD ID      NAME          STATUS   CREATED      INFRA ID      <no value>  
5cbe751d69a9  wordpress_pod_demo  Created  2 minutes ago  893b2d69c740  
5cbe751d69a9-infra
```

① Added the **--ctr-names** to show container names

*Listing 49. Verification using podman ps to get Port mapping*

```
[root@server ~]# podman ps --pod -a  
CONTAINER ID  IMAGE                                     COMMAND  
CREATED      STATUS                                     PORTS  
NAMES        POD ID         PODNAME  
893b2d69c740  registry.access.redhat.com/ubi8/pause:latest  3  
minutes ago  Created          0.0.0.0:80->80/tcp  
5cbe751d69a9-infra  5cbe751d69a9  wordpress_pod_demo
```

## 6. Create the Database container

```
[root@server ~]# podman run -d --name db \
>           -e MYSQL_USER=wordpress \
>           -e MYSQL_PASSWORD=wordpress \
>           -e MYSQL_DATABASE=wordpress \
>           -e MYSQL_ROOT_PASSWORD=somewordpress \
>           --pod wordpress_pod_demo
registry.access.redhat.com/rhscl/mysql-57-rhel7
ec64df9149af45d2ce1f26d03354cd748e28815e25a73430f69180bc8a3246d8
```

*Testing the Database to ensure it is up and ready*

You can check login to MySQL and verify the service allows login and a simple **select** command.



*Listing 50. Testing for Zero Return Code*

```
[root@server ~]# podman exec db bash -c 'mysql -u root -e "SELECT
1" &> /dev/null'; echo $?
0
```

*Copy/Pasteable Command*

*Listing 51. Launching MySQL Container*



```
podman run -d --name db \
-e MYSQL_USER=wordpress \
-e MYSQL_PASSWORD=wordpress \
-e MYSQL_DATABASE=wordpress \
-e MYSQL_ROOT_PASSWORD=somewordpress \
--pod wordpress_pod_demo registry.access.redhat.com/rhscl/mysql-
57-rhel7
```

## 7. Start the Wordpress Container

```
[root@server ~]# podman run -d --name wp \
>           -e WORDPRESS_DB_HOST=127.0.0.1:3306 \
>           -e WORDPRESS_DB_USER=wordpress \
>           -e WORDPRESS_DB_PASSWORD=wordpress \
>           -e WORDPRESS_DB_NAME=wordpress \
>           --pod wordpress_pod_demo
quay.io/redhattraining/wordpress:5.3.0
Trying to pull quay.io/redhattraining/wordpress:5.3.0...
Getting image source signatures
Copying blob 60f22fbcd07a [=====>-----] 57.4MiB / 73.1MiB
Copying blob 60f22fbcd07a [=====>-----] 57.5MiB / 73.1MiB

... OUTPUT OMITTED ...

Writing manifest to image destination
Storing signatures
793b3d5de8108204ed9161fdc65cf81214be99a13676e9f8f251f439852c70f2
```

*Copy/Pasteable Command*

*Listing 52. Launching Wordpress Application*



```
podman run -d --name wp \
-e WORDPRESS_DB_HOST=127.0.0.1:3306 \
-e WORDPRESS_DB_USER=wordpress \
-e WORDPRESS_DB_PASSWORD=wordpress \
-e WORDPRESS_DB_NAME=wordpress \
--pod wordpress_pod_demo quay.io/redhattraining/wordpress:5.3.0
```

## 8. Check Podman Pod and Container Status

```
[root@server ~]# podman pod ps --ctr-names
POD ID      NAME          STATUS   CREATED      INFRA ID    <no
value>
5cbe751d69a9  wordpress_pod_demo  Running  13 minutes ago  893b2d69c740  wp
,5cbe751d69a9-infra ,db ①
```

- ① The **wp** container is running for Wordpress. The **infra** container is running for the Pod. The **db** container is running to provide the database for Wordpress

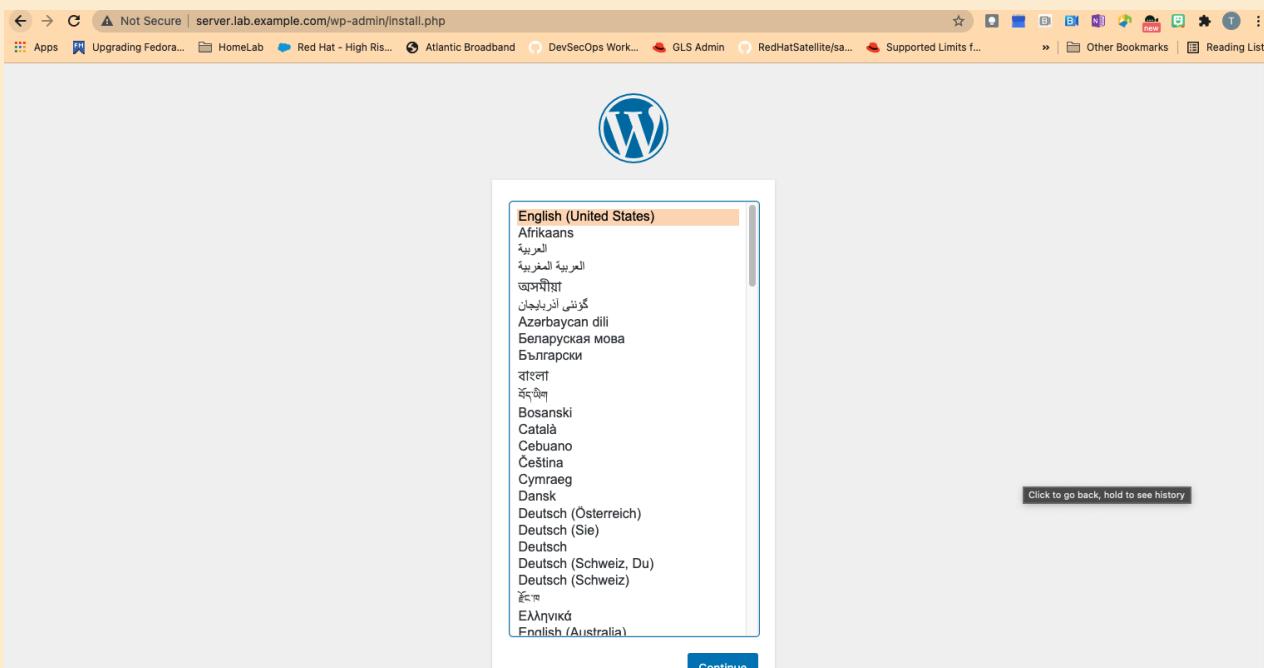
## 9. Verify Website from CLI

```
[root@server ~]# curl -I http://127.0.0.1/wp-admin/install.php
HTTP/1.1 200 OK
Date: Mon, 25 Oct 2021 16:34:17 GMT
Server: Apache/2.4.38 (Debian)
X-Powered-By: PHP/7.3.12
Expires: Wed, 11 Jan 1984 05:00:00 GMT
Cache-Control: no-cache, must-revalidate, max-age=0
Content-Type: text/html; charset=utf-8
```

10. Open Firewall ports on **server.lab.example.com** for port 80/tcp and check from web browser.

*Listing 53. Opening Port*

```
[root@server ~]# firewall-cmd --add-service=http --permanent ; firewall-cmd
--reload
success
success
```



*Figure 48. Wordpress Installation Page*

11. Cleanup Pods and Containers

*Listing 54. Removing Pods and Containers*

```
[root@server ~]# podman pod rm wordpress_pod_demo --force
```

## 3.3. Podman Image and Container Pruning

It is extremely important to cleanup containers, pods, and container images/volumes when the containers are no longer running as resources are prevented from being used even if the container is stopped. More importantly, it is important from a systems management and maintenance standpoint.

### 3.3.1. Using Podman Prune Commands

The **podman <subsystem> prune** command can remove and cleanup Pods, containers, container images, and volumes. This command can be useful as it will only cleanup what isn't being used. For example, if containers are stopped, but will be re-run at a later point, there is no reason to cleanup a volume or the container image because the image is being used, even if it is being used by a stopped container.

#### 3.3.1.1. Cleaning Up Pods

*Listing 55. Looking at Pods*

```
[root@server Buildah]# podman pod ps
POD ID      NAME          STATUS   CREATED      INFRA ID      # OF CONTAINERS
35870e1ae2d0  wordpress_demo_pod  Running  6 minutes ago  bfe7065204b2
```

*Listing 56. Stopping a Pod*

```
[root@server Buildah]# podman pod stop wordpress_demo_pod
35870e1ae2d067e6bc84e4df7ccd2b30f6df6ba4cdb4c098814311ddec1f5b41
```

*Listing 57. Looking at Stopped Pod*

```
[root@server Buildah]# podman ps -ap ①
CONTAINER ID  IMAGE                                     COMMAND
CREATED      STATUS                                     NAMES
ID           PODNAME
bfe7065204b2  registry.access.redhat.com/ubi8/pause:latest    10
minutes ago  Exited (0) 2 minutes ago  0.0.0.0:8880->80/tcp  35870e1ae2d0-infra
35870e1ae2d0  wordpress_demo_pod
21ff11d8a20a  registry.access.redhat.com/rhscl/mysql-57-rhel7  run-mysqld    10
minutes ago  Exited (0) 2 minutes ago  0.0.0.0:8880->80/tcp  db
35870e1ae2d0  wordpress_demo_pod
fb2162542a72  quay.io/redhattraining/wordpress:5.3.0        apache2-foregrou...  10
minutes ago  Exited (0) 2 minutes ago  0.0.0.0:8880->80/tcp  wp
35870e1ae2d0  wordpress_demo_pod
```

① The **-ap** will list all containers and the **pod** to which they belong.

***Listing 58. Pruning Stopped Pods and Containers for Pods***

```
[root@server Buildah]# podman pod prune
WARNING! This will remove all stopped/exited pods..
Are you sure you want to continue? [y/N] y
35870e1ae2d067e6bc84e4df7cccd2b30f6df6ba4cdb4c098814311ddec1f5b41

[root@server Buildah]# podman ps -ap
CONTAINER ID  IMAGE   COMMAND   CREATED   STATUS    PORTS   NAMES   POD ID   PODNAME
```

**3.3.1.2. Cleaning Up Containers*****Listing 59. Running a Sample Container***

```
[root@server Buildah]# podman run --name demo-container-buildah -d -p 8880:80
localhost/demo-container-image
e67dea0b982076219db520581f6e9d1a02e1fe168c2992507f9e0bd973d7d1e
```

***Listing 60. Stopping a Sample Container***

```
[root@server Buildah]# podman stop demo-container-buildah
e67dea0b982076219db520581f6e9d1a02e1fe168c2992507f9e0bd973d7d1ed
```

***Listing 61. Looking for Stopped Containers (podman ps -a)***

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
PORTS	NAMES			
e67dea0b9820	localhost/demo-container-image	/usr/sbin/httpd -...	2 minutes ago	Exited
(0) About a minute ago	0.0.0.0:8880->80/tcp	demo-container-buildah		①

① Image attached to a running or **stopped** container

***Cleanup of Containers***

Cleanup of exited/stopped containers is also very important as these containers are reserving system resources, storage space, namespaces, networking, etc. The **podman container prune** command can be used to cleanup and remove all stopped containers.

***Listing 62. Container Cleanup***

```
[root@server Buildah]# podman container prune
WARNING! This will remove all non running containers.
Are you sure you want to continue? [y/N] y
e67dea0b982076219db520581f6e9d1a02e1fe168c2992507f9e0bd973d7d1ed
```

### 3.3.1.3. Cleaning Up Images

#### Image Management

Image management is extremely important when using Podman as there are a limited number of resources on the system - namely storage space. Having a good strategy for container cleanup is needed in order to ensure a healthy system. The **podman image prune** command can be used to cleanup and remove all orphaned images not being used by containers. The **podman rmi** command is used to delete actual tagged images from the local image registry.

#### *Listing 63. Cleanup of Images*

```
[root@server Buildah]# podman images | grep demo ①
localhost/demo-container-image          latest      d21ee320b60a 2 hours ago
553 MB

[root@server Buildah]# podman images prune ②
REPOSITORY TAG      IMAGE ID      CREATED      SIZE

[root@server Buildah]# podman images | grep demo ③
localhost/demo-container-image          latest      d21ee320b60a 2 hours ago
553 MB
```

#### Image Cleanup

It will always be necessary to cleanup images after a pod or a container cleanup because there could be **orphaned** and **dangling** images left after all stopped pods and containers have been pruned. This happens if there are images on the system that are unused and have no tags assigned to them.

#### *Listing 64. Image Listing*



```
[root@server Buildah]# podman images
REPOSITORY                                     TAG      IMAGE ID
CREATED      SIZE
localhost/demo-container-image                latest      d21ee320b60a
3 hours ago    553 MB
```

#### *Listing 65. Image Pruning*

```
[root@server Buildah]# podman image prune
WARNING! This will remove all dangling images.
Are you sure you want to continue? [y/N] y
```



### *Image Cleanup*

Actual container image cleanup is performed with the **podman rmi \*** command which will delete all images that aren't attached to containers. It won't remove any images that are attached to currently running or stopped containers unless the **--force** option is used. However, when using **--force** this will also stop and remove all containers prior to removing the container images.

### Example 13. LAB: Managing Containers and Images

Playbooks have been setup to create and launch pods as well as various scripts to perform the launching.

1. Change to the **github/OCP\_Demos/Containers/labs/Pods** on Workstation

```
[student@workstation ~]$ cd github/OCP_Demos/Containers/labs/Pods/
```

2. Install Podman Containers Collection

```
[student@workstation Pods]$ ansible-galaxy collection install -r requirements.yml  
-p collections  
Process install dependency map  
Starting collection install process  
Installing 'containers.podman:1.8.1' to  
'/home/student/github1/OCP_Demos/Containers/labs/Pods/collections/ansible_collections/containers/podman'
```

3. Create the Registry Credentials File

```
[student@workstation Pods]$ cp vars/registry_login_demo.yml  
vars/registry_login.yml  
[student@workstation Pods]$ vim vars/registry_login.yml
```

4. Launch a Container

```
[student@workstation Pods]$ ansible-playbook Deploy_Container_Demo.yml  
  
PLAY [Deploy HTTPD Server Demo]  
*****  
  
... OUTPUT OMITTED ...
```

5. Identify Containers on Workstation

```
[student@workstation Pods]$ sudo podman ps  
CONTAINER ID  IMAGE                                COMMAND  
CREATED      STATUS      PORTS  
51ed5c1fc601  quay.io/redhattraining/httpd-parent:2.4  /bin/sh -c /usr/s...  48  
seconds ago  Up 47 seconds ago  0.0.0.0:7080->80/tcp  Website_Demo
```

## 6. Stop Container on Workstation

```
[student@workstation Pods]$ sudo podman stop Website_Demo  
51ed5c1fc601e3722b9a6288e7ec879c75e0b4eefb03707ffd8f83056d9654c5
```

## 7. Prune Containers on Workstation

```
[student@workstation Pods]$ sudo podman container prune  
51ed5c1fc601e3722b9a6288e7ec879c75e0b4eefb03707ffd8f83056d9654c5
```

## 8. Prune Images on Workstation

```
[student@workstation Pods]$ sudo podman image prune
```

## 9. List and Delete Images on Workstation

```
[student@workstation Pods]$ sudo podman images ①  
REPOSITORY TAG IMAGE ID CREATED SIZE  
quay.io/redhattraining/httpd-parent 2.4 3639ce1374d3 2 years ago 236 MB
```

```
[student@workstation Pods]$ sudo podman rmi 3639ce1374d3 ②  
Untagged: quay.io/redhattraining/httpd-parent:2.4  
Deleted: 3639ce1374d3611e80ed66decd7d5467b72d010c21e19e4f193cd8b944e8c9f5
```

```
[student@workstation Pods]$ sudo podman images ③  
REPOSITORY TAG IMAGE ID CREATED SIZE
```

① Listing Images

② Removing desired image

③ Ensuring image was removed

## 3.4. Using Podman to Create YAML Resource Files

Kubernetes and container orchestration tools typically require custom resource definition file (CRDs) that will fully describe the deployment and all settings. These YAML files are a description of what takes place in a declarative language, much like Ansible.

Just as Kubernetes and Openshift manages containers by creating and managing Pods, so too does Podman. Newer upgrades and features to Podman allow the generation of these YAML files to make transitioning to/from Kubernetes much easier. Additionally, Podman can use these local pod/container

definition files to launch and run services locally using Podman.

### 3.4.1. Creating a Deployment File from a Container

It is possible to create a deployment file for Podman from a running container. These files can simplify things when launching a job multiple times. It isn't necessary to author a bash script to leverage Podman to start and run the jobs. The resulting deployment file will create a pod with the needed container when running the **podman play kube** command as a **pod** is the smallest unit that can be managed and Kubernetes/OpenShift doesn't manage anything smaller than the pod-level.

**Example 14. LAB: Creating a Deployment File from a Container****1. SSH to `root@server`**

```
[student@workstation ~]$ ssh root@server
```

**2. Create a running container from `HTTPD-Parent`**

```
[root@server ~]# podman run -d --name yaml-web-demo -p 8888:80  
quay.io/redhattraining/httpd-parent:2.4  
fc626111af2376dc25fb3b397fce2c6536b5a3aa45db577d28d0ee41f7cdc791
```

**3. Test that the website is running in the container**

```
[root@server ~]# curl localhost:8888  
Hello from the httpd-parent container!
```

**4. Generate the YAML file**

```
[root@server ~]# podman generate kube yaml-web-demo > yaml-web-demo.yml
```

**5. Inspect the YAML file**

```
[root@server ~]# cat yaml-web-demo.yml  
# Generation of Kubernetes YAML is still under development!  
#  
# Save the output of this file and use kubectl create -f to import  
# it into Kubernetes.  
#  
# Created with podman-3.0.2-dev  
apiVersion: v1  
kind: Pod  
metadata:  
  creationTimestamp: "2021-10-27T15:41:06Z"  
  labels:  
    app: yaml-web-demo  
    name: yaml-web-demo ①  
spec:  
  containers:  
    - command:  
      - /bin/sh  
      - -c
```

```
- /usr/sbin/httpd -DFOREGROUND
env:
- name: PATH
  value: /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
- name: TERM
  value: xterm
- name: container
  value: oci
- name: DOCROOT
  value: /var/www/html
image: quay.io/redhattraining/httpd-parent:2.4 ②
name: yaml-web-demo ③
ports:
- containerPort: 80
  hostPort: 8888 ④
  protocol: TCP
resources: {}
securityContext:
  allowPrivilegeEscalation: true
  capabilities:
    drop:
    - CAP_MKNOD
    - CAP_AUDIT_WRITE
  privileged: false
  readOnlyRootFilesystem: false
  runAsGroup: 0
  runAsUser: 0
  seLinuxOptions: {}
  workingDir: /
dnsConfig: {}
status: {}
```

① Pod name is defined

② Container image is defined

③ Container name is defined

④ Container port mapping is defined



Since we didn't create a pod with a separate name, when the command was run, it created a **pod name** which was/is the same as the **container name**. This can cause issues when deploying from the file by creating a warning message. The **podman play kube** will generate a new name for the items so there are no duplicate name conflicts.

## 6. Stop and Remove Container

```
[root@server ~]# podman rm yaml-web-demo -f  
fc626111af2376dc25fb3b397fce2c6536b5a3aa45db577d28d0ee41f7cdc791
```

## 7. Start a new container with the **podman play kube** command

```
[root@server ~]# podman play kube ./yaml-web-demo.yml ①  
a container exists with the same name ("yaml-web-demo") as the pod in your YAML  
file; changing pod name to yaml-web-demo_pod  
Pod: ②  
72f468000f4899e89d0def7add5859c0c2cc75c9f0be49c85f4a7db401feed07  
Container: ③  
72d22f1867e9b8f56921a5f6bba460e2c9fc123db0ed7a119974cf46ce3305f
```

① **podman play kube** will attempt to create and start both a pod and a container because Kubernetes/OpenShift is meant to manage pods

② A Pod has been created for the resource

③ The **yaml-web-demo** container is created

## 8. Test and verify website

```
[root@server ~]# curl localhost:8888  
Hello from the httpd-parent container!
```

## 9. Show running containers and Pods

```
[root@server ~]# podman pod ps ①  
POD ID      NAME          STATUS   CREATED      INFRA ID      # OF  
CONTAINERS  
72f468000f48  yaml-web-demo_pod  Running  2 minutes ago  c6b8cb2eda97  2  
  
[root@server ~]# podman ps | grep yaml ②  
72d22f1867e9  quay.io/redhattraining/httpd-parent:2.4  
minutes ago    Up 3 minutes ago    0.0.0.0:8888->80/tcp  
yaml-web-demo_pod-yaml-web-demo
```

① Showing the **yaml-web-demo\_pod** pod

② Showing the **yaml-web-demo** in the **yaml-web-demo\_pod**

## 10. Stop and remove the **yaml-web-demo\_pod** and all containers

```
[root@server ~]# podman pod rm yaml-web-demo_pod -f  
72f468000f4899e89d0def7add5859c0c2cc75c9f0be49c85f4a7db401feed07
```

11. Verify that the pod and container has been removed

```
[root@server ~]# podman ps -a | grep yaml  
  
[root@server ~]# podman pod ps  
POD ID  NAME      STATUS  CREATED  INFRA ID  # OF CONTAINERS
```

### 3.4.2. Creating a Deployment file from a Pod

It is possible to create a deployment file for Podman from a running pod. This deployment file will contain information about all containers running in the pod and can be used for later deployments.

### Example 15. LAB: Deployment YAML File for a Pod with Multiple Containers

For this lab, we will be creating a **Wordpress Demo Pod** like we did above but exposing port 9999. Instead of using a script, we will have a YAML file which can deploy the application.

#### 1. SSH to **root@server**

```
[student@workstation ~]$ ssh root@server
```

#### 2. Create a running Pod

```
[root@server ~]# podman pod create --name wordpress_pod_demo -p 9999:80  
6690272e241f1155dde5ddc276e8fc5587c6258cf83b6701038713b95ae24971
```

#### 3. Create a Database Container in the Running Pod

```
[root@server ~]# podman run -d --name db \  
> -e MYSQL_USER=wordpress \  
> -e MYSQL_PASSWORD=wordpress \  
> -e MYSQL_DATABASE=wordpress \  
> -e MYSQL_ROOT_PASSWORD=somewordpress \  
> --pod wordpress_pod_demo registry.access.redhat.com/rhscl/mysql-57-rhel7
```

*Copy/Pasteable Command*

*Listing 66. Deploying a Database Container*



```
podman run -d --name db \  
-e MYSQL_USER=wordpress \  
-e MYSQL_PASSWORD=wordpress \  
-e MYSQL_DATABASE=wordpress \  
-e MYSQL_ROOT_PASSWORD=somewordpress \  
--pod wordpress_pod_demo registry.access.redhat.com/rhscl/mysql-  
57-rhel7
```

#### 4. Create the Wordpress Container in the Running Pod

```
[root@server ~]# podman run -d --name wp \
> -e WORDPRESS_DB_HOST=127.0.0.1:3306 \
> -e WORDPRESS_DB_USER=wordpress \
> -e WORDPRESS_DB_PASSWORD=wordpress \
> -e WORDPRESS_DB_NAME=wordpress \
> --pod wordpress_pod_demo quay.io/redhattraining/wordpress:5.3.0
```

*Copy/Pasteable Command*

*Listing 67. Deploying Wordpreess Container*



```
podman run -d --name wp \
-e WORDPRESS_DB_HOST=127.0.0.1:3306 \
-e WORDPRESS_DB_USER=wordpress \
-e WORDPRESS_DB_PASSWORD=wordpress \
-e WORDPRESS_DB_NAME=wordpress \
--pod wordpress_pod_demo quay.io/redhattraining/wordpress:5.3.0
```

## 5. Verify Podman and Container Status

```
[root@server ~]# podman pod ps --ctr-names
POD ID      NAME          STATUS   CREATED      INFRA ID      <no value>
6690272e241f  wordpress_pod_demo  Running  3 minutes ago  43def184da95
6690272e241f-infra,wp,db
```

## 6. Verify Website Status

```
[root@server ~]# curl -I http://127.0.0.1:9999/wp-admin/install.php
HTTP/1.1 200 OK
Date: Wed, 27 Oct 2021 16:06:26 GMT
Server: Apache/2.4.38 (Debian)
X-Powered-By: PHP/7.3.12
Expires: Wed, 11 Jan 1984 05:00:00 GMT
Cache-Control: no-cache, must-revalidate, max-age=0
Content-Type: text/html; charset=utf-8
```

## 7. Create YAML File for the Running Pod

```
[root@server ~]# podman generate kube wordpress_pod_demo > wordpress_pod_demo.yml
```

## 8. Examine the **wordpress\_pod\_demo.yml** file

```
[root@server ~]# cat wordpress_pod_demo.yml
# Generation of Kubernetes YAML is still under development!
#
# Save the output of this file and use kubectl create -f to import
# it into Kubernetes.
#
# Created with podman-3.0.2-dev
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: "2021-10-27T16:08:16Z"
  labels:
    app: wordpresspoddemo
    name: wordpress_pod_demo ①
spec:
  containers:
  - args:
    - apache2-foreground
    command:
    - docker-entrypoint.sh
    env:
      - name: PATH
        value: /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
      - name: TERM
        value: xterm
      - name: container
        value: podman
      - name: PHP_LDFLAGS
        value: -Wl,-O1 -Wl,--hash-style=both -pie
      - name: APACHE_ENVVARS
        value: /etc/apache2/envvars
      - name: WORDPRESS_SHA1
        value: e3edcb1131e539c2b2e10fed37f8b6683c824a98
      - name: PHP_URL
        value: https://www.php.net/get/php-7.3.12.tar.xz/from/this/mirror
      - name: WORDPRESS_DB_HOST
        value: 127.0.0.1:3306
      - name: PHP_EXTRA_CONFIGURE_ARGS
        value: --with-apxs2 --disable-cgi
      - name: WORDPRESS_VERSION
        value: "5.3"
      - name: WORDPRESS_DB_USER
        value: wordpress
      - name: WORDPRESS_DB_PASSWORD
        value: wordpress
      - name: PHP_CPPFLAGS
        value: -fstack-protector-strong -fpic -fpie -O2 -D_LARGEFILE_SOURCE
-D_FILE_OFFSET_BITS=64
```

```
- name: PHP_ASC_URL
  value: https://www.php.net/get/php-7.3.12.tar.xz.asc/from>this/mirror
- name: PHPIZE_DEPS
  value: "autoconf \t\tdpkg-dev \t\tfile \t\tg++ \t\tgcc \t\tlibc-dev \t\tmake
\t\tpkg-config \t\tre2c"
- name: PHP_SHA256
  value: aafe5e9861ad828860c6af8c88cdc1488314785962328eb1783607c1fdd855df
- name: PHP_EXTRA_BUILD_DEPS
  value: apache2-dev
- name: PHP_CFLAGS
  value: -fstack-protector-strong -fpic -fpie -O2 -D_LARGEFILE_SOURCE
-D_FILE_OFFSET_BITS=64
- name: PHP_VERSION
  value: 7.3.12
- name: GPG_KEYS
  value: CBAF69F173A0FEA4B537F470D66C9593118BCCB6
F38252826ACD957EF380D39F2F7956BC5DA04B5D
- name: PHP_MD5
- name: PHP_INI_DIR
  value: /usr/local/etc/php
- name: APACHE_CONFDIR
  value: /etc/apache2
- name: WORDPRESS_DB_NAME
  value: wordpress
image: quay.io/redhattraining/wordpress:5.3.0
name: wp ②
ports:
- containerPort: 80
  hostPort: 9999 ③
  protocol: TCP
resources: {}
securityContext:
  allowPrivilegeEscalation: true
  capabilities:
    drop:
      - CAP_MKNOD
      - CAP_AUDIT_WRITE
  privileged: false
  readOnlyRootFilesystem: false
  selinuxOptions: {}
workingDir: /var/www/html
- args:
  - run-mysqld
command:
- container-entrypoint
env:
- name: PATH
  value: /opt/app-root/src/bin:/opt/app-
```

```
root/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
  - name: TERM
    value: xterm
  - name: container
    value: oci
  - name: BASH_ENV
    value: /usr/share/container-scripts/mysql/scl_enable
  - name: PLATFORM
    value: el7
  - name: PROMPT_COMMAND
    value: . /usr/share/container-scripts/mysql/scl_enable
  - name: APP_ROOT
    value: /opt/app-root
  - name: MYSQL_ROOT_PASSWORD
    value: somewordpress
  - name: MYSQL_USER
    value: wordpress
  - name: MYSQL_PASSWORD
    value: wordpress
  - name: HOME
    value: /var/lib/mysql
  - name: STI_SCRIPTS_PATH
    value: /usr/libexec/s2i
  - name: STI_SCRIPTS_URL
    value: image:///usr/libexec/s2i
  - name: MYSQL_DATABASE
    value: wordpress
  - name: ENV
    value: /usr/share/container-scripts/mysql/scl_enable
  - name: SUMMARY
    value: MySQL 5.7 SQL database server
  - name: ENABLED_COLLECTIONS
    value: rh-mysql57
  - name: DESCRIPTION
    value: MySQL is a multi-user, multi-threaded SQL database server. The container image provides a containerized packaging of the MySQL mysqld daemon and client application. The mysqld server daemon accepts connections from clients and provides access to content from MySQL databases on behalf of the clients.
  - name: MYSQL_PREFIX
    value: /opt/rh/rh-mysql57/root/usr
  - name: CONTAINER_SCRIPTS_PATH
    value: /usr/share/container-scripts/mysql
  - name: APP_DATA
    value: /opt/app-root/src
  - name: MYSQL_VERSION
    value: "5.7"
image: registry.access.redhat.com/rhscl/mysql-57-rhel7:latest
name: db ④
```

```
resources: {}
securityContext:
  allowPrivilegeEscalation: true
  capabilities:
    drop:
      - CAP_MKNOD
      - CAP_AUDIT_WRITE
  privileged: false
  readOnlyRootFilesystem: false
  runAsGroup: 27
  runAsUser: 27
  selinuxOptions: {}
  workingDir: /opt/app-root/src
dnsConfig: {}
restartPolicy: Never
status: {}
```

- ① Pod Name for the **Wordpress\_Pod\_Demo** Pod
- ② Container name for Wordpress Container **wp**
- ③ Container port for the Wordpress Container **9999**
- ④ Container name for the Database Container **db**

## 9. Remove Running Pod and all Containers

```
[root@server ~]# podman pod rm wordpress_pod_demo -f
6690272e241f1155dde5ddc276e8fc5587c6258cf83b6701038713b95ae24971
```

## 10. Verify Pod and Containers Removed

```
[root@server ~]# podman pod ps
POD ID  NAME      STATUS  CREATED  INFRA ID  # OF CONTAINERS

[root@server ~]# podman ps -a | grep wp

[root@server ~]# podman ps -a | grep db
```

## 11. Deploy Wordpress Pod from YAML File with **podman play kube** command

```
[root@server ~]# podman play kube ./wordpress_pod_demo.yml
```

```
... OUTPUT OMITTED ...
```

Pod:

```
8fbfbdaaf4c17a60180fa918b8fe426f568e697e72f68c22da7a3b70b592756eb
```

Containers:

```
b044e3cbc3cbdfaae6a74c3646a43fd971ec044fe1b566f97802688307aae7a  
8a4e1ce9e3375d945ad82a503490472b9f955350098afed7716d3a1a13e76c53
```

## 12. Verify Pod Status

```
[root@server ~]# podman pod ps --ctr-names  
POD ID      NAME          STATUS    CREATED           INFRA ID      <no  
value>  
8fbfbdaaf4c17  wordpress_pod_demo  Running  About a minute ago  0a19088da270  
8fbfbdaaf4c17-infra,wordpress_pod_demo-db,wordpress_pod_demo-wp
```

## 13. Verify Website

```
[root@server ~]# curl -I http://127.0.0.1:9999/wp-admin/install.php  
HTTP/1.1 200 OK  
Date: Wed, 27 Oct 2021 16:16:37 GMT  
Server: Apache/2.4.38 (Debian)  
X-Powered-By: PHP/7.3.12  
Expires: Wed, 11 Jan 1984 05:00:00 GMT  
Cache-Control: no-cache, must-revalidate, max-age=0  
Content-Type: text/html; charset=utf-8
```

## 14. Cleanup pod and containers

```
[root@server ~]# podman pod rm wordpress_pod_demo -f
```

## 15. Verify Cleanup

```
[root@server ~]# podman pod ps  
POD ID  NAME      STATUS    CREATED   INFRA ID  # OF CONTAINERS
```

```
[root@server ~]# podman ps -a | grep wp
```

```
[root@server ~]# podman ps -a | grep db
```

### References

**Managing Containers and Pods:** [https://developers.redhat.com/blog/2019/01/15/podman-managing-containers-pods?ts=1634314817672#podman\\_pods\\_what\\_you\\_need\\_to\\_know](https://developers.redhat.com/blog/2019/01/15/podman-managing-containers-pods?ts=1634314817672#podman_pods_what_you_need_to_know)

**Managing Containers using Podman and Skopeo:** <https://www.tecmint.com/manage-containers-using-podman-in-rhel/>

**Podman:** <https://docs.podman.io/en/latest/>

**Moving from docker-compose to Podman pods:** <https://www.redhat.com/sysadmin/compose-podman-pods>

**Podman: Managing pods and containers in a local container runtime:** <https://developers.redhat.com/blog/2019/01/15/podman-managing-containers-pods#>

**Stefano's Project on Github:** <https://github.com/sstagnaro/rht-contrib>

**Spinning up and Managing Pods with multiple containers with Podman:** <https://mohitgoyal.co/2021/04/23/spinning-up-and-managing-pods-with-multiple-containers-with-podman/>

**What is a dangling image and what is an unused image?:** <https://newbedev.com/what-is-a-dangling-image-and-what-is-an-unused-image>

**Podman can now ease the transition to Kubernetes and CRI-O:** <https://developers.redhat.com/blog/2019/01/29/podman-kubernetes-yaml#>

**Moving from docker-compose to Podman pods:** <https://www.redhat.com/sysadmin/compose-podman-pods>

**Working with pods with podman generate and podman play:** <https://mohitgoyal.co/2021/05/10/working-with-pods-with-podman-generate-and-podman-play/>

**Containers vs. Pods - Taking a Deeper Look:** <https://iximiuz.com/en/posts/containers-vs-pods/>

**man pages:** *podman-system-prune*, *podman-container-cleanup*, *podman-pod-prune*, *podman-container-prune*, *podman-image-prune*, and *podman-volume-prune*.

# 4. Container Images

Container images are required and the basis of all running containers. Container images can be built from scratch using various methods, build by extending existing containers (either manually or through Dockerfiles/Containerfiles), or they can be build as part of a CI/CD pipeline with OpenShift or other Source-to-Image (s2i) tools. With the introduction of Podman into the RHEL8 **container-tools** packages also came **buildah**. **Buildah** is capable of creating container images from scratch as well as building and creating images from Dockerfiles/Container files.

## 4.1. Introduction to Buildah

While Podman allows building of containers with Dockerfiles/Container files or committing a running container into a new custom container image, **Buildah** provides a much cleaner approach. Using **Buildah** to construct and publish a container interactively allows much more control over the process and can result in a clean image being committed. Buildah also includes the **buildah config** which is able to commit and add various parameters to the container just like you would in the Dockerfile/Containerfile.

**Buildah**: A low-level tool for building and manipulating container images. Provides additional tools for building container images and **buildah** libraries are used by **podman** allowing it to have **podman build**. On the back-end, **podman** is using **Buildah** to build the container images.

OpenShift Source-to-Image process also leverages **Buildah** to construct and build the container images. There are also other projects and applications that rely on **Buildah** to construct and build container images such as **ansible-blender**. In Chapter 6, we will be using a simple playbook leveraging Ansible Collections to build and publish an image to a container registry from a **Containerfile**.

Buildah is capable of producing very small OCI container images as it doesn't include any of the build tools within the image itself. Additionally, the resulting container image also doesn't include any compilers or other tools thereby making the image smaller and more secure. Since Buildah images are OCI images, they are portable and can run on a variety of different container images.

### 4.1.1. Buildah Commands and Overview

Just like Podman, **buildah** commands have very similar options.

#### *Buildah Commands*

- **buildah inspect**: Inspects an image
- **buildah from scratch**: Builds a new container image from scratch
- **buildah containers**: View containers
- **buildah images**: View images in local registry
- **buildah rm**: Removes container
- **buildah rmi**: Removes container image

- **buildah pull:** Pulls image from remote registry
- **buildah push:** Pushes image to remote registry
- **buildah commit:** Saves and tags the image to the local image registry
- **buildah bud or buildah build-using-dockerfile:** Builds container image from Dockerfile/Containerfile

Almost all **Buildah** commands have the equivalent syntax and options to Podman. There are some slight exceptions and any help with **Buildah** commands is available in the man pages or using **buildah <command> --help**

*Listing 68. The buildah images Command*

```
[root@server ~]# buildah images
REPOSITORY                                TAG      IMAGE ID      CREATED
SIZE
quay.local:80/redhattraining/hello-world-nginx    latest    8d990e08937e  2 years ago
269 MB
quay.io/redhattraining/httpd-parent           2.4      3639ce1374d3  2 years ago
236 MB
quay.local/travis/httpd-parent                2.4      3639ce1374d3  2 years ago
236 MB
```

*Listing 69. The buildah inspect Command*

```
[root@server ~]# buildah inspect quay.local:80/redhattraining/hello-world-nginx
{
  "Type": "buildah 0.0.1",
  "FromImage": "quay.local:80/redhattraining/hello-world-nginx:latest",

  ... OUTPUT OMITTED ...

  "OCId": {
    "created": "2019-06-26T22:59:13.751737399Z",
    "architecture": "amd64",
    "os": "linux",
    "config": {
      "User": "1001",
      "ExposedPorts": {
        "8080/tcp": {}
      },
      "Env": [
        "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
        "container=oci"
      ],
      "Cmd": [
        "/bin/sh",
        "-c"
      ]
    }
  }
}
```

```
"-c",
"nginx -g \"daemon off;\""

... OUTPUT OMITTED ...

},
"history": [
{
    "created": "2019-04-25T16:26:20.260497942Z",
    "comment": "Imported from -"
},
{
    "created": "2019-04-25T16:26:28.425556Z"
},
{
    "created": "2019-06-26T22:33:55.170741575Z",
    "created_by": "/bin/sh -c yum install -y --disableplugin=subscription
manager --nodoscs nginx && yum clean all"
},
{
    "created": "2019-06-26T22:34:02.362527585Z",
    "created_by": "/bin/sh -c #(nop) ADD index.html /usr/share/nginx/html"
},
{
    "created": "2019-06-26T22:59:11.622403847Z",
    "created_by": "/bin/sh -c #(nop) ADD nginxconf.sed /tmp/"
},
{
    "created": "2019-06-26T22:59:12.412697298Z",
    "created_by": "/bin/sh -c sed -i -f /tmp/nginxconf.sed
/etc/nginx/nginx.conf"
},
{
    "created": "2019-06-26T22:59:13.277995379Z",
    "created_by": "/bin/sh -c touch /run/nginx.pid && chgrp -R 0
/var/log/nginx /run/nginx.pid && chmod -R g+rwx /var/log/nginx /run/nginx.pid"
},
{
    "created": "2019-06-26T22:59:13.414802016Z",
    "created_by": "/bin/sh -c #(nop) EXPOSE 8080"
},
{
    "created": "2019-06-26T22:59:13.593338162Z",
    "created_by": "/bin/sh -c #(nop) USER 1001"
},
{
    "created": "2019-06-26T22:59:13.751737399Z"
}
```

```
        ],
},
"Docker": {
    "created": "2019-06-26T22:59:13.751737399Z",
    "container_config": {
        "Hostname": "",
        "Domainname": "",
        "User": "1001",
        "AttachStdin": false,
        "AttachStdout": false,
        "AttachStderr": false,
        "ExposedPorts": {
            "8080/tcp": {}
        },
...
    ... OUTPUT OMITTED ...
    "config": {
        "Hostname": "",
        "Domainname": "",
        "User": "1001",
        "AttachStdin": false,
        "AttachStdout": false,
        "AttachStderr": false,
        "ExposedPorts": {
            "8080/tcp": {}
        },
        "Tty": false,
        "OpenStdin": false,
        "StdinOnce": false,
        "Env": [
            "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
            "container=oci"
        ],
        "Cmd": [
            "/bin/sh",
            "-c",
            "nginx -g \"daemon off;\""
        ],
...
    ... OUTPUT OMITTED ...
}
}
```

## 4.2. Building Containers with Buildah

The DO180 course has taught how to build containers with an existing Dockerfile/Containerfile using Podman as well as taking a running container and using **podman commit** to create the new container image. Since **podman build** essentially uses **Buildah** on the back-end, we will only be exploring using Buildah to build containers from **scratch** interactively on the system and then publishing those containers.

### 4.2.1. Building images as the root user

**Example 16. LAB: Creating a Custom Container Image Using Buildah**

1. Create a container with Buildah

*Listing 70. Creating a Custom Container*

```
[root@server ~]# buildah from scratch  
working-container
```

2. Inspect the Container

*Listing 71. Naming and Inspecting a Custom Container*

```
[root@server ~]# buildah config --label name=Demo-Container working-container  
[root@server ~]# buildah inspect working-container  
{  
    "Type": "buildah 0.0.1",  
    "FromImage": "",  
    "FromImageID": "",  
  
    ... OUTPUT OMITTED ...  
  
    "History": null,  
    "Devices": []  
}
```

3. Mount container disk image and prepare for installation and customization

*Listing 72. Installing Packages on Working Container*

```
[root@server ~]# buildah mount working-container  
/var/lib/containers/storage/overlay/47059b1cb7f2c518e7f98d905b7ae1bcad78e896257536  
1bdf7dda36701d38a9/merged ①  
  
[root@server ~]# export  
Container_Disk_Image=/var/lib/containers/storage/overlay/47059b1cb7f2c518e7f98d905  
b7ae1bcad78e8962575361bdf7dda36701d38a9/merged ②  
  
[root@server ~]# echo $Container_Disk_Image  
/var/lib/containers/storage/overlay/47059b1cb7f2c518e7f98d905b7ae1bcad78e896257536  
1bdf7dda36701d38a9/merged ③
```

① Mount container image filesystem for modification

② Export Container Disk Image Mountpoint for easy reference

③ Verify **Container\_Disk\_Image** Variable

#### 4. Prepare container image for installation of RPM Packages

*Listing 73. Download Red Hat Release RPM for installation*

```
[root@server ~]# yumdownloader --destdir=/tmp redhat-release-server  
Last metadata expiration check: 23:16:26 ago on Sun 24 Oct 2021 01:57:13 PM EDT.
```

#### 5. Install the Red Hat Release RPM to setup GPG keys for repository

*Listing 74. Install Red Hat Release RPM*

```
[root@server ~]# rpm -ivh --root $Container_Disk_Image /tmp/redhat-release-8.4-  
0.6.el8.x86_64.rpm  
warning: /tmp/redhat-release-8.4-0.6.el8.x86_64.rpm: Header V3 RSA/SHA256  
Signature, key ID fd431d51: NOKEY  
Verifying... ##### [100%]  
Preparing... ##### [100%]  
Updating / installing... ##### [100%]  
1:redhat-release-8.4-0.6.el8 ##### [100%]
```

#### 6. Setup the Repository on Disk Image

*Listing 75. Create repository for container image so files can be installed*

```
[root@server ~]# cp /etc/yum.repos.d/rhel_dvd.repo  
$Container_Disk_Image/etc/yum.repos.d/
```

#### 7. Install the **httpd** Apache Pacakge to the Container

*Listing 76. Install the HTTP package for a webserver*

```
[root@server ~]# yum install --installroot $Container_Disk_Image httpd
```

#### 8. Create a custom **index.html** File

*Listing 77. Create an **index.html** file for the webserver*

```
[root@server ~]# echo "This is a custom webserver container for me" >>  
$Container_Disk_Image/var/www/html/index.html
```

#### 9. Install the Apache Manual for Webserver Documentation

*Listing 78. Install the Apache manual for reference documentation*

```
[root@server ~]# yum install --installroot $Container_Disk_Image httpd-manual
```

*Poor Practice*

It is a poor practice to have unneeded items, packages, and documentation in containers. This step normally wouldn't be performed.

**10. Set the Container's Runtime Command***Listing 79. Configure webserver to run*

```
[root@server ~]# buildah config --cmd "/usr/sbin/httpd -DFOREGROUND" working-container
```

**11. Configure and EXPORT the webserver default port***Listing 80. Configure and open port 80 for the TCP protocol for the container*

```
[root@server ~]# buildah config --port 80/tcp working-container
```

**12. Set and Configure the Container Image Author**

```
[root@server ~]# buildah config --author "Travis Michette <tmichett@redhat.com>" working-container
```

**13. Cleanup unneeded files to conserve space on container image.***Listing 81. Clean up yum data to minimize required disk space*

```
[root@server ~]# yum clean all --installroot $Container_Disk_Image  
13 files removed
```

**14. Unmount Container Filesystem image.***Listing 82. Unmount the container image filesystem*

```
[root@server ~]# buildah unmount working-container  
fb5387521fd7b0c9f6dd567ca0f76222178f0ffbcf38fc990df9d49a60079e1
```

**15. Commit container image to local container registry**

*Listing 83. Commit the container image*

```
[root@server ~]# buildah commit working-container demo-container-image ①
Getting image source signatures
Copying blob 1a73c59066af done

... OUTPUT OMITTED ...

Storing signatures
d21ee320b60ad274b010328df6f10f6dfed578623f00855748d8aaa3c29b5e94
```

① This will create a container image tagged **demo-container-image** with the **latest** tag.

## 16. View and List Container Images

*Listing 84. List container images*

```
[root@server ~]# buildah images | grep demo
localhost/demo-container-image          latest      d21ee320b60a  About
a minute ago   553 MB
```

## 17. Test the Container image by launching a container

*Listing 85. Testing the Container Image*

```
[root@server ~]# podman run --name demo-container-buildah -d -p 8880:80
localhost/demo-container-image
be3bbc8898fc6cab1cef5d78721f46b985e957d30984d591271fffc5b906994

[root@server ~]# curl localhost:8880
This is a custom webserver container for me

[root@server ~]# curl http://localhost:8880/manual/
```

## 18. Test image in Web Browser

*Listing 86. Open Firewall Ports*

```
[root@server ~]# firewall-cmd --add-port=8880/tcp --permanent ; firewall-cmd
--reload
success
success
```

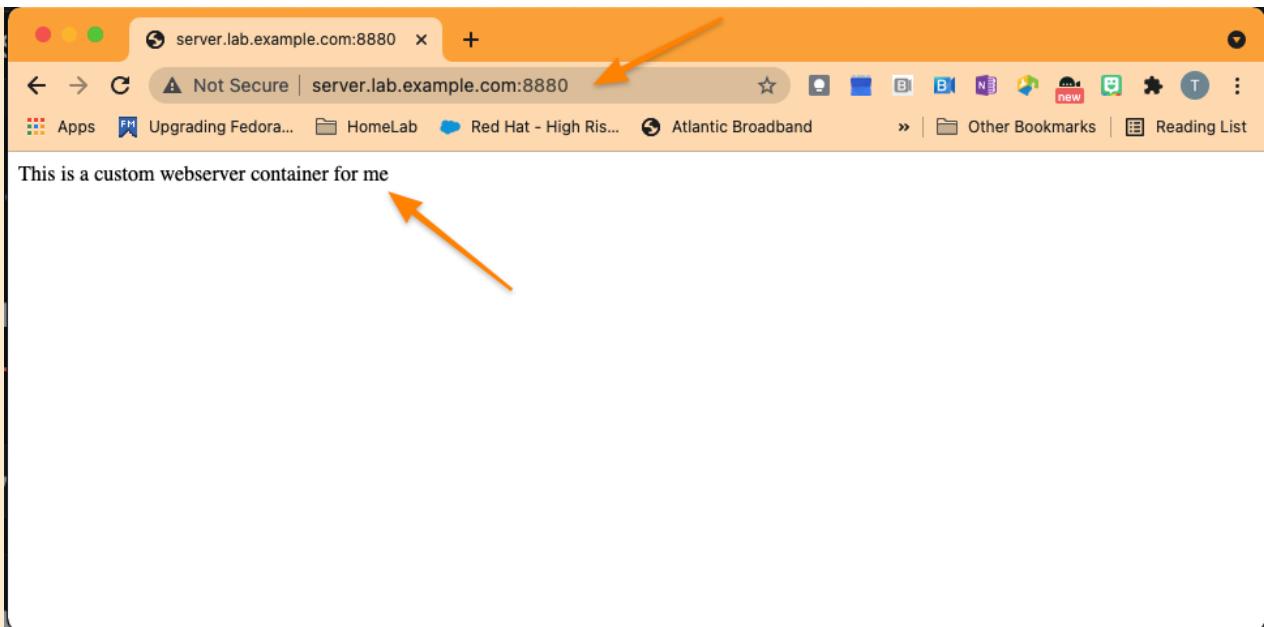


Figure 49. Custom \*index.html

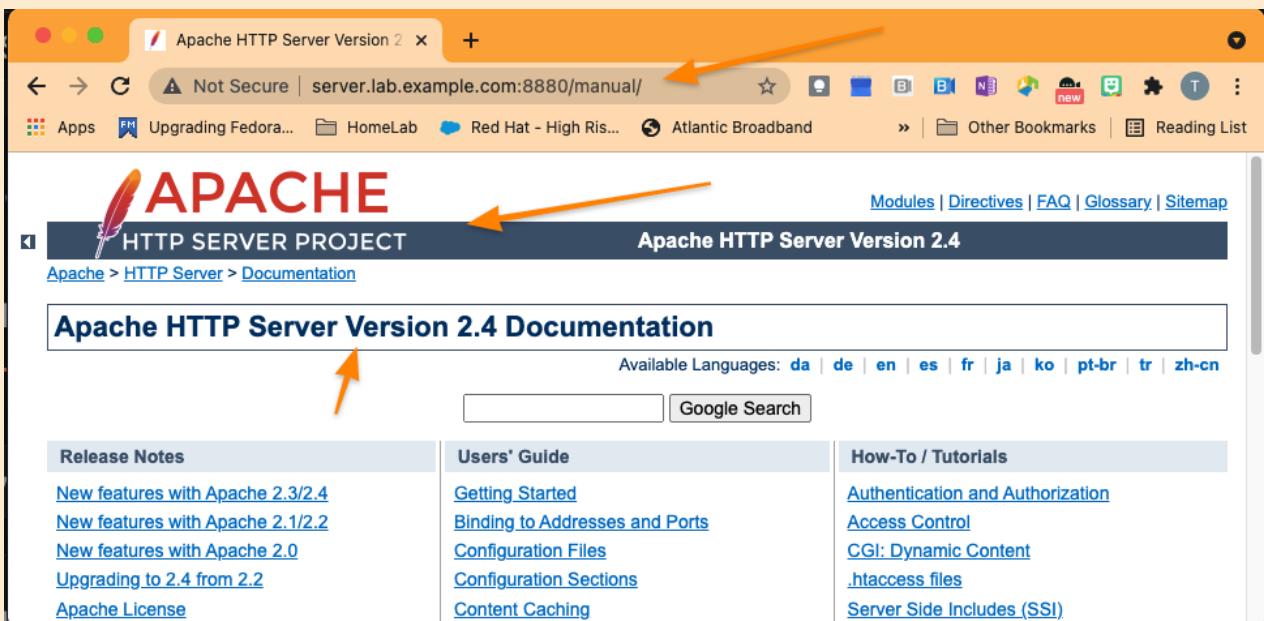


Figure 50. Apache Manual

### Cleaning up the Container

1. Stop and Remove the Container

```
[root@server ~]# podman rm demo-container-buildah --force  
be3bbc8898fc6cab1cef5d78721f46b985e957d30984d591271fffc5b906994
```



### *Image Removal*

Normally we could remove images and cleanup, but we will need to check this image into a registry later. This image shouldn't be removed until it has been checked into a remote container registry.

```
[root@server ~]# podman rmi localhost/demo-container-image
```

2. Delete the **working container** image from the system

*Listing 87. Delete Working Container from System*

```
[root@server ~]# buildah delete working-container  
fb5387521fd7b0c9f6dd567ca0f76222178f0ffbcf38fc990df9d49a60079e18
```

## 4.2.2. Building an Image Using Buildah Rootless

Just like **Podman**, **Buildah** can be used to build container images in **Rootless** mode. The same setup and system configuration applies to building an image in **rootless** mode. The biggest problem with leveraging **Buildah** in rootless mode is the **buildah mount** command. **Buildah** must be used in the **userspace** of the system just like FUSEFS with rootless-podman. Therefore, the **buildah mount** command must be used in conjunction with **buildah unshare** as **buildah unshare** provides the "magic" to allow **buildah** to enter the user namespace.

### Example 17. Lab: Building a Container Image in Rootless mode

We will be completing the steps from above where we build a demo container as a **root** user, but this time we will be leveraging a script to save time. This will be the exact same container image, but will be completed using **buildah unshare <script\_name>**.

1. Setup the lab and distribute the script to the **server** system.

*Listing 88. Change to Proper Directory*

```
[student@workstation ~]$ cd github/OCP_Demos/Containers/labs/Buildah/
```

*Listing 89. Run Ansible Playbook*

```
[student@workstation Buildah]$ ansible-playbook Deploy_Exercise_Files.yml  
... OUTPUT OMITTED ...  
PLAY RECAP ****  
server : ok=2    changed=1    unreachable=0    failed=0  
skipped=0   rescued=0   ignored=0
```

2. SSH to the **server** system as **Student**

```
[student@workstation Buildah]$ ssh student@server  
student@server's password:
```

3. Build the container image using the **rootless\_buildah\_demo.sh** script and Buildah in **rootless** mode

```
[student@server ~]$ buildah unshare ./rootless_buildah_demo.sh
```

4. Verify image was built and exists

```
[student@server ~]$ buildah images  
REPOSITORY      TAG      IMAGE ID      CREATED      SIZE  
localhost/demo-container-image  latest  c283aa80afa7  52 seconds ago  553 MB
```

5. Inspect the Image

```
[student@server ~]$ buildah inspect localhost/demo-container-image  
{
```

```
"Type": "buildah 0.0.1",
"FromImage": "localhost/demo-container-image:latest",

... OUTPUT OMITTED ...


"history": [
  {
    "created": "2021-10-28T19:39:43.657057139Z",
    "created_by": "/bin/sh",
    "author": "Travis Michette <tmichett@redhat.com>"
  }
],
"Docker": {
  "created": "2021-10-28T19:39:37.118906563Z",
  "container_config": {
    "Hostname": "",
    "Domainname": "",
    "User": "",
    "AttachStdin": false,
    "AttachStdout": false,
    "AttachStderr": false,
    "ExposedPorts": {
      "80/tcp": {}
    },
    "Tty": false,
    "OpenStdin": false,
    "StdinOnce": false,
    "Env": null,
    "Cmd": [
      "/usr/sbin/httpd",
      "-DFOREGROUND"
    ]
  }
}

... OUTPUT OMITTED ...


"author": "Travis Michette <tmichett@redhat.com>",
"config": {
  "Hostname": "",
  "Domainname": "",
  "User": "",
  "AttachStdin": false,
  "AttachStdout": false,
  "AttachStderr": false,
  "ExposedPorts": {
    "80/tcp": {}
  },
  "Tty": false,
  "OpenStdin": false,
```

```
"StdinOnce": false,  
"Env": null,  
"Cmd": [  
    "/usr/sbin/httpd",  
    "-DFOREGROUND"  
  
... OUTPUT OMITTED ...  
  
"history": [  
    {  
        "created": "2021-10-28T19:39:43.657057139Z",  
        "author": "Travis Michette <tmichett@redhat.com>",  
        "created_by": "/bin/sh"  
    }  
... OUTPUT OMITTED ...  
  
"History": [  
    {  
        "created": "2021-10-28T19:39:43.657057139Z",  
        "created_by": "/bin/sh",  
        "author": "Travis Michette <tmichett@redhat.com>"  
    }  
,  
"Devices": null  
}
```

## 4.3. Managing Images and System Storage

Podman doesn't have inherent orchestration and management components. It is extremely important that both developers and administrators be aware of the storage space that leftover containers and container images leave behind as well as the possible impact on the system and system resources/performance.

When building multiple images on a workstation or server, it is important to push those images to a remote registry for permanent storage. It is also equally important to clean up any unused images from the system. You've already learned about some of the **podman prune** operations, but it is also necessary to use either **buildah** or **podman** commands to remove images that were built, tested, and pushed to production that are no longer needed on the development system.

### 4.3.1. Pushing Images to an Image Registry with Buildah

Normally after building images and testing the images locally on the developer's workstation, these images should be pushed to a remote container registry for use in production and other environments. We will be leveraging **Buildah** to push the images to the remote container registry.

### Example 18. LAB: Pushing Image to Remote Container Registry

#### 1. Tag the image with **Buildah**

```
[root@server ~]# buildah tag localhost/demo-container-image quay.io/tmichett/demo-container-image
```

#### 2. Login to Container Registry

```
[root@server ~]# buildah login quay.io
Username: tmichett
Password:
Login Succeeded!
```

#### 3. Push the **demo-container-image** to the remote registry

```
[root@server ~]# buildah push quay.io/tmichett/demo-container-image
Getting image source signatures
Copying blob 5b447f9f7dfc done
Copying config bb055dc001 done
Writing manifest to image destination
Copying config bb055dc001 [-----] 0.0b / 538.0b
Writing manifest to image destination
Storing signatures
```

#### *Image and Storage Management*



Generally container images are built locally on a developer workstation. It is important to keep the workstation clean of container images not being used. Once built and pushed to the remote registry, it is a good practice to delete the images that might no longer be needed.

### 4.3.2. Deleting Images from the local image registry with Buildah

In a previous chapter and examples, we've used the **podman rmi** command to remove and delete images and image tags from the local image repository. We will now be using those same options with the **buildah** command to cleanup images built that are no longer needed.

**Example 19. LAB: Using \*buildah rmi to Remove Images\***

We will be using various Buildah commands to list images and remove any unneeded/unused images from the system. We need to ensure that we get all tags removed for an image as well to officially and permanently remove the image from local storage.

**1. List local images with the Buildah commands**

```
[root@server ~]# buildah images
quay.io/tmichett/demo-container-image          latest      bb055dc001a5   14
minutes ago  553 MB ①
localhost/demo-container-image                 latest      bb055dc001a5   14
minutes ago  553 MB ②
```

① Tagged Demo image for pushing

② Original tagged image built using Buildah



While it appears there are two images, both of the **tagged** images have the same image ID of **bb055dc001a5**. In order to remove the image completely, both tags must be used or the image must be removed by the image ID.

**2. Remove image by image ID**

```
[root@server ~]# buildah rmi -f bb055dc001a5
bb055dc001a5c469f7d49d656889250bbc8bd5c60687dcbb25b437b09d950373
```



In order to remove an image by ID that has multiple tags, it is necessary to use the **-f** option to force image removal.

### References

**Getting into the weeds with Buildah:** The buildah unshare command:  
<https://www.redhat.com/sysadmin/buildah-unshare-command>

**How rootless Buildah works: Building containers in unprivileged environments:**  
<https://opensource.com/article/19/3/tips-tricks-rootless-buildah>

**Building and managing container images with Buildah:** <https://mohitgoyal.co/2021/05/16/building-and-managing-container-images-with-buildah/>

**podman-image-prune:** <https://docs.podman.io/en/latest/markdown/podman-image-prune.1.html>

 **podman-system-prune:** <https://docs.podman.io/en/latest/markdown/podman-system-prune.1.html>

**podman-container-prune** <https://docs.podman.io/en/latest/markdown/podman-container-prune.1.html>

**Build Containers the Hard Way:** <https://github.com/tmichett/build-containers-the-hard-way>

**Ansible Blender Project:** <https://github.com/TomasTomecek/ansible-bender>

**Man Pages:** *man podman-image-prune, man podman-system-prune, man podman-container-prune, man buildah, man podman*

# 5. Managing Containers as System Services

## 5.1. Starting Containers Automatically with the Server

**Systemd** user unit files can be created for rootless containers allowing them to be managed with **systemctl** commands similar to regular Linux services. By enabling the services, the associated containers can start when the host machine starts. Furthermore, by running in "rootless" mode the services can be managed from a non-privileged account.

## 5.2. Running Systemd Services as a Regular User

In addition to managing system services, **SystemD** can also manage user services. Users can create their own unit files for their services which can be managed with **systemctl** without requiring **root** access.

When services are enabled as a user service (non-root user), the service automatically starts when you open your first session. The service will stop when you've closed your last session.

### *Forcing User-Enabled Services to Start/Stop with the Server*



It is possible to change the default behavior of user services (**systemd.units**) by using **loginctl enable-linger** command to override the behavior and enable the **Linger** feature. It can be disabled with **loginctl disable-linger**. It is possible to show the **Linger** status for a user using the **loginctl show-user *username*** command.

### 5.2.1. Creating and Managing Systemd User Services

In order to define **systemd** user services, the `~/.config/systemd/user` directory must be created to store the unit files. User **systemd.unit** files have the same syntax as a regular system unit file.

### *More Information on SYSTEMD*



For more information the **systemd.unit** and **systemd.service** man pages should be used.

### *User Services with systemctl*

User services can be controlled with the **systemctl** command. However, the **--user** option must be provided on the command line before the [enable, start, stop, reload] or other options and the service name.

### *Using systemctl --user Commands*



In order for the **systemctl --user** command to function properly, you must be logged in directly as using **sudo** or **su** commands will not work. The **systemctl** command interacts with a per user **systemd --user** process.

Table 2. Classroom Machines

SystemD Tasks	System Services	User Services
<b>Storing custom unit files</b>	/etc/systemd/system/unit.service	~/.config/systemd/user/unit.service
<b>Reloading unit files</b>	systemctl daemon-reload	systemctl --user daemon-reload
<b>Starting and stopping a service</b>	systemctl start UNIT systemctl stop UNIT	systemctl --user start UNIT systemctl --user stop UNIT
<b>Starting a service when the machine starts</b>	systemctl enable UNIT	loginctl enable-linger systemctl --user enable UNIT

## 5.2.2. Managing Containers Using Systemd Services

With a single container host, running small number of containers, user-based **systemd** files can be configured to automatically start the containers with the server. For larger and production based systems, Red Hat OpenShift should be used.

### 5.2.2.1. Creating a Dedicated User Account to Run Containers

To simplify management of rootless containers a dedicated container user account can be created for running all containers.

#### *A Container User Account*

If you are creating a user account for containers, the user account and group should be a regular user account. It cannot be created with the **--system** option as it would then be considered a system account and out of the reserved account IDs for regular user accounts.

### 5.2.2.2. Creating the Systemd Unit File

It is possible to use **podman** to generate the **systemd** unit files. In order to generate a **systemd** unit file, you will use **podman generate systemd** command. This works best when you are in the **~/.config/systemd/user/** directory.

#### *Generating systemd Files with Podman*

The **podman generate systemd** uses a container as a model to create the configuration file, so the container must be deleted after the creation of the **systemd** file as **systemd** expect the container to not already exist.

### podman generate systemd Command Options

- **--name ContainerName** - Specifies name of container to use as a model for generating the unit file. This also defines the name of the container built from the **systemd** unit file.
- **--files** - Instructs Podman to create the unit file in the current directory
- **--new** - Instructs Podman to configure the **systemd** service to create the container when the service starts and delete the container when the service stops. Without the **--new** option, Podman configures the service to start/stop the existing container without deleting it.

#### 5.2.2.3. Starting and Stopping Containers Using Systemd

The **systemctl** command can control containers.

*Avoid using podman for SystemD containers*



Containers managed with **systemctl** commands are controlled by **systemd**. **systemd** monitors the container status and restarts if a failure is detected. Never use **podman** to start/stop **systemd** containers.

#### 5.2.2.4. Configuring Containers to Start When the Host Machine Starts

By default, **systemd** user services start when a user opens the first session and stops when a user closes the last session on a container host. It is possible to enable the service to start/stop with the server. In order to enable starting/stopping automatically **loginctl enable-linger** must be run.

*Listing 90. Enable Login Linger*

```
loginctl enable-linger
```

*Listing 91. Enable a Container to Start Automatically*

```
systemctl --user enable container-web
```

*Listing 92. Disable a Container to Starting Automatically*

```
systemctl --user disable container-web
```

#### 5.2.3. Managing Containers Running as Root with Systemd

It is also possible to configure containers that should be run as **root** to be managed with **SystemD** unit files. The advantage to using containers as services running as root is that they work exactly like normal system unit files.

##### *Configuring root SystemD Container Files*

- No dedicated user needed

- When **podman generate systemd** is run, it should be in the **/etc/systemd/ system** directory so that the **systemd** unit file is in the correct directory.
- You can use **systemctl** and do not need the **--user** option
- There is no need to enable **Linger** with the **logind enable-linger** command.

#### 5.2.4. Orchestrating Containers at Scale

Red Hat Training offers other courses, starting with the free technical overview course Deploying Containerized Applications (DO080) and continuing with Introduction to Containers, Kubernetes, and OpenShift (DO180). For more information, visit <https://www.redhat.com/training>.

A number of resources are available there, including ways to try out OpenShift for yourself using tools like CodeReady Containers [<https://developers.redhat.com/products/codeready-containers/overview>]. See <https://www.openshift.com/try> for more information.

### 5.3. DEMO - Managing Containers as Services

*Example 20. DEMO - Managing Containers as Services*

## 1. Create a Container User

```
[root@server ~]# useradd cuser
[root@server ~]# passwd cuser
Changing password for user cuser.
New password: redhat
BAD PASSWORD: The password is shorter than 8 characters
Retype new password: redhat
passwd: all authentication tokens updated successfully
```

2. Login to **server** with the **cuser** that was just created

```
[student@workstation ~]$ ssh cuser@servera
Activate the web console with: systemctl enable --now cockpit.socket

This system is not registered to Red Hat Insights. See https://cloud.redhat.com/
To register this system, run: insights-client --register

[cuser@servera ~]$
```

## 3. Create Directories and Config Files for Container Registry

```
[cuser@server ~]$ mkdir -p ~/.config/containers/
[cuser@server ~]$ cd ~/.config/containers/
[cuser@server containers]$ scp
student@workstation:github/OCP_Demos/Containers/labs/SystemD/registries.conf .
```

## 4. Create Content for Webserver

```
[cuser@server ~]$ mkdir -p ~/www/html
[cuser@server ~]$ echo "I am a demo webserver" > ~/www/html/index.html
```

## 5. Login to the container registry

```
[cuser@server ~]$ podman login quay.io  
Username: <UN>  
Password: <PW>  
Login Succeeded!
```

## 6. Create and Run Reference Container

```
[cuser@server ~]$ podman run -d --name demoweb -p 8080:80 -v ~/www:/var/www:Z  
quay.io/redhattraining/httpd-parent
```

## 7. Verify webserver is running

```
[cuser@server ~]$ curl localhost:8080  
I am a demo webserver
```

## 8. Prepare the directory and create a **systemd** unit file

```
[cuser@server ~]$ mkdir -p ~/.config/systemd/user/  
  
[cuser@server ~]$ cd ~/.config/systemd/user/  
  
[cuser@server user]$ podman generate systemd --name demoweb --files --new  
/home/cuser/.config/systemd/user/container-demoweb.service
```

## 9. View **systemd** Unit File

```
[cuser@server user]$ cat /home/cuser/.config/systemd/user/container-demoweb.service
# container-demoweb.service
# autogenerated by Podman 3.0.2-dev
# Tue Oct 26 11:22:02 EDT 2021

[Unit]
Description=Podman container-demoweb.service
Documentation=man:podman-generate-systemd(1)
Wants=network.target
After=network-online.target

[Service]
Environment=PODMAN_SYSTEMD_UNIT=%n
Restart=on-failure
TimeoutStopSec=70
ExecStartPre=/bin/rm -f %t/container-demoweb.pid %t/container-demoweb.ctr-id
ExecStart=/usr/bin/podman run --common-pidfile %t/container-demoweb.pid --cidfile
%t/container-demoweb.ctr-id --cgroups=no-common --replace -d --name demoweb -p
8080:80 -v /home/cuser/www:/var/www:Z quay.io/redhattraining/httpd-parent
ExecStop=/usr/bin/podman stop --ignore --cidfile %t/container-demoweb.ctr-id -t 10
ExecStopPost=/usr/bin/podman rm --ignore -f --cidfile %t/container-demoweb.ctr-id
PIDFile=%t/container-demoweb.pid
Type=forking

[Install]
WantedBy=multi-user.target default.target
```

## 10. Remove the Reference Container and Verify

```
[cuser@server user]$ podman rm demoweb -f
c0c418905cb571cbbade303c94486568b3ac87ca3afe72be92a957be421bfcc4

[cuser@server user]$ podman ps -a
CONTAINER ID  IMAGE      COMMAND     CREATED    STATUS    PORTS      NAMES
```

## 11. Use **systemctl** to Start and Enable the Container Service

```
[cuser@server user]$ systemctl --user enable container-demoweb --now
Created symlink /home/cuser/.config/systemd/user/multi-
user.target.wants/container-demoweb.service →
/home/cuser/.config/systemd/user/container-demoweb.service.
Created symlink /home/cuser/.config/systemd/user/default.target.wants/container-
demoweb.service → /home/cuser/.config/systemd/user/container-demoweb.service.
```



### SystemD Container Services

Once a containerized application has been provisioned as a SystemD service, it cannot be managed and should not be managed with **podman** instead, the service should be managed with the **user systemctl** SystemD commands.

## 12. Enable **user-linger** so that the service starts when the system boots

```
[cuser@server ~]$ loginctl enable-linger cuser
```



### Must Enable User Linger Service

The **enable-linger** will allow SystemD services to start as a user when the system boots. Failure to enable this will result in services being offline until the user logs in for the first time.

## 13. Verify Container is Running

```
[cuser@server user]$ curl localhost:8080
I am a demo webserver

[cuser@server user]$ podman ps
CONTAINER ID  IMAGE                               COMMAND             CREATED
STATUS        PORTS                NAMES
6da1c129c3e2  quay.io/redhattraining/httpd-parent /bin/sh -c /usr/s...  About a
minute ago   Up About a minute ago  0.0.0.0:8080->80/tcp  demoweb
```

## 14. Reboot the server

```
[cuser@server user]$ su -
Password:
Last login: Fri Nov  5 16:12:41 EDT 2021 from 172.25.250.9 on pts/0

[root@server ~]# reboot
```

## 15. Log back into the server as any user other than **cuser**

```
[student@workstation Buildah]$ ssh root@server
```

## 16. Verify Status of Website

```
[root@server ~]# curl localhost:8080  
I am a demo webserver
```

## 17. Verify that Container isn't running as root

```
[root@server ~]# podman ps -a  
CONTAINER ID  IMAGE   COMMAND   CREATED   STATUS    PORTS     NAMES
```

## 18. Cleanup Service and Files for Upcoming Exercises

*Listing 93. Disable and Stop Container and Service*

```
[cuser@server ~]$ systemctl --user disable container-demoweb.service --now  
Removed /home/cuser/.config/systemd/user/multi-user.target.wants/container-  
demoweb.service.  
Removed /home/cuser/.config/systemd/user/default.target.wants/container-  
demoweb.service.
```



### *Warning Header*

The session must be logged into **cuser** through SSH in order for **Linger** and **SystemD** to recognize the **systemctl --user** command!

*Listing 94. Verify Container was Stopped and Removed*

```
[cuser@server ~]$ podman ps -a  
CONTAINER ID  IMAGE   COMMAND   CREATED   STATUS    PORTS     NAMES
```

### References

**Managing containerized system services with Podman:**  
<https://developers.redhat.com/blog/2018/11/29/managing-containerized-system-services-with-podman>

**Improved systemd integration with Podman 2.0:** <https://www.redhat.com/sysadmin/improved-systemd-podman>

**RH134 Course - Section 13.11: Managing Containers as Services:**  
<https://rol.redhat.com/rol/app/courses/rh134-8.2/pages/ch13s11>

**podman generate systemd:** <https://docs.podman.io/en/latest/markdown/podman-generate-systemd.1.html>

**podman auto-update:** <https://github.com/containers/podman/blob/v2.0/docs/source/markdown/podman-auto-update.1.md>

**Running containers with Podman and shareable systemd services:**  
<https://www.redhat.com/sysadmin/podman-shareable-systemd-services>

**How I learned to stop worrying and love systemd:** <https://www.redhat.com/sysadmin/love-systemd>

**Mastering systemd: Securing and sandboxing applications and services:**  
<https://www.redhat.com/sysadmin/mastering-systemd>

**man pages:** *podman-generate-systemd, podman-auto-update*



# 6. Container Management with Ansible

Containers can now be managed with the Ansible Podman collection leveraging Ansible Automation Platform (AAP). Red Hat curates and maintains several supported Ansible Collections at Ansible Automation Hub. The community collections are available to be installed from Ansible Galaxy.

## 6.1. Ansible Refresher

The following is a small reminder about Ansible and the basics around an Ansible Control node for managing systems with Ansible.

### 6.1.1. Ansible Basics

In order to use Ansible, the following items are required:

- **ansible** is installed on the Control Node
- An **ansible.cfg** file exists and points to a working inventory
- An **inventory** file exists containing managed nodes

#### 6.1.1.1. Ansible Concepts and Architecture

##### *Ansible Machine Types*

- **Control Node:** Location where Ansible is installed and used to run playbooks and execute Ansible ad-hoc commands
- **Managed Host:** Network device that is managed by an Ansible control node

The required Ansible components are the following:

- **ansible.cfg:** Ansible configuration file with directives on how Ansible should work
- **inventory:** Listing and organization of ansible managed nodes/hosts
- **module:** Small piece of code to perform a variety of tasks (generally written in Python or Powershell)
- **plugins:** Code that can be used to extend and adapt Ansible to new uses. Capable of manipulating data and extracting information from output. (*Used more in the next course DO447*)
- **Ansible Tower/API:** Framework to control and manage Ansible at scale. Ansible Tower is not a core part of Ansible, but is an add-on product to more effectively utilize Ansible with teams.

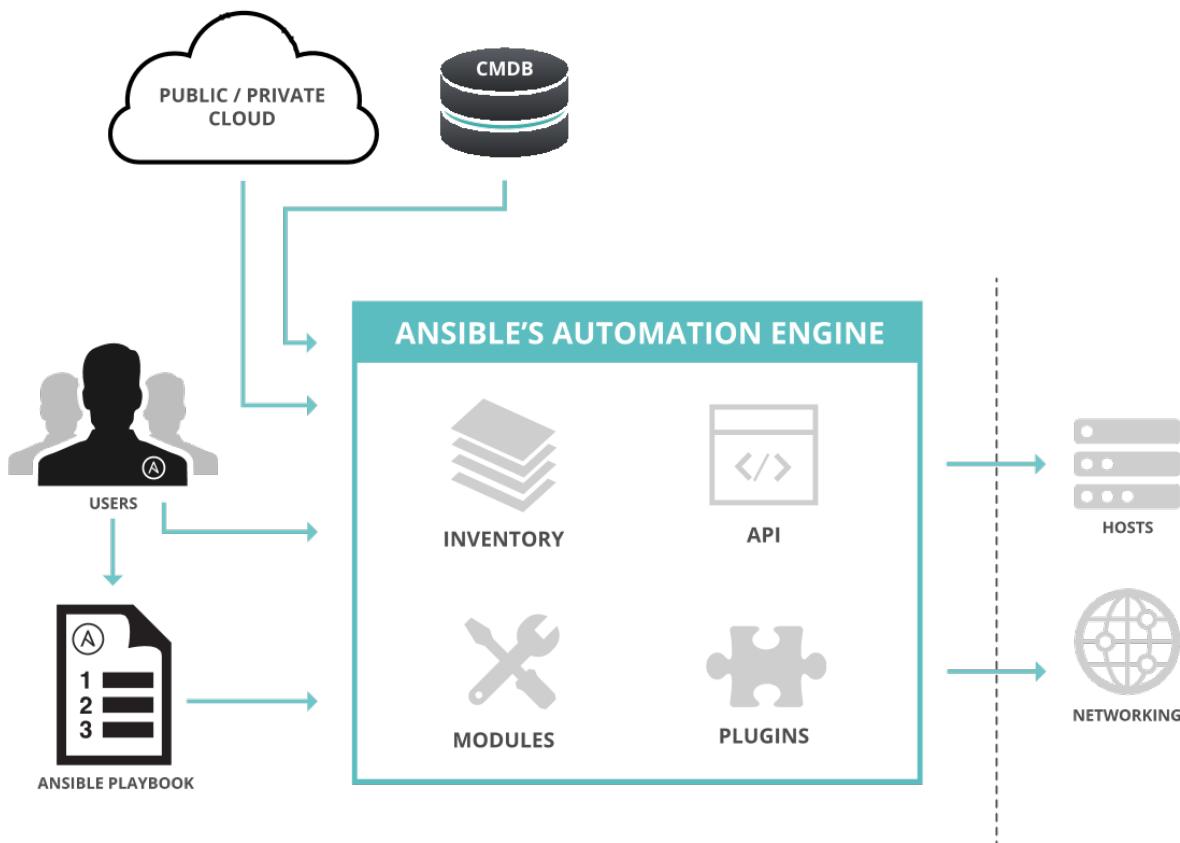


Figure 51. Ansible Architecture

### 6.1.1.2. Building an Ansible Inventory

An **inventory** file is the minimum needed item for Ansible to run. Inventory files provide a listing of hosts for Ansible to manage. There are multiple ways to define an inventory file.

#### 6.1.1.2.1. Static Inventory

Static inventory files are generally defined in the INI format.

*Listing 95. Simple Inventory*

```
web1.example.com
web2.example.com
db1.example.com
db2.example.com
192.0.2.42
```

```
[webservers]
web1.example.com
web2.example.com
192.0.2.42
```

### 6.1.1.3. Ansible Configuration Files

It is important to note, there can be multiple Ansible configuration files. Ansible will look for and process configuration files in order.

1. Location specified in **ANSIBLE\_CONFIG**
2. The **ansible.cfg** file in the current working directory
3. The **.ansible.cfg** file in the user's home directory
4. The default **/etc/ansible/ansible.cfg** file



It is important to note that the first file in this order Ansible sees is what will be used. Therefore, if anything is setup for the **ansible.cfg** file in locations 1-3, it won't ever use the default file **/etc/ansible/ansible.cfg**



**Ansible Configuration file documentation:** [http://docs.ansible.com/ansible/intro\\_configuration.html](http://docs.ansible.com/ansible/intro_configuration.html)

Table 3. Ansible Settings

Setting	Command Line Option
<b>inventory</b>	Location of the Ansible inventory file.
<b>remote_user</b>	The remote user account used to establish connections to managed hosts.
<b>become</b>	Enables or disables privilege escalation for operations on managed hosts.
<b>become_method</b>	Defines privilege escalation method on managed hosts.
<b>become_user</b>	User account to escalate privileges to on managed hosts.
<b>become_ask_pass</b>	Defines whether privilege escalation on managed hosts should prompt for password.

### *Ansible Collections*

Starting with Ansible 2.9 and the new Ansible (AAP), collections have been introduced and allow modules, roles, and other components to be stored in Ansible collections. Similar to Ansible Roles, Ansible Collections must be installed on the system and available for Ansible playbooks.

Ansible Collections can be specified in the **ansible.cfg** file for the path where collections have been installed. As with Ansible Roles, it is common to install Ansible collections to the working directory in a sub-directory called **collections**. When working with Ansible Collections it is necessary to provide this information in the **ansible.cfg** file.



#### *Listing 96. ansible.cfg for Using Collections*

```
[defaults]
remote_user = root
inventory = inventory
COLLECTIONS_PATHS =
~/.ansible/collections:/usr/share/ansible/collections:./collections

[privilegeEscalation]
become = False
become_method = sudo
become_user = root
become_ask_pass = False
```

## 6.2. Ansible Podman Collection

The Ansible Podman collection is a required component in order to have the Ansible Modules needed to manage containers using Podman. Currently, the **containers.podman** is only available via Ansible Galaxy.

### 6.2.1. Obtaining Podman Collections

In order to leverage Ansible to officially manage Podman containers, the **containers.podman** collection must be installed on the Ansible Control node. Currently, there are two main sources for installation of Ansible collections.

#### *Ansible Collections*

- Red Hat supported collections for Ansible can be downloaded from Ansible Automation Hub (<https://console.redhat.com/ansible/automation-hub>).
- Ansible Community collections can be downloaded from Ansible Galaxy (<https://galaxy.ansible.com/containers/podman>)

### *Ansible Automation Hub and Ansible Galaxy*

It is required that you have a Red Hat Subscription for Ansible Automation Platform (AAP2.0) in order to access Red Hat Supported Ansible collections. The lab and exercises will use the Ansible Galaxy Podman collection.

#### *Listing 97. Installing Ansible Galaxy Podman Collection*



```
ansible-galaxy collection install containers.podman
```

#### **containers.podman Documentation**

<https://galaxy.ansible.com/containers/podman>   <https://docs.ansible.com/ansible/latest/collections/containers/podman/index.html>   [https://docs.ansible.com/ansible/latest/collections/containers/podman/podman\\_container\\_module.html](https://docs.ansible.com/ansible/latest/collections/containers/podman/podman_container_module.html)

## **6.2.2. Installing and Using the containers.podman Collection**

There are multiple methods for the Podman collection to be installed, however, for this course and the exercises, the collection will be installed locally to the **./collections** sub-folder using a requirements.yml file. The **ansible.cfg** will point to this as already available in the path and the Ansible playbooks being utilized will reference the **collections** at the top of the playbook similar to Ansible roles so that throughout the playbook tasks, the shorter module name can be referenced.

### Example 21. LAB: Installing the `containers.podman` Collection

#### *Installing the Podman Collection*

Similar to installation of Ansible Roles, Ansible Collections can be installed in the local working directory in a sub-directory called **collections**. The collections needed for the Ansible projects can be specified in a **requirements.yml** file and installed just like Ansible Roles.

1. Change to the `github/OCP_Demos/Containers/labs/Ansible_Image/` Directory

```
[student@workstation ~]$ cd github/OCP_Demos/Containers/labs/Ansible_Image/
```

2. Create a **requirements.yml** File

#### *Listing 98. Podman Collection requirements.yml File*

```
collections:  
  - name: containers.podman ①
```

① Listing of collections to download and install

3. Install the roles/collections from the **requirements.yml** File

#### *Listing 99. Installing the Collections*

```
[student@workstation Ansible_Image]$ ansible-galaxy collection install -r  
requirements.yml -p ./collections  
Process install dependency map  
Starting collection install process  
Installing 'containers.podman:1.8.1' to  
'/home/student/github/OCP_Demos/Containers/labs/Ansible_Image/collections/ansible_  
collections/containers/podman'
```

4. Verify the collection(s) are installed

*Listing 100. Verifying Installed Collections*

```
[student@workstation Ansible_Image]$ tree collections/
collections/
└── ansible_collections
    └── containers
        └── podman
            ├── ansible-collection-containers-podman.spec
            ├── CHANGELOG.rst
            └── changelog

... OUTPUT OMITTED ...

    └── sanity
        ├── ignore-2.10.txt
        ├── ignore-2.11.txt
        ├── ignore-2.12.txt
        ├── ignore-2.9.txt
        └── requirements.txt

64 directories, 146 files
```

### Using Collections

Referencing the **collection** and using shorter module names. This allows a cleaner and more streamlined approach and is generally consistent with the older Ansible versions. It is 100% fully acceptable to utilize the fully qualified module names, however, for the course and ease of use we will continue referencing Ansible modules by the shortened/condensed name.

*Listing 101. Sample Playbook with Collections*



```
---
- name: Deploy Quay Mirror
  hosts: quay
  vars:
    QUAY_DIR: /quay
  vars_files:
    - registry_login.yml
  collections: ①
    - containers.podman

  tasks:

## Podman Collections Needed for Login
  - name: Login to Container Registry
    podman_login: ②
      username: "{{ registry_un }}"
      password: "{{ registry_pass }}"
      registry: "{{ registry_url }}"
```

① Importing the collection(s) to be used and referenced by short module names in the playbook.

② Leveraging modules by short module names in the playbook.

## 6.3. Building Container Images Using Ansible

The **podman\_image** module from the **containers.podman** collection is needed in order to work with and manipulate container images. In order to build images, it is necessary to have either a valid **Containerfile** or valid **Dockerfile** for image building.

**Example 22. LAB: Building Container Images with Ansible***Building Podman Container Images with Ansible*

1. Change to the **github/OCP\_Demos/Containers/labs/Ansible\_Image/** Directory

```
[student@workstation ~]$ cd github/OCP_Demos/Containers/labs/Ansible_Image/
```

2. Verify the **ansible.cfg** file for proper collection usage

*Listing 102. ansible.cfg*

```
[defaults]
remote_user = root
inventory = inventory
COLLECTIONS_PATHS =
~/ansible/collections:/usr/share/ansible/collections:./collections
```

3. Edit the **httpd\_ansible\_demo.yml** file

```
[student@workstation Ansible_Image]$ vim httpd_ansible_demo.yml
---
- name: Build and Upload a Container Image
  hosts: localhost
  vars_files:
    ## Add Variables
  collections:
    ## Add Collections
  tasks:
    - name: Login to Container Registry
      ## Add Login task
      - name: Build and Push Custom Container Image
        ## Add Container build task
```

4. Start with the top playbook definitions and directives

*Listing 103. Ansible Playbook Directives*

```
---
- name: Build and Upload a Container Image
  hosts: localhost
  vars_files:
    - vars/registry_login.yml ①
  collections:
    - containers.podman ②
...
... OUTPUT OMITTED ...
```

① Always provide registry login information

② Always specify the collections to be used

## 5. Create Playbook with Tasks

*Listing 104. Playbook Tasks*

```
tasks:
- name: Login to Container Registry ①
  podman_login:
    username: "{{ registry_un }}"
    password: "{{ registry_pass }}"
    registry: "{{ registry_url }}"
- name: Build and Push Custom Container Image ②
  podman_image:
    name: httpd_demo_ansible ③
    path: ./ ④
    push: yes ⑤
    username: "{{ registry_un }}" ⑥
    password: "{{ registry_pass }}" ⑦
    push_args: ⑧
    dest: "{{ registry_url }}/{{ registry_un }}"
```

① Initial task to login to registry

② Task using the **podman\_image** module to build and push an image

③ Name/Tag of the image being built

④ Path to the **Containerfile** or **Dockerfile**

⑤ Define whether or not to push to remote image registry

⑥ Registry Username

⑦ Registry Password

⑧ Arguments used for pushing to remote registry. Works in conjunction with **push: yes**. Must

have registry URL specified along with the location (*which is generally the repository username*).

## 6. Explore entire playbook

```
[student@workstation Ansible_Image]$ cat httpd_ansible_demo.yml
---
- name: Build and Upload a Container Image
  hosts: localhost
  vars_files:
    - vars/registry_login.yml ①
  collections:
    - containers.podman

  tasks:
    - name: Login to Container Registry ②
      podman_login:
        username: "{{ registry_un }}"
        password: "{{ registry_pass }}"
        registry: "{{ registry_url }}"

    - name: Build and Push Custom Container Image ③
      podman_image:
        name: httpd_demo_ansible ④
        path: ./ ⑤
        push: yes ⑥
        username: "{{ registry_un }}" ⑦
        password: "{{ registry_pass }}" ⑧
        push_args: ⑨
        dest: "{{ registry_url }}/{{ registry_un }}"
```

① Variables file allowing login to registry

② Initial task to login to registry

③ Task using the **podman\_image** module to build and push an image

④ Name/Tag of the image being built

⑤ Path to the **Containerfile** or **Dockerfile**

⑥ Define whether or not to push to remote image registry

⑦ Registry Username

⑧ Registry Password

⑨ Arguments used for pushing to remote registry. Works in conjunction with **push: yes**. Must have registry URL specified along with the location (*which is generally the repository username*).

## 7. Edit the `vars/registry_login.yml` file

```
[student@workstation Ansible_Image]$ cp vars/registry_login_demo.yml  
vars/registry_login.yml ; vim vars/registry_login.yml  
registry_un: UN_Goes_Here ①  
registry_pass: Password_Goes_Here ②  
registry_url: quay.io
```

① Replace with your Quay Username

② Replace with your Quay Password

## 8. Run the playbook to build the image

```
[student@workstation Ansible_Image]$ ansible-playbook httpd_ansible_demo.yml
```

Playbook will fail if you don't have the `containers.podman` collection installed.



*Listing 105. Installation of Collections*

```
[student@workstation Ansible_Image]$ ansible-galaxy collection  
install -r requirements.yml -p ./collections
```

## 9. Verify that the `httpd_demo_ansible` and added to the local registry.

```
[student@workstation Ansible_Image]$ podman images  
REPOSITORY                                     TAG      IMAGE ID  
CREATED          SIZE  
localhost/httpd_demo_ansible                  latest   95857ed812de  
About a minute ago   255 MB
```

## 6.4. Deploying Podman Containers Using Ansible

The `podman_container` module from the `containers.podman` collection is needed in order to create/run/remove containers. In order to create containers from container images, it is necessary to have access to the container images locally.

**Example 23. LAB: Creating and Running Containers with Ansible***Creating and Running Podman Containers with Ansible*

1. Change to the /home/student/github/OCP\_Demos/Containers/labs/Ansible\_Container Directory

```
[student@workstation ~]$ cd  
/home/student/github/OCP_Demos/Containers/labs/Ansible_Container
```

2. Verify the **ansible.cfg** file for proper collection usage

*Listing 106. ansible.cfg*

```
[defaults]  
remote_user = root  
inventory = inventory  
COLLECTIONS_PATHS =  
~/.ansible/collections:/usr/share/ansible/collections:./collections
```

3. Create a **requirements.yml** File

*Listing 107. Podman Collection requirements.yml File*

```
[student@workstation Ansible_Container]$ vim requirements.yml  
collections:  
- name: containers.podman ①
```

① Listing of collections to download and install

4. Install the roles/collections from the **requirements.yml** File

*Listing 108. Installing the Collections*

```
[student@workstation Ansible_Image]$ ansible-galaxy collection install -r  
requirements.yml -p ./collections  
Process install dependency map  
Starting collection install process  
Installing 'containers.podman:1.8.1' to  
'/home/student/github/OCP_Demos/Containers/labs/Ansible_Image/collections/ansible_  
collections/containers/podman'
```

5. Verify the collection(s) are installed

*Listing 109. Verifying Installed Collections*

```
[student@workstation Ansible_Image]$ tree collections/
collections/
└── ansible_collections
    └── containers
        └── podman
            ├── ansible-collection-containers-podman.spec
            ├── CHANGELOG.rst
            └── changelog

... OUTPUT OMITTED ...

└── sanity
    ├── ignore-2.10.txt
    ├── ignore-2.11.txt
    ├── ignore-2.12.txt
    ├── ignore-2.9.txt
    └── requirements.txt
```

64 directories, 146 files

6. Edit the **deploy-httd-ansible-demo.yml** Starter Playbook

```
[student@workstation Ansible_Container]$ vim deploy-httd-ansible-demo.yml
```

## 7. Start with the top playbook definitions and directives

*Listing 110. Ansible Playbook Directives*

```
---
- name: Launch Container from Red Hat Training Image
  hosts: server
  collections:
    - containers.podman ①
... OUTPUT OMITTED ...
```

① Always specify the collections to be used

## 8. Create the Playbook with Tasks

- Use the **quay.io/redhattraining/httpd-parent:2.4**
- Launch Container with network port on 7080
- Test Website

*Listing 111. Playbook Tasks*

```
tasks:  
  
## Start and Run the HTTPD Container  
- name: Start the Apache Container  
  podman_container:  
    name: Website_Demo  
    image: quay.io/redhattraining/httpd-parent:2.4  
    state: started  
    restart: yes  
    ports:  
      - "7080:80"  
  
## Open Firewall for Automated Testing  
- name: Open Firewall for Website  
  firewalld:  
    port: 7080/tcp  
    permanent: true  
    state: enabled  
    immediate: yes  
  
- name: Test Website  
  hosts: localhost  
  tasks:  
    - name: Test Website  
      uri:  
        url: http://server:7080  
        return_content: yes  
        status_code: 200  
      register: result  
  
    - name: Test Website Output  
      debug:  
        msg: 'The website returned: "{{ result.content }}"'
```

## 9. Verify entire playbook

```
---
- name: Deploy HTTPD Server Demo
  hosts: server
  collections:
    - containers.podman

  tasks:

## Start and Run the HTTPD Container
  - name: Start the Apache Container
    podman_container:
      name: Website_Demo
      image: quay.io/redhattraining/httpd-parent:2.4
      state: started
      restart: yes
      ports:
        - "7080:80"

## Open Firewall for Automated Testing
  - name: Open Firewall for Website
    firewalld:
      port: 7080/tcp
      permanent: true
      state: enabled
      immediate: yes

  - name: Test Website
    hosts: localhost
    tasks:
      - name: Test Website
        uri:
          url: http://server:7080
          return_content: yes
          status_code: 200
        register: result

      - name: Test Website Output
        debug:
          msg: 'The website returned: "{{ result.content }}"'
```

## 10. Run Ansible Playbook and Verify Success

```
[student@workstation Ansible_Container]$ ansible-playbook deploy-httpd-ansible-demo.yml
```

## 6.4.1. Analyzing Running Containers with Ansible

The **podman** collection has several components to manage and collect information from containers. The **podman\_container\_info** module can be used to get information about running containers. It requires the name of the container and collects all information as a list/dictionary.

### *Podman Container Info Module*

It is important to note that many of the Ansible **fact** modules have been replaced with **info** modules so the facts are returned as lists/dictionaries. You should consult each module's information page and examine the **Return Values** section.

**podman\_container\_info:** [https://docs.ansible.com/ansible/latest/collections/containers/podman/podman\\_container\\_info\\_module.html#ansible-collections-containers-podman-podman-container-info-module](https://docs.ansible.com/ansible/latest/collections/containers/podman/podman_container_info_module.html#ansible-collections-containers-podman-podman-container-info-module)

It is also important to know that in order to access specific elements from the information returned, a mapping of the value must be performed.

### *Listing 112. Mapping Element Needed*



```
- name: Gather Information on Running Containers
  podman_container_info: ①
    name: Website_Demo
    register: container_info

- name: Set Fact for Running Container Image
  set_fact: ②
    ImageName: "{{ container_info.containers | map(attribute='ImageName') | list }}"

- name: Display Running Container Image
  debug: ③
    var: ImageName
```

① Using the **podman\_container\_info** to gather all facts and information

② Using the **set\_fact** module to map the attribute of the information to a known variable name

③ Displaying the information from the set fact

**Example 24. LAB: Analyze a running container with Ansible**

1. Change to `/home/student/github/OCP_Demos/Containers/labs/Ansible_Container` if not already there.

```
[student@workstation Ansible_Container]$ cd  
/home/student/github/OCP_Demos/Containers/labs/Ansible_Container
```

2. Explore the **Gather\_Info\_Demo.yml** playbook

```
[student@workstation Ansible_Container]$ cat Gather_Info_Demo.yml  
---  
- name: Container Information  
  hosts: localhost  
  
  collections:  
    - containers.podman  
  
  tasks:  
  
    - name: Gather Information on Running Containers  
      podman_container_info:  
        name: Website_Demo  
        register: container_info  
  
    - name: Running Container Information  
      debug:  
        var: container_info  
  
    - name: Set Fact for Running Container Launch Command  
      set_fact:  
        Args: "{{ container_info.containers | map(attribute='Args') | list }}"  
  
    - name: Display Running Container Launch Command  
      debug:  
        var: Args  
  
    - name: Set Fact for Running Container Image  
      set_fact:  
        ImageName: "{{ container_info.containers | map(attribute='ImageName') |  
list }}"  
  
    - name: Display Running Container Image  
      debug:  
        var: ImageName
```

### 3. Run the `Gather_Info_Demo.yml` Playbook

```
[student@workstation Ansible_Container]$ ansible-playbook Gather_Info_Demo.yml

PLAY [Container Information] ****

TASK [Gathering Facts] ****
ok: [server]

TASK [Gather Information on Running Containers] ****
ok: [server]

TASK [Running Container Information] ****
ok: [server] => {
    "container_info": {

        ... OUTPUT OMITTED ...

        "Image": "quay.io/redhattraining/httpd-parent:2.4",

        ... OUTPUT OMITTED ...

        "PortBindings": {
            "80/tcp": [
                {
                    "HostIp": "",
                    "HostPort": "7080"
                }
            ]
        }

        ... OUTPUT OMITTED ...

        "Name": "Website_Demo",
        "Namespace": "",
        "NetworkSettings": {

            ... OUTPUT OMITTED ...

        }

        TASK [Set Fact for Running Container Launch Command] ****
        ok: [server]

        TASK [Display Running Container Launch Command] ****
        ok: [server] => {
            "Args": [
                [
                    "-c",
                    "/usr/sbin/httpd -DFOREGROUND"
                ]
            ]
        }

    }
}
```

```
}
```

```
TASK [Set Fact for Running Container Image] ****
ok: [server]
```

```
TASK [Display Running Container Image] ****
ok: [server] => {
    "ImageName": [
        "quay.io/redhattraining/httpd-parent:2.4"
    ]
}
```

```
PLAY RECAP ****
server                  : ok=7      changed=0      unreachable=0      failed=0
skipped=0      rescued=0      ignored=0
```

## 6.4.2. Controlling Running Containers with Ansible

The **podman collection** has several components to manage and collect information from containers. The **podman\_container** module can be used to interact with containers.

*Listing 113. Example of Stopping / Removing a Container and Deleting a Container Image*

```
---
```

```
- name: Container Information
  hosts: localhost
  vars_files:
    - vars/registry_login.yml
  collections:
    - containers.podman

  tasks:

    - name: Stop Running Container
      podman_container:
        name: Website_Demo
        state: stopped

    - name: Remove Stopped Container
      podman_container:
        name: Website_Demo
        state: absent

    - name: Remove Container Image
      podman_image:
        name: quay.io/redhattraining/httpd-parent:2.4
        state: absent
```

**Example 25. LAB: Clean up containers and container images****1. Get Listing of Running Containers**

```
[student@workstation Ansible_Container]$ ssh root@server "podman ps -a | grep -i website"  
2ad6b0954ed8 quay.io/redhattraining/httpd-parent:2.4 /bin/sh -c  
/usr/s... 11 minutes ago Up 11 minutes ago 0.0.0.0:7080->80/tcp  
Website_Demo
```

**2. Get Listing of Container Images**

```
[student@workstation Ansible_Container]$ ssh root@server "podman images | grep -i httpd-parent"  
quay.io/redhattraining/httpd-parent 2.4 3639ce1374d3 2 years ago 236 MB
```

**3. Analyze the `Stop_and_Remove_Demo.yml` Playbook**

```
[student@workstation Ansible_Container]$ cat Stop_and_Remove_Demo.yml  
---  
- name: Container Information  
  hosts: server  
  collections:  
    - containers.podman  
  
  tasks:  
  
    - name: Stop Running Container  
      podman_container:  
        name: Website_Demo  
        state: stopped  
  
    - name: Remove Stopped Container  
      podman_container:  
        name: Website_Demo  
        state: absent  
  
    - name: Remove Container Image  
      podman_image:  
        name: quay.io/redhattraining/httpd-parent:2.4  
        state: absent
```

**4. Run the `Stop_and_Remove_Demo.yml` Playbook**

```
[student@workstation Ansible_Container]$ ansible-playbook Stop_and_Remove_Demo.yml

PLAY [Container Information] ****
TASK [Gathering Facts] ****
ok: [server]

TASK [Stop Running Container] ****
changed: [server]

TASK [Remove Stopped Container] ****
changed: [server]

TASK [Remove Container Image] ****
changed: [server]

PLAY RECAP ****
server : ok=4    changed=3    unreachable=0    failed=0
skipped=0   rescued=0   ignored=0
```

5. Verify container is gone.

```
[student@workstation Ansible_Container]$ ssh root@server "podman ps -a | grep -i website"
```

6. Verify container image is gone.

```
[student@workstation Ansible_Container]$ ssh root@server "podman images | grep -i httpd-parent"
```

There may be other versions of the image on the system as we are only removing the image tagged with the version tag "2.4". Depending on cleanup, it is possible that a version tagged with "latest" might exist.



```
[student@workstation Ansible_Container]$ ssh root@server "podman
images | grep -i httpd-parent"
quay.io/redhattraining/httpd-parent      latest
4346d3cace25  2 years ago   236 MB
```

*References*

**Podman Collections Github Project:** <https://github.com/containers/ansible-podman-collections>



**Podman Collection Ansible Galaxy:** <https://galaxy.ansible.com/containers/podman>

**Podman Collections from Ansible Docs:** <https://docs.ansible.com/ansible/latest/collections/containers/podman/>

# 7. Quay Image Registry

The Quay image registry depends on multiple sources and configuration files. The exercises here will go through a local "Proof of Concept" deployment. It will be necessary to open up firewall ports as well as provide persistent storage mount points that can be mounted and accessed within the containers.

There are multiple options for deploying Quay locally. For the purpose of this course, we will be installing the Quay registry with both the Security Scanner and the Mirroring Worker.

## Quay Deployment Options

- Stand-Alone Quay Registry
- Quay Registry with a Mirroring Worker
- Quay Registry with Clair Security Scanner
- **Quay Registry with Clair Security Scanner and Mirroring Worker**

### *SSL and Security*



All deployment scenarios allow the use of creating SSL/TLS certificates to secure the network connection. For the purpose of this content, we will be omitting the steps for generating the SSL/TLS certificates and accessing the registry via HTTPS.

*Table 4. Network Ports for Firewalls and Port Mapping*

Service Name	Network Port
Quay HTTPS	8443/TCP
Quay HTTP	8080/TCP
Quay HTTP	80/TCP
Quay HTTPS	443/TCP
Postgresql Quay	5432/TCP
Postgresql ClairV4	5433/TCP
Redis for Quay	6379
ClairV4 HTTP	8081/TCP

### *Local Quay Instance Details*

- **Server FQDN:** quay.local

- **Images Used:**

- registry.redhat.io/rhel8/postgresql-10:1
- registry.redhat.io/rhel8/redis-5:1
- registry.redhat.io/quay/quay-rhel8:v3.6.0-62
- registry.redhat.io/quay/clair-rhel8:v3.6.0-70

*Table 5. Container Port Mapping*

Container Name	Mapped Port
clairv4	8001:8080
postgres-clairv4	5433:5432
redis	6379:6379
postgres	5432:5432
quay ( <i>Config Container Only</i> )	8080:8080
quay ( <i>Running Container</i> )	80:8080

*Table 6. Storage Mount Points*

Container Name	Mount Point on Host System	Purpose
quay	/quay	Storage for configuration files and images. Multiple sub-directories under this local directory mounted to Quay, Postresql, Redis for persistent storage.
clairv4	/etc/clairv4 ⇒ /clair	Storage for configuration files and images.

## 7.1. Installing the Quay Image Registry Manually

The Quay registry can be installed as a PoC locally. There is a manual process documented at [https://access.redhat.com/documentation/en-us/red\\_hat\\_quay/3/html/deploy\\_red\\_hat\\_quay\\_for\\_proof-of-concept\\_non-production\\_purposes/index](https://access.redhat.com/documentation/en-us/red_hat_quay/3/html/deploy_red_hat_quay_for_proof-of-concept_non-production_purposes/index) and there is an automated method with Ansible and playbooks created for an Ansible Workshop at [https://github.com/tmichett/quay\\_lab\\_poc](https://github.com/tmichett/quay_lab_poc).

### Quay Server Requirements

- RHEL8 Server with Podman Installed
- Access to **registry.redhat.io**
- Firewall ports opened
- **/etc/hosts** or DNS entry for Quay server
- Postgresql and Redis containers for Quay
- Postresql container for Clair (if using scanning)

For this workshop, the setup to prepare the environment in Chapter 1 provided the changes for the network needed to be able to successfully install Quay and register the FQDN as **quay.local** on both the **workstation** and **server** systems.

#### 7.1.1. Preparing the Server for Quay and Running Supporting Containers

The server needs prepared to run Quay as a service. Firewall ports will need to be opened and the infrastructure containers and components need setup.

### Example 26. LAB: Preparing Infrastructure

1. SSH to **server** as the root user

```
[student@workstation ~]$ ssh root@server
```

2. Configure Firewall Ports

```
[root@server ~]# firewall-cmd --add-port=80/tcp --add-port=8080/tcp --add-port=443/tcp --add-port=8443/tcp --permanent ; firewall-cmd --reload
success
success
```

If you completed and created the **quay\_lab.xml** custom firewall service, you can use the following command.



```
[root@server ~]# firewall-cmd --add-service=quay_lab --permanent ;
firewall-cmd --reload
success
success
```

3. Create Mountpoint and directory and initialize the Quay variable

```
[root@server ~]# mkdir Quay ; export QUAY=/Quay
```

4. Create Postgresql Directory for Persistent Storage

*Listing 114. Creating Directory*

```
[root@server ~]# mkdir -p $QUAY/postgres-quay
```

*Listing 115. Setting up ACL*

```
[root@server ~]# setfacl -m u:26:-wx $QUAY/postgres-quay
```

**Example 27. LAB: Setting up Infrastructure Containers****1. Install container-tools Module**

```
[root@server ~]# yum module install container-tools
```

**2. Login to the container registry**

```
[root@server ~]# podman login registry.redhat.io
Username: tmichett@redhat.com
Password:
Login Succeeded!
```

**3. Launch the Postgresql container**

```
[root@server ~]# 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
Trying to pull registry.redhat.io/rhel8/postgresql-10:1...
... OUTPUT OMITTED ...
Writing manifest to image destination
Storing signatures
c21bf1231538b75d5510b4bd81732ad8f56fa61a98582b985124bd3d5de2ae04
```

*Copy/Paste version of Command*



```
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
```

**4. Modify the Postgresql database to ensure that pg\_trgm module has been installed.**

```
[root@server ~]# podman exec -it postgresql-quay \
> /bin/bash -c \
> 'echo "CREATE EXTENSION IF NOT EXISTS pg_trgm" \
> | psql -d quay -U postgres'
```

*Copy/Paste version of Command*



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

## 5. Setup the Redis Container

```
[root@server ~]# podman run -d --rm --name redis \
> -p 6379:6379 \
> -e REDIS_PASSWORD=strongpassword \
> registry.redhat.io/rhel8/redis-5:1
Trying to pull registry.redhat.io/rhel8/redis-5:1...
... OUTPUT OMITTED ...
```

Storing signatures  
c847b459ded23acdd30e2a6ec1acfdeb3dd79f1457f7f99082cedb0a40447d73

*Copy/Paste version of Command*



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

**Example 28. LAB: Launch the Quay Configuration Container****1. Run the Quay container in configuration mode**

```
[root@server ~]# podman run --rm -it --name quay_config \
> -p 8080:8080 \
> registry.redhat.io/quay/quay-rhel8:v3.6.0-62 config secret
Trying to pull registry.redhat.io/quay/quay-rhel8:v3.6.0-62...
...
... OUTPUT OMITTED ...

2021-10-26 17:54:08,319 INFO success: config-editor entered RUNNING state, process
has stayed up for > than 1 seconds (startsecs)
config-editor stdout | time="2021-10-26T17:54:07Z" level=warning msg="An error
occurred loading TLS: No public key provided for HTTPS. Server falling back to
HTTP."
config-editor stdout | time="2021-10-26T17:54:07Z" level=info msg="Running the
configuration editor with HTTP on port 8080 with username quayconfig"
```

*Copy/Paste version of Command*



```
podman run --rm -it --name quay_config \
-p 8080:8080 \
registry.redhat.io/quay/quay-rhel8:v3.6.0-62 config secret
```

**2. Access a Web Browser and begin the Quay Configuration**

- URL: **quay.local:8080**
- UN: **quayconfig**
- PW: **secret**

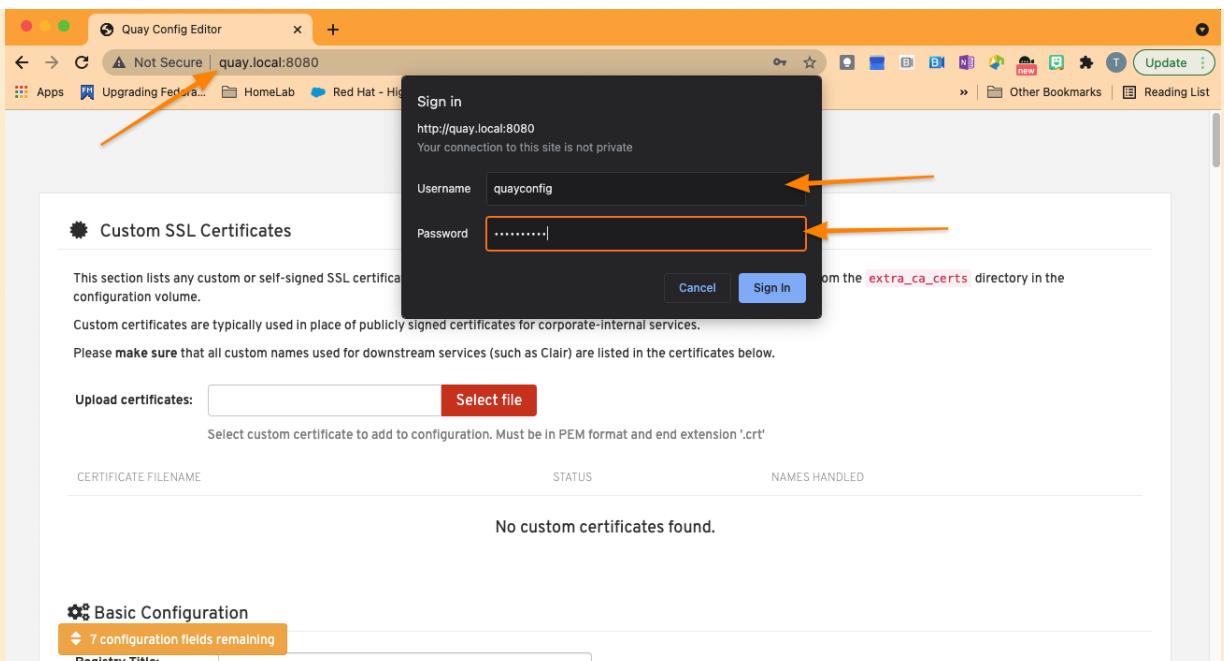


Figure 52. Quay Configuration Login

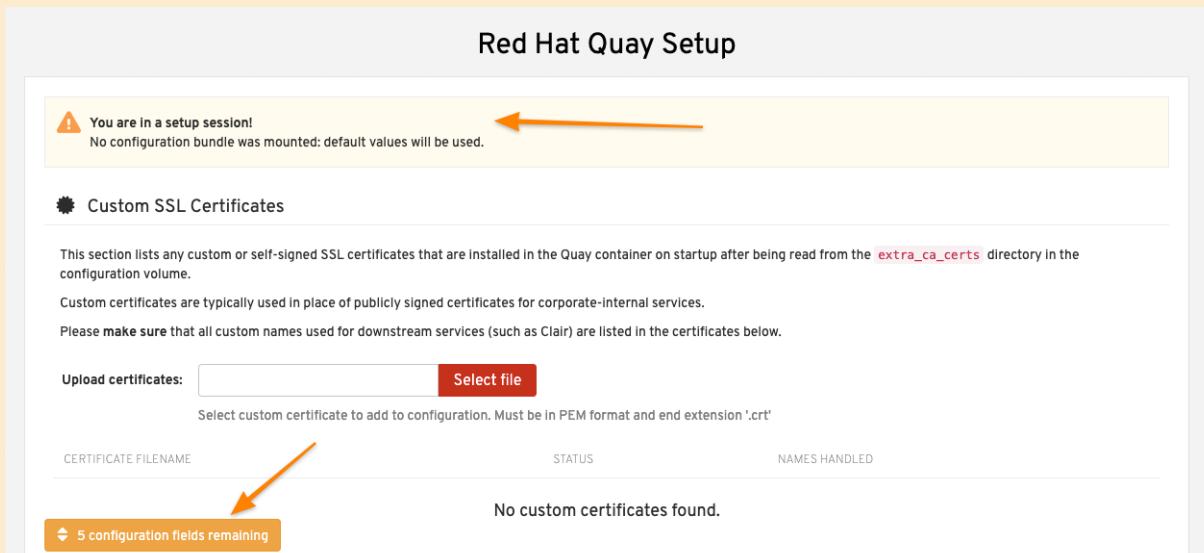


Figure 53. Quay Configuration Site

### 3. Setup the Server Configuration

#### a. Server Hostname: **quay.local**



Figure 54. Quay Server Hostname

#### 4. Setup Postgresql Database

- Postgresql Database Server: **quay.local**
- Postgresql User: **quayuser**
- Postgresql Password: **quaypass**
- Postgresql Database: **quay**
- Postgresql Admin Password: **adminpass**

The screenshot shows the 'Database' setup page for Quay. It includes the following fields with orange arrows indicating the values from the list above:

- Database Type:** Postgres
- Database Server:** quay.local
- Username:** quayuser
- Password:** (redacted)
- Database Name:** quay
- SSL Certificate:** No file chosen

Below the form, a note states: "Quay uses a database as its primary metadata storage."

Figure 55. Quay Postgresql DB Setup

#### 5. Setup Redis Information

- Redis Hostname: **quay.local**
- Redis Port: **6379**
- Redis Password: **strongpassword**

The screenshot shows the 'Redis' setup page for Quay. It includes the following fields with orange arrows indicating the values from the list above:

- Redis Hostname:** quay.local
- Redis port:** 6379
- Redis password:** (redacted)

Below the form, a note states: "A redis key-value store is required for real-time events and build logs."

Figure 56. Quay Redis Setup



At this point, there is enough configuration to validate and download the Quay config file for launching and running Quay. We will continue to setup **Repository Mirroring**, **Clair Image Scanning**, and **Admin Accounts**.

## Setting up Repository Mirroring

### 1. Enable Repository Mirroring

Figure 57. Quay Mirroring Setup



Remember to disable SSL as SSL isn't being used for this setup.

## Setting up Security Scanning

### 1. Select Enable Security Scanning

- Endpoint: <http://quay.local:8081>
- Select **Generate PSK** (Make note of this as it will need to be used)
- We will use the PSK of: **MTU5YzA4Y2ZkNzJoMQ==**

Figure 58. Quay Clair Image Scanning Setup



You **MUST** use the Pre-Shared-Key (PSK) of **MTU5YzA4Y2ZkNzJoMQ==** as we will be leveraging a pre-build CLAIR config.yml file.

## Configuring Super Users (Admins)

## 1. Add superusers to the system

- a. UN: **quayadmin**
- b. UN: **<your\_username>**

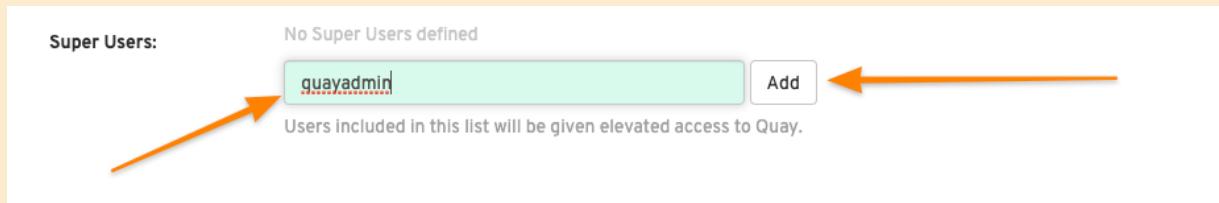


Figure 59. Quay Superuser Setup

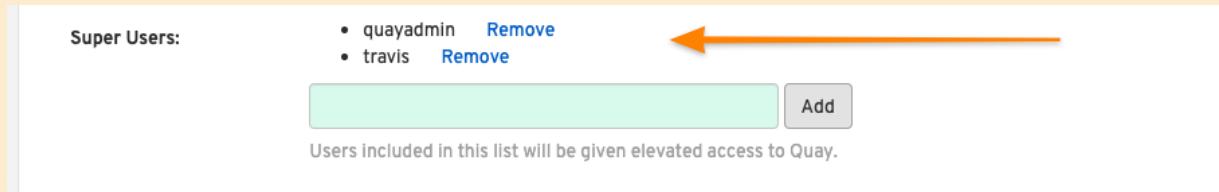


Figure 60. Quay Superuser Setup - Validation

## Completing Quay Configuration - Validating and Download Configuration File

### 1. Validate Configuration Changes

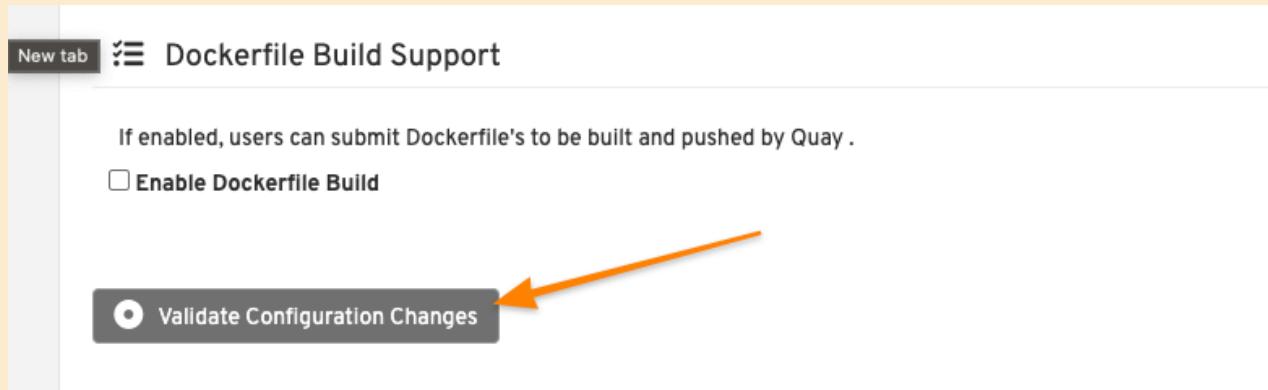


Figure 61. Quay Setup - Validation

### 2. Download Quay Configuration File



Figure 62. Quay Configuration - File Download

### 3. Stop and Remove the Quay\_Config Container

*Listing 116. Killing the Container from server by hitting CTRL-C*

```
... OUTPUT OMITTED ...

Engine"
config-editor stdout | time="2021-10-26T18:30:52Z" level=debug msg="Validating
ElasticSearch"
config-editor stdout | time="2021-10-26T18:30:52Z" level=debug msg="Validating
ActionLogArchiving"
config-editor stdout | time="2021-10-26T18:30:52Z" level=debug msg="Validating
Email"
config-editor stdout | time="2021-10-26T18:30:52Z" level=debug msg="Validating
OIDC"

^C

2021-10-26 18:36:55,589 WARN received SIGINT indicating exit request
2021-10-26 18:36:56,591 INFO waiting for stdout, config-editor to die
2021-10-26 18:36:57,596 INFO stopped: config-editor (terminated by SIGTERM)
2021-10-26 18:36:59,599 INFO waiting for stdout to die
2021-10-26 18:37:01,603 INFO stopped: stdout (terminated by SIGTERM")
```

#### Quay Next Steps



Based on the options in the Quay deployment configuration, it is necessary to setup and configure the Clair Image scanning services before starting the Quay container.

## 7.1.2. Preparing Quay and Clair Containers

It is necessary to create a storage location for Quay container images and images that will be loaded into the registry. It is also necessary to copy the Quay configuration file and Clair Configuration files

and label them appropriately for use. The Clair scanner will use and leverage Postgresql, so it will also be necessary to setup and configure a Postgresql container for Clair.

#### Example 29. LAB: Prepare Quay Storage

1. Create Directory for Quay Config

```
[root@server ~]# mkdir $QUAY/config
```

2. Copy download configuration file to \$QUAY/config

```
[root@server ~]# [root@server ~]# scp  
student@workstation:~/github/OCP_Demos/Containers/labs/Quay/quay-config.tar.gz  
$QUAY/config
```



We are coping the **quay-config.tar.gz** from the project to ensure that the correct parameters are used. This should have been the same exact configuration archive that was created and downloaded on the Workstation system in the **Downloads** directory.

3. Extract the files for use

```
[root@server ~]# cd $QUAY/config ; tar xvf quay-config.tar.gz  
extra_ca_certs/  
config.yaml
```

4. Verify Files and Return to **Home** directory

```
[root@server config]# ls ; cd  
config.yaml extra_ca_certs quay-config.tar.gz
```

5. Create and Prepare Local storage

```
[root@server ~]# mkdir $QUAY/storage ; setfacl -m u:1001:-wx $QUAY/storage
```

### Example 30. LAB: Prepare and Launch Clair

#### 1. Create the Clair Postgres Storage

```
[root@server ~]# mkdir -p $QUAY/postgres-clairv4 ; setfacl -m u:26:-wx $QUAY/postgres-clairv4
```

#### 2. Launch Postgresql Container

```
[root@server ~]# 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
e0db014a5a2f35cc895ef4ad136695a56ae2cda81c86db804810082ad38b9529
```

*Copy/Pasteable Command*



```
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
```

#### 3. Install Postgres uuid-ossp module

```
[root@server ~]# podman exec -it postgresql-clairv4 \
> /bin/bash -c \
> 'echo "CREATE EXTENSION IF NOT EXISTS \"uuid-ossp\"" \
> | psql -d clair -U postgres'
```

*Copy/Pasteable Command*

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

4. Create Clair Configuration File and place at [/etc/clairv4/config/config.yml](#)
  - a. <https://github.com/quay/clair/blob/main/Documentation/reference/config.md>
  - b. Use the one already created from [https://github.com/tmichett/quay\\_lab\\_poc](https://github.com/tmichett/quay_lab_poc)

*Listing 117. Make the /etc/clairv4/config Directory*

```
[root@server ~]# mkdir -p /etc/clairv4/config/
```

*Listing 118. Copying Config File from Workstation*

```
[root@server ~]# scp
student@workstation:~/github/quay_lab_poc/files/clair_config/config.yaml
/etc/clairv4/config/
student@workstation's password':
config.yaml
00:00
100% 845 559.3KB/s
```



The PSK from earlier is used in this file. Please ensure you use the same PSK.

5. Deploy the Clair Container

```
[root@server ~]# 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:v3.6.0-70
Trying to pull registry.redhat.io/quay/clair-rhel8:v3.6.0-70...
... OUTPUT OMITTED ...
```

Storing signatures

7f5ec8bd697c83fc2ad9277fd76d4c8c8b48048ec45d59f948cd58f87e03295e

*Copy/Pasteable Command*

```
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:v3.6.0-70
```

#### 7.1.2.1. Starting Quay and the Mirroring Containers

After all configurations have been made and the supporting containers are running, it is possible to launch the Quay and Quay Mirroring containers.

**Example 31. LAB: Starting Quay and the Mirroring Containers****1. Launch the Quay Container**

```
[root@server ~]# podman run -d --rm -p 80:8080 \
> --name=quay \
> -v $QUAY/config:/conf/stack:Z \
> -v $QUAY/storage:/datastorage:Z \
> registry.redhat.io/quay/quay-rhel8:v3.6.0-62
0357e5cf31fa0336cdc0561bd2d39e332451ba781671abd5b6f2fac2b24c7f0
```

*Copy/Pasteable Command*

```
podman run -d --rm -p 80:8080 \
--name=quay \
-v $QUAY/config:/conf/stack:Z \
-v $QUAY/storage:/datastorage:Z \
registry.redhat.io/quay/quay-rhel8:v3.6.0-62
```

**2. Launch the Quay Mirroring Container**

```
[root@server ~]# podman run -d --name mirroring-worker \
> -v $QUAY/config:/conf/stack:Z \
> registry.redhat.io/quay/quay-rhel8:v3.6.0-62 repomirror
adcd500ae3e0ca7510134d91b4389812cc55ebb2f1f2ebefcfc318f7ce2f6fd9
```

*Copy/Pasteable Command*

```
podman run -d --name mirroring-worker \
-v $QUAY/config:/conf/stack:Z \
registry.redhat.io/quay/quay-rhel8:v3.6.0-62 repomirror
```

**3. Verify all containers are running**

```
[root@server ~]# podman ps
CONTAINER ID  IMAGE                                COMMAND
CREATED      STATUS      PORTS
NAMES
c21bf1231538  registry.redhat.io/rhel8/postgresql-10:1    run-postgresql  About
an hour ago   Up About an hour ago  0.0.0.0:5432->5432/tcp
postgresql-quay
c847b459ded2  registry.redhat.io/rhel8/redis-5:1        run-redis      About
an hour ago   Up About an hour ago  0.0.0.0:6379->6379/tcp
redis
e0db014a5a2f  registry.redhat.io/rhel8/postgresql-10:1    run-postgresql  20
minutes ago   Up 20 minutes ago   0.0.0.0:5433->5432/tcp
postgresql-clairv4
7f5ec8bd697c  registry.redhat.io/quay/clair-rhel8:v3.6.0-70  6
minutes ago   Up 6 minutes ago   0.0.0.0:8081->8081/tcp, 0.0.0.0:8089-
>8089/tcp clairv4
0357e5cf31f   registry.redhat.io/quay/quay-rhel8:v3.6.0-62  registry      3
minutes ago   Up 3 minutes ago   0.0.0.0:80->8080/tcp
quay
444cf7561b02  registry.redhat.io/quay/quay-rhel8:v3.6.0-62  repomirror    3
seconds ago   Up 3 seconds ago  mirroring-worker
```

#### *Quay Setup Completed*



At this point the Quay setup has been completed with the Mirroring Worker and Clair v4 Image Scanner. You can proceed to the next steps and skip the section called: **Installing the Quay Image Registry with Ansible**. You should proceed to the section titled **Setting up the Quay Web Console and Testing Quay**.

## 7.2. Installing the Quay Image Registry with Ansible

There is a Github project that can be used to install a Quay image registry locally which includes a Mirroring and ClairV4 scanning process. The Github repository should have been cloned as part of the exercise in Chapter 1. The Github repository is: [https://github.com/tmichett/quay\\_lab\\_poc](https://github.com/tmichett/quay_lab_poc) and can deploy a fully functional Quay image registry locally.

The Quay playbooks have been broken down so that they can be run to easily deploy Quay. There are options for deploying based on the configuration files being pre-populated as well as using the Quay container to create the TGZ configuration. There is also a special playbook created to bring already deployed containers back online.

### 7.2.1. Deploying Quay with Ansible

Alternatively, there has been a project created with several playbooks allowing Quay to be deployed in the PoC method via Ansible playbooks.

### Example 32. LAB: Deploying Quay with Ansible

1. Change to the **github/quay\_lab\_poc** directory

```
[student@workstation ~]$ cd github/quay_lab_poc/
```

2. Create a registry credential file and update the file with your information

*Listing 119. Copy `registry_login.yml_example` to `registry_login.yml`*

```
[student@workstation quay_lab_poc]$ cp registry_login.yml_example  
registry_login.yml
```

*Listing 120. Edit the file `registry_login.yml`*

```
[student@workstation quay_lab_poc]$ vim registry_login.yml  
registry_un: UN_Goes_Here ①  
registry_pass: Password_Goes_Here ②  
registry_url: registry.redhat.io
```

① Replace with your Red Hat Login ID

② Replace with your Red Hat Login Password

3. Run the **Quay\_Prepares.yml** playbook

```
[student@workstation quay_lab_poc]$ ansible-playbook Quay_Prepare.yml

PLAY [Installation of Packages and Preparing the System]
*****
*****
```

TASK [Gathering Facts]
\*\*\*\*\*
\*\*\*\*\*

```
ok: [quay.local]
```

... OUTPUT OMITTED ...

TASK [Start the Quay Config Container]
\*\*\*\*\*
\*\*\*\*\*

```
changed: [quay.local]
```

TASK [Open the Quay Config Container Site]
\*\*\*\*\*
\*\*\*\*\*

```
[Open the Quay Config Container Site] ①
Go to the Quay Configuration container website and enter the appropriate
configuration values. For help, look at
https://github.com/tmichett/quay\_lab/References/Quay-3.5\_Deployment.pdf. Login
with the credentials provided which are UN: quayconfig and PW: secret. Press
'Enter' when you've completed the configuration and downloaded the file. The file
should be placed in the files directory for this playbook.: ②
```

... OUTPUT OMITTED ...'

PLAY RECAP
\*\*\*\*\*
\*\*\*\*\*

```
quay.local : ok=13    changed=10    unreachable=0    failed=0
skipped=0   rescued=0   ignored=0
```

① This allows you to open the Quay Config Site to create a custom configuration file.

② Once you've opened the Quay config site and completed the configuration as well as downloaded the file you can hit "Enter". It is also possible to hit "Enter" and skip this step so that the already existing configuration file can be used.

**Ansible Failure Possible**

It is possible that you will receive an Ansible failure message like this



```
[student@workstation quay_lab_poc]$ ansible-playbook  
Quay_Prepare.yml  
ERROR! vars file registry_login.yml was not found ①  
Could not find file on the Ansible Controller.  
If you are using a module and expect the file to exist on the  
remote, see the remote_src option
```

① This error means you forgot to create/edit the **registry\_login.yml** file.

4. Run the configuration file deployment based on using the TAR config file or the file-based configuration method. **NOTE: You can only choose one method for configuration.**

***Listing 121. File-Based Configuration Method***

```
[student@workstation quay_lab_poc]$ ansible-playbook Quay_Config_Deploy_Files.yml
```

***File-Based Configuration Considerations***

The **./files/config/config.yaml** file will be used and deployed to control the configuration of the Quay environment.

***Listing 122. TAR File Configuration Method***

```
[student@workstation quay_lab_poc]$ ansible-playbook Quay_Config_Deploy_Tar.yml
```

***TAR-Based Configuration Considerations***

The **./files/quay-config.tar.gz** file will be used and deployed to control the configuration of the Quay environment. This file MUST have been created as part of the Quay configuration container process with the WebUI and it must be placed in the **./files/quay-config.tar.gz** before running the playbook.

5. Deploy the ClairV4 scanning image by executing the **Quay\_Clair\_Deploy.yml** playbook

```
[student@workstation quay_lab_poc]$ ansible-playbook Quay_Clair_Deploy.yml
```

6. Deploy the Quay container registry by executing the **Quay\_Deploy.yml** playbook.

```
[student@workstation quay_lab_poc]$ ansible-playbook Quay_Deploy.yml
```

7. Deploy the Quay Mirroring Container by executing the **Quay\_Mirror\_Deploy.yml** playbook.

```
[student@workstation quay_lab_poc]$ ansible-playbook Quay_Mirror_Deploy.yml
```

## 7.3. Setting up the Quay Web Console and Testing Quay

After all Quay containers have been configured and installed, it is necessary to setup the Admin (Superuser) for Quay as well as test out the system for both image scanning and the ability to mirror container images from upstream repositories.

### 7.3.1. Configuring the Quay Super User

After the Quay registry has been deployed, it is important to finish configuring the super users (admins) that were defined as part of the setup and configuration file (**config.yaml**) that was created during the Quay preparation section.

It is necessary to look at the **config.yaml** file and configure these users with a password and create the accounts officially before moving forward with utilizing the Quay container registry and the lab environment.

#### *Configure Quay Super Users*

It is possible to either look in the configuration file of the **quay-config.tar.gz** or the actual **config.yaml** file for the **SUPER\_USERS** section. This is where the usernames are defined that will function as Quay super users.



#### *Listing 123. Quay Super Users*

##### **SUPER\_USERS:**

- **quayadmin**
- **travis**

**Example 33. LAB: Configuring Quay and Super Users**

1. Open the Quay web console by navigating to it in your favorite browser using <http://Quay-FQDN>

- a. Quay-FQDN: quay.local

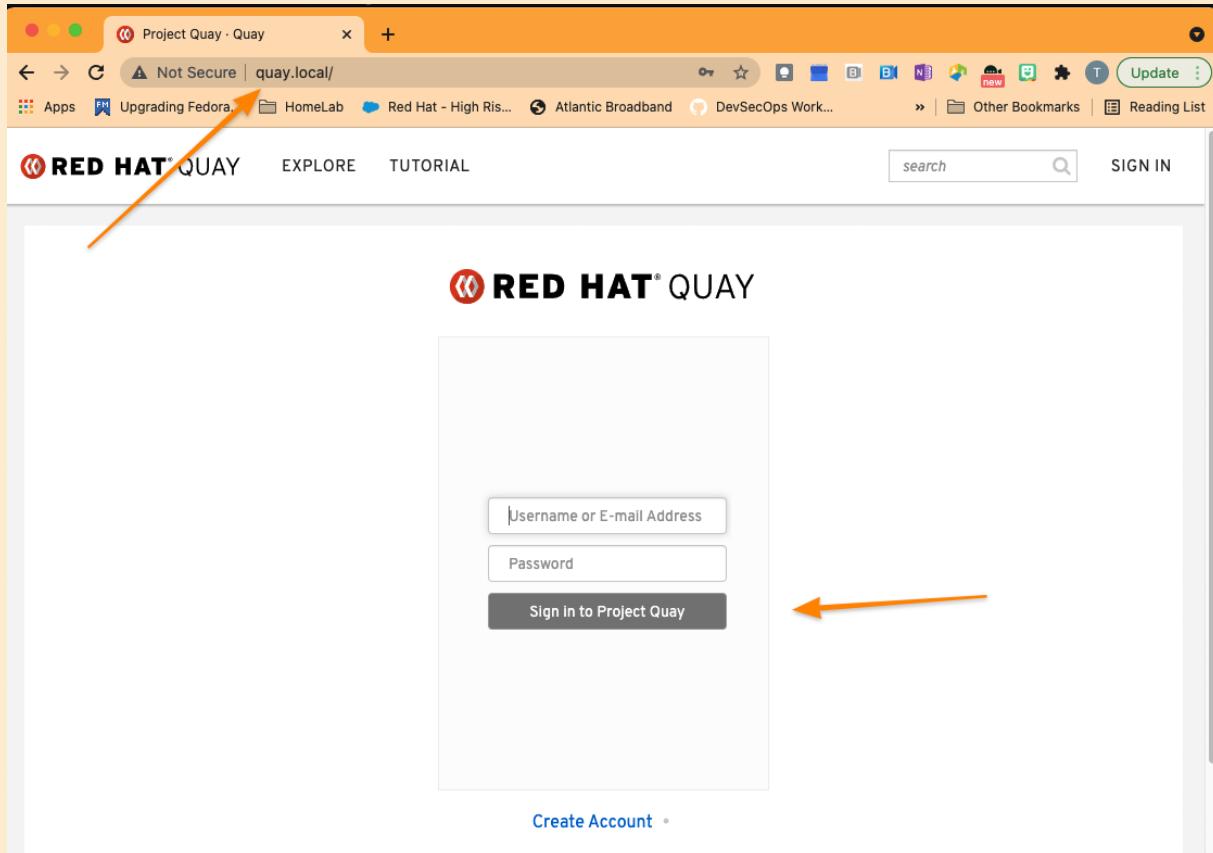


Figure 63. Quay Login Page

2. Click **Create Account** to create the administrator/superuser accounts for Quay as defined in the **config.yaml** file.
  - Repeat this step for all super users in the **config.yaml** file.

 Username or E-mail Address PasswordSign in to Project QuayA large orange arrow points from the bottom left towards the "Create Account" link.[Create Account](#) •

# Repositories



Create new account

Username:

quayadmin

E-mail address:

tmichett@redhat.com

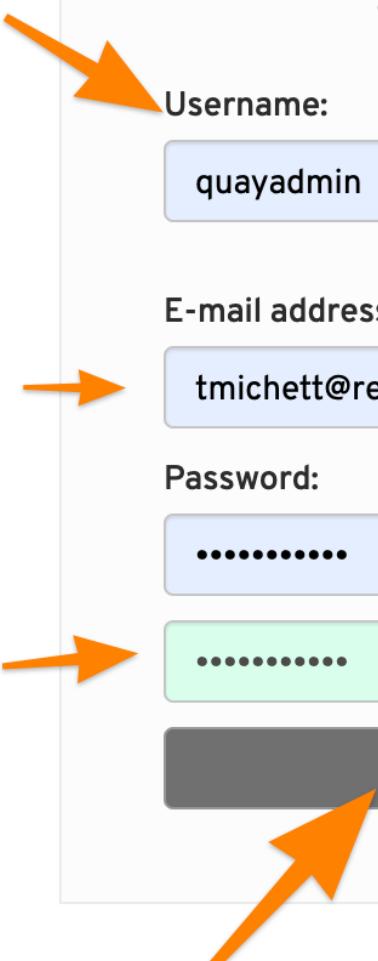
Password:

.....

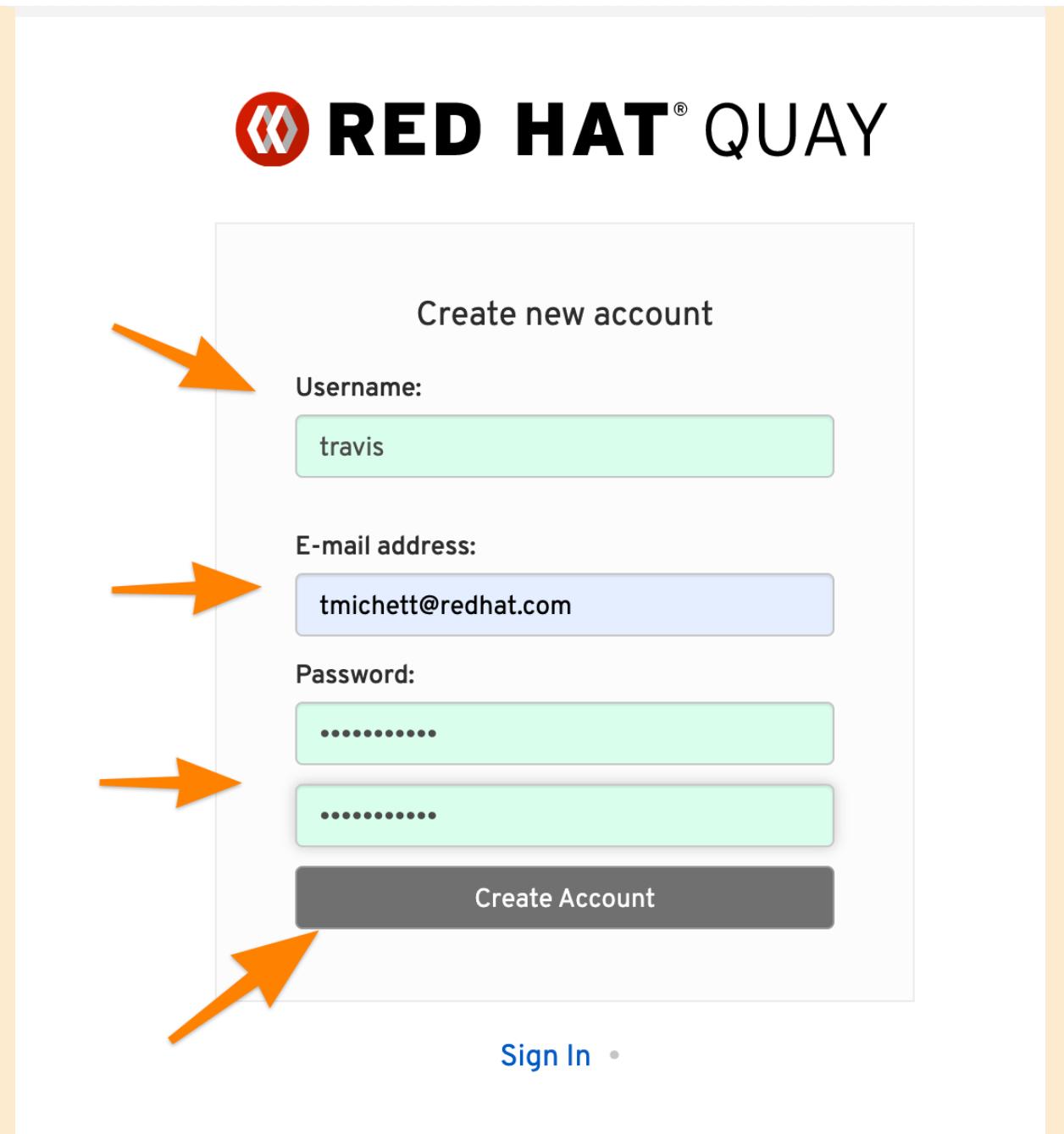
.....

Create Account

Sign In •



Three orange arrows point to the 'Username', 'E-mail address', and 'Create Account' button fields. The 'Username' field is light blue, the 'E-mail address' field is light blue, and the 'Create Account' button is dark grey.



3. Verify the account was setup properly and you have **Super User** rights by clicking your Username and looking for **Super User Admin Panel**.

The screenshot shows the 'Repositories' page of the Red Hat Quay web interface. The top navigation bar includes links for 'EXPLORE', 'REPOSITORIES', and 'TUTORIAL'. A search bar and a user dropdown menu ('travis') are also present. The main content area displays a message stating 'This namespace doesn't have any viewable repositories.' and 'Either no repositories exist yet or you may not have permission to view any. If you have permission, try [creating a new repository](#)'. On the right side, a vertical sidebar shows 'Account Settings', 'Super User Admin Panel' (which is highlighted with an orange arrow), 'Sign out all sessions', 'Users and Organizations', 'travis' (with an orange arrow pointing to it), and '+ Create New Organization'.

### 7.3.2. Testing Quay and ClairV4 Image Scanning

In order to test the scanning capabilities and ensure that things function properly, update a basic image into the Quay Repository

**Example 34. LAB: Testing Image Scanning**

1. Login to Local Quay Repository (test authentication and account)

*Listing 124. podman Authentication*

```
[root@quay ~]# podman login --tls-verify=false quay.local
Username: travis
Password:
Login Succeeded!
```

2. Pull and Download an Image, Tag it, then upload to repository

- a. Image to download: **quay.io/redhattraining/httpd-parent:2.4**
- b. Tag Name: **quay.local/travis/httpd-parent:2.4**

*Listing 125. Downloading image*

```
[root@server ~]# podman pull quay.io/redhattraining/httpd-parent:2.4
Trying to pull quay.io/redhattraining/httpd-parent:2.4...
Getting image source signatures
Copying blob a3ed95caeb02 done
Copying blob a3ed95caeb02 done
Copying blob a3ed95caeb02 done
Copying blob 787f47dbeaac done
Copying blob a3ed95caeb02 skipped: already exists
Copying blob a3ed95caeb02 skipped: already exists
Copying blob a3ed95caeb02 skipped: already exists
Copying blob 6a5240d60dc4 done
Copying blob 408208567b9a done
Copying blob 08b8c9fdec44 done
Writing manifest to image destination
Storing signatures
3639ce1374d3611e80ed66decd7d5467b72d010c21e19e4f193cd8b944e8c9f5
```

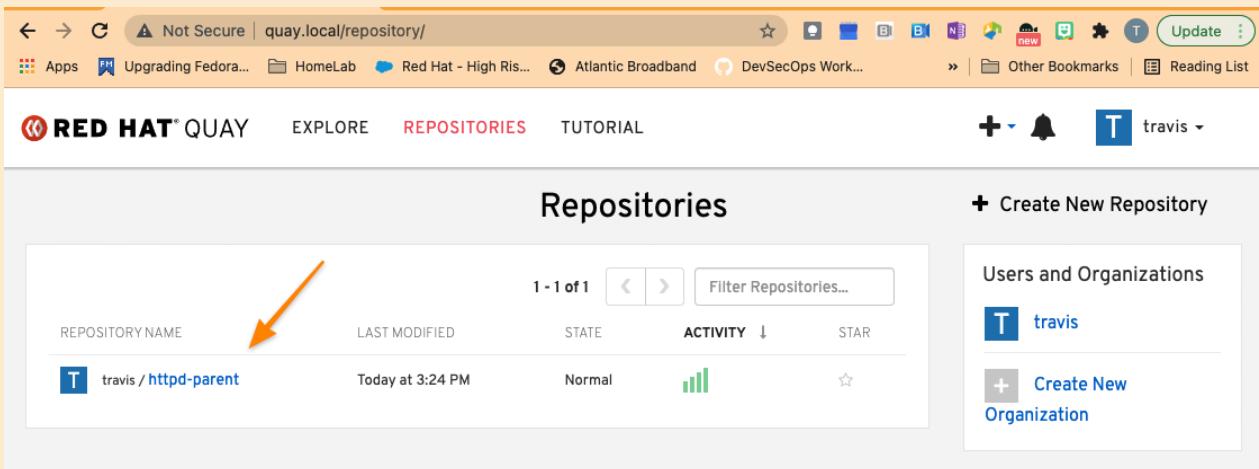
*Listing 126. Tagging image*

```
[root@server ~]# podman tag quay.io/redhattraining/httpd-parent:2.4
quay.local/travis/httpd-parent:2.4
```

*Listing 127. Push image*

```
[root@server ~]# podman push --tls-verify=false quay.local/travis/httpd-parent:2.4
Getting image source signatures
Copying blob c613b100be16 done
Copying blob a3ed95caeb02 done
Copying blob a3ed95caeb02 done
Copying blob a3ed95caeb02 done
Copying blob 574bcc187eda done
Copying blob 24d85c895b6b done
Copying blob a3ed95caeb02 done
Copying blob a3ed95caeb02 skipped: already exists
Copying blob a3ed95caeb02 skipped: already exists
Copying blob 7f9108fde4a1 done
Writing manifest to image destination
Storing signatures
```

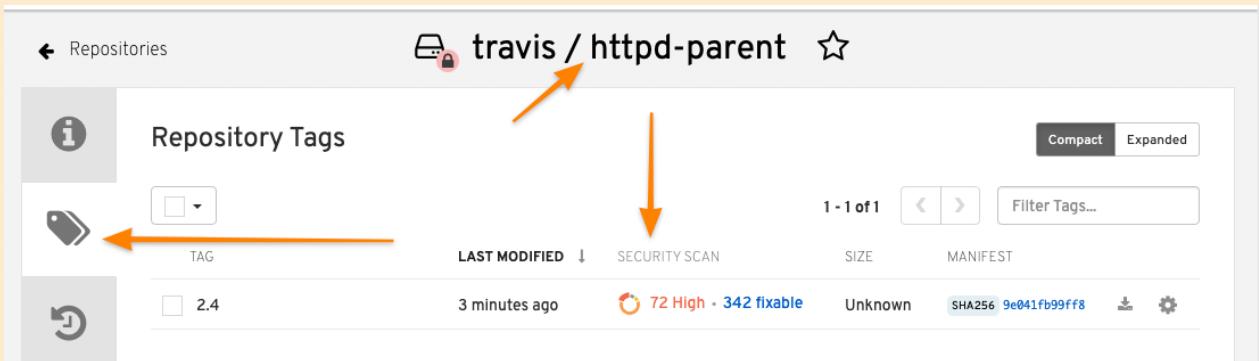
## 3. Verify image exists in Quay



The screenshot shows the Red Hat Quay web interface. At the top, there's a browser header with tabs like 'Not Secure | quay.local/repository/' and various system icons. Below the header, the Quay logo is on the left, followed by navigation links for 'EXPLORE', 'REPOSITORIES', and 'TUTORIAL'. On the right side of the header, there are buttons for '+', a bell icon, and a user profile for 'travis'. The main content area is titled 'Repositories' and shows a single entry: 'travis / httpd-parent'. This entry has a small blue icon with a white 'T', the repository name, 'LAST MODIFIED' (Today at 3:24 PM), 'ACTIVITY' (Normal), and a star icon. To the right of the repository list, there's a sidebar with a '+ Create New Repository' button, a 'Users and Organizations' section showing 'travis', and a 'Create New Organization' button.

Figure 64. HTTPD Parent Image

## 4. Navigate to image tags and see if the security scan has completed



The screenshot shows the 'Repository Tags' page for the 'travis / httpd-parent' repository. At the top, the repository name is displayed with a lock icon. An orange arrow points to this name. Below it, there's a table with columns for 'TAG', 'LAST MODIFIED', and 'SECURITY SCAN'. The table contains one row with a tag of '2.4', last modified '3 minutes ago', and a security scan result of '72 High + 342 fixable'. To the right of the table, there are buttons for 'Compact' and 'Expanded' views, and a 'Filter Tags...' search bar. The 'SECURITY SCAN' column shows a progress bar and the text '72 High + 342 fixable'.

Figure 65. HTTPD Parent Image - Image Tags

5. Click on Security scan to view the vulnerabilities

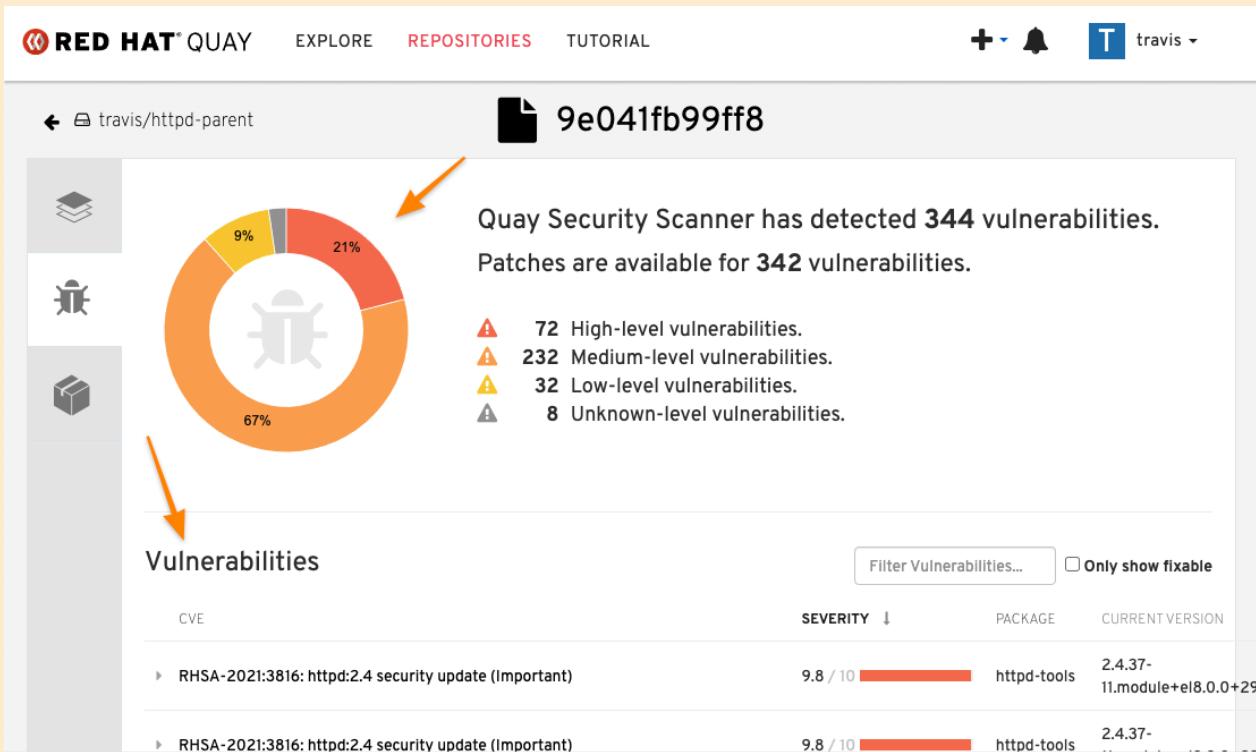


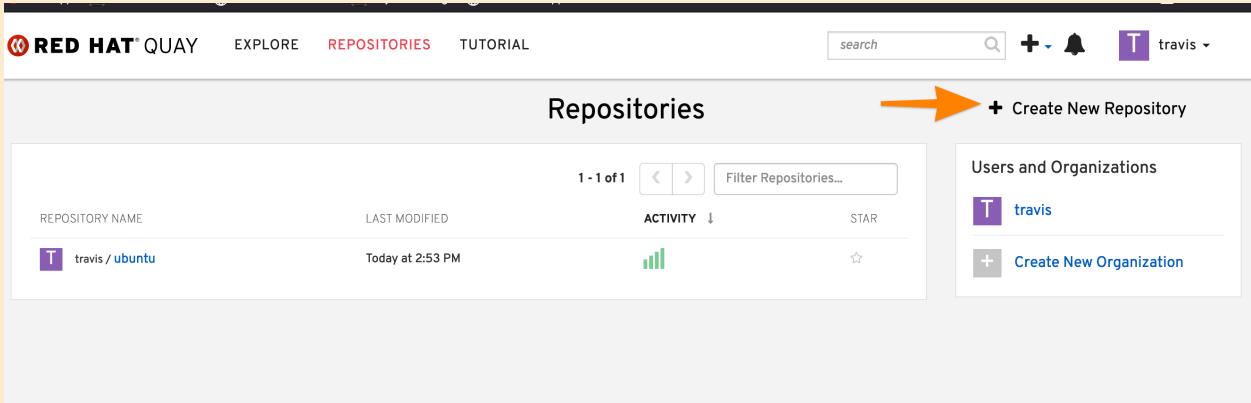
Figure 66. HTTPD Parent Image - Image Vulnerabilities

### 7.3.3. Testing Repository Mirroring

The next step is to ensure that the QUAY Image mirroring container is working and that you can successfully mirror container images from upstream repositories.

### Example 35. LAB: Testing Repository Mirroring

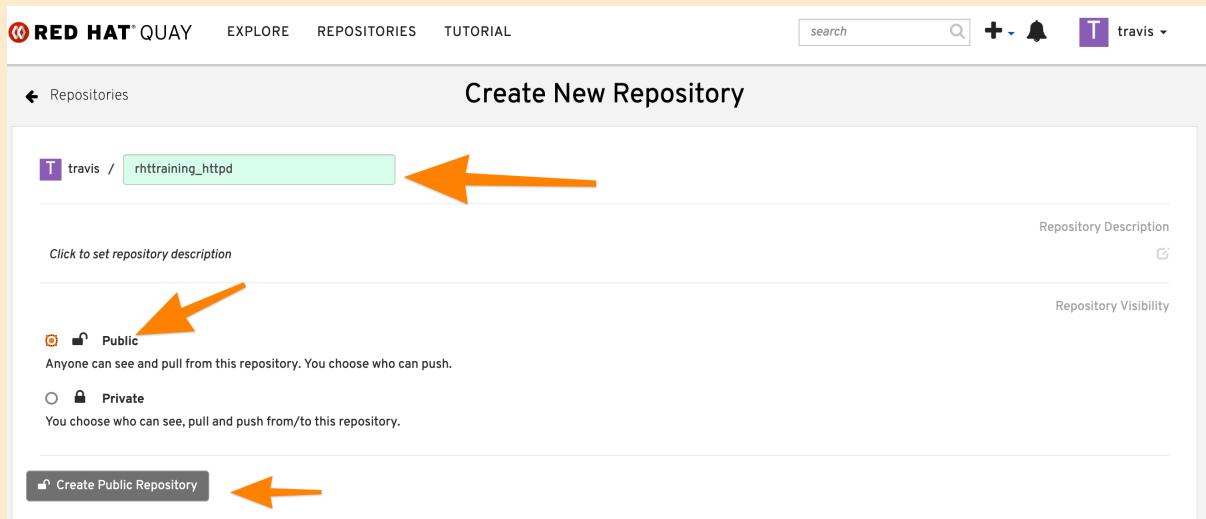
1. Create a new repository in Quay by clicking **Create New Repository**



The screenshot shows the Red Hat Quay web interface. At the top, there's a navigation bar with links for 'EXPLORE', 'REPOSITORIES', and 'TUTORIAL'. On the right side of the header, there's a search bar, a '+' icon, a bell icon, and a user profile for 'travis'. Below the header is a main content area titled 'Repositories'. It displays a single repository entry: 'travis / ubuntu'. To the right of the repository list is a sidebar titled 'Users and Organizations' which shows 'travis' and a link to 'Create New Organization'. An orange arrow points from the text in the previous step to the '+ Create New Repository' button located at the top right of the 'Repositories' section.

2. Give repository a name and setup the repository visibility

- a. Name: rhttraining\_httpd



The screenshot shows the 'Create New Repository' form. At the top, it says 'Create New Repository' and has a back arrow to 'Repositories'. In the center, there's a text input field containing 'travis / rhttraining\_httpd', with an orange arrow pointing to it. Below the input field is a note: 'Click to set repository description'. To the right of the input field are sections for 'Repository Description' and 'Repository Visibility'. Under 'Repository Visibility', there are two options: 'Public' (selected) and 'Private'. The 'Public' option is highlighted with an orange arrow. At the bottom left of the form, there's a button labeled 'Create Public Repository' with an orange arrow pointing to it.

3. In the newly created repository, click the **Settings** option from the left-side navigation menu. Set the **Repository State** to **Mirror**.

The screenshot shows the Quay Web Console interface. On the left, there is a vertical navigation menu with icons for User Permissions, Events and Notifications, Repository Visibility, and Repository State. An orange arrow points to the Repository State icon. The main content area shows a user named 'travis' with an 'Admin' role. It displays a message stating 'No notifications have been setup for this repository.' and a button to 'Create Notification'. Below that, it shows 'Repository Visibility' as 'public' and a 'Make Private' button. Under 'Repository State', it says 'Mirror' and provides a link to change it. The URL in the address bar is `https://quay.local:8080/repositories/travis/rhttraining_httpd`.

4. In the newly created repository, click the **Mirroring** option from the left-side navigation menu.

The screenshot shows a repository page for 'travis / rhttraining\_httpd'. The left sidebar has icons for Tag, Refresh, Description, and Settings. An orange arrow points to the 'Description' icon. The main content shows the repository name, a star icon, and a 'Repository Activity' section with a timeline for June, July, and August. A Docker pull command is shown: `docker pull quay.local:8080/travis/rhttraining_httpd`. The URL in the address bar is `https://quay.local:8080/repositories/travis/rhttraining_httpd`.

5. In the **Mirroring** tab, complete the required information for the repository and create a **Robot User**. Click **Enable Mirror**
  - a. **Registry Location:** `quay.io/redhattraining/httpd-parent`
  - b. **Tags:** latest and 2.4

 Create robot account X

Provide a name for your new robot account:

Choose a name to inform your teammates about this robot account. Must match ^[a-z][a-zA-Z0-9\_]{1,254}\$.

Provide an optional description for your new robot account:

Enter a description to provide extra information to your teammates about this robot account.

Create robot account Cancel

◀ Repositories

travis / rhttraining\_httpd ⭐

## Repository Mirroring

This feature will convert [travis/rhttraining\\_httpd](#) into a mirror. Changes to the external repository will be duplicated here. While enabled, users will be unable to push changes to this repository.

**External Repository**

Registry Location: quay.io/redhattraining/httpd-parent

Tags: latest, 2.4

Start Date: August 25, 2021 3:24 PM

Sync Interval: 5 days

Robot User: travis+travis\_robot

**Credentials**  
Required if the external repository is private.

Username: [REDACTED]

The screenshot shows the 'Repository Mirroring' configuration page for the 'travis / rhttraining\_httpd' repository. The 'External Repository' section is highlighted with a green background. Three orange arrows point from the right side of the screen towards the 'Registry Location', 'Tags', and 'Robot User' fields. A fourth orange arrow points upwards from the bottom towards the 'Credentials' section.

Credentials  
Required if the external repository is private.

Username: [REDACTED]  
Password: [REDACTED]

Advanced Settings

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

HTTP Proxy: proxy.example.com

HTTPs Proxy: proxy.example.com

No Proxy: example.com

**Enable Mirror**

6. Click "Sync Now" to perform immediate synchronization

Enabled  
Scheduled mirroring enabled.   
Immediate sync available via Sync Now.

External Repository: quay.io/redhattraining/httpd-parent >

Tags: latest 2.4 >

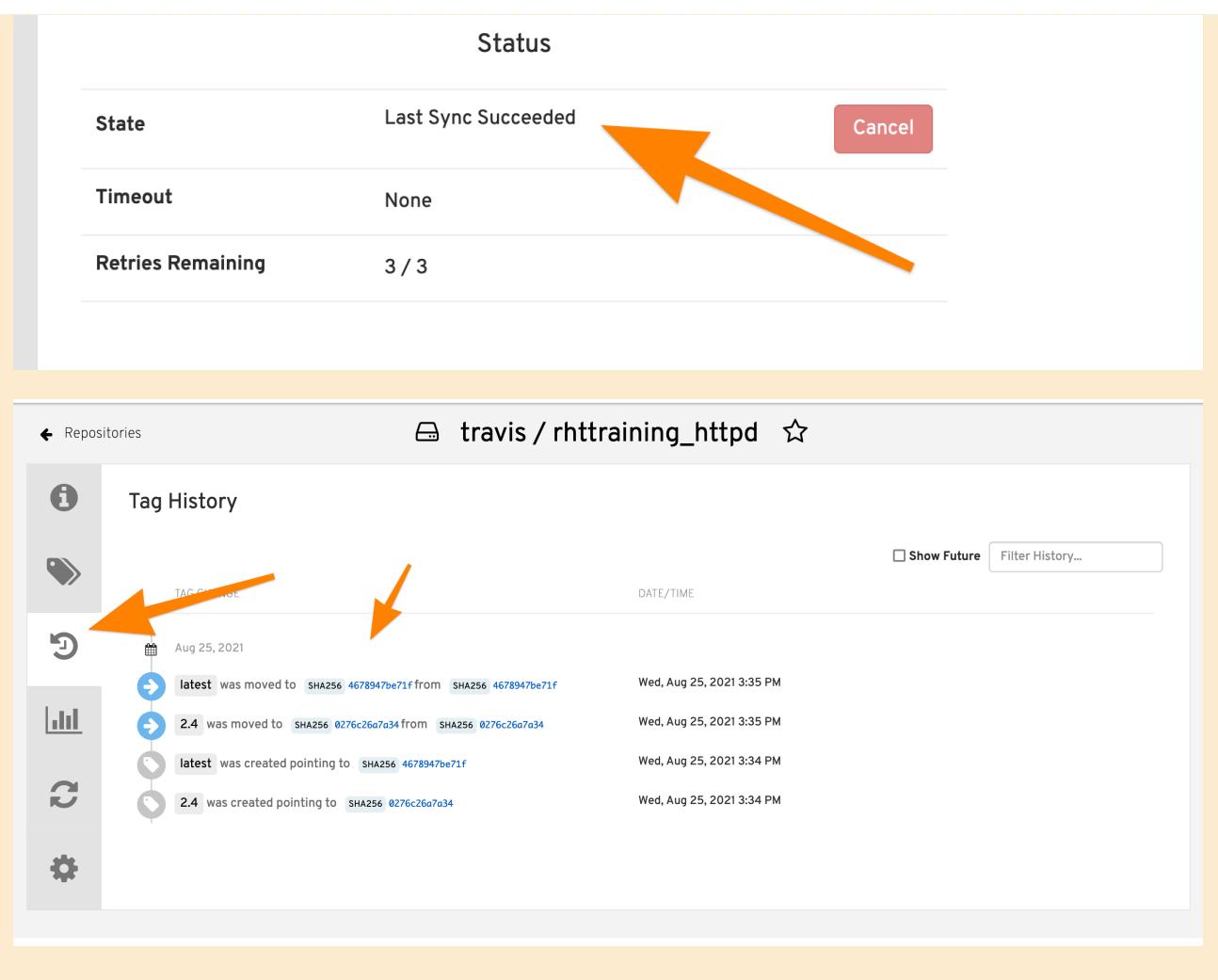
Sync Interval: 5 days >

Next Sync Date: Aug 25, 2021 3:24 PM > **Sync Now**

Robot User: travis+travis\_robot

Advanced Settings

7. Verify synchronization completed on the Mirroring tab as well as the Tag History



The screenshot shows the Quay Image Registry interface. At the top, there's a 'Status' section with three rows: 'State' (Last Sync Succeeded), 'Timeout' (None), and 'Retries Remaining' (3 / 3). A large orange arrow points from the 'Cancel' button in this section towards the 'Tag History' section below. Below the status, there's a navigation bar with 'Repositories' and a repository name 'travis / rhttraining\_httpd'. The main area is titled 'Tag History' and contains a sidebar with icons for info, tag, history, chart, refresh, and gear. The 'Tag' icon is highlighted with an orange arrow. The 'DATE/TIME' column header is also highlighted with an orange arrow. The tag history table has four columns: Action, Tag, SHA256, and Date/Time. The data is as follows:

Action	Tag	SHA256	Date/Time
TAG MOVE	latest	4678947be71f	Wed, Aug 25, 2021 3:35 PM
Moved to	2.4	0276c26a7a34	Wed, Aug 25, 2021 3:35 PM
Created pointing to	latest	4678947be71f	Wed, Aug 25, 2021 3:34 PM
Created pointing to	2.4	0276c26a7a34	Wed, Aug 25, 2021 3:34 PM

### 7.3.4. Using the Quay Image Registry

After testing the initial Quay image registry, we will need to use the registry similar to the one located at Quay.io. It is important to understand that since this repository/registry is local, it is fully maintained by the administrators that installed it. There were one or more **Superuser** accounts created as part of the installation process. These accounts can create organizations or modify permissions to organizations as well as maintain the infrastructure.

#### 7.3.4.1. Configuring Quay as the Superuser

**Example 36. LAB: Configuring QUAY****1. Login to the Quay site and open the Super User Admin Panel**

The screenshot shows the Red Hat Quay web interface. At the top, there's a navigation bar with links for 'EXPLORE', 'REPOSITORIES' (which is currently selected), and 'TUTORIAL'. A search bar and a user dropdown menu are also present. The main content area is titled 'Repositories' and displays a message: 'This namespace doesn't have any viewable repositories.' Below this, there's a link to 'creating a new repository'. On the right side, there's a sidebar with options like 'Account Settings', 'Super User Admin Panel' (which has an orange arrow pointing to it), 'Sign out all sessions', 'Users and Organizations', and 'Create New Organization'. The user 'travis' is logged in.

Figure 67. Quay Superuser Panel Access

The screenshot shows the 'Red Hat Quay Management' interface. The left sidebar has icons for 'Users', 'Organizations', 'Keys', 'Metrics', and 'Logs'. The main content area is titled 'Red Hat Quay Management' and shows a 'Users' section. It lists one user, 'travis', with a 'Create User' button. The user is identified as 'YOU SUPERUSER'. An orange arrow points from the left sidebar to the 'Users' icon, and another orange arrow points from the top right towards the 'Red Hat Quay Management' title.

Figure 68. Quay Superuser - User Management

2. Explore the various items on the left-hand menu.
3. Navigate to "Messages" and click **Create Message** to create a system-wide message

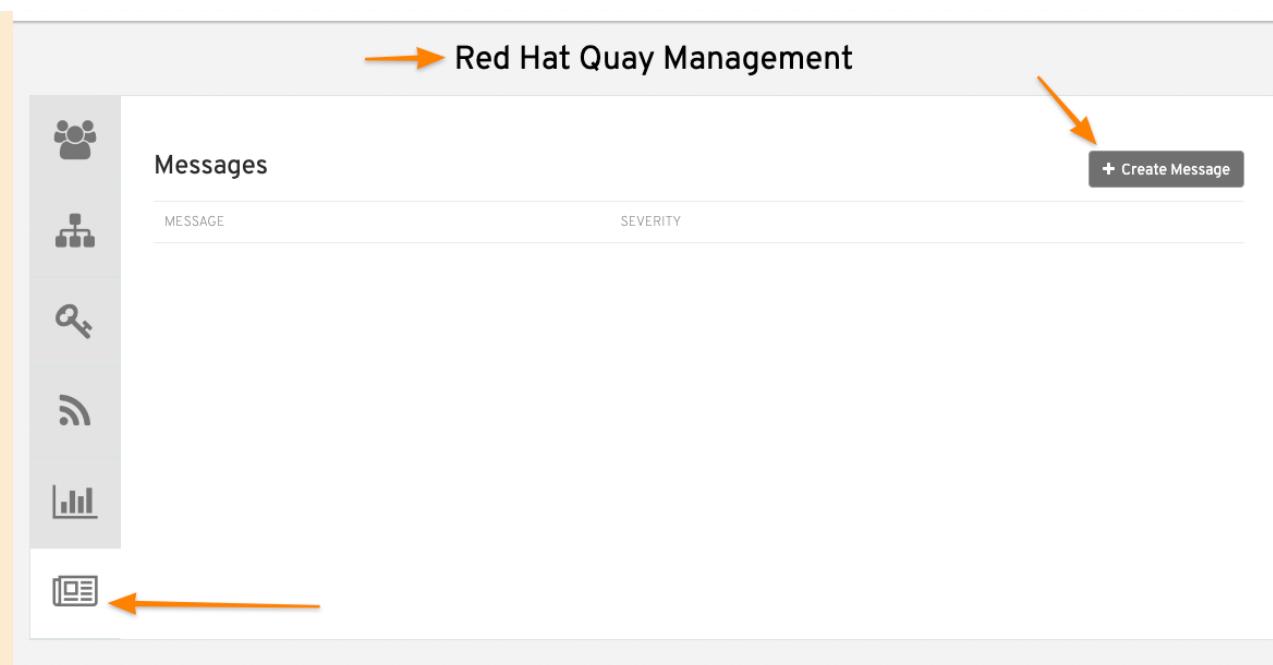


Figure 69. Quay Superuser - System Message

4. Create a system message

a. SYSTEM Message:

- i. **Attention** The system will be scheduled for maintenance on October 31, 2021. It will be in read-only mode for 24-hours. During this time, you can pull existing containers, but accounts and organizations cannot be created and new images will not be able to be published.

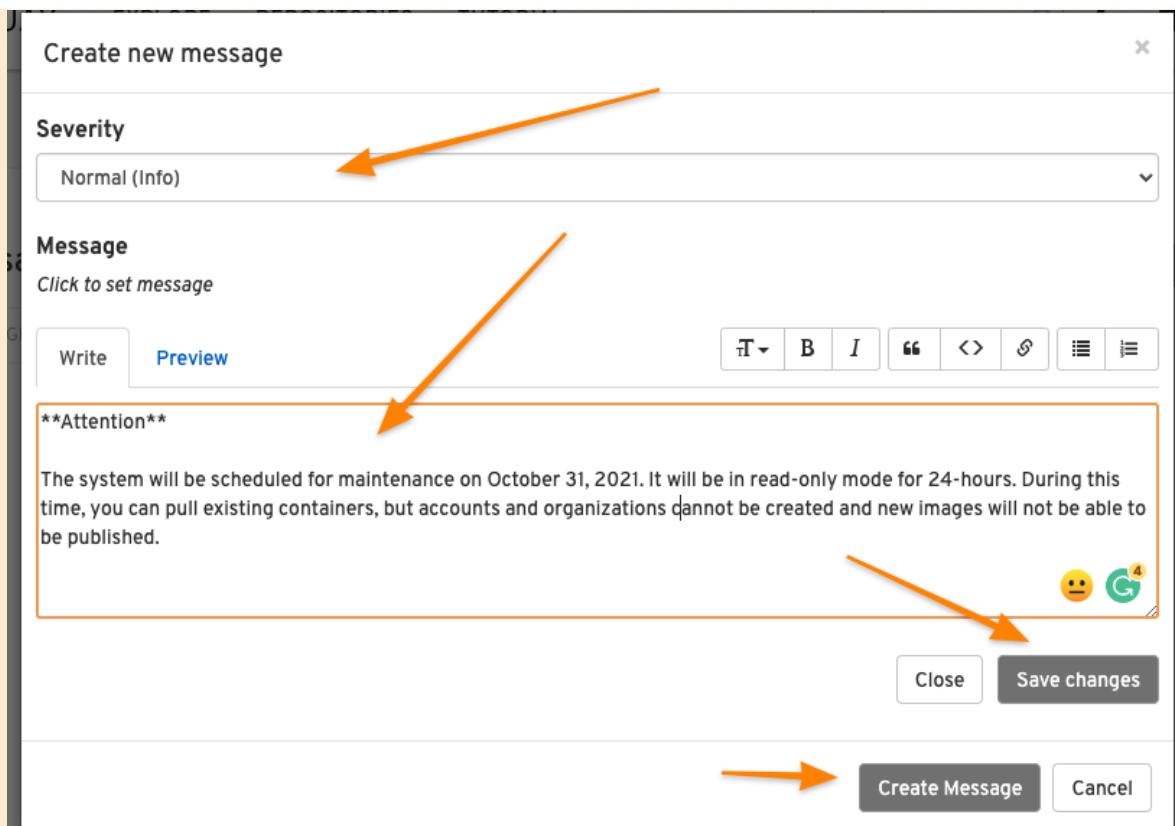


Figure 70. Quay Superuser - System Message Creation

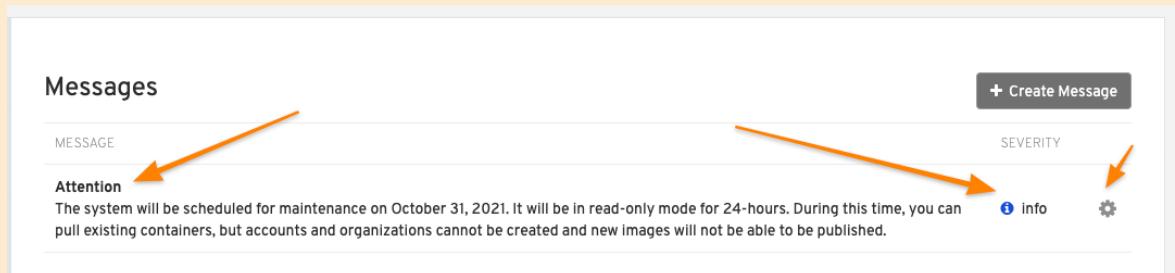


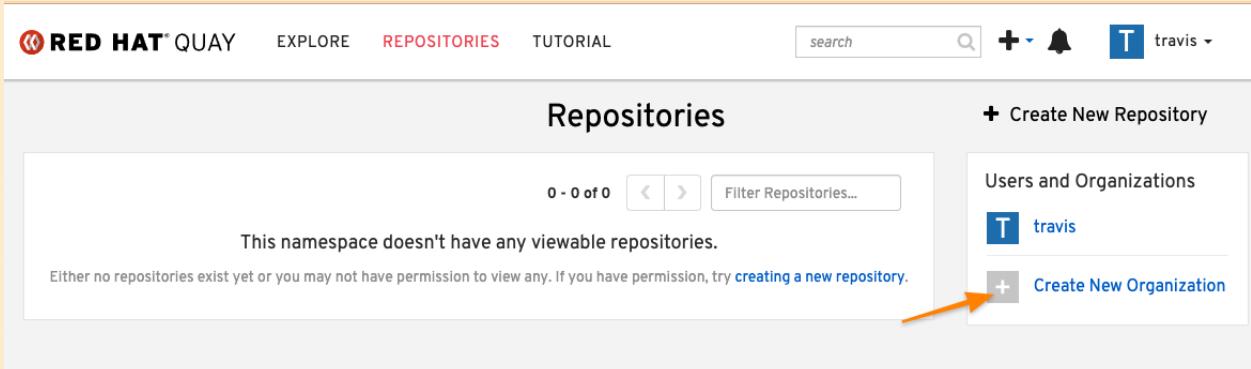
Figure 71. System Message Verification

#### 7.3.4.2. Creating Organizations

Organizations can be created so that multiple users can share container images. After creating an organization, users can be added to the organization so that they can manage images within that organization.

### Example 37. LAB: Configuring/Creating Organizations

#### 1. Click Create New Organization

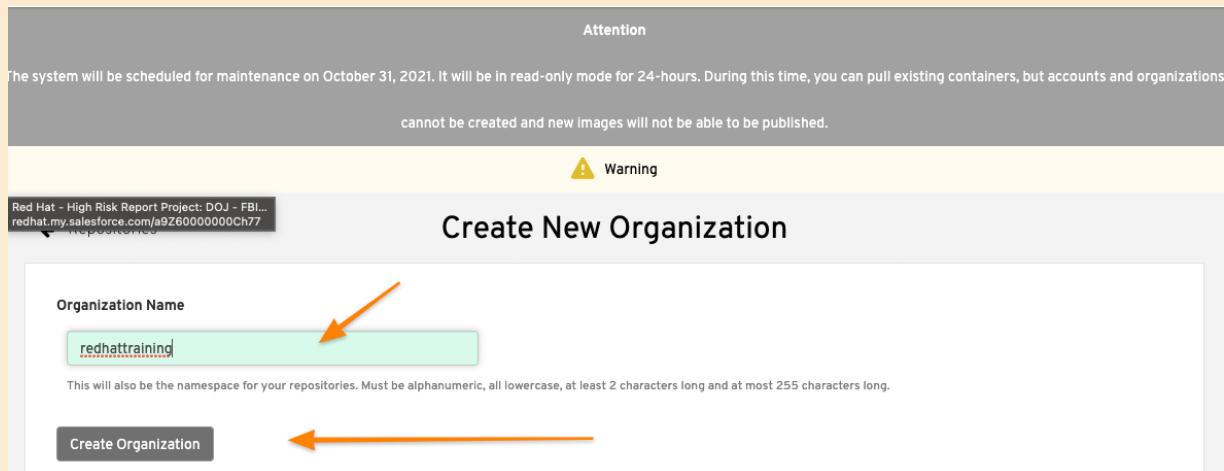


The screenshot shows the Red Hat Quay interface. At the top, there's a navigation bar with links for 'EXPLORE', 'REPOSITORIES', and 'TUTORIAL'. On the right side of the header, there's a search bar, a notification icon, and a user profile for 'travis'. Below the header, the main content area is titled 'Repositories' and displays a message: 'This namespace doesn't have any viewable repositories.' It also includes a link to 'creating a new repository.' To the right of the main content, there's a sidebar titled 'Users and Organizations' which lists 'travis'. At the bottom of this sidebar is a button labeled '+ Create New Organization'.

Figure 72. Creating an Organization

#### 2. Provide name for Organization and click Create Organization

##### a. Organization Name: redhattraining



The screenshot shows the 'Create New Organization' form. At the top, there's an 'Attention' message about system maintenance scheduled for October 31, 2021. Below that is a 'Warning' message stating that accounts and organizations cannot be created during maintenance. The main form has a title 'Create New Organization'. It contains a field labeled 'Organization Name' with the value 'redhattraining'. A note below the field says, 'This will also be the namespace for your repositories. Must be alphanumeric, all lowercase, at least 2 characters long and at most 255 characters long.' At the bottom left of the form is a 'Create Organization' button.

Figure 73. Red Hat Training Organization



When naming the organization, it is important to use all lowercase letters and not have any spaces. Additionally it should be noted that only newer registry versions support naming with "-" and "." in the name.

#### 3. Manage teams and membership

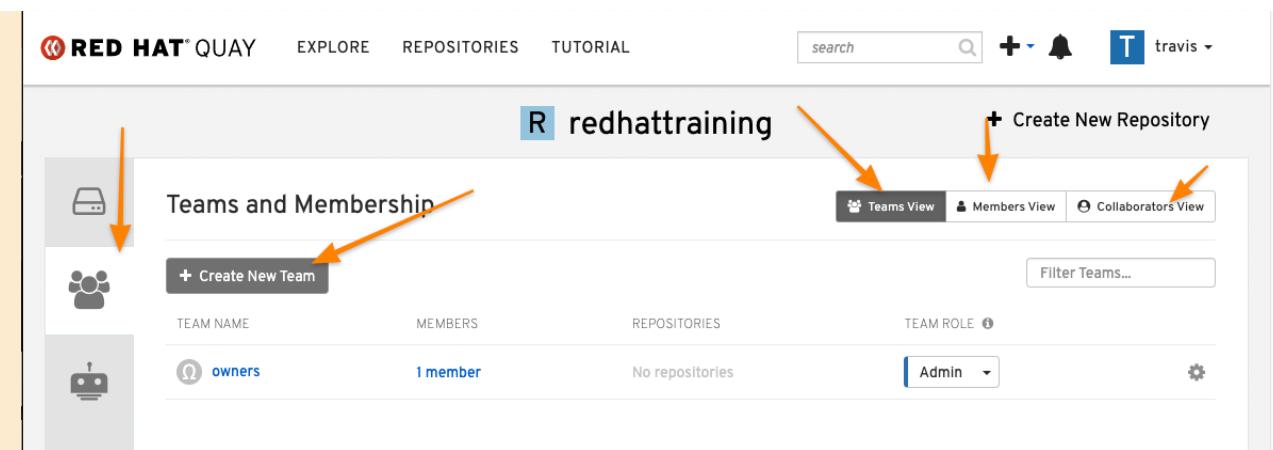


Figure 74. Red Hat Training Organization - Membership

4. Create a team by clicking **Create New Team** and proving a team name.

a. **Team Name:** devteam

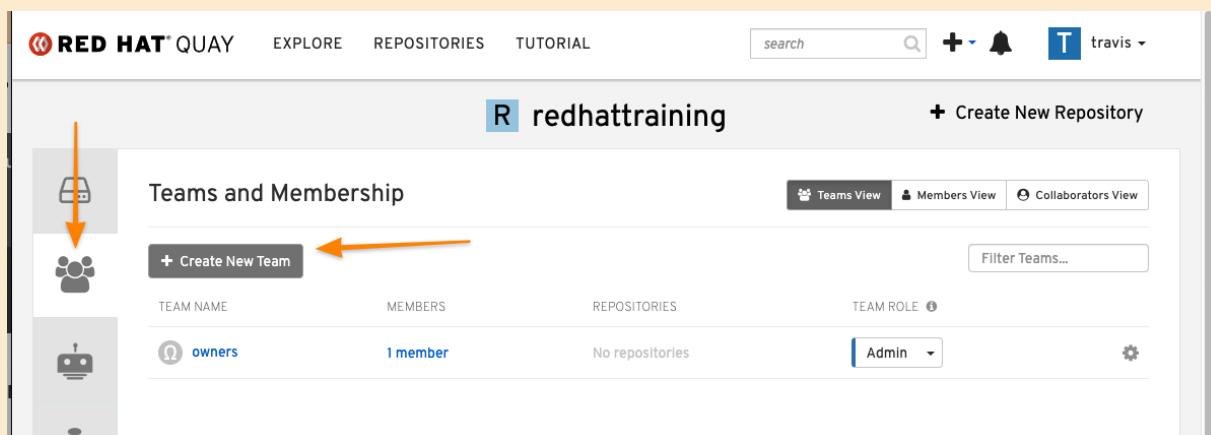


Figure 75. Red Hat Training Organization - Creating a Team

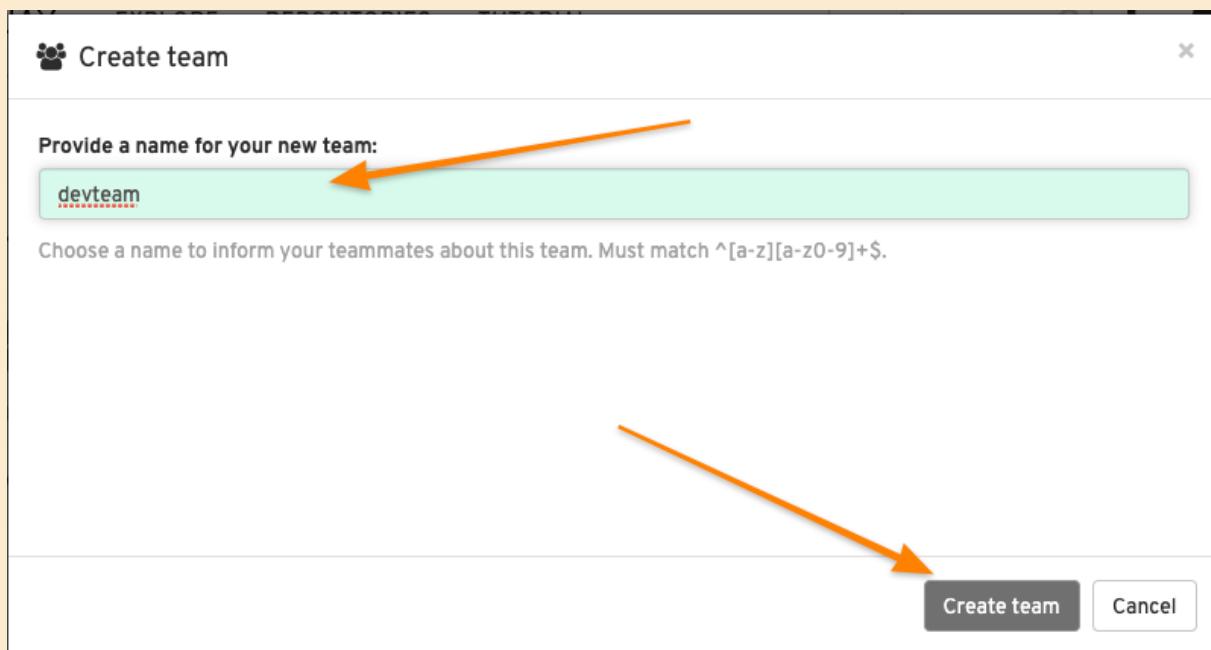
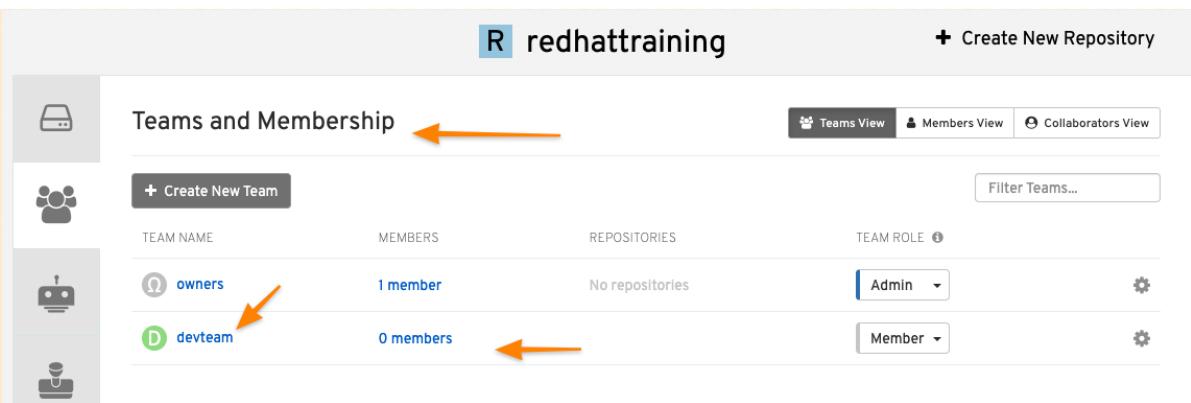


Figure 76. Red Hat Training Organization - Team Creation



TEAM NAME	MEMBERS	REPOSITORIES	TEAM ROLE
owners	1 member	No repositories	Admin
devteam	0 members		Member

Figure 77. Red Hat Training Organization - Membership Verification

5. Add members to the team by clicking the gear and then clicking **Manage Team Members**

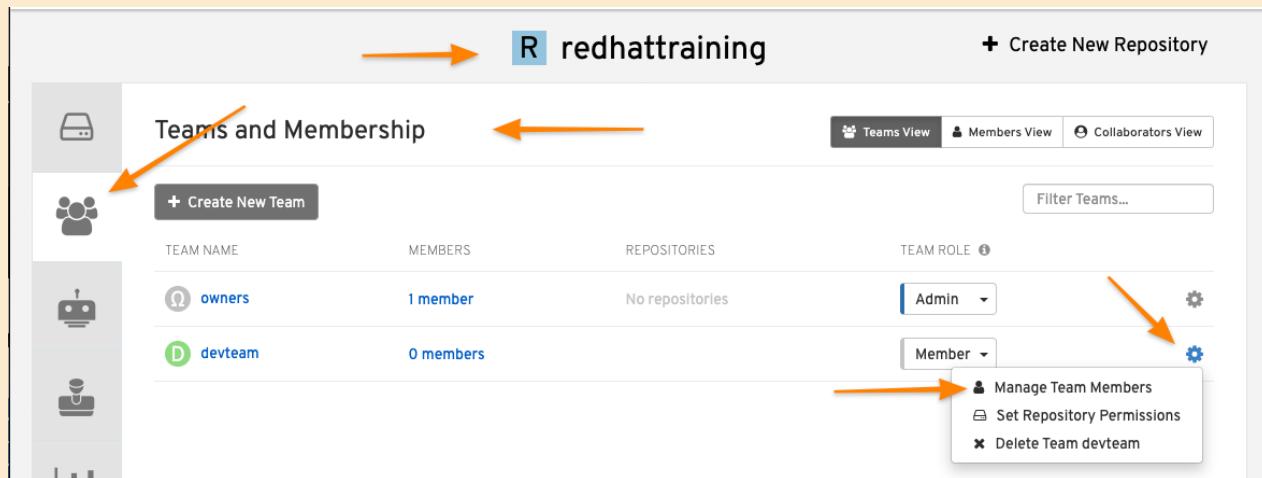


Figure 78. devteam - Membership Modification

6. Search for and add team members and add a team description

- Team Description:** Red Hat Training and Development Team
- Team Members:** <\_add your user\_>

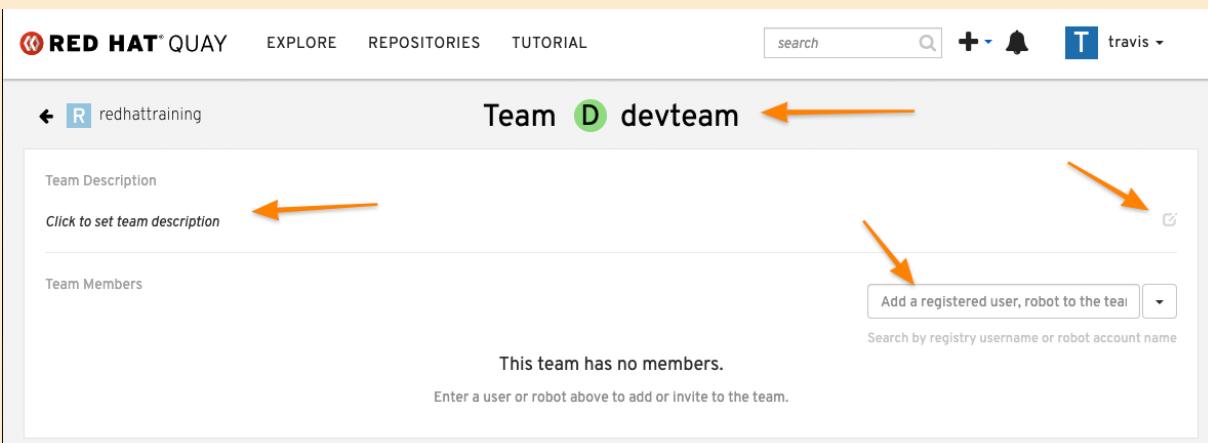


Figure 79. devteam - Edit team and members

The screenshot shows the 'Team' page for 'devteam'. At the top, it says 'Team D devteam'. Below that is a 'Team Description' field containing 'Red Hat Training - Development Team'. Under 'Team Members', there is a list with one item: 'travis'. To the right of the member list is a search bar with the placeholder 'Search by registry username or robot account name'. A gear icon is in the bottom right corner. Three orange arrows are overlaid on the image: one pointing to the team name, one pointing to the member list, and one pointing to the search bar.

Figure 80. devteam - Final Team Details

7. Create default organization permissions. Make selections below and click **Create Permission**
  - a. Permission Applies to Anyone
  - b. **Permission:** Write
  - c. **Applied To:** devteam

The screenshot shows the 'Default Permissions' section of the organization's permissions page. On the left is a sidebar with icons for users, robots, and groups. The main area has a heading 'Default Permissions' with a note: 'The Default permissions panel defines permissions that should be granted automatically to a repository when it is created, in addition to the default of the repository's creator. Permissions are assigned based on the user who created the repository.' Below this is a note: 'Note: Permissions added here do not automatically get added to existing repositories.' At the bottom, it says 'No default permissions defined.' and 'Click the "Create Default Permission" button above to create a new default permission.' An orange arrow points to the 'Create Default Permission' button at the top right of the main area.

Figure 81. Red Hat Training Organization - Permissions

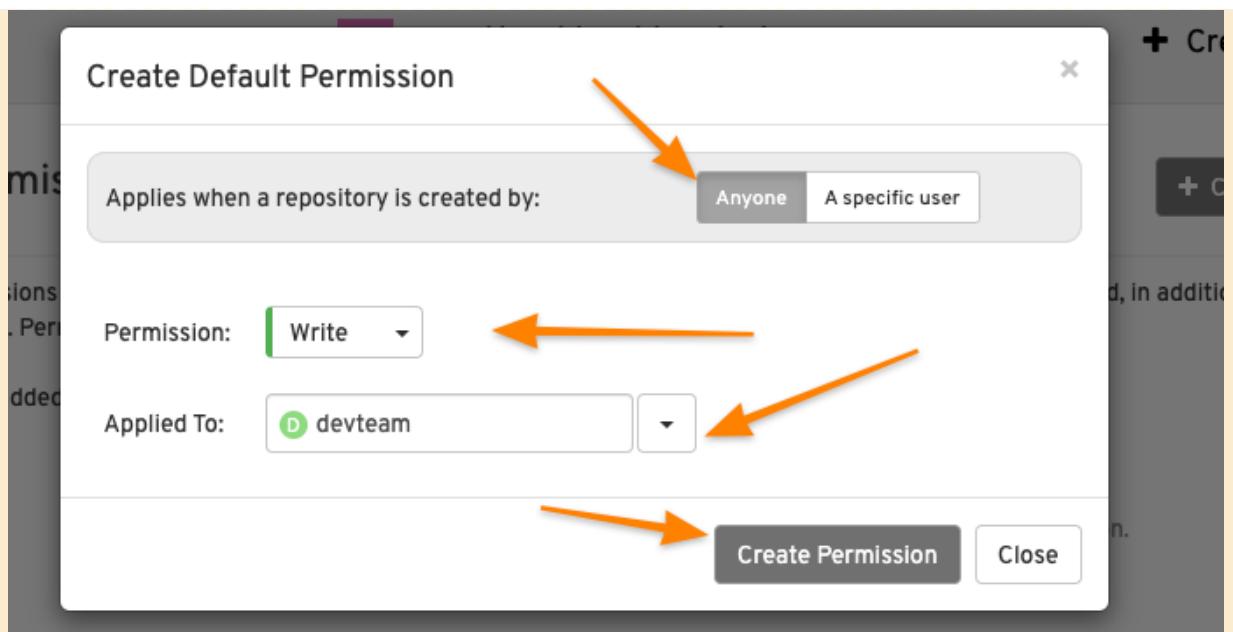


Figure 82. Red Hat Training Organization - Dev Team with Write Permissions

	Default Permissions		<a href="#">+ Create Default Permission</a>
	REPOSITORY CREATED BY	PERMISSION APPLIED TO	PERMISSION
	(ORGANIZATION DEFAULT)	D devteam	Write

Figure 83. Red Hat Training Organization - Permission Verification

### 7.3.4.3. Testing Organizations

Once an organization has been created it is important to test the organization and ensure that members can login and create repositories and are also able to push images to those repositories.

**Example 38. LAB: Testing Organizations**

1. Login to **quay.local** from the workstation machine

```
[student@workstation ~]$ podman login quay.local --tls-verify=false  
Username: travis  
Password:  
Login Succeeded!
```

2. Pull an image down from Quay.io

```
[student@workstation ~]$ podman pull quay.io/redhattraining/hello-world-nginx  
Trying to pull quay.io/redhattraining/hello-world-nginx...  
Getting image source signatures  
  
... OUTPUT OMITTED ...  
  
Writing manifest to image destination  
Storing signatures  
8d990e08937e299ce1d9e629e4df86ef824744e9c9d2057a8883553650d25ba9
```



The **quay.io/redhattraining/hello-world-nginx** image is public so it shouldn't require login and authentication. However, if that changes and you are unable to access the image, login to **quay.io** before pulling the image.

3. Tag image for upload to **quay.local**

```
[student@workstation ~]$ podman tag quay.io/redhattraining/hello-world-nginx  
quay.local/redhattraining/hello-world-nginx
```



The downloaded image from Quay.io has been tagged so it can be uploaded to the local Quay repository at **quay.local**.

4. Verify images and tags

```
[student@workstation ~]$ podman images
REPOSITORY                                     TAG      IMAGE ID      CREATED
SIZE
quay.io/redhattraining/hello-world-nginx     latest   8d990e08937e  2 years ago
269 MB
quay.local/redhattraining/hello-world-nginx   latest   8d990e08937e  2 years ago
269 MB
```

## 5. Upload (push) image to **Quay.local** Repository in the **redhattraining** Organization

```
[student@workstation ~]$ podman push quay.local/redhattraining/hello-world-nginx
--tls-verify=false
Getting image source signatures

... OUTPUT OMITTED ...

Writing manifest to image destination
Storing signatures
```



We are using **podman push** with the **--tls-verify=false** option because we are using HTTP on port 80 as a transport method and we don't have this registry configured in the **registries.conf** file. Additionally, the instance of **quay.local** doesn't currently support TLS/SSL secure connections.

## 6. Verify repository appears in Quay WebUI

The screenshot shows the Red Hat Quay web interface. At the top, there's a browser header with tabs like 'Not Secure | quay.local/repository/' and various bookmarks. Below the header, the Quay logo is on the left, followed by navigation links for 'EXPLORE', 'REPOSITORIES' (which is the active tab), and 'TUTORIAL'. To the right of these are a search bar, a '+' button for creating new repositories, a notification bell, and a user profile for 'travis'. The main content area is titled 'Repositories' and shows a single entry: 'redhattraining / hello-world-nginx'. This entry has columns for 'REPOSITORY NAME', 'LAST MODIFIED', 'STATE', 'ACTIVITY', and 'STAR'. A green activity bar is shown under 'ACTIVITY'. On the right side of the main content, there's a sidebar titled 'Users and Organizations' which lists 'travis' and 'redhattraining', each with a small blue 'T' icon. There are also buttons for '+ Create New Repository' and '+ Create New Organization'.

Figure 84. Hello World NGINX Container Image Repository

## 7. Verify the security scanning worked on image

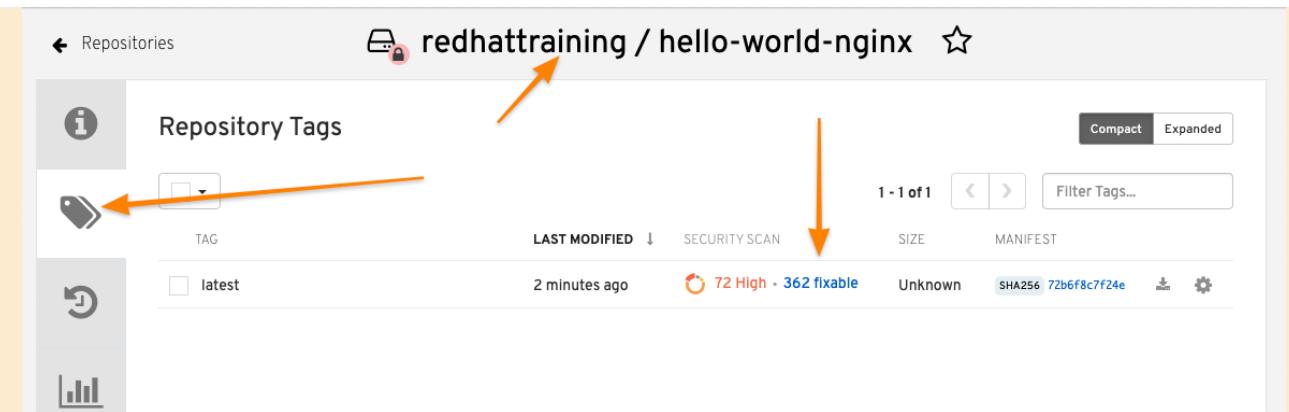


Figure 85. *hello-world-nginx - Image Tags and Scan Results*

8. Observe the status of the repository

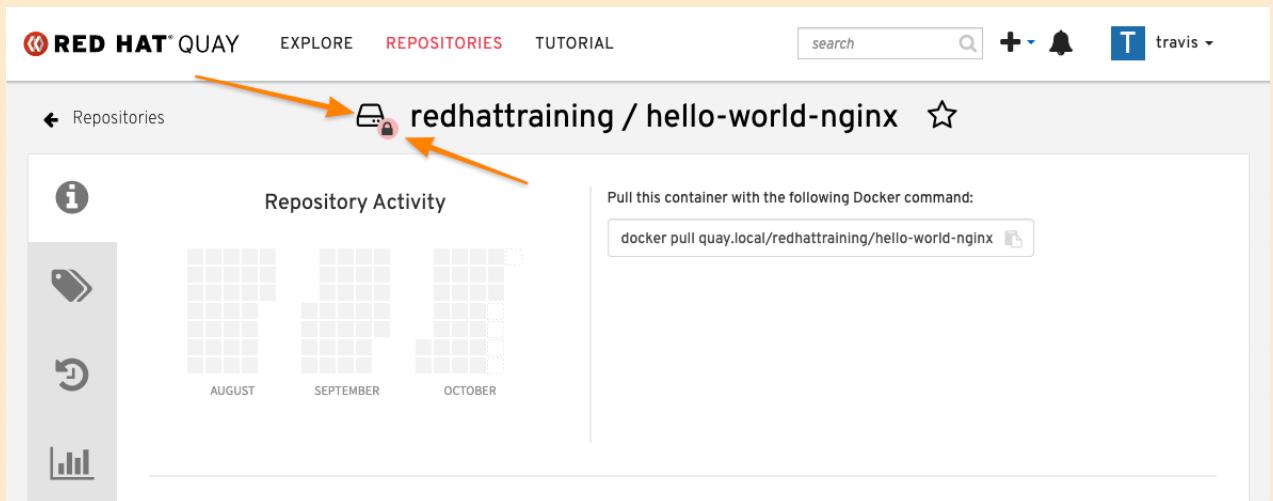
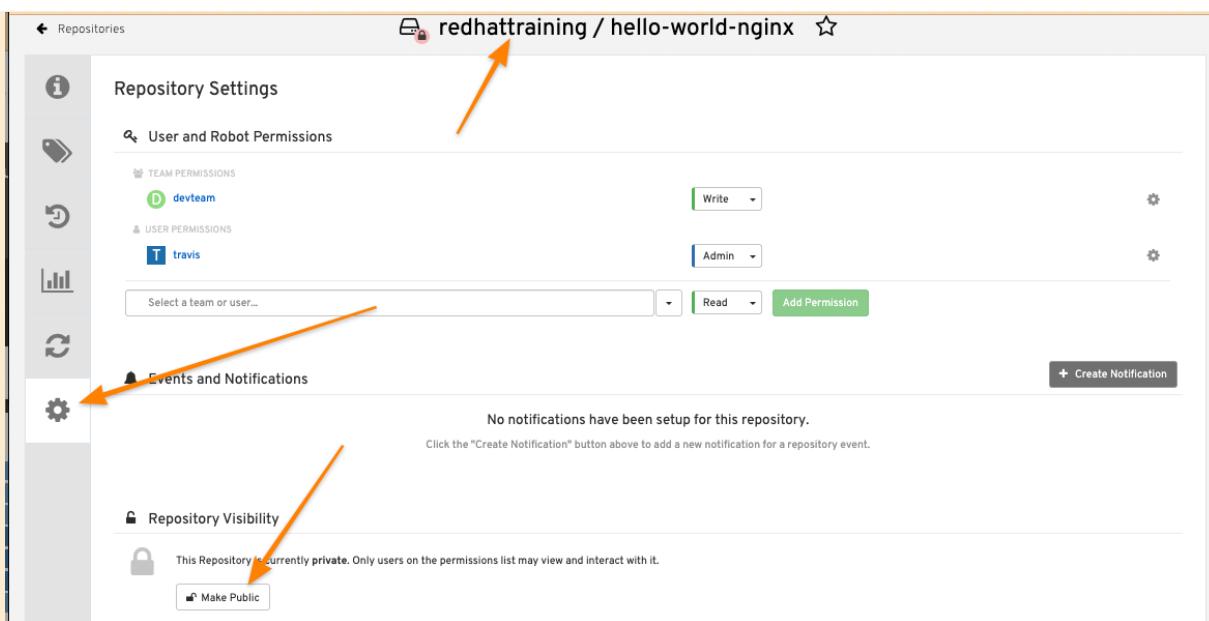
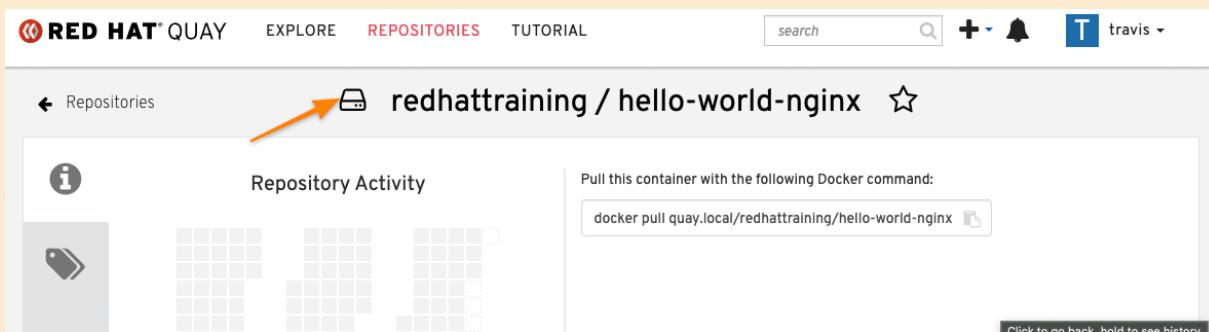


Figure 86. *hello-world-nginx - Private (Locked) Repository*

9. Unlock Repository for Public Use of Image

- Click the **Gear** on the repository navigation left-side bar menu
- Under **Repository Visibility** click **Make Public**

Figure 87. *hello-world-nginx - Making Repository Public*Figure 88. *hello-world-nginx - Verifying Unlocked*

## 10. SSH to **server** as root

```
[student@workstation ~]$ ssh root@server
```

## 11. Login to the Quay registry

```
[root@server ~]# podman login quay.local --tls-verify=false
Username: travis
Password:
Login Succeeded!
```

## 12. Launch container from the **quay.local/redhattraining/hello-world-nginx** container image.

```
[root@server ~]# podman run -d --name quay_test_nginx -p 10080:8080  
quay.local:80/redhattraining/hello-world-nginx  
Trying to pull quay.local:80/redhattraining/hello-world-nginx:latest.  
... OUTPUT OMITTED ...  
  
Storing signatures  
9c9251222721e8ebbd1e38b8a9623b035ba1d814cbfc4dd25fc0256fded25777
```

You might receive an error message about the HTTPS connection to the repository.



#### Error Message

```
Trying to pull quay.local:80/redhattraining/hello-world-nginx:latest... Get "https://quay.local:80/v2/": http: server gave HTTP response to HTTPS client Error: Error initializing source docker://quay.local:80/redhattraining/hello-world-nginx:latest: error pinging docker registry quay.local:80: Get "https://quay.local:80/v2/": http: server gave HTTP response to HTTPS client
```

The error message means that **podman** is still trying to connect via HTTPS to an HTTP registry. If this occurs, update the **registries.conf** file for the user or for the system.

#### *Listing 128. Rgistries.Conf*



```
[root@server ~]# vim /etc/containers/registries.conf  
... OUTPUT OMITTED ...  
  
[registries.search]  
registries = ['registry.access.redhat.com', 'registry.redhat.io',  
'docker.io', 'quay.io', 'quay.local:80'] ①  
  
... OUTPUT OMITTED ...  
  
[registries.insecure]  
registries = ['quay.local:80'] ②
```

- ① Add the new **quay.local** registry with the port to the registries search order
- ② Add the new **quay.local:80** so that **podman** knows it is using HTTP and not HTTPS or if it has a self-signed certificate.

### 13. Verify webserver is running

```
[root@server ~]# curl localhost:10080
<html>
<body>
  <h1>Hello, world from nginx!</h1>
</body>
</html>
```

## 7.4. Inspecting Images with Skopeo on Remote Registries

Skopeo is another new command-line utility that can work in conjunction with both Podman, and Buildah. Skopeo can be run as a regular user or root-level user to interact with container images. While both Podman and Buildah work with container images, Skopeo is designed to work with container images and image repositories. Podman is only able to inspect images in the local image registry whereas Skopeo can inspect both local and remote images.

In addition to gaining information about images from remote registries, Skopeo can also perform a wide array of tasks.

#### *Skopeo Abilities*

- Delete images from local or remote image repositories
- Copy images from one image registry to another
- Inspecting remote images
- Syncing images for offline copying to internal registries (air-gapped / isolated environments)
- Login and authenticate to registries

### 7.4.1. Using Skopeo to Inspect Images and Copy Images

The following exercises will leverage Skopeo to inspect the `quay.local/redhattraining/hello-world-nginx` image remotely. They will also copy the image from the `quay.local/redhattraining` to your private repository.

**Example 39. LAB: Inspecting Remote Images with Skopeo****1. SSH to server**

```
[student@workstation ~]$ ssh root@server
```

**2. Login to quay.local with Skopeo**

```
[root@server ~]# skopeo login quay.local --tls-verify=false
Authenticating with existing credentials... ①
Existing credentials are valid. Already logged in to quay.local
```

① Skopeo capable of using cached credentials from Podman

**3. Inspect the Container from Remote Registry**

```
[root@server ~]# skopeo inspect docker://quay.local:80/redhattraining/hello-world-
nginx
{
    "Name": "quay.local:80/redhattraining/hello-world-nginx",
    "Tag": "latest",
    "Digest":
"sha256:72b6f8c7f24e852f6307661bbb0d7bc153137c2c23aa49f6dd16cf0a8e8093b",
    "RepoTags": [
        "latest"
    ],
    "Created": "2019-06-26T22:59:13.751737399Z",
    "DockerVersion": "",
    "Labels": {
        "architecture": "x86_64",
        "authoritative-source-url": "registry.access.redhat.com",
        "build-date": "2019-04-25T16:26:03.051400",
        "com.redhat.build-host": "cpt-0013.osbs.prod.upshift.rdu2.redhat.com",
        "com.redhat.component": "ubi8-container",
        "com.redhat.license_terms": "https://www.redhat.com/licenses/eulas",
        "description": "The Universal Base Image is designed and engineered to be
the base layer for all of your containerized applications, middleware and
utilities. This base image is freely redistributable, but Red Hat only supports
Red Hat technologies through subscriptions for Red Hat products. This image is
maintained by Red Hat and updated regularly.",
        "distribution-scope": "public",
        "io.k8s.description": "The Universal Base Image is designed and engineered
to be the base layer for all of your containerized applications, middleware and
utilities. This base image is freely redistributable, but Red Hat only supports
Red Hat technologies through subscriptions for Red Hat products. This image is
maintained by Red Hat and updated regularly."
    }
}
```

```

        maintained by Red Hat and updated regularly.",
        "io.k8s.display-name": "Red Hat Universal Base Image 8",
        "io.openshift.expose-services": "",
        "io.openshift.tags": "base rhel8",
        "maintainer": "Red Hat, Inc.",
        "name": "ubi8",
        "release": "122",
        "summary": "Provides the latest release of Red Hat Universal Base Image
8.",
        "url":
"https://access.redhat.com/containers/#/registry.access.redhat.com/ubi8/images/8.0
-122",
        "vcs-ref": "d5ff1490fad8f1b57e451d384d3b331e94cf6fe4",
        "vcs-type": "git",
        "vendor": "Red Hat, Inc.",
        "version": "8.0"
},
"Architecture": "amd64",
"os": "linux",
"Layers": [
    "sha256:4fc3a6f12060dda18cbd448606cbd27c8dee5e0179f9f76a50a8a1de9e5aa827",
    "sha256:cf8bdd5396ac2c59b14aec87491c8d3c3e6c7be355804b6423f8feeffccbd41",
    "sha256:d5072897bb5d09c20671d4a3714bf9203b13887a3e97df6e7770c65e8e5d949d",
    "sha256:e0ed7e500605c8a40f7717871e9b359e65b239236a094544b89d7b2dda95ba3c",
    "sha256:2eb03ea4690d1339f41ef5082b8644ef44e265b73a008c0e05996f1badeca761",
    "sha256:8e4505e250a0379536b9f1a681943a8585bf258e6c218e29f4f4f70ffee8519a",
    "sha256:a8ab12e20ea0ea02fe1f4b0962365d0c146b988c914a112cdd43605f48a44dc1",
    "sha256:1dbcab28ce46b65c0174e5e82658492107396fead31e9144c343e6bc96e471c7",
    "sha256:1dbcab28ce46b65c0174e5e82658492107396fead31e9144c343e6bc96e471c7",
    "sha256:1dbcab28ce46b65c0174e5e82658492107396fead31e9144c343e6bc96e471c7"
],
"Env": [
    "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
    "container=oci"
]
}

```

#### 4. Use Skopeo to inspect configuration and layers of the container

```

[root@server ~]# skopeo inspect --config
docker://quay.local:80/redhattraining/hello-world-nginx
{
    "created": "2019-06-26T22:59:13.751737399Z",
    "architecture": "amd64",
    "os": "linux",
    "config": {
        "User": "1001",

```

```
"ExposedPorts": {
    "8080/tcp": {}
},
"Env": [
    "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
    "container=oci"
],
"Cmd": [
    "/bin/sh",
    "-c",
    "nginx -g \"daemon off;\""
],
"Labels": {
    "architecture": "x86_64",
    "authoritative-source-url": "registry.access.redhat.com",
    "build-date": "2019-04-25T16:26:03.051400",
    "com.redhat.build-host": "cpt-0013.osbs.prod.upshift.rdu2.redhat.com",
    "com.redhat.component": "ubi8-container",
    "com.redhat.license_terms": "https://www.redhat.com/licenses/eulas",
    "description": "The Universal Base Image is designed and engineered to be the base layer for all of your containerized applications, middleware and utilities. This base image is freely redistributable, but Red Hat only supports Red Hat technologies through subscriptions for Red Hat products. This image is maintained by Red Hat and updated regularly.",
    "distribution-scope": "public",
    "io.k8s.description": "The Universal Base Image is designed and engineered to be the base layer for all of your containerized applications, middleware and utilities. This base image is freely redistributable, but Red Hat only supports Red Hat technologies through subscriptions for Red Hat products. This image is maintained by Red Hat and updated regularly.",
    "io.k8s.display-name": "Red Hat Universal Base Image 8",
    "io.openshift.expose-services": "",
    "io.openshift.tags": "base rhel8",
    "maintainer": "Red Hat, Inc.",
    "name": "ubi8",
    "release": "122",
    "summary": "Provides the latest release of Red Hat Universal Base Image 8.",
    "url":
"https://access.redhat.com/containers/#/registry.access.redhat.com/ubi8/images/8.0-122",
    "vcs-ref": "d5ff1490fad8f1b57e451d384d3b331e94cf6fe4",
    "vcs-type": "git",
    "vendor": "Red Hat, Inc.",
    "version": "8.0"
}
},
"rootfs": {
```

```
"type": "layers",
"diff_ids": [
    "",
    "",
    "",
    "",
    "",
    "",
    "",
    "",
    "",
    "",
    "",
    ""
],
},
"history": [
{
    "created": "2019-04-25T16:26:20.260497942Z",
    "comment": "Imported from -"
},
{
    "created": "2019-04-25T16:26:28.425556Z"
},
{
    "created": "2019-06-26T22:33:55.170741575Z",
    "created_by": "/bin/sh -c yum install -y --disableplugin=subscription
manager --nodocs nginx \u0026\u0026 yum clean all"
},
{
    "created": "2019-06-26T22:34:02.362527585Z",
    "created_by": "/bin/sh -c #(nop) ADD index.html /usr/share/nginx/html"
},
{
    "created": "2019-06-26T22:59:11.622403847Z",
    "created_by": "/bin/sh -c #(nop) ADD nginxconf.sed /tmp/"
},
{
    "created": "2019-06-26T22:59:12.412697298Z",
    "created_by": "/bin/sh -c sed -i -f /tmp/nginxconf.sed
/etc/nginx/nginx.conf"
},
{
    "created": "2019-06-26T22:59:13.277995379Z",
    "created_by": "/bin/sh -c touch /run/nginx.pid \u0026\u0026 chgrp -R
0 /var/log/nginx /run/nginx.pid \u0026\u0026 chmod -R g+rwx /var/log/nginx
/run/nginx.pid"
},
{
    "created": "2019-06-26T22:59:13.414802016Z",
}
```

```
"created_by": "/bin/sh -c #(nop) EXPOSE 8080"
},
{
  "created": "2019-06-26T22:59:13.593338162Z",
  "created_by": "/bin/sh -c #(nop) USER 1001"
},
{
  "created": "2019-06-26T22:59:13.751737399Z"
}
]
```

## 5. Copy **hello-world-nginx** to your private repository

```
[root@server ~]# skopeo copy --tls-verify=false
docker://quay.local/redhattraining/hello-world-nginx
docker://quay.local/travis/hello-world-nginx
WARN[0000] '--tls-verify' is deprecated, please set this on the specific
subcommand
Getting image source signatures
Copying blob e0ed7e500605 done
Copying blob cf8bdd5396ac done
Copying blob 8e4505e250a0 done
Copying blob d5072897bb5d done
Copying blob a8ab12e20ea0 done
Copying blob 1dbcab28ce46 done
Copying blob 1dbcab28ce46 skipped: already exists
Copying blob 1dbcab28ce46 skipped: already exists
Copying blob 4fc3a6f12060 done
Copying blob 2eb03ea4690d done
Writing manifest to image destination
Storing signatures
```

Ensure you are logged in



```
[root@server ~]# skopeo login quay.local --tls-verify=false
Username: travis
Password:
Login Succeeded!
```

## 6. Verify that the **hello-world-nginx** now exists in the **Travis** repository/registry.

### a. Click **Repositories**

#### i. Look for **travis/hello-world-nginx**

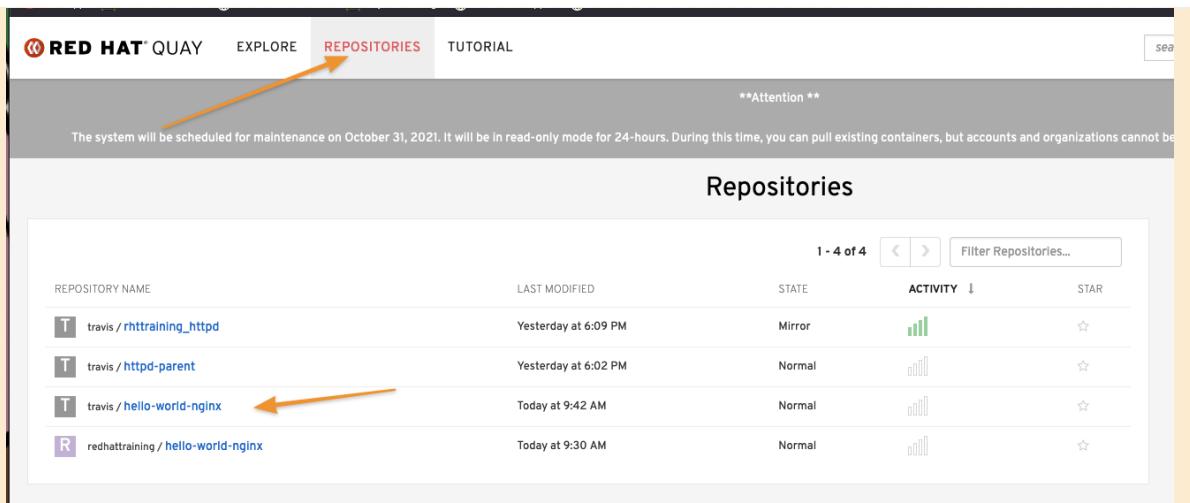


Figure 89. Verification of Successful Skopeo Copy

### References

**Deploy Red Hat Quay for proof-of-concept (non-production) purposes:**  
[https://access.redhat.com/documentation/en-us/red\\_hat\\_quay/3/html/deploy\\_red\\_hat\\_quay\\_for\\_proof-of-concept\\_non-production\\_purposes/index](https://access.redhat.com/documentation/en-us/red_hat_quay/3/html/deploy_red_hat_quay_for_proof-of-concept_non-production_purposes/index)

**Deploying Quay as a local registry server using local storage and containerized services.**

**Quay Lab PoC:** [https://github.com/tmichett/quay\\_lab\\_poc](https://github.com/tmichett/quay_lab_poc) Deploying the Quay Registry locally based on the Proof-of-Concept local deployment.

**Skopeo - Exercise from redhatgov.io:** [http://redhatgov.io/workshops/security\\_openshift/exercise1.4/](http://redhatgov.io/workshops/security_openshift/exercise1.4/)

**Skopeo on Github:** <https://github.com/containers/skopeo>

**How to run Skopeo in a container:** <https://www.redhat.com/sysadmin/how-run-skopeo-container>

**Introduction to using buildah, podman and skopeo to work on containers:** [http://redhatgov.io/workshops/rhel\\_8/exercise1.8/](http://redhatgov.io/workshops/rhel_8/exercise1.8/)



# Appendix A: Running Containers

## A.1. Introducing Containers

### A.1.1. Introducing Container Technology

Containers provide a way to package and deploy applications avoiding package conflicts and dependencies with runtime environments. A container can be a bundled package or packages that are assembled and deployed in a standard way which run isolated from the host system environment.

#### A.1.1.1. Comparing Containers to Virtual Machines

Containers provide the same benefits as VMs:

- Security
- Storage Isolation
- Network Isolation

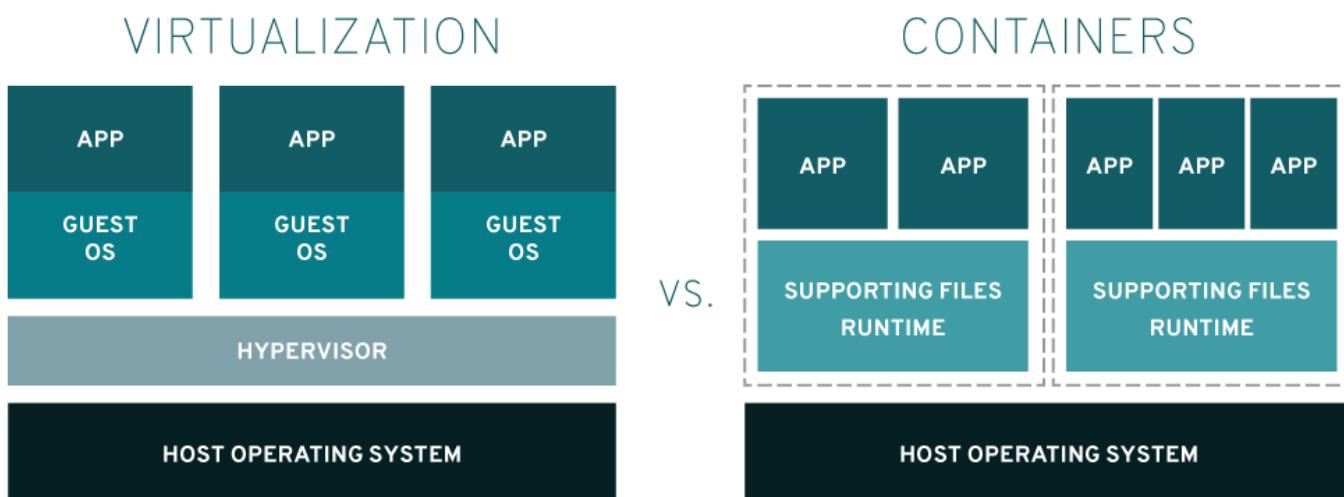


Figure 90. Virtualization vs. Containers

Table 7. Container and Virtualization Differences

Virtualization	Container
Multiple operating systems run on a single hardware platform	Applications run natively on the operating system and share hardware and OS resources across all containers
Uses hypervisor to allocate hardware resources into multiple virtual systems	Shares same OS kernel, but isolates containerized application processes from the rest of the system

Virtualization	Container
Requires a complete OS in the virtual environment to support the application	Requires fewer hardware resources allowing for quick start/stop. Also, uses the host OS and any software that is compatible with the kernel.

### A.1.1.2. Exploring the Implementation of Containers

Containers use the following core components of Linux:

- **Control Groups (cgroups)** - Used for resource management
- **Namespaces** - Used for process isolation
- **SELinux and Secomp (Secure Computing Mode)** - Used to enforce security boundaries.

### A.1.1.3. Planning for Containers

Containers provide reusability and portability for hosted and containerized applications. These containers can easily be moved between environments and are considered temporary. If containers rely on permanent data, persistent storage should be presented to the container.

### A.1.1.4. Running Containers from Container Images

Containers are run from **images**. Images serve as blueprints for creating containers.

Container images package an application together with:

- System libraries
- Programming language runtimes and libraries
- Configuration settings
- Static files

Container images are immutable and built according to specifications such as Open Container Initiative (OCI) image format.

### A.1.1.5. Designing Container-based Architectures

It is possible to install complex software applications and services in a single container, but it is often harder to manage. Instead, it is recommended to have each component of an application in a container, forming a type of microservices.

## A.1.2. Managing Containers with Podman

To learn about containers, it is best to work with individual containers on a server which serves as a container host. The **Container Tools** package will provide the OCI tools for RHEL allowing containers

to be built and run on the host.

#### Container Tools

- **podman** - Manages containers and container images
- **skopeo** - Used to inspect, copy, delete, and sign container images
- **buildah** - Used to create new container images

The tools above are compatible with the OCI and can be used to manage Linux containers created by OCI-compatible container engines such as docker.

#### A.1.2.1. Running Rootless Containers

The **podman** command is used to run containers as the **root** user or by non-privileged users. Containers run by non-privileged users are called *rootless containers*.

Rootless containers are more secure but can't do as much, including publishing services through the host's privileged ports (ports below 1024).

##### *Security Concerns - Running a Container as Root*



Containers can be run directly as root, but system security is weakened as an attacker could compromise the container.

#### A.1.3. Managing Containers at Scale

Managing containers at scale is extremely difficult. **Kubernetes** was developed as an orchestration service to assist with managing and deploying containers at scale in an enterprise environment. Red Hat provides a Kubernetes distribution called Red Hat OpenShift Container Platform (RHOC or OCP).

## A.2. Running a Basic Container

#### A.2.1. Installing Container Management Tools

In RHEL8, Container Tools are in the **container-tools** module. The **container-tools** module provides the container tools needed to run containers (specifically **podman**, **skopeo**, and **buildah**).

#### A.2.2. Selecting Container Images and Registries

Container registries are repositories for storing and retrieving container images. Container images are generally uploaded or pushed to a registry by a developer. Images are downloaded or pulled from a container registry to a local system so that they can be used to run containers.

##### *Red Hat Container Registries*

- **registry.redhat.io** - Containers based on official Red Hat products
- **registry.connect.redhat.com** - Containers based on third-party products

### *Older Red Hat Registry*



Red Hat is phasing out the older **registry.access.redhat.com** container registry in favor of the new container registries.

### *Red Hat Container Catalog*



The Red Hat Container Catalog (RHCC) is available at <https://access.redhat.com/containers> and provides a web-based interface for searching container registries for containers.

### *Classroom Container Registry*



The container registry used for the classroom **registry.lab.example.com** is a private container registry running on he

#### A.2.2.1. Container Naming Conventions

Container images are based on a *fully qualified image name* syntax as follows:

**registry\_name/user\_name/image\_name:tag**

- **registry\_name** - Name of registry storing the image.
- **user\_name** - Name of user or organization where the image is located
- **image\_name** - Name of the image. Must be unique in the user namespace
- **tag** - Identifies image version. If no image tag is provided, it is assumed that the **latest** image tag is used.

#### A.2.3. Running Containers

Before running containers on a system a container image must first be pulled from a registry. Podman can be used to download an image from a container registry using the **podman pull** command.

*Listing 129. Getting the latest UBI Image (Universal Base Image)*

```
[student@servera ~]$ podman pull registry.access.redhat.com/ubi8/ubi:latest
Trying to pull registry.access.redhat.com/ubi8/ubi:latest...
Getting image source signatures
Copying blob c4d668e229cd done
...
Storing signatures
ecbc6f53bba0d1923ca9e92b3f747da8353a070fccbae93625bd8b47dbe772e
[student@servera ~]
```

Images that have been downloaded with the Podman command can be seen with the **podman images** command.

*Listing 130. Listing Local Images*

```
[student@servera ~]$ podman images
REPOSITORY                      TAG      IMAGE ID      CREATED       SIZE
registry.access.redhat.com/ubi8/ubi    latest   ecbc6f53bba0  2 weeks ago  211 MB
registry.lab.example.com/rhel8/httpd-24  latest   7e93f25a9468  8 weeks ago  430 MB
```

A container is created and run by using the **podman run** command. When executed, Podman creates and starts a new container based on the container image specified in the command.

*Listing 131. Running the UBI Image*

```
[student@servera ~]$ podman run --name ubi-demo -it
registry.access.redhat.com/ubi8/ubi:latest

[root@d58f1e8fb4ca /]# cat /etc/redhat-release
Red Hat Enterprise Linux release 8.2 (Ootpa)
```

### **podman run Options**

The **podman run** command supports many options.

- **-t or --tty** - Allocates a **pseudo-tty** (pseudo-terminal) to the container.
- **i or --interactive** - Allows the container to accept standard input.
- **-d or --detach** - Allows the container to run in the background and displays generated container ID.
- **--name** - Allows a name to be specified to the container which must be unique, but allows access to the container with a user generated name rather than a container ID.

### *Containers Cleanup*

It is important to cleanup containers and images after a container has been run. If a user is wanting to execute a quick command within a container, the **podman run --rm** command can be used in order to execute a command in the container and remove the container once the command has been completed.

```
[student@servera ~]$ podman run --rm registry.access.redhat.com/ubi8/ubi
cat /etc/redhat-release
Red Hat Enterprise Linux release 8.2 (Ootpa)
```

## A.2.4. Analyzing Container Isolation

Containers provide runtime isolation of resources by utilizing Linux namespaces. Processes running in a container are isolated from all other processes on the host machine.

*Listing 132. Viewing Processes in a Container*

```
[student@servera ~]$ podman run -it registry.access.redhat.com/ubi7/ubi /bin/bash

[root@9fd13da02e18 /]# cat /etc/redhat-release
Red Hat Enterprise Linux Server release 7.8 (Maipo)

[root@9fd13da02e18 /]# ps -eaf
UID          PID      PPID    C STIME TTY          TIME CMD
root           1        0  0 20:28 pts/0    00:00:00 /bin/bash
root          15        1  0 20:29 pts/0    00:00:00 ps -eaf

[root@9fd13da02e18 /]# id
uid=0(root) gid=0(root) groups=0(root)
```

## A.2.5. DEMO - Running a Basic Container

*Example 40. DEMO - Running a Basic Container**Before Beginning*

```
[student@workstation ~]$ lab containers-basic start
```

1. Install **podman** and other container tools packages

*Listing 133. Installing Container Tools*

```
[student@servera ~]$ sudo yum module install container-tools
```

*Container Tools*

The container tools module installs many container applications.



- buildah
- Cockpit Podman
- podman
- runc
- skopeo

2. Login to the registry

```
[student@servera ~]$ podman login registry.lab.example.com -u admin  
Password:  
Login Succeeded!
```

*Credential Security*

While it is possible to specify the **-u** and a username on the command line as well as the **-p** and a password, normally, the password is left to be interactive to prevent the password from being stored in the shell history.

3. Download a Container image.

```
[student@servera ~]$ podman pull registry.lab.example.com/rhel8/httpd-24  
Trying to pull registry.lab.example.com/rhel8/httpd-24...  
Getting image source signatures  
... output omitted ...  
7e93f25a946892c9c175b74a0915c96469e3b4845a6da9f214fd3ec19c3d7070
```

#### 4. List available images

```
[student@servera ~]$ podman images
REPOSITORY          TAG      IMAGE ID      CREATED
SIZE
registry.lab.example.com/rhel8/httpd-24    latest   7e93f25a9468  8 weeks ago
430 MB
```

#### 5. Run image in a container called demo

```
[student@servera ~]$ podman run --name demo registry.lab.example.com/rhel8/httpd-
24 &
```



The Apache HTTPD container is set to run the HTTPD service in the foreground. The & is added to push the service to the background.

#### 6. Open an interactive shell in the **demo** container.

```
[student@servera ~]$ podman exec -it demo /bin/bash
bash-4.4$
```

#### 7. Explore the container

*Listing 134. Determine the Version of Apache*

```
bash-4.4$ httpd -v
Server version: Apache/2.4.37 (Red Hat Enterprise Linux)
Server built:   Dec 2 2019 14:15:24
```

*Listing 135. Determine UID from Container*

```
bash-4.4$ id
uid=1001(default) gid=0(root) groups=0(root)
```

#### 8. Exit the container interactive shell

```
bash-4.4$ exit
exit
```

#### 9. Stop and remove container

```
[student@servera ~]$ podman rm demo --force
[Wed Sep 16 15:47:36.131159 2020] [mpm_event:notice] [pid 1:tid 139897800703424]
AH00491: caught SIGTERM, shutting down
71aacac8f4af690b1abfb38a8426a90deb44096c3e4eb8db1f5d07189ed813f
```

### **podman ps -a** Can List All Containers on a System

The **podman ps** command lists running containers. However, some containers either exit for various reasons or the system administrator has stopped the container. In these instances, the containers are not running, however, they are still available on the system. The **podman ps -a** command can show the status of all containers (running and stopped).

*Listing 136. Listing All Containers*

```
[student@servera ~]$ podman ps -a
CONTAINER ID  IMAGE                                     COMMAND
CREATED      STATUS          PORTS  NAMES
9fd13da02e18  registry.access.redhat.com/ubi7/ubi:latest /bin/bash
About a minute ago  Exited (0) 3 seconds ago          cool_shannon
d58f1e8fb4ca  registry.access.redhat.com/ubi8/ubi:latest /bin/bash  9
minutes ago    Exited (127) 4 minutes ago           ubi-demo
```



Containers can be restarted if they are **stopped** and will contain the same data. However, if a new container is launched from a container image, any data from the *stopped* container is not accessible.

Once a container is no longer of use, it should be cleaned up and removed from the system. The **podman rm** command can remove single containers by container name or container ID. It is also possible to use **podman rm -a** to remove all containers. If you want to stop and remove a container, it is possible to use the **--force** to kill a container process so that it can be removed in a single command.

*Listing 137. Removing All Containers*

```
[student@servera ~]$ podman rm -a
9fd13da02e181d2dff09808e3f1ac860d637c1be8d57638d90080aec179e850
d58f1e8fb4caaf960fb51d1942861e33b98070b8af415cc414620bce43ba9f6c
```

## A.3. Finding and Managing Container Images

### A.3.1. Configuring Container Registries

Podman uses the **registry.conf** file on the system to retrieve information for container registries. The file is located at **/etc/containers/registries.conf**. The list of registries Podman uses are configured in the **[registries.search]** section and the **[registries.insecure]** section of the configuration file.

#### *Rootless Podman Configuration*



The system-wide **registries.conf** file located at **/etc/containers/registries.conf** can be overridden by a user's configuration file located in **\$HOME/.config/containers** directory.

#### *Container Image Registries and Searching*



While it is best practice to specify a fully qualified image, it is possible to provide just an image name on the command line. If an image name is specified, it will search the registries in the listed order to determine which image to select.

The **podman info** command can display information about Podman as well as configured registries.

*Listing 138. podman info Demo*

```
[student@servera ~]$ podman info
host:
arch: amd64

... output omitted ...

registries:
registry.lab.example.com:
  Blocked: false
  Insecure: true
  Location: registry.lab.example.com
  MirrorByDigestOnly: false
  Mirrors: null
  Prefix: registry.lab.example.com
search:
- registry.lab.example.com
```

#### A.3.1.1. Registry Security

Insecure registries can be listed in the **[registries.insecure]** section of the **registries.conf** file. If a registry is insecure, that means it is not protected with SSL/TLS encryption. In order for an insecure registry to be searchable, it should exist in both the **[registries.search]\*** section and the **[registries.insecure]** section of the **registries.conf** file.



### *Image Repositories Requiring Authentication*

Some container registries require authentication. In this instance, the **podman login** command can be used to login to a container registry.

## A.3.2. Finding Container Images

The **podman search** command is used to search container registries for a specific container image.

*Listing 139. Using podman search to Find UBI7 Image*

```
[student@servera ~]$ podman search registry.redhat.io/ubi7
INDEX      NAME                           DESCRIPTION
STARS      OFFICIAL   AUTOMATED
redhat.io  registry.redhat.io/ubi7/ubi    The Universal Base Image is designed
and eng...  0
redhat.io  registry.redhat.io/ubi7/ubi-minimal  The Universal Base Image Init is
designed to...  0
...
... output omitted ...
```

*Table 8. Podman Search Options*

Option	Description
<b>--limit &lt;number&gt;</b>	Limits number of listed images per registry
<b>--filter &lt;filter=value&gt;</b>	Filters output based on conditions: <ul style="list-style-type: none"><li>• <b>stars=&lt;number&gt;</b>: Images must contain at least X stars</li><li>• <b>is-automated=&lt;true/false&gt;</b>: Show only automatically build images</li><li>• <b>is-official=&lt;true/false&gt;</b>: Show only images flagged with official</li></ul>
<b>--tls-verify &lt;true/false&gt;</b>	Enables or disables HTTPS certificate validation. Default= <b>true</b>

### A.3.2.1. Using the Red Hat Container Catalog

Repositories maintained by Red Hat containing certified container images. The standard **podman** command is compatible with the repositories referenced by the Red Hat Container Catalog (RHCC).

### A.3.3. Inspecting Container Images

Information about images can be obtained before downloading them to the system by using the **skopeo inspect** command to inspect images in a container registry and display information about the image.

#### skopeo inspect



The **skopeo inspect** command can inspect different image formats from multiple sources including remote registries or local directories. The **docker://** transport mechanism instructs skopeo to query a container image registry.

*Listing 140. Inspecting an Image*

```
[student@servera ~]$ skopeo inspect docker://registry.redhat.io/ubi7/ubi
{
    "Name": "registry.redhat.io/ubi7/ubi",
    ...
    ... output omitted ...

    "Created": "2020-08-03T09:32:22.835719Z",
    "DockerVersion": "1.13.1",
    ...
    ... output omitted ..

    "description": "The Universal Base Image is designed and engineered to be the
base layer for all of your containerized applications, middleware and utilities. This
base image is freely redistributable, but Red Hat only supports Red Hat technologies
through subscriptions for Red Hat products. This image is maintained by Red Hat and
updated regularly.",
    ...
    ... output omitted ..
```

### A.3.4. Removing Local Container Images

Container images are immutable and do not change. In order to update software in a container, a new image is created to replace the old image. The newest image will always have the **latest** tag associated with it. As container images are replaced with new images, it becomes necessary for older images to be removed from the system to manage storage and images effectively. The **podman rmi** command can remove and delete the images from the local system. The **podman images** command can be used to list local images, and then the **podman rmi** command can remove the image from the system.

***Listing 141. Performing Image Cleanup***

```
[student@servera ~]$ podman images
REPOSITORY                      TAG      IMAGE ID      CREATED     SIZE
registry.access.redhat.com/ubi8/ubi    latest   ecbc6f53bba0  2 weeks ago  211 MB
registry.access.redhat.com/ubi7/ubi    latest   87dd8ec61bbc  6 weeks ago  215 MB
registry.lab.example.com/rhel8/httpd-24 latest   7e93f25a9468  8 weeks ago  430 MB

[student@servera ~]$ podman rmi registry.access.redhat.com/ubi7/ubi
Untagged: registry.access.redhat.com/ubi7/ubi:latest
Deleted: 87dd8ec61bbc977ed1b2bd29ba089489a8a0281e2e0f62bf43c3fb9c20414194

[student@servera ~]$ podman images
REPOSITORY                      TAG      IMAGE ID      CREATED     SIZE
registry.access.redhat.com/ubi8/ubi    latest   ecbc6f53bba0  2 weeks ago  211 MB
registry.lab.example.com/rhel8/httpd-24 latest   7e93f25a9468  8 weeks ago  430 MB
```

### A.3.5. DEMO - Finding and Managing Container Images

**Example 41. DEMO - Finding and Managing Container Images**

1. Verify that the registries are configured

```
[student@servera ~]$ cat .config/containers/registries.conf
unqualified-search-registries = ['registry.lab.example.com']

[[registry]]
location = "registry.lab.example.com"
insecure = true
blocked = false
```

2. Search for an image

```
[student@servera ~]$ podman search registry.lab.example.com/nginx
INDEX      NAME          DESCRIPTION      STARS
OFFICIAL   AUTOMATED
example.com  registry.lab.example.com/library/nginx           0
```

3. Inspect the image with Skopeo

```
[student@servera ~]$ skopeo inspect
docker://registry.lab.example.com/library/nginx
{
    "Name": "registry.lab.example.com/library/nginx"
    ...
    ... output omitted ...
}
```

It may be necessary to login and authenticate to the container registry.



*Listing 142. Autentication to Container Registry*

```
[student@servera ~]$ podman login registry.lab.example.com
```

4. Pull the image down

```
[student@servera ~]$ podman pull registry.lab.example.com/nginx
Trying to pull registry.lab.example.com/nginx...
... output omitted ...

Storing signatures
4bb46517cac397bdb0bab6eba09b0e1f8e90ddd17cf99662997c3253531136f8
```

## 5. Inspect the image with **podman inspect**

```
[student@servera ~]$ podman inspect registry.lab.example.com/nginx
[
{
    "Id": "4bb46517cac397bdb0bab6eba09b0e1f8e90ddd17cf99662997c3253531136f8",
    ... output omitted ...
```

## 6. Run the image as a container

```
[student@servera ~]$ podman run -d --name nginxdemo registry.lab.example.com/nginx
b90de12239975532a66bae67e87dcc487252765bee018970be45bd0ed71115af
```

## 7. Verify Image is Running

```
[student@servera ~]$ podman ps
CONTAINER ID IMAGE COMMAND CREATED
STATUS PORTS NAMES
b90de1223997 registry.lab.example.com/nginx:latest nginx -g daemon o... 29
seconds ago Up 29 seconds ago nginxdemo
```

## 8. Delete image and running container in a single command.

```
[student@servera ~]$ podman rmi registry.lab.example.com/nginx --force
Untagged: registry.lab.example.com/nginx:latest
Deleted: 4bb46517cac397bdb0bab6eba09b0e1f8e90ddd17cf99662997c3253531136f8
```

## 9. Verify container was stopped and image was deleted and removed

*Listing 143. Source Description*

```
[student@servera ~]$ podman ps -a
CONTAINER ID  IMAGE   COMMAND   CREATED   STATUS    PORTS   NAMES
[student@servera ~]$ podman images
REPOSITORY          TAG      IMAGE ID      CREATED       SIZE
registry.access.redhat.com/ubi8/ubi     latest    ecbc6f53bba0  2 weeks ago   211
MB
registry.lab.example.com/rhel8/httpd-24  latest    7e93f25a9468  8 weeks ago   430
MB
```

## A.4. Performing Advanced Container Management

### A.4.1. Administering Containers with Podman

The **podman** command can be used to manage running or stopped containers.

### A.4.2. Configuring Containers

The **podman run** command can start containers from an image. It can also be used to provide network access, container storage, and pass environment variables and settings to the container instead of modifying the container image.

#### A.4.2.1. Mapping Container Host Ports to the Container

When a network port on the container host is mapped to a port in the container, network traffic sent to the host port is redirected and received by the container. The **podman run -p** is used to map network ports from the host to a container port. Another reminder is that the **-d** option can cause a container to run in detached mode (daemon / background mode).

*Listing 144. Running Apache and Mapping Network Ports*

```
[student@servera ~]$ podman run -d -p 8000:8080 --name webdemo
registry.redhat.io/rhel8/httpd-24
```

*Listing 145. Verifying the Port Mapping*

```
[student@servera ~]$ podman port -a
0d1d688ebbbd  8080/tcp -> 0.0.0.0:8000
```

### Container Host Firewall

It is important to note that while there is no firewall service running in the container, the container host must allow the network traffic on whatever port has been forwarded to the container.

*Listing 146. Using FirewallD on Container Host*



```
[student@servera ~]$ sudo firewall-cmd --add-port=8000/tcp  
[sudo] password for student:  
success
```

A rootless container cannot open ports on the container host below 1024. Ports above 1024 can be mapped from the container host to any port in the container even when running a rootless container.

#### A.4.2.2. Passing Environment Variables to Configure a Container

Environment variables can be used to pass values to a container without modifying the container image. In order to see the environment variables a container image accepts, you can use the **podman inspect** command. There are different ways on obtaining information a container can use ... one is to look at the **URL** which will point to a web page documenting the possible environment variables and how to use the container image.

The other method of determining possible environment variables can be the **usage** portion returned from **podman inspect** as this will often show how to run the container and provide environment variables with the **-e** on the command line.

*Listing 147. Inspecting a Container*

```
[student@servera ~]$ podman pull registry.redhat.io/rhel8/mariadb-103:1-102  
  
[student@servera ~]$ podman inspect registry.redhat.io/rhel8/mariadb-103:1-102 | grep url  
    "io.openshift.s2i.scripts-url": "image:///usr/libexec/s2i",  
    "io.s2i.scripts-url": "image:///usr/libexec/s2i",  
    "url":  
"https://access.redhat.com/containers/#/registry.access.redhat.com/rhel8/mariadb-103/images/1-102",  
  
[student@servera ~]$ podman inspect registry.redhat.io/rhel8/mariadb-103:1-102 | grep usage  
    "usage": "podman run -d -e MYSQL_USER=user -e MYSQL_PASSWORD=pass -e  
MYSQL_DATABASE=db -p 3306:3306 rhel8/mariadb-103",
```

The **podman run -e VARNAME=Value** will pass environment variables to the container and process this inside the container image.

### A.4.3. Managing Containers

The **podman ps** command will list running containers while the **podman ps -a** command will list all containers, even if they have been stopped.

*Table 9. Podman Commands*

Podman Command	IP addresses
<b>podman run</b>	Starts a container based on a given container image and accepts other parameters such as <b>--name</b> specifying container name, <b>-d</b> for daemon/detached mode, <b>-p</b> to perform port forwarding, <b>it</b> to allow interactive commands, and more.
<b>podman stop</b>	Stops a container gracefully by name or container ID. The <b>podman stop --all</b> can stop all running containers
<b>podman rm</b>	The <b>podman rm &lt;ContainerName/ID&gt;</b> removes the stopped container from the host. Using <b>podman rm -a</b> will remove all stopped containers. It is also possible to use the <b>podman rm -a -f</b> to force stop and remove containers from the system.
<b>podman rmi / podman image rm</b>	Removes the specified image from the system.
<b>podman restart</b>	Restarts a stopped container. Creating a new container with same container ID and reuses the container state and filesystem.
<b>podman kill</b>	Sends UNIX signals to main process in the container. These signals are the same as those used by the <b>kill</b> command.

#### *Removing a Container and an Image*



It is possible to delete an image for a running container if you use the **-f** option with **podman rmi** command. This will stop and remove the containers using the image and it will then delete the image from the local system.

**podman stop** *Note*

The **podman stop** command will attempt to stop the container gracefully, but if the stop command fails, it sends a **SIGTERM** and **SIGKILL** signal to the container.

#### A.4.4. Running Commands in a Container

In addition to running commands when launching the container with the **-it** option, it is possible to launch an interactive shell in order to debug a container or inspect the contents and other items in a container. The **podman exec** command can be used on a running container to provide an interactive shell within the container.

*Listing 148. Using podman exec*

```
[student@servera ~]$ podman exec -it demo /bin/bash
bash-4.4$ id
uid=1001(default) gid=0(root) groups=0(root)

bash-4.4$ exit
exit
```

#### A.4.5. DEMO - Performing Advanced Container Management

*Example 42. DEMO - Advanced Container Management*

## 1. Login to the Registry

```
[student@servera ~]$ podman login registry.lab.example.com
Username: admin
Password: redhat321
Login Succeeded!
```

## 2. Launch a MySQL Database

```
[student@servera ~]$ podman run -d --name demodb -e MYSQL_USER=demouser -e
MYSQL_PASSWORD=redhat -e MYSQL_DATABASE=items -e MYSQL_ROOT_PASSWORD=redhat -p
3306:3306 registry.lab.example.com/rhel8/mariadb-103:1-102
```

## 3. List running containers

```
[student@servera ~]$ podman ps
CONTAINER ID  IMAGE                                     COMMAND
CREATED      STATUS          PORTS          NAMES
9a9bf84dcaab  registry.lab.example.com/rhel8/mariadb-103:1-102  run-mysqld  19
seconds ago   Up 19 seconds ago  0.0.0.0:3306->3306/tcp  demodb
```

## 4. List Port Forwards

```
[student@servera ~]$ podman port -a
9a9bf84dcaab  3306/tcp -> 0.0.0.0:3306
```

## 5. Attempt to connect to MySQL Instance

*Listing 149. Install MariaDB Client Package*

```
[student@servera ~]$ sudo yum install mariadb
```

*Listing 150. Connect to MySQL Instance*

```
[student@servera ~]$ mysql -u demouser -p --port=3306 --host=127.0.0.1
Enter password:
Welcome to the MariaDB monitor. Commands end with ; or \g.
Your MariaDB connection id is 8
Server version: 10.3.17-MariaDB MariaDB Server

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> show databases;
+-----+
| Database      |
+-----+
| information_schema |
| items          |
| test           |
+-----+
3 rows in set (0.002 sec)
```

## 6. Stop the Container

```
[student@servera ~]$ podman stop demodb
9a9bf84dcaabf794ed5bc965df38a6ab5ff8856ea5608b46165ad32f48589744
```

## 7. List running containers

```
[student@servera ~]$ podman ps
CONTAINER ID  IMAGE   COMMAND   CREATED   STATUS    PORTS   NAMES
```

## 8. Restart SQL Container

```
[student@servera ~]$ podman restart demodb
9a9bf84dcaabf794ed5bc965df38a6ab5ff8856ea5608b46165ad32f48589744
[student@servera ~]$ podman ps
CONTAINER ID  IMAGE   COMMAND   CREATED   STATUS    PORTS   NAMES
9a9bf84dcaab  registry.lab.example.com/rhel8/mariadb-103:1-102  run-mysqld  6
minutes ago   Up 2 seconds ago  0.0.0.0:3306->3306/tcp  demodb
```

## 9. Verify connectivity and data

```
[student@servera ~]$ podman ps
CONTAINER ID  IMAGE                                     COMMAND
CREATED      STATUS        PORTS          NAMES
9a9bf84dcaab  registry.lab.example.com/rhel8/mariadb-103:1-102  run-mysqld  6
minutes ago   Up 2 seconds ago  0.0.0.0:3306->3306/tcp  demodb
[student@servera ~]$ mysql -u demouser -p --port=3306 --host=127.0.0.1
Enter password:
Welcome to the MariaDB monitor. Commands end with ; or \g.
Your MariaDB connection id is 8
Server version: 10.3.17-MariaDB MariaDB Server

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> show databases;
+-----+
| Database      |
+-----+
| information_schema |
| items          |
| test           |
+-----+
3 rows in set (0.001 sec)
```

## 10. List running containers

```
[student@servera ~]$ podman ps
CONTAINER ID  IMAGE                                     COMMAND
CREATED      STATUS        PORTS          NAMES
9a9bf84dcaab  registry.lab.example.com/rhel8/mariadb-103:1-102  run-mysqld  10
minutes ago   Up 4 minutes ago  0.0.0.0:3306->3306/tcp  demodb
```

## 11. Delete and stop container

```
[student@servera ~]$ podman rm demodb -f
9a9bf84dcaabf794ed5bc965df38a6ab5ff8856ea5608b46165ad32f48589744
```

## 12. Verify container was stopped and removed

```
[student@servera ~]$ podman ps -a
CONTAINER ID  IMAGE                                COMMAND
CREATED      STATUS     PORTS
0d1d688ebbbd  registry.redhat.io/rhel8/httpd-24:latest  /usr/bin/run-http...  3
hours ago    Exited (0) 41 minutes ago  0.0.0.0:8000->8080/tcp  webdemo
```

## A.5. Attaching Persistent Storage to a Container

### A.5.1. Preparing Permanent Storage Locations

Container storage is *ephemeral* meaning any data is gone once the container has been removed. If you want data to be persistent and preserved, then you must provide and mount persistent storage to the container.

### A.5.2. Providing Persistent Storage from the Container Host

The easiest way to provide persistent storage to a container is to use a directory on the container host and mount it to the container for persistent storage. Conceptually, this is similar to adding a remote network volume and mounting it to the host file system. When the container is removed/deleted, files from the persistent storage volume remain on the container host filesystem.

#### A.5.2.1. Preparing the Host Directory

When providing storage to a container, it must be configured so container processes can access the storage. This generally involves:

- Configuring ownership and directory permissions
- Setting appropriate SELinux contexts

The user account within container application must have access to the host directory. Additionally, SELinux context types should be set for **container\_file\_t** as podman uses this SELinux context type to control what files on a host system the container can access. This is very important for applications running as **root** inside a root container. Without the SELinux contexts being provided and setup correctly, it is possible for the container and application to have **root** access on the host system.

#### A.5.2.2. Mounting a Volume

After creating and configuring the host directory, it is possible to mount the directory to the container using **--volume** or **-v** and specifying the **HostDir : ContainerDir:Z** which will automatically apply the SELinux **container\_file\_t** context type to the host directory.

### A.5.3. DEMO - Attaching Persistent Storage to a Container

*Example 43. DEMO - Adding Persistent Storage to a Container*

1. Create a directory for the container storage

```
[student@servera ~]$ mkdir -p webfiles/html
```

2. Place content in the directory for the webserver container

```
[student@servera ~]$ echo "This is a demo website" > webfiles/html/index.html
```

3. Launch a container with the Apache Service

```
[student@servera ~]$ podman run --name webdemo -d -p 8000:8080 -v ~/webfiles:/var/www:Z registry.lab.example.com/rhel8/httpd-24:1-104
```

4. Verify Website is Running

```
[student@servera ~]$ curl http://localhost:8000
This is a demo website
```

5. Stop and Remove the container

```
[student@servera ~]$ podman rm webdemo -f
7c8616095932675d32ba3f7adc0e678375a317c293591b4adb62d7735c176ac7
```

# Appendix B: System Security Policy and Compliance

System security and compliance is a primary concern for people when thinking about the protection of systems and integrity of data. This set of hands-on procedures will focus on obtaining the content and necessary packages to perform a basic scan and remediate the system based on scan results. As part of the lab, you will be customizing your own SCAP content for a scan, view the results, and generate an Ansible playbook based on the failed results.

## B.1. Customizing SCAP Content

Red Hat includes the SCAP Workbench application as a GUI application which allows scanning, customizing, and saving SCAP scans and results. The SCAP Workbench can be used to perform scans of remote systems over an SSH connection or it can be utilized to scan the local system. Since the SCAP Workbench is a GUI application, it must run on a system with X-Windows installed.

### *Servers to Configure*

- workstation

### *Packages to Install*

- scap-workbench



Since we are using an application on a VM that requires a GUI, we can use X11 forwarding with the SSH connection by specifying a **-X** on the command line.

### Step 1 - SSH to Workstation

The first step is to connect to the workstation and forward X11 traffic back to the local system.

#### *Example 44. Connecting to the Workstation VM*

##### *Listing 151. Connecting to Workstation Using SSH*

```
# ssh root@workstation -X
```

### Step 2 - Install Packages

After connecting to the Workstation VM, you will need to install the SCAP Workbench and its dependencies.

*Example 45. Installing SCAP Workbench**Listing 152. Using Yum to Install SCAP Workbench*

```
# yum install scap-workbench

Loaded plugins: langpacks, search-disabled-repos
Resolving Dependencies
--> Running transaction check
---> Package scap-workbench.x86_64 0:1.1.6-1.el7 will be installed
--> Processing Dependency: openscap-utils >= 1.2.0 for package: scap-workbench-1.1.6-
1.el7.x86_64
--> Processing Dependency: scap-security-guide for package: scap-workbench-1.1.6-
1.el7.x86_64
--> Running transaction check
---> Package openscap-utils.x86_64 0:1.2.16-8.el7_5 will be installed
--> Processing Dependency: openscap-containers = 1.2.16-8.el7_5 for package:
openscap-utils-1.2.16-8.el7_5.x86_64
---> Package scap-security-guide.noarch 0:0.1.36-9.el7_5 will be installed
--> Processing Dependency: openscap-scanner >= 1.2.5 for package: scap-security-
guide-0.1.36-9.el7_5.noarch
--> Running transaction check
---> Package openscap-containers.noarch 0:1.2.16-8.el7_5 will be installed
---> Package openscap-scanner.x86_64 0:1.2.16-8.el7_5 will be installed
--> Finished Dependency Resolution
```

Dependencies Resolved

```
=====
Package           Arch    Version        Repository      Size
=====
Installing:
scap-workbench   x86_64  1.1.6-1.el7   rhel--server-dvd  1.8 M
Installing for dependencies:
openscap-containers noarch 1.2.16-8.el7_5  rhel_updates    27 k
openscap-scanner   x86_64  1.2.16-8.el7_5  rhel_updates    61 k
openscap-utils     x86_64  1.2.16-8.el7_5  rhel_updates    27 k
scap-security-guide noarch 0.1.36-9.el7_5  rhel_updates    2.6 M
```

Transaction Summary

```
=====
Install 1 Package (+4 Dependent packages)
```

Total download size: 4.5 M

Installed size: 64 M

Is this ok [y/d/N]:



Some things might already be installed for you, if SCAP Workbench is already installed, please move on to the next step. Also note the dependencies for SCAP Workbench as they are automatically installed.

### Step 3 - Launching SCAP Workbench

In order to run a scan or customize SCAP content, you will need to launch the SCAP Workbench application.



You **must** use SCAP Workbench from a GUI, so it will need to run either locally or through an SSH connection with X11 forwarded.

*Example 46. Launching SCAP Workbench**Listing 153. Using SCAP Workbench*

```
# scap-workbench
```

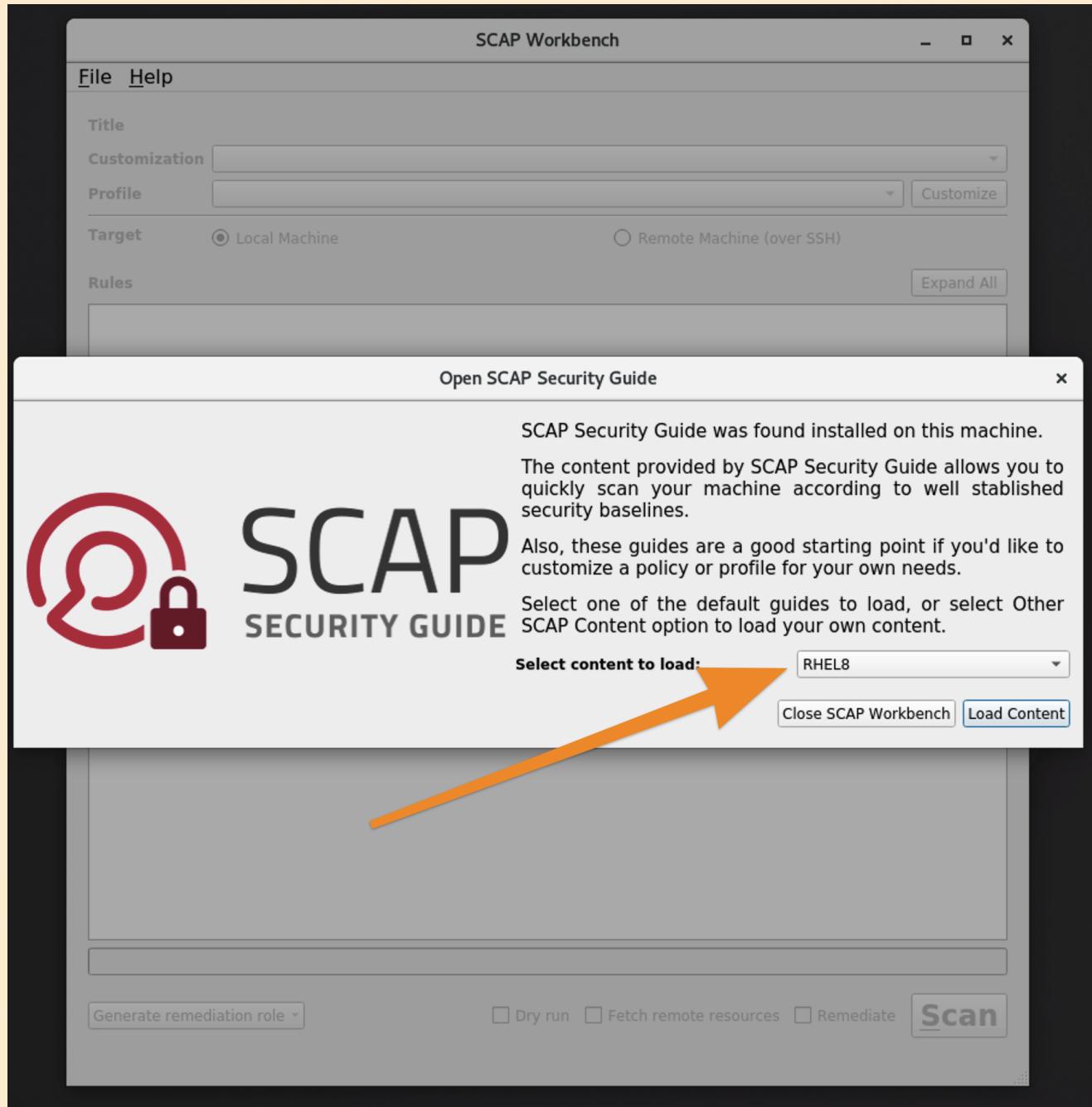


Figure 91. SCAP Workbench Startup

#### Step 4 - Creating Custom Content

Once SCAP Workbench has been launched, select the content to load. For this lab, we will be using the RHEL7 content.

*Example 47. Creating Custom Content*

1. For **Select content to load:** select "RHEL8", then click "**Load Content**"
2. Select the **Profile** you want to use to start customization



For this example, we will use the **OSPP - Protection Profile for General Purpose Operating Systems Baseline**

The screenshot shows the SCAP Workbench interface with the title bar "ssg-rhel8-ds.xml - SCAP Workbench". The main window contains the following configuration settings:

- Title:** Guide to the Secure Configuration of Red Hat Enterprise Linux 8
- Customization:** None selected
- Profile:** OSPP - Protection Profile for General Purpose Operating Systems (187)
- Target:**  Local Machine  Remote Machine (over SSH)

**Rules:** A large list of 187 rules, many of which begin with "Record Unsuccessful ...".

- ▶ Ensure Logs Sent To Remote Host
- ▶ Ensure cron Is Logging To Rsyslog
- ▶ Ensure auditd Collects Information on Kernel Module Loading and Unloading - modprobe
- ▶ Ensure auditd Collects Information on Kernel Module Loading - init\_module
- ▶ Ensure auditd Collects Information on Kernel Module Unloading - delete\_module
- ▶ Ensure auditd Collects Information on Kernel Module Loading - insmod
- ▶ Ensure auditd Collects Information on Kernel Module Unloading - rmmod
- ▶ Record Unsuccessful Permission Changes to Files - chmod
- ▶ Record Unsuccessful Ownership Changes to Files - chown
- ▶ Record Unsuccessful Permission Changes to Files - removexattr
- ▶ Record Unauthorized Access Attempts to Files (unsuccessful) - ftruncate
- ▶ Record Unsuccessful Ownership Changes to Files - lchown
- ▶ Record Unsuccessful Permission Changes to Files - fremovexattr
- ▶ Record Unauthorized Modification Attempts to Files - open O\_TRUNC
- ▶ Record Unsuccessful Ownership Changes to Files - fchown
- ▶ Record Unauthorized Modification Attempts to Files - openat O\_TRUNC
- ▶ Record Unsuccessful Permission Changes to Files - fsetxattr
- ▶ Record Unsuccessful Ownership Changes to Files - fchownat
- ▶ Record Unauthorized Creation Attempts to Files - open\_by\_handle\_at O\_CREAT
- ▶ Record Unauthorized Access Attempts to Files (unsuccessful) - creat
- ▶ Record Unauthorized Creation Attempts to Files - open O\_CREAT

At the bottom, there is a progress bar showing "0% (0 results, 187 rules selected)" and a set of buttons: "Generate remediation role", "Dry run", "Fetch remote resources", "Remediate", and a large "Scan" button.

Figure 92. SCAP Workbench OSPP Profile

3. Click "Customize" to create custom SCAP content based on the chosen profile, and give it a name.

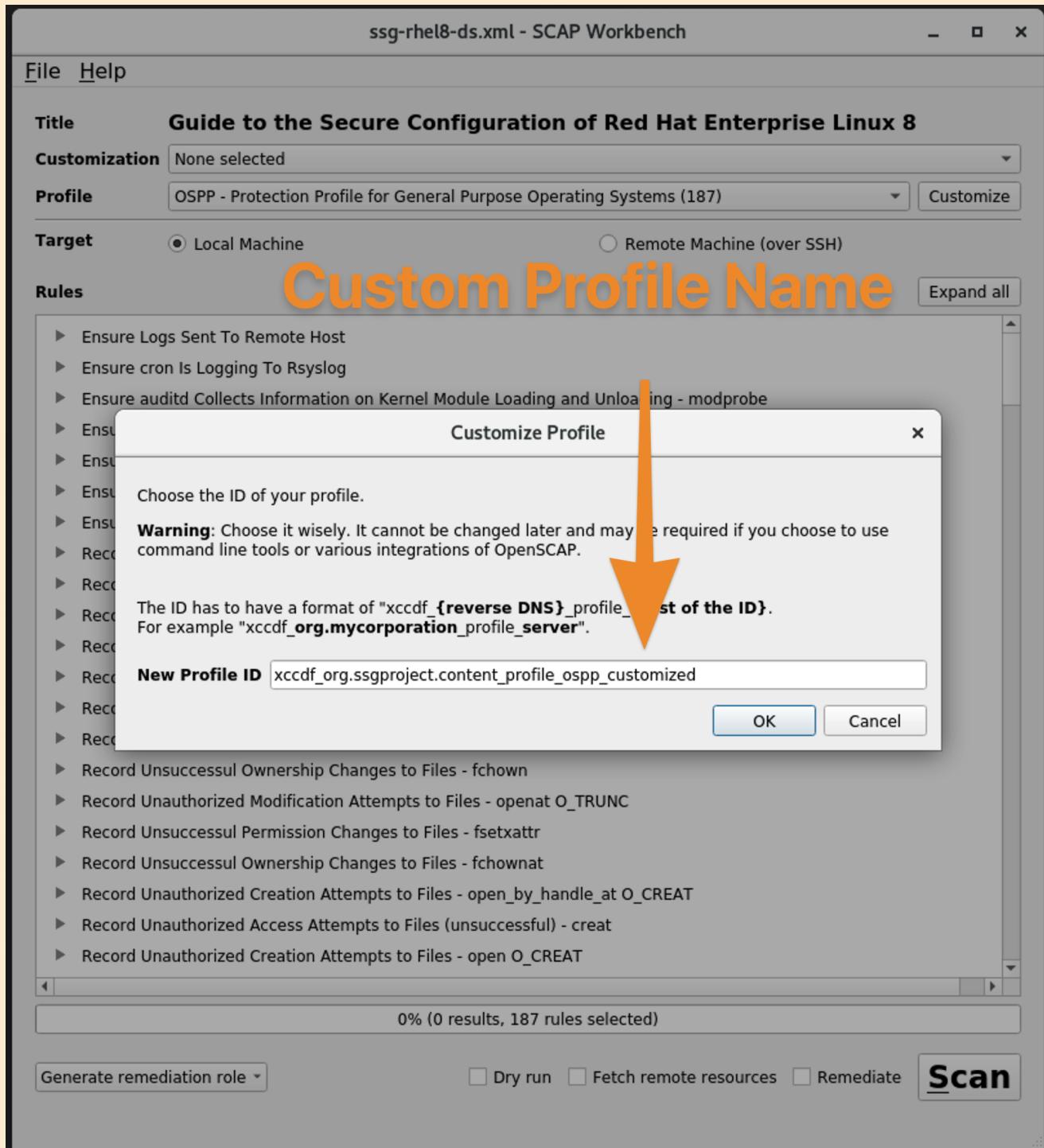


Figure 93. SCAP Custom OSPP Profile Creation



The name for this is: **xccdf\_org.ssgproject.content\_profile\_ospp\_customized**

4. Click "Deselect All" so that you can select the items you wish to include in your custom scan profile. NOTE: we are doing this to also limit it to a few checks for the example.

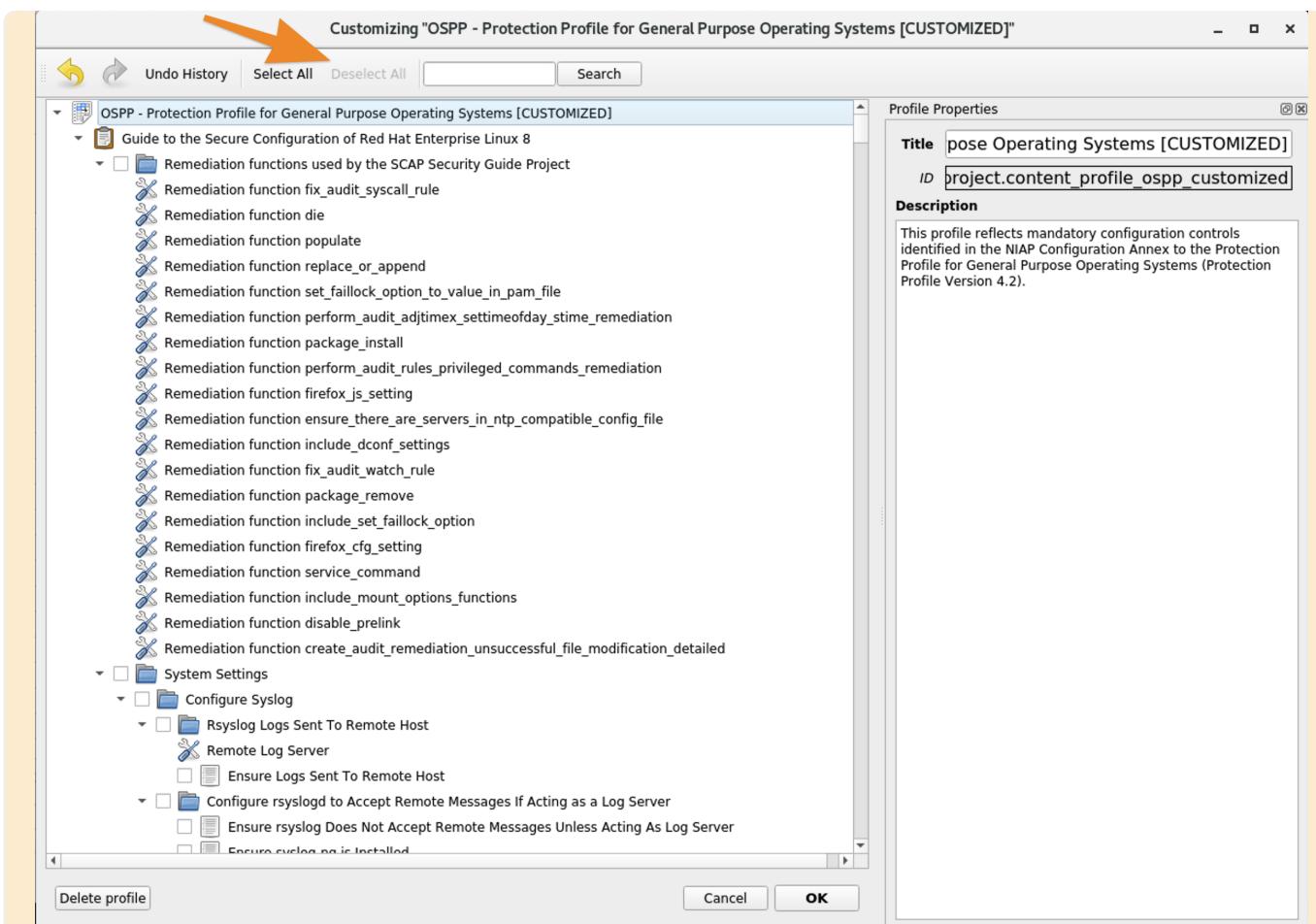


Figure 94. SCAP Custom Profile Selections



For this lab, we will be setting the minimum password length and PAM Password quality settings

5. Search for Password to set **minimum password length** and set the values in **login.defs**. Check **Set Password Minimum Length** in **login.defs** and click on the **minimum password length** and set the value to **18**

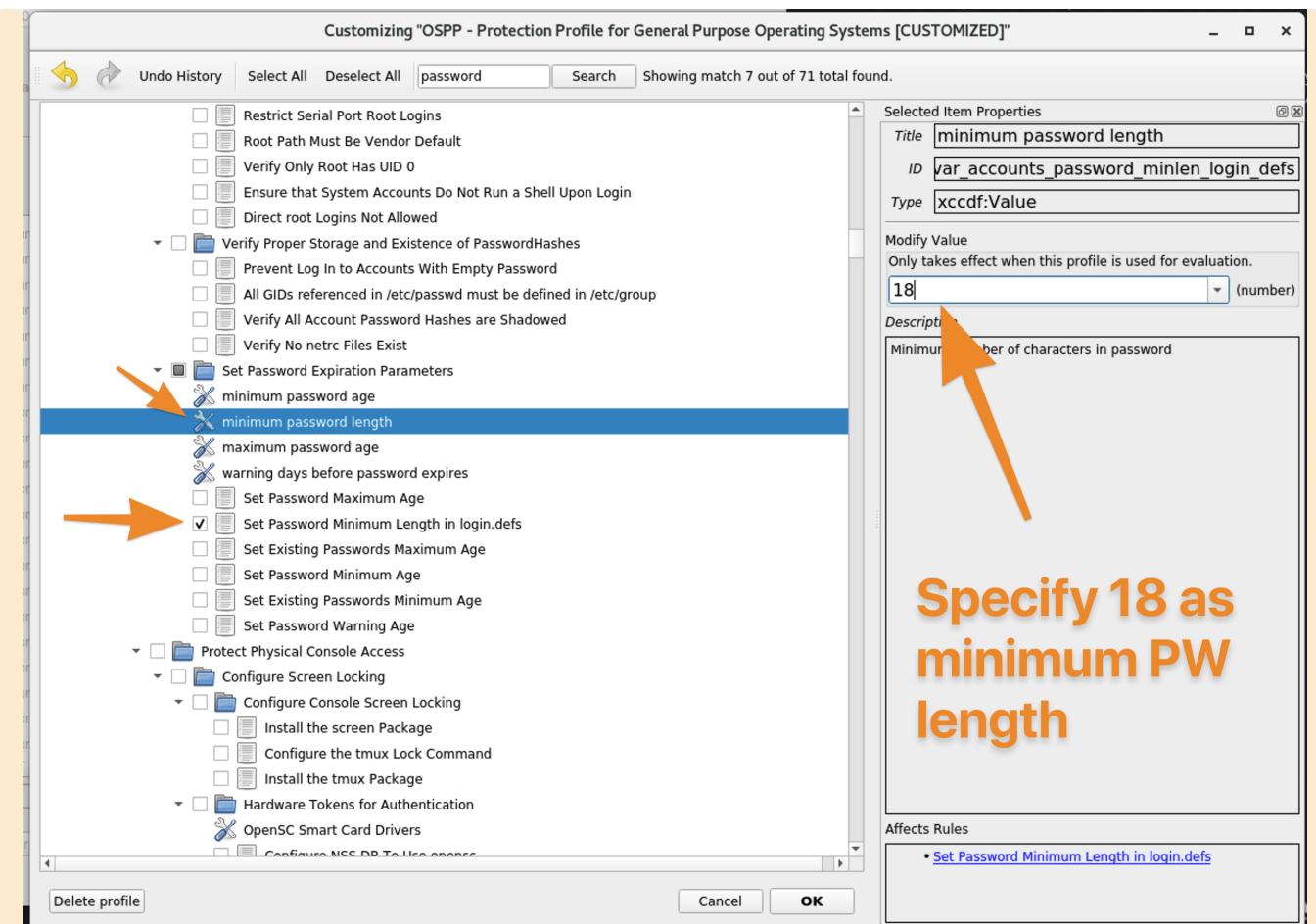


Figure 95. SCAP Custom Profile Password Settings for Login.Defs

6. Set password quality requirements with PAM. Search for the minlen and set it to 18. Also, place a checkbox in **Set Password Quality Requirements with pam\_quality**. Then click "OK"

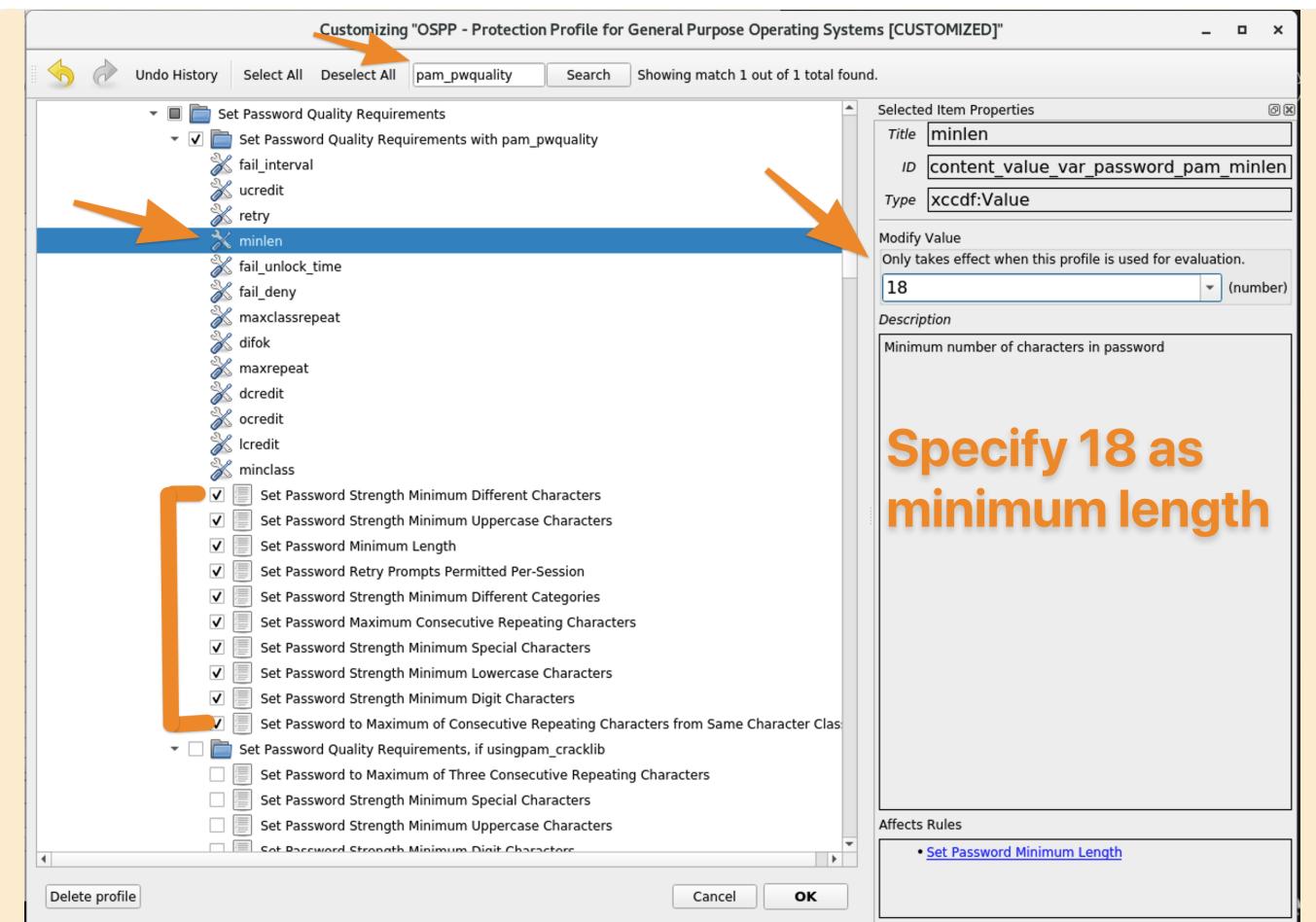
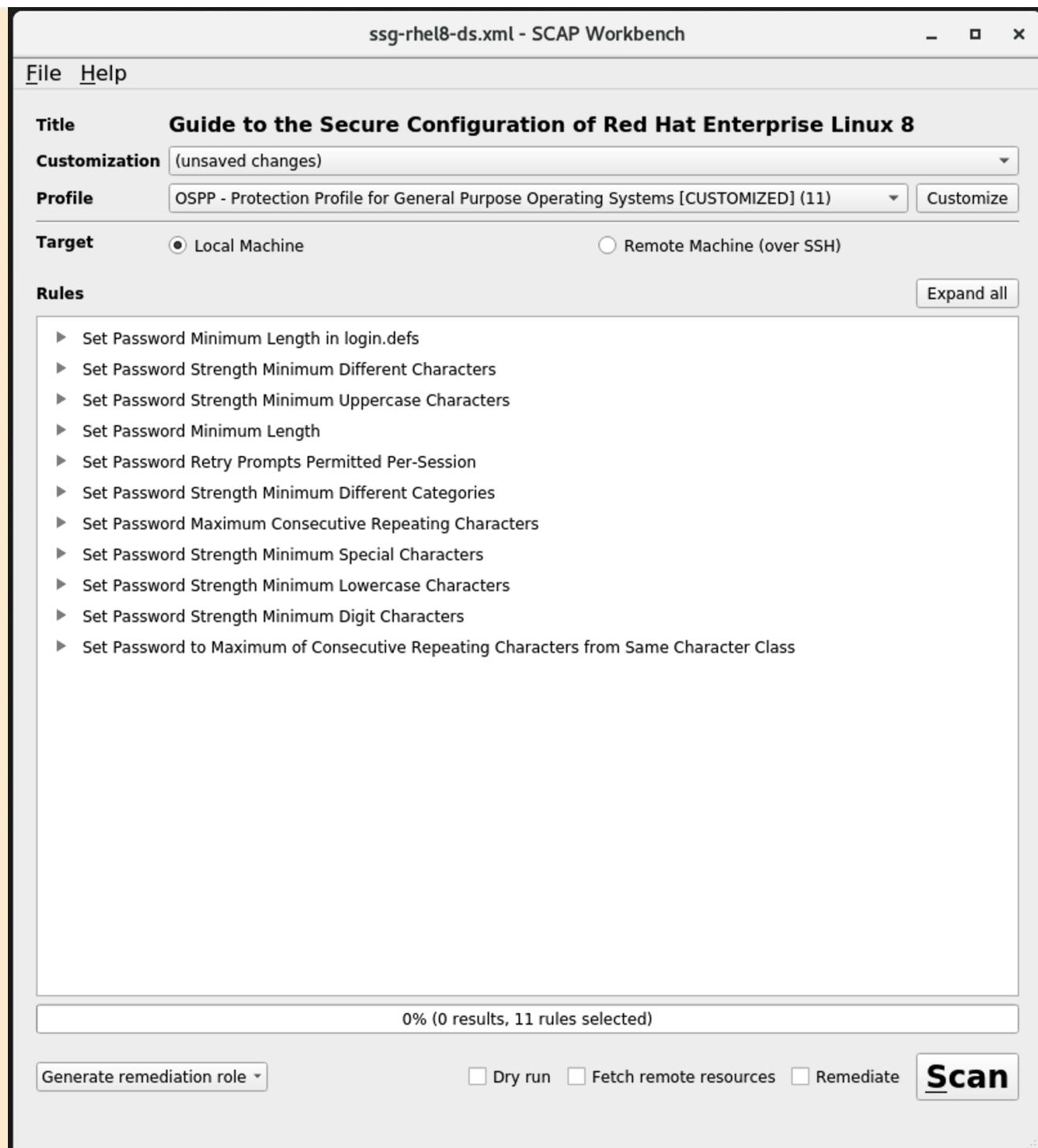


Figure 96. SCAP Custom Profile PAM Quality Requirements

7. At this point, we have taken the default settings from the OSPP profile with only the tailored pieces that we selected. The next step is to click "File ⇒ Save Customization Only" to save the custom content



The screenshot shows the SCAP Workbench interface with the title "ssg-rhel8-ds.xml - SCAP Workbench". The main area displays a list of selected rules under the heading "Guide to the Secure Configuration of Red Hat Enterprise Linux 8". The rules listed are:

- ▶ Set Password Minimum Length in login.defs
- ▶ Set Password Strength Minimum Different Characters
- ▶ Set Password Strength Minimum Uppercase Characters
- ▶ Set Password Minimum Length
- ▶ Set Password Retry Prompts Permitted Per-Session
- ▶ Set Password Strength Minimum Different Categories
- ▶ Set Password Maximum Consecutive Repeating Characters
- ▶ Set Password Strength Minimum Special Characters
- ▶ Set Password Strength Minimum Lowercase Characters
- ▶ Set Password Strength Minimum Digit Characters
- ▶ Set Password to Maximum of Consecutive Repeating Characters from Same Character Class

At the bottom, there is a status bar showing "0% (0 results, 11 rules selected)". Below the status bar are three checkboxes: "Dry run", "Fetch remote resources", and "Remediate". To the right of these checkboxes is a large "Scan" button.

Figure 97. SCAP Custom Profile Selected Settings View

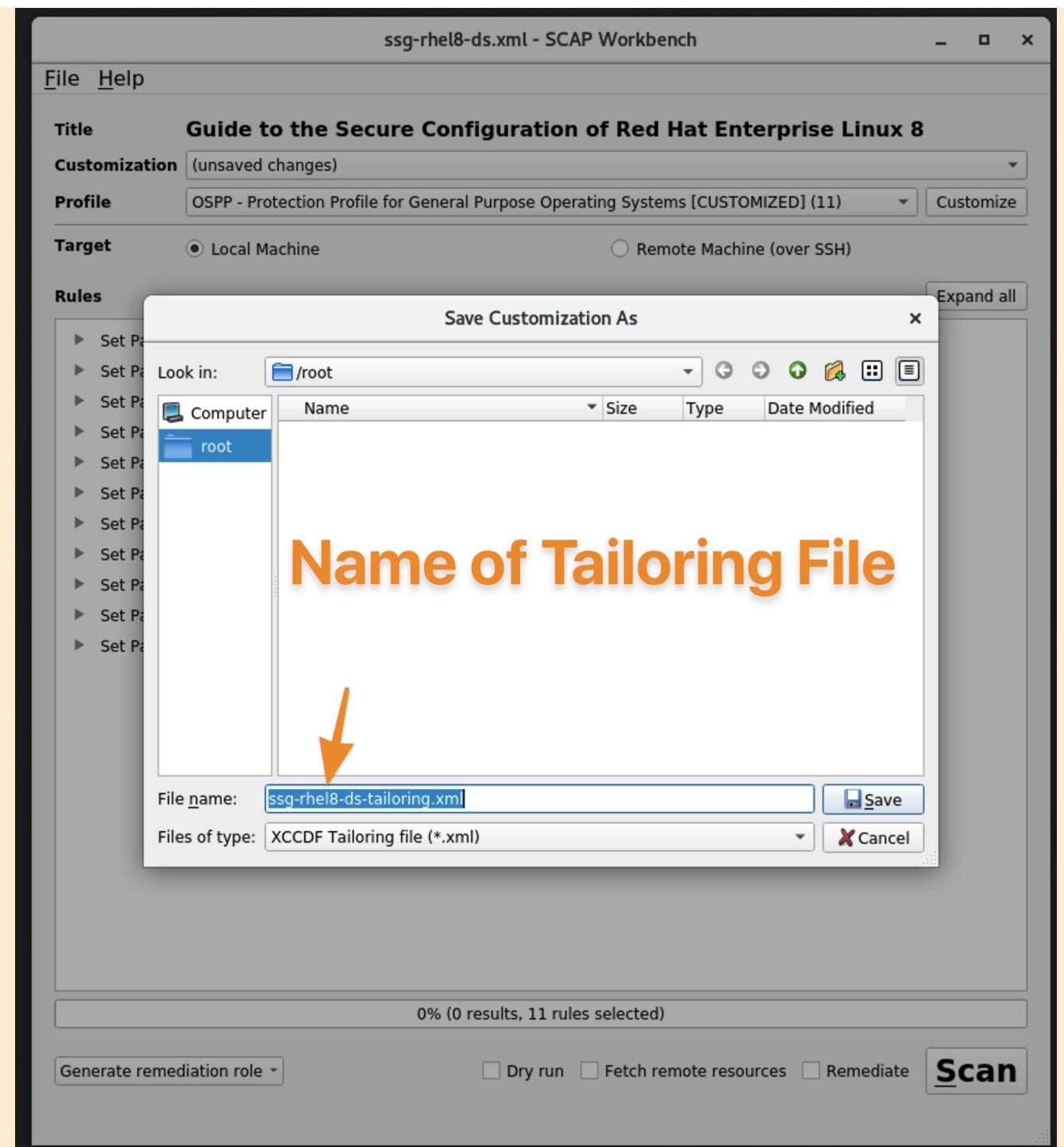


Figure 98. SCAP Custom Profile Creation Saving

ssg-rhel8-ds.xml - SCAP Workbench

[File](#) [Help](#)

**Title** **Guide to the Secure Configuration of Red Hat Enterprise Linux 8**

**Customization** /root/ssg-rhel8-ds-tailoring.xml

**Profile** OSPP - Protection Profile for General Purpose Operating Systems [CUSTOMIZED] (11) [Customize](#)

**Target**  Local Machine  Remote Machine (over SSH)

**Rules** [Expand all](#)

- ▶ Set Password Minimum Length in login.defs
- ▶ Set Password Strength Minimum Different Characters
- ▶ Set Password Strength Minimum Uppercase Characters
- ▶ Set Password Minimum Length
- ▶ Set Password Retry Prompts Permitted Per-Session
- ▶ Set Password Strength Minimum Different Categories
- ▶ Set Password Maximum Consecutive Repeating Characters
- ▶ Set Password Strength Minimum Special Characters
- ▶ Set Password Strength Minimum Lowercase Characters
- ▶ Set Password Strength Minimum Digit Characters
- ▶ Set Password to Maximum of Consecutive Repeating Characters from Same Character Class

0% (0 results, 11 rules selected)

[Generate remediation role](#)  Dry run  Fetch remote resources  Remediate **Scan**

Figure 99. SCAP Custom Profile Final View

8. Copy the custom tailoring file to the server(s) being scanned. In this case, we will want to copy the file to **servera**

*Listing 154. Copy custom content*

```
[root@workstation ~]# scp ssg-rhel8-ds-tailoring.xml root@servera:  
ssg-rhel8-ds-tailoring.xml                                100%   28KB  10.7MB/s  00:00  
[root@workstation ~]#
```

## B.2. Running a SCAP Scan with Custom Content

### *Servers to Configure*

- servera

### *Packages to Install*

- openscap-scanner
- scap-security-guide

### **Step 1 - SSH to servera**

The first step is to connect to the server.

*Example 48. Connecting to the servera VM*

*Listing 155. Connecting to servera Using SSH*

```
# ssh root@servera
```

### **Step 2 - Install packages on servera**

The second step is to install software on the server.

**Example 49. Install software on servera*****Listing 156. Installing Software on servera***

```
[root@servera ~]# yum install scap-security-guide
Last metadata expiration check: 0:47:44 ago on Thu 16 Apr 2020 08:26:15 AM EDT.
Dependencies resolved.
=====
Package           Arch    Version      Repository      Size
=====
Installing:
scap-security-guide          noarch 0.1.42-11.el8  rhel-8.0-for-x86_64-appstream-rpms 3.4 M
Installing dependencies:
openscap          x86_64  1.3.0-7.el8   rhel-8.0-for-x86_64-appstream-rpms 3.3 M
openscap-scanner  x86_64  1.3.0-7.el8   rhel-8.0-for-x86_64-appstream-rpms 66 k
xml-common        noarch 0.6.3-50.el8  rhel-8.0-for-x86_64-baseos-rpms   39 k

Transaction Summary
=====
Install 4 Packages

Total download size: 6.9 M
Installed size: 132 M
Is this ok [y/N]: y

... output omitted ...

Verifying : openscap-1.3.0-7.el8.x86_64                                1/4
Verifying : openscap-scanner-1.3.0-7.el8.x86_64                            2/4
Verifying : scap-security-guide-0.1.42-11.el8.noarch                         3/4
Verifying : xml-common-0.6.3-50.el8.noarch                                4/4

Installed:
scap-security-guide-0.1.42-11.el8.noarch      openscap-1.3.0-7.el8.x86_64
openscap-scanner-1.3.0-7.el8.x86_64          xml-common-0.6.3-50.el8.noarch

Complete!
```

**Learning about SCAP Commands**

The SSG man page is a very good source of information for usage of the **oscap** tool as well as provides examples of how to use the SCAP SSG Guide profiles itself.



*Listing 157. Looking at SCAP Security Guide (SSG) Man Page*

```
# man scap-security-guide

scap-security-guide(8)      System Manager's Manual      scap-security-
guide(8)

NAME
    SCAP Security Guide - Delivers security guidance, baselines, and
associated validation mechanisms utilizing the Security Content
Automation
Protocol (SCAP).

... output omitted ...

EXAMPLES
    To scan your system utilizing the OpenSCAP utility against the
ospp-
rhel7 profile:

        oscap xccdf eval --profile ospp-rhel7 --results /tmp/'hostname'-
ssg-
        results.xml --report /tmp/'hostname'-ssg-results.html --oval
-results
        /usr/share/xml/scap/ssg/content/ssg-rhel7-xccdf.xml
```

*Listing 158. Looking at oscap Man Page*

```
# man oscap

OSCAP(8)           System Administration Utilities
OSCAP(8)

NAME
    oscap - OpenSCAP command line tool

SYNOPSIS
    oscap [general-options] module operation [operation-options-and-
arguments]

DESCRIPTION
    oscap is Security Content Automation Protocol (SCAP) toolkit based
on
    OpenSCAP library. It provides various functions for different
SCAP
specifications (modules).

    OpenSCAP tool claims to provide capabilities of Authenticated
Configuration Scanner and Authenticated Vulnerability Scanner as
defined by
    The National Institute of Standards and Technology.

... output omitted ...

EXAMPLES
    Evaluate XCCDF content using CPE dictionary and produce html
report. In
        this case we use United States Government Configuration
Baseline
        (USGCB) for Red Hat Enterprise Linux 5 Desktop.

        oscap xccdf eval --fetch-remote-resources --oval-results \
                    --profile
united_states_government_configuration_baseline
 \
                    --report usgcb-rhel5desktop.report.html \
                    --results usgcb-rhel5desktop-xccdf.xml.result.xml \
                    --cpe usgcb-rhel5desktop-cpe-dictionary.xml \
                    usgcb-rhel5desktop-xccdf.xml
```

**Step 3 - Running oscap scan**

We will run the **oscap** utility to generate a report and a results file that can be sent back to the **workstation** system so that we can create an Ansible playbook for remediation and view the results of the report.



Be very careful about the name of the profile as this was selected during the creation of the custom profile/tailoring file portion when doing SCAP Workbench customizations.

**Example 50. Scanning servera****Listing 159. Using oscap and the tailoring profile to scan servera**

```
# [root@servera ~]# oscap xccdf eval \
--profile xccdf_org.ssgproject.content_profile_ospp_customized \
--tailoring-file ssg-rhel8-ds-tailoring.xml \
--results custom_scan_results.xml \
/usr/share/xml/scap/ssg/content/ssg-rhel8-ds.xml

Title  Set Password Minimum Length in login.defs
Rule   xccdf_org.ssgproject.content_rule_accounts_password_minlen_login_defs
Ident  CCE-80652-1
Result fail

Title  Set Password Strength Minimum Different Characters
Rule   xccdf_org.ssgproject.content_rule_accounts_password_pam_difok
Ident  CCE-80654-7
Result fail

Title  Set Password Strength Minimum Uppercase Characters
Rule   xccdf_org.ssgproject.content_rule_accounts_password_pam_ucredit
Ident  CCE-80665-3
Result fail

Title  Set Password Minimum Length
Rule   xccdf_org.ssgproject.content_rule_accounts_password_pam_minlen
Ident  CCE-80656-2
Result fail

Title  Set Password Retry Prompts Permitted Per-Session
Rule   xccdf_org.ssgproject.content_rule_accounts_password_pam_retry
Ident  CCE-80664-6
Result fail

Title  Set Password Strength Minimum Different Categories
Rule   xccdf_org.ssgproject.content_rule_accounts_password_pam_minclass
Result fail

Title  Set Password Maximum Consecutive Repeating Characters
Rule   xccdf_org.ssgproject.content_rule_accounts_password_pam_maxrepeat
Result fail

Title  Set Password Strength Minimum Special Characters
Rule   xccdf_org.ssgproject.content_rule_accounts_password_pam_ocredit
Ident  CCE-80663-8
Result fail
```

```
Title Set Password Strength Minimum Lowercase Characters
Rule xccdf_org.ssgproject.content_rule_accounts_password_pam_lcredit
Ident CCE-80655-4
Result fail
```

```
Title Set Password Strength Minimum Digit Characters
Rule xccdf_org.ssgproject.content_rule_accounts_password_pam_dcredit
Ident CCE-80653-9
Result fail
```

```
Title Set Password to Maximum of Consecutive Repeating Characters from Same
Character Class
Rule xccdf_org.ssgproject.content_rule_accounts_password_pam_maxclassrepeat
Result fail
```

### Getting Custom Profile Name from Tailoring File

If you need to locate the profile used for the custom scanning content from the tailoring file, you can search for it with `grep`.



```
[root@servera ~]# grep "Profile id" ssg-rhel8-ds-tailoring.xml
<xccdf:Profile
  id="xccdf_org.ssgproject.content_profile_ospp_customized"
  extends="xccdf_org.ssgproject.content_profile_ospp">
```

## Step 4 - Creating a Results Report

You can create a results report file from the results file so you have a nice HTML file that is easy to ready with the results from the SCAP scan.

*Example 51. Creating a SCAP Report from a Results File*

### *Listing 160. Generating a Report*

```
[root@servera ~]# oscap xccdf generate report \
custom_scan_results.xml > Custom_Scan_Report.html
```

### *Combining Steps 3 & 4*

 It is possible to perform a custom content scan which will generate the results file and the report for transfer back to the workstation for review.

### **Need to Specify**

- **--results**

- **--report**

*Listing 161. Creating a Results File and Report During Custom Content Scan*

```
[root@servera ~]# oscap xccdf eval \
--profile xccdf_org.ssgproject.content_profile_ospp_customized \
--tailoring-file ssg-rhel8-ds-tailoring.xml \
--results custom_scan_results_2.xml \
--report Custom_Scan_Report_2.html \
/usr/share/xml/scap/ssg/content/ssg-rhel8-ds.xml

Title  Set Password Minimum Length in login.defs
Rule   xccdf_org.ssgproject.content_rule_accounts_password_minlen_login_defs
Ident  CCE-80652-1
Result fail

Title  Set Password Strength Minimum Different Characters
Rule   xccdf_org.ssgproject.content_rule_accounts_password_pam_difok
Ident  CCE-80654-7
Result fail

Title  Set Password Strength Minimum Uppercase Characters
Rule   xccdf_org.ssgproject.content_rule_accounts_password_pam_ucredit
Ident  CCE-80665-3
Result fail

Title  Set Password Minimum Length
Rule   xccdf_org.ssgproject.content_rule_accounts_password_pam_minlen
Ident  CCE-80656-2
Result fail

Title  Set Password Retry Prompts Per-Session
Rule   xccdf_org.ssgproject.content_rule_accounts_password_pam_retry
Ident  CCE-80664-6
Result fail

Title  Set Password Strength Minimum Different Categories
Rule   xccdf_org.ssgproject.content_rule_accounts_password_pam_minclass
Result fail

Title  Set Password Maximum Consecutive Repeating Characters
Rule   xccdf_org.ssgproject.content_rule_accounts_password_pam_maxrepeat
Result fail

Title  Set Password Strength Minimum Special Characters
```

```
Rule    xccdf_org.ssgproject.content_rule_accounts_password_pam_ocredit
Ident   CCE-80663-8
Result  fail

Title   Set Password Strength Minimum Lowercase Characters
Rule    xccdf_org.ssgproject.content_rule_accounts_password_pam_lccredit
Ident   CCE-80655-4
Result  fail

Title   Set Password Strength Minimum Digit Characters
Rule    xccdf_org.ssgproject.content_rule_accounts_password_pam_dccredit
Ident   CCE-80653-9
Result  fail

Title   Set Password to Maximum of Consecutive Repeating Characters from
Same Character Class
Rule
xccdf_org.ssgproject.content_rule_accounts_password_pam_maxclassrepeat
Result  fail
```

## Step 5 - Transferring Results File and Report to Workstation

After you have the results files and the report, you should transfer it to your graphical workstation (**workstation**) for further analysis.

*Example 52. Transferring Results*

*Listing 162. Transferring the Results and Report Files*

```
[root@servera ~]# scp *.xml *.html root@workstation:
The authenticity of host 'workstation (<no hostip for proxy command>)' can't be
established.
ECDSA key fingerprint is SHA256:p0Q10JmyF2PFI+jxyFoOSCfi+1oWNsUruy2DZNjg+N0.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'workstation' (ECDSA) to the list of known hosts.
root@workstation's password:
custom_scan_results_2.xml          100% 4086KB 32.4MB/s  00:00
custom_scan_results.xml            100% 4086KB 60.0MB/s  00:00
ssg-rhel8-ds-tailoring.xml        100%   28KB 16.2MB/s  00:00
Custom_Scan_Report_2.html         100%  332KB 44.3MB/s  00:00
Custom_Scan_Report.html           100%  332KB 37.6MB/s  00:00
[root@servera ~]#
```

## Step 6 - Viewing the SCAP scan report

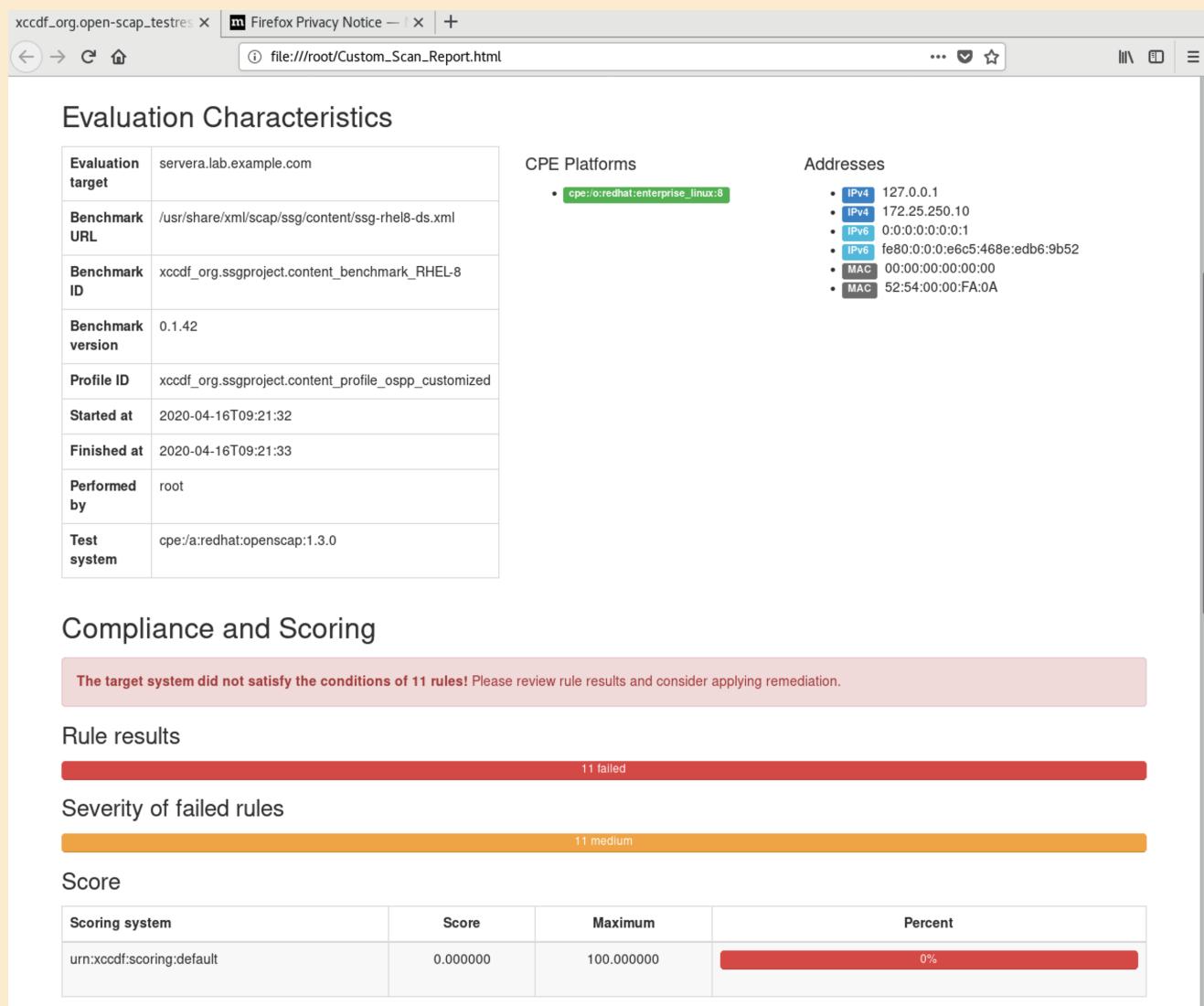
After you have transferred the results file to **workstation** you can open the HTML report in a web

browser. In this case we will use **firefox** to open the file.

#### *Example 53. Viewing the SCAP Report*

*Listing 163. Opening the SCAP HTML Report with Firefox*

```
[root@workstation ~]# firefox Custom_Scan_Report.html
```



*Figure 100. SCAP Scan Results Report in Firefox*



Firefox may not open the file based on SELinux context triggers. In order to get around this you can use the command prompt and do **setenforce 0** to allow you to open the report.

## B.3. Creating an Ansible Remediation Playbook Based on SCAP Scan Results

The OpenSCAP project and content created by Red Hat can automatically remediate findings from OpenSCAP scans. The findings can be remediated in many ways (**BASH**, **Ansible**, etc.). While things are mostly complete, there are some automated remediations that have not yet been developed.



There are multiple automatic remediation methods developed, but at this time, there isn't a script to fix everything.

### *Servers to Configure*

- servera



We will continue to use **workstation** as our master SCAP system as it should have Ansible and SCAP Workbench installed.

### **Step 1 - Creating an Ansible Playbook from Results**

The first step will be to generate an Ansible playbook from the SCAP scan results for system remediation.

### Example 54. Generating Ansible Playbook

#### *Listing 164. Ansible Playbook Generation*

```
[root@workstation ~]# oscap xccdf generate fix \
--profile xccdf_org.ssgproject.content_profile_ospp_customized \
--tailoring-file ssg-rhel8-ds-tailoring.xml \
--fix-type ansible \
--result-id "" \
custom_scan_results.xml > Custom_Scan_Fix.yml
```

#### Viewing Remediation Playbook

It is also a good idea to view the created playbook for the system prior to running it.

```
[root@workstation ~]# cat Custom_Scan_Fix.yml
---
#####
#####
#
# Ansible remediation role for the results of evaluation of profile
xccdf_org.ssgproject.content_profile_ospp_customized
# XCCDF Version: unknown
#
# Evaluation Start Time: 2020-04-16T09:21:32
# Evaluation End Time: 2020-04-16T09:21:33
#
# This file was generated by OpenSCAP 1.3.0 using:
# $ oscap xccdf generate fix --result-id xccdf_org.openscap_testresult_xccdf_org.ssgproject.content_profile_ospp_customized
--template urn:xccdf:fix:script:ansible xccdf-results.xml
#
# This script is generated from the results of a profile evaluation.
# It attempts to remediate all issues from the selected rules that
failed the test.
#
# How to apply this remediation role:
# $ ansible-playbook -i "localhost," -c local playbook.yml
# $ ansible-playbook -i "192.168.1.155," playbook.yml
# $ ansible-playbook -i inventory.ini playbook.yml
#
#####
#####

- hosts: all
```



```
vars:
    var_accounts_password_minlen_login_defs: !!str 18
    var_password_pam_difok: !!str 8
    var_password_pam_ucredit: !!str -1
    var_password_pam_minlen: !!str 18
    var_password_pam_retry: !!str 3
    var_password_pam_minclass: !!str 3
    var_password_pam_maxrepeat: !!str 3
    var_password_pam_ocredit: !!str -1
    var_password_pam_lcredit: !!str -1
    var_password_pam_dcredit: !!str -1
    var_password_pam_maxclassrepeat: !!str 4

tasks:

- name: "Set Password Minimum Length in login.defs"
  lineinfile:
    dest: /etc/login.defs
    regexp: "^\$PASS_MIN_LEN *[0-9]*"
    state: present
    line: "PASS_MIN_LEN      {{ var_accounts_password_minlen_login_defs }}"
  tags:
    - accounts_password_minlen_login_defs
    - medium_severity
    - restrict_strategy
    - low_complexity
    - low_disruption
    - CCE-80652-1
    - NIST-800-53-IA-5(f)
    - NIST-800-53-IA-5(1)(a)
    - NIST-800-171-3.5.7
    - CJIS-5.6.2.1

...
... Output Omitted ...

- name: Ensure PAM variable maxclassrepeat is set accordingly
  lineinfile:
    create: yes
    dest: "/etc/security/pwquality.conf"
    regexp: '^\$#?\s*maxclassrepeat'
    line: "maxclassrepeat = {{ var_password_pam_maxclassrepeat }}"

  tags:
    - accounts_password_pam_maxclassrepeat
    - medium_severity
```

- restrict\_strategy
- low\_complexity
- low\_disruption
- NIST-800-53-IA-5
- NIST-800-53-IA-5(c)

### Ansible is not setup for the lab

Before we can do the next steps, we will download an Ansible config file and an inventory file so we can properly run the playbook.

*Listing 165. Error Output Message*

```
[root@workstation ~]# ansible-playbook Custom_Scan_Fix.yml  
[WARNING]: provided hosts list is empty, only localhost is available.
```

Note

that the implicit localhost does not match 'all'



```
PLAY [all]  
*****  
skipping: no hosts matched  
  
PLAY RECAP  
*****  
[root@workstation ~]#
```

### Step 2 - Downloading Ansible Config and Ansible Inventory Files

This step is needed so that our Ansible system can be configured with various configuration options and the inventory files so we can run the given playbook.

*Example 55. Downloading Ansible Files**Listing 166. Downloading Ansible Files*

```
[root@workstation ~]# wget http://people.redhat.com/~tmichett/rh354/inventory
--2020-04-16 09:38:50-- http://people.redhat.com/~tmichett/rh354/inventory
Resolving people.redhat.com (people.redhat.com)... 209.132.183.19
Connecting to people.redhat.com (people.redhat.com)|209.132.183.19|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 24
Saving to: "inventory"

inventory          100%[=====]      24  --.-KB/s   in 0s

2020-04-16 09:38:50 (2.69 MB/s) - "inventory" saved [24/24]

[root@workstation ~]# wget http://people.redhat.com/~tmichett/rh354/ansible.cfg
--2020-04-16 09:39:34-- http://people.redhat.com/~tmichett/rh354/ansible.cfg
Resolving people.redhat.com (people.redhat.com)... 209.132.183.19
Connecting to people.redhat.com (people.redhat.com)|209.132.183.19|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 159
Saving to: "ansible.cfg"

ansible.cfg        100%[=====]      159  --.-KB/s   in 0s

2020-04-16 09:39:35 (13.8 MB/s) - "ansible.cfg" saved [159/159]
```

### *Reviewing Ansible Configurations*

The **inventory** file provided only has a single host **servera** in there. On real systems, you must be very cautious of running remediation playbooks against an inventory file as it could apply to unintended systems. Additionally the **ansible.cfg** file provided was created for use in this lab environment. Both of these items should be taken into account when doing going through the process on production systems.



```
[root@workstation ~]# cat inventory
servera.lab.example.com

[root@workstation ~]# cat ansible.cfg
[defaults]
roles_path = /etc/ansible/roles:/usr/share/ansible/roles
log_path   = /tmp/ansible.log
inventory  = ./inventory

[privilegeEscalation]
become=True
[root@workstation ~]#
```

### **Step 3 - Run the Ansible Playbook**

This step will utilize the **workstation** system which is configured as your Ansible management node and will run the playbook to remediate the results on the **servera** system.

*Example 56. Remediation of **serverc** with Ansible Playbook*

*Listing 167. Running the Ansible Playbook*

```
[root@workstation ~]# ansible-playbook Custom_Scan_Fix.yml

PLAY [all] ****
TASK [Gathering Facts] ****
ok: [servera.lab.example.com]

TASK [Set Password Minimum Length in login.defs] ****
changed: [servera.lab.example.com]

TASK [Ensure PAM variable difok is set accordingly] ****
changed: [servera.lab.example.com]

TASK [Ensure PAM variable ucredit is set accordingly] ****
changed: [servera.lab.example.com]

TASK [Ensure PAM variable minlen is set accordingly] ****
changed: [servera.lab.example.com]

TASK [Set Password Retry Prompts Per-Session - system-auth (change)] ***
ok: [servera.lab.example.com]

TASK [Set Password Retry Prompts Per-Session - system-auth (add)] ****
changed: [servera.lab.example.com]

TASK [Ensure PAM variable minclass is set accordingly] ****
changed: [servera.lab.example.com]

TASK [Ensure PAM variable maxrepeat is set accordingly] ****
changed: [servera.lab.example.com]

TASK [Ensure PAM variable ocredit is set accordingly] ****
changed: [servera.lab.example.com]

TASK [Ensure PAM variable lcredit is set accordingly] ****
changed: [servera.lab.example.com]

TASK [Ensure PAM variable dcredit is set accordingly] ****
changed: [servera.lab.example.com]

TASK [Ensure PAM variable maxclassrepeat is set accordingly] ****
changed: [servera.lab.example.com]

PLAY RECAP ****
servera.lab.example.com      : ok=13    changed=11    unreachable=0    failed=0
skipped=0      rescued=0    ignored=0
```



After running the playbook, you can see that there were 10 changes that were made to the system and exactly which parameters were changed. The next thing to do is perform another scan of the system to ensure that it is now fully compliant.

#### Step 4 - Rescan System and Review Results

*Example 57. Scanning System after Fixes and Verifying Results**Listing 168. Performing SCAP Verification Scan*

```
[root@servera ~]# oscap xccdf eval \
--profile xccdf_org.ssgproject.content_profile_ospp_customized \
--tailoring-file ssg-rhel8-ds-tailoring.xml \
--results custom_scan_results_fixed.xml \
--report Custom_Scan_Report_Fixed.html \
/usr/share/xml/scap/ssg/content/ssg-rhel8-ds.xml
```

*Listing 169. Copying Results to Workstation*

```
[root@servera ~]# scp custom_scan_results_fixed.xml Custom_Scan_Report_Fixed.html
workstation:
root@workstation's password:
custom_scan_results_fixed.xml          100% 4086KB 60.3MB/s 00:00
Custom_Scan_Report_Fixed.html           100% 282KB 48.4MB/s 00:00
```

*Listing 170. Viewing Results on Workstation*

```
[root@workstation ~]# firefox Custom_Scan_Report_Fixed.html
```

The screenshot shows a Firefox browser window with the title "xccdf\_org.open-scap\_testres" and the URL "file:///root/Custom\_Scan\_Report\_Fixed.html". The page content is as follows:

**Performed by:** root  
**Test system:** cpe:/a:redhat:openscap:1.3.0

## Compliance and Scoring

There were no failed or uncertain rules. It seems that no action is necessary.

### Rule results

11 passed

### Severity of failed rules

(No data shown)

### Score

Scoring system	Score	Maximum	Percent
urn:xccdf:scoring:default	100.000000	100.000000	100%

## Rule Overview

Filter options:  pass,  fail,  notchecked,  fixed,  error,  notapplicable,  informational,  unknown

Search through XCCDF rules:  Search

Group rules by: Default

Title	Severity	Result
► Guide to the Secure Configuration of Red Hat Enterprise Linux 8		

Figure 101. Fixed SCAP Scan Results Report in Firefox