Table of Contents

RHEL 8 Readiness Training

Container Tools

Course: CEE-RL-021
Version: 1.0, April 2019

How to use this module:

- Look for gray < and > marks on either the bottom or the left and right sides of this pane, depending on the size of the window. Click those to navigate to the previous or next page, respectively.
- Jump to a specific page using the navigation links at the left.
- Play audio for a page using the player at the top of that page. Audio often provides more complete information than the text and graphics alone. A transcript is available from a link on the same page.

Objectives

On completing this training, you should be able to:

- Identify software available in the RHEL 8 *container-tools* module
- Install the *container-tools* module in RHEL 8
- Use Podman in RHEL 8 to manage and build containers
- Use Buildah in RHEL 8 to build a container image

Prerequisites

This training assumes that you have the following:

- Red Hat Certified Engineer (RHCE) or equivalent experience with Red Hat Enterprise Linux (RHEL)
- Working knowledge of the Docker command line interface and the Dockerfile format
- Experience with container management basics similar to what's included in Docker Essential Training: 1 Installation and Configuration (Linkedin Learning)*

* *Red Hat internal associates, see this Mojo page about changes to your Lynda.com URLs after April 30, 2019.*

▼ Show transcript

On completing this training, you should be able to:

Identify software available in the RHEL 8 container-tools module, install that container-tools module in RHEL 8, use Podman in RHEL 8 to manage and build containers, and use Buildah in RHEL 8 to build a container image.

This training assumes that you are a Red Hat Certified Engineer or that you have equivalent experience with Red Hat Enterprise Linux. It also assumes that you have working knowledge of the Docker command line interface and the Dockerfile format, and experience with container management basics similar to what's included in the Lynda.com course listed here.

Lab Environment

Successful completion for this training includes hands-on lab activities hosted in a cloud-based lab environment.

PROVISIONING

(1) Log in to the OpenTLC lab portal.

(2) On the far left, mouse over **Services** and select **Catalogs** from the pop-up menu.

(3) Select to expand **All Services** and **Support Labs**.

(4) Select **cee-rl-021** under that list.

(5) Select **Order**.

(6) Complete the application request: read the **Runtime Warning**, check the box to confirm the runtime and expiration dates, and select **Submit**.

IMPORTANT: Expect **up to 20 minutes** to provision your lab environment.

(7) Look for information on how to access your lab environment from one of two places:

- **Information email**
  Look for an email from *Red Hat OPENTLC <noreply@opentlc.com>* with the Subject similar to: *Your Red Hat OPENTLC service provision request for OTLC-LAB_COMPLETED has completed.* This email may arrive before the environment is ready to use. If you don't receive this email within 15 minutes, you can generate a new one from OpenTLC: **Services > Active Services > OTLC-LAB-NAME* > App Control > Status > Submit**
- **The OpenTLC UI**
  Look in the *Custom Attributes* section on the right in OpenTLC: **Services > Active Services > OTLC-LAB-*NAME***

SYSTEM INFO

System	IP	Credentials	Description
servera.example.com	172.25.250.10	root/shamrock masher positron tweet	RHEL 8 container-tools host

SSH ACCESS

(1) Use the SSH command shown here to access your environment, modifying the command based on the information you received by email:

```
$ ssh flastname-redhat.com@classroom-guid.red.osp.opentlc.com
```

(2) When prompted, log in to your lab environment using one of these options:

- A password set by OpenTLC and provided in the information email.
- An SSH key pair configured as described here: http://www.opentlc.com/ssh.html

```
$ ssh flastname-redhat.com@classroom-guid.red.osp.opentlc.com
The authenticity of host 'classroom-guid.red.osp.opentlc.com (169.47.191.199)' can't be established.
ECDSA key fingerprint is SHA256:v01n4XWXr0lphfGpBiSSvbasmrlQZul2ntS8g0Kbmdk.
Are you sure you want to continue connecting (yes/no)? yes

flastname-redhat.com@classroom-guid.red.osp.opentlc.com's password: <PASSWORD>

[flastname-redhat.com@classroom-guid ~]$ sudo su -
Last login: Thu Oct 24 14:19:41 EDT 2019 from 61.0.147.106 on pts/0
[root@classroom-guid ~]#
```

CONSOLE ACCESS

If you need console access to any of the machines in this environment, follow these steps:

(1) Retrieve the **Master Console** URL from the information email you received on provisioning your lab environment. Look for a line that's similar to this one:

```
Master Console: https://console-redvnc.apps.shared.na.openshift.opentlc.com
```

(2) Open this console URL in your web browser, and select **Log in with OpenShift**.

(3) Enter your OpenTLC username and password at the OpenShift login prompt.

(4) If a dialog appears requiring you to *Authorize Access* for a service account, choose to allow the selected permissions to continue.

(5) Select **Access Console** for a given virtual machine to open a VNC console session with that system.

LOCAL WEB BROWSER ACCESS (HOSTED WEB UI)

(1) Use the same *ssh* command from your local system as for command line access, but add the argument **-CfND 8080**

```
[user1@laptop ~]$ ssh flastname-redhat.com@classroom-guid.red.osp.opentlc.com -CfnND 8080
```

(2) Configure your local web browser to send all web traffic through **localhost:8080**.

```
[user1@laptop ~]$ google-chrome --proxy-server="socks5://127.0.0.1:8080" --host-resolver-rules="MAP * 0.0.0.0 , EXCLUDE localh
ost" &
```

▼ Show transcript

Successful completion for this training includes hands-on lab activities. Use the information on this page to launch you cloud-based lab environment, locate the URLs and credentials to access that environment, familiarize yourself with the network setup, and use SSH or a local web browser to access lab systems.

container-tools

▼ Show transcript

This section covers the tools used to create and manage containers in RHEL 8.

What container tools are in RHEL 8?

RHEL 8 ships with a set of tools useful for managing and creating containers and container images.

Podman - Client tool for managing containers. Can replace most features of the *docker* command for working with individual containers and images.

podman-docker - Provides a script that emulates the Docker CLI by executing *podman* commands. Also creates links between all Docker CLI manual pages and Podman.

Buildah - Client tool for building container images compliant with the Open Container Initiative (OCI).

Skopeo - Client tool for copying container images to and from container registries. Includes features for signing and authenticating images as well.

runC - Container runtime client for running and working with OCI format containers.

Note: Container tools in Red Hat OpenShift Container Platform (RHOCP) 4

Some of the tools distributed in RHEL 8 are also used in other Red Hat products, including RHOCP 4. This module focuses on the standalone RHEL 8 usage for the tools listed here, and it does **NOT** cover container tools and technologies used with RHOCP 4 (e.g. Kubernetes and CRI-O).

▼ Show transcript

What container tools are in RHEL 8? RHEL 8 ships with a set of tools useful for managing and creating containers and container images. These tools include those listed here.

Podman is a client tool used for managing containers. It replaces most of the features that the docker command provided.

Podman-docker is an optional package that provides a script that emulates the Docker CLI by executing podman commands. If you have other utilities that expect the Docker CLI, then this package can provide backwards compatibility. It also creates links between all Docker CLI manual pages and Podman.

Buildah is a client tool for building images compliant with the Open Container Initiative, or OCI. Fun fact: the name for this tool was influenced by leading Red Hat software contributor Dan Walsh's strong Boston accent when pronouncing "builder".

Skopeo is a client tool used for copying container images to and from container registries. It also includes features for signing and authenticating images.

Runc is a container runtime client for running and working with OCI format containers.

Take a moment to read the note on this page before going on to the next page.

Container image formats

RHEL 8 container tools can create and manage container images in these formats:

- Open Container Initiative (OCI) images
- Docker version 2 images

Why OCI? The OCI specification format has these advantages over Docker version 2:

- Not bound to higher level constructs, such as a particular client or orchestration stack
- Not tightly associated with any particular commercial vendor or project
- Portable across a wide variety of operating systems, hardware, CPU architectures, and public clouds

Open Container Initiative FAQ

▼ Show transcript

There are two container image formats supported in RHEL 8: OCI images and Docker version 2 images.

Why OCI? The OCI specification format has the advantages listed here over the Docker version 2 format. First, OCI format images are not bound to higher level constructs, such as a particular client or orchestration stack. Also, OCI format images are not tightly associated with any particular commercial vendor or project. Finally, OCI format images are portable across a wide variety ofoperating systems, hardware, CPU architectures, and public clouds.

For additional information about the Open Container Initiative, see the FAQ page linked here.

container-tools artifacts

- The **container-tools** module is available in RHEL 8 in the *appstream* repository.
- Install *container-tools* with *yum* to include all the artifacts shown here.

```
[root@servera ~]# yum module info container-tools

Name             : container-tools
Stream           : rhel8 [d][e][a]
Version          : 820190211172150
Context          : 20125149
Profiles         : common [d] [i]
Default profiles : common
Repo             : rhel-8-for-x86_64-appstream-rpms
Summary          : Common tools and dependencies for container runtimes
Description      : Contains SELinux policies, binaries and other dependencies for use with container runtimes
Artifacts        : buildah-0:1.5-3.gite94b4f9.module+el8+2769+577ad176.x86_64
                 : container-selinux-2:2.75-1.git99e2cfd.module+el8+2769+577ad176.noarch
                 : containernetworking-plugins-0:0.7.4-3.git9ebe139.module+el8+2769+577ad176.x86_64
                 : containers-common-1:0.1.32-3.git1715c90.module+el8+2769+577ad176.x86_64
                 : fuse-overlayfs-0:0.3-2.module+el8+2769+577ad176.x86_64
                 : oci-systemd-hook-1:0.1.15-2.git2d0b8a3.module+el8+2769+577ad176.x86_64
                 : oci-umount-2:2.3.4-2.git87f9237.module+el8+2769+577ad176.x86_64
                 : podman-0:1.0.0-2.git921f98f.module+el8+2785+ff8a053f.x86_64
                 : podman-docker-0:1.0.0-2.git921f98f.module+el8+2785+ff8a053f.noarch
                 : runc-0:1.0.0-54.rc5.dev.git2abd837.module+el8+2769+577ad176.x86_64
                 : skopeo-1:0.1.32-3.git1715c90.module+el8+2769+577ad176.x86_64
                 : slirp4netns-0:0.1-2.dev.gitc4e1bc5.module+el8+2769+577ad176.x86_64
```

▼ Show transcript

The container-tools module is available in RHEL 8 in the appstream repository. Install container-tools with yum to include all the artifacts shown here in this "yum module info" command output.

Take some time to familiarize yourself with this list of software, and notice that it includes all the container tools mentioned earlier.

Install container-tools

- First, enable both the **baseos** and **appstream** repos.
- Then, install the **container-tools** Yum 4 module.

```
[root@servera ~]# subscription-manager repos --enable=rhel-8-for-x86_64-baseos-rpms
[root@servera ~]# subscription-manager repos --enable=rhel-8-for-x86_64-appstream-rpms

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

▼ Show transcript

To install the container-tools Yum 4 module, first enable both the "baseos" and "appstream" repos. Then run the "yum module install" command as shown here.

Lab 1.1

Objective	Lab activities
Identify software available in the RHEL 8 container-tools module	You are able to correctly identify software packages from a list that are provided by the container-tools module.

(1) If you have not already, launch and access your lab environment.

(2) Open a command terminal to *servera.example.com* in your lab environment (modify this command as appropriate based on the server's name):

```
[user@laptop ~]$ ssh -p 10001 root@serveraexamplecom-ceerl021-abc1234.srv.ravcloud.com
```

(3) Register the system to Red Hat using your Red Hat Customer Portal credentials, and attach a valid subscription that will provide access to the *Red Hat Enterprise Linux for x86_64* product:

```
[root@servera ~]# subscription-manager register --force
Registering to: subscription.rhsm.redhat.com:443/subscription
Username: rhn-support-XXXXXX
Password:
[root@servera ~]# subscription-manager list --available
[root@servera ~]# subscription-manager attach --pool=<POOL_ID>
```

(4) Enable the BaseOS and AppStream repos:

```
[root@servera ~]# subscription-manager repos --enable=rhel-8-for-x86_64-baseos-rpms
[root@servera ~]# subscription-manager repos --enable=rhel-8-for-x86_64-appstream-rpms
```

(5) Inspect the software packages provided by the container-tools module available from those repositories. Use that information to answer the question that follows.

```
[root@servera ~]# yum module info container-tools
```

Which of the following software packages are included in the container-tools Yum 4 module in RHEL 8? Select all that apply.

- [ ] podman
- [ ] runc
- [ ] kubernetes
- [ ] buildah
- [ ] cri-o
- [ ] docker
- [ ] podman-docker

Lab 1.2

Objective	Lab activities
Install the container-tools module in RHEL 8	Your system is subscribed to the "appstream" repo, and you have installed the latest container-tools Yum 4 module.

(1) If you don't have one open already, open a command terminal on *servera.example.com* as the root user as described for your lab environment.

(2) Assuming successful completion of the previous lab activity, install the **container-tools** module using *yum*.

(3) After installing the Yum module, run the following to verify your work, and submit the completion code from its output as prompted below:

```
[root@servera ~]# cee-rl-021 grade1.2
```

> COMPLETION CODE for Lab 1.2: _____
>
> ans: OMIT

Podman

▼ Show transcript

This section defines the container tool Podman and highlights common container management actions using Podman.

Podman in RHEL 8

Podman replaces both the Docker daemon and the Docker CLI.

- **Does not require a daemon** to manage containers or container images
- Uses the low-level runtime **runc** to directly manage (run) containers

Command syntax:

- *podman* syntax works like the *docker* syntax for pull, push, build, run, commit, and tag
- *podman-docker* (optional package) has a script emulates the Docker CLI using *podman*

Other *podman* features:

- **-a**|**--all** flag for *podman rm* and *podman rmi*, making container image cleanup easier
- Container storage in Docker is **/var/lib/docker** but Podman uses **/var/lib/containers**

▼ Show transcript

In RHEL 8, Podman replaces both the Docker daemon and the Docker CLI. Unlike Docker, Podman does not require a demon to manage containers or container images. Instead, Podman uses the low-level runtime runC to directly manage, or run, containers.

The "podman" command syntax works like the "docker" command syntax you're familiar with for pull, push, build, run, commit, and tag. If you choose to install the optional "podman-docker" package, a script emulates the Docker CLI using "podman" commands, allowing you to continue running the "docker" commands you're familiar with.

Beyond the parallel Docker CLI commands, "podman" has a ""--all" for "podman rm" and "podman rmi" actions, making container image cleanup easier.

Also notice that the default root container storage for Podman is "/var/lib/containers" instead of "/var/lib/docker".

podman login

- Red Hat's supported container registry transitioned from *registry.access.redhat.com* to ***registry.redhat.io***.
- Red Hat also maintains a registry for third party software at ***registry.connect.redhat.com***.
- Both of these registries require authentication.
- See this Red Hat Container Registry Authentication article for additional details.

Two authentication methods with Podman: **username/password** & **service account tokens**

▼ Show transcript

After installation, the first action you'll likely take is to set up authentication. The supported container registry has transitioned from registry.access.redhat.com to registry.redhat.io. Red Hat also maintains a registry for third party software at registry.connect.redhat.com. Both of these will require authentication. See the Red Hat Container Authentication article noted on this page for additional details.

You can choose from one of two authentication methods with Podman: username and password, or service account tokens. The next two pages cover how to use each of these with the "podman login" command.

podman login - username/password

To gain access to Red Hat supported container images from *registry.redhat.io*, use the **podman login** command with your valid Customer Portal account credentials.

```
[root@servera ~]# grep registry /etc/containers/registries.conf
registries = ['registry.redhat.io', 'quay.io', 'docker.io']

[root@servera ~]# podman login -u rhn-support-ablum https://registry.redhat.io
Password:
Login Succeeded!

[root@servera ~]# podman logout https://registry.redhat.io
Removed login credentials for registry.redhat.io
```

▼ Show transcript

By default, the registries used are defined in /etc/containers/registries.conf. Notice in the command output here that registry.redhat.io is the first registry configured in RHEL 8. To log in using a username and password, use the "-u" flag with the "podman login" command as shown in the examples here.

The username provided must be a valid Customer Portal account with a valid subscription associated with it.

To log out, use a "podman logout" command similar to the one shown here.

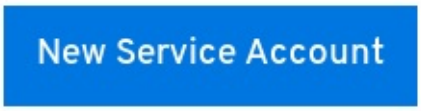
See the Red Hat Container Authentication article noted on this page for additional details.

podman login - token

Create a named token to use for this:

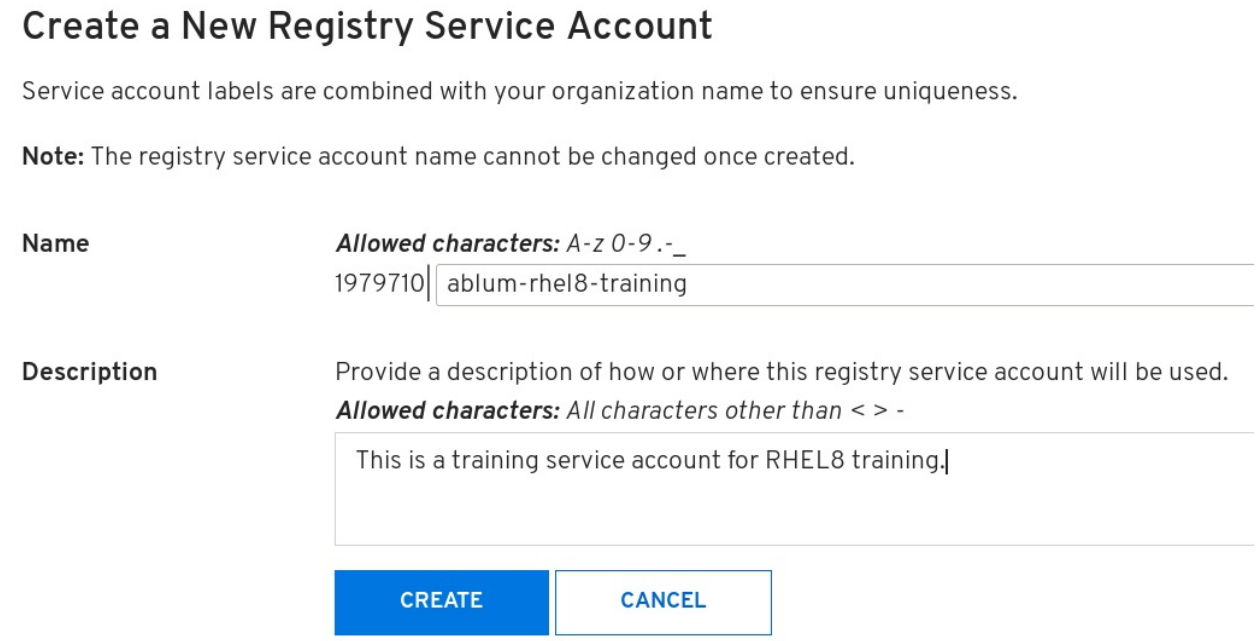
(1) Log in to *https://access.redhat.com/terms-based-registry/#/accounts* using your Red Hat Customer Portal credentials.

(2) Click *New Service Account*.



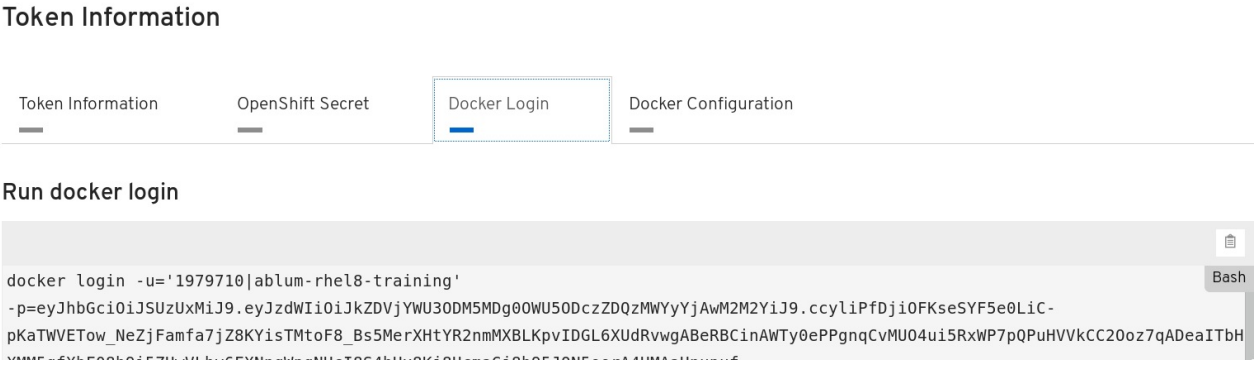
(3) Enter a unique *Name* and *Description* for the new registry service account, and then click **Create**.

Click here to see what this looks like



(4) Click the *Docker Login* (or Podman) tab to see the new command. It should look something like this:

Click here to see what this looks like



(5) Copy the command, then run that command on the RHEL container host system.

```
[root@servera ~]# podman login -u='1979710|ablum-rhel8-training' -p=eyJh...SNIP...DVUEuY registry.redhat.io

Login Succeeded!
```

▼ Show transcript

Again, Red Hat also supports using a service account token to log in to container registries. You'll need to use steps on this page to create and use a named token.

First, log in to the "terms-based-registry" URL shown in this first step using your Red Hat Customer Portal credentials.

Then, click "New Service Account."

When creating the new Service Account in step 3, make sure to comply with the allowed characters for your name and description. Then, click CREATE.

After creation, you'll be able to see the service account token in the UI. Click the Docker Login tab and copy the command provided.

To log in, run that command on the RHEL container host system. If you've installed the "podman-docker" package, then you can use this "docker" command directly. Otherwise, replace "docker" with "podman" in the command as shown in this last example.

podman pull

- Use this to copy an image from a registry to a local machine
- Command syntax: **podman pull [options] name[:tag|@digest]**
- If you don't specify an image tag, this pulls the **latest** tag by default

Expand each section, and familiarize yourself with these examples:

Pull the 'rhel7' image from registry.redhat.io

```
[root@servera ~]# podman pull registry.redhat.io/rhel7
Trying to pull registry.redhat.io/rhel7...Getting image source signatures
Copying blob 7585ac2ccc88: 0 B / 72.72 MiB [----------------------------------]
Copying blob 7585ac2ccc88: 70.70 MiB / 72.72 MiB [============================]
Copying blob 7585ac2ccc88: 72.72 MiB / 72.72 MiB [============================] 6s
Copying blob 1568bf6457e5: 1.29 KiB / 1.29 KiB [============================] 6s
Copying config bd0240457182: 6.37 KiB / 6.37 KiB [============================] 0s
Writing manifest to image destination
Storing signatures
```

Pull the 'fedora' image from the Docker hub with a specific tag

```
[root@servera ~]# podman pull docker.io/library/fedora:26
Trying to pull docker.io/library/fedora:26...Getting image source signatures
Copying blob fef9491d900a: 75.73 MiB / 75.73 MiB [============================] 6s
Copying config f36d549d2474: 2.20 KiB / 2.20 KiB [============================] 0s
Writing manifest to image destination
Storing signatures
f36d549d2474f7689939a24aedf9690d7dcdc8010250cb98482fe7d2a24cf4d4
```

Pull the 'rhel7' image with a specific digest

```
[root@servera ~]# podman pull registry.redhat.io/rhel7@sha256:135cbbec4581cd8b2f550dd90dea06affb55def73c7421e64091dc3f638d05e4
Trying to pull registry.redhat.io/rhel7@sha256:135cbbec4581cd8b2f550dd90dea06affb55def73c7421e64091dc3f638d05e4...Getting imag
e source signatures
Copying blob dab9f87f3be2: 1.20 KiB / ? [------------------------------=---] 0s
Copying blob c181936b24e2: 70.39 MiB / ? [------------------------------=------]
Copying blob c181936b24e2: 71.39 MiB / ? [-=-----------------------------] 6s
Copying blob dab9f87f3be2: 1.20 KiB / ? [------------------------------=---] 6s
Writing manifest to image destination
Storing signatures
33a3ad89f9ab42d8ab8b462f5b3c9f79aa135e8bc5d62815450724d474775335
```

As a non-root user, pull the 'rhel7' image from registry.redhat.io

```
[user1@servera ~]$ podman pull registry.redhat.io/rhel7
Trying to pull registry.redhat.io/rhel7...Getting image source signatures
Copying blob da59b306fcf5: 72.31 MiB / 72.31 MiB [============================] 6s
Copying blob e23b0afac3fa: 1.23 KiB / 1.23 KiB [============================] 6s
Copying config b8fffd14574a: 6.31 KiB / 6.31 KiB [============================] 0s
Writing manifest to image destination
Storing signatures
b8fffd14574a044315ebd7afb12cedde603bcf1e03f97b08e8a30d7a462f3144

[user1@servera ~]$ grep rhel7 ~/.local/share/containers/storage/overlay-images/images.json
[{"id":"b8fffd14574a044315ebd7afb12cedde603bcf1e03f97b08e8a30d7a462f3144","digest":"sha256:326768aa8c86dc7785a49f9711ec44a3cd7
d5975b007c6530e97a8ba5934e851","names":["registry.redhat.io/rhel7:latest"]
```

IMPORTANT NOTES:

- Running containers as a non-root user will be in Technical Preview at the time of RHEL 8.0 GA.
- Authentication is required (*podman login*) before pulling images from *registry.redhat.io*.
  - Without authentication, expect errors like "unable to retrieve auth token: invalid username/password"

- If you have trouble logging in, see Troubleshooting Authentication Issues with registry.redhat.io

▼ Show transcript

After successfully authenticating, you can use the "podman pull" command to make a local copy of an image from a registry. Expand each of the sections here and familiarize yourself with the examples and we walk through those.

The first example shows how to pull a copy of the "rhel7" image from registry.redhat.io.

The second example shows how to use "podman pull" to copy a Fedora image with a specific tag. The tag in this example is 26.

You can also pull an image by its digest to insure that a specific version is copied. This is useful when specific versions of software are QA'd or in busy build environments when tags are being reused or re-purposed.

The last example shows how "podman pull" supports the ability for non-root users to pull and copy images from a registry. In this example, a "rhel7" image is copied to a local container storage location in user1's home directory.

Read the important notes at the end of this page before going on to the next page.

podman run

- Use **podman run** to start a process with its own file system, its own networking, and its own isolated process tree (same as with *docker run*).
- Define defaults for a process within the image or by adding them to the *podman run* command.
- Inspect what defaults are defined for an image using **podman inspect <image-name>**.
  image-name = the name of a local image or the image ID as given by *podman images*

Some of the files added to the container's file system when running with *podman run*:

- **/etc/hosts** for networking
- **/etc/hostname** for networking
- **/etc/resolv.conf** for networking
- **/run/.containerenv** to indicate to programs that are running in a container

---

▼ Show transcript

Use "podman run" to start a process with its own file system, its own networking, and its own isolated process tree, same as with "docker run".

You can define defaults for a process within the image or by adding them to the *podman run* command. You can inspect what defaults are defined for an image using "podman inspect <image-name>" where "image-name" can be the name of a local image or the image ID as given by "podman images".

When running a container using "podman run," there are several files added to the container's file system. These include /etc/hosts, /etc/hostname, and /etc/resolv.conf for networking, and /run/.containerenv to indicate to programs that are running in a container.

podman run example

What can you expect when running containers with *podman run*?

Expand these sections to step through an example:

Run a RHEL7 container with an interactive shell

```
[root@servera ~]# podman run -it registry.redhat.io/rhel7 /bin/bash
Trying to pull registry.redhat.io/rhel7...Getting image source signatures
Copying blob 7585ac2ccc88: 0 B / 72.72 MiB [------------------------------------]
Copying blob 7585ac2ccc88: 70.70 MiB / 72.72 MiB [============================]
Copying blob 7585ac2ccc88: 72.72 MiB / 72.72 MiB [============================] 6s
Copying blob 1568bf6457e5: 1.29 KiB / 1.29 KiB [============================] 6s
Copying config bd0240457182: 6.37 KiB / 6.37 KiB [============================] 0s
Writing manifest to image destination
Storing signatures

[root@a2c2c12e9811 /]#  yum install httpd -y

[root@a2c2c12e9811 /]# ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 19:23 pts/0    00:00:00 /bin/bash
root        21     1  0 19:23 pts/0    00:00:00 ps -ef

[root@a2c2c12e9811 /]# ls -l /run/.containerenv
-rw-r--r--. 1 root root 0 Mar 26 19:25 /run/.containerenv
```

List processes running outside of this container's namespace

```
(from a 2nd terminal to the host system)
[root@servera ~]# ps -ef | grep podman

root      2749  1470  5 08:52 pts/0    00:00:11 podman run -it registry.redhat.io/rhel7 /bin/bash

root      2950     1  0 08:53 ?        00:00:00 /usr/libexec/podman/conmon -s -c a2c2c12e9811b31daa58843e72c9eab907a2c78ef486e
33a3536d35094ef2a6f -u a2c2c12e9811b31daa58843e72c9eab907a2c78ef486e33a3536d35094ef2a6f -r /usr/bin/runc -b /var/lib/container
s/storage/overlay-containers/a2c2c12e9811b31daa58843e72c9eab907a2c78ef486e33a3536d35094ef2a6f/userdata -p /var/run/containers/
storage/overlay-containers/a2c2c12e9811b31daa58843e72c9eab907a2c78ef486e33a3536d35094ef2a6f/userdata/pidfile -l /var/lib/conta
iners/storage/overlay-containers/a2c2c12e9811b31daa58843e72c9eab907a2c78ef486e33a3536d35094ef2a6f/userdata/ctr.log --exit-dir
/var/run/libpod/exits --socket-dir-path /var/run/libpod/socket -t --log-level error
```

Remove a specific running container

```
[root@servera ~]# podman ps
CONTAINER ID  IMAGE                            COMMAND    CREATED        STATUS            PORTS  NAMES
a2c2c12e9811  registry.redhat.io/rhel7:latest  /bin/bash  6 minutes ago  Up 6 minutes ago         modest_bartik

[root@servera ~]# podman rm a2c2c12e9811
cannot remove container a2c2c12e9811b31daa58843e72c9eab907a2c78ef486e33a3536d35094ef2a6f as it is running - running or paused
containers cannot be removed: container state improper

[root@servera ~]# podman stop a2c2c12e9811

[root@servera ~]# podman rm a2c2c12e9811
a2c2c12e9811b31daa58843e72c9eab907a2c78ef486e33a3536d35094ef2a6f

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

Remove all containers and container images

```
[root@servera ~]# podman rm -a
486f52aa8dc135ae1c0c3fb029af607449662b816a61c3b4ce27e0717f3a6b16
[root@servera ~]# podman rmi -a
a80dad1c19537b0961e485dfa0a43fbe3c0a71cec2cca32d3264e087e396a211

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

[root@servera ~]# ps -ef | grep podman
root      3909  3118  0 15:53 pts/1    00:00:00 grep --color=auto podman
[root@servera ~]# ps -ef | grep conmon
root      3911  3118  0 15:53 pts/1    00:00:00 grep --color=auto conmon
```

Warning: Don't mistake "conmon" for a daemon for Podman

One *conmon* process runs in addition to the processes running in a given container's namespace. When the container's process exits or is killed, so will its associated *conmon* process. Thus, it is ***not*** a daemon, but rather a process that wraps each container launched. For more information, see Question, Podman architecture #1175 (from libpod on GitHub)

▼ Show transcript

What can you expect when running containers with podman run? Expand these sections as we narrate an example for you.

Consider a RHEL container run with an interactive shell. The "-i" in the example "podman run" command keeps standard input open, allowing for keyboard interactions with the container after the "run" command. The "-t" option allocates a pseudo-TTY and attaches it to the standard input of the container. The "/bin/bash" argument is the command that will be run within the container's isolated namespace. Once in the interactive shell, you could install software or run other commands like "ps" if available.

What processes are running outside of this container's namespace? This second section shows listing those processes. The "ps" output here shows that the "podman run" command executed, but notice also there is an additional process on the host called "conmon".

To remove a specific running container, notice that the "podman rm" in the third section returns an error. To remove it properly, you should first stop the container using "podman stop" command, then remove it.

One convienent feature available with "podman rm" or "podman rmi" is the "-a" flag. This option removes all stopped containers and container images that are not in use. Take care when using the "-a" and "-f" flags together when removing containers because together they will forcibly remove all containers and container images from a system.

Read the warning here about the "conmon" process before going on to the next page.

podman build

- Use either *podman* or *buildah* to build a container image.
  Buildah is covered more later in this training.
- Two ways to build a container with *podman*:
  - Option 1: Use **podman build** and a Dockerfile
  - Option 2: Commit changes made in a running container to a new container image tag

Note: podman build is a subset of Buildah

The *podman build* command provides a subset of functionality and code from the *buildah* command.

▼ Show transcript

Use either podman or buildah to build a container image. Buildah is covered more later in this training.

There are two ways to build a container using "podman". Option 1 is to use "podman build" and a Dockerfile. Option 2 is to commit changes made in a running container to a new container image tag. The next two pages cover each of these options.

Note that the "podman build" command provides a subset of functionality and code from the "buildah" command, which is covered later.

Build a container image with a Dockerfile

podman build <build context directory>
build context directory = local directory path to the Dockerfile

The Dockerfile format that Red Hat supports in the *podman build* command is equivalent to what's supported using a *docker build* command.

Consider this example Dockerfile:

```
[root@servera tmp]# vi Dockerfile

FROM registry.redhat.io/rhel:latest
ENTRYPOINT ["/bin/ps","-e","f"]
RUN yum install procps-ng -y
```

You can use this file to build a container image with *podman build*:

```
[root@servera tmp]# podman build -t rhel:with_ps .
STEP 1: FROM registry.redhat.io/rhel:latest
STEP 2: ENTRYPOINT ["/bin/ps","-e","f"]
--> 77625bb5fded060d1e3c5ecbceb6cc40307adc17053bb3ab8c368f0b500835cd
STEP 3: FROM 77625bb5fded060d1e3c5ecbceb6cc40307adc17053bb3ab8c368f0b500835cd
STEP 4: RUN yum install procps-ng -y
Loaded plugins: ovl, product-id, search-disabled-repos, subscription-manager
rhel-7-server-rpms                                   | 3.5 kB     00:00
(1/3): rhel-7-server-rpms/7Server/x86_64/group         | 774 kB   00:00
(2/3): rhel-7-server-rpms/7Server/x86_64/updateinfo    | 3.1 MB   00:01
(3/3): rhel-7-server-rpms/7Server/x86_64/primary_db    |  56 MB   00:03
Package procps-ng-3.3.10-23.el7.x86_64 already installed and latest version
Nothing to do
--> ead2aab00c5ae80f55962b3b6a079907b7a2533b92ce791a4fdc946aeb385e64
STEP 5: COMMIT rhel:with_ps
[root@servera tmp]# cat Dockerfile


[root@servera tmp]# podman images
REPOSITORY               TAG       IMAGE ID       CREATED              SIZE
localhost/rhel           with_ps   ead2aab00c5a   About a minute ago   605 MB
```

▼ Show transcript
In the "podman build" command, you should specify the build context directory, which is the relative local directory path to the Dockerfile. For that, you can also use the HTTP or HTTPS URL of an archive, or a Git repository. The Dockerfile format that Red Hat supports in the "podman build" command is equivalent to what's supported using a "docker build" command.

Consider the example Dockerfile shown here. This example has 3 lines in it.

The FROM instruction in the first line creates an initial image storage layer based on the "rhel" image.

ENTRYPOINT in this example runs the command "ps -ef" without starting a command shell. This format is known as the "exec" format, and it's the preferred way of running a command when a shell or shell variables are not needed.

The RUN instruction in the file installs the "procps-ng" package and its dependencies.

To build a container image based on these 3 instructions, you can use this file with "podman build" as shown in this second example. The "-t" option specifies the name to assign to the resulting image if the build process completes successfully. Don't overlook the period at the end of the "podman build" example here: that argument represents the build context directory, which here is the relative directory path to the Dockerfile.

Create a new container image from a running container

Use **podman commit** to commit changes made in a running container to a ***new*** container image tag.

Consider these example changes to the running container from the container image *registry.redhat.io/rhel7:latest*:

```
[root@servera ~]# podman run -it registry.redhat.io/rhel7 /bin/bash
[root@c7e6e5edcdbf /]# pstree
bash: pstree: command not found
[root@c7e6e5edcdbf /]# yum install psmisc -y
[root@c7e6e5edcdbf /]# pstree
bash---pstree
[root@c7e6e5edcdbf /]# exit
```

After making those changes, they can be committed to a new image by running:

```
[root@servera ~]# podman ps -a
CONTAINER ID  IMAGE                            COMMAND    CREATED             STATUS                     PORTS  NAMES
c7e6e5edcdbf  registry.redhat.io/rhel7:latest  /bin/bash  About a minute ago  Exited (0) 2 seconds ago         sharp_morse

[root@servera ~]# podman commit c7e6e5edcdbf rhel7:with_pstree
Getting image source signatures
Skipping blob cd645bd4dc85 (already present): 205.13 MiB / 205.13 MiB [=====] 0s
Skipping blob e9f742d619c9 (already present): 10.00 KiB / 10.00 KiB [=======] 0s
Skipping blob cd645bd4dc85 (already present): 205.13 MiB / 205.13 MiB [=====] 9s
Skipping blob cd645bd4dc85 (already present): 205.13 MiB / 205.13 MiB [====] 10s
Skipping blob e9f742d619c9 (already present): 10.00 KiB / 10.00 KiB [======] 10s
Copying blob 387093245481: 407.77 MiB / 407.77 MiB [======================] 10s
Copying config 965f536a0535: 3.20 KiB / 3.20 KiB [=========================] 0s
Writing manifest to image destination
Storing signatures
965f536a05358e7f4e0aaf9277c1203b8edab30bb4554775aa2fcaa52bfc9567

[root@servera ~]# podman images
REPOSITORY                 TAG           IMAGE ID       CREATED          SIZE
localhost/rhel7            with_pstree   965f536a0535   46 seconds ago   643 MB
registry.redhat.io/rhel7   latest        bd0240457182   10 days ago      215 MB
```

▼ Show transcript

Another way to build a container image using "podman" is to use "podman commit" to commit the changes made in a running container to a NEW container image tag.

Consider these example changes to the running container from the "rhel7" container image. The container is running interactively, and while it is running, a package is installed, which provides the pstree binary.

One you exit the shell is exited the container process ends, but the image associated with it still exists as given in the "podman ps" output shown here with the "-a" flag. To create a new container image that includes these changes, notice how "podman commit" is used with that container ID, and new image name is specified with its optional tag.

After the successful commit, a new container image exists in the local registry. This new container image includes the pstree binary that was installed in the original running container.

Lab 2.1

Objective	Lab activities
Use Podman in RHEL 8 to manage and build containers	Identify key container attributes from a running container.
	Run a container from an image stored in registry.connect.redhat.com.
	Remove specific containers and container images on a RHEL 8 system.

(1) To set up this lab activity, launch the following command as the **root** user on **servera.example.com**:

```
[root@servera ~]# cee-rl-021 setup2.1
Initiating cee-rl-021 with option(s) setup2.1
running a simple podman command        PASS
build and run container image...    PASS
pulling some auspicious containers...    PASS
launching several containers.....    PASS
```

This command should build and launch a container with the name **alexwhen-2048**.

(2) Use *podman* commands to answer the following questions about this container and image:

Which host port is alexwhen-2048's container port is mapped to?

- 80
- 8080
- 8087
- 9080

> Enter the tag associated with the alexwhen-2048 image: _____
>
> ans: OMIT

---

(3) Use *podman login* to log in to *registry.connect.redhat.com* using your personal Customer Portal credentials.

(4) Use *podman search* to find the "official Axibase Time Series Database images for Docker containers" (atsd) repository in the *registry.connect.redhat.com* registry. **Hint:** Read through `man 1 podman-search` for some ways to search within a given registry. Make sure you use the repository that matches the description "official Axibase Time Series Database".

(5) Use *podman pull* to copy the latest **atsd** container image from *registry.connect.redhat.com*.

(6) Use *podman run* to start a container based on the latest **atsd** container image. The container must be:

- Running in detached mode
- Running with the name **atsd-lab2.1**
- Exposing container ports to equivalent host ports (containerPort:hostPort):
  - 8088:8088
  - 8443:8443

(7) Clean up (remove) all running containers *and* container images on *servera.example.com* **EXCEPT** those from either *registry.redhat.io* or *registry.connect.redhat.com*.

(8) After completing these steps, run the following to verify your work, and submit the completion code from its output as prompted below:

```
[root@servera ~]# cee-rl-021 grade2.1
```

> COMPLETION CODE for Lab 2.1: _____
>
> ans: OMIT

Info: Fun fact! 2048 is a game you can play

The container used in this exercise is actually a game you can play called 2048.

If you setup an SSH tunnel to the learning environment, you can proxy your browser's traffic to play. See the "LOCAL WEB BROWSER ACCESS" section of the the lab environment page for the command to set up this tunnel.

From there, you can launch the game by running your web browser's executable. For Google Chrome on Fedora and port 8080, for example, you might run:

*google-chrome --proxy-server="socks5://127.0.0.1:8080" --host-resolver-rules="MAP * 0.0.0.0 , EXCLUDE localhost" http://servera.example.com:8087*

Lab 2.2

Objective	Lab activities
Use Podman in RHEL 8 to manage and build containers	Create a container image using a Dockerfile.

(1) Create a new working directory for this activity:

```
[root@servera ~]# mkdir /root/lab2.2
[root@servera ~]# cd lab2.2/
[root@servera lab2.2]#
```

(2) Use your personal Customer Portal account to log into the *registry.redhat.io* registry.

(3) Pull the latest **rhel7** container image from *registry.redhat.io*.

(4) Create a Dockerfile that meets these requirements:

- Uses **registry.redhat.io/rhel7** as the base container image
- Installs the **redhat-support-tool** package
- Creates the file **/opt/DONE** that contains the string **lab2.2**

(5) Use your new Dockerfile with *podman build* to create a container image with repo:tag name **rhel7:lab2.2**.

(6) Run a container from the newly created image to verify it includes the software and expected contents.

(7) After completing these steps, run the following to verify your work, and submit the completion code from its output as prompted below:

```
[root@servera ~]# cee-rl-021 grade2.2
```

> COMPLETION CODE for Lab 2.2: _____
>
> ans: OMIT

Buildah

▼ Show transcript
This section introduces the container image building tool Buildah available in RHEL 8.

Buildah in RHEL 8

Command: **buildah**

Using Buildah, you can create container images from:

- A working container
- A Dockerfile
- "Scratch" (built from its basic parts)

The resulting images are OCI-compliant, so they will work on any container runtime that meet the OCI Runtime Specification (e.g. Podman, Docker, and CRI-O).

Buildah stores images in an area identified as *containers-storage* in */var/lib/containers*

▼ Show transcript
Using Buildah, you can create container images from a working container, a Dockerfile, or from scratch, meaning from assembling basic parts. The resulting images are OCI-compliant, so they will work on any container runtime that meet the OCI Runtime Specification (such as Podman, Docker, and CRI-O). Buildah stores images in an area identified as "containers-storage" in /var/lib/containers.

Buildah advantages...

...or how building container images with Buildah is different from Docker:

- **There's no daemon.** Like Podman, Buildah does not require a daemon like the Docker daemon.
- You can build a container using **either an existing base image or an empty image**.
- It builds **smaller images** than Docker.
- It **makes the image more secure** by not including the build tools (e.g. gcc, make, dnf) within that resulting image.

▼ Show transcript

Listed here are the ways in which building container images with Buildah is different from building with Docker. First, there's no daemon. Just like with Podman, Buildah does not require a daemon like the Docker daemon.

Second, you can start building a container using either an existing base image or an empty image.

Third, Buildah builds smaller images than Docker. And fourth, Buildah makes the image more secure by not including the build tools used to build the image within that resulting image.

buildah build-using-dockerfile

Buildah supports building OCI-compliant container images from a Dockerfile:

```
#> buildah build-using-dockerfile
#> buildah bud
```

This command shares code with *podman build*.

First, create a working directory and Dockerfile (as with podman build)

```
[root@servera ~]# mkdir myecho
[root@servera ~]# cd myecho
[root@servera myecho]# ls
Dockerfile  myecho
[root@servera myecho]# cat Dockerfile
FROM registry.redhat.io/rhel8-beta/rhel
ADD myecho /usr/local/bin
ENTRYPOINT "/usr/local/bin/myecho"
[root@servera myecho]# cat myecho
echo "This container works!"
[root@servera myecho]# ls -l myecho
-rwxr-xr-x. 1 root root 29 Apr  1 09:28 myecho
[root@servera myecho]#
```

Then, build the container image with buildah bud

```
[root@servera myecho]# buildah bud -t myecho --creds RHNID --format oci .
Password: RHNPASSWD
STEP 1: FROM registry.redhat.io/rhel8-beta/rhel
Getting image source signatures
Copying blob sha256:619051b1fc41546ce2c2d6911145f66472d72caf7a4aaf8ffcad782f27227e9c
 66.48 MiB / 66.48 MiB [====================================================] 5s
Copying blob sha256:386105ae8b6231e5c25160d9a40bec1da1fdb822455f6e3094bef2b6e877d865
 1.33 KiB / 1.33 KiB [======================================================] 0s
Copying config sha256:a80dad1c19537b0961e485dfa0a43fbe3c0a71cec2cca32d3264e087e396a211
 6.33 KiB / 6.33 KiB [======================================================] 0s
Writing manifest to image destination
Storing signatures
STEP 2: ADD myecho /usr/local/bin
STEP 3: ENTRYPOINT "/usr/local/bin/myecho"
STEP 4: COMMIT containers-storage:[overlay@/var/lib/containers/storage+/var/run/containers/storage:overlay.override_kernel_che
ck=true]localhost/myecho:latest
Getting image source signatures
Skipping fetch of repeat blob sha256:fba35a2d01b7bb0adfb3c6efeac57bf2f8ba85e21d2bc6c5ae8b441ed12b1d23
Skipping fetch of repeat blob sha256:848ae511b438d35ce6defcecfbbd287bf84e7fca37cfacd42d5af079c528c750
Copying blob sha256:96b9c9c8261779543d18b9264a6e9ff2d883bbdb85e0081438c882c9676dac23
 206 B / 206 B [============================================================] 0s
Copying config sha256:e7604cc5442aecc4090fc60310f5491897afbf4235f75f3c5a200a76391e550d
 3.12 KiB / 3.12 KiB [======================================================] 0s
Writing manifest to image destination
Storing signatures
--> e7604cc5442aecc4090fc60310f5491897afbf4235f75f3c5a200a76391e550d
```

Note

You may see errors indicating that "HOSTNAME is not supported for OCI image format," which is a known issue in Buildah and should be addressed in a future release.

▼ Show transcript

Before looking at Buildah as an independent container image, note its backward compatibility feature to support images from Docker. Buildah supports supports building OCI-compliant container images from a Dockerfile using the its build-using-dockerfile feature. This command also shares code with "podman build" covered earlier.

Because this can be a long command to type, there is an abbreviated "buildah bud" command you can use.

To build a container using "buildah bud," first create a working directory and Dockerfile as if you were using "podman build". Expand this first section to see the commands and outputs. In the example Dockerfile, notice the ENTRYPOINT uses shell form rather than the exec form. The shell form used here will execute "myecho" in a /bin/sh shell.

With the working directory and Dockerfile in place, build the container image using the "buildah bud" command. Expand this second section to see how the arguments were used.

The "-t" option specifies the name that should be assigned to the resulting image if the build process completes successfully.

The "--creds" option identifies the credentials to use when pulling an image from a registry that requires authentication. Using the "--creds" is not required if you've already logged in using using "podman login". Red Hat recommends logging in rather than using the "--creds" option.

The "--format" option controls which image format to build: either "oci" or "docker". You may see errors indicating that "HOSTNAME is not supported for OCI image format," which is a known issue in Buildah and should be addressed in a future release.

The period at the end of the "buildah bud" example here is the build context directory. As mentioned before, this can be the HTTP or HTTPS URL of an archive, a Git repository, or a local file path to the Dockerfile. In this example, the relative current working directory contains the Dockerfile.

buildah images & buildah containers

Use **buildah images** to list available images:

```
[root@servera ~]# buildah images
IMAGE NAME                                      IMAGE TAG            IMAGE ID             CREATED AT             SIZE
registry.redhat.io/rhel8-beta                   latest               a80dad1c1953         Nov 13, 2018 13:11     210
MB
registry.redhat.io/rhel7                        latest               6979ec30598b         Apr 8, 2019 09:39      214
MB
docker.io/library/nginx                         latest               bb776ce48575         Apr 10, 2019 17:22     113
MB
```

Use **buildah containers** to list working containers:

```
[root@servera ~]# buildah containers
CONTAINER ID  BUILDER  IMAGE ID     IMAGE NAME                       CONTAINER NAME
3f71cb46dced     *     a80dad1c1953 registry.redhat.io/rhel8-beta:latest rhel8-beta-working-container
```

▼ Show transcript

Use the commands "buildah images" and "buildah containers" to list images and containers respectively. The buildah images command displays locally stored images. In the example here, there are 3 different stored images. The buildah containers command shows a list of working containers only.

buildah from

- Creates a working container based on the specified image
- Useful for:
  - Testing an existing container image
  - When you need greater control over the build process beyond what's allowed in a Dockerfile

```
[root@servera myecho]# buildah images
IMAGE NAME                                          IMAGE TAG            IMAGE ID             CREATED AT             SIZE
localhost/myecho                                    latest               e7604cc5442a         Apr 1, 2019 09:48      210
MB

[root@servera myecho]# buildah from myecho
myecho-working-container

[root@servera myecho]# buildah containers
CONTAINER ID  BUILDER  IMAGE ID     IMAGE NAME                       CONTAINER NAME
0b6840873b5c     *     e7604cc5442a localhost/myecho:latest          myecho-working-container
```

This working container can be modified and then committed.

▼ Show transcript

Another way of creating a container image is to create a working container as starting material and make iterative changes to it. The "buildah from" command creates a working container based on the specified image. This is useful for testing an existing container image, or when you need greater control over the build process beyond what is allowed in a Dockerfile.

The first example here shows listing Buildah images and then creating a working container based on that image. The second example shows listing the available containers and their base images. This working container can be modified and then committed.

buildah mount

- Mounts a given container's root file system in a location accessible from the host
- Returns the local mountpoint on the host system

You can use that path to directly modify files in a working container's root file system.

Example:

```
[root@servera ~]# buildah containers
CONTAINER ID  BUILDER  IMAGE ID     IMAGE NAME                       CONTAINER NAME
0b6840873b5c     *     e7604cc5442a localhost/myecho:latest          myecho-working-container

[root@servera ~]# buildah mount 0b6840873b5c
/var/lib/containers/storage/overlay/17b4d245e03730f2ba6906c9dfb28255571fa8e85d7c1c1b854caada4a538bbb/merged

[root@servera ~]# echo "...and now it has been changed !" >> /var/lib/containers/storage/overlay/17b4d245e03730f2ba6906c9dfb28
255571fa8e85d7c1c1b854caada4a538bbb/merged/usr/local/bin/myecho
```

To unmount, use **buildah umount** with that same container ID:

```
[root@servera ~]# buildah umount 0b6840873b5c
```

▼ Show transcript

After creating a working container using "buildah from", you can use "buildah mount" to mount a given container's root file system in a location accessible from the host. When running "buildah mount", the path returned will be the local mountpoint on the host system. You can use that path to directly modify files in a working container's root file system.

In this example, the myecho file in the myecho-working-container is modified directly from the host.

After making changes, you can unmount the container's file system using "buildah umount" with that same container ID.

buildah run

- Another way to modify a working container
- ***Not*** the same as *podman run*
- The same as the **RUN** action in a Dockerfile
- Use when you want to run a single command in a working container for the purpose of *building* a container

Example:

```
[root@servera ~]# buildah from rhel8-beta/rhel
rhel-working-container

[root@servera ~]# buildah run rhel-working-container yum install httpd -y
```

Add some content for this "httpd" container to serve:

```
[root@servera ~]# buildah mount rhel-working-container
/var/lib/containers/storage/overlay/9f51055e238f4d230a10e99eaedceeba77bf74328129b4ee0fc6fca3057c2816/merged
[root@servera ~]# echo "hello world" > /var/lib/containers/storage/overlay/9f51055e238f4d230a10e99eaedceeba77bf74328129b4ee0fc
6fca3057c2816/merged/var/www/html/hello.html
[root@servera ~]# chmod 644 /var/lib/containers/storage/overlay/9f51055e238f4d230a10e99eaedceeba77bf74328129b4ee0fc6fca3057c28
16/merged/var/www/html/hello.html
[root@servera ~]# buildah umount rhel-working-container
```

Modify the working container to run a specific command and expose a port:

```
[root@servera ~]# buildah config --cmd "/usr/sbin/httpd -DFOREGROUND" rhel-working-container
[root@servera ~]# buildah config --port 80/tcp rhel-working-container
[root@servera ~]# buildah commit rhel-working-container myhttpd
[root@servera ~]# podman run -d -p 80:80 myhttpd
```

▼ Show transcript

Another way to modify a working container is to use the "buildah run" command. This is NOT the same as "podman run". Rather, "buildah run" is the same as the "RUN" action in a Dockerfile. Use "buildah run" when you want to run a single command in a working container for the purpose of *building* a container.

Consider the example here of using "buildah" and "buildah run" to create an "httpd" container that serves up some custom content. First, use "buildah from" to create a working container based on the "rhel8-beta" container image. In the next command, "buildah run" installs "httpd" and its dependencies in the working container.

Notice how the example progresses to modifying the working container while referencing it by the same *rhel-working-container* name. To create a working "httpd" container, you can also modify the default command and exposed port. To commit all the changes made to this working container to a new container image, use "buildah commit", which we'll look at closer on the next page.

buildah commit

Use **buildah commit** to commit the changes made to a working container to a new container image.

```
[root@servera ~]# buildah commit 0b6840873b5c myecho:modified
Getting image source signatures
Skipping fetch of repeat blob sha256:fba35a2d01b7bb0adfb3c6efeac57bf2f8ba85e21d2bc6c5ae8b441ed12b1d23
Skipping fetch of repeat blob sha256:848ae511b438d35ce6defcecfbbd287bf84e7fca37cfacd42d5af079c528c750
Skipping fetch of repeat blob sha256:402a86247ba740f956c0dd164b95ecd11c3e0bc11b24536795895a52c09f4be8
Copying blob sha256:d8d4b909f8e2b46113298f65bee974c1a19fbd0c1637176cf8b2f88e8f5e55c5
 231 B / 231 B [============================================================] 0s
Copying config sha256:8cd3b597a6feed5f0f115a341758c23fbb348167fecfc62cacbb94ad7dc4b809
 3.23 KiB / 3.23 KiB [======================================================] 0s
Writing manifest to image destination
Storing signatures
8cd3b597a6feed5f0f115a341758c23fbb348167fecfc62cacbb94ad7dc4b809

[root@servera ~]# buildah images
IMAGE NAME                                      IMAGE TAG            IMAGE ID             CREATED AT             SIZE
registry.redhat.io/rhel8-beta/rhel              latest               a80dad1c1953         Nov 13, 2018 13:11     210
MB
localhost/myecho                                latest               e7604cc5442a         Apr 1, 2019 09:48      210
MB
localhost/myecho                                modified             8cd3b597a6fe         Apr 1, 2019 11:57      210
MB
```

▼ Show transcript

Use "buildah commit" to commit the changes made to a working container to a new container image. Like in the "httpd" example on the previous page, the iterative changes made to "myecho" can be written to a new container image. The working container's image ID used in this example started with "0b68". As demonstrated earlier, you can view the working containers and their IDs using "buildah containers". Once a working container is committed, it should be listed in "buildah images" output alongside other container images that are available to run or modify.

Lab 3

Objective	Lab activity
Use Buildah in RHEL 8 to build a container image	Build a container image that uses the "centos:latest" container and runs the "sl" command.

(1) To set up this lab activity, launch the following command as the *root* user on *servera.example.com*:

```
[root@servera ~]# cee-rl-021 setup3
Initiating cee-rl-021 with option(s) setup3
running a simple buildah command  PASS
pulling centos container image        PASS
```

(2) Use *buildah* to run a working container using the **centos:latest** image, which was pulled in the previous step.

(3) Install **/root/training/sl-5.02-1.el7.x86_64.rpm** (from servera container host system) into the *centos* working container.

(4) Modify the *centos* working container to use **/usr/bin/sl** as the *ENTRYPOINT*.

(5) Commit the previous changes made to the centos working container to a new container image called **lab3**

(6) Test running this container with commands like *podman run lab3*.

Note: Resetting the terminal

The **sl** command can leave your terminal configuration in an undesirable state. To reset the terminal in the learning environment, run **tput init** or **tput reset**.

(7) After completing these steps, run the following to verify your work, and submit the completion code from its output as prompted below:

```
[root@servera ~]# cee-rl-021 grade3
```

> COMPLETION CODE for Lab 3: _____
>
> ans: OMIT

Resources

This list includes both resources we used to develop this training and resources you can use for ongoing reference or additional learning:

Podman and Buildah for Docker users by William Henry, Senior Consulting Engineer, Red Hat Inc.

Building, running, and managing containers RHEL 8.0 Beta Product Documentation

Open Container Initiative (GitHub)

axibase/atsd Installation: Docker Image (GitHub)

Manual pages installed with the *container-tools* module in RHEL:

```
man 1 podman-run
man 1 podman-pull
man 1 buildah-from
man 1 buildah-containers
man 1 buildah-images
```

Feedback

Thank you for taking time to provide your feedback on this training using the form below.

How likely are you to recommend this training module to other associates?

Not at all Likely ○0 ○1 ○2 ○3 ○4 ○5 ○6 ○7 ○8 ○9 ○10 Extremely likely

Enter additional comments here...

Submit FeedbackReset