Creating and Running OCI Containers

At the end of this chapter, students will be able to explain how to create, deploy, and run containers using Buildah and Podman tools.

Student Objectives:

- Explain OCI
- Describe Buildah and Podman software overview.
- Compare Buildah to Dockerfile and explain building a container image using both methods.
- Run a container and verify it works.

Using Containers

Containers are programs or services that have been packaged with all necessary libraries and dependancies included in the container. In order to ensure that images are compatable with all platforms and container management platforms, **Open Container Initiative** is an open source entity tasked with creating standards around the way containers are created and used. This helps to ensure that the same experience with a container will be observed without regard to which OS platform or container management solution the end user chooses. When a container image is created using the OCI standard, any container engine that is OCI compliant could be used to run the container.

There are many choices of container engine available for use. Some examples are Podman, Docker, and LXC amongst others. All of these examples are OCI compliant engines and can be used to manage containerized services.

Container management software solutions

Buildah compared to Dockerfile

Buildah is a container utility that can be used to create OCI compliant containers. They can be created either manually, from a backup image, or from Dockerfiles. OCI compliance means that these images could be used by any OCI compliant container engine. The buildah CLI allows for efficient creation of container images by mirroring Dockerfiles commands without requiring root access or a pre-existing Dockerfile. A Dockerfile is a configuration file that provides all of the necessary information for either **buildah** or **podman** to build a container with whatever customizations were needed. An example of a Dockerfile might look like this for a custom httpd application:

```
FROM registry.redhat.io/rhscl/httpd-24-rhel7:latest
LABEL description="This is a custom httpd container"
EXPOSE 80
ENV HTTPD_MPM=event
COPY /vhost.conf /etc/httpd/conf.d/
RUN systemctl restart httpd.service
RUN wget -O /var/www/html/index.html http://apachefun.com/customsite.html
```

Buildah could perform the same actions using the command line client. Since buildah commands can be run from the command line, they can also be put into a script as follows:

```
#!/bin/bash
container=CustomApache
buildah config --label description="This is a custom httpd container"
buildah config --port 80
buildah config --env HTTPD_MPM=event
buildah copy /vhost.conf /etc/httpd/conf.d/
buildah run $container /vhost.conf /etc/httpd/conf.d/
buildah run $container wget -O /var/www/html/index.html
http://apachefun.com/customsite.html
buildah commit $container
```

Using podman to run a container and verify functionality

Podman is a container management engine created by RedHat used to manage containers. There are many commands that can be run using the podman utility. These commands range in function in a few different catagories.

Logging in and working with images

command	effect
login	Login to container registry using credentials
search	Search registry for images containing a specific string
images	List locally downloaded images
pull	Download remote image locally for future use

podman run subcommands and options.

command	effect
-d	Run container as a background process

-it /bin/bash	Run container in an interactive terminal
--name <name>	Provide running container a name
-p	Map a port between host_port:container_port
-v	Map storage between host_location:container_location:Z

There are also management commands like **create start stop restart** etc.

Working with container resources

command	effect
mount	Mount containers root filesystem
import	Import tar archive as a filesystem image
export	Export filesystem image as a tar archive
cp	Copy files between local filesystem to containers filesystem
save	Save an image to a container archive
ps	Run ps to see container's running processes
top	Run top to see an auto updating container's running processes
port	List a container's port mappings
exec	Execute commands within running container

Basic podman workflow.

A basic podman workflow consists of a few steps to authenticate, search, download a container image, and run that image. A basic Apache webserver container workflow might look like this example:
1) First login to the registry.redhat.io online container registry allowing images to be searched and downloaded using the podman utility.

```
podman login registry.redhat.io
Username: opencloudjedi
Password:
Login Succeeded!
```

2) Search the registry for an Apache image. Since the output can be quite lengthy, the --limit option can be used to keep the resulting output short.

```
podman search httpd --limit 2
INDEX        NAME                                            DESCRIPTION
redhat.com   registry.access.redhat.com/rhscl/httpd-24-rhel7   Apache HTTP 2.4
Server
redhat.com   registry.access.redhat.com/rhmap45/httpd          Provides an
extension to the RHSCL Httpd ima...
redhat.io    registry.redhat.io/rhscl/httpd-24-rhel7           Apache HTTP 2.4
Server
redhat.io    registry.redhat.io/rhmap45/httpd                  Provides an
extension to the RHSCL Httpd ima...
docker.io    docker.io/library/httpd                           The Apache HTTP
Server Project
docker.io    docker.io/centos/httpd-24-centos7                 Platform for
running Apache httpd 2.4 or bui...
```

3) Using the image name above pull from the online registry to the local machine.

```
podman pull registry.access.redhat.com/rhscl/httpd-24-rhel7
Trying to pull registry.access.redhat.com/rhscl/httpd-24-rhel7...
Getting image source signatures
Copying config 649628158f done
...output omitted...
Writing manifest to image destination
Storing signatures
649628158f41cf64f7b155497993fc4f81a598867e9a7ee16c9f6da29e0d0976
```

4) The **podman run** command can then be used to start the container service. In this particular example the httpd service should be given some persistant storage for web content. The syntax needed for persistant storage is **-v host_location:containerlocation:Z**. The **:Z** portion on the end is to set SELinux contexts on the host_location to allow the container to use the storage provided. This also requires that the user running this command initially has write privilege to the directory being used by the container. One more option shown in this example maps a network port between the host and container. The networking syntax **-P host_port:container_port** maps a host side port to the container port. The **-d** option allows th container to run as a background process.

```
mkdir ~/Webfiles
podman run -d --name Apache -v ~/Webfiles:/var/www/html:Z -p 8080:8080
registry.redhat.io/rhscl/httpd-24-rhel7
```

5) The **podman ps** command will verify that the Apache container is running and is mapping port 8080 on the host to port 8080 on the container.

```
podman ps
CONTAINER ID  IMAGE                                           COMMAND
46a153dea911  registry.redhat.io/rhscl/httpd-24-rhel7:latest  /usr/bin/run-http...
CREATED        STATUS            PORTS                   NAMES
9 minutes ago  Up 9 minutes ago  0.0.0.0:8080->8080/tcp  Apache
```

6) The **ls -dZ ~/Webfiles** command will verify that permissions and SELinux contexts are set correctly on the persistant storage. The SELinux context should be **container_file_t**

```
ls -dZ ~/Webfiles
system_u:object_r:container_file_t:s0:c291,c319 /home/vagrant/Webfiles
```

7) The container can also be setup to run as a user systemd service. The process starts with the creation of a directory in the desired user's home directory. Changing to this working directory will make the files go into the correct location.

```
mkdir -pv /home/vagrant/.config/systemd/user
mkdir: created directory '/home/vagrant/.config/systemd'
mkdir: created directory '/home/vagrant/.config/systemd/user'
cd /home/vagrant/.config/systemd/user
```

8) The **podman generate** command will create the systemd unit files needed to have the service be managed by systemd as a daemon.

```
podman generate systemd --name Apache --files --new
/home/vagrant/.config/systemd/user/container-Apache.service
```

9) The container must be stopped and deleted before the systemctl unit can be added.

```
podman stop Apache
46a153dea91149c1a78c0d790afce67b165bd402dddaaef573716b8236307fa4
podman rm Apache
46a153dea91149c1a78c0d790afce67b165bd402dddaaef573716b8236307fa4
```

10) The **loginctl enable-linger** command will allow the service to start at boot time.

```
loginctl enable-linger
```

11) Systemd can now be instructed to enable and start the user container service.

```
systemctl --user enable container-Apache.service --now
Created symlink /home/vagrant/.config/systemd/user/multi-
user.target.wants/container-Apache.service →
/home/vagrant/.config/systemd/user/container-Apache.service.
Created symlink /home/vagrant/.config/systemd/user/default.target.wants/container-
Apache.service → /home/vagrant/.config/systemd/user/container-Apache.service.
```

References

podman(1) buildah(1) ss(8) systemctl(1) loginctl(1) https://github.com/containers/buildah/tree/master/docs/tutorials
https://docs.docker.com/develop/develop-images/dockerfile_best-practices/
https://docs.docker.com/engine/reference/builder/#from
https://opensource.com/article/18/6/getting-started-buildah
https://catalog.redhat.com/software/containers/rhscl/httpd-24-rhel7/57ea8d049c624c035f96f42e?container-tabs=gti
https://asciidoctor.org/docs/asciidoc-writers-guide/
https://asciidoctor.cn/docs/convert-asciidoc-to-pdf/