A CLOUD GURU

Lesson Guide - Creating a Container Image Using Buildah and a Dockerfile

Another way to create a custom container image is to start with a Dockerfile. In this lesson, we will use Buildah to create a container image from a Dockerfile. Upon completion of this lesson, you will be able to create a basic Dockerfile and use Buildah to create a container image from the Dockerfile.

Resources

Buildah

Getting Started with Buildah - Red Hat

Building an Image from a Dockerfile with Buildah - Red Hat

Dockerfile Reference - Docker

Best Practices for Writing Dockerfiles - Docker

What Is a Dockerfile? - CloudBees

Docker Basics: How to Use Dockerfiles - The New Stack

Instructions

Let's create our own Apache container image!

Using a Dockerfile, we're going to create our own Apache web server container image, based on the latest Fedora Linux image, and add some content to it. We'll then launch five instances of our custom `my-fedora-httpd` container image and test our work.

Let's do it!

Commands Covered

- `buildah images`: lists locally stored images
- `buildah containers`: lists containers which appear to be Buildah working containers
- `buildah bud`: builds an OCI image using instructions in one or more Dockerfiles
- `buildah rmi`: removes one or more locally stored images
- `podman ps`: displays information about containers and pods
- `podman run`: runs a command in a new container
- `podman stop`: stops one or more containers
- `podman rm`: removes one or more containers from the host

Creating a Custom Container Image Using a Dockerfile

Before we start, let's check the status of our Podman environment:

```
podman ps -a
```

```
buildah images
```

```
buildah containers
```

We're going to use the following Dockerfile to build a custom container image:

Dockerfile:

```
FROM fedora:latest
LABEL maintainer fedora-apache-container <apache@podman.rulez>

RUN dnf install -y httpd && dnf clean all

RUN echo "Test File 1" > /var/www/html/test1.txt
RUN echo "Test File 2" > /var/www/html/test2.txt
RUN echo "Test File 3" > /var/www/html/test3.txt
RUN echo "Test File 4" > /var/www/html/test4.txt
RUN echo "Test File 5" > /var/www/html/test5.txt

EXPOSE 80
CMD mkdir /run/httpd ; /usr/sbin/httpd -D FOREGROUND
```

We're going to copy the contents of our Dockerfile into a file on our system.

```
vi Dockerfile
```

Copy and paste the Dockerfile contents, then save and exit.

Let's build our custom `my-fedora-httpd` container image:

```
buildah bud -t my-fedora-httpd:latest .
```

Checking our work:

```
buildah images
```

```
buildah containers
```

We see our new `my-fedora-httpd:latest` container image! We don't see any containers, as we have already committed what we had to our new `my-fedora-httpd:latest` container image.

Running Our `my-fedora-httpd` Containers

```
podman run -d --name my-fedora-httpd-1 -p 8081:80 localhost/my-fedora-
httpd
```

```
podman run -d --name my-fedora-httpd-2 -p 8082:80 localhost/my-fedora-
httpd
```

```
podman run -d --name my-fedora-httpd-3 -p 8083:80 localhost/my-fedora-
httpd
```

```
podman run -d --name my-fedora-httpd-4 -p 8084:80 localhost/my-fedora-
httpd
```

```
podman run -d --name my-fedora-httpd-5 -p 8085:80 localhost/my-fedora-
httpd
```

Checking our work:

```
podman ps -a
```

Testing Our `my-fedora-httpd` Containers

```
curl -s http://localhost:8081/test1.txt
```

```
curl -s http://localhost:8082/test2.txt
```

```
curl -s http://localhost:8083/test3.txt
```

```
curl -s http://localhost:8084/test4.txt
```

```
curl -s http://localhost:8085/test5.txt
```

Mixing it up a little:

```
curl -s http://localhost:8085/test1.txt
```

We can also test a connection using our web browser and the IP address of our host machine, plus the port of the container we're looking to connect to (8081-8085).

Cleaning Up!

Let's clean up after ourselves!

```
podman stop -a
```

```
podman rm -a
```

```
buildah rmi -a
```

```
buildah images
```

```
podman ps -a
```

Congratulations! You just created a custom container image using a Buildah and a Dockerfile!

Notes

Recording - Environment used: Cloud Playground - Medium 3 unit RHEL 8 Cloud Server

Environment Setup:

Create your Cloud Playground server and log in.

Install the `container-tools` Application Stream:

```
sudo yum -y module install container-tools
```

You're ready to go!