A CLOUD GURU

Lesson Guide - Using Buildah to Manage Container Images - Part 1

From time to time, we will need to perform various management operations on our container images. In this lesson, we will look at how we can use Buildah to manage container images. Upon completion, you will have a solid understanding of how to manage container images using Buildah.

Resources

Buildah

Building Container Images with Buildah - Red Hat

Buildah and Podman Relationship - podman.io

Instructions

Let's take a look at the rest of Buildah!

Now that we've mastered building custom container images using Buildah, let's take a look at all the other things we can do. We're going to look at the other Buildah commands that make the experience complete.

Let's jump in!

Commands Covered

- `buildah`: tool that facilitates building OCI images

General Buildah Commands

These are commands that provide information on things in the `buildah` environment.

Commands: `help`/`info`/`inspect`/`version`

Quite possibly the most important command we're going to cover is the `buildah help` command:

```
buildah help | more
```

```
buildah --help | more
```

Either way works.

You can drill down into `buildah` subcommands:

```
buildah help --help | more
```

Don't forget about the `man` page for `buildah`:

```
man buildah
```

These two online resources should have you covered for most situations. I encourage you to dig deeper and explore Buildah through the documentation.

If you'd like to get more information about your Buildah system:

```
buildah info | more
```

We can get more information about a container or image using `buildah inspect`:

```
buildah inspect localhost/my-fedora-httpd | more
```

If you want to know what version of `buildah` is installed, run:

```
buildah version
```

This command provides detailed version information about our installed version of `buildah`.

Let's move on.

Content Management Using Buildah

These commands are used to add content (file/URL/directory) to a container.

Commands: `add`/`copy`

Let's add some more content to our `my-fedora-httpd` container image.

Let's create some files to move:

```
mkdir ~/testfiles
```

```
for i in `seq 1 5` ; do echo "Add Test File "$i > ~/testfiles/add$i.txt ;
echo "Copy Test File "$i > ~/testfiles/copy$i.txt ; done
```

```
cat ~/testfiles/*
```

We have five test files to use with the `buildah add` command, and five to use with the `buildah copy` command.

If you look at the help information for the two commands:

```
buildah add --help | more
```

```
buildah copy --help | more
```

You will notice the help contents are identical. Let's see how they behave.

First, let's copy the files using `buildah add`. We'll need to create a container first:

```
container=$(buildah from fedora:latest)
```

```
echo $container
```

We see our new container's name. Checking with `buildah containers`:

```
buildah containers
```

Adding the files:

```
buildah add fedora-working-container 'testfiles/add*.txt' '/var/www/html/'
```

Next, let's try the same using `buildah copy`:

```
buildah copy fedora-working-container 'testfiles/copy*.txt'
'/var/www/html/'
```

They appear to work the same. Remember, your source can be a file, directory, or URL. We'll view the results in a second.

Let's check out filesystem management to see them.

Filesystem Management Using Buildah

Sometimes we want to interact directly with a container's filesystem when creating or modifying container images. The following commands allow you to do this.

Commands: `mount`/`umount`

Let's try mounting our container filesystem to confirm that we were able to add/copy those files:

```
buildah unshare
```

```
buildah mount fedora-working-container
```

We see our mount point.

Let's check for those files:

```
ls -la <mountpoint>/var/www/html
```

```
cat <mountpoint>/var/www/html *.txt
```

We see our files!

Let's unmount our container filesystem and exit from our `buildah unshare` session:

```
buildah umount fedora-working-container
```

```
exit
```

Great work, Cloud Gurus! We're going to continue in Part 2.

Notes

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

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
```

Environment Setup:

Create your Cloud Playground server and log in.

Install the `container-tools` Application Stream:

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

Copy the contents of the Dockerfile into a file called `Dockerfile`.

Create a container image using your Dockerfile:

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

Check your work:

```
buildah images
```

Set Up a Container Registry

How to Implement a Simple Personal/Private Linux Container Image Registry for Internal Use - Red Hat

I used `localhost` for the hostname, as I'm going to access the container registry using `localhost`.

Steps to deploy:

Become `root`:

```
sudo -i
```

Install `httpd-tools`:

```
yum install -y httpd-tools
```

Create the directories for our container registry:

```
mkdir -p /opt/registry/{auth,certs,data}
```

Create our container registry credentials:

```
htpasswd -bBc /opt/registry/auth/htpasswd cloud_user registry
```

Generate our SSL certificate. I used `localhost` for the hostname, as I'm going to access the container registry using `localhost`:

```
openssl req -newkey rsa:4096 -nodes -sha256 -keyout
/opt/registry/certs/domain.key -x509 -days 365 -out
/opt/registry/certs/domain.crt
```

Add our certificate:

```
cp /opt/registry/certs/domain.crt /etc/pki/ca-trust/source/anchors/
```

Update our CA certificates:

```
update-ca-trust
```

Check your work:

```
trust list | grep -i localhost
```

Start your container registry:

```
podman run --name myregistry -p 5000:5000 -v
/opt/registry/data:/var/lib/registry:z -v /opt/registry/auth:/auth:z -e
"REGISTRY_AUTH=htpasswd" -e "REGISTRY_AUTH_HTPASSWD_REALM=Registry Realm"
-e REGISTRY_AUTH_HTPASSWD_PATH=/auth/htpasswd -v
/opt/registry/certs:/certs:z -e
"REGISTRY_HTTP_TLS_CERTIFICATE=/certs/domain.crt" -e
"REGISTRY_HTTP_TLS_KEY=/certs/domain.key" -e
REGISTRY_COMPATIBILITY_SCHEMA1_ENABLED=true -d
docker.io/library/registry:latest
```

Test your container registry:

```
curl -u cloud_user:registry https://localhost:5000/v2/_catalog
```

Exit `root`:

```
exit
```

You're ready to go!