A CLOUD GURU

Lesson Guide - Managing Pods Using Podman

One constant in the universe is change, and you can be sure that once you get your pod(s) up and running you will need to manage them. In this lesson, we're going to go beyond simply running pods and will examine how to manage our pod(s) after we bring them into existence. When you are finished with this lesson you should have a solid understanding of how to manage pods using Podman.

Resources

Building, Running, and Managing Linux Containers on Red Hat Enterprise Linux 8 - Red Hat

Instructions

It's time to take charge of our Podman pods!

We've been put in charge of both a rootless and rootfull pod, and now need to use Podman to manage these pods. We'll take a look at the most used podman pod commands, those used to manipulate pod resources, commands used to change the state of the pod, and commands used for monitoring and information gathering.

Let's seize the day!

Commands Covered

- `podman pod`: manages Podman pods
- `podman ps`: displays information about containers and pods

Working With Rootless Pods

Commands Used: start/stop/restart/kill/inspect/ps/top

We're going to take a look at some of the common commands we can use with both rootless and rootfull containers.

Let's take a look at our running pods and containers:

```
podman pod ps
```

```
podman ps -a --pod
```

We see our `wp-pod` pod, with the `wp-web` and `wp-db` containers, as well as our Infra container. Everything is up and running.

We can stop the pod, along with all its containers, using the `podman pod stop` command:

```
podman pod stop wp-pod
```

Checking the status of our pod and its containers:

```
podman ps -a --pod
```

We can see our `wp-pod` pod, and its containers, stopped.

We can start our pod using `podman pod start`:

```
podman pod start wp-pod
```

Checking the status of our pod and its containers:

```
podman ps -a --pod
```

We see that our `wp-pod` and its containers are back up!

Similarly, we can restart our pod using `podman pod restart`:

```
podman pod restart wp-pod
```

Checking the status of our pod and its containers:

```
podman ps -a --pod
```

We see that our `wp-pod` and its containers are back up!

We can get information about our pod using `podman pod inspect`:

```
podman pod inspect wp-pod
```

This provides detailed information on our `wp-pod` pod.

For a listing of the `wp-pod` pod's processes, use:

```
podman pod top wp-pod
```

Now that we've covered the `podman pod` commands that work with both rootless and rootfull pods, let's take a look at some operations for rootfull pods.

Working With Rootful Pods

Commands Used: create/prune/pause/unpause/stats

There are a number of additional commands we can use with rootfull pods. Before we can explore these, we need to deploy a rootful pod.

Deploy a Rootful Pod:

Become `root`:

```
sudo -i
```

We're going to create our rootfull pod, then add one `nginx` container and one `mariadb` container. We want to publish port `80` in the pod to `8081` on the host. We want to name the pod `root-pod`.

To do this, we run:

```
podman pod create --name root-pod -p 8081:80
```

Again, we can use the `podman pod create` command with rootless containers as well.

Next, let's start a `mariadb` container:

```
podman run -d --restart=always --pod=root-pod -e
MYSQL_ROOT_PASSWORD="dbpass" -e MYSQL_DATABASE="rootdb" -e
MYSQL_USER="dba" -e MYSQL_PASSWORD="dbapass" --name=root-db mariadb
```

Finally, we'll start the `nginx` container:

```
podman run -d --restart=always --pod=root-pod --name root-web nginx
```

Checking our pods and containers:

```
podman ps -a --pod
```

Our rootfull pod is fully running!

Checking with a `curl` command:

```
curl -s http://localhost:8081
```

We see the default `nginx` index page!

Manage a rootfull pod:

We can pause rootfull pods:

```
podman pod pause root-pod
```

Checking our pods and containers:

```
podman ps -a --pod
```

```
podman pod ps
```

We see that both our pod and its containers are in the `paused` status.

We can resume our pod and its containers with the `podman pod unpause` command:

```
podman pod unpause root-pod
```

Checking our pods and containers:

```
podman ps -a --pod
```

```
podman pod ps
```

We see that our `root-pod` pod and its containers are running again!

We can get performance statistics for rootful pods and its containers using:

```
podman pod stats root-pod
```

We can see the statistics for our `root-pod` pod and its containers. We can use `CTRL-C` to break out of the statistics display.

Let's stop our `root-pod` pod:

```
podman pod stop root-pod
```

Checking for reclaimable space:

```
podman system df
```

We see we have some reclaimable space.

Let's try the `podman pod prune` command to reclaim that space:

```
podman pod prune
```

We can see we freed up some space.

Checking our pods and containers:

```
podman ps -a --pod
```

```
podman pod ps
```

There are no more rootfull containers or pods!

Let's exit `root`:

```
exit
```

Cleaning Up Our Rootless Pods and Containers

Commands Used: rm/prune

Let's clean up our rootless pod and its containers:

Checking our pods and containers:

```
podman ps -a --pod
```

```
podman pod ps
```

We see our `wp-pod` pod and its containers.

Let's stop and remove the `wp-pod` pod and its containers:

```
podman pod stop wp-pod
```

```
podman pod rm wp-pod
```

Checking our pods and containers one more time:

```
podman ps -a --pod
```

```
podman pod ps
```

The `wp-pod` pod and its containers have been removed.

Let's clean up the rest of the mess:

```
podman system prune -a
```

Checking our work:

```
podman system df
```

We have no reclaimable space left.

Excellent work!

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
```

Deploy a Rootless WordPress Pod:

We'll start by creating our pod. Remember, we want to publish port `80` in the pod to `8080` on the host. We want to name the pod `wp-pod`.

To do this, we run:

```
podman pod create --name wp-pod -p 8080:80
```

Next, let's start the `mariadb` container:

```
podman run -d --restart=always --pod=wp-pod -e
MYSQL_ROOT_PASSWORD="dbpass" -e MYSQL_DATABASE="wp" -e
MYSQL_USER="wordpress" -e MYSQL_PASSWORD="wppass" --name=wp-db mariadb
```

Finally, we'll start the WordPress container:

```
podman run -d --restart=always --pod=wp-pod -e WORDPRESS_DB_NAME="wp" -e
WORDPRESS_DB_USER="wordpress" -e WORDPRESS_DB_PASSWORD="wppass" -e
WORDPRESS_DB_HOST="127.0.0.1" --name wp-web wordpress
```

Checking for containers:

```
podman ps -a --pod
```

Our WordPress pod is fully running!

Checking with a `curl` command:

```
curl -s http://localhost:8080
```

```
echo $?
```

We can see that the WordPress login page is working! We can now log in with our web browser, using port 8080, if we want.

You're ready to go!