A CLOUD GURU

Lesson Guide - Monitoring Containers Using Podman

Sometimes things go wrong with containers, even in the hands of the most experienced engineers. In this lesson, we're going to examine how we can monitor containers and the status of our Podman environment — for planning, troubleshooting, and to just have a grip on what's going on. Upon completion of this lesson, you will have the knowledge and skills required to monitor your Podman containers and environment.

Resources

Monitoring Container Vitality and Availability with Podman - Red Hat Developer

Building, Running, and Managing Containers - Monitoring Containers - Red Hat

Instructions

Our containers need a checkup!

It's one thing to get a bunch of containers up and running, but managing them is more than executing commands. We're going to create `healthchecks` to assure that our containers don't fall ill. In addition, we're going to take a look at how to monitor the Podman environment.

Put your gloves on and let's get started!

Commands Covered

- `podman run`: runs a command in a new container from the given image
- `podman inspect`: displays information on an object identified by name or ID
- `podman healthcheck`: manages healthchecks on containers
- `podman events`: monitors Podman events
- `podman system`: used to manage Podman

Using Container Healthchecks

Let's take a look at a simple `healthcheck` using a `nginx` container. We're going to monitor the health of the web server inside our container.

Launch the `nginx` container:

```
podman run -dt --name nginx1 --health-cmd 'curl http://localhost || exit
1' --health-interval=0 nginx
```

Checking the status of our containers:

```
podman ps -a
```

Now we can run our healthcheck:

```
podman healthcheck run nginx1
```

We can see that `nginx` is healthy and serving its default web page.

If we check the exit code:

```
echo $?
```

We see it's zero.

Let's take a look at what a `healthcheck` failure looks like. We're going to start two `ubi` containers, one `ubi7` and one `ubi8`. We're going to run the same `healthcheck` against both, one that checks for the character `8` in the file `/etc/redhat-release`.

Let's start the two containers, with the `healthcheck`:

```
podman run -dt --name myubi7 --health-cmd 'grep 8 /etc/redhat-release ||
exit 1' --health-interval=0 ubi7:latest
```

```
podman run -dt --name myubi8 --health-cmd 'grep 8 /etc/redhat-release ||
exit 1' --health-interval=0 ubi8:latest
```

Checking the status of our containers:

```
podman ps -a
```

Now, let's run our healthchecks, starting with the `ubi8` container:

```
podman healthcheck run myubi8
```

If we check the exit code:

```
echo $?
```

We see it's zero.

Running the `healthcheck` command manually:

```
podman exec -i myubi8 grep 8 /etc/redhat-release
```

Now, the `ubi7` container:

```
podman healthcheck run myubi7
```

If we check the exit code:

```
echo $?
```

We see it's non-zero.

Running the `healthcheck` command manually:

```
podman exec -i myubi7 grep 8 /etc/redhat-release
```

The expected behavior occurs. Since there's an `8` in `/etc/redhat-release`, the check succeeds on the `ubi8` container, but fails on the `ubi7` container.

Let's wrap up `healthchecks` and move on to managing Podman itself.

Managing the Podman System

Managing containers is only one part of keeping an eye on your Podman environment. Podman has the `podman system` command, which is used to manage Podman:

```
podman system --help
```

A great place to start is with the `podman system info` command:

```
podman system info | more
```

This gives us information about the system that Podman is installed and running on. This can be handy for troubleshooting.

We can get a report on Podman's disk space utilization using `podman system df`:

```
podman system df
```

This gives us a short summary, and includes information on reclaimable space.

If we'd like more detailed information, we can add the `-v` switch:

```
podman system df -v
```

This gives us a summary of storage utilization, including individual containers and images.

You can clean up unused data using the `podman system prune` command:

```
podman system prune -a
```

Checking our Podman disk utilization now:

```
podman system df
```

We can see that extra space has been reclaimed.

I encourage you to dig down further into the `podman system` command, using `podman system --help` or `man podman-system`.

On to events!

Working With Podman Events

Sometimes we need to look into the past and see what our Podman environment has been up to. Fortunately for us, Podman has provided the `podman events` command:

```
podman events --help
```

The `podman events` command without any arguments will produce live streaming output. If we'd like to take a look at events that have happened in the past, we can use the `--since` option.

Let's take a look at events from the last ten minutes:

```
podman events --since 10m
```

We can use `CTRL-C` to break from the command.

Maybe we want to filter our output. Let's take a look:

```
podman events --since 15m --filter event=prune
```

We can see all `prune` events for the past 15 minutes.

The `podman events` command is straightforward and easy to use.

Great job, Cloud Gurus!

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
```

Next, let's pull a container image that we're not going to use:

```
podman pull httpd
```

We're going to use this for our `podman system prune` operation.

You're ready to go!