A CLOUD GURU

Lesson Guide - Managing Container Networking Using Podman

Sometimes the application(s) running in our container(s) will want to communicate with the outside world via the network. In this lesson, we’re going to examine how we can enable an application or service in our container to be available via the network. Upon completion of this lesson, you should be able to configure container networking in Podman.

Resources

Podman.io - Networking

Configuring Container Networking with Podman

Understand Networking in Podman

Exposing Podman Containers Fully on the Network

Installing Nextcloud 20 on Fedora Linux with Podman

Instructions

We need to access our `nginx` containers!

Up to this point, we’ve been using nginx containers with great success, but without network access. We’re going to take a look at a number of ways that we can manage container networking, using both rootless and rootfull containers.

Let’s go!

Commands Covered

- `podman network`: execute Podman networking commands
- `podman port`: display published Podman ports
- `podman inspect`: display detailed information on an object

Podman Networking - Rootless Containers

Generally, when we want to make a service or application available on Podman rootless containers, we do it by publishing that application's ports.

Let's say we want to deploy an `nginx` container and we want to be able to access the web server from outside the container. We can publish the container's ports, which makes them available. One way to do this is to have `podman` handle it automatically, using the `-P` or `--publish-all` option. Let's check it out.

```
podman run -dt --name web1 --publish-all nginx
```

Checking our containers:

```
podman ps -a
```

We see our running container, and that port `80` on the container has been published to a *random* port that's been chosen by `podman`.

Let's try to connect to our web server on the port that `podman` has chosen for us:

```
curl http://localhost:<port>
```

If we'd like to view all published ports, we can use:

```
podman port -a
```

This will display a list off all published ports for the user's containers.

Most of the time, we will want to specify which ports we are publishing to. We can use the `-p` or `--publish` option.

Let's start another `nginx` container, but publish port `80` in the container to port `8080` on the host:

```
podman run -dt --name web2 -p 8080:80 nginx
```

Checking our containers:

```
podman ps -a
```

Checking our published ports:

```
podman port -a
```

Let's start one more `nginx` container, but publish port `80` in the container to port `8081` on the host:

```
podman run -dt --name web3 -p 8081:80 nginx
```

Checking our containers:

```
podman ps -a
```

Checking our published ports:

```
podman port -a
```

Let's try to connect to our just-launched web servers on ports `8080` and `8081`:

```
curl http://localhost:8080
```

```
curl http://localhost:8081
```

We can access the `nginx` web servers via ports `8080` and `8081`!

Podman Networking - Rootfull Containers

Podman containers that are run with `root` privileges, either directly or using `sudo`, enjoy additional network functionality. These containers are given an IP address.

Let's become `root` and explore:

```
sudo su -
```

Podman gives us a list of its networks using:

```
podman network ls
```

We can see the default `podman` network.

If we'd like to get information about the `podman` network, we can use:

```
podman network inspect podman | more
```

Podman returns detailed information, including the network, routing and firewall information.

Let's launch a container and see how it works:

```
podman run -dt --name rootweb1 --publish-all nginx
```

Checking our containers:

```
podman ps -a
```

Checking our published ports:

```
podman port -a
```

We can see our container and the *random* port that our `nginx` web server has been published to.

Let's try to connect to our web server on the port that `podman` has chosen for us:

```
curl http://localhost:<port>
```

We see our default index page for `nginx`.

If we want to get the IP address of our `rootweb1` container, we can use:

```
podman inspect rootweb1 | grep IPAddress
```

Now, we can try to access our `nginx` web server using the IP address of our container:

```
curl http://<ip_address>
```

Once again, we see our default index page for `nginx`.

If we wanted to create a second `podman` network called `test-net`, we can do that using:

```
podman network create test-net
```

Pulling a list of its networks again:

```
podman network ls
```

We can see our new network, `test-net`.

We can connect our `rootweb1` container to our new `test-net` network using:

```
podman network connect test-net rootweb1
```

Checking our `rootweb1` container's IP addresses now:

```
podman inspect rootweb1 | grep IPAddress
```

We can see two IP addresses now!

Accessing our web server via the new IP address:

```
curl http://<ip_address>
```

Once again, we see our default index page for `nginx`, using our IP address on the `test-net` network.

Le's disconnect and remove the `test-net` network:

```
podman network disconnect test-net rootweb1
```

```
podman network rm test-net
```

Checking our networks now:

```
podman network ls
```

We see our `test-net` network has been removed.

Looking good!

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!