

Lesson Guide - Configuring Persistent `systemd` Containers and Pods

Podman has the ability to generate `systemd` unit files, making quick work of configuring `systemd` containers. In this lesson, we will examine how to use Podman to generate `systemd` unit files using existing Podman containers. Once you have successfully generated `systemd` unit files from your Podman containers, you will want to configure them for persistence. Upon completion of this lesson, you will understand how to use Podman to generate `systemd` unit files from existing containers and pods and configure persistent `systemd` containers using the `systemd` unit files you generated with Podman.

Resources

[Porting Containers to systemd Using Podman - Red Hat](#)

Instructions

Our Web Team needs some WordPress pods!

Each team member on the Web Team has received a new development laptop running RHEL. They would like us to help set up a `systemd` WordPress pod for each team member, which they can use for web development.

How do we do that?

Commands Covered

- `podman generate systemd`: generates `systemd` unit files
- `podman ps -a --pod`: prints out information about containers
- `podman pod ps`: lists all pods on system including their names, IDs, and current state
- `podman pod create`: creates a new empty pod
- `podman pod stop`: stops one or more pods
- `podman pod rm`: removes one or more pods
- `podman run`: runs a command in a new container
- `systemctl --user`: queries or sends control commands to the `systemd` manager
- `loginctl`: sends control commands to or queries the login manager

Start Our WordPress Pod

We're going to stand up a WordPress instance in a pod. Let's see how it's done!

Let's start by checking for existing containers and pods. We can use the `--pod` option to show the ID and name of the pod that the containers belong to:

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

We can display pods using:

```
podman pod ps
```

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
```

First, 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
```

Then, 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
```

Test Our WordPress Pod

Checking for containers again:

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

And pods:

```
podman pod ps
```

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.

Configure the User's `systemd` Environment

Create the `~/.config/systemd/user` directory:

```
mkdir -p ~/.config/systemd/user
```

Change directory to the new directory:

```
cd ~/.config/systemd/user
```

Generate Our `systemd` Unit Files

Before we actually generate our file, let's check out the `help` information for the `podman generate systemd` command:

```
podman generate systemd --help
```

We're going to generate our `systemd` unit files from our running pod:

```
podman generate systemd --files --new --name wp-pod
```

We can see that three files were created, one for our pod, and one for each of our containers.

Let's take a look at our unit files:

```
more *.service
```

Shut Down and Clean Up Our Donor WordPress Pod

Before we can start our new pod using **systemd**, we need to shut down our "donor" pod and remove it:

```
podman pod stop -a
```

```
podman pod rm -a
```

Checking for containers again:

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

And checking for pods:

```
podman pod ps
```

Enable and Start Our WordPress Pod Using **systemd**

Reload **systemd** to pick up the new unit:

```
systemctl --user daemon-reload
```

Enable and start the container service as the user:

```
systemctl --user enable --now pod-wp-pod
```

Checking the status of our new **systemd** WordPress pod:

```
systemctl --user status pod-wp-pod
```

Test Our WordPress Pod

Checking for containers again:

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

Checking for pods:

```
podman pod ps
```

Our new 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.

Make Our WordPress Pod Persistent

We're going to configure our container to start when the system boots.

Checking the setting for `linger` for `cloud_user`:

```
loginctl show-user cloud_user | grep -i linger
```

Enable `linger` for `cloud_user`:

```
loginctl enable-linger
```

Checking the setting for `linger` for `cloud_user` again:

```
loginctl show-user cloud_user | grep -i linger
```

Test Persistence

Let's reboot and test our pod for persistence:

```
sudo systemctl reboot
```

Checking for containers again:

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

Checking for pods:

```
podman pod ps
```

Our new 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.

Congratulations, Cloud Guru! You just created a persistent **systemd pod!**

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!