Chapter 2. Podman Basics

Abstract

Goal	Manage and run containers with Podman.
Objectives	• Run a containerized service with Podman. • Describe how containers communicate with each other. • Expose ports to access containerized services. • Explore running containers. • List, stop, and delete containers with Podman.
Sections	• Creating Containers with Podman (and Guided Exercise) • Container Networking Basics (and Quiz) • Accessing Containerized Network Services (and Guided Exercise) • Accessing Containers (and Guided Exercise) • Managing the Container Lifecycle (and Guided Exercise)
Lab	• Podman Basics

Creating Containers with Podman

Objectives

- Run a containerized service with Podman.

An Introduction to Podman

Podman is an open source tool that you can use to manage your containers locally. With Podman, you can find, run, build or deploy OCI (Open Container Initiative) containers and container images.

By default, Podman is daemonless. A daemon is a process that is always running and ready for receiving incoming requests. Some other container tools use a daemon to proxy the requests, which brings a single point of failure. In addition, a daemon might require elevated privileges, which is a security concern. Podman interacts directly with containers, images, and registries without a daemon.

Podman comes in the form of a command-line interface (CLI), which is supported for several operating systems. Along with the CLI, Podman provides two additional ways to interact with your containers and automate processes, the RESTful API and a desktop application called *Podman Desktop*.

Working with Podman

After you install Podman, you can use it by running the `podman` command. The following command displays the version that you are using.

```
[user@host ~]$ podman -v
podman version VERSION
```

Pulling and Displaying Images

Before you can run your application in a container, you must create a container image.

With Podman, you fetch container images from image registries by using the `podman pull` command. For example, the following command fetches a containerized version of Red Hat Enterprise Linux 7 from the Red Hat Registry.

```
[user@host ~]$ podman pull registry.redhat.io/rhel7/rhel:7.9
Trying to pull registry.redhat.io/rhel7/rhel:7.9...
Getting image source signatures
...output omitted...
Writing manifest to image destination
Storing signatures
b85986059f7663c1b89431f74cdcb783f6540823e4b85c334d271f2f2d8e06d6
```

A container image is referenced in the form `NAME:VERSION`. In the previous example, you fetch the `7.9` version of the `registry.redhat.io/rhel7/rhel` image.

After the execution of the `pull` command, the image is stored locally in your system. You can list the images in your system by using the `podman images` command.

```
[user@host ~]$ podman images
REPOSITORY                        TAG         IMAGE ID      CREATED       SIZE
registry.redhat.io/rhel7/rhel     7.9         52617ef413bd  4 weeks ago   216 MB
```

Container and Container Images

A container is an isolated runtime environment where applications are executed as isolated processes. The isolation of the runtime environment ensures that they do not interrupt other containers or system processes.

A container image contains a packaged version of your application, with all the dependencies necessary for the application to run. Images can exist without containers, but containers are dependent of images because containers use container images to build a runtime environment to execute applications.

Running and Displaying Containers

With the image stored in your local system, you can use the `podman run` command to create a new container that uses the image. The RHEL image from previous examples accepts Bash commands as an argument. These commands are provided as an argument and are executed within a RHEL container.

```
[user@host ~]$ podman run registry.redhat.io/rhel7/rhel:7.9 echo 'Red Hat'
Red Hat
```

In the previous example, the `echo 'Red Hat'` command is provided as an argument to the `podman run` command. Podman executes the `echo` command inside the RHEL container and displays the output of the command.

NOTE

If you run a container from an image that is not stored in your system then Podman tries to pull the image before running the container. Therefore, it is not necessary to execute the `pull` command first.

When the container finishes the execution of `echo`, the container is stopped because no other process keeps it running. You can list the running containers by using the `podman ps` command.

```
[user@host ~]$ podman ps
CONTAINER ID  IMAGE       COMMAND     CREATED     STATUS      PORTS       NAMES
```

By default, the `podman ps` command lists the following details for your containers.

- the container's ID
- the name of the image that the container is using
- the command that the container is executing
- the time that the container was created
- the status of the container
- the exposed ports in the container

- the name of the container.

However, *stopping* a container is not the same as *removing* a container. Although the container is stopped, Podman does not remove it. You can list all containers (running and stopped) by adding the `--all` flag to the `podman ps` command.

```
[user@host ~]$ podman ps --all
CONTAINER ID  IMAGE                  COMMAND       CREATED  STATUS    PORTS  NAMES
20236410bcef  registry.redhat.io...  echo Red Hat  ...      Exited...        ...
```

You can also automatically remove a container when it exits by adding the `--rm` option to the `podman run` command.

```
[user@host ~]$ podman run --rm registry.redhat.io/rhel7/rhel:7.9 echo 'Red Hat'
Red Hat
[user@host ~]$ podman ps --all
CONTAINER ID     IMAGE     COMMAND     CREATED       STATUS      PORTS     NAMES
```

The container lifecycle is covered later in this course.

If a name is not provided during the creation of a container, then Podman generates a random string name for the container. It is important to define a unique name to facilitate the identification of your containers when managing their lifecycle.

You can assign a name to your containers by adding the `--name` flag to the `podman run` command.

```
[user@host ~]$ podman run --name rhel7 \
registry.redhat.io/rhel7/rhel:7.9 echo 'Red Hat'
Red Hat
```

Podman can identify the containers either by the *Universal Unique Identifier (UUID)* short identifier, which is composed of twelve alphanumeric characters, or by the UUID long identifier, which is composed of 64 alphanumeric characters, as shown in the example.

```
[user@host ~]$ podman ps --all
CONTAINER ID  IMAGE              COMMAND       CREATED  STATUS    PORTS  NAMES
20236410bcef  .../rhel7/rhel:7.9 echo Red Hat  ...      Exited...        rhel7
```

In this example, `20236410bcef` is the container's ID, or UUID short identifier. Additionally, `rhel7` is listed as the container's name.

If you want to retrieve the UUID long container ID, then you can add the `--format=json` flag to the `podman ps --all` command.

```
[user@host ~]$ podman ps --all --format=json
{
    "AutoRemove": false,
    "Command": [
      "echo",
      "Red Hat"
    ],
...output omitted...
    "Id": "2023...b0b2",  1
    "Image": "registry.redhat.io/rhel7/rhel:7.9",
...output omitted...
```

1 UUID long container ID.

You can retrieve information about a container in either the JSON format or as a Go template.

Exposing Containers

Many applications, such as web servers or databases, keep running indefinitely waiting for connections. Therefore, the containers for these applications must run indefinitely. At the same time, it is usually necessary for these applications to be accessed externally through a network protocol.

You can use the `-p` option to map a port in your local machine to a port inside the container. This way, the traffic in your local port is forwarded to the port inside the container, thus, allowing you to access the application from your computer.

The following example creates a new container that runs an Apache HTTP server by mapping the 8080 port in your local machine to the 8080 port inside the container.

```
[user@host ~]$ podman run -p 8080:8080 \
 registry.access.redhat.com/ubi8/httpd-24:latest
...output omitted...
[Thu Apr 21 12:58:57.048491 2022] [ssl:warn] [pid 1:tid 140257793613248] AH01909: 10.0.2.100:8443:0 server certificate does N
OT include an ID which matches the server name
[Thu Apr 21 12:58:57.048899 2022] [:notice] [pid 1:tid 140257793613248] ModSecurity for Apache/2.9.2 (http://www.modsecurity.
org/) configured.
...output omitted...
[Thu Apr 21 12:58:57.136272 2022] [mpm_event:notice] [pid 1:tid 140257793613248] AH00489: Apache/2.4.37 (Red Hat Enterprise L
inux) OpenSSL/1.1.1k configured -- resuming normal operations
[Thu Apr 21 12:58:57.136332 2022] [core:notice] [pid 1:tid 140257793613248] AH00094: Command line: 'httpd -D FOREGROUND'
```

You can access the HTTP server at `http://localhost:8080`.

If you want the container to run in the background, to avoid the terminal being blocked, then you can use the `-d` option.

```
[user@host ~]$ podman run -d -p 8080:8080 \
  registry.access.redhat.com/ubi8/httpd-24:latest
b7eb467781106e4f416ba79cede91152239bfc74f6a570c6d70baa4c64fa636a
```

The networking capabilities of Podman are covered later in this course.

Using Environment Variables

Environment variables are variables used in your applications that are set outside of the program. The operating system or the environment where the application runs provides the value of the variable. You can access the environment variable in your application at runtime. For example, in Node.js, you can access environment variables by using `process.env.`*`VARIABLE_NAME`*.

Environment variables are a useful and safe way of injecting environment-specific configuration values into your application. For example, your application might use a database hostname that is different for each application environment, such as the `database.local`, `database.stage`, or `database.test` hostnames.

You can pass environment variables to a container by using the `-e` option. In the following example, an environment variable called `NAME` with the value `Red Hat` is passed. Then, the environment variable is printed by using the `printenv` command inside the container.

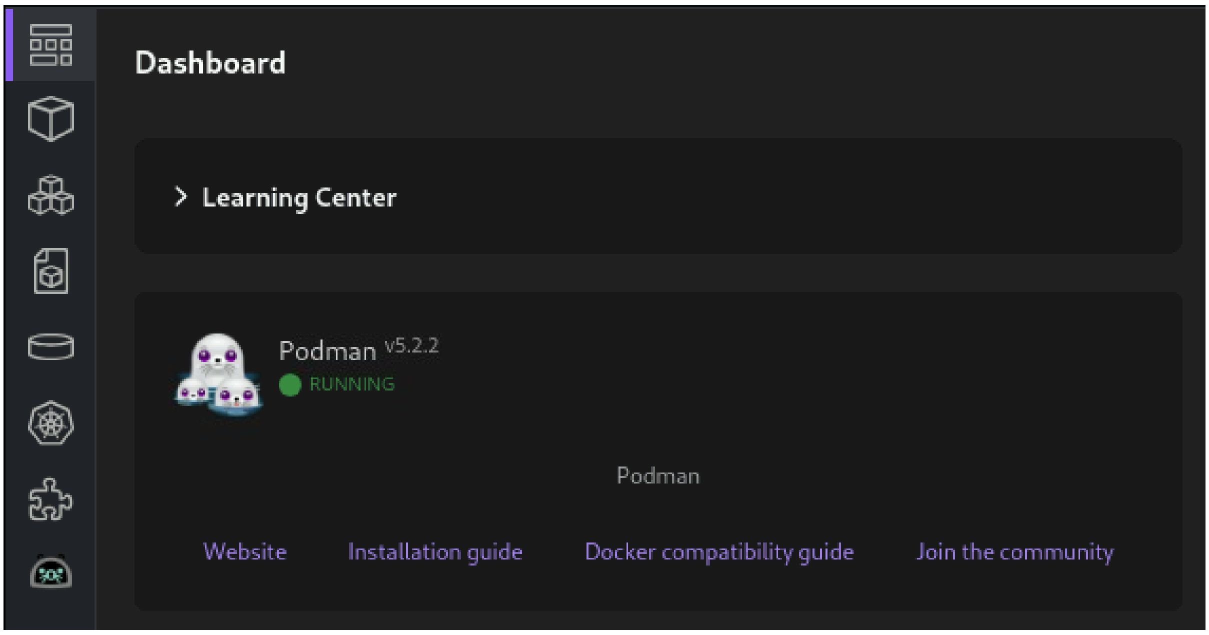
```
[user@host ~]$ podman run -e NAME='Red Hat' \
  registry.redhat.io/rhel7/rhel:7.9 printenv NAME
Red Hat
```

Environment variables are covered later in this course.

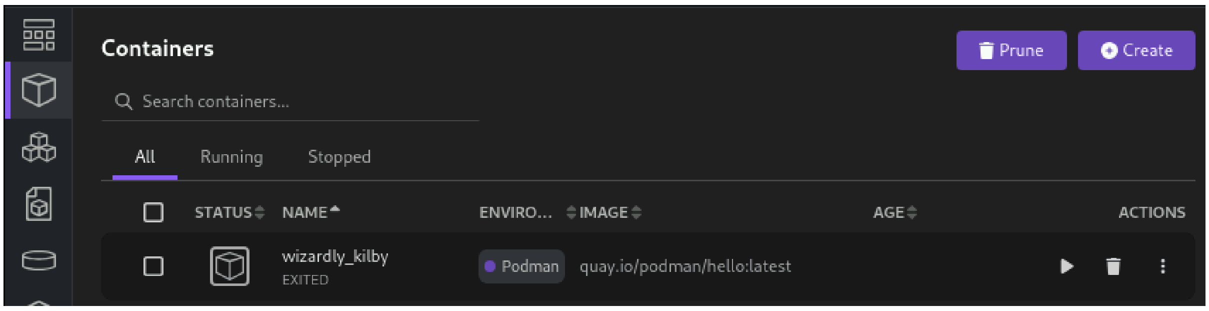
Podman Desktop

Podman Desktop is a graphical user interface, which is used to manage and interact with containers in local environments. It uses the Podman engine by default, and supports other container engines as well, such as Docker.

On launch, Podman Desktop displays a dashboard with information about the status of the Podman engine.

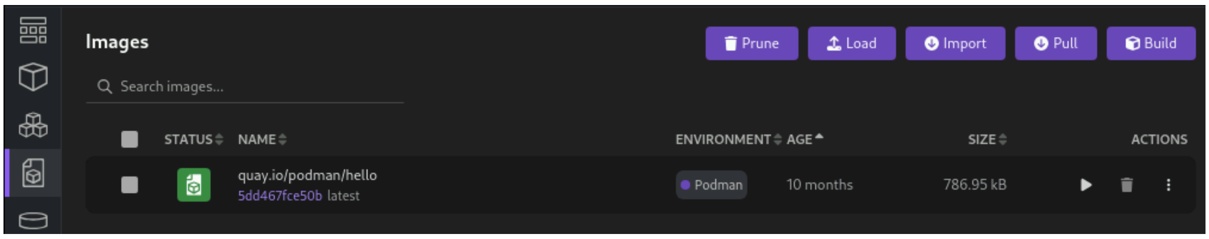


With Podman Desktop, you can perform many of the tasks that you can do with the `podman` CLI, such as pulling images and creating containers. For example, you can create, list, and run containers from the `Containers` section, as the following image shows:



STATUS	NAME	ENVIRO...	IMAGE	AGE	ACTIONS
	wizardly_kilby EXITED	Podman	quay.io/podman/hello:latest		

Similarly, you can pull, list images, and create containers from those images in the `Images` section.



STATUS	NAME	ENVIRONMENT	AGE	SIZE	ACTIONS
	quay.io/podman/hello 5dd467fce50b latest	Podman	10 months	786.95 kB	

Podman Desktop is an addition to the `podman` CLI, rather than a replacement. For more advanced commands or options, which are covered later in the course, the CLI is required. Still, Podman Desktop can be useful for users who prefer graphical environments for common tasks, and for beginners who are learning about containers.

Podman Desktop is modular and extensible. You can use and create extensions that provide additional capabilities. For example, with the *Red Hat OpenShift* extension, Podman Desktop can deploy containers to Red Hat OpenShift Container Platform.

Podman Desktop is available for Linux, MacOS, and Windows. For specific installation instructions, refer to the Podman Desktop documentation.

REFERENCES

Podman Official Documentation

Formatting output with Podman

Podman Desktop Documentation