Podman

team

Oct 31, 2022

CONTENTS:

Podman is a daemonless, open source, Linux native tool designed to make it easy to find, run, build, share and deploy applications using Open Containers Initiative (OCI) Containers and Container Images. Podman provides a command line interface (CLI) familiar to anyone who has used the Docker Container Engine. Most users can simply alias Docker to Podman (*alias docker=podman*) without any problems. Similar to other common Container Engines (Docker, CRI-O, containerd), Podman relies on an OCI compliant Container Runtime (runc, crun, runv, etc) to interface with the operating system and create the running containers. This makes the running containers created by Podman nearly indistinguishable from those created by any other common container engine.

Containers under the control of Podman can either be run by root or by a non-privileged user. Podman manages the entire container ecosystem which includes pods, containers, container images, and container volumes using the libpod library. Podman specializes in all of the commands and functions that help you to maintain and modify OCI container images, such as pulling and tagging. It allows you to create, run, and maintain those containers and container images in a production environment.

There is a RESTFul API to manage containers. We also have a remote Podman client that can interact with the RESTFul service. We currently support clients on Linux, Mac, and Windows. The RESTFul service is only supported on Linux.

If you are completely new to containers, we recommend that you check out the *Introduction*. For power users or those coming from Docker, check out our *Tutorials*. For advanced users and contributors, you can get very detailed information about the Podman CLI by looking at our *Commands* page. Finally, for Developers looking at how to interact with the Podman API, please see our API documentation *Reference*.

CHAPTER

ONE

INTRODUCTION

Containers simplify the production, distribution, discoverability, and usage of applications with all of their dependencies and default configuration files. Users test drive or deploy a new application with one or two commands instead of following pages of installation instructions. Here's how to find your first Container Image:

```
podman search busybox
```

Output:

```
INDEX       NAME                                  DESCRIPTION                                        ␣
↪          STARS    OFFICIAL    AUTOMATED
docker.io   docker.io/library/busybox             Busybox base image.                                ␣
↪          1882     [OK]
docker.io   docker.io/radial/busyboxplus          Full-chain, Internet enabled, busybox␣
↪made f...   30                  [OK]
docker.io   docker.io/yauritux/busybox-curl       Busybox with CURL                                  ␣
↪          8
...
```

The previous command returned a list of publicly available container images on DockerHub. These container images are easy to consume, but of differing levels of quality and maintenance. Let's use the first one listed because it seems to be well maintained.

To run the busybox container image, it's just a single command:

```
podman run -it docker.io/library/busybox
```

Output:

```
/ #
```

You can poke around in the busybox container for a while, but you'll quickly find that running small container with a few Linux utilities in it provides limited value, so exit out:

```
exit
```

There's an old saying that "nobody runs an operating system just to run an operating system" and the same is true with containers. It's the workload running on top of an operating system or in a container that's interesting and valuable.

Sometimes we can find a publicly available container image for the exact workload we're looking for and it will already be packaged exactly how we want. But, more often than not, there's something that we want to add, remove, or customize. It could be as simple as a configuration setting for security or performance, or as complex as adding a complex workload. Either way, containers make it fairly easy to make the changes we need.

Container Images aren't actually images, they're repositories often made up of multiple layers. These layers can easily be added, saved, and shared with others by using a Containerfile (Dockerfile). This single file often contains all the instructions needed to build a new container image and can easily be shared with others publicly using tools like GitHub.

Here's an example of how to build a Nginx web server on top of a Debian base image using the Dockerfile maintained by Nginx and published in GitHub:

```
podman build -t nginx https://git.io/Jf8ol
```

Once, the image build completes, it's easy to run the new image from our local cache:

```
podman run -d -p 8080:80 nginx
curl localhost:8080
```

Output:

```
...
<p><em>Thank you for using nginx.</em></p>
...
```

Building new images is great, but sharing our work with others let's them review our work, critique how we built them, and offer improved versions. Our newly built Nginx image could be published at quay.io or docker.io to share it with the world. Everything needed to run the Nginx application is provided in the container image. Others could easily pull it down and use it, or make improvements to it.

Standardizing on container images and Container Registries enable a new level of collaboration through simple consumption. This simple consumption model is possible because every major Container Engine and Registry Server uses the Open Containers Initiative (OCI) format. This allows users to find, run, build, share and deploy containers anywhere they want. Podman and other Container Engines like CRI-O, Docker, or containerd can create and consume container images from docker.io, quay.io, an on premise registry or even one provided by a cloud provider. The OCI image format facilitates this ecosystem through a single standard.

For example, if we wanted to share our newly built Nginx container image on quay.io it's easy. First log in to quay:

```
podman login quay.io
```

Input:

```
Username: USERNAME
Password: ********
Login Succeeded!
```

Next, tag the image so that we can push it into our user account:

```
podman tag localhost/nginx quay.io/USERNAME/nginx
```

Finally, push the image:

```
podman push quay.io/USERNAME/nginx
```

Output:

```
Getting image source signatures
Copying blob 38c40d6c2c85 done
Copying blob fee76a531659 done
Copying blob c2adabaecedb done
```

(continues on next page)

(continued from previous page)

```
Copying config 7f3589c0b8 done
Writing manifest to image destination
Copying config 7f3589c0b8 done
Writing manifest to image destination
Storing signatures
```

Notice that we pushed four layers to our registry and now it's available for others to share. Take a quick look:

```
podman inspect quay.io/USERNAME/nginx
```

Output:

```
[
    {
        "Id": "7f3589c0b8849a9e1ff52ceb0fcea2390e2731db9d1a7358c2f5fad216a48263",
        "Digest":
↪"sha256:7822b5ba4c2eaabdd0ff3812277cfafa8a25527d1e234be028ed381a43ad5498",
        "RepoTags": [
            "quay.io/USERNAME/nginx:latest",
...
```

To summarize, Podman makes it easy to find, run, build and share containers.

- Find: whether finding a container on dockerhub.io or quay.io, an internal registry server, or directly from a vendor, a couple of podman search, and podman pull commands make it easy
- Run: it's easy to consume pre-built images with everything needed to run an entire application, or start from a Linux distribution base image with the podman run command
- Build: creating new layers with small tweaks, or major overhauls is easy with podman build
- Share: Podman lets you push your newly built containers anywhere you want with a single podman push command

For more instructions on use cases, take a look at our *Tutorials* page.

CHAPTER

TWO

COMMANDS

Podman (Pod Manager) Global Options, Environment Variables, Exit Codes, Configuration Files, and more

attach Attach to a running container

auto-update Auto update containers according to their auto-update policy

build Build an image using instructions from Containerfiles

commit Create new image based on the changed container

container Manage Containers

cp Copy files/folders between a container and the local filesystem

create Create but do not start a container

diff Inspect changes on container's file systems

events Show podman events

exec Run a process in a running container

export Export container's filesystem contents as a tar archive

generate Generated structured data

healthcheck Manage Healthcheck

history Show history of a specified image

image Manage images

images List images in local storage

import Import a tarball to create a filesystem image

info Display podman system information

init Initialize one or more containers

inspect Display the configuration of a container or image

kill Kill one or more running containers with a specific signal

load Load an image from container archive

login Login to a container registry

logout Logout of a container registry

logs Fetch the logs of a container

machine Manage podman's virtual machine

manifest Create and manipulate manifest lists and image indexes

mount Mount a working container’s root filesystem

network Manage Networks

pause Pause all the processes in one or more containers

play Play a pod

pod Manage pods

port List port mappings or a specific mapping for the container

ps List containers

pull Pull an image from a registry

push Push an image to a specified destination

rename Rename an existing container

restart Restart one or more containers

rm Remove one or more containers

rmi Removes one or more images from local storage

run Run a command in a new container

save Save image to an archive

search Search registry for image

secret Manage podman secrets

start Start one or more containers

stats Display a live stream of container resource usage statistics

stop Stop one or more containers

system Manage podman

tag Add an additional name to a local image

top Display the running processes of a container

unmount Unmounts working container’s root filesystem

unpause Unpause the processes in one or more containers

unshare Run a command in a modified user namespace

untag Removes one or more names from a locally-stored image

version Display the Podman Version Information

volume Manage volumes

wait Block on one or more containers

CHAPTER THREE

REFERENCE

Show the API documentation for version:

- latest (main branch)
- version 4.1
- version 4.0
- version 3.4
- version 3.3
- version 3.2
- version 3.1

CHAPTER

FOUR

TUTORIALS

Here are a number of useful tutorials to get you up and running with Podman. If you are familiar with the Docker Container Engine the command in Podman should be quite familiar. If you are brand new to containers, take a look at our *Introduction*.

- Basic Setup and Use of Podman: Learn how to setup Podman and perform some basic commands with the utility.
- Basic Setup and Use of Podman in a Rootless environment: The steps required to setup rootless Podman are enumerated.
- Podman Mac/Windows tutorial: Special setup for running the Podman remote client on a Mac or Windows PC and connecting to Podman running on a Linux VM are documented.
- How to sign and distribute container images using Podman: Learn how to setup and use image signing with Podman.
- Podman remote-client tutorial: A brief how-to on using the Podman remote-client.
- How to use libpod for custom/derivative projects: How the libpod API can be used within your own project.
- How to use Podman's Go RESTful bindings: An introduction to using our RESTful Golang bindings in an external application.
- Common network setups: A basic guide to common network setups for Podman.

CHAPTER

FIVE

SEARCH