Chapter 4. Custom Container Images

Abstract

Goal	Build custom container images to containerize applications.
Objectives	• Create a Containerfile by using basic commands. • Create a Containerfile that uses best practices. • Run rootless containers with Podman.
Sections	• Create Images with Containerfiles (and Guided Exercise) • Build Images with Advanced Containerfile Instructions (and Guided Exercise) • Rootless Podman (and Guided Exercise)
Lab	• Custom Container Images

Create Images with Containerfiles

Objectives

- Create a Containerfile by using basic commands.

Creating Images with Containerfiles

A *Containerfile* lists a set of instructions that a container runtime uses to build a container image. You can use the image to create any number of containers.

Each instruction causes a change that is captured in a resulting image layer. These layers are stacked together to form the resulting container image.

> **NOTE**
>
> You might have seen or used Dockerfiles instead of Containerfiles. These files are largely the same and mainly differ in historical context.
>
> This course uses Containerfiles because they are not associated with any specific container runtime engine. Podman supports both Dockerfiles and Containerfiles.

Choosing a Base Image

When you build an image, podman executes the instructions in the Containerfile and applies the changes on top of a container *base image*. A base image is the image from which your Containerfile and its resulting image is built.

The base image you choose determines the Linux distribution and its components, such as:

- Package manager
- Init system
- Filesystem layout
- Preinstalled dependencies and runtimes

The base image can also influence other factors, such as image size, vendor support, and processor compatibility.

Red Hat provides container images intended as a common starting point for containers known as *universal base images (UBI)*. These UBIs come in four variants: `standard`, `init`, `minimal`, and `micro`. Additionally, Red Hat also provides UBI-based images that include popular runtimes, such as Python and Node.js.

These UBIs use Red Hat Enterprise Linux (RHEL) at their core and are available from the Red Hat Container Catalog. The main differences are as follows:

Standard
This is the primary UBI, which includes DNF, systemd, and utilities such as `gzip` and `tar`.

Init
Simplifies running multiple applications within a single container by managing them with systemd.

Minimal
This image is smaller than the `init` image and still provides nice-to-have features. This image uses the `microdnf` minimal package manager instead of the full-sized version of DNF.

Micro
This is the smallest available UBI because it only includes the bare minimum number of packages. For example, this image does not include a package manager.

Containerfile Instructions

Containerfiles use a small domain-specific language (DSL) consisting of basic instructions for crafting container images. The following are the most common instructions.

`FROM`
Sets the base image for the resulting container image. Takes the name of the base image as an argument.

`WORKDIR`
Sets the current working directory within the container. Instructions that follow the `WORKDIR` instruction run within this directory.

`COPY` and `ADD`
Copy files from the build host into the file system of the resulting container image. Relative paths use the host current working directory, known as the build context. Both instructions use the working directory within the container as defined by the `WORKDIR` instruction.

The `ADD` instruction adds the following functionality:

- Copying files from URLs.
- Unpacking `tar` archives in the destination image.

Because the `ADD` instruction adds functionality that might not be obvious, developers tend to prefer the `COPY` instruction for copying local files into the container image.

`RUN`
Runs a command in the container and commits the resulting state of the container to a new layer within the image.

`ENTRYPOINT`
Sets the executable to run when the container is started.

`CMD`
Runs a command when the container is started. This command is passed to the executable defined by `ENTRYPOINT`. Base images define a default `ENTRYPOINT`, which is usually a shell executable, such as Bash.

> NOTE
>
> Neither `ENTRYPOINT` nor `CMD` run when building a container image. Podman executes them when you start a container from the image.

`USER`
Changes the active user within the container. Instructions that follow the `USER` instruction run as this user, including the `CMD` instruction. It is a good practice to define a different user other than `root` for security reasons.

`LABEL`
Adds a key-value pair to the metadata of the image for organization and image selection.

`EXPOSE`
Adds a port to the image metadata indicating that an application within the container binds to this port. This instruction does not bind the port on the host and is for documentation purposes.

`ENV`

Defines environment variables that are available in the container. You can declare multiple `ENV` instructions within the Containerfile. You can use the `env` command inside the container to view each of the environment variables.

ARG
Defines build-time variables, typically to make a customizable container build. Developers commonly configure the `ENV` instructions by using the `ARG` instruction. This is useful for preserving the build-time variables for runtime.

VOLUME
Defines where to store data outside of the container. The value configures the path where Podman mounts persistent volume inside of the container. You can define more than one path to create multiple volumes.
Each Containerfile instruction runs in an independent container by using an intermediate image built from every previous command. This means each instruction is independent from other instructions in the Containerfile. The following is an example Containerfile for building a simple Apache web server container:

```
# This is a comment line (1)
FROM        registry.redhat.io/ubi8/ubi:8.6 (2)
LABEL       description="This is a custom httpd container image" (3)
RUN         yum install -y httpd (4)
EXPOSE      80 (5)
ENV         LogLevel "info" (6)
ADD         http://someserver.com/filename.pdf /var/www/html (7)
COPY        ./src/   /var/www/html/ (8)
USER        apache (9)
ENTRYPOINT  ["/usr/sbin/httpd"] (10)
CMD         ["-D", "FOREGROUND"] (11)
```

1. Lines that begin with a hash, or pound, sign (`#`) are comments.
2. The `FROM` instruction declares that the new container image extends the `registry.redhat.io/ubi8/ubi:8.6` container base image.
3. The `LABEL` instruction adds metadata to the image.
4. `RUN` executes commands in a new layer on top of the current image. The shell that is used to execute commands is `/bin/sh`.
5. `EXPOSE` configures metadata indicating that the container listens on the specified network port at runtime.
6. `ENV` is responsible for defining environment variables that are available in the container.
7. The `ADD` instruction copies files or directories from a local or remote source and adds them to the container's file system.
8. `COPY` copies files from a path relative to the working directory and adds them to the container's file system.
9. `USER` specifies the username or the UID to use when running the container image for the `RUN`, `CMD`, and `ENTRYPOINT` instructions.
10. `ENTRYPOINT` specifies the default command to execute when users instantiate the image as a container.
11. `CMD` provides the default arguments for the `ENTRYPOINT` instruction.

Container Image Tags

When building a container image, you can specify an image name to later identify the image. The image name is a string composed of letters, numbers, and some special characters.

An *image tag* comes after the image name and is delimited by a colon (`:`). When you omit an image tag, podman uses the default `latest` tag.

> **NOTE**
> It is a best practice to specify tags in addition to `latest` tag. Because `latest` is the default tag applied to new images, references that use the `latest` tag can change unintentionally.

The full name of the image includes both a name and an optional image tag.

For example, in the full image name `my-app:1.0`, the name is `my-app` and the tag is `1.0`.

> **REFERENCES**
> What is a container image?
>
> Best practices for building images that pass Red Hat Container Certification

For more information on the differences between UBI variants, refer to the *Characteristics of UBI Images* chapter in the *Building, running, and managing containers* guide for RHEL 9 at https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/9/html-single/building_running_and_managing_containers/index#con_characteristics-of-ubi-images_assembly_types-of-container-images

The difference between an image *name* and an image *tag* is explained in the Podman documentation at https://docs.podman.io/en/v5.2.2/markdown/podman-tag.1.html