Accessing Containers

Objectives

- Explore running containers.

Container Transparency

Developers commonly package applications as containers to, among other benefits, isolate the application process. However, process isolation also means that developers lose immediate visibility into the state of the containerized process and its environment.

To regain that visibility, containerization tools such as Podman provide a way to start new processes within containers in the running state. This is useful for example when you want to read a log file, verify the value of an environment variable, or debug a process.

An Introduction to Container Layers

Container images are characterized as *immutable* and *layered*. Each image layer consists of a set of file system differences, or *diffs*. A diff signals a file system change from the previous layer, such as adding or modifying a file.

When you start a container, the container creates a new ephemeral layer over its base container image layers called *container layer*. This layer is the only read/write storage available for the container by default, and it is used for any runtime file system operations, such as creating working files, temporary files, and log files.

Files that are created in the container layer are considered volatile, which means that the files are deleted when you delete the container. The container layer is exclusive to the running container, so if you create another container from the same base image, then the new container creates another container layer. This ensures that each container's runtime data is isolated from other containers.

Ephemeral container storage is not sufficient for applications that need to keep data beyond the life of the container, such as databases. You can use persistent storage to support such applications. Persistent container storage is covered later in this course.

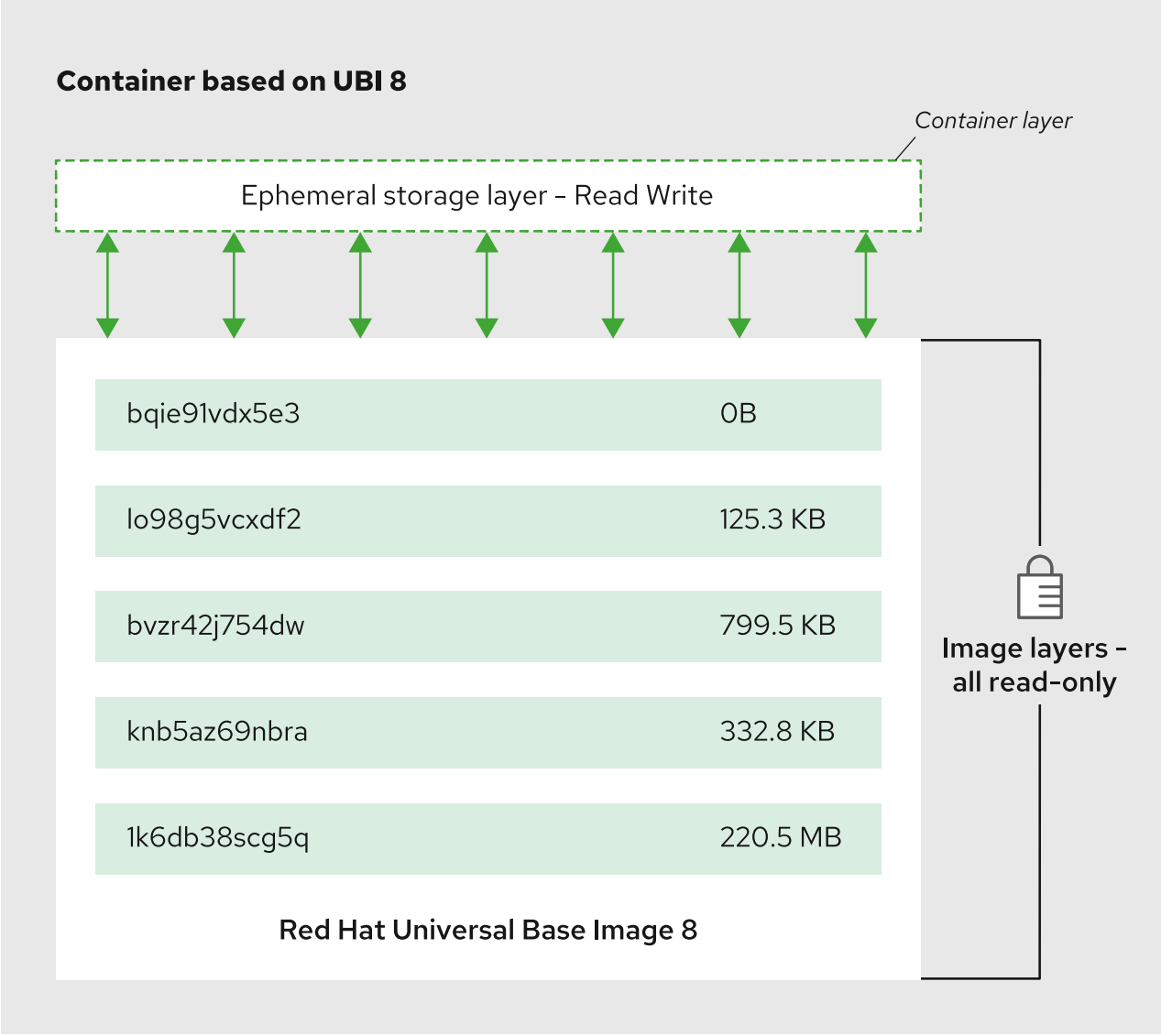

Figure 2.9: Examining example container image layers

Start Processes in Containers

Use the `podman exec` command to start a new process in a running container. The command uses the following syntax:

```
podman exec [options] container [command ...]
```

> **NOTE**
>
> Most Podman commands accept a container name or a container ID when identifying containers. You can see the IDs of all running containers by executing the `podman ps` command.

In the preceding syntax explanation, parts of the command in square brackets, such as `[options]`, are optional. For example, the following command prints the `/etc/httpd/conf/httpd.conf` file by using the `cat` command in a running container called `httpd`:

```
[user@host ~]$ podman exec httpd cat /etc/httpd/conf/httpd.conf
```

Additionally, `podman exec` provides a number of options, such as:

- Use `--env` or `-e` to specify environment variables.
- Use `--interactive` or `-i` to instruct the container to accept input.
- Use `--tty` or `-t` to allocate a pseudo terminal.
- Use `--latest` or `-l` to execute the command in the last created container.

> **NOTE**
>
> The `--latest` and `-l` flags are not available when using the Podman remote client. This includes when running via Podman Machine on macOS and Windows, except when using WSL2 on the latter.

The following command sets the `ENVIRONMENT` variable and then executes the `env` command to print all environment variables. In this example, the container name is not necessary because the `-l` option is used

```
[user@host ~]$ podman exec -e ENVIRONMENT=dev -l env
```

After the `env` process finishes, the `ENVIRONMENT` variable is unset. To make the `ENVIRONMENT` variable persistent, stop and remove the running container, and rerun the `podman run` command with the `-e ENVIRONMENT=dev` option.

Open an Interactive Session in Containers

Use the combination of the `--tty` and `--interactive` options to open an interactive shell in a running container.

If you open a shell program such as Bash or PowerShell in a container without providing any options, then the `podman exec` command opens the shell program, receives no input, and exits successfully:

```
[user@host ~]$ podman exec -l /bin/bash
[user@host ~]$
```

If you open a shell program with the `--interactive` option, the `podman exec` executes the shell program and commands, but the session does not behave like a regular terminal.

For example, features such as command history are not available in this mode and you cannot use programs that require a TTY such as vim. Although command output is visible, it is hard to differentiate from the input. Only use this mode when you do not need a full terminal experience.

The following example shows such an interactive terminal session:

```
[user@host ~]$ podman exec -il /bin/bash
pwd
/opt/app-root/src
ls ../
etc
scl_enable
src
```

> **NOTE**
>
> The preceding example combines multiple options. The `podman exec -il` command is identical to `podman exec -i -l`.

If you open a shell program with the `--tty` option, the `podman exec` executes the shell program and opens a pseudo terminal, but receives no input:

```
[user@host ~]$ podman exec -tl /bin/bash
bash-4.4$ (1)
```

(1) The shell program opens in a pseudo terminal, but you cannot pass any input to the shell.

To open an interactive shell in a running container, use the `--interactive` option. Also include the `--tty` option to make the session behave like a regular terminal session. Use the combination of both options to open a remote session in the container and have it behave in a typical way, such as in the following example:

```
[user@host ~]$ podman exec -til /bin/bash
bash-4.4$ pwd (1)
/opt/app-root/src
bash-4.4$ ls ../
etc  scl_enable  src
bash-4.4$ exit (2)
[user@host ~]$ (3)
```

(1) Execute the `pwd` command to print the current working directory.

(2) Execute the `exit` command to exit the interactive session.

(3) The container exits successfully.

Copy Files In and Out of Containers

Some containers do not provide the programs necessary to debug a running process. For example, to print the content of a file, you might use the `cat` utility. However, in production-ready containers, a best practice is to package only the libraries required by the application runtime. Such a container might not include basic utilities, such as `cat`, or an editor such as `vi`.

You can work around the issue in several ways, one of which is copying the file you want to modify to your host machine, modifying it, and then replacing the original file.

Use the `podman cp` command to copy files to and from a running container. The command uses the following syntax:

```
podman cp [options] [container:]source_path [container:]destination_path
```

Use the following command to copy the `/tmp/logs` file from a container with ID `a3bd6c81092e` to the current directory:

```
[user@host ~]$ podman cp a3bd6c81092e:/tmp/logs .
```

> **NOTE**
>
> The dot (`.`) in the previous command signifies the current working directory.

Use the following command to copy the `nginx.conf` file to the `/etc/nginx/` directory in a container called `nginx`:

```
[user@host ~]$ podman cp nginx.conf nginx:/etc/nginx
```

The preceding command assumes that the `nginx.conf` file exists in your working directory.

Use the following command to copy the `nginx.conf` file from the `nginx-test` container to the `nginx-proxy` container:

```
[user@host ~]$ podman cp nginx-test:/etc/nginx/nginx.conf nginx-proxy:/etc/nginx
```

> **REFERENCES**
>
> podman-exec(1) man page
>
> podman-cp(1) man page