Chapter 5. Persisting Data

Abstract

Goal	Run database containers with persistence.
Objectives	• Describe the process for mounting volumes and common use cases. • Build containerized databases.
Sections	• Volume Mounting (and Guided Exercise) • Working with Databases (and Guided Exercise)
Lab	• Persisting Data

Volume Mounting

Objectives

- Describe the process for mounting volumes and common use cases.

Copy-on-write (COW) File System

When you build a container image, each instruction that modifies the container file system creates a new read-only data layer. Because you cannot modify data in an existing layer, each layer contains a set of changes, or *diffs*, from the previous layer.

Consider the following Containerfile:

```
FROM registry.access.redhat.com/ubi8/ubi-minimal

RUN microdnf install httpd
RUN microdnf clean all
RUN rm -rf /var/cache/yum

CMD httpd -DFOREGROUND
```

You can inspect the image layers by using the `podman image tree` command:

```
[user@host ~]$ podman image tree ubi-httpd
Image ID: bdd298c12db7
Tags:     [localhost/ubi-httpd:latest]
Size:     166.8MB
Image Layers
├── ID: 647a854c512b Size: 94.77MB
├── ID: 3a15f4cd9c23 Size: 20.48kB Top Layer of: [registry.access.redhat.com/ubi8/ubi-minimal:latest]
├── ID: d8dcba576e2d Size: 68.45MB
├── ID: e9d8a486b7ac Size: 3.581MB
└── ID: 4c71c24f9911 Size: 4.608kB Top Layer of: [localhost/ubi-httpd:latest]
```

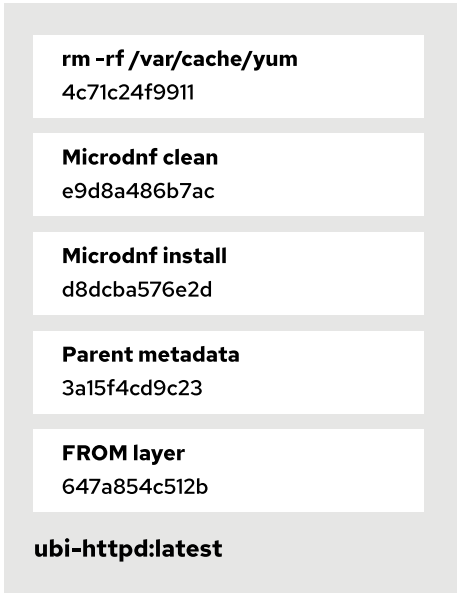

Figure 5.1: The ubi-httpd container layers

Note that because the `microdnf clean all` command creates a new layer, the resulting container image still contains the data that the `microdnf clean all` command removed. Though not accessible at runtime, the data is contained in the previous layer, the `d8dcba576e2d` layer, and contributes to the overall container size.

To remove unnecessary data from the `microdnf install` command, use the `microdnf clean all` and the `rm -rf /var/cache/yum` commands in the same container layer. For example, chain the commands in one `RUN` instruction:

```
FROM registry.access.redhat.com/ubi8/ubi-minimal

RUN microdnf install httpd && \
    microdnf clean all && \
    rm -rf /var/cache/yum

CMD httpd -DFOREGROUND
```

When you instantiate the container image, for example by using the `podman run` command, Podman creates a *thin* read/write container layer on top of the previous layers. This means that multiple containers that use one container image share read-only layers and differ in the runtime read/write layer. When you delete a container, Podman destroys the read/write layer. This means container runtime data is ephemeral.

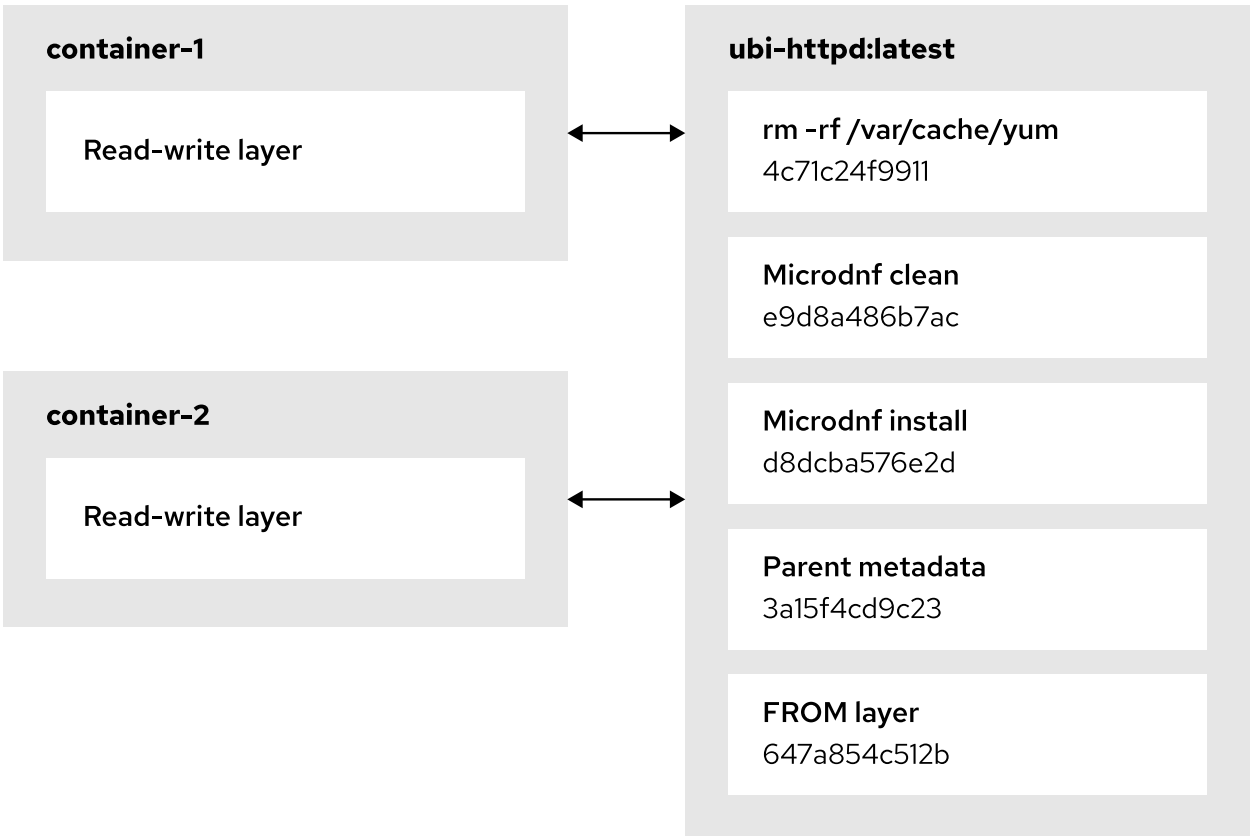

Figure 5.2: Multiple containers share immutable read-only layers

See the references section for more information about Podman's implementation of the overlay union file system.

Implications of a COW File System

Files that are in the container image layers are available to the read/write data layer by using a union file system. At runtime, the container file system consists of a union of files in the defined layers.

Modifying a file or its attributes at runtime is possible in several steps:

- Podman locates the requested file in the closest (highest) layer and copies it to the runtime layer.
- In the runtime layer, when the file is available for both read and write operations, a containerized process can modify the file.

The implementation of how files behave when a process attempts to read or write a file depends on the storage driver.

Using a union file systems provides efficient read operations. The COW data architecture also promotes the sharing and reuse of data layers between separate containers, because the layers are immutable.

However, union file systems introduce a performance bottleneck for write-intensive containerized processes.

Store Data on Host Machine

Developers often write applications that require storing data persistently. For such cases, Podman implements external mounts by using *volumes* and *bind mounts*.

This is useful for the following reasons:

Persistence
Mounted data is persistent across container deletions. Because containers are ephemeral, data from the runtime read/write layer is not accessible after container deletion.

Use of Host File System
Mounted data typically does not implement the COW file system. Consequently, write-heavy containerized processes can write data to a mount without the limitations of the COW file systems for write operations.

Ease of Sharing
Mounted data can be shared between multiple containers at the same time. This means that one container can write to a mount and another container can read the same data.

Additionally, mounts are not limited to the container host machine. You can host data mounts over the network, for example by using the NFS protocol.

Volumes are data mounts managed by Podman. Bind mounts are data mounts managed by the user.

Both volumes and bind mounts can use the `--volume` (or `-v`) parameter.

```
--volume /path/on/host:/path/in/container:OPTIONS
```

In the preceding syntax, the *:OPTIONS* part is optional. Note that you can specify host paths by using absolute paths, such as `/home/user/www`, or relative paths, such as `./www`, which refers to the `www` directory in the current working directory.

Use `-v volume_name:/path/in/container` to refer to a volume.

Alternatively, you can use the `--mount` parameter with the following syntax:

```
--mount type=TYPE,source=/path/on/host,destination=/path/in/container
```

The `--mount` parameter explicitly specifies the volume type, such as:

- `bind` for bind mounts.
- `volume` for volume mounts.
- `tmpfs` for creating memory-only, ephemeral mounts.

The `--mount` parameter is the preferred way of mounting directories in a container. However, because the `-v` parameter is still widely used, this course uses both styles.

Developers commonly use bind mounts for testing, or for mounting environment-specific files, such as property files. Because Podman manages the volume file system, use volumes to ensure consistent mount behavior across systems. Additionally, use volumes for advanced uses, such as mounting a remote volume over NFS.

Storing Data with Bind Mounts

Bind mounts can exist anywhere on the host system.

For example, to mount the `/www` directory on your host machine to the `/var/www/html` directory inside the container with the read-only option, use the following `podman run` command:

```
[user@host ~]$ podman run -p 8080:8080 --volume  /www:/var/www/html:ro \
  registry.access.redhat.com/ubi8/httpd-24:latest
```

Troubleshoot Bind Mounts

When you use bind mounts, you must configure file permissions and SELinux access manually. SELinux is an additional security mechanism used by Red Hat Enterprise Linux (RHEL) and other Linux distributions.

Consider the following bind mount example:

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

By default, the `Httpd` process has insufficient permissions to access the `/var/www/html` directory. This can be a file permission issue or an SELinux issue.

To troubleshoot file permission issues, use the `podman unshare` command to execute the `ls -l` command. The `podman unshare` command executes provided Linux commands in a new namespace such as the one Podman creates for the container. This process maps user IDs as they are mapped in a new container, which is useful for troubleshooting user permissions.

```
[user@host ~]$ podman unshare ls -l /www/
total 4
-rw-rw-r--. 1 root root 21 Jul 12 15:21 index.html
[user@host ~]$ podman unshare ls -ld /www/
drwxrwxr-x. 1 root root 20 Jul 12 15:21 /www/
```

The previous example reveals that the `/www` directory is accessible to all users:

- The `/www/index.html` file provides read permissions to all users.
- The `/www` directory is viewed as owned by the `root` user and the `root` group in a new namespace.

- The `/www` directory provides the execute permissions to all users, which gives all users access to the directory contents.

This means that the file and directory permissions are correct in the bind mount.

To troubleshoot SELinux permission issues, inspect the `/www` directory SELinux configuration by running the `ls` command with the `-Z` option. Use the `-d` option to print only the directory information.

```
[user@host ~]$ ls -Zd /www
system_u:object_r:default_t:s0:c228,c359 /www
```

The output shows the SELinux context label `system_u:object_r:default_t:s0:c228,c359`, which has the `default_t` type. A container must have the `container_file_t` SELinux type to have access to the bind mount. SELinux is out of scope for this course.

To fix the SELinux configuration, add the `:z` or `:Z` option to the bind mount:

- Lower case `z` lets different containers share access to a bind mount.
- Upper case `Z` provides the container with exclusive access to the bind mount.

```
[user@host ~]$ podman run -p 8080:8080 --volume /www:/var/www/html:Z \
  registry.access.redhat.com/ubi8/httpd-24:latest
```

After adding the corresponding option, run the `ls -Zd` command and notice the right SELinux type.

```
[user@host ~]$ ls -Zd /www
system_u:object_r:container_file_t:s0:c240,c717 /www
```

The `container_file_t` SELinux type allows the container to access the bind mount files.

> **IMPORTANT**
>
> Changing the SELinux label for system directories might lead to unexpected issues.

Storing Data with Volumes

Volumes let Podman manage the data mounts. You can manage volumes by using the `podman volume` command.

To create a volume called `http-data`, use the following command:

```
[user@host ~]$ podman volume create http-data
d721d941960a2552459637da86c3074bbba12600079f5d58e62a11caf6a591b5
```

You can inspect the volume by using the `podman volume inspect` command:

```
[user@host ~]$ podman volume inspect http-data
[
    {
        "Name": "http-data",
        "Driver": "local",
        "Mountpoint": "/home/user/.local/share/containers/storage/volumes/http-data/_data",
        "CreatedAt": "2022-07-12T17:10:12.709259987+02:00",
        "Labels": {},
        "Scope": "local",
        "Options": {}
    }
]
```

For rootless containers, Podman stores local volume data in the `$HOME/.local/share/containers/storage/volumes/` directory.

To mount the volume into a container, refer to the volume by the volume name:

```
[user@host ~]$ podman run -p 8080:8080 --volume  http-data:/var/www/html \
  registry.access.redhat.com/ubi8/httpd-24:latest
```

Because Podman manages the volume, you do not need to configure SELinux permissions.

Exporting and Importing Data with Volumes

You can import data from a tar archive into an existing Podman volume by using the `podman volume import` *`VOLUME_NAME ARCHIVE_NAME`* command.

```
[user@host ~]$ podman volume import http_data web_data.tar.gz
...no output expected...
```

You can also export data from an existing Podman volume and save it as a tar archive on the local machine by using the `podman volume export` *`VOLUME_NAME`* `--output` *`ARCHIVE_NAME`* command.

```
[user@host ~]$ podman volume export http_data --output web_data.tar.gz
...no output expected...
```

Storing Data with a tmpfs Mount

Some applications cannot use the default COW file system in a specific directory for performance reasons, but use persistence or data sharing for that directory.

For such cases, you can use the `tmpfs` mount type, which means that the data in a mount is ephemeral but does not use the COW file system:

```
[user@host ~]$ podman run -e POSTGRESQL_ADMIN_PASSWORD=redhat --network lab-net \
  --mount  type=tmpfs,tmpfs-size=512M,destination=/var/lib/pgsql/data \
  registry.redhat.io/rhel9/postgresql-13:1
```

REFERENCES

Overlay.go

SELinux and Container File Permissions

Podman is gaining rootless overlay support