

Section 9 - Persistent Data and Volumes

1 Persistent Data

Section Overview

- Defining the problem of persistent data
- Key concepts of containers: immutable, ephemeral
- Data Volumes
- Bind Mounts
- LAB

Containers: immutable & ephemeral (1)

- Containers are usually immutable and ephemeral =>
- immutable => Containers do not change (e.g. we usually don't update the binaries included)
- ephemeral => Containers are temporary/disposable (e.g. to use an updated version of the application we delete the old container and we re-create a new one from an updated image)
- This is not a limitation of the containers but a design goal, a best best practice.

Containers: immutable & ephemeral (2)

- This is the concept of the immutable infrastructure, where we do not change the application components once they are running. If an upgrade operation must be performed, then new container are re-created.
- The main benefits of the immutable infrastructure are:
 - Reliability
 - Consistency
 - Changes are reproducible

What about unique data?

- What about the unique data produced by an application?
 - Databases or
 - Anything else that an application will store to a file
- The containers should not contain *unique data* mixed with the application binaries.
 - This is known as separation of concerns
- Docker provides us with two solutions to preserve *unique data*:
 1. Data Volumes
 - anonymous volumes
 - named volumes

2. Bind Mounts

Data Volumes

- Volumes are created and managed from Docker
- Volumes are the preferred mechanism for persisting data generated by containers.
- Volumes contents exist outside the lifecycle of a given container.
- When a Docker container is destroyed, it's entire file system is destroyed too. So if we want to keep this data, it is necessary to use Docker volumes.
- The container sees it like a local file path.
- Docker volumes are attached to containers during a `docker run` command by using the `-v` option.

Bind Mounts

- A bind mount is a file or folder located in the user's workspace.
- A bind mount is a file or folder available on the Docker host filesystem, mounted into a running container.
- The main difference a *bind mount* has from a *volume* is that since it can exist anywhere on the host filesystem, processes outside of Docker can also modify it.
- Bind mounts have limited functionality compared to volumes.
- The container sees it like a local file path.
- Docker bind mount volumes are attached to containers during a `docker run` command by using the `-v` option.

Different Types of Volumes

- host volume

```
docker run -v /path/on/host:/path/in/container ...
```

- anonymous volume

```
FROM debian  
...  
VOLUME /path/to/data-dir  
...
```

- named volume

```
docker volume create somevolumename  
docker run -v name:/path/in/container ...
```


Different Types of Volumes (2)

- Ref: [link](#)

Container data

- Because we stopped the container or restarted the host, it doesn't mean that the container's file changes are lost
- Only when we remove the container, the container's data are deleted.