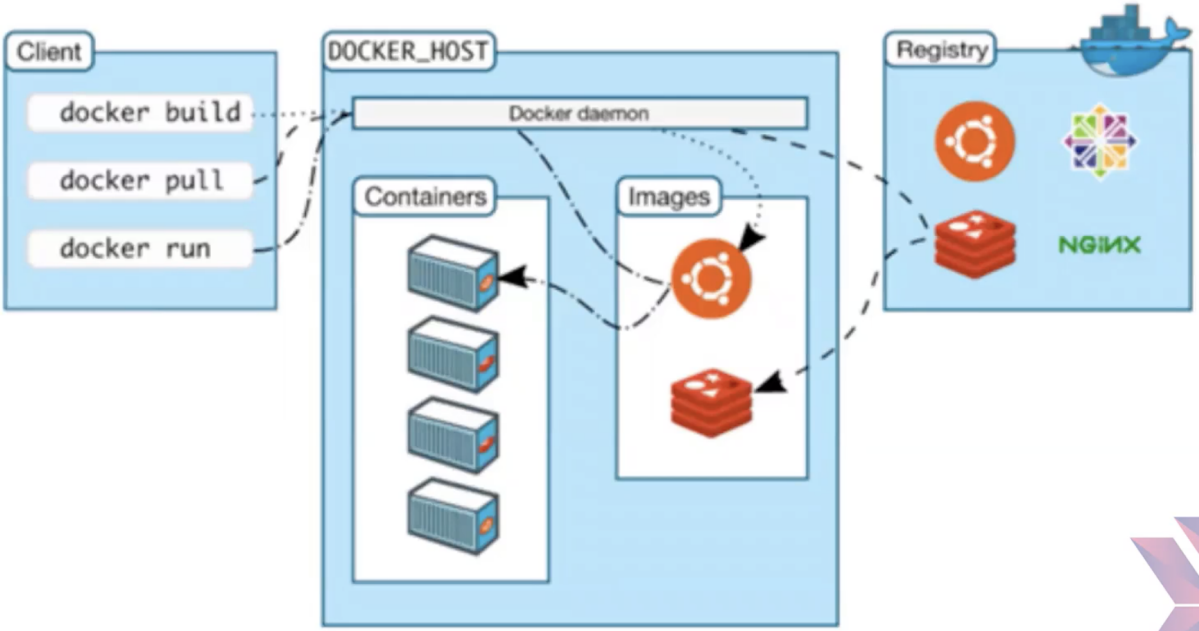
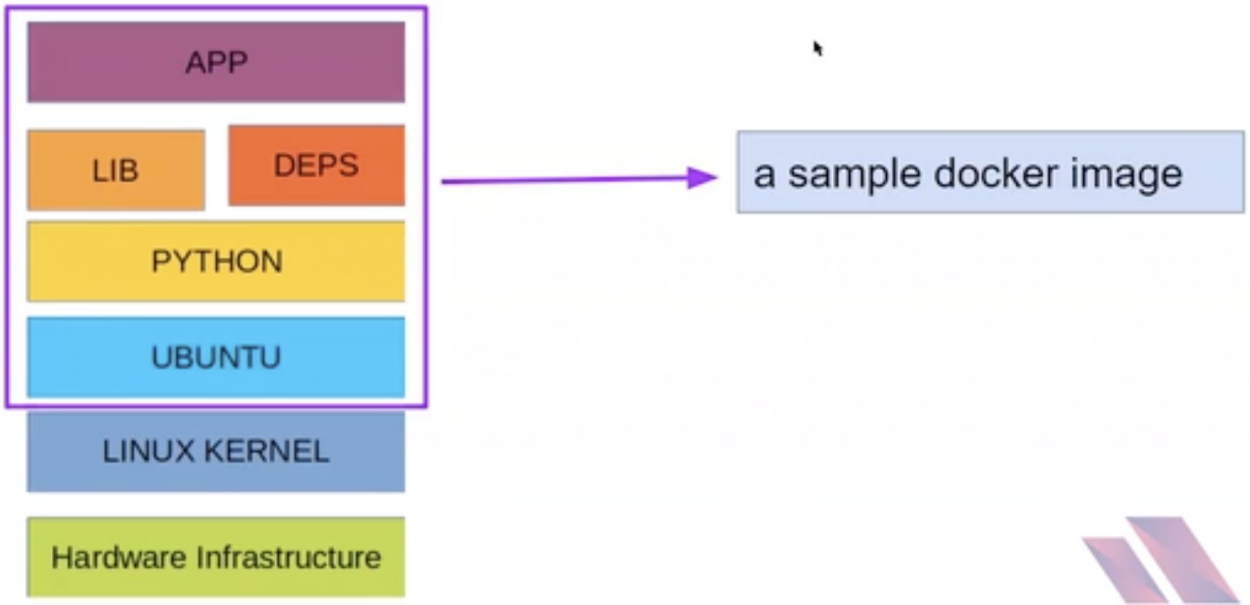
Docker Volumes

Table of Contents

Docker Architecture



Container Layers



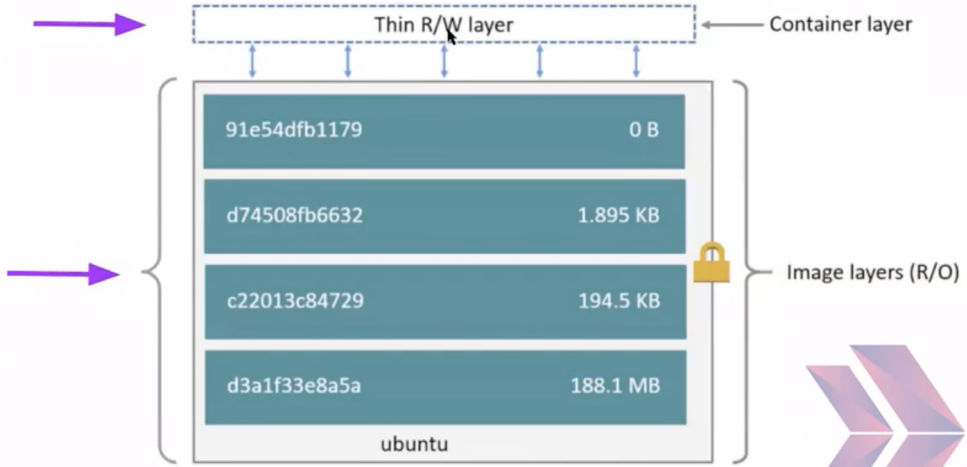
Container Layers

- A Docker image is built up from a series of layers. Each layer represents an instruction in the image's Dockerfile. Each layer except the very last one is read-only.

```
docker run imageName
```

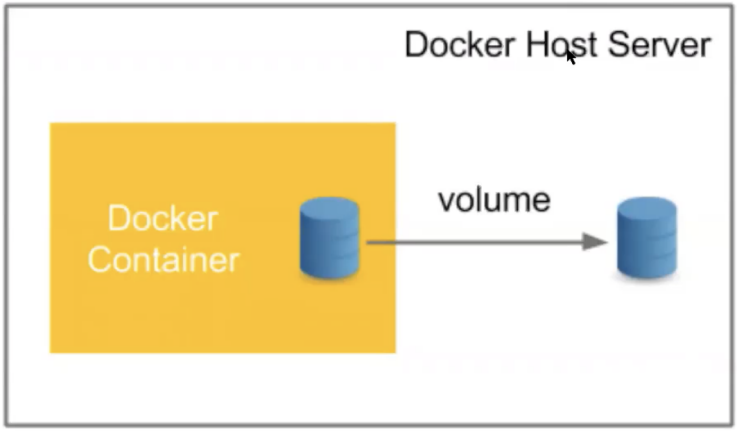
```
FROM ubuntu
COPY . /app
RUN apt-get install python3.6
CMD python3 /app/index.py
```

Dockerfile



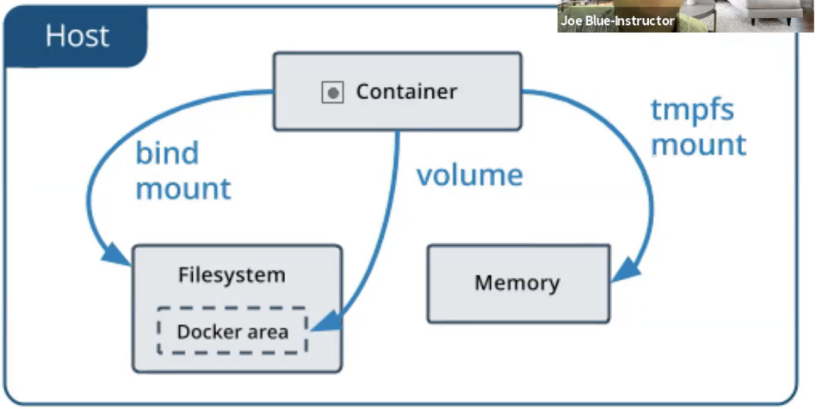
Manage data in Docker

- By default, all files created inside a container are stored on a **writable container layer**. This means that the data doesn't **persist** when that container no longer exists.
- Docker volumes, which are special directories in a container, store files in the host machine so that the files are **persisted** even after the container stops.



Types of mounts

Volumes are stored in a part of the host filesystem which is *managed* by Docker (*/var/lib/docker/volumes/* on Linux). Non-Docker processes should not modify this part of the filesystem. **Volumes are the best way to persist data in Docker.**



Bind mounts may be stored anywhere on the host system. They may even be important system files or directories. Non-Docker processes on the Docker host or a Docker container can modify them at any time.

tmpfs mounts are stored in the host system's memory only, and are never written to the host system's filesystem.

Manage data in Docker

Volumes are created and managed by Docker. You can create a volume explicitly using the docker volume create command.

```
$ docker volume create firstvolume
```

Manage data in Docker

- When you create a volume, it is stored within a directory on the Docker host. When you mount the volume into a container, this directory is what is mounted into the container.

```
$ docker volume inspect firstvolume
[
    {
        "CreatedAt": "2020-07-12T13:19:27Z",
        "Driver": "local",
        "Labels": {},
        "Mountpoint": "/var/lib/docker/volumes/firstvolume/_data",
        "Name": "firstvolume",
        "Options": {},
        "Scope": "local«
    }
]
```

Manage data in Docker

- When you create a volume, it is stored within a directory on the Docker host. When you mount the volume into a container, this directory is what is mounted into the container.

```
$ docker volume inspect firstvolume
[
    {
        "CreatedAt": "2020-07-12T13:19:27Z",
        "Driver": "local",
        "Labels": {},
        "Mountpoint": "/var/lib/docker/volumes/firstvolume/_data",
        "Name": "firstvolume",
        "Options": {},
        "Scope": "local«
    }
]
```

Docker Volume Behaviours

- **Volumes** are managed by Docker and are isolated from the core functionality of the host machine.
- **Bind mounts** are dependent on the directory structure of the host machine.

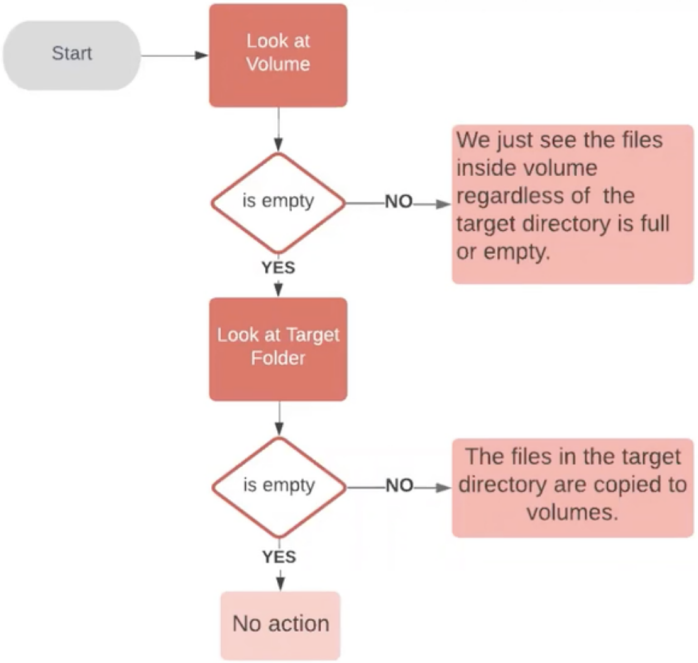
Docker Volume Behaviours

- **Volumes** are easier to back up or migrate than **bind mounts**.
- You can manage volumes using Docker CLI commands or the Docker API.
- Volumes work on both Linux and Windows containers.
- Volumes can be more **safely shared** among multiple containers.
- Volume drivers let you store volumes on **remote hosts** or **cloud providers**, to encrypt the contents of volumes, or to add other functionality.
- New volumes can have their **content pre-populated** by a container.

Docker Volume Behaviours

Situation	Behaviour
If there is no target directory.	The target directory is created and files inside volume are copied to this directory.
If there is target directory, but it is empty.	The files in volume are copied to target directory.
If there is target directory and it is not empty, but volume is empty.	The files in the target directory are copied to volumes.
If volume is not empty.	We just see the files inside volume regardless of the target directory is full or empty.

Docker Volume Behaviours



Declaration of volumes

- **`-v` or `--volume`**: Consists of three fields, separated by colon characters (`:`). The fields must be in the correct order, and the meaning of each field is not immediately obvious.
    - In the case of named volumes, the first field is the name of the volume, and is unique on a given host machine. For anonymous volumes, the first field is omitted.
    - The second field is the path where the file or directory are mounted in the container.
    - The third field is optional, and is a comma-separated list of options, such as `ro`.
- **`--mount`**: Consists of multiple key-value pairs, separated by commas and each consisting of a `<key>=<value>` tuple. The `--mount` syntax is more verbose than `-v` or `--volume`, but the order of the keys is not significant, and the value of the flag is easier to understand.
    - The `type` of the mount, which can be `bind`, `volume`, or `tmpfs`. This topic discusses volumes, so the type is always `volume`.
    - The `source` of the mount. For named volumes, this is the name of the volume. For anonymous volumes, this field is omitted. May be specified as `source` or `src`.
    - The `destination` takes as its value the path where the file or directory is mounted in the container. May be specified as `destination`, `dst`, or `target`.
    - The `readonly` option, if present, causes the bind mount to be mounted into the container as read-only.
    - The `volume-opt` option, which can be specified more than once, takes a key-value pair consisting of the option name and its value.

Declaration of volumes

- Volumes can be declared on the command-line, with the --volume or -v flag for docker run.
- -v or --volume: Consists of three fields, separated by colon characters (:). The fields must be in the correct order.

```
--volume <volume_name>:<path>:<list of options>
```

Declaration of volumes

--volume <volume_name>:<path>:<list of options>

- The first field is the name of the volume, and is unique on a given host machine.
- The second field is the path where the file or directory are mounted in the container.
- The third field is optional, and is a comma-separated list of options, such as ro (read only).

Declaration of volumes

--mount

```
$ docker run -d \
  --name devtest \
  --mount source=myvol2, target=/app \
  nginx:latest
```

-v (--volume)

```
$ docker run -d \
  --name devtest \
  -v myvol2:/app \
  nginx:latest
```

```
$ docker run -d \
  --name=nginxtest \
  --mount source=nginx-vol, destination=/usr/share/nginx/html \
  nginx:latest
```

```
$ docker run -d \
  --name=nginxtest \
  -v nginx-vol:/usr/share/nginx/html \
  nginx:latest
```

docker volume Commands

Command	Description
docker volume create	Create a volume
docker volume inspect	Display detailed information on one or more volumes
docker volume ls	List volumes
docker volume prune	Remove all unused local volumes
docker volume rm	Remove one or more volumes

Start a container with a bind m


```
$ docker run -d \
  --name devtest \
  --mount type=bind,source="$(pwd)"/target,target=/app \
  nginx:latest
```

```
$ docker run -d \
  --name devtest \
  -v "$(pwd)"/target:/app \
  nginx:latest
```

Use `docker inspect devtest` to verify that the bind mount was created correctly. Look for the `Mounts` section:

```
"Mounts": [
    {
        "Type": "bind",
        "Source": "/home/ubuntu/target",
        "Destination": "/app",
        "Mode": "",
        "RW": true,
        "Propagation": "rprivate"
    }
],
```