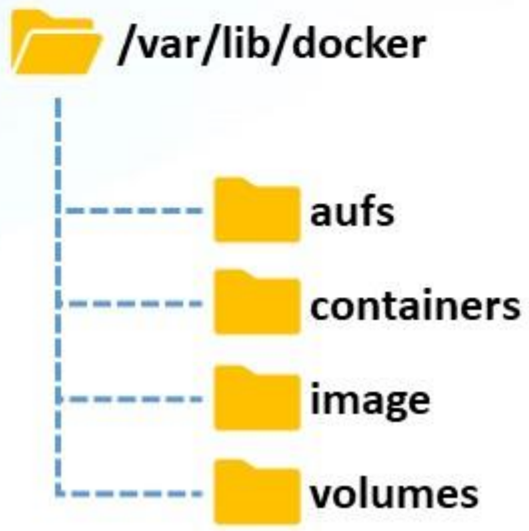
SkillAssure

# Docker Storage

## File System

- On installation, docker creates its folder structure at var/lib/docker
- This is where Docker stores all its data by default.



SkillAssure

## Layered Architecture

Dockerfile

```
FROM Ubuntu

RUN apt-get update  && apt-get -y install python

RUN pip install flask flask-mysql

COPY . /opt/source-code

ENTRYPOINT FLASK_APP=/opt/source-code/app.py  flask run
```

Dockerfile2

```
FROM Ubuntu

RUN apt-get update  && apt-get -y install python

RUN pip install flask flask-mysql

COPY app2.py /opt/source-code

ENTRYPOINT FLASK_APP=/opt/source-code/app2.py  flask run
```

```
docker build Dockerfile –t prabhav/my-custom-app
```

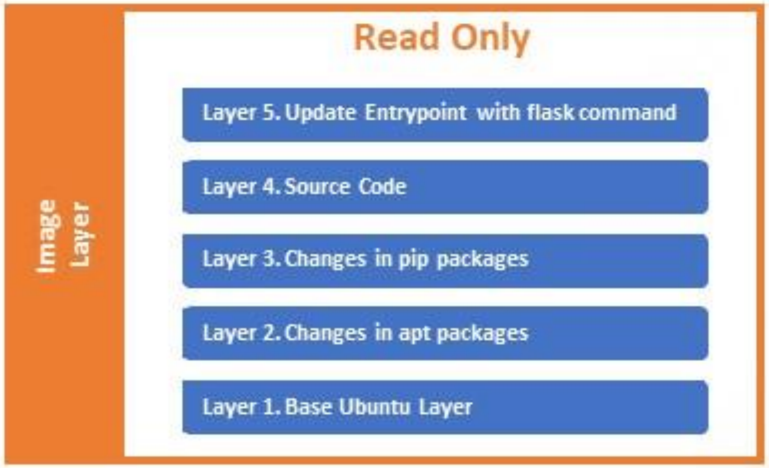
| | | |
|---|---|---|
| 1 | Layer 1. Base Ubuntu Layer | 120 MB |
| 2 | Layer 2. Changes in apt packages | 306 MB |
| 3 | Layer 3. Changes in pip packages | 6.3 MB |
| 4 | Layer 4. Source Code | 229 B |
| 5 | Layer 5. Update Entrypoint with flask command | 0 B |

```
docker build Dockerfile2 –t prabhav/my-custom-app2
```

| | | |
|---|---|---|
| 1 | Layer 1. Base Ubuntu Layer | |
| 2 | Layer 2. Changes in apt packages | |
| 3 | Layer 3. Changes in pip packages | |
| 4 | Layer 4. Source Code | 229 B |
| 5 | Layer 5. Update Entrypoint with flask command | 0 B |

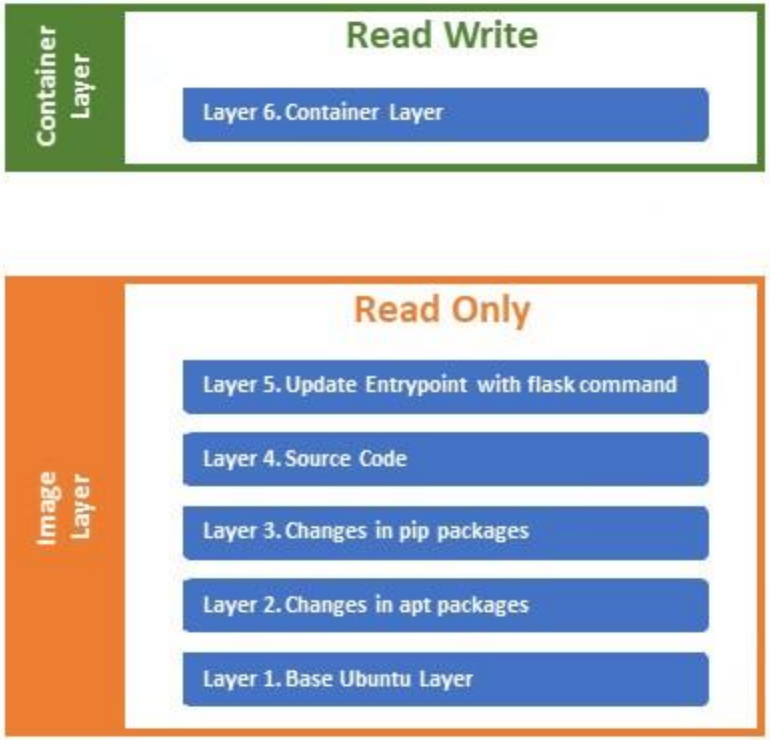
## Layered Architecture

- All these layers are created when we build the Docker image.
- These are called Image Layers
- Once the build is complete, contents of these layers cannot be modified.
- These layers are Read Only
- These layers can be modified by initiating a new build


Read Only
Image Layer
Layer 5. Update Entrypoint with flask command
Layer 4. Source Code
Layer 3. Changes in pip packages
Layer 2. Changes in apt packages
Layer 1. Base Ubuntu Layer

## Layered Architecture

- When we run a container, docker creates a container from these layers and creates a new writable layer on top of the image layer.
- The writable layer is called as container layer
- The writable layer is used to store data created by the container
  - e.g. log files, temporary files or just any file modified by the user on that container.
- The life of this layer is only as long as the container is alive.
- When the container is destroyed this layer and all the changes stored in it are also destroyed.



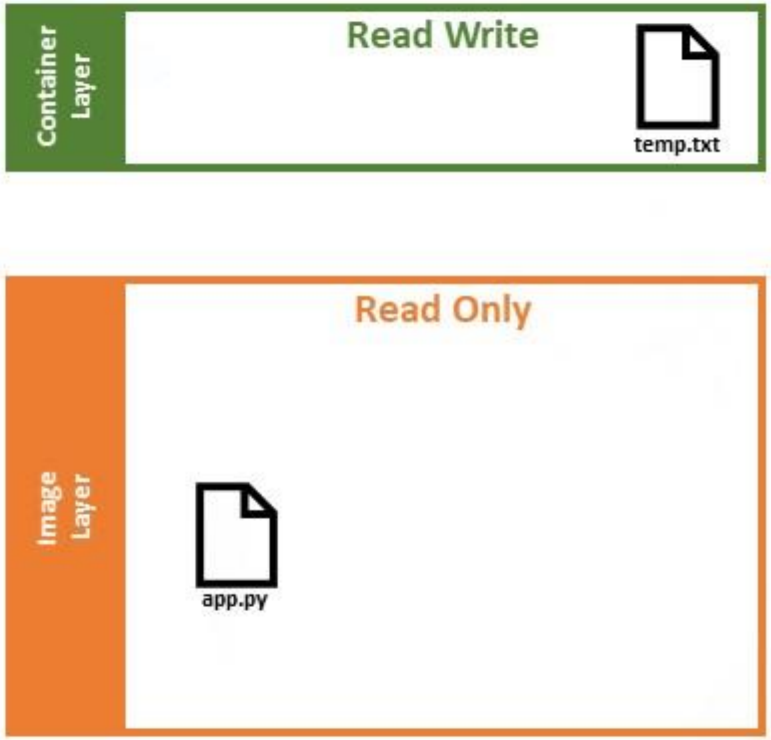
Container Layer
Read Write
Layer 6. Container Layer
Image Layer
Read Only
Layer 5. Update Entrypoint with flask command
Layer 4. Source Code
Layer 3. Changes in pip packages
Layer 2. Changes in apt packages
Layer 1. Base Ubuntu Layer

SkillAssure

## Layered Architecture

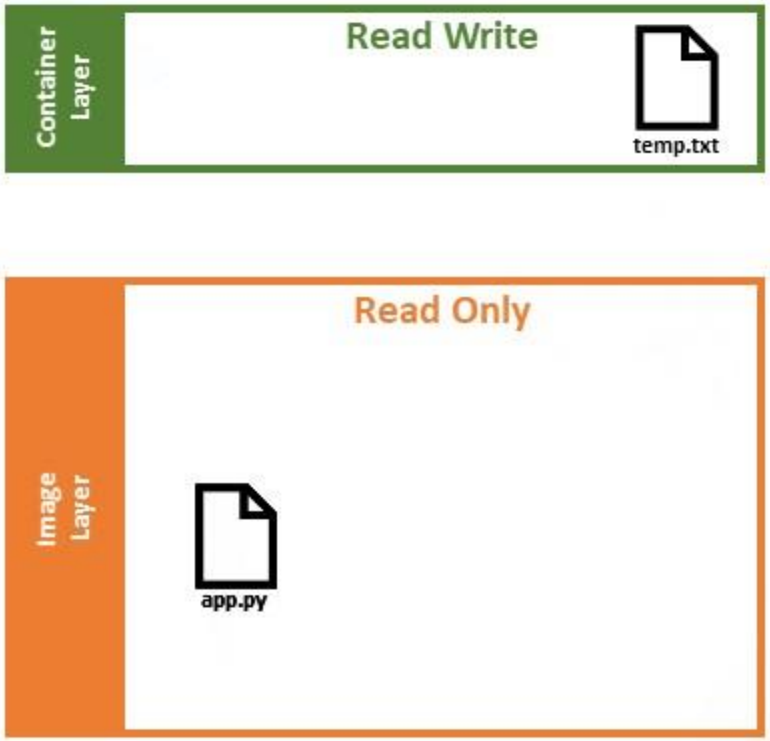
- Let’s say we login to the newly created container and create a new file - temp.txt.
- It will get created in the container layer which is read and write.

- Our application code is in Image layer.
- If I wish to modify the application code
- How will the modification work? As our Image layer is read only.



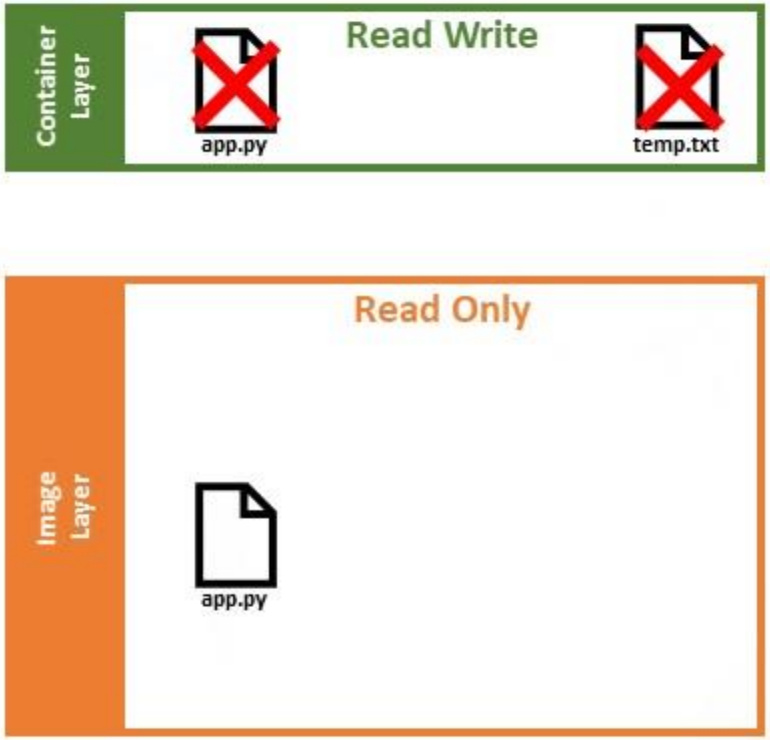
# COPY-ON-WRITE

- Before I save the modified file, docker automatically creates a copy of the file in the read write layer
- This allow us to modify the application code
- All future modifications will be done on this copy of the file in the read write layer.
- This is called **copy on write mechanism**
- Image layer being read only just means that the files in these layers will not be modified in the image itself
- Image will remain the same all time until you rebuild

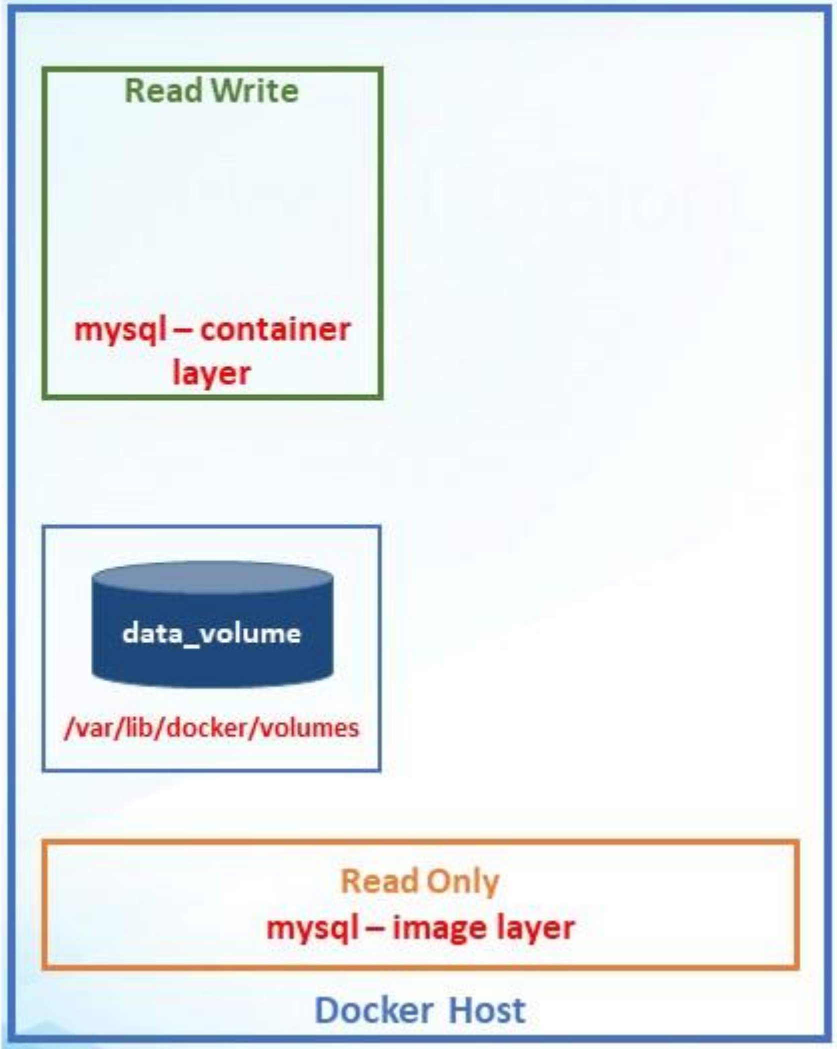


## COPY-ON-WRITE

- What happens when I delete the container?
- All data stored in the container layer also gets deleted.
- The change we made to the app.py and temp.txt will also get removed.



## Volumes

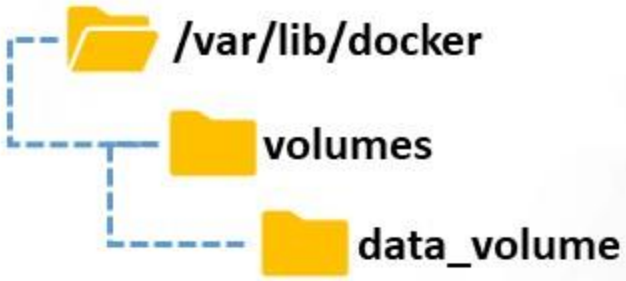


- How do we persist the data?
- Let's say I want to run MySQL and my databases to be persistent.

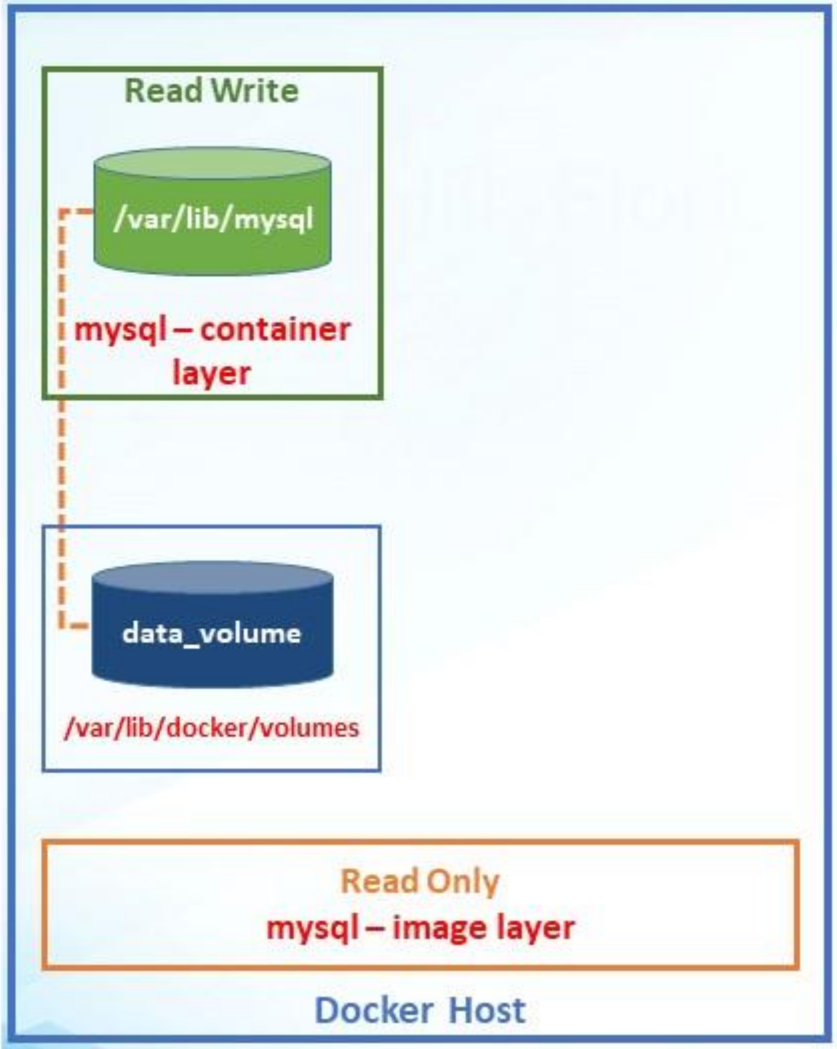
- We can add a persistent volume to the container
- Create a volume with name "data_volume"

```
docker volume create data_volume
```

- It creates a folder called data_volume under var/lib/docker/volumes



## Volumes



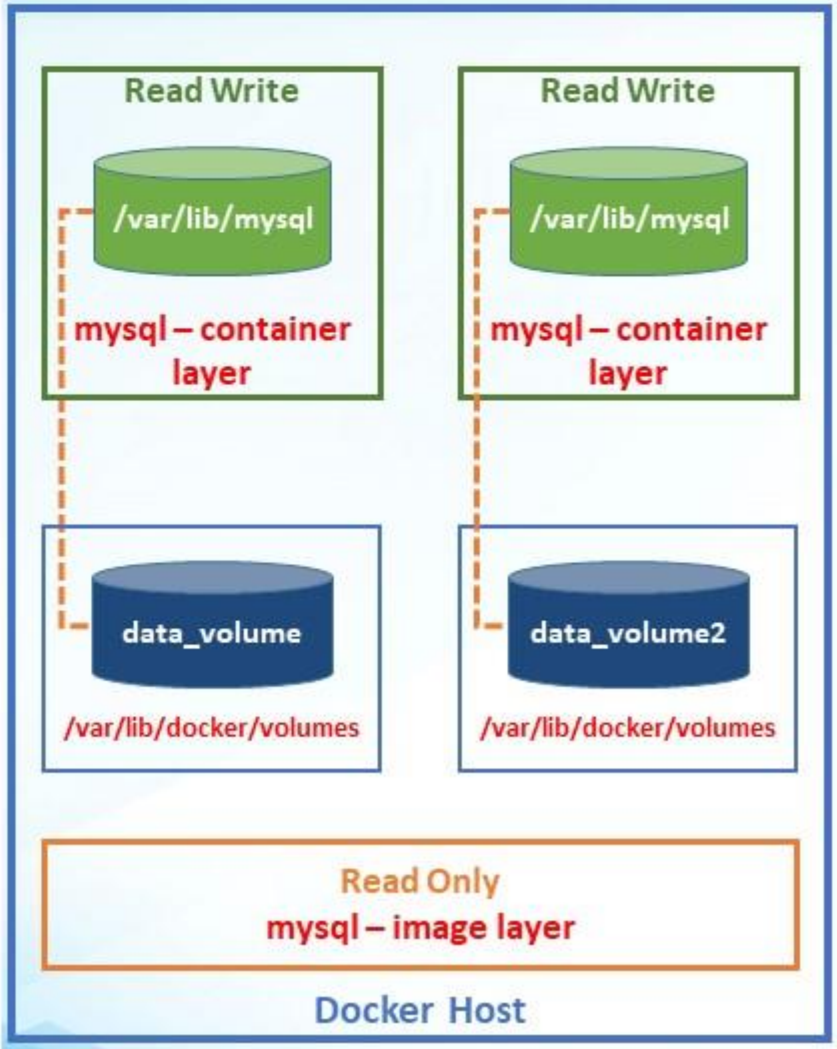
- **var/lib/mysql** is the default location where MySQL stores data
- To persist data I can mount this volume to MySql default location while running the container.

```
docker run -v data_volume:/var/lib/mysql mysql
```

- All data written on database is stored on the volume created on the docker host.
- Even if the container is destroyed the data will be available.

SkillAssure

## Volumes



- What if you directly run the container without creating the volume?

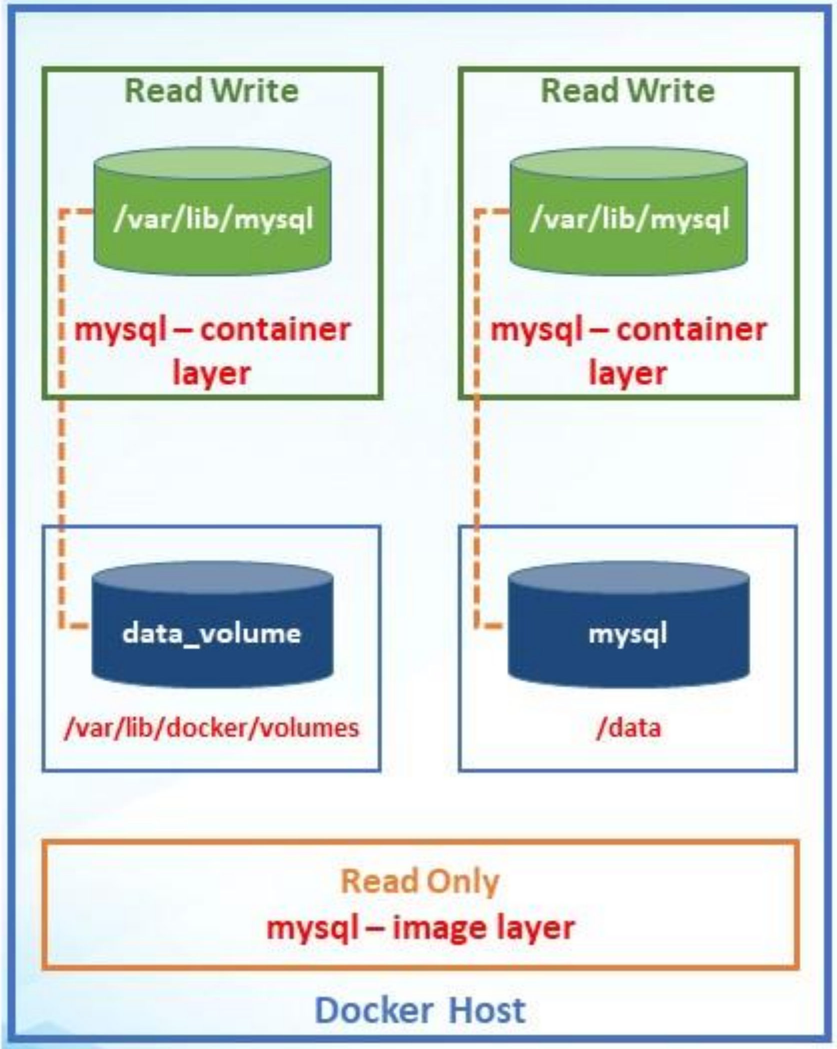
- To persist data I can mount this volume to MySql default location while running the container.

```
docker run -v data_volume2:/var/lib/mysql mysql
```

- Docker will automatically create a volume named data_volume2 and mount it to the container.

- This is called **volume mounting**

## Volumes



- What if our data is already present at another location, e.g. /data
- We would like to store database data on /data and not in the default /var/lib/docker/volumes folder.
- Create a container and provide complete path of folder to mount.

```
docker run -v /data/mysql:/var/lib/mysql mysql
```

- This is called **bind mounting**

## Volumes

- –v is an old style for mounting

```
docker run -v /data/mysql:/var/lib/mysql mysql
```

- --mount is the preferred way as it is more verbose.
- Each parameter can be specified in a key=value format

```
docker run --mount type=bind,source=/data/mysql,target=/var/lib/mysql mysql
```

## Volume Commands

- Create a volume

```
docker volume create my-vol
```

- List volumes

```
docker volume ls
```

- Inspect a volume:

```
docker volume inspect my-vol
```

- Remove a volume

```
docker volume rm my-vol
```

SkillAssure