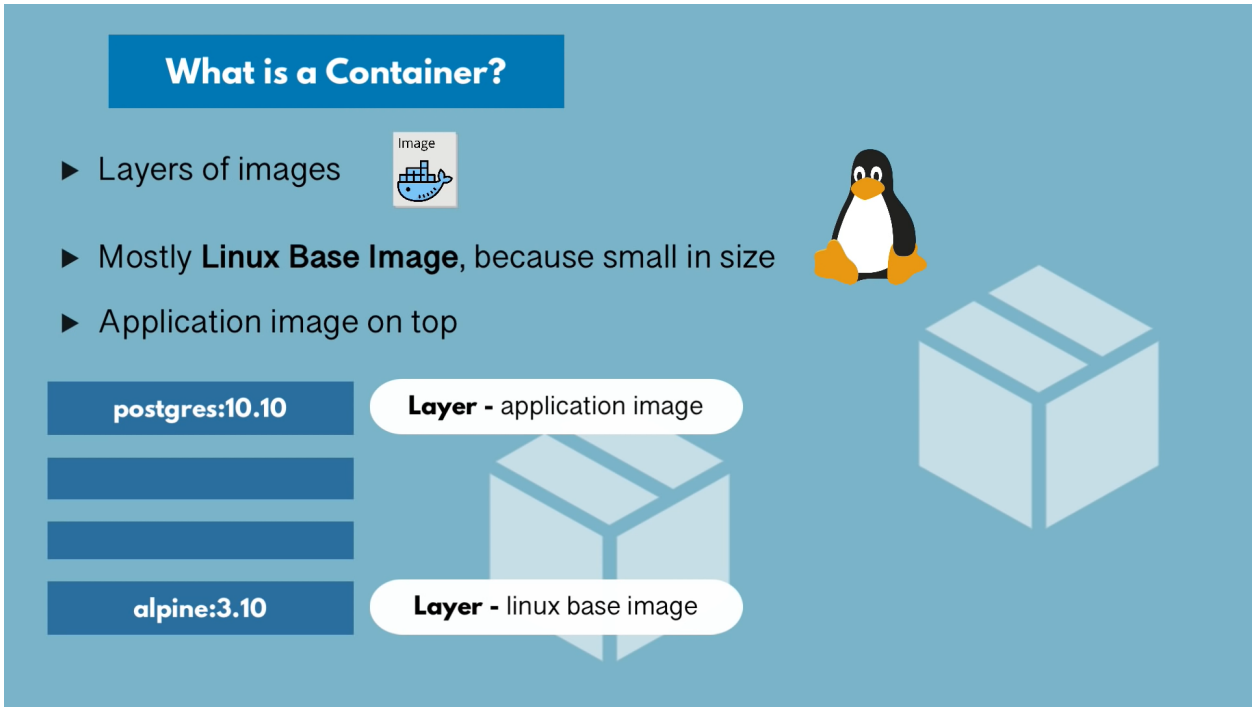
**Container**: A way of packaging application with all necessary dependencies and configurations.

A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings.

Helps in phases like

- Application Development (installing softwares on developer machine irrespective of OS)
- Application Deployment




If the same layered dependency is present it won't pull that again from the hub.
First it checks whether the image exists locally and then decides whether to pull from the hub.

The base layer will be the OS usually alpine(because it is lightweight). We may even have windows as base for applications like windows server, etc.,

The top layer will be the application.

Container images become containers at runtime and in the case of Docker containers – images become containers when they run on Docker Engine. Available for both Linux and Windows-based applications, containerized software will always run the same, regardless of the infrastructure. Containers isolate software from its environment and ensure that it works uniformly despite differences for instance between development and staging.

The same postgres application with a different version can be run on the machine.

We can run multiple containers of the same image and expose different ports. ( We may need to check the volumes)

Docker virtualizes at application level and uses the host kernel
VM virtualizes at the kernel level

Compatibility: VM can run on OS host.
We need to check if the OS supports docker natively or docker image supports the host kernel(problem with windows 10 less than, mac) to make it work we need to install docker toolbox which abstracts the kernel.

https://labs.play-with-docker.com/ – We can do lot more

Container: Running instance of an image..( fileSystem,port-binding, etc.,)

To see the images: docker images
List of running containers: docker ps
List of running and stopped containers: docker ps -a

Detached Mode:
If we don't want our terminal to be connected with the container, we can use detached mode while running the container.
—---- docker run -d redis

- - docker run – It always creates a new container from the image.It does pull if it does not exist locally and runs. No need for us to use –docker pull.

To remove a container stopped container – docker rm containerID

To start a container (by default it runs in detached mode) (restart a stopped container)

– docker start containerID

Containers can run on same port inside the docker engine
Ex: We are running two redis containers with the same 6379 as the port. However we need to change the ports while mapping the ports to the host

To get the logs of the container – docker logs containerId

To connect the terminal with a running container – docker exec -it containerId /bin/bash
Sometimes bash may not be part of the underlying OS, then we can try with sh
env – to get the environmental variables

Docker Network: Needs to be studied more

- Container to container
- Outside to container

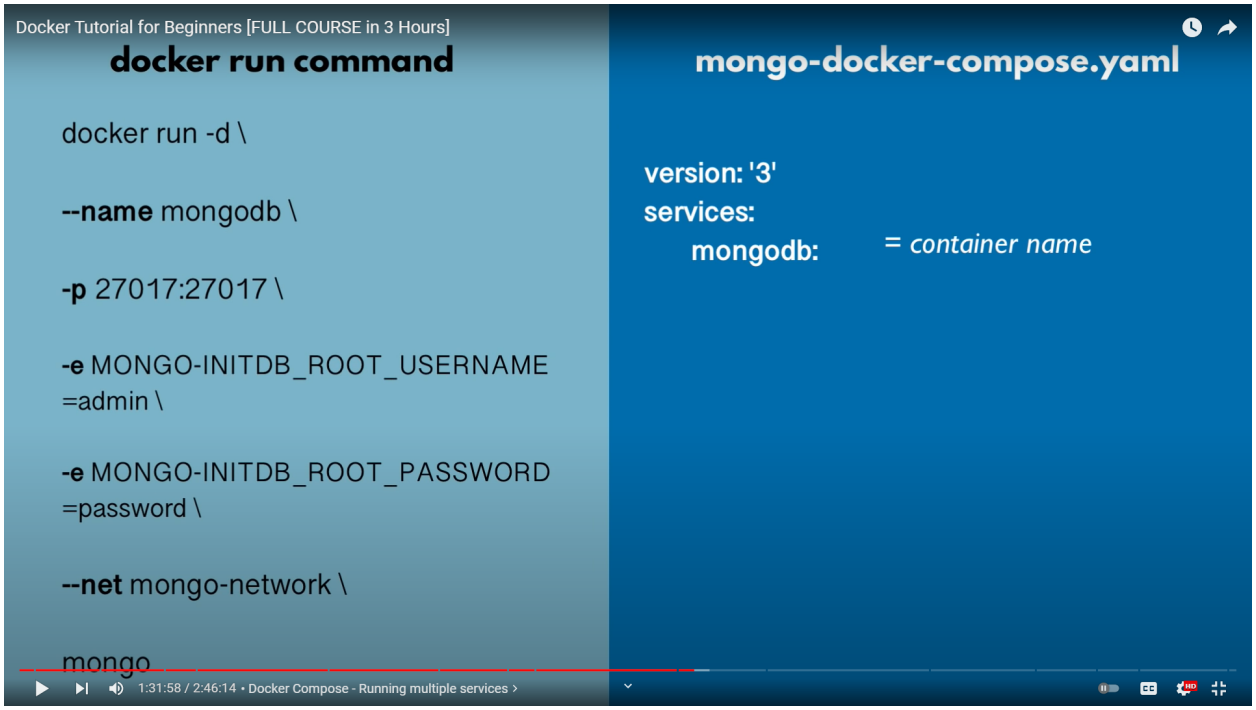
Docker compose:
To run multiple containers we can use docker-compose where we need to structure the commands

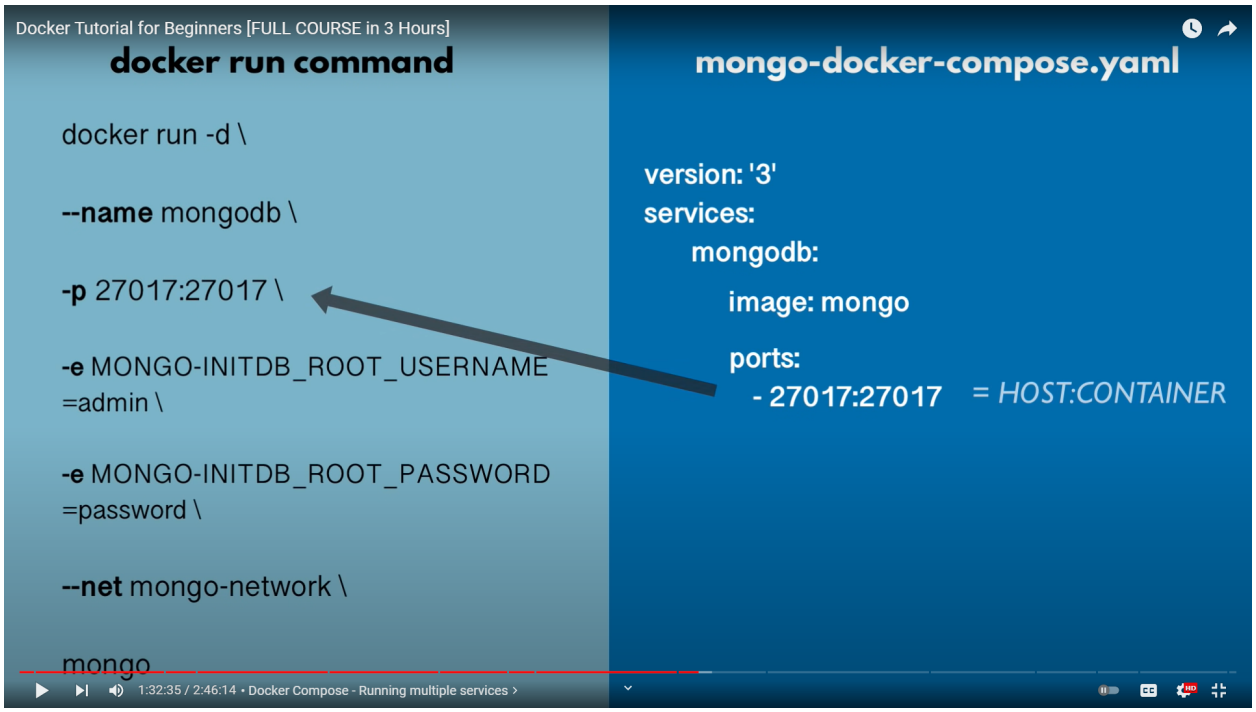
**docker run command**

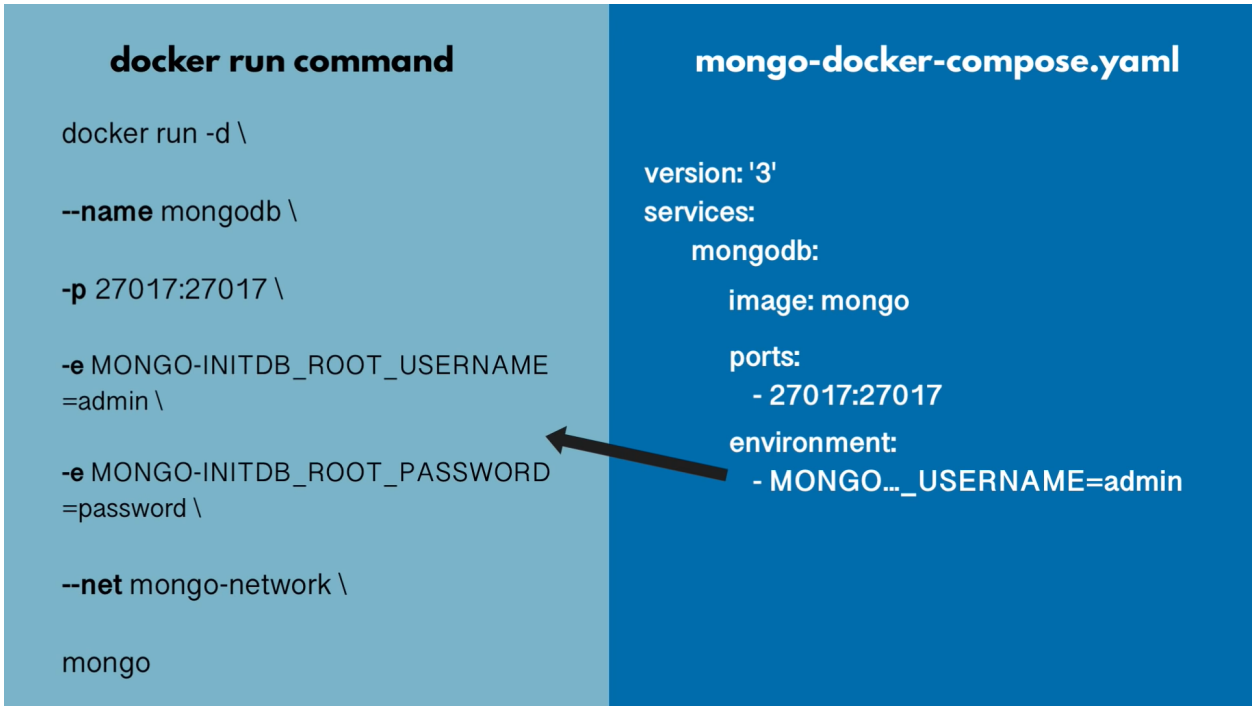
```
docker run -d \

--name mongodb \

-p 27017:27017 \

-e MONGO-INITDB_ROOT_USERNAME
=admin \

-e MONGO-INITDB_ROOT_PASSWORD
=password \

--net mongo-network \

mongo
```

**mongo-docker-compose.yaml**

```
version: '3'          = version of docker-compose
services:
    mongodb:
```


Docker Tutorial for Beginners [FULL COURSE in 3 Hours]
docker run command
docker run -d \
--name mongodb \
-p 27017:27017 \
-e MONGO-INITDB_ROOT_USERNAME =admin \
-e MONGO-INITDB_ROOT_PASSWORD =password \
--net mongo-network \
mongo
1:31:58 / 2:46:14 • Docker Compose - Running multiple services ›
mongo-docker-compose.yaml
version: '3'
services:
mongodb:
= container name


Docker Tutorial for Beginners [FULL COURSE in 3 Hours]
docker run command
docker run -d \
--name mongodb \
-p 27017:27017 \
-e MONGO-INITDB_ROOT_USERNAME =admin \
-e MONGO-INITDB_ROOT_PASSWORD =password \
--net mongo-network \
mongo
1:32:35 / 2:46:14 • Docker Compose - Running multiple services ›
mongo-docker-compose.yaml
version: '3'
services:
mongodb:
image: mongo
ports:
- 27017:27017
= HOST:CONTAINER

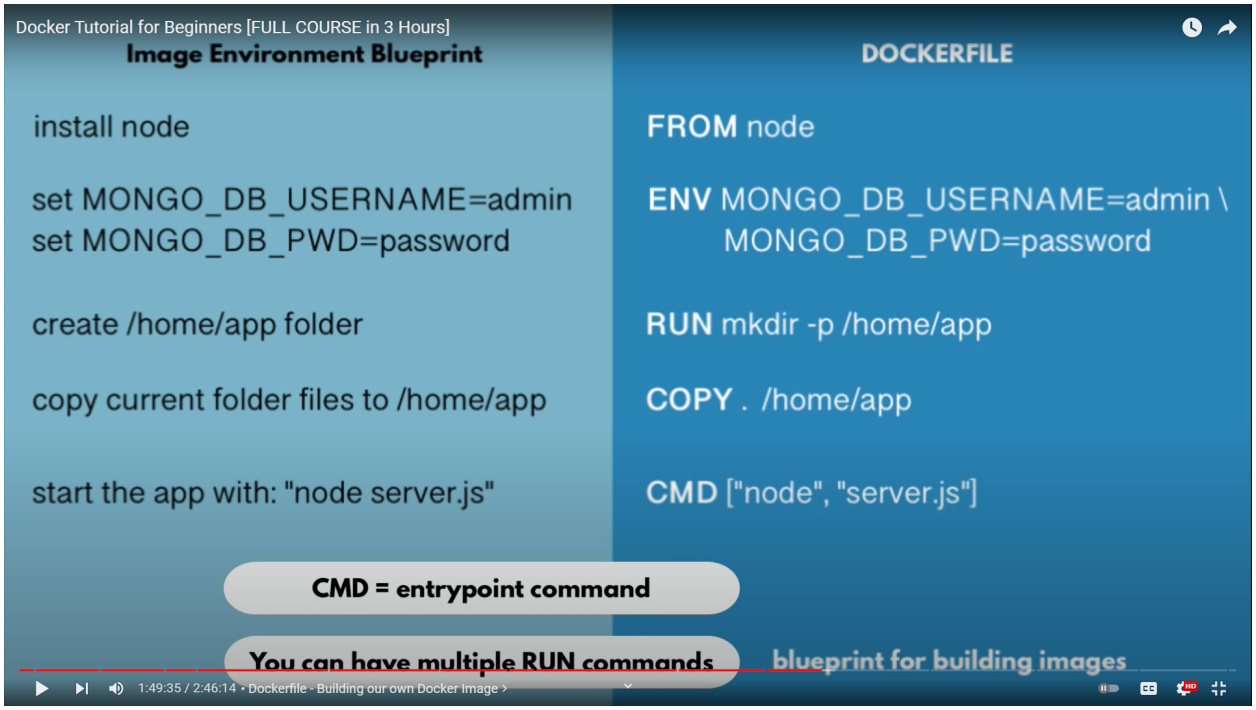



Docker Compose takes care of creating network for these containers in the yaml under the same network

docker-compose

- up - start and create containers
- down - stop and remove containers, networks, images and volumes

Whenever a container is stopped all the data that is being held will be erased.. To do the persistence we need to use volumes




The name of the docker file must be Dockerfile

To build a container from Dockerfile

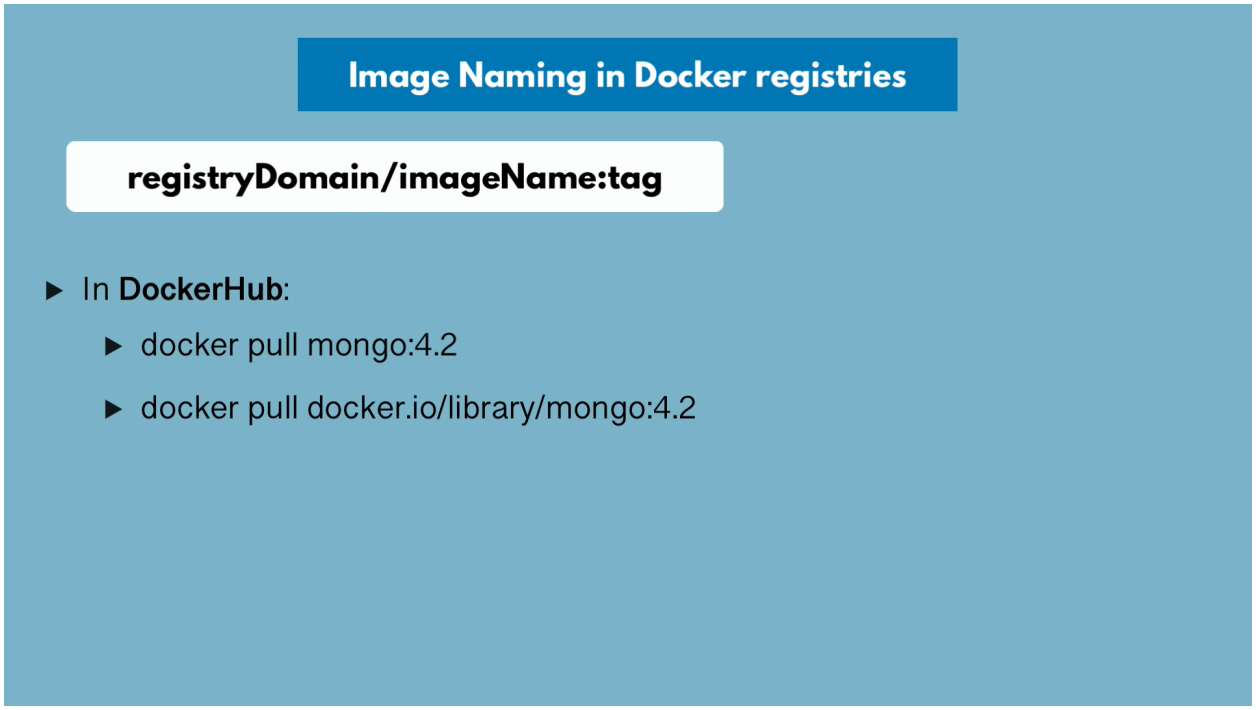
```
docker build -t tagName:version pathOfDockerfile
```

To remove an image

```
docker rmi imageTag
```

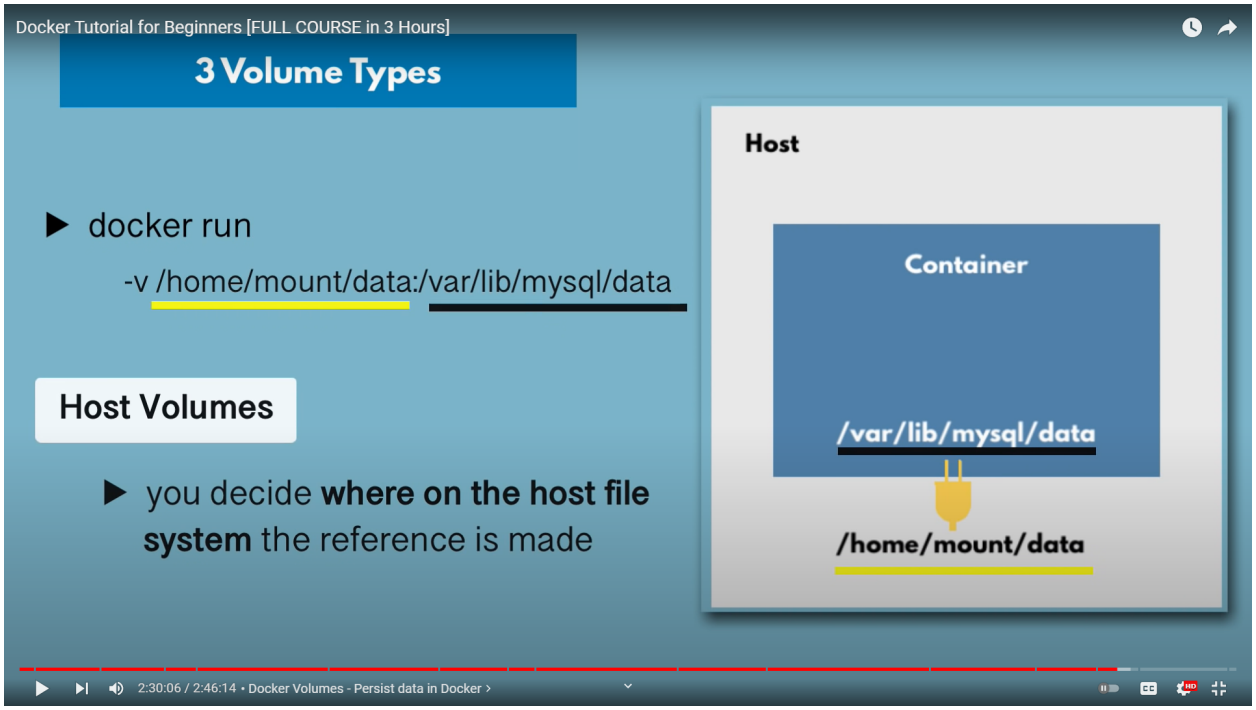
Docker tag = rename the image

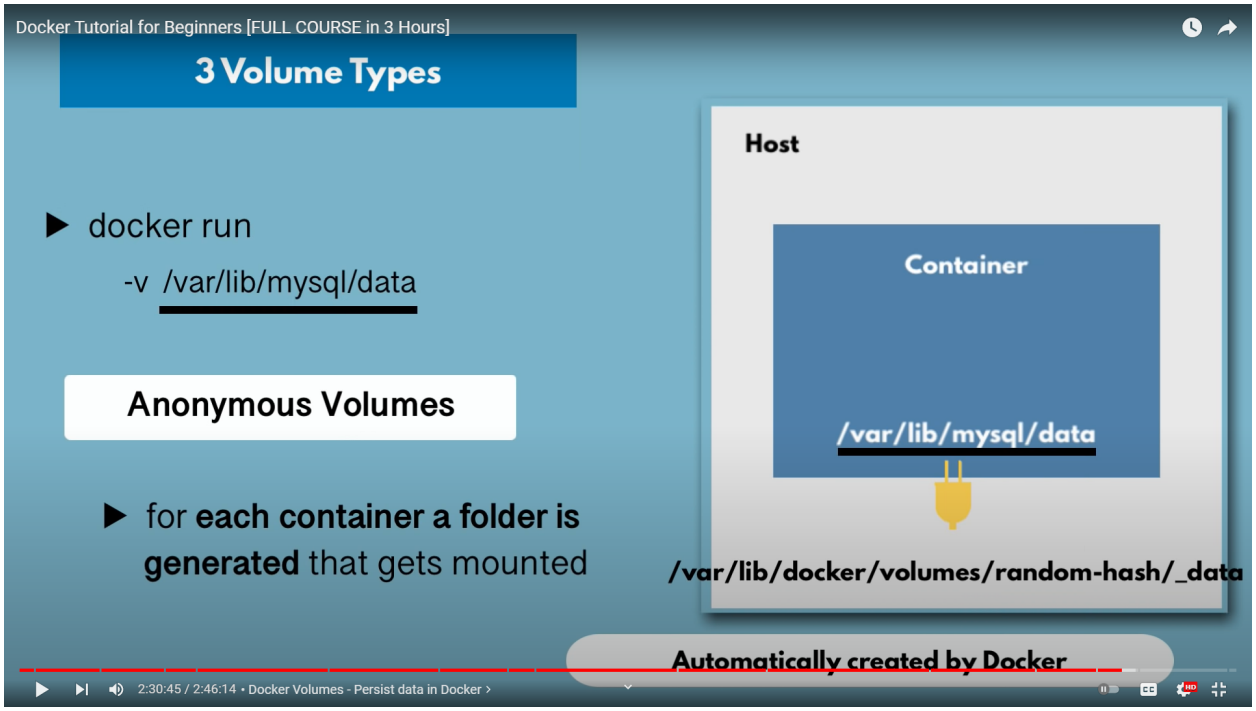
Again we need to have a final Dockerfile to make up the application fully functional with databases, logging, etc.,
We need to add the built image that pushed to private repository and all the service to deploy the whole




Docker volumes:
To persist data
For databases, stateful applications, etc.,

These are three types


Docker Tutorial for Beginners [FULL COURSE in 3 Hours]
3 Volume Types
Host
docker run
-v /home/mount/data:/var/lib/mysql/data
Container
/var/lib/mysql/data
Host Volumes
you decide where on the host file system the reference is made
/home/mount/data
2:30:06 / 2:46:14 • Docker Volumes - Persist data in Docker

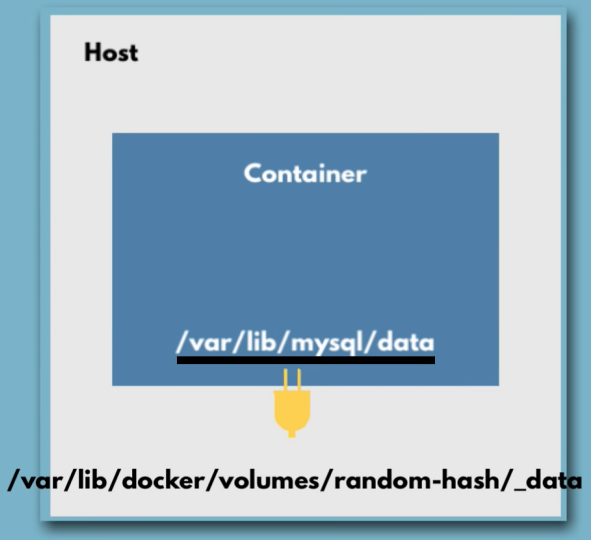

Docker Tutorial for Beginners [FULL COURSE in 3 Hours]
3 Volume Types
Host
docker run
-v /var/lib/mysql/data
Container
/var/lib/mysql/data
Anonymous Volumes
for each container a folder is generated that gets mounted
/var/lib/docker/volumes/random-hash/_data
Automatically created by Docker
2:30:45 / 2:46:14 • Docker Volumes - Persist data in Docker

## 3 Volume Types

- docker run
  -v **name:**/var/lib/mysql/data

**Named Volumes**

- you can **reference** the volume **by name**
- should be used in production

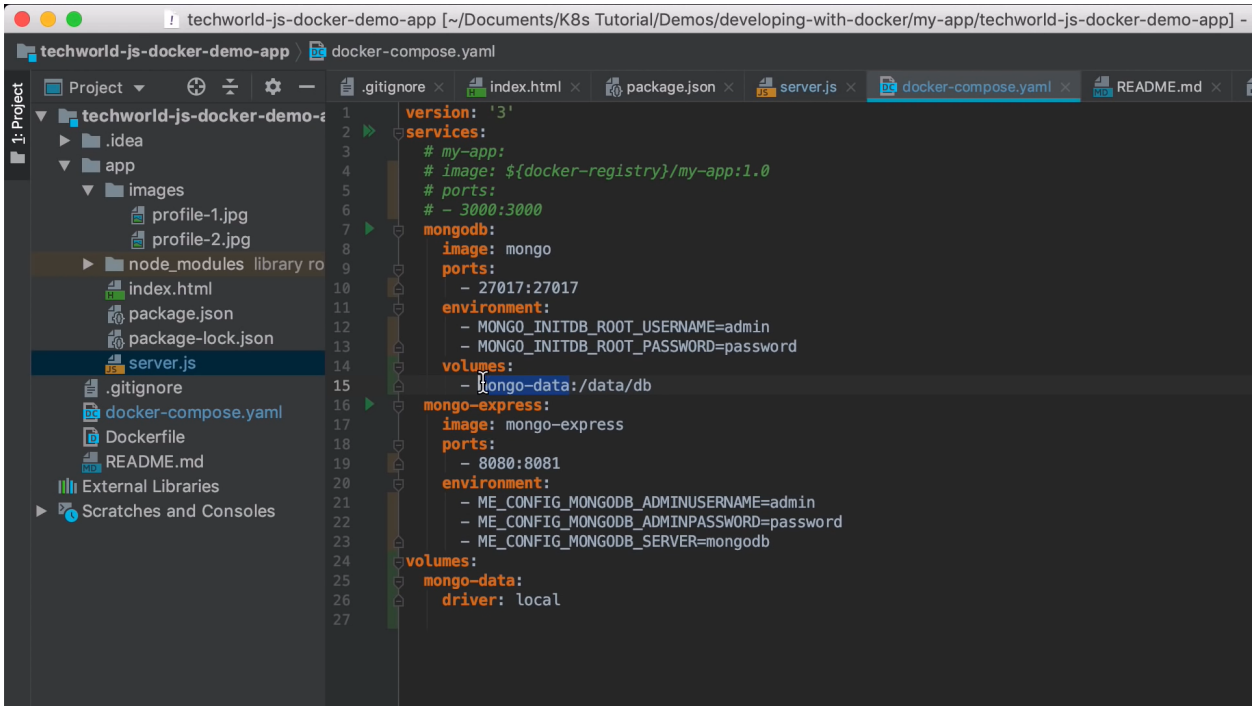


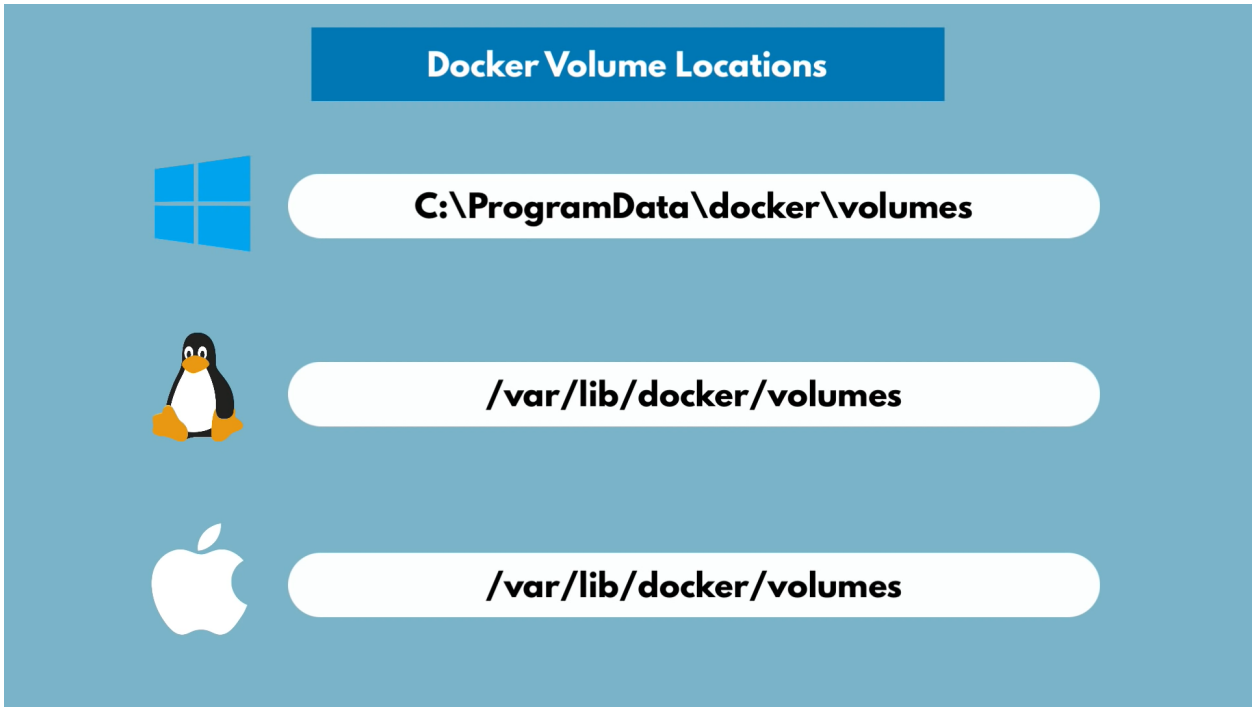

## Docker Volumes in docker-compose

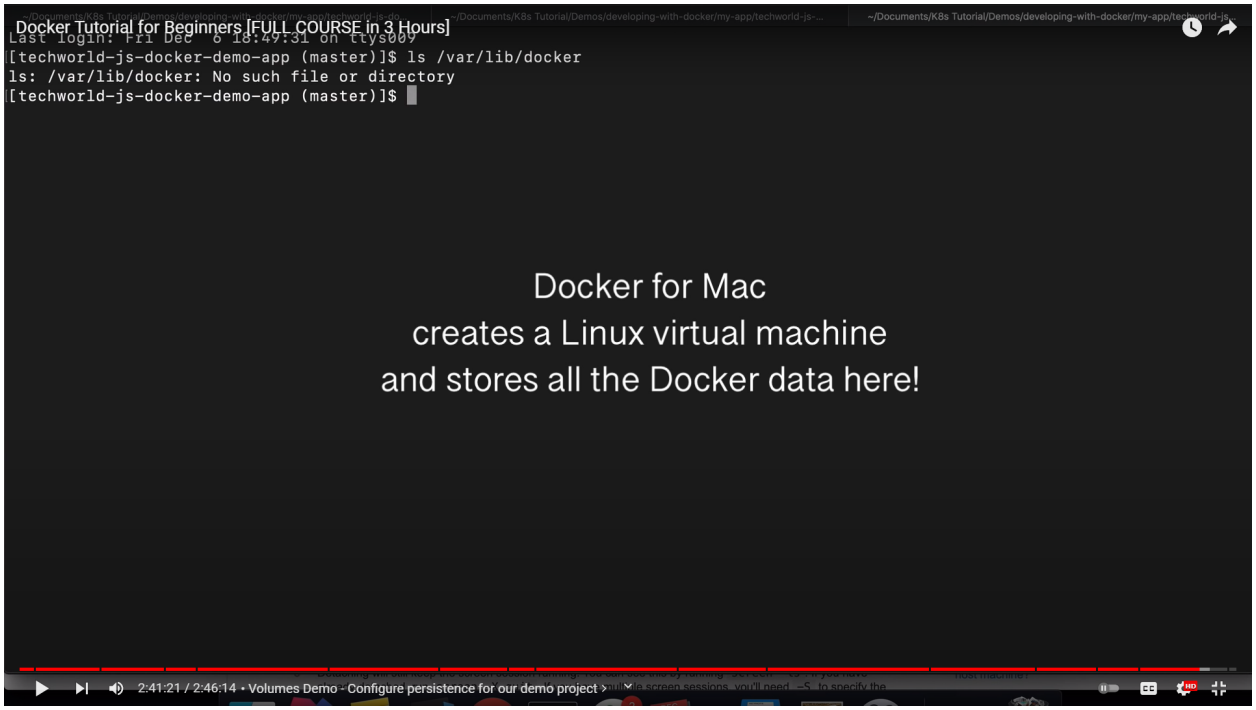
**Named Volume**

**mongo-docker-compose.yaml**

```
version: '3'

services:

    mongodb:

        image: mongo

        ports:
         - 27017:27017

        volumes:
         - db-data:/var/lib/mysql/data

    mongo-express:

        image: mongo-express
        ...

volumes:
    db-data
```


techworld-js-docker-demo-app [~/Documents/K8s Tutorial/Demos/developing-with-docker/my-app/techworld-js-docker-demo-app] -
techworld-js-docker-demo-app
docker-compose.yaml
Project
1: Project
techworld-js-docker-demo-a
.idea
app
images
profile-1.jpg
profile-2.jpg
node_modules library ro
index.html
package.json
package-lock.json
server.js
.gitignore
docker-compose.yaml
Dockerfile
README.md
External Libraries
Scratches and Consoles
.gitignore
index.html
package.json
server.js
docker-compose.yaml
README.md
version: '3'
services:
# my-app:
# image: ${docker-registry}/my-app:1.0
# ports:
# - 3000:3000
mongodb:
image: mongo
ports:
- 27017:27017
environment:
- MONGO_INITDB_ROOT_USERNAME=admin
- MONGO_INITDB_ROOT_PASSWORD=password
volumes:
- mongo-data:/data/db
mongo-express:
image: mongo-express
ports:
- 8080:8081
environment:
- ME_CONFIG_MONGODB_ADMINUSERNAME=admin
- ME_CONFIG_MONGODB_ADMINPASSWORD=password
- ME_CONFIG_MONGODB_SERVER=mongodb
volumes:
mongo-data:
driver: local


Docker Volume Locations
C:\ProgramData\docker\volumes
/var/lib/docker/volumes
/var/lib/docker/volumes


Docker Tutorial for Beginners [FULL COURSE in 3 Hours]
Last login: Fri Dec  6 18:49:31 on ttys009
[techworld-js-docker-demo-app (master)]$ ls /var/lib/docker
ls: /var/lib/docker: No such file or directory
[techworld-js-docker-demo-app (master)]$
Docker for Mac
creates a Linux virtual machine
and stores all the Docker data here!
2:41:21 / 2:46:14 • Volumes Demo - Configure persistence for our demo project