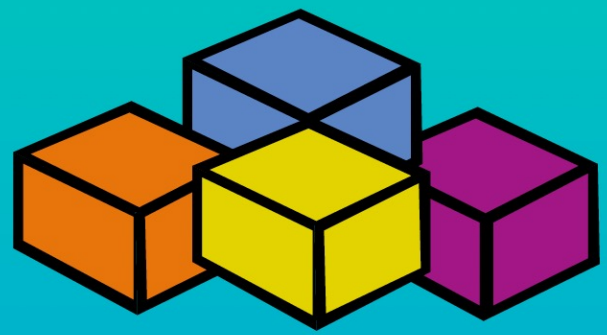
What is Docker?

- Docker is an open source **containerization platform**
- Enables developers to **package applications into containers**
- Containers existed already before Docker
- Docker made containers popular

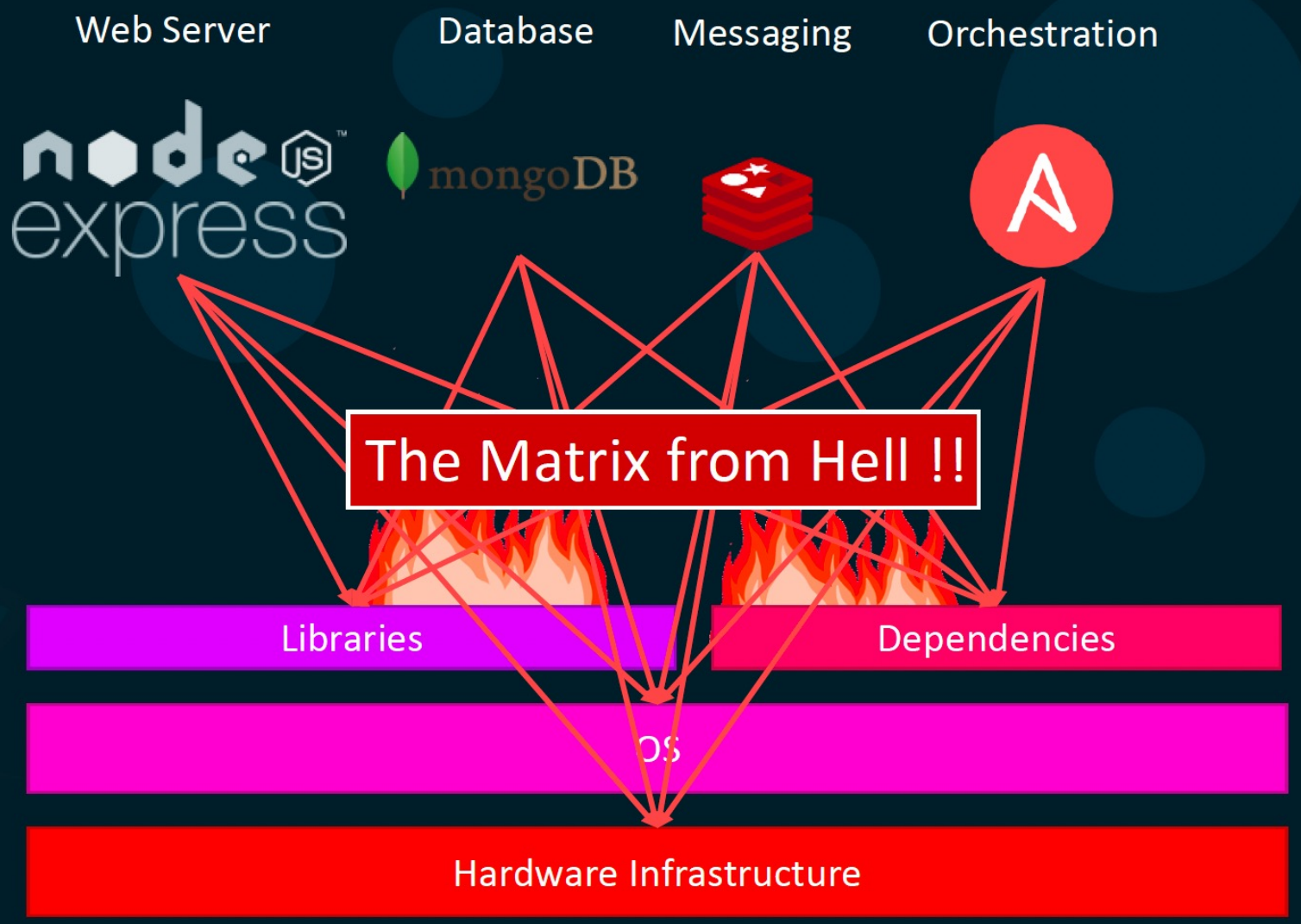
What is a container?



- A way to **package** application with **all** the **necessary dependencies** and **configuration**
- **Portable standardized artifact** for development, shipment and deployment
- Makes development and deployment **more efficient**

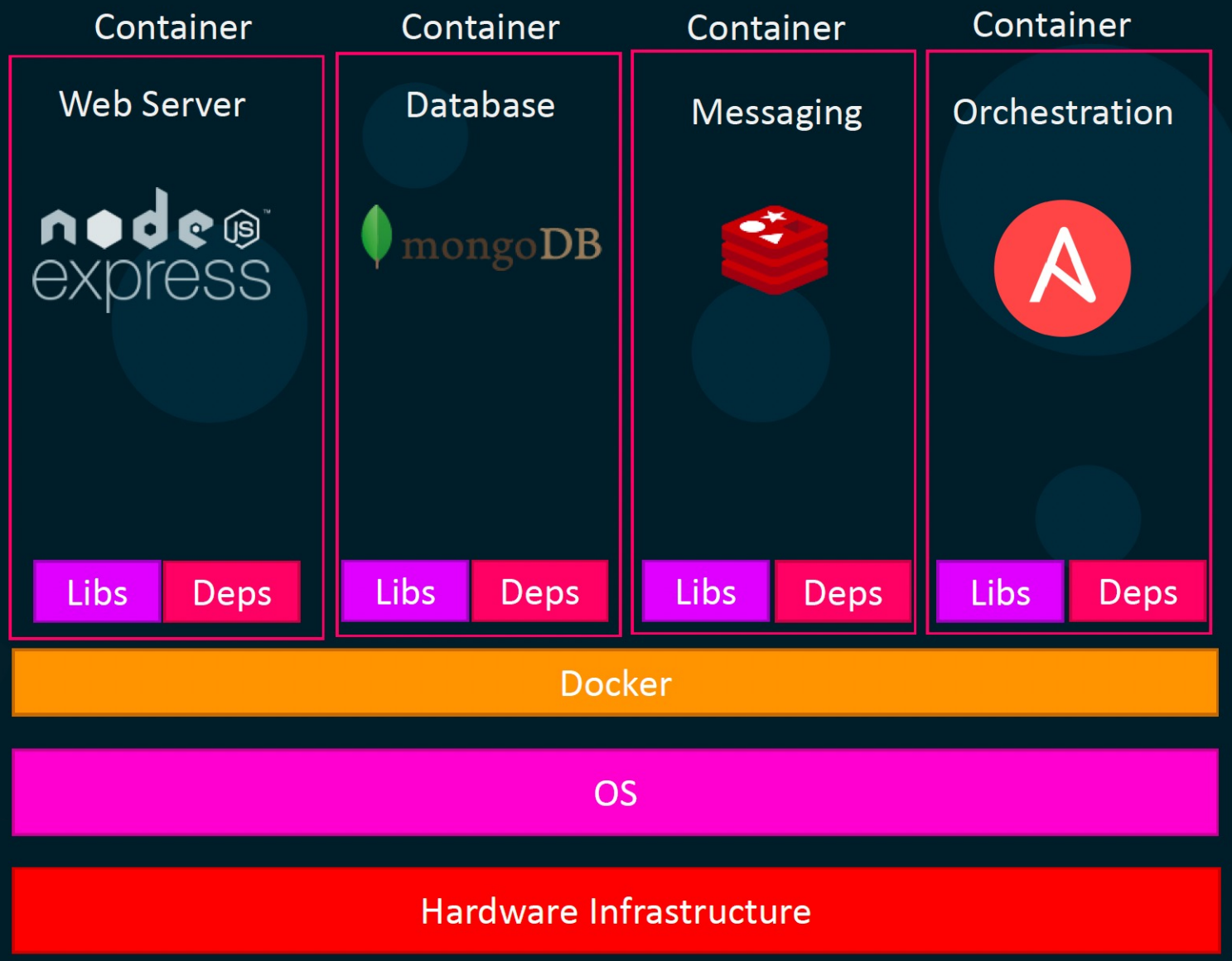
Why do you need docker?

- Compatibility/Dependency
- Long setup time
- Different Dev/Test/Prod environments



What can it do?

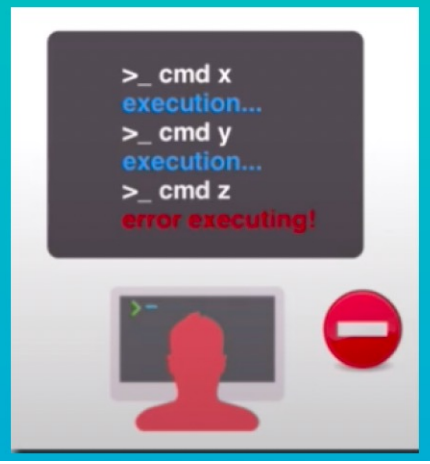
- Containerize Applications
- Run each service with its own dependencies in separate containers

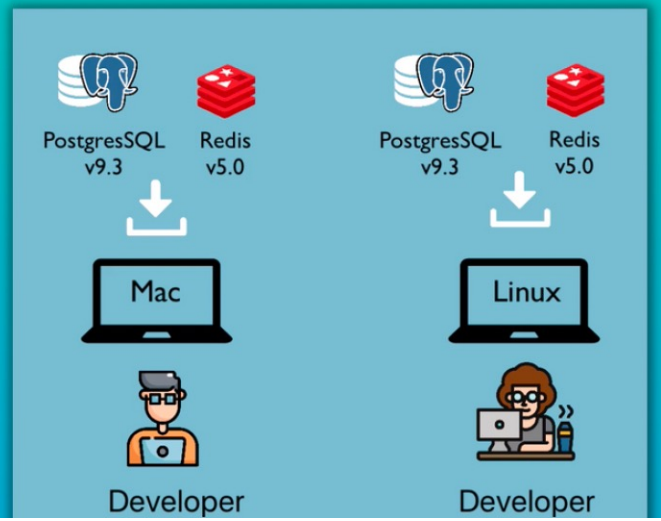


Application DEVELOPMENT before and after Docker

Before Containers

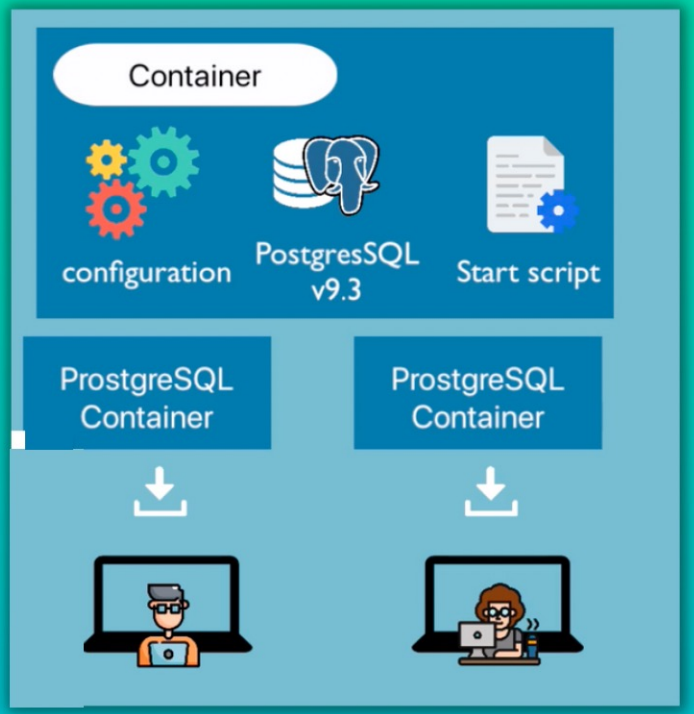
- **Installation process different** on each OS environment
- **Many steps** where something could go wrong



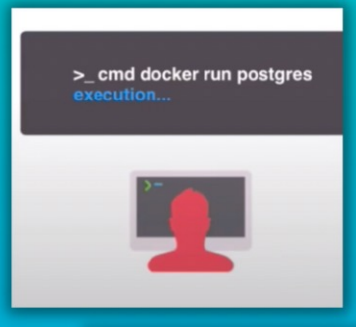


After Containers

- Own **isolated environment**
- Packaged with all needed configurations



- **1 command** to install the application
- Easily run same application with 2 different versions



Application DEPLOYMENT before and after Docker

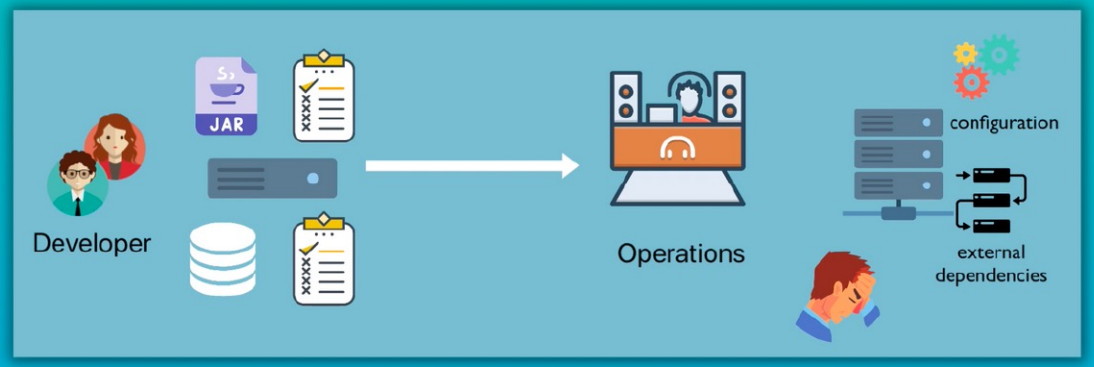
Before Containers

- Configuration on the server needed

❌ Dependency version conflicts

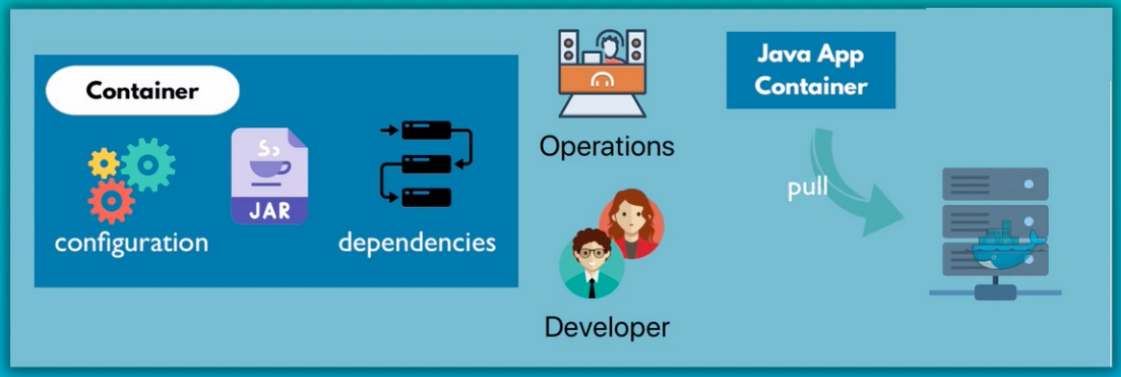
- Textual guide for deployment

❌ Misunderstanding

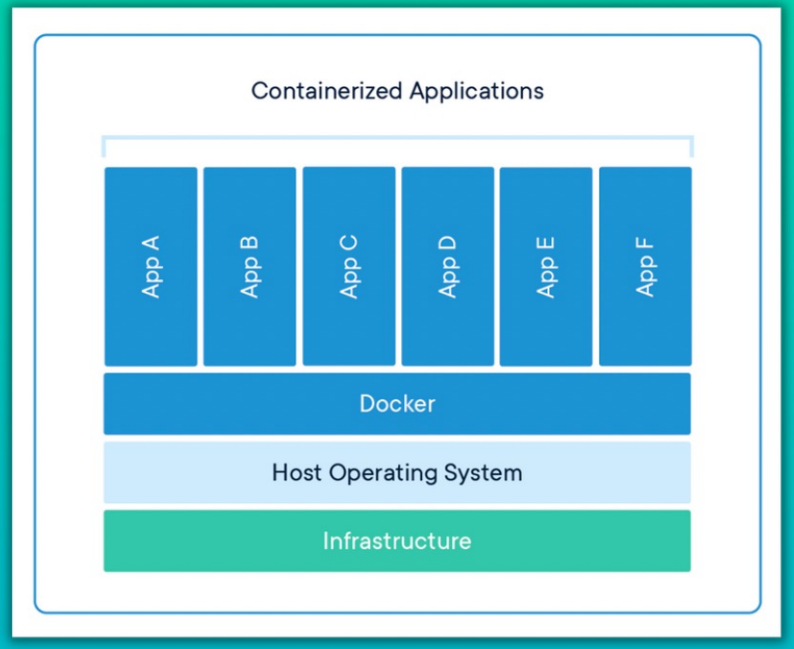


After Containers

- Devs & Ops work together to package the application in a container
- **No environment configuration needed** on server (except Container Runtime)



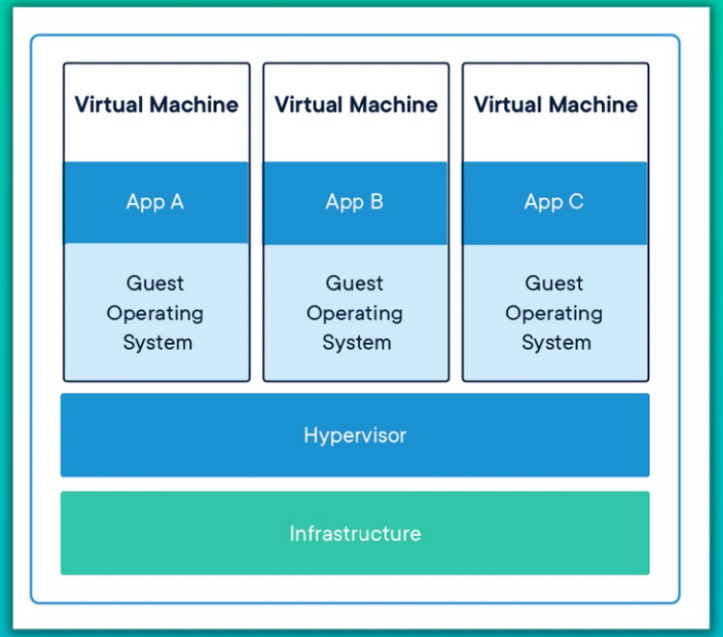
Containers vs Virtual Machines



Size: Docker Image much smaller

Speed: Docker containers start and run much faster

Compatibility: VM of any OS can run on any OS host

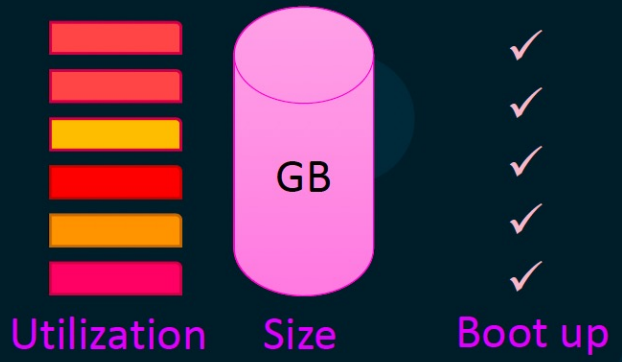


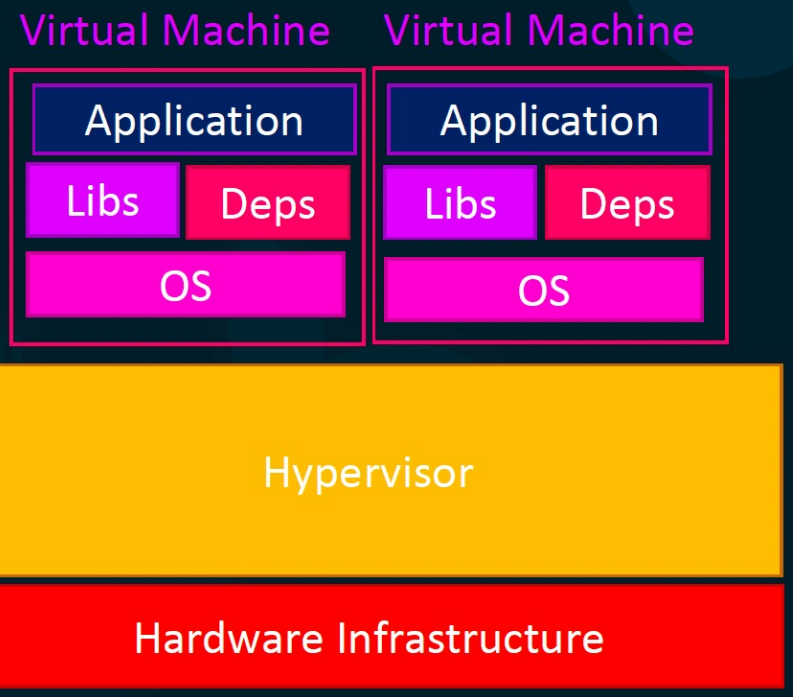
- Abstraction at the app layer
- Multiple containers share the OS kernel

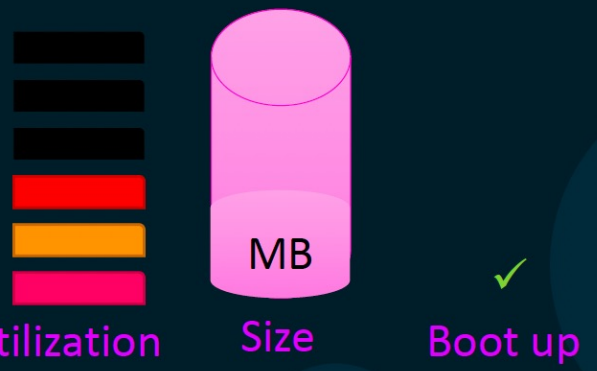
- Abstraction of physical hardware
- Each VM includes a full copy of an OS

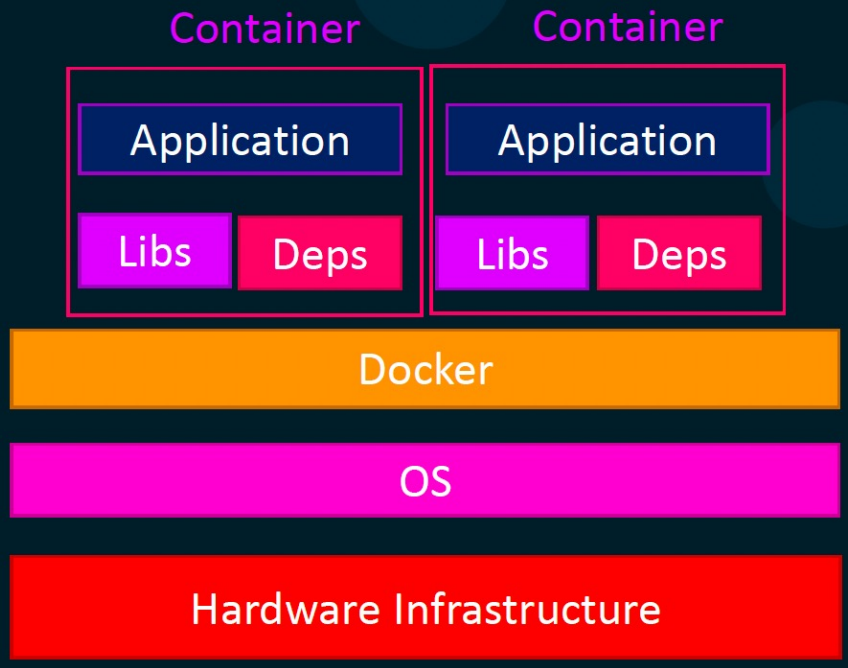
Both are **virtualization** tools

Containers vs Virtual Machines

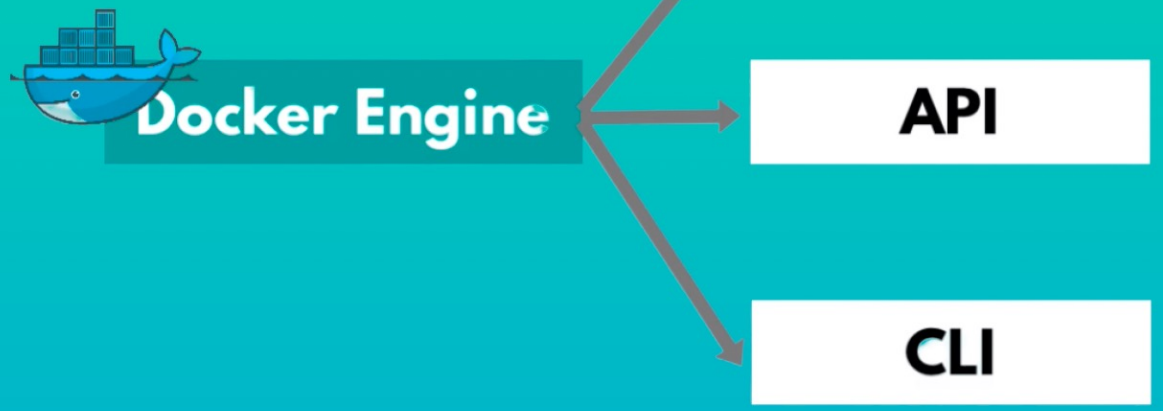








Docker Architecture & its components - 1



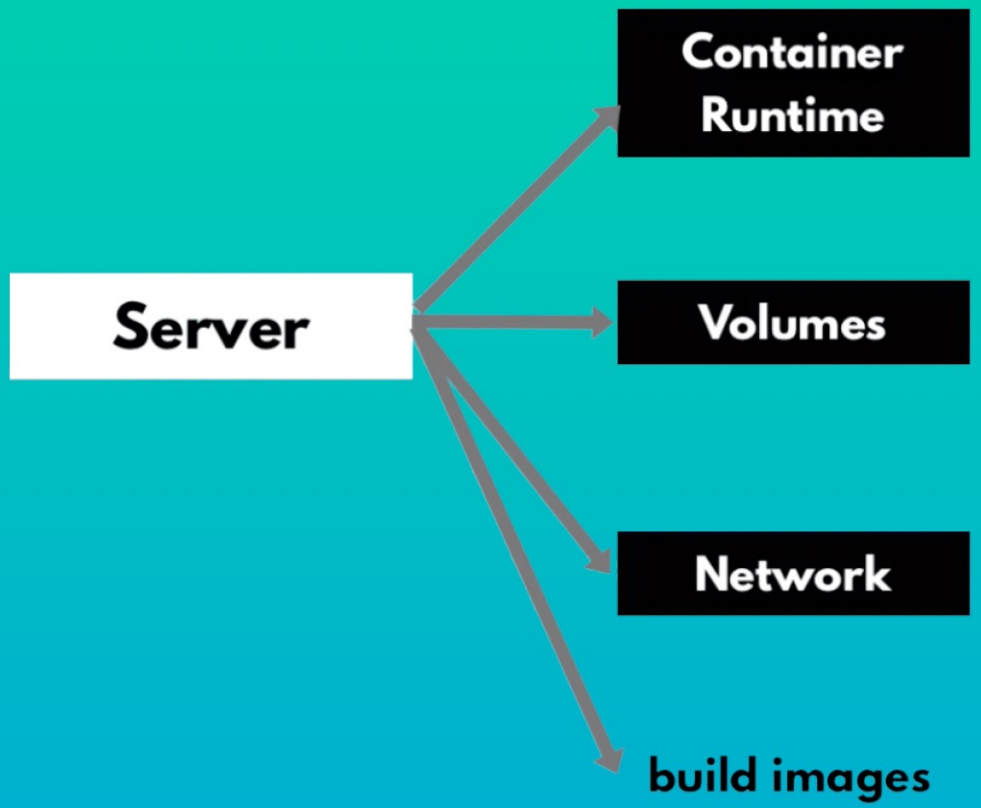
- Pulling images
- Managing images & containers

- Interacting with Docker Server

- Command Line Interface:

  Client to execute Docker commands

Docker Architecture & its components - 2

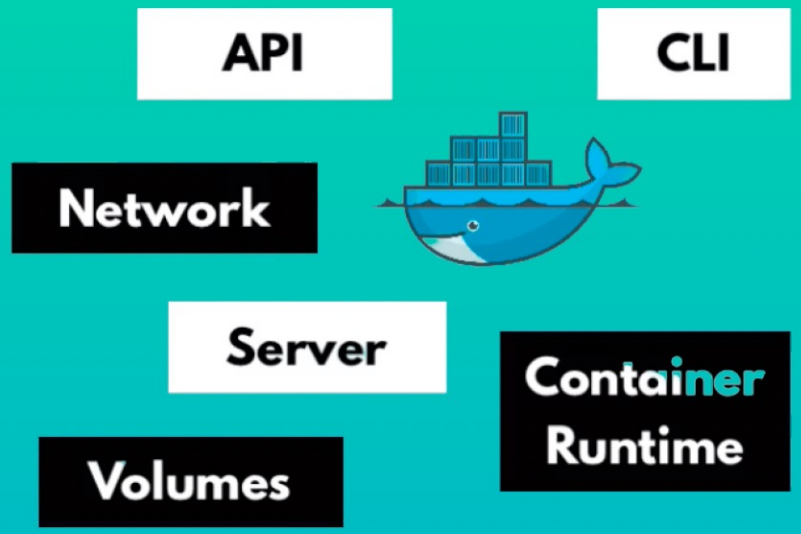


- Pulling images
- Managing container lifecycle

- Persisting data

- Configuring network for container communication

Docker Architecture & its components - 3



Docker has many functionalities in 1 application

Alternatives, if you need **only** a **container runtime**?

containerd

cri-o

Need to build an image?

buildah

Docker Installation

Linux

native support

Mac + Windows

Docker Desktop

Get Docker

Docker is an open platform for developing, shipping, and running applications. Docker enables you to separate your applications from y[redacted] can manage your infrastructure in the same ways you [redacted] ies for shipping, testing, and deploying code quickly, you can significantly reduce the delay between writing code and running it in production.

Link in Checklist

You can download and install Docker on multiple platforms. Refer to the following section and choose the best installation path for you.

Docker Desktop for Mac

A native application using the macOS sandbox security model which delivers all Docker tools to your Mac.

Docker Desktop for Windows

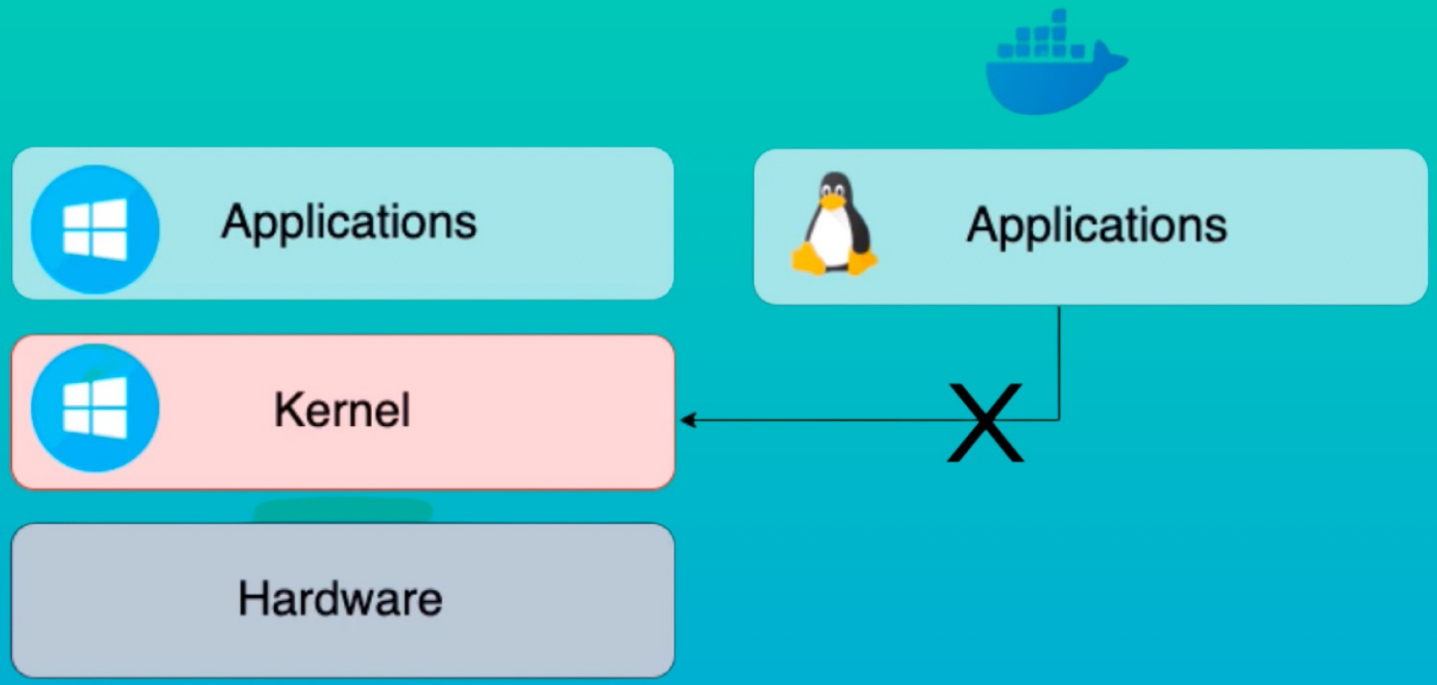
A native Windows application which delivers all Docker tools to your Windows computer.

Docker for Linux

Install Docker on a computer which already has a Linux distribution installed.

You can't run Linux container on Windows host, but for that there is

Docker Desktop for Windows and Mac



Main Docker Commands

```
docker run {image}
docker pull {image}
docker start {container}
docker stop {container}
docker images
docker ps
docker ps -a
```

1. **docker run**: creates a container from an image
2. **docker pull**: pull images from the docker repository
3. **docker start**: starts one or more stopped container
4. **docker stop**: stops a running container
5. **docker images**: lists all the locally stored docker images
6. **docker ps:** lists the running containers
7. **docker ps -a:** show all the running and exited containers

Debug Commands

1. **docker logs**: fetch logs of a container
2. **docker exec -it**: creates a new bash session in the container

```
docker run redis
```

```
Using default tag: latest
latest: Pulling from library/redis
f5d23c7fed46: Pull complete
Status: Downloaded newer image for redis:latest

1:C 31 Jul 2019 09:02:32.624 # oO0OoO0OoO0Oo Redis is starting oO0OoO0OoO0Oo
1:C 31 Jul 2019 09:02:32.624 # Redis version=5.0.5, bits=64, commit=00000000, modified=0, pid=1, just started
1:M 31 Jul 2019 09:02:32.626 # Server initialized
```

```
docker run redis:4.0
```

TAG

```
Unable to find image 'redis:4.0' locally
4.0: Pulling from library/redis
e44f086c03a2: Pull complete
Status: Downloaded newer image for redis:4.0

1:C 31 Jul 09:02:56.527 # oO0OoO0OoO0Oo Redis is starting oO0OoO0OoO0Oo
1:C 31 Jul 09:02:56.527 # Redis version=4.0.14, bits=64, commit=00000000, modified=0, pid=1, just started
1:M 31 Jul 09:02:56.530 # Server initialized
```

```
docker pull nginx
```

```
Using default tag: latest
latest: Pulling from library/nginx
fc7181108d40: Pull complete
d2e987ca2267: Pull complete
0b760b431b11: Pull complete
Digest:
sha256:96fb261b66270b900ea5a2c17a26abbfabe95506e73c3a3c65869a6dbe83223a
Status: Downloaded newer image for nginx:latest
```

```
docker run ubuntu
```

```
docker ps
```

```
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS
```

```
docker ps -a
```

```
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS                      PORTS
45aacca36850        ubuntu              "/bin/bash"         43 seconds ago      Exited (0) 41 seconds ago
```

```
docker run ubuntu sleep 5
```

docker images

```
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
nginx               latest              f68d6e55e065        4 days ago          109MB
redis               latest              4760dc956b2d        15 months ago       107MB
ubuntu              latest              f975c5035748        16 months ago       112MB
alpine              latest              3fd9065eaf02        18 months ago       4.14MB
```

docker rmi nginx

```
Untagged: nginx:latest
Untagged: nginx@sha256:96fb261b66270b900ea5a2c17a26abbfabe95506e73c3a3c65869a6dbe83223a
Deleted: sha256:f68d6e55e06520f152403e6d96d0de5c9790a89b4cfc99f4626f68146fa1dbdc
Deleted: sha256:1b0c768769e2bb66e74a205317ba531473781a78b77feef8ea6fd7be7f4044e1
Deleted: sha256:34138fb60020a180e512485fb96fd42e286fb0d86cf1fa2506b11ff6b945b03f
Deleted: sha256:cf5b3c6798f77b1f78bf4e297b27cfa5b6caa982f04caeb5de7d13c255fd7a1e
```

docker logs blissful_hopper

```
 This is a sample web application that displays a colored background.
 A color can be specified in two ways.

 1. As a command line argument with --color as the argument. Accepts one of
red,green,blue,blue2,pink,darkblue
 2. As an Environment variable APP_COLOR. Accepts one of
red,green,blue,blue2,pink,darkblue
 3. If none of the above then a random color is picked from the above list.
 Note: Command line argument precedes over environment variable.
```

Ports in Docker - 1

- **Multiple containers** can run on your host machine

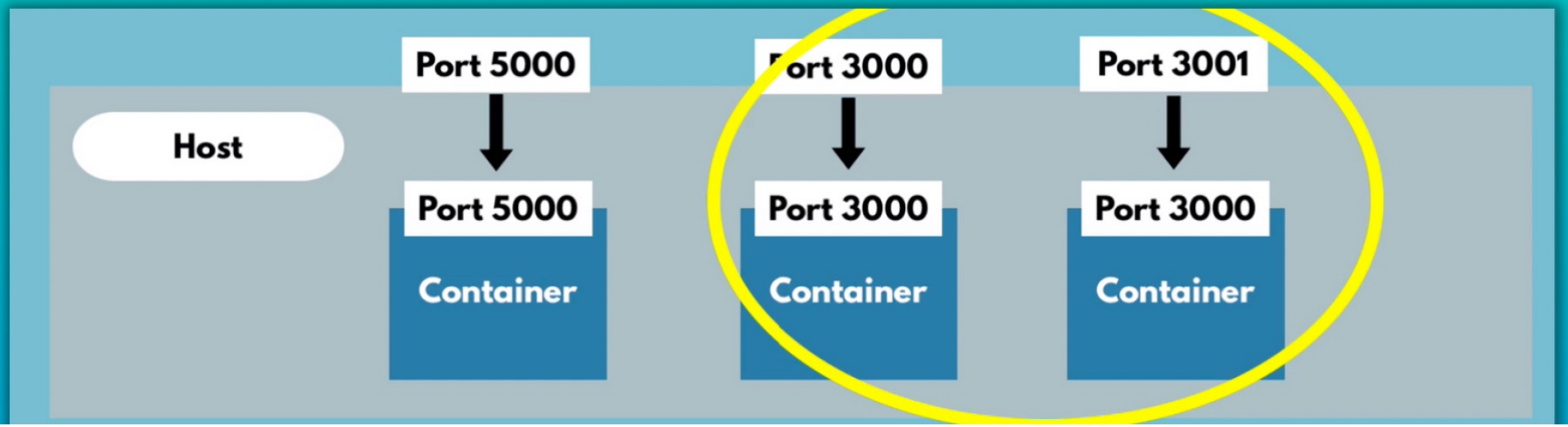
Problem:

- But your laptop has only certain ports available
- **Conflict when same port** on host machine, so we need to map to a free port on host machine:

Port Mapping:

```
docker run -p 8080:8080 {image}
```

- host port
- container port



How to create my own image?

Dockerfile

```
FROM Ubuntu

RUN apt-get update
RUN apt-get install python

RUN pip install flask
RUN pip install flask-mysql

COPY . /opt/source-code

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

1. OS - Ubuntu
2. Update apt repo
3. Install dependencies using apt
4. Install Python dependencies using pip
5. Copy source code to /opt folder
6. Run the web server using “flask” command

```
docker build Dockerfile -t mmumshad/my-custom-app
```

```
docker push mmumshad/my-custom-app
```

Dockerfile

Dockerfile

```
INSTRUCTION ARGUMENT
```

Dockerfile

```
FROM Ubuntu

RUN apt-get update
RUN apt-get install python

RUN pip install flask
RUN pip install flask-mysql

COPY . /opt/source-code

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

Start from a base OS or another image

Install all dependencies

Copy source code

Specify Entrypoint

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
```

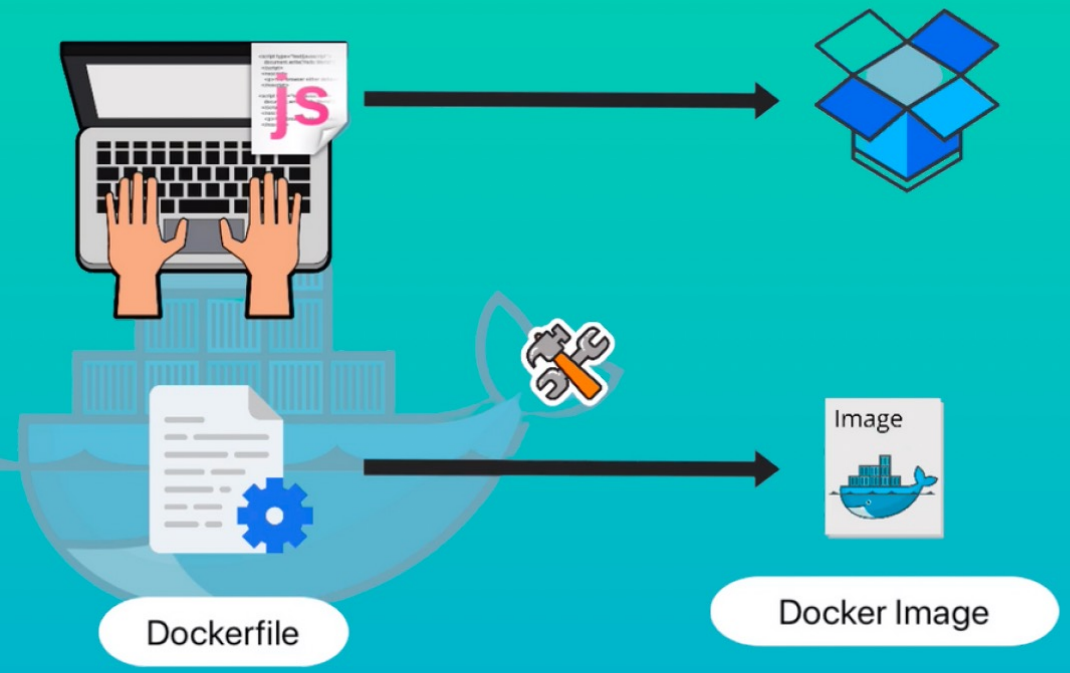
```
docker build Dockerfile -t mmumshad/my-custom-app
```

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

```
root@osboxes:/root/simple-webapp-docker # docker history mmumshad/simple-webapp
IMAGE               CREATED             CREATED BY                                      SIZE                COMMENT
1a45ba829f10        About an hour ago   /bin/sh -c #(nop)  ENTRYPOINT ["/bin/sh" "...   0B
37d37ed8fe99        About an hour ago   /bin/sh -c #(nop) COPY file:29b92853d73898...   229B
d6aaebf8ded0        About an hour ago   /bin/sh -c pip install flask flask-mysql        6.39MB
e4c055538e60        About an hour ago   /bin/sh -c apt-get update && apt-get insta...   306MB
ccc7a11d65b1        2 weeks ago         /bin/sh -c #(nop)  CMD ["/bin/bash"]            0B
<missing>           2 weeks ago         /bin/sh -c mkdir -p /run/systemd && echo '...   7B
<missing>           2 weeks ago         /bin/sh -c sed -i 's/^#\s*\(deb.*universe\...   2.76kB
<missing>           2 weeks ago         /bin/sh -c rm -rf /var/lib/apt/lists/*          0B
<missing>           2 weeks ago         /bin/sh -c set -xe   && echo '#!/bin/sh' >...   745B
<missing>           2 weeks ago         /bin/sh -c #(nop) ADD file:39d3593ea220e68...   120MB
```

Dockerfile - 1

A simple text file that consists of **instructions to build Docker images**



- Some common Dockerfile commands:

Image Environment Blueprint	DOCKERFILE
install node	**FROM** node
set MONGO_DB_USERNAME=admin set MONGO_DB_PWD=password	**ENV** MONGO_DB_USERNAME=admin \ MONGO_DB_PWD=password
create /home/app folder	**RUN** mkdir -p /home/app
copy current folder files to /home/app	**COPY** . /home/app
start the app with: "node server.js"	**CMD** ["node", "server.js"]

CMD = entrypoint command

You can have multiple RUN commands

blueprint for building images

Dockerfile - 2

- Each instruction in a Dockerfile results in an Image Layer:

- It's common to start with an existing base image in your application's Dockerfile:

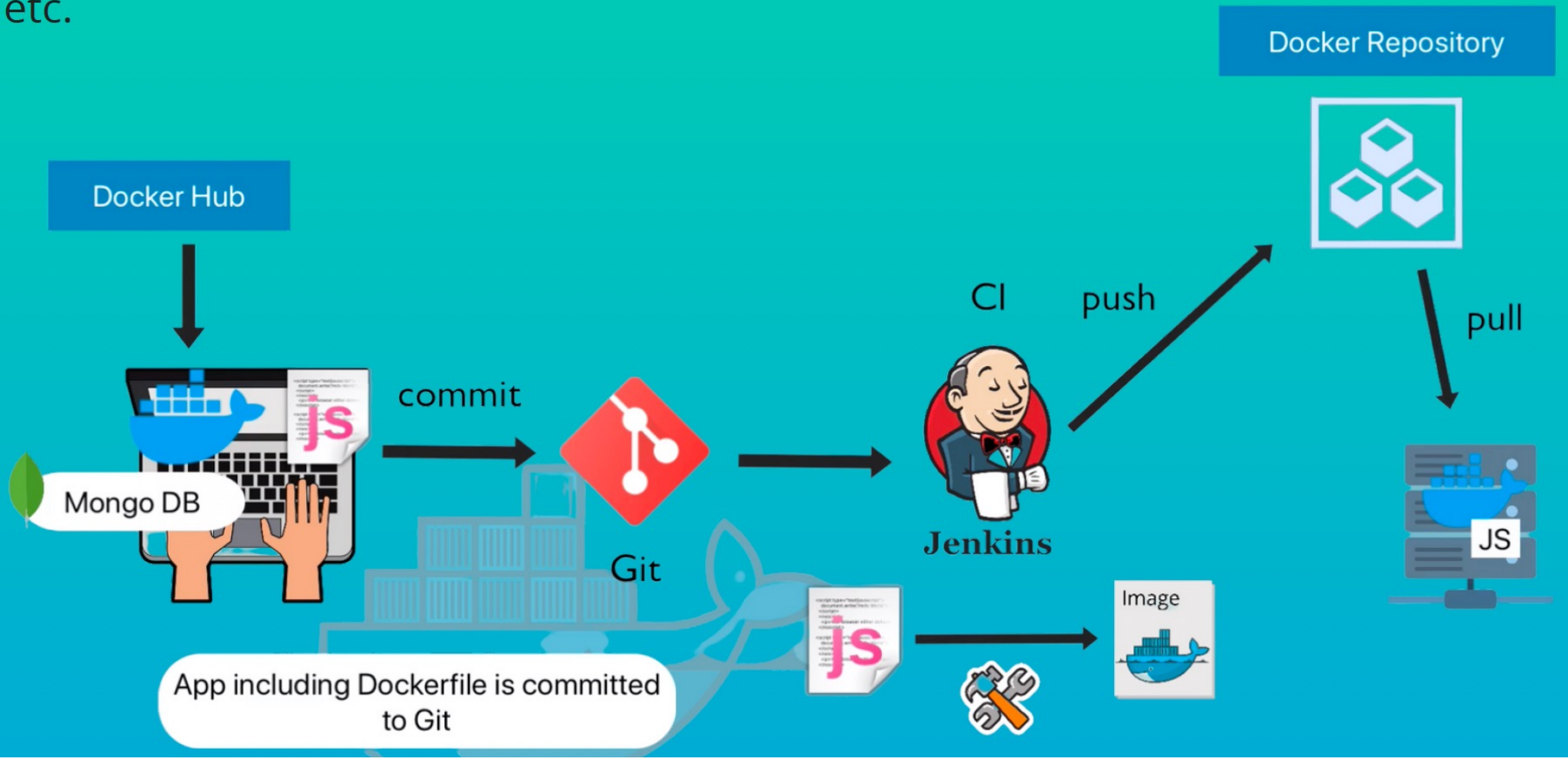
- Command to build an image from a Dockerfile and a context
- The build's context is the set of files at a specified location

```
docker build -t my-app:1.0 .
```

Uses the current directory (.) as build context

Dockerfile - 3

- Dockerfile is **used in CI/CD** to build the docker image artifact, which will be pushed to Docker repo,
- Docker image can then be pull to multiple remote servers or pulled to locally for development and testing etc.



Docker Compose - 1

Docker Compose is a tool for defining and **running multiple docker containers**

- YAML file to configure your application's services:
- You can **maintain and update configuration more easily** than with *docker run* command

docker run command

```
docker run -d \
--name mongo-express \
-p 8080:8080 \

-e ME_CONFIG_MONGODB_
   ADMINUSERNAME=admin \
-e ME_CONFIG_MONGODB_
   ADMINPASSWORD=password \
-e ME_CONFIG_MONGODB_
   SERVER=mongodb \

--net mongo-network \
mongo-express
```

mongo-docker-compose.yaml

```
version: '3'
services:
    mongodb:
        image: mongo
        ...
    mongo-express:
        image: mongo-express
        ports:
          - 8080:8080
      environment:
        - ME_CONFIG_MONGODB_A....
        ...
```

- Docker Compose automatically creates a **common docker network** for docker containers

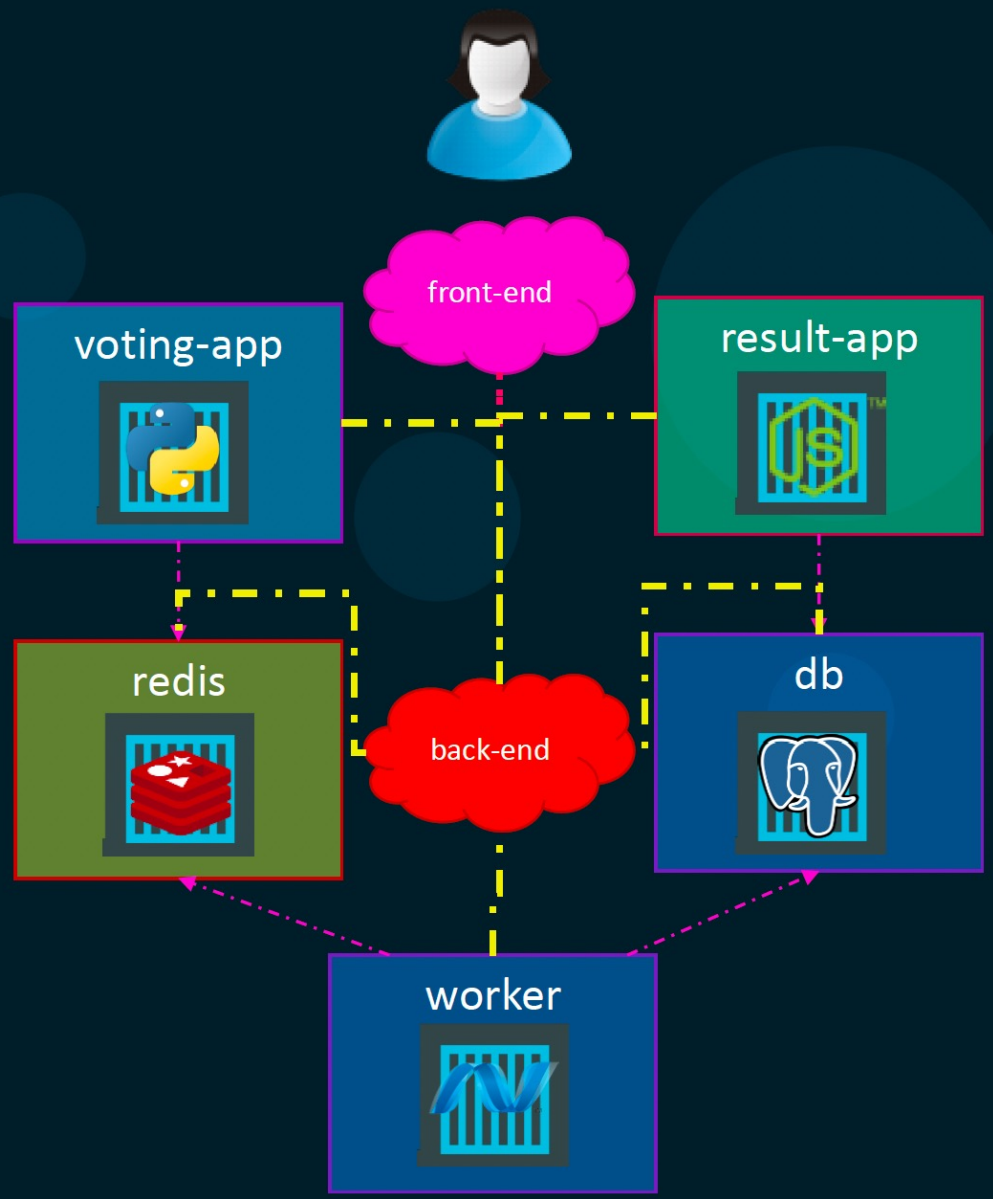
Docker compose

docker-compose.yml

```
version: 2
services:
    redis:
        image: redis

        networks:
            - back-end
    db:
        image: postgres:9.4
        networks:
            - back-end
    vote:
        image: voting-app
        networks:
            - front-end
            - back-end
    result:
        image: result
        networks:
            - front-end
            - back-end

networks:
    front-end:
    back-end:
```



Private Docker Repository - 1

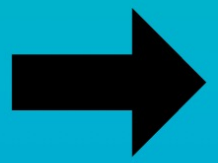
- When you work in a company, you will be working with a private docker registry
- **Public = DockerHub**, Example for **Private** = Amazon Elastic Container Registry "**AWS ECR**"

Difference to DockerHub

1. You need to **login,** so authenticate with the registry before fetching or pushing the image
2. **Tag** your image with the **registry address and name**,
3. Push the tagged image

```
docker login
- reponame
- username
- password

docker tag {repo-name}:{image-version}
docker push {tagged-image}
```



registryDomain/imageName:tag

Private Docker Repository - 2

In DockerHub

- **docker.io is default** repository

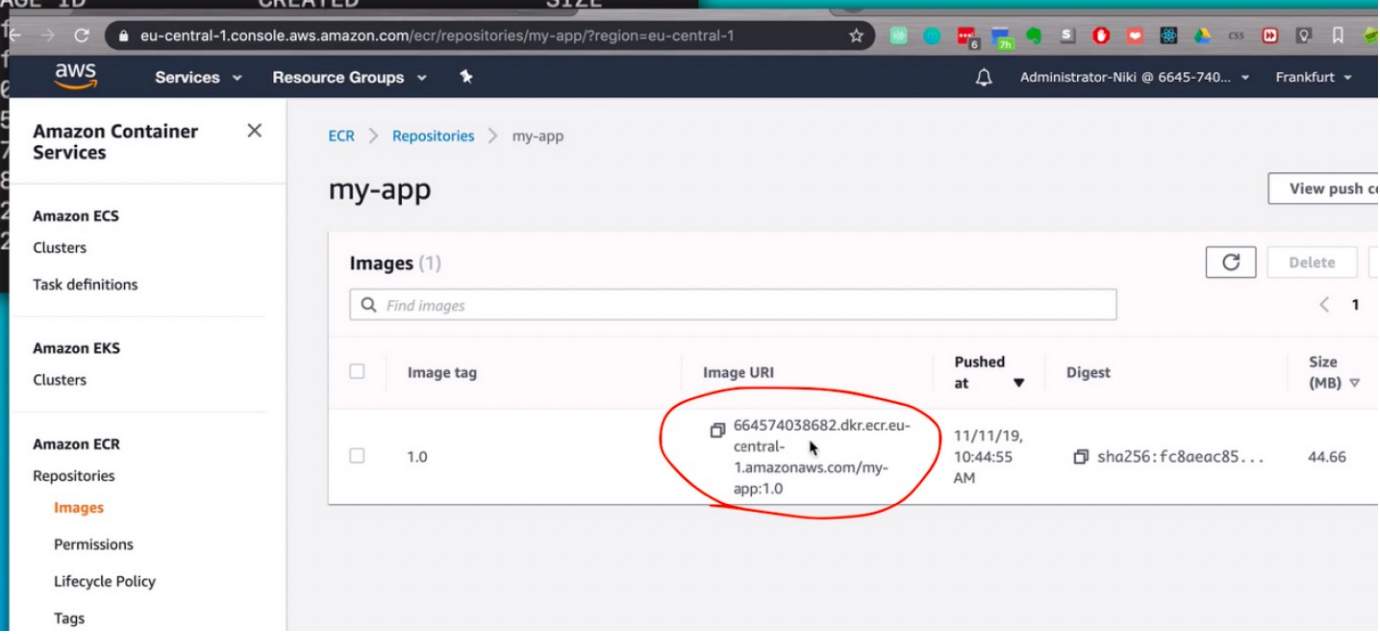
`docker pull mongo:4.2` = `docker pull docker.io/library/mongo:4.2`

In AWS ECR or any other private registry

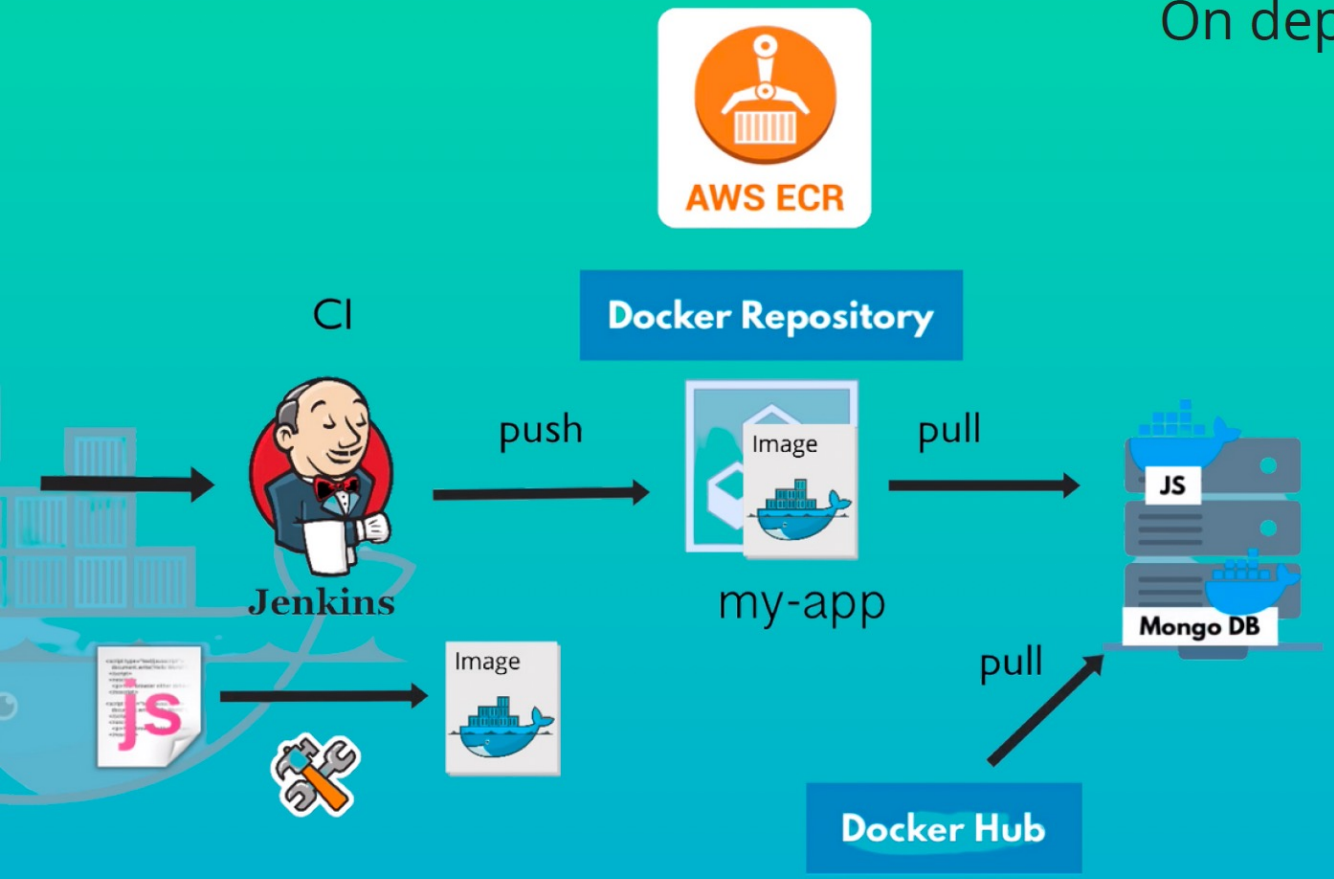
- Need to include the registry domain

`docker pull 523450290.dkr.ecr.eu-central-1.amazonaws.com/my-app:1.0`

```
[~]$ docker tag my-app:1.0 664574038682.dkr.ecr.eu-central-1.amazonaws.com/my-app:1.0
[~]$ docker images
REPOSITORY                                                  TAG                 IMAGE ID            CREATED             SIZE
664574038682.dkr.ecr.eu-central-1.amazonaws.com/my-app      1.0
my-app                                                      1.0
node                                                        13-alpine
mongo                                                       latest
mongo-express                                               latest
redis                                                       4.0
redis                                                       latest
postgres                                                    9.6.5
[~]$ docker push 664574038682.dkr.ecr.eu-central-1.amazonaws.com/my-app:1.0
```



Deploy Docker Containers on a Remote Server



On deployment server, you can pull from both:

Private Images (Your JS application, company internal libraries)

↓

from **private docker repository** (AWS ECR)

Public Images (Mysql, MongoDB)

↓

from **public docker repository** (DockerHub)

Docker Volumes - 1

Volumes are the way to **persist data** generated by and used by Docker containers

Why Volume are needed for persistence?

- Data is stored on the virtual file system of the container. So when container is removed, the data is deleted as well.

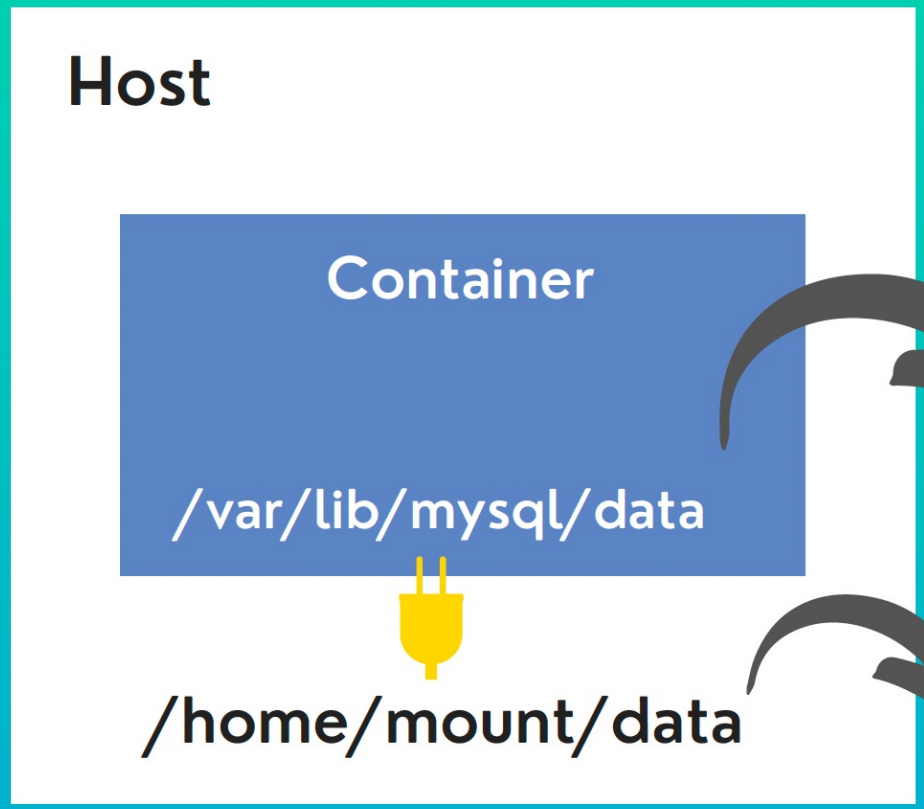
Important for

- Databases
- Other stateful applications

By default:
Data is gone when removing the container

Docker Volumes - 2

How Docker Volumes work?



- Folder in physical host file system is **mounted** into the virtual file system of Docker
- Data gets **automatically replicated**

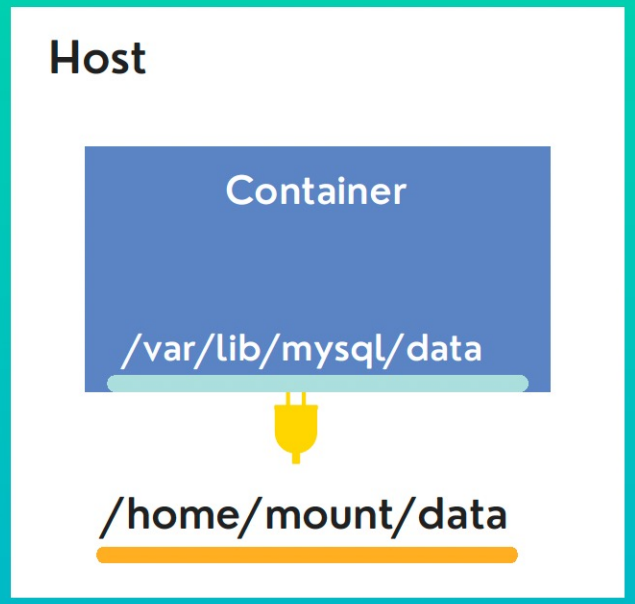
Virtual File System inside container

Host File System (physical)

Docker Volumes - 3

3 Volume Types

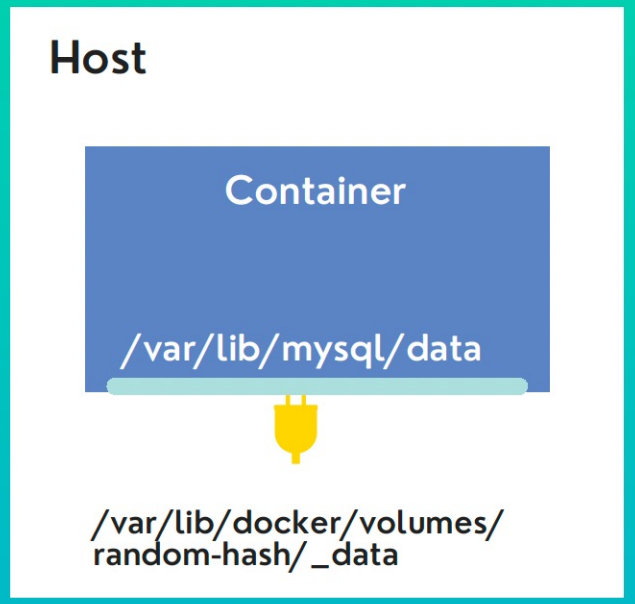
Host Volumes



- You decide **where on the host file system** the reference is made

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

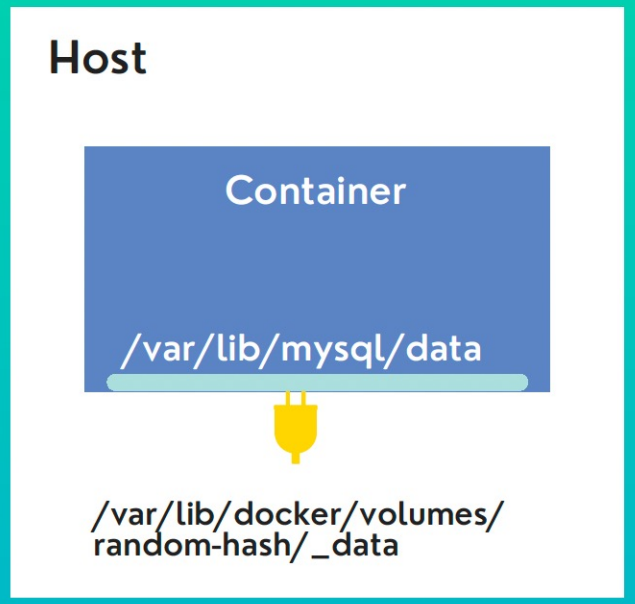
Anonymous Volumes



- For **each container a folder is generated** that gets mounted

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

Named Volumes



- You can **reference** the volume **by name**
- Should be used in production

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

Docker Volumes - 4

Docker Volumes in docker-compose file:

mongo-docker-compose.yaml

Docker Best Practices - 1

- **Security**: Only use official/trusted Docker images as base image to avoid malware
- **Security**: Use specific image versions
- **Size & Security**: Use minimal base images (e.g. prefer alpine-based images over full-fledged system OS images)
- **Size**: Optimize caching image layers
- **Size**: Use .dockerignore to exclude files and folders
- **Cleaner Dockerfile**: Make use of "Multi-Stage Builds"

Improve Security

Optimize Image Size

Write cleaner and more maintainable Dockerfiles

Docker Best Practices - 2

- **Security**: Use least privileged user (create a dedicated user and group with minimal permissions to run the application)
- **Security**: Scan your images for vulnerabilities
- **Security**: Don't leak sensitive information to Docker images

Improve Security

Optimize Image Size

Write cleaner and more maintainable Dockerfiles