

Containers with Docker

Key Takeaways

Introduction to Containers & Docker

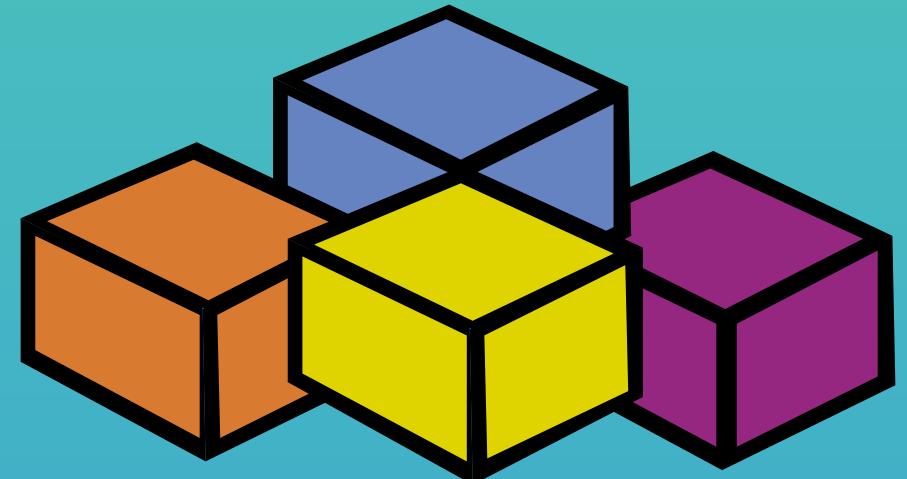
What is Docker?

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



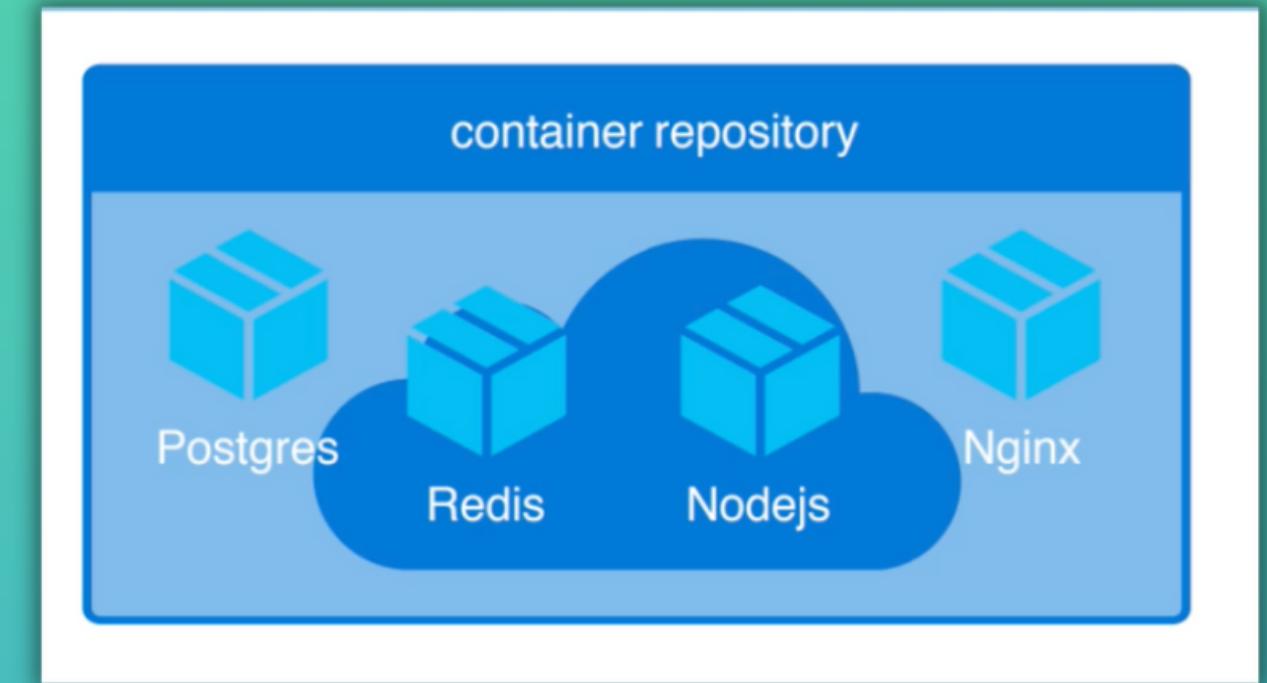
What is a container?

- 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**



Where container artifacts are hosted

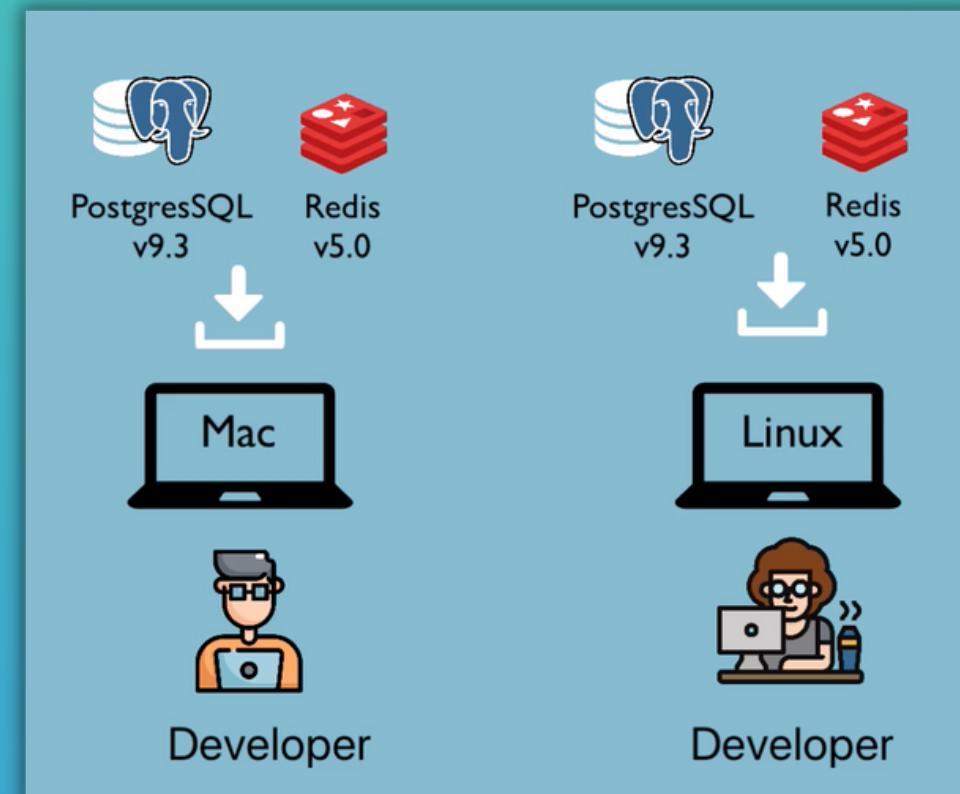
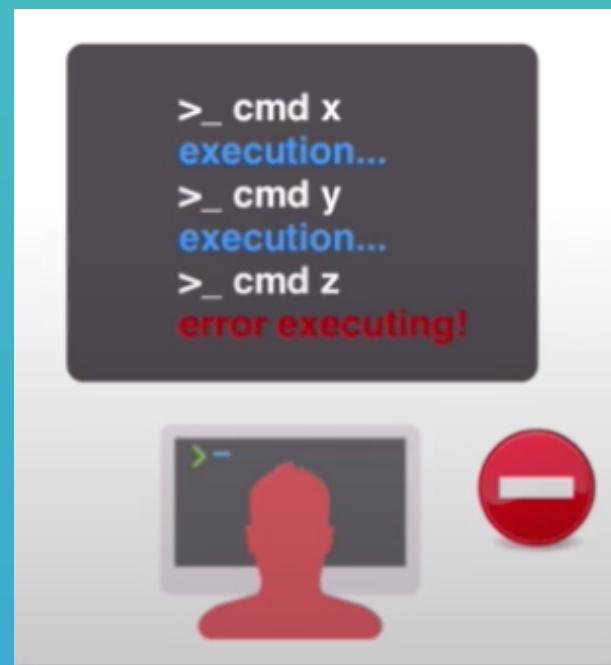
- Hosted in **container repositories**
- There are private and public repositories
- Public repository for Docker:



Application DEVELOPMENT before and after Docker

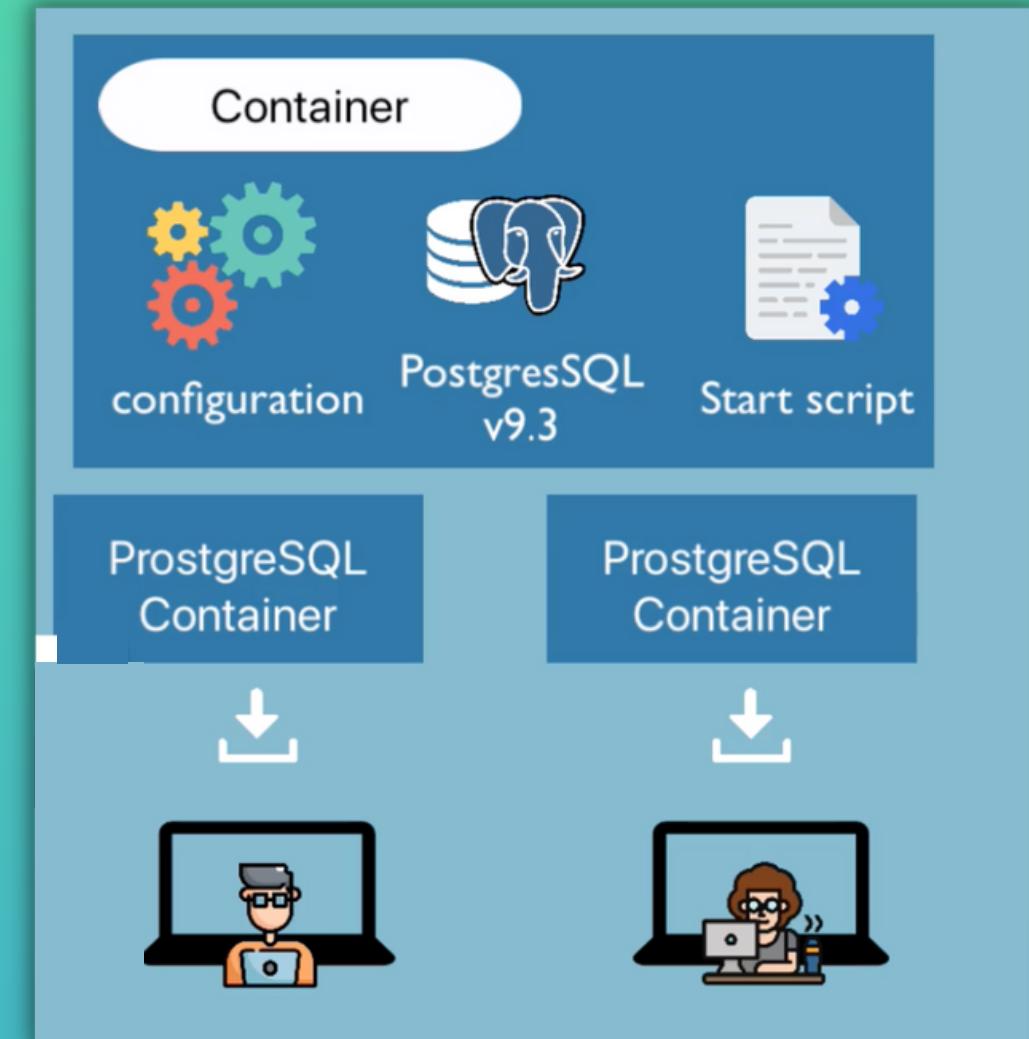
Before Containers

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

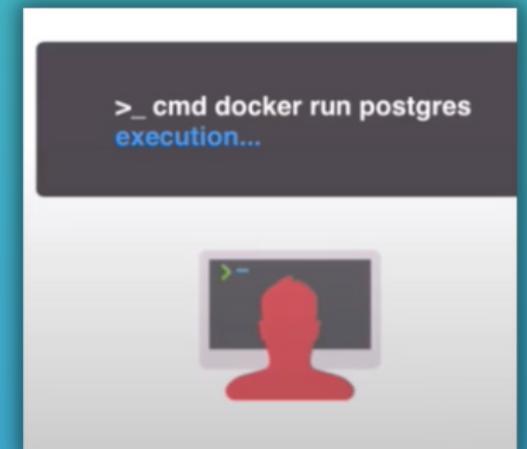


After Containers

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



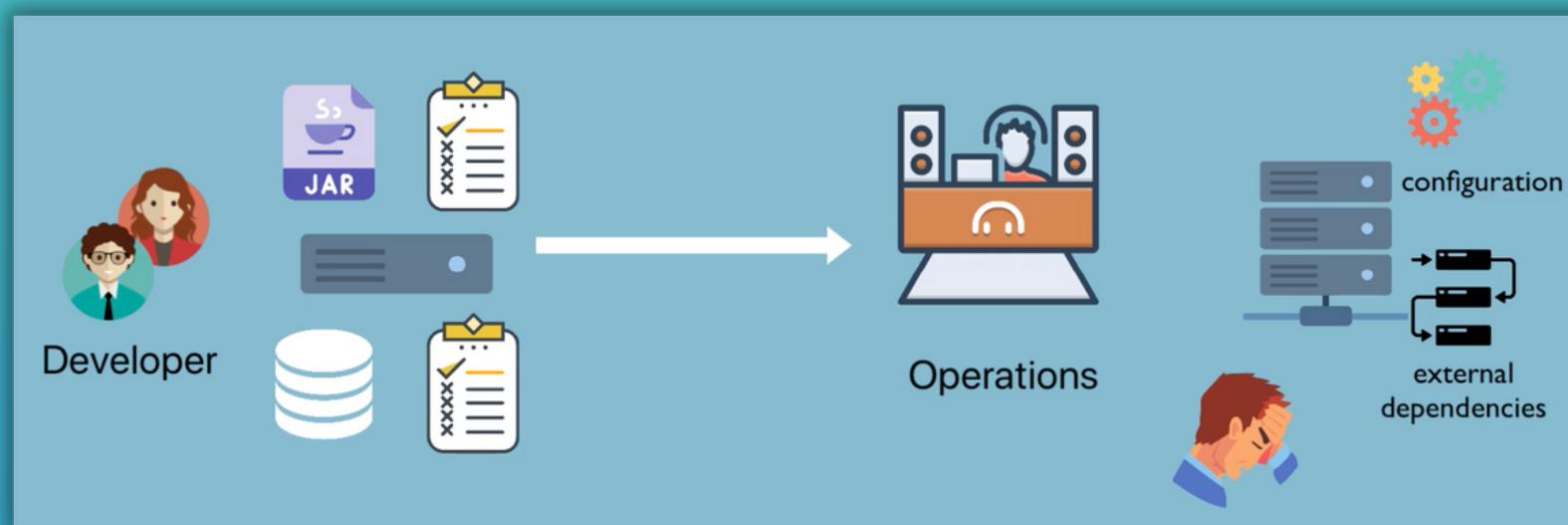
- 1 **command** to install the application
- Easily run same application with 2 different versions



Application DEPLOYMENT before and after Docker

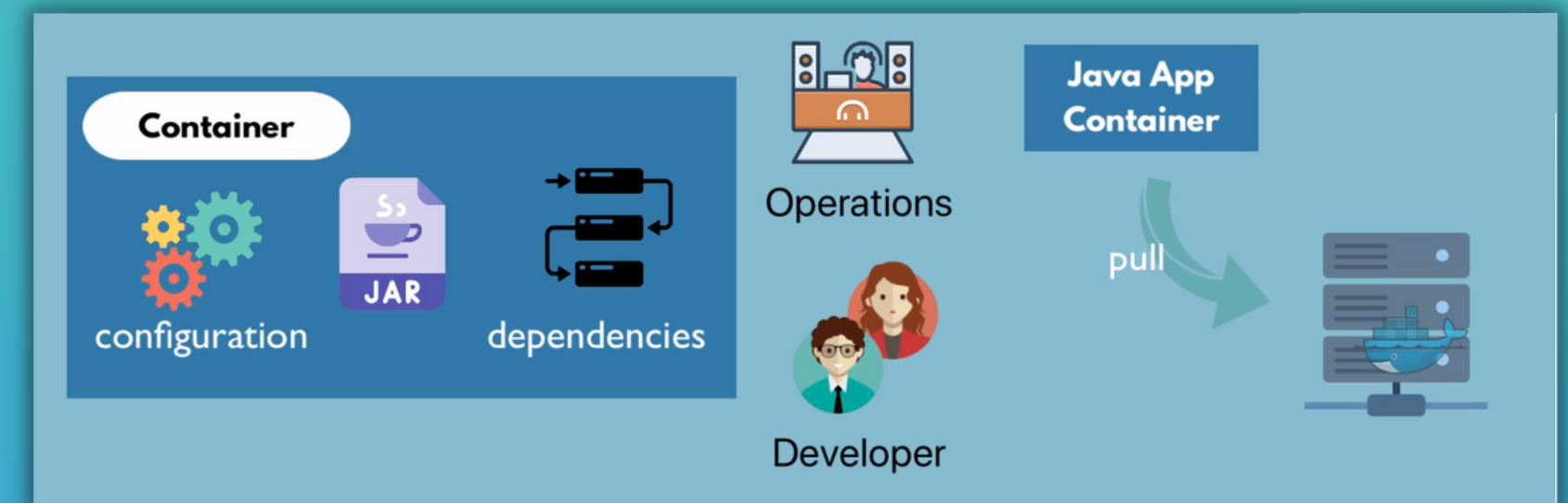
Before Containers

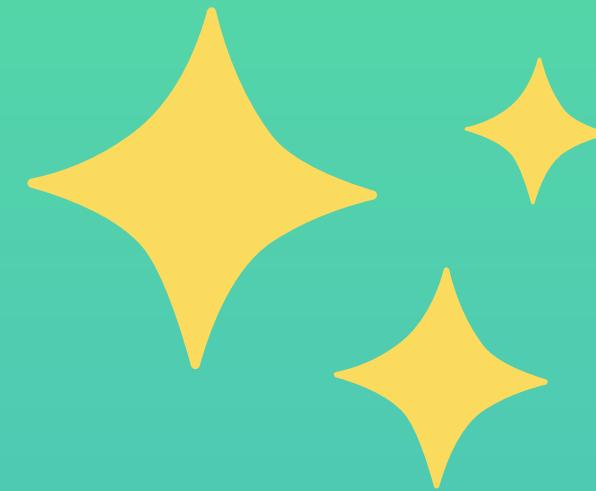
- Configuration on the server needed
- ✗ Dependency version conflicts
- Textual guide for deployment
- ✗ Misunderstanding



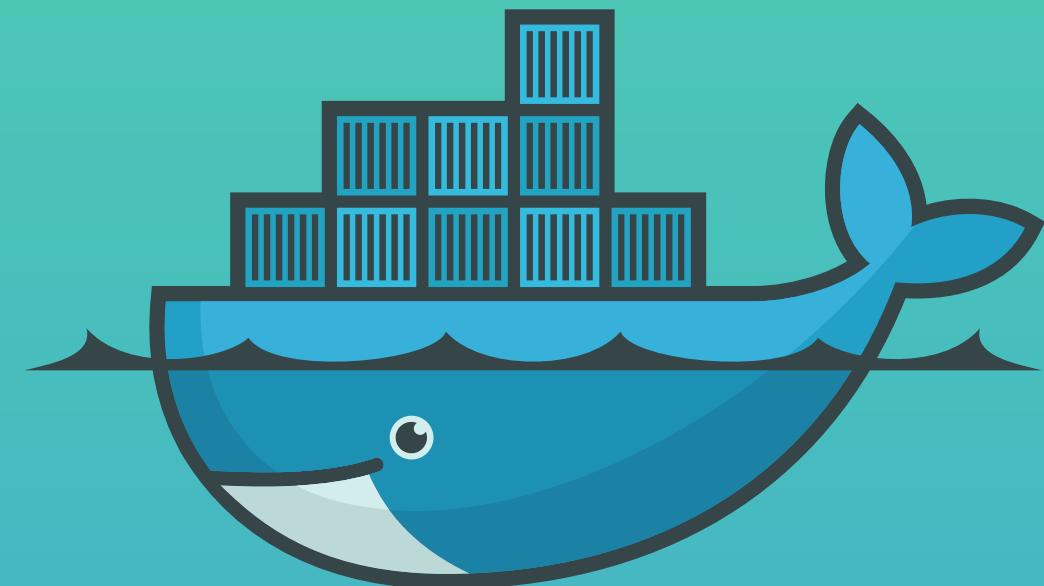
After Containers

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





Docker = most popular container technology



docker

Others:

containerd



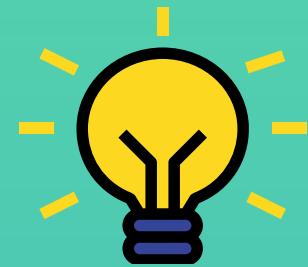
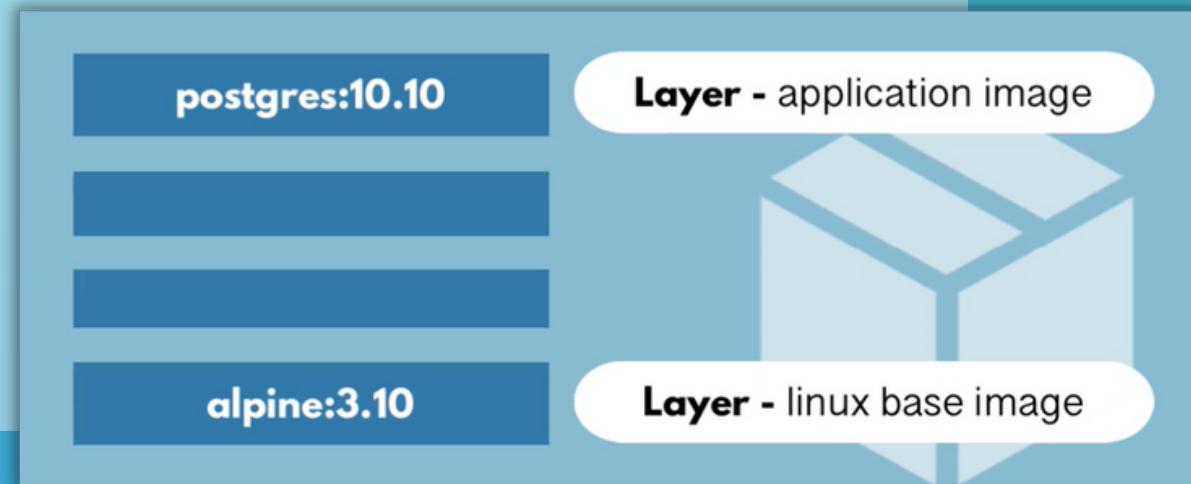
cri-o

Docker Image vs Docker Container

Docker Image

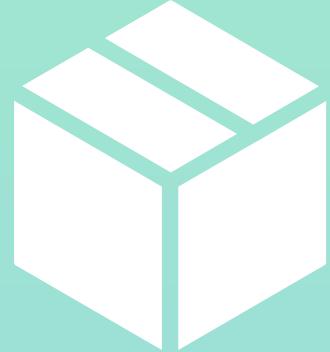


- The actual package (file)
- **Artifact** that can be moved around
- Not in "running" state
- Consists of several layers
- Mostly Linux Base Image, application image on top

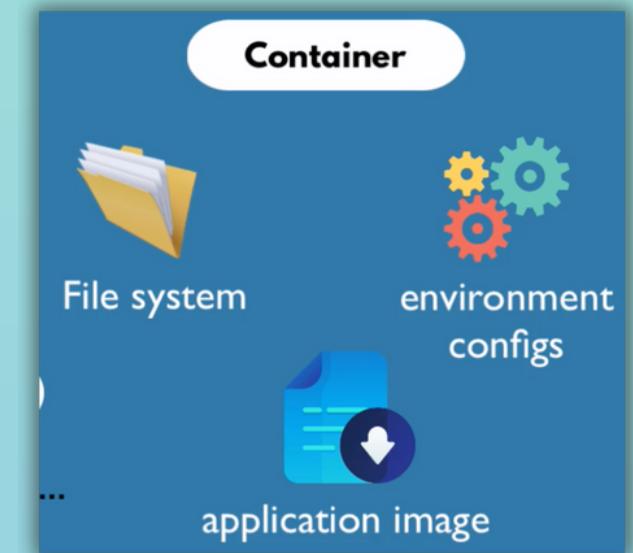


Images become
containers at runtime

Docker Container



- Actually **starts the application**
- Is a **running environment** defined in the image

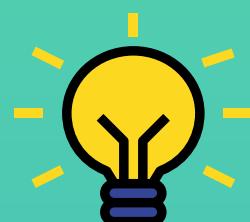
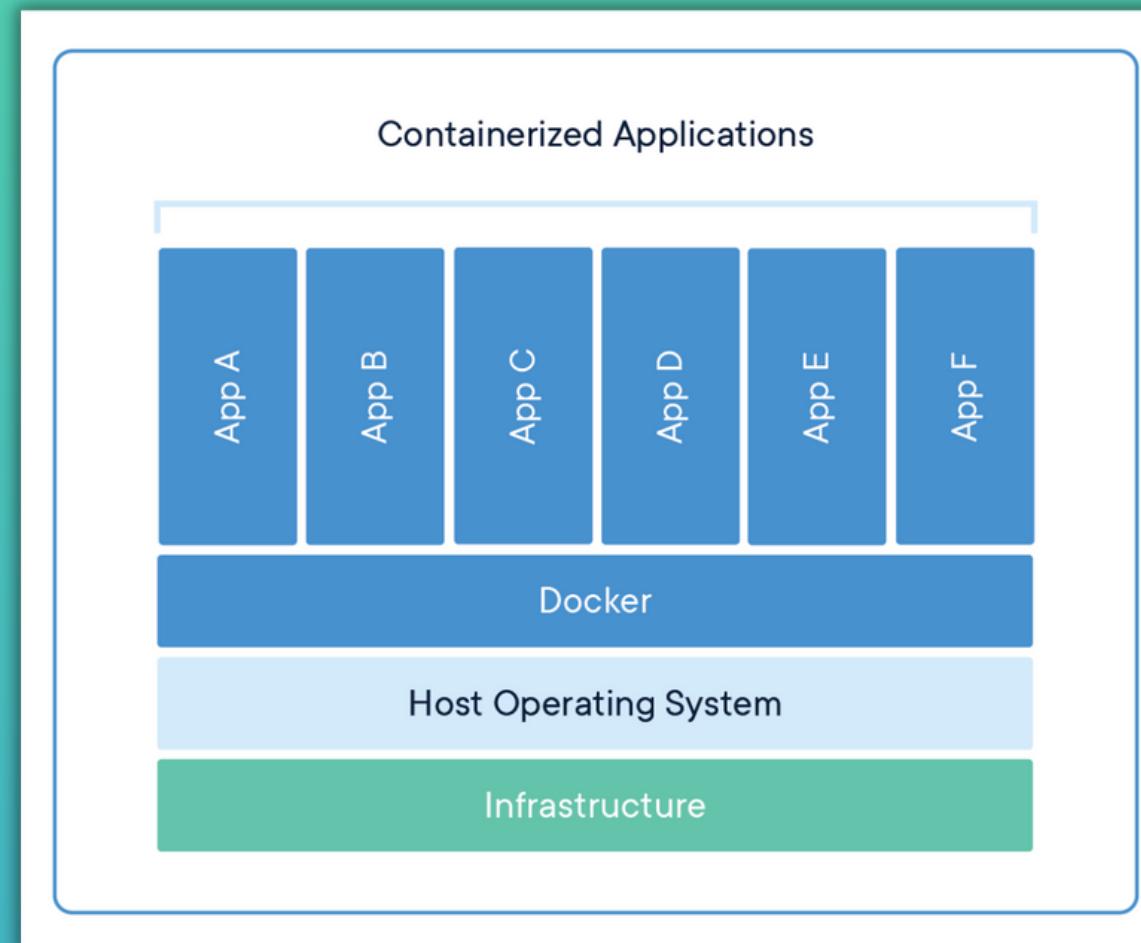


- Virtual file system
- Port binding: talk to application running inside of container

Containers

vs

Virtual Machines



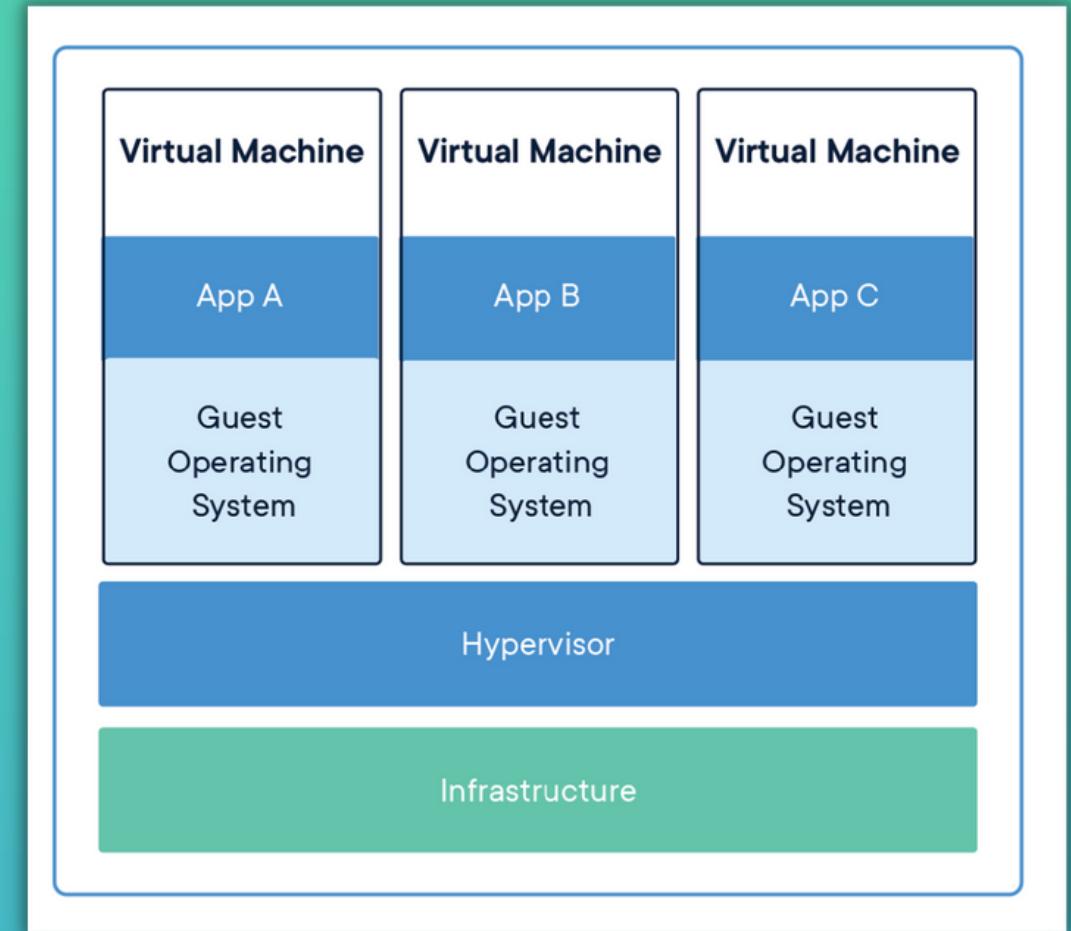
Size: Docker Image
much smaller



Speed: Docker
containers start and
run much faster



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

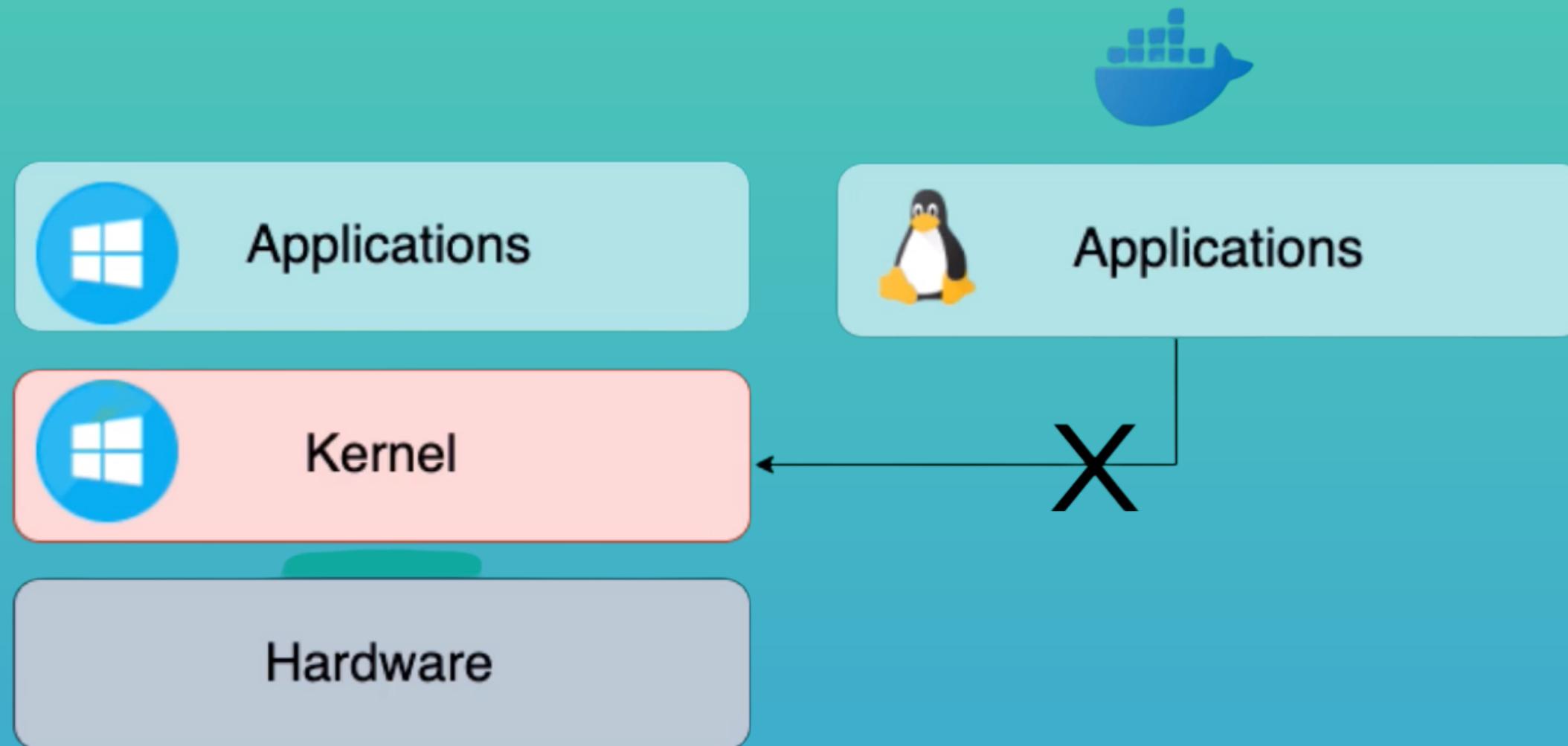


- Abstraction at the app layer
- Multiple containers share the OS kernel

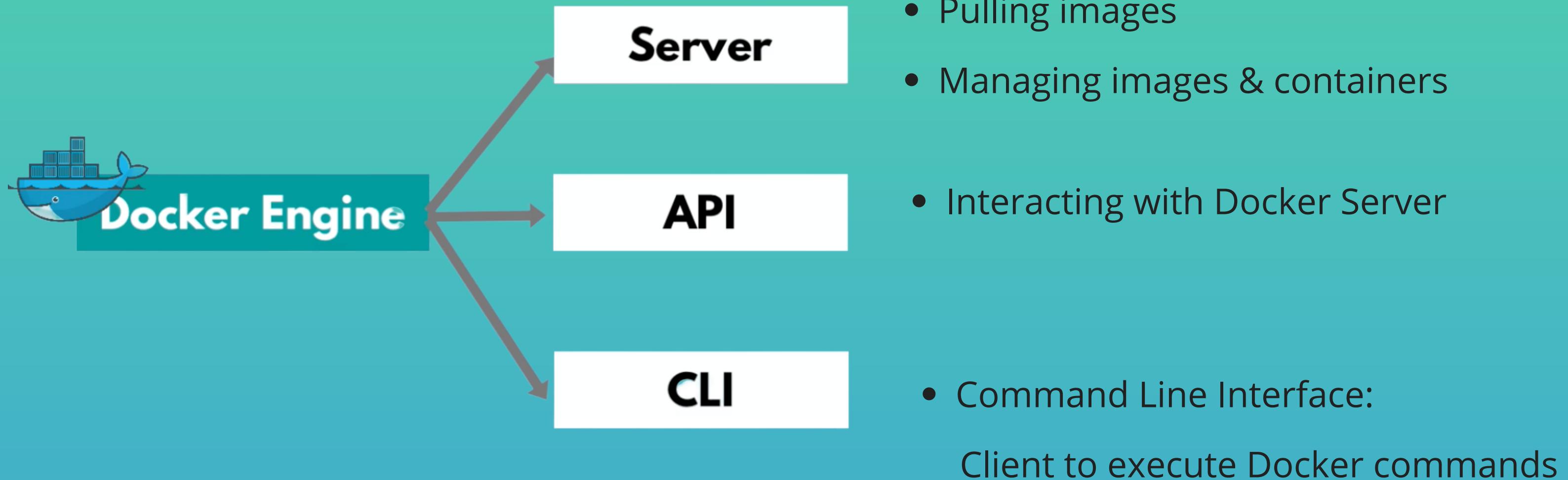
- Abstraction of physical hardware
- Each VM includes a full copy of an OS

Both are **virtualization** tools

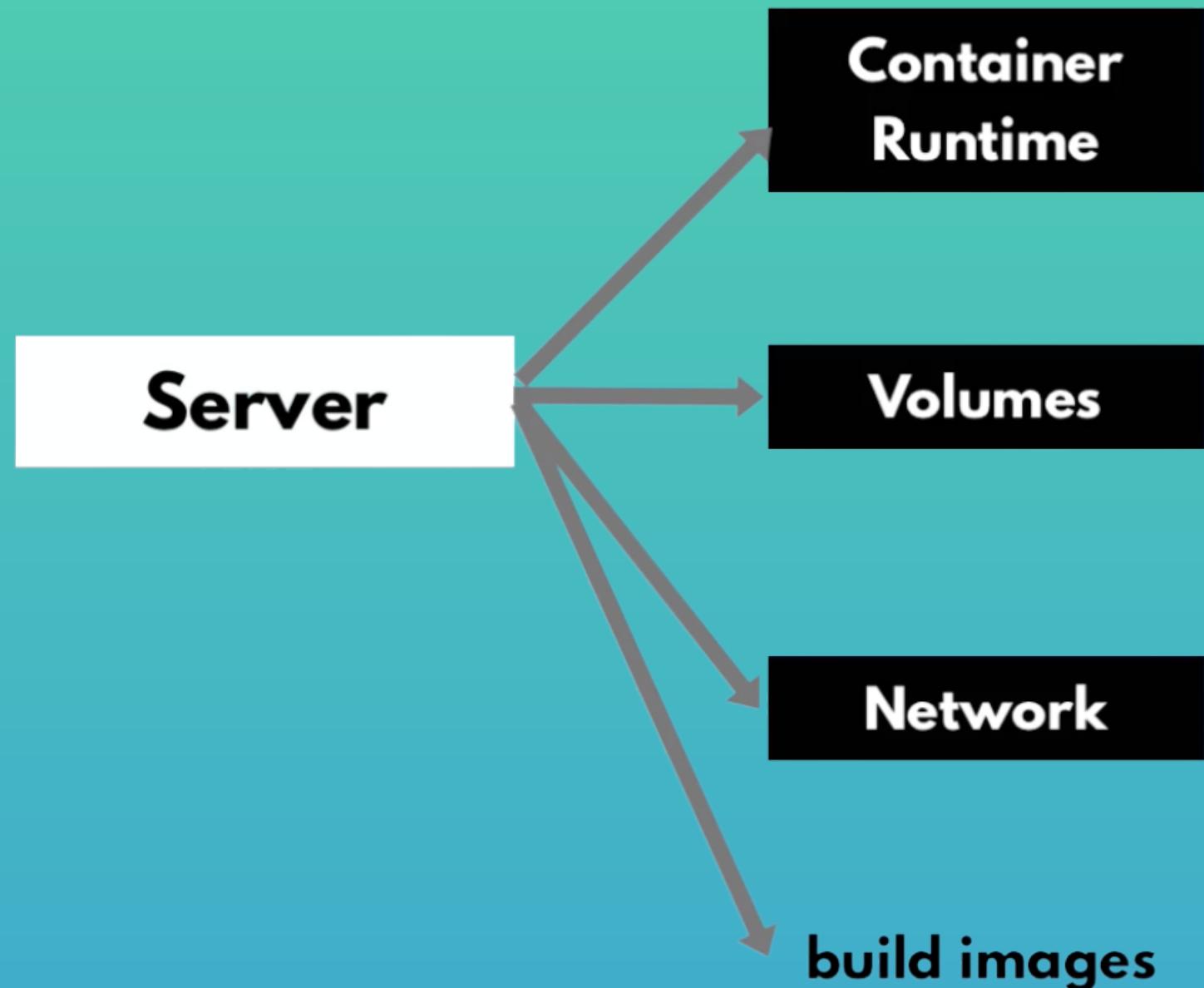
You can't run Linux container on Windows host, but for that there is
Docker Desktop for Windows and Mac



Docker Architecture & its components - 1

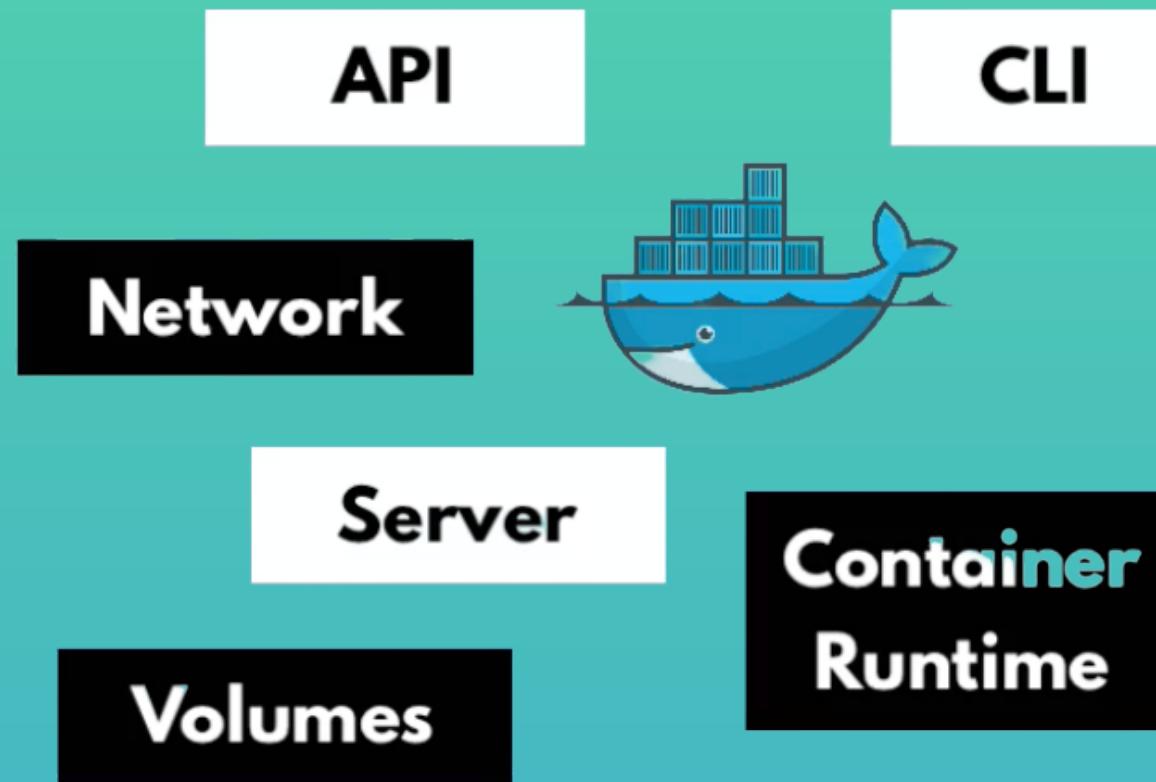


Docker Architecture & its components - 2



- 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`



Need to build an image?



`buildah`

Deep Dive into Docker

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 your infrastructure so you can manage your infrastructure in the same way you manage your applications. By using Docker for shipping, testing, and deploying code quickly, you can significantly reduce the delay between writing code and running it in production.

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.



TECHWORLD
WITH NANA

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

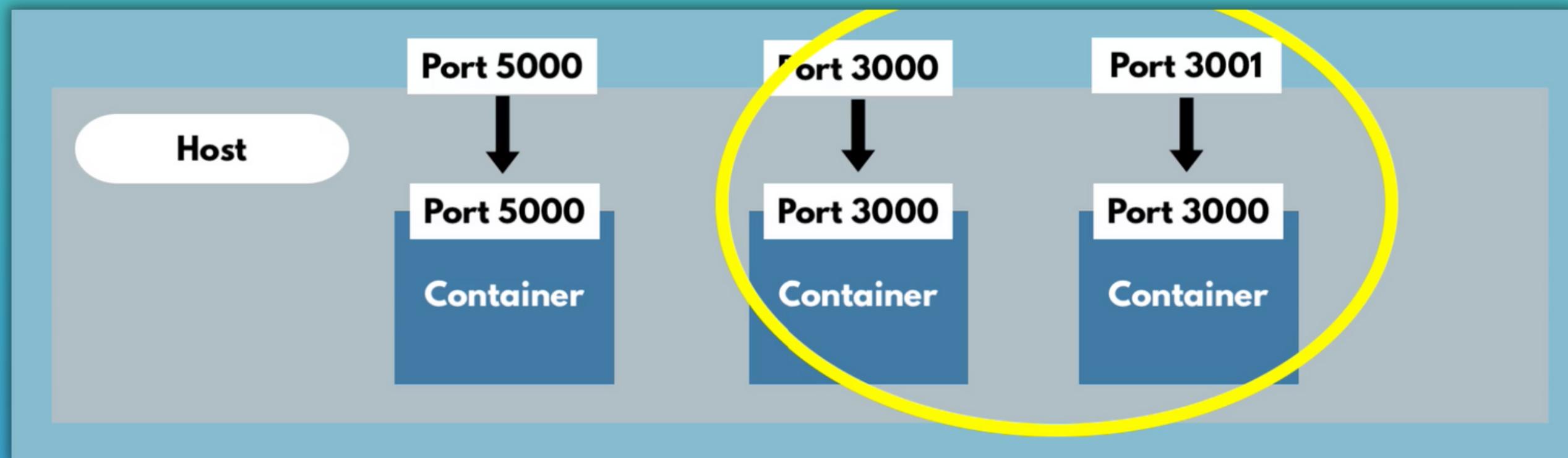
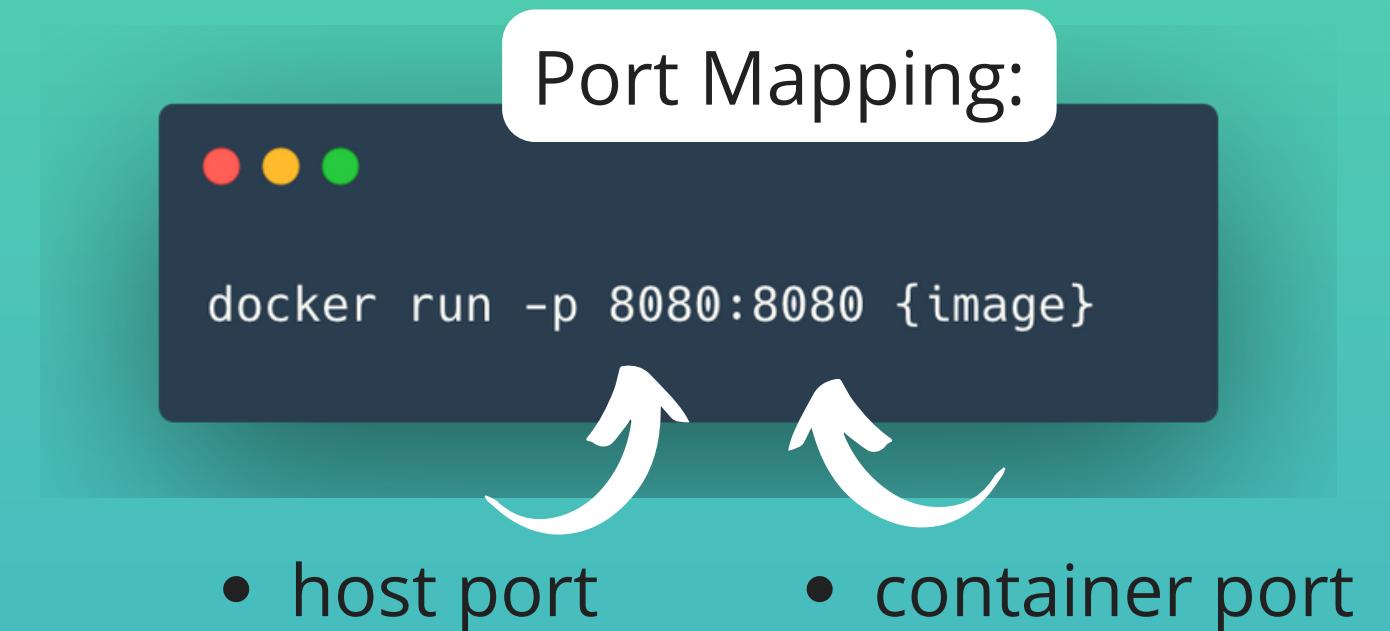
1. **docker logs**: fetch logs of a container
2. **docker exec -it**: creates a new bash session in the container

Ports in Docker - 1

- Multiple containers can run on your host machine

Problem:

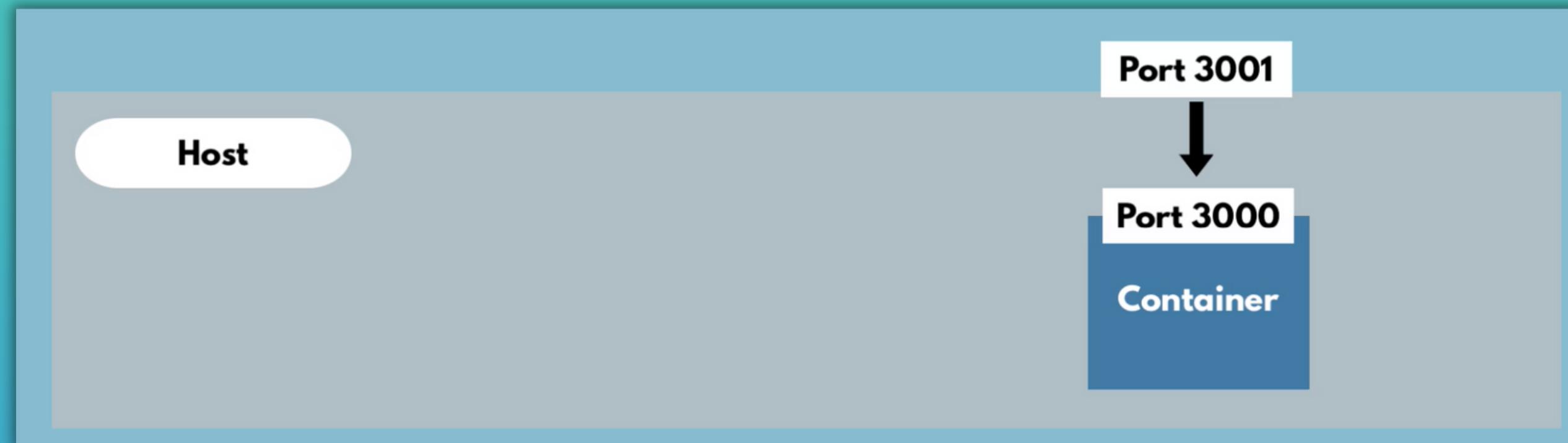
- But your laptop has only certain ports available
- Conflict when same port on host machine, so we need to map to a free port on host machine:



Ports in Docker - 2

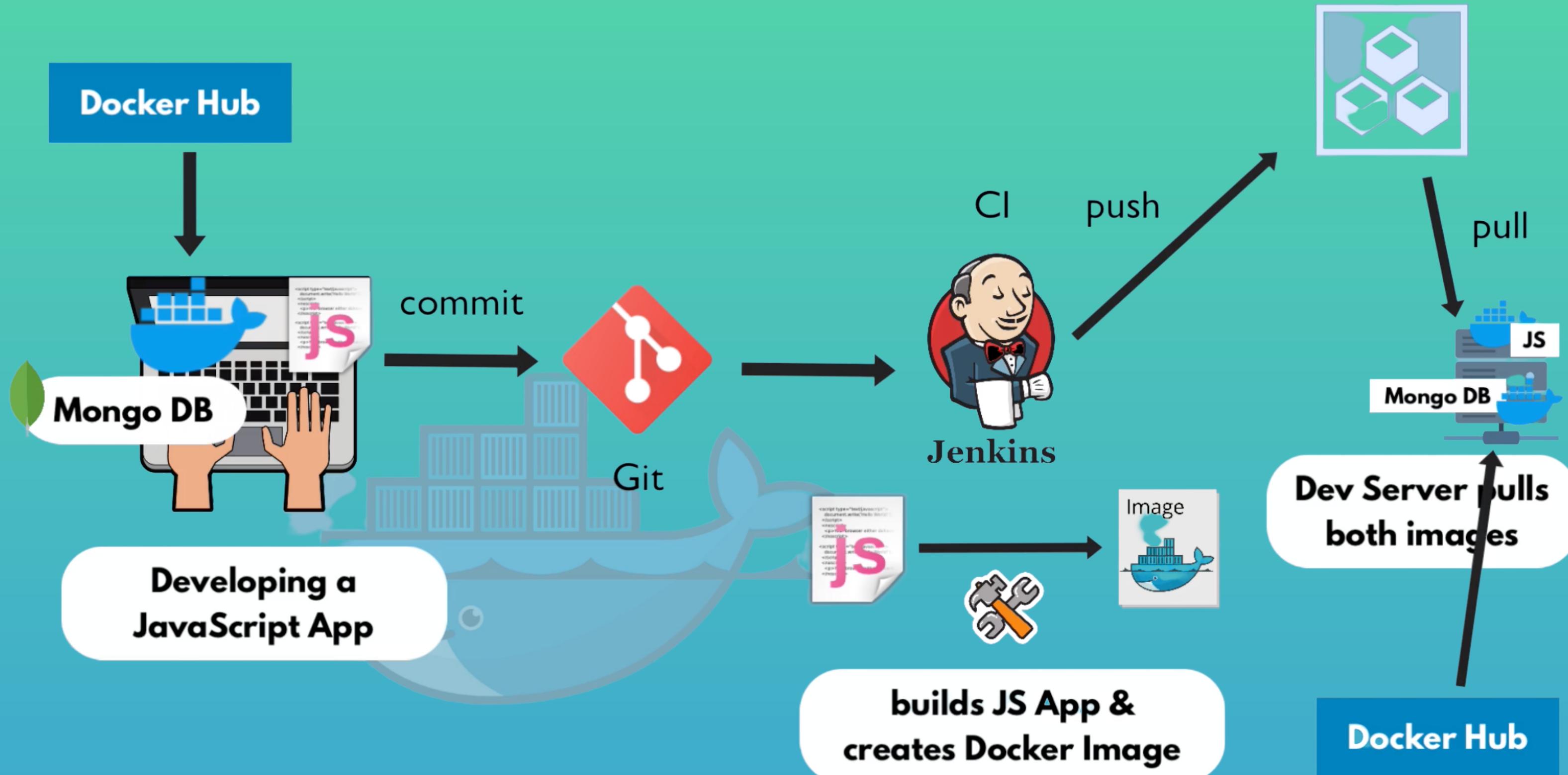
- **Container Port** = Port used in container
- **Host Port** = Port on the host machine

some-app://localhost:3001



Workflow with Docker

Docker Repository



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 in it (*--net* option in docker run)

Docker Compose - 2

Example docker-compose.yaml file

```
1  version: '3'  
2  services:  
3    mongodb:  
4      image: mongo  
5      ports:  
6        - 27017:27017  
7      environment:  
8        - MONGO_INITDB_ROOT_USERNAME=admin  
9        - MONGO_INITDB_ROOT_PASSWORD=password  
10    mongo-express:  
11      image: mongo-express  
12      ports:  
13        - 8080:8081  
14      environment:  
15        - ME_CONFIG_MONGODB_ADMINUSERNAME=admin  
16        - ME_CONFIG_MONGODB_ADMINPASSWORD=password  
17        - ME_CONFIG_MONGODB_SERVER=mongodb
```

Dockerfile - 1

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

- Some common Dockerfile commands:

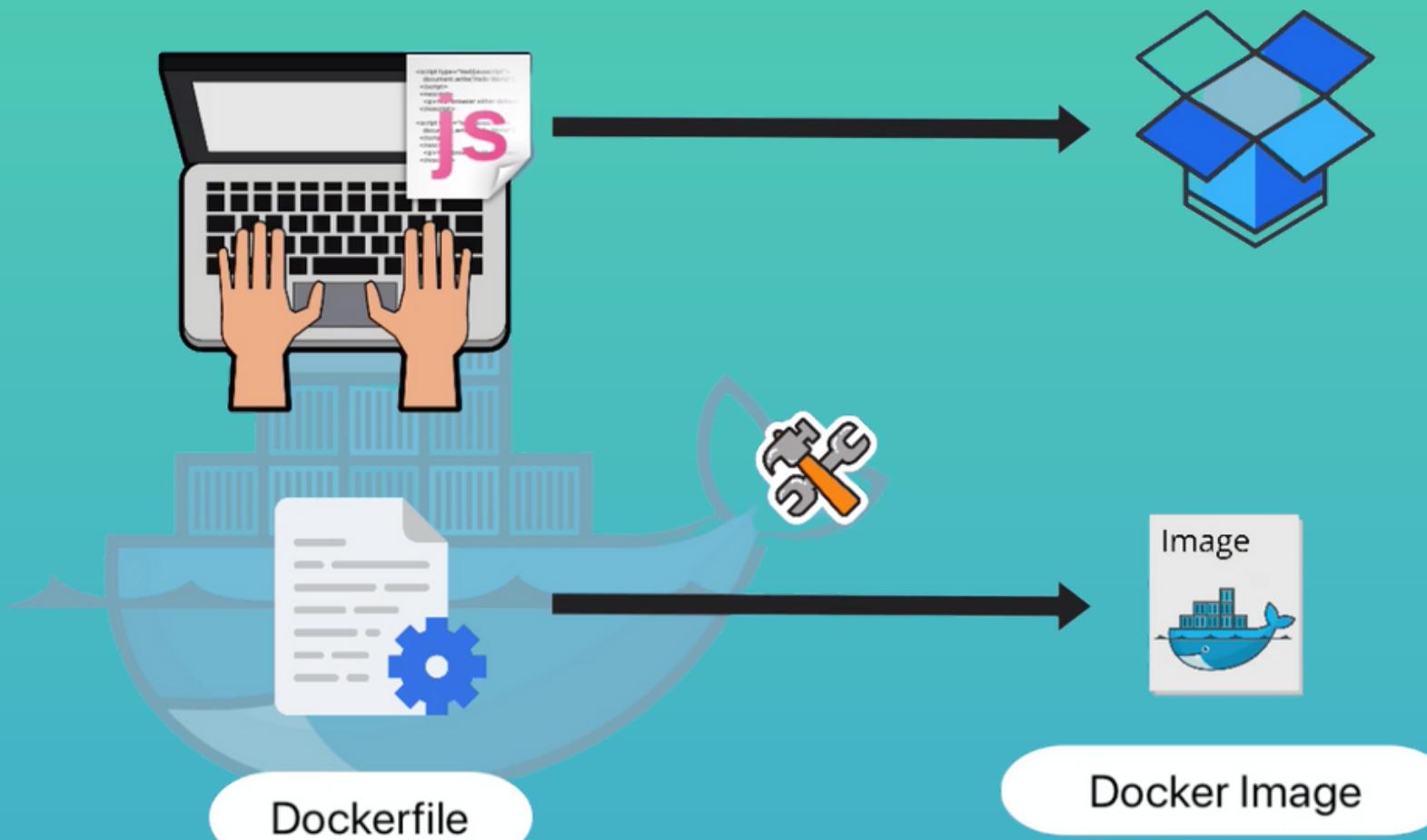


Image Environment Blueprint	DOCKERFILE
install node	FROM node
set MONGO_DB_USERNAME=admin	ENV MONGO_DB_USERNAME=admin \
set MONGO_DB_PWD=password	MONGO_DB_PWD=password
create /home/app folder	RUN mkdir -p /home/app
copy current folder files to /home/app	COPY . /home/app
start the app with: "node server.js"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

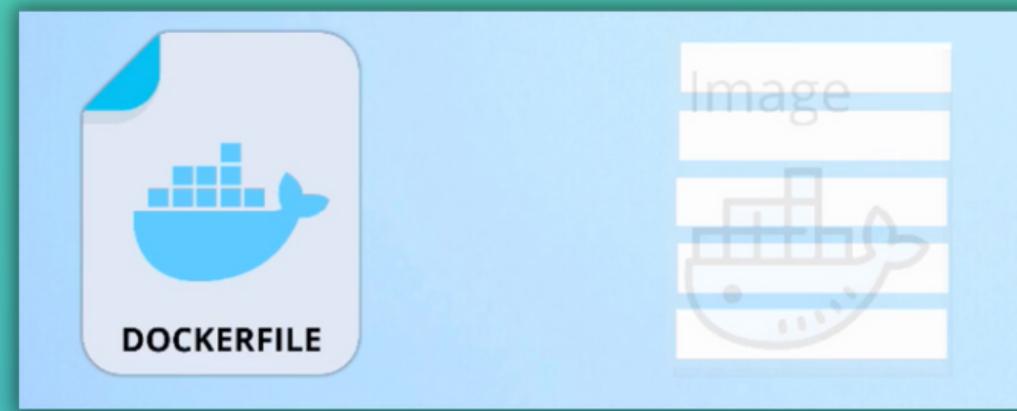
CMD = entrypoint command

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

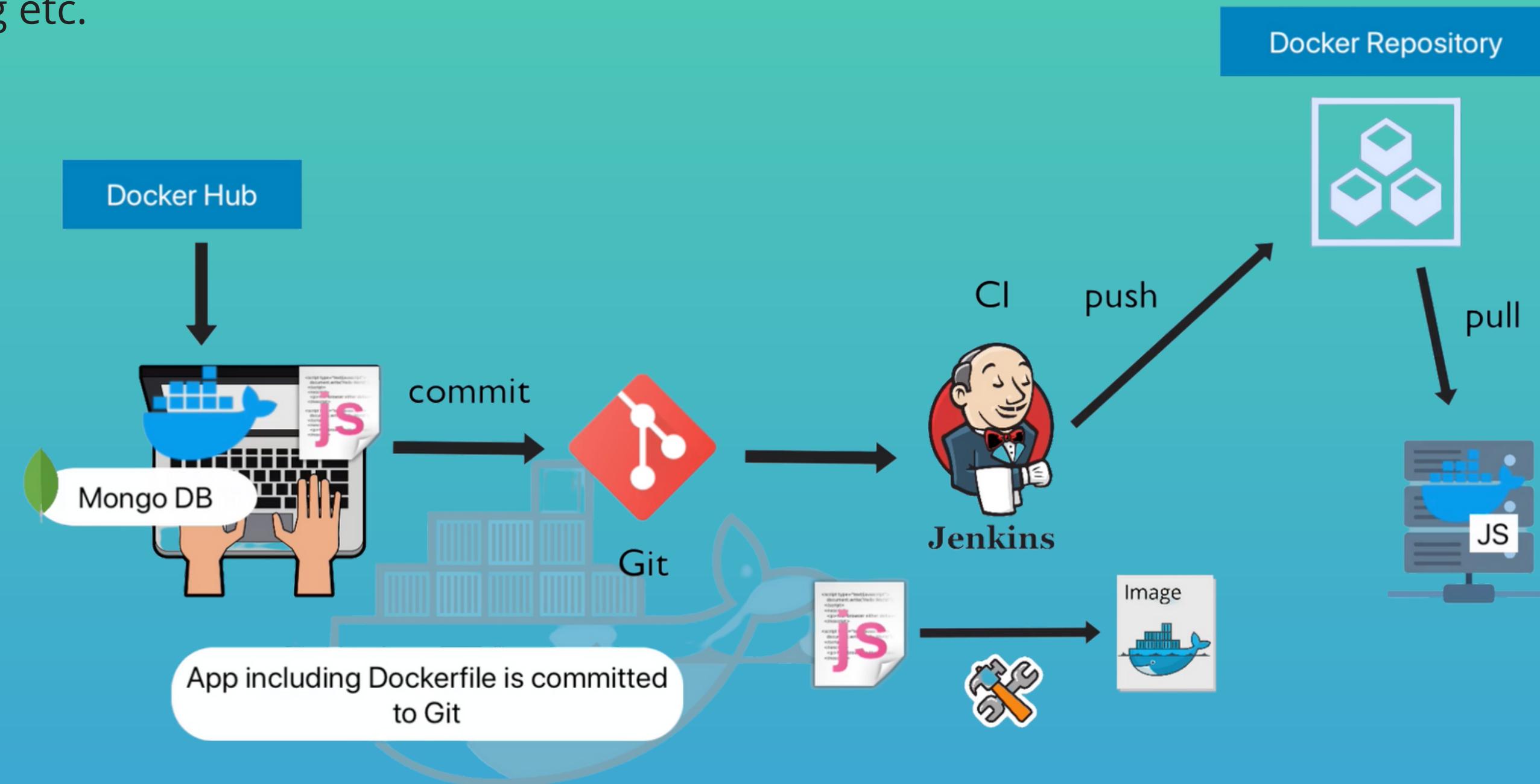


A dark blue rounded rectangle representing a terminal window. At the top, there are three small colored circles (red, yellow, green) representing a progress bar. Below them, the command 'docker build -t my-app:1.0 .' is displayed in white text.

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.



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

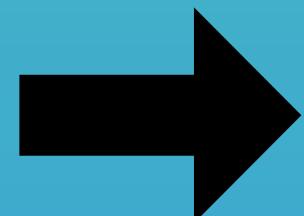


```
docker login
```

```
- 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

The screenshot illustrates the process of publishing a Docker image to a private registry (AWS ECR). On the left, a terminal window shows the command:

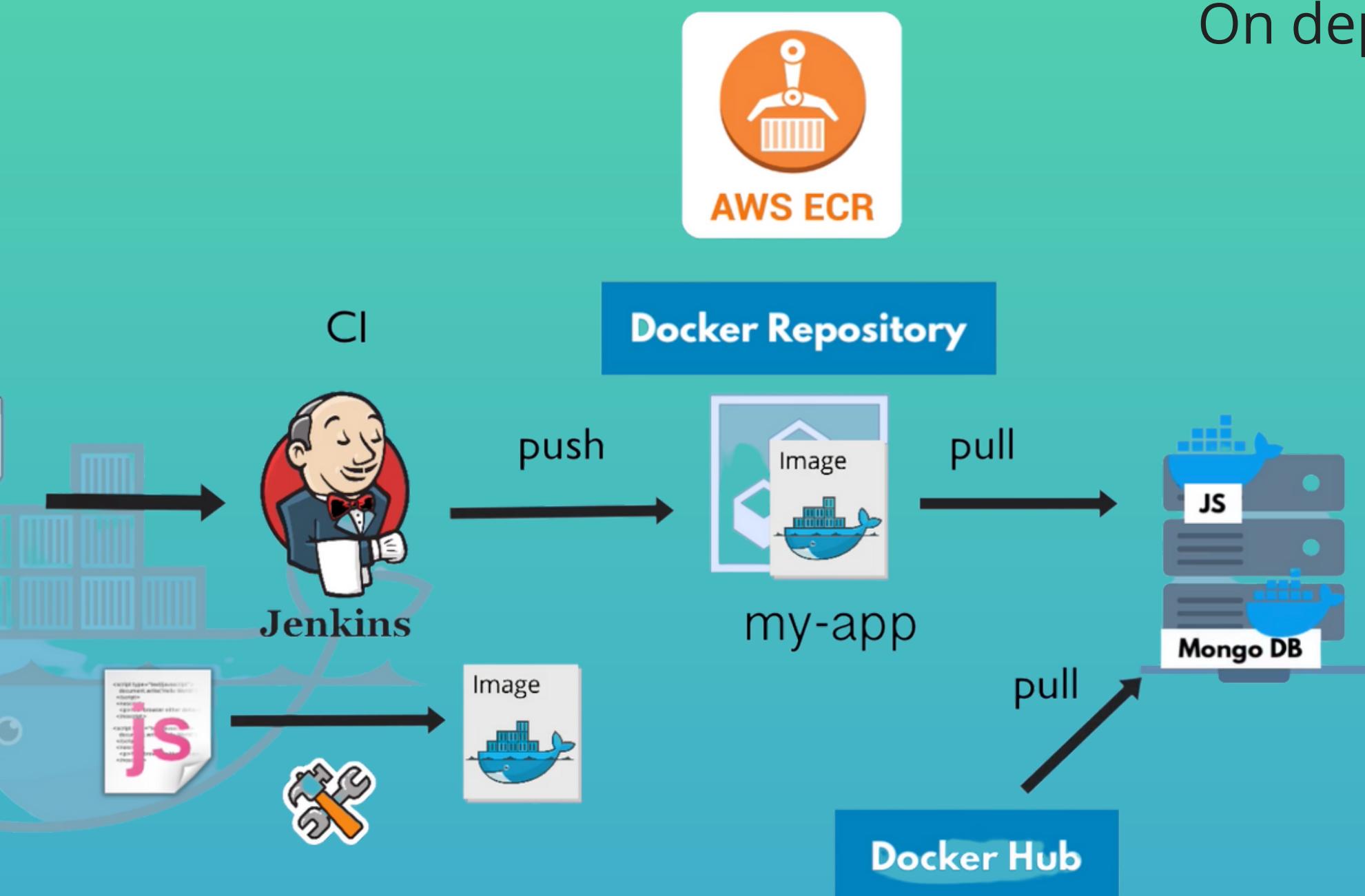
```
[~]$ docker tag my-app:1.0 664574038682.dkr.ecr.eu-central-1.amazonaws.com/my-app:1.0
```

On the right, the AWS ECR console displays the repository 'my-app'. The 'Images' section shows one image entry:

Image tag	Image URI	Pushed at	Digest	Size (MB)
1.0	664574038682.dkr.ecr.eu-central-1.amazonaws.com/my-app:1.0	11/11/19, 10:44:55 AM	sha256:fc8aeac85...	44.66

A red circle highlights the image URI in the table, which matches the one specified in the terminal command.

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.



By default:

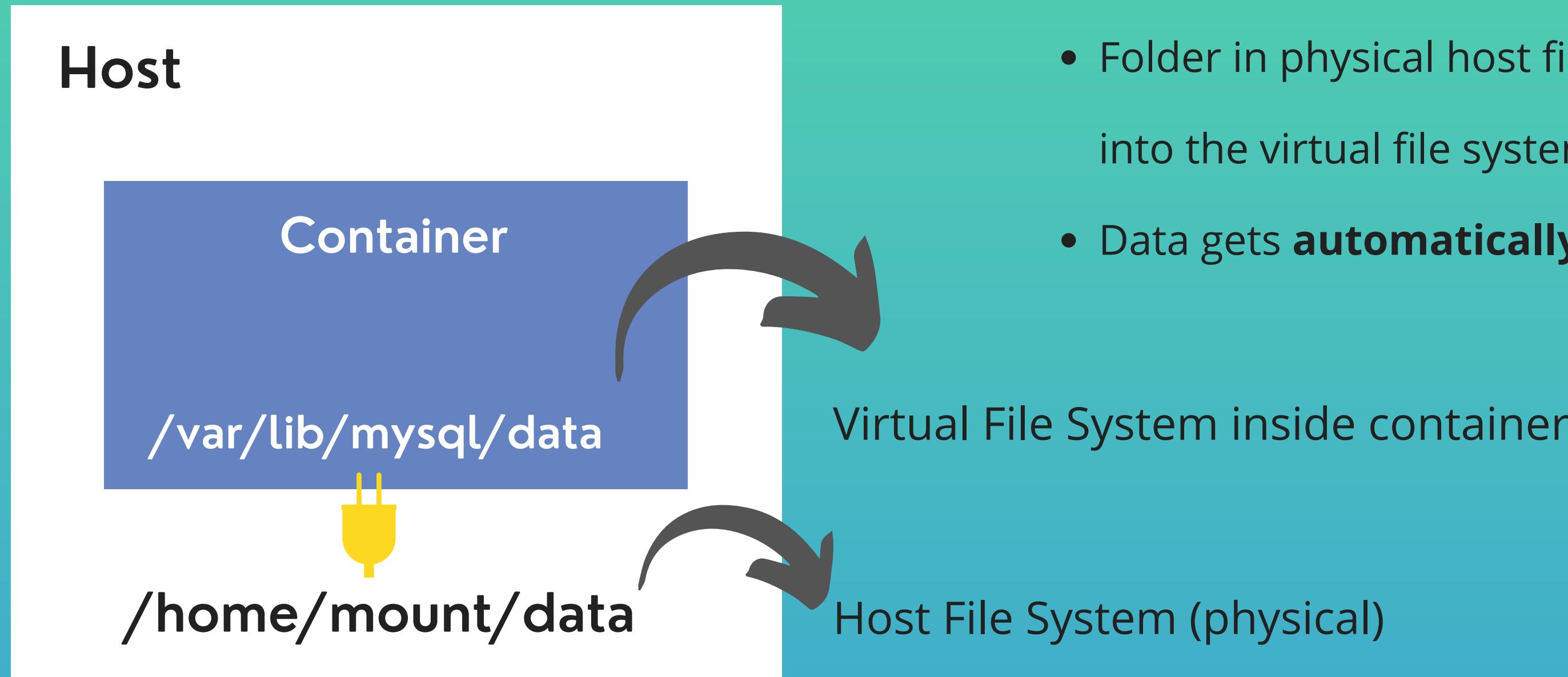
Data is gone when removing the container

Important for

- Databases
- Other stateful applications

Docker Volumes - 2

How Docker Volumes work?

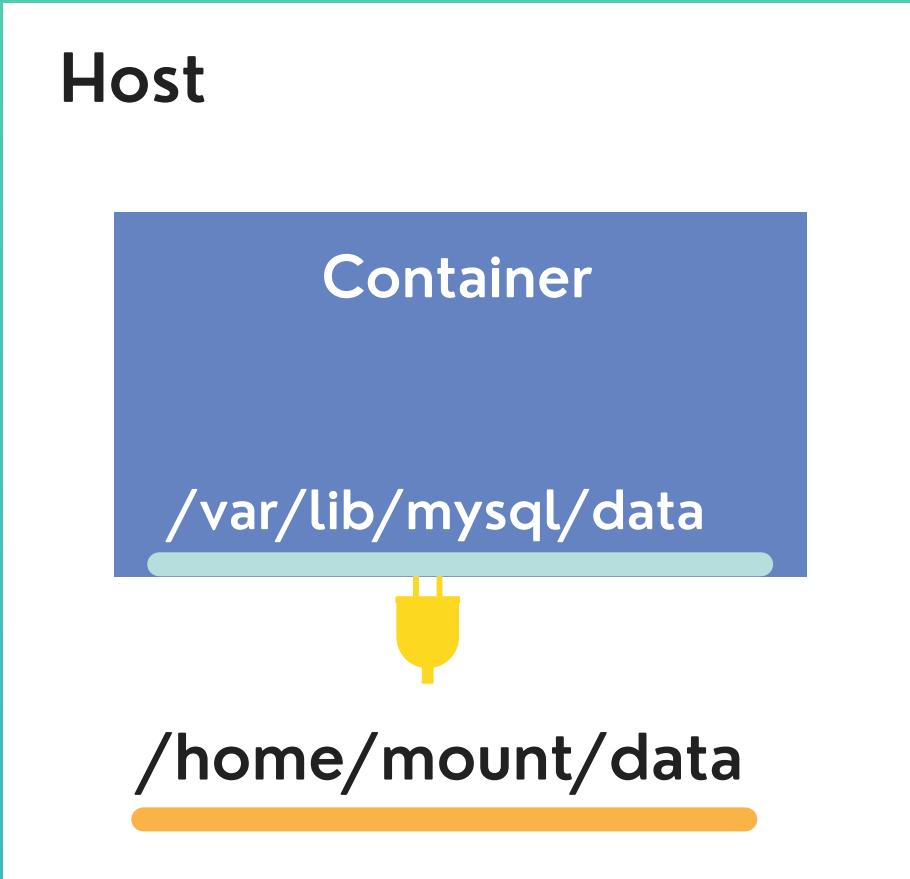


- Folder in physical host file system is **mounted** into the virtual file system of Docker
- Data gets **automatically replicated**

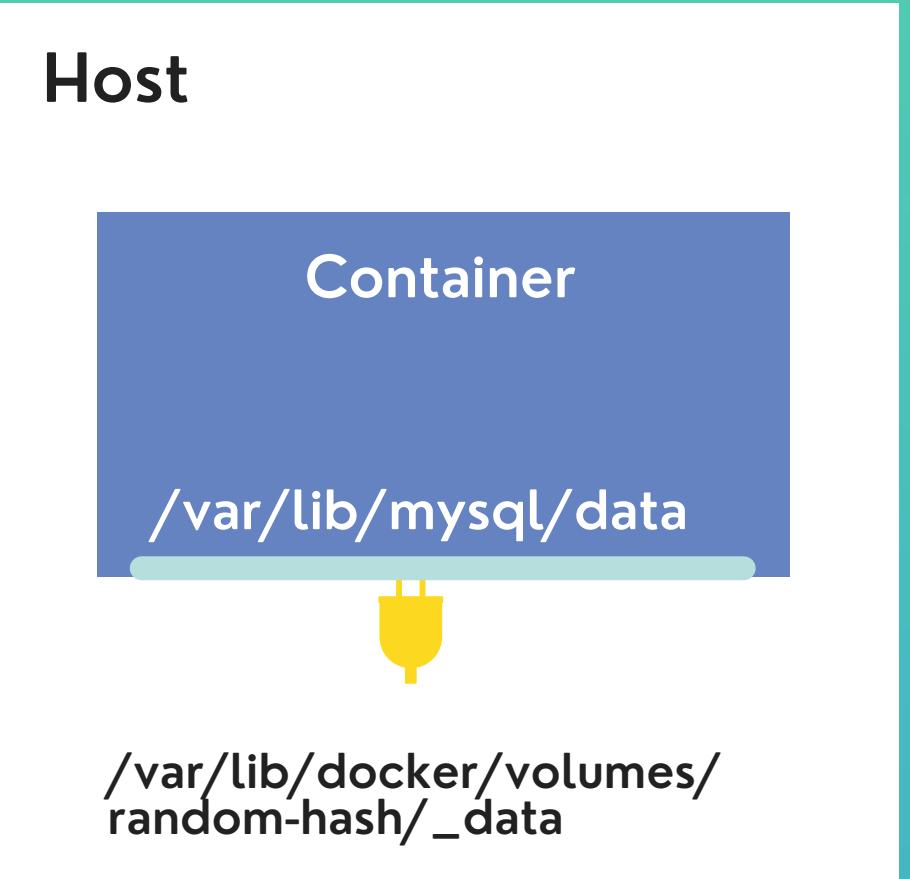
Docker Volumes - 3

3 Volume Types

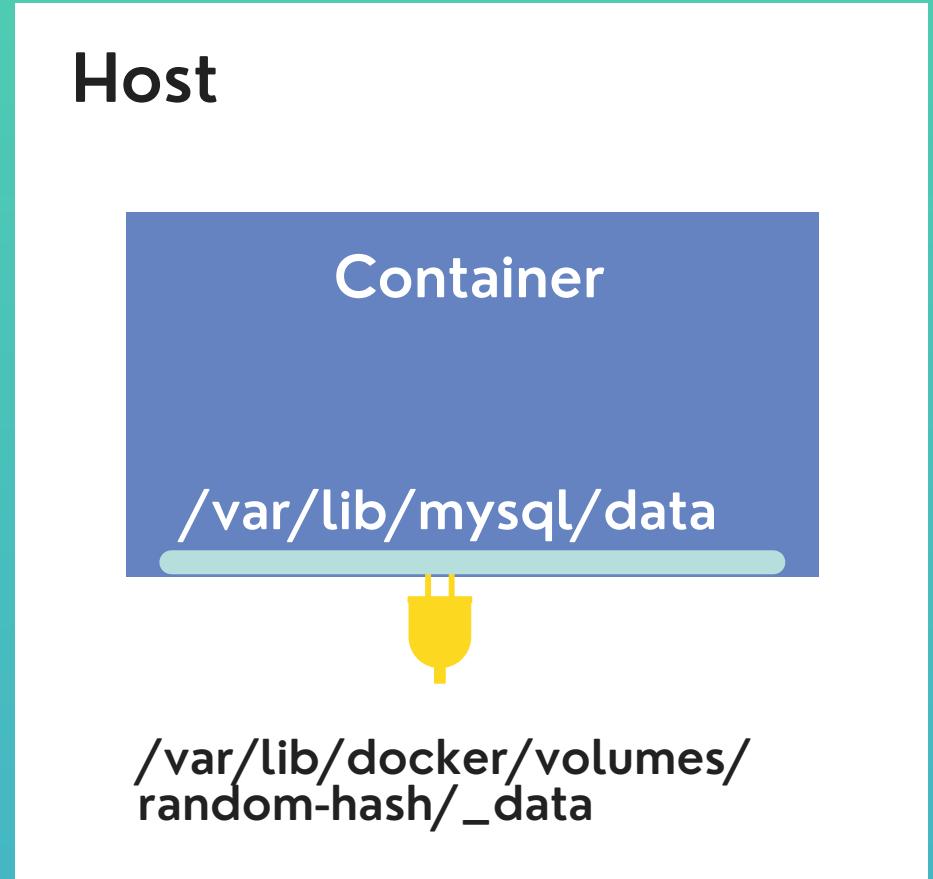
Host Volumes



Anonymous Volumes



Named Volumes



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

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

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

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

- 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

Named Volume

```
version: '3'

services:
  mongodb:
    image: mongo
    ports:
      - 27017:27017
    volumes:
      - db-data:/var/lib/mysql/data
  mongo-express:
    image: mongo-express
    ...
volumes:
  db-data
```

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



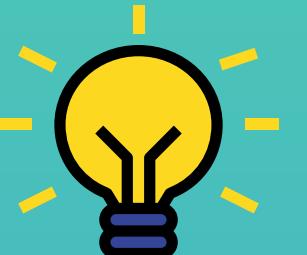
Optimize Image Size



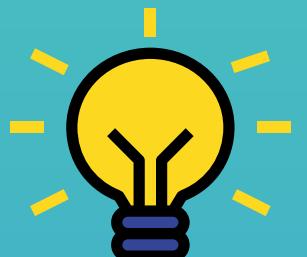
Write cleaner and more maintainable Dockerfiles

Docker Best Practices - 2

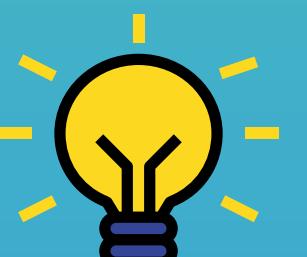
- **Security:** Use least privileged user (create a dedicated user and group with minimal permissions to run the application)
- **Security:** Scan your images for vulnerabilities
- **Security:** Don't leak sensitive information to Docker images



Improve Security



Optimize Image Size



Write cleaner and more maintainable Dockerfiles