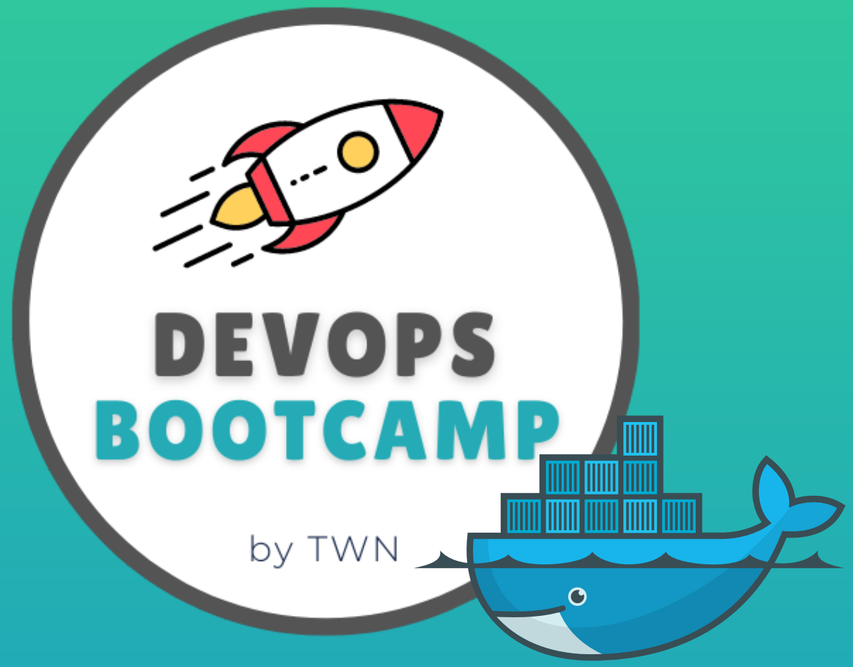


Containers with Docker

Key Takeaways

TECHWORLD WITH NANA

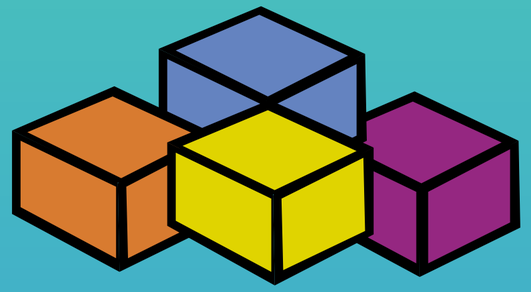
Introduction to Containers & Docker

TECHWORLD WITH NANA

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

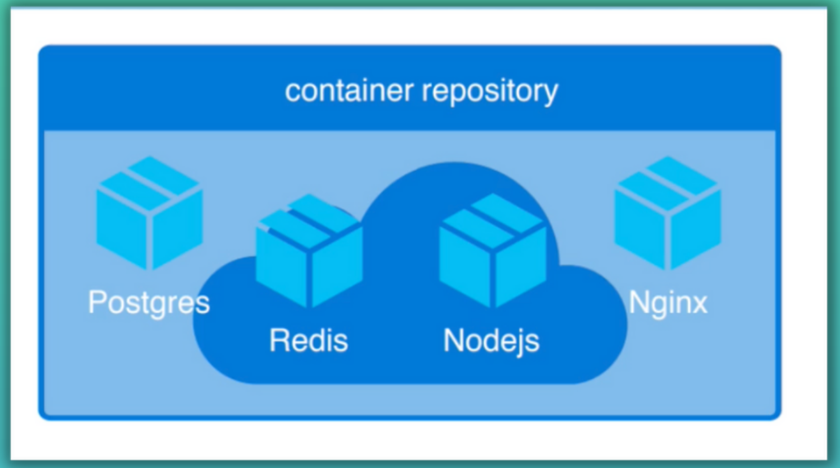
What is a container?



- A way to **package** an 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

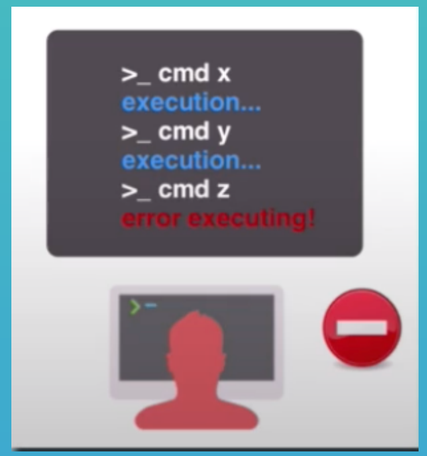
- Hosted in **container repositories**
- There are private and public repositories
- Public repository for Docker:

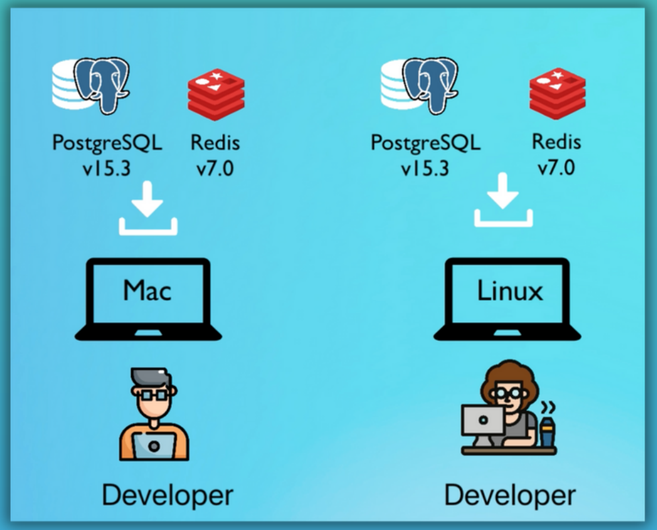


Application DEVELOPMENT before and after Docker

Before Containers

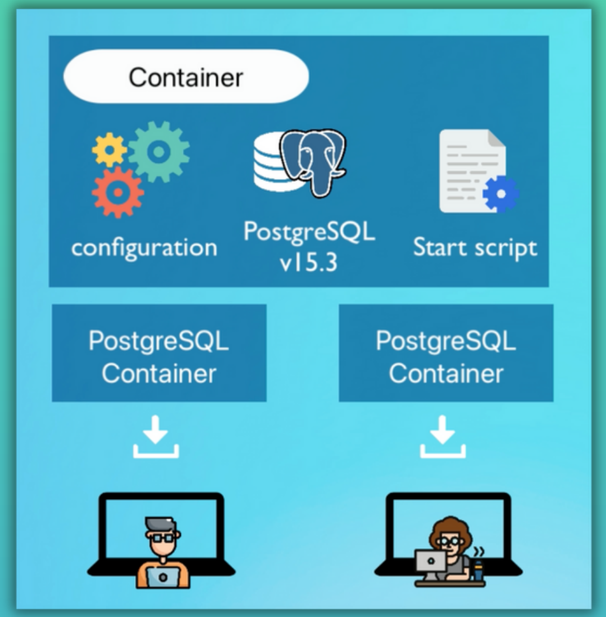
- **Installation process different** on each OS environment
- **Many steps** where something could go wrong



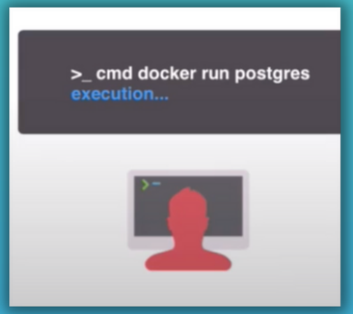


After Containers

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



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



TECHWORLD WITH NANA

Application DEPLOYMENT before and after Docker

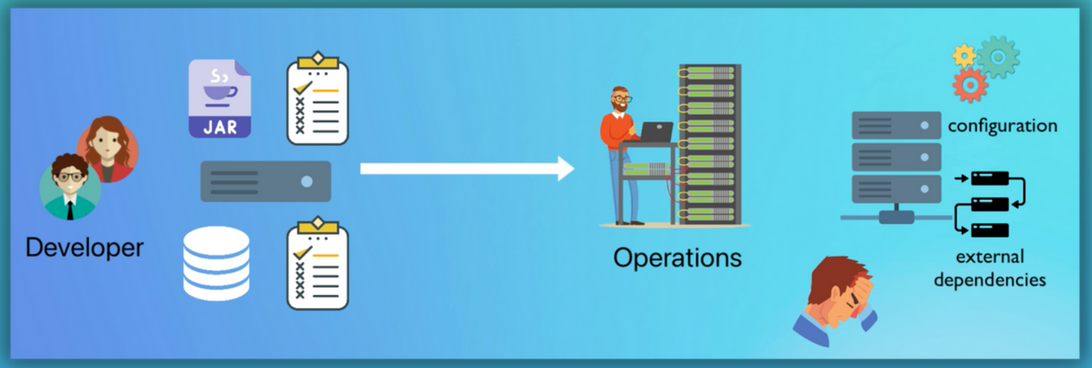
Before Containers

- Configuration on the server needed

❌ Dependency version conflicts

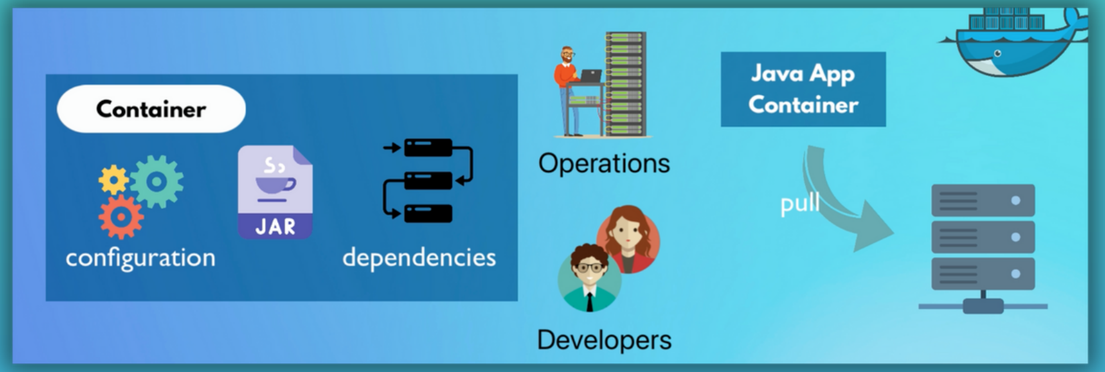
- Textual guide for deployment

❌ Misunderstandings



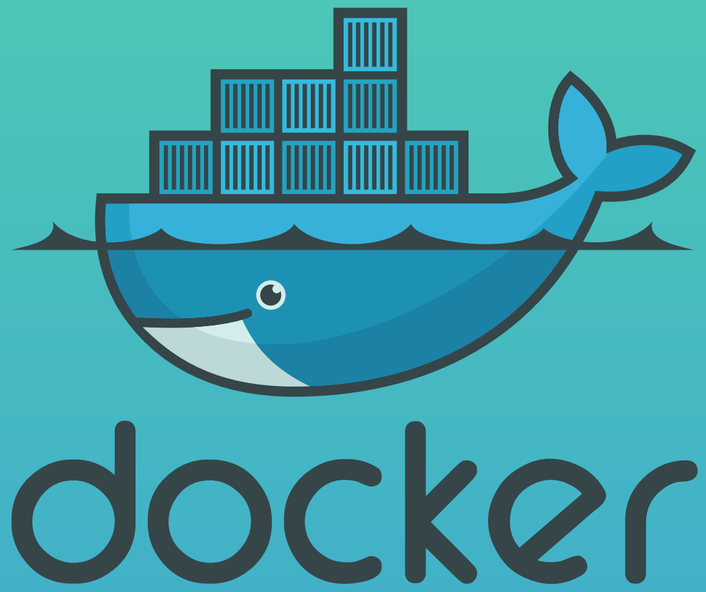
After Containers

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



TECHWORLD WITH NANA

Docker = most popular container technology



Others:

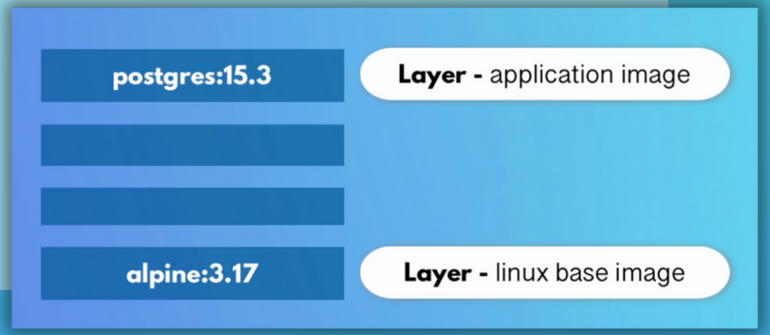
containerd

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

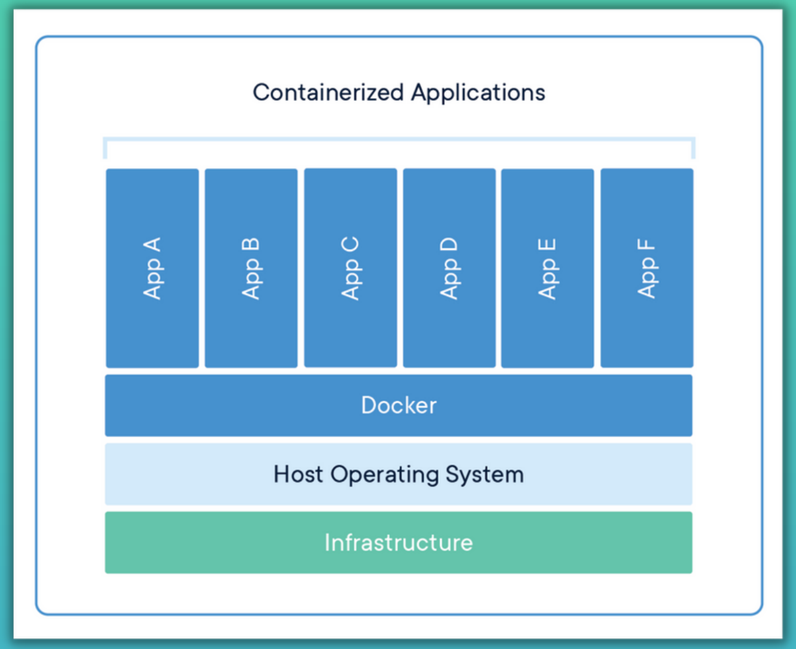


Images become containers at runtime

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

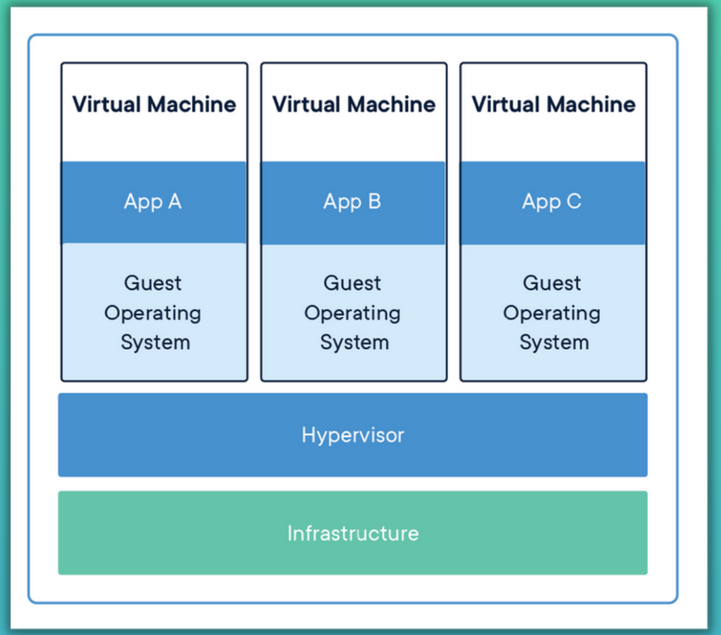
Containers vs Virtual Machines



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

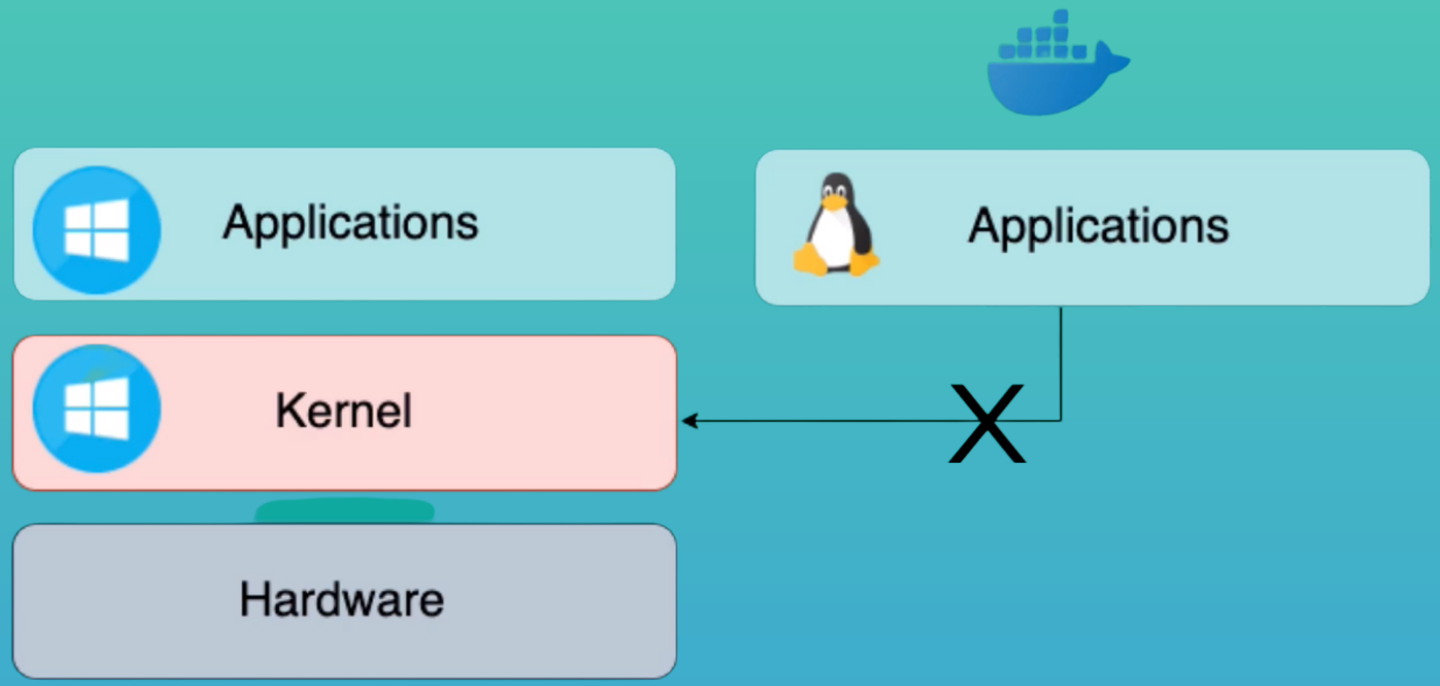
- Abstraction of physical hardware
- Each VM includes a full copy of an OS

Both are **virtualization** tools

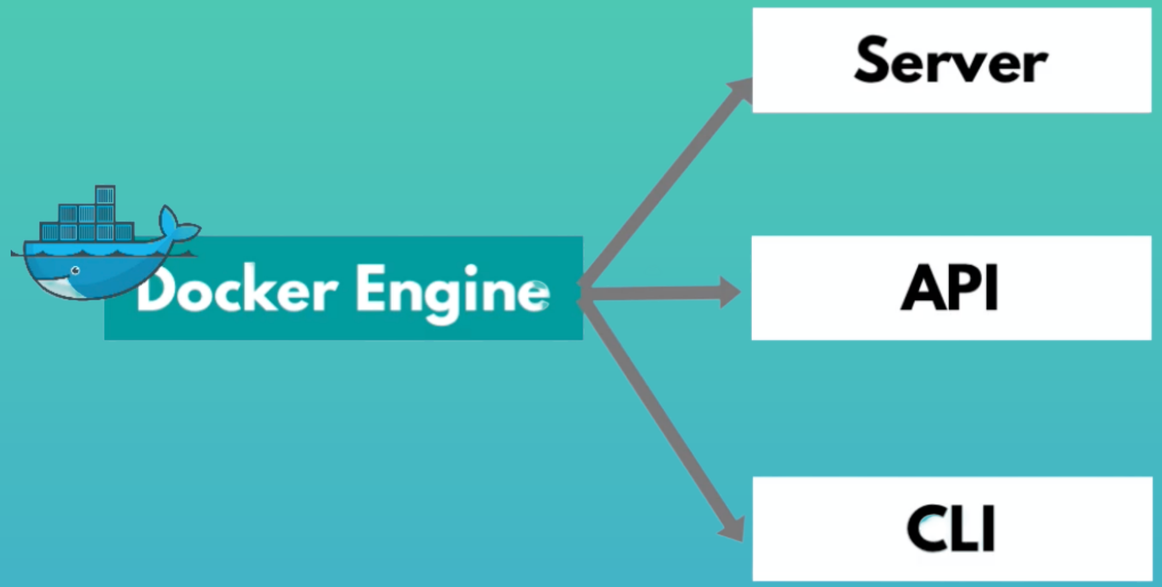
TECHWORLD WITH NANA

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

Docker Desktop for Windows and Mac



Docker Architecture & its components - 1



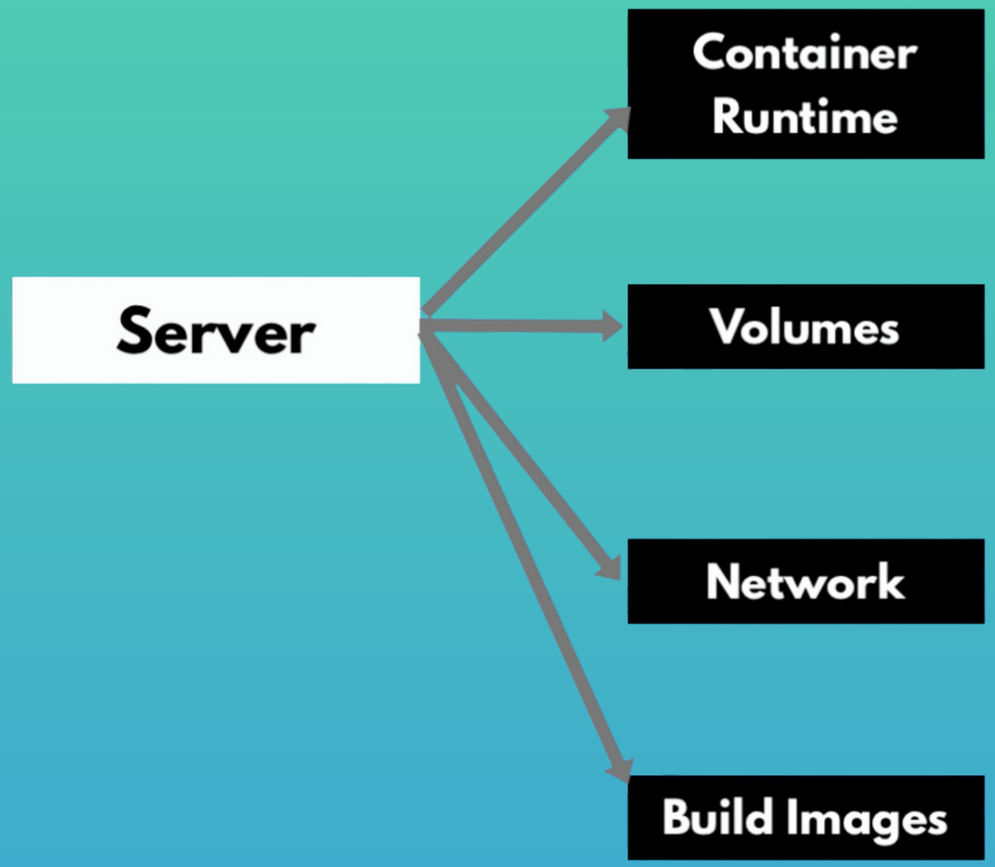
- Pulling images
- Managing images & containers

- Interacting with Docker Server

- Command Line Interface:

  Client to execute Docker commands

Docker Architecture & its components - 2

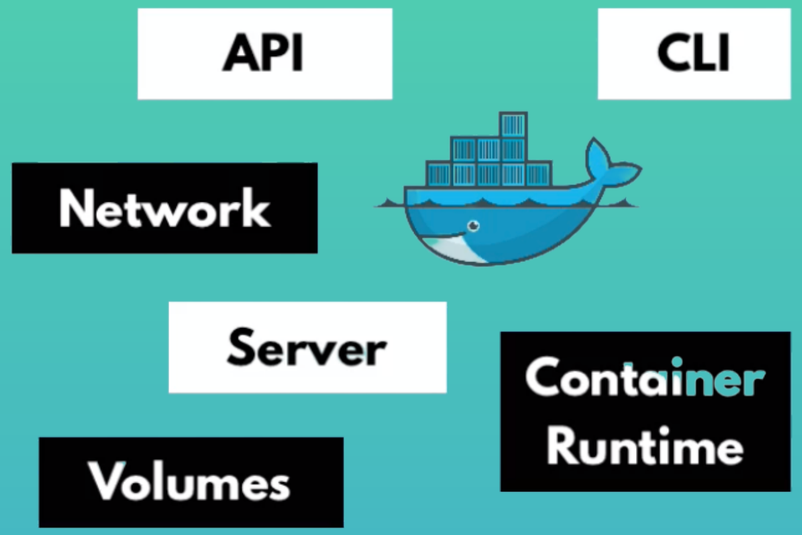


- Pulling images
- Managing container lifecycle

- 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

Deep Dive into Docker

TECHWORLD WITH NANA

Docker Installation

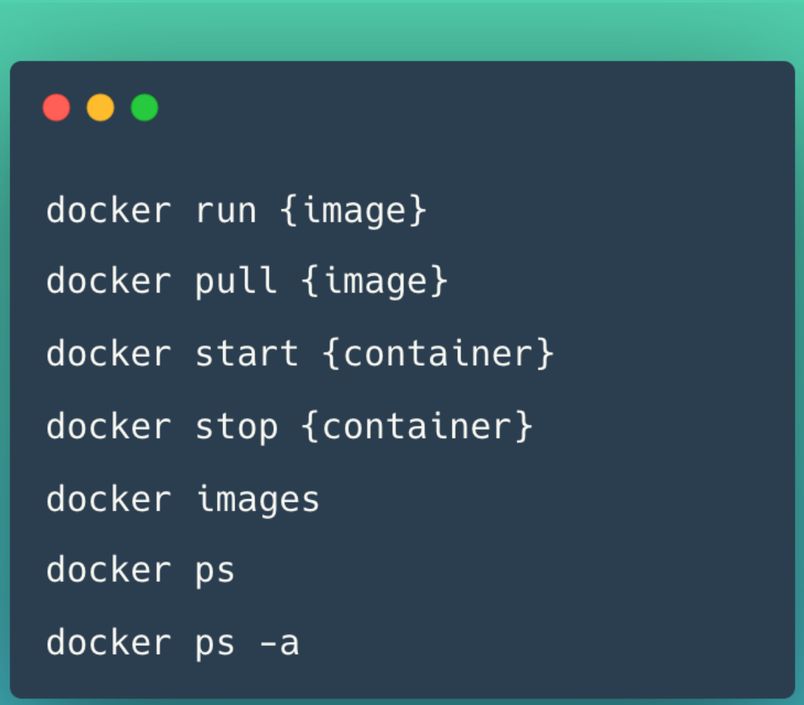
Linux

native support

Mac + Windows

Docker Desktop

Main Docker Commands



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(s)
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

TECHWORLD WITH NANA

Ports in Docker - 1

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

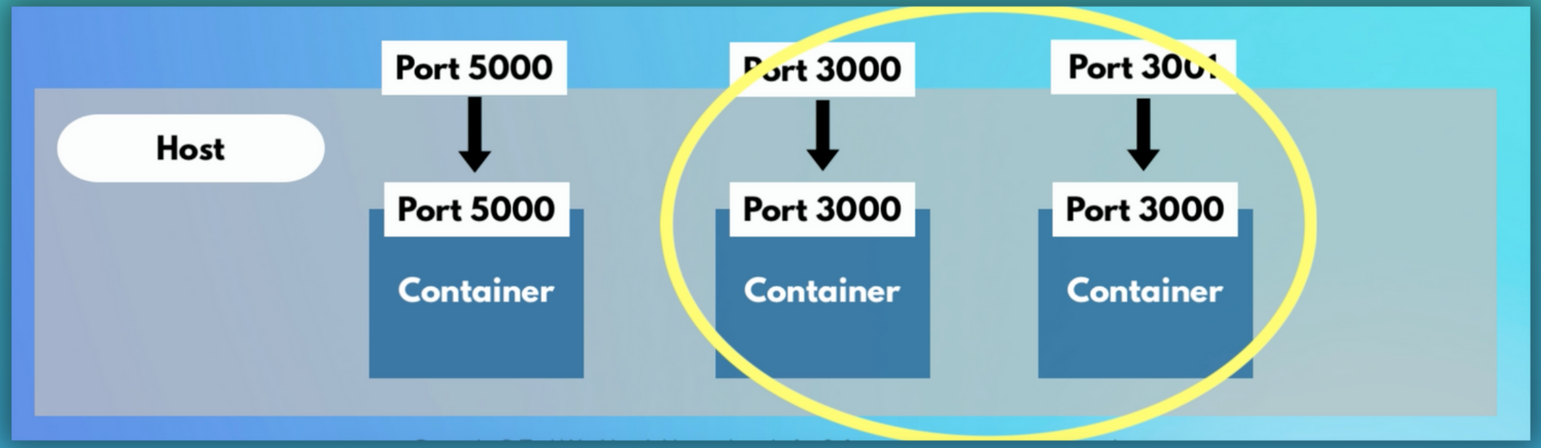
Problem:

- 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}
```

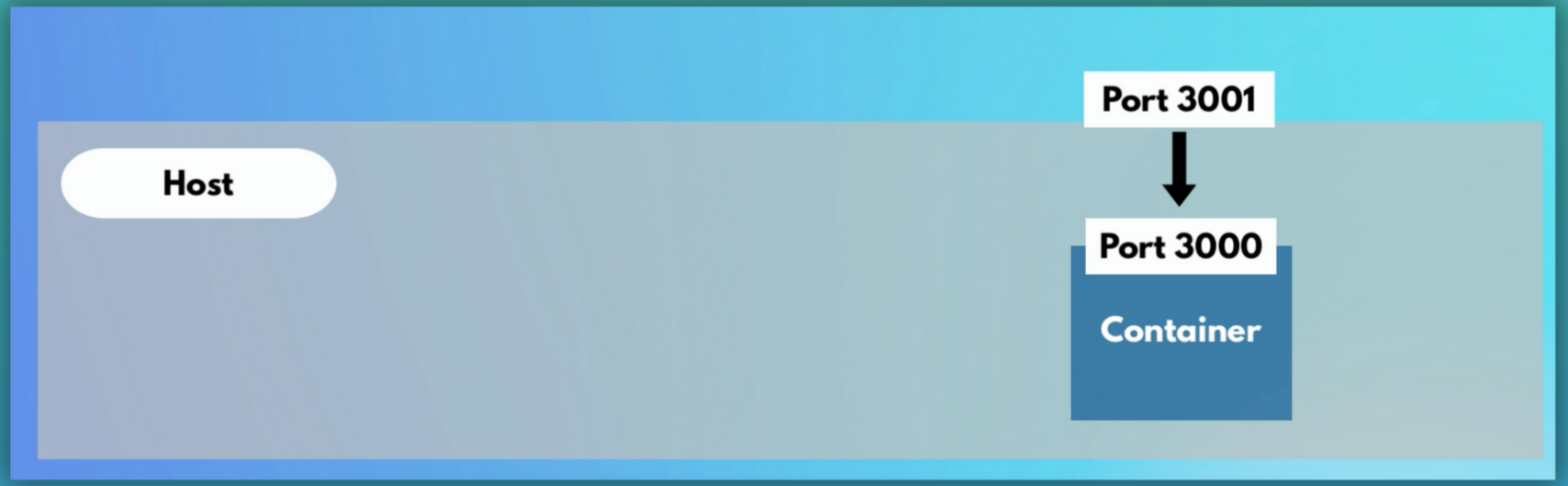
- host port
- container port



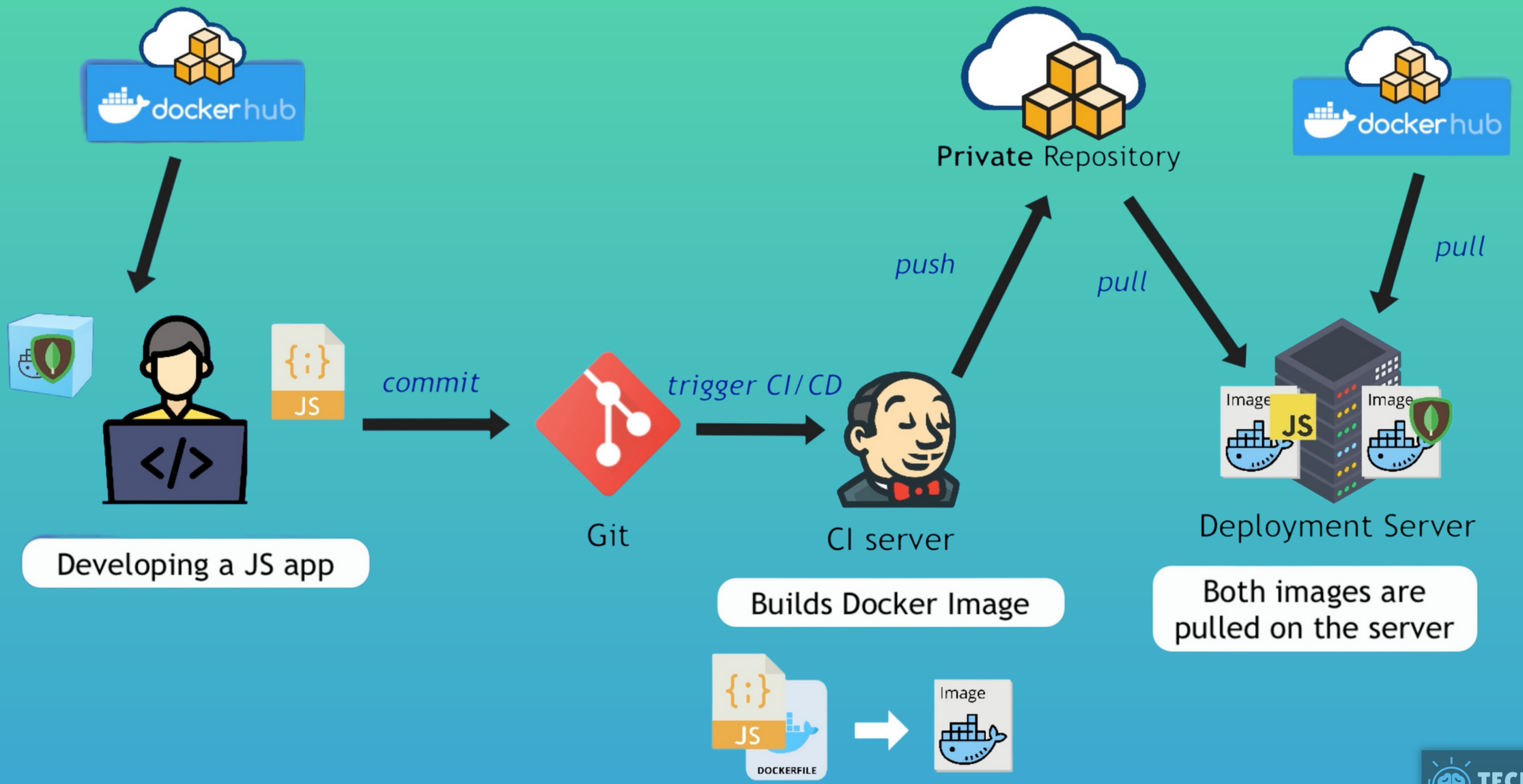
Ports in Docker - 2

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

some-app://localhost:3001



Workflow with Docker



Docker 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 8081:8081 \

-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:
          - 8081:8081
        environment:
          - ME_CONFIG_MONGODB_A....
        ...
```

- Docker Compose automatically creates a **common docker network** for docker containers in it (*--net* option in docker run)

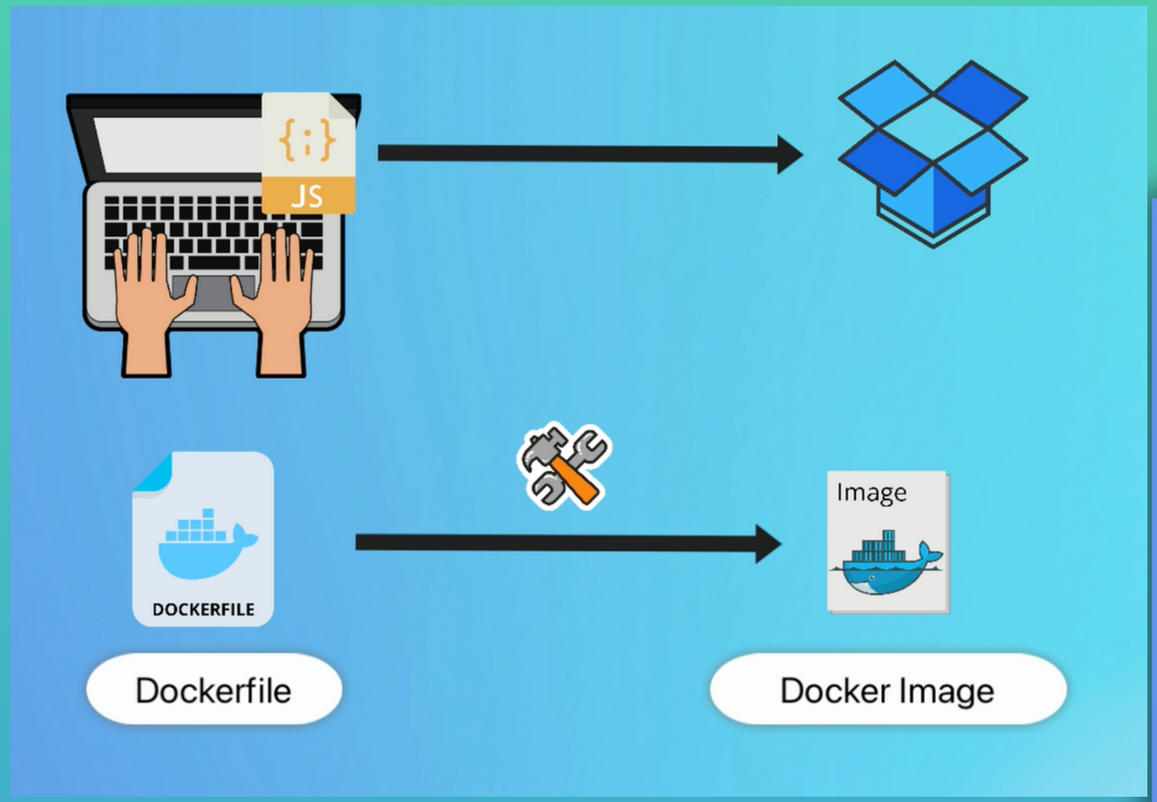
Docker Compose - 2

Example docker-compose.yaml file

```yaml
version: '3'
services:
  mongodb:
    image: mongo
    ports:
    - 27017:27017
    environment:
    - MONGO_INITDB_ROOT_USERNAME=admin
    - MONGO_INITDB_ROOT_PASSWORD=password
  mongo-express:
    image: mongo-express
    ports:
    - 8080:8081
    environment:
    - ME_CONFIG_MONGODB_ADMINUSERNAME=admin
    - ME_CONFIG_MONGODB_ADMINPASSWORD=password
    - 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 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

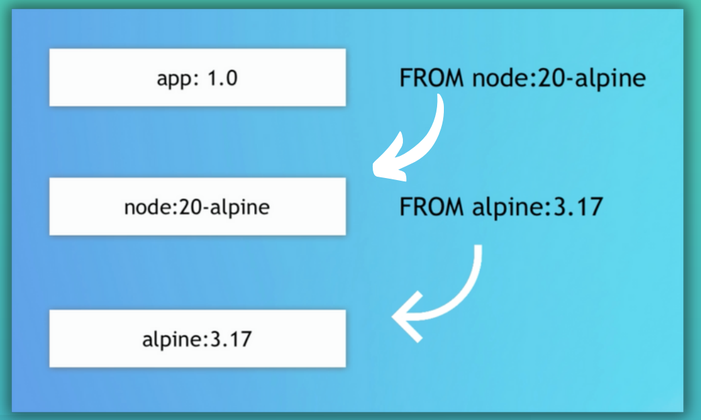
TECHWORLD

TECHWORLD WITH NANA

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:



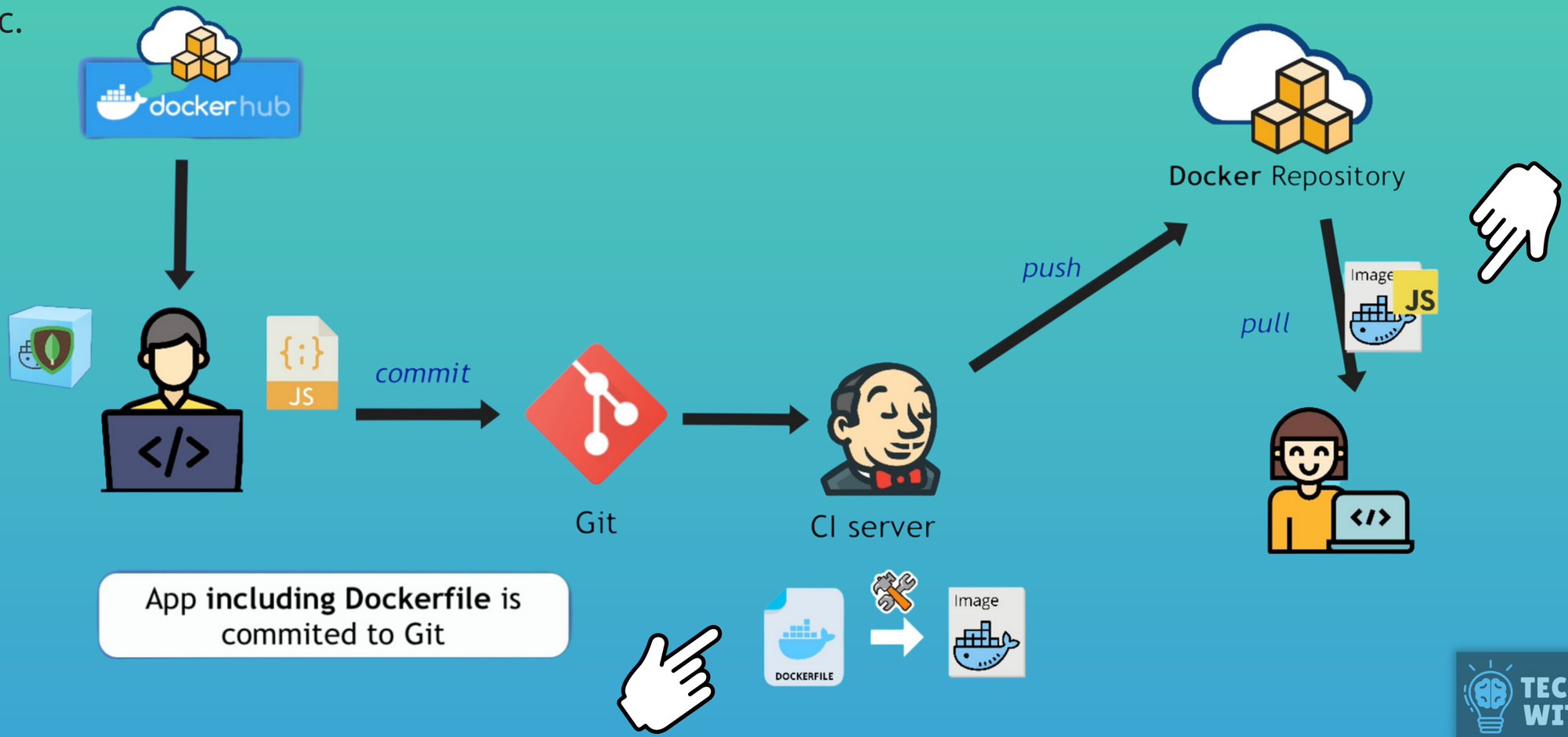
- Command to build an image from a Dockerfile and a context
- The build's context is the set of files at a specified location

```
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 pushed to multiple remote servers or pulled locally for development and testing etc.



Private Docker Repository - 1

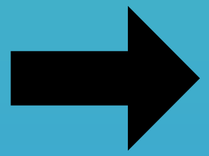
- When you work in a company, you will be working with a private docker registry
- **Public = DockerHub**, Example for **Private** = Amazon Elastic Container Registry "**AWS ECR**"

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

TECHWORLD WITH NANA

Private Docker Repository - 2

In DockerHub

- **docker.io is default** repository

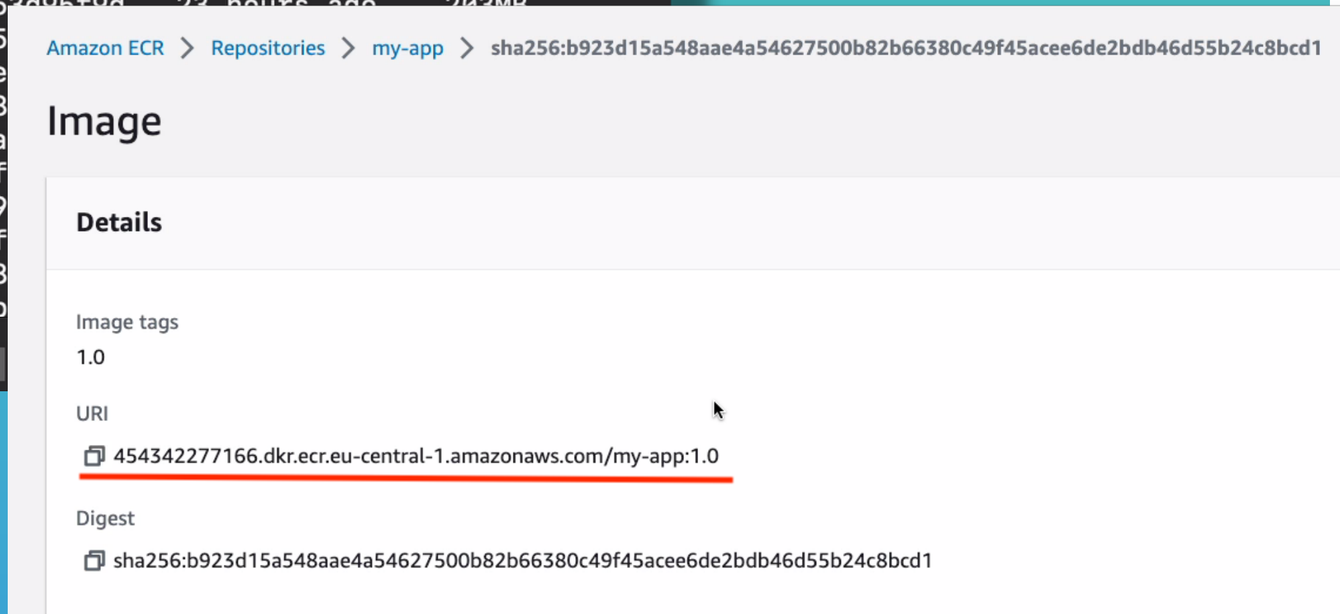
docker pull mongo:4.2 = *docker pull docker.io/library/mongo:4.2*

In AWS ECR or any other private registry

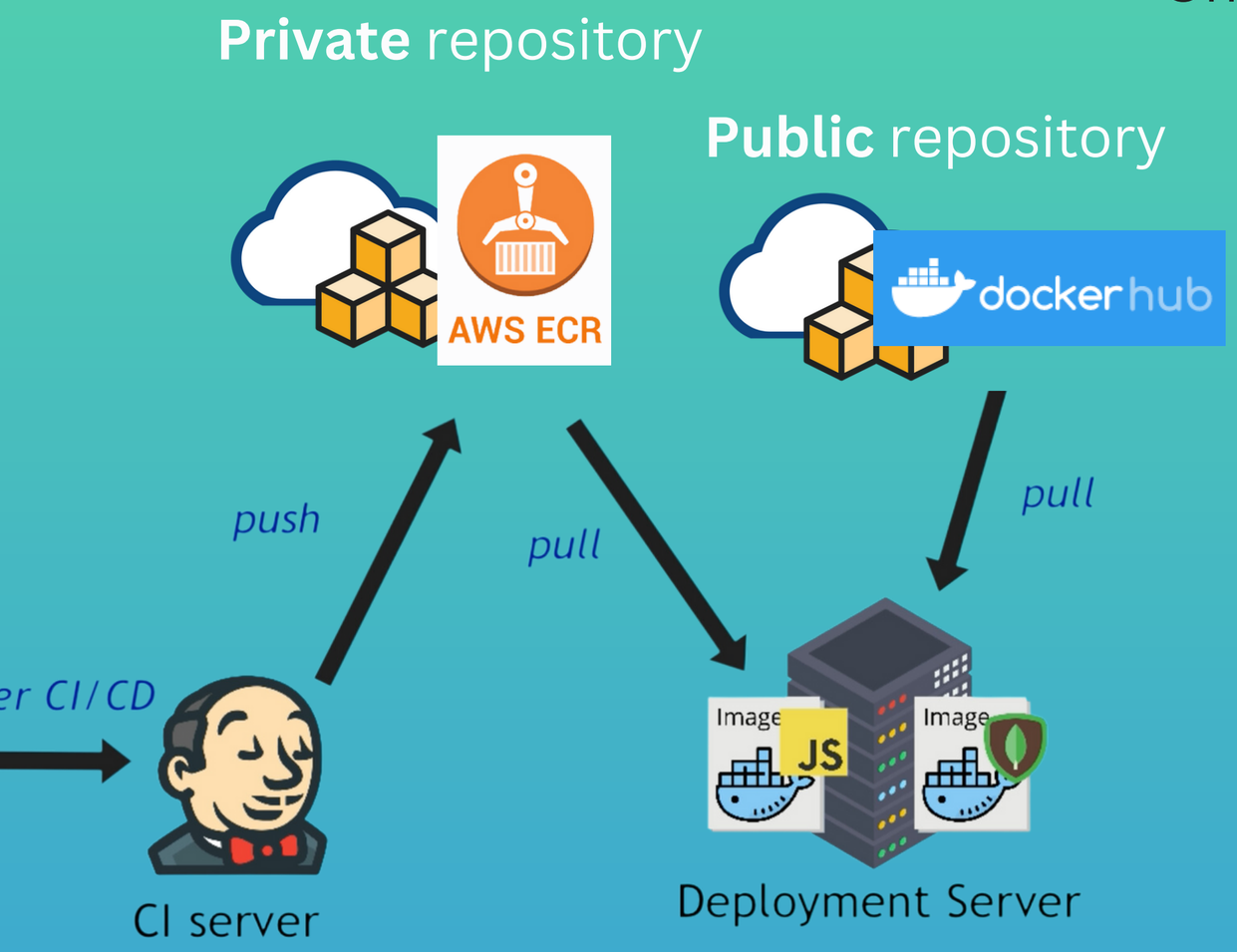
- Need to include the registry domain

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

```
% docker tag my-app:1.0 454342277166.dkr.ecr.eu-central-1.amazonaws.com/my-app:1.0
nana@macbook /Users/nana
% docker images
REPOSITORY                                                TAG       IMAGE ID       CREATED          SIZE
454342277166.dkr.ecr.eu-central-1.amazonaws.com/my-app    1.0       4ef253d95f9d   23 hours ago     203MB
my-app                                                    1.0       4ef25
mongo                                                     latest    8b33e
redis                                                     6.2       f3d38
redis                                                     latest    116ca
postgres                                                  13.10     c562f
postgres                                                  14.7      1c159
debian                                                    latest    34b4f
mongo-express                                             0.54.0    05378
mongo-express                                             latest    2d2fb
nana@macbook /Users/nana
% docker push 454342277166.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** (like AWS ECR, Azure Container Registry etc.)

Public Images (Mysql, MongoDB)

↓

from **public docker repositories** (like DockerHub)

TECHWORLD WITH NANA

Docker Volumes - 1

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

Why are volumes 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.

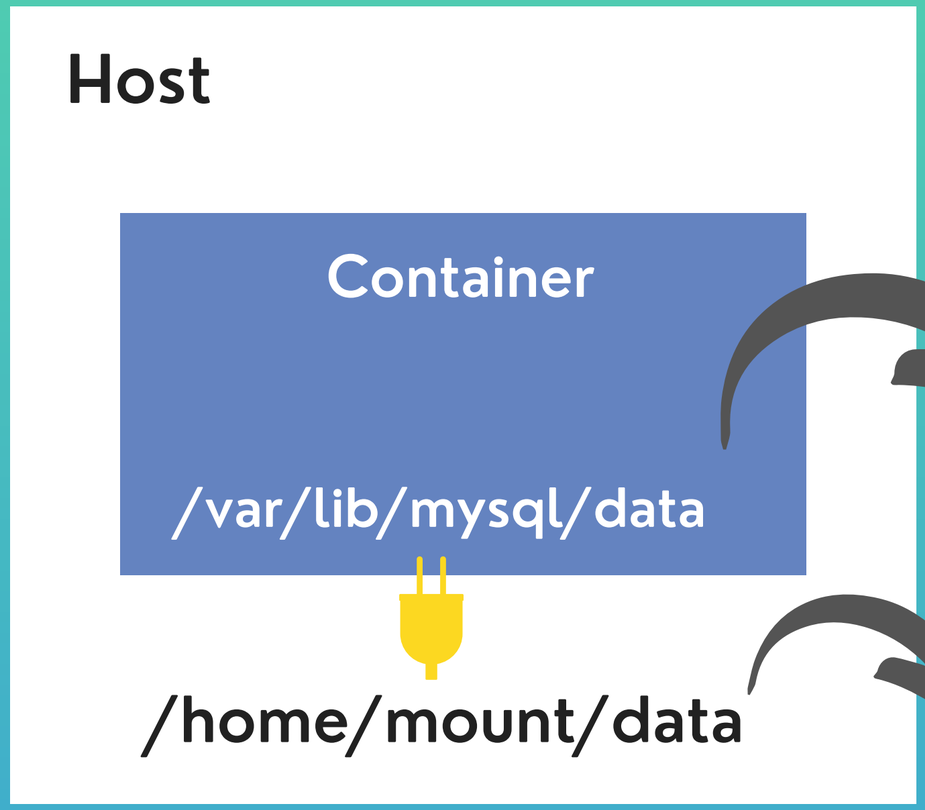
Important for

- 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**

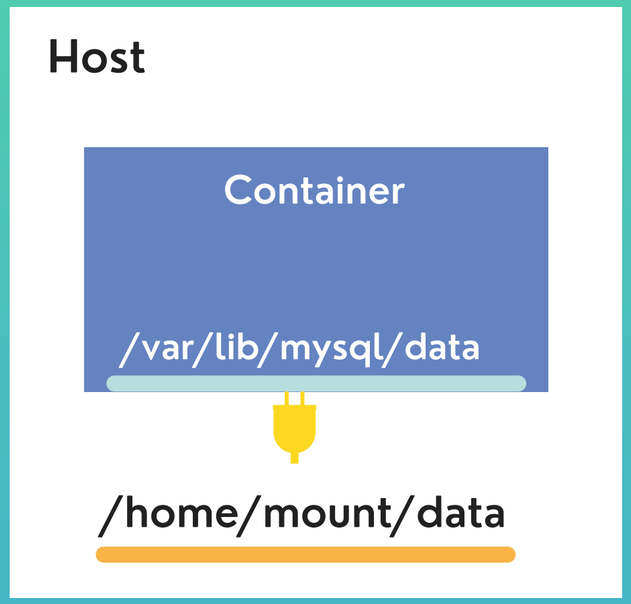
Virtual File System inside container

Host File System (physical)

Docker Volumes - 3

3 Volume Types

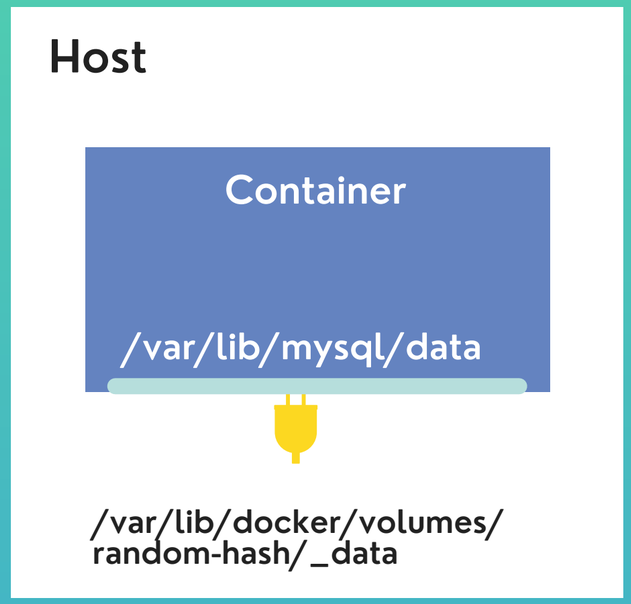
Host Volumes



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

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

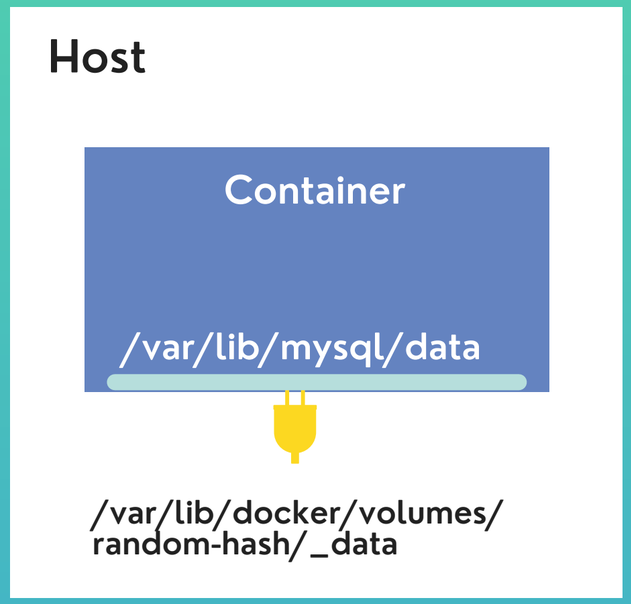
Anonymous Volumes



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

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

Named Volumes



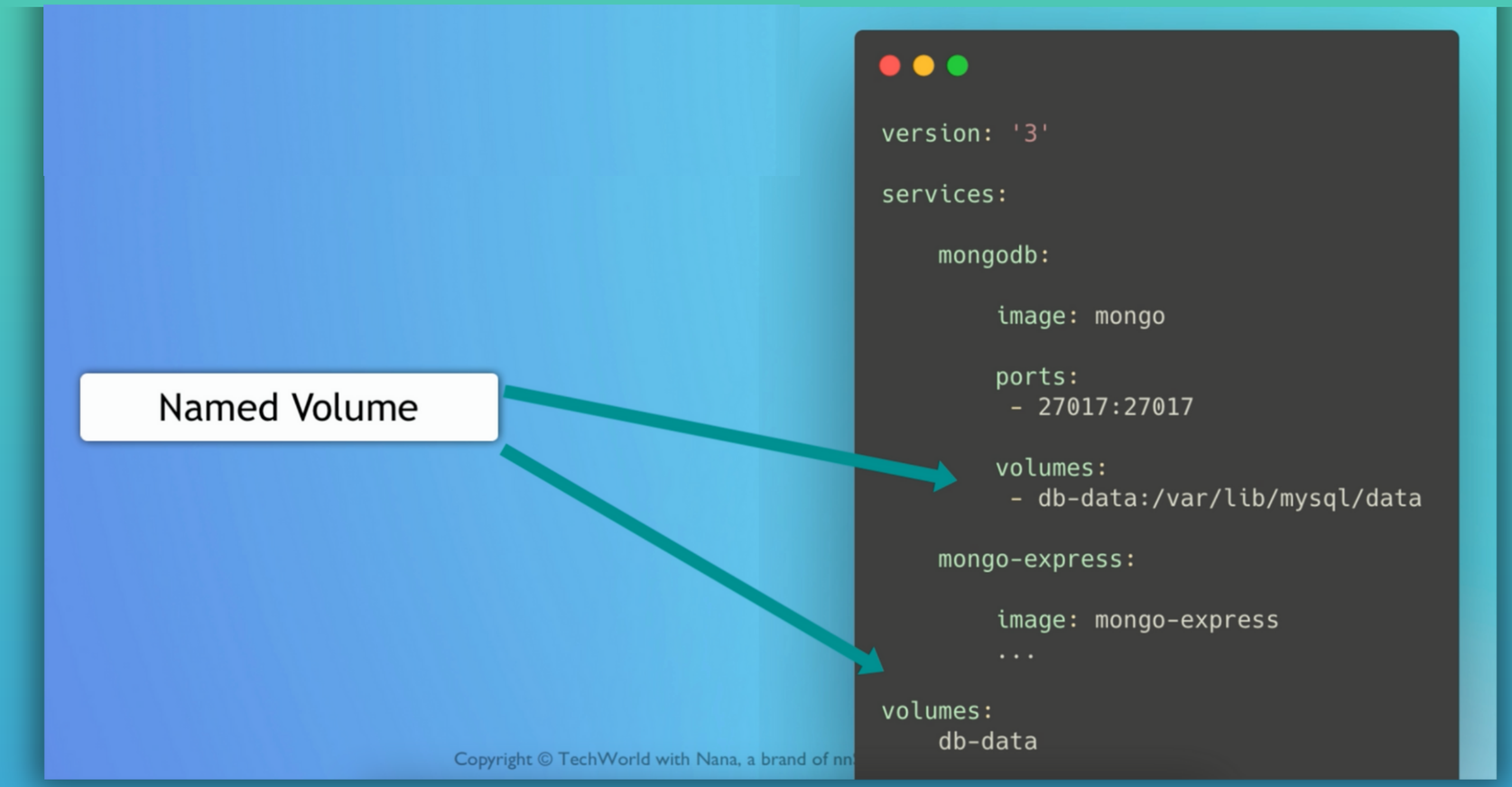
- You can **reference** the volume **by name**
- Should be used in production

```
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