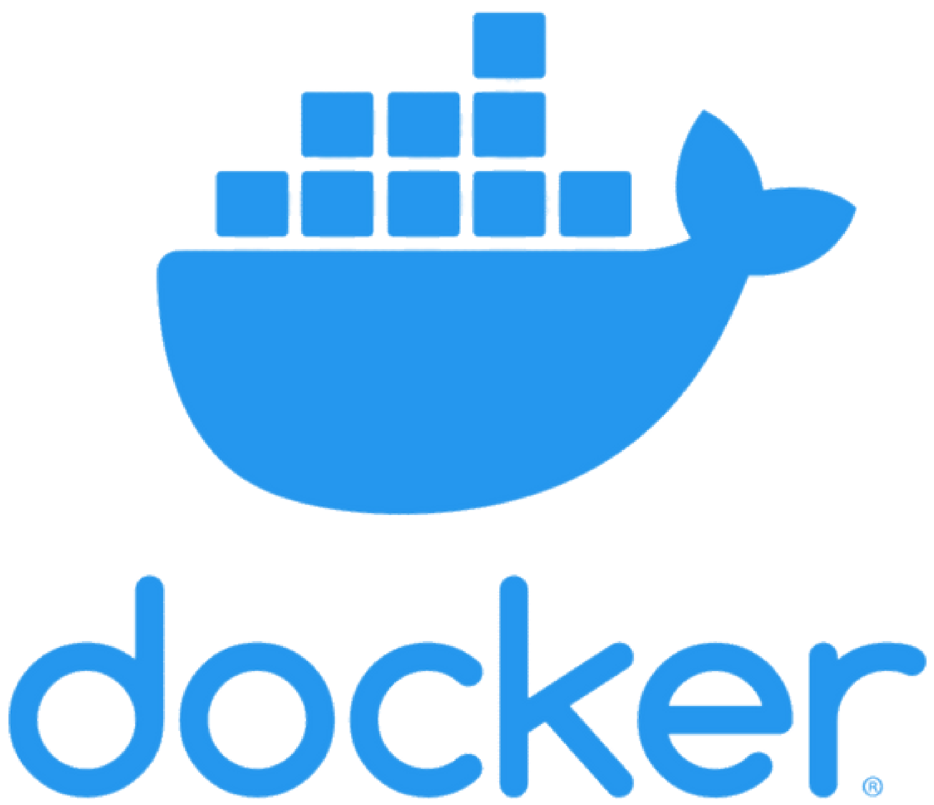
# Introduction à Docker



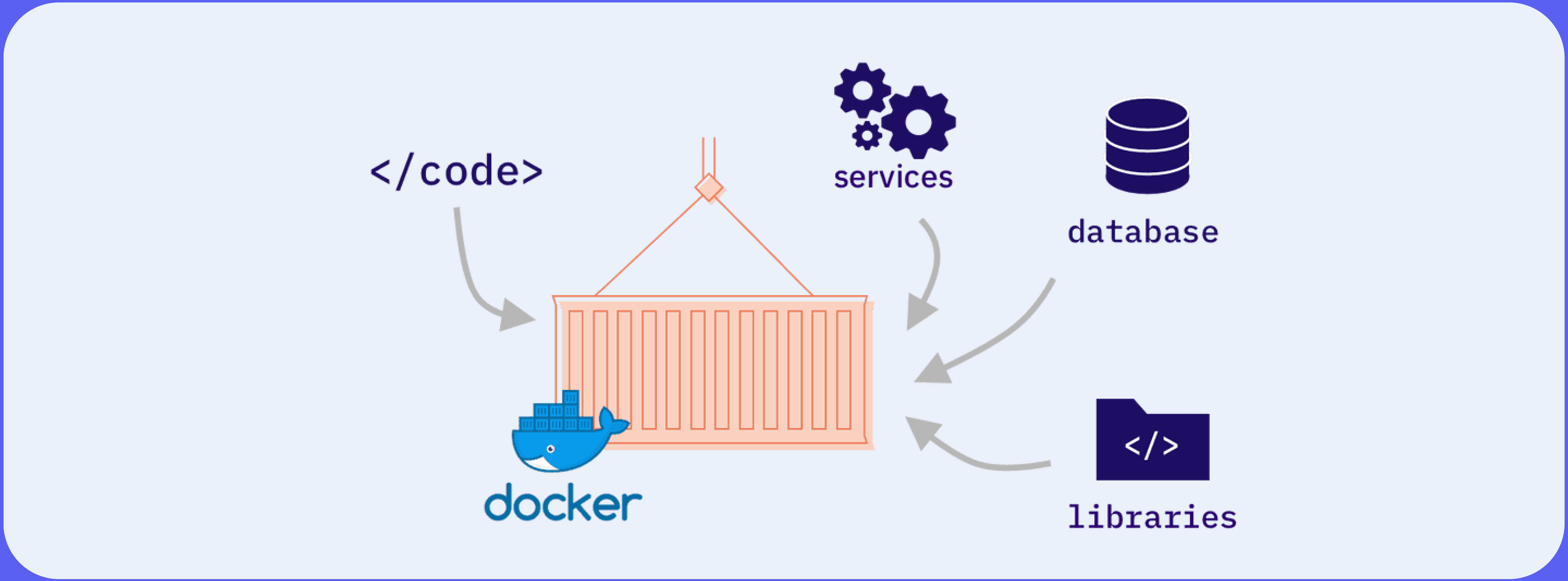
# Planning

“A goal without a plan is just a wish.” - Me

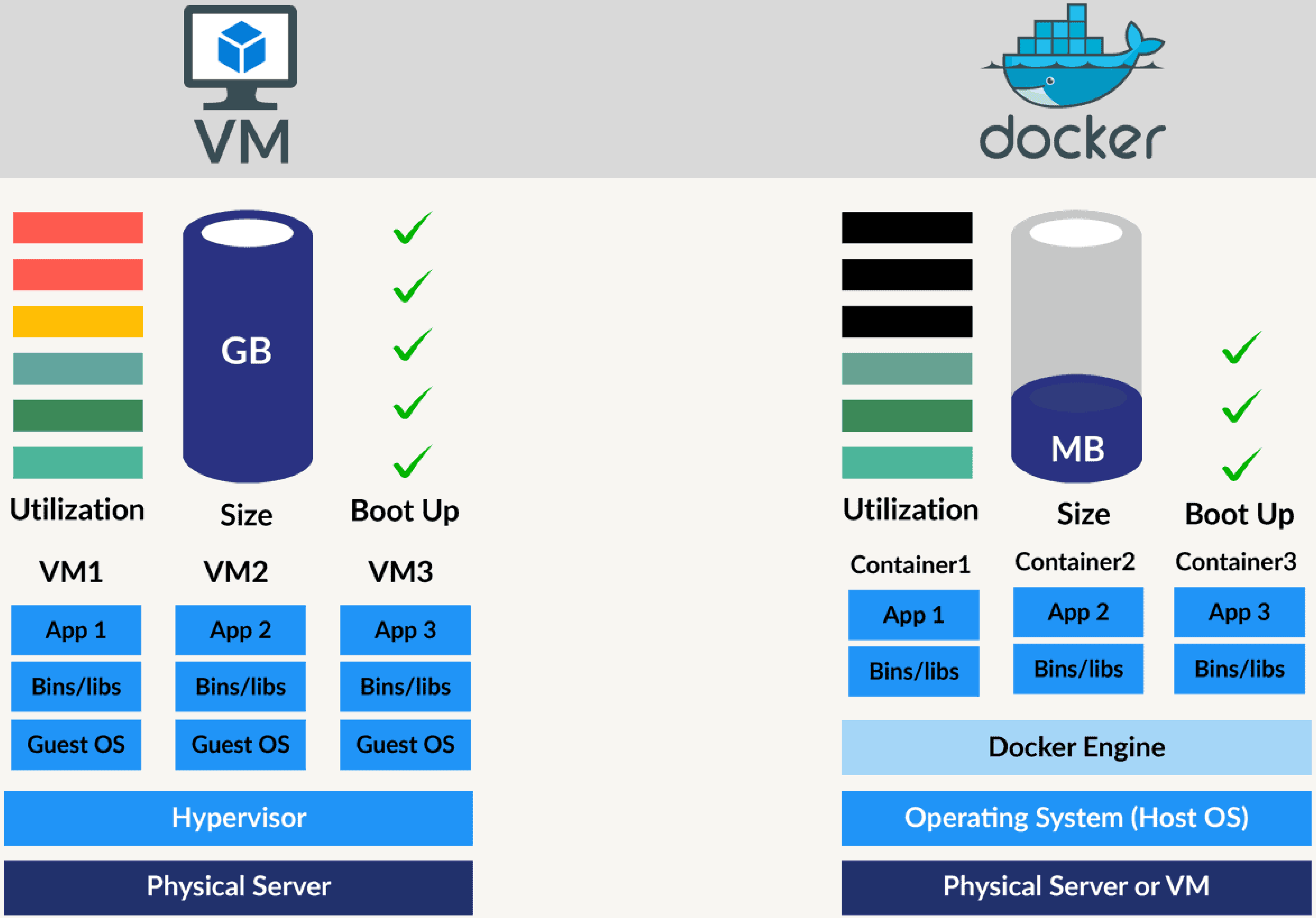
- Run a container with a simple Dockerfile
- Deal with communication between containers
- Configure your whole project with Docker Compose

# What is Docker ?

“Docker is a platform and tool that allows you to develop, deploy, and run applications in containers.” - ChatGPT



# Docker vs Virtual Machine



VM

GB

Utilization

Size

Boot Up

VM1 VM2 VM3

App 1 App 2 App 3

Bins/libs Bins/libs Bins/libs

Guest OS Guest OS Guest OS

Hypervisor

Physical Server

docker

MB

Utilization

Size

Boot Up

Container1 Container2 Container3

App 1 App 2 App 3

Bins/libs Bins/libs Bins/libs

Docker Engine

Operating System (Host OS)

Physical Server or VM

# Part 0

Installation of Docker

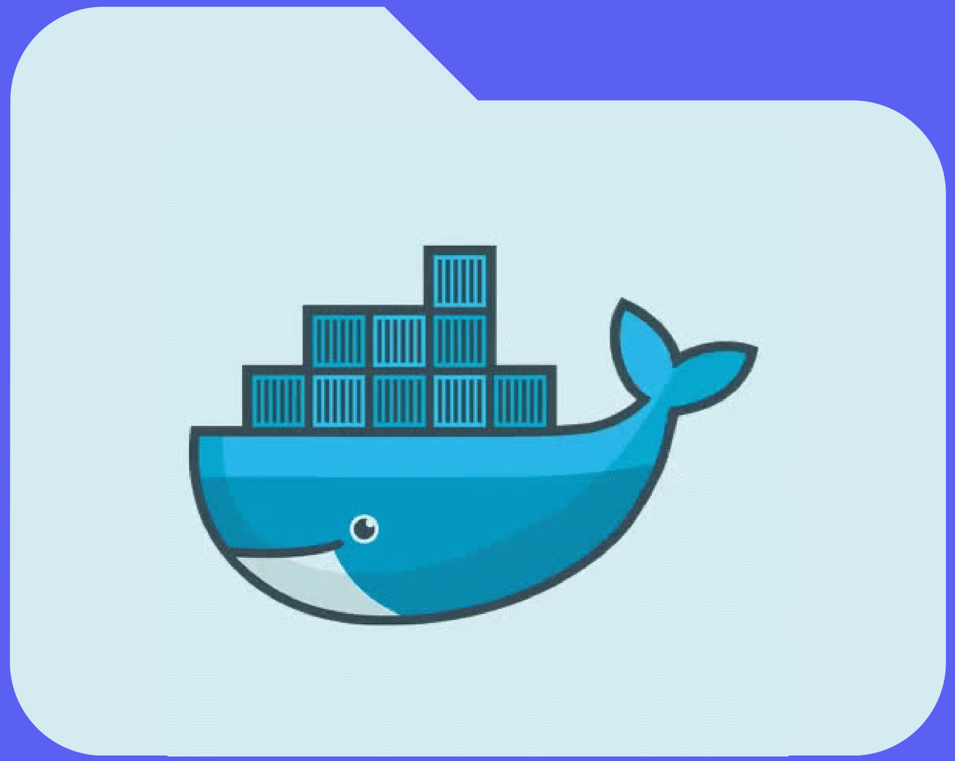
# How to install

| | | |
|---|---|---|
| Windows | ✚ | https://docs.docker.com/desktop/install/windows-install/ |
| MacOS | ✚ | https://docs.docker.com/desktop/install/mac-install/ |
| Ubuntu | ✚ | https://docs.docker.com/engine/install/ubuntu/#install-using-the-repository |

“If you have WSL on Windows, you will install on WSL (Ubuntu)” - Me

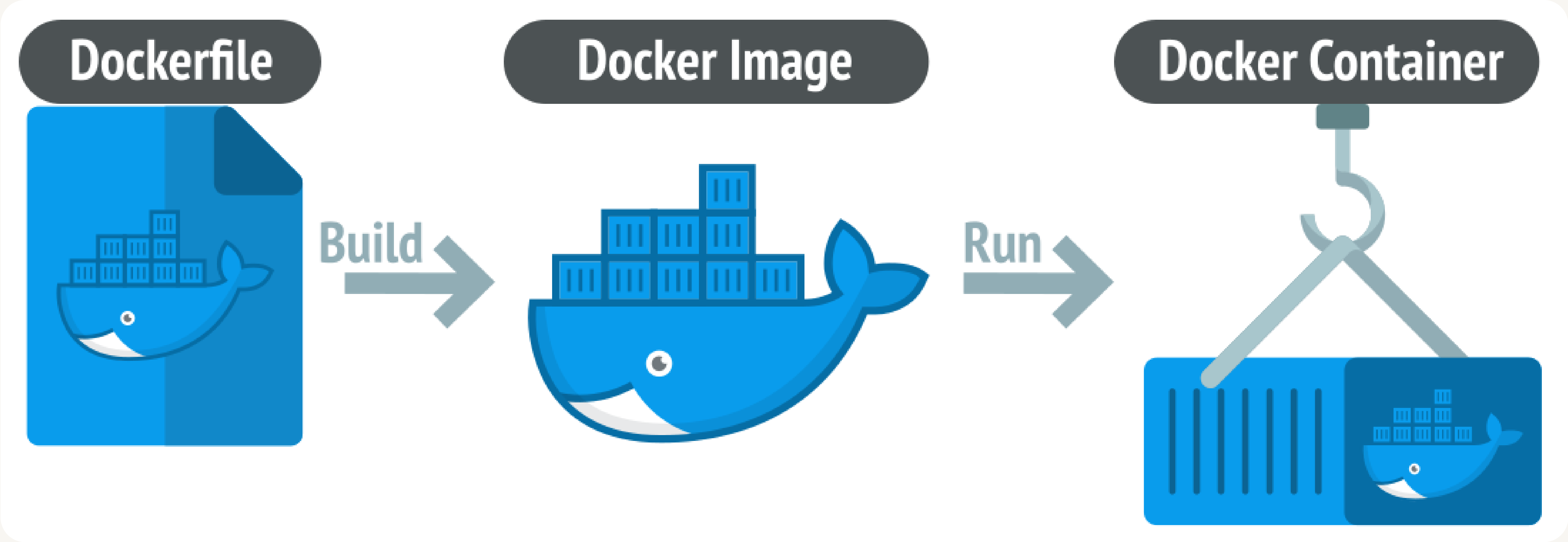
# Part I

Creating your first Dockerfile



# What is a Dockerfile ?

“A Dockerfile is a plain text configuration file used in Docker to define and create a Docker image.” - ChatGPT



# Your first Dockerfile

## Let’s learn the basic keywords of Dockerfile

**FROM**: The FROM command specifies the base image from which your Docker image will be built.

**RUN**: The RUN command executes a command in the image during the image-building process.

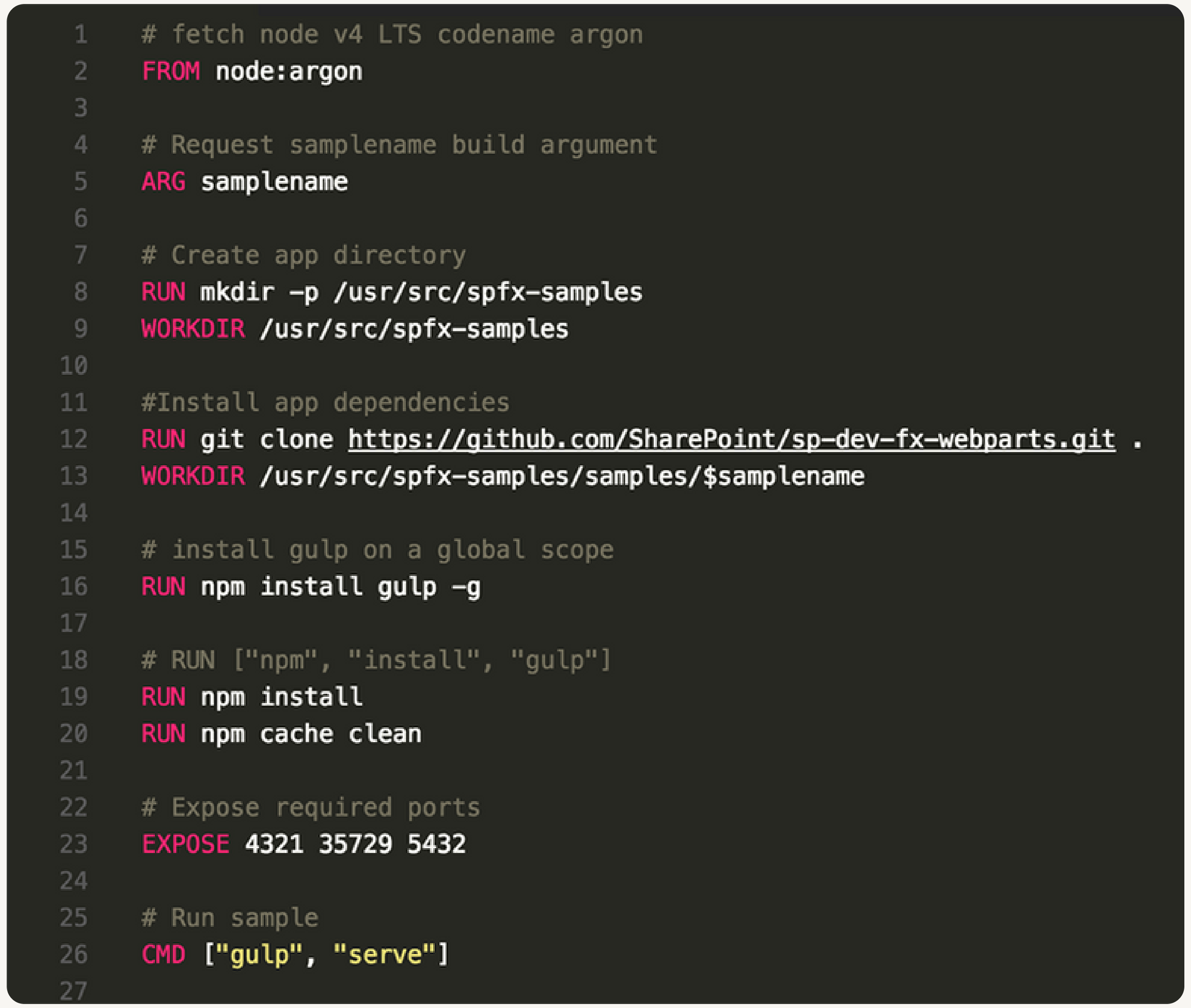
**COPY**: The COPY command is used to copy files and directories from your local system into the image.

**WORKDIR**: The WORKDIR command sets the working directory within the container.

**EXPOSE**: The EXPOSE command specifies which network ports the container should listen on.

**CMD:** The CMD command defines the default command that should be executed when a container is started from the image.

## Here is an exemple of a Dockerfile



```
# fetch node v4 LTS codename argon
FROM node:argon

# Request samplename build argument
ARG samplename

# Create app directory
RUN mkdir -p /usr/src/spfx-samples
WORKDIR /usr/src/spfx-samples

#Install app dependencies
RUN git clone https://github.com/SharePoint/sp-dev-fx-webparts.git .
WORKDIR /usr/src/spfx-samples/samples/$samplename

# install gulp on a global scope
RUN npm install gulp -g

# RUN ["npm", "install", "gulp"]
RUN npm install
RUN npm cache clean

# Expose required ports
EXPOSE 4321 35729 5432

# Run sample
CMD ["gulp", "serve"]

```

# Docker client commands

## Let’s learn the basic commands of the client

**docker version:** This command displays the Docker client and server version information

**docker images:** This command lists all the Docker images that are currently available on your system.

**docker build:** This command create an image from the specified Dockerfile

**docker ps:** This command shows a list of running containers

**docker run:** This command is used to create and start a new container from a specific image

**docker stop:** This command stop to halt a running container

**docker start:** This command restarts a stopped container

# A MySQL Dockerfile

## Now follow those instructions to create a Dockerfile for your MySQL database

1. You must start from the **mysql:latest** image
2. Set up environment variables for the **root password** and the **database**
3. Copy the **init.sql** file into the **/docker-entrypoint-initdb.d/** folder.
4. Expose the **3306** port

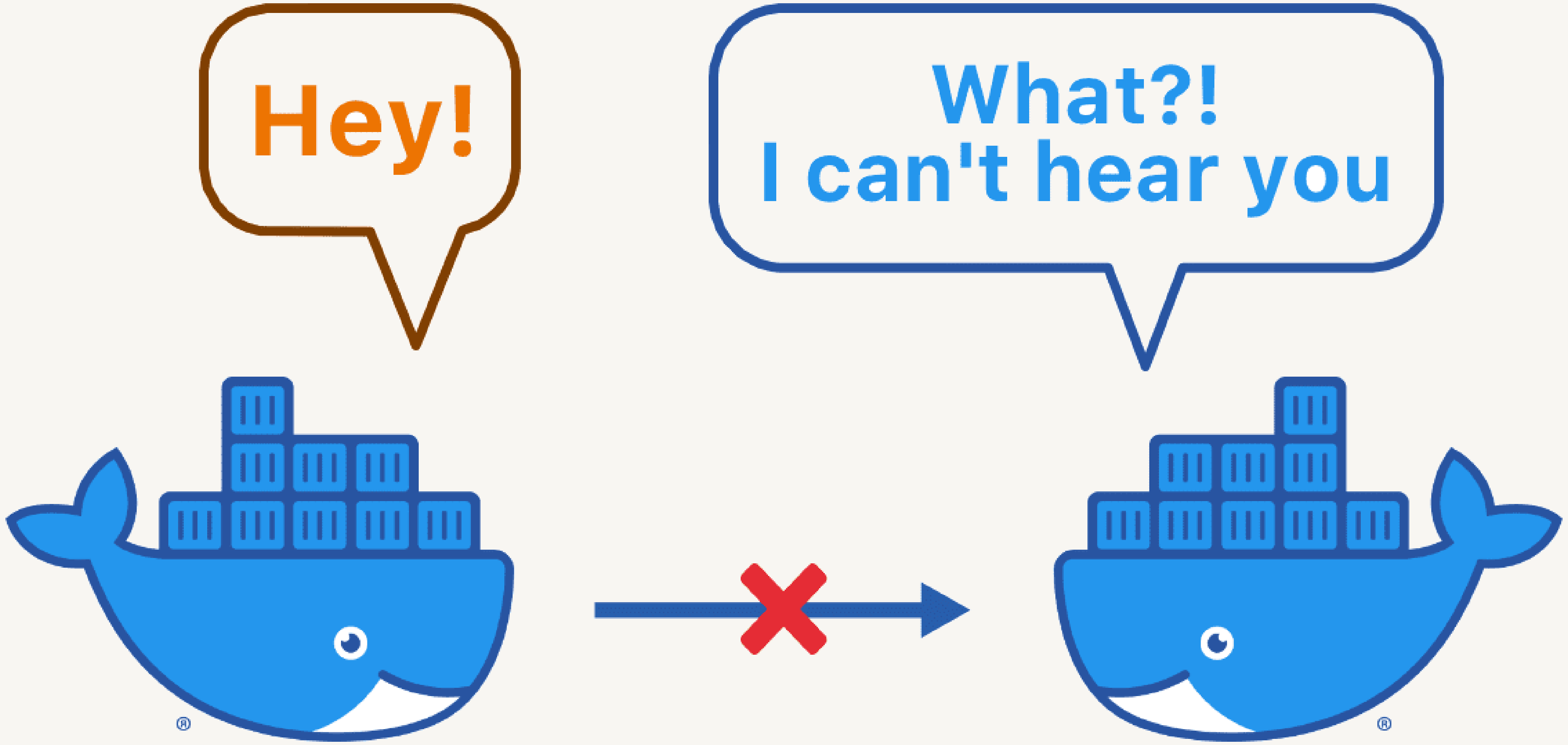
“If you want to try your MySQL database, just connect to it.” - Edouard Chhang

# Part II

Connecting multiple docker containers

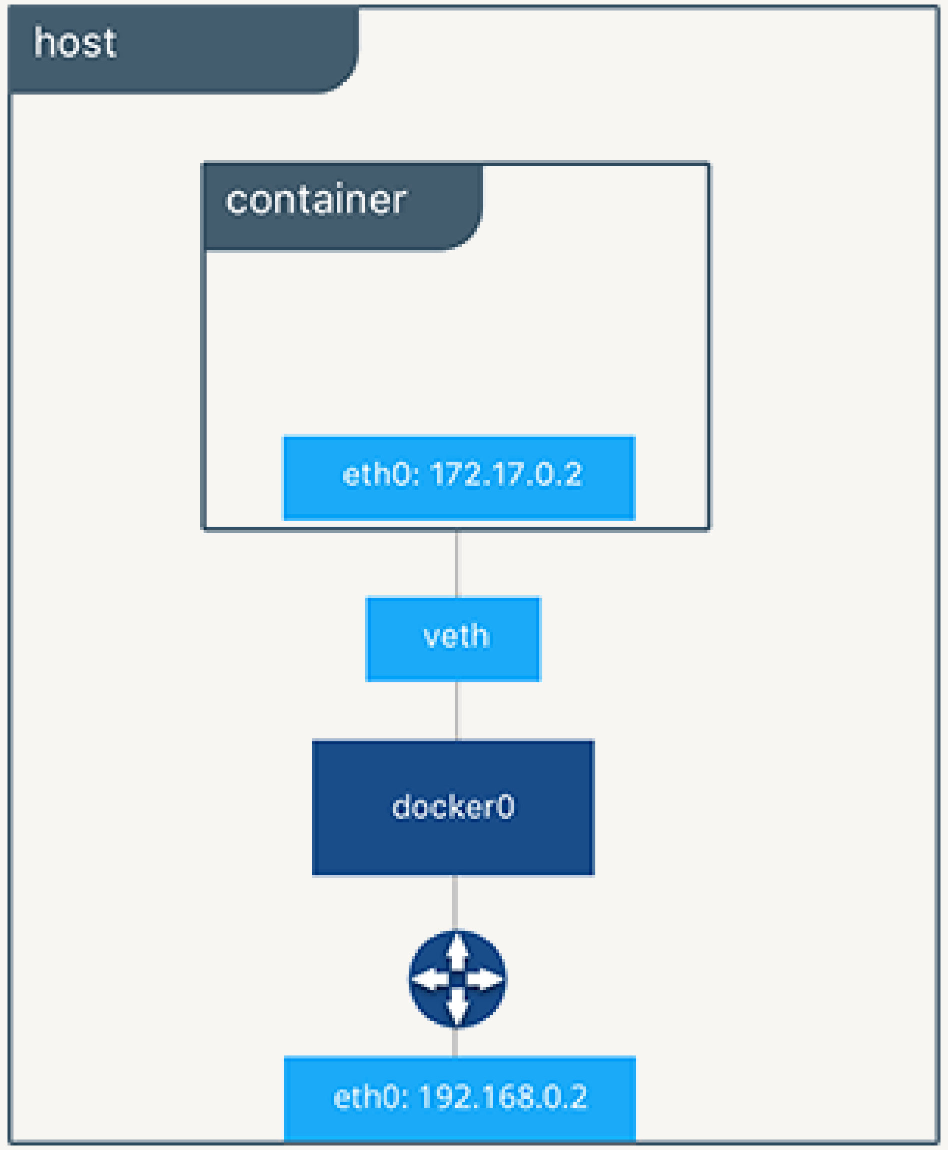
# What is a Docker network ?

“A Docker network is a virtual network that allows Docker containers to communicate with each other and with external networks in a controlled and isolated manner.” - ChatGPT



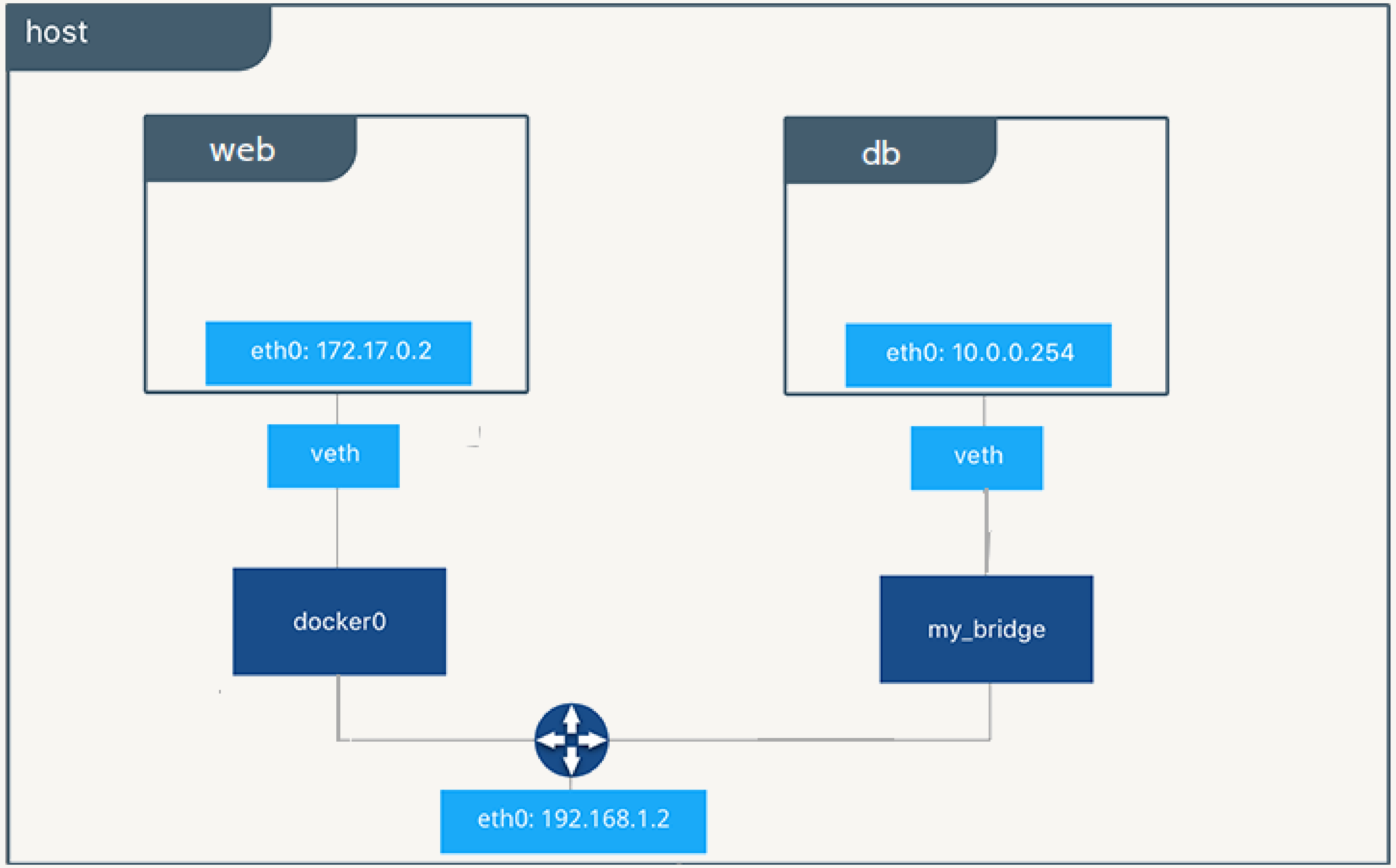
# Communication between containers

## Initial state of a container in networks



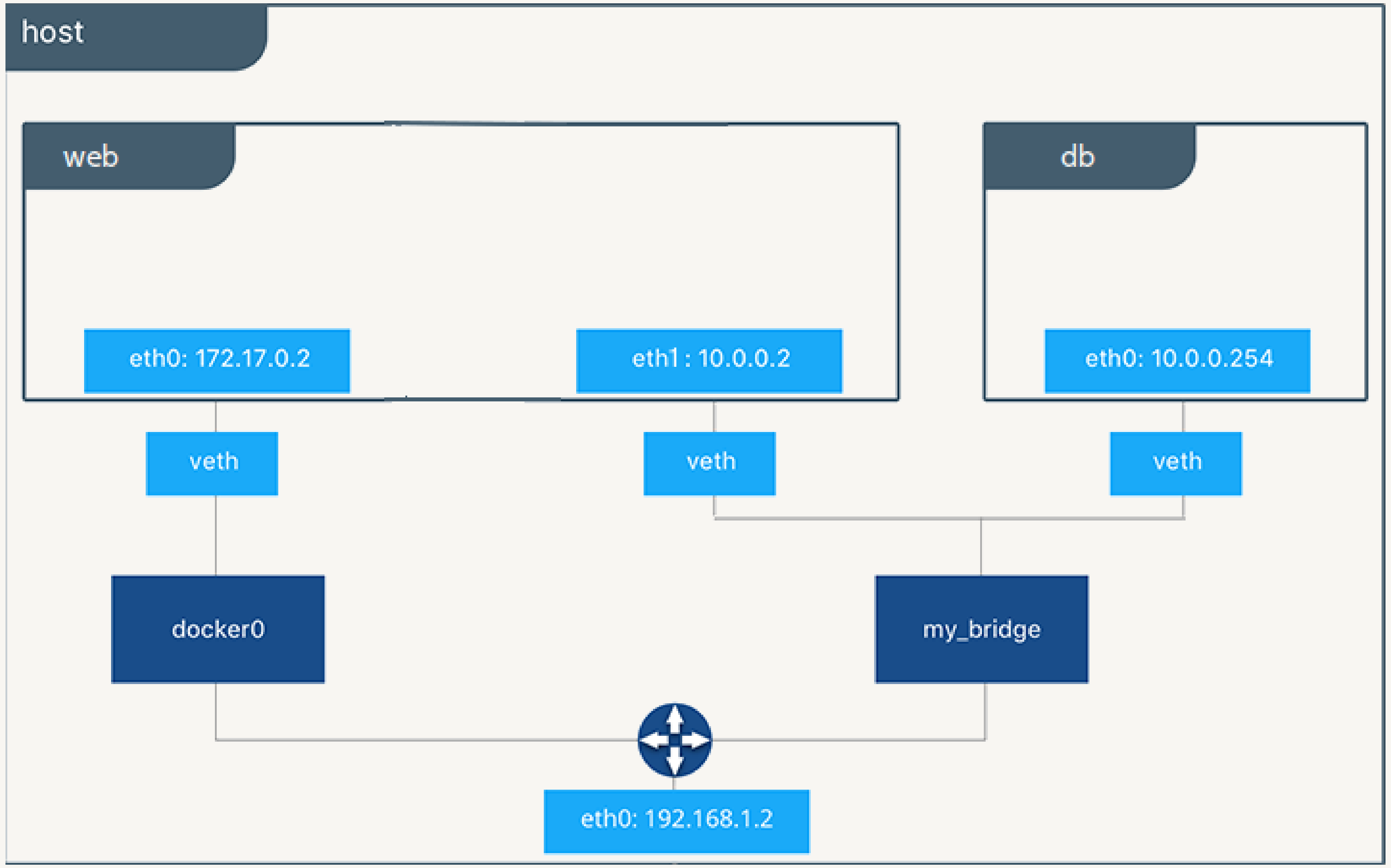
# Communication between containers

## Two containers without any network configuration



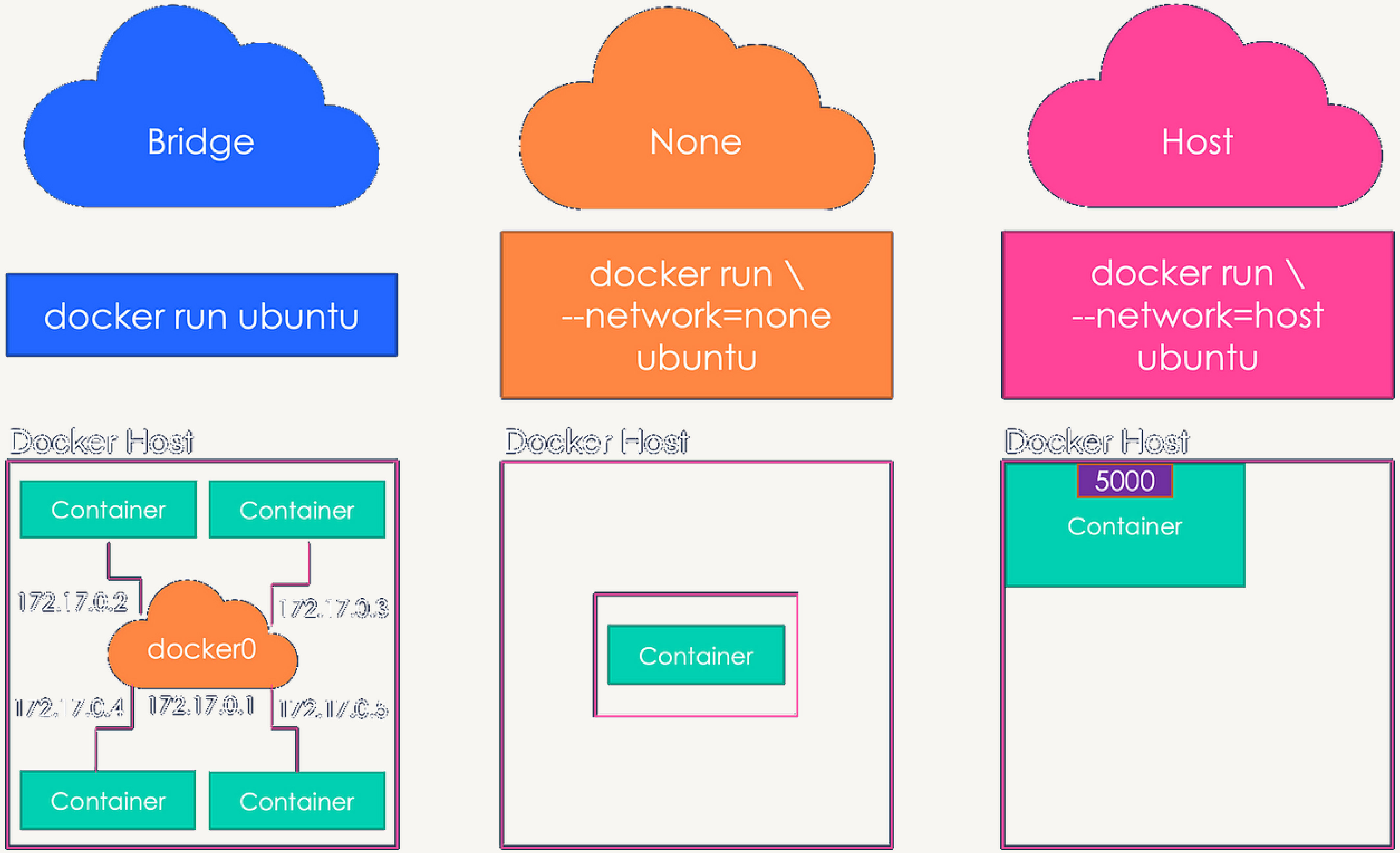
# Communication between containers

## Two containers with network configuration



# Different types of networks driver

## The three basic network drivers



```
docker run ubuntu
```

```
docker run \
--network=none
ubuntu
```

```
docker run \
--network=host
ubuntu
```

# A Node.js Dockerfile

## Now follow those instructions to create a Dockerfile for your Node.js API

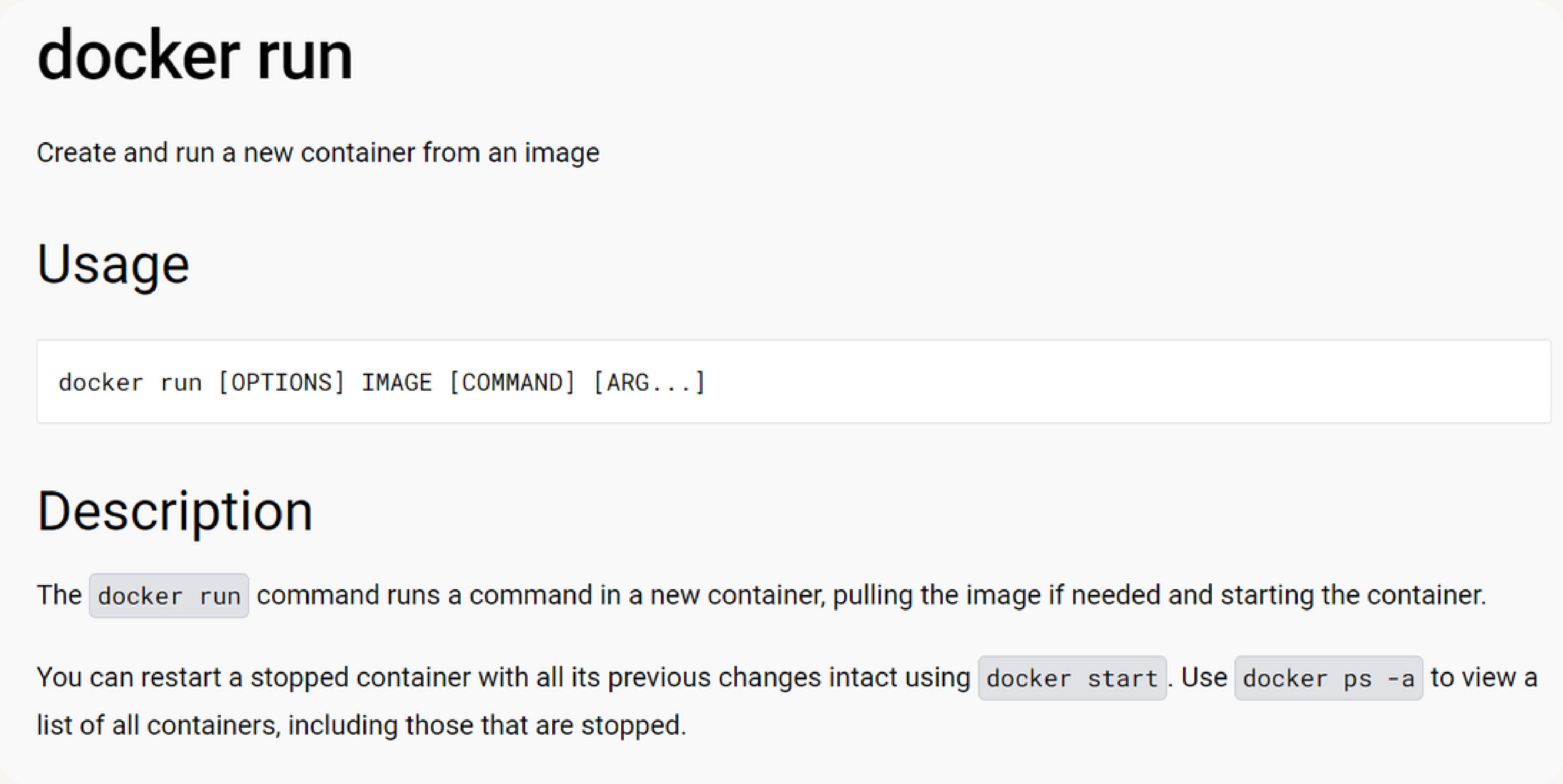
1. You must start from the **node:18-alpine3.18** image
2. Set the working directory to **/app**
3. Copy the **application code** to the working directory
4. Install dependencies by running **npm install**
5. Expose the **3300** port
6. Finally start the application by running **npm start**

“If you want to try your Node.js API, just send a request to it.” - Edouard Chhang

# Run containers with the right network

## Let's take a look at the docker run command documentation to choose the right network !

Find by yourself on the documentation what option to specify with the docker run command so you can select a network for your container.



## docker run

Create and run a new container from an image

### Usage

```
docker run [OPTIONS] IMAGE [COMMAND] [ARG...]
```

### Description

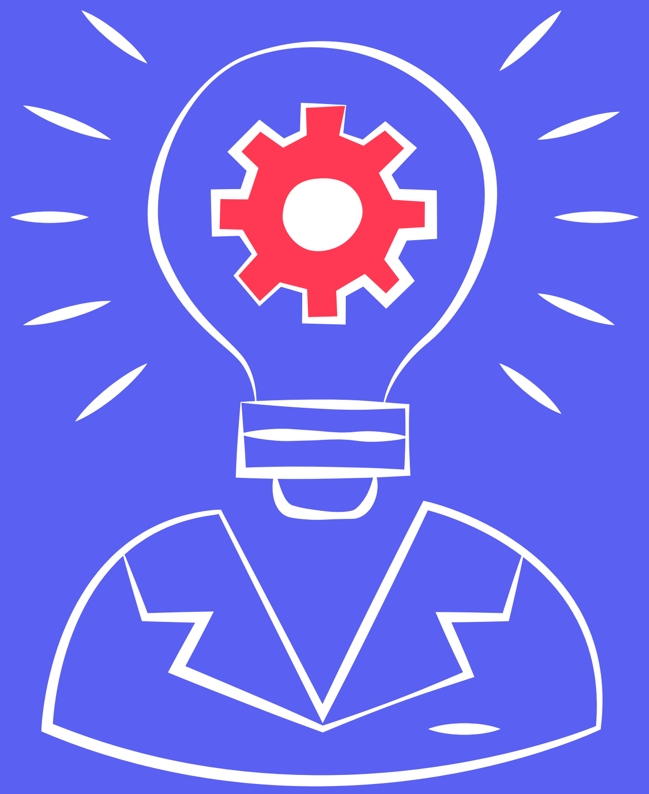
The `docker run` command runs a command in a new container, pulling the image if needed and starting the container.

You can restart a stopped container with all its previous changes intact using `docker start`. Use `docker ps -a` to view a list of all containers, including those that are stopped.

"If you want to find something, just CTRL+F it." - Edouard Chhang

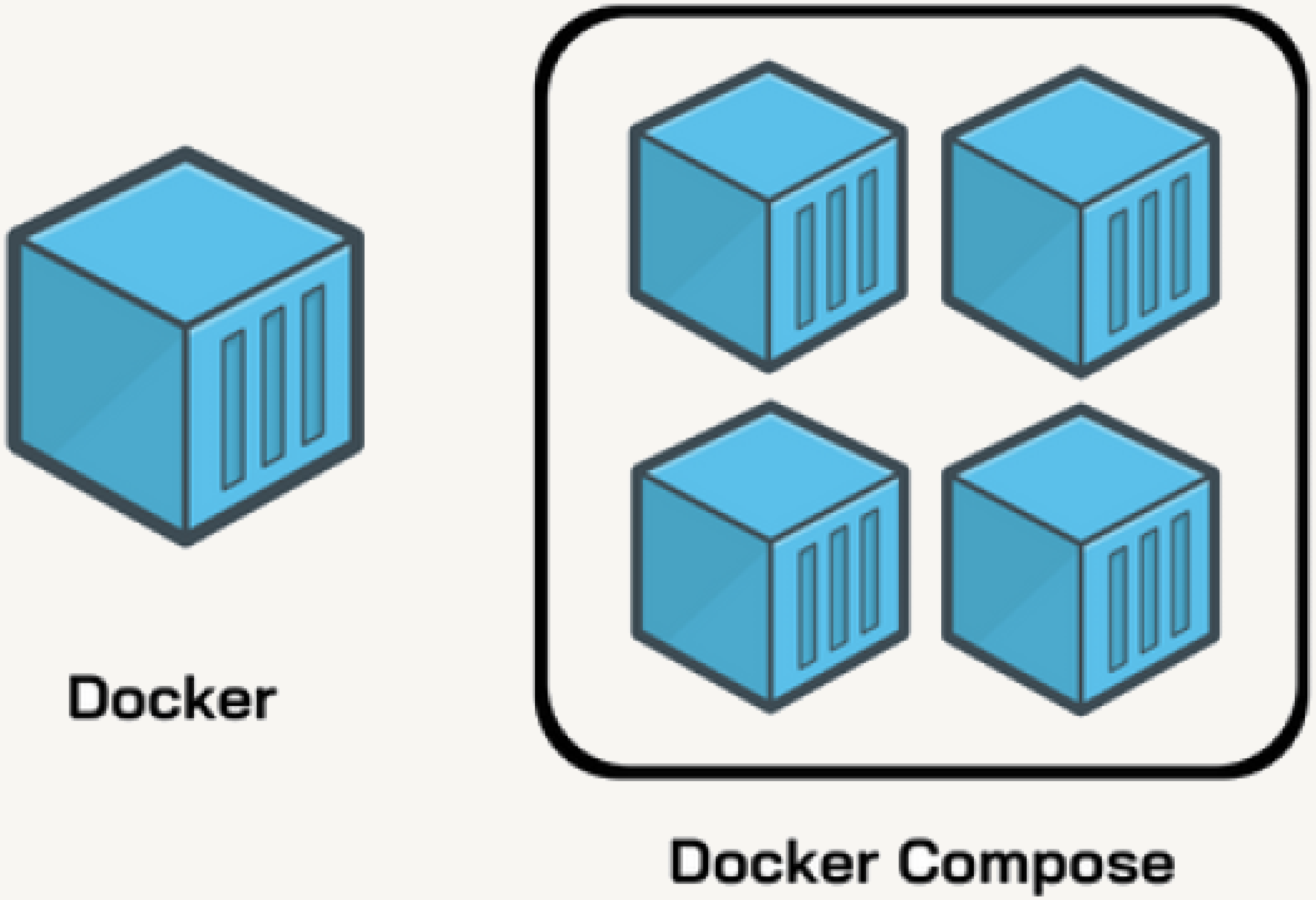
# Part III

Configure the whole project all together



# What is a Docker Compose ?

“Docker Compose is a tool for defining and running multi-container Docker applications.” - ChatGPT



# Your first Docker Compose

## Let’s learn the basic keywords of Docker Compose

**version:** Specifies the Compose file version.

**services:** Defines the services that make up your application.

**image:** Specifies the Docker image to use for a service.

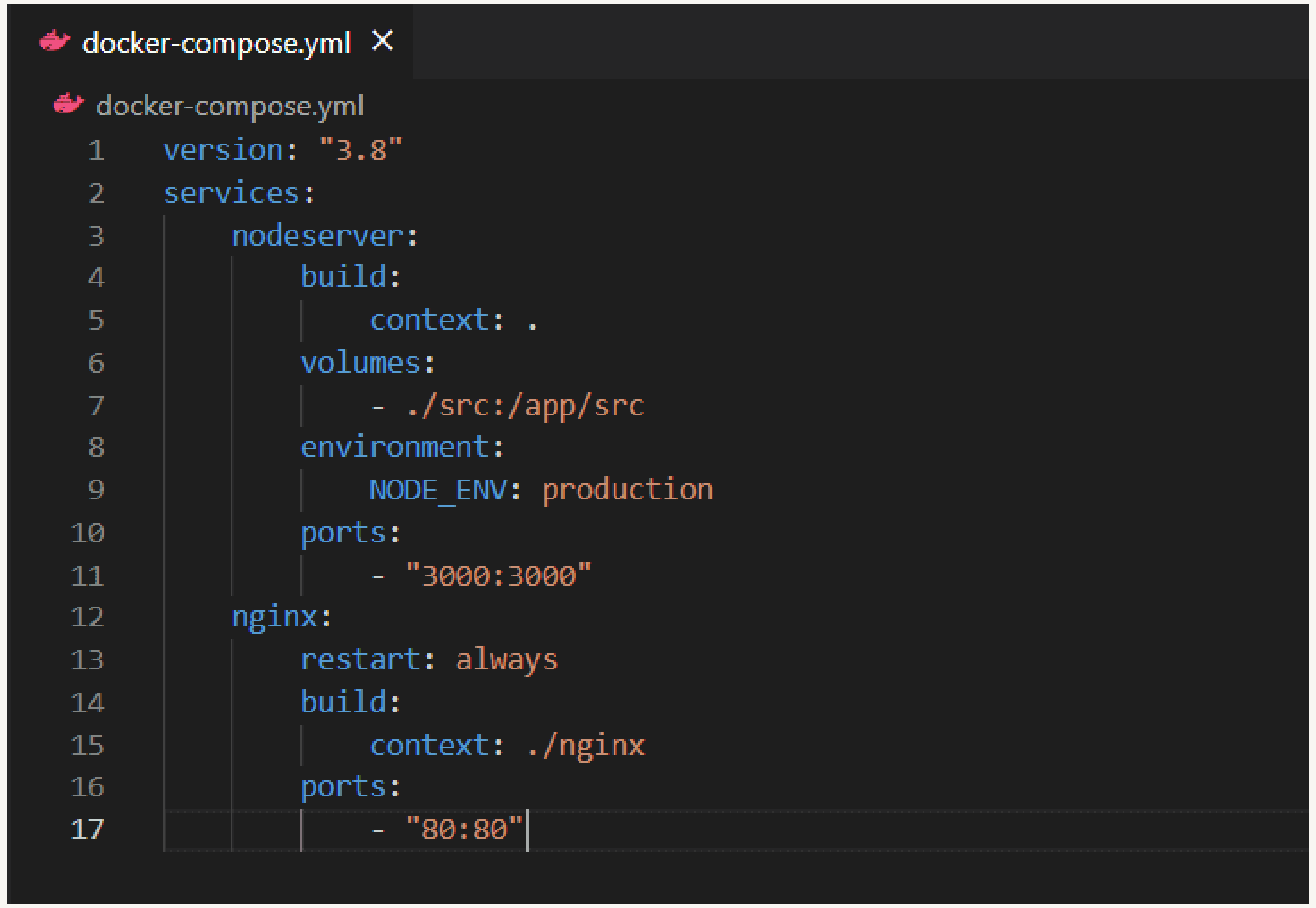
**build:** Instead of specifying an image, you can use the build directive to build an image from a Dockerfile

**ports:** Maps container ports to host ports.

**networks:** Specifies custom networks for services and connects services to these networks.

**environment:** Sets environment variables within the container.

**Here is an exemple of a Docker Compose**



```yaml
version: "3.8"
services:
    nodeserver:
        build:
            context: .
        volumes:
            - ./src:/app/src
        environment:
            NODE_ENV: production
        ports:
            - "3000:3000"
    nginx:
        restart: always
        build:
            context: ./nginx
        ports:
            - "80:80"
```

# Docker Compose commands

## Let’s learn the basic commands of Docker Compose

**docker-compose up:** Create and start all containers defined in the Compose file.

**docker-compose down:** Stop and remove all containers defined in the Compose file.

**docker-compose build:** Build or rebuild the images for the services defined in the Compose file.

**docker-compose scale:** Change the number of containers for a specific service.

# A Docker Compose for your application

**It’s now up to you, reuse and adapt previous Dockerfile with a Docker Compose so it runs your API and Database together**

“I don’t have anymore wisdom to give you.” - Edouard Chhang

# Summary

Containerization Technology

Docker Images and Containers

Isolation and Efficiency

Orchestration and Ecosystem

# Thank you!

Email us at edouard.chhang@epitech.eu
if you have more questions.