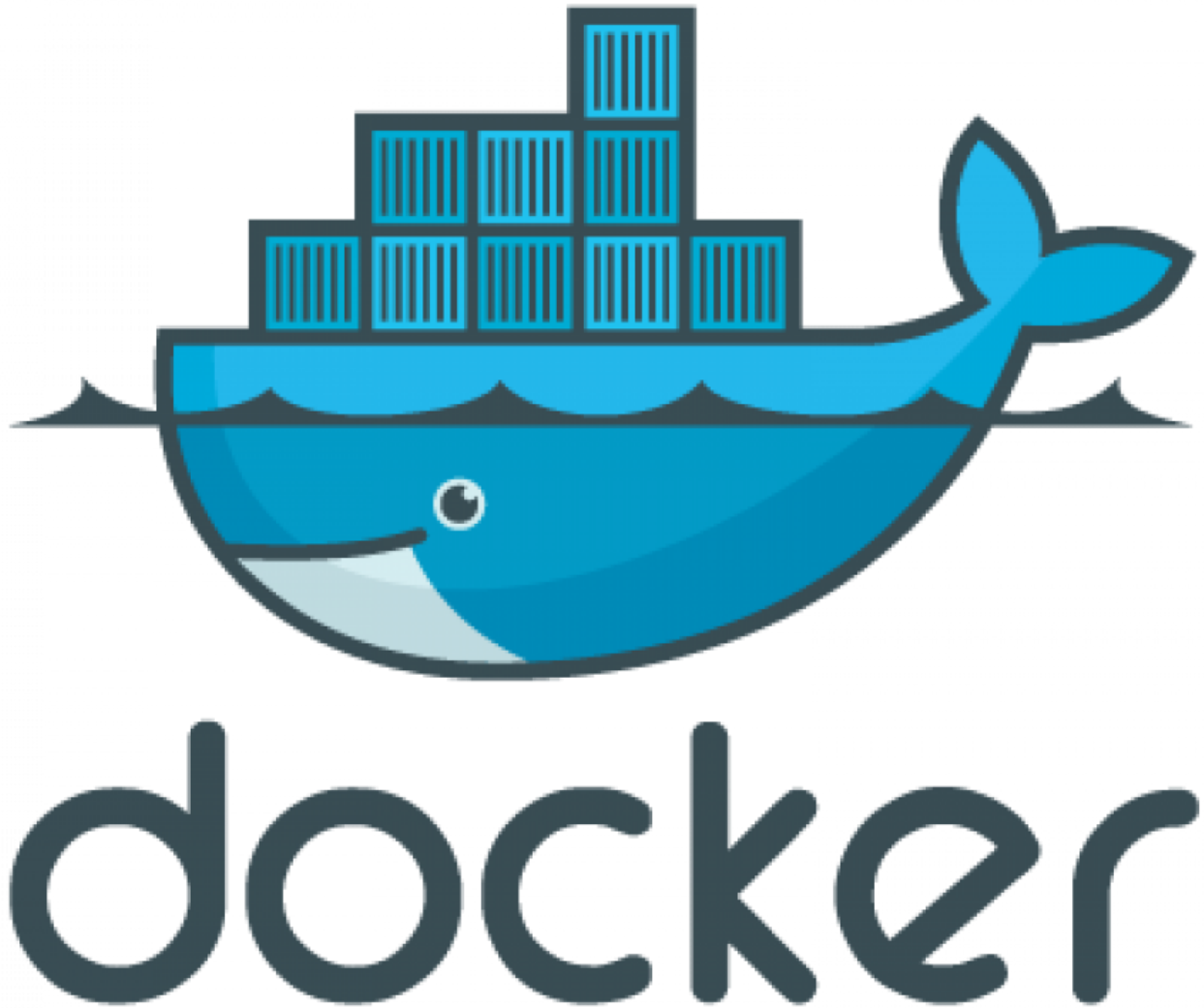

WORKED FINE IN DEV
OPS PROBLEM NOW
memegenerator.net

# DevOps

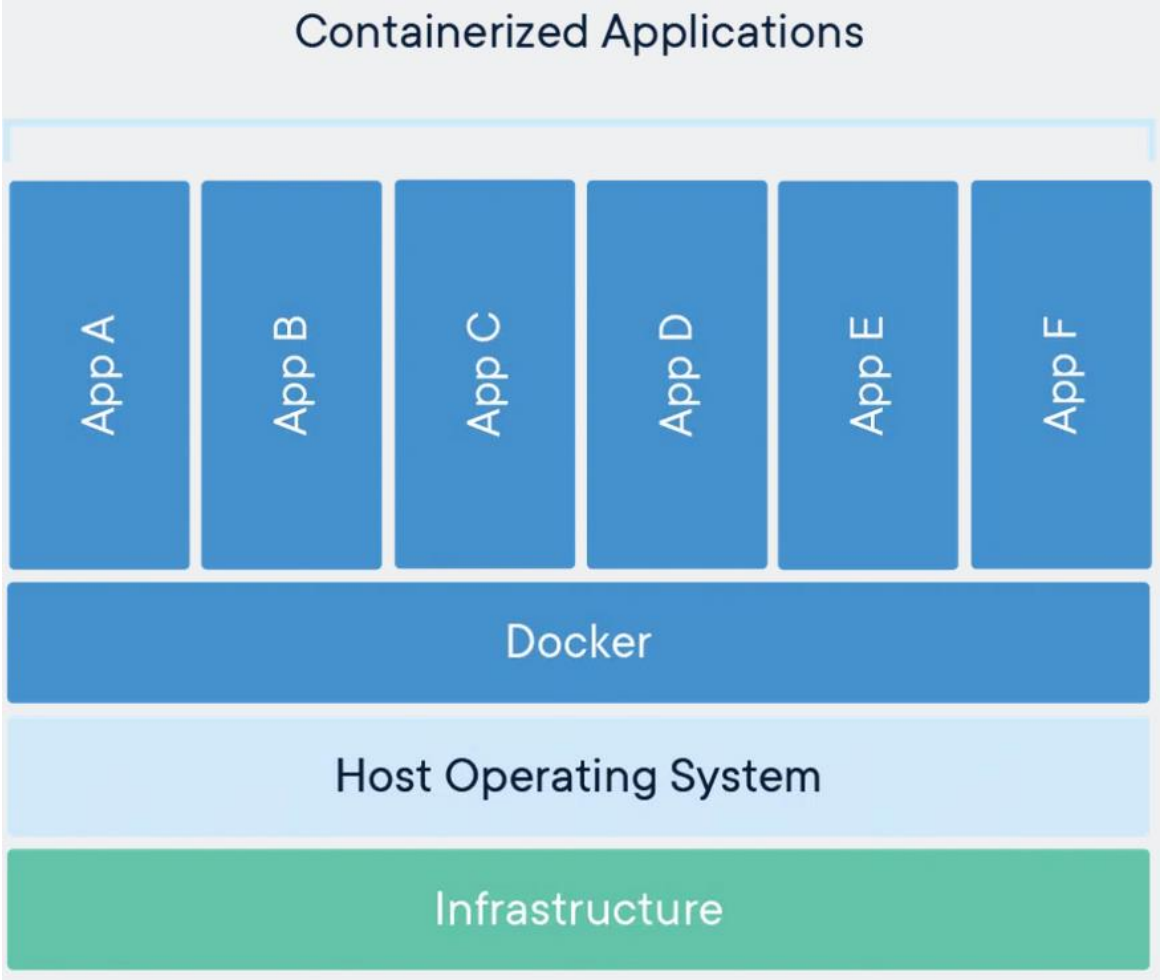
Framework Project 2


docker

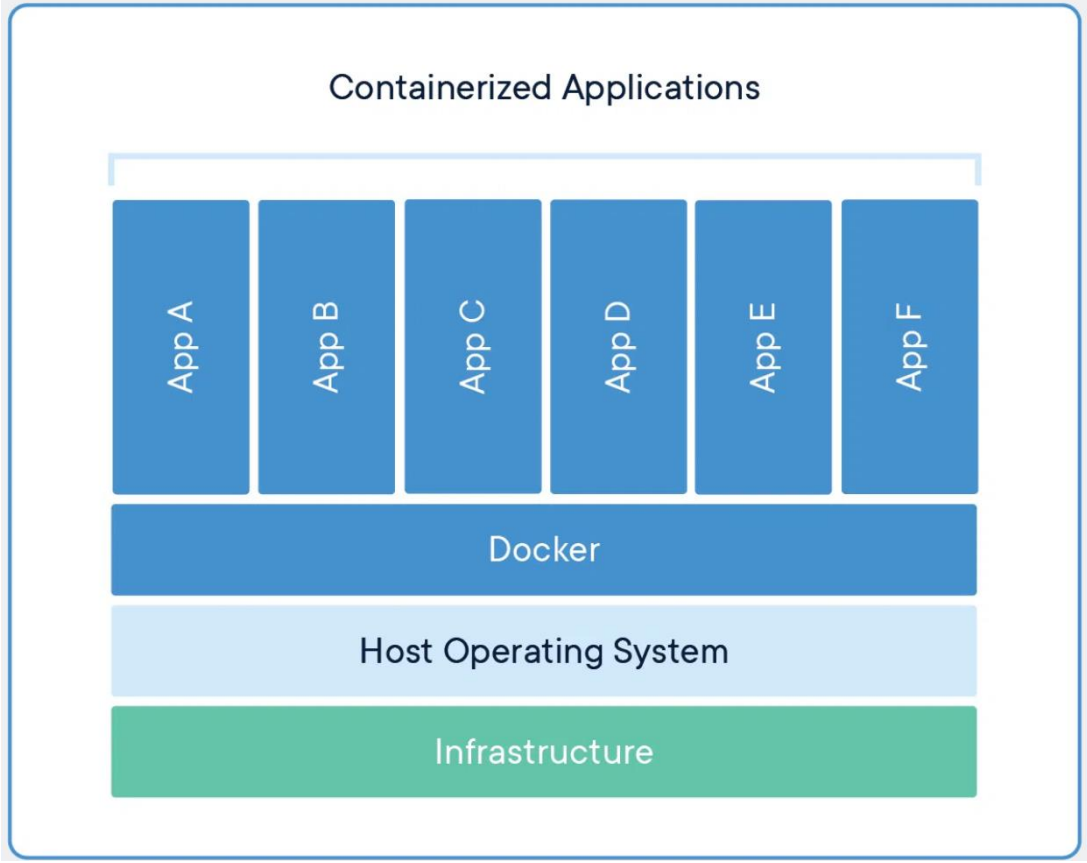
# https://www.docker.com/resources/what-is-a-container/

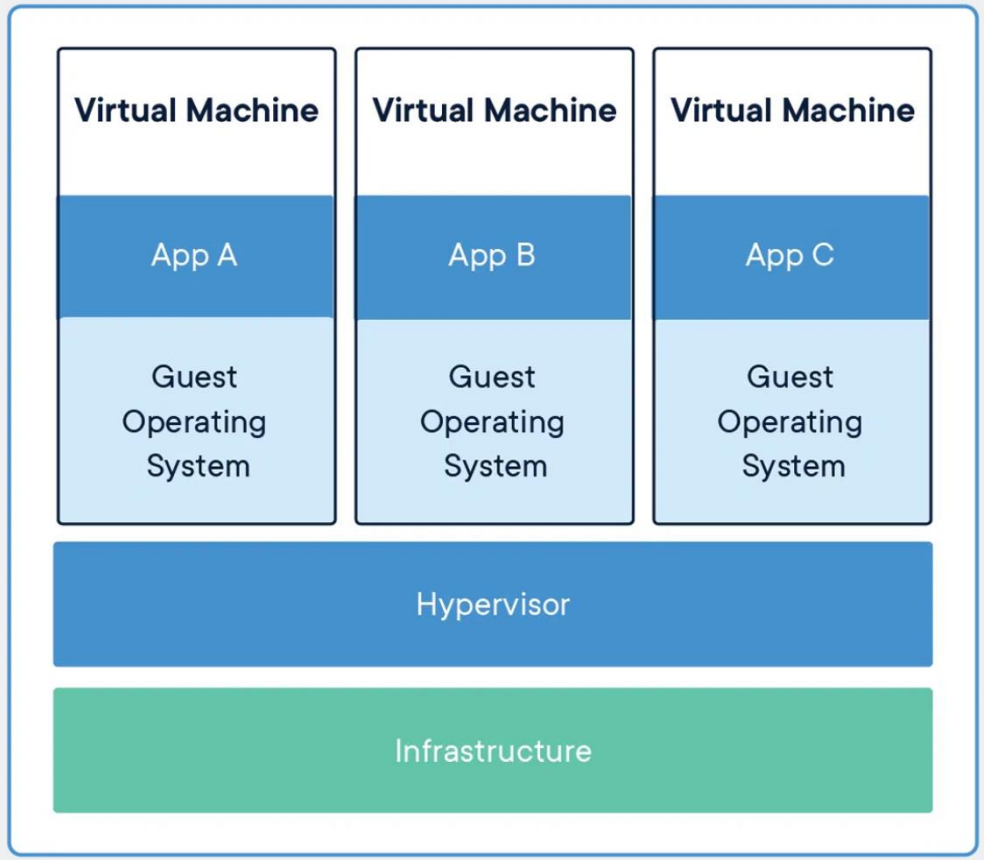
A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another. A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings.

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.



https://www.docker.com/resources/what-is-a-container/




## CONTAINERS

Containers are an abstraction at the app layer that packages code and dependencies together. Multiple containers can run on the same machine and share the OS kernel with other containers, each running as isolated processes in user space. Containers take up less space than VMs (container images are typically tens of MBs in size), can handle more applications and require fewer VMs and Operating systems.

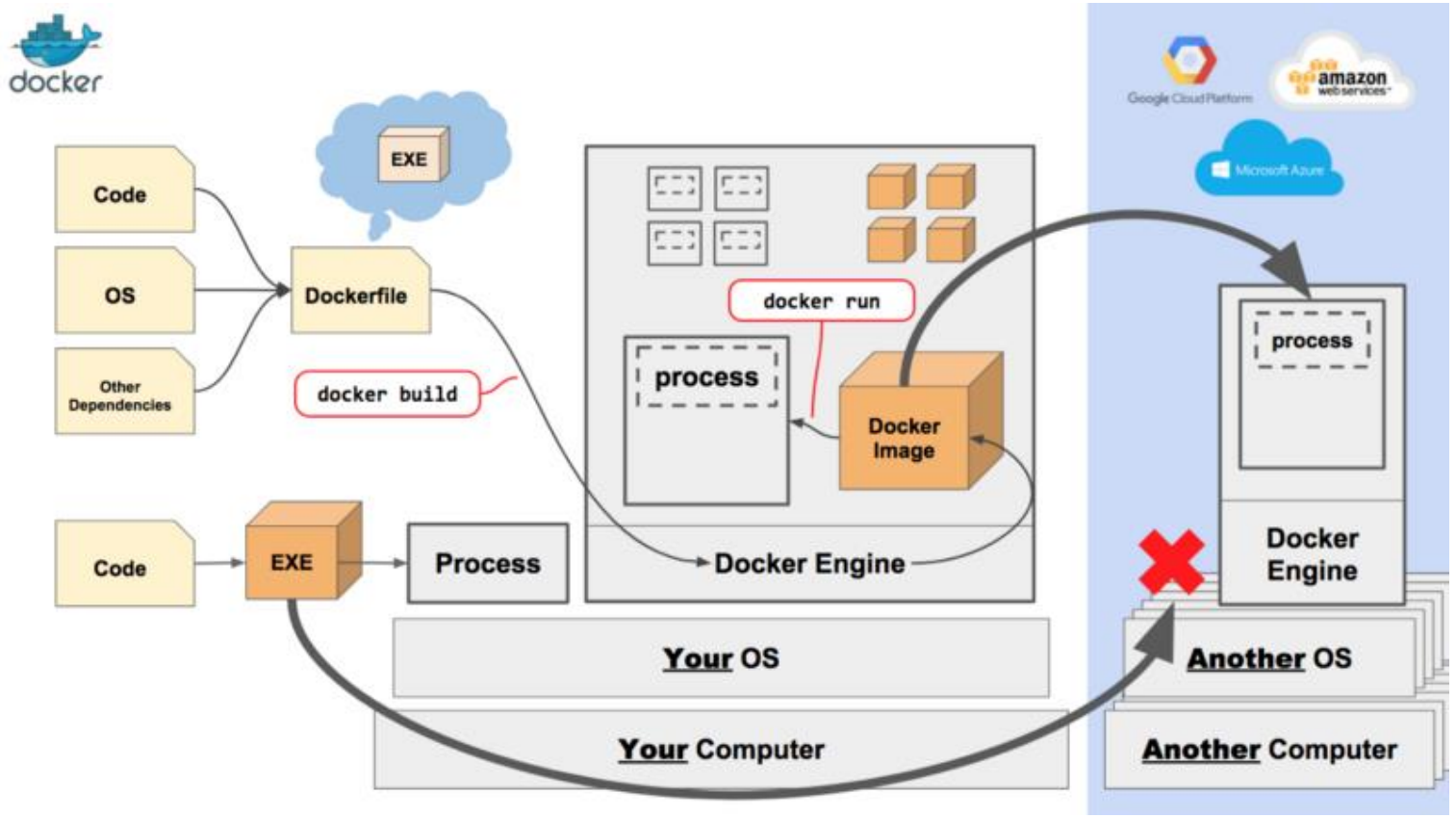
## VIRTUAL MACHINES

Virtual machines (VMs) are an abstraction of physical hardware turning one server into many servers. The hypervisor allows multiple VMs to run on a single machine. Each VM includes a full copy of an operating system, the application, necessary binaries and libraries – taking up tens of GBs. VMs can also be slow to boot.

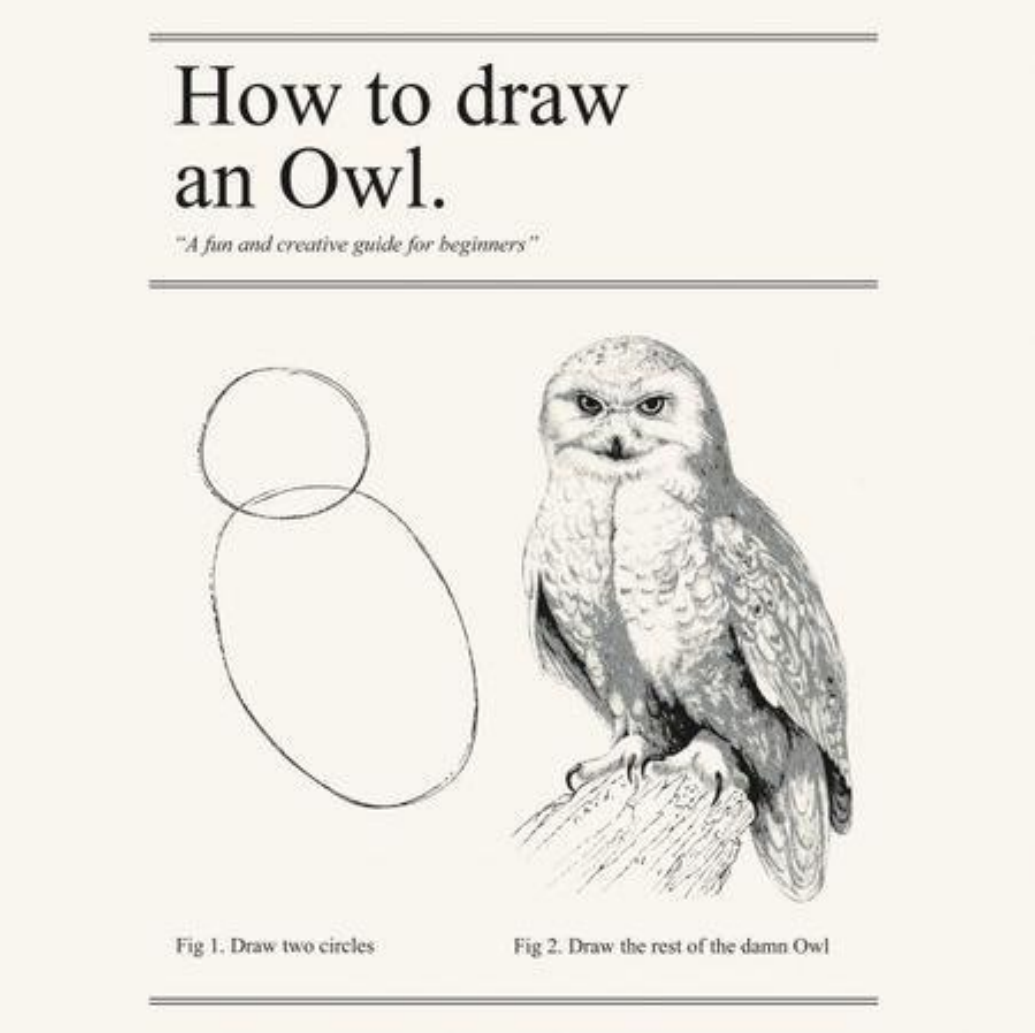
# Basically

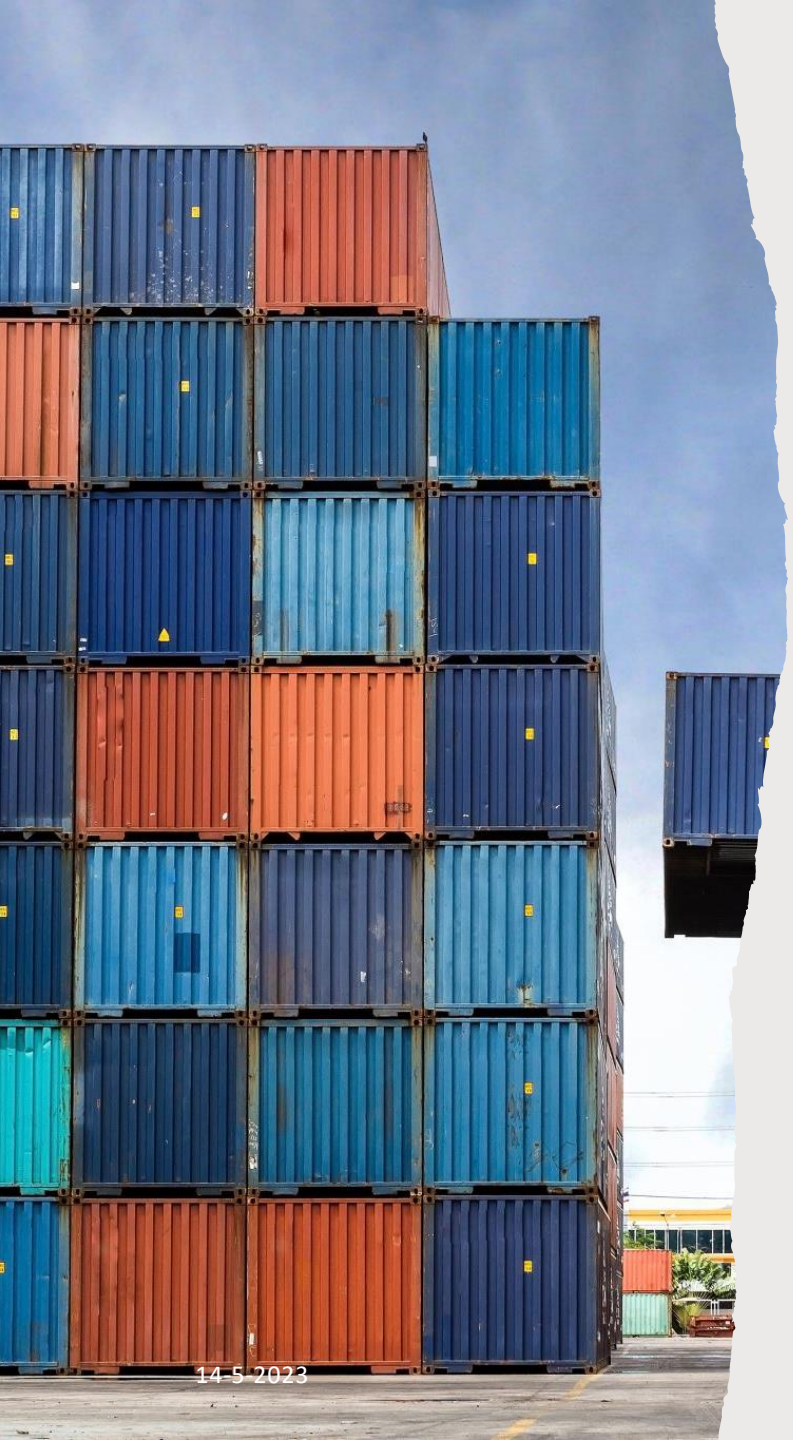
Docker is awesome

# Docker tutorials



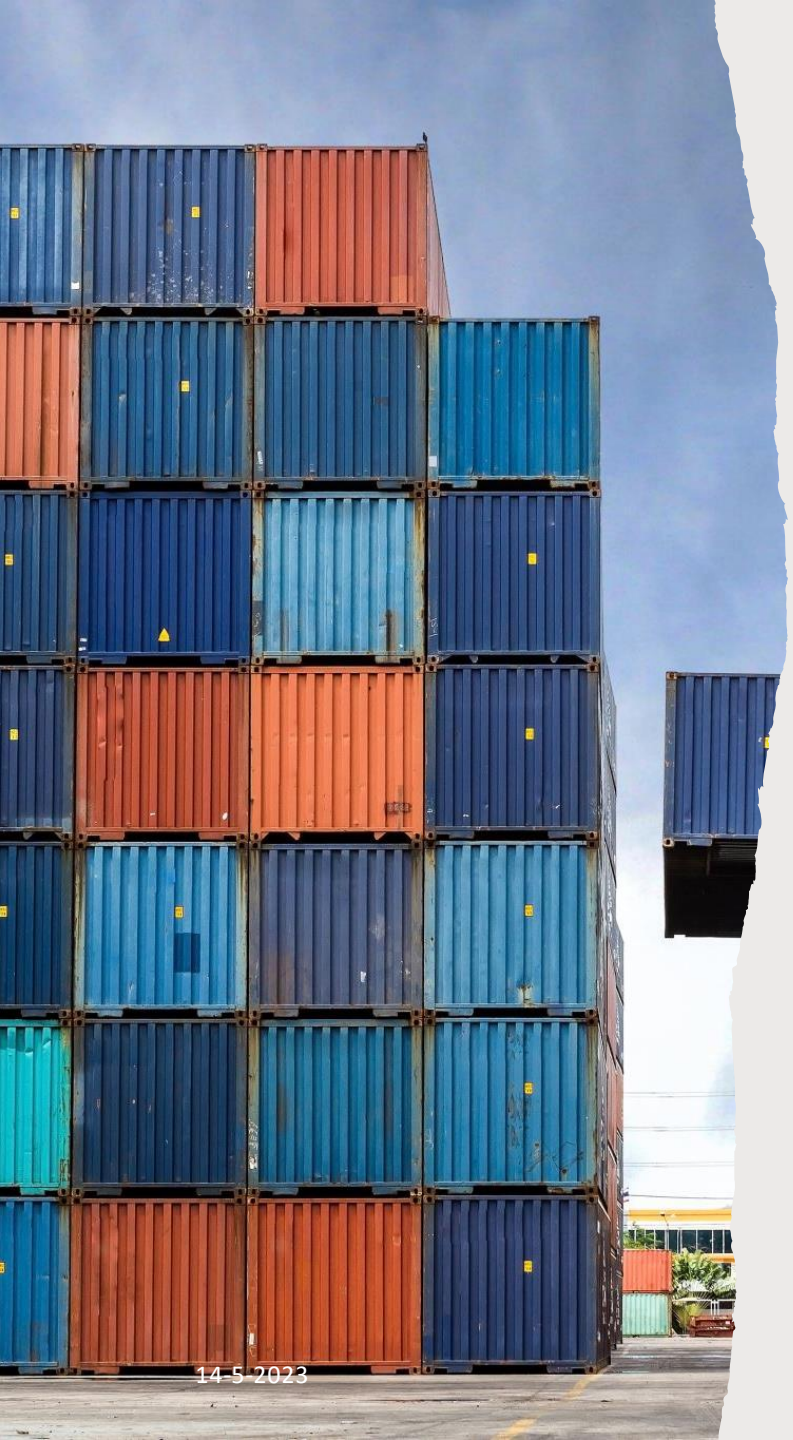
# They feel like this





# Docker

- Containers
- Images
- Dockerfiles
- Compose
- Kubernetes (k8s)
- Orchestration



# Docker

| | |
|---|---|
| • Containers | Last lecture + today |
| • Images | Today |
| • Dockerfiles | Today |
| • Compose | Next lecture |
| • Kubernetes (k8s) | Won’t do |
| • Orchestration | Won’t do |

# Today

- What is a Docker Image?
- What is a Docker container?
- What is a Dockerfile?
- How do you use all of that?

# What did we do last lecture?

- Processes
- Namespaces

# Exercise for last sprint

- Install Docker for your OS
- Watch https://www.youtube.com/watch?v=-YnMr1lj4Z8
- (optional) Find https://github.com/HZ-HBO-ICT/docker-course, read and execute *Lecture 1* again

# Remember

A Docker container is a (group of) process(es) that run in a seperated namespace

They use your system’s kernel, but are isolated from all other processes that are not in the same namespace

# Jargon

- Images
- Containers
- Dockerfile

# Jargon <3 OOP

**Docker**

- Images
- Containers
- Dockerfiles

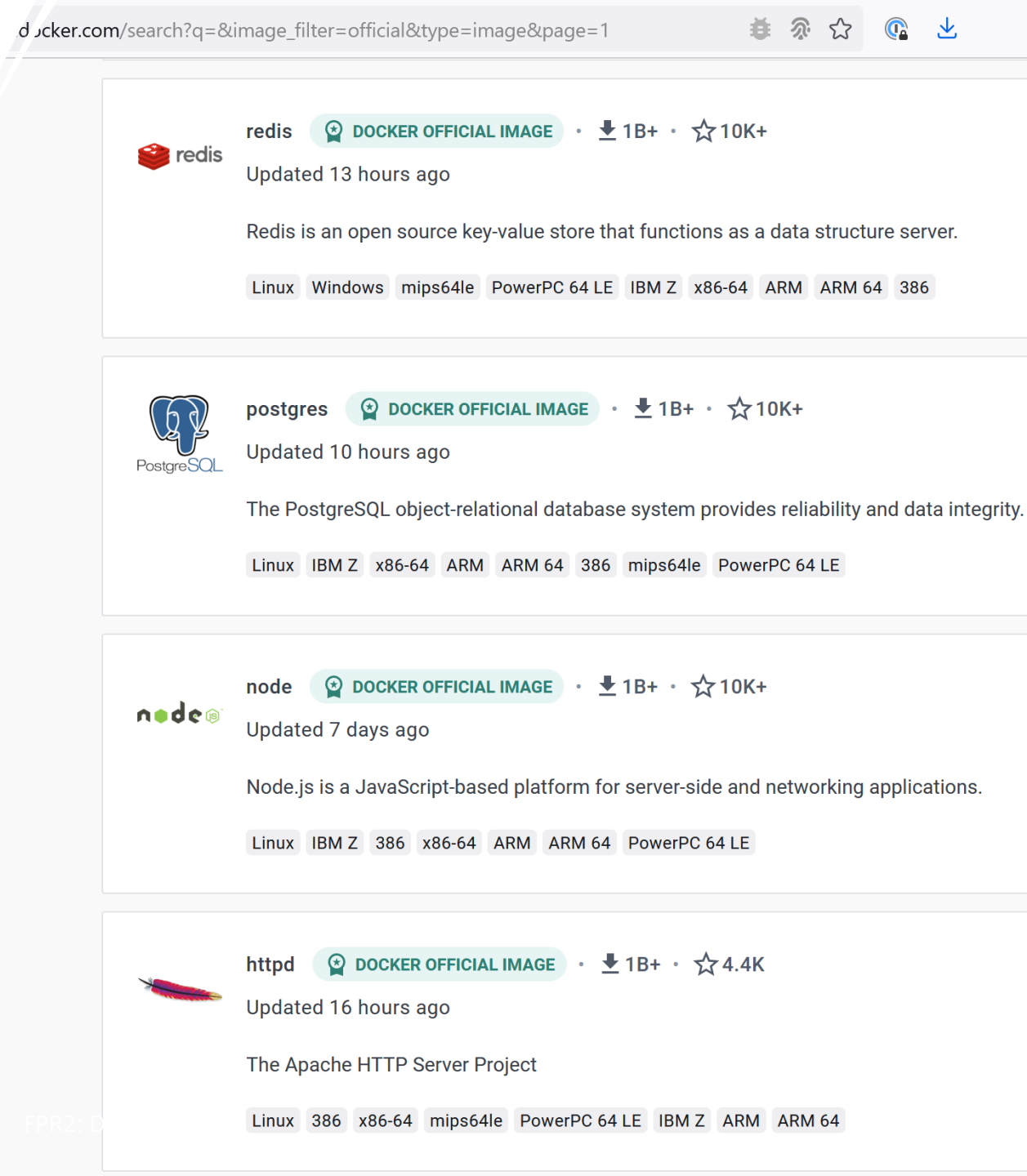
**OOP**

- Classes
- Objects in runtime
- The source code files

# Docker image

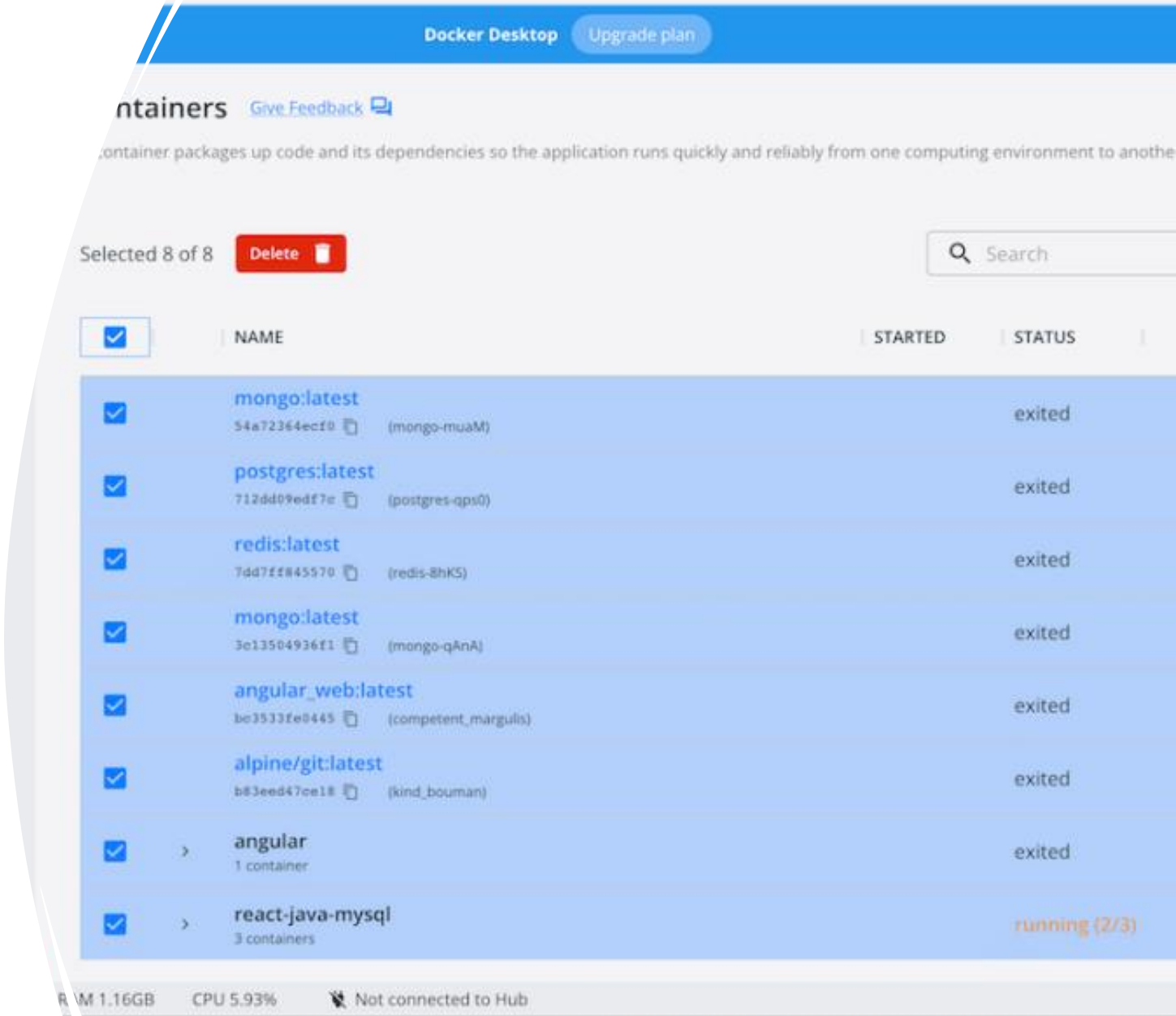
- Pre-built blueprint for a container
- Usually an application or runtime

- Download one and then run as many containers from it as your RAM allows



# Docker container

- Instance of an image

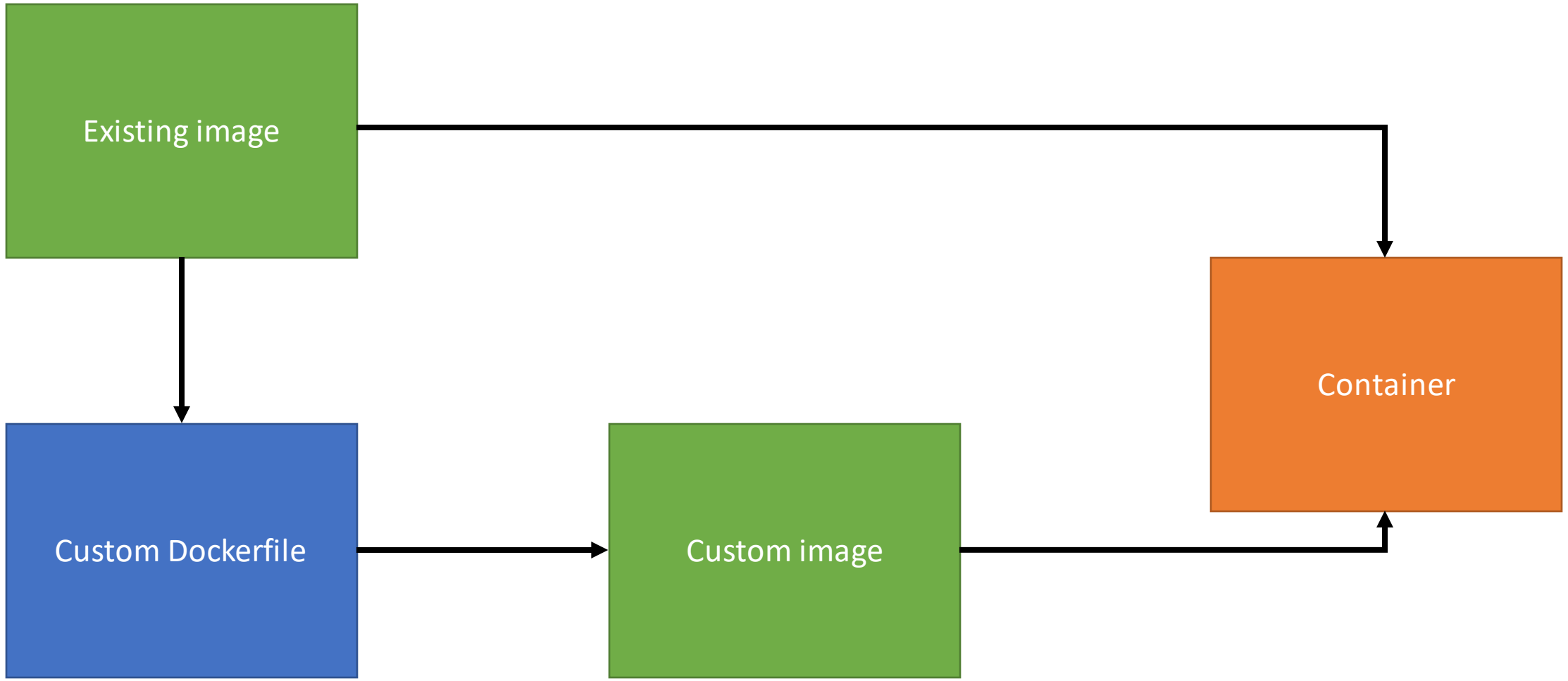


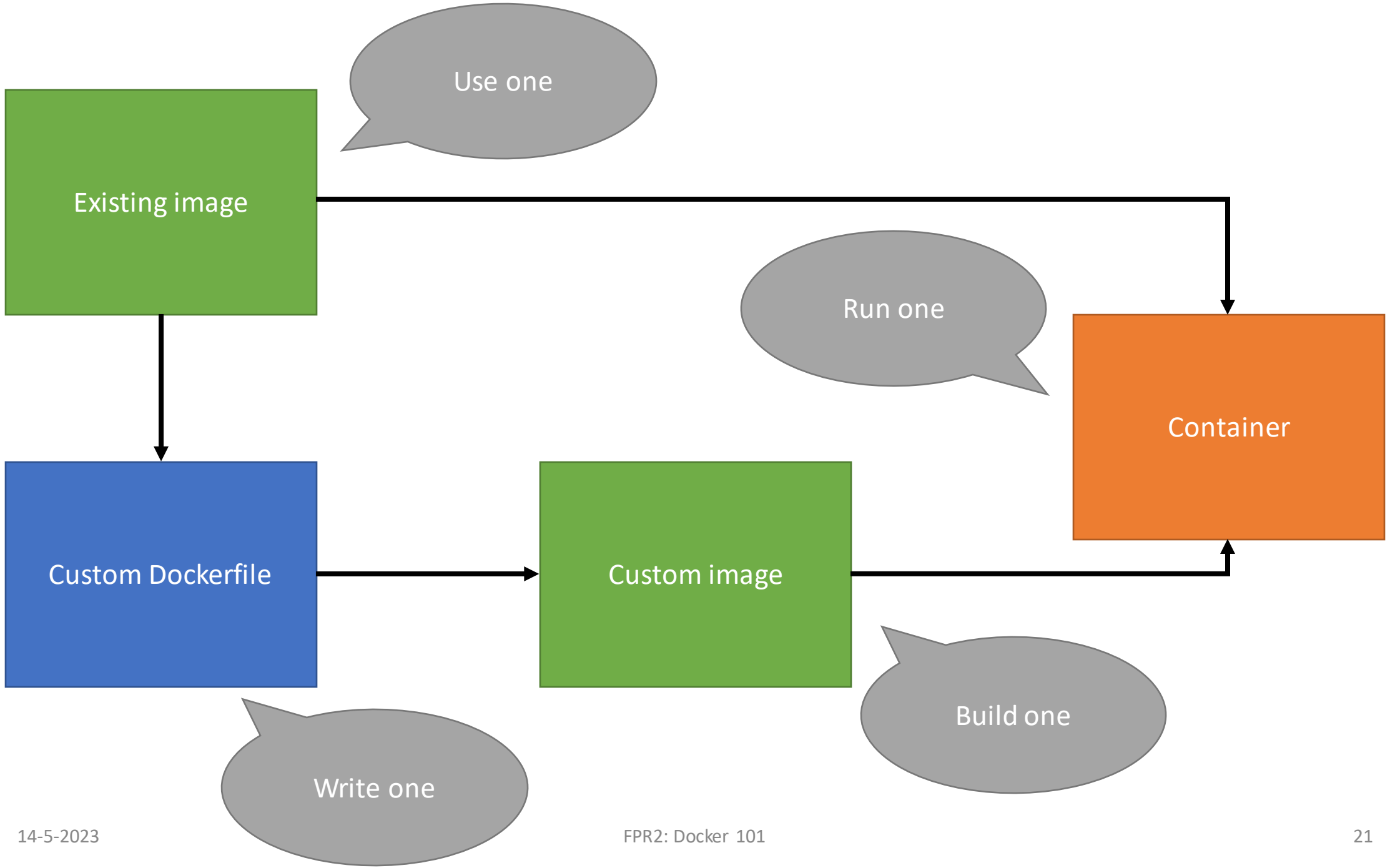
# Dockerfile

- File that contains instructions to create an image
- Can extend existing images (!!!)

Dockerfile - myhttpd:0.1

```
1  # A simple web app served by httpd
2  FROM httpd:2.4
3
4  LABEL AUTHOR=user@example.com
5
6  LABEL VERSION=0.1
7
8  # COPY mypage.html /usr/local/apache2/htdocs/mypage.html
9  # WORKDIR /usr/local/apache2
10
11 COPY mypage.html htdocs/mypage.html
```


Existing image
Container
Custom Dockerfile
Custom image


Use one
Existing image
Run one
Container
Custom Dockerfile
Custom image
Build one
Write one


GOOD LUCK


THE BEATLES
ALL TOGETHER NOW
APPLE
STEREO

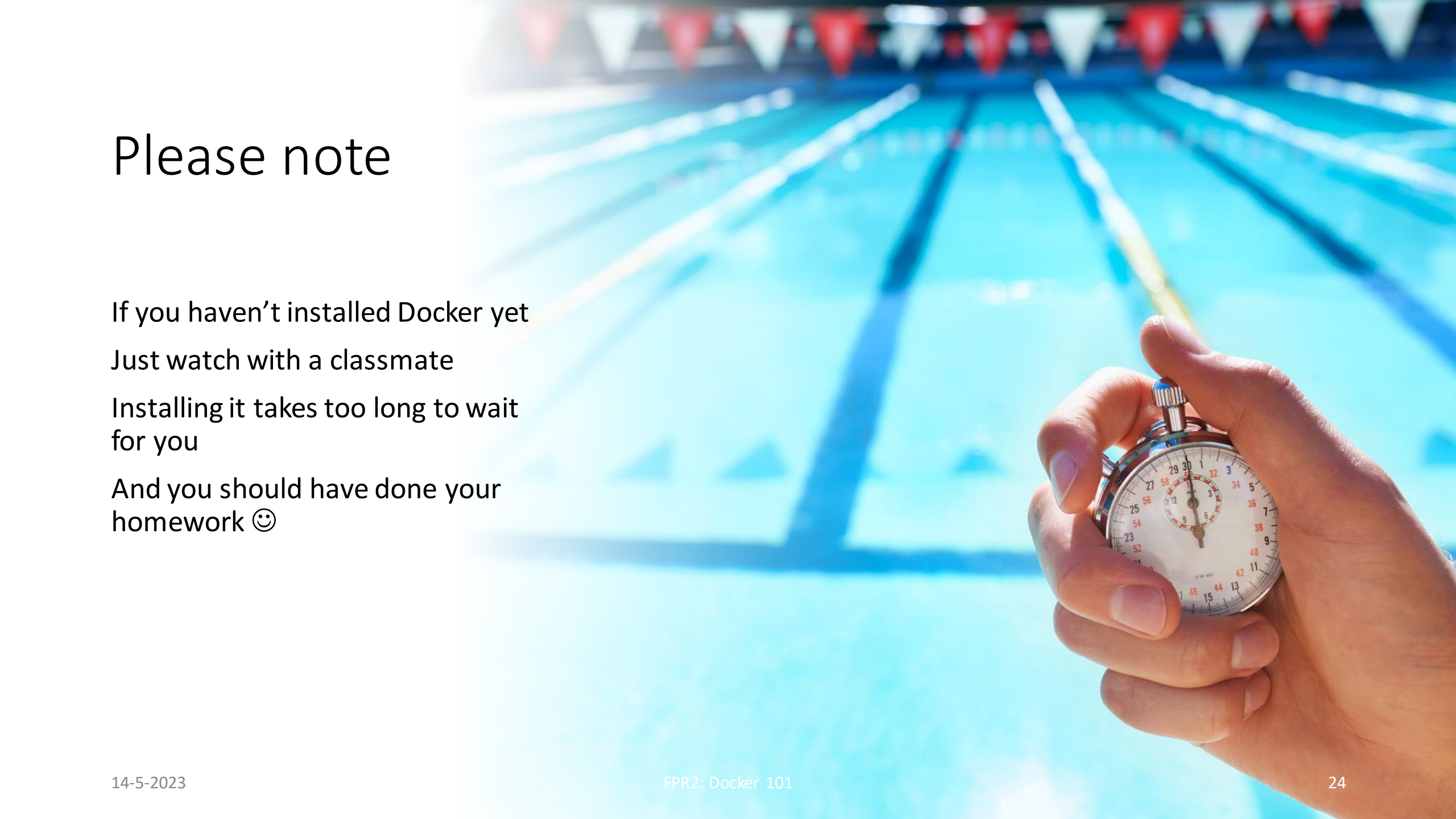
# Please note

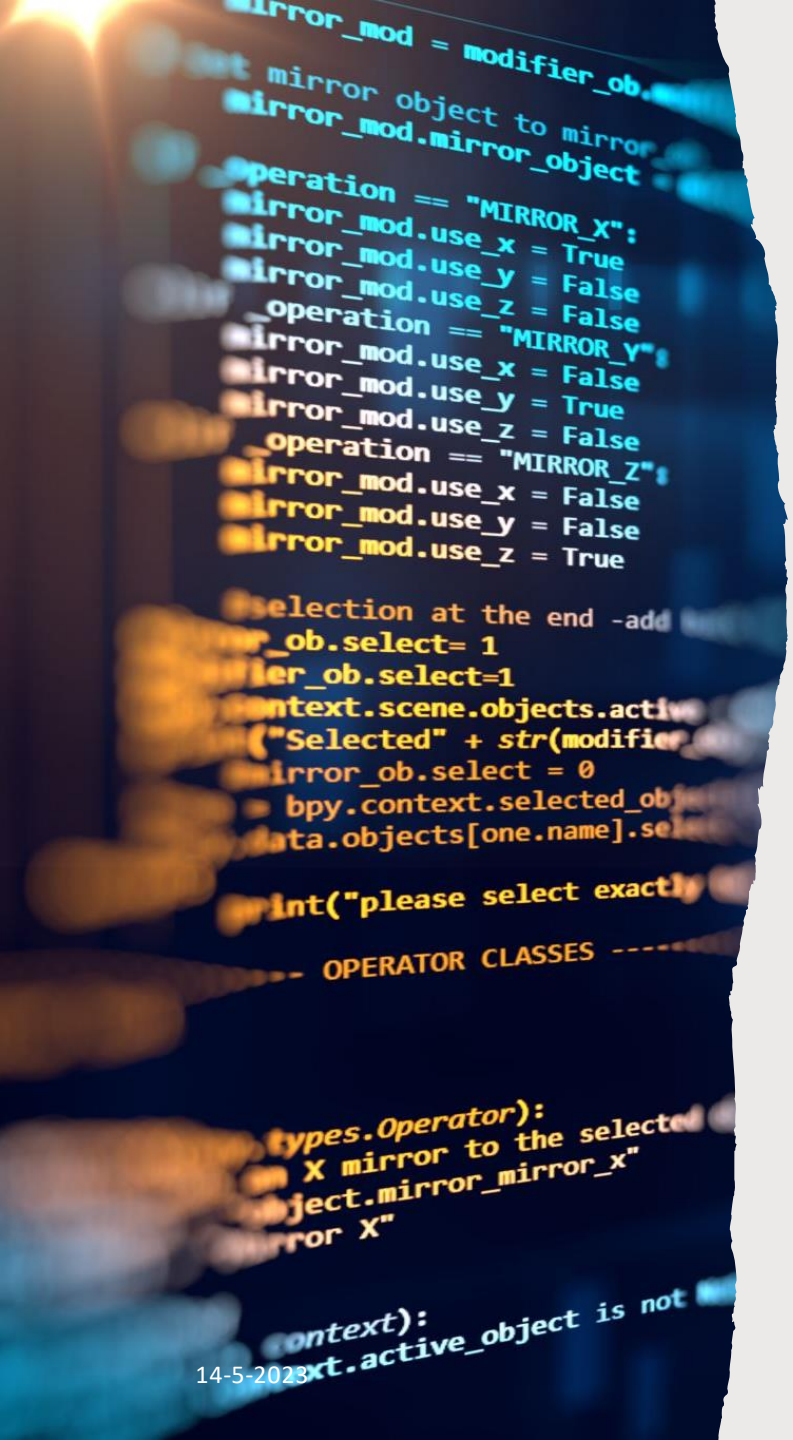
If you haven’t installed Docker yet

Just watch with a classmate

Installing it takes too long to wait for you

And you should have done your homework ☺






# Goal

**Part 1**: create a Dockerfile that runs simple php scripts

**Part 2**: create a Dockerfile that runs simple php scripts in combination with a database

# Create a folder

# Create 2 files:

1. index.php
2. Dockerfile (no extension, with Capital!)

Part 1 – Step 2

Copy these contents to index.php

```php
<?php

echo "<h1>Hello world!</h1>";
```

Run the file. What happens?
(No XAMPP! Just run the file in your browser)

Part 1 – Step 3

# Copy these contents to Dockerfile

```
1   # Extend from latest greatest PHP Apache image
2   FROM php:apache
3
4   # Copy files into the Apache folder of the container
5   COPY . /var/www/html
```

## Part 1 – Step 4

Execute the following commands in terminal (prefer powershell or bash, no cmd)

```
docker build -t example-app ./
docker run example-app
```

```
# docker --> run docker binary
# build --> build an image from a Dockerfile
# -t example-app --> tag the image with a name. Here "example-app"
# ./ --> build from the files in the current directory
docker build -t example-app ./

# docker --> run docker binary
# run --> create a container from an image and run it
# example-app --> use the image named "example-app"
docker run example-app
```

# Part 1 – Step 5

Turn of XAMPP or other stuff interfering

Browse to IP address listed in terminal

# Part 1 – Step 6

Let index.php print something else

Refresh the browser window

# Part 1 – Step 6

Stop the container with ctrl + c in terminal

Start a new container

```
docker run example-app
```

Part 1 – Step 6

# Refresh the browser window


I DO NOT KNOW
WHAT
IS EVEN HAMPENING.

# Docker fundamentals

Docker images are classes

Containers are objects at runtime

We edited the source code files but never "compiled" (built) it to a class!


DOES THIS
COUNT AS ANNOYING?

# Comparing production vs development

**Production**

- Stable
- Reliable
- Versioned
- Not updated too often

**Development**

- Breaks all the time
- Buggy as hell
- Branches might work
- Updated all the time

# Volumes

In development envs

Prefer "volumes"

Synchronize files on your computer in realtime with your container

# Remove the COPY statement from the Dockerfile

# Run a new container

```
docker build -t example-app

# Unix-y systems (macOS, WSL, Linux)
docker run -v $PWD:/var/www/html example-app

# Powershell
docker run -v ${PWD}:/var/www/html example-app
```

# Part 1 – Step 7

```
# docker ⟶ run the docker binary
# run ⟶ create a container from an image and run it
# -v ⟶ create a volume
# $PWD:/var/www/html ⟶ sync the present working directory with /var/www/html
# example-app ⟶ use the image named "example-app"
docker run -v $PWD:/var/www/html example-app
```

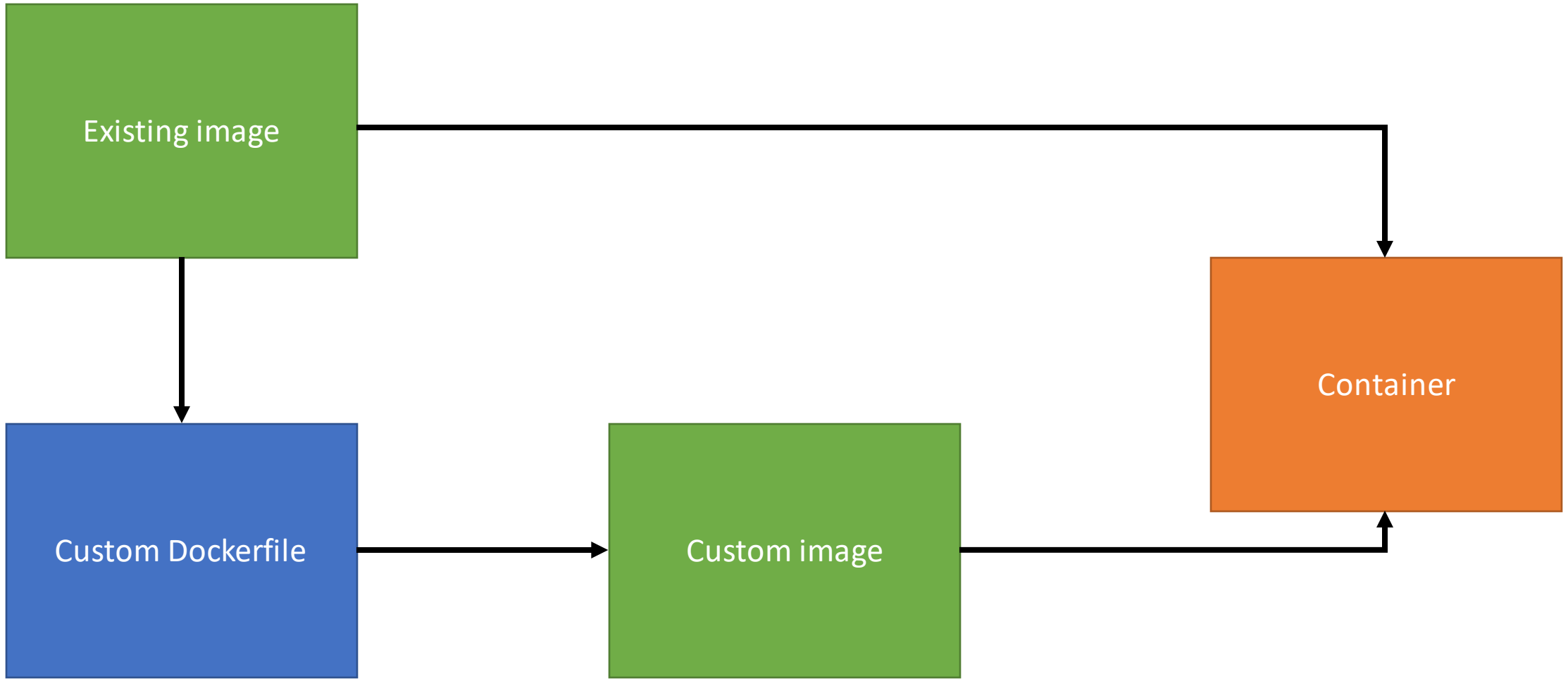
# But my Dockerfile?

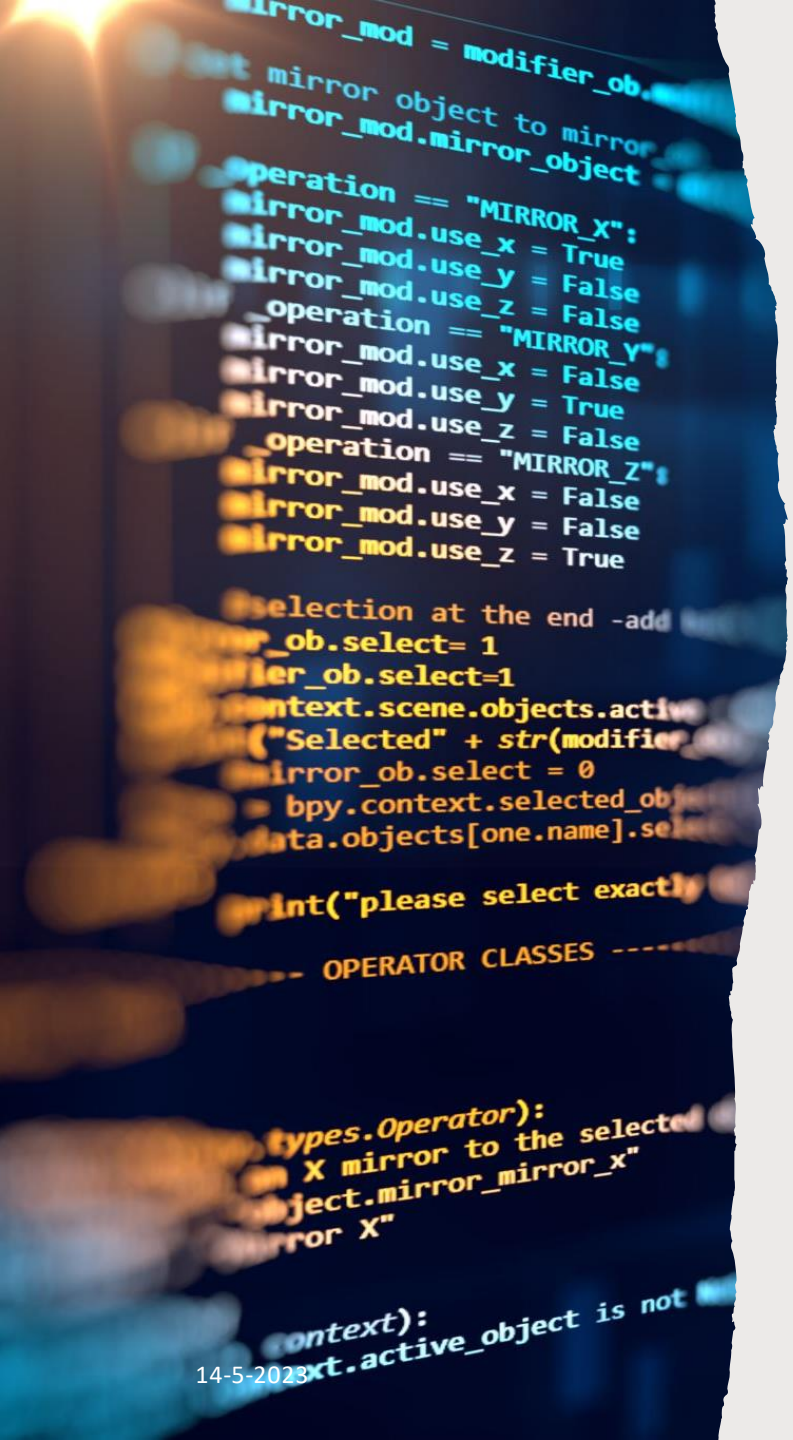
It contains only 1 line:

It imports an existing image

You can choose not to use a Dockerfile in cases like this:

```
# docker ⟶ run the docker binary
# run ⟶ create a container from an image and run it
# -v ⟶ create a volume
# $PWD:/var/www/html ⟶ sync the present working directory with /var/www/html
# php:apache ⟶ use the image named "example-app"
docker run -v $PWD:/var/www/html php:apache
```


Existing image
Container
Custom Dockerfile
Custom image




# Goal

~~**Part 1**: create a Dockerfile that runs simple php scripts~~

**Part 2**: create a Dockerfile that runs simple php scripts in combination with a database

# Part 2 – Step 1 – Copy paste to index.php please!

```php
$servername = "myserver";
$username = "root";
$password = "ActuallyNotThatBadAPassword";

try {
  $conn = new PDO("mysql:host=$servername;dbname=myDB", $username, $password);
  // set the PDO error mode to exception
  $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
  echo "Connected successfully";
} catch(PDOException $e) {
  echo "Connection failed: " . $e->getMessage();
}
```

## Part 2 – Step 2

Create a subfolder "database" with a new Dockerfile

Note that you don't actually have to create it

```
FROM mysql:8
```

Part 2 – Step 2

# Php Dockerfile now requires you to have a Dockerfile!

```
FROM php:apache

RUN docker-php-ext-install pdo pdo_mysql
```

## Part 2 – Step 3

Build a new image for php app

Builda new image for the database

```
docker build -t example-app ./
docker build -t database ./database
```

Part 2 – Step 4

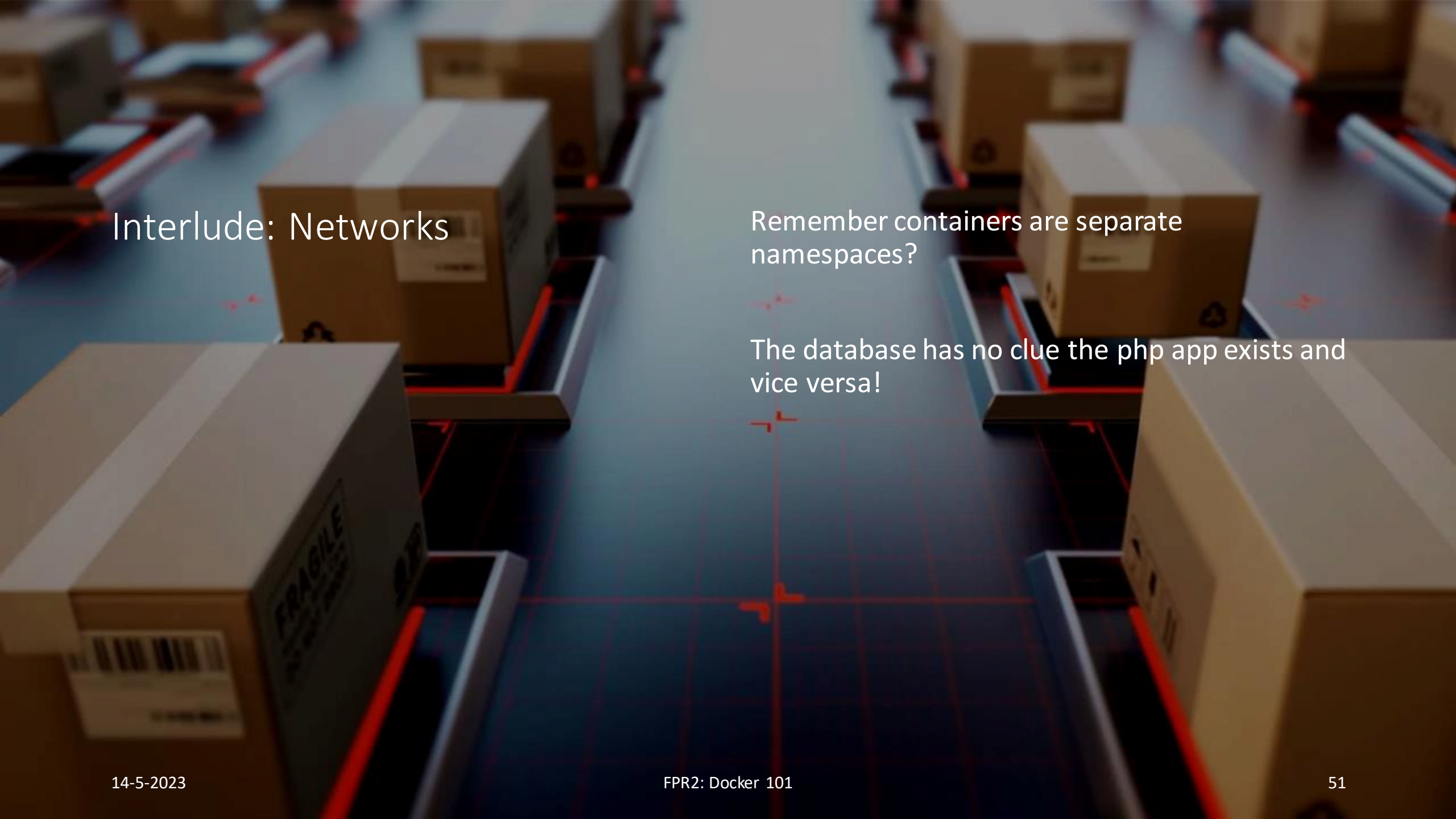
# Run both images as a new container

```
# --name ⟶ name the container so you can easily connect to it later
#        Note that we used the hostname from our script... Wonder why?
# -e MYSQL_ROOT_PASSWORD ⟶ MySQL containers must always have a root password
docker run --name myserver -e MYSQL_ROOT_PASSWORD=ActuallyNotThatBadAPassword database

# Powershell: use ${PWD}
docker run -v $PWD:/var/www/html example-app
```

Part 2 – Step 5

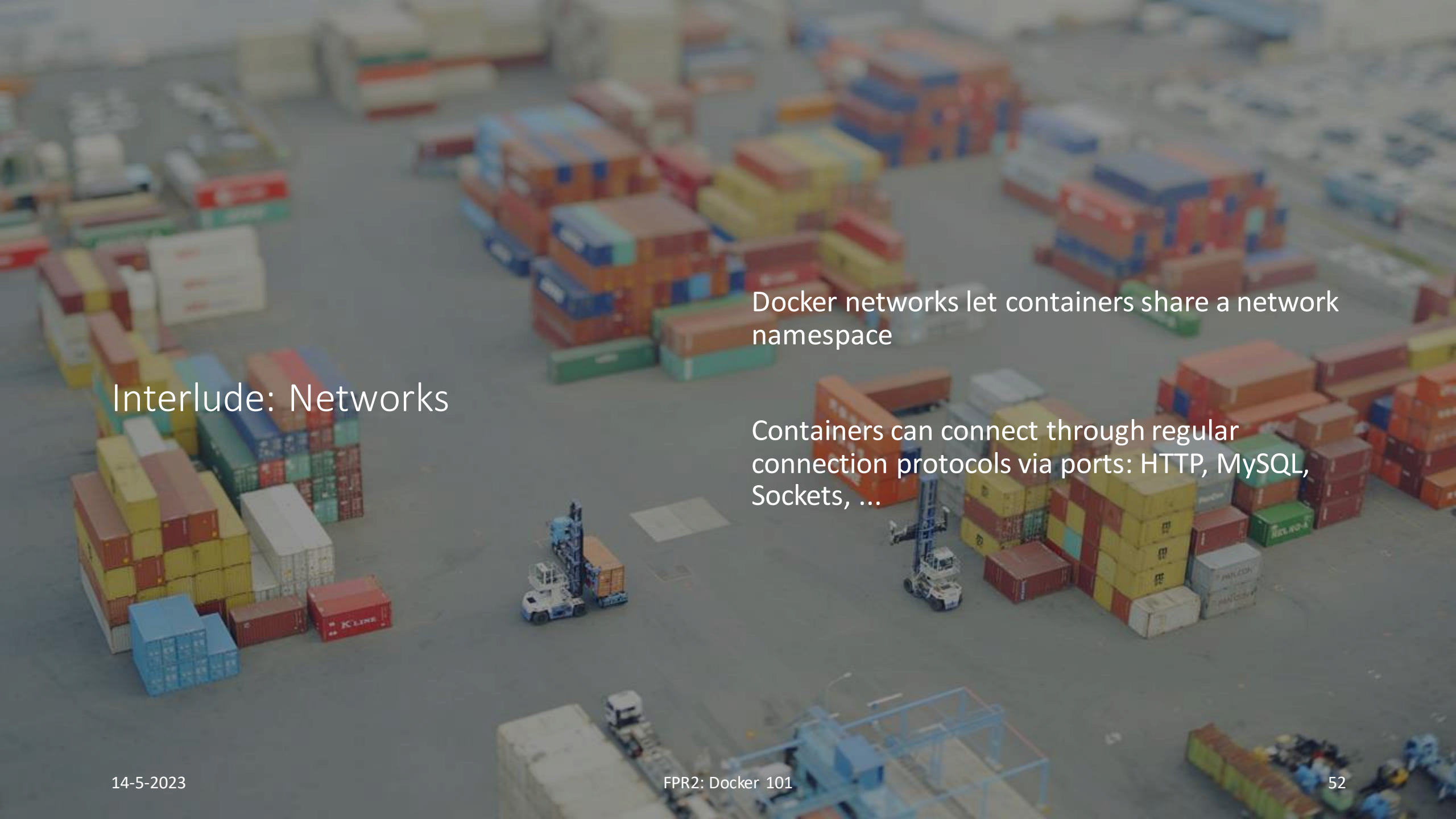
# Browse to IP address




# Interlude: Networks

Remember containers are separate namespaces?

The database has no clue the php app exists and vice versa!



# Interlude: Networks

Docker networks let containers share a network namespace

Containers can connect through regular connection protocols via ports: HTTP, MySQL, Sockets, ...

Part 2 – Step 6

Kill all existing containers

Create network

```
docker system prune
docker network create -d bridge my-network
```

Part 2 – Step 7

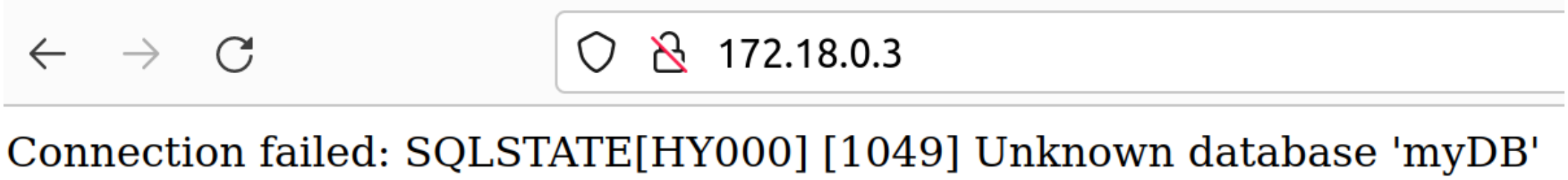
# Start new database and php app containers and connect them to the network

```
docker run --name myserver --network my-network -e MYSQL_ROOT_PASSWORD=ActuallyNotThatBadAPassword database

# # Powershell: use ${PWD}
docker run -v --network my-network $PWD:/var/www/html example-app
```

# Part 2 – Step 8 – Grande Finale

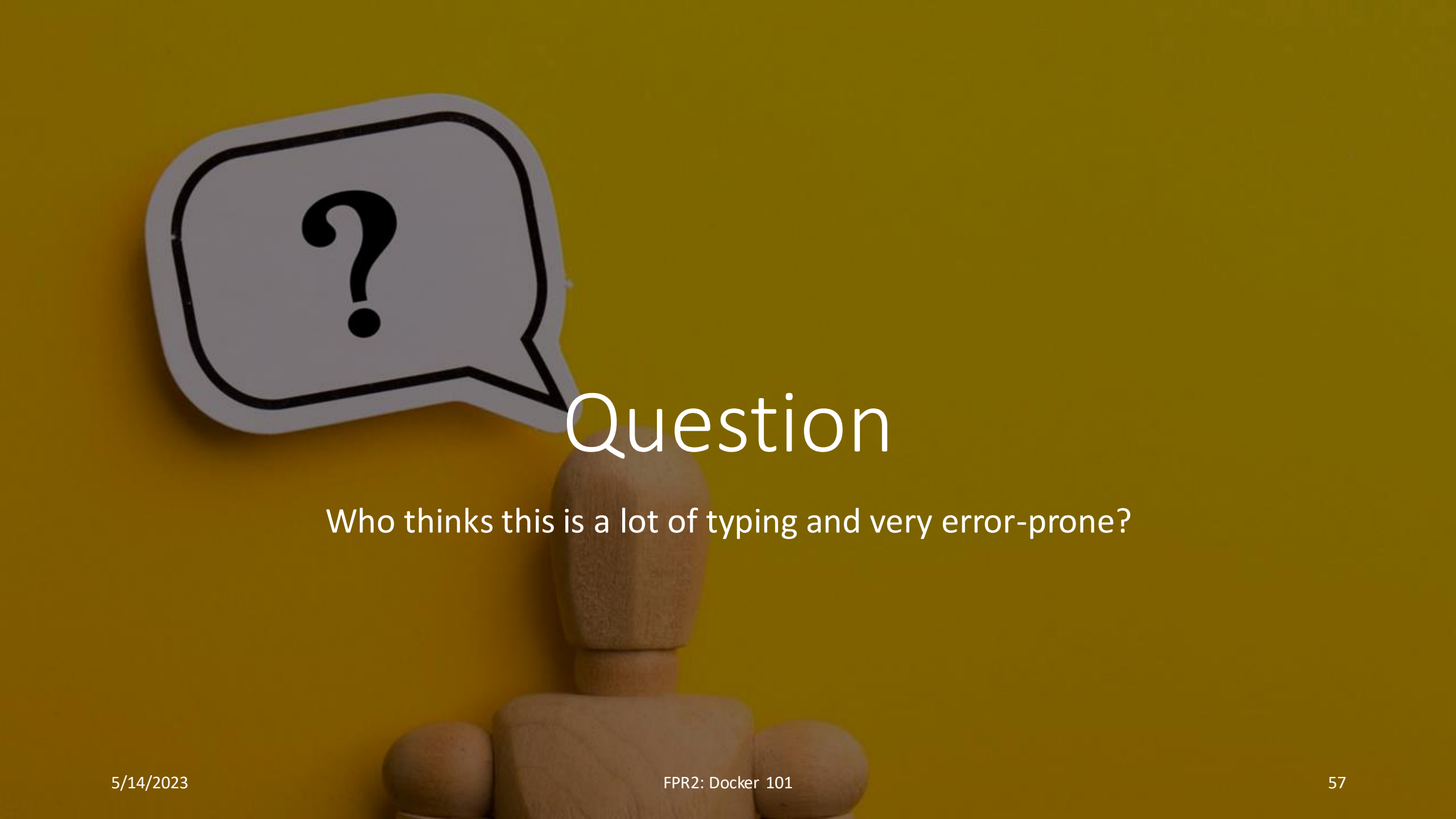
## Browse to IP address




Example is silly, but it shows that you are now actually connecting to the database container. There just isn't a database created in the container yet.

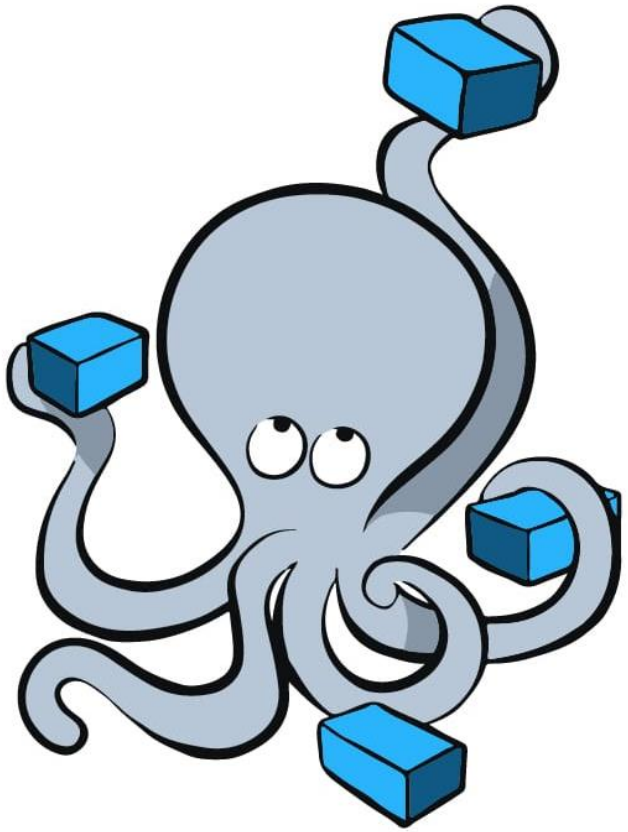
# Question

Who thinks this is quite cool?



# Question

Who thinks this is a lot of typing and very error-prone?



docker Compose

Next lecture solves both problems!

# Exercise for this sprint

- Execute DevOps assignment 1
- It's not meant to be easy
- Spend some time on it (40 points is about 18 hours of work!)

- (optional) Find https://github.com/HZ-HBO-ICT/docker-course, read and execute *Lecture 2* again