Docker Training: Integrating Docker in Web Development

Marlon I. Tayag

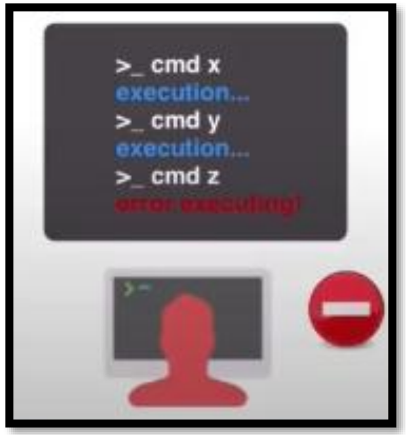
Docker

- Docker is a platform designed to help developers build, deploy, and run applications in containers. A container packages an application and its dependencies, allowing it to run consistently across various computing environments.

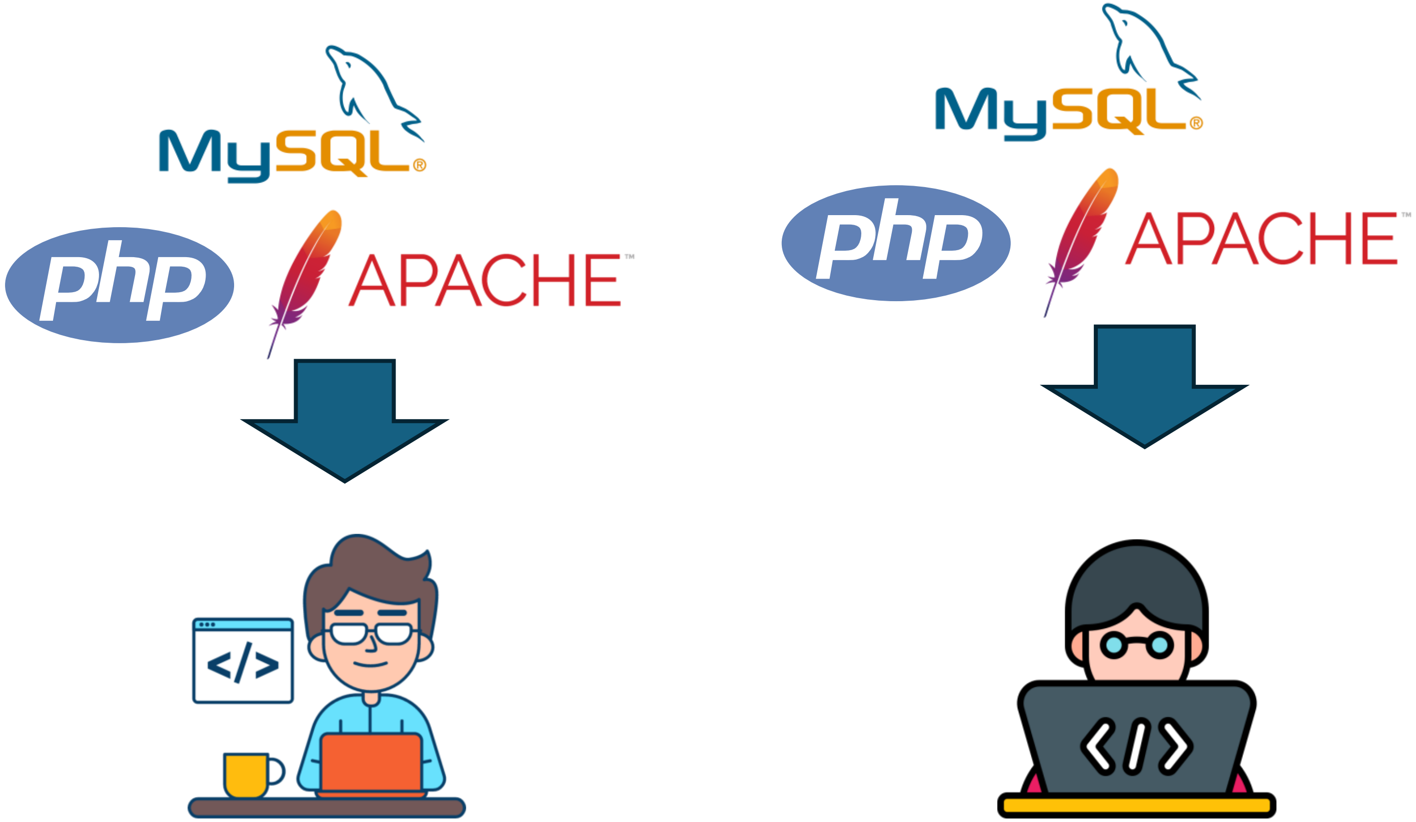
Application Development (Before Container)

- **Installation process different**
  On each OS environment

- **Many steps**
  Something could go wrong



```
>_ cmd x
execution...
>_ cmd y
execution...
>_ cmd z
error executing!
```

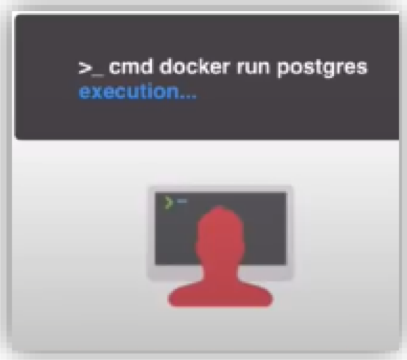


Windows

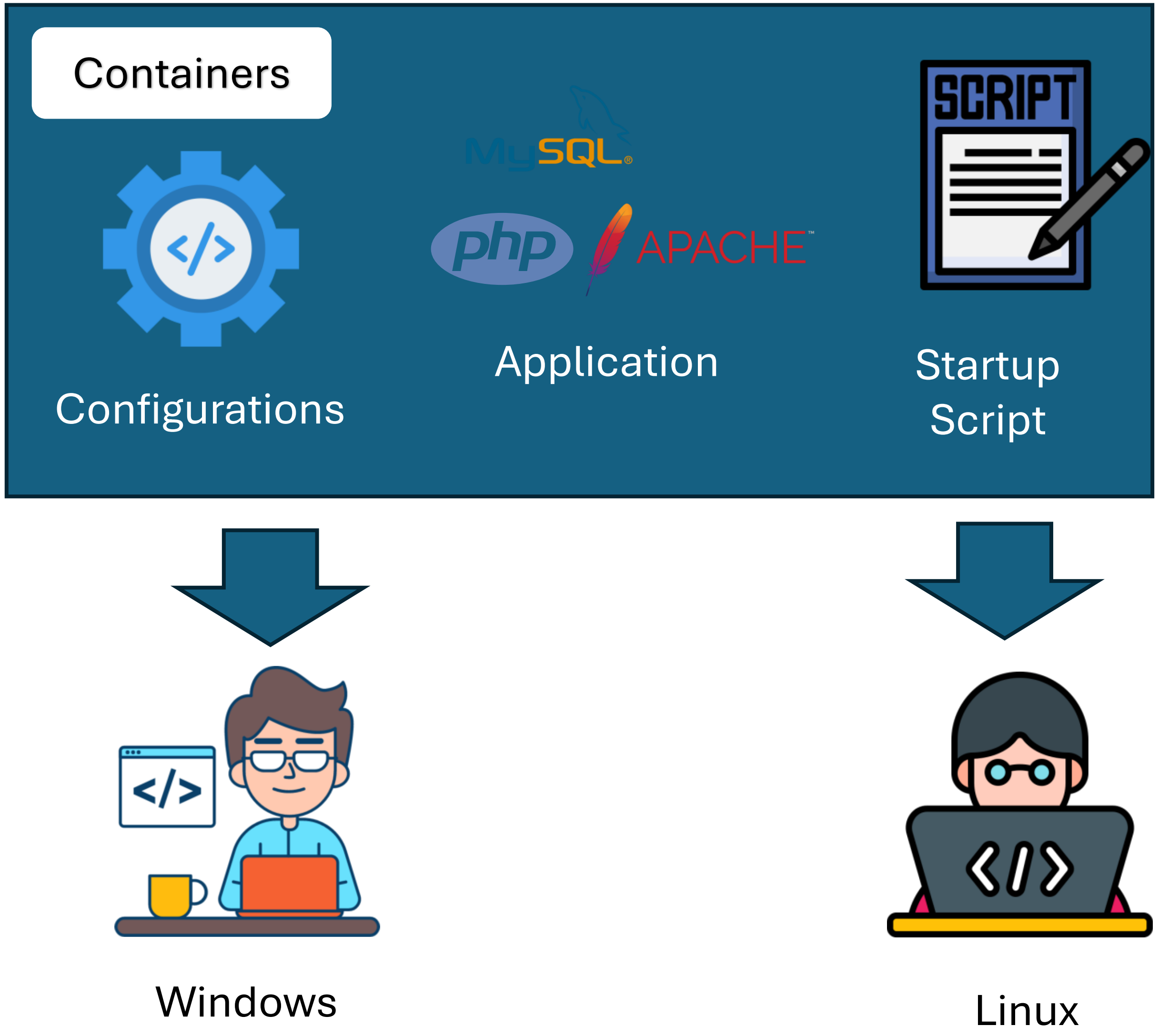
Linux

Application Development (After Containers)

- Own isolated environment
- Packaged with all the needed configuration
- One command to install the app



```
>_ cmd docker run postgres
execution...
```



Containers

Configurations

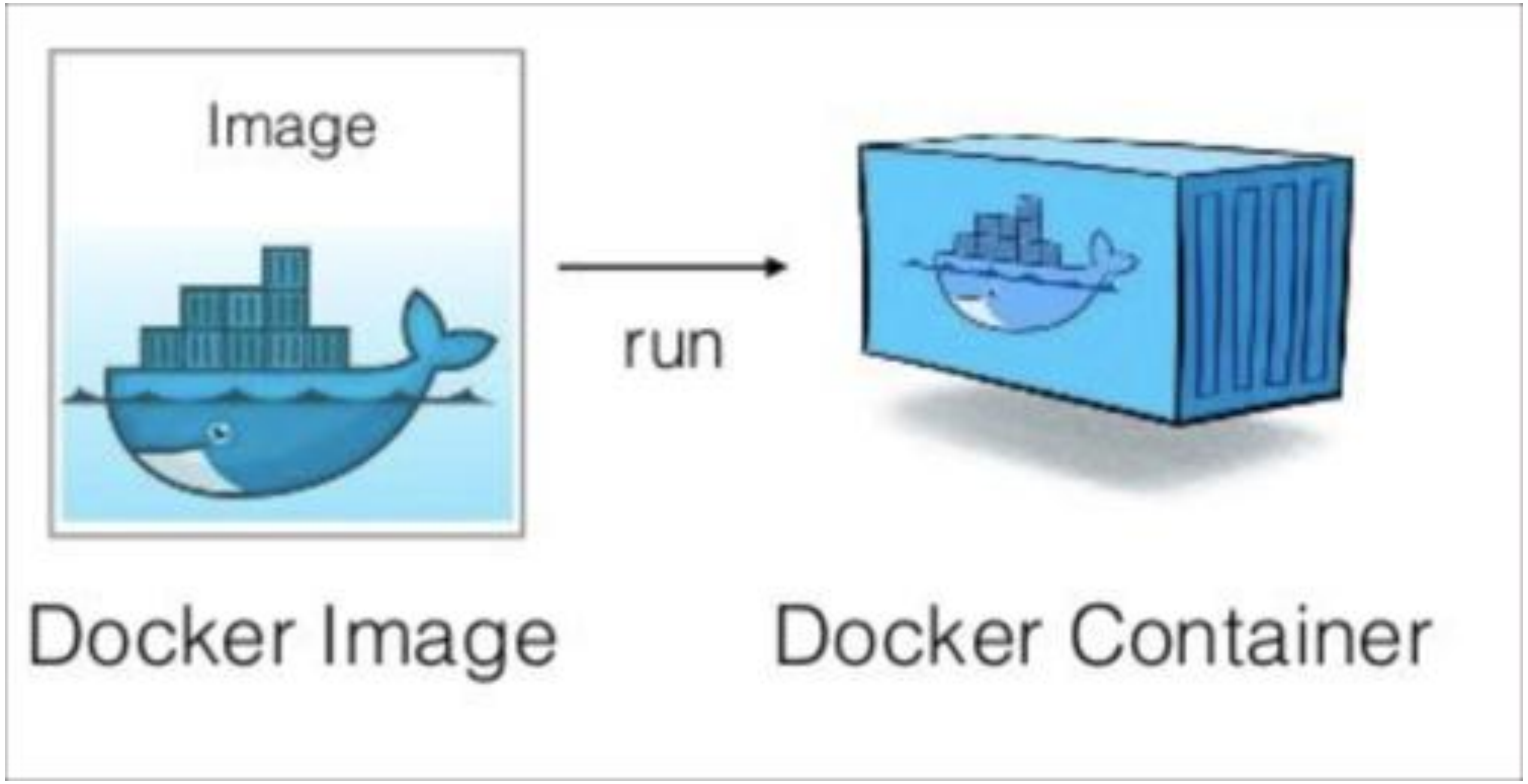
Application

Startup Script

Windows

Linux

Understanding Docker Architecture

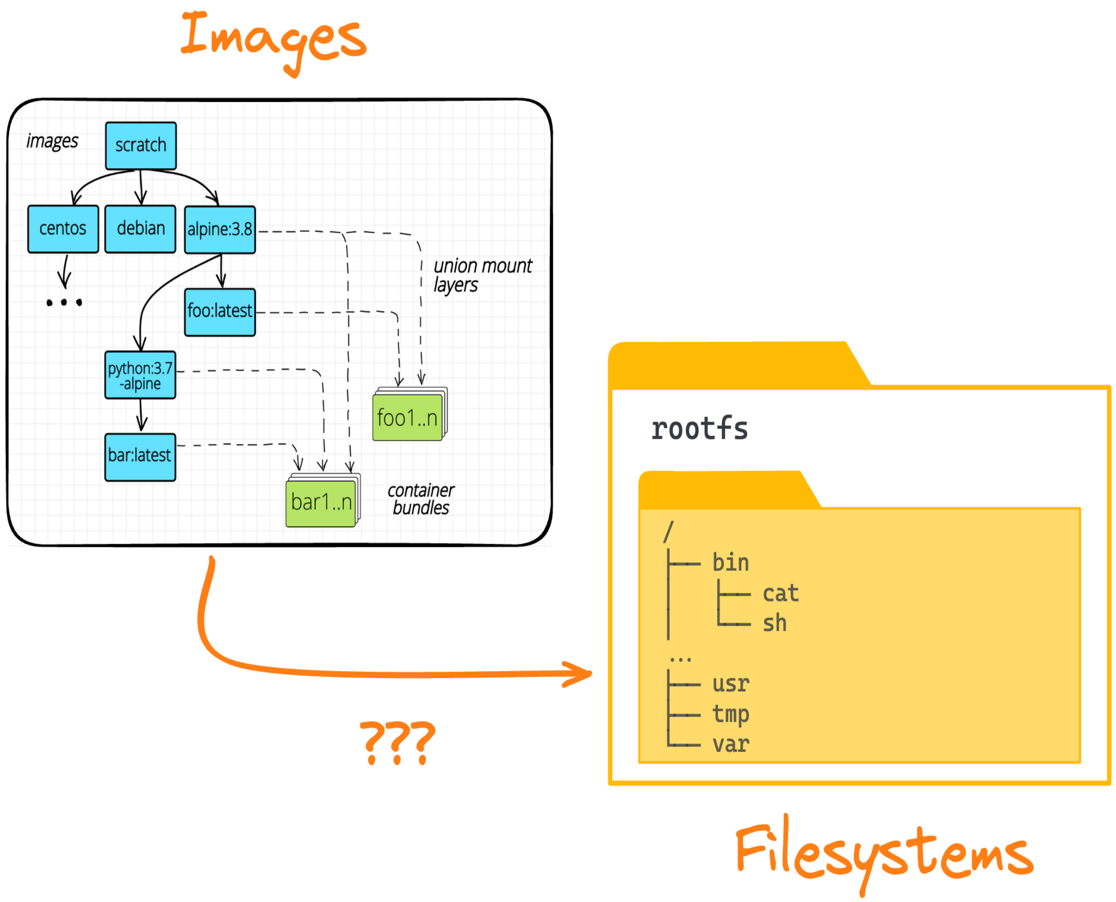


Docker Components

Component	Description
Docker for Mac	It allows one to run Docker containers on the Mac OS
Docker for Linux	It allows one to run Docker containers on the Linux OS.
Docker for Window	It allows one to run Docker containers on the Windows OS.
Docker Engine	It is used for building Docker images and creating Docker containers
Docker Hub	This is the registry which is used to host various Docker Images
Docker Compose	This is used to define applications using multiple Docker containers.

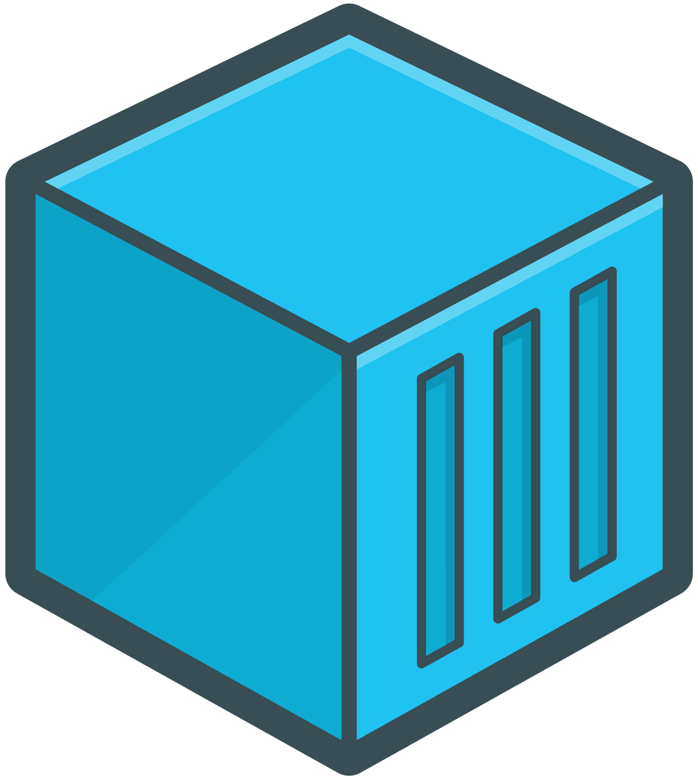
Docker Images

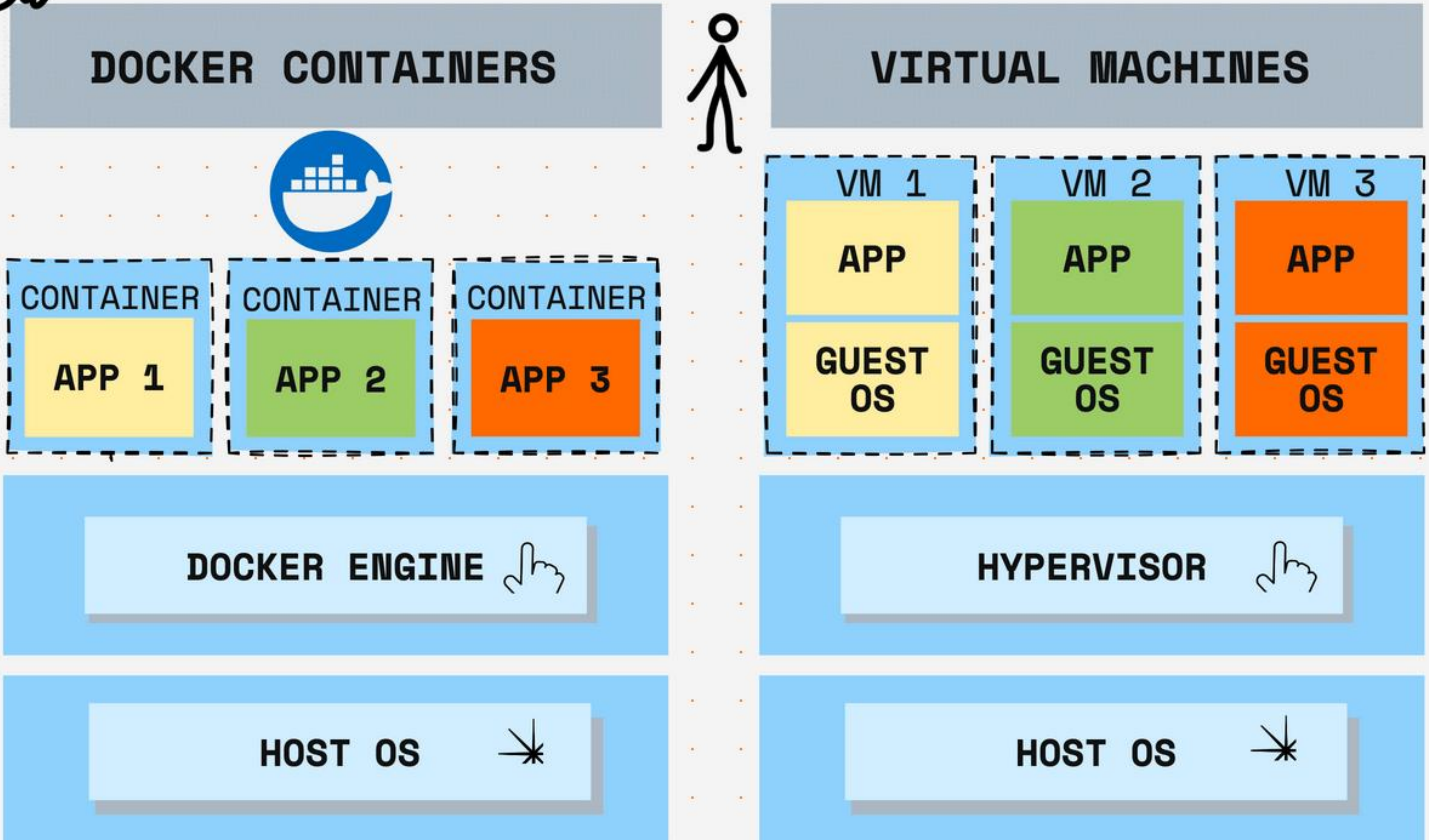
- A **Docker image** is a lightweight, standalone, and executable package that includes everything needed to run a piece of software: the code, runtime, libraries, environment variables, and configuration files. Docker images serve as the blueprint for creating Docker containers, meaning they provide a read-only snapshot that containers are based on.



Containers

- A **Docker container** is a runtime instance of a Docker image. When you run a Docker image, you create a container, which is an isolated environment where the application and its dependencies run. Containers encapsulate everything needed to execute an application, including the code, libraries, environment variables, and configuration files, ensuring consistency across various environments.





DOCKER CONTAINERS

CONTAINER

APP 1

CONTAINER

APP 2

CONTAINER

APP 3

DOCKER ENGINE

HOST OS

VIRTUAL MACHINES

VM 1

APP

GUEST OS

VM 2

APP

GUEST OS

VM 3

APP

GUEST OS

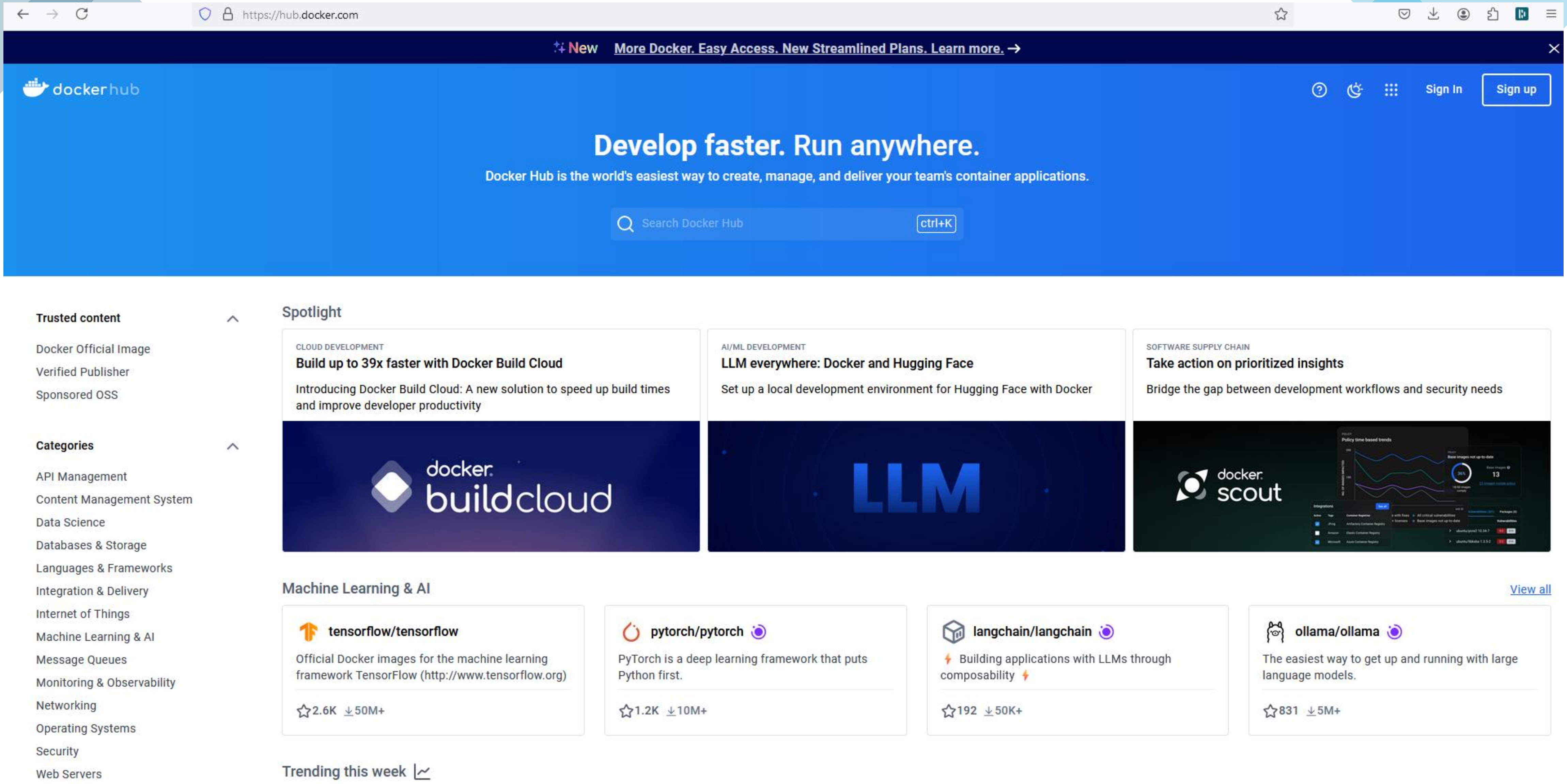
HYPERVISOR

HOST OS

Docker Hub

- **Docker Hub** is a cloud-based registry service provided by Docker where developers can store, share, and manage Docker images. It’s essentially a “hub” for container images, much like a public library where people can upload, download, and access resources (in this case, Docker images).

https://hub.docker.com/



https://hub.docker.com

✨New More Docker. Easy Access. New Streamlined Plans. Learn more. →

Develop faster. Run anywhere.

Docker Hub is the world's easiest way to create, manage, and deliver your team's container applications.

Trusted content

- Docker Official Image
- Verified Publisher
- Sponsored OSS

Categories

- API Management
- Content Management System
- Data Science
- Databases & Storage
- Languages & Frameworks
- Integration & Delivery
- Internet of Things
- Machine Learning & AI
- Message Queues
- Monitoring & Observability
- Networking
- Operating Systems
- Security
- Web Servers

Spotlight

CLOUD DEVELOPMENT
Build up to 39x faster with Docker Build Cloud
Introducing Docker Build Cloud: A new solution to speed up build times and improve developer productivity

AI/ML DEVELOPMENT
LLM everywhere: Docker and Hugging Face
Set up a local development environment for Hugging Face with Docker

SOFTWARE SUPPLY CHAIN
Take action on prioritized insights
Bridge the gap between development workflows and security needs

Machine Learning & AI View all

tensorflow/tensorflow
Official Docker images for the machine learning framework TensorFlow (http://www.tensorflow.org)
☆2.6K ↓50M+

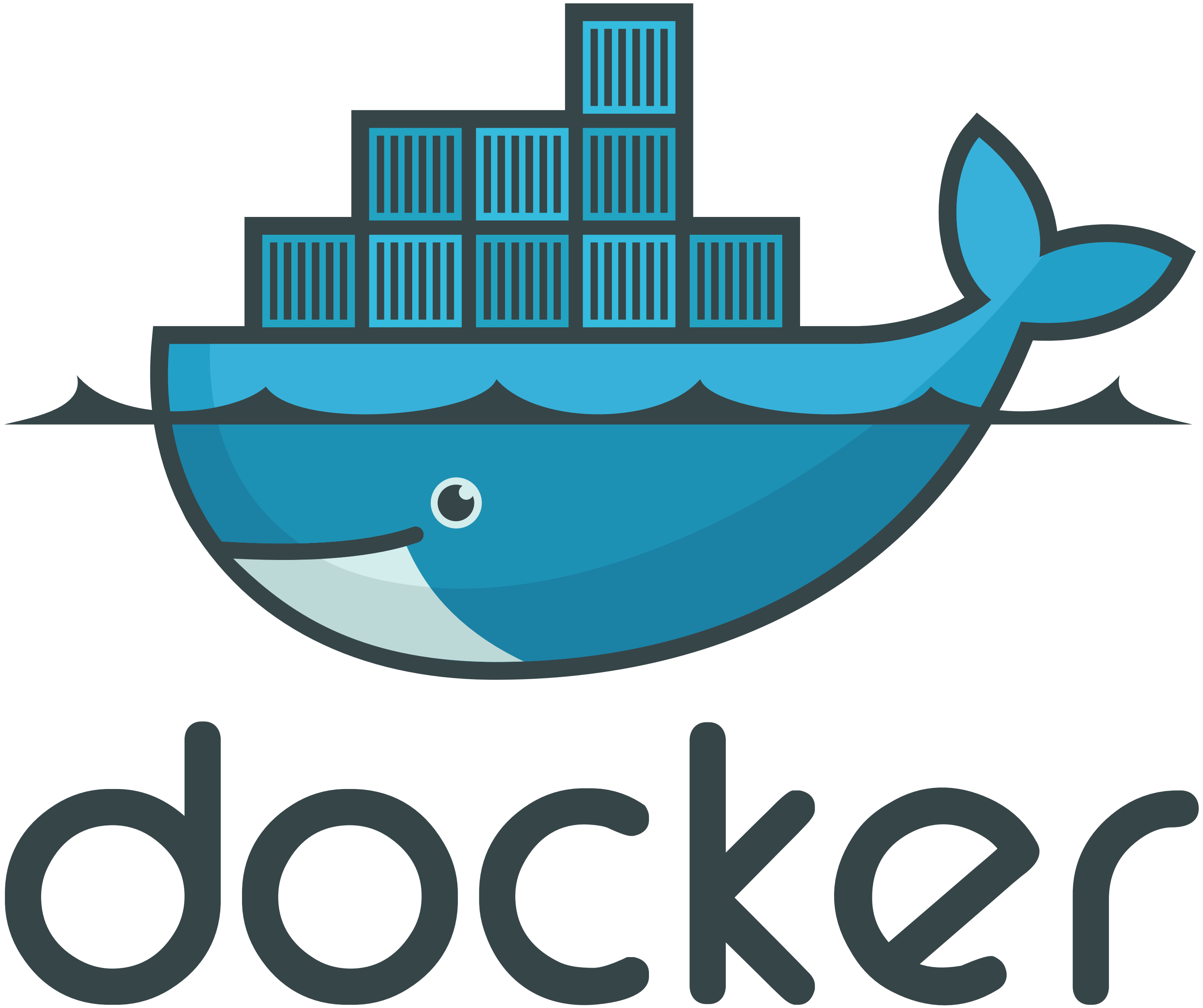
pytorch/pytorch
PyTorch is a deep learning framework that puts Python first.
☆1.2K ↓10M+

langchain/langchain
⚡ Building applications with LLMs through composability ⚡
☆192 ↓50K+

ollama/ollama
The easiest way to get up and running with large language models.
☆831 ↓5M+

Trending this week

Installing Docker

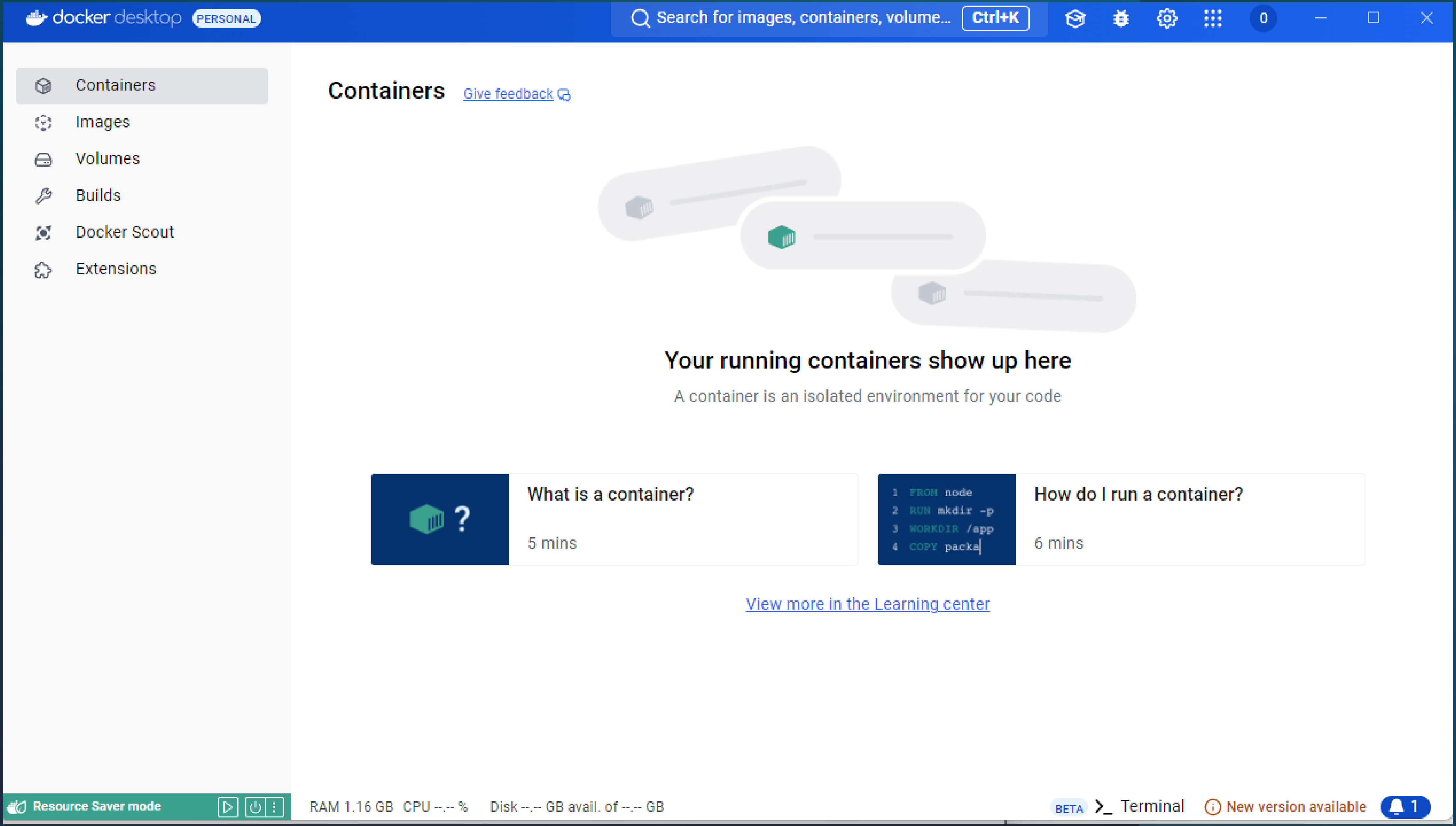


Docker Installation

Hands-On: Installing Docker in Linux

Docker for Windows

- **Docker for Windows**, also known as **Docker Desktop for Windows**, is a desktop application that enables you to build, share, and run containerized applications on Windows. It provides a streamlined, user-friendly interface to manage Docker containers, images, and environments, allowing developers to easily use Docker on Windows machines.



Containers

Give feedback

Your running containers show up here

A container is an isolated environment for your code

What is a container?

5 mins

```
FROM node
RUN mkdir -p
WORKDIR /app
COPY packa
```

How do I run a container?

6 mins

View more in the Learning center

RAM 1.16 GB CPU --.-- % Disk --.-- GB avail. of --.-- GB

Docker Installation

Hands-On: Installing Docker For Windows

Docker Command

Command	Description
docker version	To see the version of Docker running
docker info	To see more information on the Docker running on the system

docker pull

- The **docker pull** command is used to download a Docker image from a Docker registry, typically Docker Hub, to your local machine. This command fetches the specified image along with all its layers so you can create containers from it on your local system.

```
docker pull [OPTIONS] IMAGE[:TAG|@DIGEST]
```

docker pull Syntax:

- **IMAGE:** The name of the Docker image you want to download. This can include the repository and optionally a tag or digest.
- **TAG:** Specifies the image version. If no tag is specified, Docker will pull the latest version by default (usually denoted as latest).
- **DIGEST:** An alternative to the tag, this is a unique identifier of the image, typically a SHA256 hash. It ensures that you pull a specific, immutable version of the image.

docker run

Used to create and start a container based on a specified Docker image. When you execute the docker run command, Docker performs the following steps:

- **Pull the Image (if Needed)**: If the specified image does not already exist on your system, Docker will pull it from a repository, usually Docker Hub.
- **Create a Container**: Docker creates a container based on the pulled image. A container is an isolated, lightweight environment that includes everything needed to run an application.
- **Start the Container**: Docker then starts the container, allowing you to run an application or process within it.

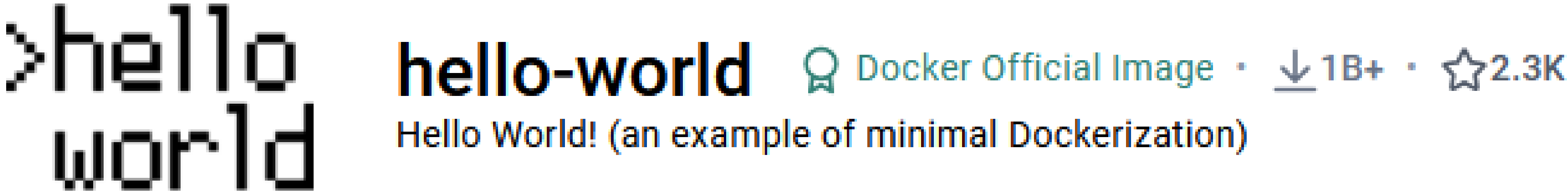
```
docker run [OPTIONS] IMAGE [COMMAND] [ARG...]
```

docker run Basic Syntax:

- **IMAGE:** The name of the Docker image to create the container from (e.g., nginx, alpine, ubuntu).
- **COMMAND:** Optional command to run inside the container, like /bin/bash.
- **OPTIONS:** Additional options to customize how the container runs, such as network settings, volume mounts, etc.

Hello-world images

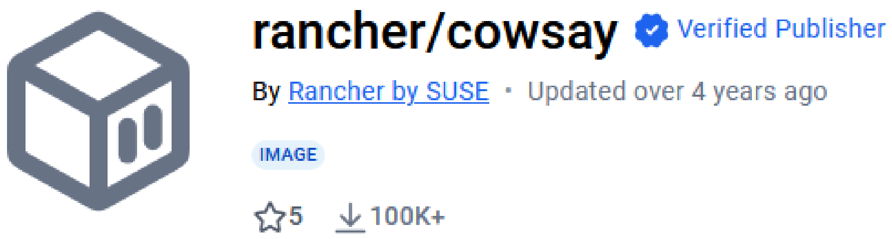
- The hello-world Docker image is a simple, minimal Docker image designed to help users verify that their Docker installation is working correctly. It is often the first image users run after installing Docker, as it provides a basic test that Docker can successfully pull and execute an image.



hello-world Docker Official Image · 1B+ · ☆2.3K

Hello World! (an example of minimal Dockerization)

Hands-on Challenge: racher/cowsay



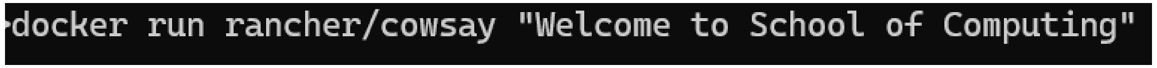
rancher/cowsay Verified Publisher

By Rancher by SUSE · Updated over 4 years ago

IMAGE

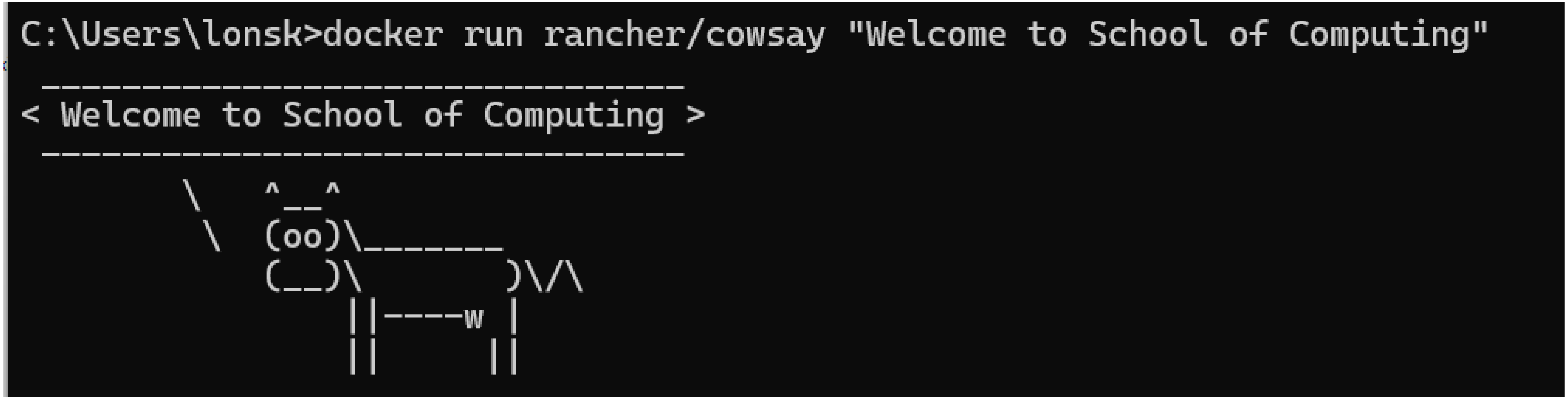
☆5 ↓100K+

1. Pull the "docker/whalesay" image from Docker Hub by using the following command:**docker pull docker/whalesay**

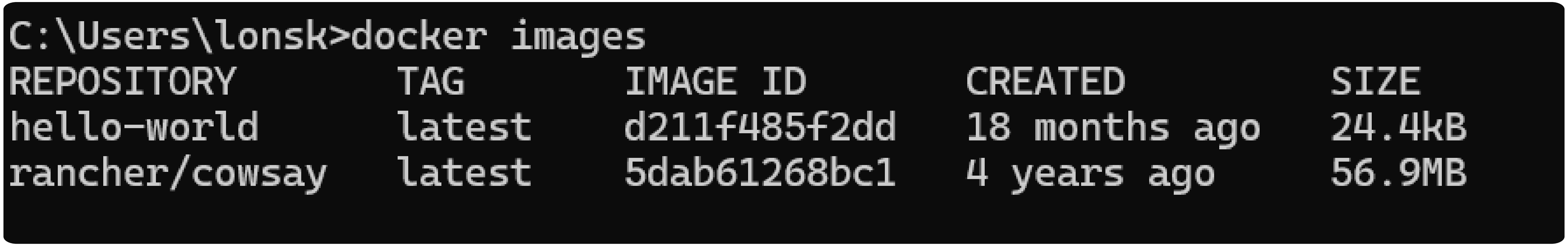


```
docker run rancher/cowsay "Welcome to School of Computing"
```

2. Run docker/whalesay with a Custom MessageUse the "docker/whalesay" image to display a custom message:



```
C:\Users\lonsk>docker run rancher/cowsay "Welcome to School of Computing"
 ________________________________
< Welcome to School of Computing >
 --------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```



```
C:\Users\lonsk>docker images
REPOSITORY        TAG       IMAGE ID       CREATED         SIZE
hello-world       latest    d211f485f2dd   18 months ago   24.4kB
rancher/cowsay    latest    5dab61268bc1   4 years ago     56.9MB
```

Displaying Docker Images

- To display all the images currently installed on the system use the docker images

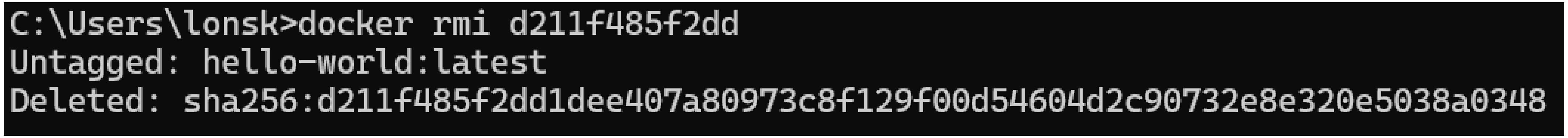
Syntax: `docker images`

Removing Docker Images

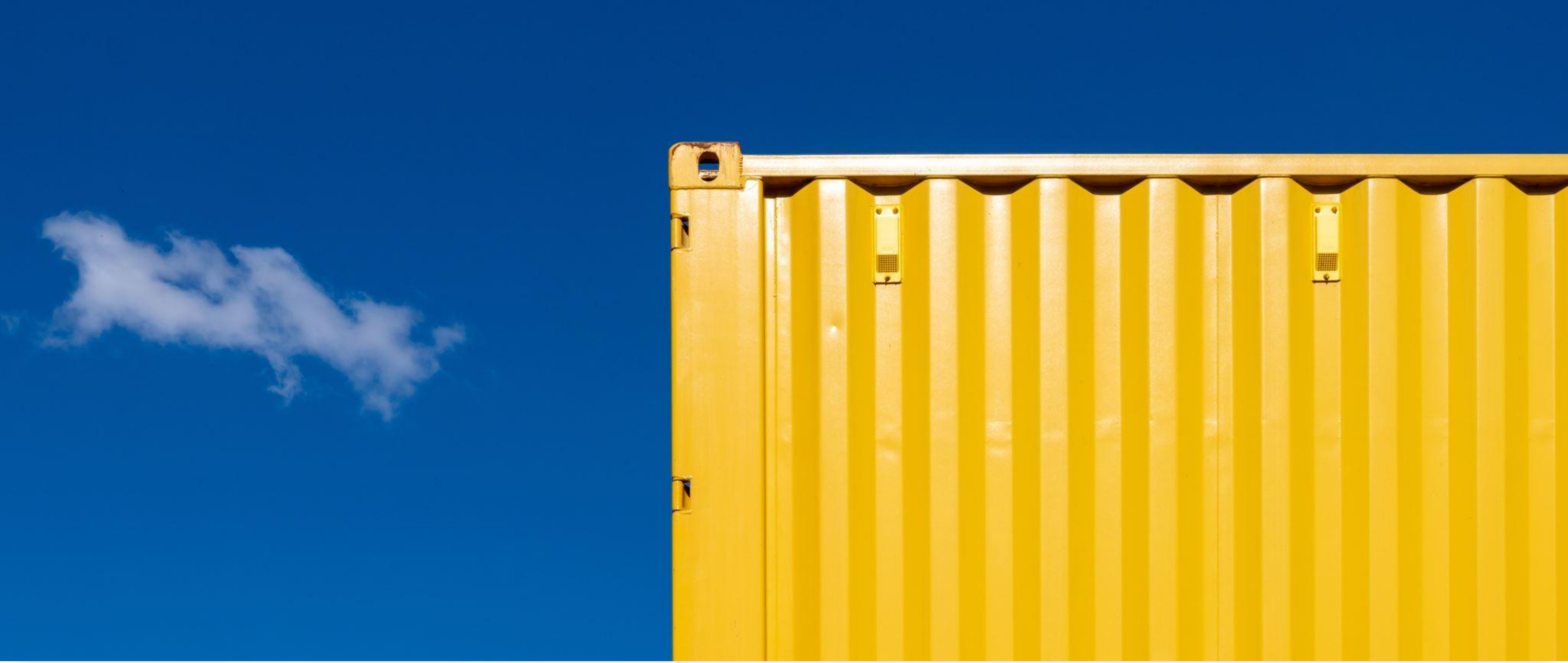
- The Docker images on the system can be removed via the docker **rmi** command.
- This command is used to remove Docker images.
- You can only delete an image that is not being use or link to a container. You need to delete the container first before you can delete the image

Syntax:

```
docker rmi ImageID
```



```
C:\Users\lonsk>docker rmi d211f485f2dd
Untagged: hello-world:latest
Deleted: sha256:d211f485f2dd1dee407a80973c8f129f00d54604d2c90732e8e320e5038a0348
```



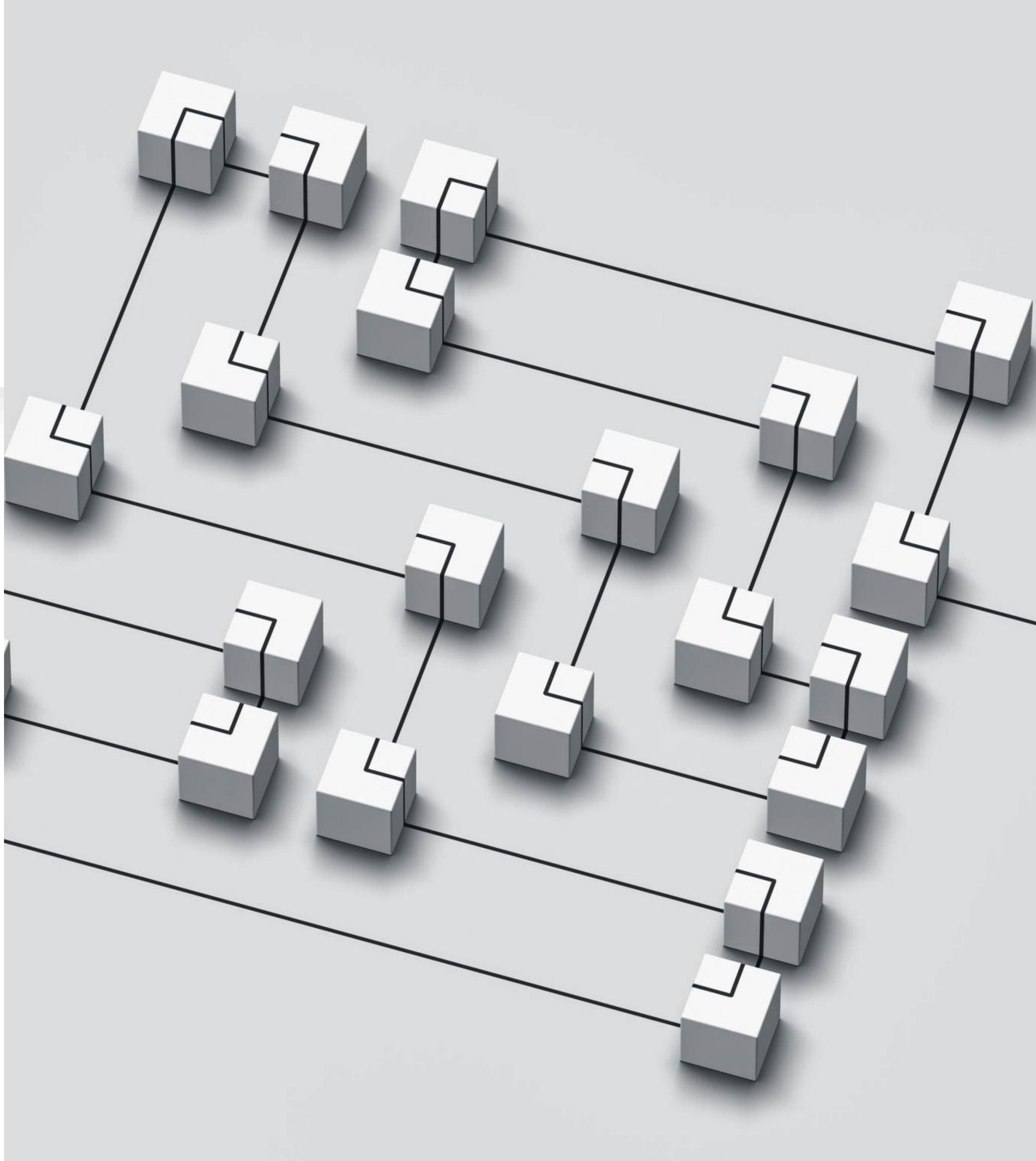
Docker - Containers

Interactive Mode Docker Image

- Interactive mode in Docker refers to running a container in such a way that you can interact with it directly via a command-line interface, just as if you were logged into a terminal session of a regular machine.

Syntax:

```
 docker run -it IMAGE_NAME
COMMAND
```



Interactive Mode Running Container

- To interactively connect to a running container and move inside its environment, you can use the docker exec command. This allows you to open a terminal session inside the container, letting you run commands interactively.

Syntax:

```
docker exec -it <container_id> /bin/bash
Or
docker exec -it <container_id> sh
```

docker stop

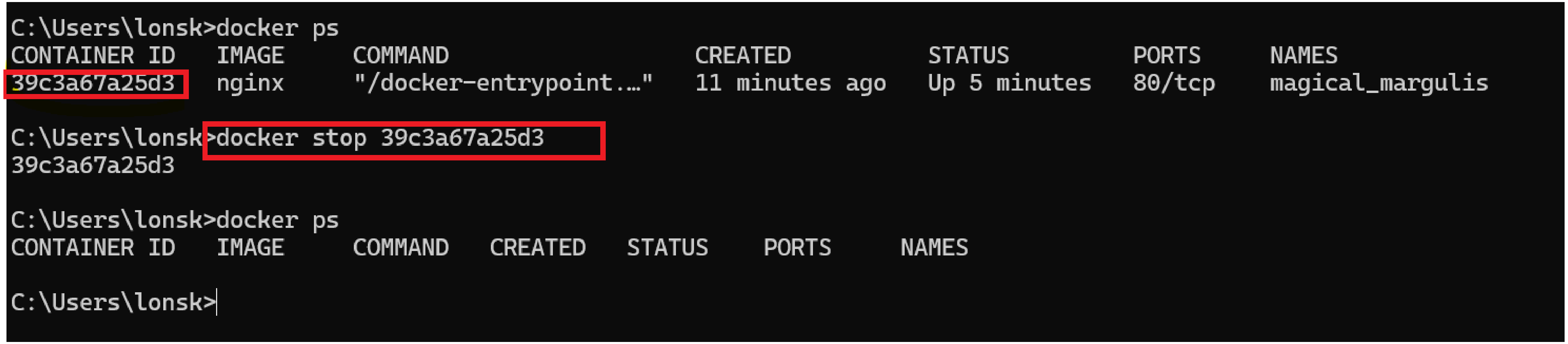
- This command is used to stop a running container.

Syntax

```
docker stop ContainerID
```

Options

- ContainerID − This is the Container ID which needs to be stopped



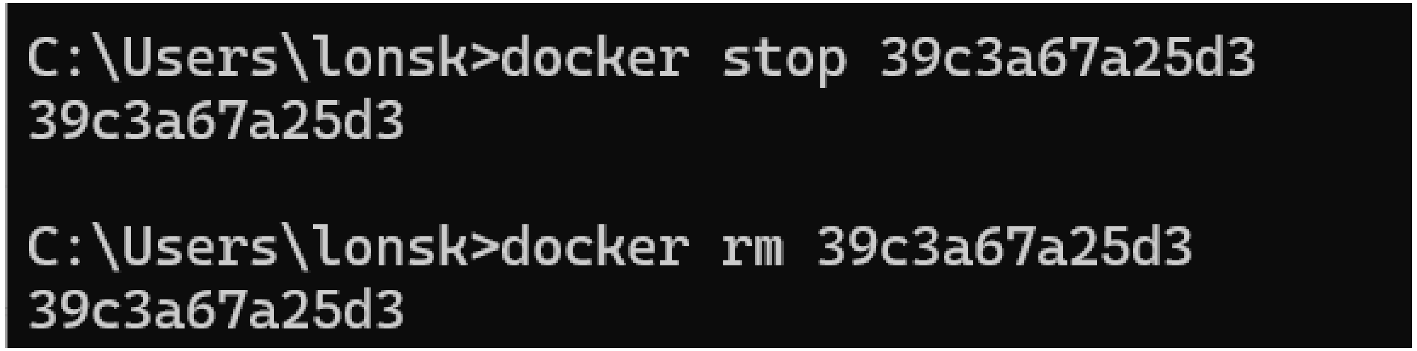
```
C:\Users\lonsk>docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS         PORTS     NAMES
39c3a67a25d3   nginx     "/docker-entrypoint.…"   11 minutes ago   Up 5 minutes   80/tcp    magical_margulis

C:\Users\lonsk>docker stop 39c3a67a25d3
39c3a67a25d3

C:\Users\lonsk>docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES

C:\Users\lonsk>
```

docker rm

- This command is used to delete a container.
- The container to be remove need to be stop first

Syntax:

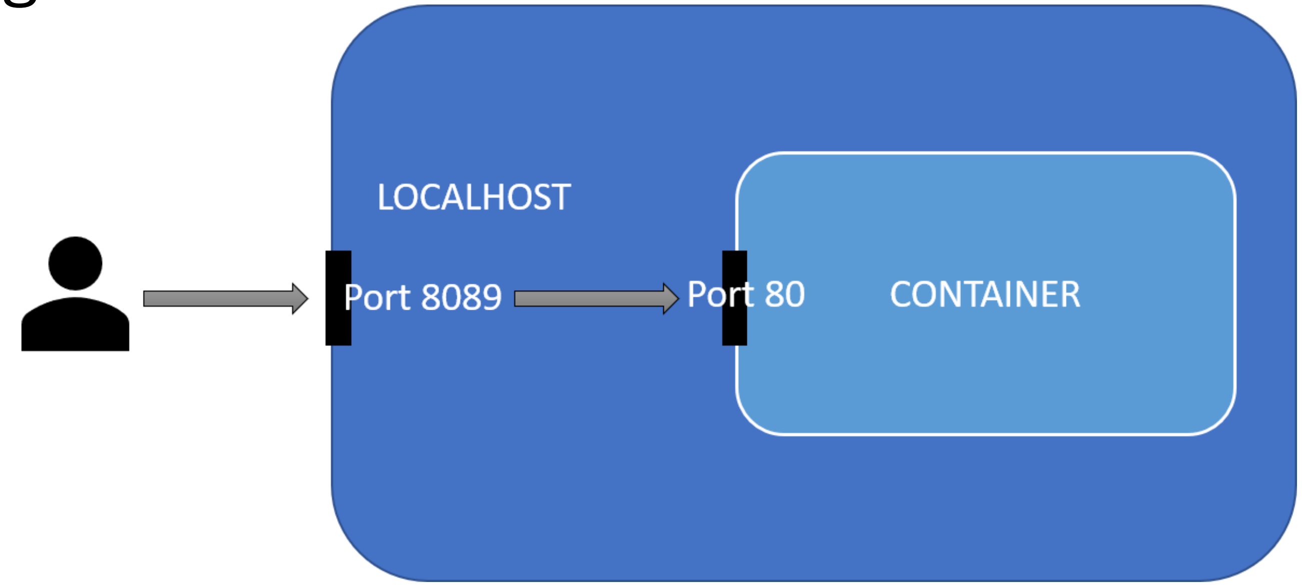
```
docker rm ContainerID
```



```
C:\Users\lonsk>docker stop 39c3a67a25d3
39c3a67a25d3

C:\Users\lonsk>docker rm 39c3a67a25d3
39c3a67a25d3
```

Exposing Ports

- Exposing ports in Docker allows services running inside a container to be accessible from outside the container, whether by the host machine or external clients. Here's a guide on how to expose ports when starting a container and after a container is already running.



Hands-on: Creating an Ubuntu Webserver

Step-by-step:

1. Create a Container “webserver” with image “ubuntu” and port mapping 80:80.

```
docker run -td --name webserver -p 80:80 ubuntu
```

```
docker run -td --name webserver -p 80:80 ubuntu
```

Hands-on: Creating an Ubuntu Webserver

- Go inside the container “webserver”

```
docker exec -it webserver /bin/bash
```

- Update Server Available Packages

```
apt-get update
```

- Install webserver package “apache2”

```
apt-get install apache2 -y
```

Hands-on: Creating an Ubuntu Webserver

- Go inside “/var/www/html” directory and create a file “index.html”

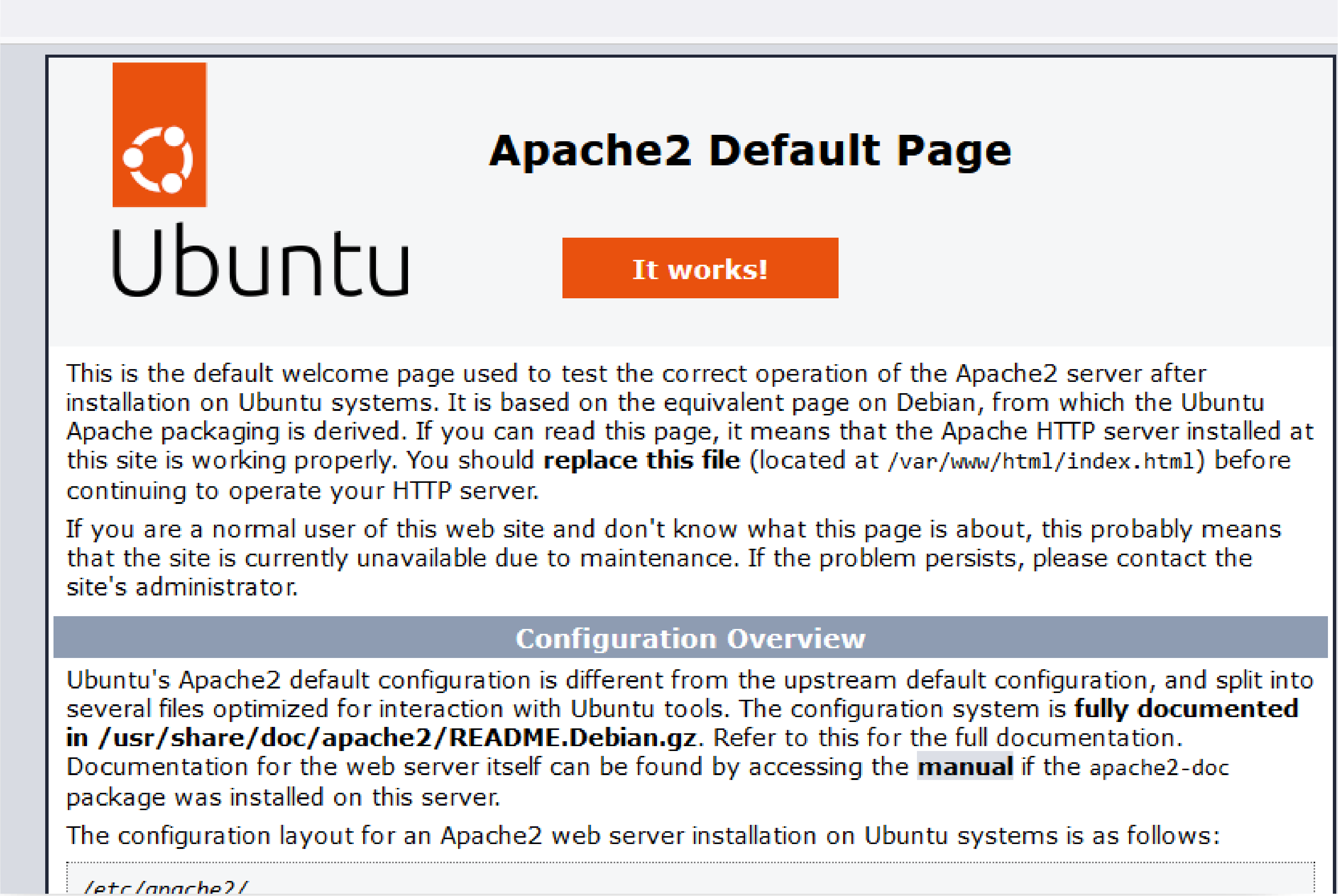
```
cd /var/www/html
```

- Start apache2 service

```
service apache2 start
```

- Verify status of apache2 service

```
service apache2 status
```



Apache2 Default Page

This is the default welcome page used to test the correct operation of the Apache2 server after installation on Ubuntu systems. It is based on the equivalent page on Debian, from which the Ubuntu Apache packaging is derived. If you can read this page, it means that the Apache HTTP server installed at this site is working properly. You should **replace this file** (located at `/var/www/html/index.html`) before continuing to operate your HTTP server.

If you are a normal user of this web site and don't know what this page is about, this probably means that the site is currently unavailable due to maintenance. If the problem persists, please contact the site's administrator.

Configuration Overview

Ubuntu's Apache2 default configuration is different from the upstream default configuration, and split into several files optimized for interaction with Ubuntu tools. The configuration system is **fully documented in /usr/share/doc/apache2/README.Debian.gz**. Refer to this for the full documentation. Documentation for the web server itself can be found by accessing the **manual** if the `apache2-doc` package was installed on this server.

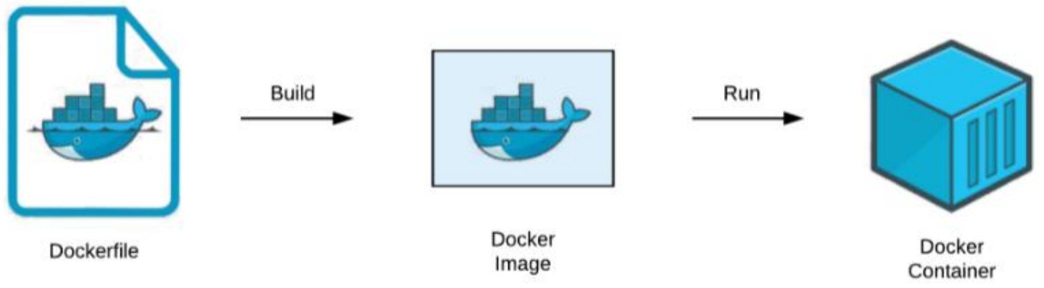
The configuration layout for an Apache2 web server installation on Ubuntu systems is as follows:

```
/etc/apache2/
```

Docker – Docker File

Dockerfile

- A **Dockerfile** is a text file that contains a set of instructions for building a Docker image. Each command in the Dockerfile tells Docker how to configure the environment, install dependencies, copy files, and define what the container will run.



Key Components of a Dockerfile

FROM:

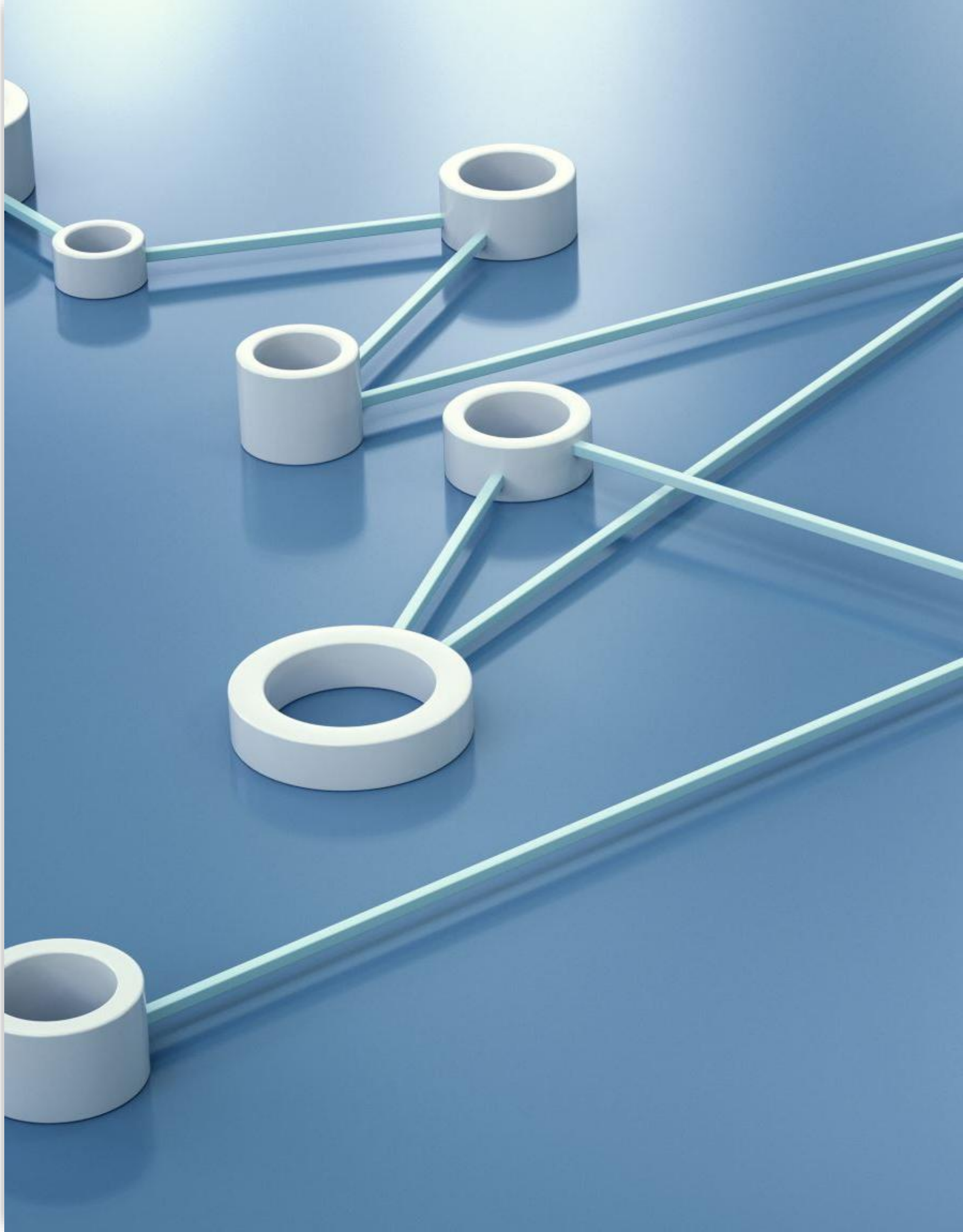
- Specifies the base image to use as the starting point for your Docker image. This can be a minimal Linux distribution like ubuntu or a specific pre-configured image.

Example: FROM ubuntu:latest

RUN:

- Executes commands in the container during the image-building process. Commonly used to install dependencies or software packages.

Example: RUN apt-get update && apt-get install -y python3



Key Components of a Dockerfile

COPY and ADD:

- COPY: Copies files or directories from the local machine to the Docker container.
- ADD: Similar to COPY, but can also handle URLs and automatically extract tar files.

Example: COPY app/ /usr/src/app

WORKDIR:

- Sets the working directory inside the container. Commands following WORKDIR will be executed relative to this directory.

Example: WORKDIR /usr/src/app

Key Components of a Dockerfile

EXPOSE:

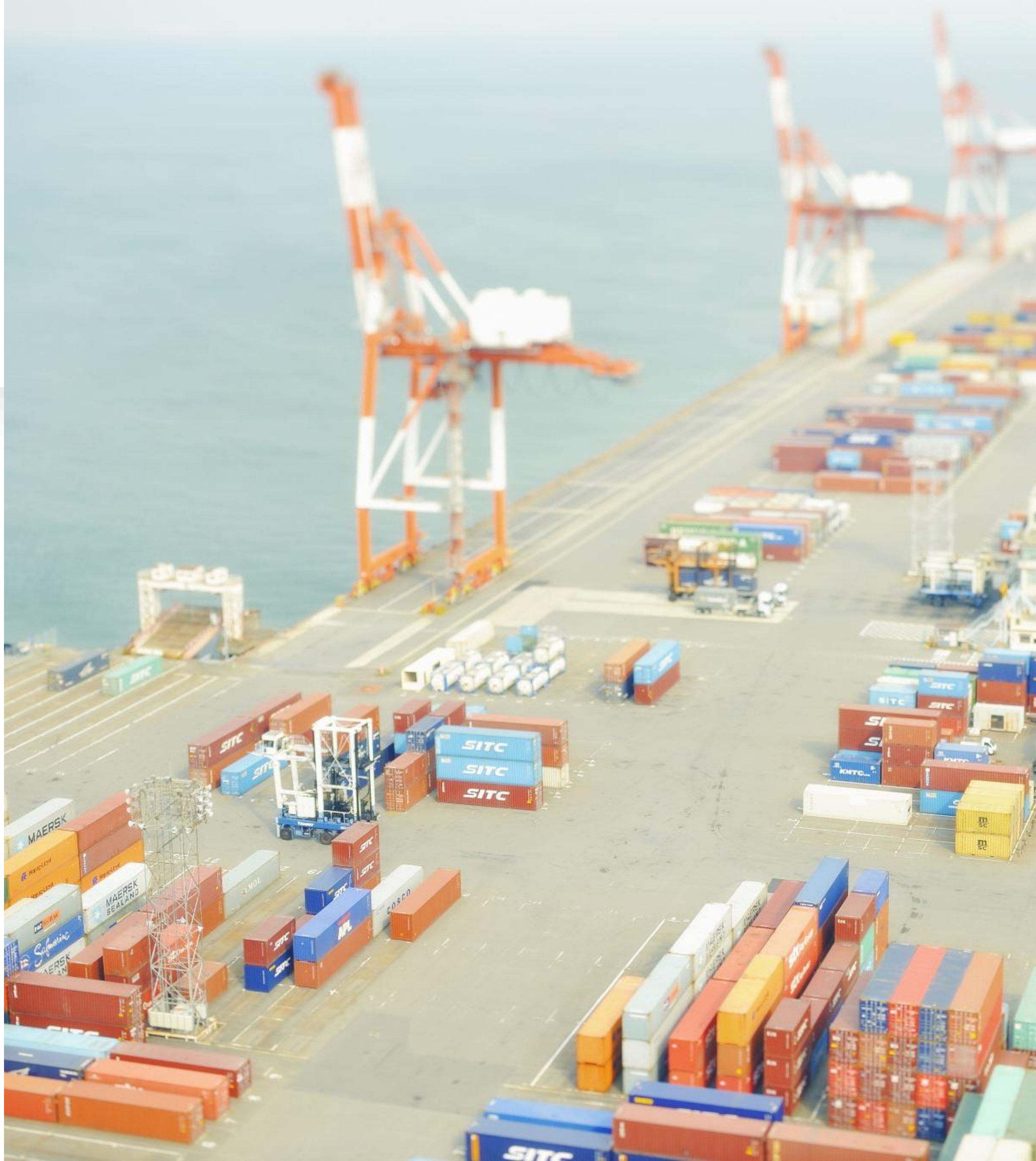
- Specifies the port(s) that the container will listen on at runtime. This doesn't automatically publish the ports; it only indicates which ports the container will use.

Example: EXPOSE 80

ENV:

- Sets environment variables in the container, which can be accessed by the application at runtime.

Example: ENV APP_ENV=production



Key Components of a Dockerfile

CMD:

- Defines the default command to run when a container is started from the image. This command can be overridden when running the container.

Example: CMD ["python3", "app.py"]

ENTRYPOINT:

- Sets a default executable that will always run when the container starts, and can be combined with CMD to pass default arguments.

Example: ENTRYPOINT ["python3"]

Key Components of a Dockerfile

VOLUME:

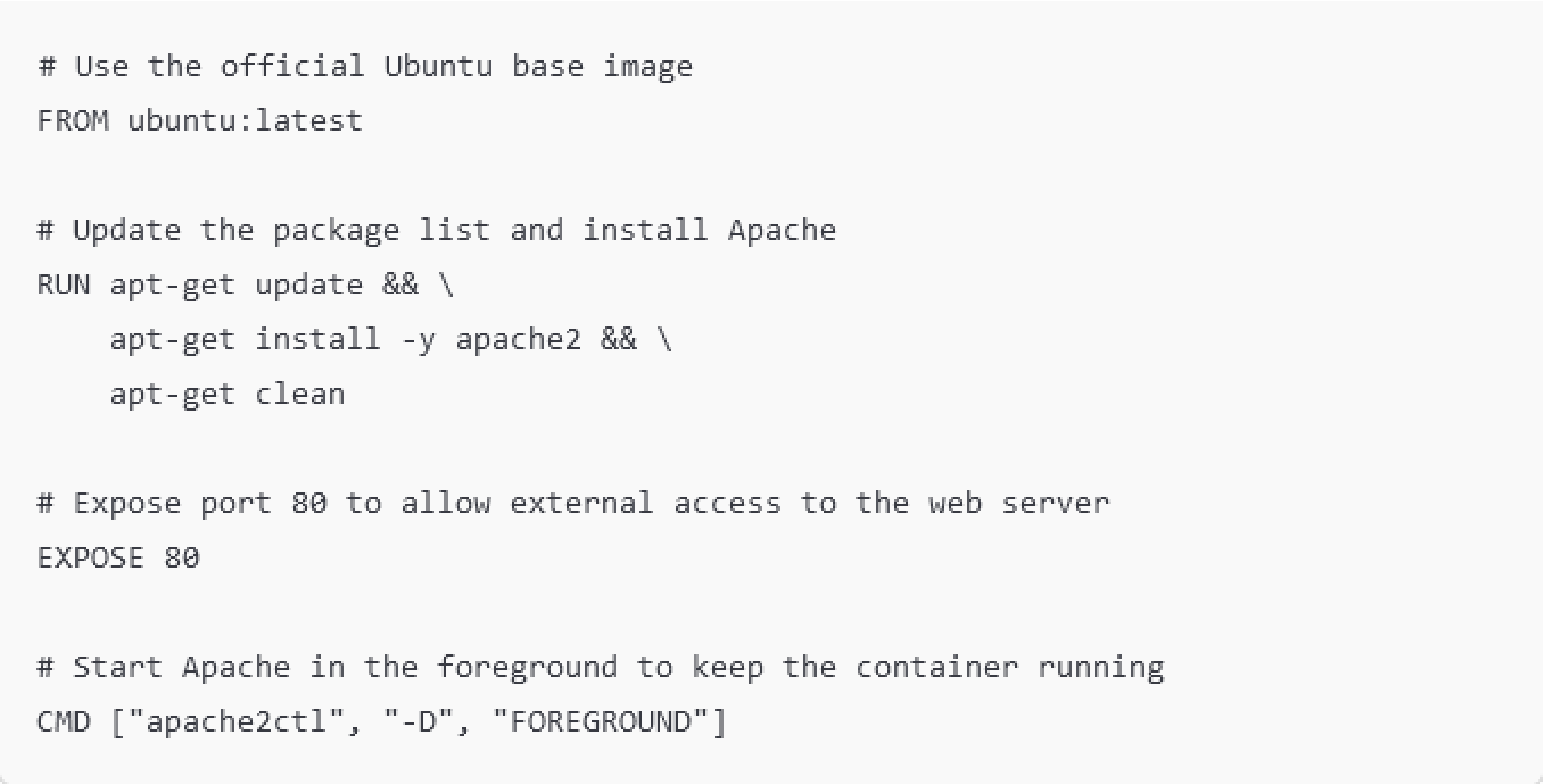
- Creates a mount point to allow persistent data storage outside of the container’s file system.

Example: VOLUME /data

USER:

- Sets the user under which the container should run.

Example: USER nonrootuser



```
# Use the official Ubuntu base image
FROM ubuntu:latest

# Update the package list and install Apache
RUN apt-get update && \
    apt-get install -y apache2 && \
    apt-get clean

# Expose port 80 to allow external access to the web server
EXPOSE 80

# Start Apache in the foreground to keep the container running
CMD ["apache2ctl", "-D", "FOREGROUND"]
```

Building and Running the Docker Image

1. Build the Docker Image

- Run this command in the terminal where your Dockerfile is located to build the image:

```
docker build -t my-apache-server .
```

2. Run the Docker Container

- Once the image is built, run the container and map port 80 on the host to port 80 in the container:

```
docker run -d -p 80:80 my-apache-server
```

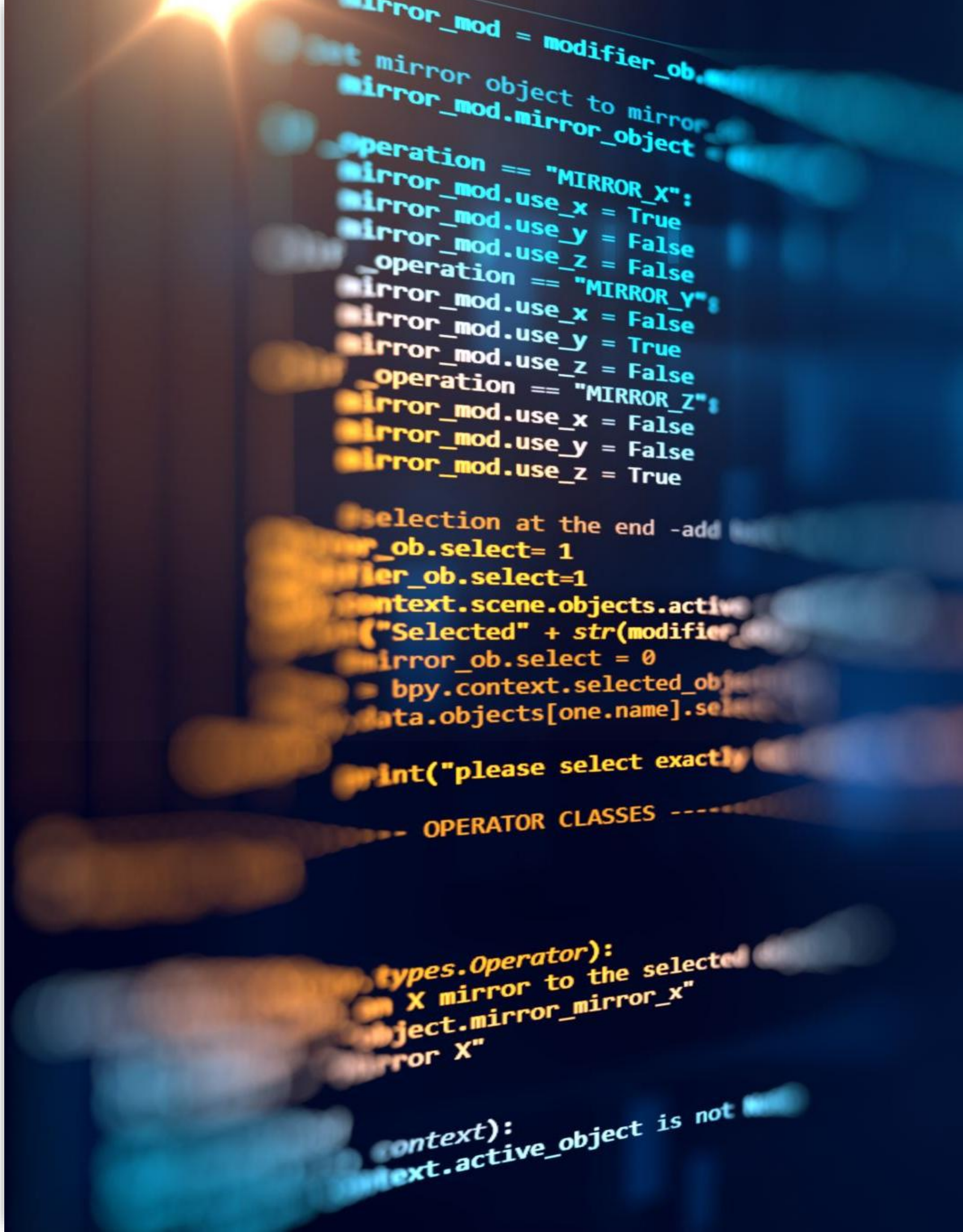
Building and Running the Docker Image

3. Access Apache in the Browser

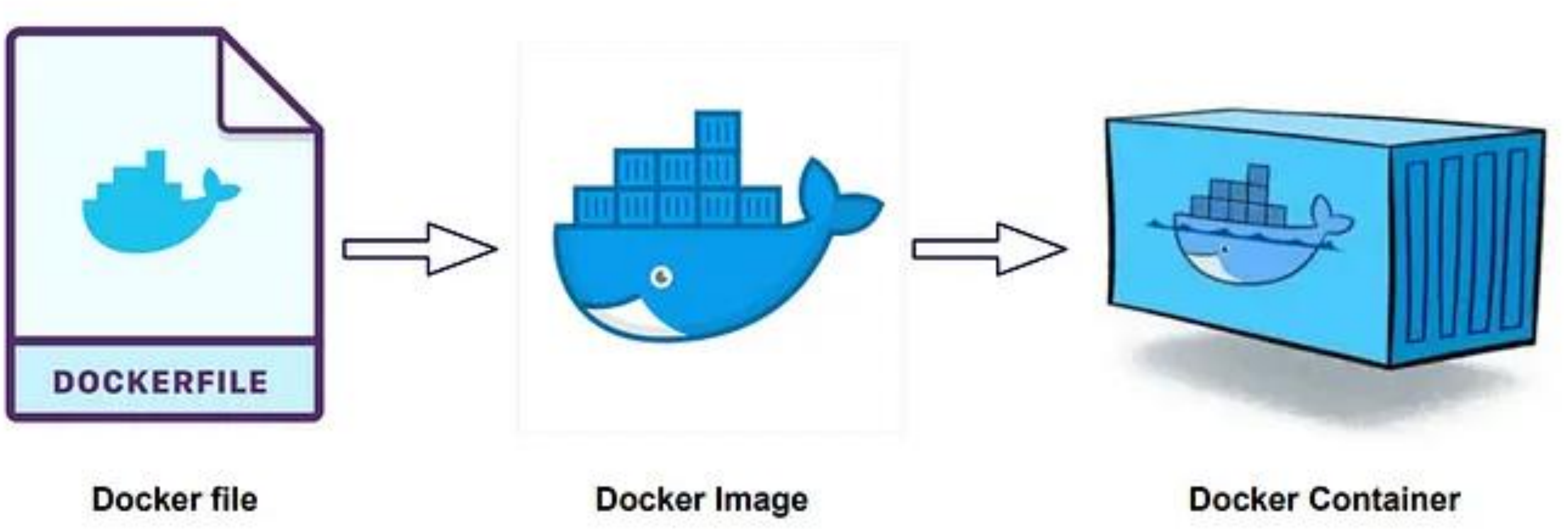
- Open your web browser and go to **http://localhost**. You should see the default Apache web server page, which confirms that Apache is running in the container.
- This Dockerfile sets up a lightweight, containerized Apache web server on Ubuntu, accessible via port 80 on your host machine.

Full Stack Development

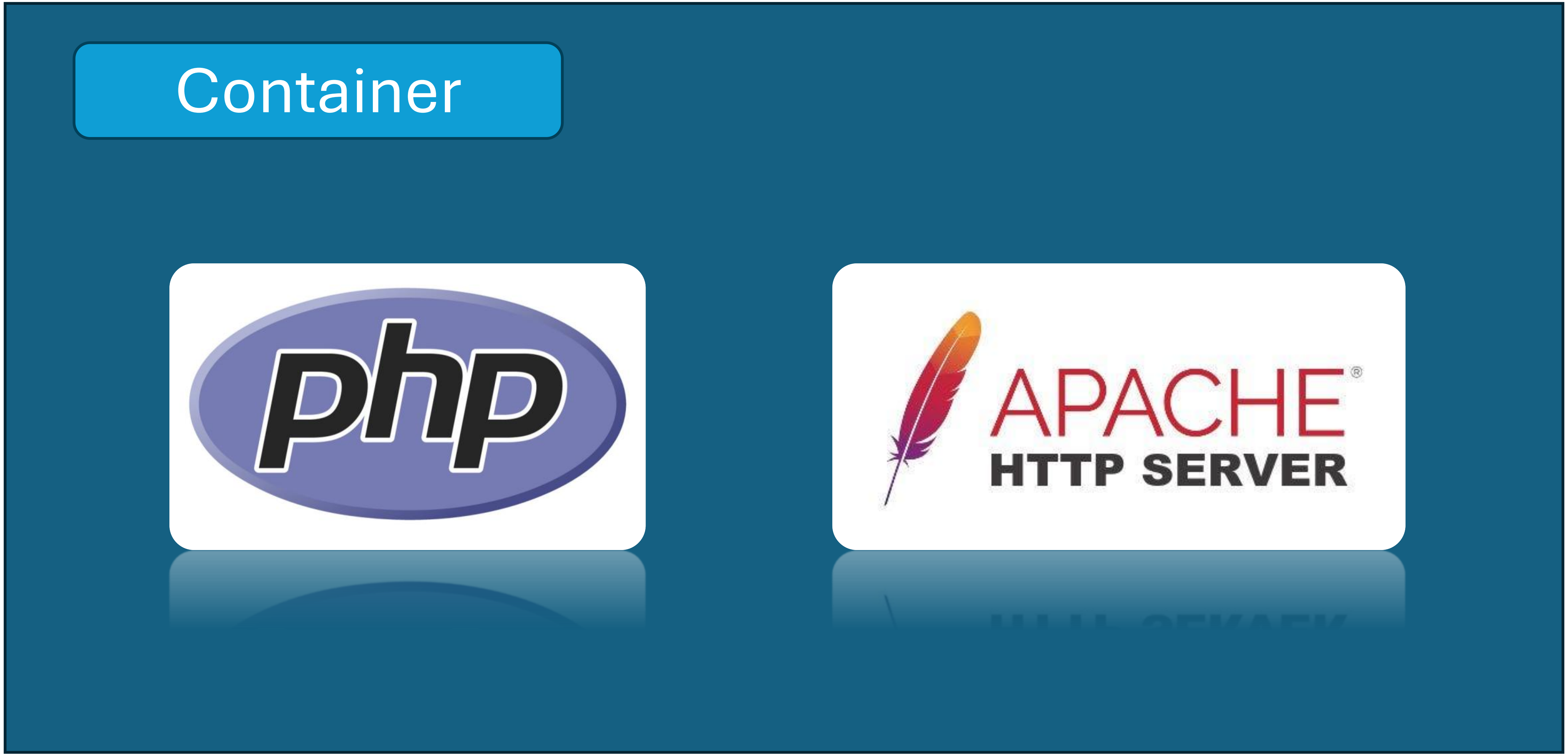
- A Full Stack Developer is a software developer skilled in both front-end and back-end development. This means they have the ability to work on the complete stack of technologies involved in building a web application. The "stack" includes all layers of software development, from designing the user interface to managing the server and database.



PHP and Apache Web Server Development Environment - Docker File



php / Apache-7.4



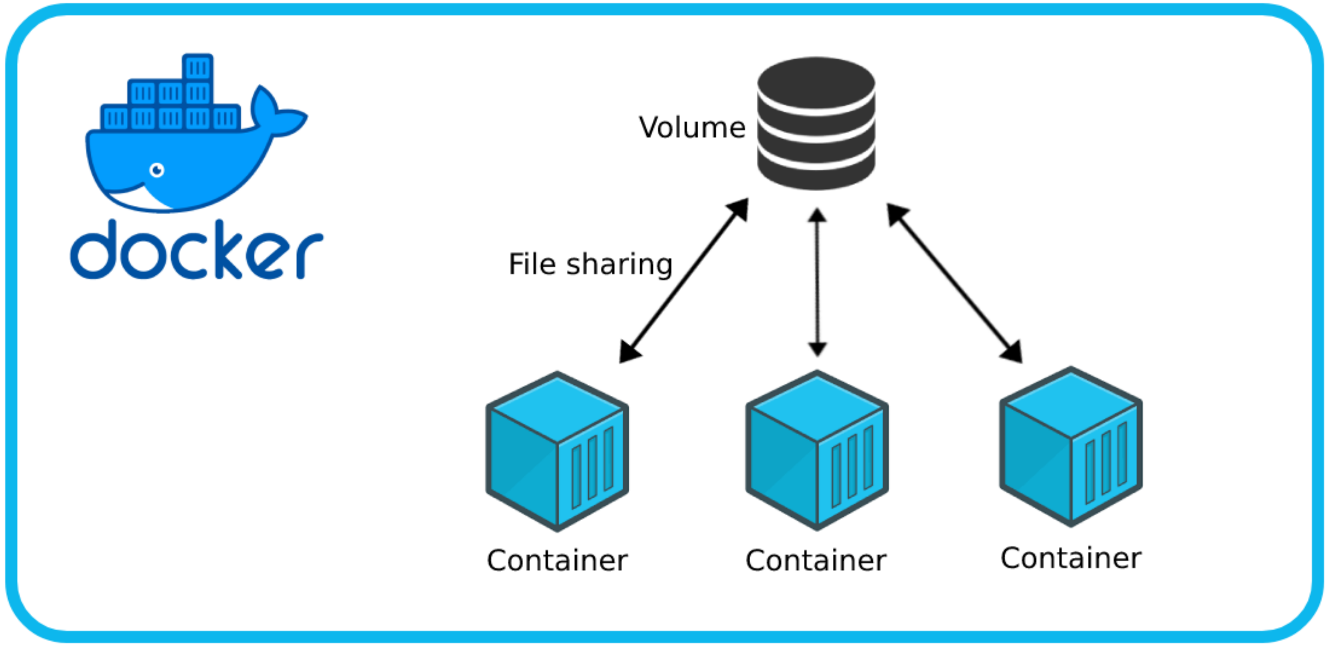
Container

Hands-on: PHP and Apache Web Server Development Environment

php-apache-webserver Dev Environment

Volume

- a **volume** is a storage mechanism that allows you to persist and manage data generated or used by Docker containers. Unlike the data stored inside a container, which is deleted when the container is removed, volumes are designed to **persist data beyond the lifecycle of a container**.



Key Characteristics of Volumes

- Persistence
- Shared Access
- Separate from Container File System
- Host Directory Mapping

Mapping a Volume

- Linking a directory from your host machine to a directory inside the container. This allows the container to access and save data directly to the host’s filesystem, which is useful for persisting data, sharing files across containers, or live development.

Docker Compose

- **Docker Compose** is a tool that allows you to define and manage multi-container Docker applications. Using a simple YAML configuration file, Docker Compose enables you to define, configure, and deploy multiple services, networks, and volumes, making it easy to manage complex applications that require several interconnected containers (such as a web server, database, and caching service).

Key Features of Docker Compose

- Multi-Container Applications
- YAML Configuration File
- Simplified Commands
- Environment Management
- Networking
- Volumes for Persistent Data

Basic Docker Compose Workflow

1. Create a **docker-compose.yml** File:

- This file defines the services, networks, and volumes needed for the

application.

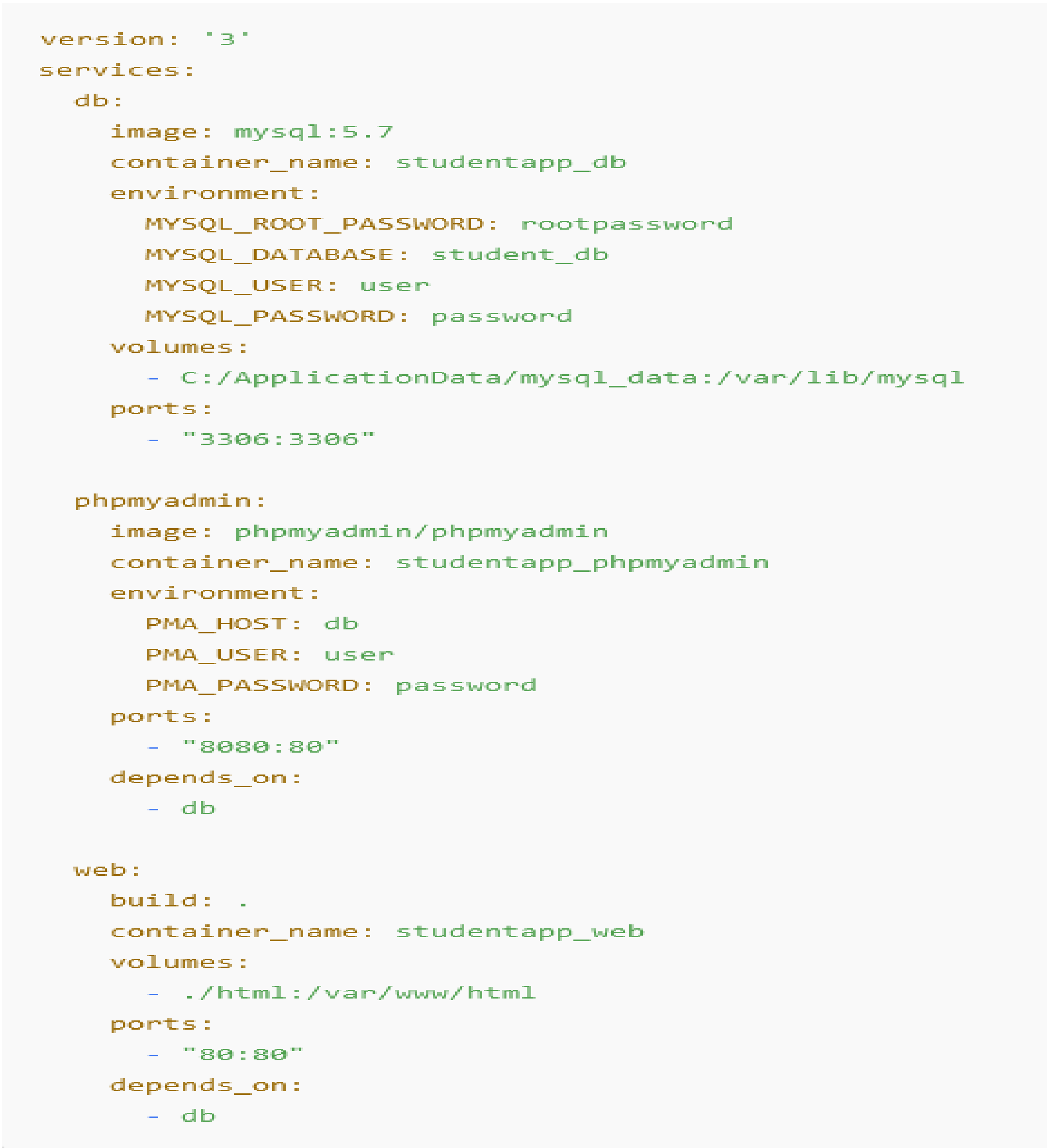
2 .Run docker-compose up:

- This command creates and starts all services defined in the docker-compose.yml file.

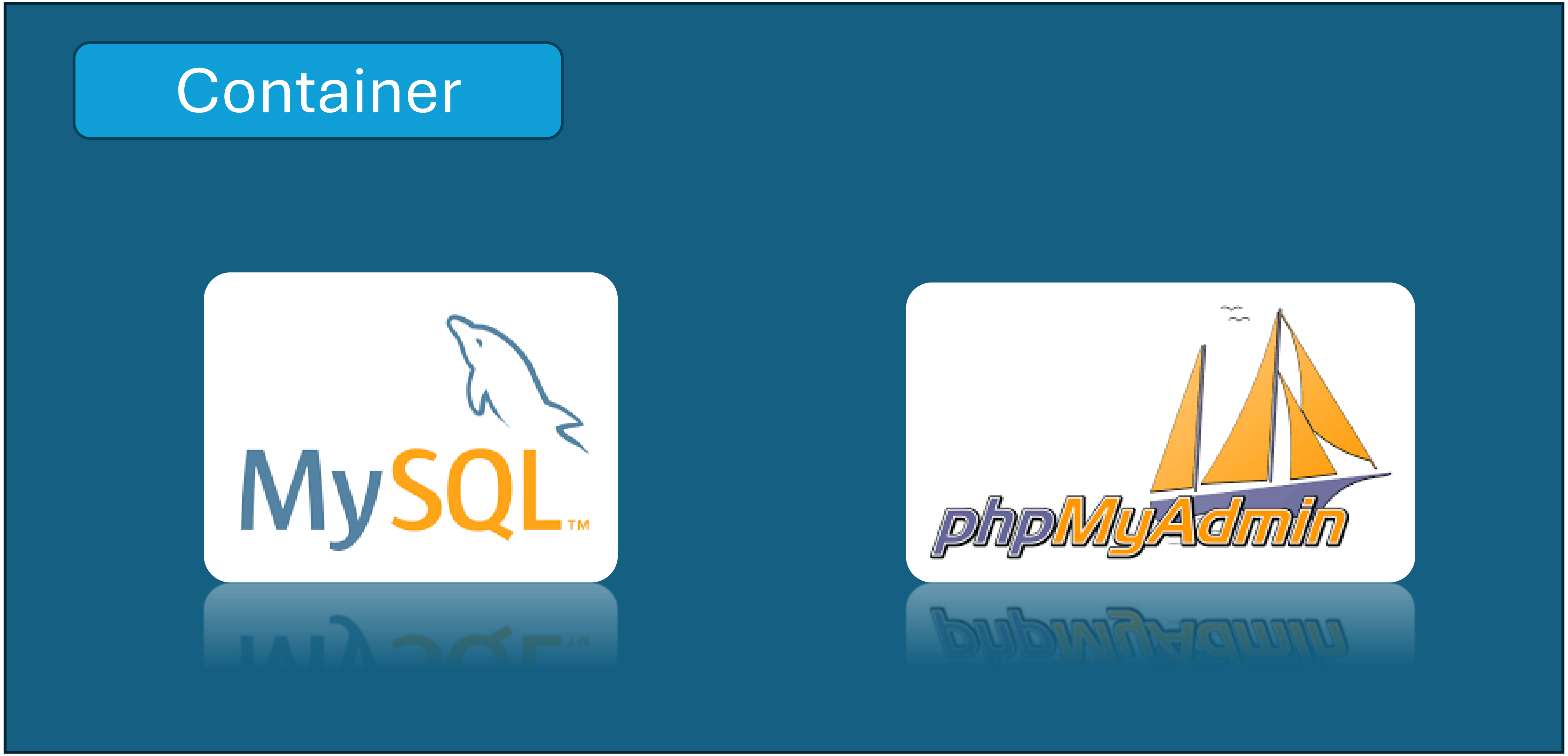
3. Manage Services:

- Use commands like docker-compose stop, docker-compose start, and docker-compose down to control your services.

Sample docker-compose.yaml

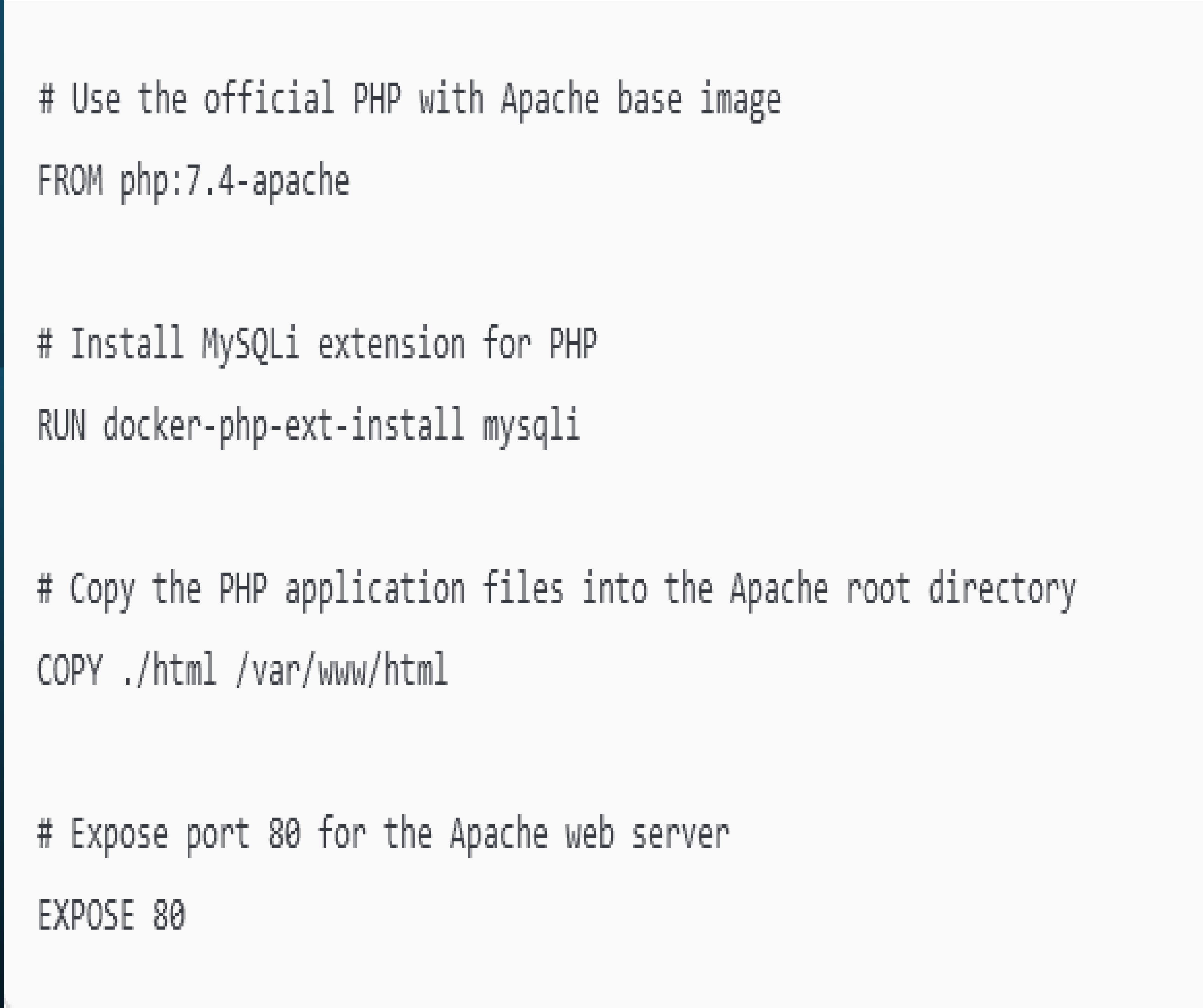


```yaml
version: '3'
services:
  db:
    image: mysql:5.7
    container_name: studentapp_db
    environment:
      MYSQL_ROOT_PASSWORD: rootpassword
      MYSQL_DATABASE: student_db
      MYSQL_USER: user
      MYSQL_PASSWORD: password
    volumes:
      - C:/ApplicationData/mysql_data:/var/lib/mysql
    ports:
      - "3306:3306"

  phpmyadmin:
    image: phpmyadmin/phpmyadmin
    container_name: studentapp_phpmyadmin
    environment:
      PMA_HOST: db
      PMA_USER: user
      PMA_PASSWORD: password
    ports:
      - "8080:80"
    depends_on:
      - db

  web:
    build: .
    container_name: studentapp_web
    volumes:
      - ./html:/var/www/html
    ports:
      - "80:80"
    depends_on:
      - db
```



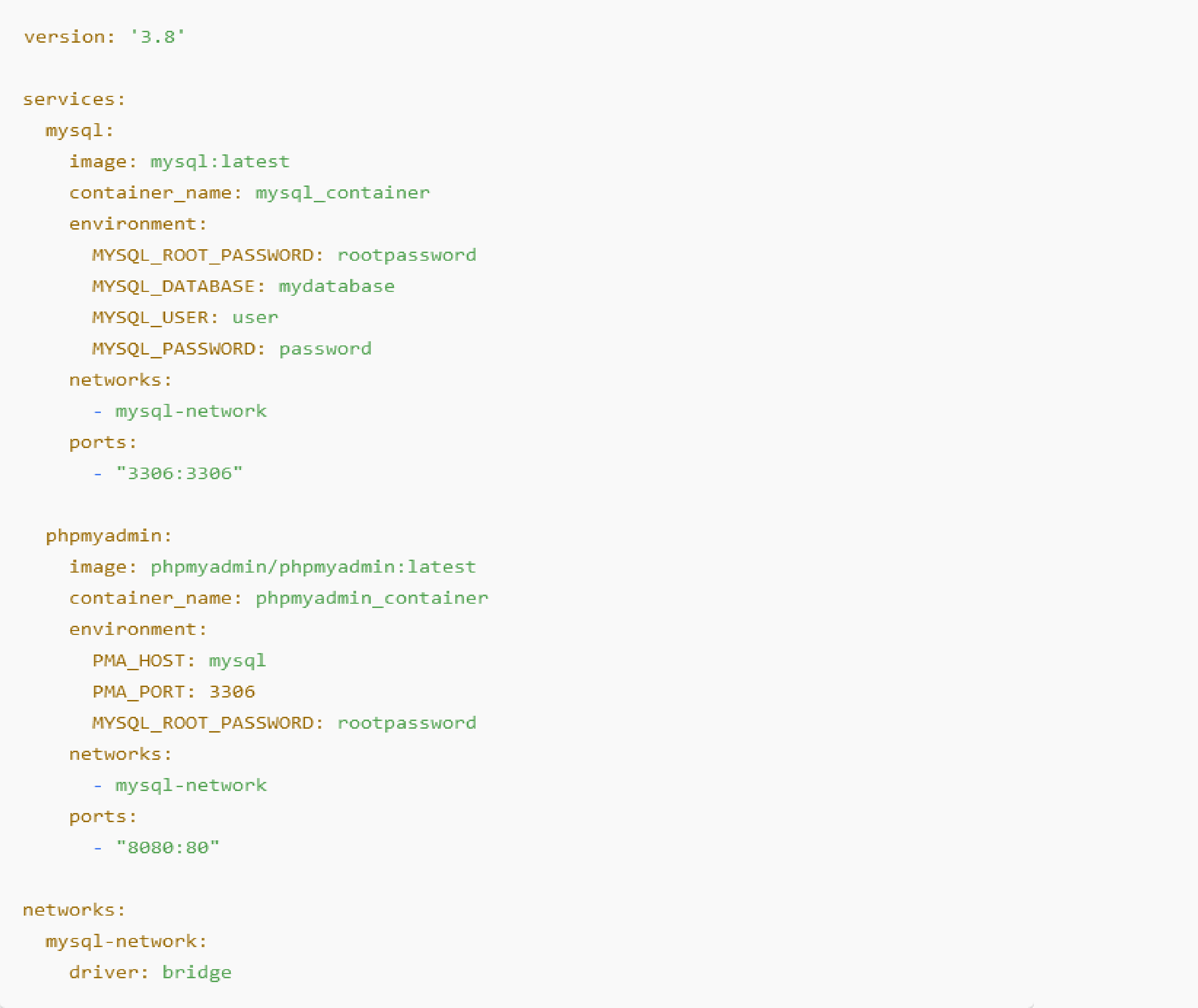
Container

MySQL and phpMyAdmin - Dockerfile



```
# Use the official PHP with Apache base image
FROM php:7.4-apache


# Install MySQLi extension for PHP
RUN docker-php-ext-install mysqli


# Copy the PHP application files into the Apache root directory
COPY ./html /var/www/html


# Expose port 80 for the Apache web server
EXPOSE 80
```

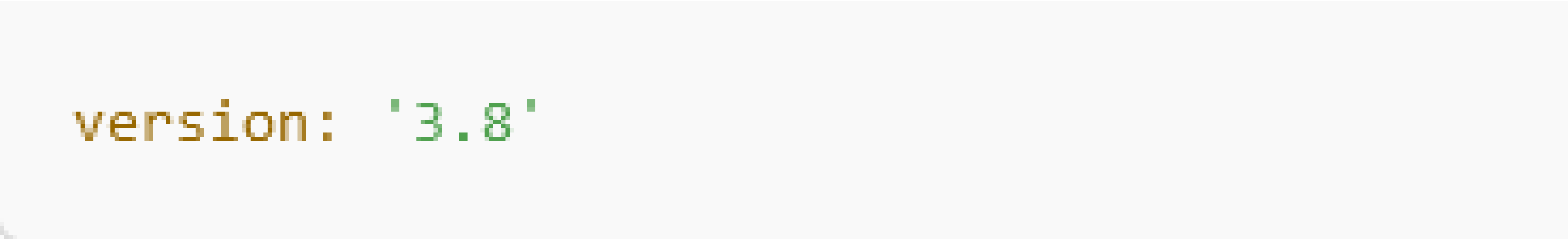
MySQL and phpMyAdmin - Dockercompose



```yaml
version: '3.8'

services:
  mysql:
    image: mysql:latest
    container_name: mysql_container
    environment:
      MYSQL_ROOT_PASSWORD: rootpassword
      MYSQL_DATABASE: mydatabase
      MYSQL_USER: user
      MYSQL_PASSWORD: password
    networks:
      - mysql-network
    ports:
      - "3306:3306"

  phpmyadmin:
    image: phpmyadmin/phpmyadmin:latest
    container_name: phpmyadmin_container
    environment:
      PMA_HOST: mysql
      PMA_PORT: 3306
      MYSQL_ROOT_PASSWORD: rootpassword
    networks:
      - mysql-network
    ports:
      - "8080:80"

networks:
  mysql-network:
    driver: bridge
```

Version

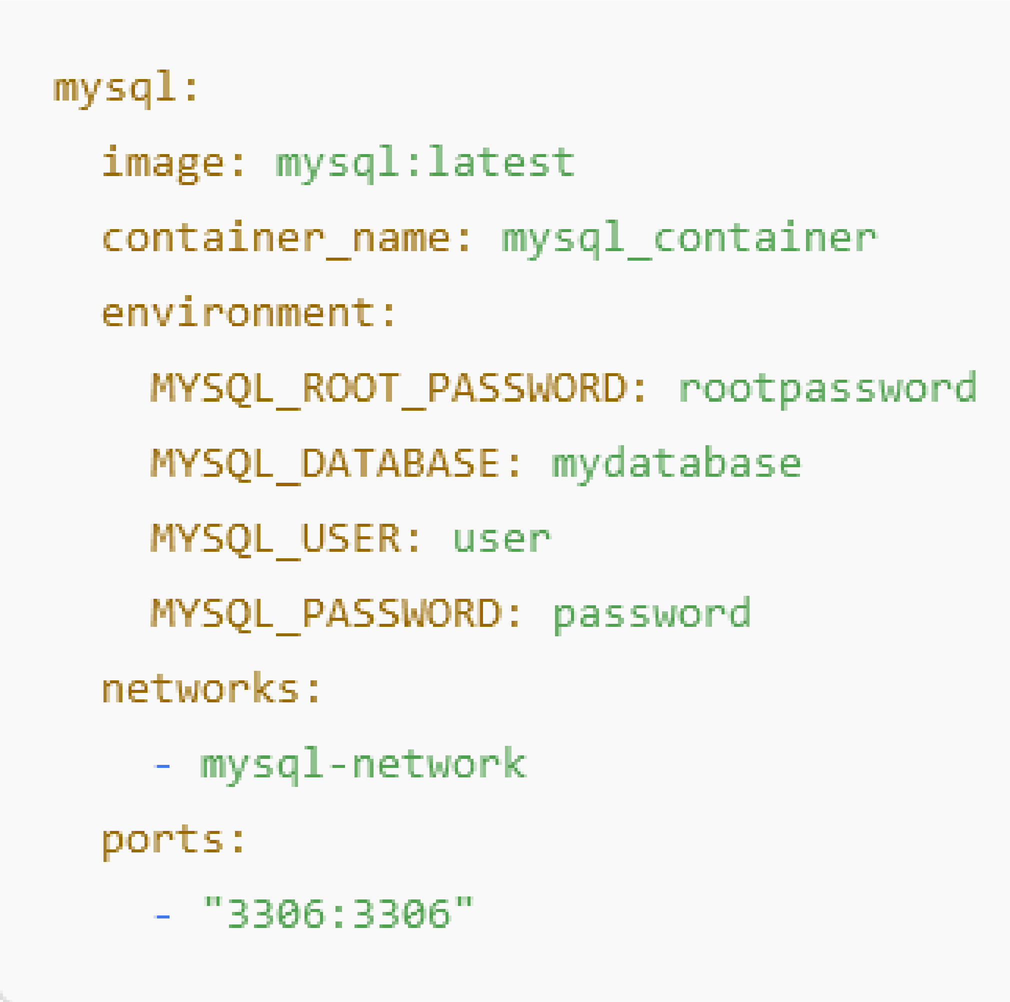
- This specifies the version of the Docker Compose file syntax. Version 3.8 is a stable version that supports various networking and service configuration options.



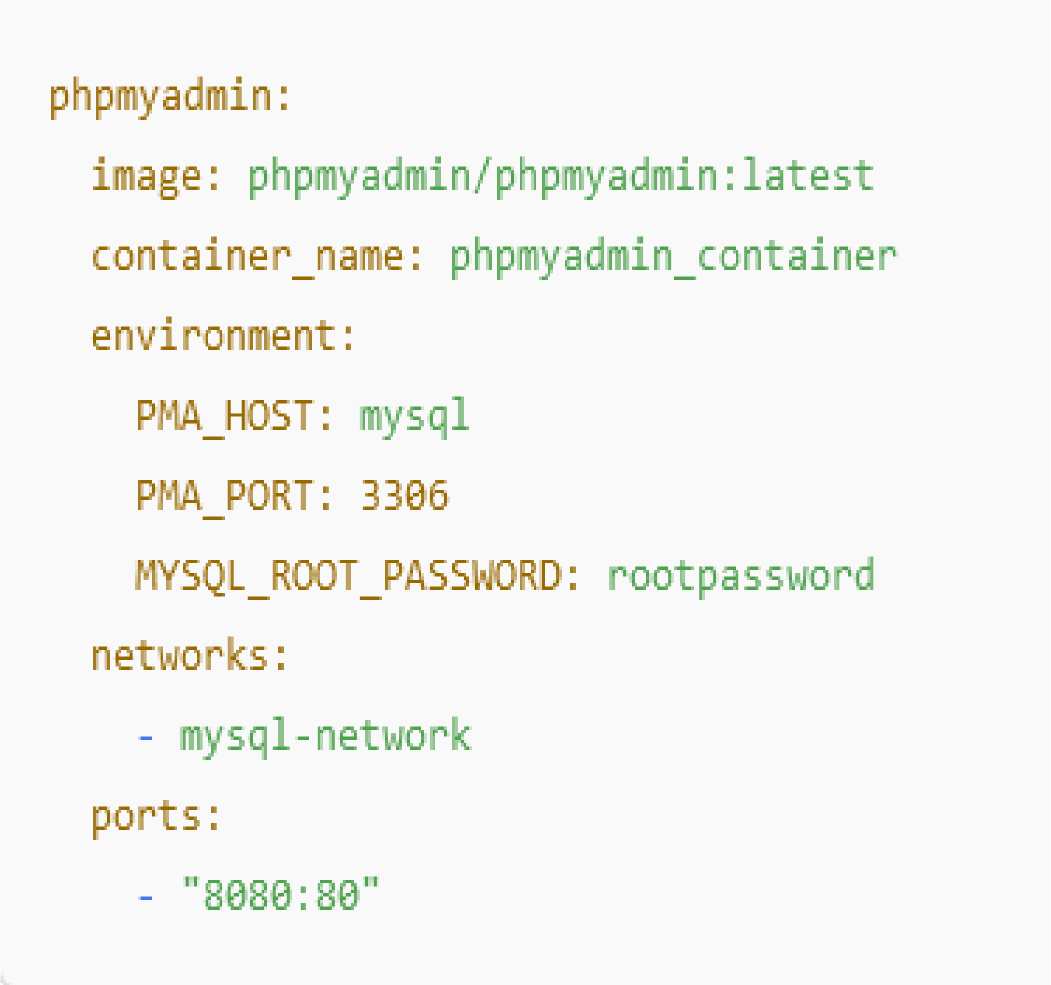
```
version: '3.8'
```

Services

- The services section defines the individual containers (or services) that will be created and managed.



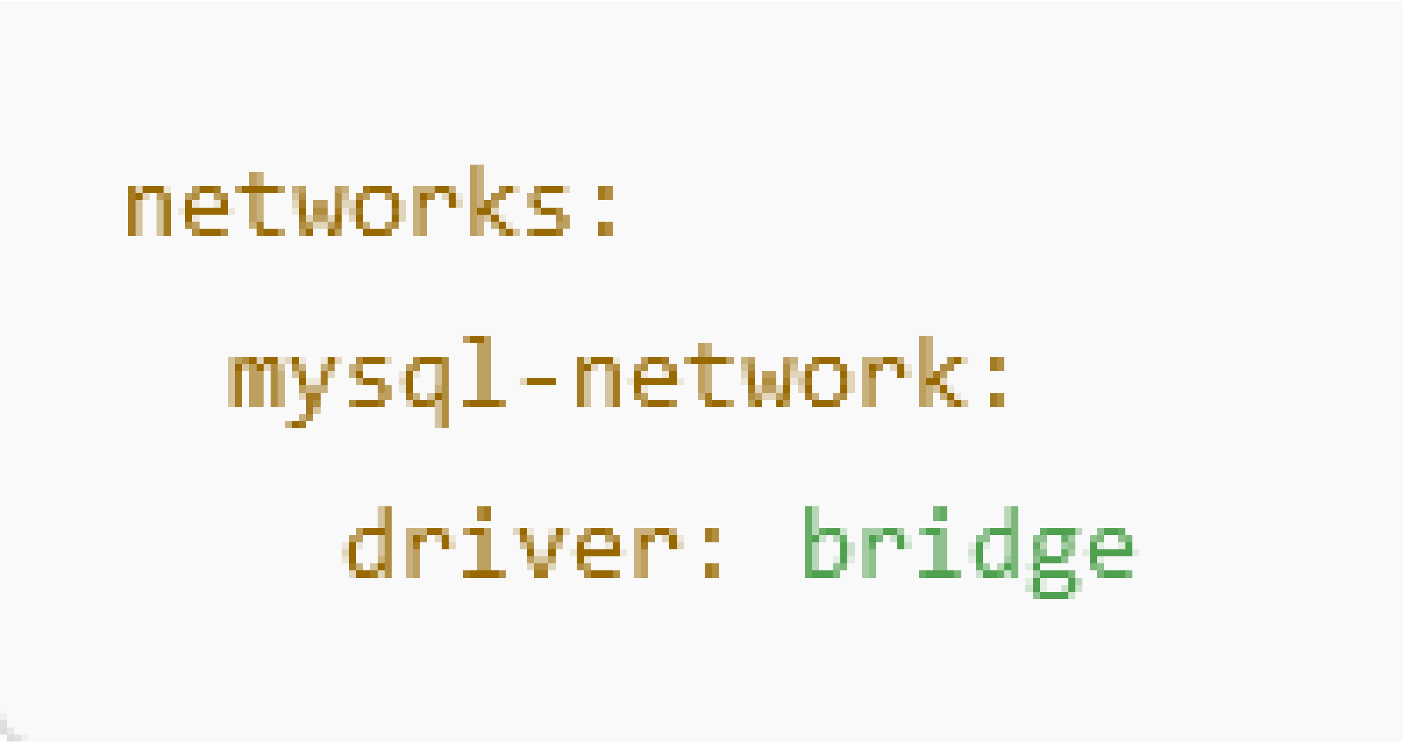
```
mysql:
  image: mysql:latest
  container_name: mysql_container
  environment:
    MYSQL_ROOT_PASSWORD: rootpassword
    MYSQL_DATABASE: mydatabase
    MYSQL_USER: user
    MYSQL_PASSWORD: password
  networks:
    - mysql-network
  ports:
    - "3306:3306"
```



```
phpmyadmin:
  image: phpmyadmin/phpmyadmin:latest
  container_name: phpmyadmin_container
  environment:
    PMA_HOST: mysql
    PMA_PORT: 3306
    MYSQL_ROOT_PASSWORD: rootpassword
  networks:
    - mysql-network
  ports:
    - "8080:80"
```

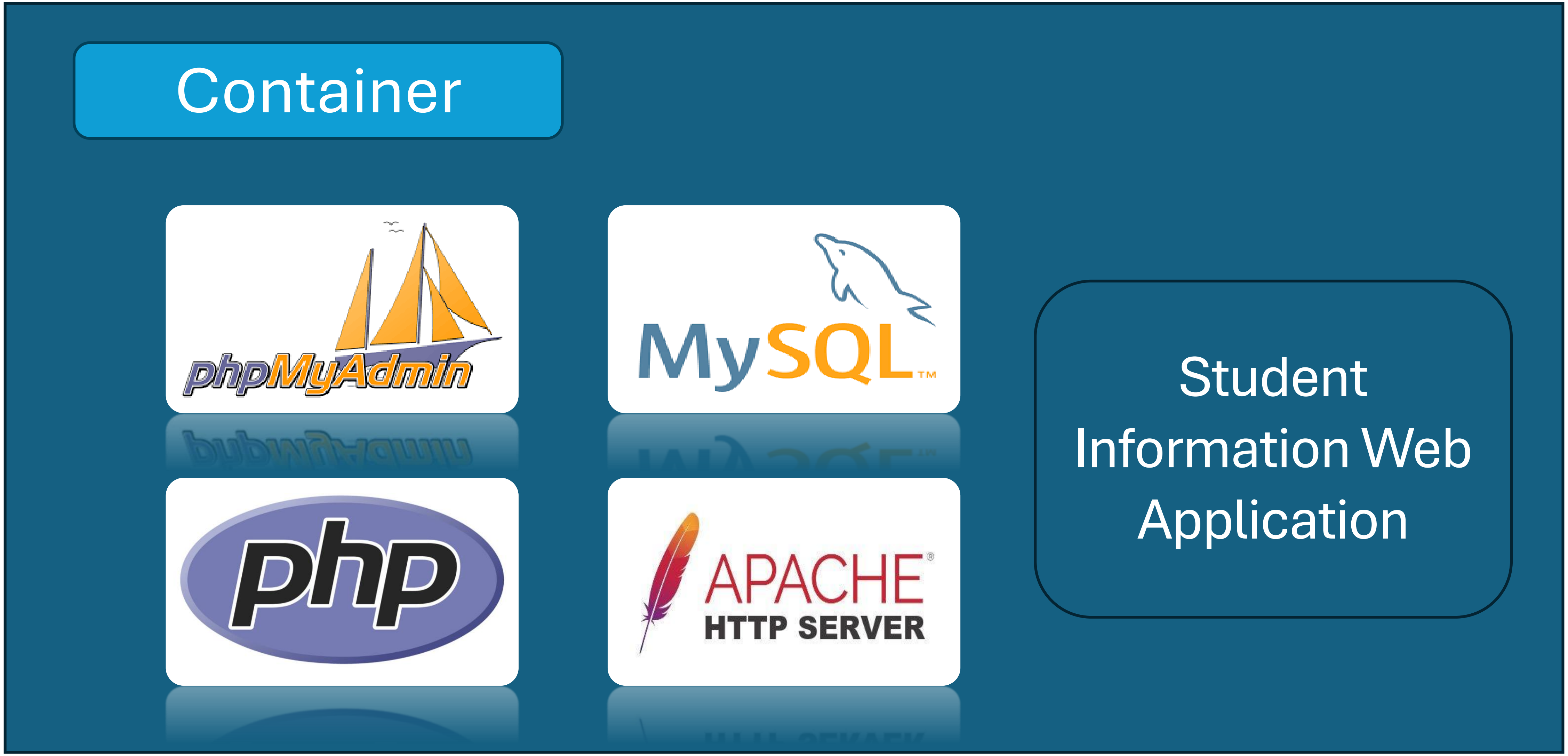
Network

- a **network** is a virtual layer that allows containers to communicate with each other and, optionally, with external networks such as the host machine's network or the internet. Docker networks help manage and control how containers interact and provide a secure way for them to communicate.



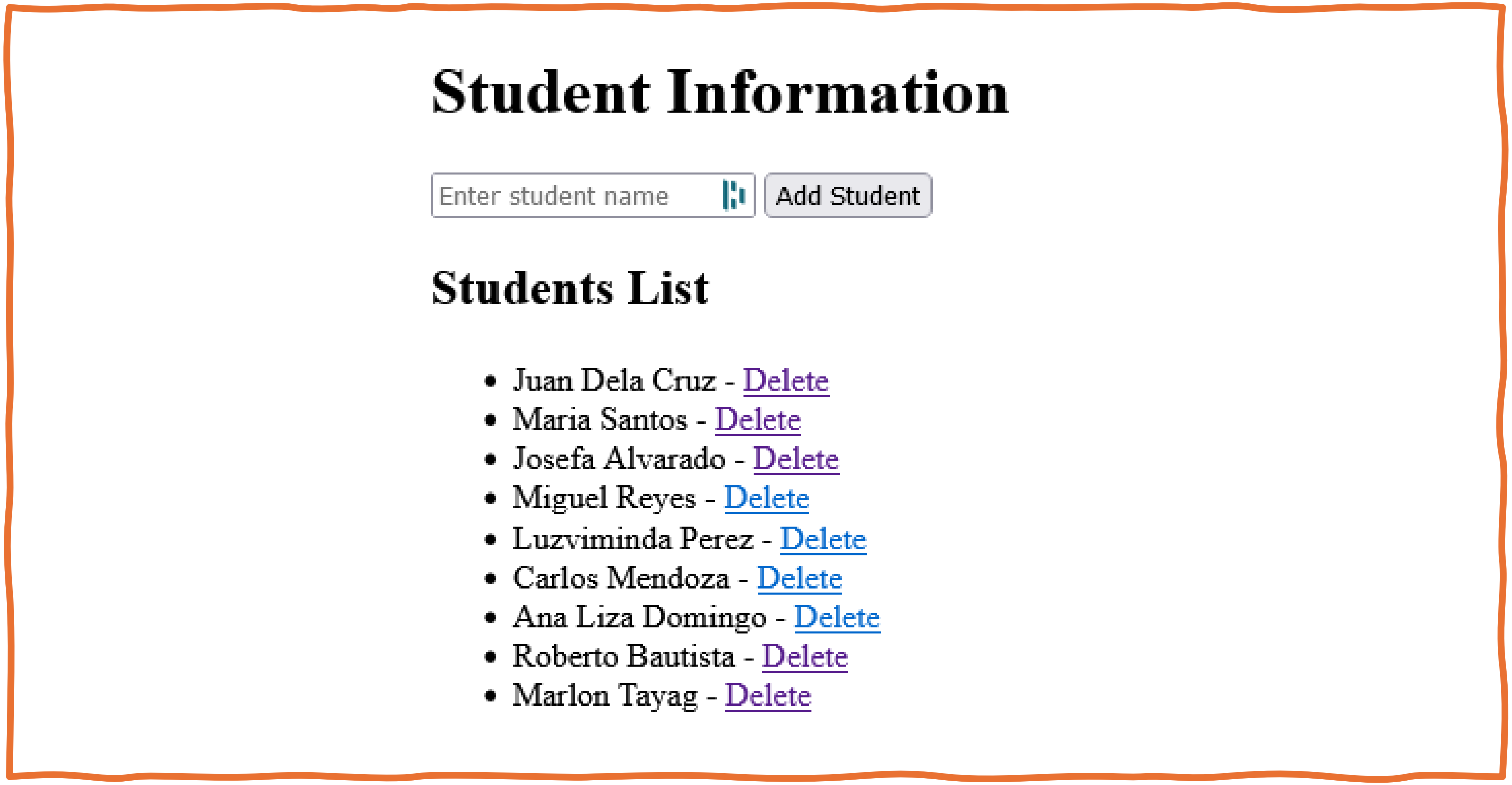
```yaml
networks:
  mysql-network:
    driver: bridge
```

Student Information Web Application with Docker



Container

Student Information Web Application



Student Information

Students List

- Juan Dela Cruz - Delete
- Maria Santos - Delete
- Josefa Alvarado - Delete
- Miguel Reyes - Delete
- Luzviminda Perez - Delete
- Carlos Mendoza - Delete
- Ana Liza Domingo - Delete
- Roberto Bautista - Delete
- Marlon Tayag - Delete

Objectives

1

Set up and deploy a Dockerized environment with MySQL, PHP, Apache, and phpMyAdmin.

2

Develop a PHP application that stores and manages student information in a persistent MySQL database.

3

Use Docker Compose to streamline the setup and deployment process.

4

Push the Docker image to a free repository for public access.

Part 1: Setting up the Project Directory

Create Project Directory

- Open Command Prompt and create a new directory for your application. Type the command:

```
mkdir student-app && cd student-app
```

Create Subdirectories

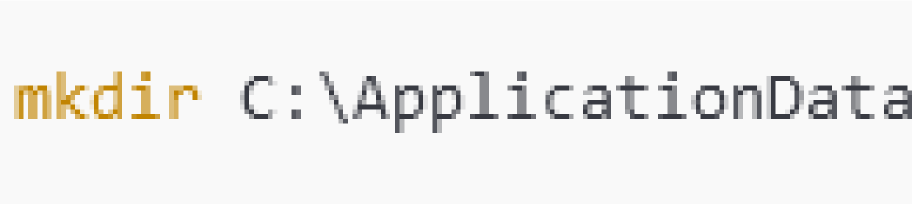
- Within the **`student-app`** directory, create the following folders:
    - src
    - mysql_data

```
mkdir src mysql_data
```

The src folder will contain your PHP application code. mysql_data will act as the persistent storage, linked to your Windows directory C:\ApplicationData.

Create Volume Directory for Persistent Data

- Create a directory for MySQL data on your Windows machine, to ensure data persists even if the container is removed.

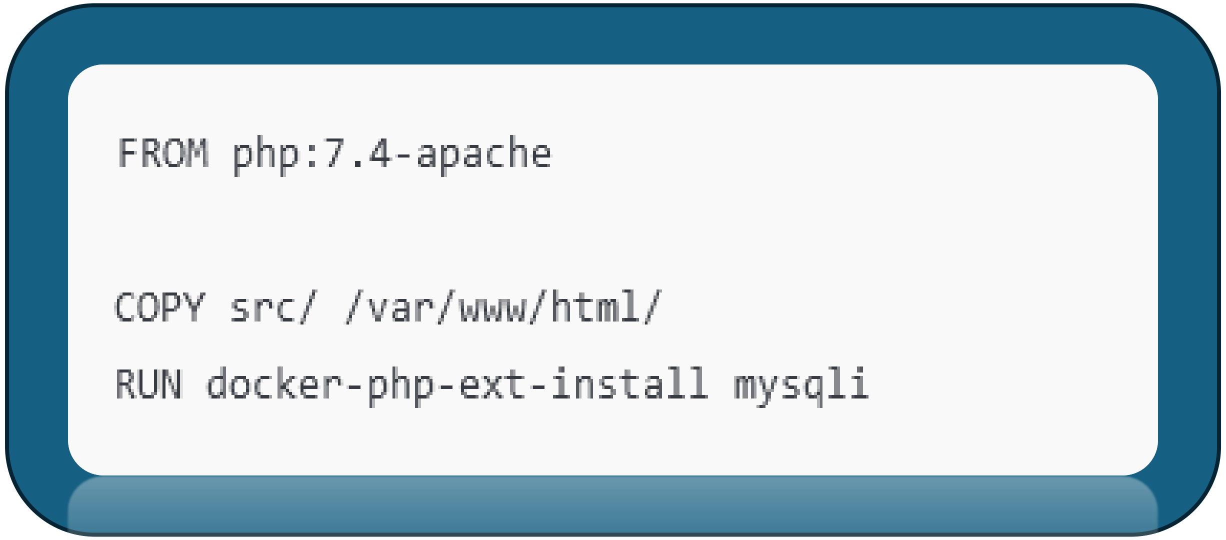


```
mkdir C:\ApplicationData
```

Part 2: Create a Dockerfile for PHP and Apache

Create a Dockerfile

- Inside the student-app directory, create a file named Dockerfile and open it in a text editor.
- Add Instructions to Dockerfile. Use the following configuration to set up the Docker environment with PHP, Apache, and MySQL.

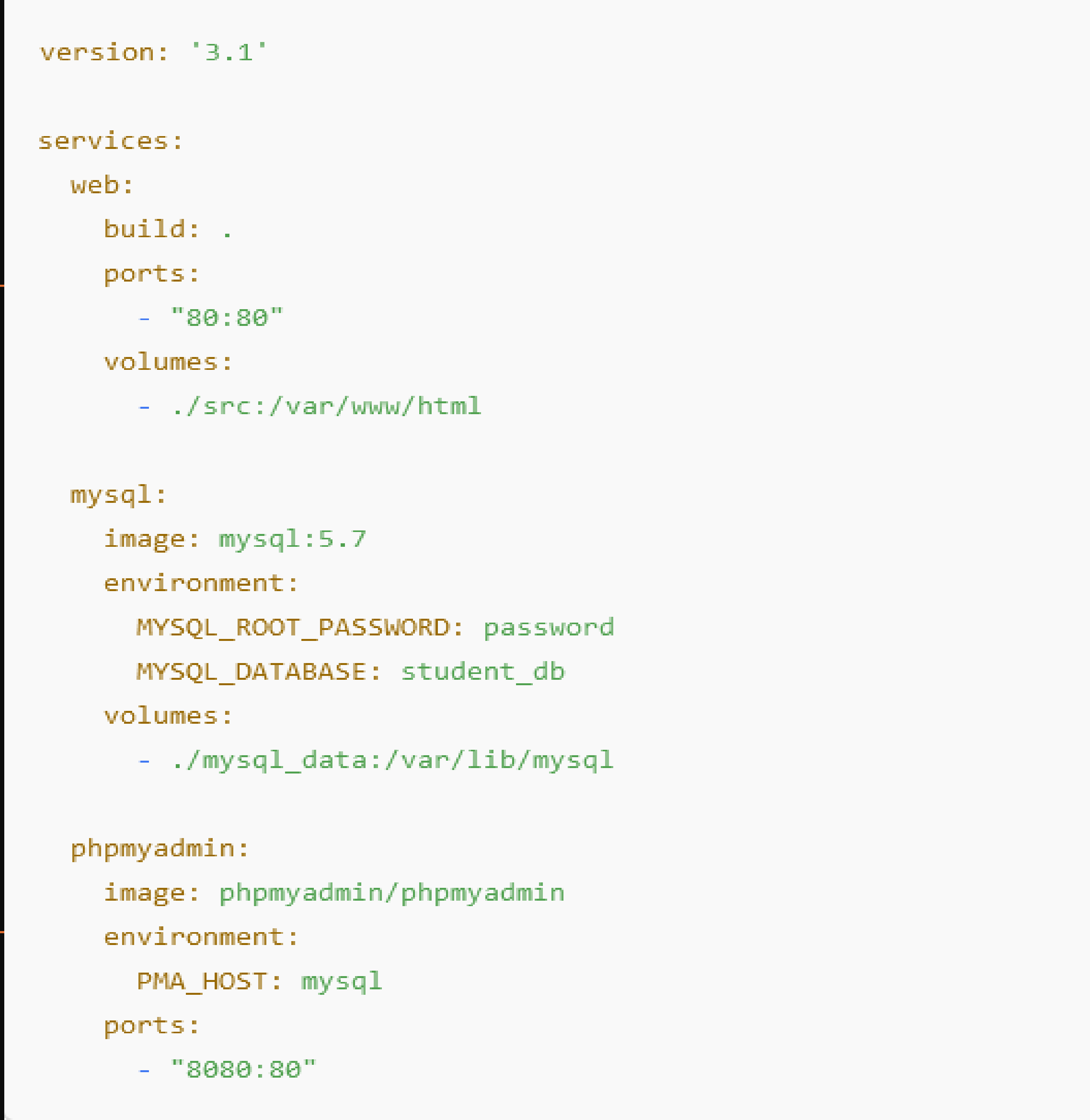


```
FROM php:7.4-apache


COPY src/ /var/www/html/
RUN docker-php-ext-install mysqli
```

Part 3: Create the MySQL Database and Docker Compose File

Create the **docker-compose.yml** file

- Inside the student-app directory, create a file named **docker-compose.yml** and open it with a text editor.
- Add the following configuration to set up the services:

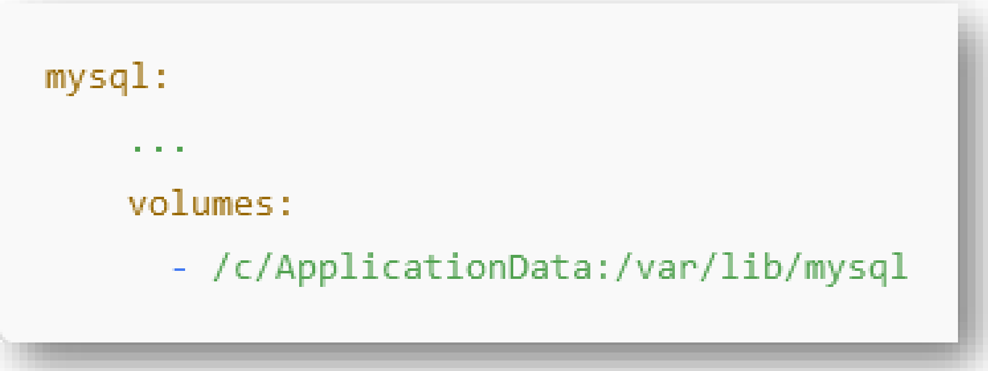


```yaml
version: '3.1'

services:
  web:
    build: .
    ports:
      - "80:80"
    volumes:
      - ./src:/var/www/html

  mysql:
    image: mysql:5.7
    environment:
      MYSQL_ROOT_PASSWORD: password
      MYSQL_DATABASE: student_db
    volumes:
      - ./mysql_data:/var/lib/mysql

  phpmyadmin:
    image: phpmyadmin/phpmyadmin
    environment:
      PMA_HOST: mysql
    ports:
      - "8080:80"
```

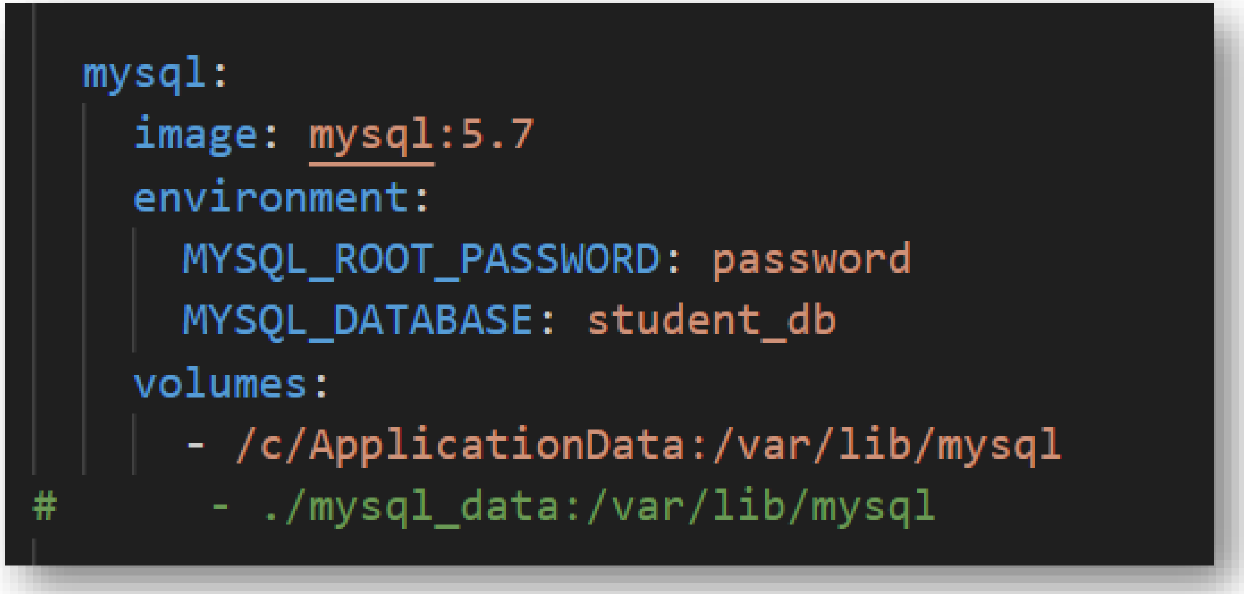
Part 4: Configure the Volume Persistence

Configure the Volume Persistence

- To ensure MySQL data persists, bind the mysql_data volume to C:\ApplicationData. Edit docker-compose.yml:



```
mysql:
    ...
    volumes:
      - /c/ApplicationData:/var/lib/mysql
```

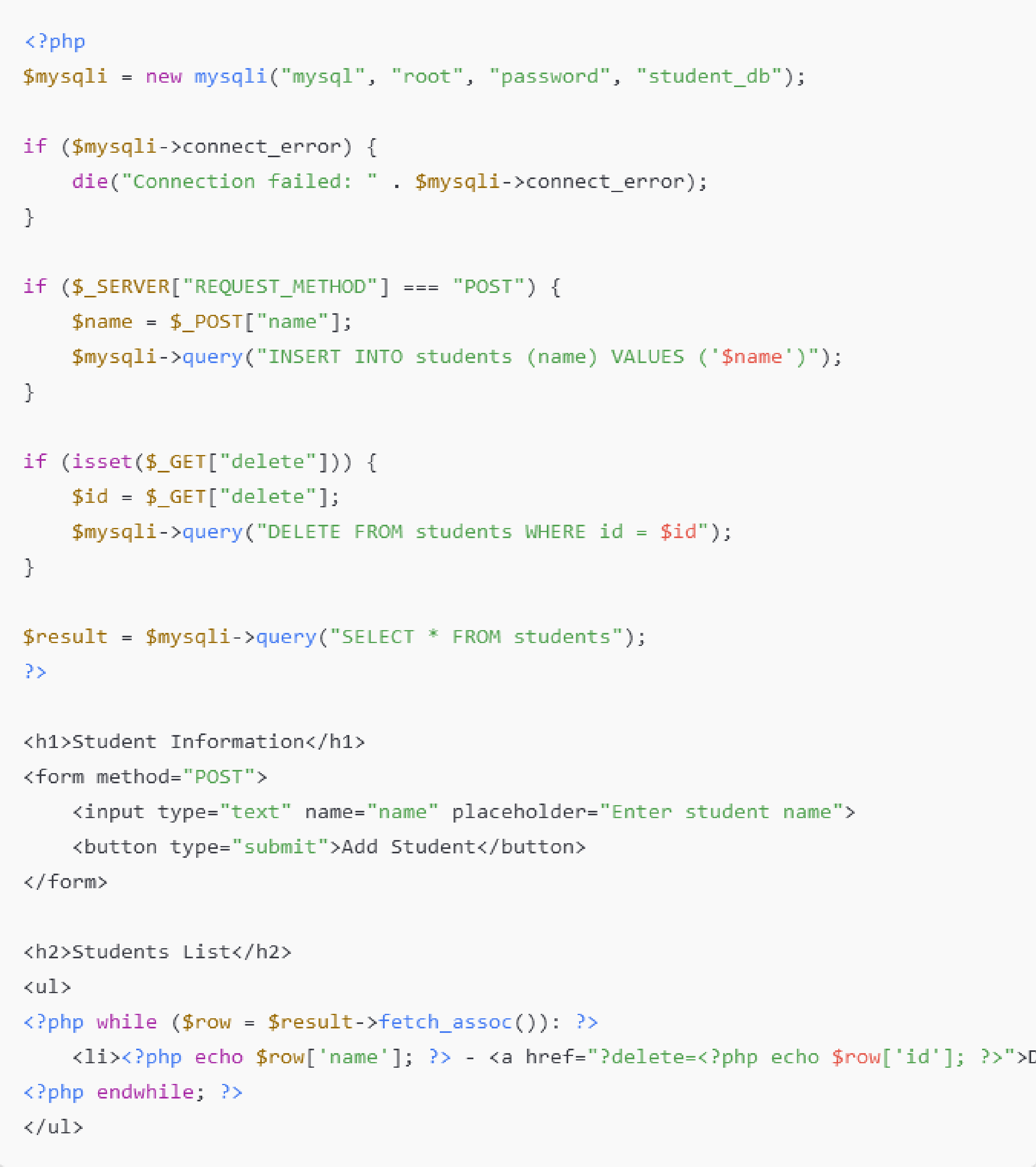


```
  mysql:
    image: mysql:5.7
    environment:
      MYSQL_ROOT_PASSWORD: password
      MYSQL_DATABASE: student_db
    volumes:
      - /c/ApplicationData:/var/lib/mysql
#      - ./mysql_data:/var/lib/mysql
```

Part 5: Developing the PHP Application

Create PHP Application Files

- In the **src** folder, create an **index.php** file with code for displaying, adding, and deleting student records

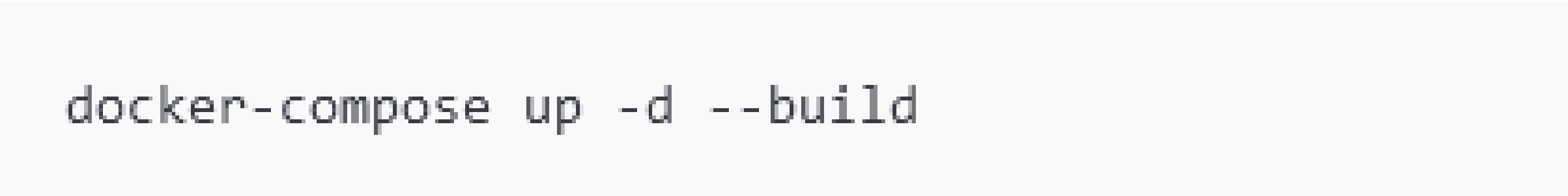


```
<?php
$mysqli = new mysqli("mysql", "root", "password", "student_db");

if ($mysqli->connect_error) {
    die("Connection failed: " . $mysqli->connect_error);
}

if ($_SERVER["REQUEST_METHOD"] === "POST") {
    $name = $_POST["name"];
    $mysqli->query("INSERT INTO students (name) VALUES ('$name')");
}

if (isset($_GET["delete"])) {
    $id = $_GET["delete"];
    $mysqli->query("DELETE FROM students WHERE id = $id");
}

$result = $mysqli->query("SELECT * FROM students");
?>

<h1>Student Information</h1>
<form method="POST">
    <input type="text" name="name" placeholder="Enter student name">
    <button type="submit">Add Student</button>
</form>

<h2>Students List</h2>
<ul>
<?php while ($row = $result->fetch_assoc()): ?>
    <li><?php echo $row['name']; ?> - <a href="?delete=<?php echo $row['id']; ?>">D
<?php endwhile; ?>
</ul>
```

Part 6: Build and Run the Application with Docker Compose

Build the Image and Container

- Run Docker Compose to build and start the containers:

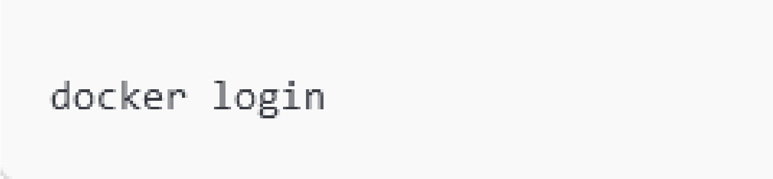


```
docker-compose up -d --build
```

Part 7: Push the Docker Image to a Repository

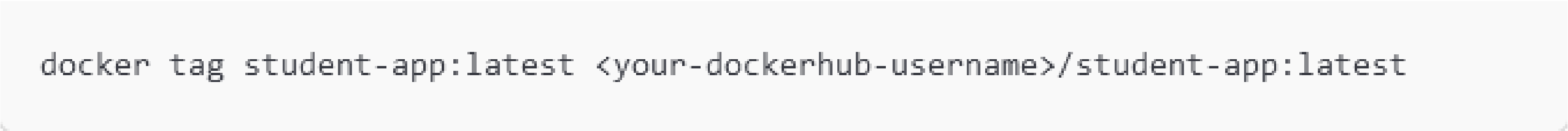
Push the Docker Image to a Repository

- Log in to Docker Hub or any free Docker image repository.



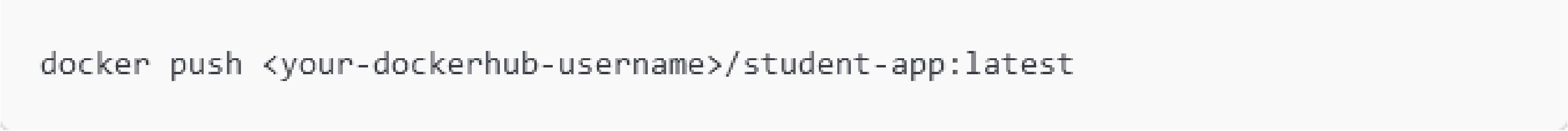
```
docker login
```

- Tag the image to push to your repository:



```
docker tag student-app:latest <your-dockerhub-username>/student-app:latest
```

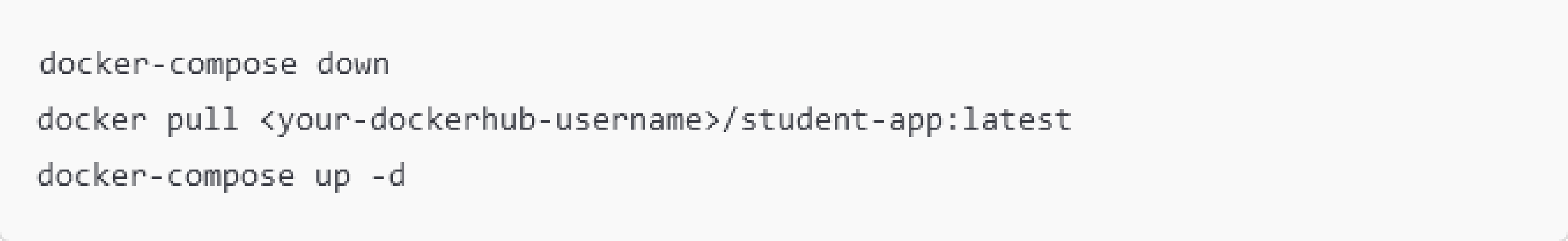
- Push the image:



```
docker push <your-dockerhub-username>/student-app:latest
```

Test Pull and Deployment from Repository

- On another machine or after deleting the existing images, pull and deploy:



```
docker-compose down
docker pull <your-dockerhub-username>/student-app:latest
docker-compose up -d
```

- Verify that all required containers (PHP app, MySQL, and phpMyAdmin) start automatically with the data persisted.

Final Hands-On Activity

- Push to Repository: Follow Step 7 to push the final image to a Docker repository.
- Documentation: Document each step with screenshots to track progress and validate each step for future reference.
- Testing: Confirm functionality on a fresh machine by pulling the image and deploying it with Docker Compose.

Thank you

Marlon I. Tayag