Machine Learning Operations (MLOps)
Docker Images and Containers, with CI/CD Pipeline

Zeham Management Technologies BootCamp

by SDAIA

October the 1st, 2024

SDAIA

الهيئة السعودية للبيانات والذكاء الاصطناعي

Saudi Data & AI Authority

Objectives

By the end of this module, trainees will have a comprehensive understanding of:

Install and build your own repository.

Learn Imputers and Encoders

Make preprocessing Pipelines.

Train and validate the model.

Evaluate and save your model.

Build an API and connect it with your Machine Learning code

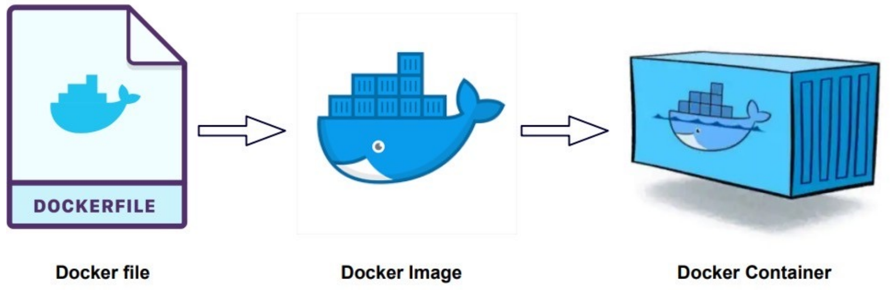
Agenda

- Composing Apps with Docker
- Setting up CI/CD Pipeline
- References

Composing Apps with Docker

What is a Docker ?

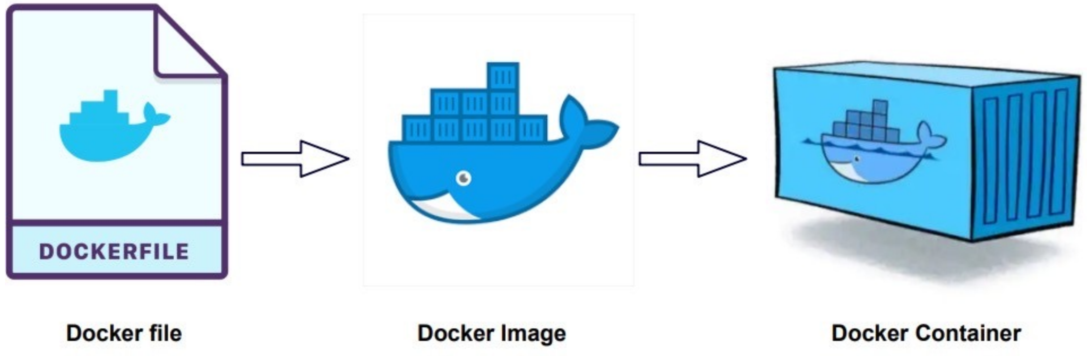
- **Docker** is a group of Platform as a Service (PaaS) products that use operating system-level virtualization to deliver software in packages called containers.
- Docker is an open-source containerization platform that allows you to bundle your application and all its dependencies into a standard unit called a container.



Source

What is a Docker ?

- **Containers** are separate from each other and include their own software, libraries, and configuration files.
- Containers can communicate with each other through specific channels.
- All containers run on a single operating system kernel, so they use fewer resources than virtual machines.

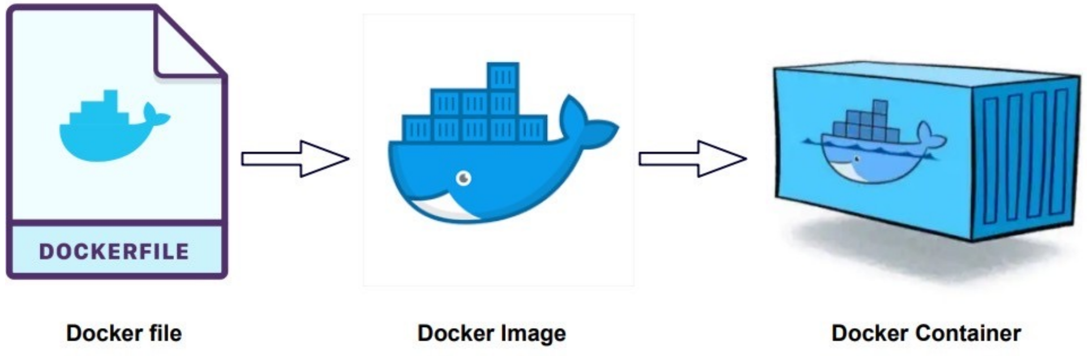


Source

Docker Popularity

- Docker is popular for several reasons:

1. Portability.
2. Reproducibility.
3. Efficiency.
4. Scalability.

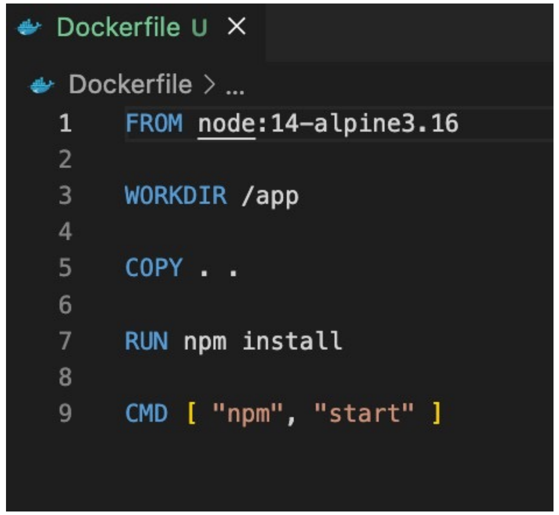


Source

What is Dockerfile ?

- The Dockerfile uses a Domain Specific Language (DSL) and includes instructions for creating a Docker image.
- It outlines the steps needed to quickly build an image.
- When developing your application, you should create a Dockerfile in sequence because the Docker daemon executes all instructions from top to bottom.

Source

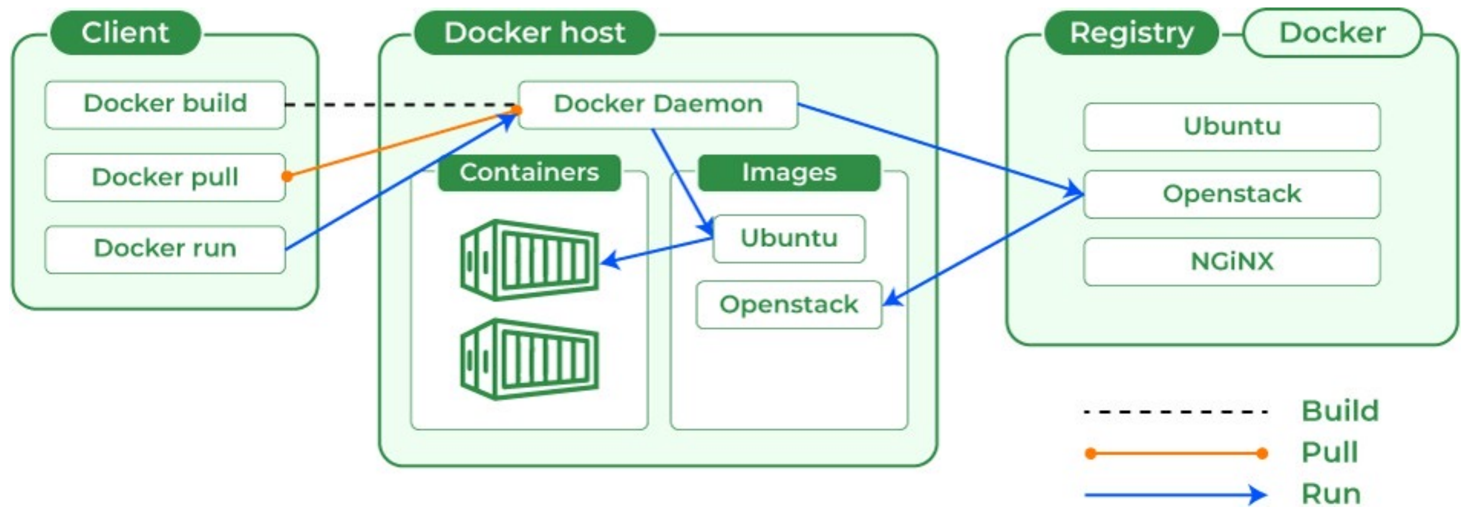


```
FROM node:14-alpine3.16

WORKDIR /app

COPY . .

RUN npm install

CMD [ "npm", "start" ]
```

Source

How Docker works ?

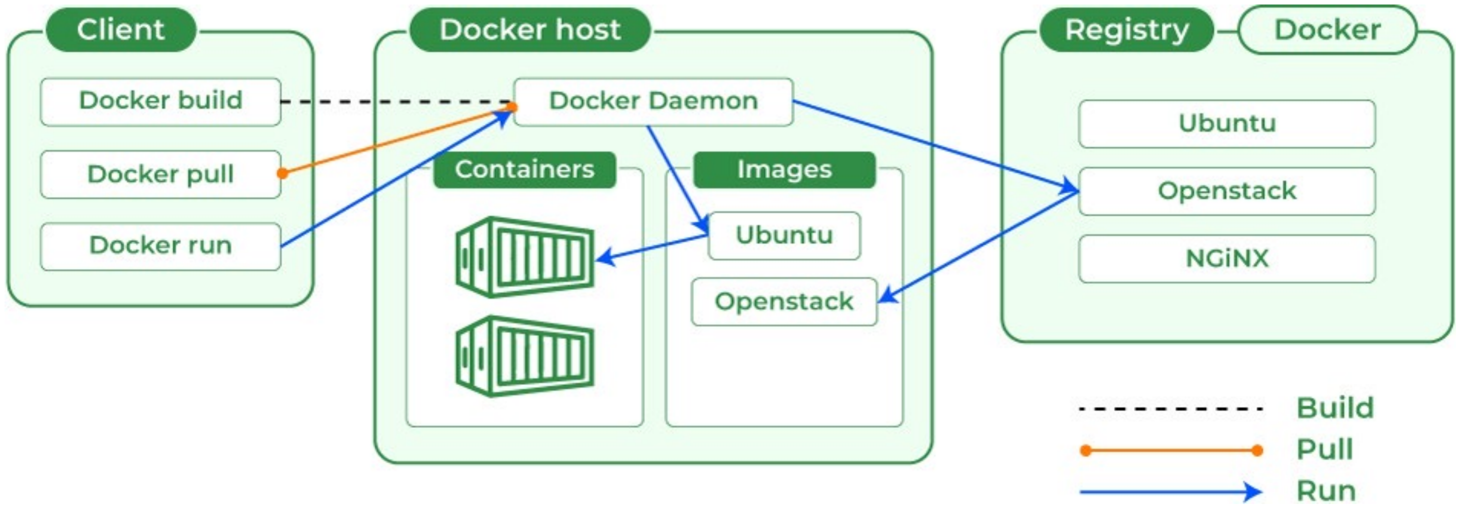
- Docker uses a client-server architecture.
- The Docker client communicates with the Docker daemon, which is responsible for building, running, and distributing Docker containers.



Source

How Docker works ?

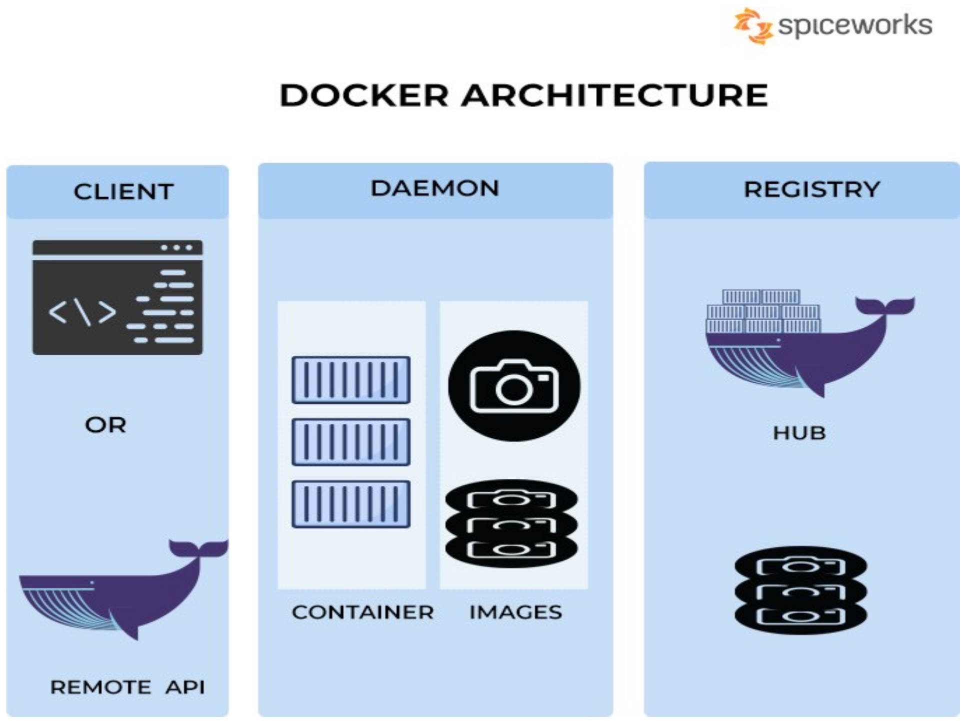
- The Docker client and daemon can run on the same system or connect remotely.
- They interact via a REST API over a UNIX socket or a network.



Source

What is a Docker Image ?

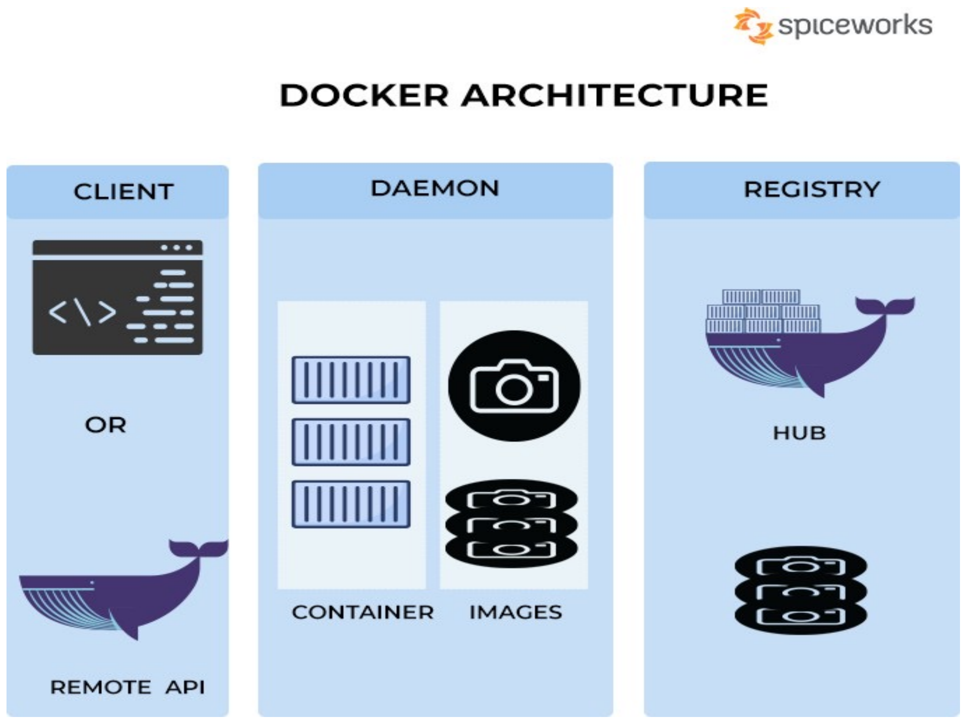
- A Docker image is a file made up of multiple layers, used to execute code in a Docker container.
- It contains a set of instructions for creating Docker containers.
- Essentially, a Docker image is an executable package of software that includes everything needed to run an application.



Source

What is a Docker Image ?

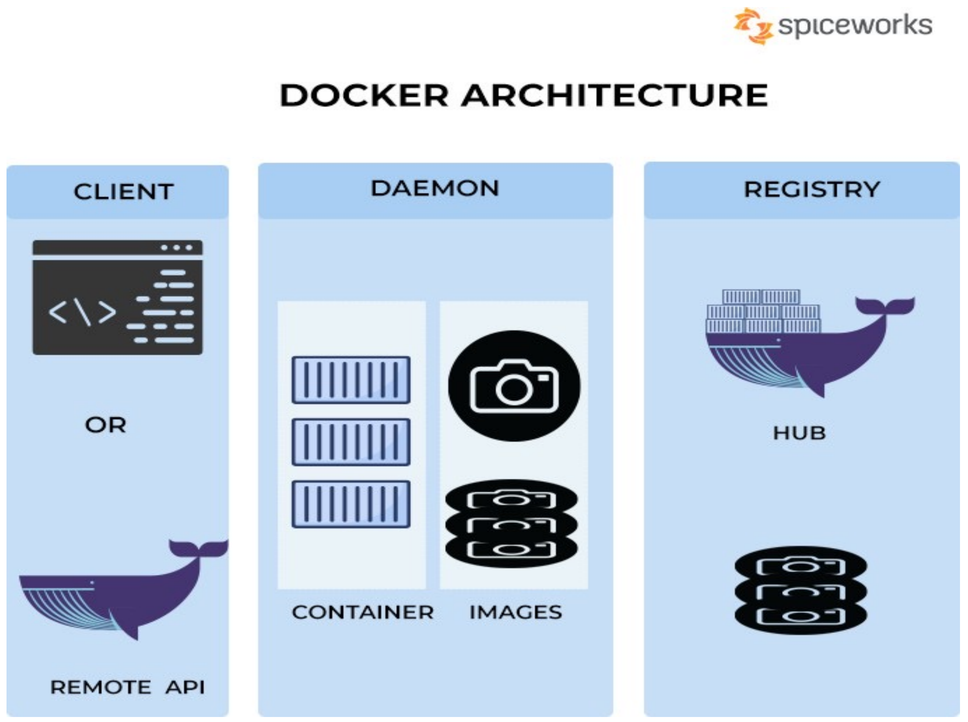
- This image defines how a container should be created, specifying which software components will run and how they will be configured.
- A Docker container, on the other hand, is a virtual environment that packages application code along with all the dependencies required to run it.



Source

What is a Docker Image ?

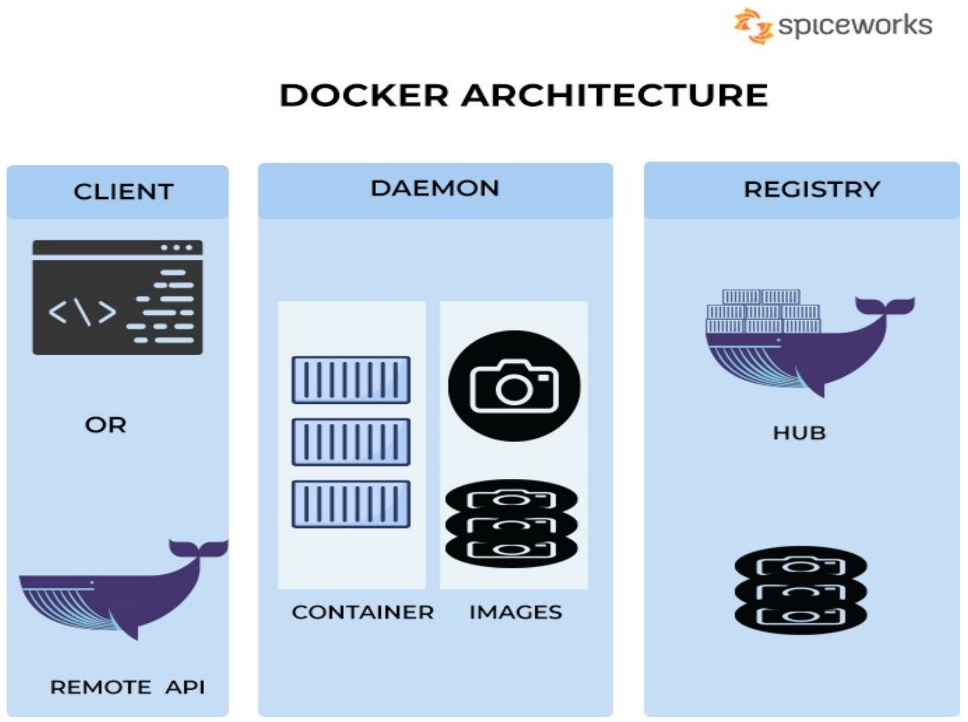
- This ensures that the application runs quickly and reliably across different computing environments.



Source

What is Docker Container ?

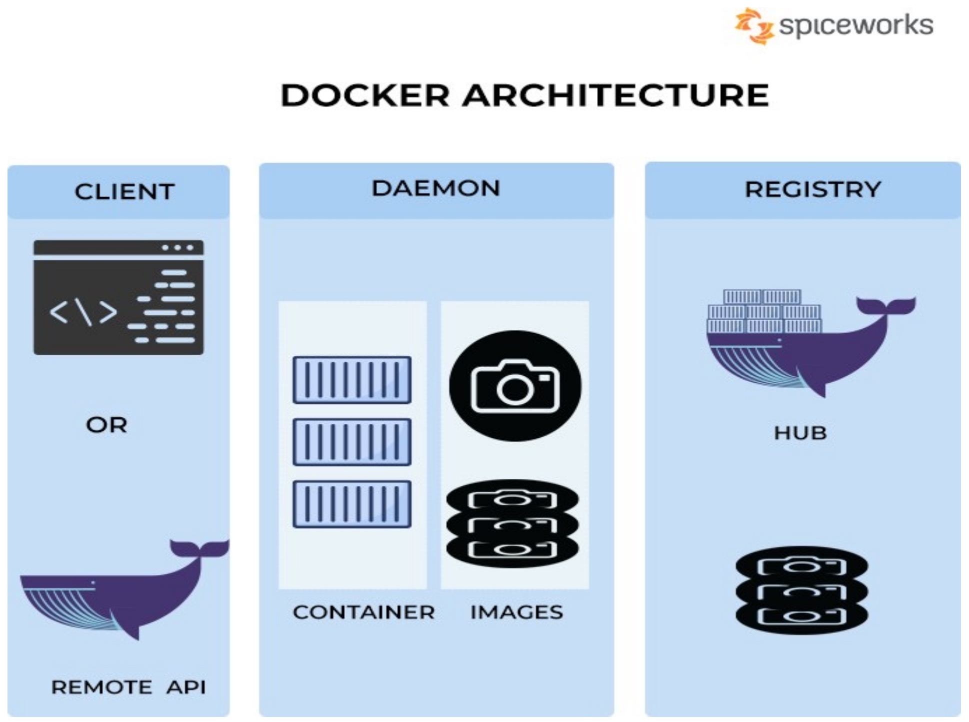
- A Docker container is a runtime instance of an image.
- It allows developers to package applications with all necessary components, such as libraries and dependencies.



Source

What are Docker Containers?

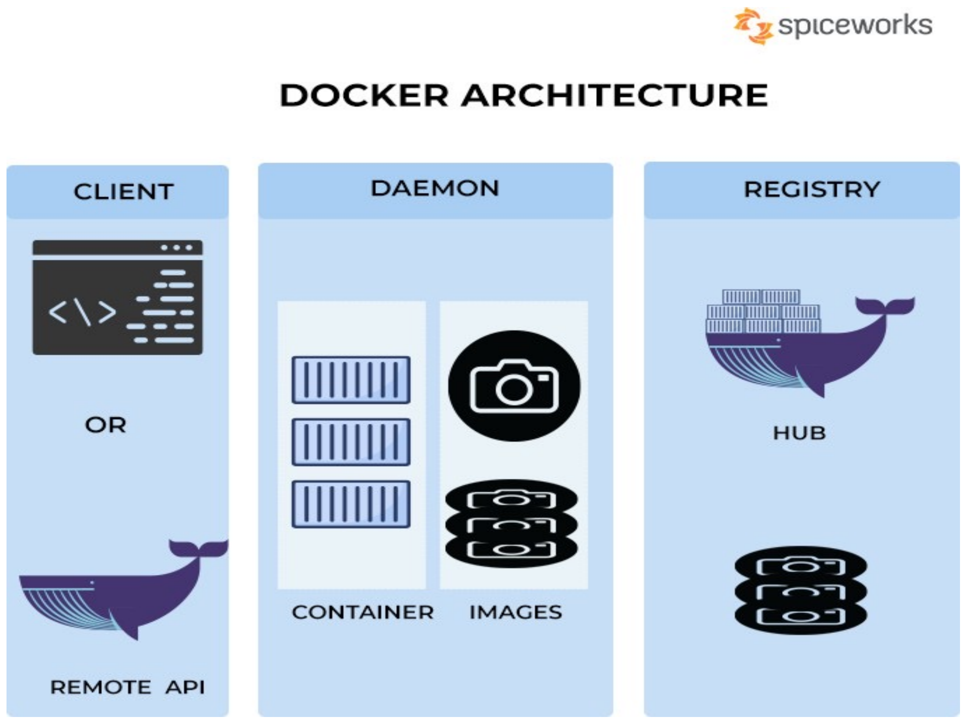
- Containers are self-contained, ensuring the application runs in an isolated environment.
- For example, if you have an image of Ubuntu OS with an NGINX server, running this image with the `docker run` command will create a container where the NGINX server operates on the Ubuntu OS.



Source

What is Docker Hub ?

- Docker Hub is a cloud-based repository service where users can push and pull Docker container images.
- It allows people to upload their container images to the cloud and download them from anywhere with internet access.

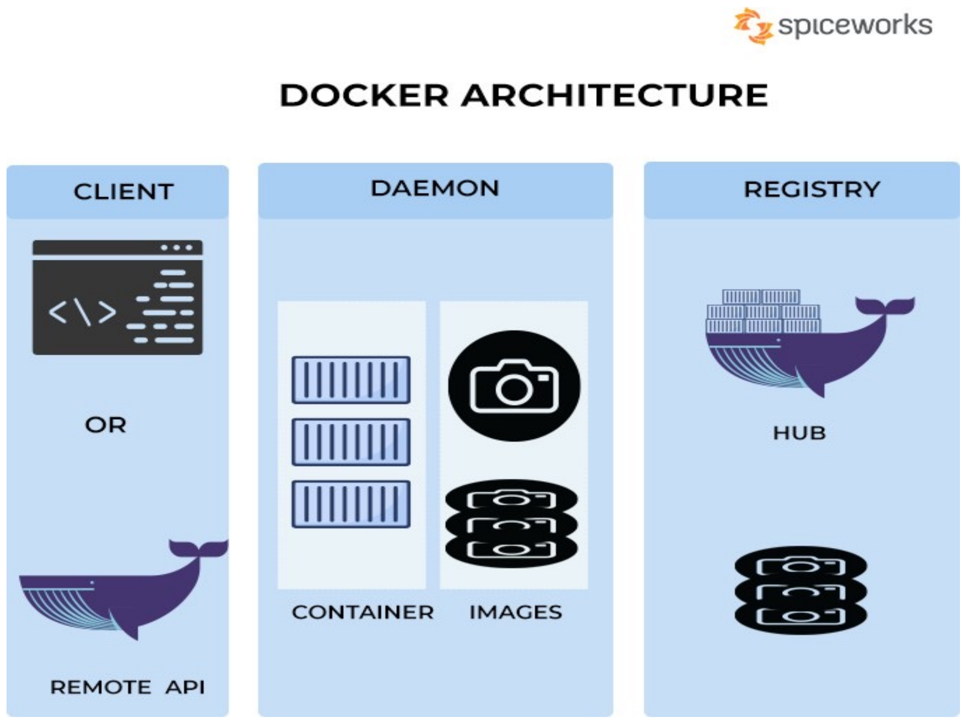


Source

What is Docker Hub ?

- Docker Hub makes it easy to find and reuse images.

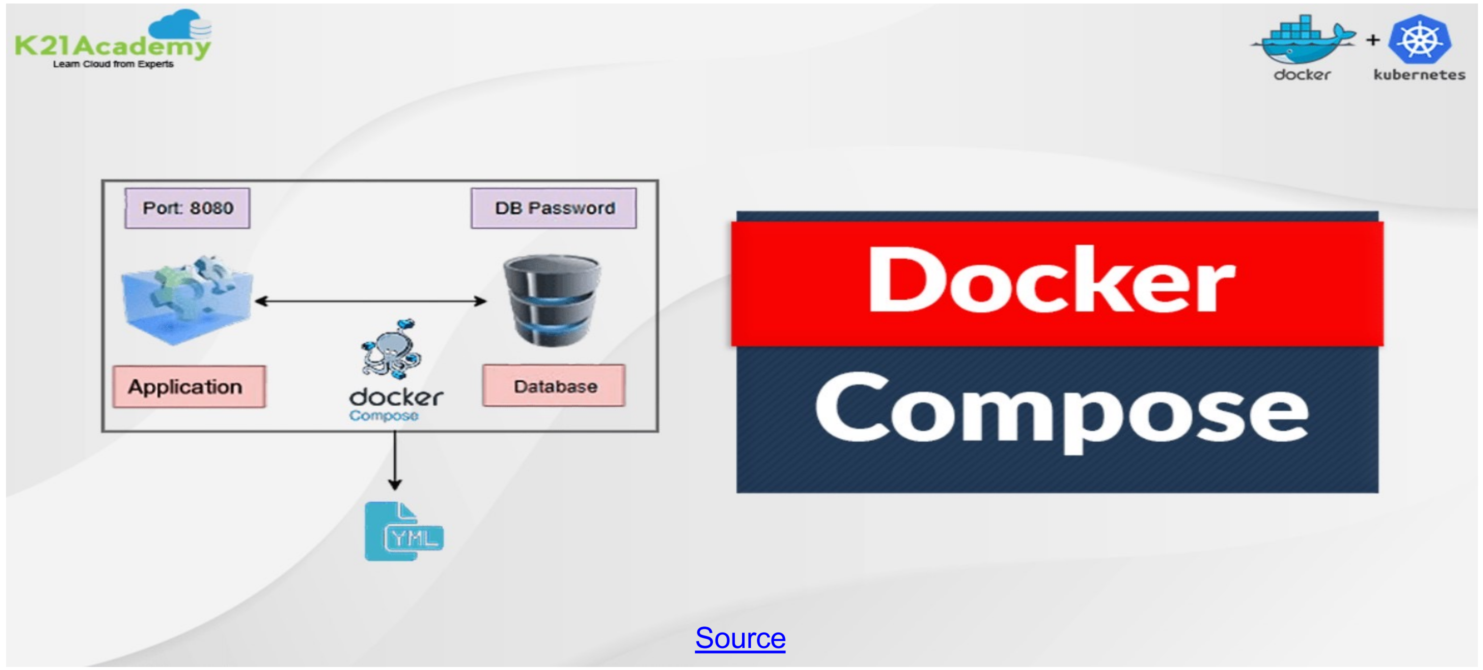
- It also offers features like private and public registries, enabling users to store and share Docker images as needed.



Source

What is Docker Compose ?

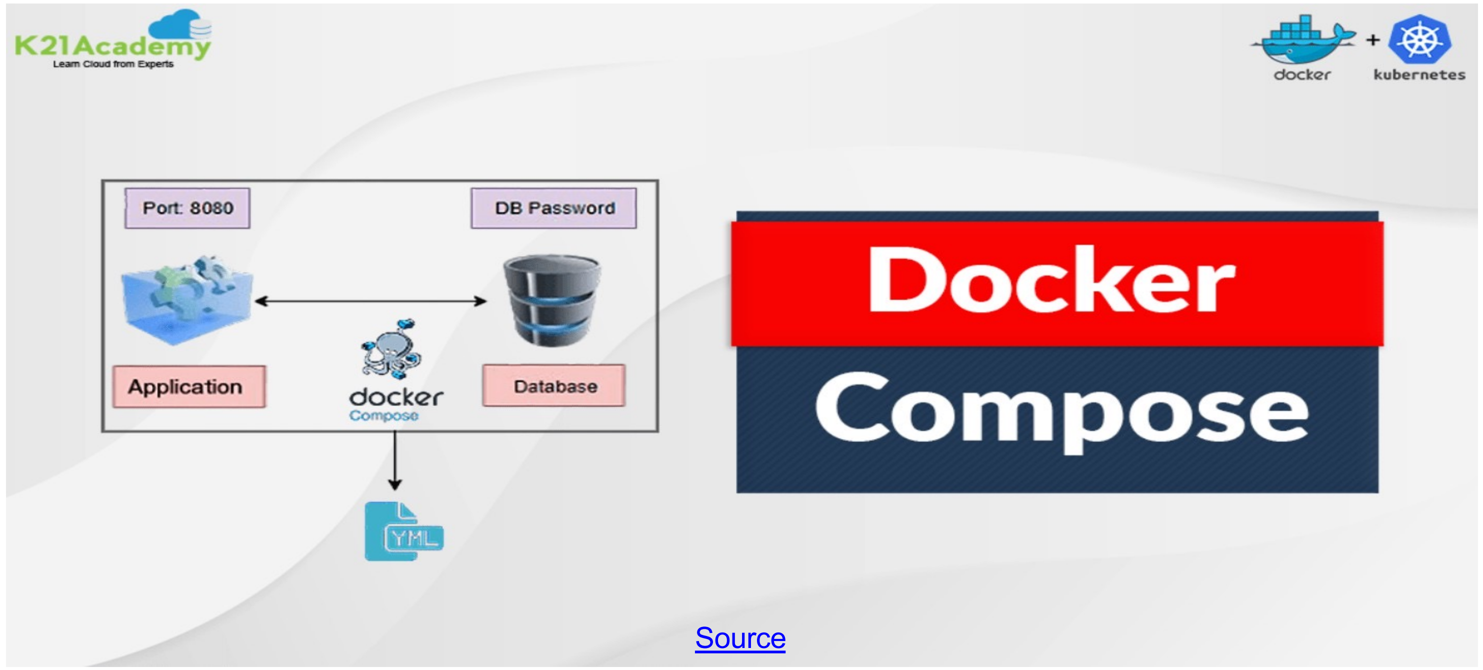
- Docker Compose executes multi-container applications using a YAML file.
- This YAML file includes all the configurations needed to deploy containers.



Source

What is Docker Compose ?

- Integrated with Docker Swarm, Docker Compose provides instructions for building and deploying these containers.
- Each container defined in Docker Compose is designed to run on a single host.

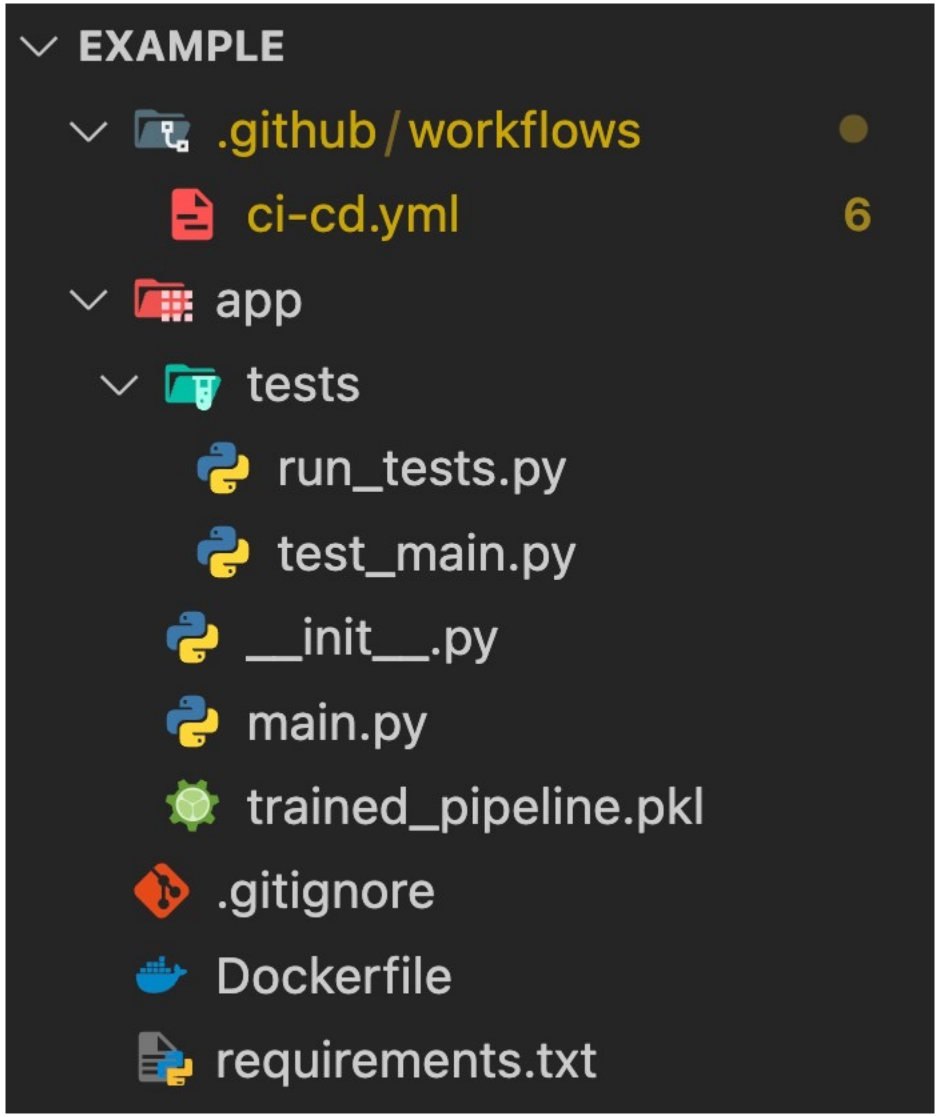


Source

Start your first Docker

To start the first Docker, make sure your file structure like the image shown. Tests is optionally at this step, but you should make tests and test cases for your application.

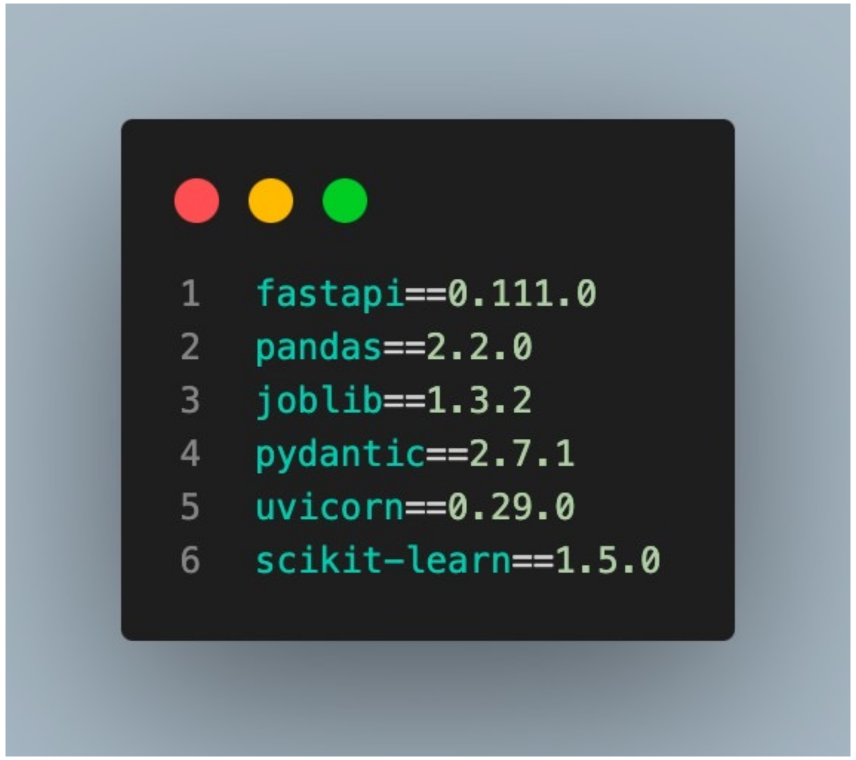
Please have a look at the uploaded example folder.



```
EXAMPLE
  .github/workflows
    ci-cd.yml                6
  app
    tests
      run_tests.py
      test_main.py
    __init__.py
    main.py
    trained_pipeline.pkl
  .gitignore
  Dockerfile
  requirements.txt
```

Start your first Docker

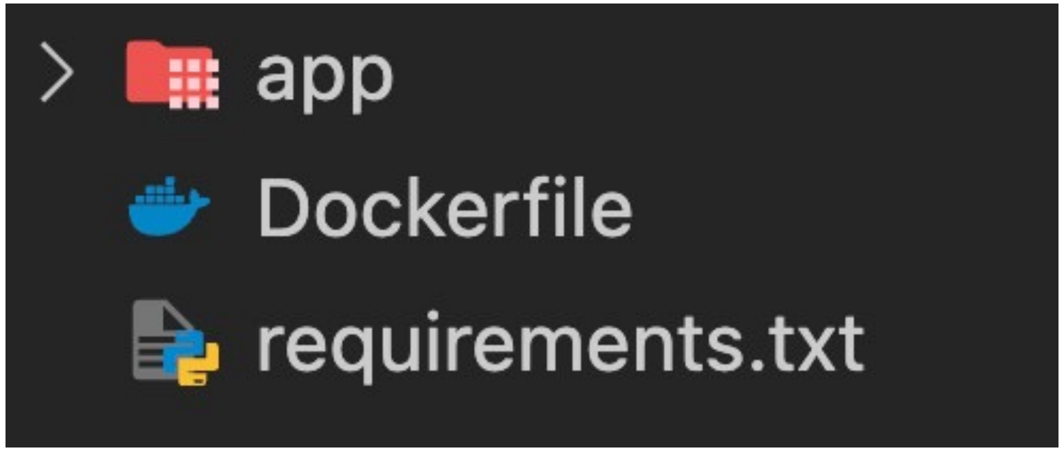
First make sure you create requirements file in the root containing all the needed dependencies, here is an example:



```
fastapi==0.111.0
pandas==2.2.0
joblib==1.3.2
pydantic==2.7.1
uvicorn==0.29.0
scikit-learn==1.5.0
```

Start your first Docker

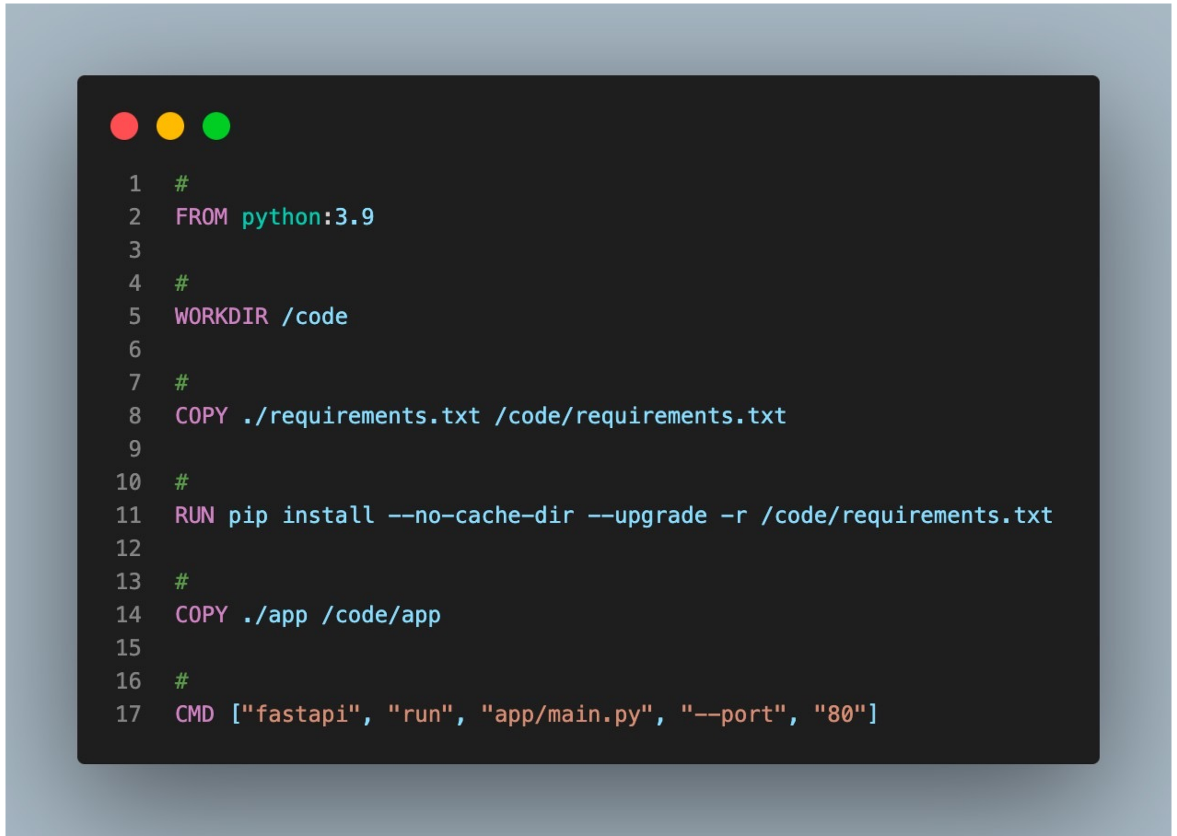
Now you will need to create a file named “Docker” with no extension in the root directory:



```
> app
  Dockerfile
  requirements.txt
```

Start your first Docker

Now you will need to create a file named “Docker” with no extension in the root:

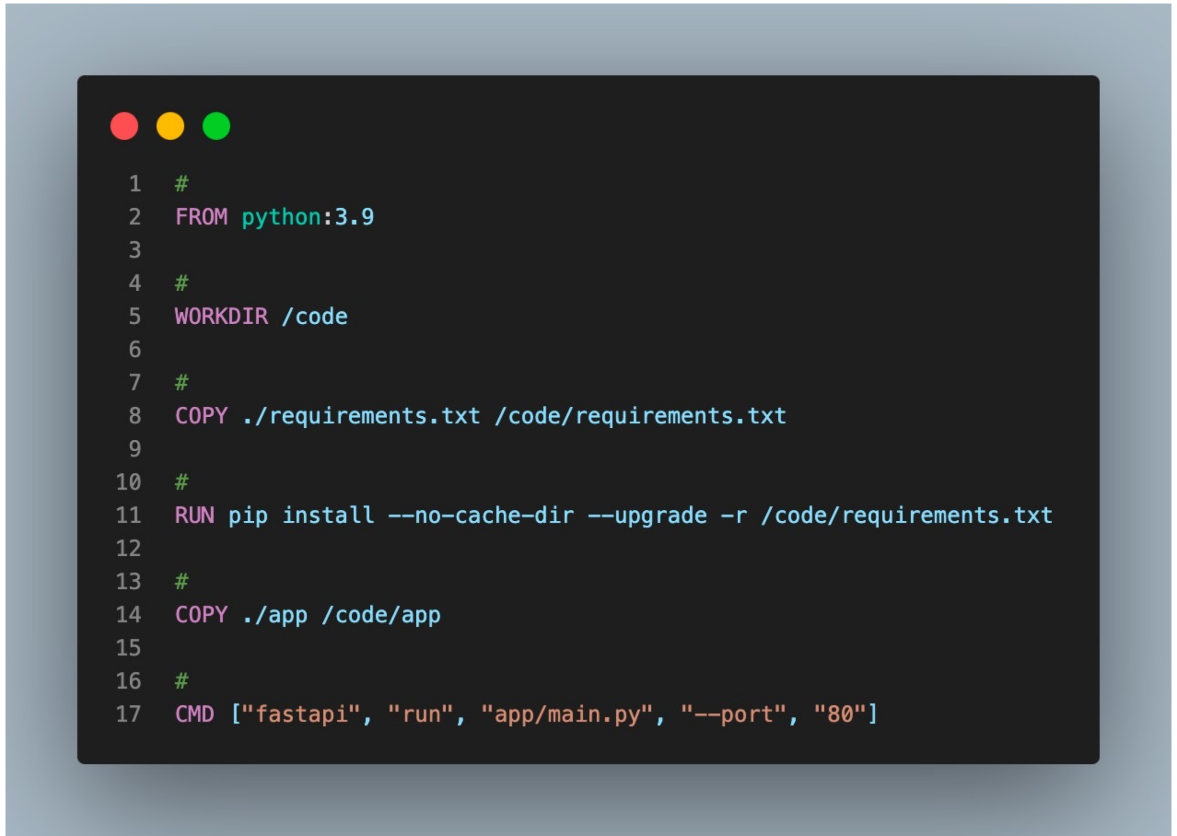


```
#
FROM python:3.9

#
WORKDIR /code

#
COPY ./requirements.txt /code/requirements.txt

#
RUN pip install --no-cache-dir --upgrade -r /code/requirements.txt

#
COPY ./app /code/app

#
CMD ["fastapi", "run", "app/main.py", "--port", "80"]
```

Start your first Docker

```
FROM python:3.9
```

This line specifies the base image to use for the Docker container.

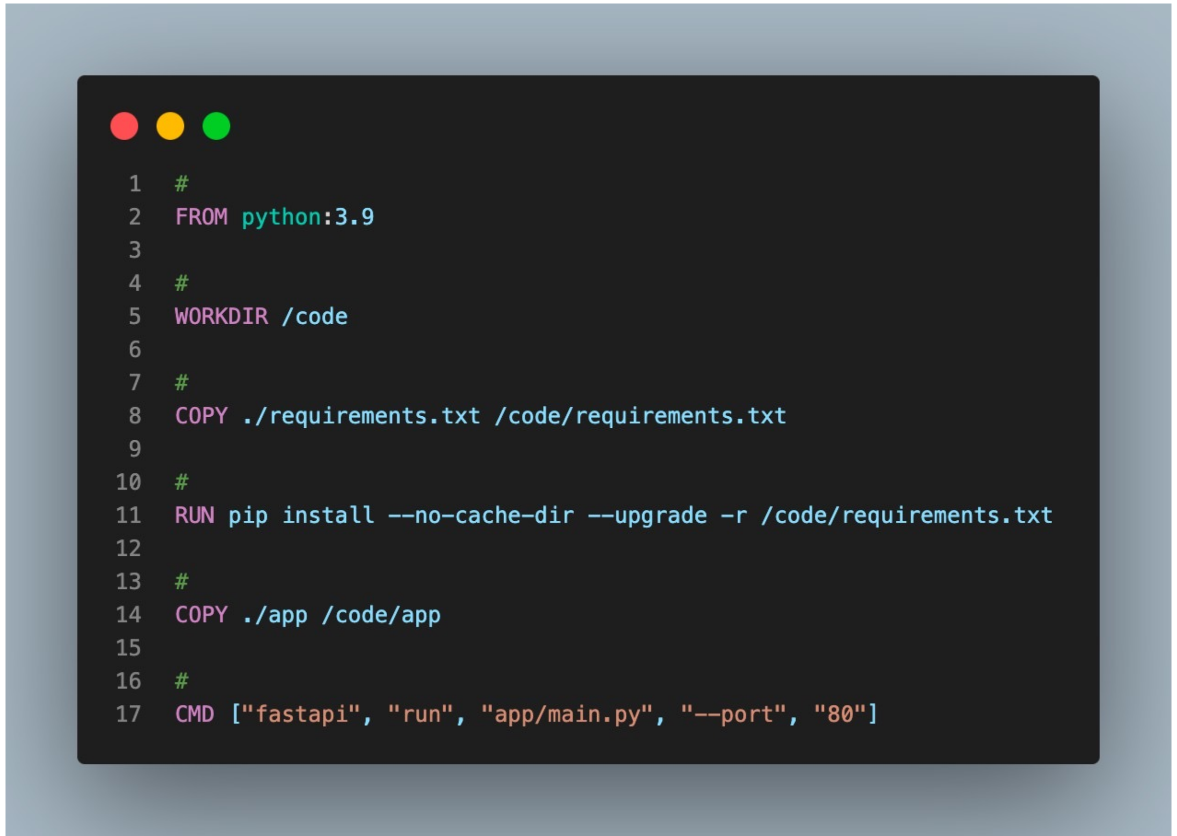


```
#
FROM python:3.9

#
WORKDIR /code

#
COPY ./requirements.txt /code/requirements.txt

#
RUN pip install --no-cache-dir --upgrade -r /code/requirements.txt

#
COPY ./app /code/app

#
CMD ["fastapi", "run", "app/main.py", "--port", "80"]
```

Start your first Docker

`FROM python:3.9`

It pulls the official Python 3.9 image from the Docker Hub.

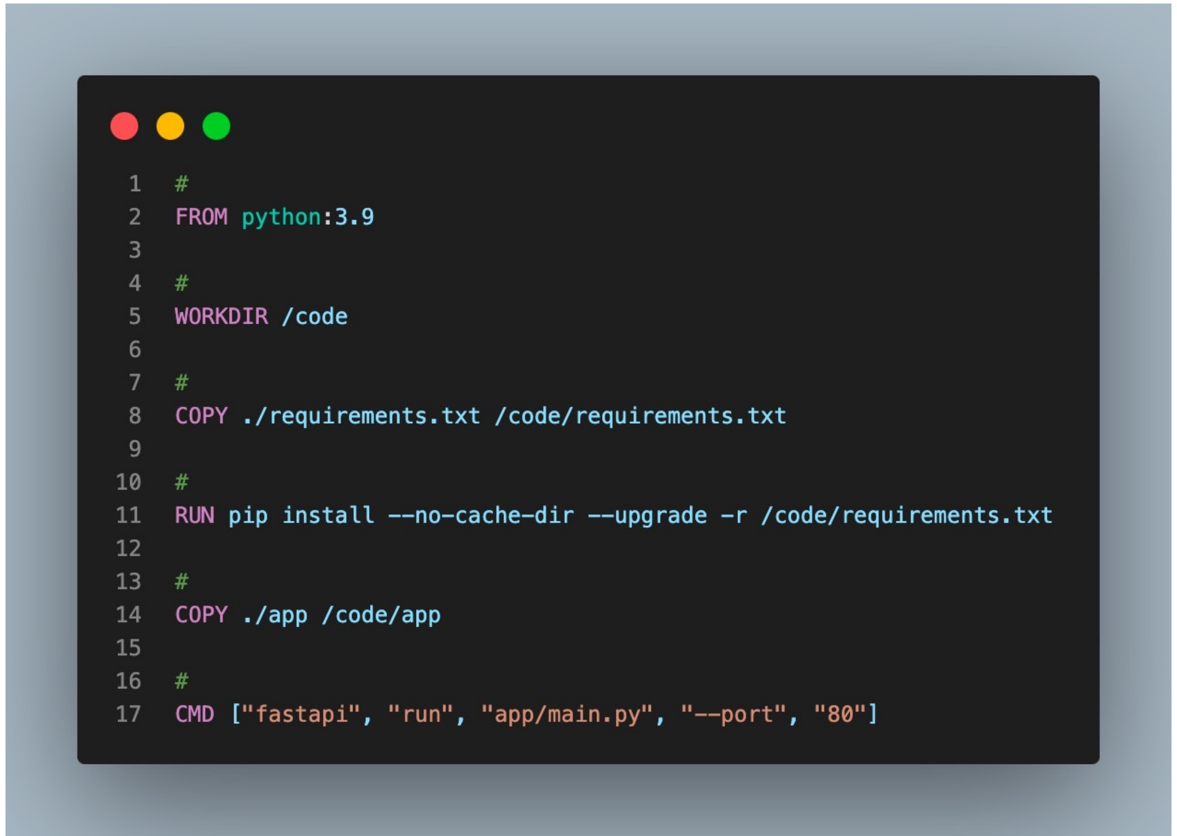


```
#
FROM python:3.9

#
WORKDIR /code

#
COPY ./requirements.txt /code/requirements.txt

#
RUN pip install --no-cache-dir --upgrade -r /code/requirements.txt

#
COPY ./app /code/app

#
CMD ["fastapi", "run", "app/main.py", "--port", "80"]
```

Start your first Docker

FROM python:3.9

The base image contains a minimal Python environment to build the application upon.

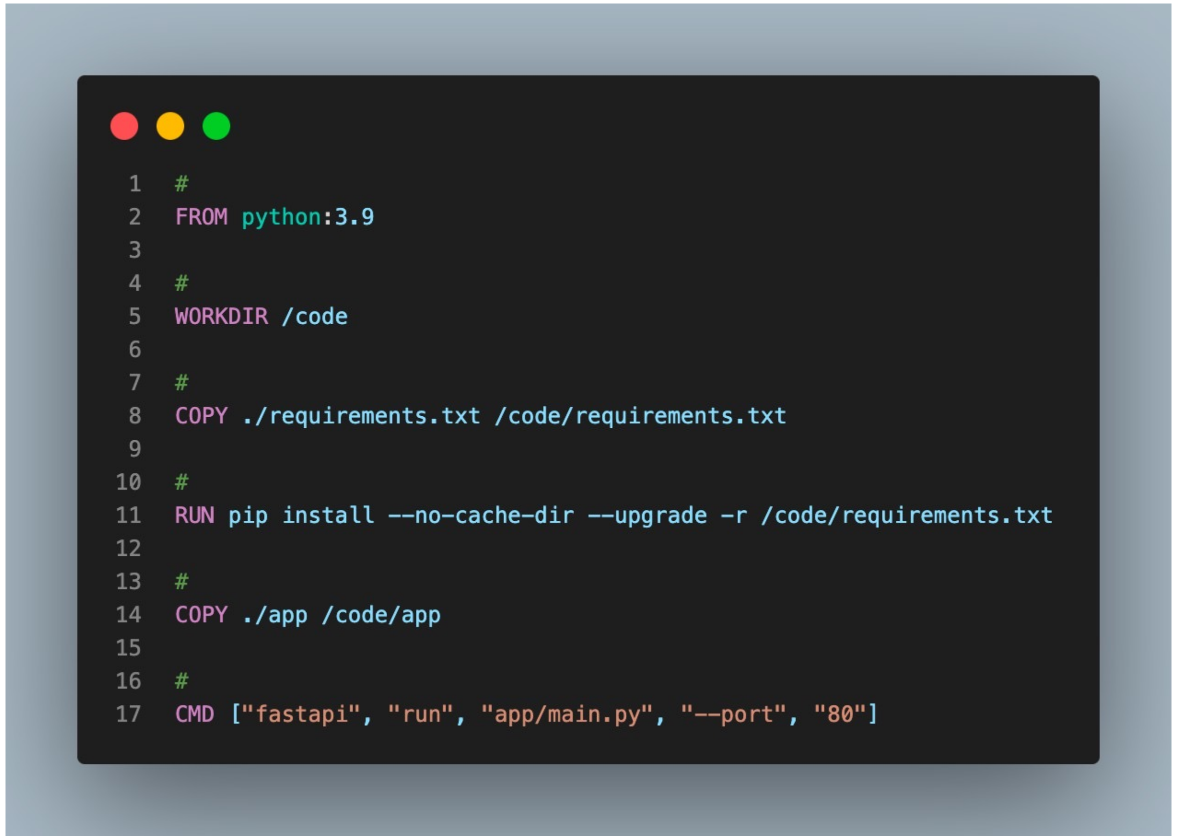


```
#
FROM python:3.9

#
WORKDIR /code

#
COPY ./requirements.txt /code/requirements.txt

#
RUN pip install --no-cache-dir --upgrade -r /code/requirements.txt

#
COPY ./app /code/app

#
CMD ["fastapi", "run", "app/main.py", "--port", "80"]
```

Start your first Docker

WORKDIR /code

This line sets the working directory inside the Docker container.

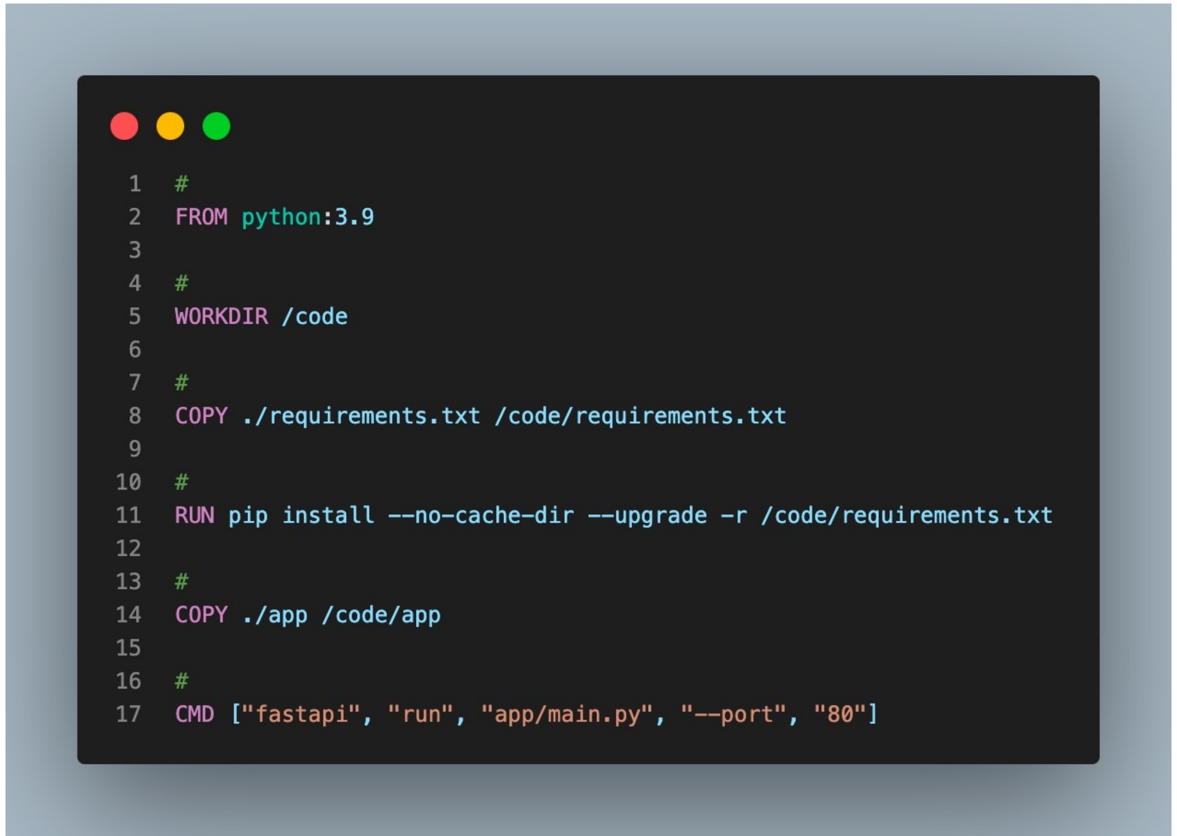


```
#
FROM python:3.9

#
WORKDIR /code

#
COPY ./requirements.txt /code/requirements.txt

#
RUN pip install --no-cache-dir --upgrade -r /code/requirements.txt

#
COPY ./app /code/app

#
CMD ["fastapi", "run", "app/main.py", "--port", "80"]
```

Start your first Docker

```
WORKDIR /code
```

All subsequent instructions in the Dockerfile will be executed in the /code directory.

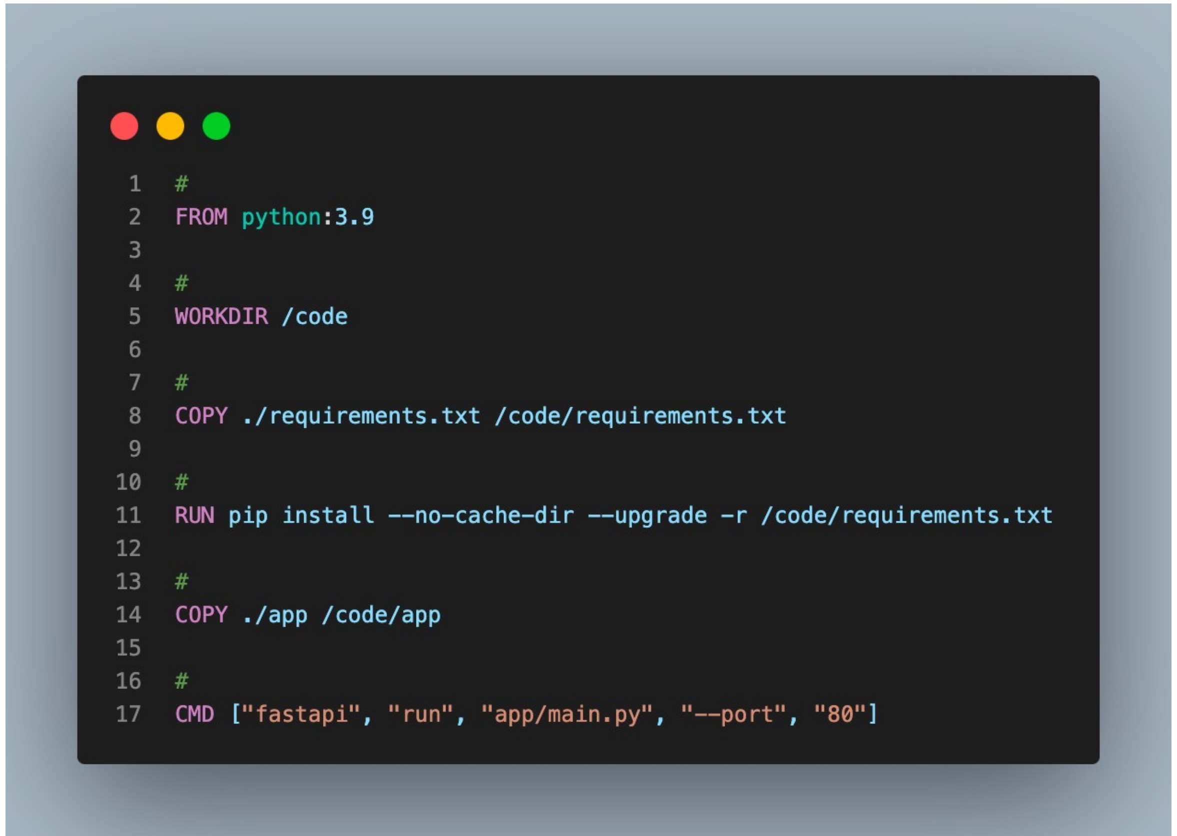


```
#
FROM python:3.9

#
WORKDIR /code

#
COPY ./requirements.txt /code/requirements.txt

#
RUN pip install --no-cache-dir --upgrade -r /code/requirements.txt

#
COPY ./app /code/app

#
CMD ["fastapi", "run", "app/main.py", "--port", "80"]
```

Start your first Docker

`WORKDIR /code`

If the directory does not exist, it will be created.

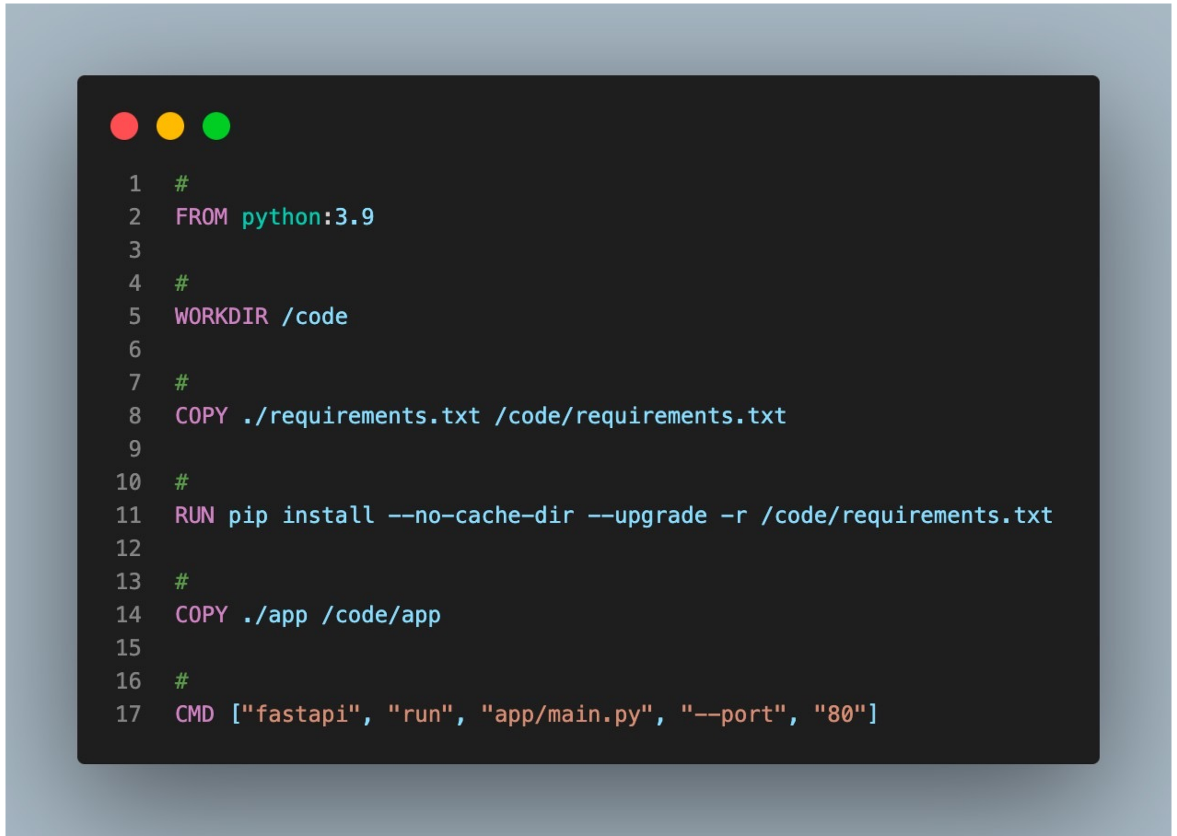


```
#
FROM python:3.9

#
WORKDIR /code

#
COPY ./requirements.txt /code/requirements.txt

#
RUN pip install --no-cache-dir --upgrade -r /code/requirements.txt

#
COPY ./app /code/app

#
CMD ["fastapi", "run", "app/main.py", "--port", "80"]
```

Start your first Docker

./requirements.txt /code/requirements.txt

This line copies the requirements.txt file from your local machine to the /code directory inside the Docker container.

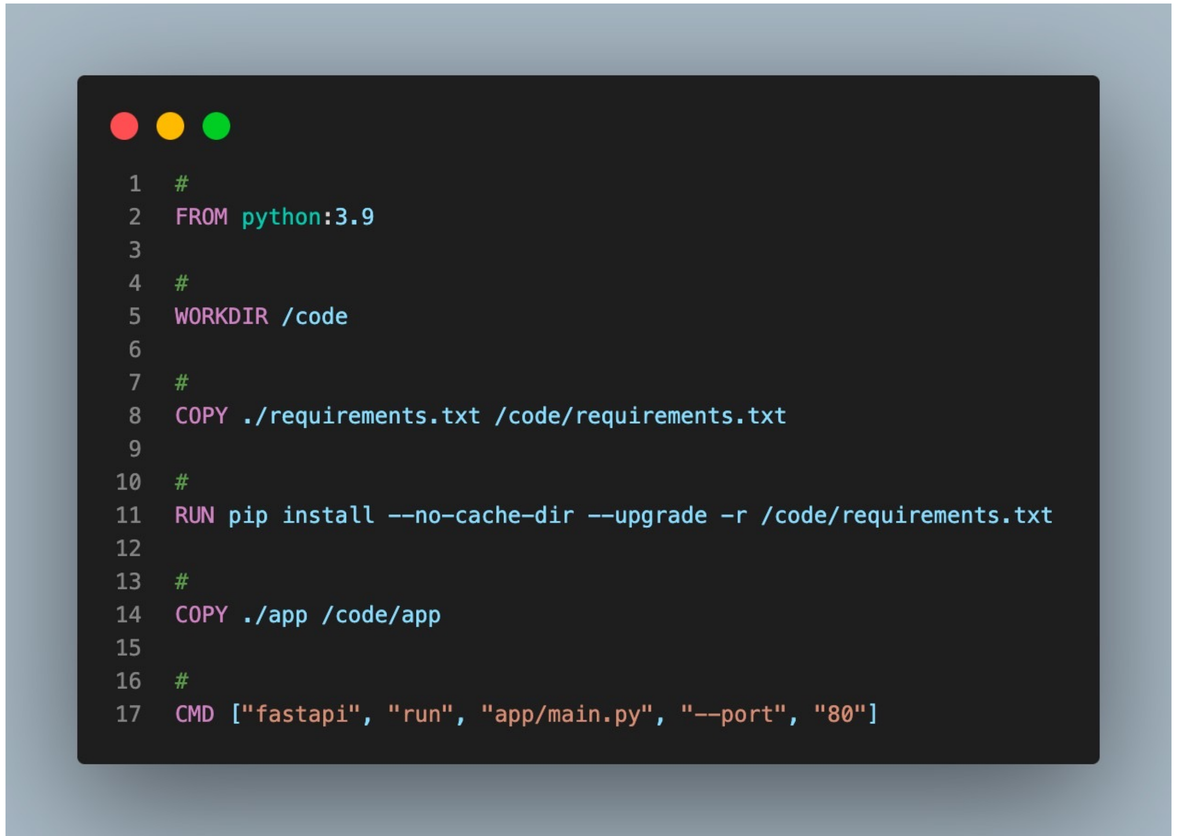


```
#
FROM python:3.9

#
WORKDIR /code

#
COPY ./requirements.txt /code/requirements.txt

#
RUN pip install --no-cache-dir --upgrade -r /code/requirements.txt

#
COPY ./app /code/app

#
CMD ["fastapi", "run", "app/main.py", "--port", "80"]
```

Start your first Docker

```
RUN pip install --no-cache-dir --upgrade -r /code/requirements.txt
```

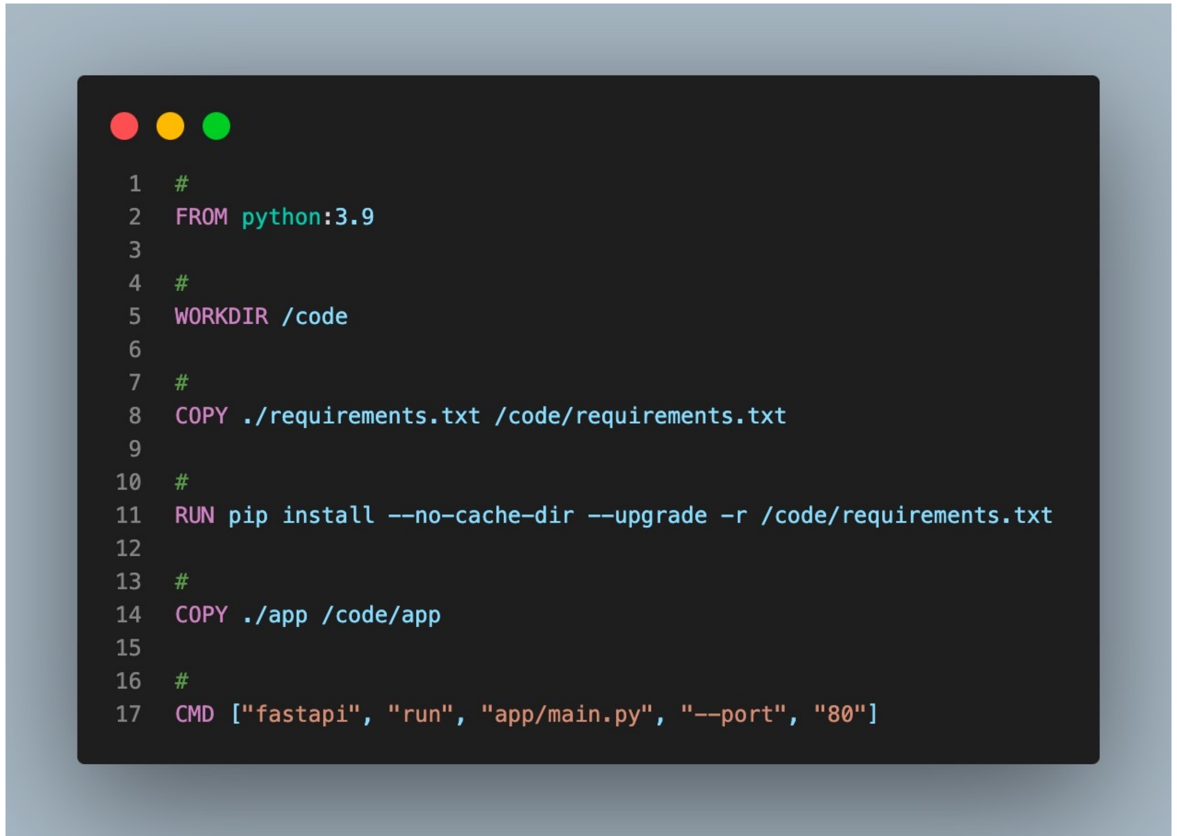
This line installs the Python dependencies listed in the requirements.txt file.



```
#
FROM python:3.9

#
WORKDIR /code

#
COPY ./requirements.txt /code/requirements.txt

#
RUN pip install --no-cache-dir --upgrade -r /code/requirements.txt

#
COPY ./app /code/app

#
CMD ["fastapi", "run", "app/main.py", "--port", "80"]
```

Start your first Docker

RUN pip install --no-cache-dir --upgrade -r /code/requirements.txt
The --no-cache-dir option tells pip not to cache the packages, which reduces the size of the image.

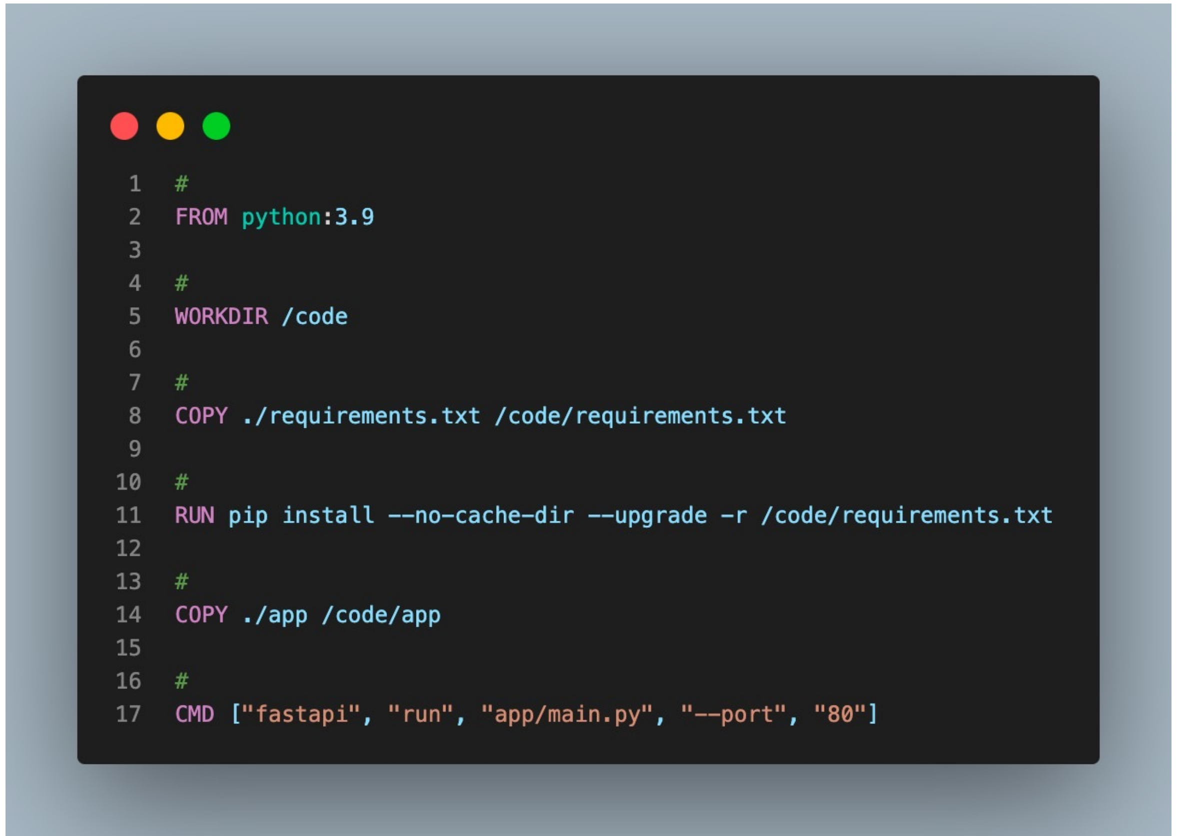


```
#
FROM python:3.9

#
WORKDIR /code

#
COPY ./requirements.txt /code/requirements.txt

#
RUN pip install --no-cache-dir --upgrade -r /code/requirements.txt

#
COPY ./app /code/app

#
CMD ["fastapi", "run", "app/main.py", "--port", "80"]
```

Start your first Docker

```
RUN pip install --no-cache-dir --upgrade -r /code/requirements.txt
```

The --upgrade option makes sure that the latest versions of the packages are installed.

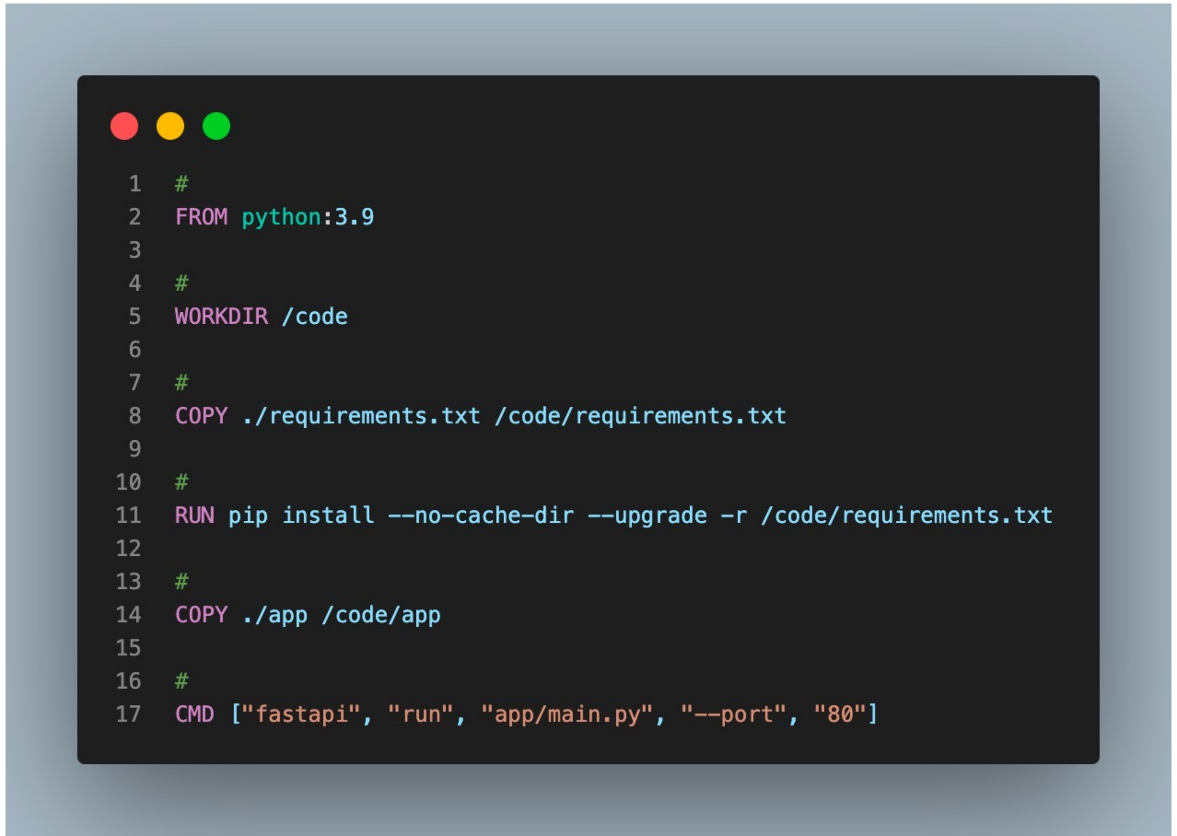


```
#
FROM python:3.9

#
WORKDIR /code

#
COPY ./requirements.txt /code/requirements.txt

#
RUN pip install --no-cache-dir --upgrade -r /code/requirements.txt

#
COPY ./app /code/app

#
CMD ["fastapi", "run", "app/main.py", "--port", "80"]
```

Start your first Docker

`COPY ./app /code/app`

This line copies the entire app directory from your local machine to the /code/app directory inside the Docker container.

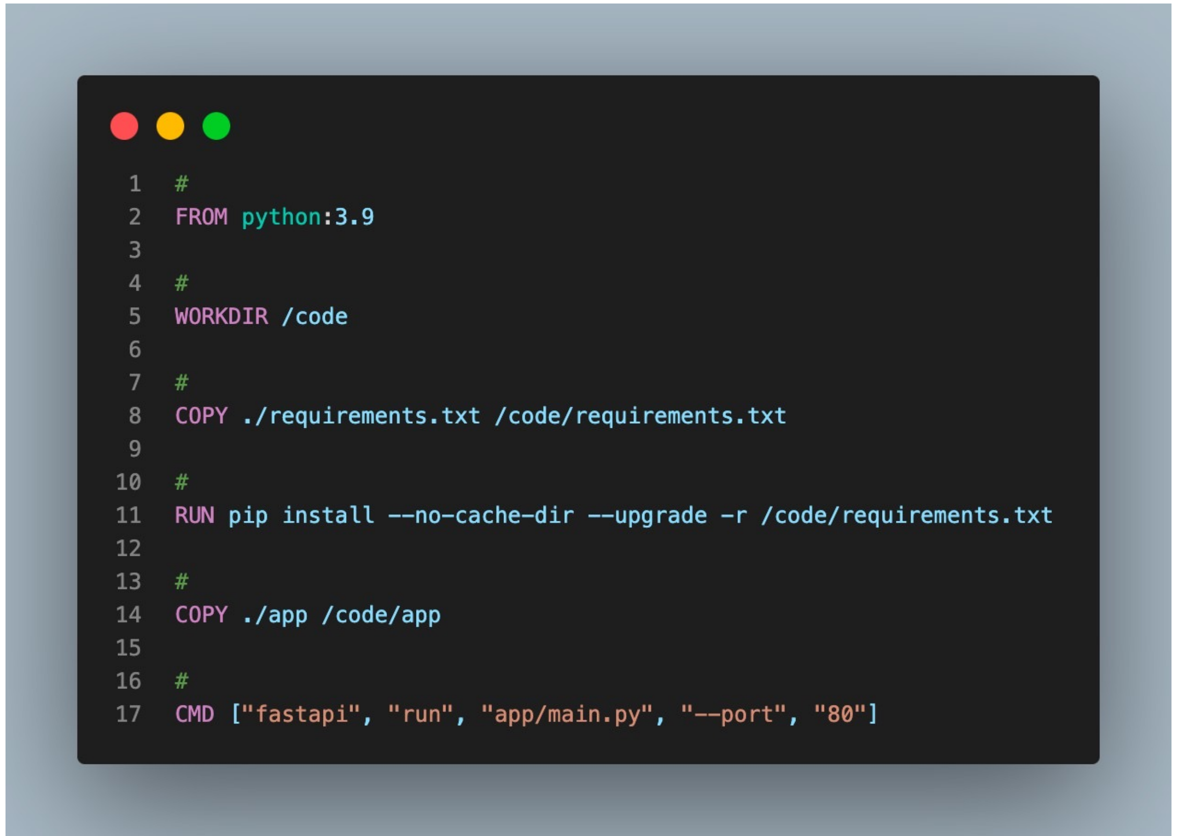


```
#
FROM python:3.9

#
WORKDIR /code

#
COPY ./requirements.txt /code/requirements.txt

#
RUN pip install --no-cache-dir --upgrade -r /code/requirements.txt

#
COPY ./app /code/app

#
CMD ["fastapi", "run", "app/main.py", "--port", "80"]
```

Start your first Docker

```
CMD ["fastapi", "run", "app/main.py", "--port", "80"]
```

This line specifies the command to run when the container starts.

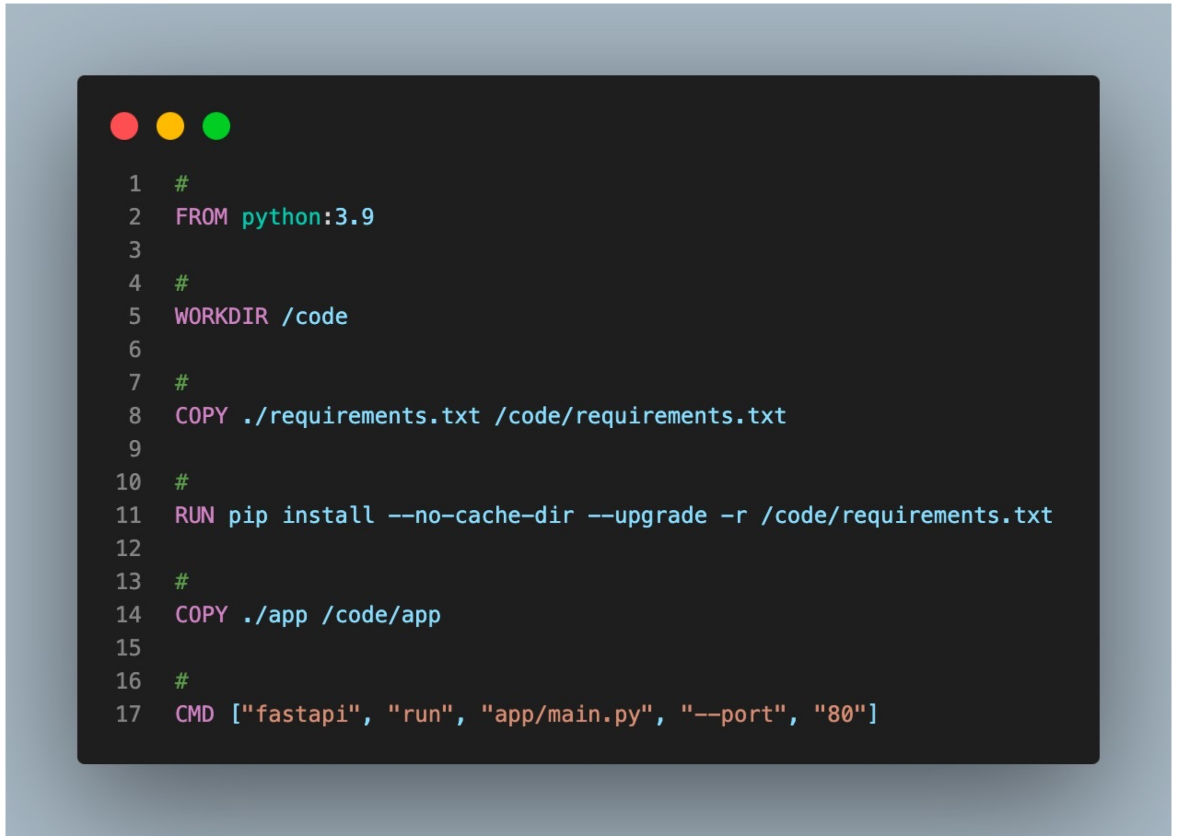


```
#
FROM python:3.9

#
WORKDIR /code

#
COPY ./requirements.txt /code/requirements.txt

#
RUN pip install --no-cache-dir --upgrade -r /code/requirements.txt

#
COPY ./app /code/app

#
CMD ["fastapi", "run", "app/main.py", "--port", "80"]
```

Start your first Docker

```
CMD ["fastapi", "run", "app/main.py", "--port", "80"]
```

‘fastapi’: It uses fastapi to start the application.

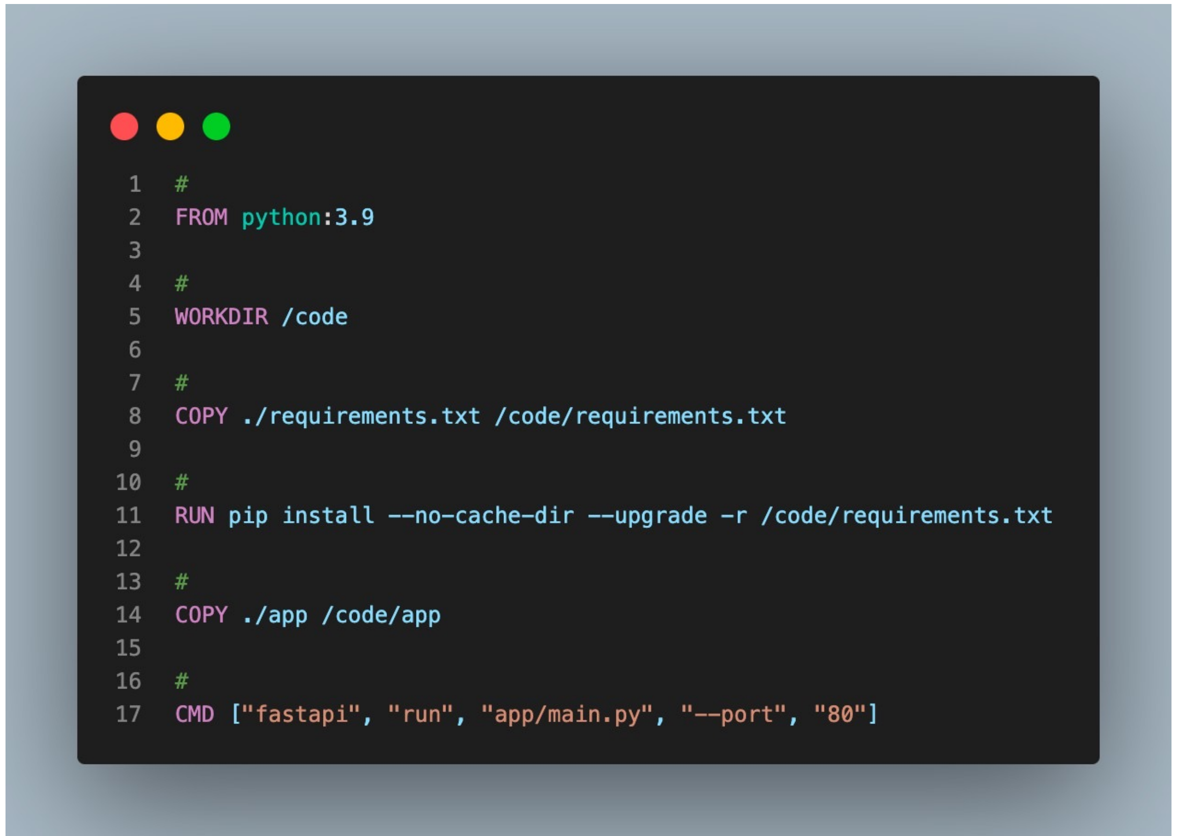


```
#
FROM python:3.9

#
WORKDIR /code

#
COPY ./requirements.txt /code/requirements.txt

#
RUN pip install --no-cache-dir --upgrade -r /code/requirements.txt

#
COPY ./app /code/app

#
CMD ["fastapi", "run", "app/main.py", "--port", "80"]
```

Start your first Docker

```
CMD ["fastapi", "run", "app/main.py", "--port", "80"]
```

'run': The run command specifies the script to execute.

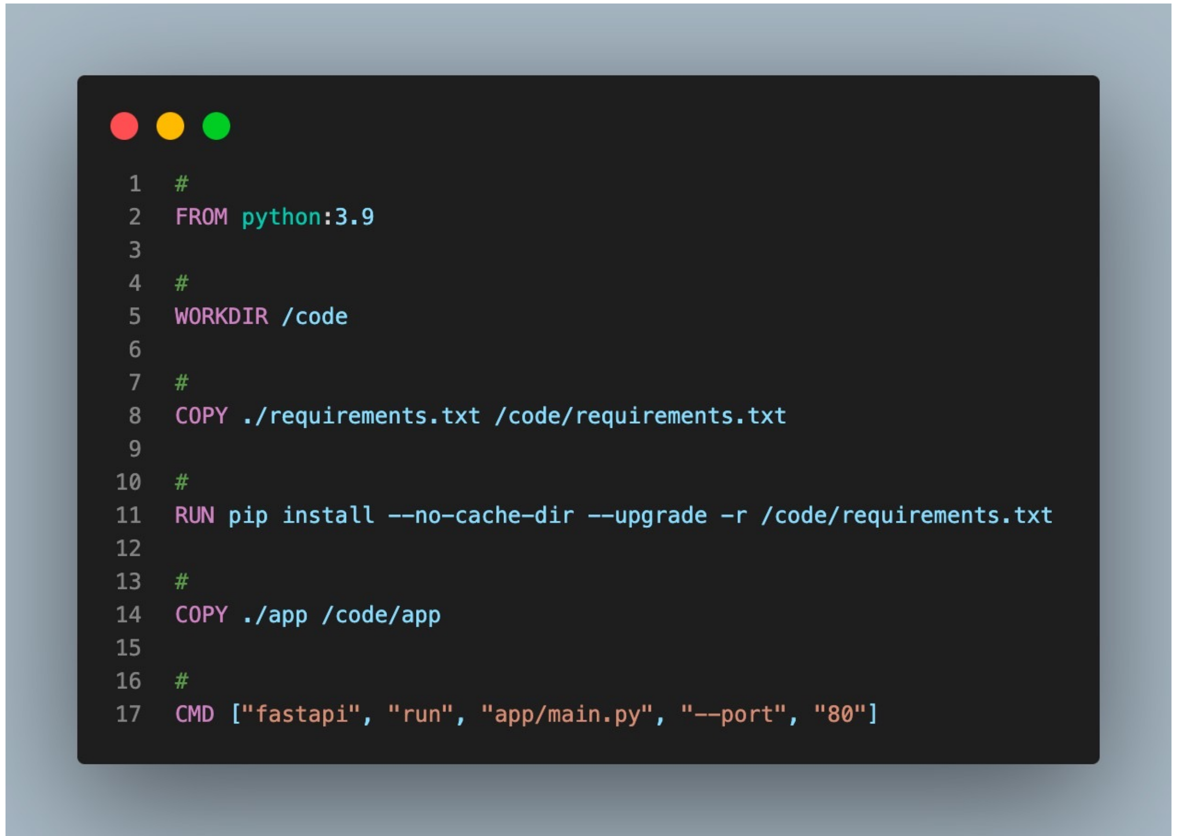


```
#
FROM python:3.9

#
WORKDIR /code

#
COPY ./requirements.txt /code/requirements.txt

#
RUN pip install --no-cache-dir --upgrade -r /code/requirements.txt

#
COPY ./app /code/app

#
CMD ["fastapi", "run", "app/main.py", "--port", "80"]
```

Start your first Docker

```
CMD ["fastapi", "run", "app/main.py", "--port", "80"]
```

‘app/main.py’: Points to the main Python script of the FastAPI application.

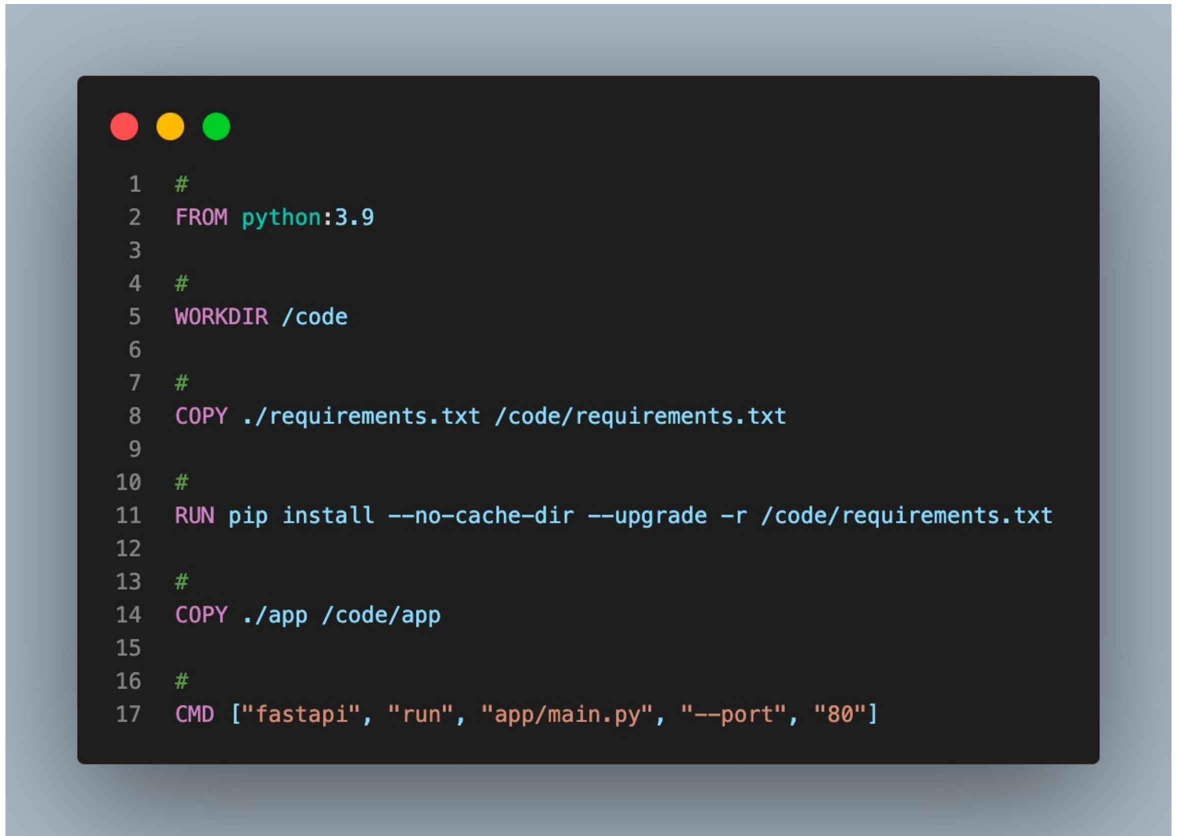


```
#
FROM python:3.9

#
WORKDIR /code

#
COPY ./requirements.txt /code/requirements.txt

#
RUN pip install --no-cache-dir --upgrade -r /code/requirements.txt

#
COPY ./app /code/app

#
CMD ["fastapi", "run", "app/main.py", "--port", "80"]
```

Start your first Docker

```
CMD ["fastapi", "run", "app/main.py", "--port", "80"]
```

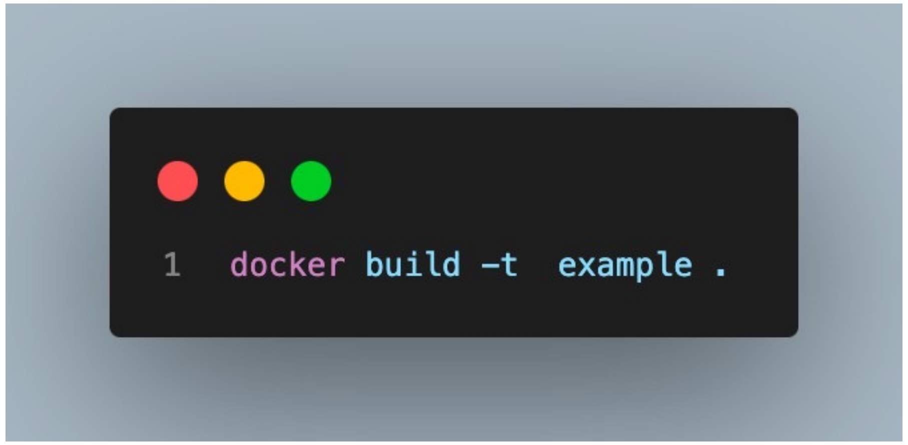
'--port', '80': Options sets the server to listen on port 80.



```
#
FROM python:3.9

#
WORKDIR /code

#
COPY ./requirements.txt /code/requirements.txt

#
RUN pip install --no-cache-dir --upgrade -r /code/requirements.txt

#
COPY ./app /code/app

#
CMD ["fastapi", "run", "app/main.py", "--port", "80"]
```

Start your first Docker

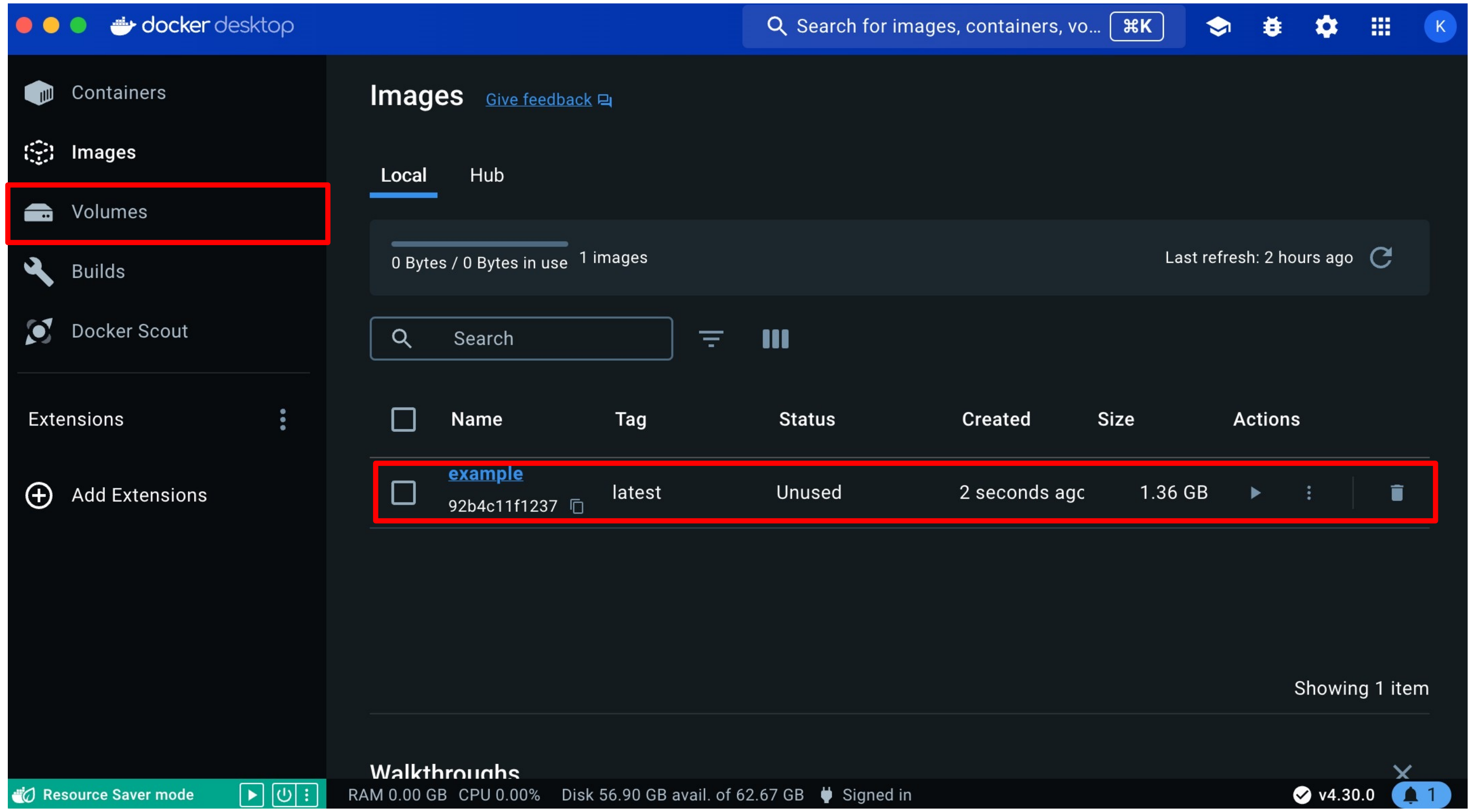
To build the Docker you will need to use the command ‘Docker Build Command’. The command creates a Docker image named "example” just like the repo and the directory name using the Dockerfile and context located in the current directory.



```
docker build -t  example .
```

Start your first Docker

Now if you open the Docker application you will find the image in the images section:



Images

0 Bytes / 0 Bytes in use 1 images Last refresh: 2 hours ago

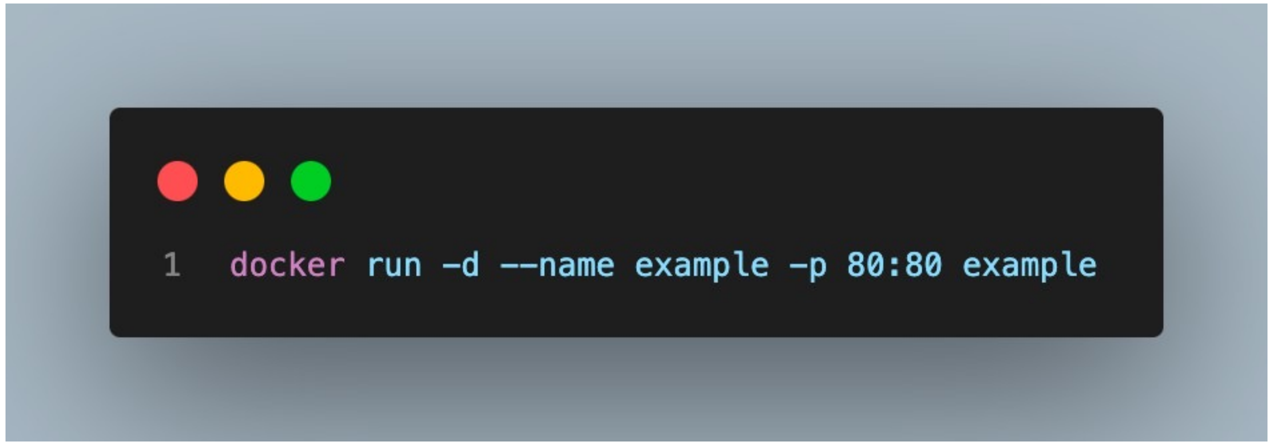
Name	Tag	Status	Created	Size	Actions
example 92b4c11f1237	latest	Unused	2 seconds agc	1.36 GB	

Showing 1 item

RAM 0.00 GB CPU 0.00% Disk 56.90 GB avail. of 62.67 GB Signed in v4.30.0 1

Start your first Docker

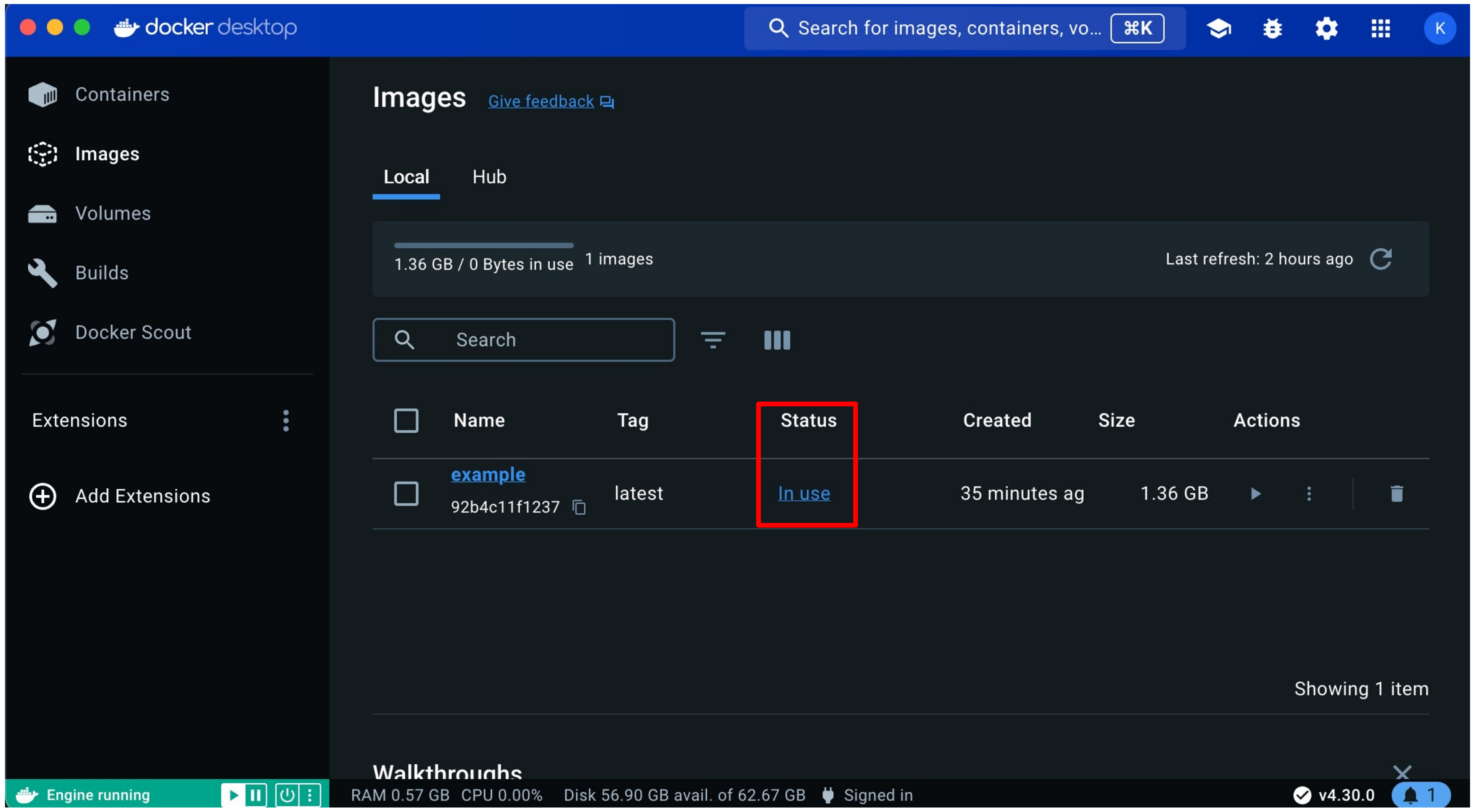
The command ‘docker run -d --name example -p 80:80 example’ starts a new container named "example" from the "example" image in detached mode, mapping port 80 on the host to port 80 on the container.



```
docker run -d --name example -p 80:80 example
```

Start your first Docker

As you can see the status now changed to in use.


docker desktop
Search for images, containers, vo... ⌘K
Containers
Images
Volumes
Builds
Docker Scout
Extensions
Add Extensions
Engine running
RAM 0.57 GB CPU 0.00% Disk 56.90 GB avail. of 62.67 GB Signed in
v4.30.0

Images Give feedback

Local Hub

1.36 GB / 0 Bytes in use 1 images Last refresh: 2 hours ago

Search

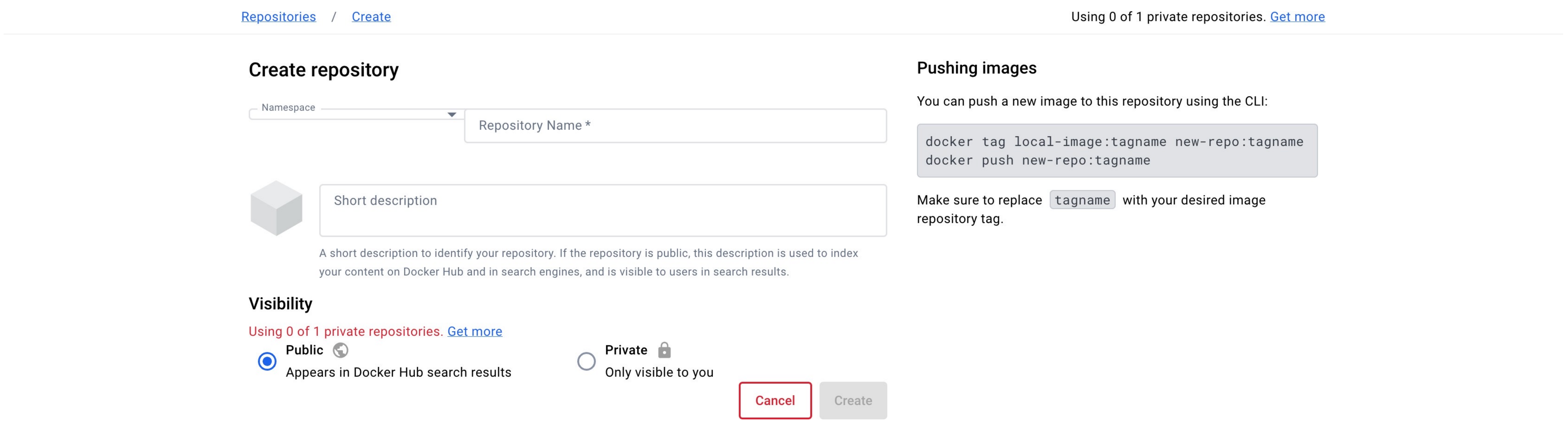
Name	Tag	Status	Created	Size	Actions
example 92b4c11f1237	latest	In use	35 minutes ag	1.36 GB	

Showing 1 item

Walkthroughs

Push the image to Ducker Hub

To push the image to the Docker Hub first create a new repo on the Ducker website.



Repositories / Create

Using 0 of 1 private repositories. Get more

Create repository

A short description to identify your repository. If the repository is public, this description is used to index your content on Docker Hub and in search engines, and is visible to users in search results.

Visibility

Using 0 of 1 private repositories. Get more

[✓ Public] Appears in Docker Hub search results

Private — Only visible to you

Pushing images

You can push a new image to this repository using the CLI:

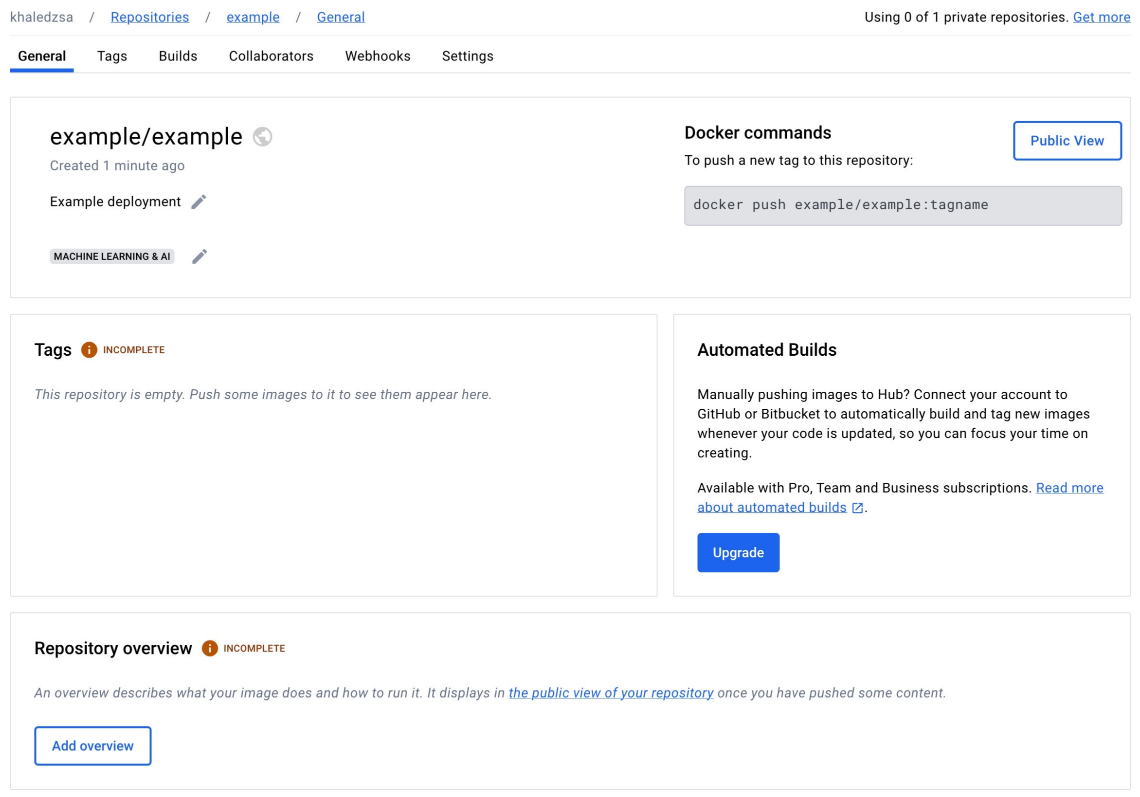
```
docker tag local-image:tagname new-repo:tagname
docker push new-repo:tagname
```

Make sure to replace `tagname` with your desired image repository tag.

https://hub.docker.com/repository/create

Push the image to Ducker Hub

Complete all the needed information.



khaledzsa / Repositories / example / General

Using 0 of 1 private repositories. Get more

General Tags Builds Collaborators Webhooks Settings

example/example

Created 1 minute ago

Example deployment

MACHINE LEARNING & AI

Docker commands

To push a new tag to this repository:

Public View

```
docker push example/example:tagname
```

Tags INCOMPLETE

This repository is empty. Push some images to it to see them appear here.

Automated Builds

Manually pushing images to Hub? Connect your account to GitHub or Bitbucket to automatically build and tag new images whenever your code is updated, so you can focus your time on creating.

Available with Pro, Team and Business subscriptions. Read more about automated builds.

Upgrade

Repository overview INCOMPLETE

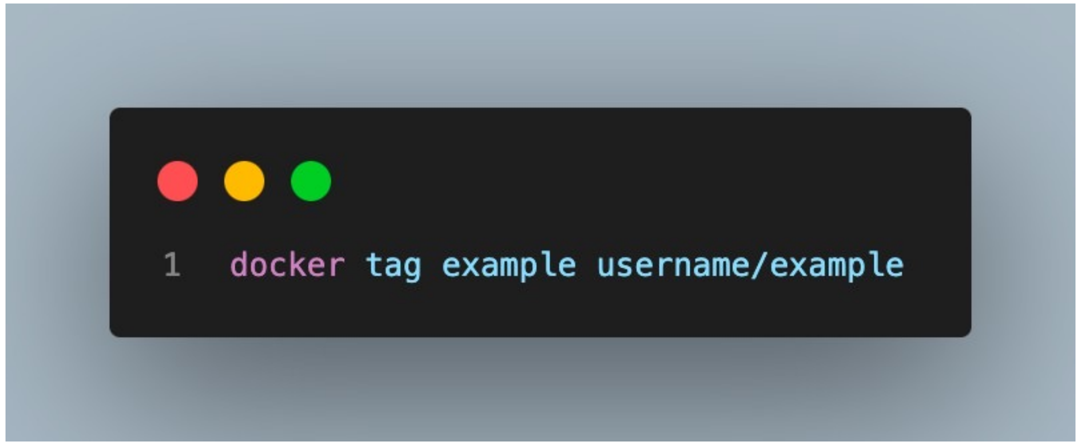
An overview describes what your image does and how to run it. It displays in the public view of your repository once you have pushed some content.

Add overview

https://hub.docker.com/repository/create

Push the image to Ducker Hub

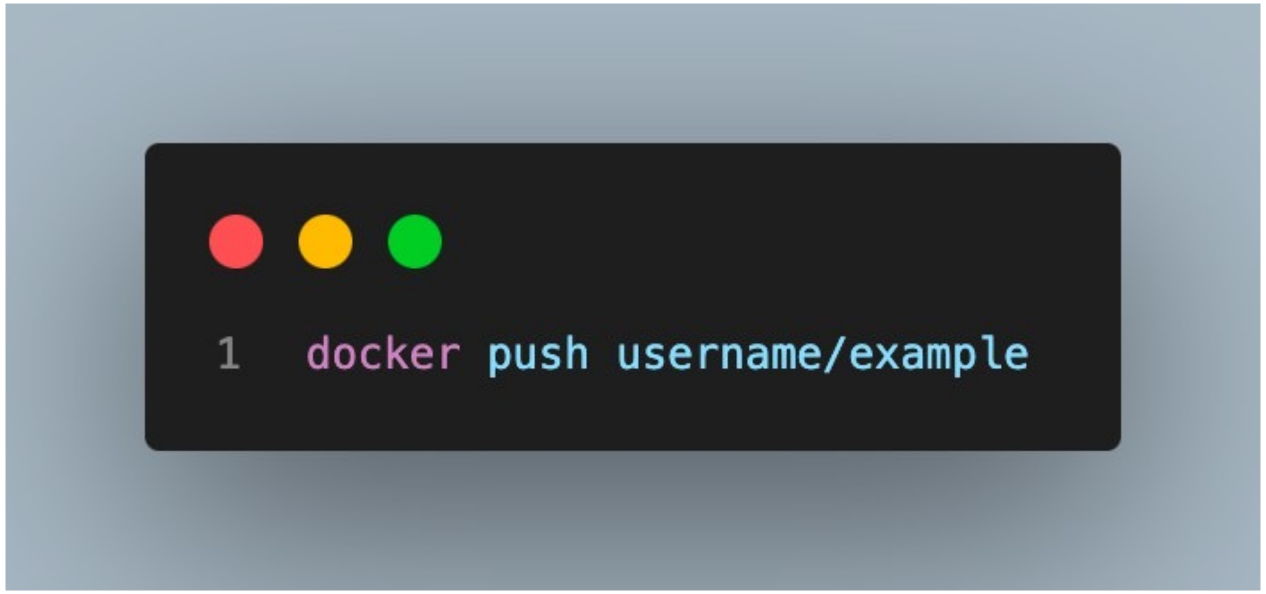
Rin this command with the local image name and the repo name on the Docker Hub.



```
docker tag example username/example
```

Push the image to Ducker Hub

Now if you can see the new image at the image section all you have to do is to push the image to the Docker Hub repo via this command:



```
docker push username/example
```

Docker Hub

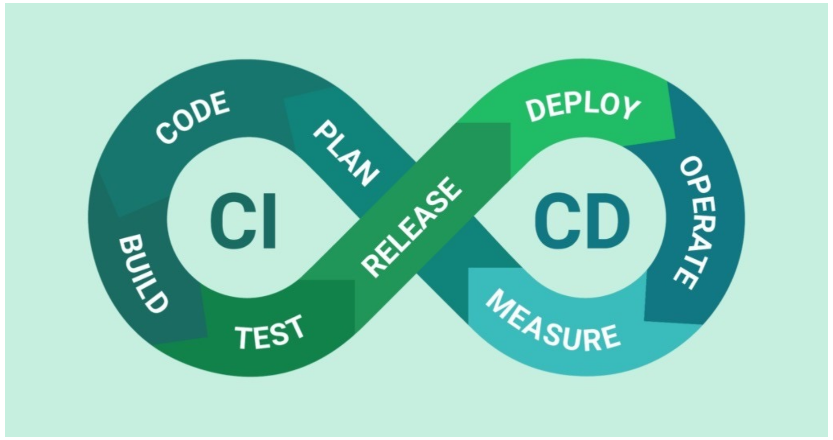
```
Your Docker repo is ready to be deployed!
For further steps we recommend you to look for the desired
cloud platform’s documentation and follow the steps.

The floder used is uploaded named ‘example’.
```

Setting up CI/CD Pipeline

Introduction to CI/CD

- **Continuous Integration (CI)** and **Continuous Deployment (CD)** are essential practices in MLOps. It helps in automating the deployment of machine learning models.

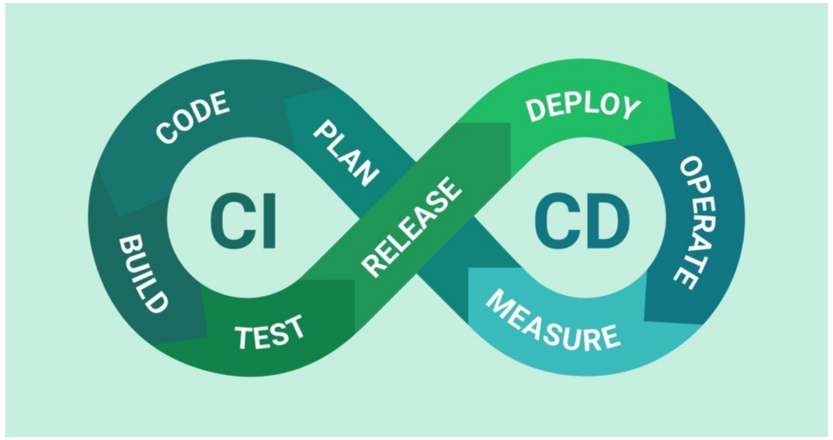
- **CI/CD** pipelines ensure that models are continuously tested, integrated, and deployed, reducing the time from development to production.



Source

Benefits of CI/CD in MLOps

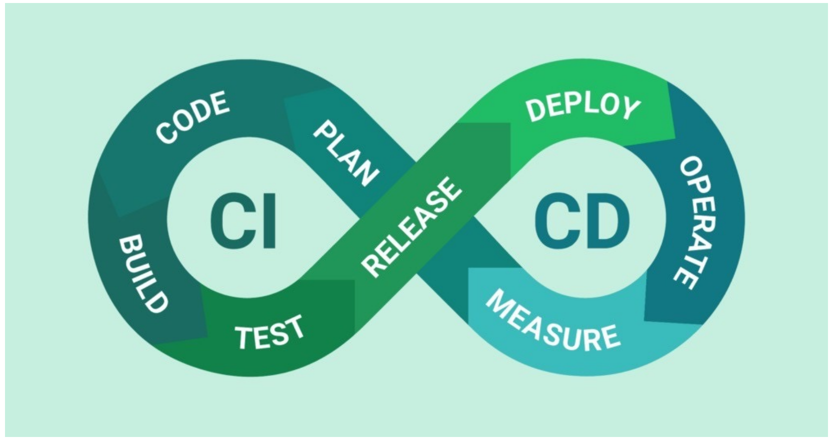
- **Automation:** Automates the repetitive tasks of testing and deploying models.
- **Consistency:** Ensures consistent and reliable deployment processes.
- **Efficiency:** Speeds up the development and deployment cycle.
- **Scalability:** Facilitates scaling of deployment processes as the number of models and updates increase.



Source

CI/CD Components

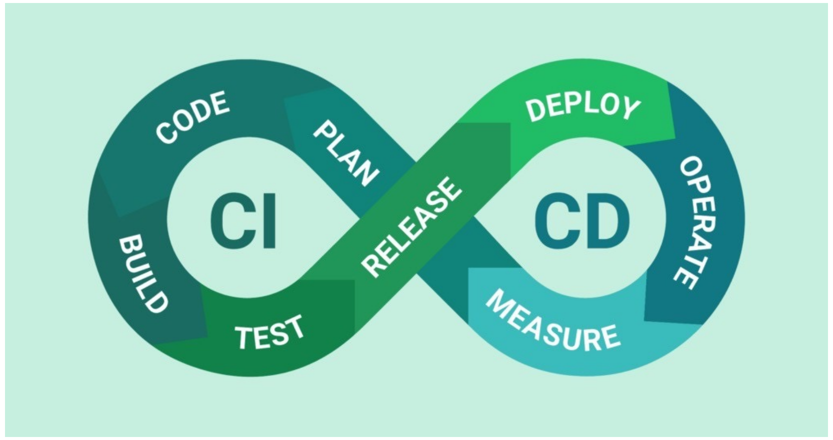
- **Source Control:** Manages code and model versions (e.g., GitHub, GitLab).
- **Continuous Integration:** Automates testing and integration of code changes.
- **Continuous Deployment:** Automates the deployment of tested changes to production.
- **Monitoring:** Tracks the performance and health of deployed models.



Source

CI/CD Pipeline for MLOps

- **Code Commit:** Developers commit code to a version control system (GitHub for example).
- **Build:** The pipeline builds the project and packages the model.
- **Test:** Automated tests are run to validate the model.
- **Deploy:** The validated model is deployed to a staging or production environment.
- **Monitor:** The deployed model is monitored for performance and issues.

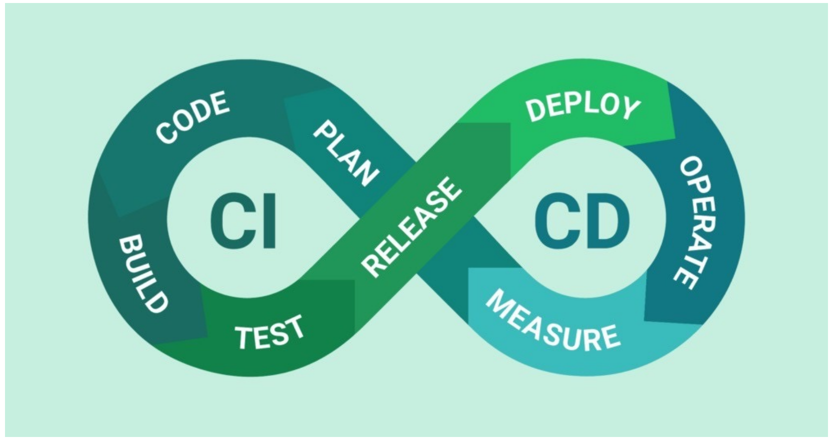


Source

CI/CD Tools for MLOps

There are many CI/CD tools, here are the most famous tools in MLOps:

- **Jenkins:** Open-source automation server for building and deploying applications.
- **GitHub Actions:** CI/CD workflows integrated with GitHub repositories.
- **GitLab CI:** CI/CD pipelines integrated with GitLab.



Source

Example CI/CD Pipeline Using GitHub Actions

In this example, we will walk through a CI/CD pipeline for a FastAPI project using GitHub Actions.

The pipeline includes steps for checking out the code, setting up Python, installing dependencies, running tests, and building a Docker image.

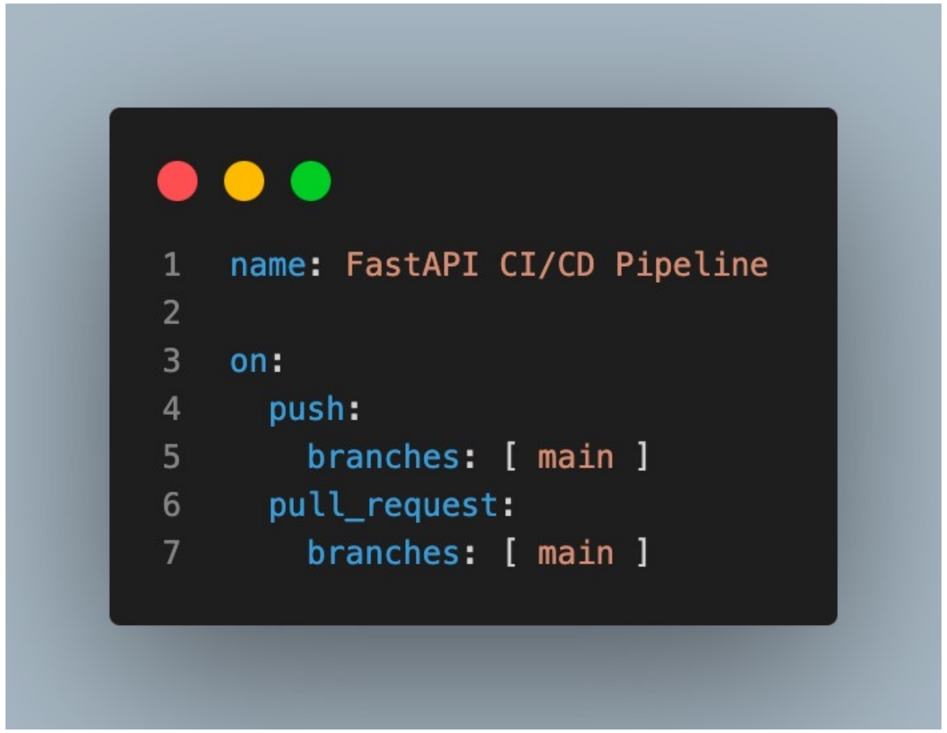
GitHub Actions

Source

Pipeline Trigger

The pipeline is triggered on two events:

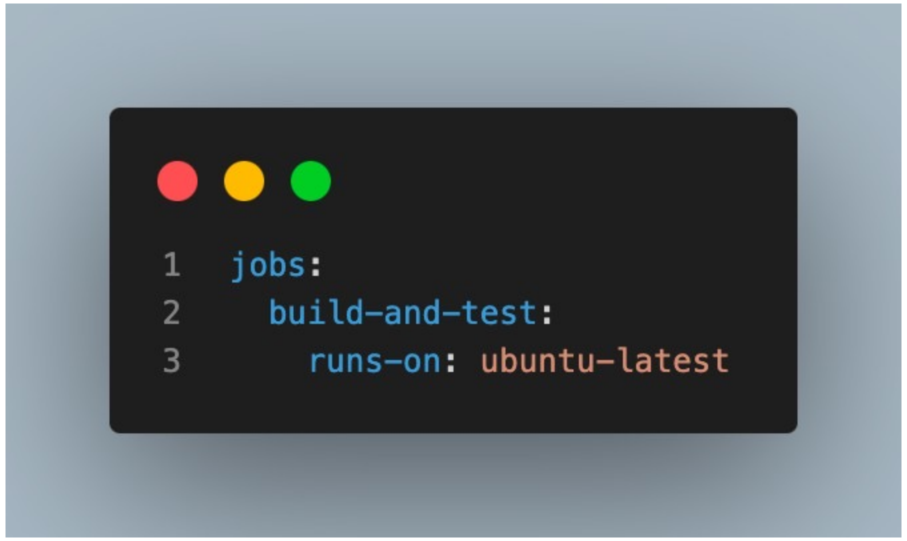
1. **push:** When changes are pushed to the main branch.
2. **pull_request:** When a pull request is created or updated targeting the main branch.



```yaml
name: FastAPI CI/CD Pipeline

on:
  push:
    branches: [ main ]
  pull_request:
    branches: [ main ]
```

Define Jobs

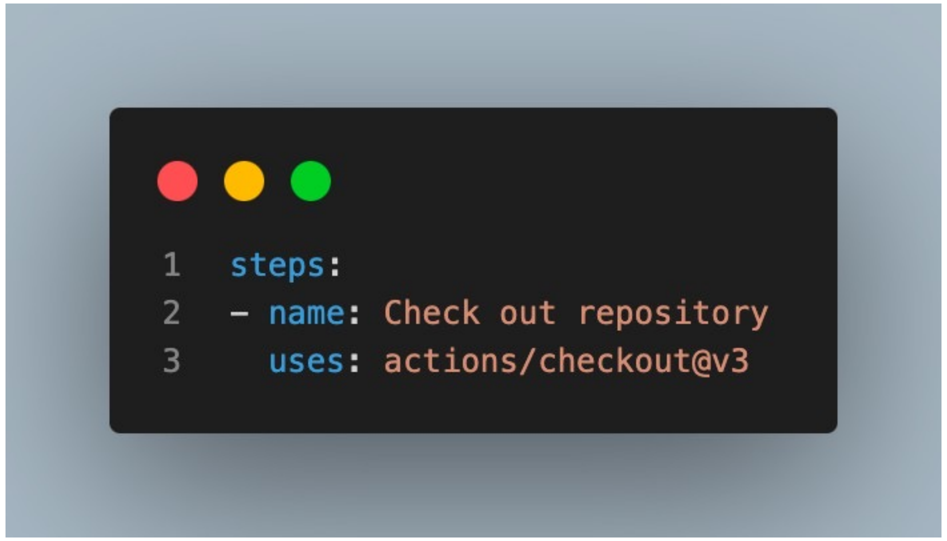
- **jobs:** Defines the jobs to be run in the workflow.
- **build-and-test:** Name of the job.
- **runs-on:** Specifies the virtual environment to use. In this case, ubuntu-latest.



```yaml
jobs:
  build-and-test:
    runs-on: ubuntu-latest
```

Define Jobs

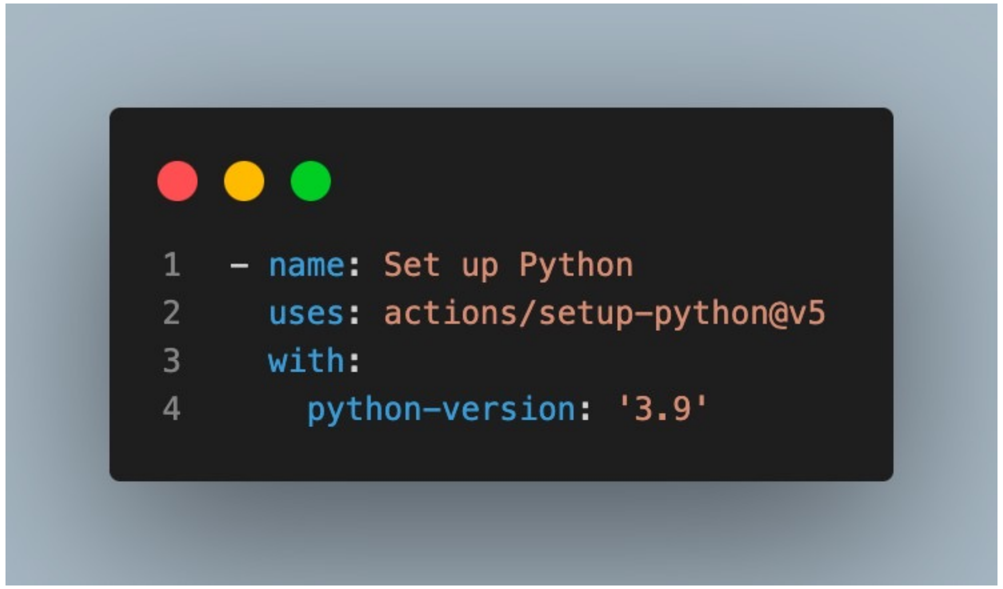
- **steps:** Defines the sequence of steps to be executed.
- **name:** Check out repository: Descriptive name for the step.
- **uses:** actions/checkout@v3: Uses the actions/checkout action to clone the repository to the runner.



```
steps:
- name: Check out repository
  uses: actions/checkout@v3
```

Set Up Python

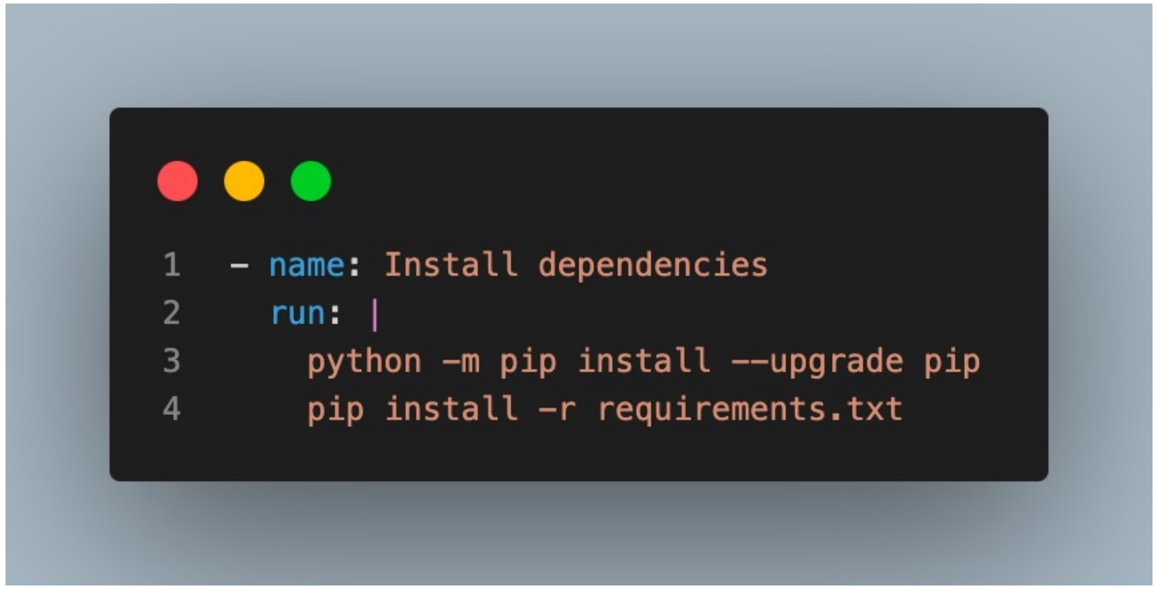
- **name:** Set up Python (step name).
- **uses:** actions/setup-python@v5: Uses the actions/setup-python action to set up a Python environment.
- **with:** Specifies the version of Python to be used, in this case, 3.9.



```
- name: Set up Python
  uses: actions/setup-python@v5
  with:
    python-version: '3.9'
```

Install Dependencies

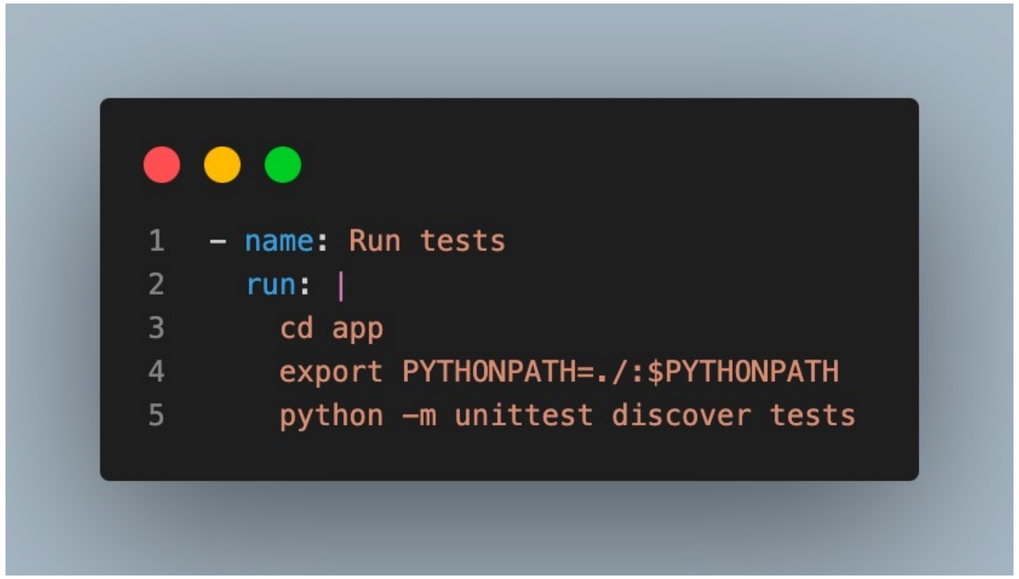
- **name:** Install dependencies (step name).
- **run:** Runs shell commands to:
  - Upgrade pip.
  - Install the project dependencies listed in requirements.txt.



```yaml
- name: Install dependencies
  run: |
    python -m pip install --upgrade pip
    pip install -r requirements.txt
```

Run Tests

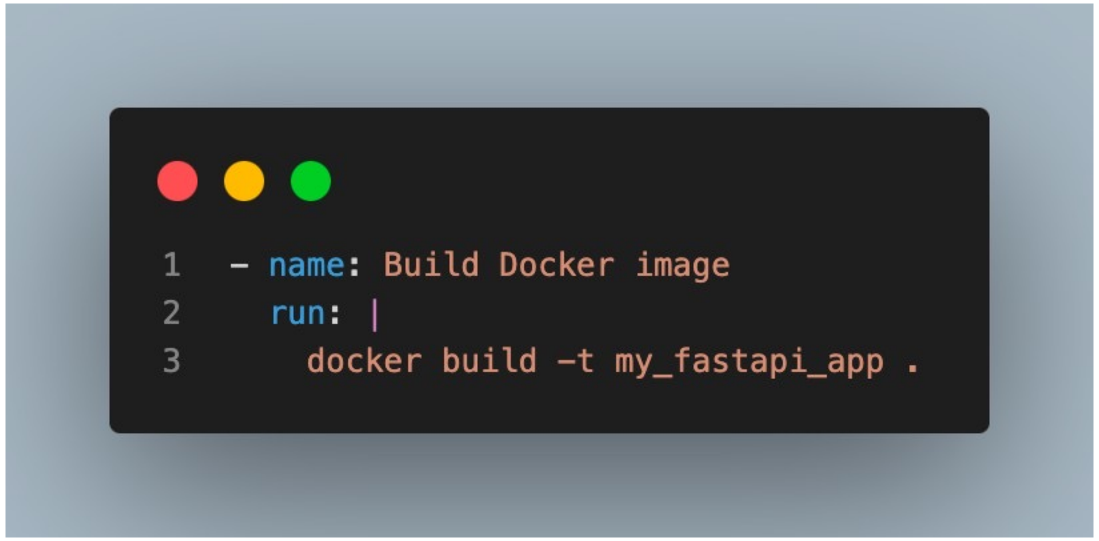
- **name:** Run tests (step name).
- **run:** Runs shell commands to:
    - Change directory to app.
    - Set the PYTHONPATH environment variable to include the current directory.
    - Use unittest to discover and run tests in the test's directory.



```yaml
- name: Run tests
  run: |
    cd app
    export PYTHONPATH=./:$PYTHONPATH
    python -m unittest discover tests
```

Build Docker Image

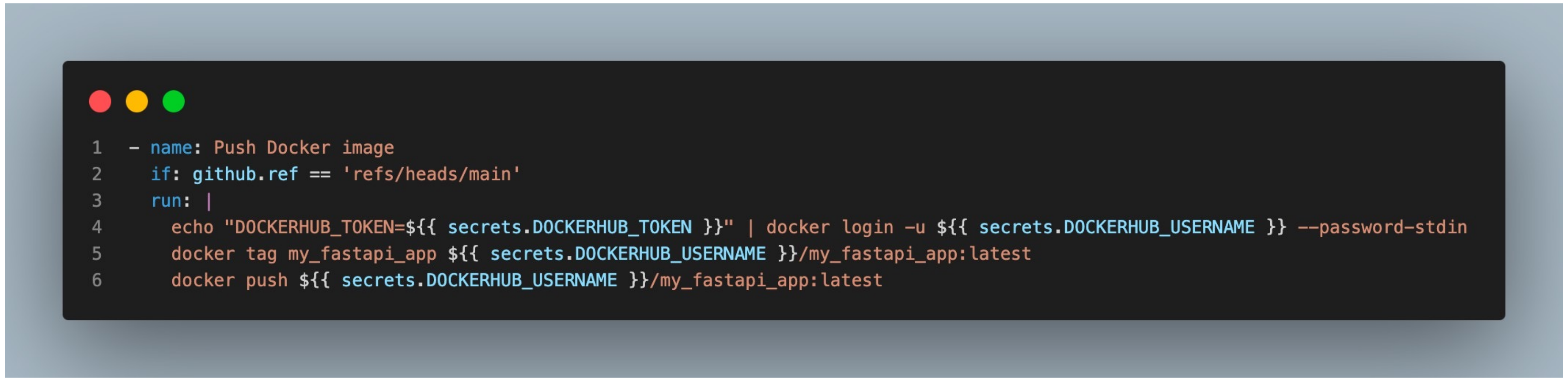
- **name:** Build Docker (step name).
- **run:** Runs a shell command to build a Docker image named my_fastapi_app using the Dockerfile in the current directory (.).



```
- name: Build Docker image
  run: |
    docker build -t my_fastapi_app .
```

Push Docker Image to Docker Hub!

- **name:** Push Docker image (step name).
- **if:** Conditional statement to execute this step only if the code is on the main branch.
- **run:** Runs a series of shell commands to push the image to the Docker Hub.

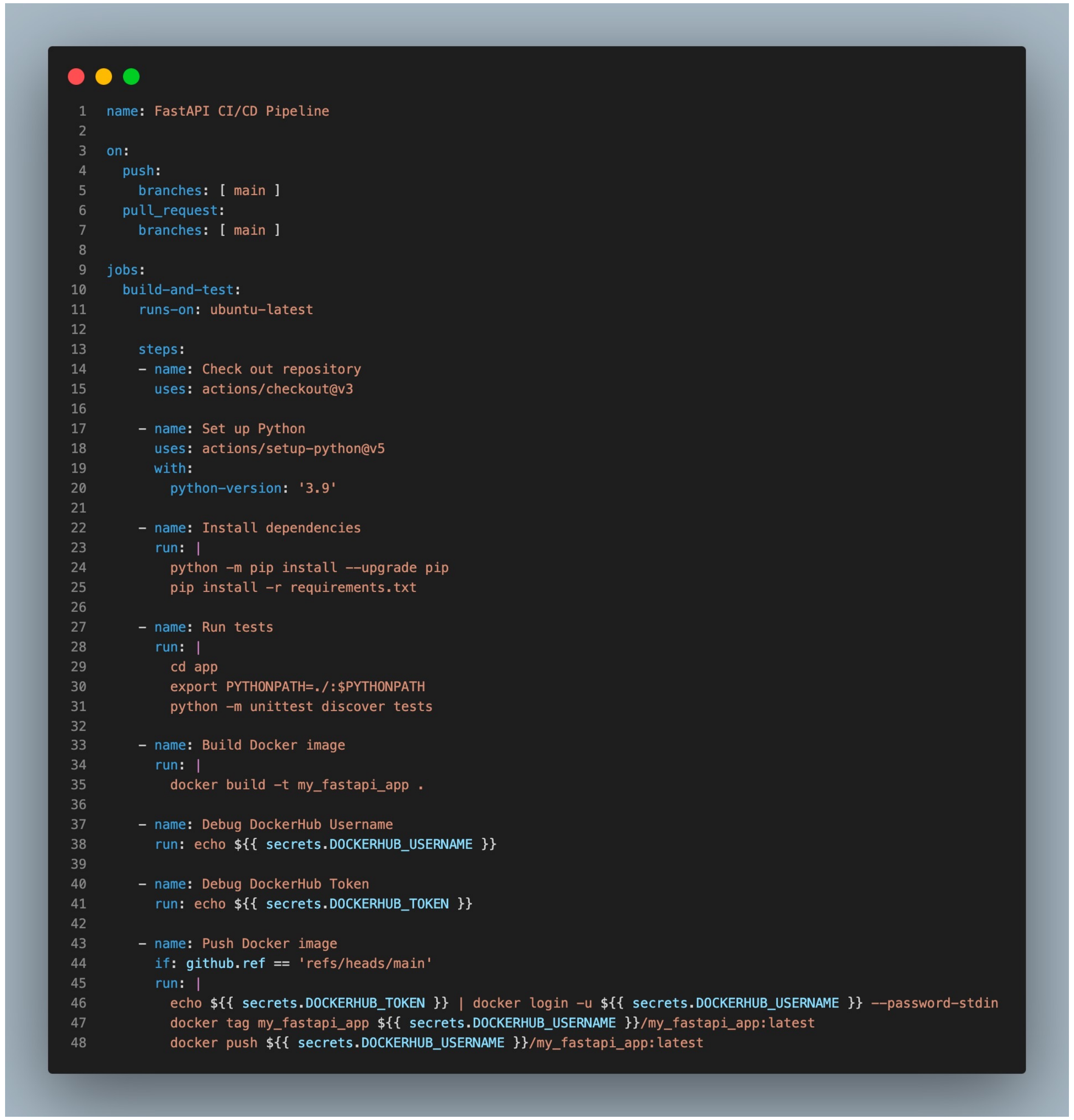


```
- name: Push Docker image
  if: github.ref == 'refs/heads/main'
  run: |
    echo "DOCKERHUB_TOKEN=${{ secrets.DOCKERHUB_TOKEN }}" | docker login -u ${{ secrets.DOCKERHUB_USERNAME }} --password-stdin
    docker tag my_fastapi_app ${{ secrets.DOCKERHUB_USERNAME }}/my_fastapi_app:latest
    docker push ${{ secrets.DOCKERHUB_USERNAME }}/my_fastapi_app:latest
```

Complete code

The complete code is uploaded under the name of Dockerfile in the example file!

The floder used is uploaded named ‘example’.



```yaml
name: FastAPI CI/CD Pipeline

on:
  push:
    branches: [ main ]
  pull_request:
    branches: [ main ]

jobs:
  build-and-test:
    runs-on: ubuntu-latest

    steps:
    - name: Check out repository
      uses: actions/checkout@v3

    - name: Set up Python
      uses: actions/setup-python@v5
      with:
        python-version: '3.9'

    - name: Install dependencies
      run: |
        python -m pip install --upgrade pip
        pip install -r requirements.txt

    - name: Run tests
      run: |
        cd app
        export PYTHONPATH=./:$PYTHONPATH
        python -m unittest discover tests

    - name: Build Docker image
      run: |
        docker build -t my_fastapi_app .

    - name: Debug DockerHub Username
      run: echo ${{ secrets.DOCKERHUB_USERNAME }}

    - name: Debug DockerHub Token
      run: echo ${{ secrets.DOCKERHUB_TOKEN }}

    - name: Push Docker image
      if: github.ref == 'refs/heads/main'
      run: |
        echo ${{ secrets.DOCKERHUB_TOKEN }} | docker login -u ${{ secrets.DOCKERHUB_USERNAME }} --password-stdin
        docker tag my_fastapi_app ${{ secrets.DOCKERHUB_USERNAME }}/my_fastapi_app:latest
        docker push ${{ secrets.DOCKERHUB_USERNAME }}/my_fastapi_app:latest
```

References:

- https://dagshub.com/glossary/mlops-monitoring/
- https://www.abtasty.com/blog/deployment-strategies/#:~:text=In%20that%20sense%2C%20a%20deployment,available%20to%20its%20intended%20users.
- https://www.harrisonclarke.com/blog/the-role-of-mlops-in-explainable-ai-use-cases-and-approaches
- https://www.qwak.com/post/top-ml-model-monitoring-tools