**Date: 06.05.2024**

**Topic: Docker**

**Docker:**

"Dockerizing" refers to the process of packaging an application and its dependencies into a Docker container. Docker is a platform that allows developers to build, ship, and run applications in containers, which are lightweight, portable, and isolated environments.

Dockerizing an application involves creating a Dockerfile, which contains instructions for building the Docker image, and then building the image using the Docker CLI. Once the image is built, it can be run as a container on any system that has Docker installed, regardless of the underlying operating system or environment.

**Docker**

**Docker**

**Registry**

**Docker**

**hub**

**Docker**

**Image**

**Docker Container**

**Docker Container:** A Docker container is a lightweight, standalone, executable software package that includes everything needed to run a piece of software, including the code, runtime, system tools, system libraries, and settings. Containers are based on Docker images, which are read-only templates that contain the instructions for creating a container.

**Docker Image:** A Docker image is a lightweight, standalone, executable software package that contains everything needed to run a piece of software, including the code, runtime, system tools, system libraries, and settings. Docker images are built from a Dockerfile, which contains instructions for creating the image.

**Docker Hub:** Docker Hub is a cloud-based repository provided by Docker that allows developers to store, manage, and share Docker images. It serves as a central registry for Docker images, where users can upload their images or download images created by others.

**Docker Registry:** A Docker registry is a service responsible for storing and distributing Docker images. It serves as a central repository where Docker images can be pushed, pulled, and managed by users. Docker Hub, which I mentioned earlier, is an example of a public Docker registry provided by Docker, but users can also set up their private registries for internal use.

**For Linux Docker Commands:**

Step 1: Create a new directory – mkdir myapp

Cd myapp

Step 2: Create a file called “index.html” – echo “Hello world!” > index.html

ls – used to list the files in the directory

cat index.html – display the content in that file

(or)

To create a new file – touch demo.html

Write inside the file – vi demo.html

Then will open a new page – click i for insert, after typing something

Click esc 🡪:wq – write and quite

Step 3: Create a file named Dockerfile – touch Dockerfile

Step 4: Open the “Dockerfile” file in a text editor and add the following line

FROM nginx

COPY index.html /usr/share/nginx/html

A docker file is a text file with instructions to build a docker Image.

When we run the Dockerfile, a Docker image is created.

When we run the docker image, containers are created.

Step 5: Start docker & build docker image from the docker file

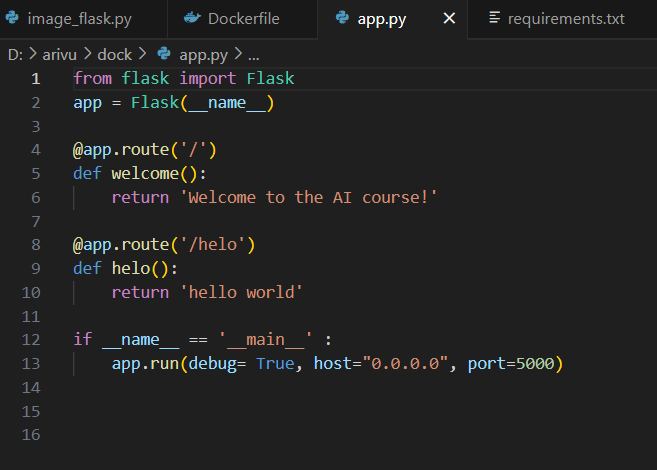
Docker build -t myapp . 🡪 This command builds a new docker image with the “myapp” using the Dockerfile in the current directory.

Docker images 🡪 We can see the images

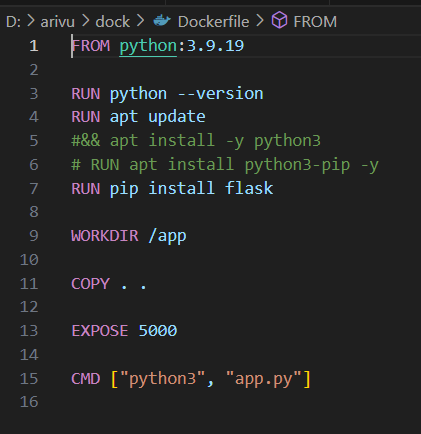
Step 6: run the docker container from the image

Docker run -p 5000:5000 myapp 🡪 This tells docker to run the myapp container and map port 5000 on your local machine to port 5000 inside the container. (we can give 8080:80 or 5000:50 like this)

Dockerizing the “Hello World” Program:



Here must define **host =** and **port =**



Here app means, creating a new working dir.

Here app.py is our python script.

A command for creating an image and run our docker container:

docker build -t dock . (dock means image name, we can give anything)

docker run -p 5000:5000 dock

**FastAPI in Docker:**

from transformers import AutoFeatureExtractor, ViTForImageClassification

from PIL import Image

#import io

import requests

import torch

from fastapi import FastAPI, Request

# Define the model and feature extractor (place outside any function)

model\_name = "facebook/deit-tiny-patch16-224"

feature\_extractor = AutoFeatureExtractor.from\_pretrained(model\_name)

model = ViTForImageClassification.from\_pretrained(model\_name)

app = FastAPI()

@app.post('/', status\_code= 200)

async def predict(info: Request):

# Check if an image file is present in the request

#if 'image' not in request.files:

#return jsonify({'error': 'No image file found in request'}), 400

#response = Request.get\_json()

response = await info.json()

print(response)

image = Image.open(requests.get(response["url"], stream=True).raw)

# Preprocess the image

inputs = feature\_extractor(images=image, return\_tensors="pt")

# Make prediction

# with torch.no\_grad(): # Disable gradient calculation for efficiency

outputs = model(\*\*inputs)

logits = outputs.logits

predicted\_class\_idx = logits.argmax(-1).item()

result = {'result': model.config.id2label[predicted\_class\_idx], 'message': 'Success'}

return result

For Dockerization, first create a separate folder with our Python program file, requirements file, and docker file.

In requirements file:

fastapi

redis

types-redis

uvicorn

transformers

pillow

torch

These dependencies are important to run our program.

Docker file:

