1) Start a new project called practice1. You're going to create an API. Put it inside a docker

Structure

├── app

│ └── main.py

│ └── requirements.txt

└── Dockerfile

Terminal:

1. **mkdir practice1**
   cd practice1
   Creating practice1 as the main directory helps keep all related files together.
2. **mkdir app** → creates a folder to contain our FastAPI app. The app folder acts like a container, making it easier to tell apart from Docker-related files.
3. **touch app/main.py** → This code sets up a basic FastAPI instance so that my app can respond to web requests.
   from fastapi import FastAPI
   app = FastAPI()
4. **Create a requirements.txt file inside the app.** Inside the .txt put
   fastapi
   uvicorn
   The requirements.txt lists the Python libraries the app depends on. Here, I've added fastapi to run the app and uvicorn to serve it. This file is essential for Docker, as it enables us to install these dependencies inside the container.
5. **cd /SCAV2024/practice1**
   touch Dockerfile

   # Dockerfile
   # Use the official Python image as a base
   FROM python:3.10-slim
   #https://katalon-inc.my.site.com/katalonhelpcenter/s/article/How-to-install-FFMPEG-on-Docker-Image
   # Set the working directory in the container
   WORKDIR /app
   # Copies requirements.txt from your local app folder into the container.
   COPY app/requirements.txt .
   # Installs FastAPI, Uvicorn, and other dependencies listed in requirements.txt.
   RUN pip install --no-cache-dir -r requirements.txt
   # Copy the FastAPI app code to the working directory

```
COPY app /app
# Exposes port 80, which is used to access the application from outside the container.
EXPOSE 80
# Starts the FastAPI app with Uvicorn, listening on all network interfaces.
CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "80"]
```

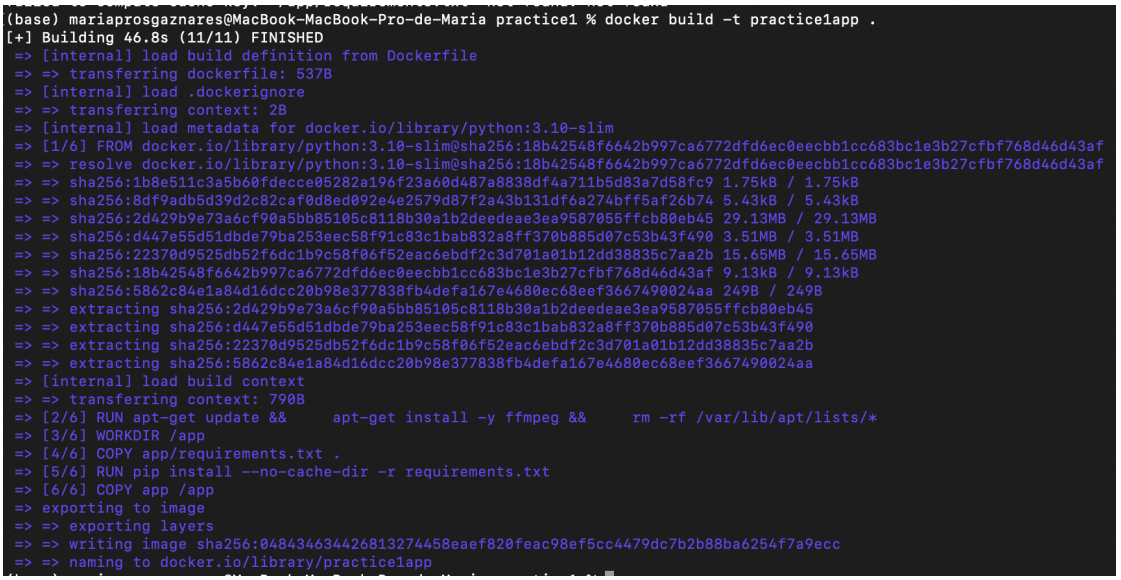
Okay now we have to test if it works. How do we do it?
We add a simple endpoint that will respond with a message when accessed in main.py:

```
@app.get("/")
async def root():
    return {"message": "Hello World!"}
```

Terminal:

1. **docker build -t practice1app .**

   Here, **docker build** tells Docker to create an image, -t practice1app tags the image with the name practice1app, and the . specifies that Docker should use the current directory (where the Dockerfile is located) to build the image. Docker will follow the steps in the Dockerfile, installing dependencies and copying files into the container.



2. **docker run -d -p 80:80 practice1app**

   Here, **docker run** starts a container from the image. The -d flag runs the container in detached mode, meaning it runs in the background, and -p 80:80 maps port 80 on your local machine to port 80 in the container, allowing you to access the FastAPI app at http://localhost/.

3. **http://localhost/**

2) Put ffmpeg inside a Docker

In the **Dockerfile** add:

```
RUN apt-get update && \
    apt-get install -y ffmpeg && \
    rm -rf /var/lib/apt/lists/*
```

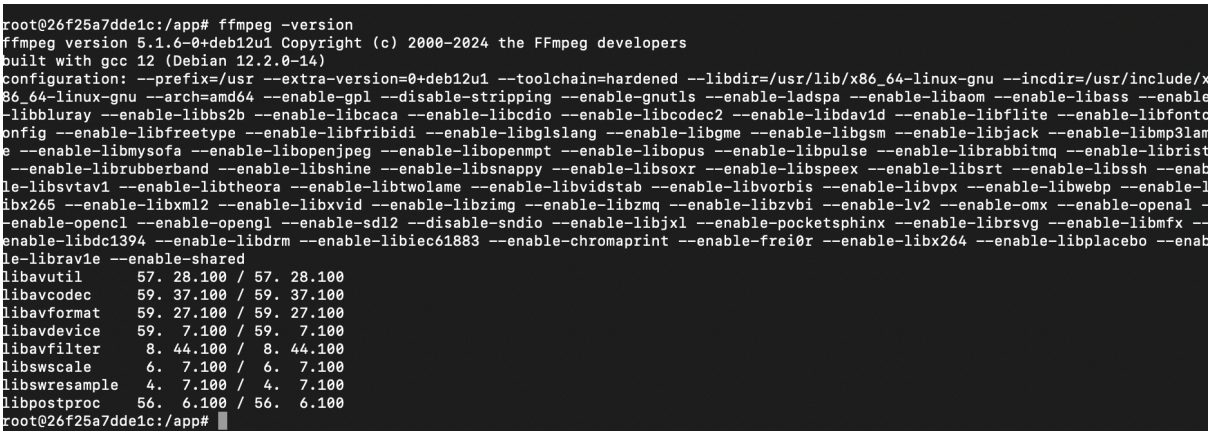
Here, ffmpeg allows for processing multimedia files. These commands update the package list, install ffmpeg, and then clean up to keep the container size small.

Okay now we have to test if it works. How do we do it?

1. **docker run -it practice1app /bin/bash**

   Runs the Docker Container in Interactive Mode: To test ffmpeg directly.

2. **ffmpeg -version**



Important Information we used to handle our docker container:

To check CONTAINER ID: **docker** ps

To stop container: **docker stop <container_id>**

Docker build and run: **docker build -t practice1app .**

docker run -d -p 80:80 practice1app

Interactive mode: **docker run -it practice1app /bin/bash**