TP04: Docker – Containerizing the TP3 Analysis

kiram mohammed ali

Introduction

The goal of TP04 is to containerize the TP3 data analysis project using Docker and Docker Hub.[1] The main tasks are:

1. Create a Docker Hub account.
2. Build a Docker image for TP3.
3. Push the image to Docker Hub.
4. Pull the uploaded image.
5. Create three containers from this image.
6. Create a custom Docker network.
7. Run the containers with port mapping.
8. Verify that all containers are running correctly.

The project contains at least the following files:

- `Dockerfile` .
- `requirements.txt` – Python dependencies (pandas, numpy, jupyter, matplotlib, seaborn, scikit-learn, etc.).
- `entrypoint.sh` – starts a simple HTTP server on port 8000 inside the container.
- Analysis code and CSV files used in TP3.

[1] See TP04 instructions.

Step 1: Create a Docker Hub Account

A personal account was created on Docker Hub. The public repository for this TP is named **mohammedali233/tp3**.

Overview

Creating a Docker Hub account allows us to store and share Docker images in a centralized cloud registry.

Screenshot

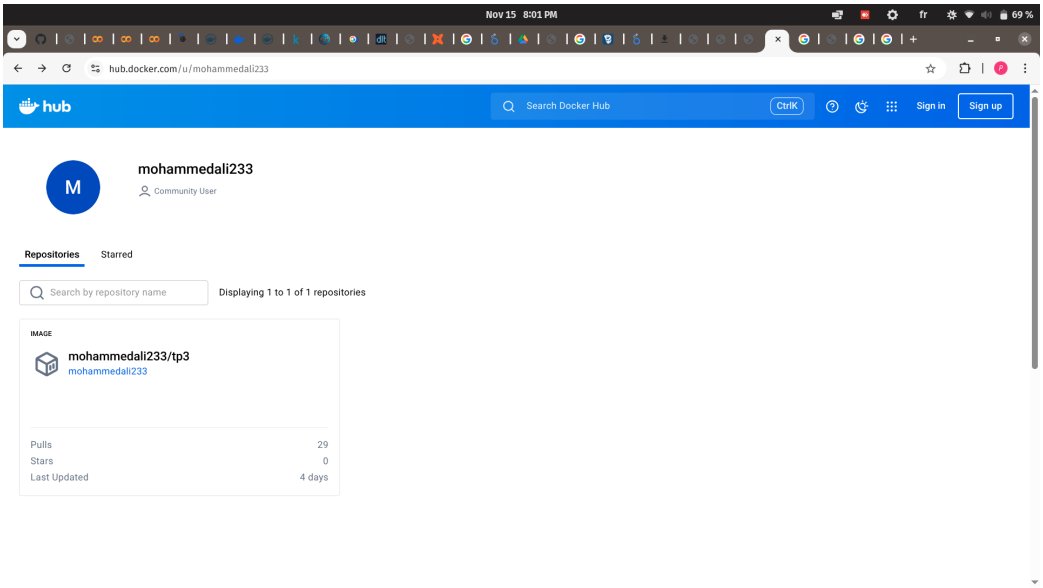

Figure 1: Docker Hub account and the `mohammedali233/tp3` repository.

Step 2: Build a Docker Image for TP3

The image is built locally from the project folder using a Dockerfile. A typical `requirements.txt` and entrypoint script are:

Requirements

```
pandas
numpy
pyarrow
papermill
jupyter
matplotlib
seaborn
scikit-learn
```

Entrypoint Script

```
#!/usr/bin/env bash
set -e

echo "[*] Serving analysis results on port 8000..."
cd /app
python3 -m http.server 8000
```

Example Dockerfile

```
FROM python:3.11-slim

WORKDIR /app

COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

COPY . .

EXPOSE 8000
RUN chmod +x entrypoint.sh
ENTRYPOINT ["./entrypoint.sh"]
```

Build Command

```
docker build -t mohammedali233/tp3 .
```

Overview

This image contains all Python dependencies and TP3 analysis files. When the container starts, `entrypoint.sh` launches a simple HTTP server on port 8000 to expose the analysis results (notebooks, CSVs, etc.).

Step 3: Push the Image to Docker Hub

After building the image locally, it is tagged and pushed:

```
docker login
docker tag mohammedali233/tp3 mohammedali233/tp3:latest
docker push mohammedali233/tp3:latest
```

Overview

Pushing the image to Docker Hub makes it available from any machine with Docker installed.

Step 4: Pull the Image from Docker Hub

On another machine (or after removing the local image), the uploaded image can be downloaded using:

```
docker pull mohammedali233/tp3:latest
```

Overview

This step validates that the repository was correctly configured and that the image is publicly accessible.

Step 5: Create Three Containers from the Image

We run three containers based on the downloaded image:

```
docker run -d --name tp3-container1 mohammedali233/tp3
docker run -d --name tp3-container2 mohammedali233/tp3
docker run -d --name tp3-container3 mohammedali233/tp3
```

Overview

Each container runs an independent instance of the TP3 analysis service. They all share the same image but are isolated processes.

Step 6: Create a Docker Network

To allow the containers to communicate, we create a custom bridge network:

```
docker network create tp3-network
```

Then we attach the containers to this network:

```
docker network connect tp3-network tp3-container1
docker network connect tp3-network tp3-container2
docker network connect tp3-network tp3-container3
```

Overview

Using a dedicated Docker network simplifies service discovery between containers and isolates them from other running services.

Step 7: Run Containers with Port Mapping

To access the HTTP server from the host, we expose container port 8000 on different host ports:

```
docker run -d --name tp3-container1 \
  --network tp3-network -p 8081:8000 mohammedali233/tp3

docker run -d --name tp3-container2 \
  --network tp3-network -p 8082:8000 mohammedali233/tp3

docker run -d --name tp3-container3 \
  --network tp3-network -p 8083:8000 mohammedali233/tp3
```

Overview

Each container listens on port 8000 internally, but the host accesses them using ports 8081, 8082, and 8083 respectively.

Screenshot

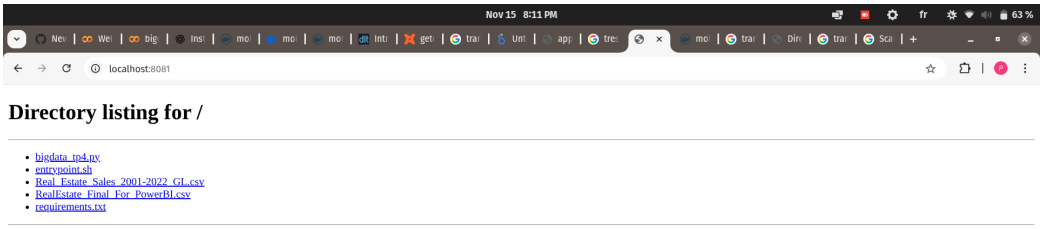

Figure 2: HTTP directory listing served by one container (`http://localhost:8081/`).

Step 8: Verify that All Containers Are Running

We verify the status of all containers with:

`docker ps`

Overview

The command `docker ps` lists all running containers, their image, status, exposed ports, and names. All three containers based on `mohammedali233/tp3` should appear with the correct port mappings.

Conclusion

In this TP, the TP3 analysis project was successfully containerized. The final setup includes:

- A reusable Docker image published on Docker Hub.
- Three running containers attached to a custom network.
- HTTP access to the analysis results via mapped ports on the host.

Docker and Docker Hub make it easy to distribute and reproduce the complete analysis environment on any machine.