■ Assignment 1 – Dockerized Python + PostgreSQL Stack

This project implements **Assignment 1 (ITCS 6190/8190)** using Docker.

It demonstrates how to containerize a simple **PostgreSQL database** and a **Python app** that connects to the DB, performs queries, and writes results to a shared volume.

■ Project Structure

CloudAssignment/
■■■ app/
■ ■■■ Dockerfile # Python app image
■ ■■■ main.py # Python script that queries DB & writes summary
■■■ db/
■ ■■■ Dockerfile # PostgreSQL image with init script
■ ■■■ init.sql # Schema + seed data (20 trips across 5 cities)
■■■ out/ # Host folder mounted into container (/out)
■ ■■■ summary.json # Output file (auto-generated by app)
■■■ compose.yaml # Docker Compose stack definition
■■■ makefile # Shortcuts: build, run, clean, etc.
■■■ README.md # Documentation (this file)

■ Database (PostgreSQL)

• Runs PostgreSQL **16** inside a container.
• Uses `init.sql` to **auto-create schema** and **insert seed data** on first start.
• Data model:

```
CREATE TABLE trips (
id SERIAL PRIMARY KEY,
city TEXT NOT NULL,
minutes INT NOT NULL,
fare NUMERIC(6,2) NOT NULL
);
```

• Seeded with **20 trips** across 5 cities (Charlotte, New York, San Francisco, Chicago, Boston).

■ Application (Python)

• Runs inside a `python:3.11-slim` container.
• Uses [psycopg3](https://www.psycopg.org/psycopg3/) to connect to PostgreSQL.
• Reads DB connection info from **environment variables** (`DB_HOST`, `DB_PORT`, etc.).

• Performs 3 queries:
1. Count total number of trips.
2. Compute **average fare per city**.
3. Find **top N longest trips** by minutes (default `N=5`, configurable via `APP_TOP_N`).
• Outputs:
• - Prints a JSON **summary** to stdout.
• - Saves the same JSON into `out/summary.json` (via mounted volume).

■■ How to Run

Option A – Using `make`

From the project root:
make all
This will:
1. Stop any running containers.
2. Clean and recreate the `out/` folder.
3. Build images and start the stack.

Option B – Using Docker Compose directly

docker compose up --build

■ Example Output

When the app runs, it prints a JSON summary:

{
"total_trips": 20,
"avg_fare_by_city": [
{"city": "Boston", "avg_fare": 23.28},
{"city": "Charlotte", "avg_fare": 18.41},
{"city": "Chicago", "avg_fare": 24.65},
{"city": "New York", "avg_fare": 24.55},
{"city": "San Francisco", "avg_fare": 23.93}
],
"top_by_minutes": [
{"city": "New York", "minutes": 42, "fare": 45.2},
{"city": "Boston", "minutes": 38, "fare": 36.25},
{"city": "Charlotte", "minutes": 35, "fare": 32.4},
{"city": "San Francisco", "minutes": 33, "fare": 35.8},

{"city": "Chicago", "minutes": 31, "fare": 30.75}
]
}

This exact file is also saved under:
out/summary.json

■ How to Stop & Clean Up

Stop containers & remove volumes

docker compose down -v

Or use Makefile

make clean

■ Troubleshooting

• **Error: `failed to read dockerfile`**
→ Ensure `db/Dockerfile` and `app/Dockerfile` exist.

• **App says "Waiting for database..."**
→ Normal on startup; the app retries until the DB is healthy.

• **Port conflict on 5432**
→ Stop local Postgres (`brew services stop postgresql`) or map to a different port in `compose.yaml`.

• **No `summary.json` generated**
→ Ensure `out/` folder exists and is mounted (`volumes:` section in compose).

• **Network timeout pulling images**
→ Restart Docker Desktop, switch to a stable network, or manually pull images:
docker pull postgres:16
docker pull python:3.11-slim

■ Reflection

Through this assignment, I learned:
• How to containerize both **database and app layers**.
• How to use **Docker Compose** for multi-service orchestration.
• How to use **volumes** to share output between containers and host.
• Importance of **healthchecks** and retry logic to ensure reliable startup.

For future improvement, I would:

• Add unit tests for the Python logic.
• Parameterize city filters and queries via environment variables.
• Push images to a private Docker Hub repo for easier reuse.

■ Submission Notes

• Repo: *(add your GitHub repo URL here)*
• One-page PDF includes:
• - Build/run commands.
• - Example summary JSON (stdout + `summary.json`).
• - Reflection (above).