AG-PST DevOps-Driven Data Ingestion and Analysis

Objective:

The objective of this assignment is to **design and implement a containerized data ingestion and analysis pipeline**. The solution demonstrates how to:

1. Select a complex dataset that maps naturally into a relational database schema.
2. Build a **Dockerized environment** with a database and ingestion service.
3. Ingest datasets into normalized tables using **Python and Pandas**.
4. Run **meaningful queries** (aggregations, joins, inserts) on demand.
5. Showcase **DevOps best practices** such as modularity, environment isolation, reproducibility, and idempotency.

Dataset Selection:

We used the **TMDB 5000 Movies dataset** (public dataset from Kaggle)
Url: https://www.kaggle.com/datasets/tmdb/tmdb-movie-metadata?select=tmdb_5000_credits.csv

This dataset is good, relational, and widely recognized. It was split into three CSVs for ingestion:

- **tmdb_5000_movies1.csv** renamed as **movie_metadata**
  Columns in this csv: id, budget, homepage, original_language, original_title
- **tmdb_5000_movies2.csv** renamed as **movie_details**
  Columns in this csv: id, popularity, release_date, revenue, runtime, status, vote_average, vote_count
- **tmdb_5000_movies3.csv** renamed as **movie_castcrew**
  Columns in this csv: movie_id, title, tagline

Reason behind selecting this dataset:

Fits **multi-table relational modeling** (metadata, details, castcrew) The **TMDB 5000 Movies dataset** was chosen because it naturally lends itself to a **relational database model** while also being rich enough to demonstrate real-world data engineering practices. The dataset contains multiple aspects of movies such as metadata (budget, language, title), details (revenue, runtime, votes, ratings), and cast/crew information that can be cleanly separated into different tables and connected through foreign keys. This ensures that the project is not limited to flat file ingestion but instead highlights **database normalization, relationships, and joins**.Additionally, this dataset supports **meaningful analytical queries** such as finding the top movies by revenue, aggregating ratings with vote thresholds, or joining movie details with cast/crew information.

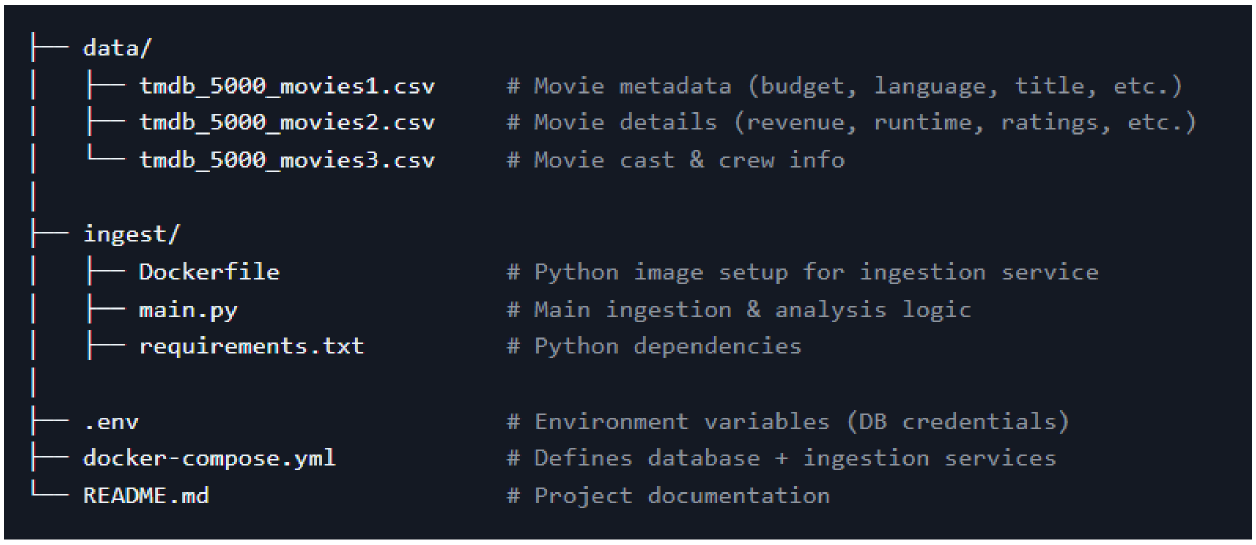
Database Choice:

We chose **PostgreSQL 15** because:

- **Open-source, stable, production-grade** relational database widely used in industry
- Excellent support for **normalized schemas** with primary/foreign keys and constraints
- Strong **Python integration** via SQLAlchemy + psycopg2, making ingestion seamless
- Official lightweight **Docker image available**, making containerization simple and reliable
- Provides **robust query optimization** and indexing for analytical queries
- Ensures **data integrity and scalability**, suitable for real-world ingestion pipeline

Alternative DBs like MySQL or DuckDB could also work, but PostgreSQL offers the most **robust relational capabilities** for this project.

Repository Setup:



```
├── data/
│   ├── tmdb_5000_movies1.csv     # Movie metadata (budget, language, title, etc.)
│   ├── tmdb_5000_movies2.csv     # Movie details (revenue, runtime, ratings, etc.)
│   └── tmdb_5000_movies3.csv     # Movie cast & crew info
│
├── ingest/
│   ├── Dockerfile                # Python image setup for ingestion service
│   ├── main.py                   # Main ingestion & analysis logic
│   ├── requirements.txt          # Python dependencies
│
├── .env                          # Environment variables (DB credentials)
├── docker-compose.yml            # Defines database + ingestion services
└── README.md                     # Project documentation
```

The repository is structured to separate concerns clearly. Each folder and file serves a distinct purpose, making the project easy to understand, maintain, and to extend.

- **data/** → Contains the raw datasets (tmdb_5000_movies1.csv, tmdb_5000_movies2.csv, tmdb_5000_movies3.csv) used for ingestion. Keeping them in a dedicated folder ensures that data is organized and version-controlled separately from code.
- **ingest/** → Holds all the ingestion logic and supporting files:
  - **Dockerfile** → Builds a lightweight Python container for ingestion.
  - **main.py** → The core ingestion script that connects to PostgreSQL, loads CSVs into tables, and runs queries.
  - **requirements.txt** → Lists dependencies (pandas, sqlalchemy, psycopg2-binary) to guarantee reproducibility.
- **.env** → Stores sensitive configuration values (DB user, password, and database name). Using environment variables follows 12-factor app principles, keeps secrets out of code, and allows different environments (dev, test, prod) without code changes.

- **docker-compose.yml** → Defines and orchestrates multiple services:
    - postgres_db (database service)
    - ingest_service (Python ingestion service)

      Both run inside a shared Docker network, ensuring secure communication.
- **README.md** → Comprehensive documentation with setup instructions, project explanation, screenshots, and usage details.

Dockerized Solution:

We used **Docker Compose** with two services mainly db (PostgreSQL) and ingest_and_query (Python)

Service 1: db (PostgreSQL)

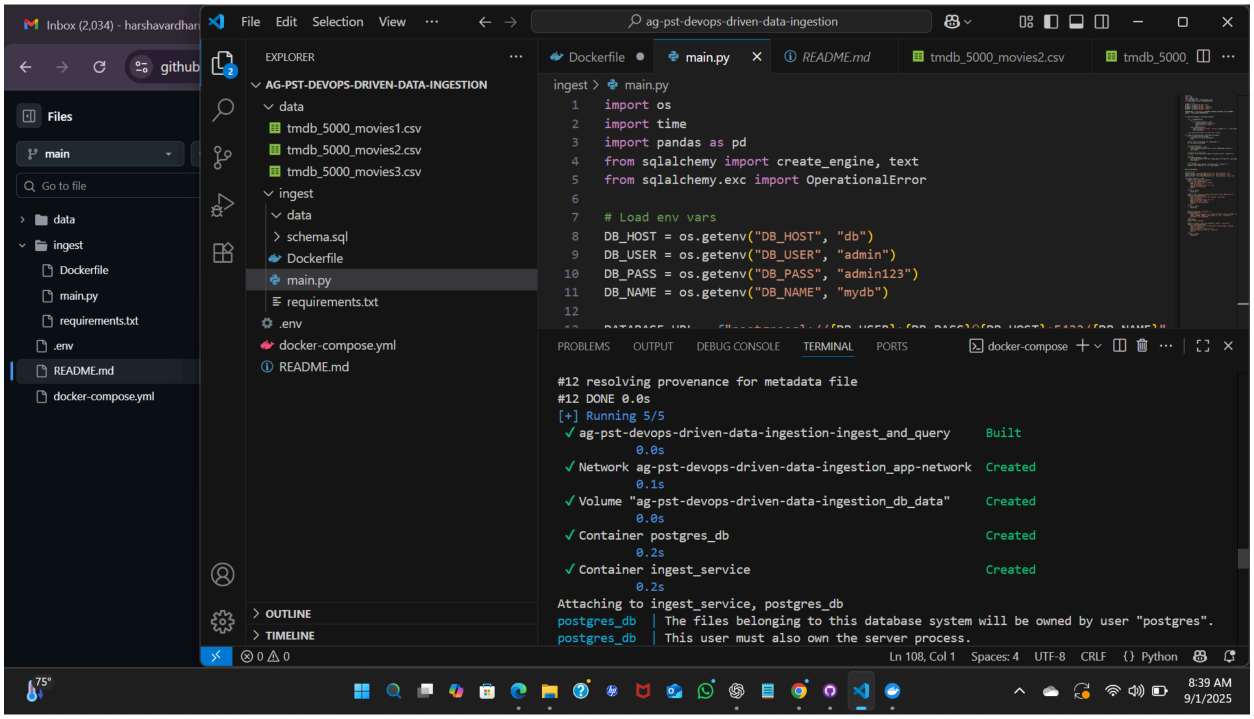
- Based on postgres 15 image.
- Credentials + DB name injected via .env.
- Exposes port 5432.
- Persists data in Docker volume db_data.
- Uses Docker volume (db_data) for persistence

Service 2: ingest_and_query (Python)

- Built with python:3.10-slim.
- Installs required libraries (pandas, sqlalchemy, psycopg2)
- Runs main.py.
    - Waits for DB readiness
    - Loads CSV data into tables (movie_metadata, movie_details, movie_castcrew)
    - Runs queries (aggregations, join, insert)
- Runs inside the same Docker network as postgres_db
- Executes sample SQL queries.

Both services run on the same **Docker network** for secure communication.

Docker Compose Build & Startup



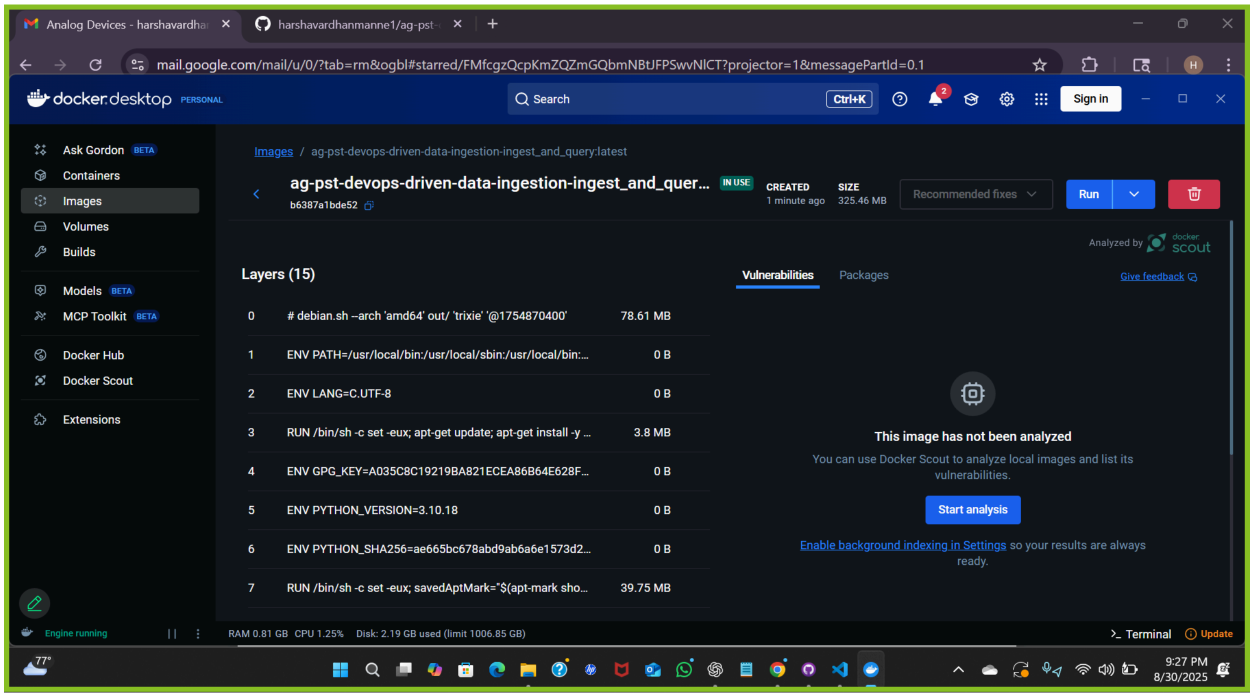
```python
import os
import time
import pandas as pd
from sqlalchemy import create_engine, text
from sqlalchemy.exc import OperationalError

# Load env vars
DB_HOST = os.getenv("DB_HOST", "db")
DB_USER = os.getenv("DB_USER", "admin")
DB_PASS = os.getenv("DB_PASS", "admin123")
DB_NAME = os.getenv("DB_NAME", "mydb")
```

```
#12 resolving provenance for metadata file
#12 DONE 0.0s
[+] Running 5/5
 ✓ ag-pst-devops-driven-data-ingestion-ingest_and_query     Built
           0.0s
 ✓ Network ag-pst-devops-driven-data-ingestion_app-network  Created
           0.1s
 ✓ Volume "ag-pst-devops-driven-data-ingestion_db_data"     Created
           0.0s
 ✓ Container postgres_db                                    Created
           0.2s
 ✓ Container ingest_service                                 Created
           0.2s
Attaching to ingest_service, postgres_db
postgres_db  | The files belonging to this database system will be owned by user "postgres".
postgres_db  | This user must also own the server process.
```

This screenshot shows the execution of docker-compose up --build, where:

- The ingest_and_query service image is built successfully.
- A dedicated Docker network (ag-pst-devops-driven-data-ingestion_app-network) is created for secure communication between services.
- A persistent volume (ag-pst-devops-driven-data-ingestion_db_data) is created to ensure PostgreSQL data survives container restarts.
- Both containers (postgres_db and ingest_service) are started successfully.

Docker Image Build



Images / ag-pst-devops-driven-data-ingestion-ingest_and_query:latest

ag-pst-devops-driven-data-ingestion-ingest_and_quer... IN USE
b6387a1bde52

CREATED 1 minute ago SIZE 325.46 MB

Layers (15)

0	# debian.sh --arch 'amd64' out/ 'trixie' '@1754870400'	78.61 MB
1	ENV PATH=/usr/local/bin:/usr/local/sbin:/usr/local/bin:...	0 B
2	ENV LANG=C.UTF-8	0 B
3	RUN /bin/sh -c set -eux; apt-get update; apt-get install -y ...	3.8 MB
4	ENV GPG_KEY=A035C8C19219BA821ECEA86B64E628F...	0 B
5	ENV PYTHON_VERSION=3.10.18	0 B
6	ENV PYTHON_SHA256=ae665bc678abd9ab6a6e1573d2...	0 B
7	RUN /bin/sh -c set -eux; savedAptMark="$(apt-mark sho...	39.75 MB

This image has not been analyzed

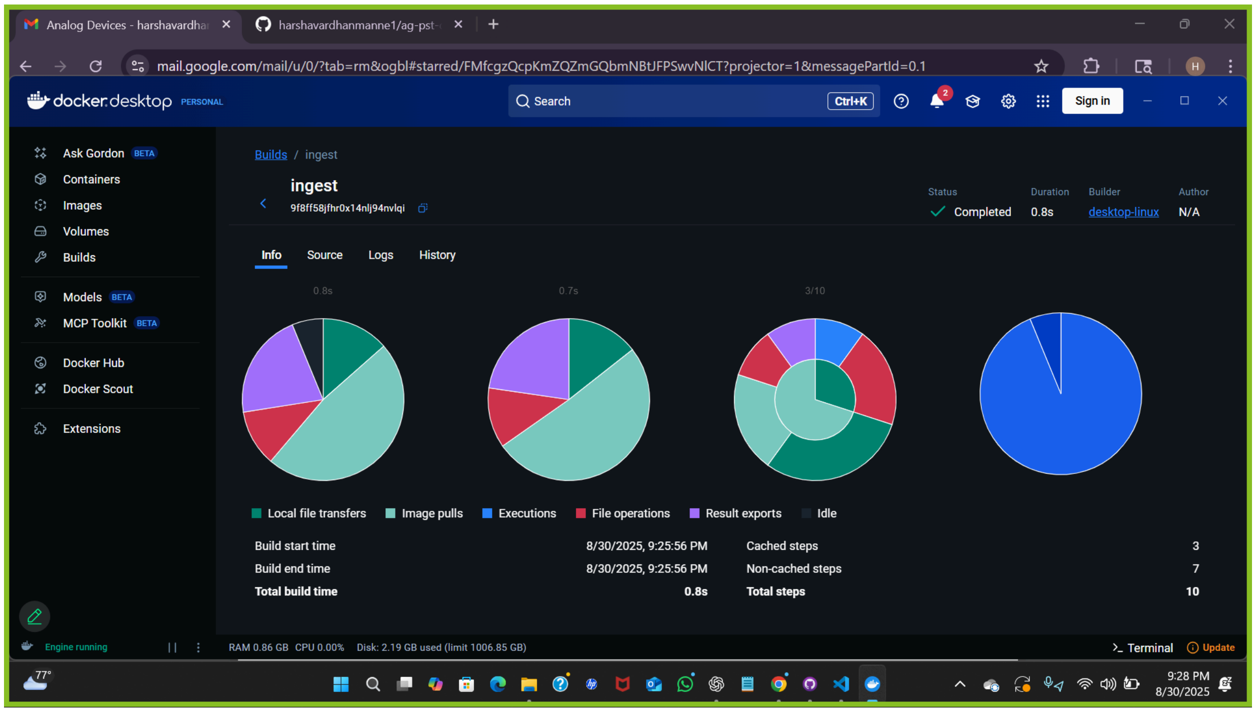
You can use Docker Scout to analyze local images and list its vulnerabilities.

Enable background indexing in Settings so your results are always ready.

RAM 0.81 GB CPU 1.25% Disk: 2.19 GB used (limit 1006.85 GB)

This screenshot shows the custom Python ingestion image successfully built using the ingest/Dockerfile. It validates that dependencies (pandas, sqlalchemy, psycopg2) were installed and the container is ready to run ingestion logic.

Docker Build Logs: These logs show the step-by-step build process for the Python service.



Builds / ingest

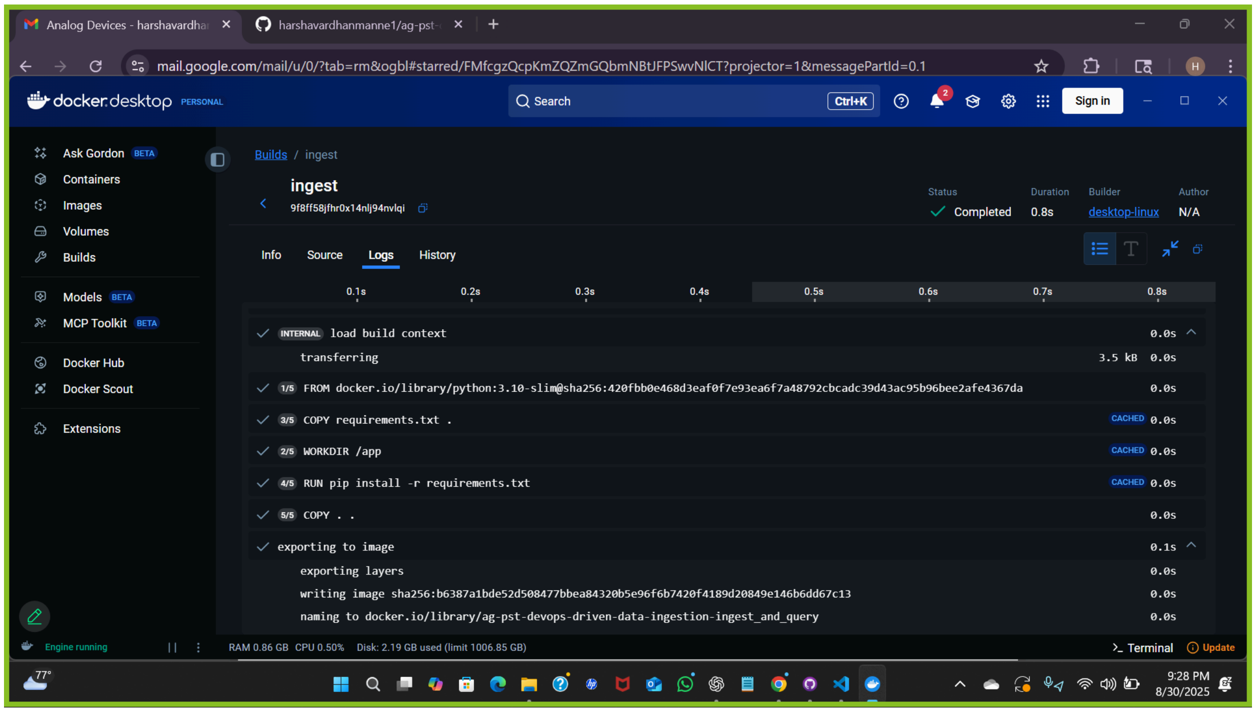
ingest
9f8ff58jfhr0x14nlj94nvlqi

Status	Duration	Builder	Author
Completed	0.8s	desktop-linux	N/A

Build start time	8/30/2025, 9:25:56 PM	Cached steps	3
Build end time	8/30/2025, 9:25:56 PM	Non-cached steps	7
Total build time	**0.8s**	**Total steps**	**10**

RAM 0.86 GB CPU 0.00% Disk: 2.19 GB used (limit 1006.85 GB)

This screenshot shows build stats & timing of Docker layers.



ingest
9f8ff58jfhr0x14nlj94nvlqi

Status: Completed · Duration: 0.8s · Builder: desktop-linux · Author: N/A

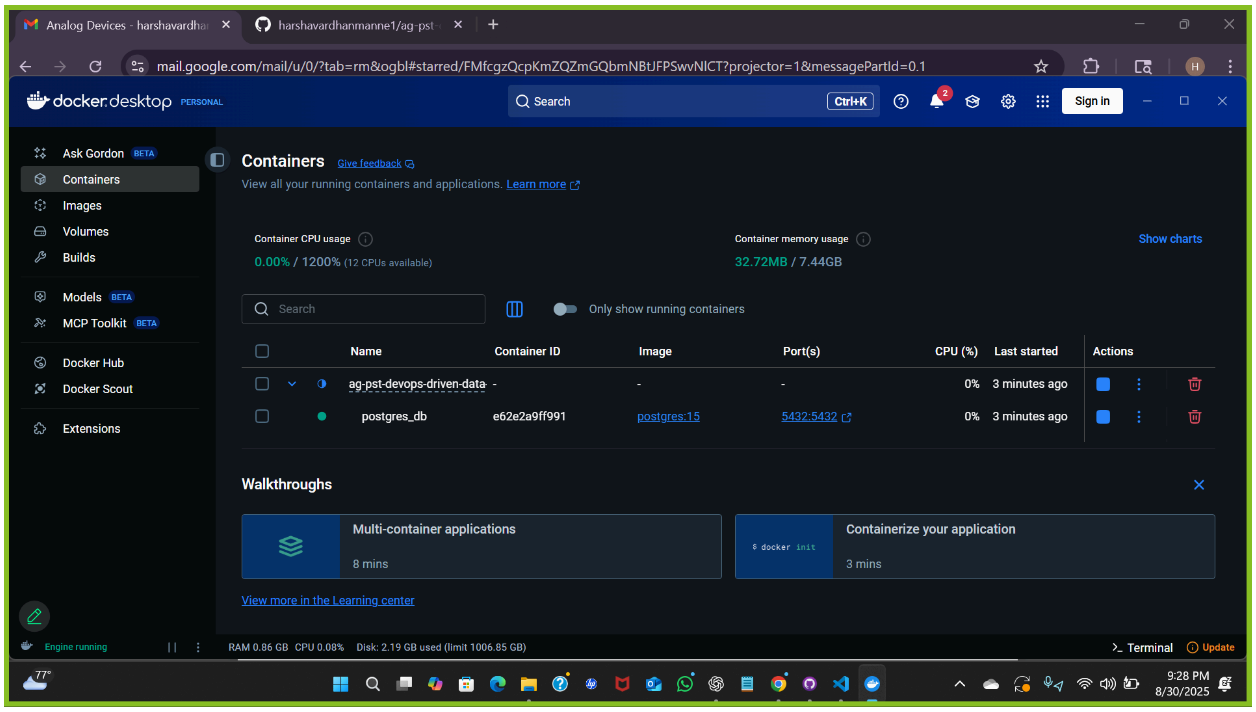
```
✓ INTERNAL load build context                                                                 0.0s
      transferring                                                                    3.5 kB  0.0s
✓ 1/5 FROM docker.io/library/python:3.10-slim@sha256:420fbb0e468d3eaf0f7e93ea6f7a48792cbcadc39d43ac95b96bee2afe4367da  0.0s
✓ 3/5 COPY requirements.txt .                                                          CACHED 0.0s
✓ 2/5 WORKDIR /app                                                                     CACHED 0.0s
✓ 4/5 RUN pip install -r requirements.txt                                              CACHED 0.0s
✓ 5/5 COPY . .                                                                                0.0s
✓ exporting to image                                                                          0.1s
      exporting layers                                                                        0.0s
      writing image sha256:b6387a1bde52d508477bbea84320b5e96f6b7420f4189d20849e146b6dd67c13   0.0s
      naming to docker.io/library/ag-pst-devops-driven-data-ingestion-ingest_and_query        0.0s
```

RAM 0.86 GB CPU 0.50% Disk: 2.19 GB used (limit 1006.85 GB)

This screenshot shows detailed installation logs, confirming all dependencies were installed correctly.

Runtime Phase

Running Containers



Containers Give feedback

View all your running containers and applications. Learn more

Container CPU usage 0.00% / 1200% (12 CPUs available)

Container memory usage 32.72MB / 7.44GB

Show charts

Only show running containers

Name	Container ID	Image	Port(s)	CPU (%)	Last started	Actions
ag-pst-devops-driven-data	-	-	-	0%	3 minutes ago	
postgres_db	e62e2a9ff991	postgres:15	5432:5432	0%	3 minutes ago	

Walkthroughs

Multi-container applications 8 mins

Containerize your application 3 mins

View more in the Learning center

RAM 0.86 GB CPU 0.08% Disk: 2.19 GB used (limit 1006.85 GB)

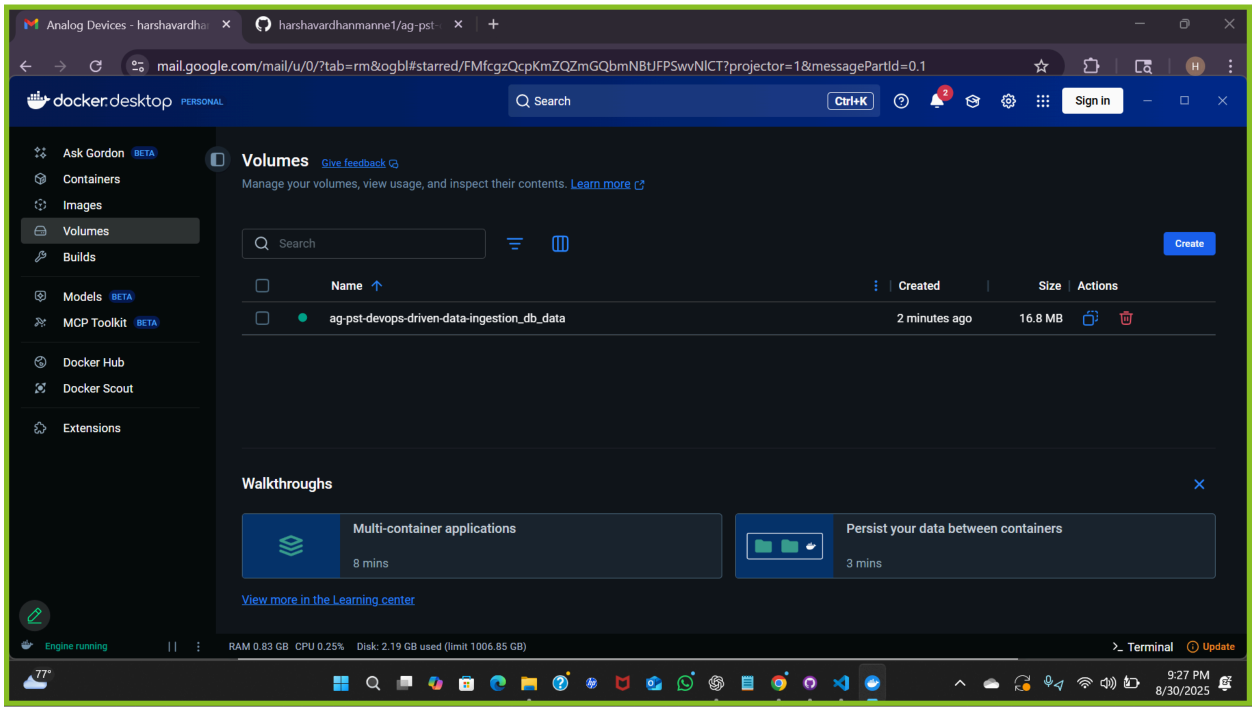
77° 9:28 PM 8/30/2025

This screenshot from Docker Desktop shows both services running:

- postgres_db → PostgreSQL container
- ingest_service → Python ingestion service

It proves that docker-compose.yml correctly orchestrated the services.

Docker Volumes


Analog Devices - harshavardhan
harshavardhanmanne1/ag-pst-
mail.google.com/mail/u/0/?tab=rm&ogbl#starred/FMfcgzQcpKmZQZmGQbmNBtJFPSwvNlCT?projector=1&messagePartId=0.1
docker desktop PERSONAL
Search
Ctrl+K
Sign in
Ask Gordon BETA
Containers
Images
Volumes
Builds
Models BETA
MCP Toolkit BETA
Docker Hub
Docker Scout
Extensions
Volumes
Give feedback
Manage your volumes, view usage, and inspect their contents. Learn more
Search
Create
Walkthroughs
Multi-container applications
8 mins
Persist your data between containers
3 mins
View more in the Learning center
Engine running
RAM 0.83 GB CPU 0.25% Disk: 2.19 GB used (limit 1006.85 GB)
Terminal
Update
77°
9:27 PM
8/30/2025

Name	Created	Size	Actions
ag-pst-devops-driven-data-ingestion_db_data	2 minutes ago	16.8 MB	

This screenshot shows the Docker volume db_data created for PostgreSQL persistence. It ensures that even if the container restarts, the data is not lost.

Database Schema:

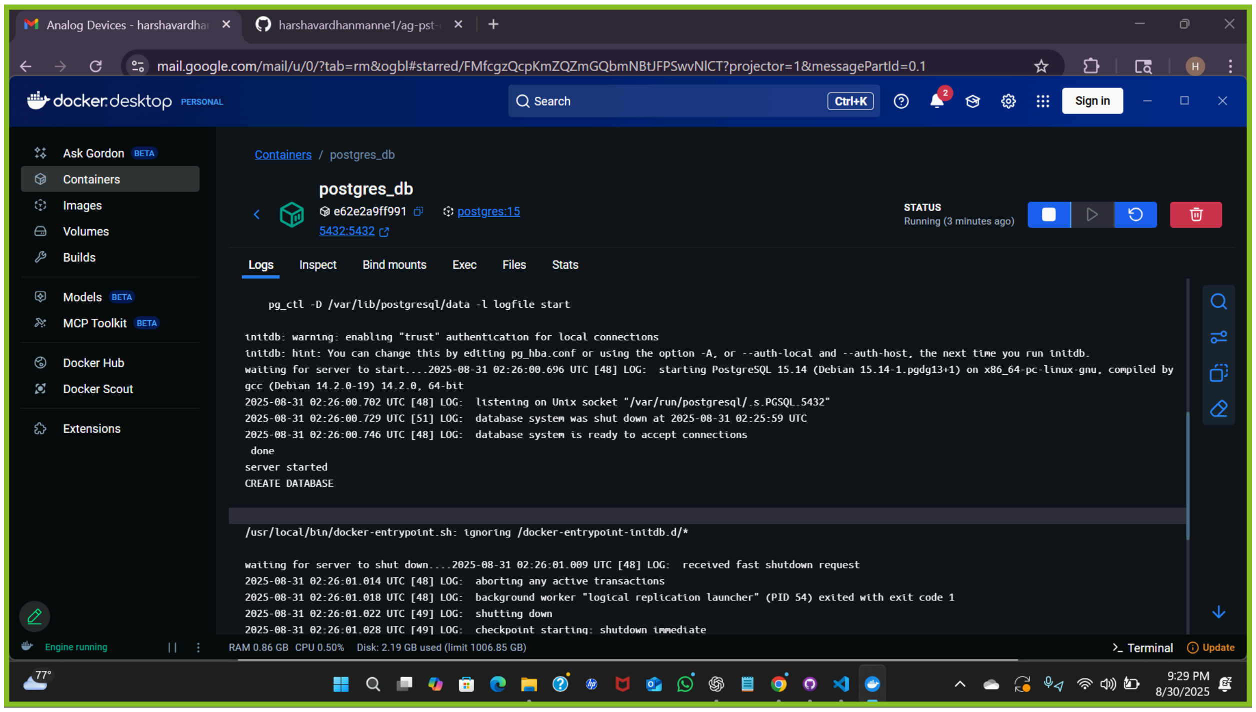
```
CREATE TABLE movie_metadata (
    id INT PRIMARY KEY,
    budget BIGINT,
    homepage VARCHAR,
    original_language VARCHAR,
    original_title VARCHAR
);

CREATE TABLE movie_details (
    id INT PRIMARY KEY,
    popularity FLOAT,
    release_date DATE,
    revenue BIGINT,
    runtime INT,
    status VARCHAR,
    vote_average FLOAT,
    vote_count INT,
    FOREIGN KEY (id) REFERENCES movie_metadata(id)
);
```

```
CREATE TABLE movie_castcrew (
    movie_id INT,
    title VARCHAR,
    tagline TEXT,
    FOREIGN KEY (movie_id) REFERENCES movie_metadata(id)
);
```

The schema was designed to be **normalized, relational, and extensible**,

- Different aspects of the dataset (metadata, details, cast/crew) are stored in separate tables, reducing redundancy.
- Foreign key relationships (e.g., movie_details.id → movie_metadata.id) ensure consistency between tables.
- New attributes or additional datasets can be added without disrupting existing tables now or later.
- Splitting into multiple tables allows targeted queries (ratings, revenue, or tagline lookups) without scanning unnecessary data.
- Normalization prevents data duplication, making storage efficient and queries faster as the dataset grows.
- Mimics how relational databases are used in production systems for movie catalogs, analytical dashboards, or recommendation engines

PostgreSQL Logs



```
    pg_ctl -D /var/lib/postgresql/data -l logfile start

initdb: warning: enabling "trust" authentication for local connections
initdb: hint: You can change this by editing pg_hba.conf or using the option -A, or --auth-local and --auth-host, the next time you run initdb.
waiting for server to start....2025-08-31 02:26:00.696 UTC [48] LOG:  starting PostgreSQL 15.14 (Debian 15.14-1.pgdg13+1) on x86_64-pc-linux-gnu, compiled by
gcc (Debian 14.2.0-19) 14.2.0, 64-bit
2025-08-31 02:26:00.702 UTC [48] LOG:  listening on Unix socket "/var/run/postgresql/.s.PGSQL.5432"
2025-08-31 02:26:00.729 UTC [51] LOG:  database system was shut down at 2025-08-31 02:25:59 UTC
2025-08-31 02:26:00.746 UTC [48] LOG:  database system is ready to accept connections
 done
server started
CREATE DATABASE


/usr/local/bin/docker-entrypoint.sh: ignoring /docker-entrypoint-initdb.d/*

waiting for server to shut down....2025-08-31 02:26:01.009 UTC [48] LOG:  received fast shutdown request
2025-08-31 02:26:01.014 UTC [48] LOG:  aborting any active transactions
2025-08-31 02:26:01.018 UTC [48] LOG:  background worker "logical replication launcher" (PID 54) exited with exit code 1
2025-08-31 02:26:01.022 UTC [49] LOG:  shutting down
2025-08-31 02:26:01.028 UTC [49] LOG:  checkpoint starting: shutdown immediate
```

This screenshot shows logs from the PostgreSQL container, confirming successful initialization and that the database is ready to accept connections. It validates the schema and DB service are running as expected.

Ingestion Workflow (main.py):

Steps performed:

1. **Wait for Database Readiness**
    - The ingestion service waits until the postgres_db container is fully up.
    - Prevents connection failures by retrying until PostgreSQL accepts connections.
2. **Read CSV Files with Pandas**
    - Loads each dataset (tmdb_5000_movies1.csv, tmdb_5000_movies2.csv, tmdb_5000_movies3.csv) into Pandas DataFrames.
    - Ensures the data is clean and structured before loading.
3. **Write to PostgreSQL using SQLAlchemy**
    - Uses df.to_sql() to insert data into corresponding tables (movie_metadata, movie_details, movie_castcrew).
    - Ensures type mapping and efficient bulk insert.
    - Uses append/replace mode to avoid duplicate schema creation.
4. **Run SQL Queries via SQLAlchemy**
    - Executes predefined SQL queries for analysis.
    - Results are printed in container logs for verification.
5. **Insert a Test Movie**
    - Demonstrates that the pipeline supports inserts in addition to reads.
    - Ensures idempotency by using ON CONFLICT DO NOTHING.
6. **Print results**
    - visible in Docker logs

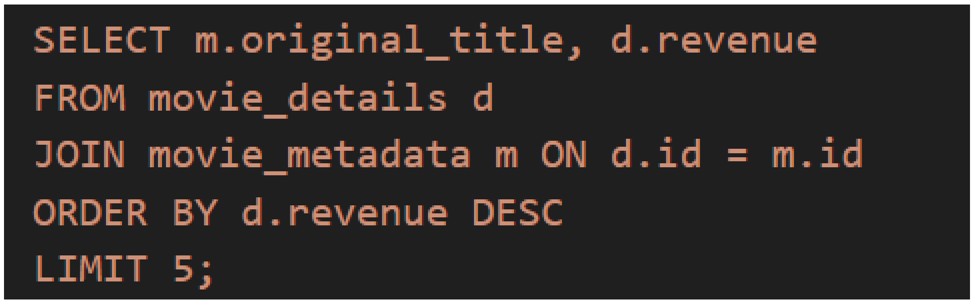
Queries Implemented:

The project includes **four categories of queries** to demonstrate practical use cases:

- Top 5 movies by revenue
- Top 5 movies by average rating (votes > 1000)
- Insert a test movie record
- Join metadata + details + castcrew

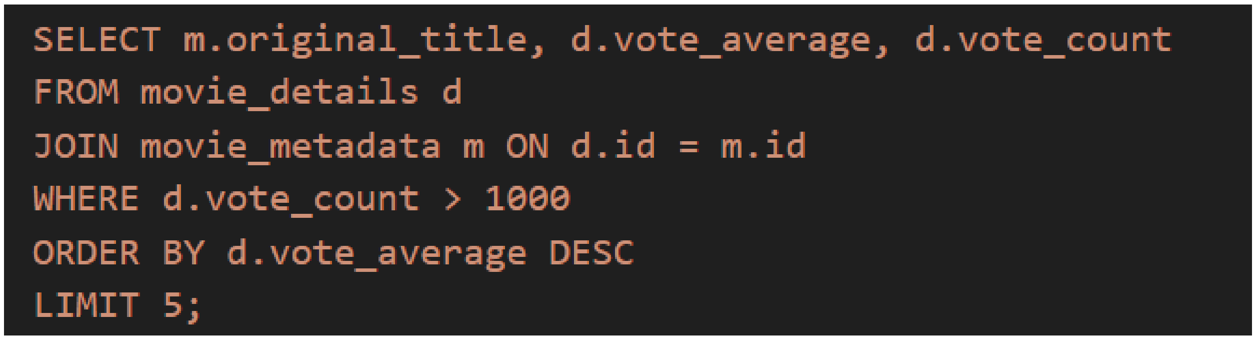
Aggregation Queries:

Top 5 Movies by Revenue



```
SELECT m.original_title, d.revenue
FROM movie_details d
JOIN movie_metadata m ON d.id = m.id
ORDER BY d.revenue DESC
LIMIT 5;
```

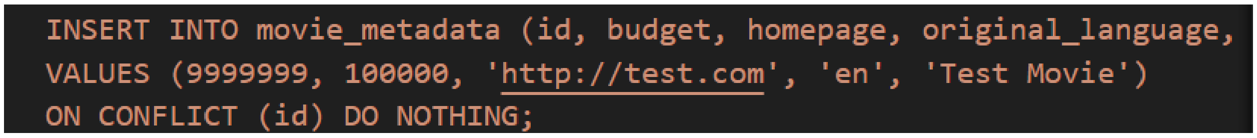
Top 5 Movies by Average Rating (votes > 1000)



```
SELECT m.original_title, d.vote_average, d.vote_count
FROM movie_details d
JOIN movie_metadata m ON d.id = m.id
WHERE d.vote_count > 1000
ORDER BY d.vote_average DESC
LIMIT 5;
```

Insert Query

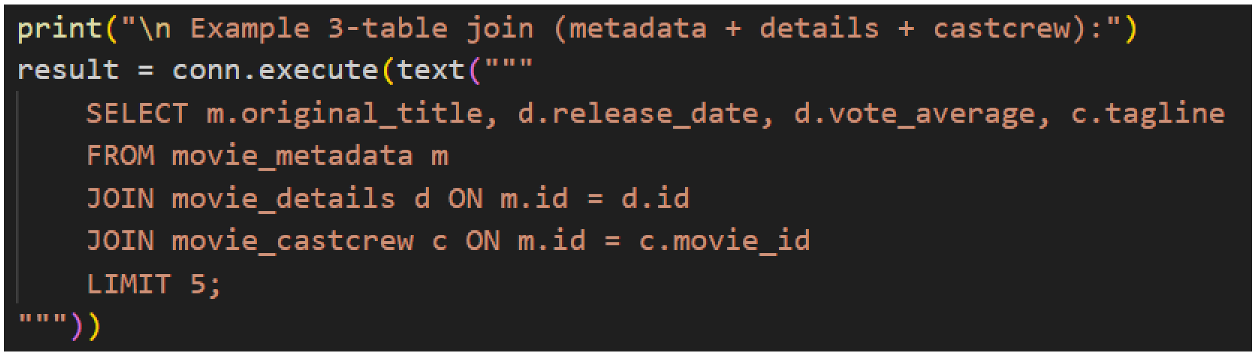
Add a Test Movie into Metadata



```
INSERT INTO movie_metadata (id, budget, homepage, original_language,
VALUES (9999999, 100000, 'http://test.com', 'en', 'Test Movie')
ON CONFLICT (id) DO NOTHING;
```

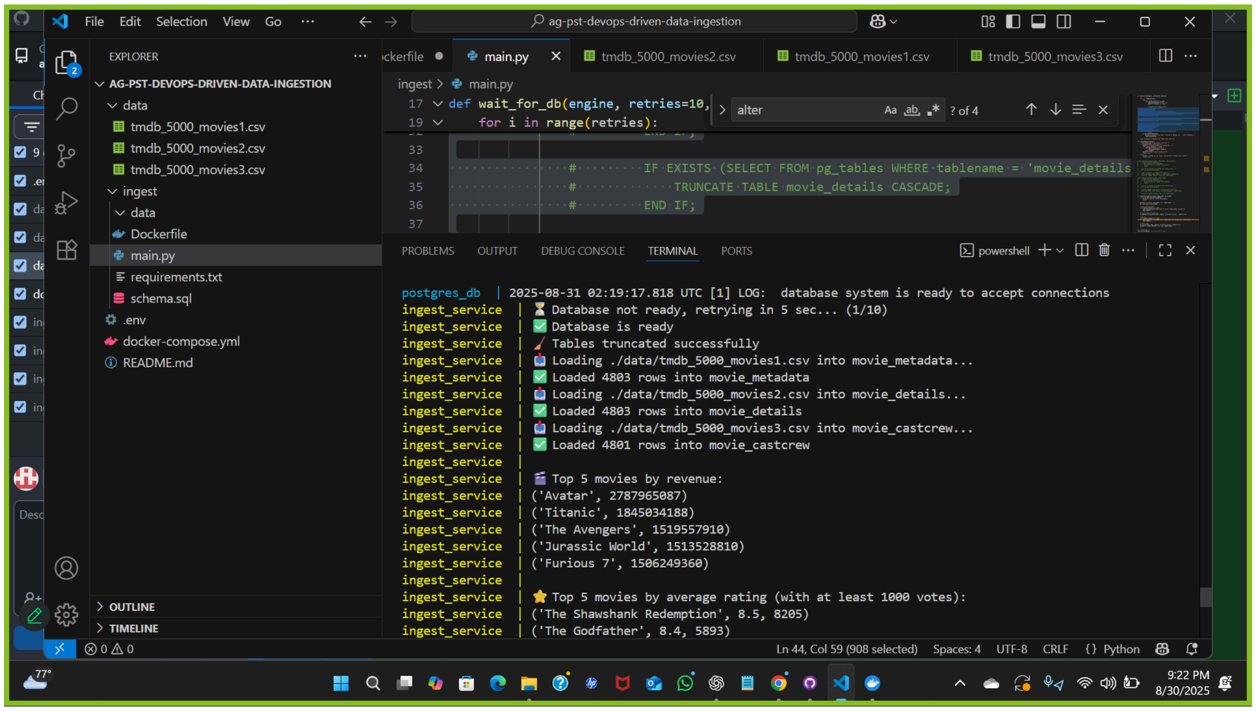
Join Query

Combine Metadata, Details, and CastCrew



```
print("\n Example 3-table join (metadata + details + castcrew):")
result = conn.execute(text("""
    SELECT m.original_title, d.release_date, d.vote_average, c.tagline
    FROM movie_metadata m
    JOIN movie_details d ON m.id = d.id
    JOIN movie_castcrew c ON m.id = c.movie_id
    LIMIT 5;
"""))
```

Query Outputs :

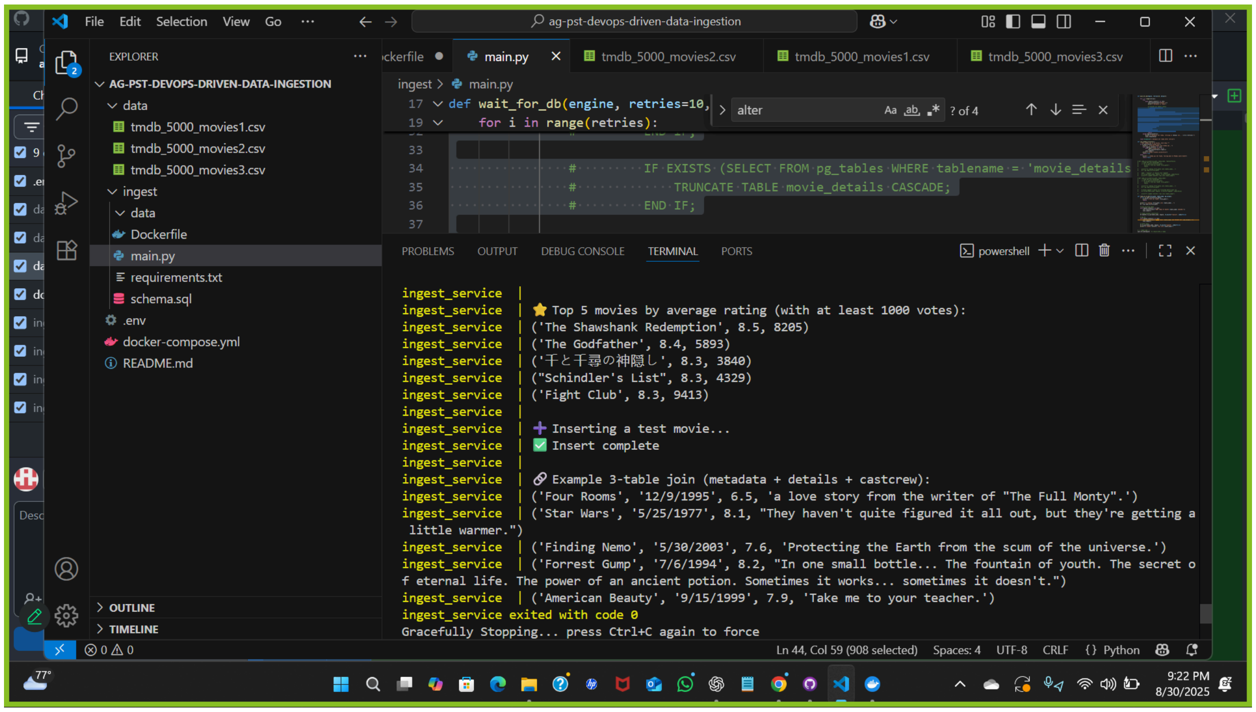


```
def wait_for_db(engine, retries=10,
    for i in range(retries):

        #        IF EXISTS (SELECT FROM pg_tables WHERE tablename = 'movie_details
        #            TRUNCATE TABLE movie_details CASCADE;
        #        END IF;
```

```
postgres_db  | 2025-08-31 02:19:17.818 UTC [1] LOG:  database system is ready to accept connections
ingest_service  | ⏳ Database not ready, retrying in 5 sec... (1/10)
ingest_service  | ✅ Database is ready
ingest_service  | 🧹 Tables truncated successfully
ingest_service  | 📥 Loading ./data/tmdb_5000_movies1.csv into movie_metadata...
ingest_service  | ✅ Loaded 4803 rows into movie_metadata
ingest_service  | 📥 Loading ./data/tmdb_5000_movies2.csv into movie_details...
ingest_service  | ✅ Loaded 4803 rows into movie_details
ingest_service  | 📥 Loading ./data/tmdb_5000_movies3.csv into movie_castcrew...
ingest_service  | ✅ Loaded 4801 rows into movie_castcrew
ingest_service  |
ingest_service  | 🎬 Top 5 movies by revenue:
ingest_service  | ('Avatar', 2787965087)
ingest_service  | ('Titanic', 1845034188)
ingest_service  | ('The Avengers', 1519557910)
ingest_service  | ('Jurassic World', 1513528810)
ingest_service  | ('Furious 7', 1506249360)
ingest_service  |
ingest_service  | ⭐ Top 5 movies by average rating (with at least 1000 votes):
ingest_service  | ('The Shawshank Redemption', 8.5, 8205)
ingest_service  | ('The Godfather', 8.4, 5893)
```

This screenshot shows the output of the query **Top 5 movies by revenue**. It validates that the dataset was ingested correctly and queries are running successfully inside the container.

Query Outputs (Rating, Insert, Join)



```
ingest_service  |
ingest_service  | ⭐ Top 5 movies by average rating (with at least 1000 votes):
ingest_service  | ('The Shawshank Redemption', 8.5, 8205)
ingest_service  | ('The Godfather', 8.4, 5893)
ingest_service  | ('千と千尋の神隠し', 8.3, 3840)
ingest_service  | ("Schindler's List", 8.3, 4329)
ingest_service  | ('Fight Club', 8.3, 9413)
ingest_service  |
ingest_service  | ➕ Inserting a test movie...
ingest_service  | ✅ Insert complete
ingest_service  |
ingest_service  | 🔗 Example 3-table join (metadata + details + castcrew):
ingest_service  | ('Four Rooms', '12/9/1995', 6.5, 'a love story from the writer of "The Full Monty".')
ingest_service  | ('Star Wars', '5/25/1977', 8.1, "They haven't quite figured it all out, but they're getting a
 little warmer.")
ingest_service  | ('Finding Nemo', '5/30/2003', 7.6, 'Protecting the Earth from the scum of the universe.')
ingest_service  | ('Forrest Gump', '7/6/1994', 8.2, "In one small bottle... The fountain of youth. The secret o
f eternal life. The power of an ancient potion. Sometimes it works... sometimes it doesn't.")
ingest_service  | ('American Beauty', '9/15/1999', 7.9, 'Take me to your teacher.')
ingest_service exited with code 0
Gracefully Stopping... press Ctrl+C again to force
```

This screenshot shows the results of multiple queries average rating, inserting test movie query and Join query (metadata + details + castcrew).