SkyLogix Weather Data Pipeline

1. Project Overview

This project implements an end-to-end data engineering pipeline that ingests real-time weather data from the OpenWeather API, stores raw data in MongoDB, transforms it into an analytics-ready format, and loads it into PostgreSQL. Apache Airflow orchestrates the pipeline, enabling scheduled and reliable data processing.

The final dataset supports analytics use cases such as weather trend analysis and correlation with logistics delays.

2. Architecture & Design

OpenWeather API

↓

Python Ingestion Script (raw JSON)

↓

MongoDB (weather_raw – upsert)

↓ (Airflow DAG)

Read from MongoDB → Transform → Load

↓

PostgreSQL (weather_readings – analytics)

↓

Dashboards / Reports

Key Design Choices

- **MongoDB** is used as a raw/staging layer to store semi-structured JSON payloads.
- **PostgreSQL** acts as the analytics warehouse with structured, query-optimized tables.
- **Airflow** schedules and orchestrates ingestion and transformation tasks.
- **Python** handles ingestion, transformation, and database interactions.

3. ETL Process Explaination

Extract

- Weather data is fetched from the OpenWeather API using Python.
- Data is ingested into MongoDB (weather_raw collection).

Transform

- Relevant fields (temperature, wind speed, rainfall, coordinates, conditions) are extracted.
- Data is normalized and timestamped.
- Incremental loading is supported using execution time.

Load

- Transformed data is upserted into PostgreSQL (weather_readings table).
- A unique constraint on (city, observed_at) prevents duplicates.

ETL successfully executed and verified via PostgreSQL queries and joins.

4. Orchestration (Airflow)

- DAG: weather_pipeline
- Schedule: Every 15 minutes
- Tasks:
    1. fetch_and_upsert_raw – API → MongoDB
    2. transform_and_load_postgres – MongoDB → PostgreSQL

Task execution and logs were validated via the Airflow UI and filesystem logs.

5. Analytics Enablement

Example Queries

- Average temperature per city (last 24 hours)
- Extreme weather detection (high wind, heavy rain)
- Weather vs logistics delay analysis (JOIN with trips table)

6. Assumptions

- - Real logistics trip data was not available; therefore, a simulated `logistics_trips` table was created.
- - This assumption allowed testing of joins, analytics queries, and downstream insights without dependency on external systems.

Weather & Logistics Correlation

- Weather observations are joined with logistics trip data using city-level matching and a 15-minute time window to account for ingestion latency.
- This enables analysis of whether weather conditions (e.g. wind, rain, cloud cover) correlate with delivery delays. Sample results show delayed trips in Lagos and Johannesburg even under mild weather conditions, demonstrating the model's ability to support root-cause analysis beyond weather alone.

7. Outcome

- The pipeline delivers a production-ready, analytics-enabled weather dataset that supports real-time monitoring and operational decision-making.

8. Technologies Used

- Python
- MongoDB
- PostgreSQL
- Apache Airflow
- SQL
- GitHub
- VS Code (WSL)