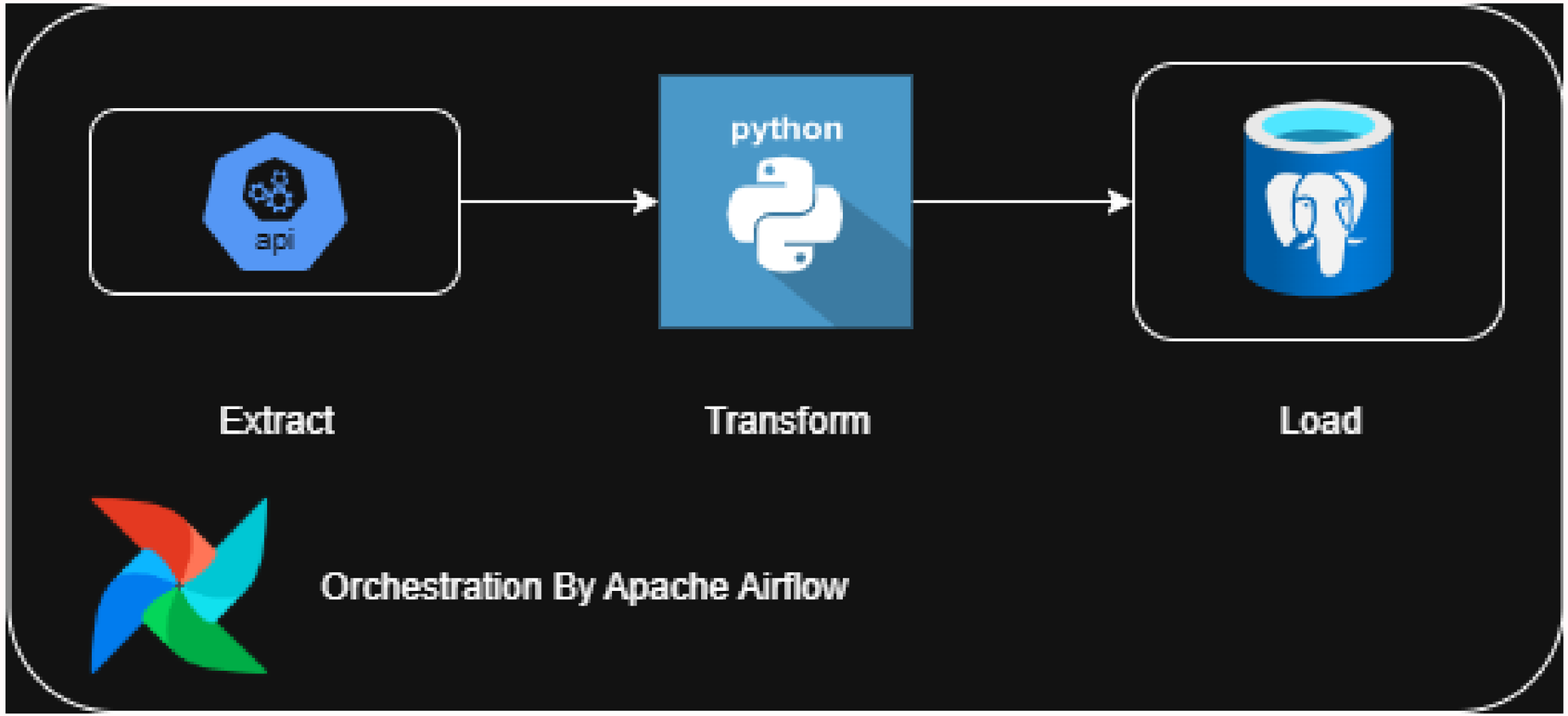
ETL Weather Data Pipeline: Implementation Guide

A comprehensive guide for implementing an automated ETL pipeline to retrieve real-time weather data from a public API, process it into a structured format, and load it into a persistent database for analysis and downstream applications.



Technology Stack Overview

Apache Airflow

Industry-standard workflow orchestration tool used to schedule, author, and monitor the data pipeline.

Docker

Containerization platform creating consistent, isolated environments for the pipeline's components.

PostgreSQL

Powerful, open-source database system used to store the processed weather data.

AWS

Target cloud platform for production deployment, utilizing Amazon RDS for managed PostgreSQL.

Prerequisites

Docker Desktop

Download from Docker website. Ensure the Docker daemon is running.

Visual Studio Code

Recommended code editor with integrated terminal. Download from VS Code website.

DBeaver

Free database tool for connecting to PostgreSQL. Download from DBeaver website.

AWS Account

Required for production deployment.

Local Environment Setup

Install Astro CLI

macOS & Linux: `/bin/bash -c "$(curl -sSL https://install.astronomer.io)"`

Windows: `Invoke-WebRequest -Uri "https://install.astronomer.io" -OutFile "install.ps1"; .\install.ps1`

Initialize Airflow Project

1. Create a dedicated folder (e.g., etl-weather-pipeline)
2. Navigate to this folder in terminal
3. Run: `astro dev init`

Project Structure

This creates the standard Airflow project structure with key files and directories:

- dags/: Directory for pipeline definition files
- docker-compose.yml: Multi-container Docker configuration
- Dockerfile: Instructions for Airflow Docker image

Configure PostgreSQL Database

Docker Compose Configuration

In the root of your project, open the `docker-compose.yml` file and add a service definition for PostgreSQL:

```
services:  postgres_db:    image: postgres:13    container_name:
postgres_db    environment:      - POSTGRES_USER=postgres      -
POSTGRES_PASSWORD=postgres      - POSTGRES_DB=postgres    ports:
- "5432:5432"    volumes:      - postgres_data:/var/lib/postgresql/data
```

Configuration Details

- image: Official PostgreSQL version 13 image
- environment: Default database credentials
- ports: Maps container port to host machine
- volumes: Creates persistent storage for data

Running the Pipeline

Start Airflow

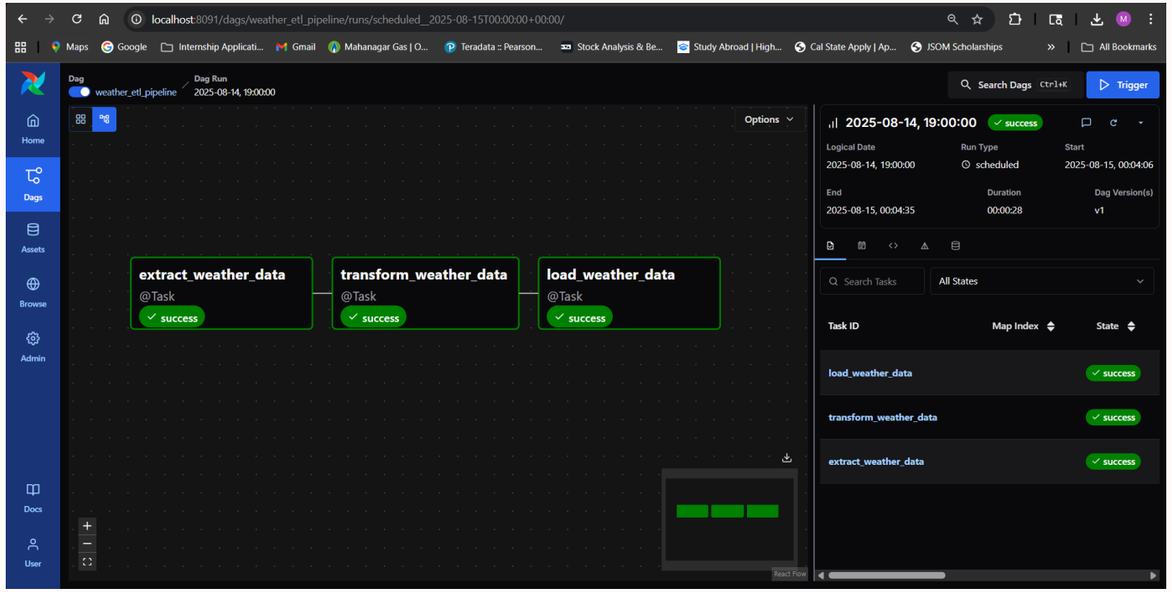
Run `astro dev start` from the project root. Access Airflow UI at http://localhost:8080 with username/password: admin/admin

Configure Connections

Set up PostgreSQL and API connections in Admin → Connections with proper credentials and endpoints

Trigger DAG

Un-pause the DAG and click "Play" to run the pipeline manually. Monitor progress in Grid and Graph views



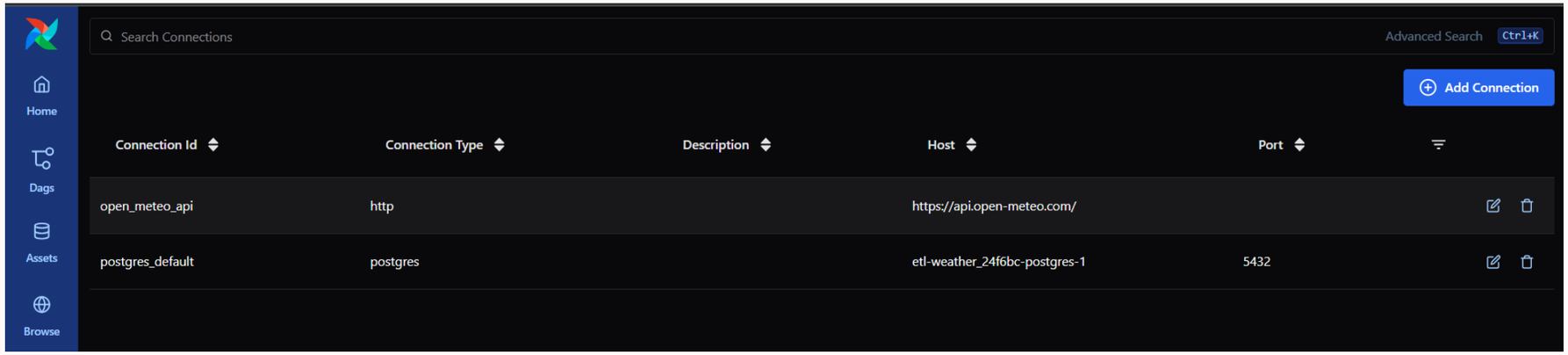
Airflow Connections Setup

PostgreSQL Connection

- Connection Id: postgres_default
- Connection Type: Postgres
- Host: postgres_db
- Schema: postgres
- Login: postgres
- Password: postgres
- Port: 5432

API Connection

- Connection Id: open_meteo_api
- Connection Type: HTTP
- Host: https://api.open-meteo.com



Verifying the Data

Connect to Database

Open DBeaver and create a new PostgreSQL connection:

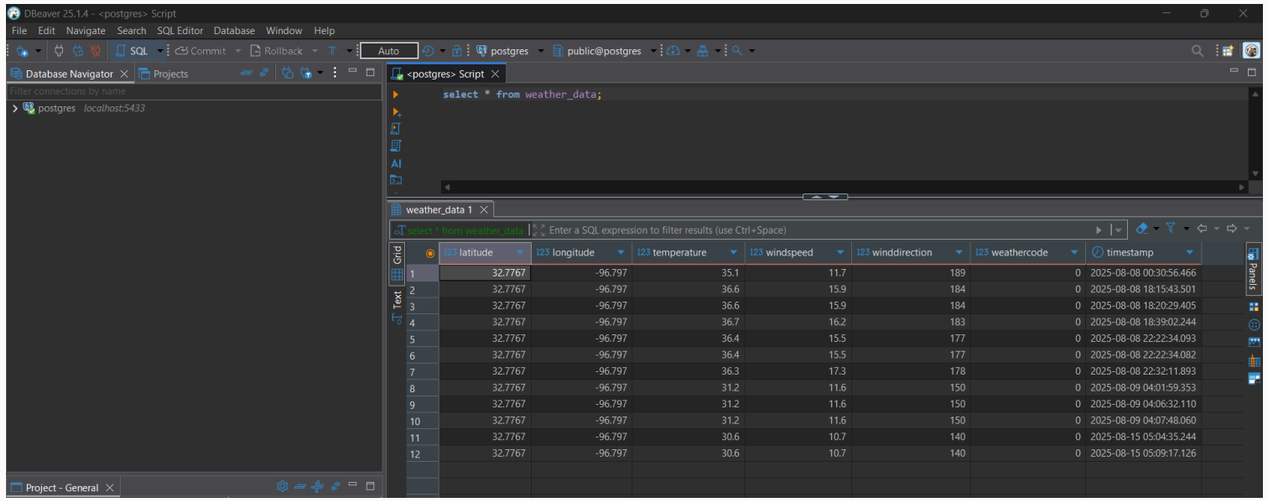
- Host: localhost
- Port: 5432
- Database: postgres
- Username: postgres
- Password: postgres

Query the Data

After connecting, run the following SQL query:

```
SELECT * FROM weather_data;
```

You should see rows of weather data that the pipeline has successfully loaded. Each DAG run adds a new row.



Production Deployment on AWS (Proposed)

Provision AWS RDS Instance

In the AWS console, create a new PostgreSQL instance using Amazon RDS. Note the endpoint URL, master username, and password.

Update Airflow Connection

In the Airflow UI, edit the postgres_default connection:

- Replace host with RDS endpoint URL
- Update login with RDS master username
- Update password with RDS master password

Deploy Airflow

Deploy Airflow to AWS using Amazon MWAA (Managed Workflows for Apache Airflow) or by running it on EC2/EKS, following Astronomer's deployment guides.

Once the connection is updated, the pipeline will load data into the production AWS database on its next scheduled run without any code changes.

Thank You