ETL Weather Data Pipeline: Implementation Guide

Date: August 11, 2025

Author: Meet Zaveri

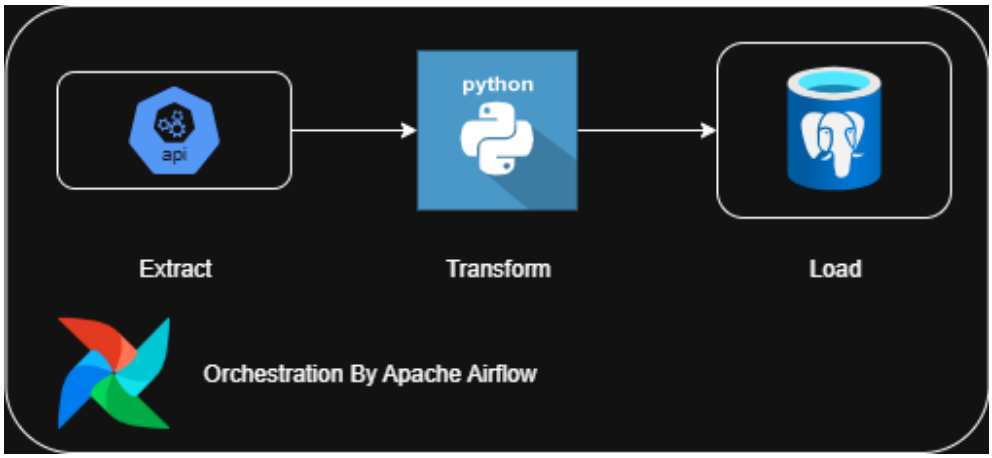
1.0 Executive Summary

This document provides a comprehensive, step-by-step guide for implementing an automated ETL (Extract, Transform, Load) data pipeline. The objective of this pipeline is 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 use in downstream applications.

This solution leverages a modern, open-source technology stack to ensure scalability, reliability, and ease of management:

- **Apache Airflow:** An industry-standard workflow orchestration tool used to schedule, author, and monitor the data pipeline.
- **Astro CLI (by Astronomer):** A command-line interface that simplifies the development and management of Airflow environments.
- **Docker:** A containerization platform used to create consistent, isolated environments for the pipeline's components, including the database.
- **PostgreSQL:** A powerful, open-source object-relational database system used to store the processed weather data.
- **Amazon Web Services (AWS):** The target cloud platform for production deployment, specifically utilizing Amazon RDS for a managed PostgreSQL instance.

By following this guide, clients can replicate this robust data pipeline within their own environments to harness timely weather data for their specific business needs.



2.0 Prerequisites

Before beginning the implementation, ensure the following software and accounts are set up on your development machine:

- **Docker Desktop:** The pipeline and its services run in containers. Download and install it from the official Docker website. Ensure the Docker daemon is running.

- **Visual Studio Code (Recommended):** A versatile code editor with an integrated terminal, which is ideal for this project. Download it from the VS Code website.
- **Astro CLI:** The command-line tool for managing the Airflow project. Installation instructions are in **Section 3.1**.
- **DBeaver (Recommended):** A free, multi-platform database tool for connecting to and verifying the data in the PostgreSQL database. Download it from the DBeaver website.
- **AWS Account (For Deployment)**

3.0 Local Environment Setup

This section covers the initial setup of the project on a local machine for development and testing.

3.1 Install the Astro CLI

The Astro CLI is the primary tool for managing the project. Open a terminal or command prompt and run the appropriate command for your operating system.

- **macOS & Linux:**
  /bin/bash -c "$(curl -sSL https://install.astronomer.io)"
- **Windows (run in PowerShell as Administrator):**
  Invoke-WebRequest -Uri "https://install.astronomer.io" -OutFile "install.ps1"; .\install.ps1

  Note: Docker Desktop for Windows requires WSL 2 or Hyper-V to be enabled.

3.2 Initialize the Airflow Project

1. Create a dedicated folder for your project (e.g., etl-weather-pipeline).
2. Navigate into this folder using your terminal.
3. Run the following command to initialize a new Astro project:
   astro dev init

This command creates the standard Airflow project structure, including the following key files and directories:

- dags/: This directory will hold all your pipeline definition (DAG) files.
- docker-compose.yml: A file to define and run multi-container Docker applications (used here for the database).
- Dockerfile: Instructions to build the custom Docker image for Airflow.
- requirements.txt: A list of Python packages to be installed in the Airflow environment.
- packages.txt: A list of OS-level packages (if needed).

4.0 Building the ETL Pipeline

The core of the pipeline is a Python script that defines the workflow as a Directed Acyclic Graph (DAG) in Airflow.

4.1 Create the DAG File

Inside the dags/ folder, create a new Python file named etl_weather_dag.py. This file will contain the logic for all three ETL stages.

4.2 Define the ETL Tasks in Python

The pipeline consists of three distinct tasks: extracting data from the API, transforming it, and loading it into the database.

```
# dags/etl_weather_dag.py

from airflow.decorators import dag, task
from airflow.providers.postgres.hooks.postgres import PostgresHook
from airflow.providers.http.hooks.http import HttpHook
from airflow.utils.dates import days_ago
import json

# Define constants for the pipeline
LATITUDE_LONDON = 51.5074
LONGITUDE_LONDON = -0.1278
POSTGRES_CONN_ID = "postgres_default"
API_CONN_ID = "open_meteo_api"

@dag(
    dag_id='weather_etl_pipeline',
    default_args={'owner': 'airflow'},
    schedule_interval='@daily',
    start_date=days_ago(1),
    catchup=False,
    tags=['weather', 'etl'],
)
def weather_etl_dag():
    """
    ### Weather ETL Pipeline DAG
    This DAG extracts weather data for a specific location, transforms it,
    and loads it into a PostgreSQL database.
    """
```

```
    @task
    def extract_weather_data():
        """
        #### Extract Task
        Gets weather data from the Open-Meteo API.
        """
        http_hook = HttpHook(method='GET', http_conn_id=API_CONN_ID)
        endpoint =
f"v1/forecast?latitude={LATITUDE_LONDON}&longitude={LONGITUDE_LONDON}&curr
ent_weather=true"
        response = http_hook.run(endpoint)
        return json.loads(response.text)

    @task
    def transform_weather_data(weather_data: dict):
        """
        #### Transform Task
        Transforms the raw API data into a structured format.
        """
        current_weather = weather_data['current_weather']
        transformed_data = {
            'latitude': weather_data['latitude'],
            'longitude': weather_data['longitude'],
            'temperature': current_weather['temperature'],
            'windspeed': current_weather['windspeed'],
            'winddirection': current_weather['winddirection'],
            'weathercode': current_weather['weathercode'],
            'time': current_weather['time']
        }
        return transformed_data

    @task
    def load_weather_data(transformed_data: dict):
        """
        #### Load Task
        Loads the transformed data into the PostgreSQL database.
        """
        pg_hook = PostgresHook(postgres_conn_id=POSTGRES_CONN_ID)
        create_table_sql = """
            CREATE TABLE IF NOT EXISTS weather_data (
```

```
            latitude REAL,
            longitude REAL,
            temperature REAL,
            windspeed REAL,
            winddirection INTEGER,
            weathercode INTEGER,
            time TIMESTAMP
        );
    """
    pg_hook.run(create_table_sql)

    insert_sql = """
        INSERT INTO weather_data (latitude, longitude, temperature, windspeed, winddirection,
weathercode, time)
        VALUES (%s, %s, %s, %s, %s, %s, %s);
    """
    pg_hook.run(insert_sql, parameters=tuple(transformed_data.values()))

# Define the workflow by calling the tasks
raw_data = extract_weather_data()
transformed_data = transform_weather_data(raw_data)
load_weather_data(transformed_data)

# Instantiate the DAG
weather_etl_dag_instance = weather_etl_dag()
```

4.3 Configure the PostgreSQL Database Container

In the root of your project, open the docker-compose.yml file and add a service definition for PostgreSQL. This ensures a database is available for Airflow to connect.

```
# docker-compose.yml

version: '3'
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

volumes:
  postgres_data:
```

- **image**: Specifies the official PostgreSQL version 13 image from Docker Hub.
- **environment**: Sets the default database credentials.
- **ports**: Maps the container's port 5432 to the host machine's port 5432, allowing external tools like DBeaver to connect.
- **volumes**: Creates a persistent volume to ensure data is not lost when the container is stopped or restarted.

5.0 Running and Monitoring the Pipeline

With the DAG and database configured, you can now start the environment and run the pipeline.

5.1 Start the Airflow Environment

In your terminal, from the project's root directory, run:

astro dev start

This command will build the Docker images and start all necessary containers for Airflow (webserver, scheduler, triggerer) and the PostgreSQL database. The first run may take several minutes.

Once complete, the Airflow UI will be accessible at http://localhost:8080.

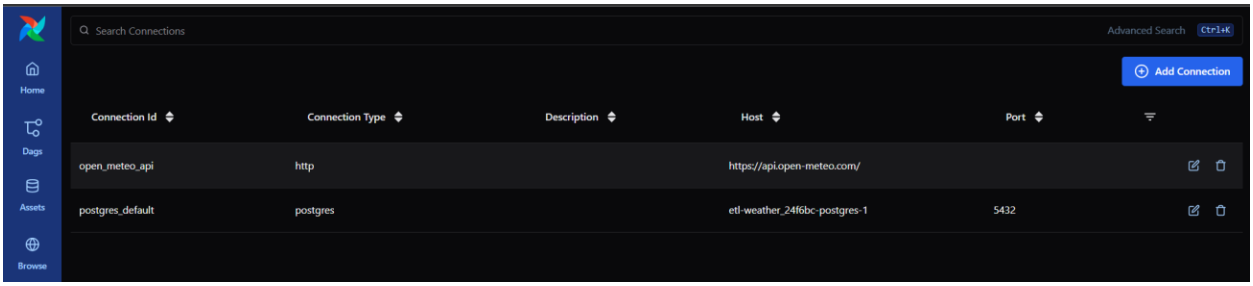
- **Username:** admin
- **Password:** admin

5.2 Configure Airflow Connections

The DAG code references connection IDs (postgres_default and open_meteo_api). These must be configured in the Airflow UI to provide the actual connection details.

1. Navigate to **Admin -> Connections** in the Airflow UI.
2. **Create the PostgreSQL Connection:**
    - Click the + button to add a new connection.
    - **Connection Id:** postgres_default
    - **Connection Type:** Postgres
    - **Host:** postgres_db (This is the service name from docker-compose.yml)
    - **Schema:** postgres
    - **Login:** postgres
    - **Password:** postgres
    - **Port:** 5432
    - Click **Save**.
3. **Create the API Connection:**
    - Click the + button again.
    - **Connection Id:** open_meteo_api

- **Connection Type:** HTTP
- **Host:** https://api.open-meteo.com
- Click **Save**.



5.3 Trigger and Monitor the DAG

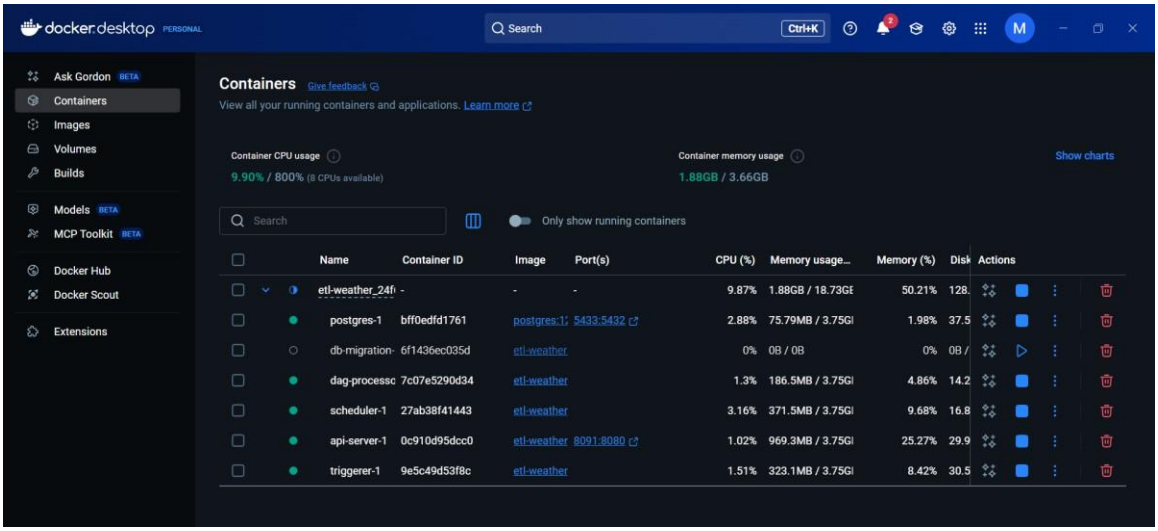
1. On the main **DAGs** page, find weather_etl_pipeline.
2. Un-pause the DAG using the toggle switch on the left.
3. To run the pipeline manually, click the "Play" button (Trigger DAG) on the right.
4. Click on the DAG name to view its progress in the **Grid** and **Graph** views. A successful run will show all three tasks outlined in green.

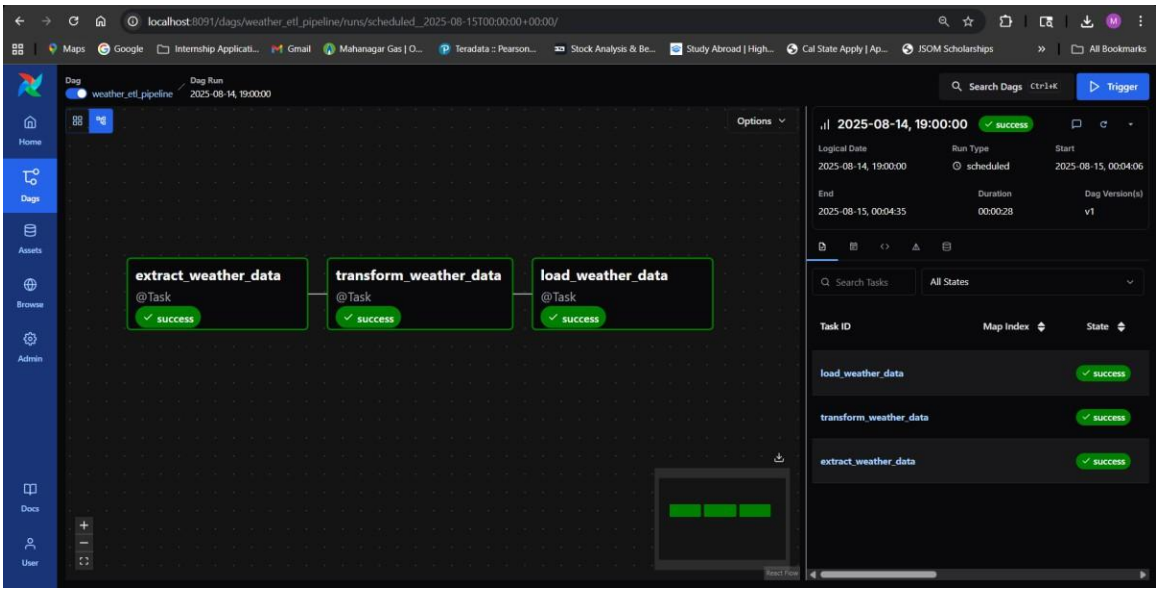
6.0 Verifying the Data

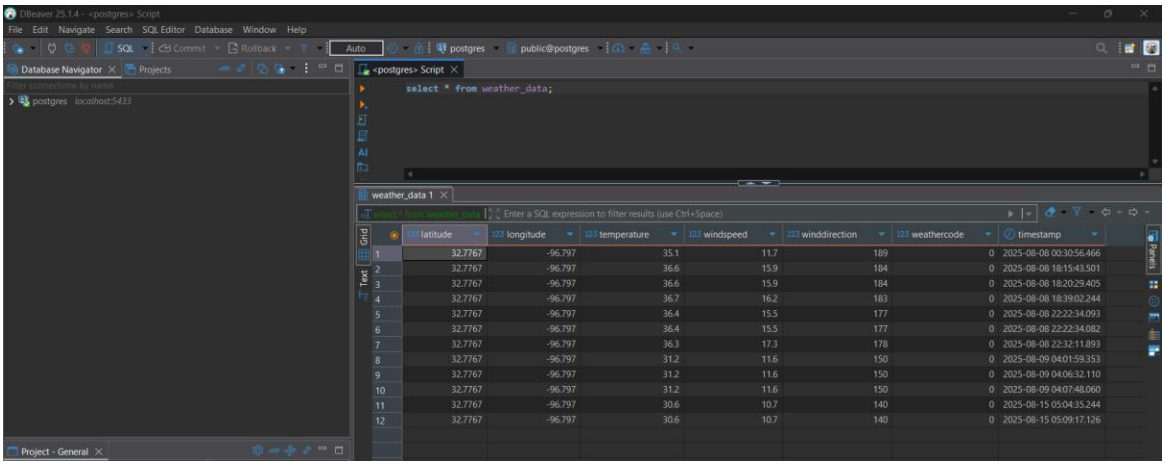
To confirm the pipeline worked correctly, connect to the PostgreSQL database and query the weather_data table.

1. Open DBeaver (or your preferred SQL client).
2. Create a new connection to PostgreSQL.
3. Use the following connection settings:
   - **Host:** localhost
   - **Port:** 5432
   - **Database:** postgres
   - **Username:** postgres
   - **Password:** postgres
4. After connecting, open a new SQL editor and run the following query:
   SELECT * FROM weather_data;

You should see the rows of weather data that the pipeline has successfully loaded. Each time the DAG runs, a new row will be added.


docker desktop PERSONAL
Search Ctrl+K
Ask Gordon BETA
Containers
Images
Volumes
Builds
Models BETA
MCP Toolkit BETA
Docker Hub
Docker Scout
Extensions
Containers
Give feedback
View all your running containers and applications. Learn more
Container CPU usage
9.90% / 800% (8 CPUs available)
Container memory usage
1.88GB / 3.66GB
Show charts
Search
Only show running containers
Name Container ID Image Port(s) CPU (%) Memory usage... Memory (%) Disk Actions
etl-weather_24f -
postgres-1 bff0edfd1761 postgres:1 5433:5432 2.88% 75.79MB / 3.75GI 1.98% 37.5
db-migration- 6f1436ec035d etl-weather 0% 0B / 0B 0% 0B /
dag-processc 7c07e5290d34 etl-weather 1.3% 186.5MB / 3.75GI 4.86% 14.2
scheduler-1 27ab38f41443 etl-weather 3.16% 371.5MB / 3.75GI 9.68% 16.8
api-server-1 0c910d95dcc0 etl-weather 8091:8080 1.02% 969.3MB / 3.75GI 25.27% 29.9
triggerer-1 9e5c49d53f8c etl-weather 1.51% 323.1MB / 3.75GI 8.42% 30.5


localhost:8091/dags/weather_etl_pipeline/runs/scheduled__2025-08-15T00:00:00+00:00/
Dag weather_etl_pipeline
Dag Run 2025-08-14, 19:00:00
Search Dags Ctrl+K
Trigger
Home
Dags
Assets
Browse
Admin
Docs
User
Options
extract_weather_data
@Task
success
transform_weather_data
@Task
success
load_weather_data
@Task
success
2025-08-14, 19:00:00
success
Logical Date
2025-08-14, 19:00:00
Run Type
scheduled
Start
2025-08-15, 00:04:06
End
2025-08-15, 00:04:35
Duration
00:00:28
Dag Version(s)
v1
Search Tasks
All States
Task ID
Map Index
State
load_weather_data
success
transform_weather_data
success
extract_weather_data
success


DBeaver 25.1.4 - <postgres> Script
File Edit Navigate Search SQL Editor Database Window Help
Database Navigator
Projects
postgres localhost:5433
<postgres> Script
select * from weather_data;
weather_data 1
Enter a SQL expression to filter results (use Ctrl+Space)
latitude longitude temperature windspeed winddirection weathercode timestamp
1 32.7767 -96.797 35.1 11.7 189 0 2025-08-08 00:30:56.466
2 32.7767 -96.797 36.6 15.9 184 0 2025-08-08 18:15:43.501
3 32.7767 -96.797 36.6 15.9 184 0 2025-08-08 18:20:29.405
4 32.7767 -96.797 36.7 16.2 183 0 2025-08-08 18:39:02.244
5 32.7767 -96.797 36.4 15.5 177 0 2025-08-08 22:22:34.093
6 32.7767 -96.797 36.4 15.5 177 0 2025-08-08 22:22:34.082
7 32.7767 -96.797 36.3 17.3 178 0 2025-08-08 22:32:11.893
8 32.7767 -96.797 31.2 11.6 150 0 2025-08-09 04:01:59.353
9 32.7767 -96.797 31.2 11.6 150 0 2025-08-09 04:06:32.110
10 32.7767 -96.797 31.2 11.6 150 0 2025-08-09 04:07:48.060
11 32.7767 -96.797 30.6 10.7 140 0 2025-08-15 05:04:35.244
12 32.7767 -96.797 30.6 10.7 140 0 2025-08-15 05:09:17.126
Project - General

7.0 Transitioning to Production (AWS)

To move this pipeline from a local environment to a production deployment on AWS, the primary change involves replacing the local Dockerized database with a managed AWS RDS instance.

1. **Provision an AWS RDS Instance:** In the AWS console, create a new PostgreSQL instance using Amazon RDS. Note the endpoint URL, master username, and password.
2. **Update the Airflow Connection:** In the Airflow UI, navigate back to **Admin -> Connections** and edit the postgres_default connection.
    - **Host:** Replace postgres_db with the endpoint URL of your new AWS RDS instance.
    - **Login:** Update with the master username for the RDS instance.
    - **Password:** Update with the master password for the RDS instance.
3. **Deploy Airflow:** The Airflow instance itself can be deployed to AWS using services like 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.