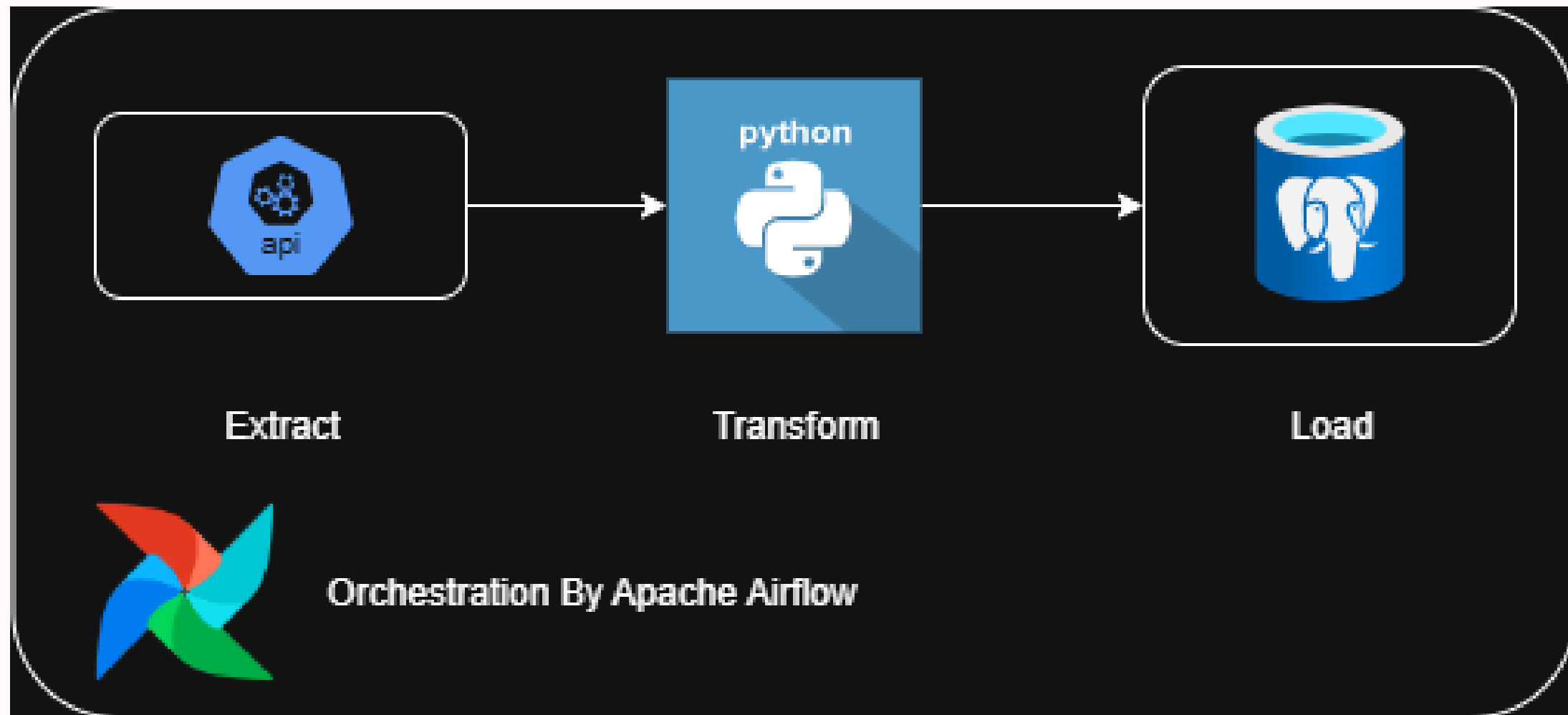


# ETL Weather Data Pipeline: Implementation Guide

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

The screenshot displays the Apache Airflow web interface in a web browser. The URL bar shows `localhost:8081/dags/weather_etl_pipeline/runs/scheduled_2025-08-15T00:00:00+00:00/`. The interface includes a left sidebar with navigation links: Home, Dags, Assets, Browse, Admin, Docs, and User. The main content area shows a DAG named 'weather\_etl\_pipeline' with a run status of 'success' for the date '2025-08-14, 19:00:00'. The DAG graph consists of three tasks: 'extract\_weather\_data', 'transform\_weather\_data', and 'load\_weather\_data', all of which are marked as 'success'. On the right side, there is a 'Trigger' button and a table showing the run details. The table includes columns for Logical Date, Run Type, Start, End, Duration, and Dag Version(s). The run details table shows a successful run on 2025-08-14, 19:00:00, with a duration of 00:00:28 and version v1. Below the DAG graph, there is a 'Task ID' table listing the tasks and their states, all of which are 'success'.

Task ID	Map Index	State
load_weather_data		success
transform_weather_data		success
extract_weather_data		success

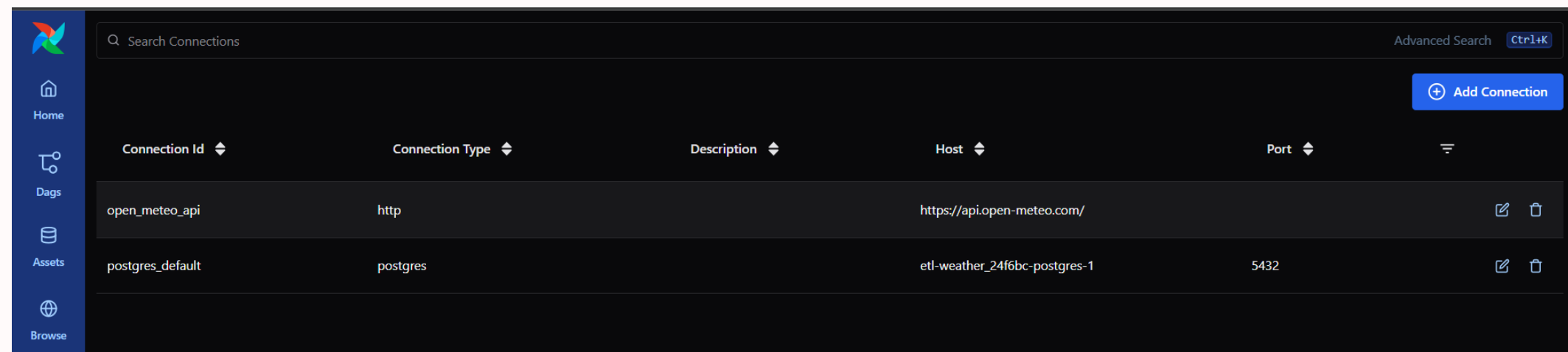
# Airflow Connections Setup

## 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



The screenshot shows the Airflow web interface for managing connections. A sidebar on the left contains navigation links: Home, Dags, Assets, and Browse. The main area features a search bar at the top with the text 'Search Connections' and an 'Advanced Search' button. Below the search bar is a table with columns: Connection Id, Connection Type, Description, Host, Port, and a menu icon. Two connections are listed: 'open\_meteo\_api' with type 'http' and host 'https://api.open-meteo.com/', and 'postgres\_default' with type 'postgres' and host 'etl-weather\_24f6bc-postgres-1' on port '5432'. Each row has edit and delete icons. An 'Add Connection' button is in the top right corner.

Connection Id	Connection Type	Description	Host	Port	
open_meteo_api	http		https://api.open-meteo.com/		 
postgres_default	postgres		etl-weather_24f6bc-postgres-1	5432	 

# Verifying the Data

## Connect to Database

Open DBeaver and create a new PostgreSQL connection:

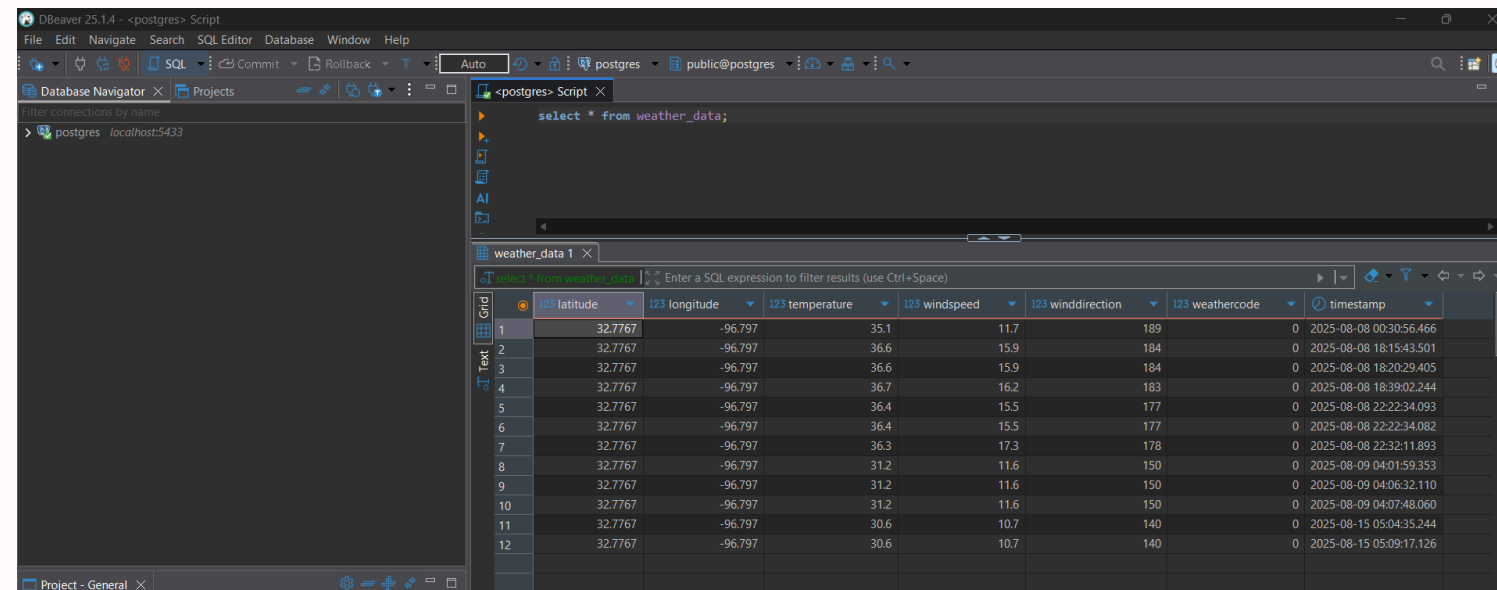
- Host: localhost
- Port: 5432
- Database: postgres
- Username: postgres
- Password: postgres

## 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.



	123 latitude	123 longitude	123 temperature	123 windspeed	123 winddirection	123 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



# 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