

## Introduction:

Welcome to the world of ETL pipelines using Apache Airflow. In this Here, we will focus on pulling weather data using the open meteo API, transforming this data, and then loading it into a Postgres database for easy access and manipulation.

## Apache Airflow:

Apache Airflow is considered an industry standard for data orchestration and pipeline management. It has become popular among data scientists, machine learning engineers, and AI practitioners for its ability to orchestrate complex workflows, manage dependencies between tasks, retry failed tasks, and provide extensive logging.

## Airflow ETL:

Airflow ETL refers to the use of Apache Airflow to manage ETL processes. To review, ETL is a type of data integration that involves extracting data from various sources, transforming it into a format suitable for analysis, and loading it into a final destination such as a data warehouse.

## Configuring our Airflow Development Environment:

We must first configure our development environment before we can start building an ETL pipeline with Airflow. For detailed information about how to configure our development environment.

We will also have to install the Astro CLI. Please check out Astronomer, who maintains the Astro CLI.

## Creating an Airflow Project:

After configuring our environment and installing the Astro CLI, we create an Airflow project. We do this by opening a terminal shell and creating a new directory with the chosen path.

~/Documents/data-engineering/ETL-pipeline/

From the root of this directory, we run the following command in order to create the required resources

```
astro dev init
```

The contents of the directory will look something like below. The exact output might vary.

```
|— dags/
|— include/
|— plugins/
|— tests/
|— airflow_settings.yaml
|— Dockerfile
|— packages.txt
└— requirements.txt
```

To get your project up and running, execute the following command:

```
astro dev start
```

Your Airflow environment will take about a minute to spin up. Once it does, navigate to `localhost : 8080` in your web browser, and you'll be greeted by the Airflow UI. You're now ready to start developing your own ETL pipeline with Airflow

## Designing an ETL pipeline:

we will design a data pipeline to pull weather data from the open meteo API, before transforming and loading this data into a Postgres database. In this case, the source

system is the open meteo API, and the destination is a Postgres database. Let's illustrate this with an image:

API

...



We know from our experience as data engineers that to prepare data to be loaded into a Postgres database, it needs to be transformed from JSON to a tabular format. We create a plan to transform our data after it has been pulled from the open meteo API. We see the three logical steps in this pipeline, which correspond to the E, T, and L of our process.

We can also translate each of these tasks into a directed acyclic graph, or DAG, which is a specific configuration that defines the entire set of tasks to be executed by Airflow, their sequence, and their dependencies on each other. DAG focuses on task-level specifics and may not encompass every bit of information.

## Building an ETL Pipeline with Airflow:

We will organize how we build out ETL pipeline by moving through the steps in order. Taking a structured approach ensures each phase is executed with precision.

### Extracting data with Airflow:

Before pulling data from the open meteo API, we'll need to create an API token by visiting open meteo and selecting the create api key button.

After we have created an API key, we are now ready to start extracting data from the open meteo API with Airflow. We use the tech specs table we created, which includes the details for our DAG configuration.

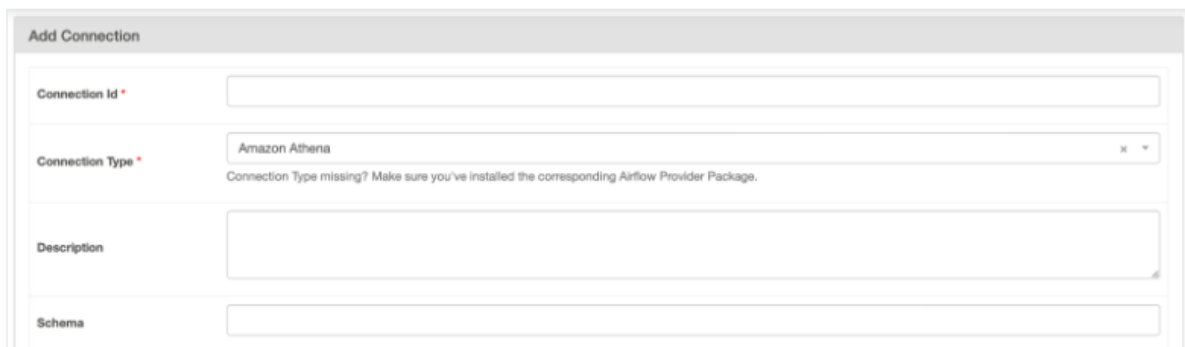
## Transforming data with Airflow:

After data's been extracted from the open meteo API, we're ready to transform it. To do this, we'll create another task using the API. The actual transformation that we'll do is quite straightforward. We're going to flatten the JSON returned by the open meteo API into a list. The catch is that we'll provide unique default values for each key if it does not exist in the response.

## Loading data with Airflow:

The last step of our ETL pipeline We have planned to do this using a Postgres database and one final task defined with the API. we'll define a single parameter when we create our task.

we'll first need to create a connection in the Airflow UI. To open the connections page



The screenshot shows the 'Add Connection' form in the Airflow UI. It has a light gray header with the title 'Add Connection'. Below the header, there are four input fields: 'Connection Id' (a text box), 'Connection Type' (a dropdown menu with 'Amazon Athena' selected), 'Description' (a text box), and 'Schema' (a text box). Below the 'Connection Type' dropdown, there is a small text message: 'Connection Type missing? Make sure you've installed the corresponding Airflow Provider Package.'

we updated our dependencies to allow data returned from the weather data to be passed to the load weather data task. The resulting graph view for our DAG looks something like this:



Graph view for ETL pipeline

## Testing:

Built your first Airflow DAG, it's time to ensure it works. There are a few ways to do this, but one of the most common is by running the DAG end-to-end.

To do this in the Airflow UI, you'll navigate to your DAG and toggle the switch from blue to active. If a task executes successfully, the box associated with it in the UI will turn green. If all tasks in the DAG are successful, the DAG will be marked as success, and the next DAG run will be triggered.