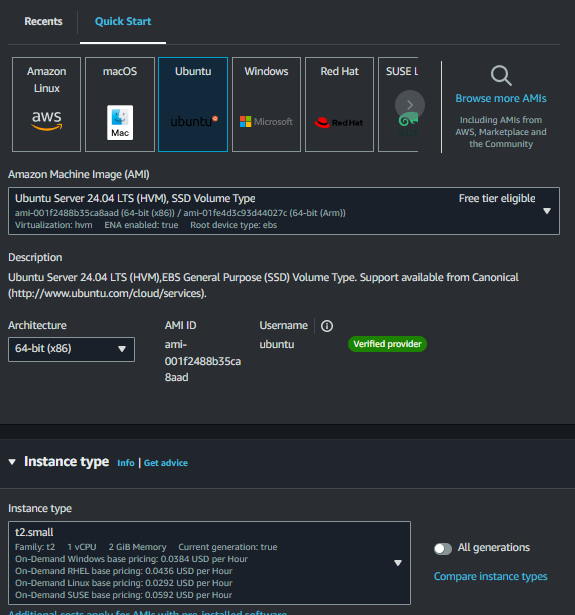
Apache Airflow Project – Extracting Data from Weather API, Transforming it in Python, and Storing it in an AWS S3 Bucket.

**Summary**

This project establishes an end-to-end ETL pipeline leveraging AWS EC2, S3, and Ubuntu, with Apache Airflow for orchestration and Python for data transformations. It orchestrates data collection from the OpenWeather API, transforms the data using Python, and stores the processed results in an S3 bucket.

**Tools:**

* AWS EC2 (Ubuntu)
* AWS S3 for storage
* VSCode for developing Apache Airflow orchestration and Python transformations
* Python
* Apache Airflow

The first step I took was to create an EC2 instance to set up a virtual environment running Ubuntu. This involved launching the instance, assigning it a name, selecting the Ubuntu image, choosing a t2.small instance type, and then proceeding with the launch.

Once launched, I adjusted the security settings to allow IP access. I accessed the EC2 instance's security settings, navigated to the security groups, and added an inbound rule for port 8080, configuring it as a custom TCP connection type.

This configuration enables the connection between the virtual environment and my local machine.

With the instance set up, I was able to open and connect to it, granting access to the command line interface (CLI). Several installation procedures needed to be completed, including:

**Installing Python**

* sudo apt update
* sudo apt install python3-pip
* sudo apt install python3-venv

**Creating a virtual environment**

* python3 -m venv airflow\_venv OR replace airflow\_venv with whatever you want to call it
* source airflow\_venv/bin/activate

**Installing Pandas**

* sudo airflow\_venv/bin/python3 -m pip install pandas

**Installing s3fs**

* sudo airflow\_venv/bin/python3 -m pip install s3fs

**Installing Airflow**

* sudo airflow\_venv/bin/python3 -m pip install apache-airflow

**Accessing Airflow**

* airflow standalone

**When restarting, all that needs to be done after connecting is:**

* source airflow\_venv/bin/activate
* airflow standalone

**Airflow UI Access**

After Airflow has been running for approximately 5 minutes, I accessed the Airflow UI by using the Public IPv4 DNS from the instance. I pasted the DNS into the web browser, appending port 8080 (e.g., ec2-..compute.amazonaws.com:8080). The username is "admin," and the password can be found in the output displayed in the Ubuntu command line where Apache Airflow was started.

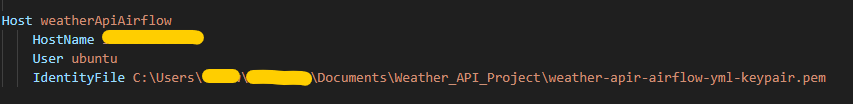
**VSCODE**

I could now proceed with implementing Apache Airflow using VSCode. The installation requirement was to enable Remote – SSH within VSCode. To create a host for managing DAGs, I followed these steps:

1. Selected the blue icon in the bottom left corner to enter the virtual environment.
2. Clicked on "Connect to a Host" at the top of the screen.
3. Added a New SSH Host.
4. Entered the name of the EC2 instance created for the virtual environment.

This action opened a configuration file where I could input the connection details for the virtual environment, including:

* **Host**: The name of the EC2 instance
* **Hostname**: The Public IP
* **User**: In this case, "ubuntu"
* **IdentityFile**: The path to the PEM file for security authentication.

****

**Everytime when starting up, the SSH will need to be reconfigured with the new IP HostName.**

With that all set up, an API connection had to be made in the Apache Airflow UI. This was done by going into the drop down Admin -> Connections.

**A screenshot of a computer

Description automatically generated**

**Data Understanding**

Like most APIs, the data is provided in JSON format, often structured in nested formats. Understanding the design of the JSON structure for the weather data is crucial, as it facilitates easier extraction of the correct columns during the transformation process.

Further details about the data can be found at: [OpenWeatherMap Current API](https://openweathermap.org/current).

With this knowledge, I could create DAGs by importing various libraries for working with API data and DAGs. First, a “dags” folder needed to be created so that Airflow could identify the location of the DAGs, followed by creating a Python file to define the DAGs.

**A screenshot of a computer

Description automatically generated**

Once I had access to the weather\_dags.py file, I could begin implementing the data.

The first DAG I created was an HTTP sensor to confirm the connection to the API. This was followed by a DAG that extracts weather data from the API. I then developed another DAG to convert the raw data into a CSV format by normalizing the JSON structure, although this step was not strictly necessary since I have access to the website mentioned earlier.

Finally, I created a DAG that transforms the data to focus on specific columns of interest for business analysis. This included various transformations such as date-time conversions and converting temperatures from Kelvin to Fahrenheit. Further details on the code can be found in the appendix.

**Additional security for bucket access**

Next, I needed to create a new key in AWS to enable access for data to be pipelined into an AWS S3 bucket. This was done by navigating to the Security Credentials section in AWS, selecting "Create Access Key," following the instructions, and downloading the CSV file containing the credentials.

With the new key, I could set up the AWS CLI in a new terminal in VSCode by executing the following steps:

* pip install awscli – Installation of the AWS CLI.
* aws configure – To enter in the access codes generated in AWS including the region.
* aws sts get-session-token – This provides a key, secret key and a token, which can be put together in a list and then called in the function.

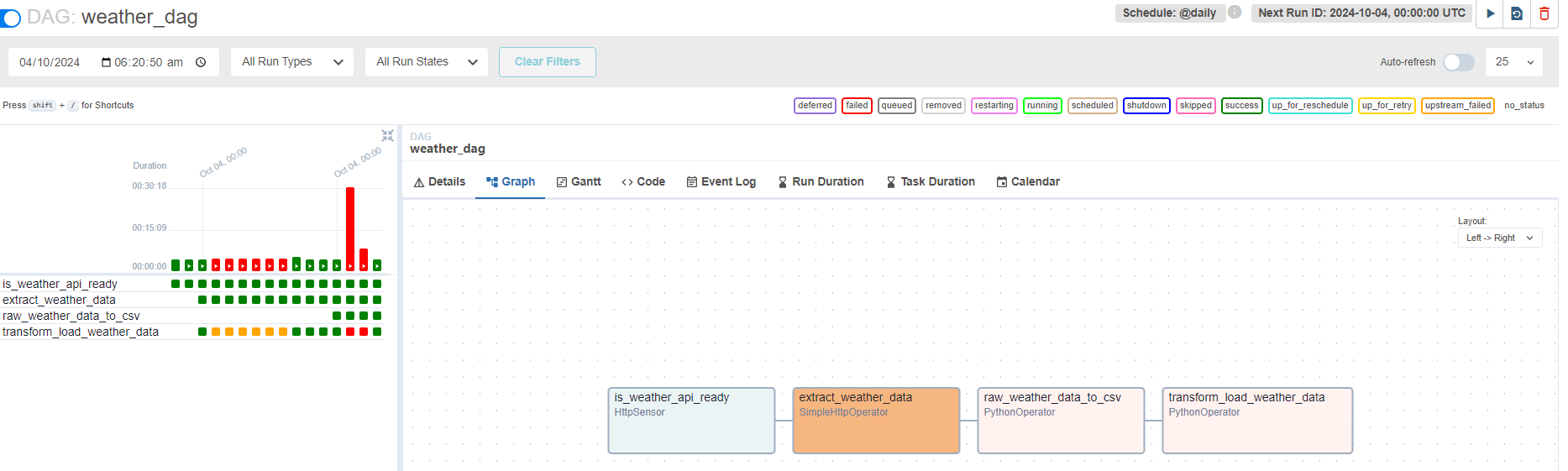
A yellow line on a black background

Description automatically generated

A black screen with yellow and white text

Description automatically generated

After saving the file, the dag could be run in the Airflow UI, and then found in the S3 bucket, making a successful ETL pipeline.



A screenshot of a computer

Description automatically generated

**Limitations of This Project**

This project currently pulls only one row of data at a time. However, with specific objectives and adjustments, it is possible to extract data from multiple sources and then combine it into a dataset efficiently. The Apache Airflow orchestration tool facilitates this process, making it an excellent solution for data collection for analysts and scientists in the field.

**Appendix**

from airflow import DAG

from datetime import timedelta, datetime

from airflow.providers.http.sensors.http import HttpSensor

import json

from airflow.providers.http.operators.http import SimpleHttpOperator

from airflow.operators.python import PythonOperator

import pandas as pd

def kelvin\_to\_fahrenheit(temp\_in\_kelvin):

    temp\_in\_fahrenheit = (temp\_in\_kelvin - 273.15) \* (8/5) + 32

    return temp\_in\_fahrenheit

def save\_raw\_data(task\_instance):

    #Extracted Api data into a pull to store into a variable data

    data = task\_instance.xcom\_pull(task\_ids="extract\_weather\_data")

    # Presave the raw data into a CSV format

    raw\_data\_df = pd.json\_normalize(data)  # Normalize the JSON data

    now = datetime.now()

    dt\_string = now.strftime("%d%m%Y%H%M%S")

    raw\_data\_filename = f"raw\_weather\_data\_{dt\_string}.csv"

    raw\_data\_df.to\_csv(raw\_data\_filename, index=False)

def transform\_load\_data(task\_instance):

    #Extracted Api data into a pull to store into a variable data

    data = task\_instance.xcom\_pull(task\_ids="extract\_weather\_data")

    #Column transformations

    city = data["name"]

    weather\_description = data["weather"][0]['description']

    temp\_farenheit = kelvin\_to\_fahrenheit(data["main"]["temp"])

    feels\_like\_farenheit= kelvin\_to\_fahrenheit(data["main"]["feels\_like"])

    min\_temp\_farenheit = kelvin\_to\_fahrenheit(data["main"]["temp\_min"])

    max\_temp\_farenheit = kelvin\_to\_fahrenheit(data["main"]["temp\_max"])

    pressure = data["main"]["pressure"]

    humidity = data["main"]["humidity"]

    wind\_speed = data["wind"]["speed"]

    time\_of\_record = datetime.utcfromtimestamp(data['dt'] + data['timezone'])

    sunrise\_time = datetime.utcfromtimestamp(data['sys']['sunrise'] + data['timezone'])

    sunset\_time = datetime.utcfromtimestamp(data['sys']['sunset'] + data['timezone'])

    #Creating a JSON format with new column variables / Key : Value

    transformed\_data = {"City": city,

                            "Description": weather\_description,

                            "Temperature (F)": temp\_farenheit,

                            "Feels Like (F)": feels\_like\_farenheit,

                            "Minimun Temp (F)":min\_temp\_farenheit,

                            "Maximum Temp (F)": max\_temp\_farenheit,

                            "Pressure": pressure,

                            "Humidty": humidity,

                            "Wind Speed": wind\_speed,

                            "Time of Record": time\_of\_record,

                            "Sunrise (Local Time)":sunrise\_time,

                            "Sunset (Local Time)": sunset\_time

                            }

    #Converting the JSON format into a list

    transformed\_data\_list = [transformed\_data]

    #Converting the list into a dataframe

    df\_data = pd.DataFrame(transformed\_data\_list)

    #AWS connection credentials

    aws\_credentials = {"key" : "", "secret" : "", "token" : ""}

    now = datetime.now()

    dt\_string = now.strftime("%d%m%Y%H%M%S")

    dt\_string = 'current\_weather\_data\_portland\_' + dt\_string

    df\_data.to\_csv(f"s3://<bucketname>/{dt\_string}.csv", index=False, storage\_options=aws\_credentials)

    #Create a CSV

    ##df\_data.to\_csv(f"{dt\_string}.csv", index=False)

default\_args = {

    'owner': 'airflow',

    'depends\_on\_past': False,

    'start\_date': datetime(2024, 9, 1),

    'email': ['carlwentzel@hotmail.com'],

    'email\_on\_failure': False,

    'email\_on\_retry': False,

    'retries': 2,

    'retry\_delay': timedelta(minutes=2)

}

with DAG('weather\_dag',

        default\_args=default\_args,

        schedule\_interval='@daily',

        catchup=False) as dag:

        is\_weather\_api\_ready = HttpSensor(

        task\_id = 'is\_weather\_api\_ready',

        http\_conn\_id = 'weathermap\_api',

        endpoint = '/data/2.5/weather?q=<location>&APPID=<APIKEY>'

        )

        extract\_weather\_data = SimpleHttpOperator(

        task\_id = 'extract\_weather\_data',

        http\_conn\_id = 'weathermap\_api',

        endpoint = '/data/2.5/weather?q=<location>&APPID=<APIKEY>'

,

        method = 'GET',

        response\_filter = lambda r: json.loads(r.text),

        log\_response=True

        )

        raw\_weather\_data\_to\_csv = PythonOperator(

            task\_id= 'raw\_weather\_data\_to\_csv',

            python\_callable=save\_raw\_data

        )

        transform\_load\_weather\_data = PythonOperator(

            task\_id= 'transform\_load\_weather\_data',

            python\_callable=transform\_load\_data

        )

        is\_weather\_api\_ready >> extract\_weather\_data >> raw\_weather\_data\_to\_csv >> transform\_load\_weather\_data

**Credit:**

Certain things may have been missed in writing up this project as I only focued on key points. For more detail, check out tuplespectra who helped me journey through this interesting project using AWS and Apache Airflow.

**How to build and automate a python ETL pipeline with airflow on AWS EC2 | Data Engineering Project:** [**https://www.youtube.com/watch?v=uhQ54Dgp6To&t=36s**](https://www.youtube.com/watch?v=uhQ54Dgp6To&t=36s)

**How to remotely SSH (connect) Visual Studio Code to AWS EC2:** [**https://www.youtube.com/watch?v=sQQjMnEkGjs&t=1224s**](https://www.youtube.com/watch?v=sQQjMnEkGjs&t=1224s)