

# **Weather ETL Pipeline Documentation**

By: Garret Gallo

Written on: July 15, 2025

## **Section 1: Introduction**

The primary objective of this project was to create a weather ETL pipeline that a weather institution might use during its daily operations. The goal of this pipeline would be its ability to consistently and reliably transport and transform raw data into quality data that could be used for future applications, which may include machine learning and data analytics.

During the construction of this pipeline, I also took into account future scalability and growth. While my current hardware might be a limiting factor, I wanted my pipeline to in theory be able to support an 'indefinite' level of growth. Orchestration was another tool I wanted to incorporate so that my pipeline would be able to run reliably on a consistent basis. This factor is especially important in the context of weather data, as satellites are constantly trying to collect and relay data. Finally, I made sure to consider what data and transformations would be most useful for the previously mentioned applications in the context of weather. This step is crucial to ensure the right data is uploaded and prepared for downstream use.

## Section 2: System Design

While I will describe my thought process in more detail, the image below serves as an overview of the final ETL pipeline that was used to create that final result.

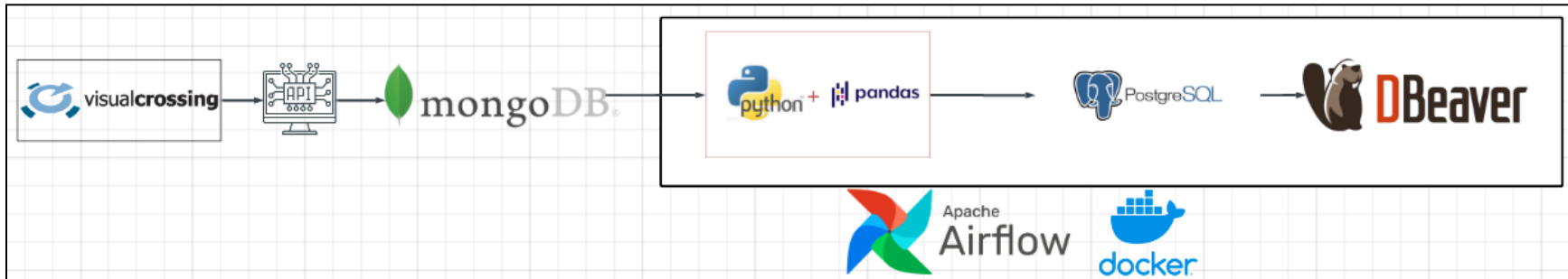


Figure 1: ETL Pipeline

### Section 2.1: Extraction

For the project, one of my primary goals was to find an excellent source to pull data from. Specifically, I wanted more than just simple daily temperatures; rather, I wanted to see if I could pull as many weather conditions as possible (wind, precipitation, UV index, etc.). The reasoning for this is that I wanted to simulate the possible volume that a weather institution would experience on a consistent basis. My final verdict was to go with VisualCrossing, as they had 40 different fields and conditions you could select from, which was exactly what I was looking for. While I had to pay a small price, it was more than worth it for the quality and amount of data I would be able to extract.

Additionally, as seen above in the chart, the initial extraction was done via a simple API call into MongoDB, and there are a few reasons for this. For one, MongoDB acts as a great NoSQL database and data lake, which is perfect for storing raw data. I can initially extract all data into this data lake without having to worry about data inconsistencies, which is exactly what I am looking for. Secondly, I wanted to do this initial extraction outside of my airflow orchestration. While implementing these first few steps in my Apache Airflow orchestration isn't challenging, I wanted to avoid any potential cost down the line while in the testing phase, and future integration can easily be added later.

The final extraction involved pulling from MongoDB into pandas so I could aggregate and transform the data. This was done by simplifying finding my collection in my cluster, then filtering through to pull out all the desired columns and factors that I wanted. Initially, I was just going to be simulating my pipeline daily, but since I am trying to emulate what a weather institution might experience, I figured hourly would be more appropriate.

```
db = client["WWAI"]
collection = db["Test.Weather"]

cursor = collection.find()

records = []
for doc in cursor:
    country = doc.get("resolvedAddress")
    city = doc.get("address")
    for day in doc.get("days", []):
        date_day = day.get("datetime")
        for hour in day.get("hours", []):
            records.append({
                "country": country,
                "city": city,
                "latitude": doc.get('latitude'),
                'longitude': doc.get('longitude'),
                'timezone': doc.get('timezone'),
                'timeZoneOffset': doc.get('tzoffset'),

                "date": date_day,
                "time": hour.get("datetime"),

                "temp": hour.get("temp"),
                'feelslike': hour.get('feelslike'),
                "humidity": hour.get("humidity"),

                "windspeed": hour.get("windspeed"),
                "winddir": hour.get("winddir"),
                'pressure': hour.get('pressure'),
                "windgust": hour.get("windgust"),

                "solarradiation": hour.get('solarradiation'),
                "solarenergy": hour.get('solarenergy'),
                "uvindex": hour.get('uvindex'),
                "severerisk": hour.get('severerisk'),

                "conditions": hour.get("conditions"),

                "precip": hour.get("precip"),
                "preciptype": hour.get("preciptype"),
            })
```

Figure 2: Extracting from mongoDB

## Section 2.2: Transformation

In the transformation stage, I wanted to make sure that my final data and databases would be transformed into actionable insights for processes like machine learning or data analytics, which is the core of data engineering.

My initial approach was to first visualize what I wanted my end result to be, then do reverse engineering to figure out the specifics. After some planning, I came up with a final relational database schema as seen below (which will be explained later in Section 2.3: Load). With this schema, it became significantly easier to know exactly how I needed to transform my data.

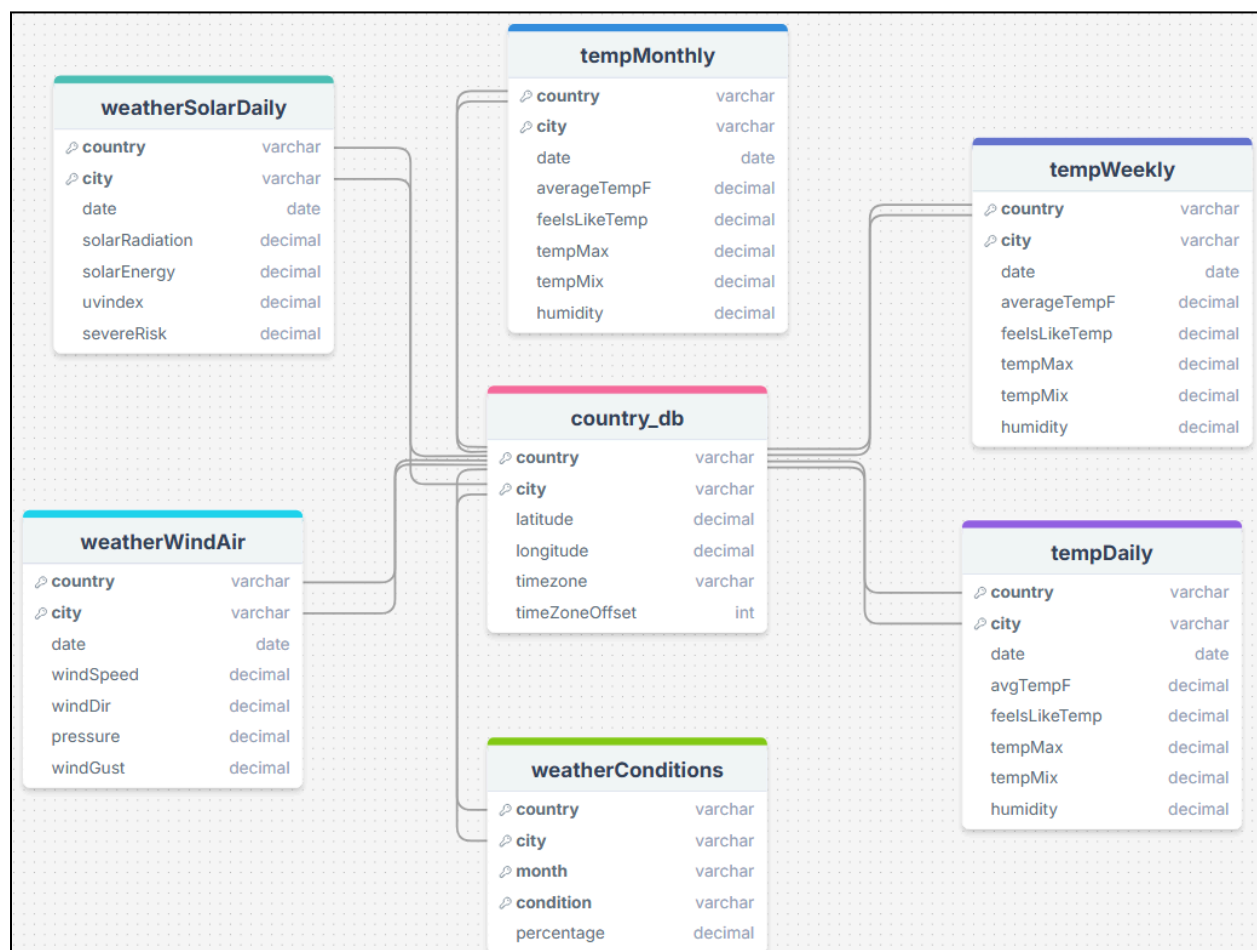


Figure 3: Final Database Schema

Transformations for the temperature tables were relatively straightforward. avgTempF, feelsLikeTemp, and humidity were calculated by finding the average for the time basis I wanted (daily, weekly, and monthly). Similarly, tempMax and tempMin were calculated by finding the maximum and minimum values.

Similar to the temperature tables, the conditions for the solar and wind/air tables were calculated based on a daily average. Finally, the weather conditions table shows how often certain weather conditions (rain, snow, etc) happen on a monthly basis. This was done by one-hot encoding to convert the conditions column to binary columns. It then grouped the data by month so it could calculate the averages. Lastly, using a melt method, I was able to reshape the table for easier visualization.

## Section 2.3: Load

As shown earlier in Figure 3, I broke down the data into multiple tables for better organization and enhanced future analysis.

First, I decided to have a central table with all cities and their respective countries in one table, along with other important information like longitude, latitude, and time zones. With this table, it acts as a great central hub for all other tables to relate to (all tables have a country and city columns that are related to the country and city columns of the countries tables)

Second was the average temperature tables, as it is arguably the most important data point when it comes to analyzing weather. I decided to go with a breakdown of daily, weekly, and monthly, as it paints a great picture of average temperatures per location, especially over long periods of time. Additionally, I decided to include the average feelsLikeTemp, tempMax, tempMix, and humidity as well, as it provided great additional context to the overall temperatures.

Next, I decided that weather conditions were the next most important factors to calculate. For simplicity, I broke down the conditions on a monthly basis and what percentage chance a country had for each condition, which can be seen below in Figure 4. Finally, I decided that solar, wind, and air factors were great additional statistics for more in-depth analysis.





A-Z  country ▼	A-Z  city ▼	A-Z  month ▼	A-Z  condition ▼	123 percentage ▼
Ciudad de México, CDMX, México	Mexico City	January	Clear	32.53
Ciudad de México, CDMX, México	Mexico City	January	Overcast	6.59
Ciudad de México, CDMX, México	Mexico City	January	Partially cloudy	58.2
Ciudad de México, CDMX, México	Mexico City	January	Rain	0
Ciudad de México, CDMX, México	Mexico City	January	Rain, Overcast	1.21
Ciudad de México, CDMX, México	Mexico City	January	Rain, Partially cloudy	1.48
Delhi, DL, India	Delhi	January	Clear	50.13
Delhi, DL, India	Delhi	January	Overcast	16.8
Delhi, DL, India	Delhi	January	Partially cloudy	32.12
Delhi, DL, India	Delhi	January	Rain	0
Delhi, DL, India	Delhi	January	Rain, Overcast	0.13
Delhi, DL, India	Delhi	January	Rain, Partially cloudy	0.81
Karachi, Pakistan	Karachi	January	Clear	83.06
Karachi, Pakistan	Karachi	January	Overcast	0.81
Karachi, Pakistan	Karachi	January	Partially cloudy	16.13
Karachi, Pakistan	Karachi	January	Rain	0
Karachi, Pakistan	Karachi	January	Rain, Overcast	0
Karachi, Pakistan	Karachi	January	Rain, Partially cloudy	0
London, England, United Kingdom	London	January	Clear	18.82
London, England, United Kingdom	London	January	Overcast	46.1
London, England, United Kingdom	London	January	Partially cloudy	23.66
London, England, United Kingdom	London	January	Rain	0.13
London, England, United Kingdom	London	January	Rain, Overcast	9.41
London, England, United Kingdom	London	January	Rain, Partially cloudy	1.08
London, England, United Kingdom	London	January	Snow, Rain, Overcast	0.81
Los Angeles, CA, United States	Los Angeles	January	Clear	71.1
Los Angeles, CA, United States	Los Angeles	January	Overcast	5.91
Los Angeles, CA, United States	Los Angeles	January	Partially cloudy	20.03

Figure 4: Weather Conditions Example

## **Section 2.4: Other Technologies and Tools**

As mentioned previously, I want to implement an orchestration tool that would help run this pipeline on a consistent basis. To achieve this, I elected to go with Apache Airflow and Docker. With Airflow, I was able to better organize how I wanted my pipeline to run and how often, which drastically improved the workflow. Additionally, Airflow is great for troubleshooting exactly where issues are arising in your pipeline and why.

Finally, one small addition I made towards the end was the incorporation of DBeaver, which is a multi-platform database management tool. While minor, it can be a great addition to view and manage all databases in one concise location.

### **Section 3: Future Improvements**

While the goal of this project was successfully completed - there are definitely a handful of improvements that could be made to improve efficiency, durability, and the visualization of the final product.

#### **1. Overcome Scalability Issues**

One of my primary goals with the project was to create a pipeline that could be scalable to include more cities and countries. While I believe my current pipeline in theory should work, hardware and software limitations would prevent that from happening. However, there are tools that could be implemented to remedy this roadblock.

Using a combination of Apache Kafka and Spark would be a key player in scalability, especially with a project of this size. Kafka enables real-time data ingestion by streaming weather data continuously, rather than relying solely on batch loads. Apache Spark would act as the consumer, being able to handle the large amount of aggregations that would be required on a consistent basis. Together, they would provide the foundation for real-time data streaming and transformation.

#### **2. Integration with the Cloud**

While running this pipeline locally works and gets the desired result, even with the integrations of Kafka and Spark, true scalability is impossible without the implementation of a cloud service provider (CSP) such as AWS.

With a CSP, you're no longer constrained by your computer's hardware. You gain access to virtually unlimited compute, memory, and storage — which is exactly where distributed tools like Spark and Kafka thrive. By deploying to the cloud, you can spin up multiple worker nodes and clusters, allowing your data transformations, aggregations, and streaming processes to run in parallel and at scale. This drastically improves overall performance, durability, and reliability, especially as the pipeline increases with scale and volume.