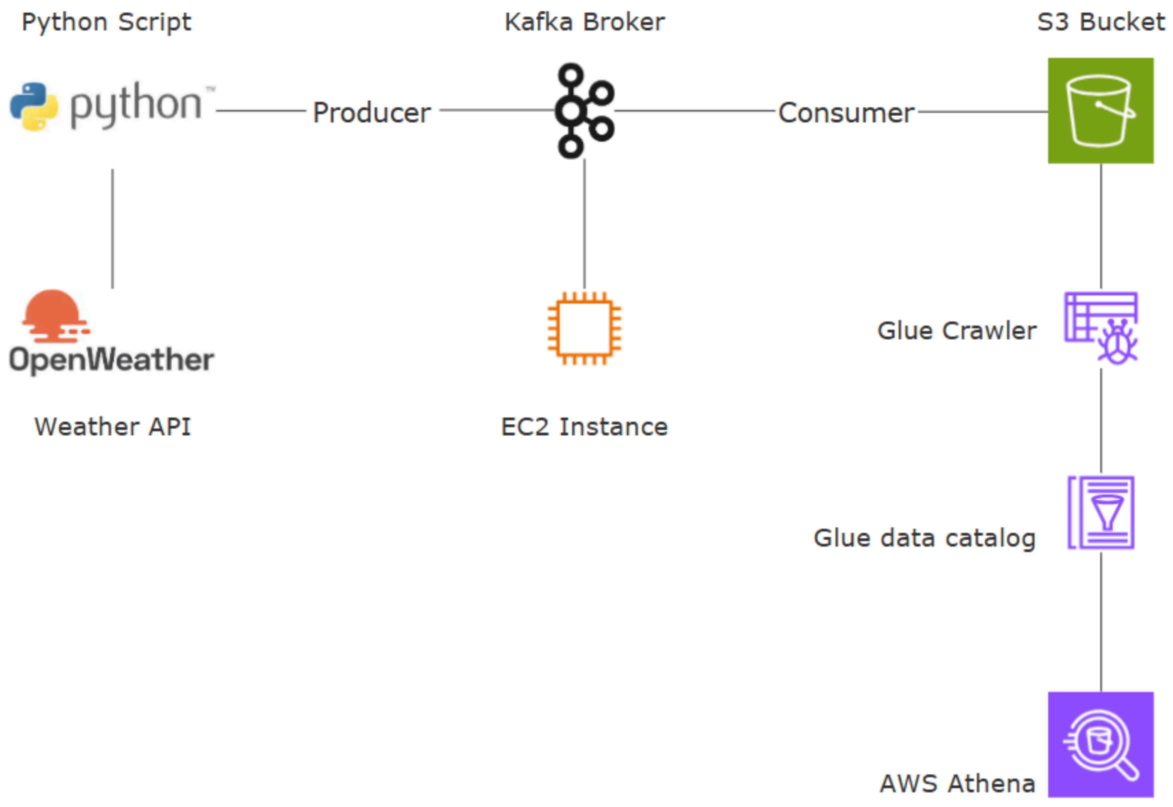
Weather Data Pipeline with Kafka and AWS Glue

- Overview

This project involves creating a data pipeline that ingests weather data from the OpenWeather API, processes it through an Apache Kafka broker, stores the data in an Amazon S3 bucket, and uses AWS Glue and AWS Athena for data analysis.

- Architecture



- Architecture Components

1. **Python Script (Producer):**
   - A Python script fetches weather data from the OpenWeather API. This data is then
   formatted and sent as a stream to the Kafka broker.
   - The script acts as the data producer in this pipeline.

2. **OpenWeather API:**
   - Provides real-time weather data which is fetched by the Python script.
   - The API response includes weather information like temperature, humidity, wind speed, etc.

3. **Apache Kafka (Broker):**
   - Hosted on an EC2 instance.
   - Receives data from the Python script and acts as the intermediary that buffers the data stream, making it available for consumers.

4. **Kafka Consumer:**
   - A Kafka consumer fetches the data from the Kafka broker and pushes it to an Amazon S3 bucket for storage.
   - The consumer could be another Python script or an application configured to read from Kafka and interact with AWS services.

5. **Amazon S3 Bucket:**
   - Acts as the storage layer for the weather data received from the Kafka consumer.
   - Data is stored in a format suitable for further processing, such as CSV or JSON.

6. **AWS Glue Crawler:**
   - Scans the S3 bucket to identify data schemas and populate the AWS Glue Data Catalog with metadata.
   - Automatically updates the schema when new data is added to the S3 bucket.

7. **AWS Glue Data Catalog:**
   - Stores metadata about the weather data, such as schema definitions.
   - Enables seamless integration with other AWS services like Athena for querying the data.

8. **AWS Athena:**
   - Provides an interactive query interface to analyze the weather data stored in the S3 bucket.
   - Uses the schema information stored in the AWS Glue Data Catalog to facilitate SQL-based querying on the data.

- Data Flow

1. **Data Ingestion:**

- The Python script connects to the OpenWeather API, fetches weather data, and sends it to the Kafka broker hosted on an EC2 instance.

2. **Data Streaming:**

- The Kafka broker buffers the incoming weather data, making it available to downstream consumers.

3. **Data Storage:**

- A Kafka consumer listens to the data stream and writes the data into an S3 bucket in the appropriate format.

4. **Data Cataloging:**

- AWS Glue Crawler scans the data stored in the S3 bucket to identify the schema and updates the Glue Data Catalog.

5. **Data Querying:**

- AWS Athena uses the metadata in the Glue Data Catalog to query the weather data for analysis.

- Technologies Used

- **Python**: For scripting data ingestion from OpenWeather API and producing messages to Kafka.
- **OpenWeather API**: Source of weather data.
- **Apache Kafka**: For real-time data streaming, hosted on an EC2 instance.
- **Amazon S3**: Storage service for weather data.
- **AWS Glue**: For creating a data catalog and schema discovery.
- **AWS Athena**: For querying and analyzing the stored weather data.

- Implementation Steps

1. Setting Up Kafka:
- Install and configure Apache Kafka on an EC2 instance.
- Create topics for the weather data stream.

2. Developing the Producer Script:
- Write a Python script to connect to the OpenWeather API.
- Process and send the weather data to the Kafka topic.

3. Configuring the Consumer:
- Implement a Kafka consumer to read data from the topic and write it to an S3 bucket.

4. Setting Up AWS Glue Crawler:
- Create an AWS Glue Crawler to scan the S3 bucket and identify the schema.
- Populate the AWS Glue Data Catalog with the metadata.

5. Using AWS Athena for Data Analysis:
- Configure Athena to use the Glue Data Catalog.
- Write SQL queries to analyze the weather data in S3.

- Challenges & Considerations

- **Data Schema**: Handling dynamic schema changes in weather data over time.

- **Security**: Ensuring data is securely transmitted and stored using encryption and secure access policies.

- **Scalability**: Kafka and S3 can handle large volumes of streaming data, but proper configuration and monitoring are necessary.

- Installation and Setup

1. Create a Python script that interacts with the Weather API, receives data, passes it to the Kafka producer, and sends it to S3.
2. Install required libraries:

   `**python-dotenv**`: To securely store the API key.
   ```
   bash
   pip install python-dotenv

   ```

   `**kafka-python**`: A Kafka client to send data from the local machine to the EC2 instance.
   ```
   bash
   pip install kafka-python

   ```

   `**s3fs**`: Library to interact with AWS SDK for S3 service.
   ```
   bash
   pip install s3fs

   ```

3. AWS CLI: - Ensure you have the AWS CLI installed and credentials set up in the `.aws/credentials` file.

4. EC2 Instance Setup
   Note: The `t2.micro` instance under the free tier is not sufficient for Kafka due to real-time data ingestion. Use at least a `t2.medium` instance.
   After you are done setting up EC2 instance from aws management console you need to setup few files and install few packages to run kafka
   a. Update EC2 instance

      - sudo apt-get update

   b. As Kafka is build on top of Java we need java installed in EC2 instance

      - sudo apt-get install default-jdk

c. We need to download binary version of kafka from the official kafka website otherwise we would encounter errors while setting up the server also this is outdated version go to kafka official website and get link of latest version

   - wget https://downloads.apache.org/kafka/3.3.1/kafka_2.12-3.3.1.tgz

d. Start Zoo-keeper:

   - bin/zookeeper-server-start.sh config/zookeeper.properties

   Before running this command make sure you have navigated into the directory where kafka is located cd {kafka-dir}

e. Start Kafka-Cluster:

   cd kafka_2.12-3.3.1
   bin/kafka-server-start.sh config/server.properties

f. Create the topic:

   cd kafka_2.12-3.3.1
   bin/kafka-topics.sh --create --topic demo_testing2 --bootstrap-server {Put the Public IP of your EC2 Instance:9092} --replication-factor 1 --partitions 1

g. Start Producer:

   bin/kafka-console-producer.sh --topic demo_testing2 --bootstrap-server {Put the Public IP of your EC2 Instance:9092}

h. Start Consumer:

   cd kafka_2.12-3.3.1
   bin/kafka-console-consumer.sh --topic demo_testing2 --bootstrap-server {Put the Public IP of your EC2 Instance:9092}

- TroubleShooting

ubuntu@ip-172-31-27-187:~/kafka_2.12-3.8.0$ **bin/kafka-topics.sh --create --topic demo_test --bootstrap-server 54.234.56.252:9092 --replication-factor 1 --partitions 1**
WARNING: Due to limitations in metric names, topics with a period ('.') or underscore ('_') could collide. To avoid issues it is best to use either, but not both.
Error while executing topic command : Timed out waiting for a node assignment. Call: createTopics [2024-09-16 06:35:46,238] ERROR org.apache.kafka.common.errors.TimeoutException: Timed out waiting for a node assignment. Call: createTopics (org.apache.kafka.tools.TopicCommand)

Solution:

Check Kafka Server Status:

- Ensure that the Kafka server is up and running on the EC2 instance.

Firewall/Security Group Configuration:

- Verify that the EC2 instance's security group is allowing inbound traffic on port `9092` (the Kafka broker port).
- Confirm that outbound rules in the security group allow traffic to the Kafka clients.
- Check the `iptables` or `ufw` (Uncomplicated Firewall) settings on the EC2 instance to ensure that port `9092` is not blocked.
- Run netstat to check whether the Kafka cluster is binding to expected interface or not

  Note: In my case when I ran the netstat command I got something like this

  ubuntu@ip-172-31-27-187:~/kafka_2.12-3.8.0$ netstat -tuln | grep 9092

  ```
  tcp6       0      0 :::9092               :::*                    LISTEN
  ```

  This means that my Kafka broker was listening to all IPv6 addresses at port 9092 but in my security group I hadn't set a rule where the inbound traffic would allow IPv6 address which was the root cause of timeout issue, so basically what you have to do is set up IPv6 rule for all traffic(just for project purpose) and then the issue wouldn't persist.

Kafka Server Configuration:

- Open the Kafka server properties file (`server.properties`) and verify the `listeners` and `advertised.listeners` settings. The `advertised.listeners` should be set to the public IP address of the EC2 instance (`54.234.56.252`):

  (Note: Kafka server properties file is located at {kafka_folder}/config/server.properties)

  ```
  listeners=PLAINTEXT://0.0.0.0:9092
  ```

  ```
  advertised.listeners=PLAINTEXT://<YOUR IP>:9092
  ```

- After making changes, restart the Kafka server:

Zookeeper Status:

- Kafka relies on Zookeeper for managing cluster state. Ensure that the Zookeeper service is running:

Network Issues:

- Check the network connectivity from the machine where the command is being run to the Kafka server. You can use `telnet` to check if the Kafka port is accessible:

  ```
  telnet <YOUR IP> 9092
  ```

- If the connection fails, it indicates a network issue, possibly related to firewall or security group settings.

- Future Enhancements

- **Data Transformation**: Use AWS Glue ETL jobs to clean and transform weather data before analysis.

- **Real-time Analytics**: Integrate with Amazon Kinesis or a similar service for real-time data analysis.

- **Visualisation**: Use AWS QuickSight or Kibana for graphical representation of weather trends.

- Conclusion

This pipeline provides a scalable and efficient way to collect, store, and analyze weather data using a combination of open-source technologies (Kafka) and AWS services (S3, Glue, Athena). The architecture is flexible and can be extended to accommodate additional data sources and analytical requirements.

- Sources

1. AWS Glue Tutorial for Beginners [FULL COURSE in 45 mins]
2. AWS Glue: An ETL Solution with Huge Potential | Medium
3. Weather API - OpenWeatherMap
4. ETL pipeline in Python. In Data world ETL stands for Extract | Medium
5. Thorough Introduction to Apache Kafka™ | HackerNoon