Real time Weather Monitoring pipeline.

{ Interim Project... }

Mini

Vinay kumar kariven

28-01-2024

Table of Contents

1. Introduction

My project entails the creation of a robust real-time weather data pipeline, aimed at efficiently managing and analyzing weather information. Leveraging technologies such as Kafka, PySpark, and MongoDB Atlas.

I have developed a scalable and versatile solution for ingesting, processing, storing, and visualizing real-time weather data. This pipeline addresses the growing need for timely and accurate weather insights across various industries, from agriculture to transportation.

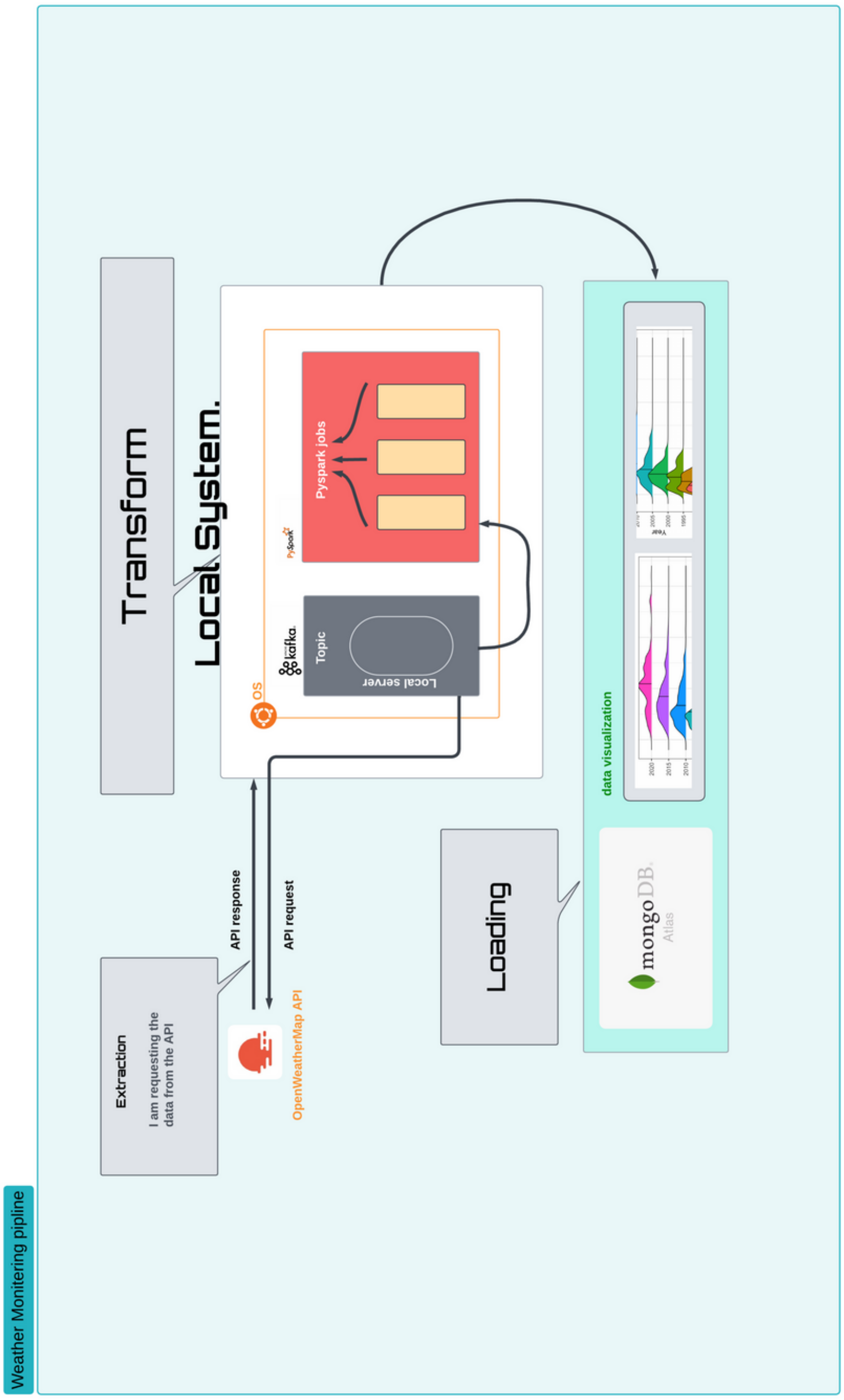
By harnessing the power of modern data processing tools, we aim to empower decision-makers with actionable insights derived from live weather data.

Through this project, I have endeavor to streamline the integration and analysis of weather data, enabling users to make informed decisions based on up-to-date information.

Our approach emphasizes flexibility, scalability, and reliability to meet the evolving demands of real-time data analysis.

2. Architecture Overview

The outline architectural view of this project



In this you can view the three step project implementation namely Extraction , Transformation and Loading.

2.1 Extraction :-

OpenWeather

In the process of extraction i have used the Openweather Api for free weather data in different city's

kafka

After getting the data from the Api. I have pushed the data into the Kafka by creating the kafka topic.

2.2 Transform :-

APACHE Spark

After extraction I have performed the transformation like converting the json data in to parquet file in a given location in real time.

2.3 Loading :-

mongoDB

After Transformation I have saved the real time data in the mongoDb cloud cluster for faulty tolerance.

mongoDB Atlas

After storing the data in mongoDb I have used the atlas for visualizing the real time data in charts.

3. Installation and Setup :-

Step 1:- Kafka Installation

I am using Ubuntu linux distribution OS. so It is preferred to download the Binary downloads of kafka from their website.

Navigate to the directory where you want to download and install Kafka

```
$ cd /opt
```

{   I am using this location   }

```
$ wget https://downloads.apache.org/kafka/2.8.1/kafka_2.13-2.8.1.tgz
```

{ This above command is used to download the kafka binary.tgr file   }

```
~$ tar -xzf kafka/2.8.1/kafka_2.13-2.8.1.tgz
```

{ This will un Zip and un-compress the .tgz file   }

The Source downloads require compilation and provide you with the source code, giving you the flexibility to modify and build the program. Binary downloads, on the other hand, provide pre-compiled executables, which are ready to be run without the need for additional compilation

```
~$ cd /opt/kafka/2.8.1/kafka_2.13-2.8.1
```

{ This will change the current directory or path ! }

```
~$ bin/kafka-server-start.sh --version
```

{ This will show the current kafka version in your local system ! }

Step 2:- spark Installation

To work with spark you need to have java in your local system minimum java-8 is required

```
~$ sudo apt update
```

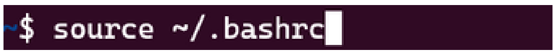
{ This will update all the patches for the OS that your using ! }

```
~$ sudo apt install openjdk-8-jdk
```

{ This will install the java-8 in your system ! }

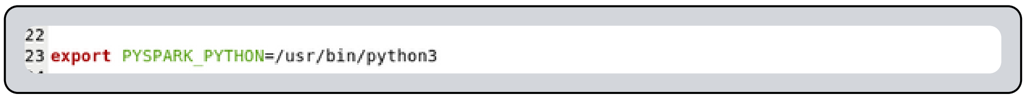
```
~$ export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
```

{ Set the JAVA_HOME environment variable to point to your Java installation. }

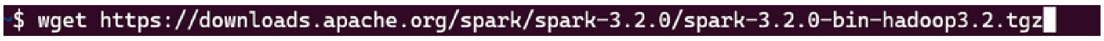


```
$ source ~/.bashrc
```

{ To source the file to apply the changes }

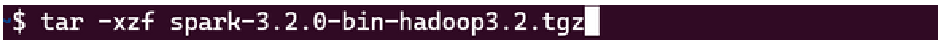


```
export PYSPARK_PYTHON=/usr/bin/python3
```



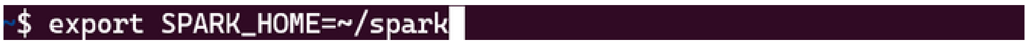
```
$ wget https://downloads.apache.org/spark/spark-3.2.0/spark-3.2.0-bin-hadoop3.2.tgz
```

{ spark file from official website using wget. }



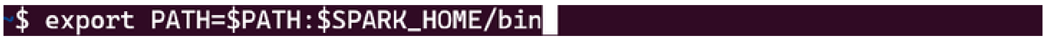
```
$ tar -xzf spark-3.2.0-bin-hadoop3.2.tgz
```

{ This command is used to extract the spark scripts.. }



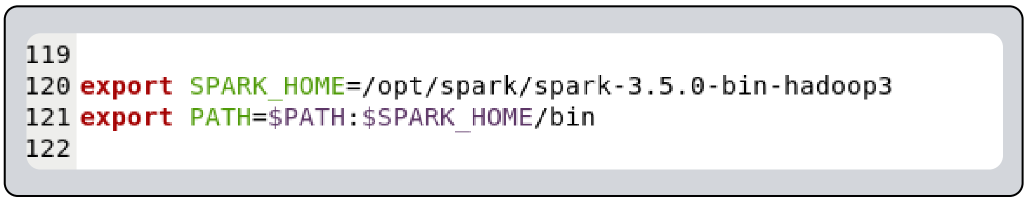
```
$ export SPARK_HOME=~/spark
```

{ setting the spark home environment variable... }

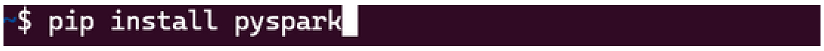


```
$ export PATH=$PATH:$SPARK_HOME/bin
```

{ Add Spark's 'bin' directory to your 'PATH'... }

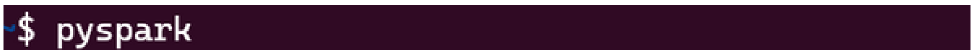


```

export SPARK_HOME=/opt/spark/spark-3.5.0-bin-hadoop3
export PATH=$PATH:$SPARK_HOME/bin

```



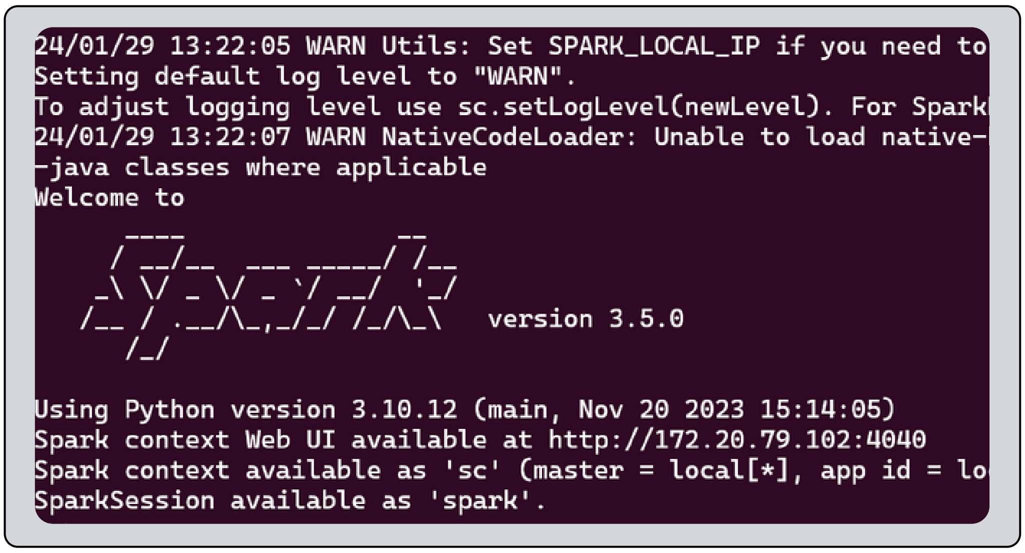
```
$ pip install pyspark
```

{ Installing pyspark... }



```
$ pyspark
```

{ To see the pyspark version in your system... }

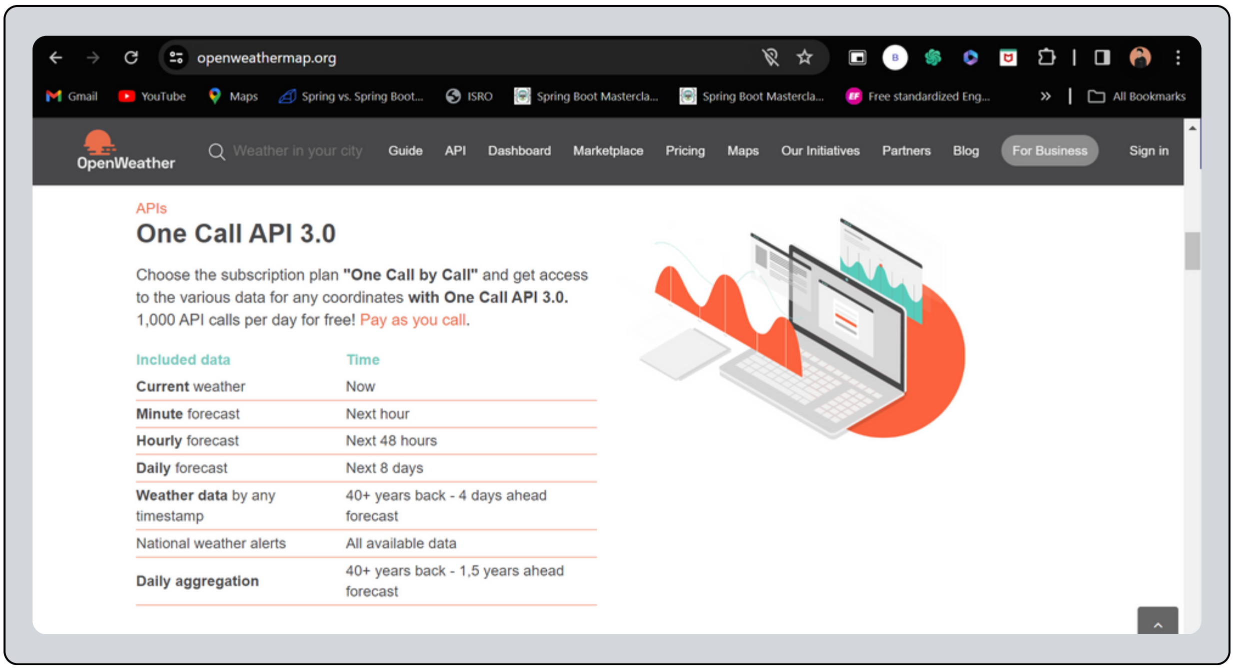


```
24/01/29 13:22:05 WARN Utils: Set SPARK_LOCAL_IP if you need to
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For Spark
24/01/29 13:22:07 WARN NativeCodeLoader: Unable to load native-
-java classes where applicable
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 3.5.0
      /_/

Using Python version 3.10.12 (main, Nov 20 2023 15:14:05)
Spark context Web UI available at http://172.20.79.102:4040
Spark context available as 'sc' (master = local[*], app id = lo
SparkSession available as 'spark'.
```

4. Components :-

4.1 Data Source: OpenWeather Api :-

To work with OpenWeather Api you need to login to their website to access to their api_end point.



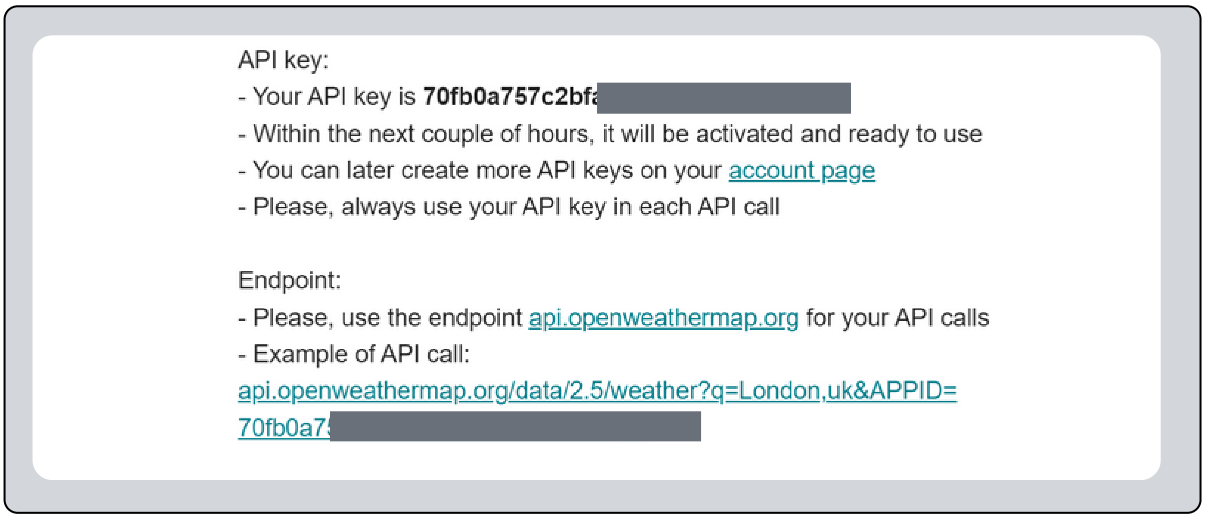
APIs

One Call API 3.0

Choose the subscription plan **"One Call by Call"** and get access to the various data for any coordinates **with One Call API 3.0.** 1,000 API calls per day for free! Pay as you call.

Included data	Time
Current weather	Now
Minute forecast	Next hour
Hourly forecast	Next 48 hours
Daily forecast	Next 8 days
Weather data by any timestamp	40+ years back - 4 days ahead forecast
National weather alerts	All available data
Daily aggregation	40+ years back - 1,5 years ahead forecast

After registering for your Api access you will get your access key and access token to your mail



API key:
- Your API key is **70fb0a757c2bf**[redacted]
- Within the next couple of hours, it will be activated and ready to use
- You can later create more API keys on your account page
- Please, always use your API key in each API call

Endpoint:
- Please, use the endpoint api.openweathermap.org for your API calls
- Example of API call:
api.openweathermap.org/data/2.5/weather?q=London,uk&APPID=
70fb0a7[redacted]

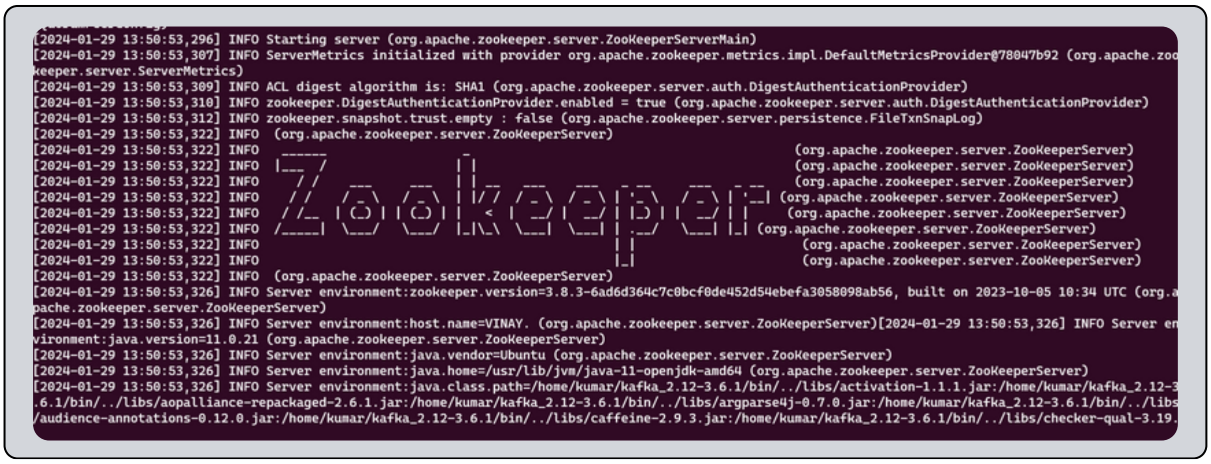
4.2 Data Ingestion: Kafka :-

In order to integrate the kafka with the api end_point first we need to run the zookeeper and kafka server in your local system.

Zookeeper server

```
~/kafka_2.12-3.6.1$ bin/zookeeper-server-start.sh config/zookeeper.properties
```

{ This will the zookeeper in your local system.. }

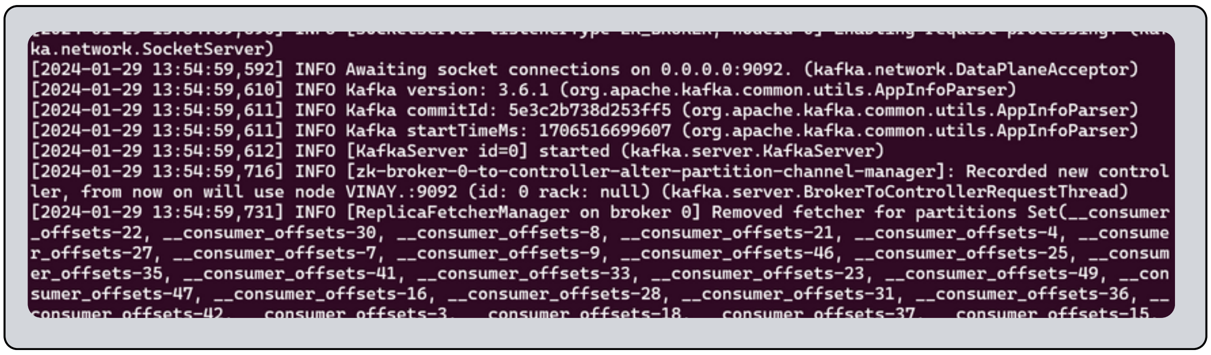


```
[2024-01-29 13:50:53,296] INFO Starting server (org.apache.zookeeper.server.ZooKeeperServerMain)
[2024-01-29 13:50:53,307] INFO ServerMetrics initialized with provider org.apache.zookeeper.metrics.impl.DefaultMetricsProvider@78047b92 (org.apache.zoo
keeper.server.ServerMetrics)
[2024-01-29 13:50:53,309] INFO ACL digest algorithm is: SHA1 (org.apache.zookeeper.server.auth.DigestAuthenticationProvider)
[2024-01-29 13:50:53,310] INFO zookeeper.DigestAuthenticationProvider.enabled = true (org.apache.zookeeper.server.auth.DigestAuthenticationProvider)
[2024-01-29 13:50:53,312] INFO zookeeper.snapshot.trust.empty : false (org.apache.zookeeper.server.persistence.FileTxnSnapLog)
[2024-01-29 13:50:53,322] INFO  (org.apache.zookeeper.server.ZooKeeperServer)
[2024-01-29 13:50:53,322] INFO   ______                  _                                          (org.apache.zookeeper.server.ZooKeeperServer)
[2024-01-29 13:50:53,322] INFO  |___  /                 | |                                         (org.apache.zookeeper.server.ZooKeeperServer)
[2024-01-29 13:50:53,322] INFO     / /    ___     ___   | | __   ___    ___   _ __     ___   _ __   (org.apache.zookeeper.server.ZooKeeperServer)
[2024-01-29 13:50:53,322] INFO    / /    / _ \   / _ \  | |/ /  / _ \  / _ \ | '_ \   / _ \ | '__| (org.apache.zookeeper.server.ZooKeeperServer)
[2024-01-29 13:50:53,322] INFO   / /__  | (_) | | (_) | |   <  |  __/ |  __/ | |_) | |  __/ | |     (org.apache.zookeeper.server.ZooKeeperServer)
[2024-01-29 13:50:53,322] INFO  /_____|  \___/   \___/  |_|\_\  \___|  \___| | .__/   \___| |_| (org.apache.zookeeper.server.ZooKeeperServer)
[2024-01-29 13:50:53,322] INFO                                               | |                     (org.apache.zookeeper.server.ZooKeeperServer)
[2024-01-29 13:50:53,322] INFO                                               |_|                     (org.apache.zookeeper.server.ZooKeeperServer)
[2024-01-29 13:50:53,322] INFO  (org.apache.zookeeper.server.ZooKeeperServer)
[2024-01-29 13:50:53,326] INFO Server environment:zookeeper.version=3.8.3-6ad6d364c7c0bcf0de452d54ebefa3058098ab56, built on 2023-10-05 10:34 UTC (org.a
pache.zookeeper.server.ZooKeeperServer)
[2024-01-29 13:50:53,326] INFO Server environment:host.name=VINAY. (org.apache.zookeeper.server.ZooKeeperServer)[2024-01-29 13:50:53,326] INFO Server en
vironment:java.version=11.0.21 (org.apache.zookeeper.server.ZooKeeperServer)
[2024-01-29 13:50:53,326] INFO Server environment:java.vendor=Ubuntu (org.apache.zookeeper.server.ZooKeeperServer)
[2024-01-29 13:50:53,326] INFO Server environment:java.home=/usr/lib/jvm/java-11-openjdk-amd64 (org.apache.zookeeper.server.ZooKeeperServer)
[2024-01-29 13:50:53,326] INFO Server environment:java.class.path=/home/kumar/kafka_2.12-3.6.1/bin/../libs/activation-1.1.1.jar:/home/kumar/kafka_2.12-3
.6.1/bin/../libs/aopalliance-repackaged-2.6.1.jar:/home/kumar/kafka_2.12-3.6.1/bin/../libs/argparse4j-0.7.0.jar:/home/kumar/kafka_2.12-3.6.1/bin/../libs
/audience-annotations-0.12.0.jar:/home/kumar/kafka_2.12-3.6.1/bin/../libs/caffeine-2.9.3.jar:/home/kumar/kafka_2.12-3.6.1/bin/../libs/checker-qual-3.19.
```

kafka server

```
~/kafka_2.12-3.6.1$ bin/kafka-server-start.sh config/server.properties
```

{ This will start the kafka server in your local system... }

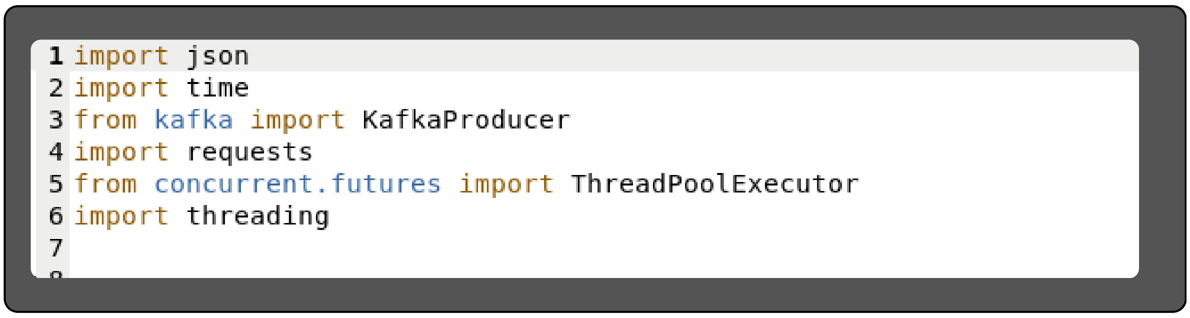


```
ka.network.SocketServer)
[2024-01-29 13:54:59,592] INFO Awaiting socket connections on 0.0.0.0:9092. (kafka.network.DataPlaneAcceptor)
[2024-01-29 13:54:59,610] INFO Kafka version: 3.6.1 (org.apache.kafka.common.utils.AppInfoParser)
[2024-01-29 13:54:59,611] INFO Kafka commitId: 5e3c2b738d253ff5 (org.apache.kafka.common.utils.AppInfoParser)
[2024-01-29 13:54:59,611] INFO Kafka startTimeMs: 1706516699607 (org.apache.kafka.common.utils.AppInfoParser)
[2024-01-29 13:54:59,612] INFO [KafkaServer id=0] started (kafka.server.KafkaServer)
[2024-01-29 13:54:59,716] INFO [zk-broker-0-to-controller-alter-partition-channel-manager]: Recorded new control
ler, from now on will use node VINAY.:9092 (id: 0 rack: null) (kafka.server.BrokerToControllerRequestThread)
[2024-01-29 13:54:59,731] INFO [ReplicaFetcherManager on broker 0] Removed fetcher for partitions Set(__consumer
_offsets-22, __consumer_offsets-30, __consumer_offsets-8, __consumer_offsets-21, __consumer_offsets-4, __consume
r_offsets-27, __consumer_offsets-7, __consumer_offsets-9, __consumer_offsets-46, __consumer_offsets-25, __consum
er_offsets-35, __consumer_offsets-41, __consumer_offsets-33, __consumer_offsets-23, __consumer_offsets-49, __con
sumer_offsets-47, __consumer_offsets-16, __consumer_offsets-28, __consumer_offsets-31, __consumer_offsets-36, __
consumer_offsets-42, __consumer_offsets-3, __consumer_offsets-18, __consumer_offsets-37, __consumer_offsets-15,
```

To integrate the kafka with the api end_poin twe need to write a python script that actively requests the api_end_point for a given interval of time and pushes back the data in to kafka topic.

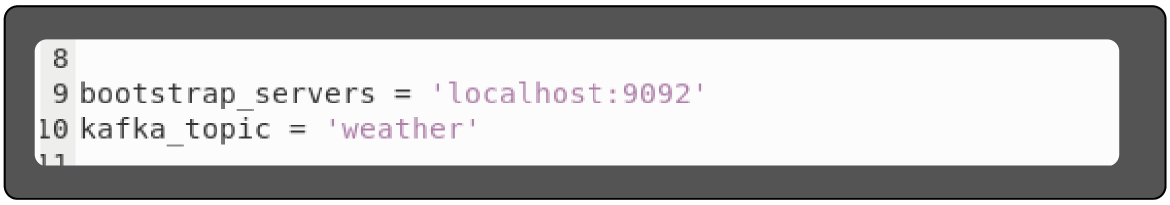
------------------ Internal View of my code {C1} ------------------

Needed libraries and packages.



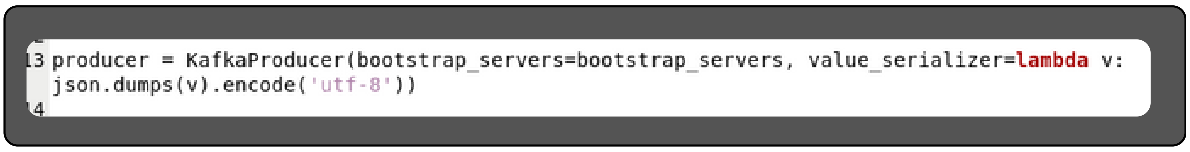
```python
import json
import time
from kafka import KafkaProducer
import requests
from concurrent.futures import ThreadPoolExecutor
import threading

```

Below are the required credentials to connect with kafka topic and the kafka server that is up and running in your local system.



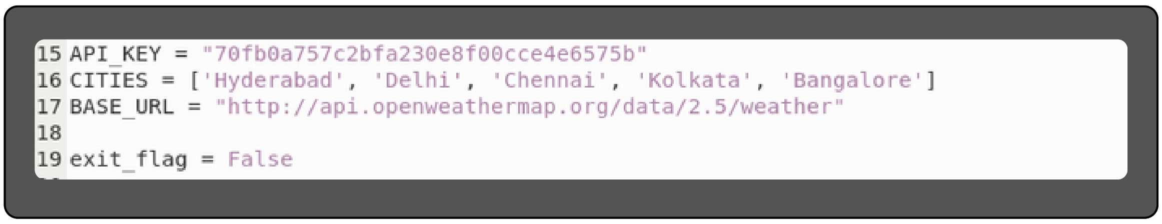
```python

bootstrap_servers = 'localhost:9092'
kafka_topic = 'weather'
```

It is creating a Kafka producer using the KafkaProducer class from the confluent_kafka library. This producer is used to publish messages to a Kafka topic

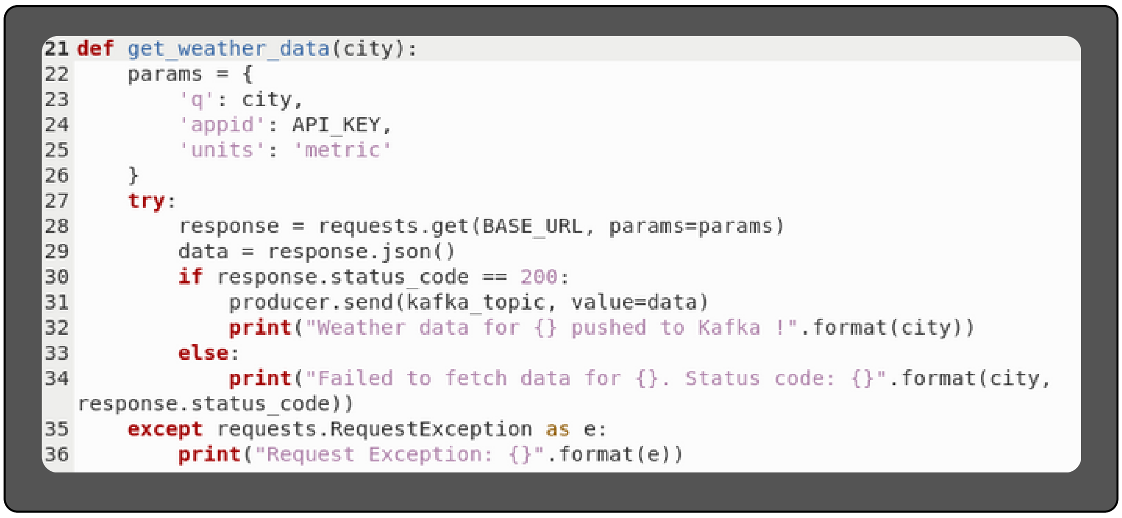


```python
producer = KafkaProducer(bootstrap_servers=bootstrap_servers, value_serializer=lambda v:
json.dumps(v).encode('utf-8'))
```

This are the Api credential's that i need to call the respective Api.

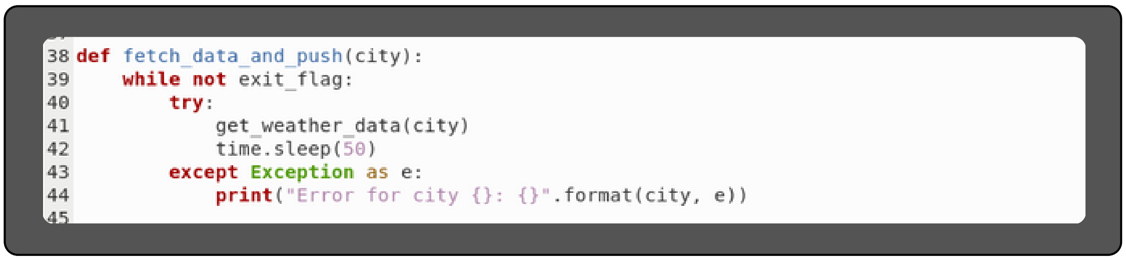


```python
API_KEY = "70fb0a757c2bfa230e8f00cce4e6575b"
CITIES = ['Hyderabad', 'Delhi', 'Chennai', 'Kolkata', 'Bangalore']
BASE_URL = "http://api.openweathermap.org/data/2.5/weather"

exit_flag = False
```



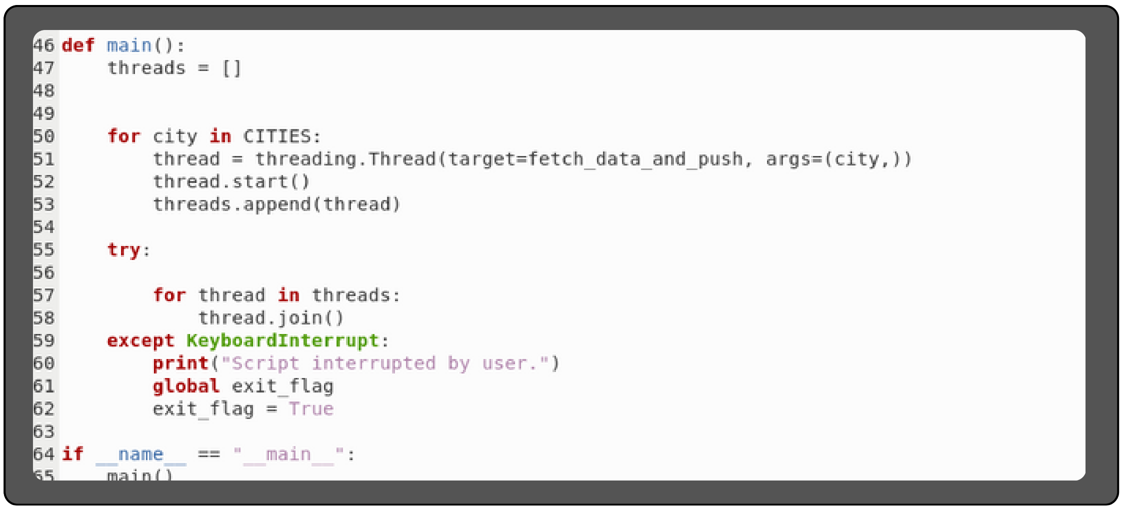
```python
def get_weather_data(city):
    params = {
        'q': city,
        'appid': API_KEY,
        'units': 'metric'
    }
    try:
        response = requests.get(BASE_URL, params=params)
        data = response.json()
        if response.status_code == 200:
            producer.send(kafka_topic, value=data)
            print("Weather data for {} pushed to Kafka !".format(city))
        else:
            print("Failed to fetch data for {}. Status code: {}".format(city,
response.status_code))
    except requests.RequestException as e:
        print("Request Exception: {}".format(e))
```

The above peace of code will call api and get the requested data fromt he api end point.



```python
def fetch_data_and_push(city):
    while not exit_flag:
        try:
            get_weather_data(city)
            time.sleep(50)
        except Exception as e:
            print("Error for city {}: {}".format(city, e))
```

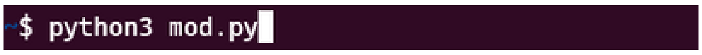
This above code will run a loop to gather data on give list of city's on different thread.



```python
def main():
    threads = []


    for city in CITIES:
        thread = threading.Thread(target=fetch_data_and_push, args=(city,))
        thread.start()
        threads.append(thread)

    try:

        for thread in threads:
            thread.join()
    except KeyboardInterrupt:
        print("Script interrupted by user.")
        global exit_flag
        exit_flag = True

if __name__ == "__main__":
    main()
```

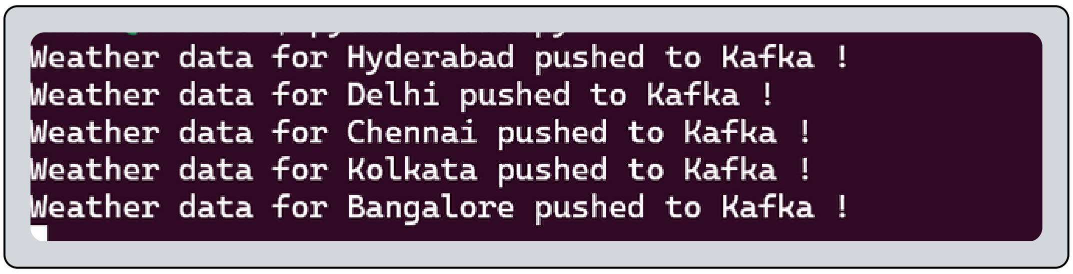
This above code will make sure gathering the each city data is happening in the different thread's and pushing the gathered data directly in to the kafka topic parallel.

To run this code .



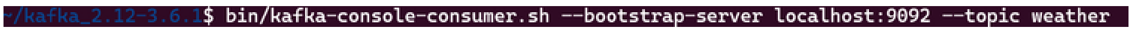
```
~$ python3 mod.py
```

The out put you will see in the terminal is .

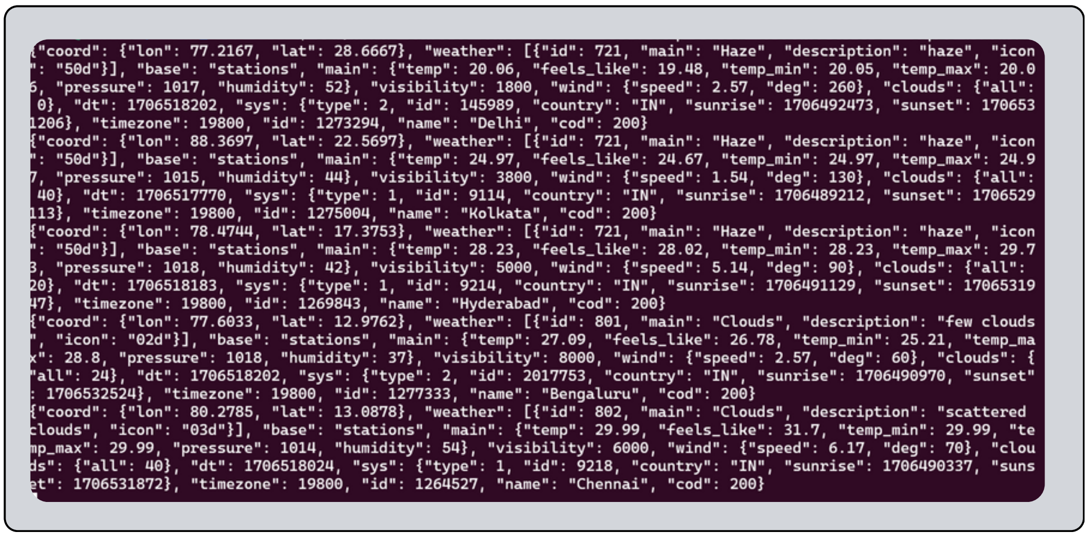


```
Weather data for Hyderabad pushed to Kafka !
Weather data for Delhi pushed to Kafka !
Weather data for Chennai pushed to Kafka !
Weather data for Kolkata pushed to Kafka !
Weather data for Bangalore pushed to Kafka !
```

You can view th edata that is been pushed in the kafka topic using kafka consumer script. in other terminal.



```
~/kafka_2.12-3.6.1$ bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic weather
```

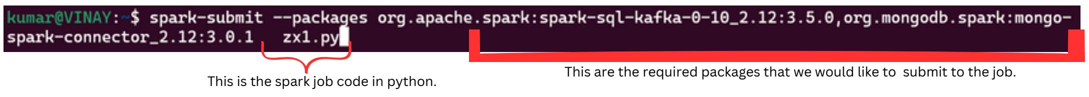


```
{"coord": {"lon": 77.2167, "lat": 28.6667}, "weather": [{"id": 721, "main": "Haze", "description": "haze", "icon
": "50d"}], "base": "stations", "main": {"temp": 20.06, "feels_like": 19.48, "temp_min": 20.05, "temp_max": 20.0
6, "pressure": 1017, "humidity": 52}, "visibility": 1800, "wind": {"speed": 2.57, "deg": 260}, "clouds": {"all":
 0}, "dt": 1706518202, "sys": {"type": 2, "id": 145989, "country": "IN", "sunrise": 1706492473, "sunset": 170653
1206}, "timezone": 19800, "id": 1273294, "name": "Delhi", "cod": 200}
{"coord": {"lon": 88.3697, "lat": 22.5697}, "weather": [{"id": 721, "main": "Haze", "description": "haze", "icon
": "50d"}], "base": "stations", "main": {"temp": 24.97, "feels_like": 24.67, "temp_min": 24.97, "temp_max": 24.9
7, "pressure": 1015, "humidity": 44}, "visibility": 3800, "wind": {"speed": 1.54, "deg": 130}, "clouds": {"all":
 40}, "dt": 1706517770, "sys": {"type": 1, "id": 9114, "country": "IN", "sunrise": 1706489212, "sunset": 1706529
113}, "timezone": 19800, "id": 1275004, "name": "Kolkata", "cod": 200}
{"coord": {"lon": 78.4744, "lat": 17.3753}, "weather": [{"id": 721, "main": "Haze", "description": "haze", "icon
": "50d"}], "base": "stations", "main": {"temp": 28.23, "feels_like": 28.02, "temp_min": 28.23, "temp_max": 29.7
3, "pressure": 1018, "humidity": 42}, "visibility": 5000, "wind": {"speed": 5.14, "deg": 90}, "clouds": {"all":
20}, "dt": 1706518183, "sys": {"type": 1, "id": 9214, "country": "IN", "sunrise": 1706491129, "sunset": 17065319
47}, "timezone": 19800, "id": 1269843, "name": "Hyderabad", "cod": 200}
{"coord": {"lon": 77.6033, "lat": 12.9762}, "weather": [{"id": 801, "main": "Clouds", "description": "few clouds
", "icon": "02d"}], "base": "stations", "main": {"temp": 27.09, "feels_like": 26.78, "temp_min": 25.21, "temp_ma
x": 28.8, "pressure": 1018, "humidity": 37}, "visibility": 8000, "wind": {"speed": 2.57, "deg": 60}, "clouds": {
"all": 24}, "dt": 1706518202, "sys": {"type": 2, "id": 2017753, "country": "IN", "sunrise": 1706490970, "sunset"
: 1706532524}, "timezone": 19800, "id": 1277333, "name": "Bengaluru", "cod": 200}
{"coord": {"lon": 80.2785, "lat": 13.0878}, "weather": [{"id": 802, "main": "Clouds", "description": "scattered
clouds", "icon": "03d"}], "base": "stations", "main": {"temp": 29.99, "feels_like": 31.7, "temp_min": 29.99, "te
mp_max": 29.99, "pressure": 1014, "humidity": 54}, "visibility": 6000, "wind": {"speed": 6.17, "deg": 70}, "clou
ds": {"all": 40}, "dt": 1706518024, "sys": {"type": 1, "id": 9218, "country": "IN", "sunrise": 1706490337, "suns
et": 1706531872}, "timezone": 19800, "id": 1264527, "name": "Chennai", "cod": 200}
```

4.3 Data Processing: PySpark :-

Now we see how to submit a spark job to process the data that we are putting into the kafka topic!

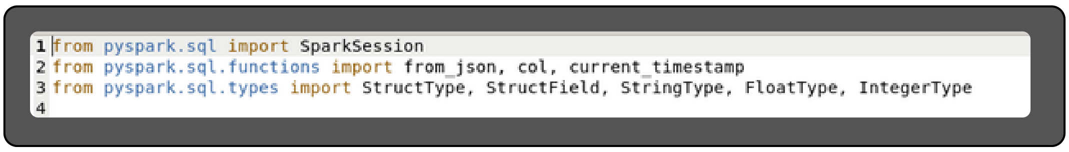
This is how you submit a spark job.



```
kumar@VINAY:~$ spark-submit --packages org.apache.spark:spark-sql-kafka-0-10_2.12:3.5.0,org.mongodb.spark:mongo-
spark-connector_2.12:3.0.1   zx1.py
```

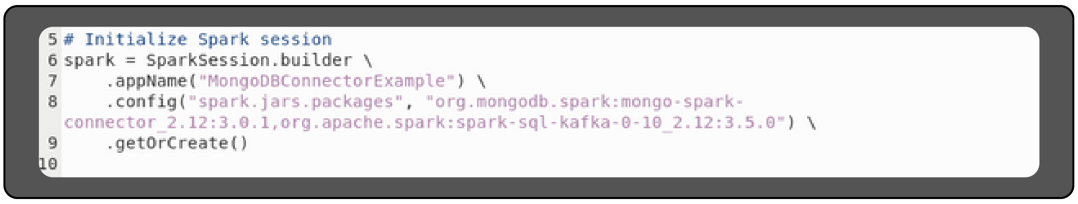
This is the spark job code in python.
This are the required packages that we would like to submit to the job.

----------------- *Internal View of my code {C2}* -----------------



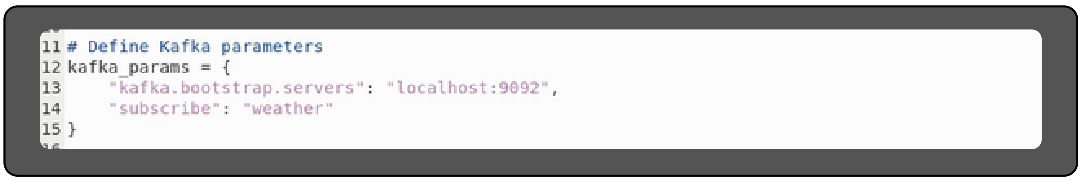
```python
from pyspark.sql import SparkSession
from pyspark.sql.functions import from_json, col, current_timestamp
from pyspark.sql.types import StructType, StructField, StringType, FloatType, IntegerType

```

This above peace of code is the require libraries and packages.



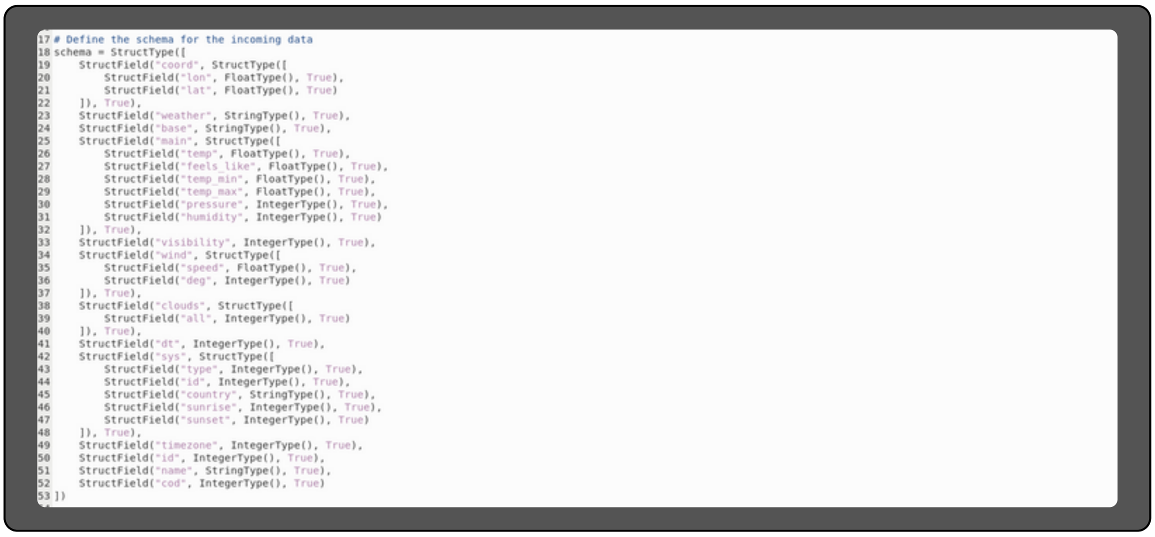
```python
# Initialize Spark session
spark = SparkSession.builder \
    .appName("MongoDBConnectorExample") \
    .config("spark.jars.packages", "org.mongodb.spark:mongo-spark-
connector_2.12:3.0.1,org.apache.spark:spark-sql-kafka-0-10_2.12:3.5.0") \
    .getOrCreate()

```

In this I have created the spark Session.



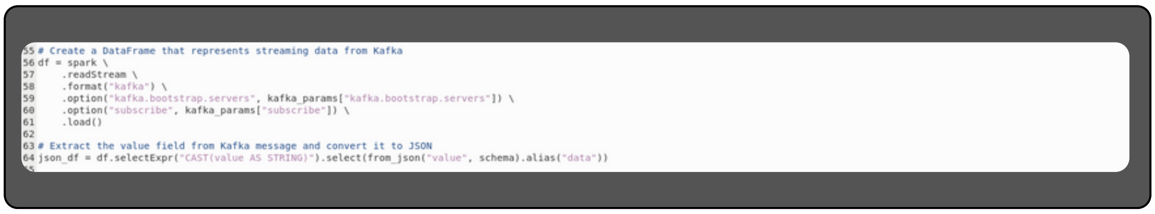
```python
# Define Kafka parameters
kafka_params = {
    "kafka.bootstrap.servers": "localhost:9092",
    "subscribe": "weather"
}
```

This are the kafka credentials to redrive the data in the kafka topic.



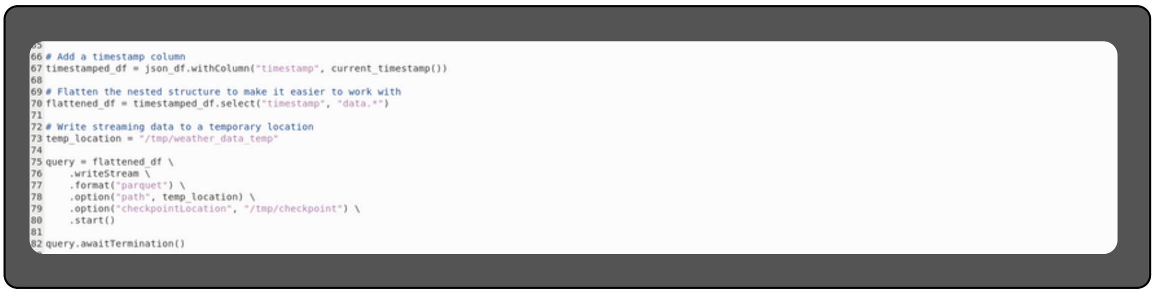
```python
# Define the schema for the incoming data
schema = StructType([
    StructField("coord", StructType([
        StructField("lon", FloatType(), True),
        StructField("lat", FloatType(), True)
    ]), True),
    StructField("weather", StringType(), True),
    StructField("base", StringType(), True),
    StructField("main", StructType([
        StructField("temp", FloatType(), True),
        StructField("feels_like", FloatType(), True),
        StructField("temp_min", FloatType(), True),
        StructField("temp_max", FloatType(), True),
        StructField("pressure", IntegerType(), True),
        StructField("humidity", IntegerType(), True)
    ]), True),
    StructField("visibility", IntegerType(), True),
    StructField("wind", StructType([
        StructField("speed", FloatType(), True),
        StructField("deg", IntegerType(), True)
    ]), True),
    StructField("clouds", StructType([
        StructField("all", IntegerType(), True)
    ]), True),
    StructField("dt", IntegerType(), True),
    StructField("sys", StructType([
        StructField("type", IntegerType(), True),
        StructField("id", IntegerType(), True),
        StructField("country", StringType(), True),
        StructField("sunrise", IntegerType(), True),
        StructField("sunset", IntegerType(), True)
    ]), True),
    StructField("timezone", IntegerType(), True),
    StructField("id", IntegerType(), True),
    StructField("name", StringType(), True),
    StructField("cod", IntegerType(), True)
])
```

In the above code i have defined the schema if the data that i am getting from the kafka topic.



```python
# Create a DataFrame that represents streaming data from Kafka
df = spark \
    .readStream \
    .format("kafka") \
    .option("kafka.bootstrap.servers", kafka_params["kafka.bootstrap.servers"]) \
    .option("subscribe", kafka_params["subscribe"]) \
    .load()

# Extract the value field from Kafka message and convert it to JSON
json_df = df.selectExpr("CAST(value AS STRING)").select(from_json("value", schema).alias("data"))
```

In above code I am creating the dataframe that represent the streaming data from kafka .



```python
# Add a timestamp column
timestamped_df = json_df.withColumn("timestamp", current_timestamp())

# Flatten the nested structure to make it easier to work with
flattened_df = timestamped_df.select("timestamp", "data.*")

# Write streaming data to a temporary location
temp_location = "/tmp/weather_data_temp"

query = flattened_df \
    .writeStream \
    .format("parquet") \
    .option("path", temp_location) \
    .option("checkpointLocation", "/tmp/checkpoint") \
    .start()

query.awaitTermination()
```

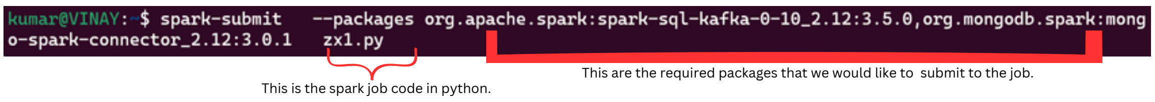
In this i am converting the received data from kafka into json and then flatining for future transformations and saving the incoming data in a file path in parquet file !

4.4 Data Storage: MongoDB Atlas :-

In this I have created a other job that take's the data that is stored in the parquet file in ot mongoDB atlas.

------------------ Internal View of my code {C3} ------------------

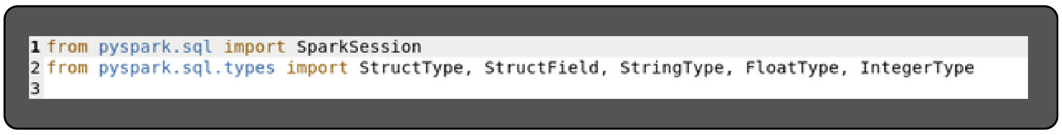
In this code i have used this to take the real time added data in ot he parquet file in to the mongoDB database in cloud .



```
kumar@VINAY:~$ spark-submit   --packages org.apache.spark:spark-sql-kafka-0-10_2.12:3.5.0,org.mongodb.spark:mong
o-spark-connector_2.12:3.0.1   zx1.py
```

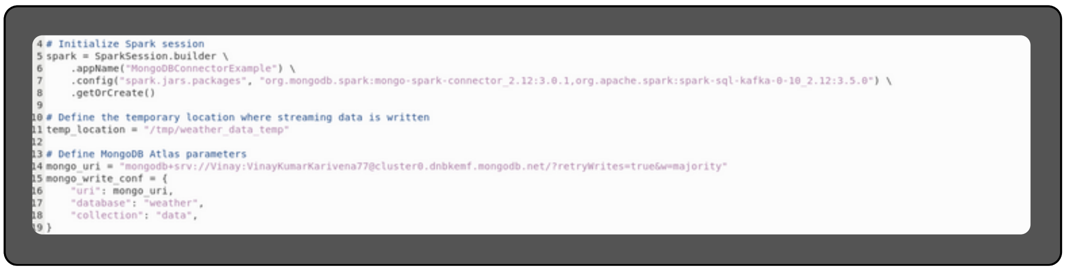
This are the required packages that we would like to submit to the job.
This is the spark job code in python.

This code can be run in the parallel with the job1 or after the job1 it works like in rea time when the new data is added in to the parquet file path.



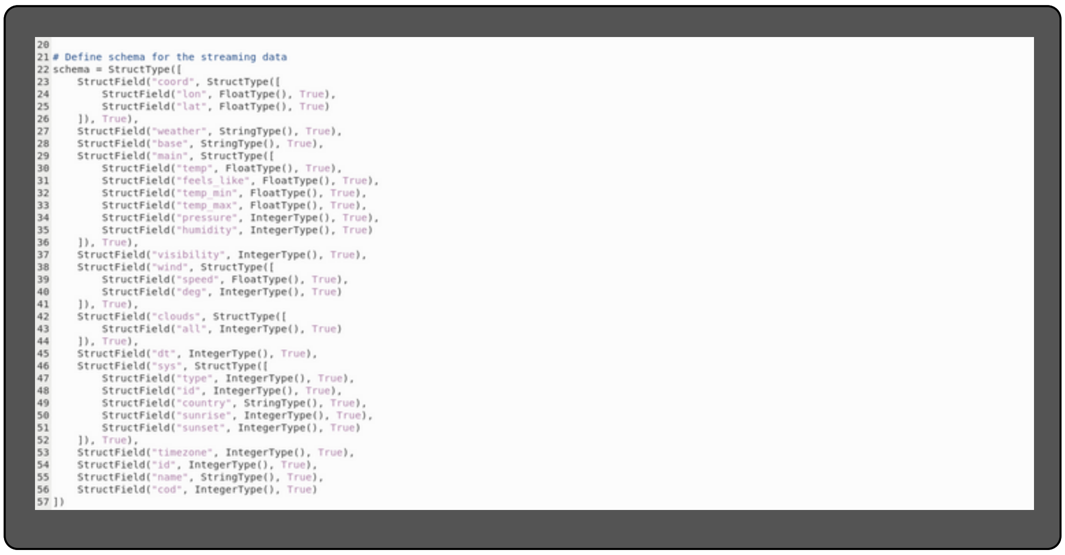
```python
from pyspark.sql import SparkSession
from pyspark.sql.types import StructType, StructField, StringType, FloatType, IntegerType

```

This are the required packages for connecting spark and the mongoDb.



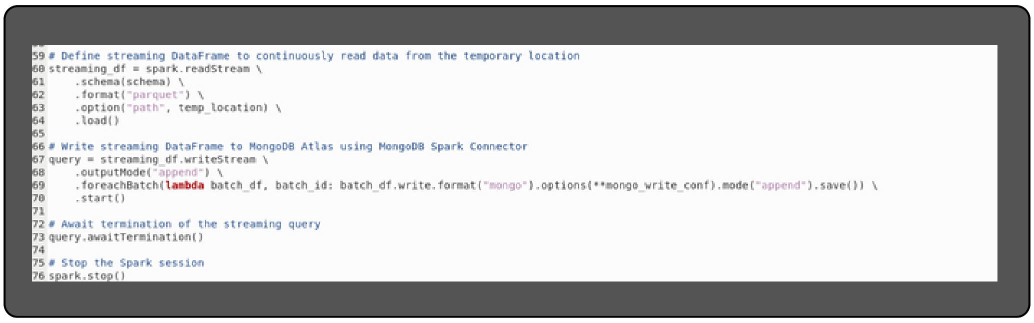
```python
# Initialize Spark session
spark = SparkSession.builder \
    .appName("MongoDBConnectorExample") \
    .config("spark.jars.packages", "org.mongodb.spark:mongo-spark-connector_2.12:3.0.1,org.apache.spark:spark-sql-kafka-0-10_2.12:3.5.0") \
    .getOrCreate()

# Define the temporary location where streaming data is written
temp_location = "/tmp/weather_data_temp"

# Define MongoDB Atlas parameters
mongo_uri = "mongodb+srv://Vinay:VinayKumarKarivena77@cluster0.dnbkemf.mongodb.net/?retryWrites=true&w=majority"
mongo_write_conf = {
    "uri": mongo_uri,
    "database": "weather",
    "collection": "data",
}
```

In this above code i am trying to create a spark session and also configuring the spark mongoDb connecter int he spark



```python

# Define schema for the streaming data
schema = StructType([
    StructField("coord", StructType([
        StructField("lon", FloatType(), True),
        StructField("lat", FloatType(), True)
    ]), True),
    StructField("weather", StringType(), True),
    StructField("base", StringType(), True),
    StructField("main", StructType([
        StructField("temp", FloatType(), True),
        StructField("feels_like", FloatType(), True),
        StructField("temp_min", FloatType(), True),
        StructField("temp_max", FloatType(), True),
        StructField("pressure", IntegerType(), True),
        StructField("humidity", IntegerType(), True)
    ]), True),
    StructField("visibility", IntegerType(), True),
    StructField("wind", StructType([
        StructField("speed", FloatType(), True),
        StructField("deg", IntegerType(), True)
    ]), True),
    StructField("clouds", StructType([
        StructField("all", IntegerType(), True)
    ]), True),
    StructField("dt", IntegerType(), True),
    StructField("sys", StructType([
        StructField("type", IntegerType(), True),
        StructField("id", IntegerType(), True),
        StructField("country", StringType(), True),
        StructField("sunrise", IntegerType(), True),
        StructField("sunset", IntegerType(), True)
    ]), True),
    StructField("timezone", IntegerType(), True),
    StructField("id", IntegerType(), True),
    StructField("name", StringType(), True),
    StructField("cod", IntegerType(), True)
])
```

In this above peace of code I am defining the schema of the data that is coming out of the file path of parquet .



```python
# Define streaming DataFrame to continuously read data from the temporary location
streaming_df = spark.readStream \
    .schema(schema) \
    .format("parquet") \
    .option("path", temp_location) \
    .load()

# Write streaming DataFrame to MongoDB Atlas using MongoDB Spark Connector
query = streaming_df.writeStream \
    .outputMode("append") \
    .foreachBatch(lambda batch_df, batch_id: batch_df.write.format("mongo").options(**mongo_write_conf).mode("append").save()) \
    .start()

# Await termination of the streaming query
query.awaitTermination()

# Stop the Spark session
spark.stop()
```

In this above code i am sending the data in to the monogDB through the give string and in append mode.

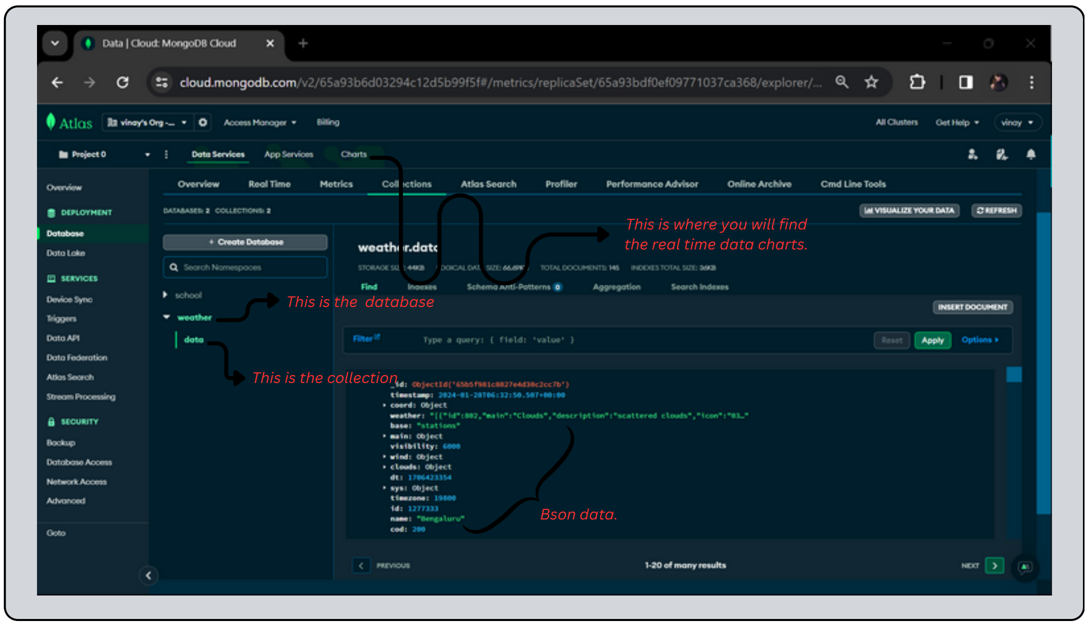
This code will terminate any exception happend or any interruption happens in the middle of the process..

This whole above code is a parallel job that can be runned and it is a real time runnable code that processes the data in the real time.

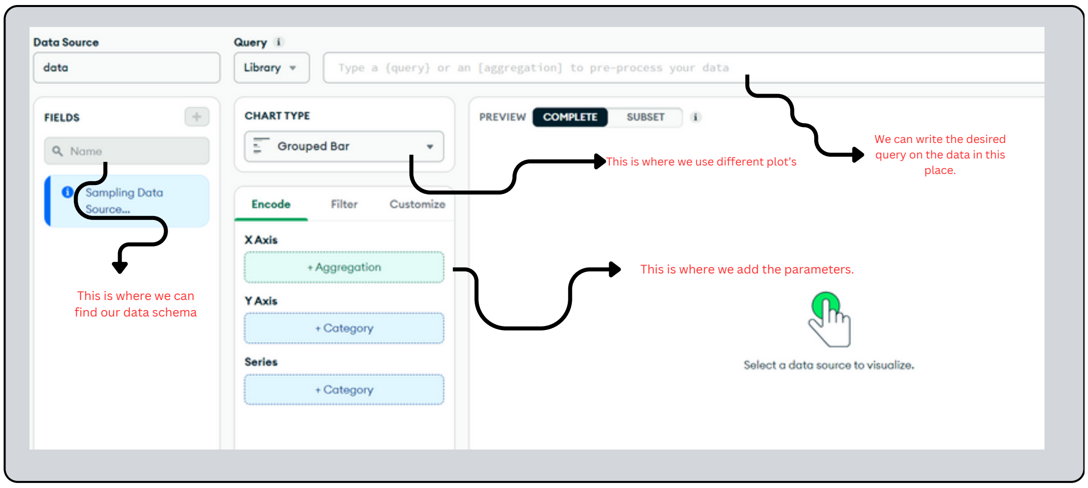
4.5 Integration and Visualization :-

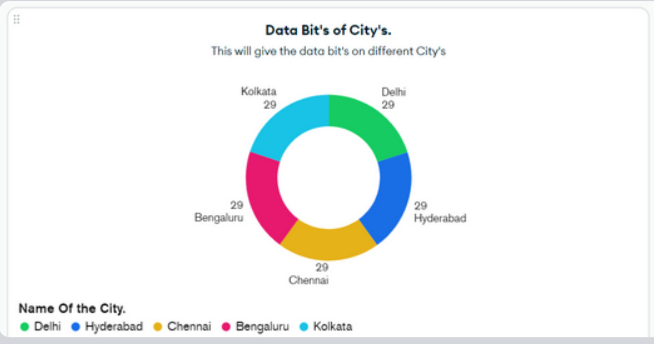
In this step i wanted to monitor the real time data that is coming through the pipeline and visualize the data .

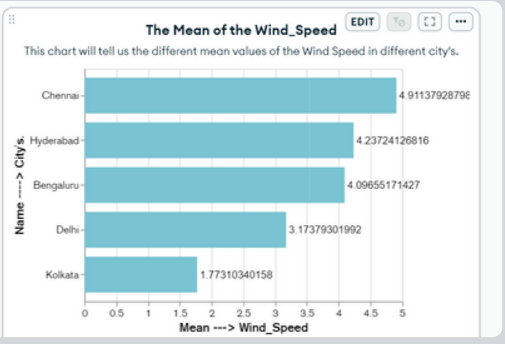
First to work with mongoDB atlas you need to create a free account from the official website { https://www.mongodb.com/atlas/database }

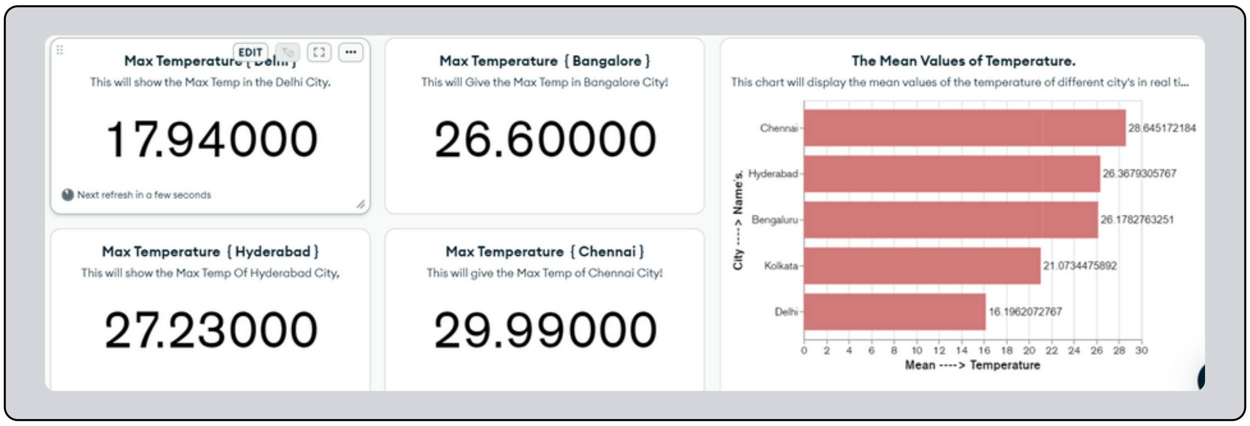


```
_id: ObjectId('65b5f981c8827e4d30c2cc7b')
timestamp: 2024-01-28T06:32:50.507+00:00
• coord: Object
weather: "[{"id":802,"main":"Clouds","description":"scattered clouds","icon":"03…"
base: "stations"
• main: Object
visibility: 6000
• wind: Object
• clouds: Object
dt: 1706423354
• sys: Object
timezone: 19800
id: 1277333
name: "Bengaluru"
cod: 200
```


Data Source
data
Query
Library
Type a {query} or an [aggregation] to pre-process your data
FIELDS
Name
Sampling Data Source...
CHART TYPE
Grouped Bar
Encode
Filter
Customize
X Axis
+ Aggregation
Y Axis
+ Category
Series
+ Category
PREVIEW
COMPLETE
SUBSET
Select a data source to visualize.
This is where we can find our data schema
This is where we use different plot's
We can write the desired query on the data in this place.
This is where we add the parameters.







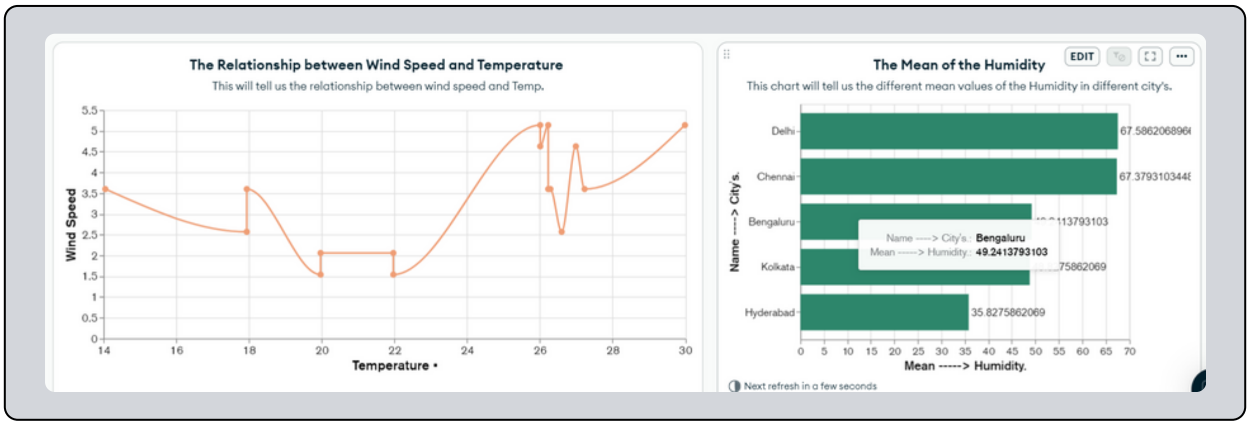
Max Temperature { Delhi }
This will show the Max Temp in the Delhi City.
17.94000
Next refresh in a few seconds

Max Temperature { Bangalore }
This will Give the Max Temp in Bangalore City!
26.60000

Max Temperature { Hyderabad }
This will show the Max Temp Of Hyderabad City.
27.23000

Max Temperature { Chennai }
This will give the Max Temp of Chennai City!
29.99000

The Mean Values of Temperature.
This chart will display the mean values of the temperature of different city's in real ti...



The Relationship between Wind Speed and Temperature
This will tell us the relationship between wind speed and Temp.

The Mean of the Humidity
This chart will tell us the different mean values of the Humidity in different city's.

Next refresh in a few seconds

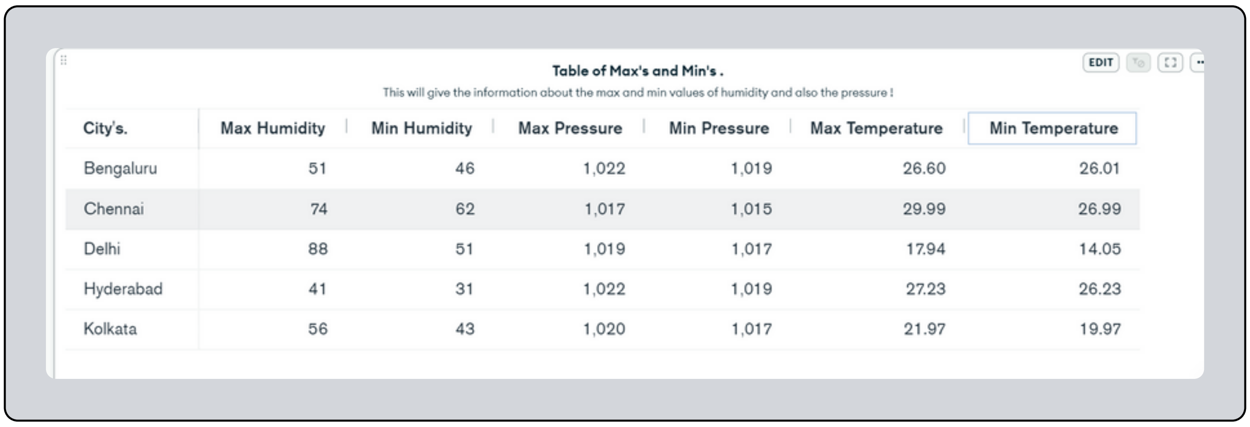
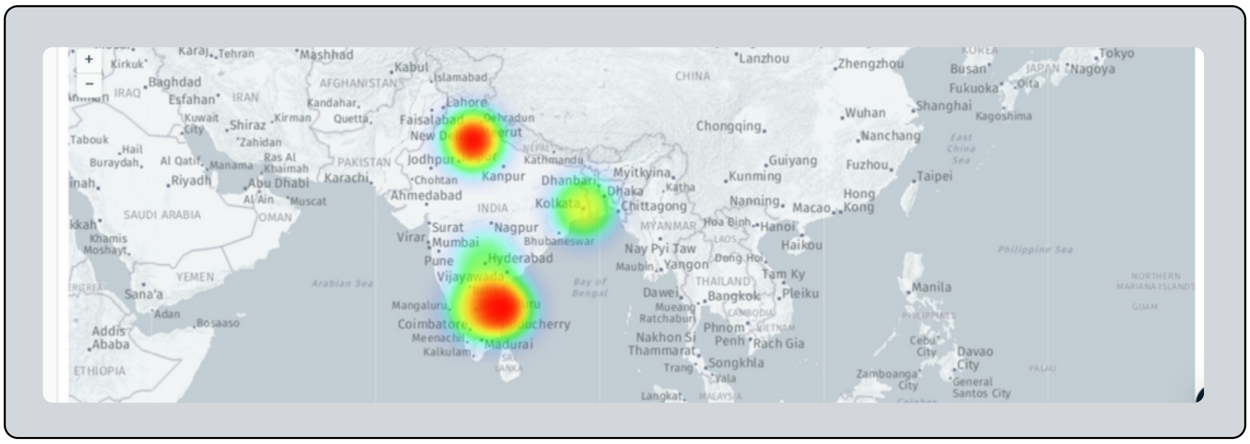


Table of Max's and Min's .
This will give the information about the max and min values of humidity and also the pressure !

City's.	Max Humidity	Min Humidity	Max Pressure	Min Pressure	Max Temperature	Min Temperature
Bengaluru	51	46	1,022	1,019	26.60	26.01
Chennai	74	62	1,017	1,015	29.99	26.99
Delhi	88	51	1,019	1,017	17.94	14.05
Hyderabad	41	31	1,022	1,019	27.23	26.23
Kolkata	56	43	1,020	1,017	21.97	19.97



5. Usage :-

Accessing the System: *To begin utilizing the system, ensure that all necessary components are installed and configured as per the instructions provided in the "Installation and Setup" section.*

Initializing Data Retrieval: *Start by accessing the OpenWeather API to fetch real-time weather data for the desired cities. Utilize the appropriate endpoints and parameters to retrieve relevant information.*

Data Ingestion with Kafka: *Once the weather data is retrieved, it is ingested into Kafka topics for efficient processing and distribution across the system. Ensure that Kafka is running and configured to receive incoming data streams.*

Data Processing with PySpark: *Set up PySpark jobs to process the incoming weather data from Kafka. Implement necessary transformations and analyses to derive meaningful insights from the raw data.*

Storing Processed Data: *Processed data is then stored in MongoDB Atlas for persistent storage and easy access. Ensure that the MongoDB connection is established and authenticated to enable seamless data storage.*

Visualization of Real-Time Data: *Utilize visualization tools or libraries to create intuitive dashboards or graphs that display real-time weather data fetched from MongoDB Atlas. This enables users to quickly interpret and analyze the current weather conditions.*

Monitoring and Maintenance: *Regularly monitor the system for any issues or anomalies. Perform necessary maintenance tasks such as updating dependencies or scaling components to ensure optimal performance and reliability.*

Troubleshooting: *In case of any errors or discrepancies, refer to system logs and error messages to identify the root cause. Troubleshoot accordingly and implement necessary fixes to restore functionality.*

User Interaction: *Users can interact with the system through defined interfaces or APIs to query specific weather data, adjust settings, or retrieve historical information as needed.*

Documentation and Support: *Refer to comprehensive documentation provided alongside the system for detailed instructions on usage, troubleshooting steps, and best practices. For further assistance, reach out to the support team for prompt resolution of any queries or issues.*

6. Scalability and Performance:-

Vertical Scalability: *Assess the system's capability to handle increased loads by vertically scaling individual components, such as upgrading hardware resources like CPU, RAM, or storage capacity, to meet growing demands.*

Horizontal Scalability: *Explore options for horizontal scaling by distributing workloads across multiple instances or nodes, allowing the system to accommodate higher volumes of data and processing tasks efficiently.*

Load Balancing: *Implement load balancing techniques to evenly distribute incoming requests or data streams across available resources, preventing overloading of any single component and optimizing system performance.*

Auto-scaling Mechanisms: *Configure auto-scaling mechanisms to automatically adjust resource allocation based on fluctuating workloads, ensuring optimal performance during peak usage periods while minimizing resource wastage during off-peak times.*

Monitoring and Optimization: *Continuously monitor system performance metrics such as CPU utilization, memory usage, and response times to identify potential bottlenecks or performance issues. Optimize configurations and resource allocation accordingly to enhance scalability and responsiveness.*

Database Sharding: *Implement database sharding techniques to horizontally partition data across multiple nodes, distributing the data processing workload and improving both scalability and performance for large-scale data storage and retrieval operations.*

Caching Strategies: *Utilize caching mechanisms to store frequently accessed data or computations, reducing the need for redundant processing and enhancing overall system performance, especially for read-heavy workloads.*

Streamlining Data Pipelines: *Streamline data ingestion, processing, and storage pipelines to minimize latency and overhead, optimizing the end-to-end workflow for real-time data processing and analysis.*

Performance Testing: *Conduct rigorous performance testing under various load conditions to assess the system's scalability limits and identify potential areas for optimization. Utilize performance testing tools and methodologies to simulate real-world scenarios and validate system resilience.*

7. Future Enhancements:-

Machine Learning Integration: *Explore opportunities to integrate machine learning algorithms for predictive analysis, such as forecasting weather trends or detecting anomalies in real-time data streams, enhancing the system's predictive capabilities.*

Enhanced Visualization Features: *Implement advanced visualization techniques, including interactive maps, charts, and geospatial analysis tools, to provide users with richer insights and a more intuitive understanding of weather patterns and trends.*

Optimized Data Processing Algorithms: *Investigate optimization techniques for data processing algorithms to improve efficiency and reduce computational overhead, enabling faster processing of large volumes of incoming data with minimal latency.*

Geographical Expansion: *Expand the scope of the system to cover additional geographic regions or cities, leveraging the OpenWeather API to retrieve weather data for a broader range of locations and enhancing the system's utility and relevance.*

Integration with External Systems: *Integrate the system with external data sources or services, such as social media feeds or environmental sensors, to augment weather data with additional contextual information and enhance the accuracy of predictions and analyses.*

Real-Time Alerts and Notifications: *Implement real-time alerting mechanisms to notify users of significant weather events or changes, allowing for proactive decision-making and timely responses to emerging weather conditions.*

Multi-platform Support: *Develop cross-platform compatibility to support deployment on a variety of environments, including cloud-based infrastructure, edge devices, and mobile platforms, ensuring accessibility and flexibility for users across different devices and use cases.*

Scalability Improvements: *Further optimize system scalability by refining resource allocation strategies, enhancing fault tolerance mechanisms, and exploring containerization or microservices architectures to facilitate seamless horizontal scaling and elastic resource provisioning.*

Community Engagement Features: *Incorporate community engagement features such as user forums, crowdsourced data collection, or collaborative analysis tools to encourage user participation and foster a sense of community around the platform.*

8. Conclusion :-

The development of this weather data processing and visualization system represents a significant step towards harnessing the power of real-time data analytics to gain insights into weather patterns and trends. By leveraging technologies such as the OpenWeather API, Kafka, PySpark, and MongoDB Atlas, we have created a scalable and performant platform capable of ingesting, processing, and visualizing vast amounts of weather data in real-time.

Throughout this project, we have demonstrated the importance of efficient data processing pipelines, scalable architectures, and intuitive visualization tools in extracting actionable insights from raw weather data. From retrieving real-time weather information to storing and visualizing processed data, each component plays a crucial role in enabling users to make informed decisions based on the latest weather forecasts and trends.

Looking ahead, there are numerous opportunities for further enhancement and refinement, including the integration of machine learning algorithms for predictive analysis, expansion to cover additional geographic regions, and the implementation of real-time alerting mechanisms. By continuously iterating and improving upon the existing framework, we aim to provide users with a comprehensive and valuable tool for understanding and responding to weather-related phenomena.

In summary, this project serves as a testament to the power of data-driven approaches in tackling complex problems such as weather forecasting and analysis. By combining cutting-edge technologies with innovative solutions, we have laid the foundation for a versatile and adaptable platform capable of meeting the evolving needs of users in an increasingly dynamic environment.

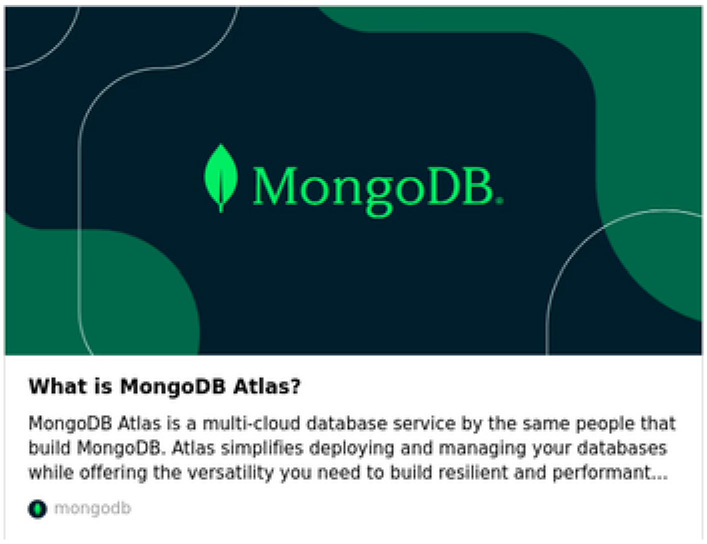
9. References :-

OpenWeather. (n.d.). OpenWeather API Documentation. Retrieved from https://openweathermap.org/api

Apache Kafka. (n.d.). Kafka Documentation. Retrieved from https://kafka.apache.org/documentation/

Apache Spark. (n.d.). PySpark Documentation. Retrieved from https://spark.apache.org/docs/latest/api/python/index.html

MongoDB. (n.d.). MongoDB Atlas Documentation. Retrieved from https://docs.atlas.mongodb.com/



What is MongoDB Atlas?

MongoDB Atlas is a multi-cloud database service by the same people that build MongoDB. Atlas simplifies deploying and managing your databases while offering the versatility you need to build resilient and performant...

mongodb

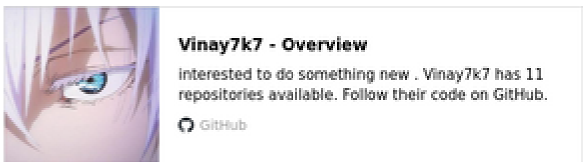
"Scalable Data Processing with Apache Kafka, Spark, and Cassandra" by Guy Harrison. (2017). O'Reilly Media.

"Big Data Analytics with Spark: A Practitioner's Guide to Using Spark for Large-Scale Data Processing, Machine Learning, and Graph Analytics" by Mohammed Guller. (2018). Apress.

"Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems" by Martin Kleppmann. (2017). O'Reilly Media.

"Real-Time Analytics: Techniques to Analyze and Visualize Streaming Data" by Byron Ellis et al. (2014). Wiley.

Github :-



Vinay7k7 - Overview

interested to do something new . Vinay7k7 has 11 repositories available. Follow their code on GitHub.

GitHub