Stock Data Streaming And Analysis

UMKC

Introduction

- The stock market is a constellation of marketplaces where securities like stocks and bonds are bought and sold. Stock markets provide you with easy, transparent access to investment assets, and they help professional investors determine fair prices for public companies.

- Data analytics refers to the process of analyzing vast quantities of data to identify commonalities, insights, and trends. While data analytics tools can be used in many industries, including healthcare, politics, retail, banking, and government organizations, they're vital for competitiveness in modern financial markets.

- In finance, data analytics are crucial to interpret the ups and downs of capital markets. With great financial data science, traders and investment advisors earn the confidence to make informed decisions about buying, selling, or holding a particular security. It allows them to manage a portfolio based on short-, mid-, or long-term objectives.

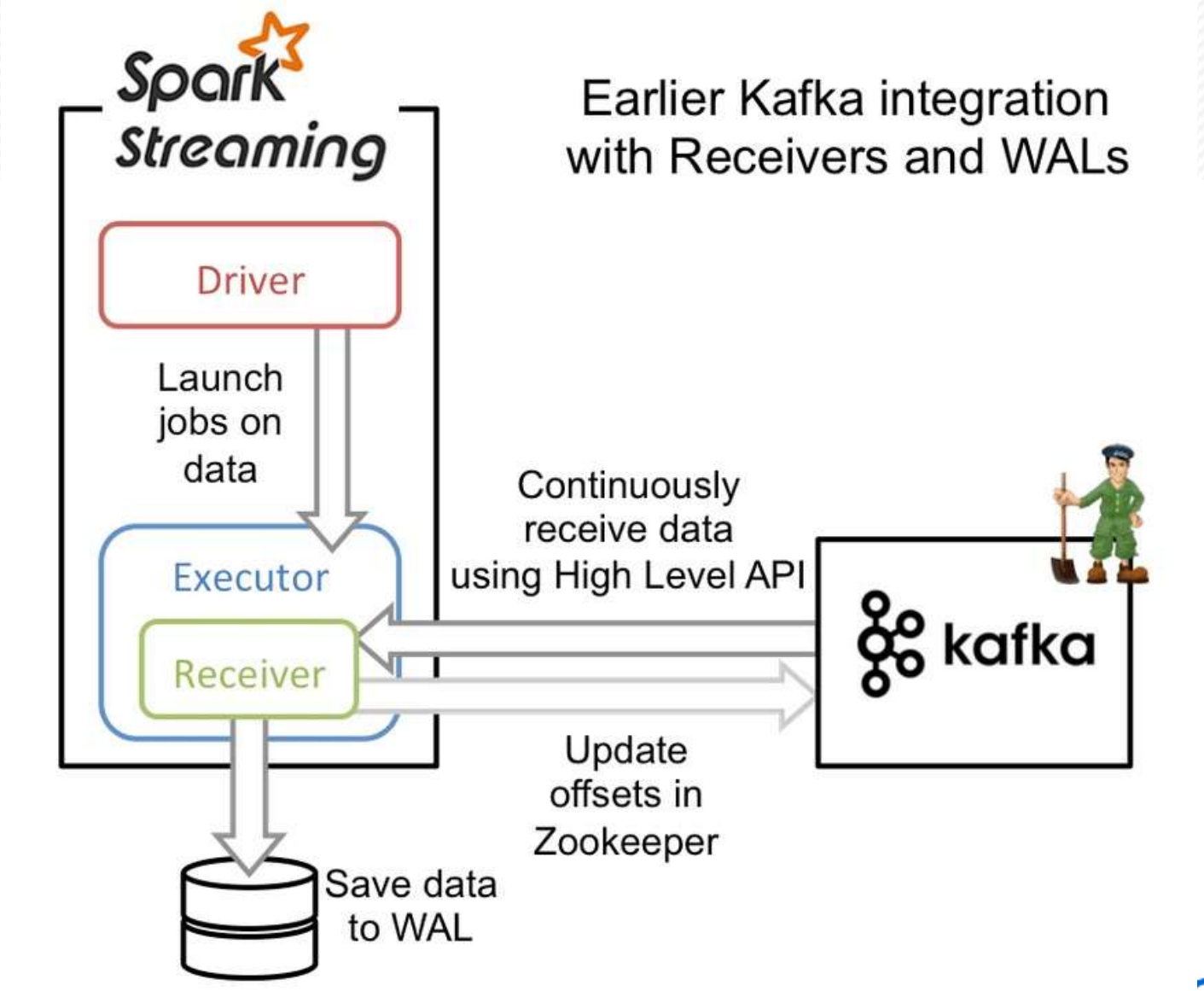
UMKC

Abstract

The main objective of the project is to visualize the analysis made from the live streaming of the stock data. The live data streaming from a stock analysis is initially used as the source data. Kafka streams the data from the Alpha Vintage website. The data stored in topics is then analyzed. For a variety of use scenarios, Jupyter notebook visualizes data analysis. The project has been improved with the aid of AWS services. The main objectives of the enhancement are to enhance data security and storage

UMKC

System Architecture



Tools & Technologies

Streaming Tools

- Requests library
- Pandas datareader library
- API : Tiingo API, Alpha Vantage API
- Data Visualization Tool: Matplotlib, Seaborn

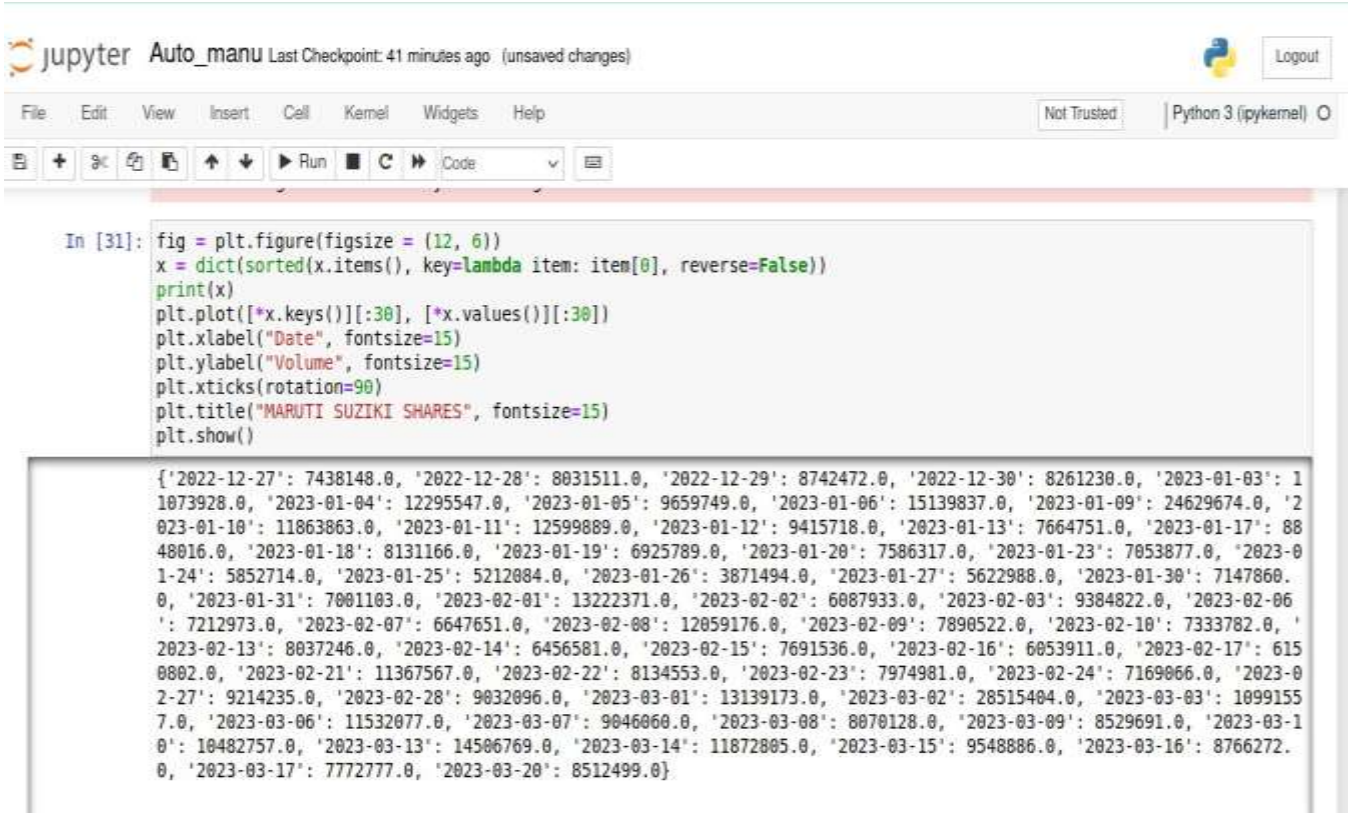
Technologies Used

- Spark
- Kafka
- Pyspark

UMKC

Source Data

- The Live Streaming of data is extracted from AlphaVintage website.
- The data is converted into Json Format which contains Metadata and Time Series (Daily).
- Data Size is 2MB.

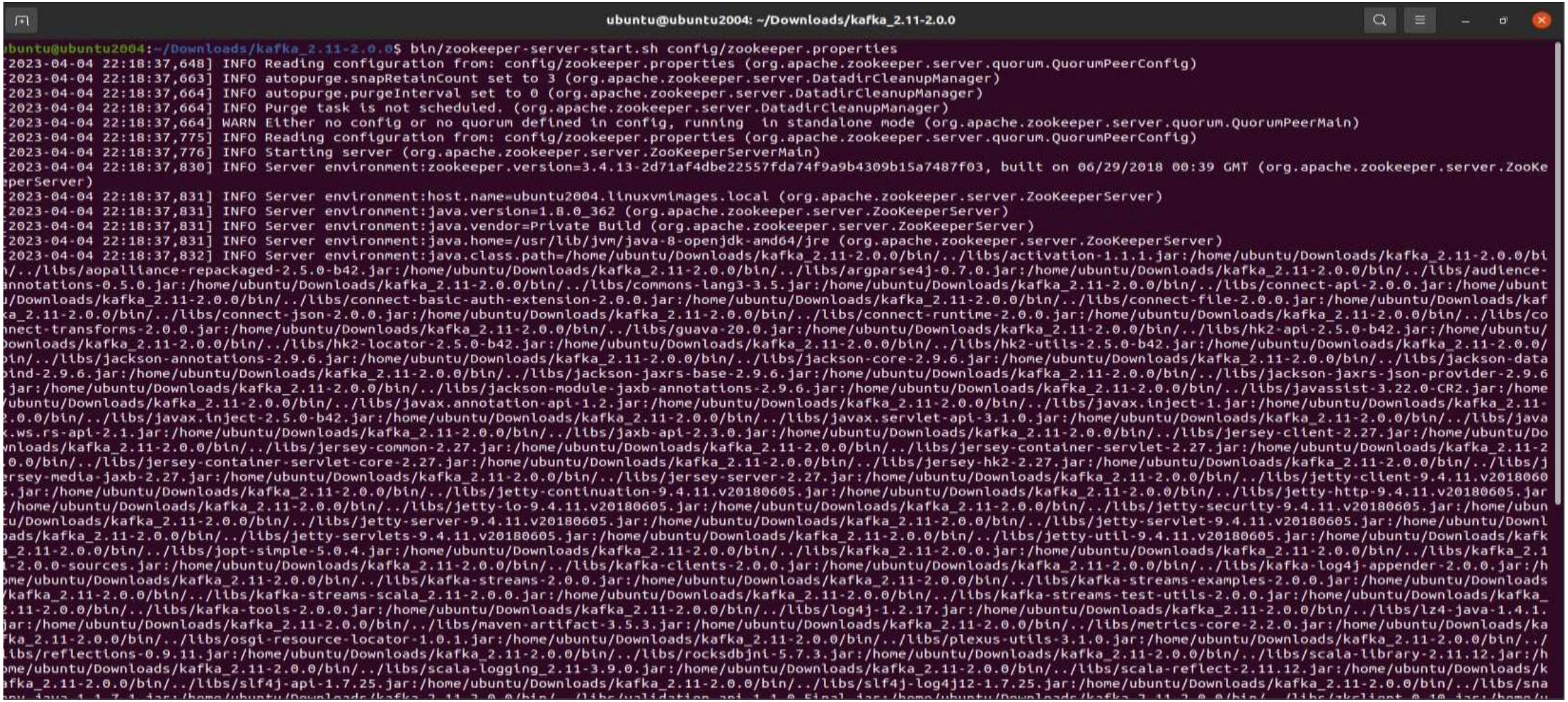


Source Code

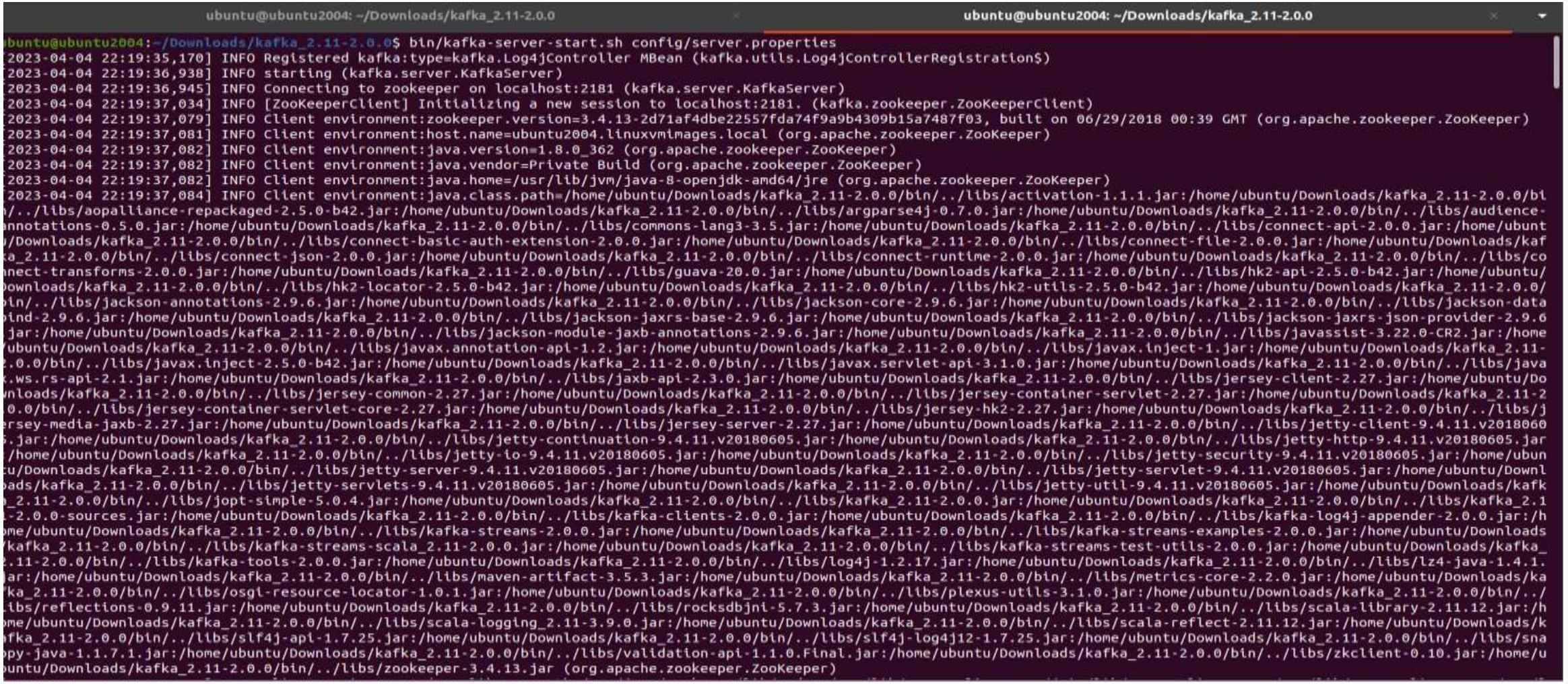
```python
from kafka import KafkaProducer
import requests
from json import dumps
import time

def on_message1(message):
    producer1.send('mn', message)
    producer1.flush()

producer1 = KafkaProducer(value_serializer=lambda m: dumps(m).encode('utf-8'), bootstrap_servers=['localhost:9092'])


# url to collect data of Mahindra and Mahindra shares
url = 'Shttps://www.alphavantage.co/query?function=TIME_SERIES_DAILY&symbol=M&M.BSE&apikey=36S0YNI4HJXTBL12&outputsize=full'
r = requests.get(url)
data=r.json()
#print(data)
#preprocessing
del(data['Meta Data'])
df=data['Time Series (Daily)']
#print(df)
jsonFile={}
j=0
for i in df.keys():
    jsonFile[j]=df[i]
    jsonFile[j]['0. date']=i
    #print(i)
    j+=1
print(jsonFile)
for i in jsonFile.keys():
        jsonFile[i]["open"]=jsonFile[i]["1. open"]
        jsonFile[i]["high"]=jsonFile[i]["2. high"]
        jsonFile[i]["low"]=jsonFile[i]["3. low"]
        jsonFile[i]["close"]=jsonFile[i]["4. close"]
        jsonFile[i]["volume"]=jsonFile[i]["5. volume"]
        jsonFile[i]["date"]=jsonFile[i]["0. date"]
        del jsonFile[i]["1. open"]
        del jsonFile[i]["2. high"]
        del jsonFile[i]["3. low"]
        del jsonFile[i]["4. close"]
        del jsonFile[i]["5. volume"]
```

```python
from pyspark.sql import SparkSession
from pyspark.sql.functions import *
from pyspark.sql.types import *

spark = SparkSession \
    .builder \
    .appName("Manufacturing_companies") \
    .getOrCreate()

raw_df = spark \
    .readStream \
    .format("kafka") \
    .option("kafka.bootstrap.servers", "localhost:9092") \
    .option("subscribe", "mn") \
    .option("startingOffsets", "latest") \
    .load() \
    .selectExpr("CAST(value AS STRING)")

# Created Schema for Structured Streaming

#timestamp,open,high,low,close,volume

schema = StructType(
    [
            StructField("open", StringType()),
            StructField("high", StringType()),
            StructField("low", StringType()),
            StructField("close", StringType()),
            StructField("volume", StringType()),
            StructField("date", StringType())
    ])
# Applied schema on data
data1= raw_df.select(from_json(raw_df.value, schema).alias("data"))

#Fetching required data for processing queries
output_df1 = data1.select(to_json(struct(col("data.volume"),col("data.date"))).alias("value"))

# Sending the data to kafka brocker
query = output_df1.writeStream.format("kafka").option("kafka.bootstrap.servers", "localhost:9092").option("checkpointLocation", "/tmp/checkpoint2").option("topic", "mn1").start()
```

Zookeeper Initiation



Kafka Initiation



Topic Creation

```
ubuntu@ubuntu2004:~/Downloads/kafka_2.11-2.0.0$ bin/kafka-topics.sh --create --zookeeper localhost:2181 --topic mm --replication-factor 1 --partitions 1
Created topic "mm".
ubuntu@ubuntu2004:~/Downloads/kafka_2.11-2.0.0$ bin/kafka-topics.sh --create --zookeeper localhost:2181 --topic mm1 --replication-factor 1 --partitions 1
Created topic "mm1".
ubuntu@ubuntu2004:~/Downloads/kafka_2.11-2.0.0$
```

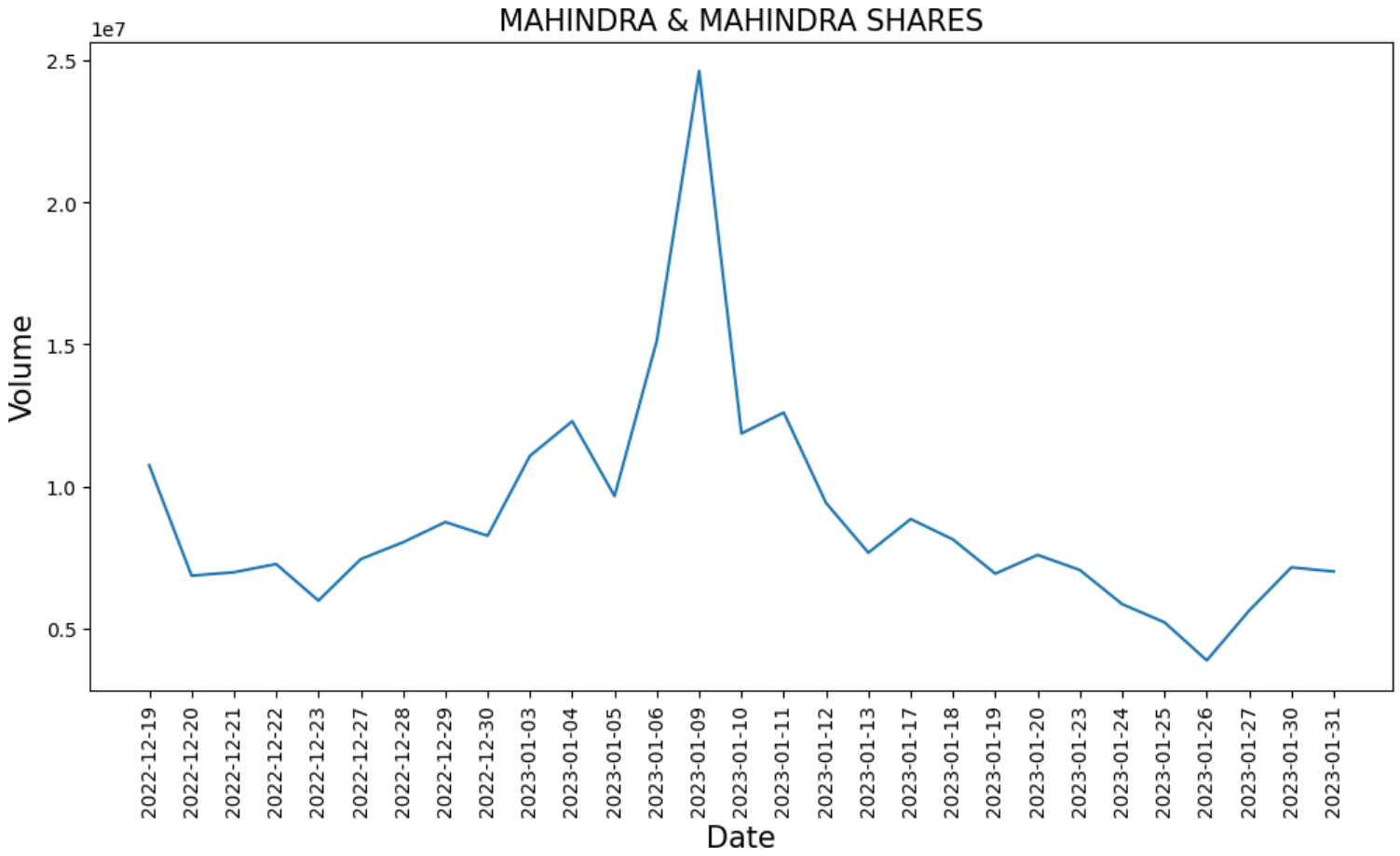
UMKC

Preprocessed Data

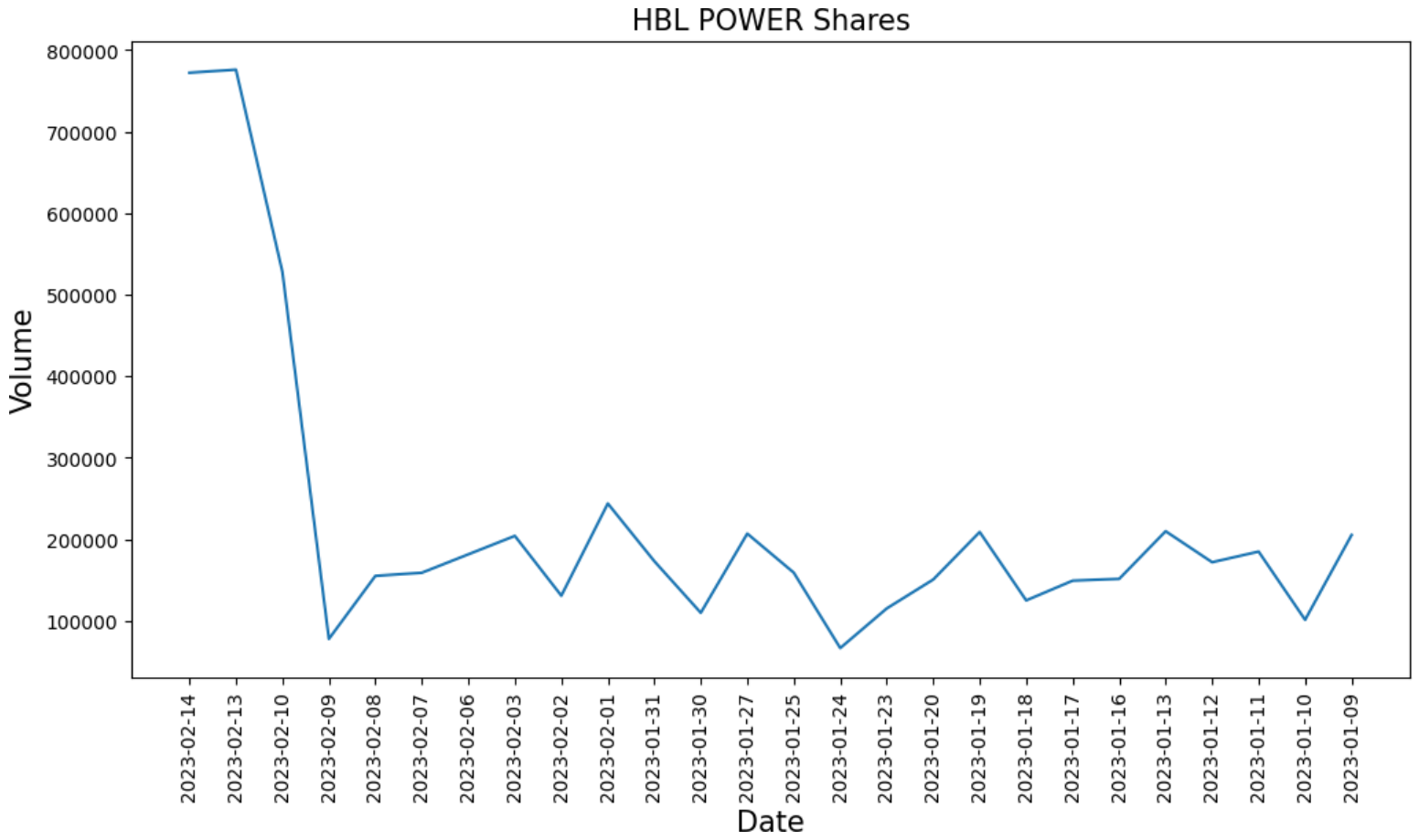
Spark - Submit

```
ubuntu@ubuntu2004:~/Downloads/spark-3.1.2-bin-hadoop3.2/bin$ spark-submit --packages org.apache.spark:spark-sql-kafka-0-10_2.12:3.1.2 /home/ubuntu/Downloads/Mahendra/mm1.py
23/04/04 22:34:32 WARN Utils: Your hostname, ubuntu2004 resolves to a loopback address: 127.0.1.1; using 192.168.86.42 instead (on interface ens33)
23/04/04 22:34:32 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
:: loading settings :: url = jar:file:/home/ubuntu/Downloads/spark-3.1.2-bin-hadoop3.2/jars/ivy-2.4.0.jar!/org/apache/ivy/core/settings/ivysettings.xml
Ivy Default Cache set to: /home/ubuntu/.ivy2/cache
The jars for the packages stored in: /home/ubuntu/.ivy2/jars
org.apache.spark#spark-sql-kafka-0-10_2.12 added as a dependency
:: resolving dependencies :: org.apache.spark#spark-submit-parent-77eeae84-0ce2-4520-93f6-0617be7a116c;1.0
        confs: [default]
        found org.apache.spark#spark-sql-kafka-0-10_2.12;3.1.2 in central
        found org.apache.spark#spark-token-provider-kafka-0-10_2.12;3.1.2 in central
        found org.apache.kafka#kafka-clients;2.6.0 in central
        found com.github.luben#zstd-jni;1.4.8-1 in central
        found org.lz4#lz4-java;1.7.1 in central
        found org.xerial.snappy#snappy-java;1.1.8.2 in central
        found org.slf4j#slf4j-api;1.7.30 in central
        found org.spark-project.spark#unused;1.0.0 in central
        found org.apache.commons#commons-pool2;2.6.2 in central
:: resolution report :: resolve 2772ms :: artifacts dl 200ms
        :: modules in use:
        com.github.luben#zstd-jni;1.4.8-1 from central in [default]
        org.apache.commons#commons-pool2;2.6.2 from central in [default]
        org.apache.kafka#kafka-clients;2.6.0 from central in [default]
        org.apache.spark#spark-sql-kafka-0-10_2.12;3.1.2 from central in [default]
        org.apache.spark#spark-token-provider-kafka-0-10_2.12;3.1.2 from central in [default]
        org.lz4#lz4-java;1.7.1 from central in [default]
        org.slf4j#slf4j-api;1.7.30 from central in [default]
        org.spark-project.spark#unused;1.0.0 from central in [default]
        org.xerial.snappy#snappy-java;1.1.8.2 from central in [default]
        ---------------------------------------------------------------------
        |                  |            modules            ||   artifacts   |
        |       conf       | number| search|dwnlded|evicted|| number|dwnlded|
        ---------------------------------------------------------------------
        |      default     |   9   |   0   |   0   |   0   ||   9   |   0   |
        ---------------------------------------------------------------------
:: retrieving :: org.apache.spark#spark-submit-parent-77eeae84-0ce2-4520-93f6-0617be7a116c
        confs: [default]
        0 artifacts copied, 9 already retrieved (0kB/264ms)
23/04/04 22:34:42 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
python3: can't open file '/home/ubuntu/Downloads/Mahendra/mm1.py': [Errno 2] No such file or directory
log4j:WARN No appenders could be found for logger (org.apache.spark.util.ShutdownHookManager).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
ubuntu@ubuntu2004:~/Downloads/spark-3.1.2-bin-hadoop3.2/bin$
```

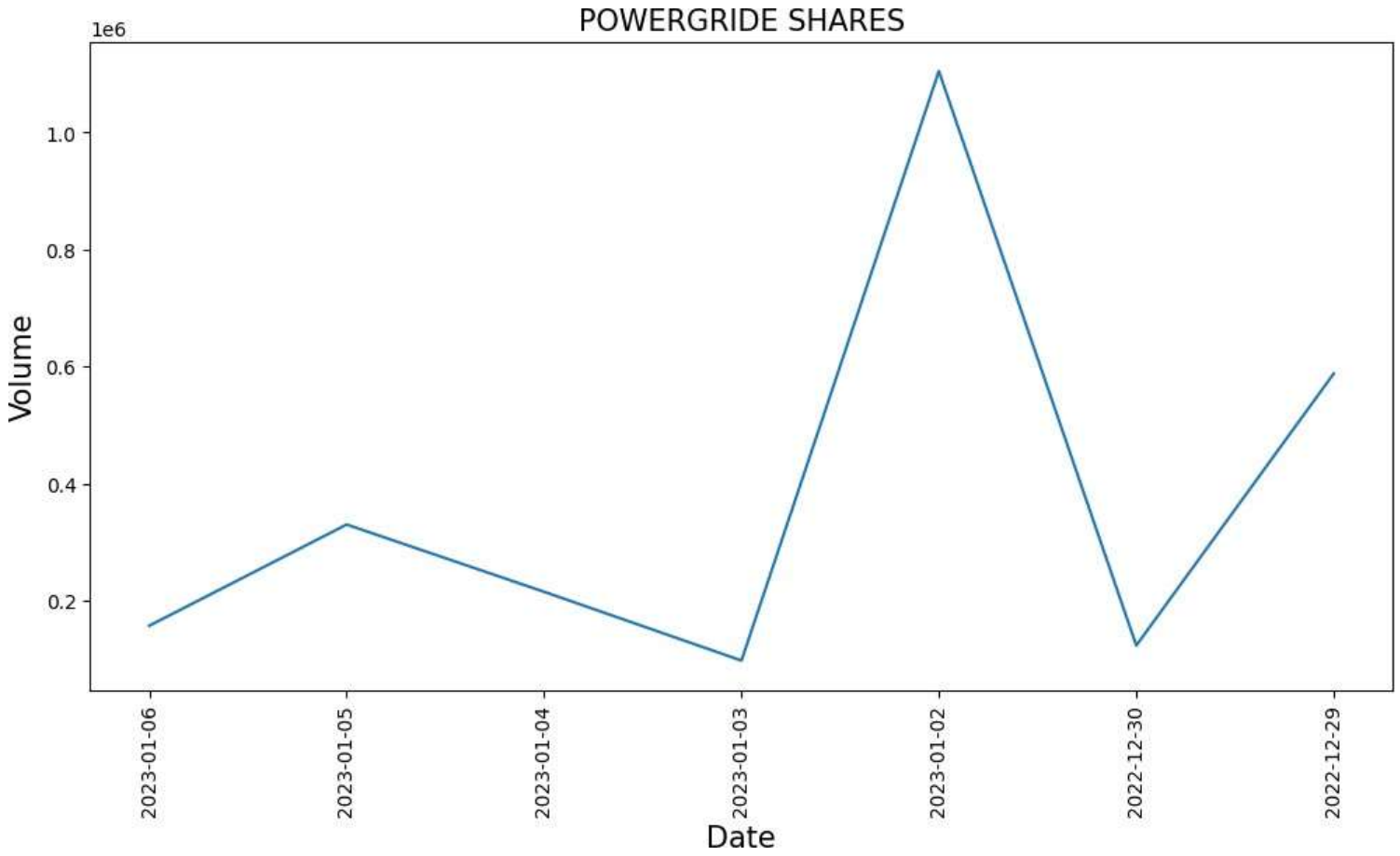
Mahendra_Auto_Manufactures



HBL_Power_Battery_Company



Powergrid_Charging_Point

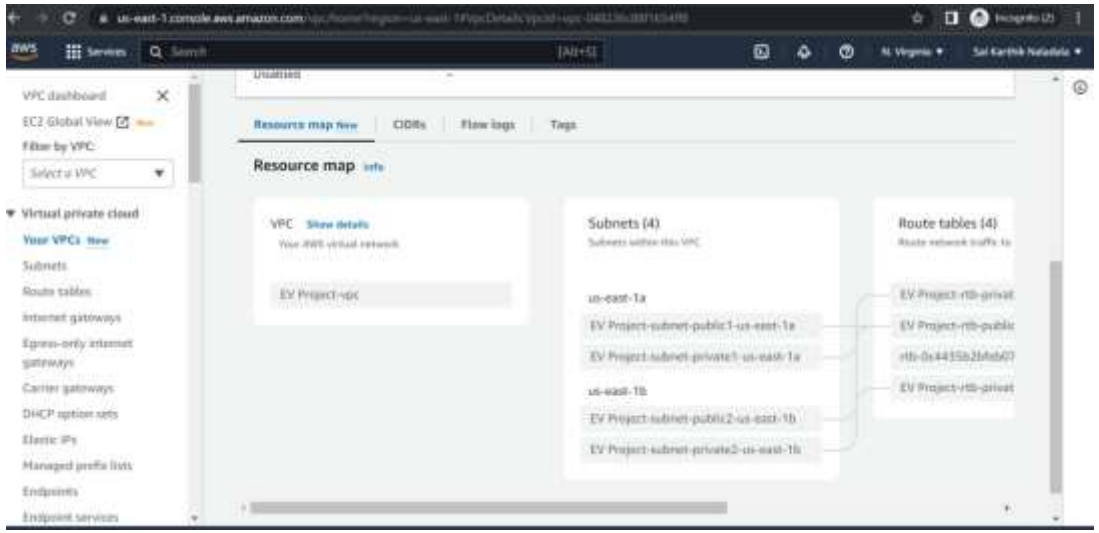


AWS Services

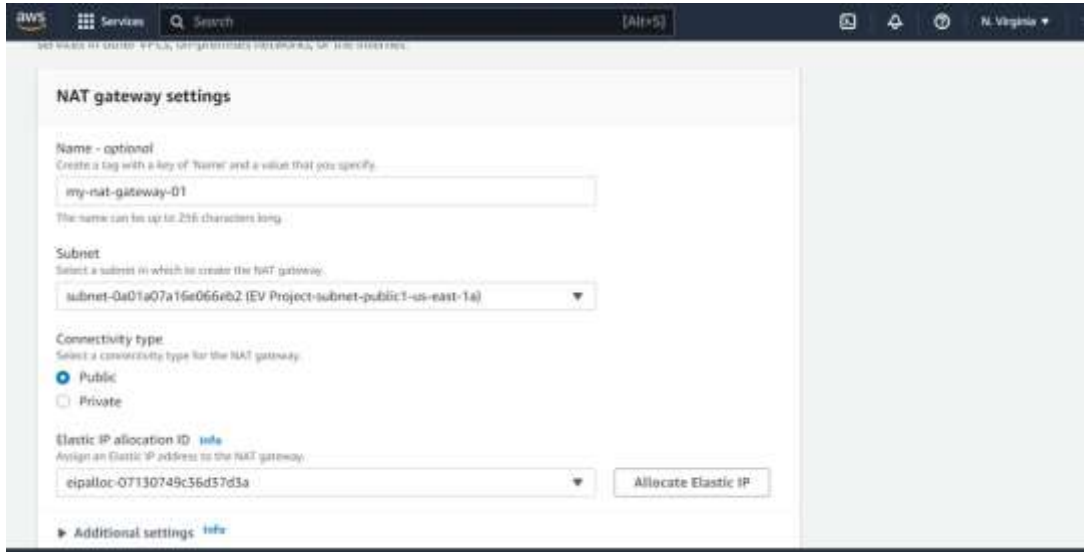
amazon web services

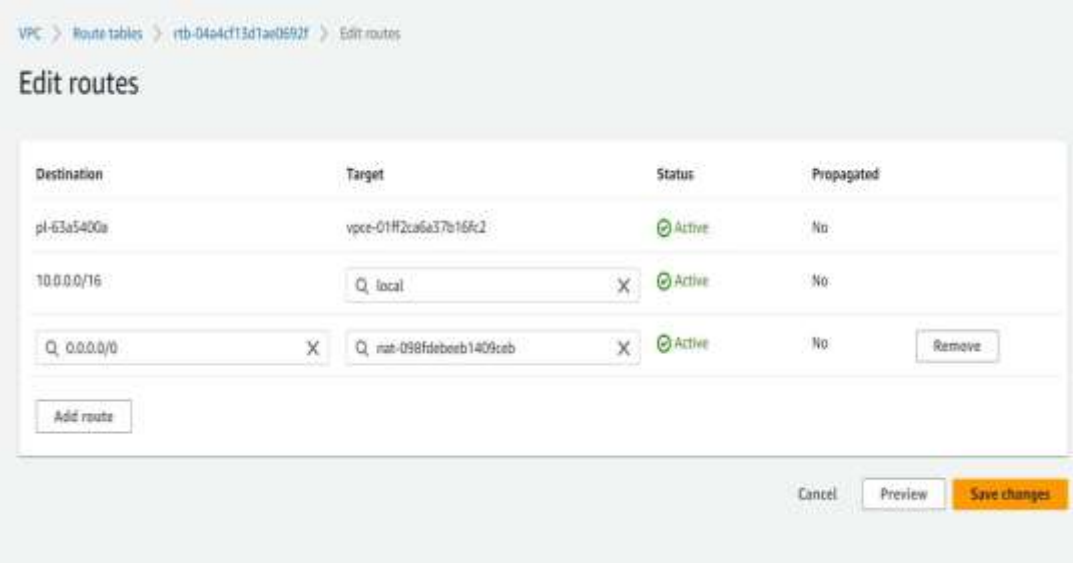
VPC Creation

- Firstly we create a separate VPC for our project with minimum two public and private subnets, though we are deploying our application private subnet but to connect to internet we need public subnets as well.
- We have created my VPC ‘EV Project-vpc’ and now as we need our private subnet to connect to internet for installing required libraries in our EC2 instance we have to create NAT gateway



Creation of NAT Gateway

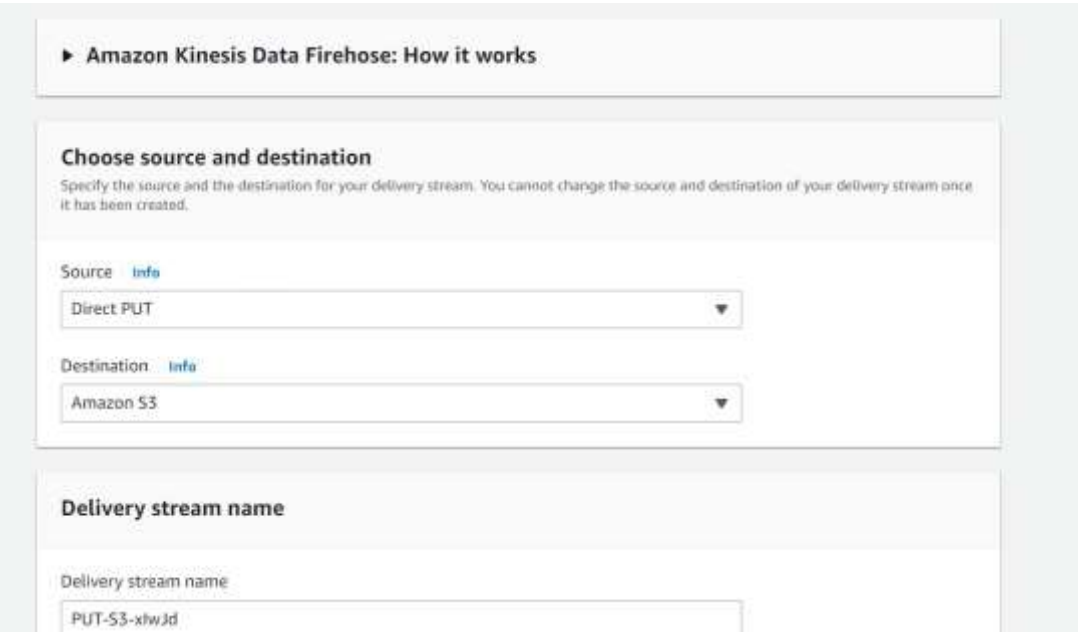


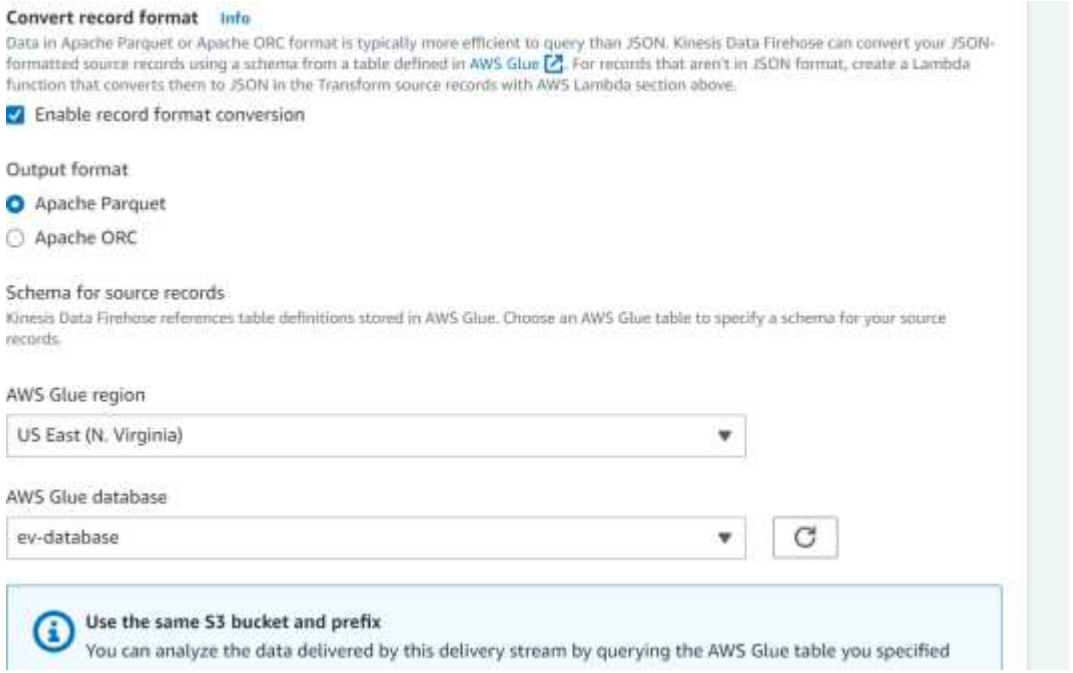


Launching MSK Cluster

- Created an Amazon MSK (Amazon Managed Streaming for Apache Kafka), which will stream the data through Kafka. To optimize cost we have launched only two brokers which will be running in each private subnet.

Kinesis Data Firehose





Lambda

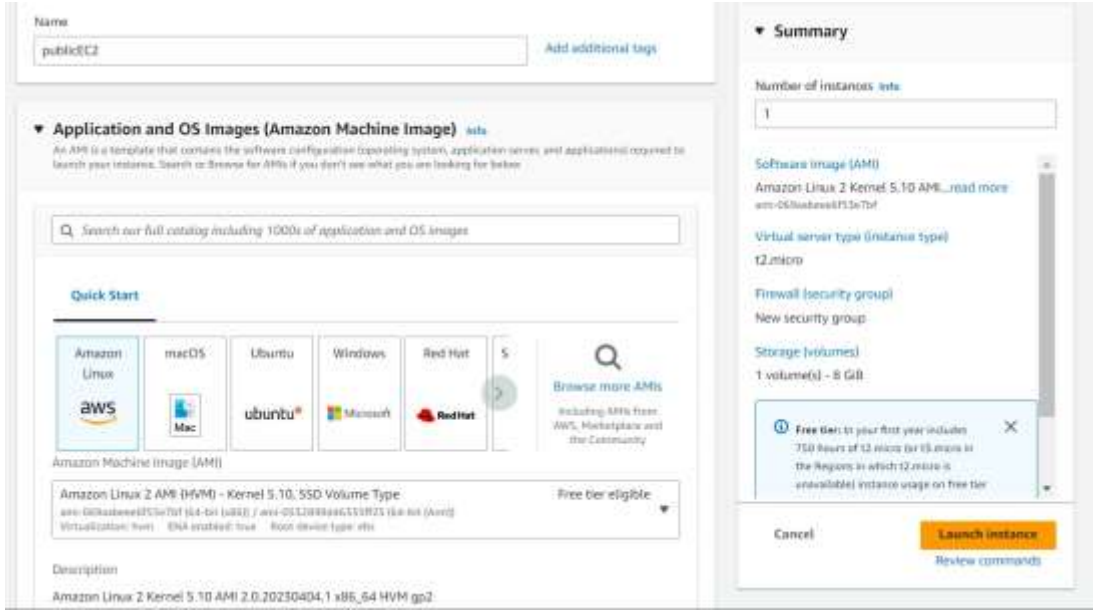
Producer Lambda:

- This lambda function will get the data from API using requests, preprocess the data and publishes the data to Kafka producer.

Consumer Lambda:

- This lambda function will consume the data from Kafka topic and pushes the data into Kinesis Firehose.

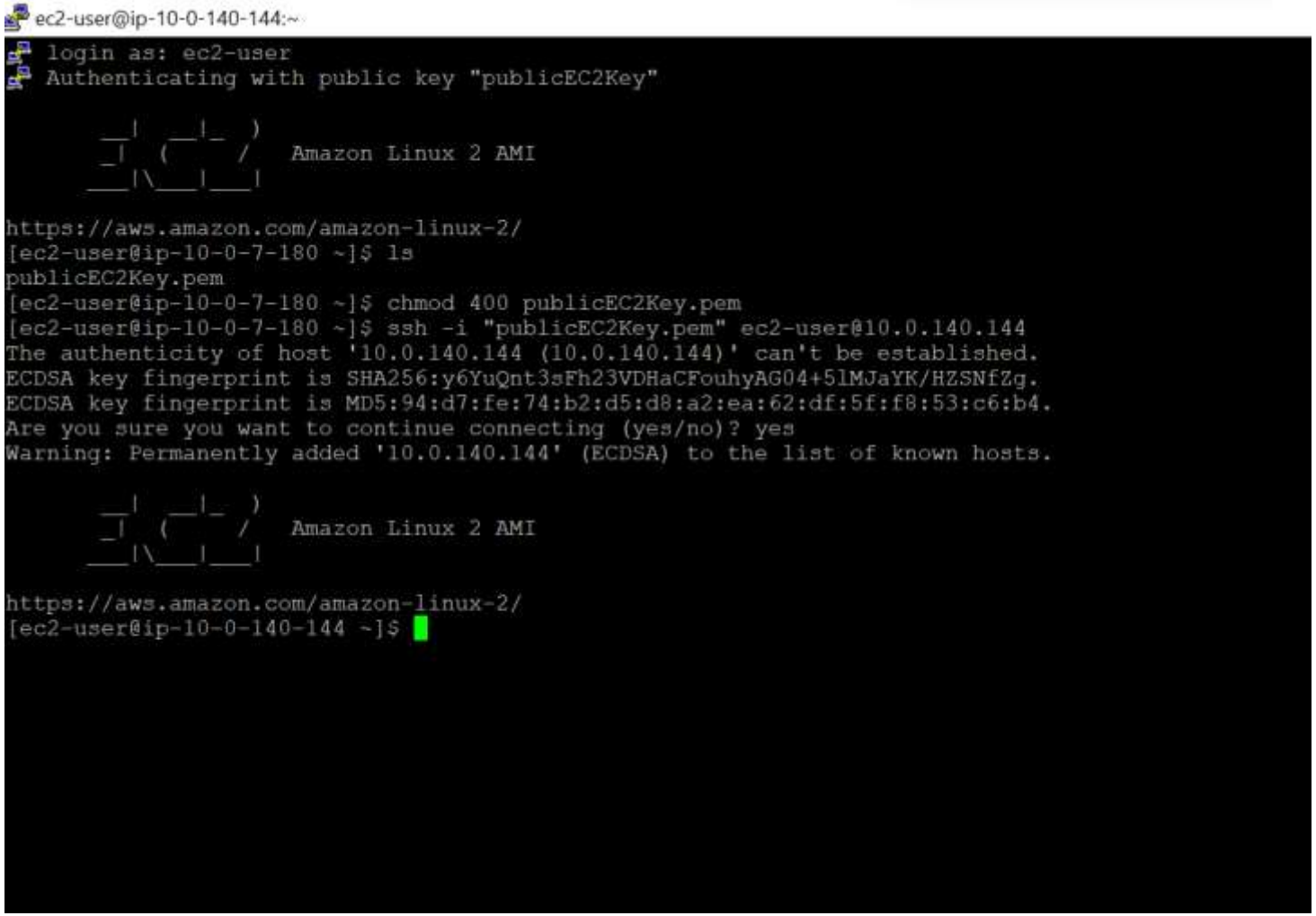
EC2 Instances



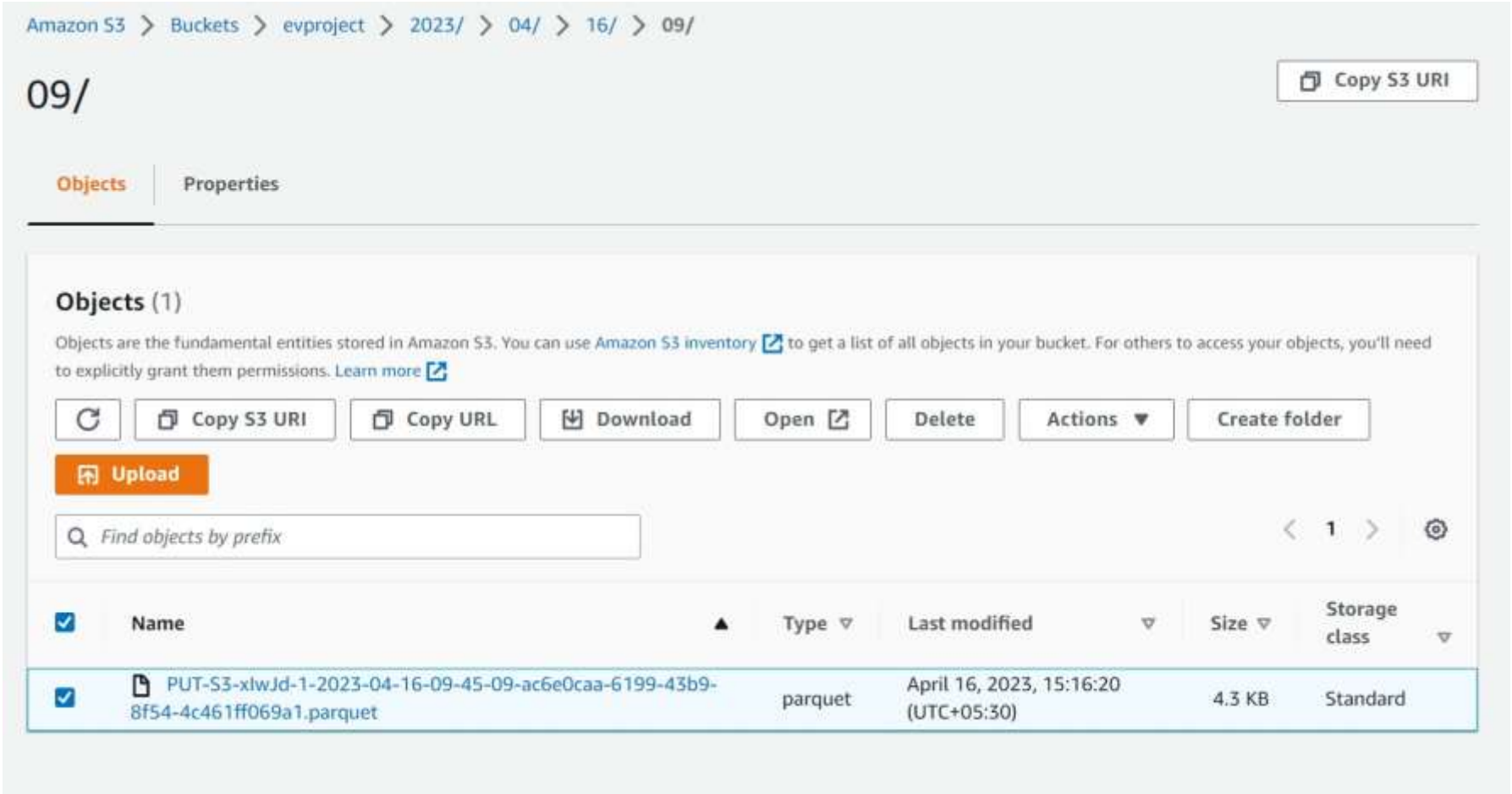
- We need an EC2 instance to create a Kafka topic for which we need to install Java and Kafka in linux machine. So we are creating two EC2 instance each in public and private subnet of our VPC.

Installing Kafka on EC2 Instance

- Now we have to send SHH into private EC2 instance from public EC2 instance using PuTTy, PuTTygen and WinSCP and install Java and Kafka using the below commands.

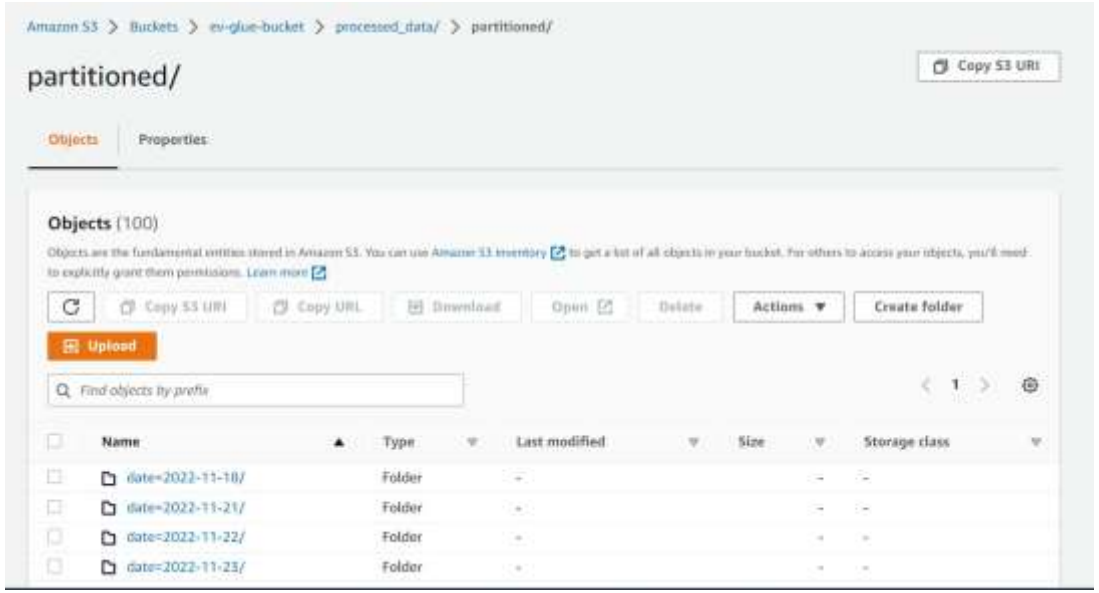


S3


Amazon S3 > Buckets > evproject > 2023/ > 04/ > 16/ > 09/
09/
Copy S3 URI
Objects
Properties
Objects (1)
Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more
Copy S3 URI
Copy URL
Download
Open
Delete
Actions
Create folder
Upload
Find objects by prefix
1
Name
Type
Last modified
Size
Storage class
PUT-S3-xlwJd-1-2023-04-16-09-45-09-ac6e0caa-6199-43b9-8f54-4c461ff069a1.parquet
parquet
April 16, 2023, 15:16:20 (UTC+05:30)
4.3 KB
Standard

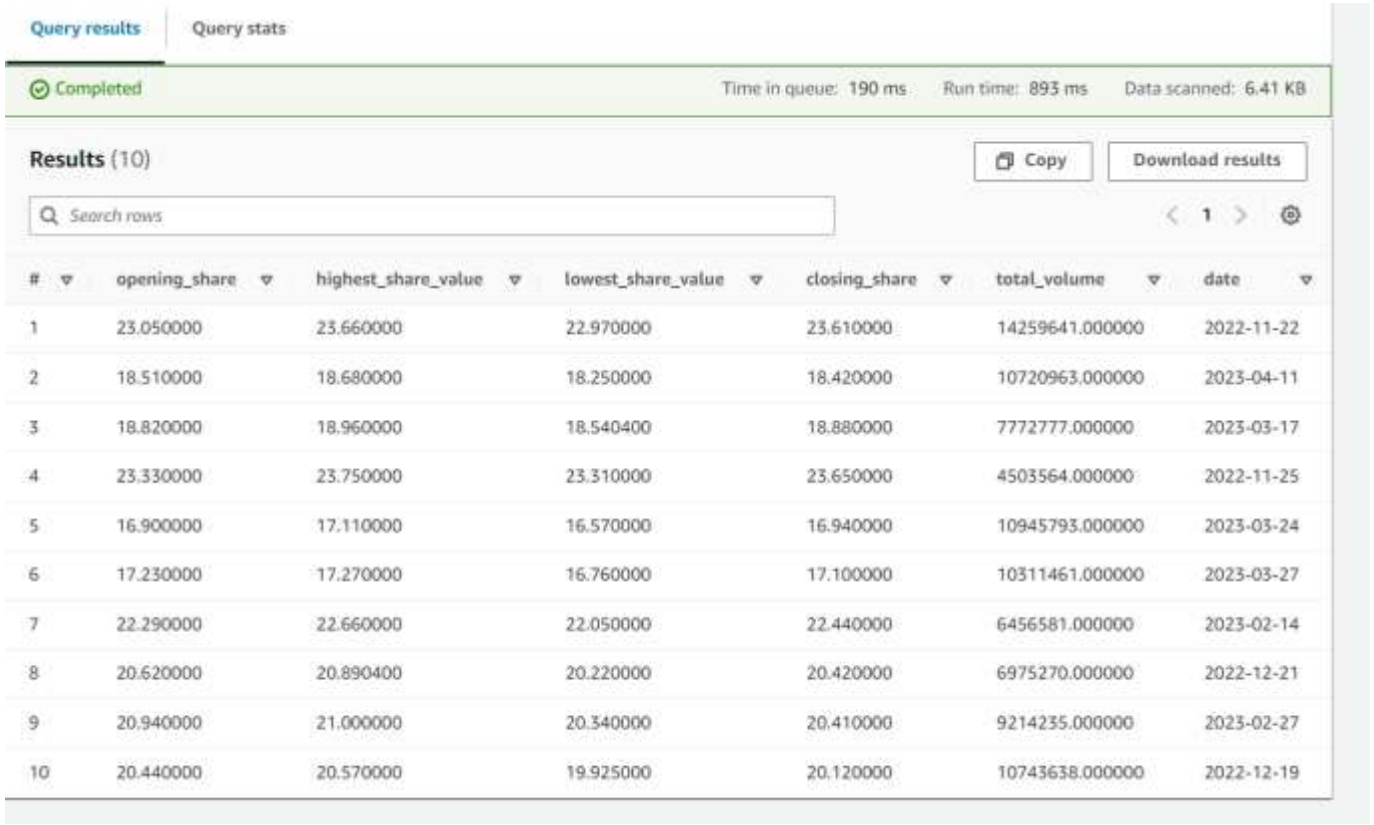
Glue

- As the data is successfully placed in s3, now we will create Glue job which will read the files from the s3 directory, process by extracting few columns and place the results in different s3 bucket and also create the Athena table in the mentioned database and stores the data.
- Processed data is successfully stored in given s3 location with date as partitioned column



Athena

- Query - select * from processed_data limit 10;



Query results | Query stats

Completed — Time in queue: 190 ms — Run time: 893 ms — Data scanned: 6.41 KB

Results (10) — Copy — Download results

Search rows — 1

#	opening_share	highest_share_value	lowest_share_value	closing_share	total_volume	date
1	23.050000	23.660000	22.970000	23.610000	14259641.000000	2022-11-22
2	18.510000	18.680000	18.250000	18.420000	10720963.000000	2023-04-11
3	18.820000	18.960000	18.540400	18.880000	7772777.000000	2023-03-17
4	23.330000	23.750000	23.310000	23.650000	4503564.000000	2022-11-25
5	16.900000	17.110000	16.570000	16.940000	10945793.000000	2023-03-24
6	17.230000	17.270000	16.760000	17.100000	10311461.000000	2023-03-27
7	22.290000	22.660000	22.050000	22.440000	6456581.000000	2023-02-14
8	20.620000	20.890400	20.220000	20.420000	6975270.000000	2022-12-21
9	20.940000	21.000000	20.340000	20.410000	9214235.000000	2023-02-27
10	20.440000	20.570000	19.925000	20.120000	10743638.000000	2022-12-19

Conclusion

The examination of the stock data is displayed using real-time stock data from various businesses. The AWS Services were utilized for the project's improvement. The enhancement overcame the issues with storage and security. The analyzed data from various companies can be compared from the visualization. The visualization helps the customer to choose the profitable stock from the analyzed data. Ultimately, the consumer might anticipate profitable results from the stock data analysis.

UMKC

References

1 https://www.digitalocean.com/community/tutorials/how-to-set-up-jupyternotebook-with-python-3-on-ubuntu-20-04-and-connect-via-ssh-tunneling

2 https://sparkbyexamples.com/kafka/apache-kafka-cluster-setup/

3 https://www.alphavantage.co/documentation/

UMKC

Thank U!