# STOCK MARKET REALTIME KAFKA-STREAMING (AWS PROJECT)

Harshda Rawool

# Problem Statement: Enhancing Investment Decision-Making with Real-Time Market Insights

In the realm of portfolio management, traditional methodologies predominantly hinge on historical data analysis and periodic reviews. However, this conventional approach often leads to missed opportunities and suboptimal portfolio allocations. The absence of a real-time market analysis capability compounds this issue, as it deprives us of timely insights into market trends, volatility, and asset performance. Consequently, our ability to make informed investment decisions and optimize portfolio returns is significantly impeded.

This projects objective is to bridge this gap by developing a robust real-time market analysis capability. This solution aims to provide timely insights into market dynamics.

# OBJECTIVES

1. Real-Time Monitoring: Implement a system to monitor market trends, volatility, and asset performance instantaneously, enabling proactive decision-making.

2. Opportunity Identification: Develop algorithms to swiftly identify and seize emerging investment opportunities as they arise in the market.

3. Dynamic Portfolio Optimization: Enable the dynamic optimization of portfolio allocations based on up-to-the-minute market insights, ensuring the most efficient use of resources.

4. Enhanced Risk Management: Implement strategies to enhance risk management practices by promptly responding to market fluctuations and minimizing exposure to downside risks.

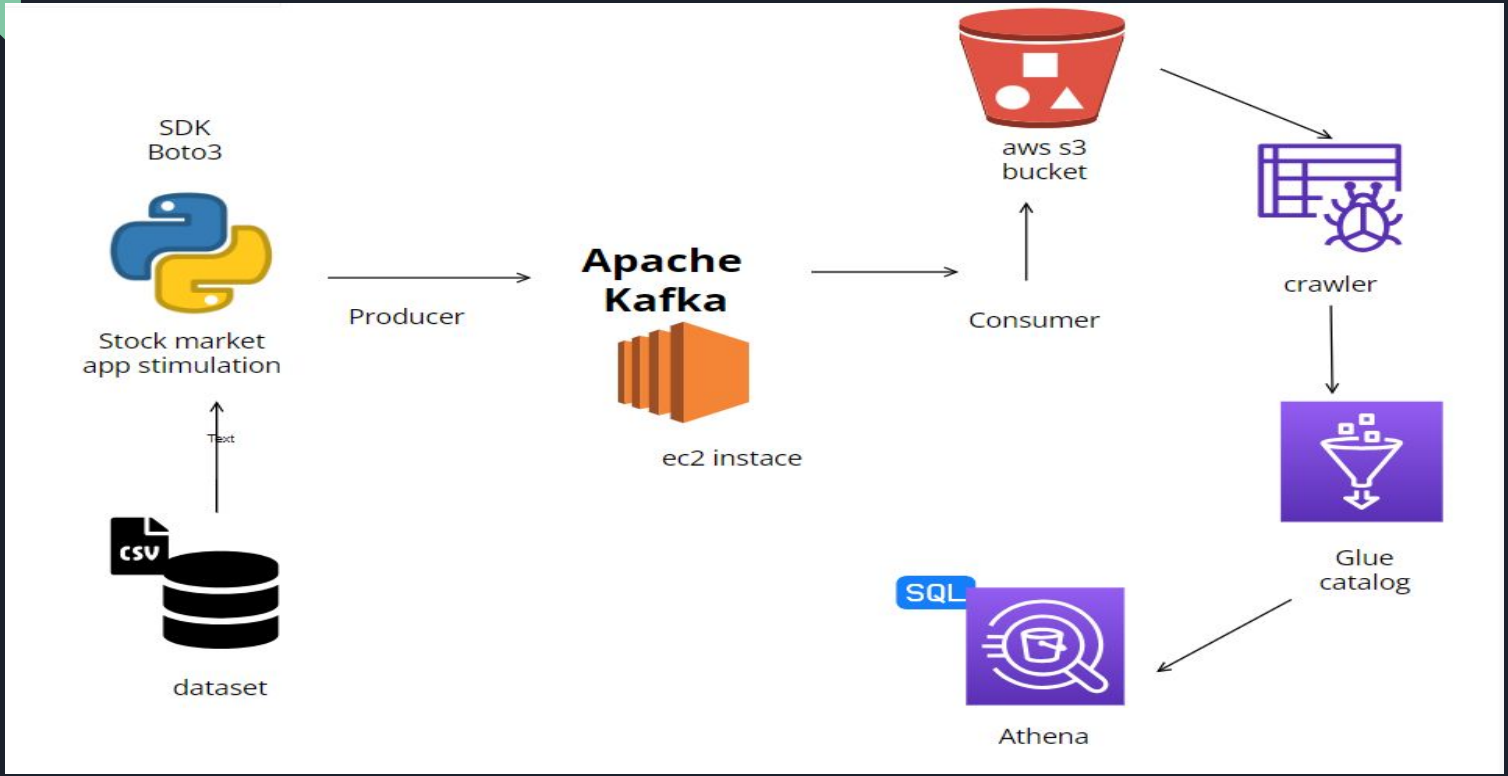
# PROJECT OVERVIEW

project focuses on delivering real-time data processing and analysis solutions tailored for informed decision-making. Leveraging Apache Kafka for continuous data ingestion and AWS infrastructure, including Glue for seamless data loading into S3, we enable efficient processing and organization of streaming financial data. By utilizing Athena for on-demand SQL querying, investment professionals can promptly analyze market trends,

# TECHNOLOGY USED :

1. Programming Language - Python
2. Amazon Web Service (AWS)
3. S3 (Simple Storage Service)
4. Athena
5. Glue Crawler
6. Glue Catalog
7. EC2
8. Apache Kafka

# ARCHITECTURE



# DATASET

a stock market dataset sourced from Kaggle, comprising index values, date, open, high, low, close, adjusted close, volume, and close USD prices,

# EC2 INSTANCE FOR INSTALLING KAFKA

- Set up Kafka on an EC2 instance, begin by launching an appropriate instance type and installing Java Development Kit (JDK).

- Next, download and extract the Kafka binaries onto the instance, and configure Kafka by modifying the necessary configuration files.

**Do a "sudo nano config/server.properties" - change ADVERTISED_LISTENERS to {public ip of the EC2 instance}**
(By doing this modification we can access kafka server from our local machine as now it will  run on public ip instead of private  this will help us to send data to producer programmatically)

```
############################# Socket Server Settings #############################

# The address the socket server listens on. If not configured, the host name will be equal to the value of
# java.net.InetAddress.getCanonicalHostName(), with PLAINTEXT listener name, and port 9092.
#   FORMAT:
#     listeners = listener_name://host_name:port
#   EXAMPLE:
#     listeners = PLAINTEXT://your.host.name:9092
#listeners=PLAINTEXT://:9092

# Listener name, hostname and port the broker will advertise to clients.
# If not set, it uses the value for "listeners".
advertised.listeners=PLAINTEXT://15.206.165.54:9092

# Maps listener names to security protocols, the default is for them to be the same. See the config documentation for m$
#listener.security.protocol.map=PLAINTEXT:PLAINTEXT,SSL:SSL,SASL_PLAINTEXT:SASL_PLAINTEXT,SASL_SSL:SASL_SSL

# The number of threads that the server uses for receiving requests from the network and sending responses to the netwo$
num.network.threads=3
```

- Start Zookeeper, a prerequisite for Kafka :

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

- and then launch the Kafka broker :

  **bin/kafka-server-start.sh config/server.properties**

- With Kafka running, create a topic using the appropriate Kafka command :

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

- Create producer:

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

- Create consumer:

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

To ensure functionality, test the producer by sending messages to the topic and subsequently test the consumer to ensure it can successfully retrieve these messages.

# DATA STORAGE AND INTEGRATION WITH AWS S3

- Store and manage data produced by the Kafka producer in an S3 bucket for further processing and analysis.
- Configure the Kafka consumer to ingest data from the Kafka topic and write it directly to an S3 bucket in real-time.

# USING PYTHON FOR STIMULATING THE DATASET

By simulating data with Python, we can emulate real-world scenarios, test the scalability and resilience of our Kafka infrastructure, and validate the functionality of our consumer applications in a controlled environment.

## Producer.py :

```python
from kafka import KafkaProducer
import pandas as pd
from json import dumps
from time import sleep

# Initialize Kafka producer
producer = KafkaProducer(
    bootstrap_servers=['15.206.165.54:9092'],  # Change IP and port here
    value_serializer=lambda x: dumps(x).encode('utf-8')
)

# Read the CSV file
df = pd.read_csv("indexProcessed.csv")

# Display the first few rows of  the DataFrame
print(df.head())

# Loop indefinitely
while True:
    # Sample one row randomly from the DataFrame and convert it to a dictionary
    dict_stock = df.sample(1).to_dict(orient="records")[0]

    # Send the sampled data to Kafka topic named 'demo1'
    producer.send('demo1', value=dict_stock)

    # Wait for 1 second before sending the next message
    sleep(1)

    # Flush the producer's message queue
    # producer.flush()
```

# consumer.py

```
from kafka import KafkaConsumer
from time import sleep
from json import dumps,loads
import json
from s3fs import S3FileSystem
consumer = KafkaConsumer(
    'demo1',
     bootstrap_servers=['15.206.165.54:9092'], #add your IP here
    value_deserializer=lambda x: loads(x.decode('utf-8')))
for c in consumer:
    print(c.value)
    s3 = S3FileSystem()
    for count, i in enumerate(consumer):
        with
s3.open("kafka-project-stockmarketdata-harshada/stock_market{}.json".for
mat(count), 'w') as file:
            json.dump(i.value, file)
    print(count)
    print(i.value)
```

# DATA CATALOGING AND ANALYSIS WITH AWS GLUE AND ATHENA

- Crawler Setup: Configure AWS Glue crawler to automatically discover and catalog the schema and structure of the data stored in the S3 bucket, enabling seamless data processing and analysis.

- AWS Glue Catalog: Utilize AWS Glue Catalog as a centralized metadata repository to store table definitions, schemas, and other metadata for the data stored in S3.

- Athena Integration: Integrate Athena with the Glue Catalog to execute ad-hoc SQL queries directly on the data stored in S3, enabling interactive querying and analysis without the need for complex data preparation or infrastructure setup.

# ATHENA QUERY RESULTS

Data

Data source

AwsDataCatalog

Database

kafka-stockmarket

Tables and views

Create

Filter tables and views

Tables (1)

kafka_project_stockmarketdata_harshada

Views (0)

Query 1 | Query 2 | Query 3

```
1 SELECT * FROM "kafka-stockmarket"."kafka_project_stockmarketdata_harshada
```

SQL Ln 1, Col 1

Run again | Explain | Cancel | Clear | Create

Completed Time in queue: 54 ms Run time: 597 ms Data scanned: 5.94 KB

Results (10)

Copy | Download results

Search rows

| # | index | date | open | high | low | close | adj close | volume |
|---|---|---|---|---|---|---|---|---|
| 1 | NYA | 1978-11-22 | 562.950012 | 562.950012 | 562.950012 | 562.950012 | 562.950012 | 0.0 |
| 2 | NSEI | 2008-08-21 | 4416.200195 | 4418.549805 | 4271.299805 | 4283.850098 | 4283.850098 | 0.0 |
| 3 | IXIC | 1988-03-22 | 379.100006 | 379.899994 | 378.600006 | 379.799988 | 379.799988 | 1.2165E8 |
| 4 | IXIC | 1983-12-02 | 283.910004 | 283.910004 | 283.910004 | 283.910004 | 283.910004 | 0.0 |
| 5 | GSPTSE | 2003-10-09 | 7593.399902 | 7620.0 | 7569.100098 | 7604.5 | 7604.5 | 1.419549E10 |
| 6 | N225 | 1980-04-16 | 6770.370117 | 6770.370117 | 6770.370117 | 6770.370117 | 6770.370117 | 0.0 |
| 7 | SSMI | 1996-03-28 | 3670.699951 | 3675.800049 | 3621.899902 | 3640.300049 | 3640.300049 | 0.0 |
| 8 | N225 | 1971-08-18 | 2328.280029 | 2328.280029 | 2328.280029 | 2328.280029 | 2328.280029 | 0.0 |
| 9 | N225 | 1985-01-30 | 11960.62988 | 11960.62988 | 11960.62988 | 11960.62988 | 11960.62988 | 0.0 |
| 10 | GSPTSE | 2018-01-09 | 16351.40039 | 16358.2002 | 16309.79981 | 16319.2002 | 16319.2002 | 2.369117E10 |