# Stock Market Real-Time Data Analysis Using Kafka

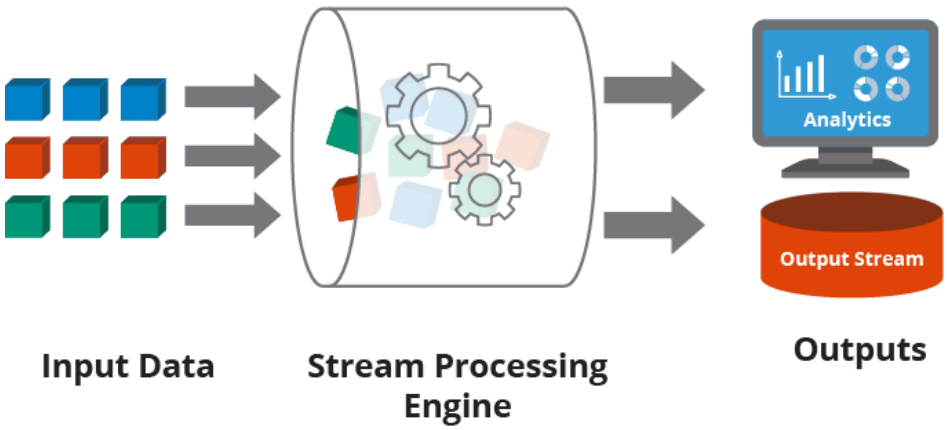
End-To-End Data Engineering Project

By- Subhesh Kumar

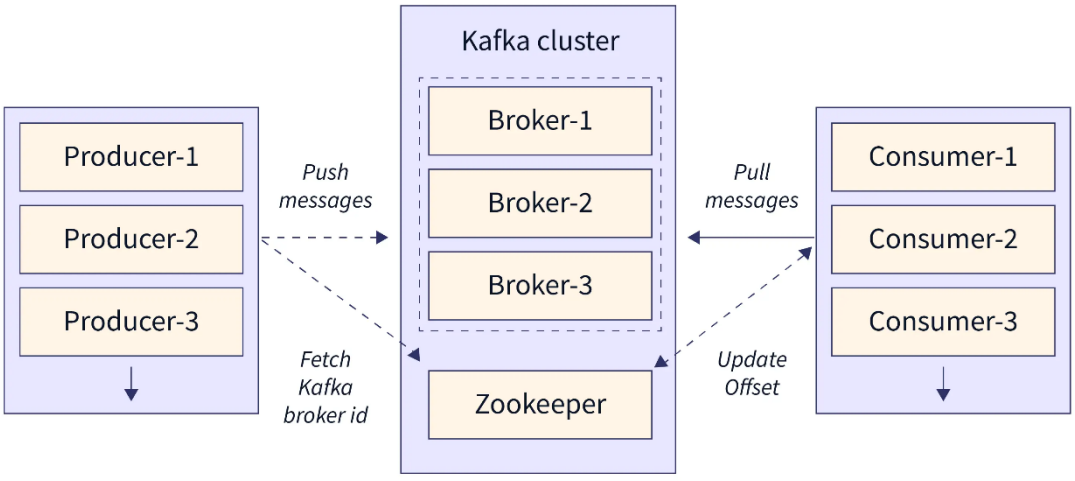
## Kafka

Analyzing stock market data in real-time using Apache Kafka can provide valuable insights for traders, investors, and financial analysts. Apache Kafka is a distributed streaming platform that can handle large volumes of data and is well-suited for real-time data streaming and analysis.

Kafka is built around the concept of a distributed commit log. It allows you to publish and subscribe to streams of records, store these records in a fault-tolerant and scalable manner, and process them in real-time or batch modes.




**Kafka Architecture:**



**Input Data Stream Processing:**

Input data stream processing in the context of Kafka involves ingesting, storing, and managing real-time data streams from various sources. Here's how it works:

**Producers**: Data producers are responsible for sending data records (messages) to Kafka topics. These producers can be applications, devices, or systems that generate data.

**Topics:** Topics are logical channels or categories where data records are published. Producers publish data records to specific topics.

**Brokers:** Kafka brokers form the Kafka cluster and store the data records in a distributed manner. They manage data replication, distribution, and fault tolerance.

**Consumers:** Data consumers subscribe to specific topics and read data records from them. Consumers process the data records, perform analyses, and take appropriate actions.

**Partitions:** Each topic is divided into partitions, which are the basic units of parallelism and distribution. Each partition can be hosted on a different broker.

**Offsets:** Each data record in a partition is assigned a unique identifier called an offset. Offsets enable consumers to keep track of the records they have processed.

**Output Data Stream Processing:**

Output data stream processing involves consuming the data streams from Kafka, performing various operations on the data, and potentially producing new data streams or results. Here's how it works:

**Consumers:** Consumers subscribe to Kafka topics and retrieve data records from the partitions. These consumers can be part of a stream processing application.

**Stream Processing:** Stream processing applications consume data records, apply transformations, filters, aggregations, and other operations in real-time or near-real-time. This can include tasks like calculating moving averages, identifying trends, filtering out noise, or performing more complex analytics.

**Result Generation:** The output of stream processing can lead to different outcomes. It can include generating alerts, updating dashboards, writing data to databases or data warehouses, and even producing new data streams for downstream consumers.

**Scaling and Parallelism:** Kafka allows you to scale your stream processing applications by adding more consumers that process different partitions concurrently. This ensures that the processing workload is distributed effectively.

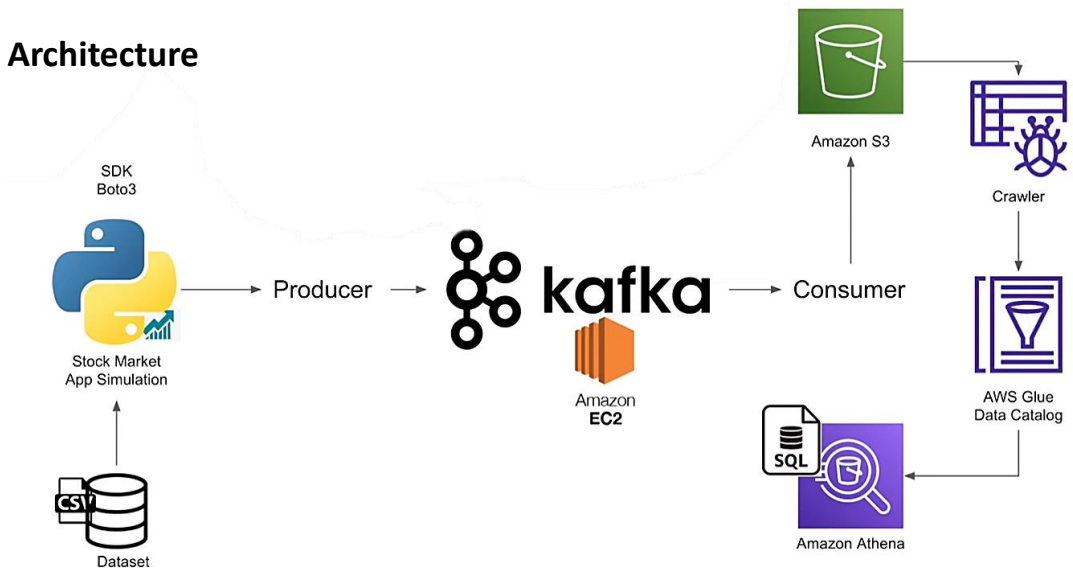
**Stateful Processing:** Some stream processing tasks require maintaining state, such as sessionization or counting occurrences. Kafka Streams, a stream processing library provided by Kafka, offers stateful processing capabilities.

**Output to External Systems:** The results of stream processing can be sent to external systems for visualization, reporting, further analysis, or triggering actions.

**Zookeeper:**

ZooKeeper is a critical role in a Kafka cluster by providing distributed coordination and synchronization services. It maintains the cluster's metadata, manages leader elections, and enables consumers to track their consumption progress.

## Project Architecture




**Technology Used**

1. **Programming Language - Python**
2. **Amazon Web Service (AWS)**
    - **i) S3 (Simple Storage Service)**
    - **i) Athena**
    - **ii) Glue Crawler**
    - **iii) Glue Catalog**
    - **iv) EC2**
    - **v) Apache Kafka**

**Steps to follow:**

1. **Get Python latest version installed in the PC.**
2. **Create a AWS Account.**
    2.1. After having AWS Account Go to Console -> EC2 -> Under Resources -> Instances
    2.2. Instances -> Launch Instances -> Give a name (kafka-stock-market-analysis)
    2.3. Application and OS Images (Amazon Machine Image) -> Amazon (Linux)
    2.4. Instance Type -> t2.micro (free tier eligible)
    2.5. Key Pair Login ( Create New Key Pair) -> give name and leave other default-> Key Pair will downloaded.
    2.6. Keep the rest as it is. Click Launch instances
    2.7. Click on Instance Id -> Connect -> SSH Client -> Copy the command -> Run it on Terminal

2.8. Before it run cd (location of the project folder with the key file) -> ls-> paste the command
2.9. Yes -> Amazon Linux 2023 will be added and connected to EC2 Instance

3. Paste -> **wget https://downloads.apache.org/kafka/3.5.1/kafka_2.12-3.5.1.tgz** on the EC2 Machine Terminal to install Kafka on the server
   3.1. Paste tar -xvf kafka-3.5.1-src.tgz

**The command tar -xvf kafka-3.5.1-src.tgz is used to extract the contents of a compressed tarball file**

4. **Install Java**
   4.1. java -version
   4.2. sudo yum install java
   4.3. java -version
   4.4. cd kafka-3.5.1-src

5. Start Zoo-keeper:
   **bin/zookeeper-server-start.sh config/zookeeper.properties**

   The command **bin/zookeeper-server-start.sh config/zookeeper.properties** is used to start the ZooKeeper server for an Apache Kafka installation.

6. Start Kafka-server:

Duplicate the session & enter in a new console --

**export KAFKA_HEAP_OPTS="-Xmx256M -Xms128M"**

**cd kafka_2.12-3.5.1**

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

It is pointing to private server , change server.properties so that it can run in public IP

To do this , you can follow any of the 2 approaches shared below –

First stop the terminal by Ctrl+C

Do a "**sudo nano config/server.properties**" - change ADVERTISED_LISTENERS to public IP of the EC2 instance in one of the terminal.

It is pointing to private server , change server.properties so that it can run in public IP

To do this , you can follow any of the 2 approaches shared belwo --

Do a "sudo nano config/server.properties" - change ADVERTISED_LISTENERS to public IP of the EC2 instance.

7. We need to provide access Security to the our local machine

7.1. Go to Security in the EC2 instance -> Security Groups -> Edit Inbound rules -> Add rule ->Type( All traffic) and Source (Anywhere)
7.2. Add one more -> Type(All traffic) and Source (Anywhere)

I edited the Inbound rules of the Security just for learning purposes **It is not a good practice, we should never allow all the traffic from everywhere in the world, it is not a secured way.

8. Create the topic:

Duplicate the session & enter in a new console --

cd kafka_2.12-3.5.1

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

9. Start Producer:

---

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

10. Start Consumer:

---

Duplicate the session & enter in a new console --

cd kafka_2.12-3.5.1

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

Now Comes to Jupyter for Stock Market App Simulation

**Jupyter**: Jupyter Notebook is an interactive web-based environment that allows users to create and share documents containing live code, visualizations, and narrative text.

11. **Kafka Consumer Code:**

```
from kafka import KafkaConsumer
from time import sleep
from json import dumps,loads
import json
```

```
from s3fs import S3FileSystem
consumer = KafkaConsumer(
    'demo_test',
     bootstrap_servers=[':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("s3://kafka-stock-market-tutorial-youtube-
darshil/stock_market_{}.json".format(count), 'w') as file:
        json.dump(i.value, file)
```

12. **For Kafka Producer:**

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

producer = KafkaProducer(bootstrap_servers=[':9092'], #change ip here
                         value_serializer=lambda x:
                         dumps(x).encode('utf-8'))

producer.send('demo_test', value={'surnasdasdame':'parasdasdmar'})

df = pd.read_csv("data/indexProcessed.csv")

df.head()

while True:
    dict_stock = df.sample(1).to_dict(orient="records")[0]
    producer.send('demo_test', value=dict_stock)
    sleep(1)

producer.flush() #clear data from kafka server
```

**Dataset:** Choose any Good Dataset from e.g, (Kaggle) related to Stock Market.

13) Amazon S3 : Amazon Simple Storage Service (Amazon S3) is an object storage service that offers industry-leading scalability, data availability, security, and performance. You can use Amazon S3 to store and retrieve any amount of data at any time, from anywhere.
    a) Create S3 Bucket -> Give Unique name -> Keep rest default -> Click Create Bucket
    b) We will next upload our data from our Python code to the S3 Bucket we have created.
    c) Before that we need to install package called s3fs( pip install s3fs)

Now to extract the data and convert it into JSON format within a loop using while true. The user can see the data moving in real-time from one source to another source. We have also done how to remove old data and send new data using the **flush command**. Finally, the user is directed to go to their AWS console and create a new S3 bucket with a unique name to store their project data.

We need to configure AWS in the local machine by going to IAM and clicking on add user, giving a name like Shell Admin Account, and allowing programmatic access.

Then we will download one CSA file which has an access key ID and a secret key ID. After saving the files properly, the user goes to their terminal, installs AWS CLI, and configures it by providing the access key ID, secret key ID, and default region. Once this is done, the user installs s3fs, creates an object s3fs, assigns numbers at the end of each file to give them unique names, and uses count and enumerate in the Python function to get the count and the actual iterator on the consumer.

**AWS CLI** : The AWS Command Line Interface (AWS CLI) is a unified tool to manage your AWS services. With just one tool to download and configure, you can control multiple AWS services from the command line and automate them through scripts.

To Configure CLI –

~ aws configure

~ Give all the information like Access key and secret key, Locations, etc.

14) Create AWS Crawler -> Give another unique name -> Click on Next -> Data Source Configuration -> Click on Not yet -> Add Data Source -> Select S3 Path -> Select S3 Bucket we have created ( to run crawler on the entire S3 Bucket) -> Add S3 data source -> Click on Next -> IAM role -> Select the role we created -> Nect -> Set Output and Scheduling -> Choose database -> Create database -> give a name to database -> Click on create -> Choose database created -> Click on Next -> Create crawler -> After creation Run the crawler

IAM role – Glue need access to connect with S3 so it need IAM role. AWS Identity and Access Management (IAM) roles are entities you create and assign specific permissions to that allow trusted identities such as workforce identities and applications to perform actions in AWS.

**Don't make any Access key or Secret key Public anywhere**

Go to IAM -> Click on role -> Create role -> Trusted Entity Type (AWS Service) -> Use case -> Browse for Glue and select it -> Next -> Permission Policies -> Give access to Administrator Access -> give a name -> Click on create role.

** A crawler accesses your data store, extracts metadata, and creates table definitions in the AWS Glue Data Catalog. The Crawlers pane in the AWS Glue console lists all the crawlers that you create. The list displays status and metrics from the last run of your crawler.

Athena : Athena provides a simplified, flexible way to analyze petabytes of data where it lives. Analyze data or build applications from an Amazon Simple Storage Service (S3) data lake and 30 data sources, including on-premises data sources or other cloud systems using SQL or Python.

Open Athena -> Refresh the data -> Select AWSDataCatalog in (database source) -> Choose the database -> One table will appear -> Click on Preview from the 3 dots -> Run some SQL Query -> It will appear in Results.

**In the case, If an error appears like ( not able to find the target location ) -> Click on Settings -> Click on Manage -> Select a S3 Path -> Create a temporary or extra S3 Bucket for athena queries -> Click Save.

For Real Time Data Analysis we need to send some data in real time.

To do this we need to add some delay ( so we don't overload the server) -> Go to Jupyter -> KafkaProducer code -> add delay of 1s -> sleep(1) -> in the while loop -> and we already have sleep package( from time import sleep) -> Run the code -> Sending data on to the Producer -> Run the code on KafkaConsumer -> last code -> to consuming data and uploading the data onto our S3 -> Go to the S3 Bucket we can see the rows of the data consumed -> Go to Athena -> Run Count Query on it ( to see the no. of rows there) -> if you run again multiple time we can see the data changes in real time .

So we are sending the data in real time and S3 bucket getting the data and we can keep quering it on Athena on real time basis.

Thank You

** I spearheaded this captivating project focused on Real-Time Data Analysis within the Stock Market, leveraging the robust capabilities of Kafka and its architecture in conjunction with Zookeeper. The project's architecture encompassed a comprehensive pipeline, commencing with a Stock Dataset, followed by a Stock Market application simulation implemented in Python. This data was then seamlessly ingested into Kafka as a producer and later consumed via Amazon EC2 instances. The data journey continued with Kafka consumers directing data flow to Amazon S3, where a custom crawler was employed for further data processing. The subsequent stages involved AWS Glue Data Catalog for cataloging and Amazon Athena for efficient query execution. This end-to-end data engineering project not only enhanced my proficiency in utilizing cutting-edge data technologies but also showcased my ability to design and manage complex data pipelines for real-time analytics. **