End-To-End Data Engineering Project

Designing a Stock market data processing engine using Kafka

1. Objective:

In this project, I took stock market data and used Python to produce the data, which was then pushed into a Kafka cluster. After consumption, the data was stored in Amazon S3. Subsequently, I crawled the data to build a Glue catalog and analyzed it using Amazon Athena with SQL.

2. Pre-requisite:

Python installed on Laptop or Jupyter notebook, AWS account

3. Skills used:

Python, Amazon Web Services (AWS free tier account), Apache Kafka, Glue, Athena, SQL

4. Theory:

In our daily lives, we heavily rely on real-time streaming applications like Google Maps, Amazon, and Uber. For instance, when placing an order on Amazon, we receive timely notifications for order confirmation, dispatch, and delivery. Similarly, in the banking sector, real-time notifications are critical, especially in detecting fraudulent transactions promptly to prevent financial losses. This underscores the vital role of real-time analytics. Apache Kafka serves as a cornerstone in facilitating these real-time data pipelines, ensuring seamless and efficient data processing and analysis, thereby enhancing user experience and safeguarding against potential risks.

Apache Kafka is a distributed event store and stream-processing platform.

Components of Kafka:

a. Producer:

Data is produced here, for example data coming from sensors, financial transactions, orders from a shipping company, tourism booking etc. All of this data generated is sent to Kafka server. Can be written in C/C++, Java, Python .NET etc

b. Kafka Broker/cluster:

A single member server of a Kafka cluster. One single node or a server is called a broker interms of Kafka. We have multiple brokers in a cluster. In this project we have only one broker. Brokers can be assumes as EC2 machines.

c. Consumer:

Until now we had producer which produces data, then a kafka broker where we write the data. Consumer consumes the data from the broker

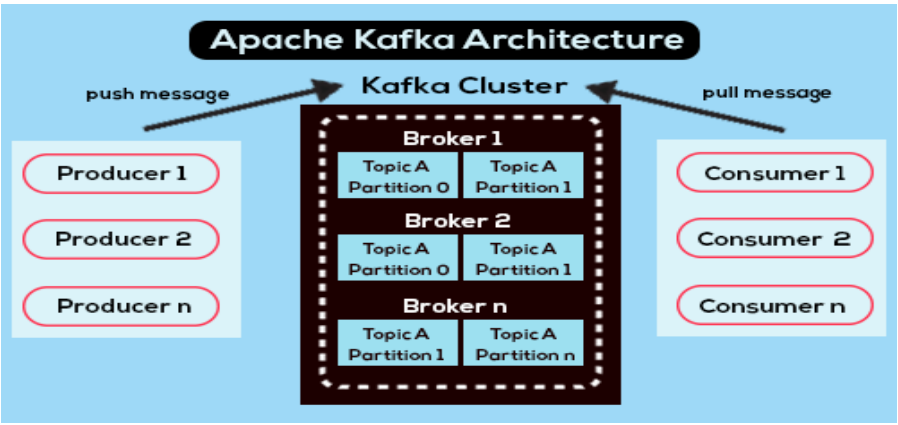


Fig 1: Apache Kafka Architecture

We can also have multiple producers, brokers and consumers too.

d. Zookeeper

Zookeeper is a resource manager as a part of Hadoop ecosystem. In terms of Kafka, Zookeeper makes sure that all the brokers are running properly. If one broker fails, zookeeper notifies all of the other brokers about the situation and makes sure that the data is in state. Additional uses are cluster management, failure detection and disaster recovery and provides distributed synchronization.

e. Topics

Logical buckets inside a Kafka broker. It categorizes messages into groups. We can have separate topics say for example order created in Amazon, it will get sent to that topic and another topic for return order, ship order etc. If I am interested only in return order, I can only focus on that particular topic. Inside each topic we have partition. A partition is like a log that stores data. Partition can be done on dates, customer ID, order ID etc.

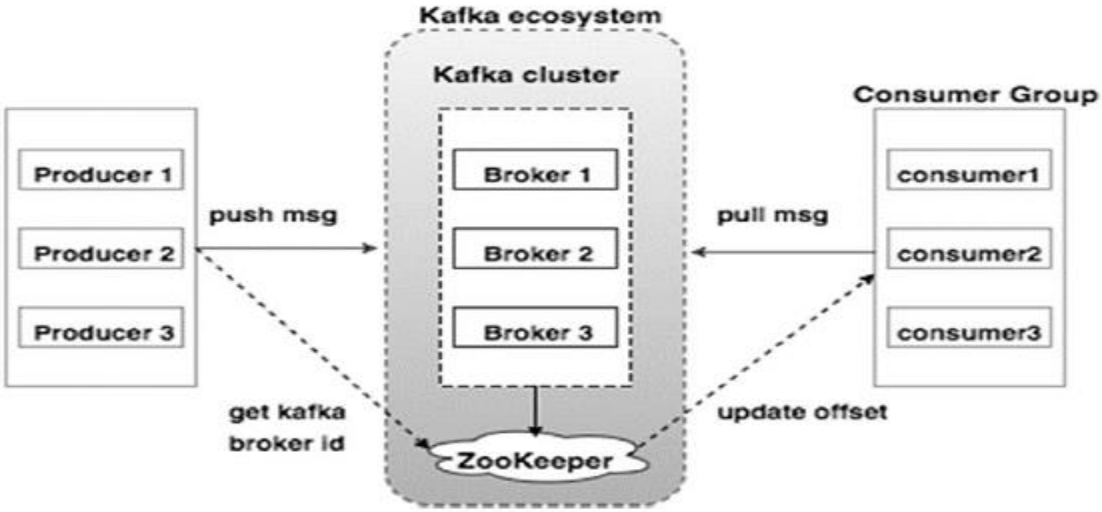


Fig 2: Kafka ecosystem

5. Architecture diagram of this project:

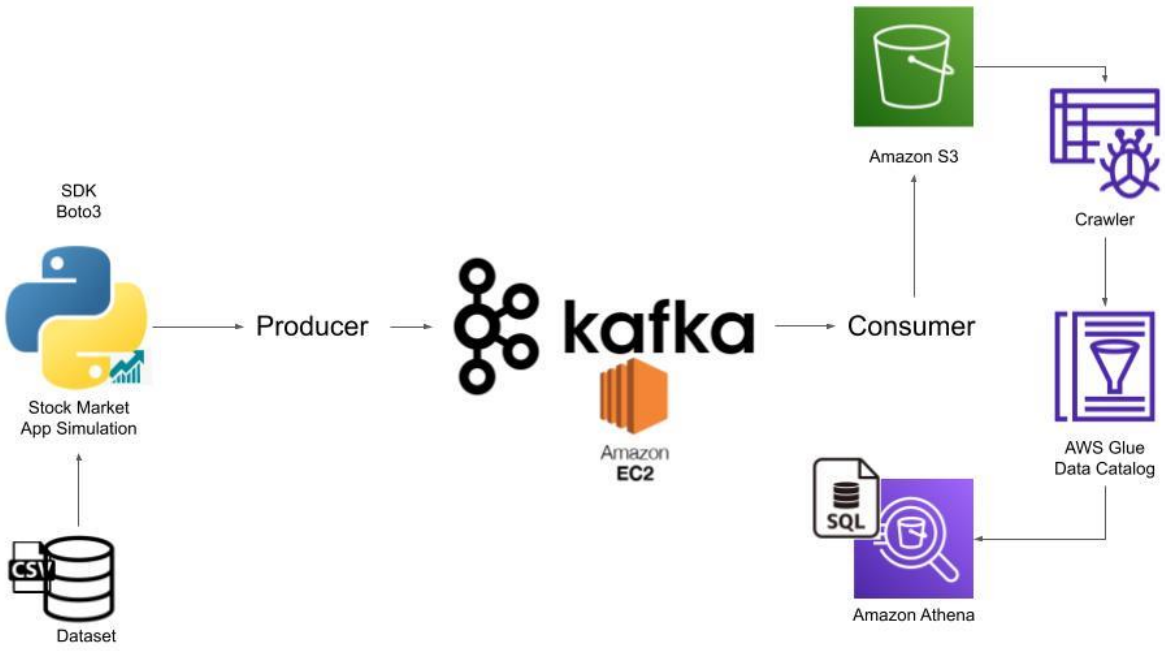


Fig 3: Project Architecture

Consider any dataset of your interest, in this case I used stock market dataset. Use python to simulate or read the data. Then write a producer code that pushes the event to kafka broker. Then we have kafka broker where kafka is installed on EC2 machine. Next we write a consumer code that will consume the data and store them on Amazon S3. S3 is an object storage so you can store any kind of files such as audio, video, csv, orc avro etc. Then we will run a Glue crawler, which is a service available on AWS. You can crawl the schema out of different files and build a catalog and then you can directly query data on top of those files. We will do these things on Amazon Athena.

Important steps:

Go to S3 on AWS console and create a bucket name as you prefer. Make sure it's a unique name.

Configure AWS on your local machine. Go to IAM, create a new user and get the access and secret key. Install AWS CLI, and go to cmd. Type aws configure and provide access and secret key id. Once configured, you can now send data from local machine to Amazon S3.

We have data into S3, now we need to write a crawler.

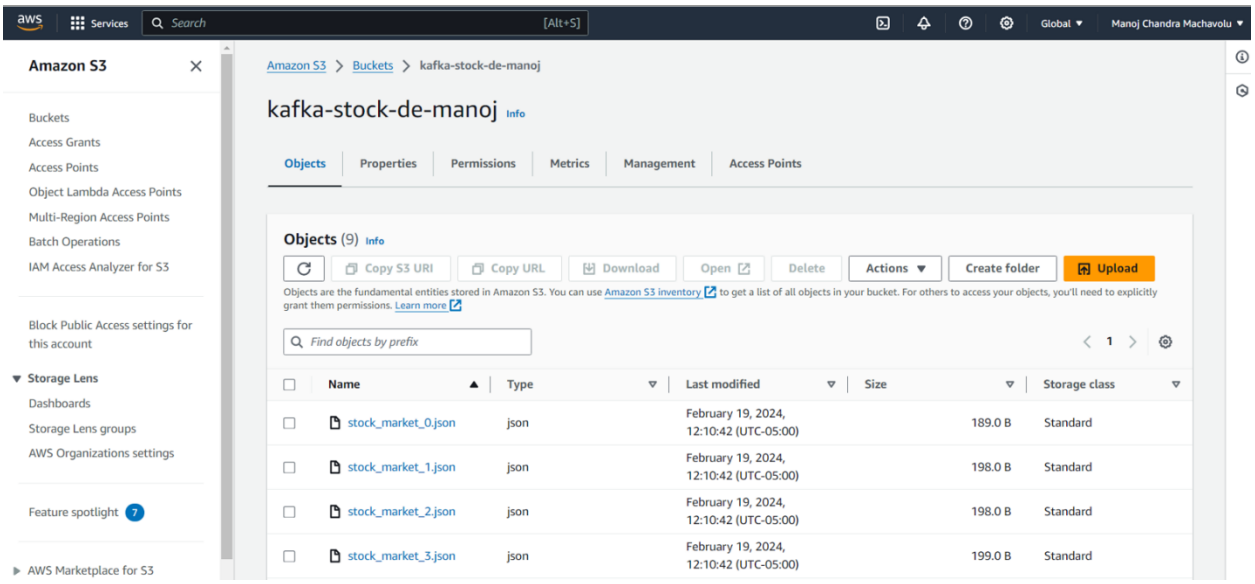


Fig 4: Data files stored on S3 bucket

AWS Glue Crawler is a component of AWS Glue that automatically discovers the schema of your data and stores the associated metadata in the AWS Glue Data Catalog.

When you have data stored in various data sources such as Amazon S3, databases, or data warehouses, AWS Glue Crawler can automatically scan these data sources, infer the schema of your data, and create metadata tables in the AWS Glue Data Catalog. This metadata includes information about the data's structure, format, and location. By cataloging your data, AWS Glue makes it easier to query and analyze your data using other AWS services like Amazon Athena.

Amazon Athena is an interactive query service that allows you to analyze data using standard SQL. Athena does not require you to set up or manage any infrastructure; you simply point it to your data stored in Amazon S3, define the schema (or use the schema discovered by AWS Glue Crawler), and start querying using SQL.

You can preview the stock market data table in AWS Athena to make sure that data is rightly imported. Make sure to provide a target location/S3 path to store queries.

To see the action in real time, we can send some data in real time by adding a delay of 1 second. This avoids server breakdown. Now run the producer code which sends the data to producer with a delay of 1 second. The consumer consumes this data and sends to S3 bucket. In S3, the data is added slowly i.e the number of rows increases gradually which means we get data in S3 bucket in real time. We can query this data in real time using AWS Athena.

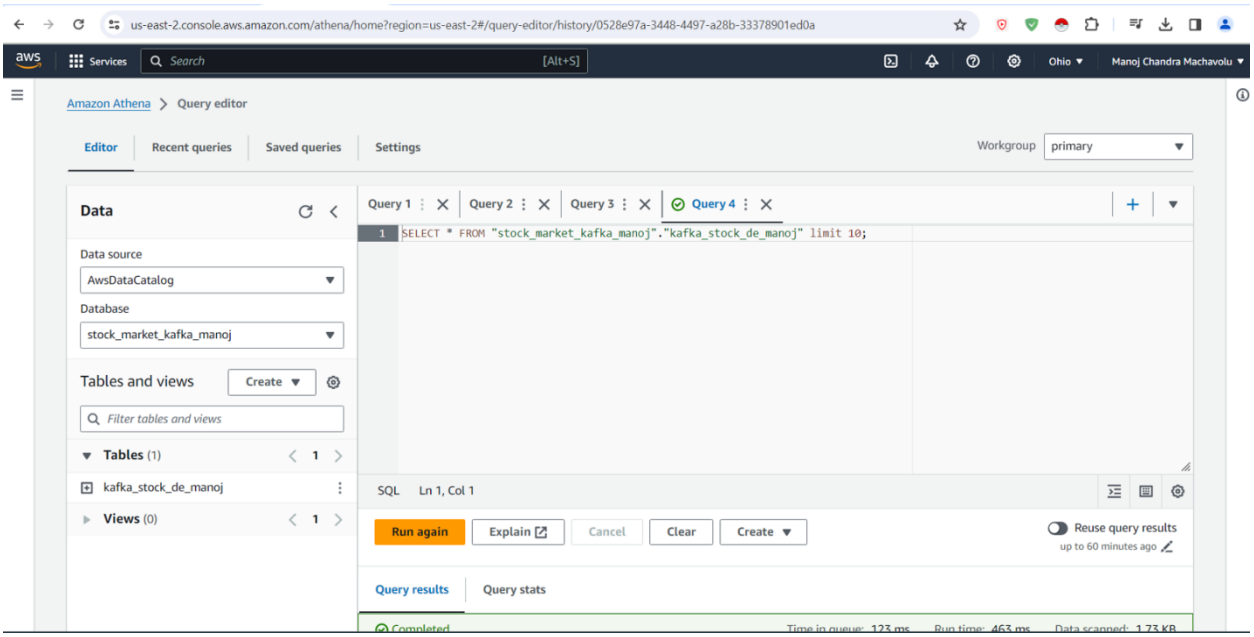
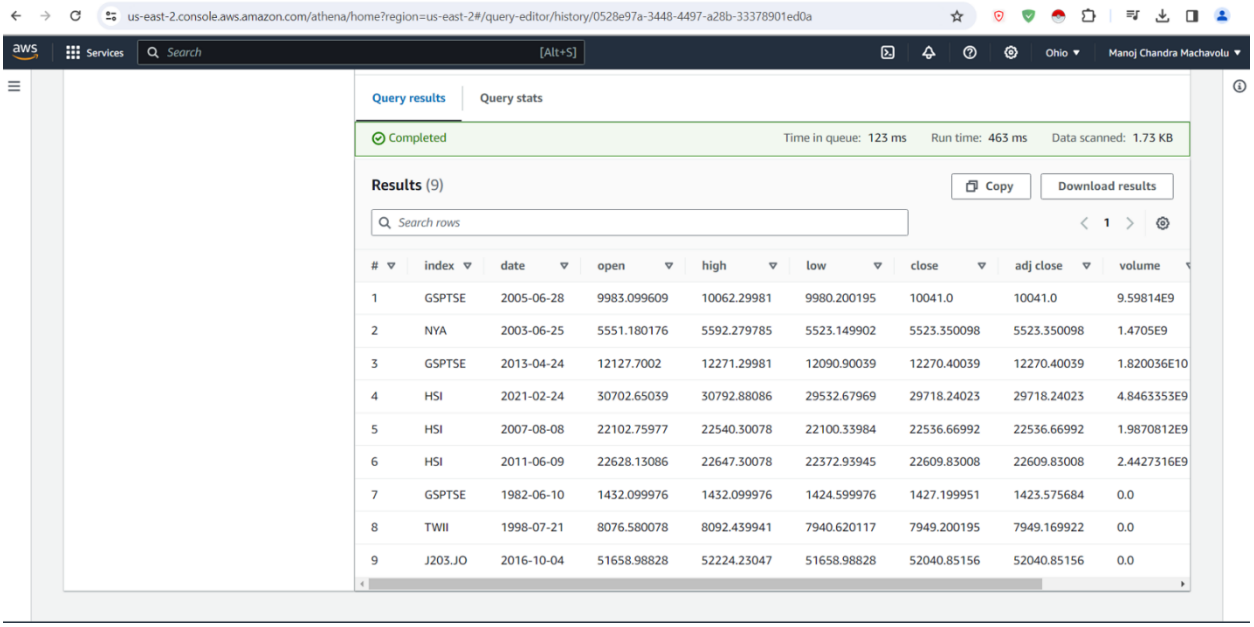


Fig 5: Query editor on Amazon Athena



#	index	date	open	high	low	close	adj close	volume
1	GSPTSE	2005-06-28	9983.099609	10062.29981	9980.200195	10041.0	10041.0	9.59814E9
2	NYA	2003-06-25	5551.180176	5592.279785	5523.149902	5523.350098	5523.350098	1.4705E9
3	GSPTSE	2013-04-24	12127.7002	12271.29981	12090.90039	12270.40039	12270.40039	1.820036E10
4	HSI	2021-02-24	30702.65039	30792.88086	29532.67969	29718.24023	29718.24023	4.8463353E9
5	HSI	2007-08-08	22102.75977	22540.30078	22100.33984	22536.66992	22536.66992	1.9870812E9
6	HSI	2011-06-09	22628.13086	22647.30078	22372.93945	22609.83008	22609.83008	2.4427316E9
7	GSPTSE	1982-06-10	1432.099976	1432.099976	1424.599976	1427.199951	1423.575684	0.0
8	TWII	1998-07-21	8076.580078	8092.439941	7940.620117	7949.200195	7949.169922	0.0
9	J203.JO	2016-10-04	51658.98828	52224.23047	51658.98828	52040.85156	52040.85156	0.0

Fig 6: Results

Thus the entire architecture diagram is covered by executing this project. This project was executed by referring to Darshil Parmar's youtube channel. You can find his video here: https://www.youtube.com/watch?v=KerNf0NANMo