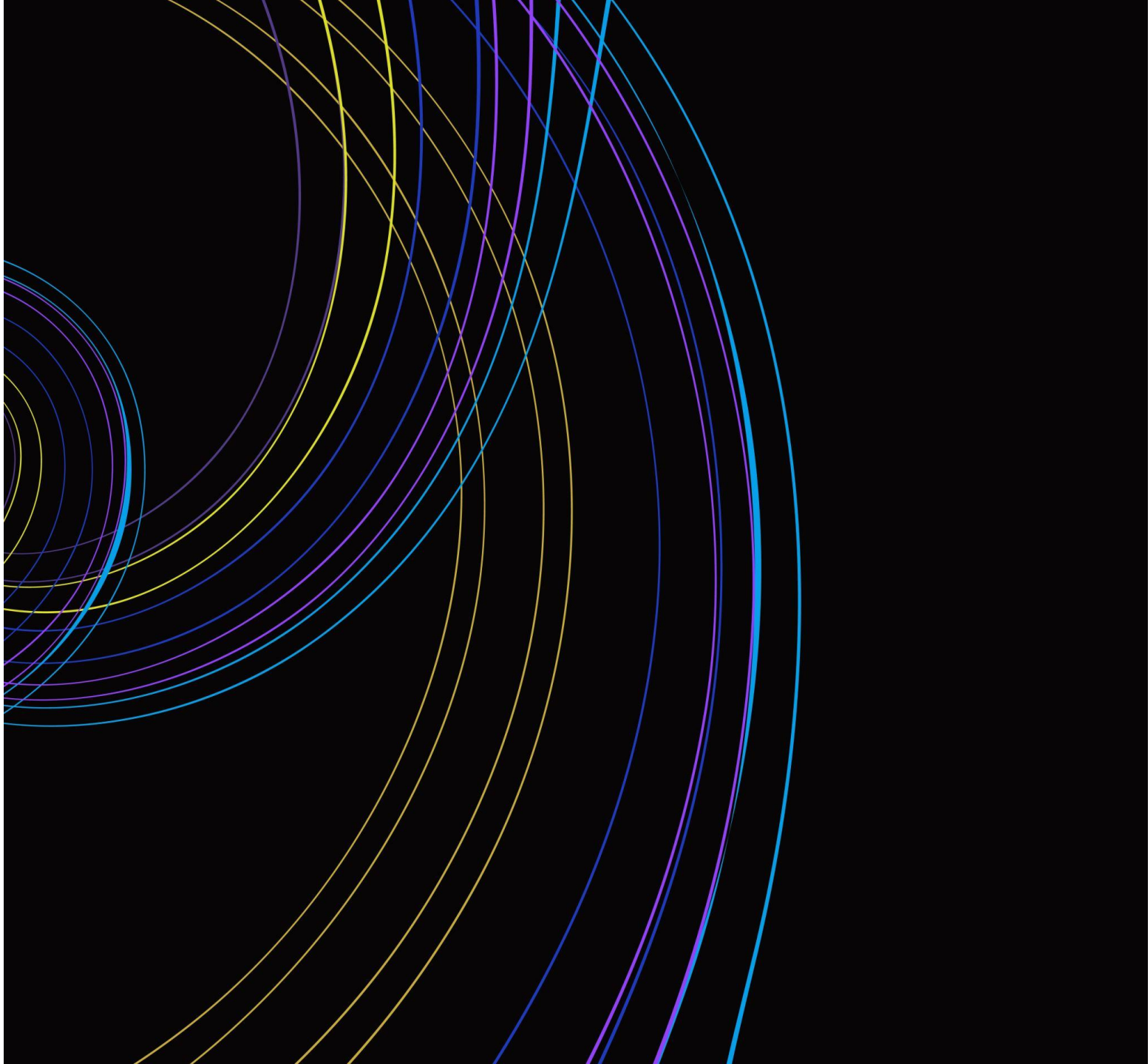


STREAMING & PROCESSING WITH KAFKA

~Siddhant Prakash More



ARCHITECTURE DIAGRAM

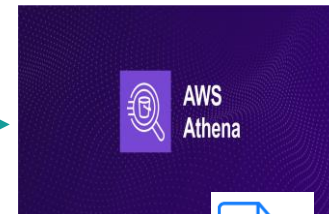
Producer: Stock Market
App Simulator using
Python



Amazon EC2



Data Catalog



Data



PRODUCER:

```
[2]: import pandas as pd
      from kafka import KafkaProducer
      from time import sleep
      from json import dumps
      import json

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

[4]: producer.send('demo_testing2', value={'surname': 'more'})

[4]: <kafka.producer.future.FutureRecordMetadata at 0x29628c84c10>

[5]: df = pd.read_csv("indexProcessed.csv")

[6]: df.head()

[6]:
```

	Index	Date	Open	High	Low	Close	Adj Close	Volume
0	HSI	1986-12-31	2568.300049	2568.300049	2568.300049	2568.300049	2568.300049	0.0
1	HSI	1987-01-02	2540.100098	2540.100098	2540.100098	2540.100098	2540.100098	0.0
2	HSI	1987-01-05	2552.399902	2552.399902	2552.399902	2552.399902	2552.399902	0.0
3	HSI	1987-01-06	2583.899902	2583.899902	2583.899902	2583.899902	2583.899902	0.0
4	HSI	1987-01-07	2607.100098	2607.100098	2607.100098	2607.100098	2607.100098	0.0

```
[7]: while True:
      dict_stock = df.sample(1).to_dict(orient="records")[0]
      producer.send('demo_testing2', value=dict_stock)
      sleep(1)
```

- Simulates real-time stock market data using a preprocessed CSV file
- Reads data using **Pandas** and converts it to JSON format.
- Sends messages to Kafka topic (demo_testing2) using KafkaProducer
- Uses a while True loop to continuously stream random stock records at 1-second intervals.
- Acts as a real-time data generator running on an EC2 instance.

CONSUMER:

```
[1]: from kafka import KafkaConsumer
    from time import sleep
    from json import dumps, loads
    import json
    from s3fs import S3FileSystem

•[2]: consumer = KafkaConsumer(
    'demo_testing2',
    bootstrap_servers=[''], #add your IP here
    value_deserializer=lambda x: loads(x.decode('utf-8')))

[3]: # for c in consumer:
    #     print(c.value)

[4]: s3 = S3FileSystem()

[5]: for count, i in enumerate(consumer):
    with s3.open("kafka-stock-market-project-sid/stock_market_{}.json".format(count), 'w') as file:
        json.dump(i.value, file)
```

- Listens to the Kafka topic demo_testing2 using KafkaConsumer.
- Deserializes incoming JSON messages using UTF-8 encoding.
- Uses s3fs with boto3 under the hood to interact with Amazon S3.
- For each consumed message, generates a new .json file.
- Uploads the file to the S3 bucket kafka-stock-market-project-sid.
- Runs on an EC2 instance to stay close to Kafka and minimize latency.

Getting started

ETL jobs

Visual ETL

Notebooks

Job run monitoring

Data Catalog tables

Data connections

Workflows (orchestration)

Zero-ETL integrations

▼ Data Catalog

Databases

Tables

Stream schema registries

Schemas

Connections

Crawlers

Classifiers

▼ Data integration and ETL

Zero-ETL integrations

ETL jobs

Visual ETL

Last updated

March 27, 2025 at 02:37:36

► Advanced properties

Schema

Partitions

Indexes

Column statistics - new

Schema (9)

View and manage the table schema.

Filter schemas

#	Column name	Data type	Partition key	Comment
1	index	string	-	-
2	date	string	-	-
3	open	double	-	-
4	high	double	-	-
5	low	double	-	-
6	close	double	-	-
7	adj close	double	-	-
8	volume	double	-	-
9	closeud	double	-	-

Amazon Athena

Query editor

kafka_stock_market_project_sid

index

date

open

high

low

close

adj close

volume

closeud

Views (0)

Run again

Explain

Cancel

Clear

Create

Query results

Query stats

Completed

Time in queue: 67 ms

Run time: 983 ms

Data scanned: 65.04 KB

Results (347)

Copy

Download results CSV

Search rows

#	Index	date	open	high	low	close	adj close	volume
1								
2	N225	2014-11-18	17188.83984	17356.75977	17186.5	17344.06055	17344.06055	1.581E8
3	000001.SS	2014-09-23	2289.091064	2311.524902	2289.023926	2309.718018	2309.718018	156900.0
4	GSPTSE	1995-01-25	4076.800049	4113.0	4076.800049	4106.399902	4106.399902	5.629E7
5	NVA	1989-01-04	1660.709961	1660.709961	1660.709961	1660.709961	1660.709961	0.0
6	N225	1998-08-06	16048.15039	16075.71973	15834.76953	15876.21973	15876.21973	0.0
7	N100	2019-04-03	1063.319946	1065.48999	1061.920044	1065.359985	1065.359985	1.881211E8
8	SSMI	2015-10-27	8842.839844	8894.379883	8806.349609	8849.919922	8849.919922	4.00302E7

kafka-stock-market-project-sid

Objects

Metadata

Properties

Permissions

Metrics

Management

Access Points

Objects (490)

Copy S3 URI

Copy URL

Download

Open

Delete

Actions

Create folder

Upload

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](#)

Find objects by prefix

<input type="checkbox"/>	Name	Type	Last modified	Size	Storage class
<input type="checkbox"/>	stock_market_0.json	json	March 26, 2025, 21:34:18 (UTC-05:00)	19.0 B	Standard
<input type="checkbox"/>	stock_market_1.json	json	March 26, 2025, 21:34:18 (UTC-05:00)	197.0 B	Standard
<input type="checkbox"/>	stock_market_10.json	json	March 26, 2025, 21:34:26 (UTC-05:00)	189.0 B	Standard
<input type="checkbox"/>	stock_market_100.json	json	March 26, 2025, 21:35:57 (UTC-05:00)	197.0 B	Standard
<input type="checkbox"/>	stock_market_101.json	json	March 26, 2025, 21:35:58 (UTC-05:00)	191.0 B	Standard
<input type="checkbox"/>	stock_market_102.json	json	March 26, 2025, 21:35:59 (UTC-05:00)	200.0 B	Standard

DATA STORAGE, CATALOGING & QUERYING

CONCLUSION:

Built a real-time data streaming pipeline using Apache Kafka on AWS EC2.

Simulated stock market data using a Python producer.

Streamed data into Kafka topics, then consumed and stored it in Amazon S3.

Automated metadata cataloging with AWS Glue Crawlers.

Enabled serverless querying and analysis using Amazon Athena.

Demonstrated how Kafka integrates with AWS for scalable and real-time data processing.