Lambton College

REAL-TIME STOCK PRICE ANALYSIS

PYSPARK AND KAFKA STREAMING

PROJECT PRESENTATION

This project builds an end-to-end real-time data pipeline to ingest, process, and analyze live stock prices using Kafka and PySpark.

Presenter : **Team Everest**

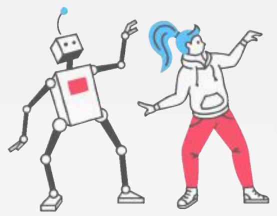
Team Members:
Bharat, Satish, Bikash, Samir, Arwind

BDM 3603: Big Data Framework   Lambton College   10 Dec 2023

Project Overview

- Introduction to Real-Time Streaming
- Introduction to Apache Kafka
- Data Ingestion & Processing Pipeline
- Set up Kafka for streaming stock data
- Kafka Producer Setup
- Kafka Consumer Setup
- Set up PySpark Streaming in Databricks
- Define schema for stock price data
- Visualization and Streaming Data Sink
- Final Dashboard Visualization
- Libraries and Modules Used
- Conclusion
- Challenges of the Project
- Project Timeline
- References

Introduction to Real-Time Streaming



Real-Time Stream Engines: They processes continuous streams of data as they arrive, enabling immediate insights and actions.

Examples: Amazon Kinesis (Case: I) & Apache Kafka (Case: II)

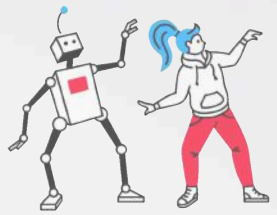
Use Case I : E-Commerce Order Tracking

- **Data Ingestion:** Kinesis streams data from user activities like clicks and purchases.
- **Processing:** Real-time analytics for personalized recommendations and inventory updates.
- **Outcome:** Faster order tracking, instant alerts for delivery issues, and better customer experience.

Use Case II : Stock Price Monitoring

- **Process:**
  - Kafka receives live stock prices from exchanges.
  - Processes data to calculate moving averages and price volatility.
  - Generates real-time alerts for price fluctuations.
- **Outcome:** Immediate visualization and decision-making for traders.

Introduction to Apache Kafka



Apache Kafka is a distributed streaming platform used for building real-time data pipelines and streaming applications. It handles large volumes of data in real time, offering fault tolerance, scalability, and high throughput.

Kafka Components:

- **Producer:** Sends data (messages) to Kafka topics.
- **Consumer:** Reads data from Kafka topics.
- **Broker:** A Kafka server that manages topics and partitions.
- **Topic:** A category for messages, where data is stored.
- **Partition:** A division of a topic to distribute data across multiple servers.

Example: Real-Time Stock Price Monitoring

1. **Scenario:** A financial firm tracks real-time stock prices across various markets.
2. **Kafka Flow:**
   - Producer: Sends stock price data to the Kafka topic stock_prices.
   - Kafka Broker: Stores and manages the stock price data in partitions.
   - Consumer: A real-time application consumes the data to calculate metrics like moving averages or trigger alerts if a price threshold is crossed.
3. **Outcome:** Real-time alerts notify traders of price changes, while dashboards visualize the data for further analysis.

Data Ingestion & Processing Pipeline

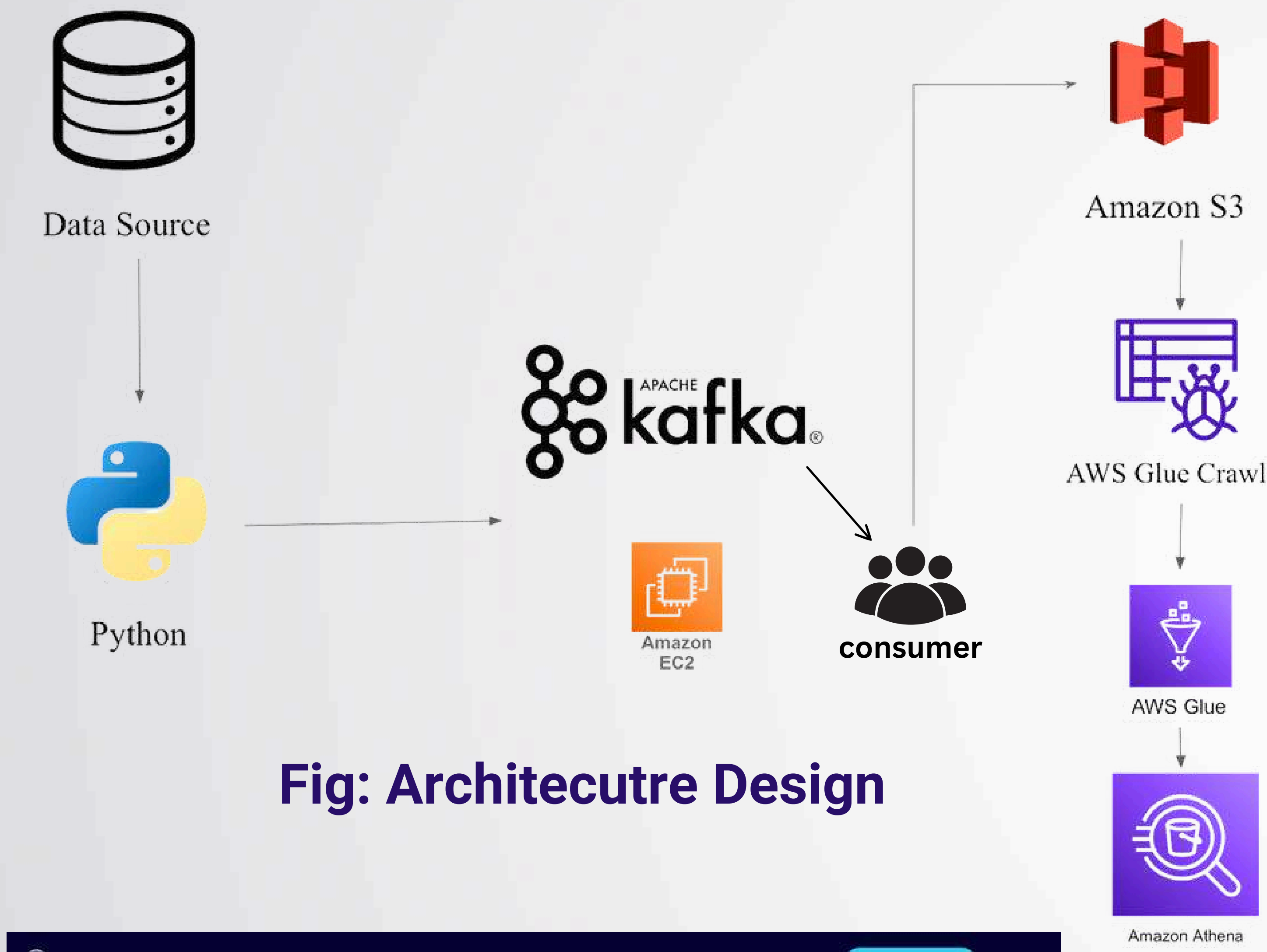
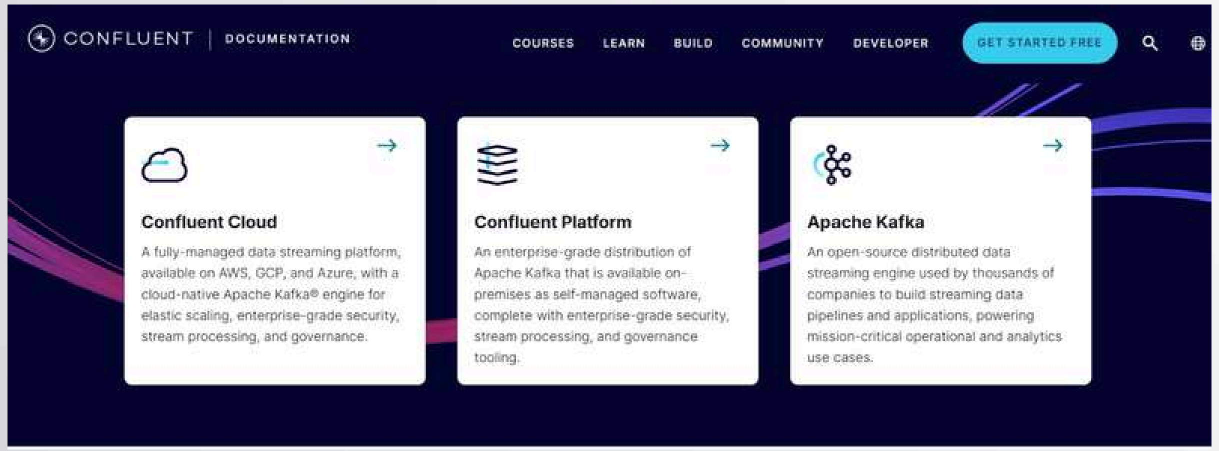


Fig: Architecutre Design



Confluent Cloud

A fully-managed data streaming platform, available on AWS, GCP, and Azure, with a cloud-native Apache Kafka® engine for elastic scaling, enterprise-grade security, stream processing, and governance.

Confluent Platform

An enterprise-grade distribution of Apache Kafka that is available on-premises as self-managed software, complete with enterprise-grade security, stream processing, and governance tooling.

Apache Kafka

An open-source distributed data streaming engine used by thousands of companies to build streaming data pipelines and applications, powering mission-critical operational and analytics use cases.

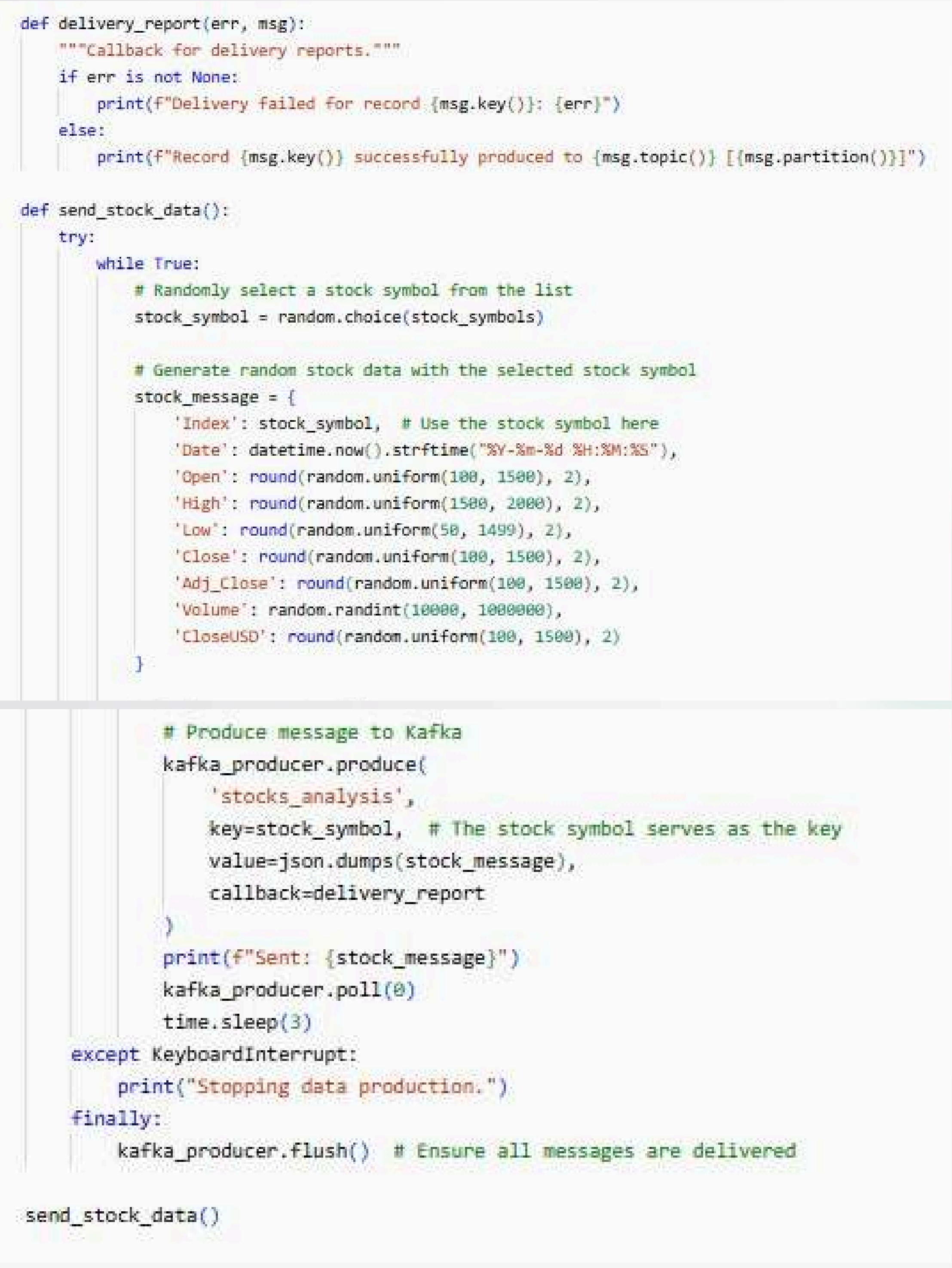
Fig: Confluent Platform

1. **Data Source:** The raw data originates from a data source, which could be a database, logs, or other systems.
2. **Python:** A Python application processes or formats the raw data before sending it to the streaming platform.
3. **Apache Kafka:** Acts as a distributed messaging system for streaming data in real time. Kafka ensures scalability and durability for event-driven data.
4. **Amazon EC2:** Hosts services to manage Kafka producers, consumers, or other processing tasks.
5. **Amazon S3:** Data is streamed and stored in Amazon S3, a scalable object storage service.
6. **AWS Glue Crawler:** Automatically detects and catalogs the schema of the data in S3.
7. **AWS Glue:** Performs ETL (Extract, Transform, Load) processes to prepare data for querying.
8. **Amazon Athena:** Provides serverless SQL query capabilities on the prepared data for insights and analytics.

Set up Kafka for streaming stock data

- **Prerequisites:** Ensure Java is installed and the system meets Kafka's requirements.
- **Download Kafka:** Get the Kafka binary from the official website and extract it.
- **Start Zookeeper:** Launch Zookeeper, which Kafka uses to manage metadata.
- **Start Kafka Broker:** Run the Kafka server to handle message distribution.
- **Configure Kafka:** Adjust the server.properties file for broker ID, log directories, and networking.
- **Create Topics:** Set up logical channels for message publishing and consumption.
- **Test the Setup:** Use Kafka’s console tools to verify producer and consumer functionality.
- **Monitor and Scale:** Track performance with tools and add brokers to grow the cluster.

Kafka
Producer Setup



```python
def delivery_report(err, msg):
    """Callback for delivery reports."""
    if err is not None:
        print(f"Delivery failed for record {msg.key()}: {err}")
    else:
        print(f"Record {msg.key()} successfully produced to {msg.topic()} [{msg.partition()}]")

def send_stock_data():
    try:
        while True:
            # Randomly select a stock symbol from the list
            stock_symbol = random.choice(stock_symbols)

            # Generate random stock data with the selected stock symbol
            stock_message = {
                'Index': stock_symbol,  # Use the stock symbol here
                'Date': datetime.now().strftime("%Y-%m-%d %H:%M:%S"),
                'Open': round(random.uniform(100, 1500), 2),
                'High': round(random.uniform(1500, 2000), 2),
                'Low': round(random.uniform(50, 1499), 2),
                'Close': round(random.uniform(100, 1500), 2),
                'Adj_Close': round(random.uniform(100, 1500), 2),
                'Volume': random.randint(10000, 1000000),
                'CloseUSD': round(random.uniform(100, 1500), 2)
            }
```

```python
            # Produce message to Kafka
            kafka_producer.produce(
                'stocks_analysis',
                key=stock_symbol,  # The stock symbol serves as the key
                value=json.dumps(stock_message),
                callback=delivery_report
            )
            print(f"Sent: {stock_message}")
            kafka_producer.poll(0)
            time.sleep(3)
    except KeyboardInterrupt:
        print("Stopping data production.")
    finally:
        kafka_producer.flush()  # Ensure all messages are delivered

send_stock_data()
```

The producer generates stock price data and sends it to a Kafka topic.

Setup and Configuration:

- Imports required libraries (Producer from confluent_kafka, json, random, etc.).
- Configures the Kafka producer with necessary parameters like bootstrap servers.

Simulated Stock Data:

- Generates random stock data for symbols like MSFT, AMZN, etc.
- Fields include Index, Date, Open, High, Low, Close, Adj_Close, Volume, and CloseUSD.

Publishing to Kafka:

- Each record is serialized into JSON format.
- Published to the Kafka topic stocks_analysis using the produce method.
- A delivery callback (delivery_report) is used to handle success or error responses.

Continuous Streaming: The producer continuously sends new stock data every 3 seconds (time.sleep(3)). Terminates gracefully on a keyboard interrupt.

- *snippest from the code*

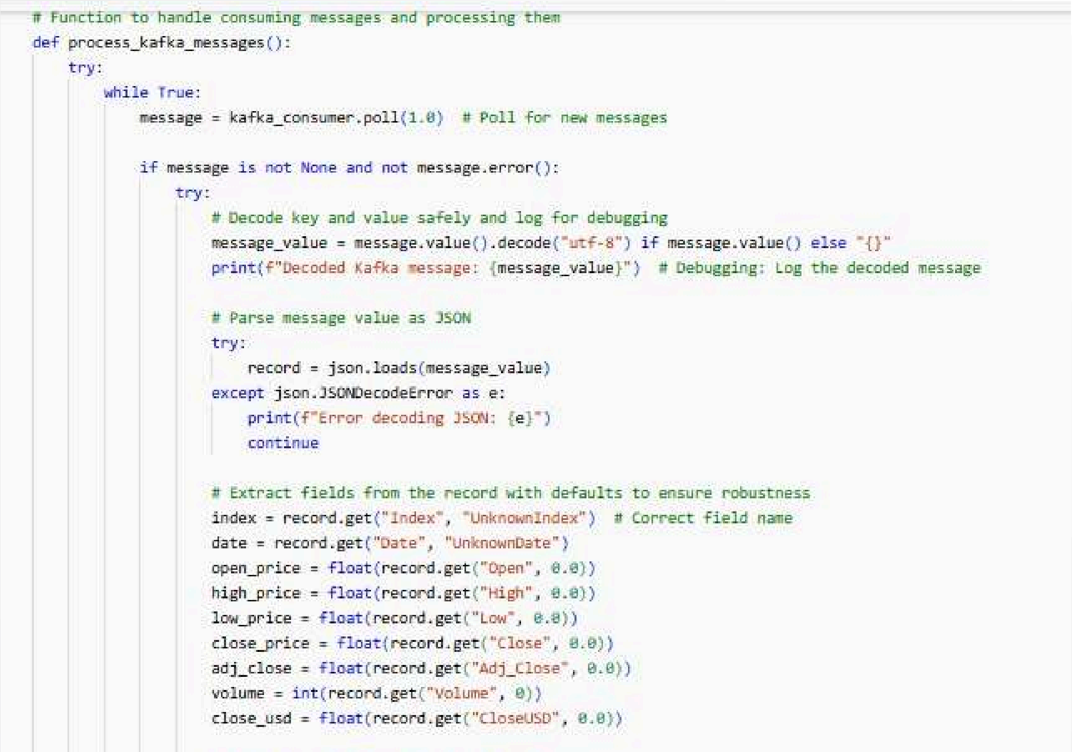
Kafka Consumer Setup

Integration with Kafka:

- **Reading Data from Kafka in PySpark:** PySpark can act as a Kafka consumer to ingest real-time streaming data, ensuring low latency for smooth processing.

Parsing JSON Data:

- **Convert JSON Strings:** Use PySpark's from_json function to parse JSON strings into structured data, based on a predefined schema.
- **Transform into Tabular Format:** Convert the raw JSON data into a structured DataFrame for more advanced analysis and real-time processing.

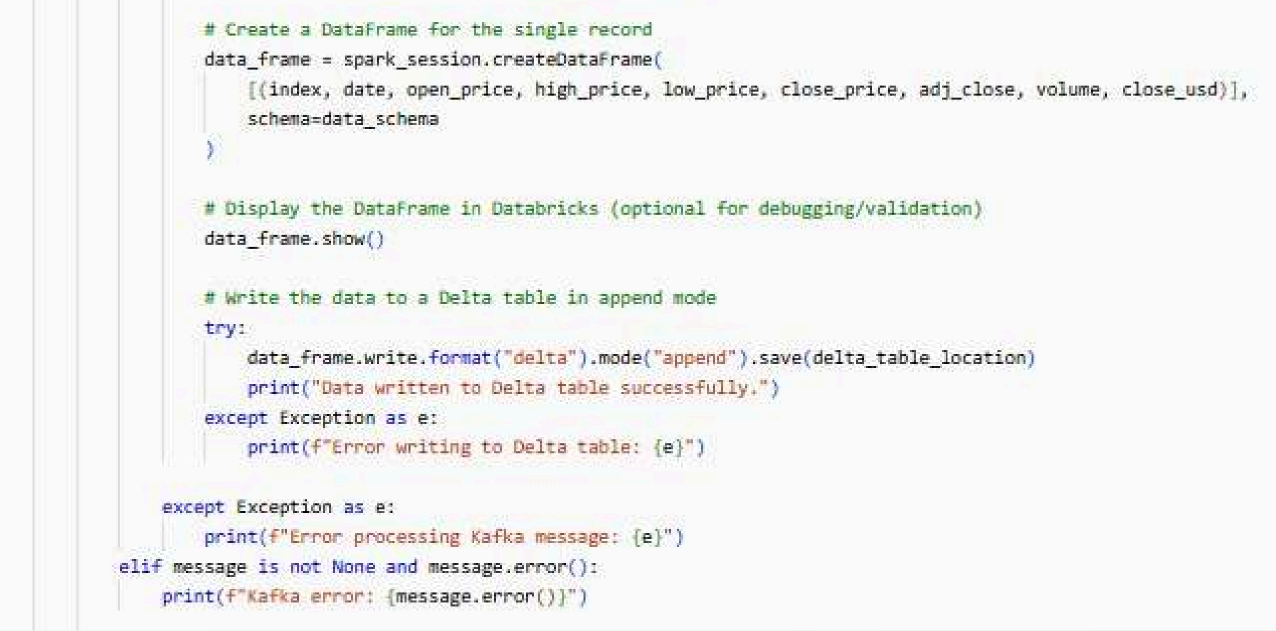


```
# Function to handle consuming messages and processing them
def process_kafka_messages():
    try:
        while True:
            message = kafka_consumer.poll(1.0)  # Poll for new messages

            if message is not None and not message.error():
                try:
                    # Decode key and value safely and log for debugging
                    message_value = message.value().decode("utf-8") if message.value() else "{}"
                    print(f"Decoded Kafka message: {message_value}")  # Debugging: Log the decoded message

                    # Parse message value as JSON
                    try:
                        record = json.loads(message_value)
                    except json.JSONDecodeError as e:
                        print(f"Error decoding JSON: {e}")
                        continue

                    # Extract fields from the record with defaults to ensure robustness
                    index = record.get("Index", "UnknownIndex")  # Correct field name
                    date = record.get("Date", "UnknownDate")
                    open_price = float(record.get("Open", 0.0))
                    high_price = float(record.get("High", 0.0))
                    low_price = float(record.get("Low", 0.0))
                    close_price = float(record.get("Close", 0.0))
                    adj_close = float(record.get("Adj_Close", 0.0))
                    volume = int(record.get("Volume", 0))
                    close_usd = float(record.get("CloseUSD", 0.0))
```



```
                    # Create a DataFrame for the single record
                    data_frame = spark_session.createDataFrame(
                        [(index, date, open_price, high_price, low_price, close_price, adj_close, volume, close_usd)],
                        schema=data_schema
                    )

                    # Display the DataFrame in Databricks (optional for debugging/validation)
                    data_frame.show()

                    # Write the data to a Delta table in append mode
                    try:
                        data_frame.write.format("delta").mode("append").save(delta_table_location)
                        print("Data written to Delta table successfully.")
                    except Exception as e:
                        print(f"Error writing to Delta table: {e}")

                except Exception as e:
                    print(f"Error processing Kafka message: {e}")
            elif message is not None and message.error():
                print(f"Kafka error: {message.error()}")
```

- *snippest from the code*

PySpark Streaming in Databricks

Why Databricks? databricks

- **Unified Platform:** Databricks integrates PySpark with collaborative tools for streamlined real-time data streaming and analysis.
- **Scalable Performance:** It handles large streaming datasets efficiently with automatic resource scaling.
- **Ease of Use:** Simplifies cluster management and monitoring, making streaming pipelines easier to build and maintain.

Steps Involved databricks

- **Import Libraries:** Load PySpark and Kafka-related libraries for handling streaming data and schemas.
- **Configure Spark Session:** Set up a Spark session with the Kafka broker details (bootstrap.servers).
- **Subscribe to Topic:** Read data from the stock_prices Kafka topic using PySpark's structured streaming API.
- **Define Schema:** Create a schema to parse the incoming JSON data for structured processing.
- **Process Stream:** Transform and display real-time data using PySpark’s streaming query capabilities.

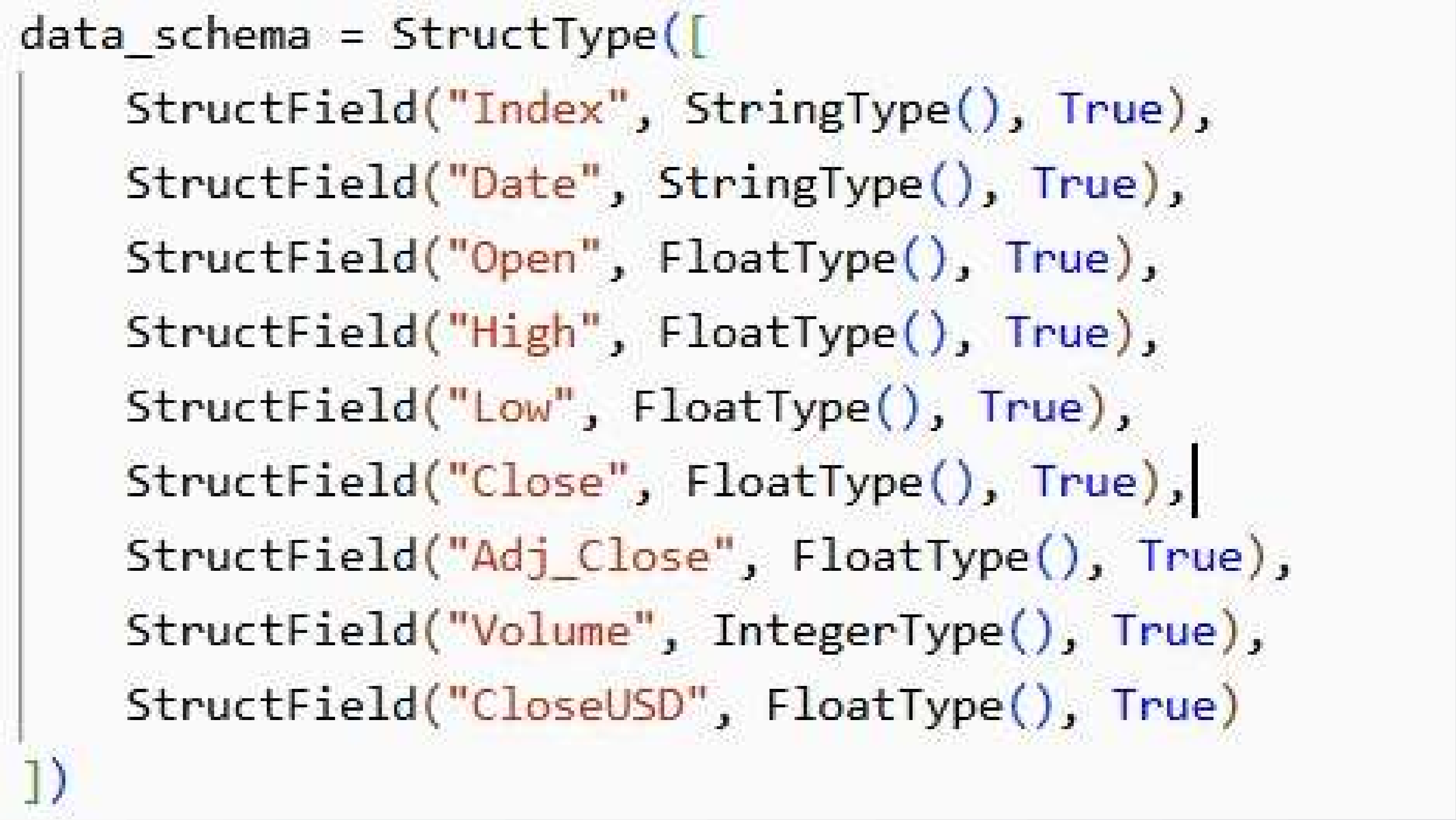
Define schema for stock price data

Why Define a Schema?

- **Simplifies Parsing:** Makes data easier to process.
- **Ensures Consistency:** Guarantees uniform data structure.

Schema Fields:

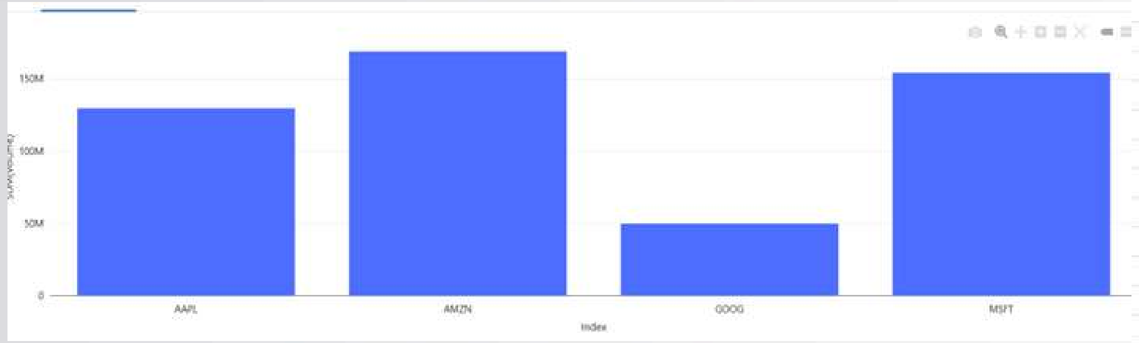
- **Index:** *String type, nullable (True).*
- **Date:** *String type, nullable (True).*
- **Open:** *Float type, nullable (True).*
- **High:** *Float type, nullable (True).*
- **Low:** *Float type, nullable (True).*
- **Close:** *Float type, nullable (True).*
- **Adj_Close:** *Float type, nullable (True).*
- **Volume:** *Integer type, nullable (True).*
- **CloseUSD:** *Float type, nullable (True).*

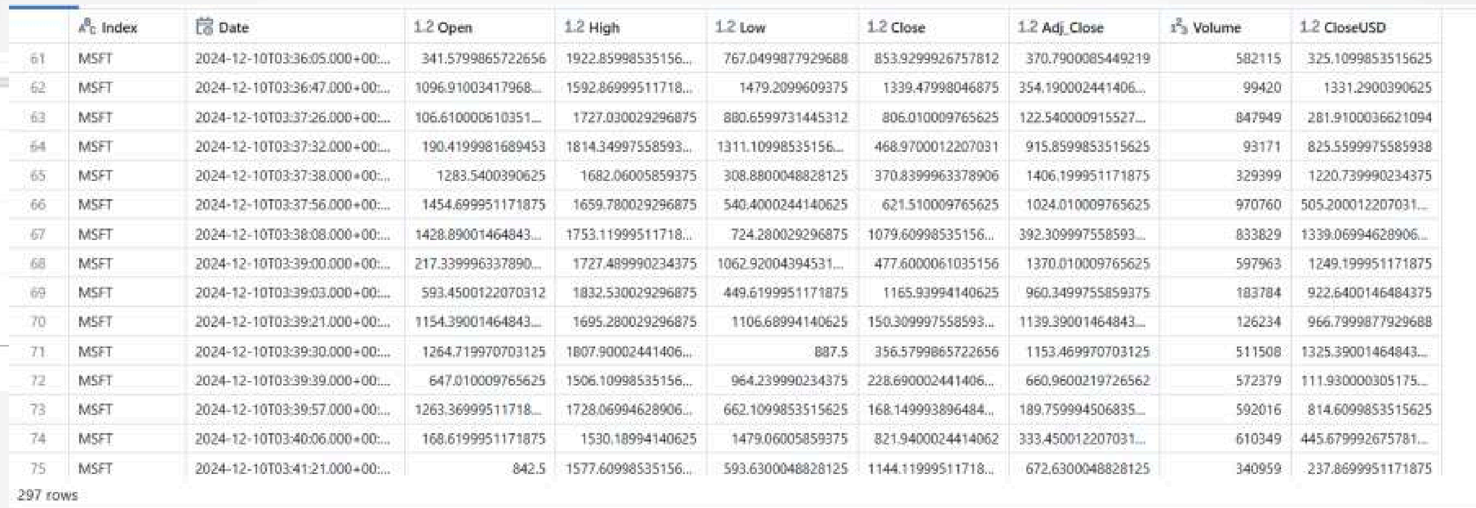


```
data_schema = StructType([
    StructField("Index", StringType(), True),
    StructField("Date", StringType(), True),
    StructField("Open", FloatType(), True),
    StructField("High", FloatType(), True),
    StructField("Low", FloatType(), True),
    StructField("Close", FloatType(), True),
    StructField("Adj_Close", FloatType(), True),
    StructField("Volume", IntegerType(), True),
    StructField("CloseUSD", FloatType(), True)
])
```

Defining schema

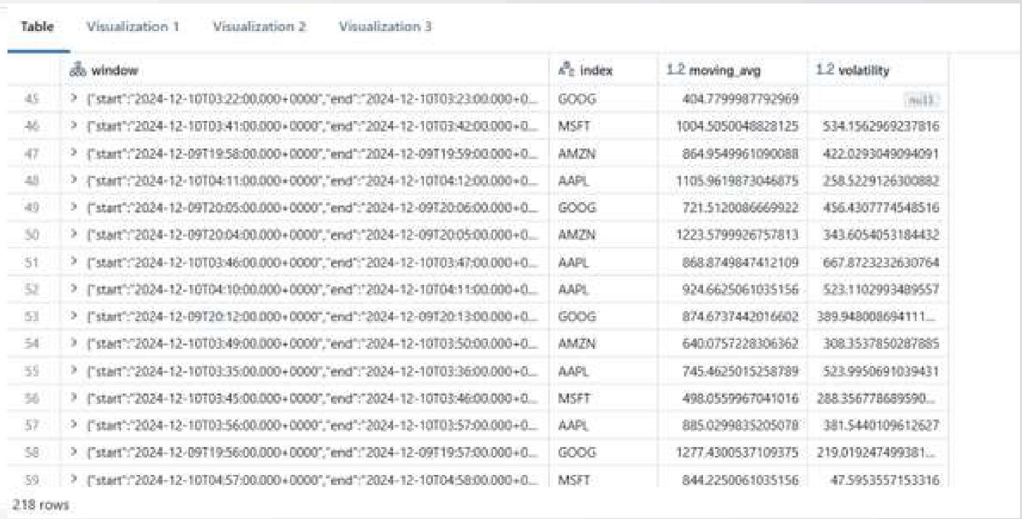
Visualization and Streaming Data Sink





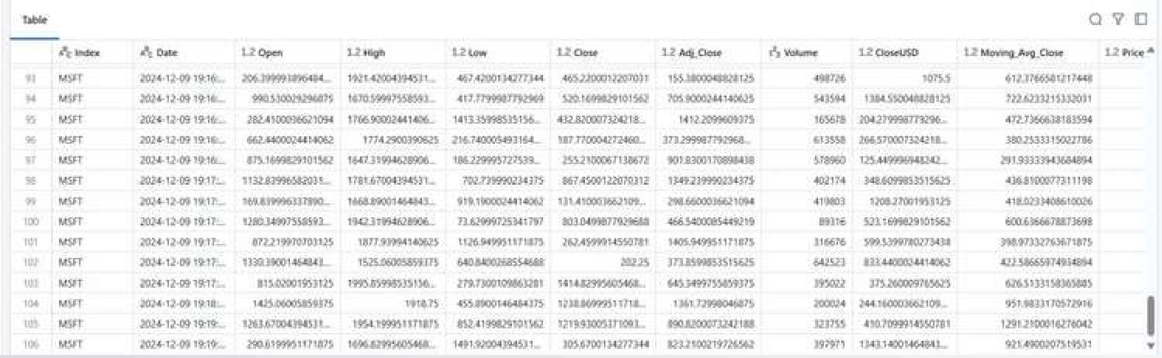
	Index	Date	1.2 Open	1.2 High	1.2 Low	1.2 Close	1.2 Adj_Close	Volume	1.2 CloseUSD
61	MSFT	2024-12-10T03:36:05.000+00:...	341.5799865722656	1922.85998535156...	767.0499877929688	853.9299926757812	370.7900085449219	582115	325.1099853515625
62	MSFT	2024-12-10T03:36:47.000+00:...	1096.91003417968...	1592.86999511718...	1479.2099609375	1339.47998046875	354.190002441406...	99420	1331.2900390625
63	MSFT	2024-12-10T03:37:26.000+00:...	106.610000610351...	1727.030029296875	880.6599731445312	806.010009765625	122.540000915527...	847949	281.9100036621094
64	MSFT	2024-12-10T03:37:32.000+00:...	190.4199981689453	1814.34997558593...	1311.10998535156...	468.9700012207031	915.8599853515625	93171	825.5599975585938
65	MSFT	2024-12-10T03:37:38.000+00:...	1283.5400390625	1682.06005859375	308.8800048828125	370.8399963378906	1406.199951171875	329399	1220.739990234375
66	MSFT	2024-12-10T03:37:56.000+00:...	1454.699951171875	1659.780029296875	540.4000244140625	621.510009765625	1024.010009765625	970760	505.200012207031...
67	MSFT	2024-12-10T03:38:08.000+00:...	1428.89001464843...	1753.11999511718...	724.280029296875	1079.60998535156...	392.309997558593...	833829	1339.06994628906...
68	MSFT	2024-12-10T03:39:00.000+00:...	217.339996337890...	1727.489990234375	1062.92004394531...	477.6000061035156	1370.010009765625	597963	1249.199951171875
69	MSFT	2024-12-10T03:39:03.000+00:...	593.4500122070312	1832.530029296875	449.6199951171875	1165.93994140625	960.3499755859375	183784	922.6400146484375
70	MSFT	2024-12-10T03:39:21.000+00:...	1154.39001464843...	1695.280029296875	1106.68994140625	150.309997558593...	1139.39001464843...	126234	966.7999877929688
71	MSFT	2024-12-10T03:39:30.000+00:...	1264.719970703125	1807.90002441406...	887.5	356.5799865722656	1153.469970703125	511508	1325.39001464843...
72	MSFT	2024-12-10T03:39:39.000+00:...	647.010009765625	1506.10998535156...	964.239990234375	228.690002441406...	660.9600219726562	572379	111.930000305175...
73	MSFT	2024-12-10T03:39:57.000+00:...	1263.36999511718...	1728.06994628906...	662.1099853515625	168.149993896484...	189.759994506835...	592016	814.6099853515625
74	MSFT	2024-12-10T03:40:06.000+00:...	168.6199951171875	1530.18994140625	1479.06005859375	821.9400024414062	333.450012207031...	610349	445.679992675781...
75	MSFT	2024-12-10T03:41:21.000+00:...	842.5	1577.60998535156...	593.6300048828125	1144.11999511718...	672.6300048828125	340959	237.8699951171875

297 rows

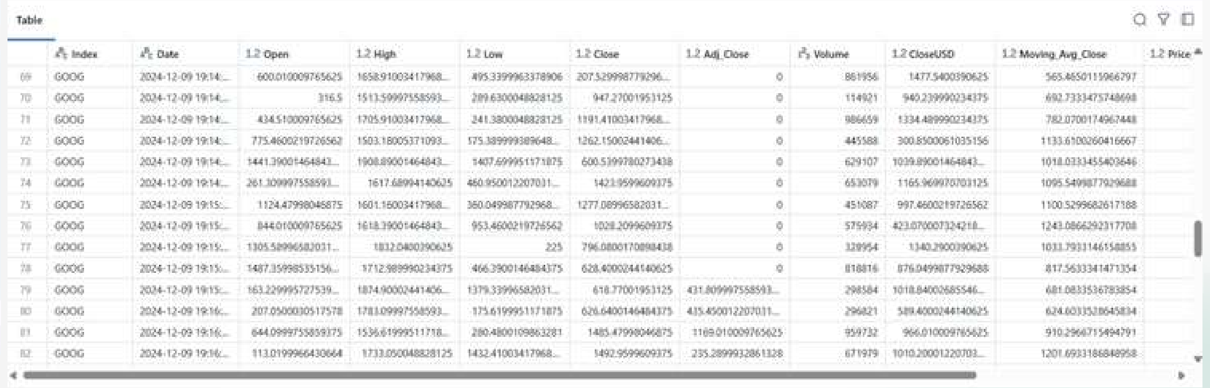


	window	index	1.2 moving_avg	1.2 volatility
45	> {"start":"2024-12-10T03:22:00.000+0000","end":"2024-12-10T03:23:00.000+0...	GOOG	404.7799987792969	null
46	> {"start":"2024-12-10T03:41:00.000+0000","end":"2024-12-10T03:42:00.000+0...	MSFT	1004.5050048828125	534.1562969237816
47	> {"start":"2024-12-09T19:58:00.000+0000","end":"2024-12-09T19:59:00.000+0...	AMZN	864.9549961090088	422.0293049094091
48	> {"start":"2024-12-10T04:11:00.000+0000","end":"2024-12-10T04:12:00.000+0...	AAPL	1105.9619873046875	258.5229126300882
49	> {"start":"2024-12-09T20:05:00.000+0000","end":"2024-12-09T20:06:00.000+0...	GOOG	721.5120086669922	456.4307774548516
50	> {"start":"2024-12-09T20:04:00.000+0000","end":"2024-12-09T20:05:00.000+0...	AMZN	1223.5799926757813	343.6054053184432
51	> {"start":"2024-12-10T03:46:00.000+0000","end":"2024-12-10T03:47:00.000+0...	AAPL	868.8749847412109	667.8723232630764
52	> {"start":"2024-12-10T04:10:00.000+0000","end":"2024-12-10T04:11:00.000+0...	AAPL	924.6625061035156	523.1102993489557
53	> {"start":"2024-12-09T20:12:00.000+0000","end":"2024-12-09T20:13:00.000+0...	GOOG	874.6737442016602	389.948008694111...
54	> {"start":"2024-12-10T03:49:00.000+0000","end":"2024-12-10T03:50:00.000+0...	AMZN	640.0757228306362	308.3537850287885
55	> {"start":"2024-12-10T03:35:00.000+0000","end":"2024-12-10T03:36:00.000+0...	AAPL	745.4625015258789	523.9950691039431
56	> {"start":"2024-12-10T03:45:00.000+0000","end":"2024-12-10T03:46:00.000+0...	MSFT	498.0559967041016	288.356778689590...
57	> {"start":"2024-12-10T03:56:00.000+0000","end":"2024-12-10T03:57:00.000+0...	AAPL	885.0299835205078	381.5440109612627
58	> {"start":"2024-12-09T19:56:00.000+0000","end":"2024-12-09T19:57:00.000+0...	GOOG	1277.4300537109375	219.019247499381...
59	> {"start":"2024-12-10T04:57:00.000+0000","end":"2024-12-10T04:58:00.000+0...	MSFT	844.2250061035156	47.5953557153316

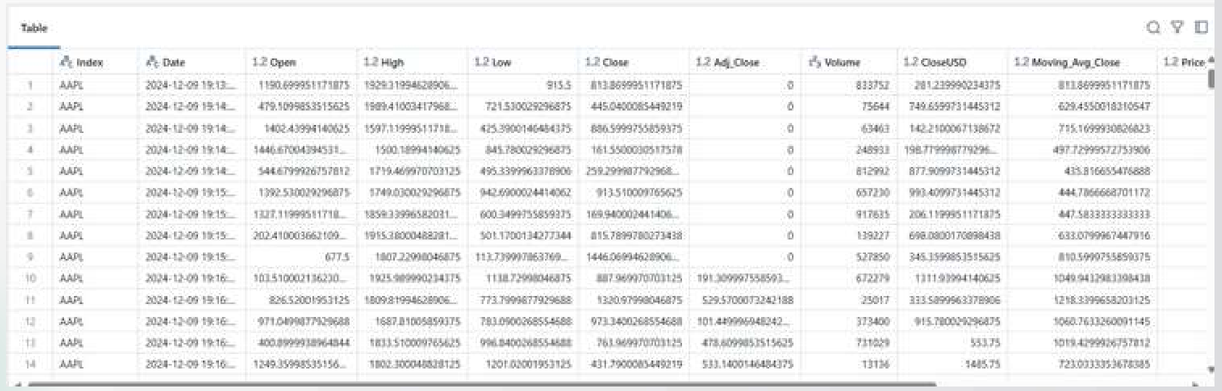
218 rows



	Index	Date	1.2 Open	1.2 High	1.2 Low	1.2 Close	1.2 Adj_Close	Volume	1.2 CloseUSD	1.2 Moving_Avg_Close	1.2 Price
93	MSFT	2024-12-09 19:16:...	206.399993896484...	1921.42004394531...	467.4200134277344	465.2200012207031	155.3800048828125	498726	1075.5	612.3766581217448	
94	MSFT	2024-12-09 19:16:...	990.530029296875	1670.59997558593...	417.7799987792969	520.1699829101562	705.9000244140625	543594	1384.550048828125	722.6233215332031	
95	MSFT	2024-12-09 19:16:...	282.4100036621094	1766.90002441406...	1413.35998535156...	432.820007324218...	1412.2099609375	165678	204.279998779296...	472.7366638183594	
96	MSFT	2024-12-09 19:16:...	662.4400024414062	1774.2900390625	216.740005493164...	187.770004272460...	373.299987792968...	613558	266.570007324218...	380.2533315022786	
97	MSFT	2024-12-09 19:16:...	875.1699829101562	1647.31994628906...	186.229995727539...	255.2100067138672	901.8300170898438	578960	125.449996948242...	291.93333943684894	
98	MSFT	2024-12-09 19:17:...	1132.83996582031...	1781.67004394531...	702.739990234375	867.4500122070312	1349.239990234375	402174	348.6099853515625	436.8100077311198	
99	MSFT	2024-12-09 19:17:...	169.839996337890...	1668.89001464843...	919.1900024414062	131.410003662109...	298.6600036621094	419803	1208.27001953125	418.0233408610026	
100	MSFT	2024-12-09 19:17:...	1280.34997558593...	1942.31994628906...	73.62999725341797	803.0499877929688	466.5400085449219	89316	523.1699829101562	600.6366678873698	
101	MSFT	2024-12-09 19:17:...	872.219970703125	1877.93994140625	1126.949951171875	262.4599914550781	1405.949951171875	316676	599.5399780273438	398.97332763671875	
102	MSFT	2024-12-09 19:17:...	1330.39001464843...	1525.06005859375	640.8400268554688	202.25	373.8599853515625	642523	833.4400024414062	422.5866597494894	
103	MSFT	2024-12-09 19:17:...	815.02001953125	1995.85998535156...	279.7300109863281	1414.82995605468...	645.3499755859375	395022	375.260009765625	626.5133158365885	
104	MSFT	2024-12-09 19:18:...	1425.06005859375	1918.75	455.8900146484375	1238.86999511718...	1361.72998046875	200024	244.160003662109...	951.9833170572916	
105	MSFT	2024-12-09 19:19:...	1263.67004394531...	1954.199951171875	852.4199829101562	1219.93005371093...	890.8200073242188	323755	410.7099914550781	1291.2100016276042	
106	MSFT	2024-12-09 19:19:...	290.6199951171875	1696.82995605468...	1491.92004394531...	305.6700134277344	823.2100219726562	397971	1343.14001464843...	921.4900207519531	



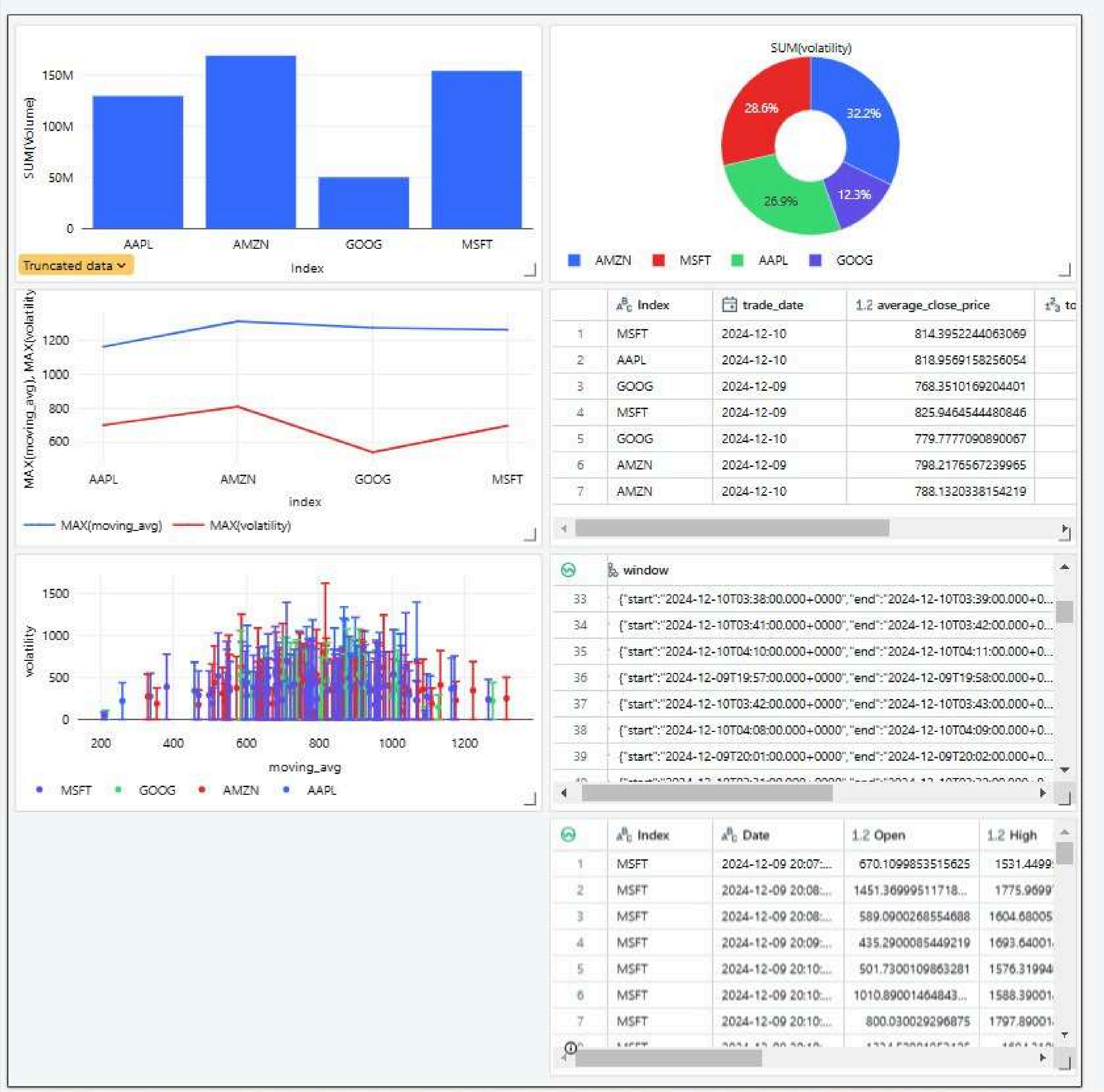
	Index	Date	1.2 Open	1.2 High	1.2 Low	1.2 Close	1.2 Adj_Close	Volume	1.2 CloseUSD	1.2 Moving_Avg_Close	1.2 Price
69	GOOG	2024-12-09 19:14:...	600.010009765625	1658.91003417968...	495.3399963378906	207.529998779296...	0	861956	1477.5400390625	565.4650115966797	
70	GOOG	2024-12-09 19:14:...	316.5	1513.59997558593...	289.6300048828125	947.27001953125	0	114921	940.239990234375	692.7333475748698	
71	GOOG	2024-12-09 19:14:...	434.510009765625	1705.91003417968...	241.3800048828125	1191.41003417968...	0	986659	1334.489990234375	782.0700174967448	
72	GOOG	2024-12-09 19:14:...	775.4600219726562	1503.18005371093...	575.389999389648...	1262.15002441406...	0	445588	300.8500061035156	1133.6100260416667	
73	GOOG	2024-12-09 19:14:...	1441.39001464843...	1908.89001464843...	1407.699951171875	600.5399780273438	0	629107	1039.89001464843...	1018.0333455403646	
74	GOOG	2024-12-09 19:14:...	261.309997558593...	1617.68994140625	460.950012207031...	1423.9599609375	0	653079	1165.969970703125	1095.5499877929688	
75	GOOG	2024-12-09 19:15:...	1124.47998046875	1601.16003417968...	360.049987792968...	1277.08996582031...	0	451087	997.4600219726562	1100.5299682617188	
76	GOOG	2024-12-09 19:15:...	844.010009765625	1618.39001464843...	953.4600219726562	1028.2099609375	0	575934	423.070007324218...	1243.0866292317708	
77	GOOG	2024-12-09 19:15:...	1305.58996582031...	1812.0400390625	225	796.0800170898438	0	328954	1340.2900390625	1033.7933146158855	
78	GOOG	2024-12-09 19:15:...	1487.35998535156...	1712.989990234375	466.3900146484375	628.4000244140625	0	818816	876.0499877929688	817.5633341471354	
79	GOOG	2024-12-09 19:15:...	163.229995727539...	1874.90002441406...	1379.33996582031...	618.77001953125	431.809997558593...	298584	1018.84002685546...	681.0833536783854	
80	GOOG	2024-12-09 19:16:...	207.0500030517578	1783.09997558593...	175.6199951171875	626.6400146484375	415.450012207031...	296821	589.4000244140625	624.6033528645834	
81	GOOG	2024-12-09 19:16:...	644.0999755859375	1536.61999511718...	280.4800109863281	1485.47998046875	1169.010009765625	959732	966.010009765625	910.2966715494791	
82	GOOG	2024-12-09 19:16:...	113.0199966430664	1733.050048828125	1432.41003417968...	1492.9599609375	235.2899932861328	671979	1010.20001220703...	1201.6933186848958	



	Index	Date	1.2 Open	1.2 High	1.2 Low	1.2 Close	1.2 Adj_Close	Volume	1.2 CloseUSD	1.2 Moving_Avg_Close	1.2 Price
1	AAPL	2024-12-09 19:13:...	1190.699951171875	1929.31994628906...	913.5	813.8699951171875	0	833752	281.239990234375	813.8699951171875	
2	AAPL	2024-12-09 19:14:...	479.1099853515625	1989.41003417968...	721.530029296875	445.0400085449219	0	75644	749.6599731445312	629.4550018310547	
3	AAPL	2024-12-09 19:14:...	1402.43994140625	1597.11999511718...	425.3900146484375	886.5999755859375	0	63463	142.2100067138672	715.1699930826823	
4	AAPL	2024-12-09 19:14:...	1446.67004394531...	1500.18994140625	845.780029296875	161.5500030517578	0	248933	198.779998779296...	497.72999572753906	
5	AAPL	2024-12-09 19:14:...	544.6799926757812	1719.469970703125	495.3399963378906	259.299987792968...	0	812992	877.9099731445312	435.816655476888	
6	AAPL	2024-12-09 19:15:...	1392.530029296875	1749.030029296875	942.6900024414062	913.510009765625	0	657230	993.4099731445312	444.7866668701172	
7	AAPL	2024-12-09 19:15:...	1327.11999511718...	1859.33996582031...	600.3499755859375	169.940002441406...	0	917635	206.1199951171875	447.5833333333333	
8	AAPL	2024-12-09 19:15:...	202.410003662109...	1915.38000488281...	501.1700134277344	815.7899780273438	0	139227	698.0800170898438	633.0799967447916	
9	AAPL	2024-12-09 19:15:...	677.5	1807.22998046875	113.739997863769...	1446.06994628906...	0	527850	345.3599853515625	810.5999755859375	
10	AAPL	2024-12-09 19:16:...	103.510002136230...	1925.989990234375	1138.72998046875	887.969970703125	191.309997558593...	672279	1311.93994140625	1049.9412983398438	
11	AAPL	2024-12-09 19:16:...	826.52001953125	1809.81994628906...	773.7999877929688	1320.97998046875	529.5700073242188	25017	333.5899963378906	1218.3399658203125	
12	AAPL	2024-12-09 19:16:...	971.0499877929688	1687.81005859375	783.0900268554688	973.3400268554688	101.449996948242...	373400	915.780029296875	1060.7633260091145	
13	AAPL	2024-12-09 19:16:...	400.8999938964844	1833.510009765625	996.8400268554688	763.969970703125	478.6099853515625	731029	553.75	1019.4299926757812	
14	AAPL	2024-12-09 19:16:...	1249.35998535156...	1802.300048828125	1201.02001953125	431.7900085449219	533.1400146484375	13136	1485.75	723.0333353678385	

- *Streaming Data Sink (Top Left)*
- *Stock Symbol Filtering (Top Middle)*
- *Real-Time Aggregation (Moving Averages & Volatility) (Top Right)*
- *Daily Summary Calculation (Bottom Left)*
- *Price Change and Volatility Calculation (Bottom Middle)*
- *Stock Price Alerts (Bottom Right)*

Final Dashboard Visualization



	Index	trade_date	1.2 average_close_price	to
1	MSFT	2024-12-10	814.3952244063069	
2	AAPL	2024-12-10	818.9569158256054	
3	GOOG	2024-12-09	768.3510169204401	
4	MSFT	2024-12-09	825.9464544480846	
5	GOOG	2024-12-10	779.7777090890067	
6	AMZN	2024-12-09	798.2176567239965	
7	AMZN	2024-12-10	788.1320338154219	

	window
33	{"start":"2024-12-10T03:38:00.000+0000","end":"2024-12-10T03:39:00.000+0...
34	{"start":"2024-12-10T03:41:00.000+0000","end":"2024-12-10T03:42:00.000+0...
35	{"start":"2024-12-10T04:10:00.000+0000","end":"2024-12-10T04:11:00.000+0...
36	{"start":"2024-12-09T19:57:00.000+0000","end":"2024-12-09T19:58:00.000+0...
37	{"start":"2024-12-10T03:42:00.000+0000","end":"2024-12-10T03:43:00.000+0...
38	{"start":"2024-12-10T04:08:00.000+0000","end":"2024-12-10T04:09:00.000+0...
39	{"start":"2024-12-09T20:01:00.000+0000","end":"2024-12-09T20:02:00.000+0...

	Index	Date	1.2 Open	1.2 High
1	MSFT	2024-12-09 20:07:...	670.1099853515625	1531.4499
2	MSFT	2024-12-09 20:08:...	1451.36999511718...	1775.9699
3	MSFT	2024-12-09 20:08:...	589.0900268554688	1604.68005
4	MSFT	2024-12-09 20:09:...	435.2900085449219	1693.64001
5	MSFT	2024-12-09 20:10:...	501.7300109863281	1576.31994
6	MSFT	2024-12-09 20:10:...	1010.89001464843...	1588.39001
7	MSFT	2024-12-09 20:10:...	800.030029296875	1797.89001

- *Bar Chart (Top Left): Shows the total trading volume for stocks (AAPL, AMZN, GOOG, MSFT), identifying the most actively traded stock.*
- *Pie Chart (Top Right): Displays the percentage contribution of each stock to overall volatility.*
- *Line Chart (Middle Left): Compares maximum moving averages (price trends) and maximum volatilities for stocks.*
- *Bar chart (Bottom Left): Correlates volatility with moving averages to analyze stock price stability and trends.*
- *Summary Table (Middle Right): Provides key metrics like average closing price for each stock by date.*
- *Window Data Table (Bottom Right): Shows real-time streaming batches with processing windows and metrics.*
- *Raw Data Table (Bottom Right Corner): Displays unprocessed stock data like Open, High, and Volume for analysis.*

LIBRARIES AND MODULES USED

CONFLUENT

databricks

kafka

Seaborn

Pandas

PySpark

Google colab

matplotlib

Conclusion

- **Confluent Kafka Integration:** Real-time stock price streaming through Kafka's message queuing.
- **PySpark Data Processing:** Utilizes PySpark for efficient and scalable data processing and analytics.
- **Interactive Dashboard:** Provides visualizations (bar charts, pie charts, line graphs) for tracking stock performance and trends.
- **Stock Symbol Coverage:** Currently limited, but can be expanded for broader market analysis.
- **Scalability Challenges:** Potential issues with scaling, addressed in future enhancements.
- **Future Enhancements:** Expanding stock symbols, adding advanced analytics, and improving scalability for better performance.

Future Work

Enhanced Data Analytics

Integrate machine learning for price predictions, sentiment analysis from news or social media, and anomaly detection to provide smarter insights for traders.

Real-Time Alerts

Implement an alert system that notifies users of significant market changes, such as price shifts or volatility spikes, enabling quick decision-making.

Cloud Deployment

Deploy the system on cloud platforms to handle larger datasets, improve scalability, and provide easier access for a broader user base.

Challenges of the Project

Data Quality and Consistency

Validating and cleaning real-time data for accuracy.

Scalability

Configuring Kafka and PySpark to handle large data volumes.

Latency

Minimizing delays to ensure real-time processing.

Fault Tolerance and Reliability

Ensuring system recovery with Kafka replication and PySpark checkpoints.

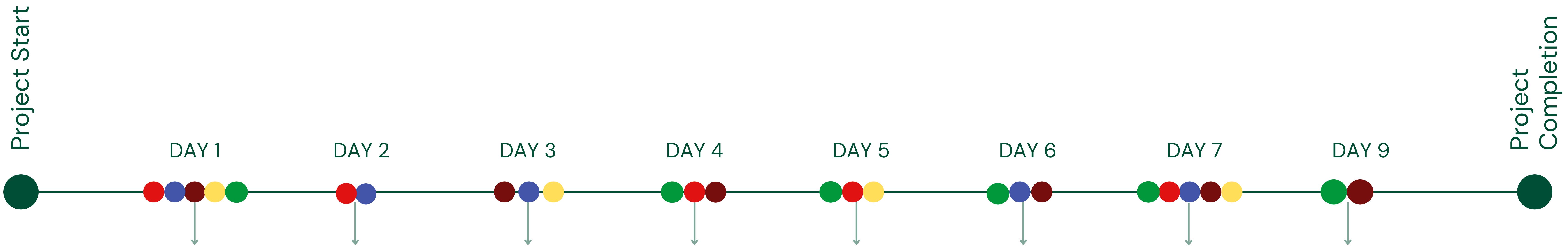
Complex Event Processing (CEP)

Detecting real-time patterns and anomalies efficiently.

Real-Time Visualization

Displaying live data and insights through dashboards.

Project Timeline



Project Start

DAY 1 🔴🔵🟤🟡🟢

DAY 2 🔴🔵

DAY 3 🟤🔵🟡

DAY 4 🟢🔴🟤

DAY 5 🟢🔴🟡

DAY 6 🟢🔵🟤

DAY 7 🟢🔴🔵🟤🟡

DAY 9 🟢🟤

Project Completion

Kafka Setup and Configuration

- Installed Apache Kafka locally, configured Kafka on cloud (if applicable), and created the stock_prices topic for streaming data.

Kafka Producer and Data Streaming

- Developed a Kafka producer script to simulate real-time stock prices. The producer publishes stock data at regular intervals to the stock_prices topic.

PySpark Environment Setup

- Set up PySpark on Databricks (or locally). Verified successful integration between PySpark and Kafka for real-time streaming.

Schema Definition and Data Parsing

- Defined the schema for stock price data, including fields like stock symbol, price, and timestamp. Implemented PySpark logic to parse incoming Kafka data into a structured format.

Data Consumption and Real-Time Processing

- Configured PySpark streaming to consume data from the Kafka stock_prices topic. Implemented real-time metrics like moving averages and price fluctuations.

Visualization and Fault Tolerance

- Developed a dashboard to visualize live stock price data, moving averages, and alerts.
- Set up Delta tables for persistent storage and configured Kafka replication and PySpark checkpointing for fault tolerance.

Final Testing, Optimization, and Reporting

Final Files Delivery

- **Dec 9:** Submit project
- **Dec 9:** Presentation

Legend:

🟤 Bharat Dhungana

🔵 Samir Bhandari

🟢 Aravind

🟡 Satish Kandel

🔴 Bikash Thapa

References

- Confluent Documentation. (n.d.). Confluent documentation. Retrieved December 8, 2024, from *https://docs.confluent.io/*
- Apache Spark. (n.d.). Cluster mode overview. Retrieved December 8, 2024, from *https://spark.apache.org/docs/latest/cluster-overview.html*
- Confluent. (n.d.). confluent_kafka API. Retrieved December 9, 2024, from *https://docs.confluent.io/platform/current/clients/confluent-kafka-python/html/index.html*
- Iwuchukwu, C. (2024, April 23). Analyzing stock prices using PySpark. Medium. Retrieved December 7, 2024, from *https://medium.com/@ceejayiwufitness/analyzing-stock-prices-using-pyspark-acdaef8a5511*
- ResearchGate. (n.d.). Real-time streaming data analysis using Spark. Retrieved December 9, 2024, from *https://www.researchgate.net/publication/322674233_Real time_Streaming_Data_Analysis_using_Spark*

Thank You

For Your Attention

Any Queries?

We are open for discussion.

Access this ppt via: bit.ly/aml2054