Kafka Data Pipeline

Overview

This Kafka data pipeline consists of a Kafka cluster with multiple topics, two producers that generate data, and three consumers that consume and process the data. The data is generated based on a realistic financial transactions use case using a Kafka client library. The consumers perform different processes on the data, such as aggregating company events by company ID, calculating real-time statistics on stock inventory, and filtering out fraudulent financial transactions. The pipeline is also integrated with an external system, such as a MongoDB or SQL database.

Data Description

Brief description of some of the most common data available on Yahoo Finance:

- Stock Price Data: Historical and real-time data on stock prices for publicly traded companies, including the opening price, closing price, high price, low price, and trading volume.
- Financial Statements: Quarterly and annual financial statements for publicly traded companies, including the balance sheet, income statement, and cash flow statement.
- Market Data: Information on market capitalization, trading volume, and other key market metrics for individual stocks and broader market indices.
- News and Analysis: Articles and analysis on breaking news and trends in the financial markets, including company news, analyst reports, and economic indicators.
- Charting and Technical Analysis: Charting tools and technical analysis indicators to help investors and traders analyze market trends and make informed decisions.
- Options and Derivatives Data: Data on options and other derivative securities, including strike price, expiration date, and open interest.

The data is generated based on a realistic financial transactions use case and sent to different Kafka topics.

Pipeline Setup

To set up and run the pipeline, follow these steps:

1. Install Apache Kafka and ZooKeeper.
2. Start ZooKeeper by running the following command in a terminal:

   Code :
   ```
   bin/zookeeper-server-start.sh config/zookeeper.properties
   ```

3. Start Kafka brokers by running the following command in a new terminal:

Code:

```
kafka-server-start.sh config/server1.properties
```

```
kafka-server-start.sh config/server2.properties
```

kafka-server-start.sh config/server3.properties

4. Create Kafka topics with different replication factors, retention periods, and partition counts. For example:

Code:

```
bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 3 --
partitions 4--   topic topic1
```

5. Create kafka producers

Code:

```
kafka-console-producer.sh --bootstrap-server localhost:9092 --topic topic1,topic2
```

```
kafka-console-producer.sh --bootstrap-server localhost:9092 --topic topic3
```

6. Create kafka consumers

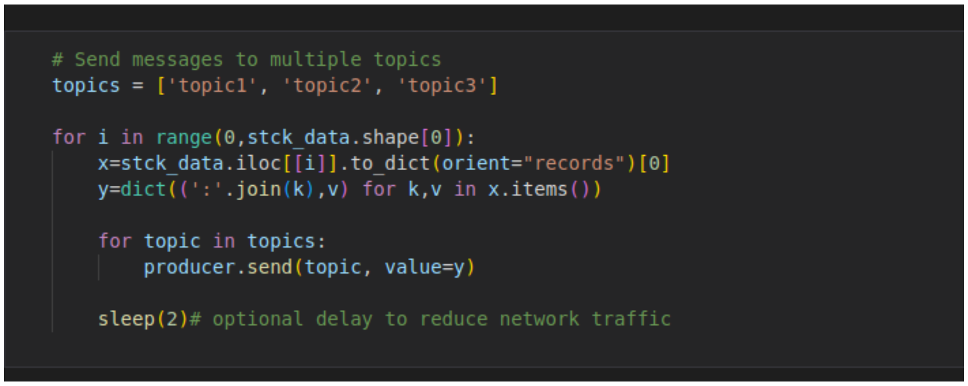
Code:

```
kafka-console-consumer.sh --bootstrap-server localhost:9092,localhost:9093,localhost:9094 --t opic
topic1
```

```
kafka-console-consumer.sh --bootstrap-server localhost:9092,localhost:9093,localhost:9094 --   topic
topic2
```

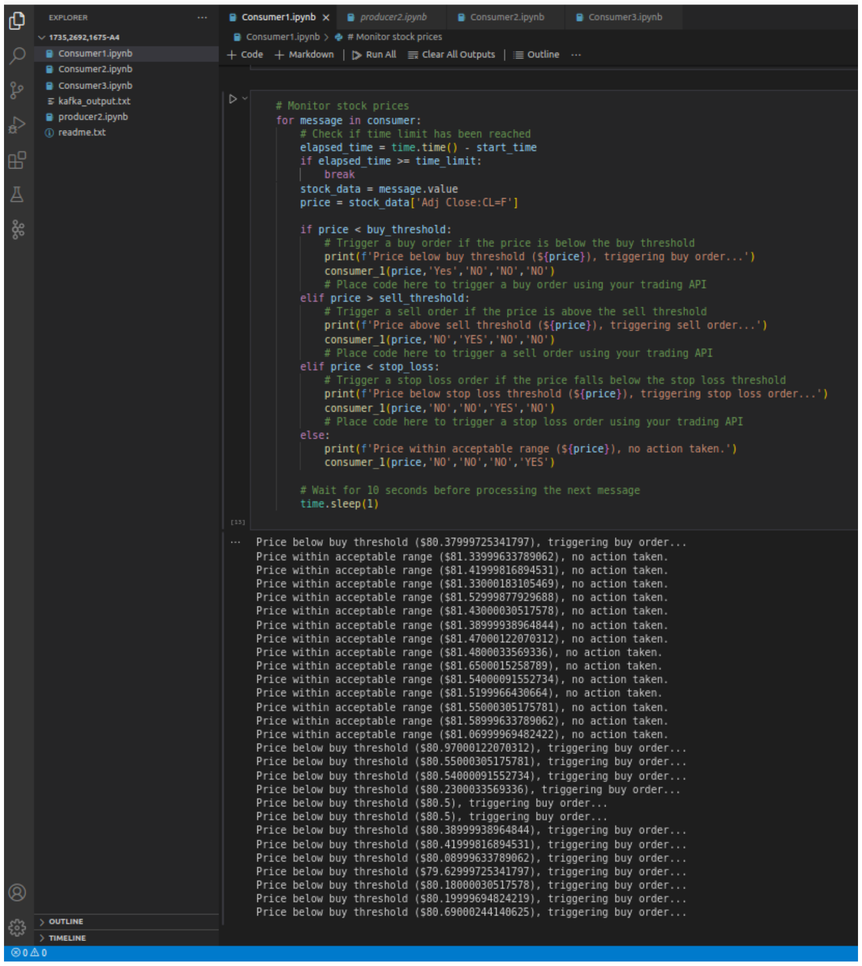
```
kafka-console-consumer.sh --bootstrap-server localhost:9092,localhost:9093,localhost:9094 --   topic
topic3
```

- ***Producer code***



```
# Send messages to multiple topics
topics = ['topic1', 'topic2', 'topic3']

for i in range(0,stck_data.shape[0]):
    x=stck_data.iloc[[i]].to_dict(orient="records")[0]
    y=dict((':'.join(k),v) for k,v in x.items())

    for topic in topics:
        producer.send(topic, value=y)

    sleep(2)# optional delay to reduce network traffic
```

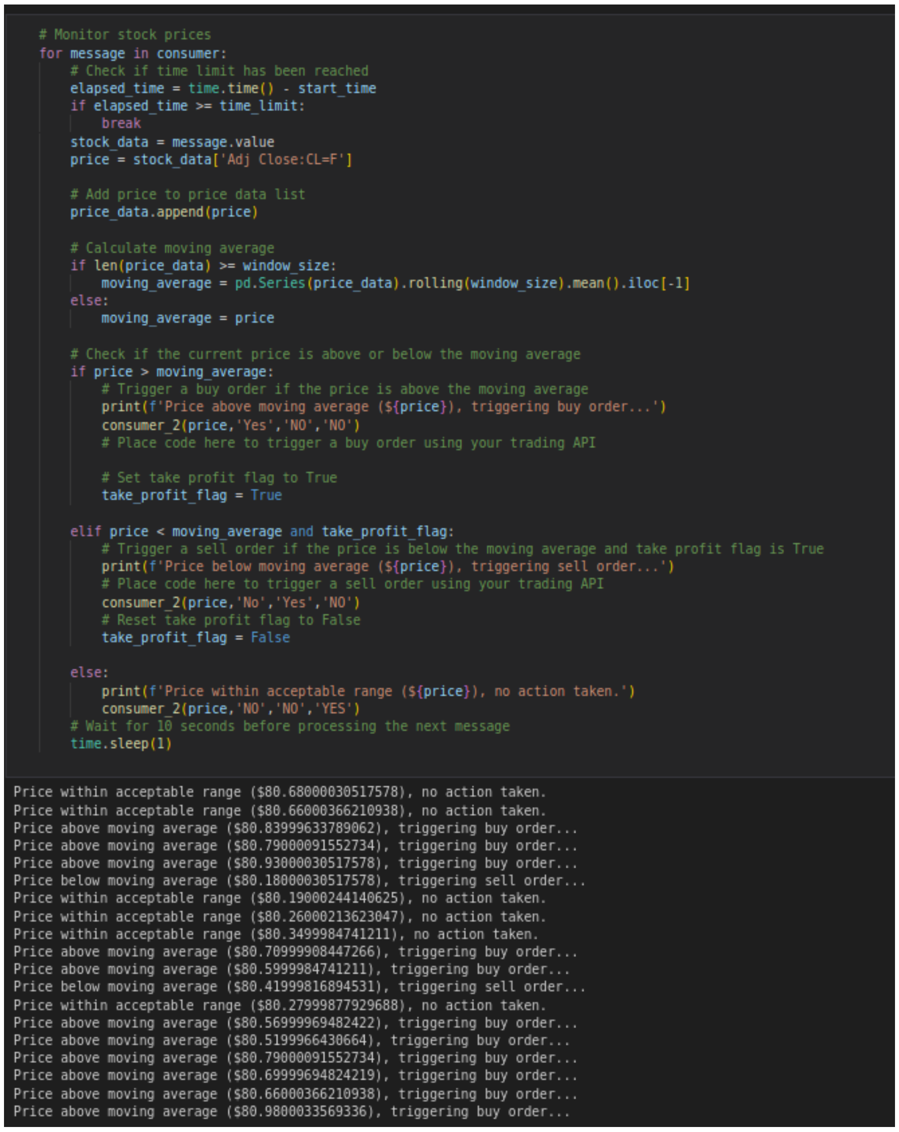
- ***Consumer 1***



```python
# Monitor stock prices
for message in consumer:
    # Check if time limit has been reached
    elapsed_time = time.time() - start_time
    if elapsed_time >= time_limit:
        break
    stock_data = message.value
    price = stock_data['Adj Close:CL=F']

    if price < buy_threshold:
        # Trigger a buy order if the price is below the buy threshold
        print(f'Price below buy threshold (${price}), triggering buy order...')
        consumer_1(price,'Yes','NO','NO','NO')
        # Place code here to trigger a buy order using your trading API
    elif price > sell_threshold:
        # Trigger a sell order if the price is above the sell threshold
        print(f'Price above sell threshold (${price}), triggering sell order...')
        consumer_1(price,'NO','YES','NO','NO')
        # Place code here to trigger a sell order using your trading API
    elif price < stop_loss:
        # Trigger a stop loss order if the price falls below the stop loss threshold
        print(f'Price below stop loss threshold (${price}), triggering stop loss order...')
        consumer_1(price,'NO','NO','YES','NO')
        # Place code here to trigger a stop loss order using your trading API
    else:
        print(f'Price within acceptable range (${price}), no action taken.')
        consumer_1(price,'NO','NO','NO','YES')

    # Wait for 10 seconds before processing the next message
    time.sleep(1)
```

```
Price below buy threshold ($80.37999725341797), triggering buy order...
Price within acceptable range ($81.33999633789062), no action taken.
Price within acceptable range ($81.41999816894531), no action taken.
Price within acceptable range ($81.33000183105469), no action taken.
Price within acceptable range ($81.52999877929688), no action taken.
Price within acceptable range ($81.43000030517578), no action taken.
Price within acceptable range ($81.38999938964844), no action taken.
Price within acceptable range ($81.47000122070312), no action taken.
Price within acceptable range ($81.4800033569336), no action taken.
Price within acceptable range ($81.6500015258789), no action taken.
Price within acceptable range ($81.54000091552734), no action taken.
Price within acceptable range ($81.5199966430664), no action taken.
Price within acceptable range ($81.55000305175781), no action taken.
Price within acceptable range ($81.58999633789062), no action taken.
Price within acceptable range ($81.06999969482422), no action taken.
Price below buy threshold ($80.97000122070312), triggering buy order...
Price below buy threshold ($80.55000305175781), triggering buy order...
Price below buy threshold ($80.54000091552734), triggering buy order...
Price below buy threshold ($80.2300033569336), triggering buy order...
Price below buy threshold ($80.5), triggering buy order...
Price below buy threshold ($80.5), triggering buy order...
Price below buy threshold ($80.38999938964844), triggering buy order...
Price below buy threshold ($80.41999816894531), triggering buy order...
Price below buy threshold ($80.08999633789062), triggering buy order...
Price below buy threshold ($79.62999725341797), triggering buy order...
Price below buy threshold ($80.18000030517578), triggering buy order...
Price below buy threshold ($80.19999694824219), triggering buy order...
Price below buy threshold ($80.69000244140625), triggering buy order...
```

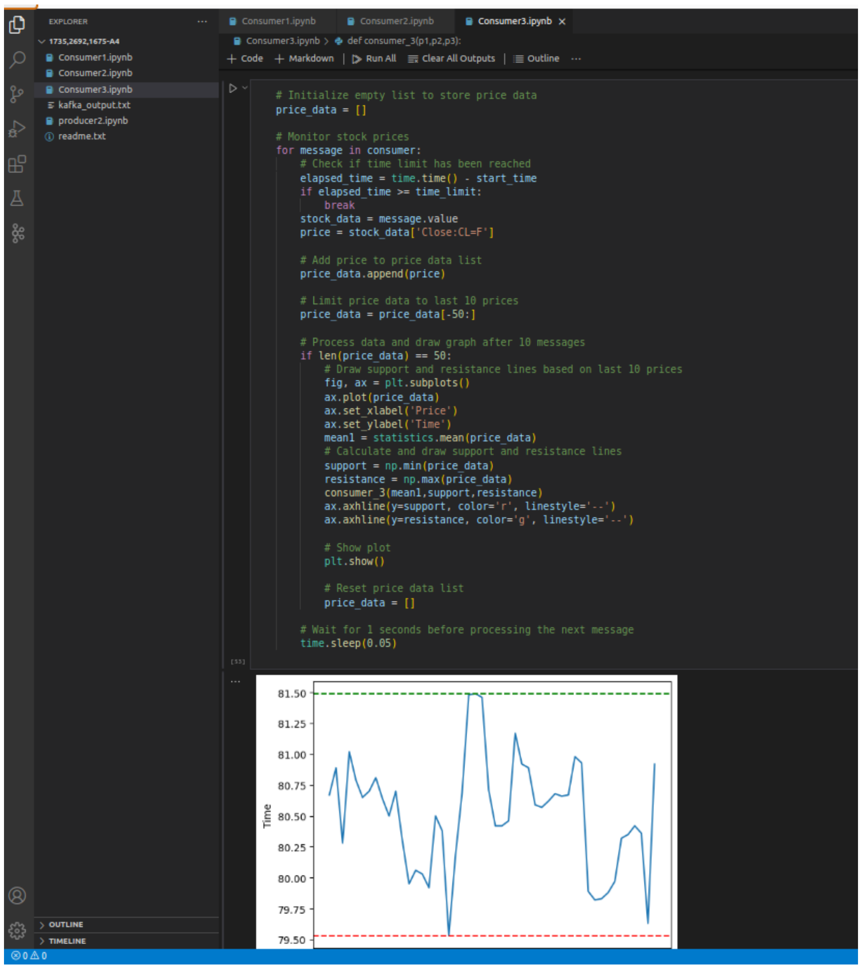
- ***Consumer 2***



```python
# Monitor stock prices
for message in consumer:
    # Check if time limit has been reached
    elapsed_time = time.time() - start_time
    if elapsed_time >= time_limit:
        break
    stock_data = message.value
    price = stock_data['Adj Close:CL=F']

    # Add price to price data list
    price_data.append(price)

    # Calculate moving average
    if len(price_data) >= window_size:
        moving_average = pd.Series(price_data).rolling(window_size).mean().iloc[-1]
    else:
        moving_average = price

    # Check if the current price is above or below the moving average
    if price > moving_average:
        # Trigger a buy order if the price is above the moving average
        print(f'Price above moving average (${price}), triggering buy order...')
        consumer_2(price,'Yes','NO','NO')
        # Place code here to trigger a buy order using your trading API

        # Set take profit flag to True
        take_profit_flag = True

    elif price < moving_average and take_profit_flag:
        # Trigger a sell order if the price is below the moving average and take profit flag is True
        print(f'Price below moving average (${price}), triggering sell order...')
        # Place code here to trigger a sell order using your trading API
        consumer_2(price,'No','Yes','NO')
        # Reset take profit flag to False
        take_profit_flag = False

    else:
        print(f'Price within acceptable range (${price}), no action taken.')
        consumer_2(price,'NO','NO','YES')
    # Wait for 10 seconds before processing the next message
    time.sleep(1)
```

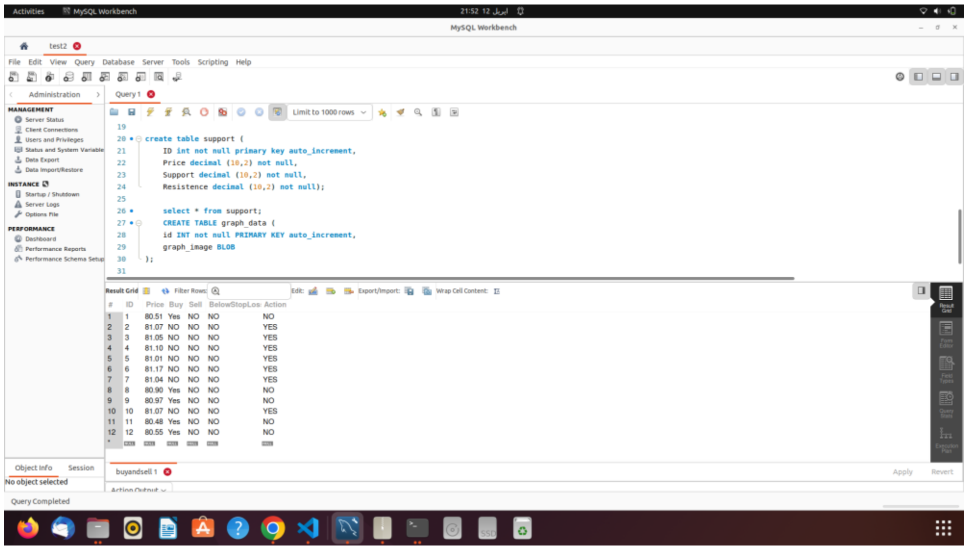
```
Price within acceptable range ($80.68000030517578), no action taken.
Price within acceptable range ($80.66000366210938), no action taken.
Price above moving average ($80.83999633789062), triggering buy order...
Price above moving average ($80.79000091552734), triggering buy order...
Price above moving average ($80.93000030517578), triggering buy order...
Price below moving average ($80.18000030517578), triggering sell order...
Price within acceptable range ($80.19000244140625), no action taken.
Price within acceptable range ($80.26000213623047), no action taken.
Price within acceptable range ($80.3499984741211), no action taken.
Price above moving average ($80.70999908447266), triggering buy order...
Price above moving average ($80.5999984741211), triggering buy order...
Price below moving average ($80.41999816894531), triggering sell order...
Price within acceptable range ($80.27999877929688), no action taken.
Price above moving average ($80.56999969482422), triggering buy order...
Price above moving average ($80.5199966430664), triggering buy order...
Price above moving average ($80.79000091552734), triggering buy order...
Price above moving average ($80.69999694824219), triggering buy order...
Price above moving average ($80.66000366210938), triggering buy order...
Price above moving average ($80.9800033569336), triggering buy order...
```

- ***<u>Consumer 3</u>***



```
# Initialize empty list to store price data
price_data = []

# Monitor stock prices
for message in consumer:
    # Check if time limit has been reached
    elapsed_time = time.time() - start_time
    if elapsed_time >= time_limit:
        break
    stock_data = message.value
    price = stock_data['Close:CL=F']

    # Add price to price data list
    price_data.append(price)

    # Limit price data to last 10 prices
    price_data = price_data[-50:]

    # Process data and draw graph after 10 messages
    if len(price_data) == 50:
        # Draw support and resistance lines based on last 10 prices
        fig, ax = plt.subplots()
        ax.plot(price_data)
        ax.set_xlabel('Price')
        ax.set_ylabel('Time')
        mean1 = statistics.mean(price_data)
        # Calculate and draw support and resistance lines
        support = np.min(price_data)
        resistance = np.max(price_data)
        consumer_3(mean1,support,resistance)
        ax.axhline(y=support, color='r', linestyle='--')
        ax.axhline(y=resistance, color='g', linestyle='--')

        # Show plot
        plt.show()

        # Reset price data list
        price_data = []

    # Wait for 1 seconds before processing the next message
    time.sleep(0.05)
```

- ***Database table 1***



```
create table support (
    ID int not null primary key auto_increment,
    Price decimal (10,2) not null,
    Support decimal (10,2) not null,
    Resistence decimal (10,2) not null);

    select * from support;
    CREATE TABLE graph_data (
    id INT not null PRIMARY KEY auto_increment,
    graph_image BLOB
);
```

#	ID	Price	Buy	Sell	BelowStopLos	Action
1	1	80.51	Yes	NO	NO	NO
2	2	81.07	NO	NO	NO	YES
3	3	81.05	NO	NO	NO	YES
4	4	81.10	NO	NO	NO	YES
5	5	81.01	NO	NO	NO	YES
6	6	81.17	NO	NO	NO	YES
7	7	81.04	NO	NO	NO	YES
8	8	80.90	Yes	NO	NO	NO
9	9	80.97	Yes	NO	NO	NO
10	10	81.07	NO	NO	NO	YES
11	11	80.48	Yes	NO	NO	NO
12	12	80.55	Yes	NO	NO	NO