STOCK MARKET SIGNALS

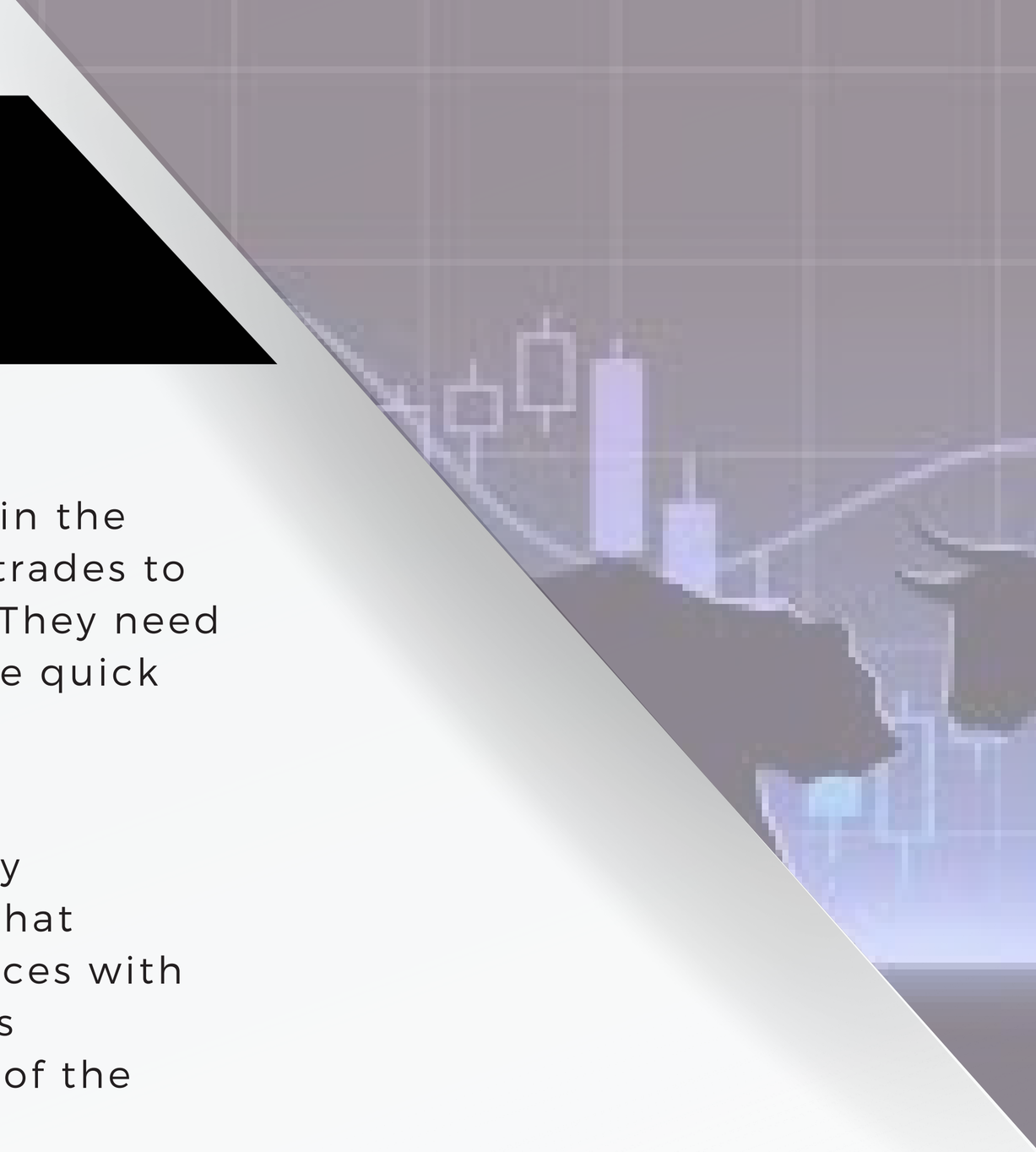
DATA ENGINEERING PROJECT

CLOUD & BIG DATA ENGINEER COURSE- MAY 2024

MIRI SHKOL

CONTENT

BACKGROUND



- Day traders buy and sell stocks within the same trading day, making multiple trades to profit from small price movements. They need timely, accurate information to make quick decisions.

- This project addresses their needs by providing real-time trading signals that combine up-to-the-minute stock prices with current market sentiment from news analysis, helping traders stay ahead of the market and make informed trades.

GOALS AND OBJECTIVES

Address Market Volatility

Provide real-time stock signals to help traders navigate the dynamic nature of the stock market.

Manage Information Overload

Implement tools that filter and interpret vast amounts of data quickly, giving traders clear and actionable insights.

Integrate Diverse Data Sources

Combine historical data, real-time price updates, and news sentiment analysis to deliver comprehensive and reliable trading signals.

TECHNICAL GOALS

01 **Utilize Course Technologies**

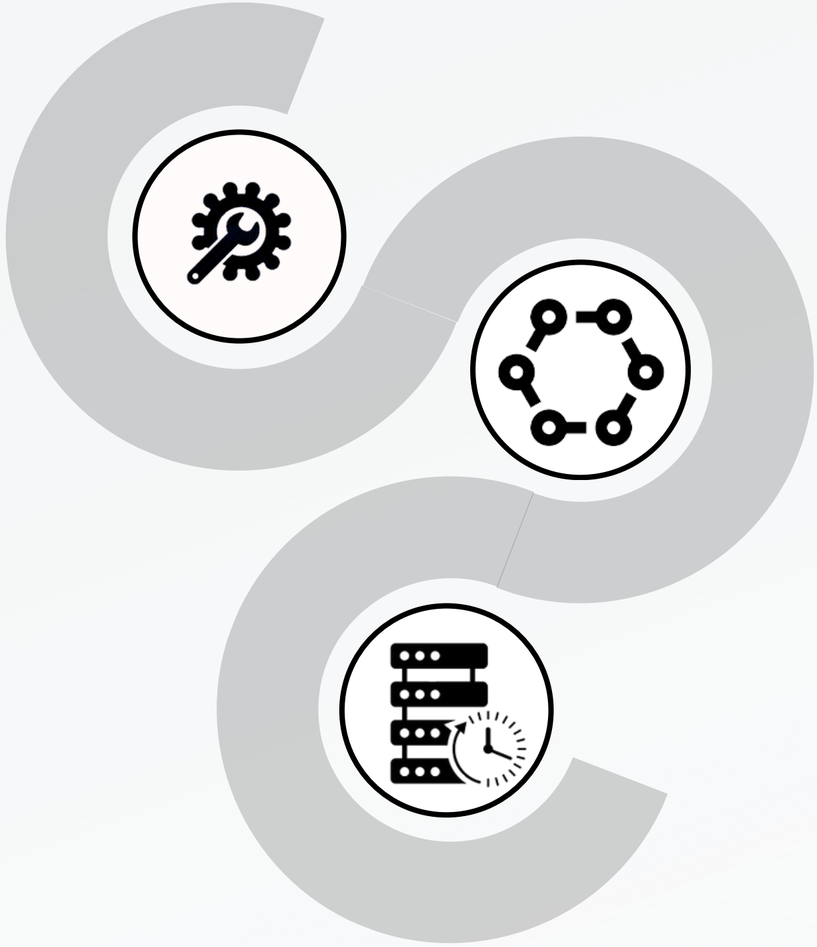
Apply tools and technologies learned in the course, such as Kafka, Spark, and HDFS.

02 **Pipeline Integration**

Perform integration between different tools and technologies to create a complete end-to-end pipeline.

03 **Real-Time Data Processing**

Implement real-time data processing to handle live stock prices and news sentiment analysis.



HIGH LEVEL ARCHITECTURE

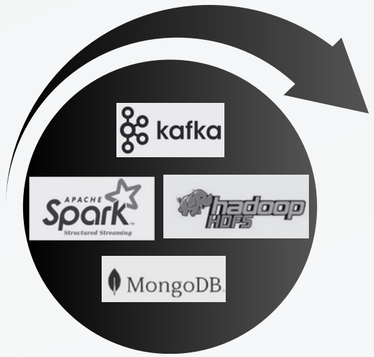
Data Sources

- Twelve Data API: Provides historical and real-time stock prices.
- News API: Provides financial news data.

Data Pipelines

- Historical Data Pipeline with daily update at the end of trading day,
- Real-Time Data Pipeline
- News Sentiment Analysis Pipeline

Processing and Storage

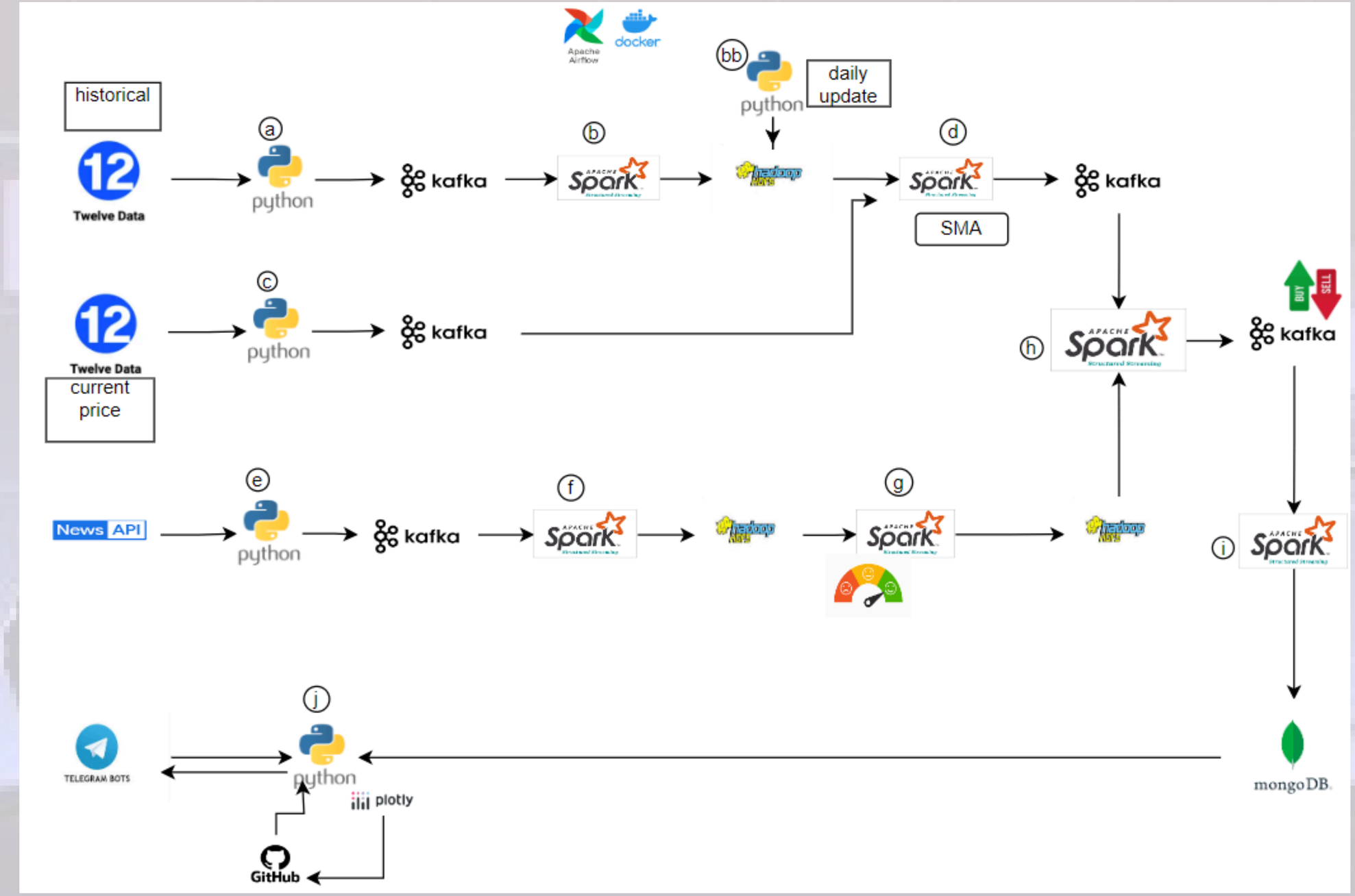


- **Apache Kafka**: Manages data streaming from various sources.
- **Apache Spark:** Handles data processing tasks.
- **HDFS :** Stores raw and processed data.
- **MongoDB:** Stores the final combined signals (SMA + sentiment) for easy retrieval.

Notifications

Telegram bot for sending real-time trading signals with plotly charts.

ARCHITECTURE



DETAILED DATA PIPELINES

THE PRICES PIPELINE

Historical Data Pipeline

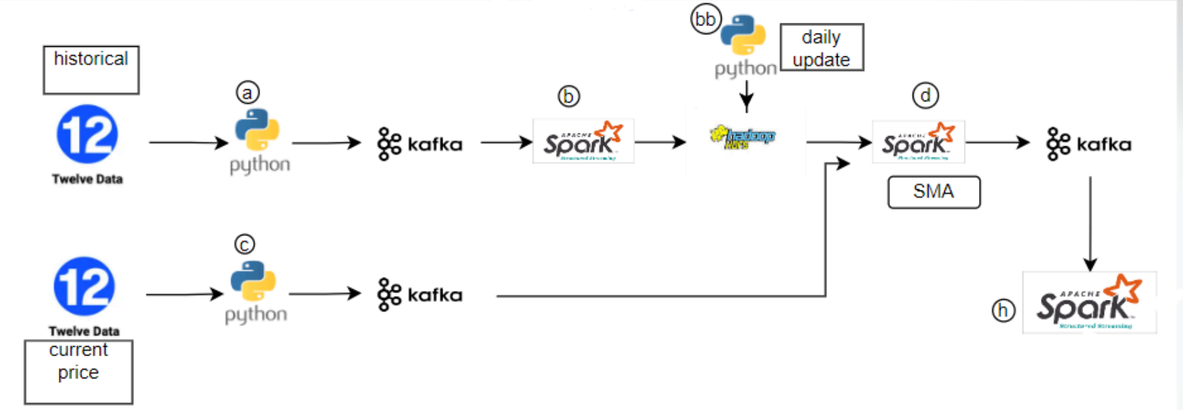
Collect daily closing prices to compute long term average and short term average. and also check if there is a crossover between these averages.

In ‘position’ column I track changes in the ‘short_above_long’ column over time.

- If it goes from 0 to 1, it means the short-term average price just crossed above the long-term average, suggesting a bullish (positive) trend.
- If it goes from 1 to 0, it means the short-term average price just crossed below the long-term average, suggesting a bearish (negative) trend.

Current Data Pipeline

Provide up-to-the-minute stock price updates and compare with historical prices averages.

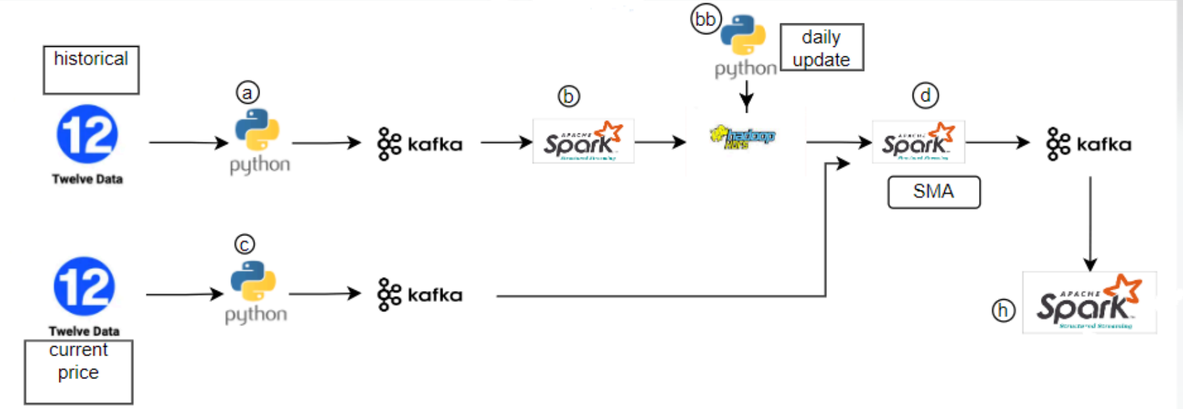



```python
# Create window specifications
short_window_spec = Window.orderBy("date").rowsBetween(-short_window_size + 1, 0)
long_window_spec = Window.orderBy("date").rowsBetween(-long_window_size + 1, 0)
previous_day_spec = Window.orderBy("date")

# Calculate the short-term and long-term SMAs
historical_df = historical_df.withColumn("SMA_short", F.avg(F.col("close_price")).over(short_window_spec))
historical_df = historical_df.withColumn("SMA_long", F.avg(F.col("close_price")).over(long_window_spec))

# Check if short above long
historical_df = historical_df.withColumn(
    "short_above_long",
    F.when(F.col("SMA_short") > F.col("SMA_long"), 1).otherwise(0)
)

# Calculate the positions
historical_df = historical_df.withColumn(
    "position",
        F.col("short_above_long") - F.lag(F.col("short_above_long"), 1)
    .over(previous_day_spec))
```

DETAILED DATA PIPELINES

THE PRICES PIPELINE

Combine both pipelines
Signals are determined based on the relationship between short-term and long-term averages and the current price, and the position changes:




```
processing_df = streaming_df_with_avg \
    .withColumn("signal",
        F.when(F.col("short_above_long") & F.col("is_significant"),
               F.when((F.col("close") > F.col("short_term_avg")), "strong_buy")
                .otherwise("hold"))
        .otherwise(
            F.when(F.col("short_above_long") & ~F.col("is_significant"),
                   F.when((F.col("close") > F.col("short_term_avg")) & (F.col("position") == "buy"), "strong_buy")
                   .when((F.col("close") > F.col("short_term_avg")) & (F.col("position") != "buy"), "buy")
                   .when((F.col("close") < F.col("long_term_avg")) & (F.col("Position") != "buy"), "sell")
                   .otherwise("hold"))
            .otherwise(
                F.when(~F.col("short_above_long") & F.col("is_significant"),
                    F.when((F.col("close") < F.col("short_term_avg")), "strong_sell")
                    .otherwise("hold"))
                .otherwise(
                    F.when(~F.col("short_above_long") & ~F.col("is_significant"),
                        F.when((F.col("close") < F.col("short_term_avg")) & (F.col("Position") == "sell"), "strong_sell")
                        .when((F.col("close") < F.col("short_term_avg")) & (F.col("Position") != "sell"), "sell")
                        .when((F.col("close") > F.col("long_term_avg")) & (F.col("Position") != "sell"), "buy")
                        .otherwise("hold"))
                    )
                )
            )
    )
```

DETAILED DATA PIPELINES

THE NEWS PIPELINE

Purpose

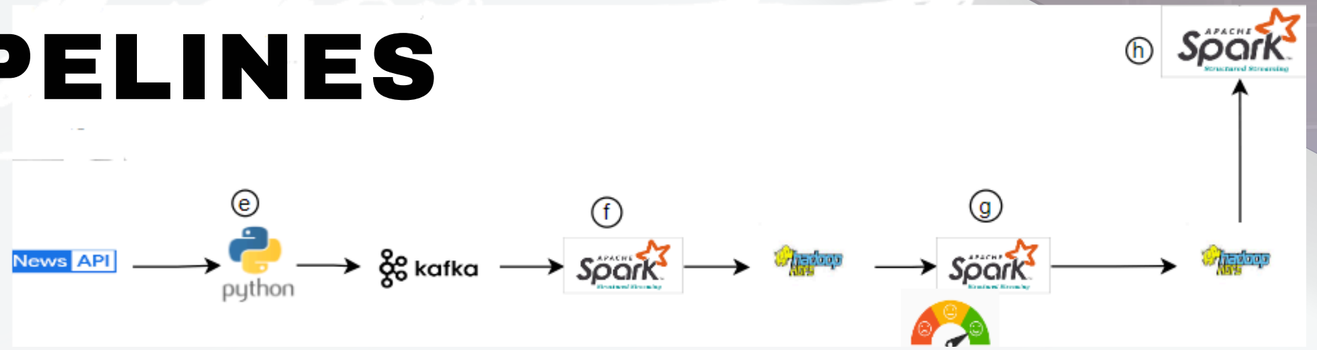
The news pipeline is designed to ingest, process, and analyze financial news data to extract sentiment, which is then combined with stock price signals to enhance trading decision-making.

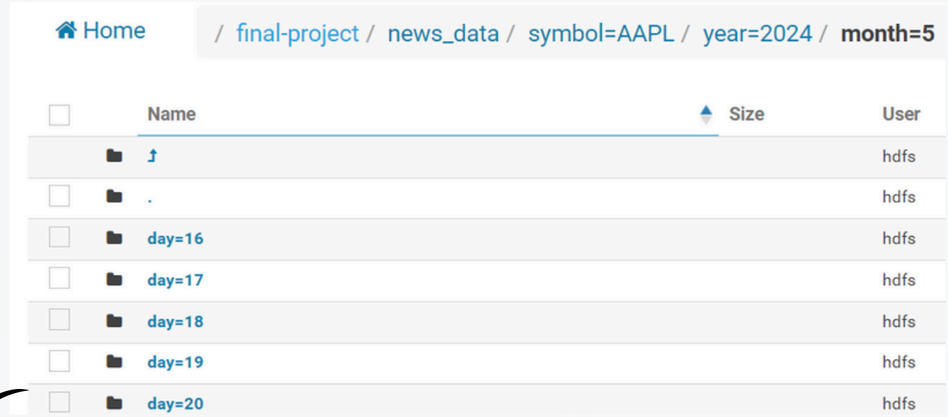
News Ingestion and Language Detection

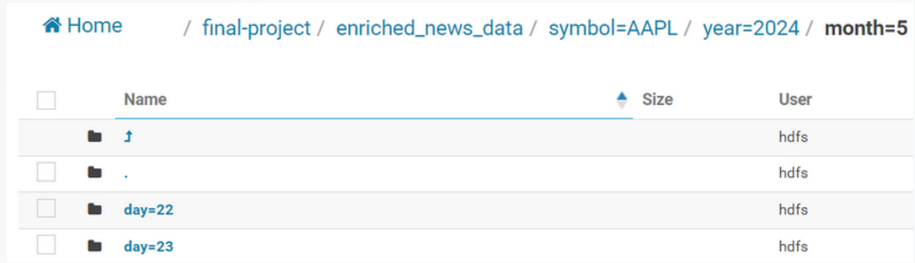
The news pipeline initiates by fetching news articles published within the past 24 hours for analysis.
Articles are filtered based on the detected language, retaining only those identified as English.

Sentiment Analysis

Incorporating sentiment analysis using TextBlob enhances the stock market signals with emotional tone insights from financial news.
It analyzes the sentiment of news articles to determine whether they are positive, negative, or neutral.







DETAILED DATA PIPELINES

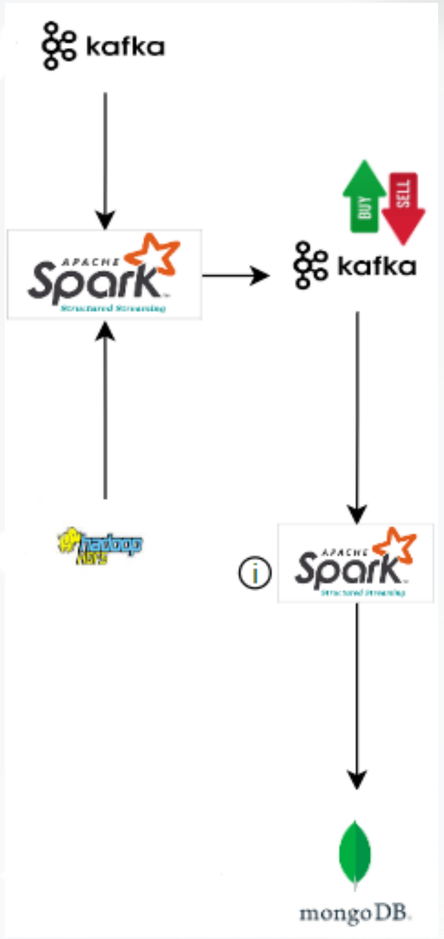
COMBINE BOTH PIPELINES FOR SIGNALS

Aggregated Sentiment

Compute aggregated sentiment metrics, including:

- Positive Count & Percentage: Number and percentage of positive sentiment articles.
- Negative Count & Percentage: Number and percentage of negative sentiment articles.
- Average Sentiment Score: Average sentiment score for positive and negative articles.

```
# Aggregate sentiment data
if news_sentiment_df:
    total_count = news_sentiment_df.count()

    # Calculate counts and percentage of positive and negative sentiments
    sentiment_aggregated_df = news_sentiment_df.groupBy("symbol").agg(
        F.count(F.when(F.col("sentiment_score") > 0, True)).alias("positive_count"),
        (F.count(F.when(F.col("sentiment_score") > 0, True)) / total_count * 100).alias("positive_percentage"),
        F.avg(F.when(F.col("sentiment_score") > 0, F.col("sentiment_score"))).alias("positive_avg_score"),
        F.count(F.when(F.col("sentiment_score") < 0, True)).alias("negative_count"),
        (F.count(F.when(F.col("sentiment_score") < 0, True)) / total_count * 100).alias("negative_percentage"),
        F.avg(F.when(F.col("sentiment_score") < 0, F.col("sentiment_score"))).alias("negative_avg_score"),
        F.count(F.when(F.col("sentiment_score") == 0, True)).alias("neutral_count"),
    )

    # Define buy, sell, hold signals based on sentiment analysis results
    sentiment_agg_df = sentiment_aggregated_df.withColumn(
        "sentiment_signal",
        F.when(((F.col("positive_percentage") > 50) & (F.col("positive_avg_score") > 0.3))
               |(F.col("positive_percentage") == 100), "buy")
        .when((F.col("positive_percentage") > 55) & (F.col("positive_avg_score") > 0.6), "strong_buy")
        .when(((F.col("negative_percentage") > 50) & (F.col("negative_avg_score") < -0.3))
              |(F.col("negative_percentage") == 100), "sell")
        .when((F.col("negative_percentage") > 55) & (F.col("negative_avg_score") < -0.6), "strong_sell")
        .otherwise("hold")
    )
```

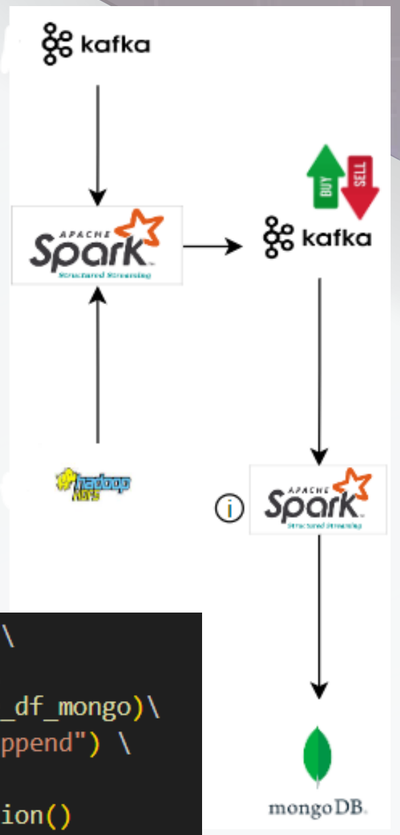


DETAILED DATA PIPELINES

COMBINE BOTH PIPELINES FOR SIGNALS

The strategy integrates the signals from SMA and aggregated sentiment analysis to generate a combined signal that takes into account both technical and sentiment indicators.

```
combined_df= combined_df\
.withColumn(
    "combined_signal",
    F.when(
        (parsed_sma_df["sma_signal"] == "strong_buy") &
        ((F.col("sentiment_signal") == "strong_buy") | (F.col("sentiment_signal") == "buy"))
        , "strong_buy"
    )
    .when(
        (parsed_sma_df["sma_signal"] == "strong_buy") &
        ((F.col("sentiment_signal") == "hold") |(F.col("sentiment_signal") == "sell"))
        , "buy"
    )
    .when(
        (parsed_sma_df["sma_signal"] == "strong_buy") &
        (F.col("sentiment_signal") == "strong_sell")
        , "hold"
    )
    .when(
        (parsed_sma_df["sma_signal"] == "buy") &
        (F.col("sentiment_signal") == "strong_buy")
        , "strong_buy"
    )
    .when(
        (parsed_sma_df["sma_signal"] == "buy") &
        ((F.col("sentiment_signal") == "buy") |(F.col("sentiment_signal") == "hold"))
        , "buy"
    )
```




```
parsed_signals_df \
    .writeStream \
    .foreach(write_df_mongo)\
    .outputMode("append") \
    .start() \
    .awaitTermination()
```


```
{
  "_id": {...},
  "symbol": "AAPL",
  "kafka_datetime": "2024-05-24T19:38:04.184000",
  "datetime": "2024-05-24T15:36:00",
  "open": 190.21001,
  "high": 190.215,
  "low": 190.11,
  "close": 190.11,
  "volume": 60694,
  "short_term_avg": 189.058998,
  "long_term_avg": 180.63560040000004,
  "short_above_long": true,
  "is_significant": true,
  "sma_signal": "strong_buy",
  "sentiment_signal": "buy",
  "combined_signal": "strong_buy"
}
```

SEE THE PROCESS

Offset: 1058 Key: AAPL Timestamp: 2024-05-24 19:38:03.979 Headers: empty


```
{
    "symbol": "AAPL",
    "datetime": "2024-05-24 15:36:00",
    "open": "190.21001",
    "high": "190.21500",
    "low": "190.11000",
    "close": "190.11000",
    "volume": "60694"
}
```


Home / final-project / historical_stock_data / symbol=AAPL

Name	Size
↑	
.	
_SUCCESS	0 bytes
year=2023	
year=2024	

Offset: 3567 Key: AAPL Timestamp: 2024-05-24 19:38:04.083 Headers: empty


```
{
    "symbol": "AAPL",
    "datetime": "2024-05-24T15:36:00.000Z",
    "open": 190.21001,
    "high": 190.215,
    "low": 190.11,
    "close": 190.11,
    "volume": 60694,
    "short_term_avg": 189.058998,
    "long_term_avg": 180.63560040000004,
    "short_above_long": true,
    "is_significant": true,
    "signal": "strong_buy"
}
```


Home / final-project / enriched_news_data / symbol=AAPL / year=2024 / month=5

Name	Size	User
↑		hdfs
.		hdfs
day=22		hdfs
day=23		hdfs

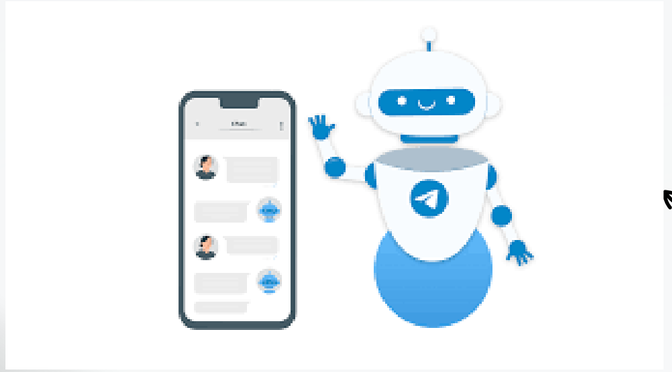
Offset: 2572 Key: AAPL Timestamp: 2024-05-24 19:38:04.184 Headers: empty


```
{
    "symbol": "AAPL",
    "datetime": "2024-05-24T15:36:00.000Z",
    "open": 190.21001,
    "high": 190.215,
    "low": 190.11,
    "close": 190.11,
    "volume": 60694,
    "short_term_avg": "189.058998",
    "long_term_avg": "180.63560040000004",
    "short_above_long": "true",
    "is_significant": "true",
    "sma_signal": "strong_buy",
    "sentiment_signal": "buy",
    "combined_signal": "strong_buy"
}
```

SEE THE PROCESS

Offset: 2572 Key: AAPL Timestamp: 2024-05-24 19:38:04.184 Headers: empty


```
{
    "symbol": "AAPL",
    "datetime": "2024-05-24T15:36:00.000Z",
    "open": 190.21001,
    "high": 190.215,
    "low": 190.11,
    "close": 190.11,
    "volume": 60694,
    "short_term_avg": "189.058998",
    "long_term_avg": "180.63560040000004",
    "short_above_long": "true",
    "is_significant": "true",
    "sma_signal": "strong_buy",
    "sentiment_signal": "buy",
    "combined_signal": "strong_buy"
}
```

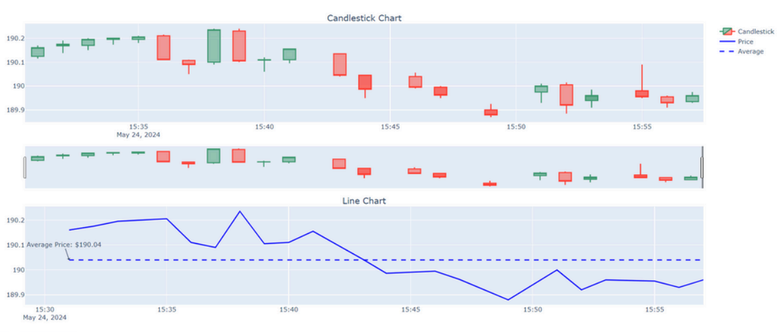

```
{
    "_id": {...},
    "symbol": "AAPL",
    "kafka_datetime": "2024-05-24T19:38:04.184000",
    "datetime": "2024-05-24T15:36:00",
    "open": 190.21001,
    "high": 190.215,
    "low": 190.11,
    "close": 190.11,
    "volume": 60694,
    "short_term_avg": 189.058998,
    "long_term_avg": 180.63560040000004,
    "short_above_long": true,
    "is_significant": true,
    "sma_signal": "strong_buy",
    "sentiment_signal": "buy",
    "combined_signal": "strong_buy"
}
```

DETAILED DATA PIPELINES

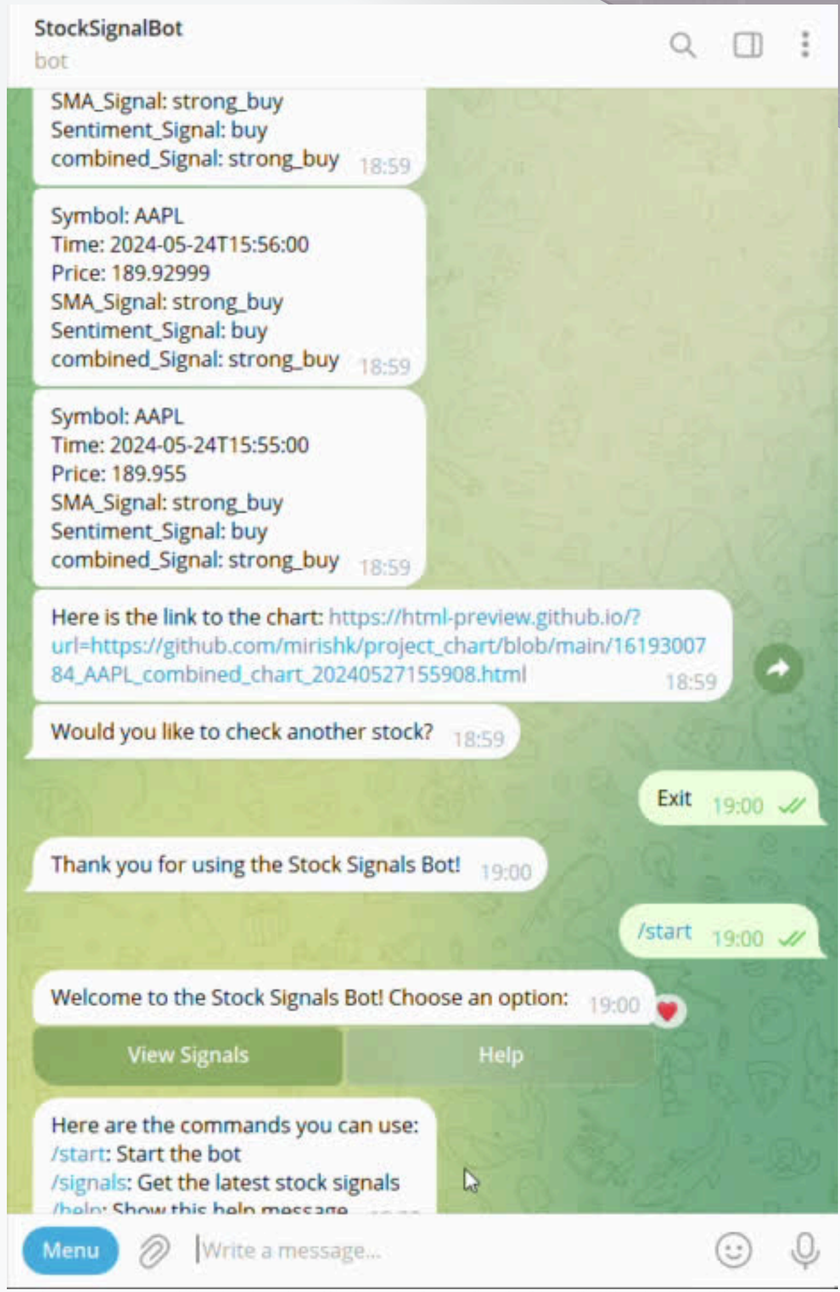
SEND SIGNALS TO USERS VIA TELEGRAM BOT

The Telegram bot enables users to interactively request and receive the latest stock market signals and visualizations.

- Signal Delivery: The bot fetches the latest 3 signals from MongoDB and sends them to the user.
- Chart Visualizations: The bot also generates and sends a link to line chart and candlestick chart for the chosen stock.



https://html-preview.github.io/?url=https://github.com/mirishk/project_chart/blob/main/1619300784_AAPL_combined_chart_20240527160103.html



ORCHESTRATION

Apache Airflow

Apache Airflow is a powerful platform for orchestrating complex data workflows. It allows you to schedule, monitor, and manage data pipelines, ensuring they run reliably and efficiently.
In this scenario, we have three distinct DAGs to handle different aspects of our data processing needs:

- **Daily Historical Data Update DAG**- To update close price at the end of each trading day.
- **Streaming Pipeline Dag**- To process real-time streaming data during trading hours.
- **Batch News Pipeline DAG**- To process news articles related to stocks at specific intervals during trading hours.

01

```
# Define the DAG
us_tz = pendulum.timezone("America/New_York")  # Set to US Eastern Timezone
dag = DAG(
    'dag_update_hdfs_end_of_trading_day',
    description='Update HDFS file with day close price at end of trading day',
    schedule_interval='5 16 * * 1-5',  # At 4:05 PM every trading day
    start_date=datetime(2024, 5, 23, tzinfo=us_tz),  # Start date with timezone
    catchup=False,
)
```

02

```
# Define the DAG
us_tz = pendulum.timezone("America/New_York")  # Set to US Eastern Timezone
dag = DAG(
    'realtime_pipeline_scripts_dag',
    description='Run trading hour scripts in real-time',
    schedule_interval='30 9 * * 1-5',  # At 09:30 on every day-of-week from Monday through Friday.
    start_date=datetime(2024, 5, 23, tzinfo=us_tz),  # Start date with timezone
    catchup=False,
)
```

03

```
# Define the DAG
dag = DAG(
    'news_pipeline_scripts_dag',
    default_args=default_args,
    description='A DAG to schedule news pipline scripts',
    schedule_interval= '30 9-16 * * 1-5',
    # At minute 30 past every hour from 9 through 16 on every day-of-week from Monday through Friday.
    catchup=False  # Don't catch up on any missed executions
)
```

ORCHESTRATION

Apache Airflow

DAG: dag_update_hdfs_end_of_trading_day Update HDFS file with day close price at end of trading day

Tree | Graph | Calendar | Task Duration | Task Tries | Landing Times | Gantt | Details | Code

2024-05-23T12:02:31Z | Runs 25 | Update

PythonOperator | queued | running | success | failed | up

May 23, 13:54 — May 23, 15:02

[DAG]
update_hdfs_with_day_close_price

DAG: realtime_pipeline_scripts_dag Run trading hour scripts in real-time

Tree | Graph | Calendar | Task Duration | Task Tries | Landing Time

2024-05-23T13:30:00Z | Runs 25 | Update

PythonOperator

May 23, 12:57 — May 23, 16:30

[DAG]
calculate_end_trading_day
run_producer_task
run_consumer_sma_signals_task
run_consumer_combine_signals_task
run_consumer_to_mongo_task
run_telegram_bot_task

DAG: news_pipeline_scripts_dag A DAG to schedule news pipline scripts

Tree | Graph | Calendar | Task Duration | Task Tries | Landing Times

2024-05-24T19:30:00Z | Runs 25 | Update

DummyOperator | PythonOperator

May 22, 22:30 — May 23, 17:30 — May 24, 22:30

[DAG]
check_trading_time
run_producer_task
run_spark_consumer_task
run_spark_enrichment_task
skip_all_tasks

TOOLS SELECTION RATIONALE

Kafka is used for real-time data streaming and message brokering between different components of the pipeline.

Reasons for Choosing Kafka

- **High Throughput**: Efficiently handles large data volumes with low latency.
- **Scalability**: Easily scales horizontally for growing data loads.
- **Reliability:** Provides durable, fault-tolerant log-based storage.
- **Integration**: Seamlessly works with Apache Spark and HDFS.

Alternatives Considered

- **RabbitMQ**: Better for complex routing but not as efficient for high-volume data streams.
- **Apache Pulsar:** Similar features but more complex and smaller community.

Why Kafka is Better

- **Performance:** Ideal for high-throughput, low-latency applications
- **Ecosystem**: Extensive community support.
- **Scalability:** Grows easily with project needs.

TOOLS SELECTION RATIONALE

Spark is used for data processing, combining historical and real-time data, and performing sentiment analysis.

Reasons for Choosing Spark

- **Speed:** In-memory processing enables fast computation for real-time analysis.
- **Versatility:** Supports multiple languages and high-level APIs for streaming, machine learning, and more.
- **Integration:** Seamlessly integrates with Kafka for real-time data and HDFS for storage.

Why Spark is Better

- **Performance and Flexibility:** In-memory processing and comprehensive ecosystem make Spark ideal for both real-time and batch processing.

Alternatives Considered

- **Hadoop MapReduce:** Reliable for batch processing but lacks real-time capabilities.
- **Apache Flink**: Excellent for stream processing but less flexible for combined batch and stream processing.

TOOLS SELECTION RATIONALE

MongoDB stores final combined data (price signals and sentiment analysis) and serves it to the Telegram bot.

Reasons for Choosing MongoDB

- **Schema Flexibility:** Schema-less design for storing varied and complex data types.
- **Scalability:** Handles large data volumes and scales horizontally.
- **Real-Time Capabilities:** Supports real-time read and write operations.
- **JSON Support:** Naturally handles JSON-like documents, making it easy to integrate with Kafka,

Why MongoDB is Better

- **Flexibility, Scalability, Real-Time Operations, and JSON Support:** Ideal for diverse, growing datasets and real-time data handling, and integrates smoothly with Kafka's JSON data streams.

Alternatives Considered

- **MySQL:** Strong consistency but less flexible and scalable for diverse datasets.
- **PostgreSQL:** Offers strong relational integrity but lacks MongoDB's flexibility and scalability.

TOOLS SELECTION RATIONALE

HDFS is used for long-term storage of historical price data and sentiment analysis results.

Reasons for Choosing HDFS

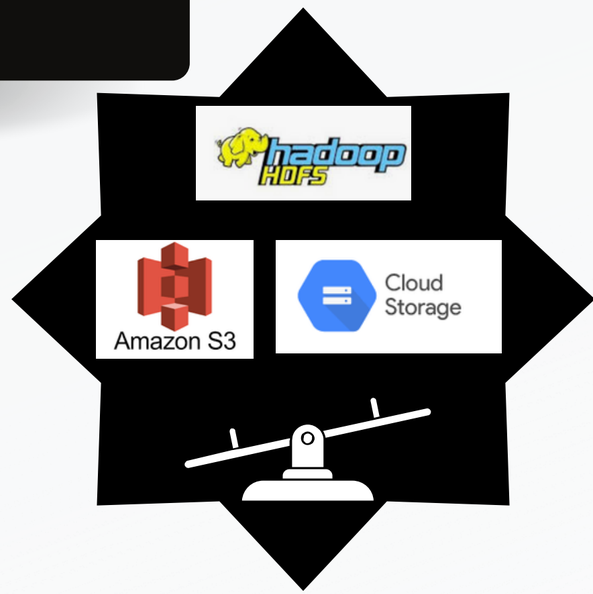
- **Reliability:** Fault tolerance and high throughput for storing large data volumes reliably.
- **Scalability:** Manages vast amounts of data across many nodes, ideal for long-term storage.
- **Cost-Effectiveness:** Economically stores large datasets, making it a cost-effective solution.

Why HDFS is Better

- **Large-Scale Storage, Fault Tolerance, and Cost-Effectiveness:** Best choice for handling extensive data storage needs economically and reliably.

Alternatives Considered

- **Amazon S3:** Scalable but incurs ongoing costs and requires AWS integration.
- **Google Cloud Storage**: Scalable but involves cloud storage costs and dependency on GCP.



CHALLENGES

stackoverflow ChatGPT

configuration

Python Version Compatibility: The Telegram bot dependencies required Python 3.8, but the default environment was Python 3.6.
Java Version Compatibility: Spark required Java 8, but the system initially had a newer version (Java 11).

Preventing Duplicates in Batch in the news pipeline

In batch processing, startingOffsets = "latest" is not applicable. Therefore, I used timestamps to ensure processing the most recent data and prevented duplicates.

```
# If no max timestamp from HDFS, start from earliest
if start_from_earliest:
    kafka_options = {
        "kafka.bootstrap.servers": bootstrap_servers,
        "subscribe": topic_name,
        "startingOffsets": "earliest"
    }

else:
    kafka_options = {
        "kafka.bootstrap.servers": bootstrap_servers,
        "subscribe": topic_name,
        "startingOffsetsByTimestamp": starting_offsets_json
    }

news_df = spark \
            .read \
            .format("kafka") \
            .options(**kafka_options) \
            .load()
```

```
base_path = f"{hdfs_path}//final-project/news_data/symbol={symbol}"
try:
    # Read data from the specified path
    df = spark.read.parquet(base_path)
    # Check if DataFrame is empty
    if df.count() == 0:
        raise ValueError("DataFrame is empty")

    # Find the maximum value of the timestamp column
    max_timestamp_row = df.agg(F.max(F.col("timestamp")).alias("max_timestamp")).collect()[0]

    # Extract max_timestamp from the Row object
    max_timestamp = max_timestamp_row["max_timestamp"]

    starting_offset_timestamp_ms = int(max_timestamp.timestamp() * 1000)

    # Convert the timestamp to a JSON string with the appropriate topic and partition offsets
    starting_offsets_json = '{"' + topic_name + '": {"0": ' + str(starting_offset_timestamp_ms) + '}}'

    print("Max Timestamp:", max_timestamp, " ",starting_offset_timestamp_ms, " ", starting_offsets_json)

except Exception as e:
    print("An error occurred:", str(e))
    print("Starting from earliest...")
    start_from_earliest = True
    max_timestamp = None
```

Language Filtering in News API

The News API's language parameter, intended to filter news articles in English, proved unreliable.
To ensure only English articles were processed, the langdetect library was used for accurate language detection.

NEXT STEPS

Additional Indicators from Twelve Data API

Utilize more technical indicators available from the Twelve Data API, such as RSI, MACD, Bollinger Bands, and others, to provide more comprehensive trading signals.

Research High-Frequency News APIs

Search for and integrate a high-frequency, near real-time news API to capture the latest market-moving news events as they happen. enhancing the timeliness of sentiment-based signals.

User Flexibility and Customization

Allow users to customize their preferences for the types of indicators and signals they want to receive. Implement user-specific alert thresholds and conditions for notifications.

Machine Learning Integration

Integrate machine learning models to predict stock price movements based on historical data and news sentiment.

Use ML models to detect anomalies or unusual patterns in stock prices and news sentiment that may indicate significant market events.

WWW.LINKEDIN.COM/IN/MIRI-SHKOL/