ĐẠI HỌC BÁCH KHOA HÀ NỘI
Course: Big Data Storage and Processing

Real-Time Stock Sentiment Analysis Pipeline

Group 14

Le Ngoc Binh	20214878
Hoang Trung Khai	20225502
Nguyen Viet Anh	20225434
Trinh Duy Phong	20220065
Luu Hoang Phan	20225516

Problem Statement

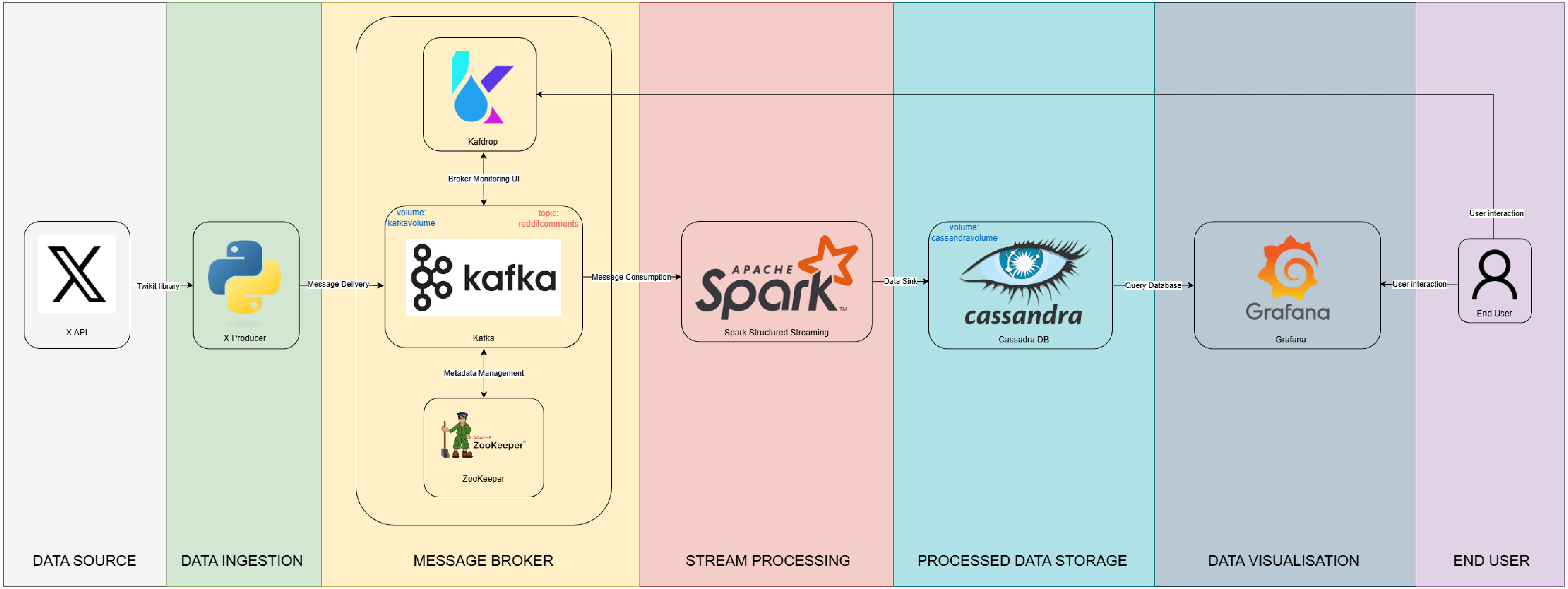
Objectives

- Build a complete **end-to-end** system capable of **collecting, processing,** and **performing real-time** sentiment analysis on **social media data** (Twitter/X).

Key Highlights

- Build a real-time streaming data pipeline
- Microservices-based architecture for modularity and flexibility.
- Process data using **Kafka + Spark Streaming**
- Fully containerized deployment using ***Docker***.
- ***Kubernetes (Minikube)*** orchestration for scalability and reliability.
- Infrastructure managed as Code (IaC) with ***Terraform*** for consistent and repeatable deployments.
- Visualize insights using **Grafana dashboards**

Overview



X API
Twikit library
X Producer
Message Delivery
Kafdrop
Broker Monitoring UI
volume:
kafkavolume
topic:
redditcomments
kafka
Kafka
Metadata Management
APACHE
ZooKeeper
ZooKeeper
Message Consumption
APACHE
Spark
Spark Structured Streaming
Data Sink
volume:
cassandravolume
cassandra
Cassadra DB
Query Database
Grafana
Grafana
User interaction
User interaction
End User
DATA SOURCE
DATA INGESTION
MESSAGE BROKER
STREAM PROCESSING
PROCESSED DATA STORAGE
DATA VISUALISATION
END USER

I. Data Ingestion

This step collects and generates data into the system

1

Twitter Producer

- Stimulates live stock-related tweets
- Reads historical dataset (Hugging Face)
- Overwrites timestamps with current UTC time
- Sends JSON messages to Kafka topic tweets

2

Stock Producer

- Simulates minute-level OHLCV stock prices
- Reads static CSV data
- Generates real-time trading minutes
- Sends data to Kafka topic stock-prices

I. Data Ingestion

This step collects and generates data into the system

1

Twitter Producer

- Stimulates live stock-related tweets
- Reads historical dataset (Hugging Face)
- Overwrites timestamps with current UTC time
- Sends JSON messages to Kafka topic tweets

2

Stock Producer

- Simulates minute-level OHLCV stock prices
- Reads static CSV data
- Generates real-time trading minutes
- Sends data to Kafka topic stock-prices

3

Kafka Cluster

- Acts as a **buffer and decoupling layer** on **Kubernetes with Zookeeper**
- Ensures scalability and fault tolerance
- Topics: *tweets, stock-prices*

Data Flow

Stage 1 - Generation

Producers generate simulated real-time data

Tweet JSON	{ "created_at": "2025-12-21T14:30:00.123456", "text": "Huge breakout for $AAPL today! Testing new highs." }
Stock Price JSON	{ "symbol": "AAPL", "trade_timestamp": "2025-12-21T14:30:00", "open_price": 150.5, "close_price": 151.2, "high_price": 151.5, "low_price": 150.0, "volume": 551923 }

Data Flow

Stage 1 - Generation

Producers generate simulated real-time data

Stage 2 - Buffering

Incoming messages are buffered in **Kafka Topics** *(tweets, stock-prices) under* UTF-8 JSON.

The data structure is exactly the same as in previous step.

Data Flow

Stage 1 - Generation

Producers generate simulated real-time data

Stage 2 - Buffering

Incoming messages are buffered in **Kafka Topics** *(tweets, stock-prices) under* UTF-8 JSON.

The data structure is exactly the same as in previous step.

Stage 3 - Processing

Spark consumes and processes streams

Spark reads JSON data, converts it into a DataFrame, and doing ETL & Analytics

Step 1: Raw Parsing (Tweet)

- Extract symbol: $AAPL → AAPL
- Convert created_at → api_timestamp

Step 2: Compute *sentiment_score*

Step 3: Aggregation: Data is aggregated by time window and topic.

Data Flow

Stage 1 - Generation

Producers generate simulated real-time data

Stage 2 - Buffering

Incoming messages are buffered in **Kafka Topics** *(tweets, stock-prices) under* UTF-8 JSON.

The data structure is exactly the same as in previous step.

Stage 3 - Processing

Spark consumes and processes streams

Spark reads JSON data, converts it into a DataFrame, and doing ETL & Analytics

Stage 4 - Storage

Store the data in MinIO (Data Lake) and Cassandra (Serving Layer).

Data Flow

Stage 1 - Generation

Producers generate simulated real-time data

Stage 2 - Buffering

Incoming messages are buffered in **Kafka Topics** *(tweets, stock-prices) under* UTF-8 JSON.

The data structure is exactly the same as in previous step.

Stage 3 - Processing

Spark consumes and processes streams

Spark reads JSON data, converts it into a DataFrame, and doing ETL & Analytics

Stage 4 - Storage

Store the data in MinIO (Data Lake) and Cassandra (Serving Layer).

Stage 5 - Visualization

Grafana queries Cassandra to plot charts and visualize the data

II. Speed Layer

- Engine: **Apache Spark Structured Streaming 3.4.0 (Python/PySpark).**
- Act as the backbone of the system, where raw data is transformed into information.

Input:

tweets	**stock-prices**
Structure: • created_at: StringType • text: StringType	Structure: • symbol: StringType • trade_timestamp: StringType • open_price: DoubleType • high_price: DoubleType • low_price: DoubleType • close_price: DoubleType • volume: LongType

II. Speed Layer

Spark performs a series of transformations on the data stream (DStream/DataFrame):

01

Read & Parse input

Read data from Kafka and decode JSON into columns.

02

Enrichment & Sentiment Analysis

Use RegEx to extract stock symbols

Calculate Sentiment

Create a unique ID for each record

03

Window Aggregation

Aggregate data by time windows (5-minute window, 1-minute sliding) and by topic.

Metrics:

- Average sentiment score
- Bullish tweet count
- Bearish tweet count
- Neutral tweet count

II. Speed Layer

Streaming Writers (Output)

- **Writer 1 (Bronze - MinIO):** Writes all `enriched_tweets` to MinIO in Parquet format, partitioned by `topic`.
- **Writer 2 (Speed - Cassandra Raw):** Writes each individual tweet record to the `twitter.tweets` table.
- **Writer 3 (Speed - Cassandra Market):** Writes stock price data to the `twitter.market_data` table.
- **Writer 5 (Gold - Cassandra Analytics):** Writes aggregated results (`windowed_sentiment`) to the `twitter.topic_sentiment_avg` table.

III. Batch Processing Layer

Overview

- The analytical engine of our Lambda Architecture, responsible for deep, historical analysis of "Big Data."
- Corrects potential errors or gaps from the real-time stream layer.

Input & Technology

- Engine : Apache Spark 3.4 (PySpark)
- Input Source : MinIO (S3 Compatible Data Lake)
- Data Format : Parquet (A highly efficient columnar storage format)

III. Batch Processing Layer

This layer performs three heavy-duty analyses:

01

Market Heatmap

Groups millions of data points by the hour and pivots the data to create a sentiment matrix with time slots as rows and stock symbols as columns.

02

Volatility & Risk Profiling

Calculates Standard Deviation to measure volatility and uses Median and 95th Percentile to identify the true central tendency, ignoring outliers.

03

Long-Term Trend Identification

Applies a Moving Average over a sliding window to smooth out noisy real-time data and reveal the underlying market trend

III. Batch Processing Layer

Desired Output (The "Gold" Layer)

The processed, high-value data is stored in a "Gold" layer in our data lake, ready for BI tools and dashboards.

Performance Optimizations

- **In-Memory Caching :** The source dataset is loaded into RAM once and reused for all three transformations, avoiding multiple slow disk reads.
- **Partitioning :** Results are written back to MinIO partitioned by topic (e.g., AAPL , GOOGL). This allows for highly efficient queries, as analysts can read data for a single stock without scanning the entire dataset.

III. Storage Layer

- The system uses a **Hybrid Storage Architecture**
- Designed to support:
    - Fast real-time queries (Dashboards)
    - Long-term storage (Analytics & ML)
- Core technologies:
    - **Apache Cassandra** – Speed / Serving Layer
    - **MinIO** – Data Lake (Bronze Layer)

III. Storage Layer

3.1. Apache Cassandra (Speed Layer / OLTP Database)

- **Role:**
  - High-performance NoSQL database
  - Serves **real-time queries** from Grafana dashboards
- **Key characteristics:**
  - Low latency
  - Horizontally scalable
- **Keyspace:**
  - twitter
  - Replication Factor: 1 (development environment)

III. Storage Layer

3.1. Apache Cassandra (Speed Layer / OLTP Database)

Cassandra Table:

- ***tweets****:* Store individual tweets after enrichment, used for detailed inspection
- ***market_data***: Store market price data, enables correlation between sentiment and price
- ***topic_sentiment_avg***: Store windowed sentiment analytics. This is data source for Grafana charts

III. Storage Layer

Apache Cassandra Table Examples

tweets

api_timestamp	sentiment_score	text
2025-12-21 07:22:05.727000+0000	0.4404	$AAPL truly the scumbaggiest stock that ever existed

market_data

symbol	trade_timestamp	close_price	volume
AAPL	2025-12-22 13:54:13.000000+0000	68.39574	551923

topic_sentiment_avg

topic	ingest_timestamp	sentiment_score_avg	bullish_count	bearish_count	neutral_count
AAPL	2025-12-21 07:18:00.000000+0000	0.06764	1	0	4

III. Storage Layer

2. MinIO (Data Lake)

- S3-compatible object storage
- **Stores:** Raw & semi-processed tweet data
- **Format:** Parquet
- **Used for:**
  - Batch analytics
  - Data science & ML workloads
- Data is automatically partitions by topic

III. Storage Layer

2. MinIO (Data Lake)

Example Structure

```
/data/twitter-bronze/tweets_v2/
├── topic=AAPL/
│   ├── part-00000-....c000.snappy.parquet (created 2025-12-21 07:xx:xx)
│   ├── part-00001-....c000.snappy.parquet (created 2025-12-21 07:yy:yy)
├── topic=GOOGL/
│   ├── part-00000-....c000.snappy.parquet
└── _spark_metadata/
```

IV. Serving Layer

- The end-user interface for monitoring the market.
- Built with **Grafana** on top of **Apache Cassandra**

Grafana Setup

- URL: *http://localhost:3000* (via Port Forwarding)
- Datasource:
    - Plugin: *hadesarchitect-cassandra-datasource*
    - Connection: *cassandra:9042*
- Template Variable (*$topic*):
    - Type: Static list of stock tickers (AAPL, GOOGL, TSLA,...)
    - Used to filter all panels in the dashboard

Grafana

IV. Serving Layer

Fear & Greed Index

- Visualization: Gauge
- Purpose: Display the latest market sentiment from ‘*twitter.topic_sentiment_avg*’

Momentum: Bulls vs Bears

- Visualization: Time Series (Line Chart).
- Purpose: Compare bullish vs bearish tweet volume over time from ‘twitter.topic_sentiment_avg’

Market Composition

- Visualization: Time Series (Stacked Area Chart)
- Purpose: Show overall discussion volume, visualize sentiment distribution

Stock Price History

- Visualization: Time Series (Line Chart)
- Purpose: Track stock price movement over time, correlate price trends with sentiment changes

Live Tweets Stream

- Visualization: Table
- Purpose: Display real tweet content in near real-time

Conclusion

The system has successfully implemented all core functionalities and demonstrated stable real-time data stream processing capabilities.

- Real-time sentiment analysis and stock price tracking.
- Managed with Terraform to ensure efficient and reproducible deployments.
- Built using Kafka, Spark, Cassandra, and Kubernetes.
- Grafana dashboards provide deep market insights.

This is
the end of
representation