BUILDING A REAL-TIME STOCK MARKET DATA PIPELINE WITH KAFKA & AWS

By Winardi

HI, I AM
WINARDI

Passionate about building reliable data pipelines and scalable data systems, with a strong foundation in statistics. Focused on data engineering to ensure clean and trustworthy data for better decision-making.

Course License :

- Data Engineer –DSAREA(Oct 2025)
- Learn AI Basics @Dicoding
- Getting Started Programming in Python @Dicoding
- Learn Basic Structured Query Language (SQL) @Dicoding
- Excel Intermediate @Koding Studio
- Data Science Bootcamp @Kelas Wok by Kelas.com
- Microsoft Power BI @MySkill

Cotact Info :

Email : awinardi1004@gmail.coml

LinkedIn : winardi-

GitHub : awinardi1004

BACKGROUND

Stock prices move every second, and processing this fast-changing data requires a system that is both reliable and real-time. Many existing solutions still rely on batch processing, which often delays insight generation.

This project was developed to build a real-time data pipeline capable of collecting, streaming, storing, and analyzing stock market data using Kafka and AWS services.

PROBLEM STATEMENT

How can we design a system that continuously handles streaming stock data end to end—from API ingestion, Kafka streaming, storage in S3, to fast querying in Athena?

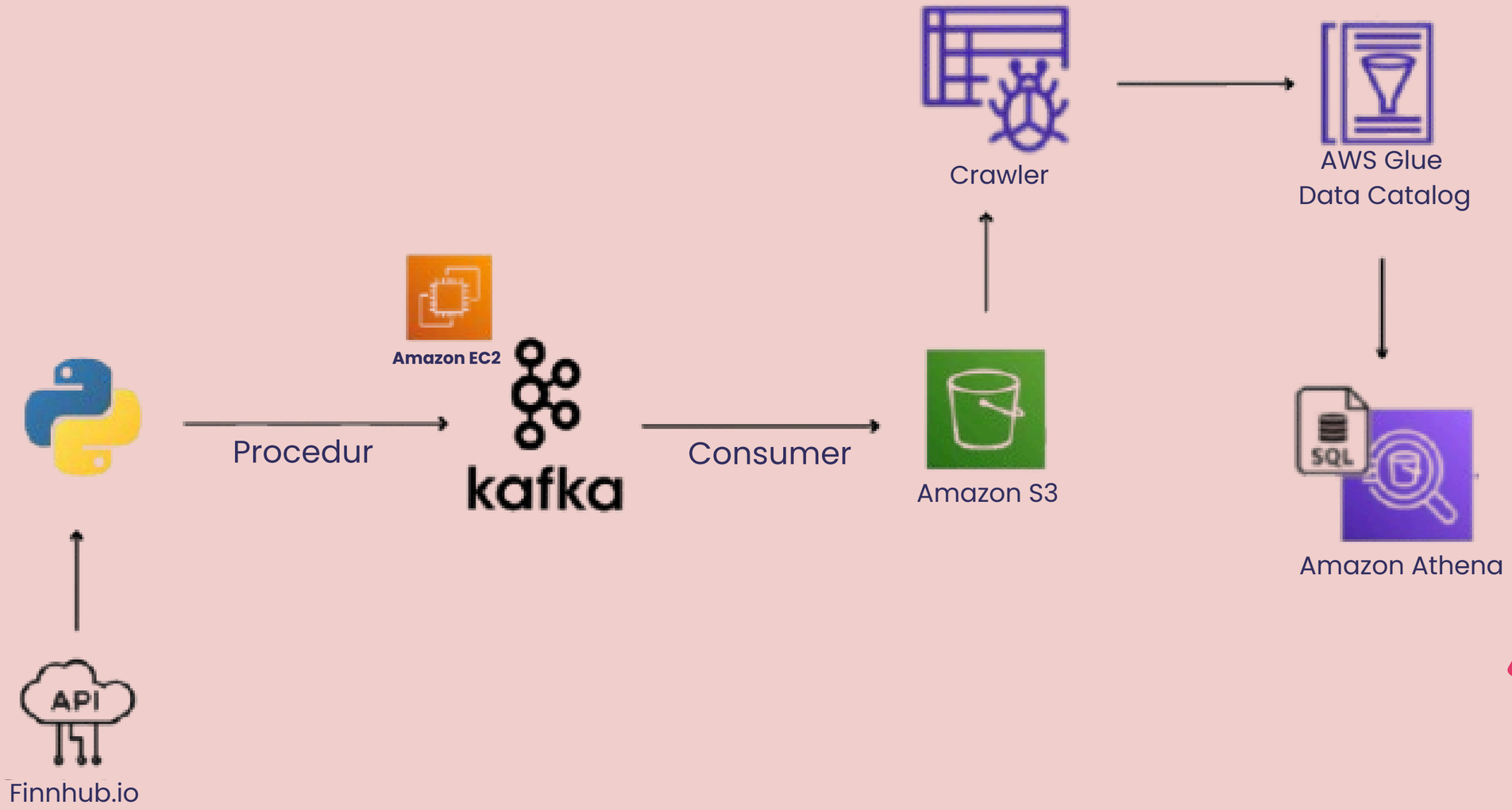
The main challenges include the speed of incoming data, dynamic data structures, and the need for efficient storage that remains easy to query.

OBJECTIVES

The objective of this project is to build a functional real-time data pipeline for U.S. stock market data. The system focuses on collecting live stock data from an API, streaming it through Kafka, and storing it in Amazon S3 in a queryable format.

This project aims to demonstrate the end-to-end process of capturing, processing, and preparing real-time data for analysis using Athena.

ARCHITECTURE



TOOLS & TECHNOLOGIES

- **Kafka :**Used as the message broker to stream live stock market data.
- **Python :** Handles data ingestion from the API and produces messages into Kafka.
- **AWS EC2 :** Hosts the Kafka broker and Python producer script.
- **AWS S3 :** Stores the streamed data in a scalable, cost-efficient data lake.
- **AWS Glue Crawler :** Automatically detects schema and prepares the data for querying.
- **Amazon Athena:** Runs SQL queries directly on S3 for analysis.

IMPLEMENTATION

REAL-TIME DATA COLLECTION

- Python script connects to Finnhub WebSocket → receives live stock ticks (AAPL, TSLA, MSFT, etc.)

MESSAGE STREAMING WITH KAFKA

- Producer sends each tick into a Kafka topic
- Consumer reads messages and processes them in real time

DATA STORAGE IN AMAZON S3

- Consumer writes validated records as files into S3 buckets
- Folder structure organized by date for easy partitioning

METADATA & SCHEMA WITH AWS GLUE

- Glue Crawler scans the S3 bucket
- Automatically creates / updates tables in AWS Glue Data Catalog

QUERYING WITH AMAZON ATHENA

- Athena reads the Parquet/CSV files using the Glue schema
- Enables SQL analysis directly on S3 data

RESULTS

- Real-time trade data successfully streamed from Finnhub into Kafka.
- Consumer batches the messages and stores them as Parquet files in S3.
- Glue Crawler updates the table schema automatically.
- Athena queries confirm the data is captured and processed correctly.

RESULTS DOCUMENTATION

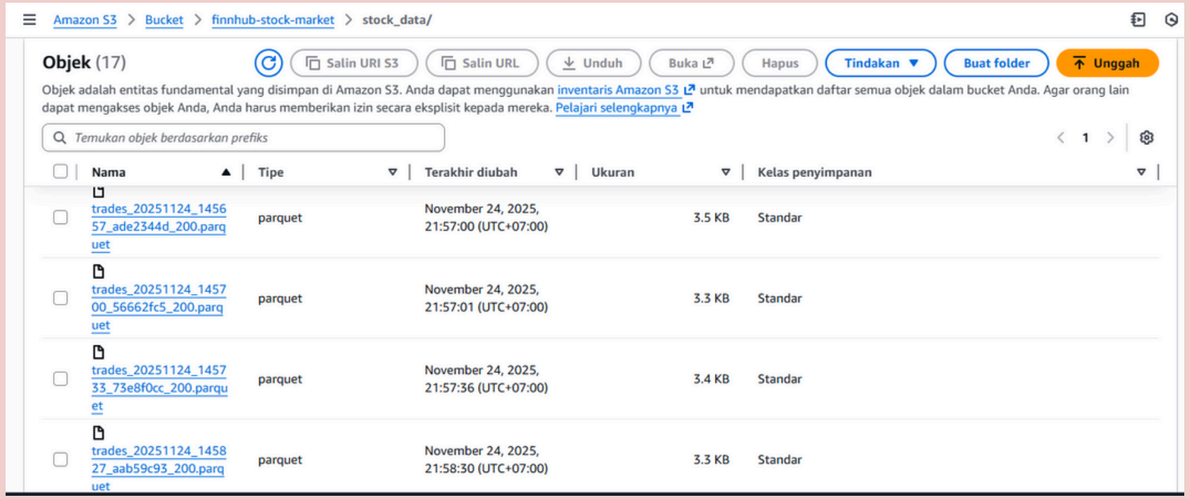
Procedure run

```
Connected to Finnhub WebSocket
Subscribed: AAPL
Subscribed: GOOGL
Subscribed: TSLA
Subscribed: MSFT
Sent to Kafka: {'c': ['1'], 'p': 415.81, 's': 'TSLA', 't': 1763997375165, 'v': 40}
Sent to Kafka: {'c': ['1', '8'], 'p': 415.89, 's': 'TSLA', 't': 1763997374633, 'v': 40}
Sent to Kafka: {'c': ['1', '8'], 'p': 415.88, 's': 'TSLA', 't': 1763997374633, 'v': 40}
```

Consumer run

```
Consumer running...
C:\Users\User\AppData\Local\Temp\ipykernel_14744\3217407254.py:23: DeprecationWarning: datetime.d
in a future version. Use timezone-aware objects to represent datetimes in UTC: datetime.datetime.
  timestamp = datetime.datetime.utcnow().strftime("%Y%m%d_%H%M%S")
Uploaded to S3: s3://finnhub-stock-market/stock_data/trades_20251124_151800_dd936e5a_200.parquet
Uploaded to S3: s3://finnhub-stock-market/stock_data/trades_20251124_151901_bb7aa701_200.parquet
```

Data stored in S3



Query successful in Athena

#	symbol	minute	avg_price	total_volume
1	MSFT	2025-11-24 14:48:00.000	470.08750000000003	1169
2	GOOGL	2025-11-24 14:48:00.000	317.2825	800
3	TSLA	2025-11-24 14:48:00.000	409.0215384615385	1164
4	TSLA	2025-11-24 14:52:00.000	409.32187500000003	1988
5	MSFT	2025-11-24 14:52:00.000	468.5201315789474	3166
6	GOOGL	2025-11-24 14:52:00.000	317.08875000000006	1798
7	AAPL	2025-11-24 14:52:00.000	273.8985714285714	1660
8	GOOGL	2025-11-24 14:53:00.000	317.18467741935484	8894
9	TSLA	2025-11-24 14:53:00.000	410.94158385093175	11566
10	MSFT	2025-11-24 14:53:00.000	469.23449438202243	5737

CONCLUSION

What began as a stream of raw market events moving through Kafka gradually transformed into a well-structured pipeline. Each component played its part: Kafka delivered the data in real time, S3 kept it safely stored, Glue brought structure to the chaos, and Athena turned it into something we could finally explore and understand. The result is a system that not only handles live market data but makes it meaningful, accessible, and ready for deeper analysis. This pipeline shows how the right tools can turn fast-moving information into clear, actionable insight.

NEXT STEPS / IMPROVEMENTS

ADD MONITORING WITH GRAFANA

Build dashboards to track Kafka throughput, consumer lag, and S3 ingestion for real-time visibility.

IMPROVE DATA QUALITY CONTROLS

Add validation and error handling to prevent corrupted or incorrect files from reaching S3.

OPTIMIZE S3 STRUCTURE WITH PARTITIONING

Organize Parquet files by symbol/date to make Athena queries faster and reduce scan costs.

AUTOMATE METADATA & ETL WITH AWS GLUE

Use crawlers and scheduled jobs to automatically manage schema, partitions, and transformations.

Feel Free to Reach Me Here !

Email : Awinardi1004@gmail.com
LinkedIn : www.linkedin.com/in/winardi-/
GitHub : https://github.com/awinardi1004