**Real Time Stock Data Streaming**

Link to github repo containing files and screenshots- [Click\_here](https://github.com/bhanuprakashnvb/Stock-Pipeline-Mongodb)

For the final project we have explored the use cases of Kafka, MongoDB and Python for data warehousing and pipeline engineering, through the lens of this project which is focused on comparing similar large-cap stocks to help users better understand the “Options” market. Options are financial derivatives that give buyers the right, but not the obligation, to buy or sell an underlying asset at an agreed-upon price and date.

Our goal is to create dashboards with data for 2-3 similarly priced stocks such that users can select the best stock to invest in. We are exploring the stocks namely – IBM , Nvidia and Apple Inc.

1. **Conceptual Schema for Database**

**A screenshot of a computer

Description automatically generated**

* **realtime\_data collection**:
  + Represents a time-series dataset.
  + Contains documents storing real-time stock data for multiple stocks.
* **Entities and Relationships**:
  + Each document is self-contained and includes:
    - **\_id**: A unique identifier for each document.
    - **Stocks (AAPL, IBM, NVDA)**: Nested subdocuments representing each stock's data.
  + No foreign key or relational schema is needed (denormalized).
* **Field Details**:
  + **\_id**: Automatically generated by MongoDB, uniquely identifying each record.
  + **Stocks**:
    - **timestamp**: The time the data was recorded.
    - **close**: The stock's closing price at that timestamp.
* **NoSQL Design Considerations**:
  + **Denormalization**: Data for all stocks is stored together in one document for quick access.
  + **Scalability**: Adding new stocks only requires appending fields to the document.
  + **Flexibility**: Supports schema changes like adding new fields (e.g., volume and open) without modifying existing documents.

1. **Database Design**

* **No Null Values-**

**To ensure data integrity, the following constraints are enforced:**

**-\_id: This field is automatically generated and cannot be null as it is a primary key.**

**- Stock Data (AAPL, IBM, NVDA):**

**- The timestamp and close fields within each stock object cannot be null.**

**- MongoDB validators ensure that these fields are mandatory when inserting a document.**

* **Uniqueness**

**- \_id Field: Uniqueness is inherently enforced by MongoDB since \_id is the primary key for each document.**

**- Stock Timestamps:**

**- For a given stock (e.g., AAPL), the timestamp field should be unique to avoid duplicate records for the same time period.**

* **Data Format**

**The following formats are strictly enforced to maintain consistency:**

**- timestamp Field:**

**- Must follow the ISO 8601 format (YYYY-MM-DD HH:mm:ss) for consistency across all stock data.**

**- Enforced using MongoDB schema validation rules ($jsonSchema).**

**- close Field:**

**- Must be a decimal or floating-point number.**

1. **Code to create database and data stream pipeline**

In this project, we create a MongoDB database and build queries to insert and retrieve stock market data using Kafka for real-time data streaming. The MongoDB database, named stock\_market, contains a collection called realtime\_data. The consumer.ipynb script listens for incoming stock data from Kafka, deserializes it, and inserts the data into MongoDB. Each record is uniquely identified by a combination of the stock symbol and timestamp to avoid duplicates. The data includes timestamps and closing prices for each stock symbol. After the data is inserted into MongoDB, we use MongoDB Atlas to create charts, such as a line graph plotting timestamps versus close prices for different stocks. All code for creating the database, inserting data, and querying the MongoDB Atlas charts can be found in the provided notebooks. The code is reusable and can be extended for additional stock symbols or data formats.

1. **Overall Contribution Summary**

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Task | Contribution | AVG Time Spent (hrs) |
| Anish Khatvakar | Built the data pipeline and deployed Kafka on AWS. | Contributed by setting up the EC2 server on AWS, deploying Kafka, and configuring the environments to ensure a successful data stream pipeline. | 12.5 |
| Bhanuprakash Narayana | Set up MongoDB for data warehousing and created charts using Atlas. | Implemented data warehousing, creating and managing the MongoDB database for storing real-time stock data, and visualizing this data with charts in MongoDB Atlas for easy analysis and reporting. | 12.5 |