Task4 Part 2: End to End System Design

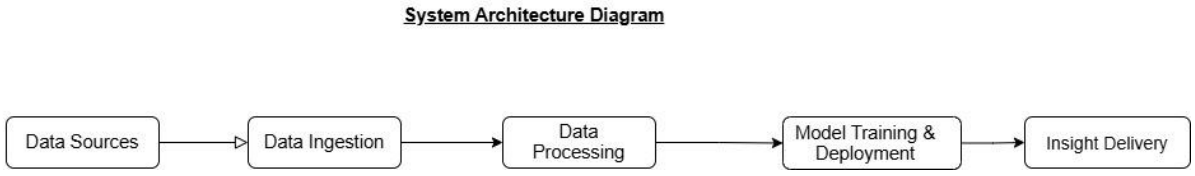
Team Name: Data Dominators

Date: 2025/03/09

System Architecture Overview

The end to end system for deploying the stock price prediction model in a production environment involves four main components: data collection, data processing, model operations, and insight delivery . Below is a simplified breakdown of each component, along with a basic system architecture diagram and justification for the design choices.

1. System Architecture Diagram



- Data Sources : Market data APIs, financial news, etc.
- Data Ingestion : A simple message queue (e.g., RabbitMQ) for real time data and a cloud storage (e.g., AWS S3) for batch data.
- Data Processing : Python scripts or a lightweight ETL tool (e.g., Pandas, Airflow) for cleaning and feature engineering.
- Model Training & Deployment : Scikit learn or TensorFlow for model training, and Flask for deploying the model as an API.
- Insight Delivery : A simple dashboard (e.g., Flask + Plotly) or API endpoints for delivering predictions.

2. Component Justification

a. Data Collection & Ingestion

Technology/Approach :

- Real time Data : RabbitMQ for streaming market data.
- Batch Data : AWS S3 for storing historical data.

Justification :

- RabbitMQ is easy to set up and manage for real time data streaming.
- AWS S3 is cost effective and widely used for storing large datasets.

Tradeoffs :

- RabbitMQ is less scalable than Kafka but simpler to use.
- AWS S3 is not suitable for real time access but works well for batch processing.

b. Data Processing

Technology/Approach :

- Python scripts or Pandas for data cleaning and feature engineering.
- Airflow for scheduling batch processing jobs.

Justification :

- Python and Pandas are easy to use and widely adopted for data processing.
- Airflow is a lightweight tool for orchestrating batch jobs.

Tradeoffs :

- Pandas is not suitable for very large datasets, but it works well for moderate sized data.
- Airflow requires some setup but is simpler than more complex tools like Spark.

c. Model Operations

Technology/Approach :

- Model Training : Scikit learn or TensorFlow for building and training the prediction model.
- Model Deployment : Flask for serving the model as an API.

Justification :

- Scikit learn is simple and effective for many machine learning tasks.
- Flask is lightweight and easy to use for deploying models as APIs.

Tradeoffs :

- Scikit learn may not be as powerful as TensorFlow for complex models.
- Flask is not as scalable as Kubernetes but is much simpler to set up.

d. Insight Delivery

Technology/Approach :

- Dashboards : Flask + Plotly for visualizing predictions.
- API Endpoints : Flask for serving real time predictions.

Justification :

- Flask + Plotly is a simple and effective way to create interactive dashboards.
- Flask is easy to use for serving predictions via API.

Tradeoffs :

- Flask + Plotly may not be as feature rich as Tableau or Power BI but is much simpler.
- Flask APIs may require additional security measures for production use.

3. Data Flow Explanation

Batch vs. Streaming :

- Batch Processing : Historical data is ingested into AWS S3 and processed in batches using Python scripts or Airflow.
- Streaming : Real time market data is streamed using RabbitMQ and processed in near real time for making immediate predictions.

Data Transformation Stages :

1. Data Ingestion : Data is collected from various sources and ingested into RabbitMQ (for real time) or S3 (for batch).
2. Data Preprocessing : Data is cleaned, normalized, and transformed using Python scripts or Pandas.
3. Feature Engineering : Additional features (e.g., moving averages) are generated.
4. Model Training : The model is trained on historical data and deployed using Flask.

5. Prediction : Real time predictions are made using the deployed model and delivered via API or dashboards.

System Interaction Points :

- Data flows from external sources (e.g., market data APIs) into the system via RabbitMQ or S3.
- Processed data is stored in a database (e.g., PostgreSQL) for easy access.
- Predictions are delivered to end users via dashboards or API endpoints.

4. Challenge Analysis

1. Challenge: Real time Data Latency

   Mitigation : Use RabbitMQ for low latency data streaming and optimize Python scripts for faster processing.

2. Challenge: Model Drift

   Mitigation : Monitor model performance and retrain the model periodically using new data.

3. Challenge: Scalability

   Mitigation : Use Flask for lightweight deployment and consider scaling horizontally if needed.

4. Challenge: Data Quality

   Mitigation : Implement basic data validation checks during ingestion and preprocessing stages.

5. Challenge: Cost Management

   Mitigation : Use cloud cost management tools (e.g., AWS Cost Explorer) to monitor and optimize resource usage.

5. Evaluation Criteria

- Completeness of Solution : The proposed system addresses all components, from data ingestion to insight delivery.
- Technical Feasibility : The design uses simple, widely adopted technologies (e.g., RabbitMQ, Flask) that are easy to implement.
- Design Justification : Each component is chosen based on its simplicity and ability to meet the system's requirements.
- Challenge Awareness : Potential challenges are identified, and basic mitigation strategies are proposed.

Conclusion

This simplified end to end system design provides a practical and easy to implement solution for deploying the stock price prediction model in a production environment. By using lightweight technologies like RabbitMQ, Flask, and Pandas, the system can handle both real time and batch processing, ensuring that predictions are delivered to end users with minimal latency. The design also addresses potential challenges, such as model drift and data quality, ensuring the system remains robust over time.