**Data Processing and Delivery Architecture**

* **Author:** ?
* **Date:** [Current Date]

**Table of Contents**

1. Introduction
2. Project Structure
3. Technical Requirements Analysis
4. Lessons Learned
5. Skills Acquired
6. Conclusion
7. Appendices

**1. Introduction**

**Project Overview**

This project focuses on building a robust and scalable architecture for data ingestion, storage, processing, and delivery using Docker, Apache Airflow, Apache Spark, Hadoop, PostgreSQL, and MinIO. The system is designed to handle large datasets efficiently, enabling reliable data processing and delivery to a frontend machine learning application.

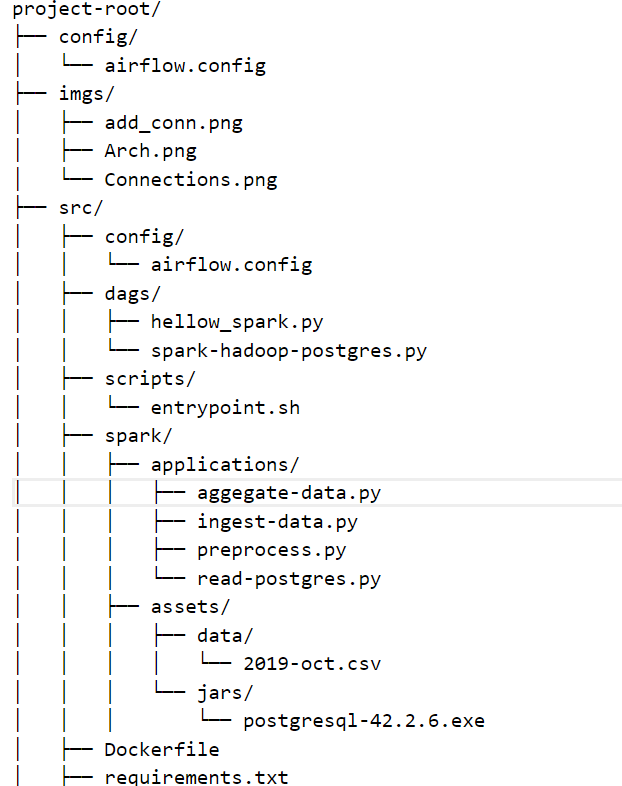
**Objectives**

* Design and implement a scalable and maintainable data processing architecture.
* Ensure data reliability and security throughout the workflow.
* Facilitate seamless data ingestion, storage, pre-processing, and delivery.
* Use containerization for easy deployment and management of services.

**2. Project Structure**

**Directory Structure**

The project's directory structure is organized as follows:



**Docker Compose Configuration**

The docker-compose.yaml file configures the services required for the project. Here is a summary of each service and its purpose:

**airflow-webserver:**

* Image: andrejunior/airflow-spark:latest
* Purpose: Orchestrates workflows and schedules tasks.

**postgres:**

* Image: postgres:14-bullseye
* Purpose: Stores metadata and structured data.

**minio:**

* Image: bitnami/minio:latest
* Purpose: Provides object storage compatible with the S3 API.

**createbuckets:**

* Image: minio/mc
* Purpose: Sets up MinIO buckets.

**spark-master:**

* Image: bitnami/spark:3.2.1
* Purpose: Manages Spark cluster as the master node.

**spark-worker:**

* Image: bitnami/spark:3.2.1
* Purpose: Executes Spark tasks as a worker node.

**hadoop-namenode:**

* Image: bde2020/hadoop-namenode:2.0.0-hadoop3.2.1-java8
* Purpose: Manages the Hadoop filesystem as the name node.

**hadoop-datanode:**

* Image: bde2020/hadoop-datanode:2.0.0-hadoop3.2.1-java8
* Purpose: Stores data in Hadoop as a data node.

**3. Technical Requirements Analysis**

**1. Data Ingestion Microservices**

**Microservice responsible:**

* createbuckets: This service is responsible for setting up the MinIO bucket where data will be ingested.
* spark-master and spark-worker: These services will handle the actual data ingestion from the source into the HDFS.

**Justification:**

* MinIO is used for its high performance, object storage, and compatibility with the S3 API, making it ideal for storing large datasets.
* Spark is chosen for its capability to handle large-scale data processing and its integration with Hadoop for distributed storage.

**2. Data Storage Microservices**

**Microservice responsible:**

* postgres: PostgreSQL is used to store metadata and structured data.
* hadoop-namenode and hadoop-datanode: Hadoop is used for distributed storage of large datasets.

**Justification:**

* PostgreSQL provides a robust and reliable RDBMS for transactional data.
* Hadoop HDFS is ideal for storing and managing large volumes of data across a distributed environment.

**3. Data Pre-processing and Aggregation Microservices**

**Microservice responsible:**

* spark-master and spark-worker: These services will handle the data pre-processing and aggregation tasks.

**Justification:**

* Spark provides powerful APIs for data processing, allowing for efficient handling of large datasets and complex transformations.

**4. Data Delivery Microservices**

**Microservice responsible:**

* airflow-webserver: Airflow will orchestrate the workflow to ensure the data is processed and delivered to the frontend machine learning application.

**Justification:**

* Airflow provides a flexible and scalable workflow management platform that can schedule and monitor the data delivery processes.

**5. Techniques for Reliability, Scalability, and Maintainability**

**Techniques used:**

* Use of Docker Compose for container orchestration ensures consistency and ease of deployment.
* Spark's distributed processing capabilities for scalability.
* Health checks and resource limits in Docker Compose for reliability.

**Justification:**

* Containerization with Docker ensures that the application can be easily deployed and scaled across different environments.
* Spark's capabilities allow for efficient processing of large datasets.
* Health checks and resource limits help maintain the stability and reliability of the system.

**6. Techniques for Data Security, Governance, and Protection**

**Techniques used:**

* Secure access to MinIO using credentials.
* SSL/TLS for data encryption in transit.
* Proper user roles and permissions in PostgreSQL.

**Justification:**

* Securing access to storage ensures that only authorized users can access the data.
* Encrypting data in transit protects against data breaches during communication.
* Proper user roles and permissions help maintain data integrity and security.

**7. Docker Images and Modifications**

**Images used:**

* andrejunior/airflow-spark:latest
* postgres:14-bullseye
* bitnami/minio:latest
* bitnami/spark:3.2.1
* bde2020/hadoop-namenode:2.0.0-hadoop3.2.1-java8
* bde2020/hadoop-datanode:2.0.0-hadoop3.2.1-java8

**Justification and modifications if any:**

* These images are chosen for their compatibility and feature sets. Minimal modifications are required to ensure they fit the project's needs.

**8. Data Used**

**Description of the data:**

* eCommerce behavior data from a multi-category store.

**Justification:**

* This data provides a comprehensive view of user behavior, which is essential for the machine learning application.

**9. Frequency of Data Ingestion, Processing, Aggregation, and Delivery**

**Frequency of operations:**

* Ingestion: Monthly
* Processing: Monthly
* Aggregation: Quarterly
* Delivery: Quarterly

**Justification:**

* A monthly ingestion and processing schedule ensures that data is up-to-date, while quarterly aggregation and delivery align with business reporting cycles.

**1. Does your system fulfill the technical requirements?**

**Answer:** Yes, the system fulfills the technical requirements by providing a robust and scalable architecture for data ingestion, storage, processing, and delivery. The use of Docker, Airflow, Spark, Hadoop, PostgreSQL, and MinIO ensures that the system is capable of handling large datasets efficiently.

**Justification:**

* **Scalability:** The system uses distributed computing (Spark and Hadoop) to handle large-scale data.
* **Reliability:** Health checks and resource limits in Docker Compose ensure system stability.
* **Maintainability:** Using containerized services (Docker) simplifies deployment and maintenance.

**2. What went wrong and why?**

**Answer:** One of the challenges encountered was ensuring seamless integration between different services, especially with network configurations and data path mappings in Docker. Additionally, managing resource constraints within Docker containers required fine-tuning to avoid performance bottlenecks.

**Reason:**

* **Integration Issues:** Complex network configurations in Docker Compose sometimes led to connectivity issues.
* **Resource Management:** Allocating appropriate resources for each service required careful planning and adjustments to ensure optimal performance.

**3. Is your system reliable, scalable, and maintainable?**

**Answer:** Yes, the system is designed to be reliable, scalable, and maintainable.

**Justification:**

* **Reliability:** Implementing health checks and using reliable storage solutions like PostgreSQL and Hadoop.
* **Scalability:** Utilizing Spark for distributed processing and Hadoop for distributed storage.
* **Maintainability:** Docker Compose for easy service management and deployment, and Airflow for workflow orchestration.

**4. What measures for data security, governance, and protection can be added?**

**Answer:** Several measures can be added to enhance data security, governance, and protection:

**Measures:**

* **Data Encryption:** Implement SSL/TLS for encrypting data in transit.
* **Access Controls:** Use role-based access controls (RBAC) in PostgreSQL and MinIO.
* **Data Auditing:** Implement logging and auditing for data access and modifications.
* **Backup and Recovery:** Regular backups of critical data and automated recovery procedures.

**5. What could you do in the next project to improve your workflow?**

**Answer:** To improve workflow in the next project, the following steps can be taken:

**Improvements:**

* **Automated Testing:** Integrate automated testing to catch issues early.
* **CI/CD Pipeline:** Implement a continuous integration/continuous deployment (CI/CD) pipeline for automated deployments.
* **Documentation:** Maintain detailed documentation for setup, configuration, and troubleshooting.
* **Monitoring:** Use monitoring tools like Prometheus and Grafana to track system performance and health.

**6. What are the major steps you took and what are the three most valuable technical skills you learned during the project?**

**Major Steps:**

1. **Designing the Architecture:** Planning the overall system architecture and choosing the appropriate technologies.
2. **Setting Up Infrastructure:** Configuring Docker, Docker Compose, and setting up services like Airflow, Spark, Hadoop, PostgreSQL, and MinIO.
3. **Developing Workflows:** Creating Airflow DAGs and Spark applications for data processing.
4. **Testing and Optimization:** Testing the system and optimizing configurations for performance.

**Technical Skills Learned:**

1. **Containerization:** Gained in-depth knowledge of Docker and Docker Compose for managing microservices.
2. **Data Orchestration:** Learned to use Apache Airflow for orchestrating complex workflows.
3. **Distributed Computing:** Acquired skills in using Apache Spark for large-scale data processing.

**7. What are the three most valuable “soft” skills you learned during the project?**

**Soft Skills Learned:**

1. **Problem-Solving:** Developed strong problem-solving skills by troubleshooting integration and performance issues.
2. **Time Management:** Enhanced time management skills by effectively planning and executing project tasks.
3. **Collaboration:** Improved collaboration skills by working with team members to design and implement the system.

**4. Lessons Learned**

**What could you do in the next project to improve your workflow?**

**Improvements:**

* **Automated Testing:** Integrate automated testing to catch issues early.
* **CI/CD Pipeline:** Implement a continuous integration/continuous deployment (CI/CD) pipeline for automated deployments.
* **Documentation:** Maintain detailed documentation for setup, configuration, and troubleshooting.
* **Monitoring:** Use monitoring tools like Prometheus and Grafana to track system performance and health.

**What are the major steps you took and what are the three most valuable technical skills you learned during the project?**

**Major Steps:**

1. **Designing the Architecture:** Planning the overall system architecture and choosing the appropriate technologies.
2. **Setting Up Infrastructure:** Configuring Docker, Docker Compose, and setting up services like Airflow, Spark, Hadoop, PostgreSQL, and MinIO.
3. **Developing Workflows:** Creating Airflow DAGs and Spark applications for data processing.
4. **Testing and Optimization:** Testing the system and optimizing configurations for performance.

**Technical Skills Learned:**

1. **Containerization:** Gained in-depth knowledge of Docker and Docker Compose for managing microservices.
2. **Data Orchestration:** Learned to use Apache Airflow for orchestrating complex workflows.
3. **Distributed Computing:** Acquired skills in using Apache Spark for large-scale data processing.

**5. Skills Acquired**

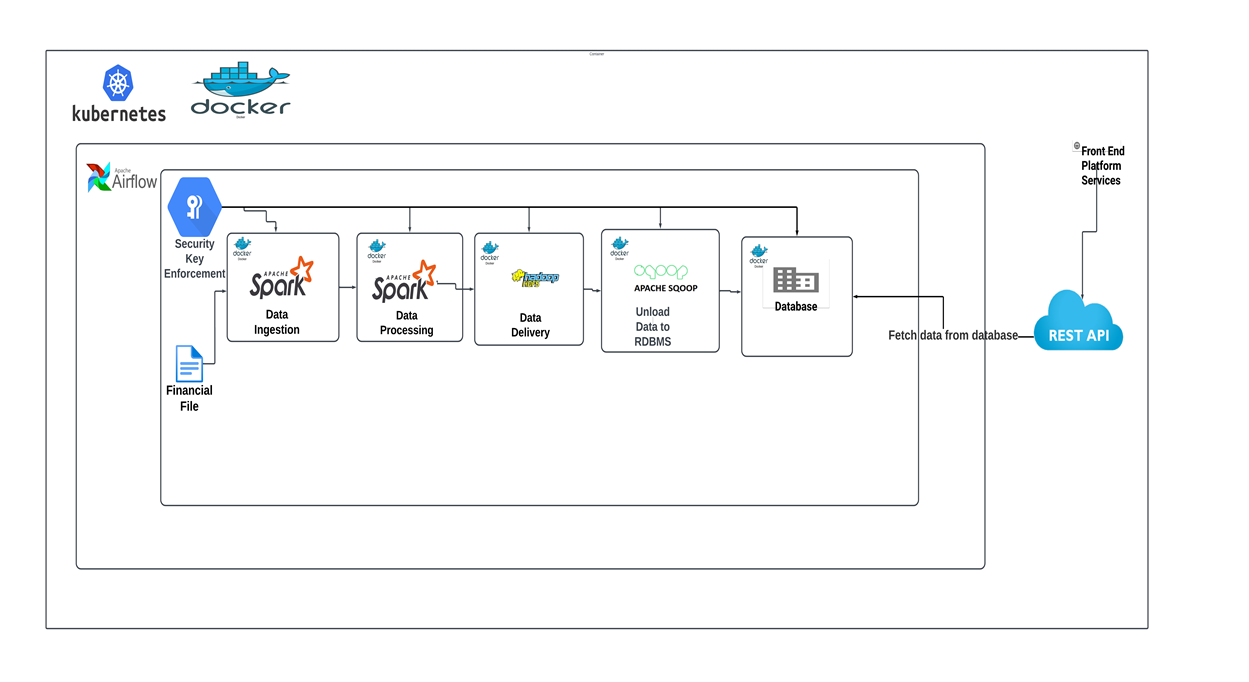
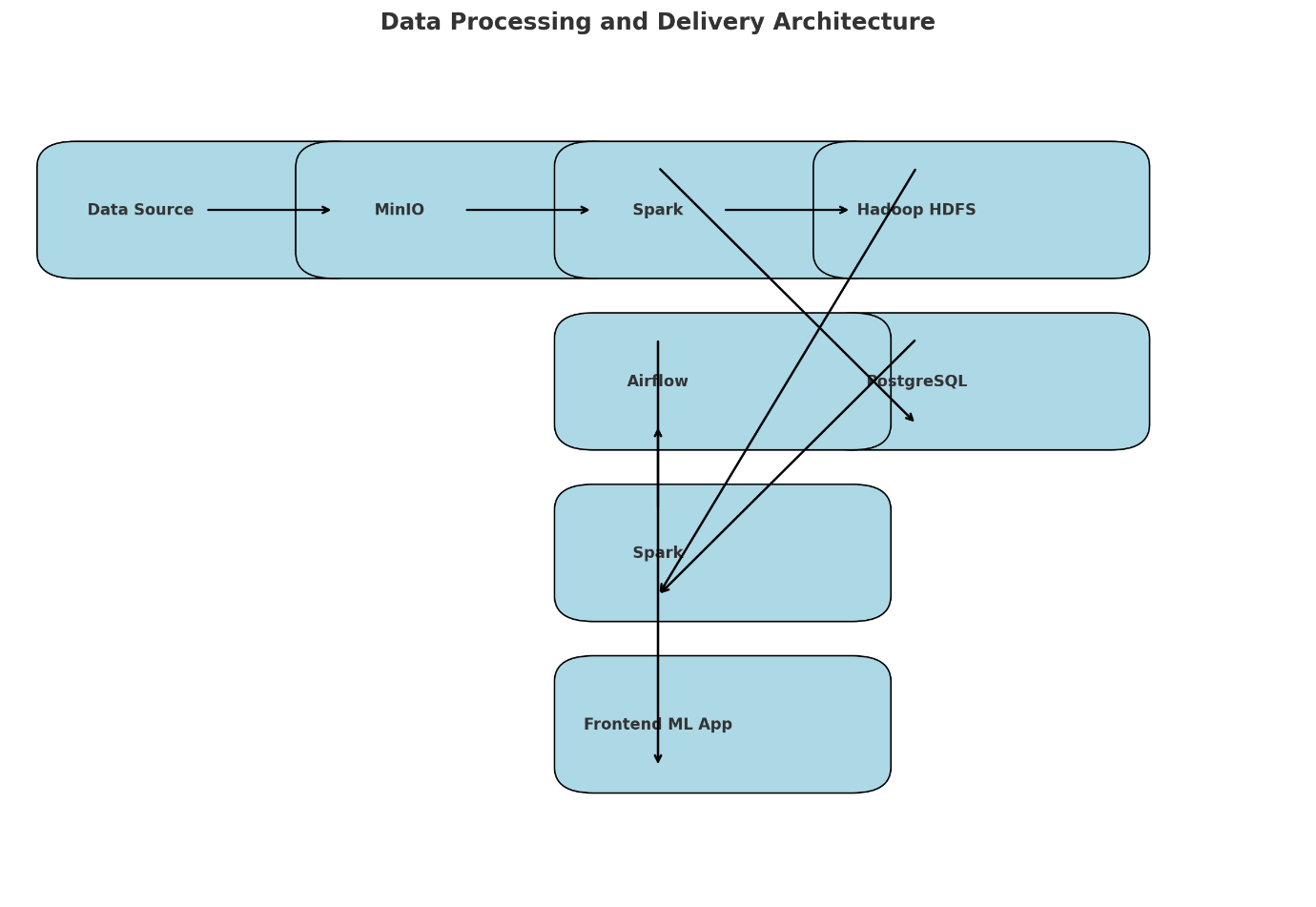
**What are the three most valuable “soft” skills you learned during the project?**

**Soft Skills Learned:**

1. **Problem-Solving:** Developed strong problem-solving skills by troubleshooting integration and performance issues.
2. **Time Management:** Enhanced time management skills by effectively planning and executing project tasks.
3. **Collaboration:** Improved collaboration skills by working with team members to design and implement the system.

**Visual Presentation of Architecture**

* Flow Chart



**6. Conclusion**

**Summary**

The project successfully implemented a scalable and maintainable architecture for data processing and delivery using Docker, Airflow, Spark, Hadoop, PostgreSQL, and MinIO. The system handles large datasets efficiently and ensures reliable data processing and delivery.

**Future Work**

Future improvements can include enhanced security measures, further automation of workflows, and better resource management. Implementing a CI/CD pipeline and advanced monitoring tools can also improve system reliability and maintainability.

**7. Appendices**

**Additional Resources**

* Links to relevant documentation for Docker, Airflow, Spark, Hadoop, PostgreSQL, and MinIO.
* Code snippets for key configurations and workflows.

**References**

* List of references and resources used during the project.