# Final Phase of Batch-Processing Data Architecture for Sentiment Analysis ML Application

The **final phase** of the **Batch-Processing-Based Data Architecture for Sentiment Analysis ML Application** focuses on finalizing the system to ensure that the data pipeline is fully operational, reliable, and scalable. This phase includes testing, optimization, deployment, and documentation, ensuring that the system is ready for production use. The goal is to ensure that all components work cohesively, the system meets the desired performance, and all documentation is complete for reproducibility.

## 1. Testing and Debugging

In this phase, thorough testing was conducted to verify that all components of the data pipeline function correctly. This involved:

- **Unit Testing**: Individual microservices (data ingestion and preprocessing) were tested in isolation to ensure they work as expected. For example, the data ingestion service was tested for handling edge cases, such as missing values or incorrect formats in the raw dataset.
- **Integration Testing**: The entire data pipeline, including data ingestion, preprocessing, and storage in BigQuery and MongoDB, was tested together. This ensured that data flowed seamlessly from one service to the next and that there were no issues with the data transfer or transformation.
- **Error Handling**: Mechanisms were put in place to handle failures during the data processing pipeline. This included setting up logging and retry strategies using **Apache Airflow** to ensure that any errors could be quickly identified and addressed.

## 2. Optimization

During the final phase, efforts were made to optimize the performance and scalability of the system. Key optimization tasks included:

- **PySpark Optimizations**: PySpark scripts were optimized for performance by ensuring efficient data processing, especially for large datasets. Techniques like caching intermediate data, filtering before transformations, and partitioning data were employed to minimize memory usage and increase processing speed.
- **Airflow Scheduling**: The scheduling of tasks in **Apache Airflow** was optimized to ensure that jobs run at optimal times, preventing conflicts and minimizing resource usage. This involved setting the appropriate dependencies between tasks and optimizing the frequency of the batch jobs.
- **Cloud Resource Configuration**: Cloud resources in **Google BigQuery** and **MongoDB Cloud** were configured for optimal performance.

**3. Deployment**
The system was deployed in the cloud using containerized microservices with Docker. Key steps included:

- Docker Compose Setup: A docker-compose.yml file was created to deploy all microservices, including data ingestion, preprocessing, and Apache Airflow for orchestration.
- Terraform Automation: Terraform scripts automated the provisioning of cloud resources (BigQuery and MongoDB), enabling quick and reproducible setups.
- CI/CD Pipeline: A CI/CD pipeline was implemented to automate testing, deployment, and monitoring, ensuring consistency and minimizing errors.

**4. Final Documentation**
The project was thoroughly documented, including:

- System Architecture: Detailed explanation of how the system components (BigQuery, MongoDB, PySpark, Airflow, Docker) work together for sentiment analysis.
- Setup Instructions: Step-by-step guide for setting up the system using Docker and Terraform to ensure reproducibility.
- Code Comments and Documentation: The code was well-commented for easy maintenance, and the repository includes guidelines for extending the system.

**5. Personal Reflection**

The final phase of the project provided valuable learning experiences related to the deployment, optimization, and documentation of a real-world data pipeline. One of the key takeaways from this phase was the importance of testing and debugging in ensuring the reliability of a data processing system. The integration of cloud services like BigQuery and MongoDB, along with orchestration tools like Apache Airflow, provided a deep understanding of how to build scalable and reliable data pipelines.

Additionally, containerizing the entire system with **Docker** and automating the deployment with **Terraform** made the system easily portable and reproducible. This is crucial for ensuring that the system can be scaled and maintained in production environments.

**6. GitHub Repository**

The final code and all related documentation are available in the GitHub repository. This repository contains the entire implementation, including the Python scripts for data ingestion and preprocessing, the Docker configuration, the Terraform scripts for infrastructure setup, and the Airflow DAGs for task orchestration.

GitHub Repository:
Bigquery-PySpark-MongoDB