E-Commerce Data Pipeline: Pipeline on GCP

Dhruvil Joshi

INFO-I535 Management, Access, and Use of Big and Complex Data

GitHub Link for this project - https://github.com/DAJ8112/Ecommerce-Pipeline

1) Introduction

Organizations today generate massive volumes of transactional data that must be processed, validated, and transformed into actionable insights. Raw e-commerce data - including customer orders, product information, and fulfillment records requires systematic processing pipelines that can handle scale, ensure data quality, and produce reliable analytics. I've focused on the **Lifecycles and Pipelines** module, addressing the challenge of building an end-to-end data pipeline that transforms raw e-commerce transactions which then can be converted into dashboards, or data for predictive models.

I've designed and implemented a complete Big Data processing pipeline on Google Cloud Platform using PySpark and Dataproc. The pipeline extracts data from BigQuery's **theLook eCommerce dataset,** validates and cleans records using a quarantine pattern, transforms data into simple business metrics loads results into BigQuery, and then makes it ready for consumption. This can be a simple dashboard - all orchestrated through automated workflow templates and ran using cloud scheduler at fixed intervals as required at the end.

2) Background

Data processing matters critically for big and complex data because organizations cannot make informed decisions from raw, unvalidated data scattered across systems. E-commerce companies process millions of transactions daily, and without proper data pipelines, issues like missing customer IDs, negative prices, or corrupted records propagate through analytics, leading to flawed business decisions. Furthermore, as data volumes grow beyond what single machines can handle, distributed processing becomes essential. A well-designed pipeline ensures data quality, enables scalability, and transforms raw information into trustworthy insights that drive revenue optimization, operational efficiency, and customer satisfaction.

This project applies three core concepts from the course.

1. **Distributed Data Processing** through PySpark on Dataproc, where computations are parallelized across multiple executors to handle large datasets efficiently using batch processing patterns.
2. **Data Lake Storage** by generating aggregated metrics and visualizations that summarize historical patterns - such as average fulfillment times by distribution center, revenue trends by geography, and profit margins by product category.

3. **Reproducibility** by structuring the pipeline with clear documentation, version-controlled code, automated orchestration, and a comprehensive README that enables others to replicate the entire system from scratch.

3) Methodology

Platform: Google Cloud Platform (GCP)

Key Services:

- BigQuery - Source data warehouse and curated output storage
- Cloud Storage (GCS) - Data lake for intermediate Parquet files
- Dataproc Serverless - Managed PySpark execution without cluster management
- Dataproc Workflow Templates - Pipeline orchestration with job dependencies
- Looker Studio - Interactive dashboard visualization
- Cloud Scheduler – Run Pipeline automatically on a schedule

3.1 Design Steps

Step 1: Infrastructure Setup

- Created GCP project with billing and budget alerts
- Enabled required APIs (BigQuery, Dataproc, Cloud Storage, Resource Manager)
- Configured VPC network and firewall rules for Dataproc communication
- Created Cloud Storage bucket with data lake folder structure
- Created BigQuery datasets for curated outputs and metadata

Step 2: Data Lake Architecture Implemented this architecture:

- Raw Layer (/raw/) - Direct extraction from source, unmodified
- Cleaned Layer (/cleaned/) - Validated and quality-checked records
- Curated Layer (/curated/) - Business-ready aggregated metrics
- Quarantine Layer (/quarantine/) - Failed validation records with error tags
- Metadata Layer (/metadata/) - Quality reports and job logs

Step 3: PySpark Job Development Created four sequential jobs:

- Extract Job - Reads from BigQuery using Spark BigQuery connector, writes Parquet to raw layer
- Clean Job - Applies validation rules, separates good/bad records, generates quality report
- Transform Job - Joins tables, calculates business metrics using aggregations and window functions
- Load Job - Writes curated Parquet files back to BigQuery tables

Step 4: Orchestration Configuration

- Created Dataproc Workflow Template
- Defined job dependencies (extract → clean → transform → load)
- Configured managed cluster settings for automatic provisioning
- Enabled single-command pipeline execution

Step 5: Visualization Development

- Connected Looker Studio to BigQuery curated dataset
- Built interactive charts for geographic sales, fulfillment performance, revenue by category, and monthly trends

3.3 Architecture Diagram

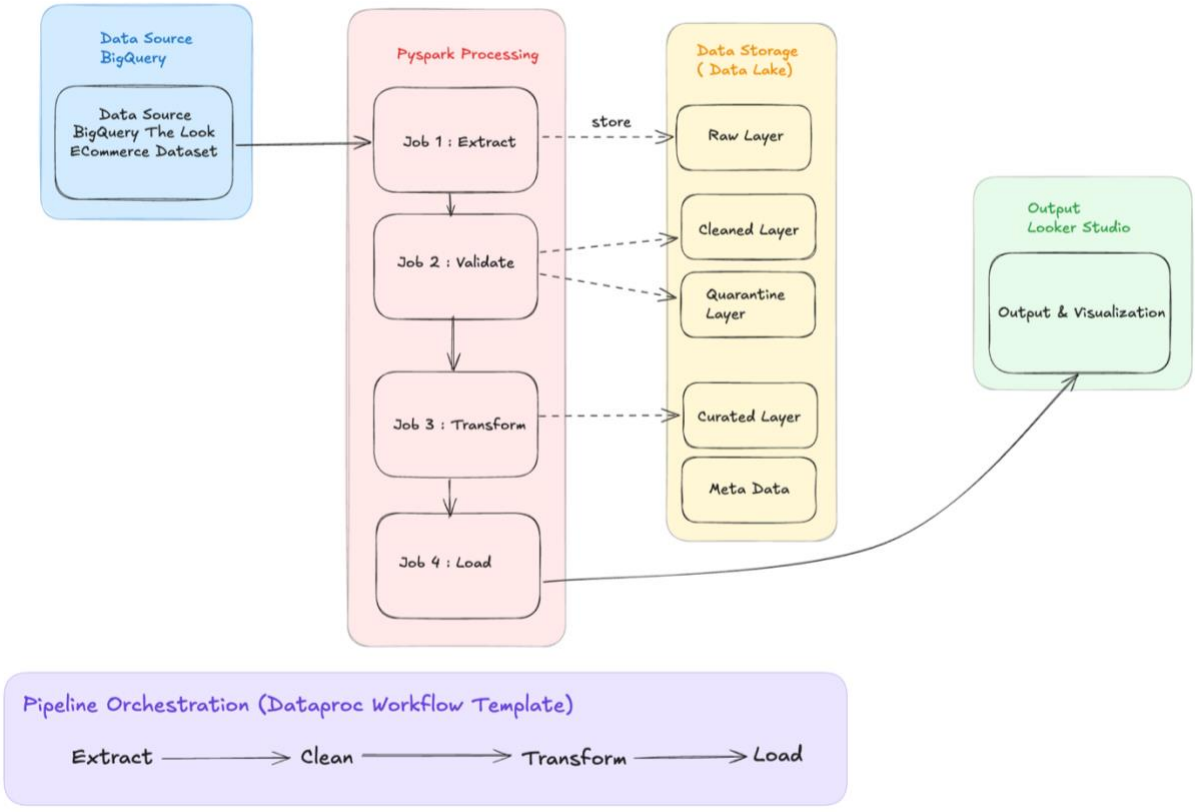

Figure 1: Brief Overview of Project Architecture

3.4 Data Description

Source: BigQuery Public Dataset - theLook eCommerce

Below is the subset of tables which have been processed for this pipeline

Table	Records	Description
orders	125,161	Order transactions with status and timestamps
order_items	181,667	Individual line items with prices and fulfillment dates
users	100,000	Customer demographics and geographic information
products	29,120	Product catalog with costs and categories
distribution_centers	10	Warehouse locations

Total Volume: ~436,000 records across 5 tables

Date Range: January 2019 to November 2025

Data Types: Integers (IDs, counts), Floats (prices, costs), Timestamps (order/ship/delivery dates), Strings (names, categories, locations)

3.5 Validation Plan

The pipeline includes a set of data quality checks across all core tables. For Orders, validations ensure num_of_item is greater than zero and key fields such as order_id, user_id, and created_at are not null. Order Items are checked for nulls on identifiers and verified that sale_price is present and non-negative. Users undergo null checks on id, email, and created_at, with an additional rule requiring age to fall between 0 and 120. Products are validated for non-null id, and both cost and retail_price must be present and ≥ 0.

Validation metrics tracked include total records processed, number of clean versus quarantined records, pass rates, and detailed failure reasons. The pipeline is considered successful when all jobs complete without errors, each layer (raw → cleaned → curated) receives data, BigQuery tables reflect expected row counts, and a quality report summarizing validation results is generated.

4) Results

End-to-End Pipeline Execution

What: Successfully executed the complete data pipeline through orchestrated workflow, processing data from source extraction through to final visualization-ready tables.

How: Instantiated Dataproc Workflow Template that automatically managed cluster provisioning, job sequencing, dependencies, and resource cleanup.

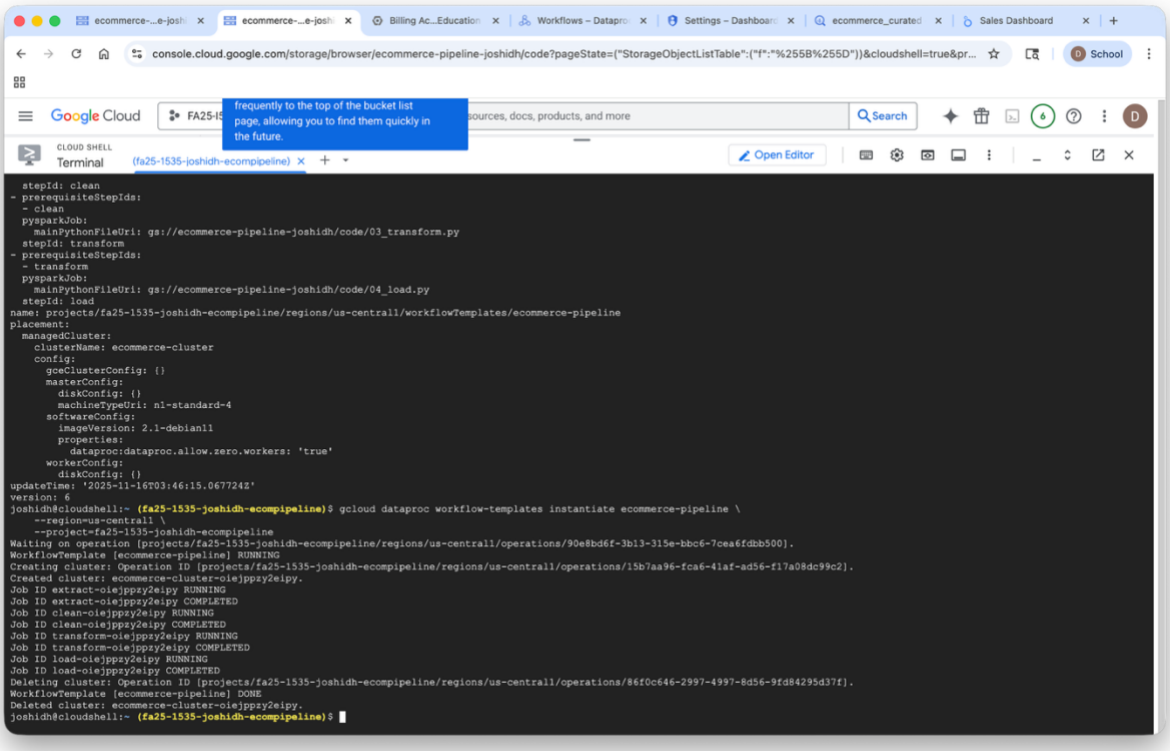

Figure 2: Output of code after running the orchestrated pipeline using dataproc Workflow Template

Pipeline Execution Summary:

Cluster Creation: Automatically provisioned n1-standard-4 instance

Job 1 (Extract): COMPLETED - 436,958 records extracted across 5 tables

Job 2 (Clean): COMPLETED - 100% validation pass rate, quality report generated

Job 3 (Transform): COMPLETED - 9 aggregated metric tables created

Job 4 (Load): COMPLETED - All tables loaded to BigQuery

Cluster Deletion: Automatically terminated after completion

Interpretation**:** The workflow template successfully demonstrated production-grade pipeline orchestration. A single command triggered the entire data processing sequence with proper dependency management - each job waited for its predecessor to complete before starting. The automatic cluster lifecycle management (provisioning at start, deletion at end) ensures cost optimization by eliminating idle resources. This proves the pipeline is reproducible and can be scheduled for regular automated execution.

Data Lake Architecture Implementation:

What: Established proper data lake layering with Bronze (raw), Silver (cleaned), and Gold (curated) tiers following industry-standard medallion architecture.

How: PySpark jobs wrote intermediate results to designated Cloud Storage paths in Parquet format.

Evidence:

```
joshidh@cloudshell:~ (fa25-1535-joshidh-ecompipeline)$ gsutil ls gs://ecommerce-pipeline-joshidh/
gs://ecommerce-pipeline-joshidh/cleaned/
gs://ecommerce-pipeline-joshidh/curated/
gs://ecommerce-pipeline-joshidh/metadata/
gs://ecommerce-pipeline-joshidh/quarantine/
gs://ecommerce-pipeline-joshidh/raw/
joshidh@cloudshell:~ (fa25-1535-joshidh-ecompipeline)$
```

Figure 3: Storage structure on data lake

```
joshidh@cloudshell:~ (fa25-1535-joshidh-ecompipeline)$ gsutil ls gs://ecommerce-pipeline-joshidh/raw/
gs://ecommerce-pipeline-joshidh/raw/.keep
gs://ecommerce-pipeline-joshidh/raw/distribution_centers/
gs://ecommerce-pipeline-joshidh/raw/order_items/
gs://ecommerce-pipeline-joshidh/raw/orders/
gs://ecommerce-pipeline-joshidh/raw/products/
gs://ecommerce-pipeline-joshidh/raw/users/
joshidh@cloudshell:~ (fa25-1535-joshidh-ecompipeline)$ gsutil ls gs://ecommerce-pipeline-joshidh/raw/orders/
gs://ecommerce-pipeline-joshidh/raw/orders/
gs://ecommerce-pipeline-joshidh/raw/orders/_SUCCESS
gs://ecommerce-pipeline-joshidh/raw/orders/part-00000-5c64cee2-80a9-463f-98e6-bf9b9d147d12-c000.snappy.parquet
joshidh@cloudshell:~ (fa25-1535-joshidh-ecompipeline)$
```

Figure 4: Contents in Parquet Structure inside a table

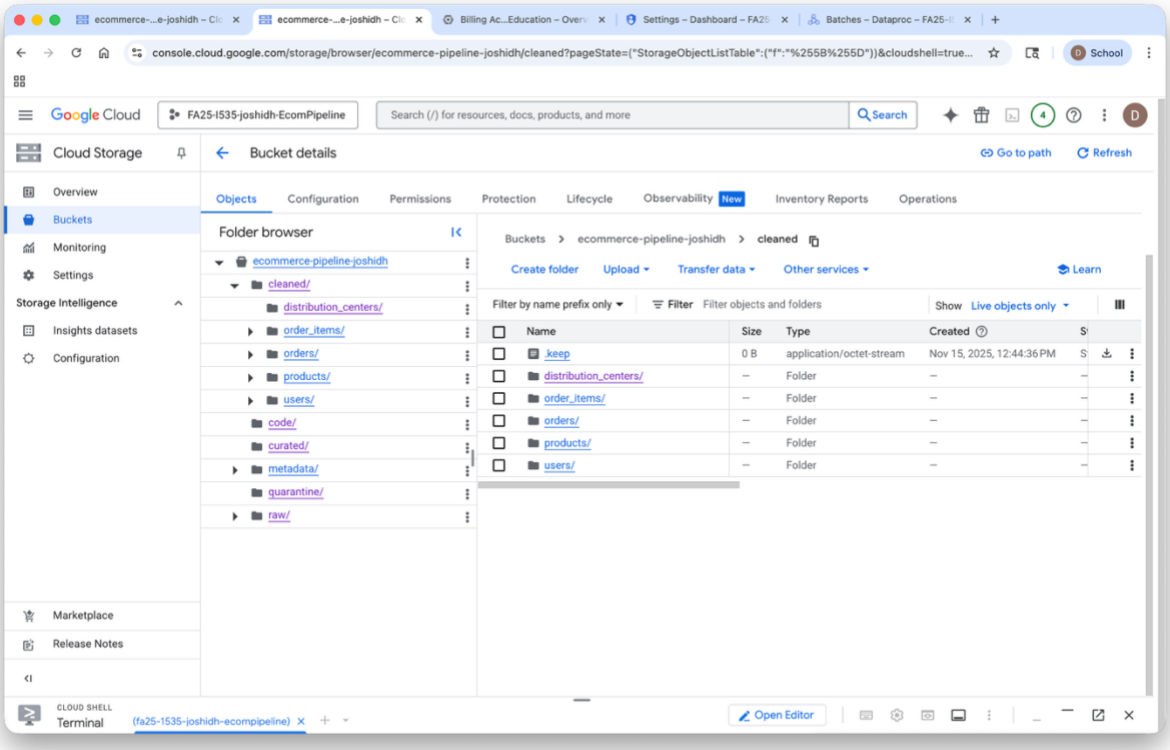

Figure 5: Contents in the Google Cloud GUI

Visualization Integration:

While the focus of this project was not on analytics but from the course module, I wanted to demonstrate something for the “consume” part in the pipeline i.e use the transformed data in some way. For this I built a very simple dashboard to demonstrate something like this could be done.

What: Connected BigQuery curated tables to Looker Studio, creating interactive dashboard for pipeline outputs.

How: Established BigQuery data source connection in Looker Studio and built charts from loaded tables.

Evidence:

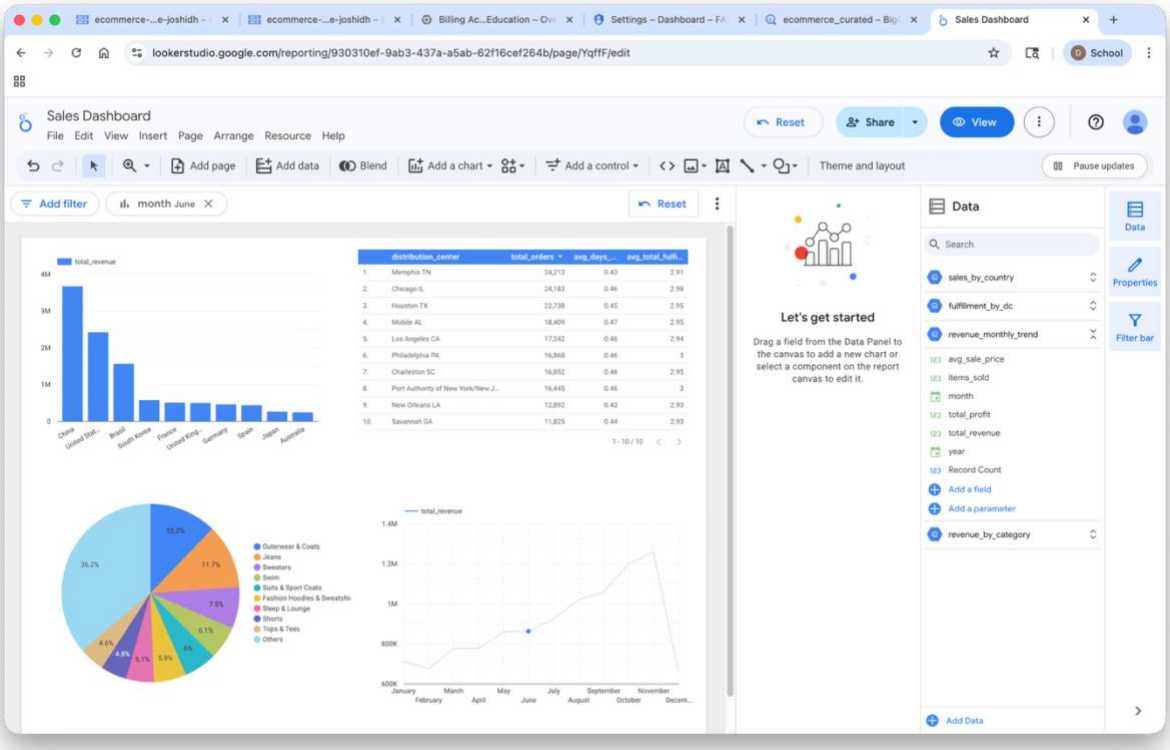

Figure 6: Simple Looker Studio Dashboard

Interpretation: The visualization layer confirms end-to-end pipeline completion - data successfully flows from source (BigQuery theLook) through processing (Dataproc/GCS) back to analytical warehouse (BigQuery curated) and finally to business intelligence tools (Looker Studio). The dashboard auto-refreshes when underlying tables update, meaning subsequent pipeline runs automatically reflect in visualizations without manual intervention.

5) Discussion

Course Concepts in Practice:

The course emphasis on end-to-end pipelines shaped how I approached this project. Rather than building isolated scripts, I designed a complete system where each stage feeds cleanly into the next: extraction produces Parquet files that cleaning jobs consume, cleaned data flows into transformation logic, and curated metrics land in a queryable warehouse. This end-to-end perspective prevented brittle point solutions and yielded a system where each component has clear inputs, outputs, and responsibilities.

Batch processing emerged as both a strength and limitation. The course material on batch versus streaming helped me understand that batch jobs trade latency for efficiency - processing large volumes at scheduled intervals rather than reacting to events in real-time. For historical analytics like revenue trends, batch processing makes perfect sense. However, I became aware of its blind spots: if a fulfillment center experiences sudden delays, the pipeline won't surface that issue until the next scheduled run. These limitations aren't design failures; they're inherent trade-offs of the batch processing paradigm.

Surprises:

Finding clean source data was a surprise; 100% of records passed validation, rendering my elaborate quarantine framework seemingly unnecessary. Yet this demonstrated that validation frameworks are insurance policies - essential when things go wrong, even if unused on clean data.

Troubles:

JAR version compatibility presented an unexpected barrier. Explicitly specifying BigQuery connector JARs via --jars triggered cryptic errors, teaching me to use declarative dependency resolution (spark.jars.packages) instead. Schema discovery failures also taught valuable lessons - I assumed order_items contained product_distribution_center_id for direct joins but had to redesign through the products table. These experiences reinforced the importance of validating assumptions against actual schemas before writing transformation logic.

What Can I Improve Next:

The most impactful enhancement would be evolving this batch pipeline into a streaming architecture with full automation. Currently, weekly batch processing works for historical analysis but creates significant latency for operational insights. Implementing streaming with Pub/Sub and Dataflow would enable near-real-time analytics - fulfillment teams could respond to delays immediately, marketing could capitalize on traffic surges as they happen, and revenue dashboards would track performance minute-by-minute. Coupling this with Cloud Scheduler running daily or hourly (rather than weekly) would make the pipeline production-ready, processing only incremental changes rather than full refreshes to manage costs efficiently.

The course emphasis on reproducibility influenced a lot as well. Configuration lives in clearly marked variables, the data lake follows the medallion architecture pattern, workflow templates make dependencies explicit, and comprehensive documentation enables others to recreate the system. This matters because pipelines outlive their creators - without clear patterns and

documentation, systems become unmaintainable technical debt. Reproducibility also enables troubleshooting: when jobs fail, I can trace data flow through each layer and identify exactly where things broke. This transparency distinguishes professional data engineering from quick-and-dirty scripts.

6) Conclusion

The primary takeaway from this project is that building production-grade Big Data pipelines requires equal attention to infrastructure, data quality, and automation - not just analytical logic. While PySpark transformations are intellectually interesting, the majority of effort went into GCP configuration, error handling, and ensuring reproducibility. The quarantine pattern for data quality, the medallion architecture for data organization, and workflow orchestration for automation are patterns that translate directly to enterprise data engineering roles. These structural elements transform ad-hoc analysis into reliable, maintainable systems that organizations can trust for decision-making.

Given more time, the most impactful improvement would be implementing **streaming data ingestion** using Google Cloud Pub/Sub and Dataflow. This would evolve the pipeline from batch processing (periodic snapshots) to real-time analytics, enabling dashboards that update within seconds of new transactions. Real-time fulfillment monitoring could alert operations teams to emerging delays before customers notice, and live revenue tracking could inform immediate marketing decisions. This enhancement would demonstrate additional Big Data concepts (stream processing, event-driven architecture) while significantly increasing the pipeline's business value.

7) References

1. Google Cloud. (2024). Dataproc Serverless Documentation. https://cloud.google.com/dataproc-serverless/docs
2. Google Cloud. (2024). BigQuery Public Datasets - theLook eCommerce. https://console.cloud.google.com/marketplace/product/bigquery-public-data/thelook-ecommerce
3. Apache Spark. (2024). PySpark Documentation. https://spark.apache.org/docs/latest/api/python/
4. Google Cloud. (2024). Dataproc Workflow Templates. https://cloud.google.com/dataproc/docs/concepts/workflows/overview
5. Databricks. (2023). Medallion Architecture. https://www.databricks.com/glossary/medallion-architecture
6. Google Cloud. (2024). BigQuery Spark Connector. https://cloud.google.com/dataproc/docs/tutorials/bigquery-connector-spark-example
7. Apache Parquet. (2024). Parquet Documentation. https://parquet.apache.org/docs/