ETL PIPELINE FOR SALES DATA ANALYSIS USING GOOGLE CLOUD

Software and Data Engineering (Mini Project)

Richard David (M24CSE019)

Vishwanath Singh (M24CSE030)

Sachin Singh (M24CSE033)

Abstract

This project develops a comprehensive sales data pipeline using Google Cloud services to automate data extraction, transformation, and secure storage. By leveraging tools such as Cloud Composer, Cloud Data Fusion, and BigQuery, the pipeline integrates data masking techniques to ensure sensitive information is protected during processing. The final output is visualized using Looker Studio, enabling real-time analysis and actionable insights for better decision-making. In addition to the cloud-based solution, a non-cloud implementation using PySpark and Matplotlib is developed for comparison, showcasing how local processing can serve as an alternative for smaller datasets or resource-limited scenarios. The comparison highlights the strengths of cloud platforms in terms of scalability, automation, and flexibility, while also demonstrating where non-cloud solutions may still be viable. This dual approach underscores the adaptability of both methods, providing a holistic view of ETL pipeline design for diverse business requirements.

Table of Contents

1. Introduction

In today's information-driven world, organizations rely heavily on sales data for decision-making. However, handling sensitive data presents challenges, particularly in ensuring privacy and security. This project addresses these issues by creating a comprehensive data pipeline to extract, transform, and load (ETL) sales data into a scalable cloud-based data warehouse. By using Google Cloud services like Composer, Data Fusion, BigQuery, and Looker Studio, the pipeline integrates privacy measures such as data masking to comply with security regulations. It automates the process of visualizing sales data, providing secure, real-time insights to support informed decision-making.

The project primarily focuses on efficiently handling sales data from various sources and loading it into BigQuery for analysis and reporting. With Looker Studio, the system enables real-time trend analysis, helping stakeholders access actionable insights on sales performance. Additionally, a non-cloud implementation was developed to compare the benefits and limitations of each approach. The non-cloud pipeline, using PySpark and Matplotlib, processes and visualizes data locally, offering a solution for organizations with smaller datasets or limited cloud resources.

By comparing both implementations, this project highlights the advantages of cloud infrastructure—scalability, automation, and real-time processing—while recognizing the usefulness of non-cloud solutions for certain use cases. This dual approach emphasizes the flexibility of both methods in meeting diverse business needs, illustrating how cloud-based platforms can streamline data operations, enhance analysis, and help businesses remain agile. The non-cloud comparison provides additional insights into scenarios where local solutions may still be applicable, especially for smaller projects or developmental testing.

2. Project Objectives

The main objective of this project is to design and implement a cloud-based ETL pipeline that securely processes sales data by extracting, transforming, and loading it into a scalable data warehouse for analysis. Using Google Cloud tools such as Cloud Composer, Data Fusion, and BigQuery, the pipeline ensures data privacy through masking techniques and provides real-time data visualization via Looker Studio. Additionally, the project aims to compare this cloud-based approach with a non-cloud implementation using PySpark and Matplotlib, offering insights into the benefits and limitations of each, particularly in terms of scalability, automation, and ease of use.

3. Literature Review

Recent studies highlight the growing importance of automated data pipelines in addressing the challenges of data-driven decision-making. As businesses increasingly rely on data to optimize operations and enhance customer experiences, the need for efficient data processing becomes critical. Automated pipelines, such as ETL (Extract, Transform, Load) and ELT (Extract, Load, Transform), simplify the movement and transformation of large datasets without manual intervention. [1] These pipelines enable real-time data analysis and seamless integration, allowing organizations to generate actionable insights more effectively. The literature underscores how automated data pipelines are transforming data processing by improving efficiency, scalability, and decision-making capabilities.

In the age of digitalization, the vast amount of data generated from various sources, referred to as Big Data, poses significant challenges for storage, processing, and analysis using traditional methods. Cloud computing offers a solution, providing scalable storage, processing power, and remote access to resources on a pay-as-you-go model.[2] Cloud platforms such as Google Cloud, AWS, IBM, and Microsoft offer robust solutions for managing and analyzing Big Data. This paper focuses on Google's BigQuery, a data warehouse service designed for real-time analysis and decision-making, which enables users to handle large datasets efficiently and cost-effectively, offering substantial advantages for business intelligence and quick data-driven insights.

The literature emphasizes the increasing complexity of modern data pipelines, especially in organizations handling large and complex datasets. Traditional data pipelines involve cleaning, aggregating, and transferring data, but as data volumes grow to petabytes, modern pipelines must be faster, more robust, and capable of providing real-time analysis. [3] These pipelines must also operate in the cloud while offering notifications and reliability for repeated tasks. Tools like Apache Airflow have become essential for managing such pipelines, as they simplify pipeline creation using DAGs (Directed Acyclic Graphs) and Operators, requiring only basic Python knowledge. This paper highlights the implementation of Airflow for stock-exchange data pipelines, showcasing its practical application in handling large-scale data tasks efficiently.

The literature underscores the increasing complexity of handling vast amounts of data generated daily across multiple platforms such as social media and search engines. Traditional data processing methods struggle to cope with the volume, velocity, and variety of this data, necessitating modern solutions like Apache Spark.[4] Apache Spark, an in-memory cluster computing system, provides faster data processing capabilities, making it a key player in Big Data Analytics. The study explores the Spark ecosystem, including its components, libraries, and deployment modes, while also comparing Python and Scala as programming languages for Spark. The findings suggest that both languages are effective for Spark, with the choice depending on project-specific requirements. This comparative analysis aids programmers in selecting the appropriate language for their Apache Spark projects based on desired features and performance.

Cloud migration is a critical topic for organizations seeking to transition from on-premise architectures to cloud-based solutions. Research on this subject emphasizes the importance of understanding migration processes across the three primary layers of cloud computing: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS).[5] Studies based on provider-driven case studies and expert interviews highlight common migration activities specific to each layer. These processes reveal both commonalities and differences, while identifying challenges such as integration, security, and scalability. The literature suggests that addressing these issues is crucial for smooth cloud transitions, with additional insights provided from an independent systems integration perspective.

Drawing insights from the research on data-driven decision-making, automated data pipelines, and cloud migration processes, we are developing our project with a focus on building a comprehensive, cloud-based ETL pipeline. The project leverages the scalability and automation capabilities of cloud platforms, enabling seamless extraction, transformation, and loading of sales data. By incorporating best practices from studies on cloud infrastructure, platform services, and data pipeline efficiency, the project aims to streamline data processing, ensure real-time insights, and enhance decision-making processes. Our approach aligns with the literature by focusing on efficiency, security, and scalability, essential for modern data-driven businesses.

4. Tools and Technologies Used

This project leverages a combination of cloud-based services and local tools to implement and compare the sales data pipeline:

i. **Google Cloud Composer (Airflow):** Google Cloud Composer is fully managed workflow orchestration service based on Apache Airflow and creates easier scheduling and more complex monitoring of data workflows. Cloud Composer offers a flexible environment for defining tasks and dependencies, which can help data engineers and analysts automate data pipelines and streamline their data processing to make sure it is delivered reliably and on time.

ii. **Google Cloud Data Fusion:** Google Cloud Data Fusion is the fully managed, cloud-native service that is specifically designed to help users build and manage their pipelines for their data. This visually designed interface makes it easier for any organization to quickly integrate data from all sorts of sources. With all its connectors in-built and data transformation ability, Data Fusion makes it easier to move and prepare data for analysis.

iii. **Google Cloud Storage:** The main objective of Google Cloud Storage is to designed for storing and retrieving virtually any volume of user-generated data at any time. It offers high level of durability and availability which are more suitable for applications where large volumes of unstructured data, such as images, videos, and the like are stored. Thus, Cloud Storage provides flexible and cost-effective solutions for its numerous user applications, with several classes tailored for different use cases.

iv. **Google BigQuery:** It is a fully managed, serverless data warehouse that gives a fast SQL query and big data set analysis. It is to be used for big data analytics, thus allowing users to quickly run complex queries using a pay-as-you-go model. BigQuery architecture supports real-time analytics and machine learning capabilities.

v. **Looker Studio:** Looker Studio, formerly Google Data Studio, is a powerful tool designed to help create interactive dashboards and reports for data visualization and reporting. With the user-friendly interface of Looker Studio, users can easily connect to various sources of data, view them through customizable charts and graphs, and share insights across teams. Generally, this will improve data-driven decision-making by making information accessible and available to stakeholders in meaningful visualizations and reports.

vi. **PySpark:** In the non-cloud implementation, PySpark is employed for local data processing and transformation. PySpark provides the ability to handle large datasets efficiently by distributing the workload across multiple processors in a local environment. It performs the same ETL (Extract, Transform, Load) functions as Google Cloud Data Fusion but requires more manual setup and coding. PySpark allows for operations like filtering, grouping, and aggregating data, making it a powerful tool for handling sales data locally. However, it is limited by the processing power of the machine it's run on, making it less scalable compared to cloud-based solutions.

vii. **Matplotlib:** Matplotlib is used for data visualization in the non-cloud implementation. This Python-based plotting library is ideal for creating static visualizations such as pie charts, bar graphs, and histograms. In this project, Matplotlib is used to visualize the sales data after processing it with PySpark. Though it provides a simpler, more basic approach to visualizing data compared to cloud tools like Looker Studio, it is highly customizable and useful for local, small-scale projects. However, it lacks the interactivity and real-time update features of cloud-based visualization platforms.

5. Architecture Overview

The architecture of the sales data pipeline is built using a combination of Google Cloud services, ensuring a secure, scalable, and automated data processing workflow. The diagram illustrates the flow of data from extraction to visualization, emphasizing privacy and security through the use of cloud services.

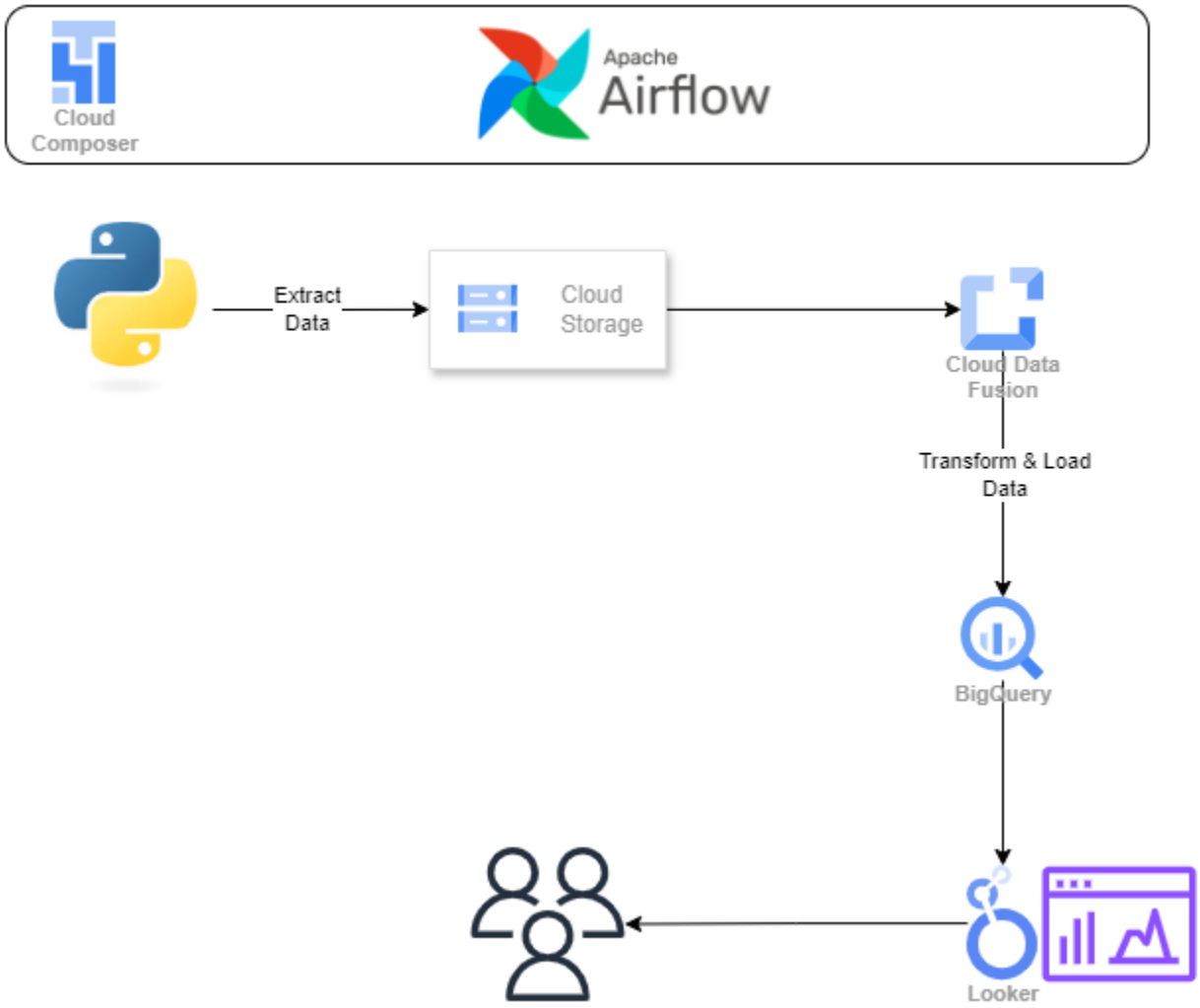


Figure 5: Architecture Diagram

1. **Data Extraction**: The pipeline begins with a Python script, managed by Cloud Composer (Airflow), that extracts sales data from various sources (e.g., databases, APIs, or flat files). This extracted data is stored temporarily in Google Cloud Storage (GCS) for further processing.

2. **Cloud Storage**: Once the data is extracted, it is stored in Google Cloud Storage, acting as the intermediary between extraction and transformation. GCS ensures secure, reliable, and scalable data storage.

3. **Data Transformation and Loading**: The next step involves Cloud Data Fusion, which ingests the raw sales data from Cloud Storage. Data Fusion applies necessary transformations, including masking sensitive information such as customer details. The transformed data is then securely loaded into Google BigQuery, a cloud-based data warehouse designed for fast querying and analysis of large datasets.

4. **Data Visualization**: Finally, Looker Studio is used for data visualization. It connects to the BigQuery dataset and creates interactive dashboards, allowing stakeholders to visualize sales trends and performance metrics in real-time. Looker provides secure, scalable dashboards that can be shared across the organization.

5. **Orchestration**: The entire process is orchestrated using Apache Airflow within Cloud Composer, automating the data pipeline. Airflow ensures that tasks such as data extraction, transformation, and loading are scheduled and executed without manual intervention.

This architecture is designed to handle both large and small datasets efficiently, ensuring data privacy and real-time insights. The system leverages cloud resources for scalability and automation, providing a comprehensive solution for sales data analysis.

6. Cloud Implementation using Google Cloud Platform

The cloud implementation of the sales data pipeline using Google Cloud Platform (GCP) integrates several services to create a scalable, automated, and secure ETL process. Cloud Composer (Apache Airflow) orchestrates the workflow, automating the extraction of sales data and its storage in Google Cloud Storage (GCS). Cloud Data Fusion then transforms the data, applying necessary transformations such as data masking to ensure privacy before loading it into BigQuery, GCP's scalable data warehouse. Finally, Looker Studio provides real-time, interactive visualizations for stakeholders to analyze sales performance. This cloud-based solution is ideal for large-scale data operations, offering flexibility, security, and cost-efficiency.

6.1. Step-by-Step Process for Cloud Implementation

This section outlines the entire process of building and automating the sales data pipeline. Each step is described in detail, providing a clear understanding of how the various tools and technologies interact within the architecture.

Step 1: Setting Up Cloud Composer Environment

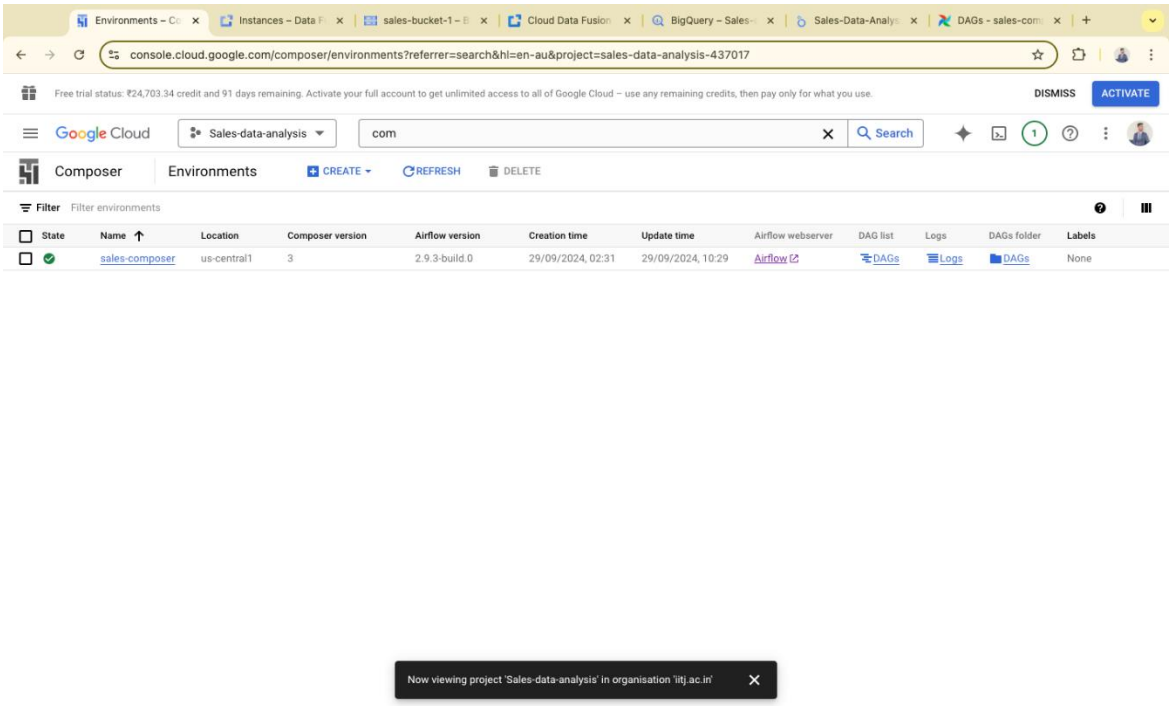


Figure 6.1: Cloud Composer Enviroment

To begin the project, we set up a Cloud Composer environment to orchestrate the pipeline using Apache Airflow. Start by navigating to the Google Cloud Console and selecting Cloud Composer from the navigation menu. Click Create environment, provide a name for the environment, and choose a location (e.g., us-central1). Depending on the expected scale of your DAG (Directed Acyclic Graph) tasks, configure the environment settings, such as the Node count and Machine type. Once these settings are finalized, click Create and wait for the environment to be provisioned. This setup process may take a few minutes.

Step 2: Create a Cloud Data Fusion Instance

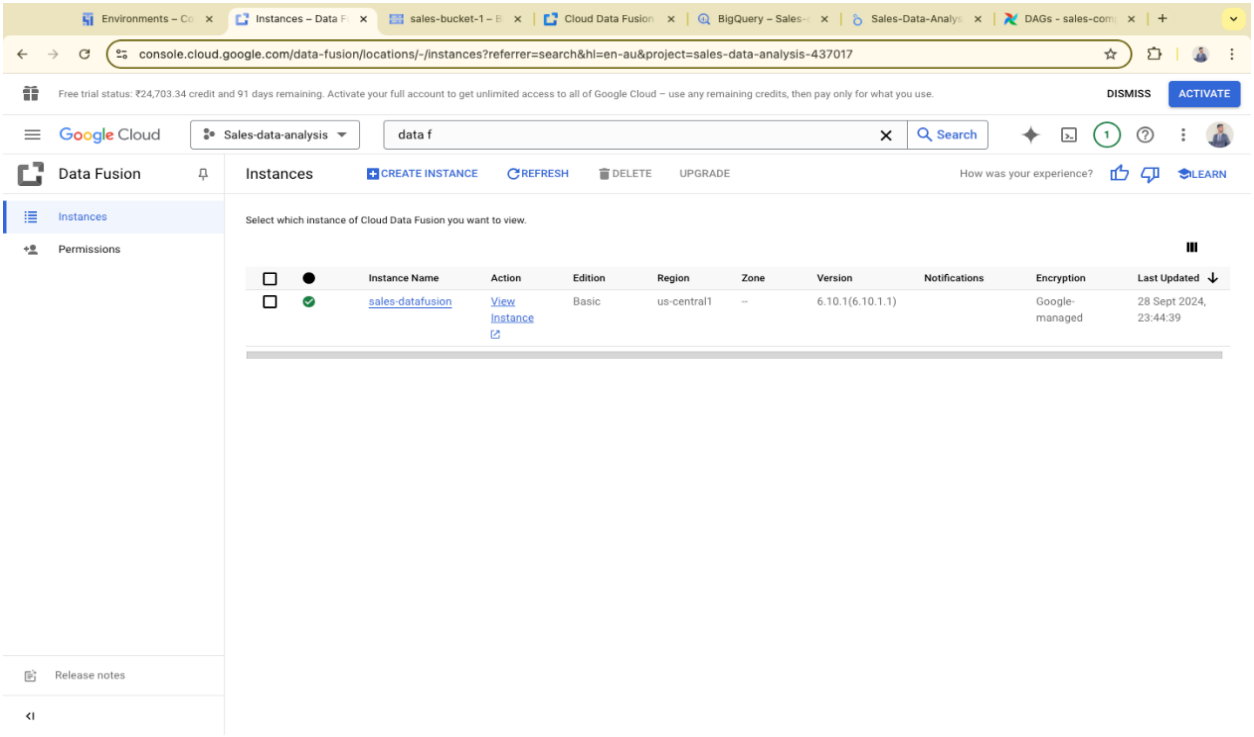


Figure 6.2: Cloud Data Fusion Instance

Next, create a Cloud Data Fusion instance to manage data transformation tasks. Search for Cloud Data Fusion in the Google Cloud Console and click Create Instance. Choose the appropriate region (e.g., us-central1) and provide a unique instance name (e.g., sales-datafusion). You will be prompted to choose between the Basic or Enterprise edition based on your project needs. After making your selection, click Create. Once the instance is ready, you can start building transformation pipelines.

Step 3: Create a Google Cloud Storage (GCS) Bucket

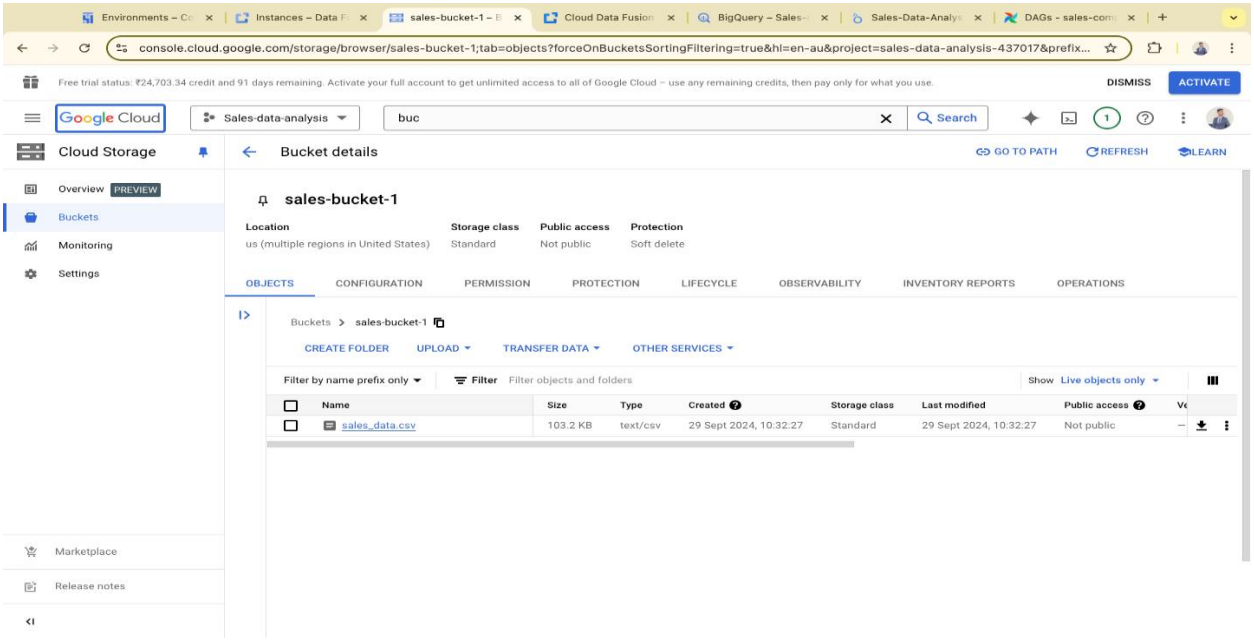


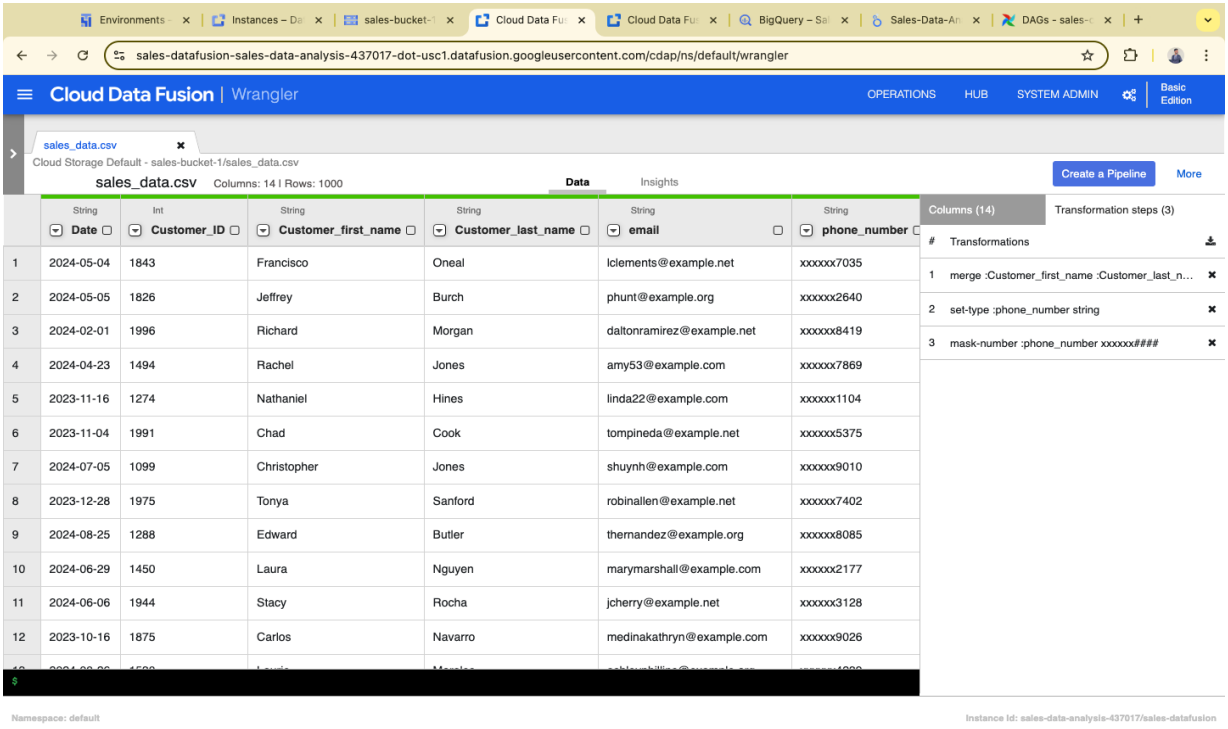
Figure 6.3: Google Cloud Storage (GCS)

With the Cloud Data Fusion instance in place, the next step is to create a Google Cloud Storage (GCS) bucket where extracted sales data will be stored temporarily. In the Google Cloud Console, navigate to Cloud Storage and click Create bucket. Choose a globally unique name for the bucket (e.g., sales-bucket-1), select the same region as your Data Fusion instance and Composer environment, and configure the storage class (e.g., Standard). Once configured, click Create to complete the setup.

Step 4: Run Python Code and Check the Bucket

With the GCS bucket ready, the next task is to run a Python script (e.g., extract.py) that generates synthetic or real sales data. You can execute this code on your local machine or within Cloud Shell. Once the data has been generated, upload the resulting CSV file (sales_data.csv) to the GCS bucket. This can be done using the gsutil command. After uploading, verify that the file was successfully added by checking the GCS bucket through the Google Cloud Console.

Step 5: Perform Data Transformation (Masking Personal Information)



	Date	Customer_ID	Customer_first_name	Customer_last_name	email	phone_number
1	2024-05-04	1843	Francisco	Oneal	lclements@example.net	xxxxxx7035
2	2024-05-05	1826	Jeffrey	Burch	phunt@example.org	xxxxxx2640
3	2024-02-01	1996	Richard	Morgan	daltonramirez@example.net	xxxxxx8419
4	2024-04-23	1494	Rachel	Jones	amy53@example.com	xxxxxx7869
5	2023-11-16	1274	Nathaniel	Hines	linda22@example.com	xxxxxx1104
6	2023-11-04	1991	Chad	Cook	tompineda@example.net	xxxxxx5375
7	2024-07-05	1099	Christopher	Jones	shuynh@example.com	xxxxxx9010
8	2023-12-28	1975	Tonya	Sanford	robinallen@example.net	xxxxxx7402
9	2024-08-25	1288	Edward	Butler	thernandez@example.org	xxxxxx8085
10	2024-06-29	1450	Laura	Nguyen	marymarshall@example.com	xxxxxx2177
11	2024-06-06	1944	Stacy	Rocha	jcherry@example.net	xxxxxx3128
12	2023-10-16	1875	Carlos	Navarro	medinakathryn@example.com	xxxxxx9026

Figure 6.4: Data Transformation

After the data is stored in GCS, it must be transformed before being loaded into BigQuery. In Cloud Data Fusion, add a Data Transformation step to the pipeline to clean the data and apply transformations such as masking sensitive information. For example, replace sensitive phone numbers with a masked format (XXXX-XXXX), or hide personal information such as email addresses. You can use Data Fusion's built-in transformation features or write custom transformation logic depending on your requirements.

Step 6: Create a Data Fusion Pipeline

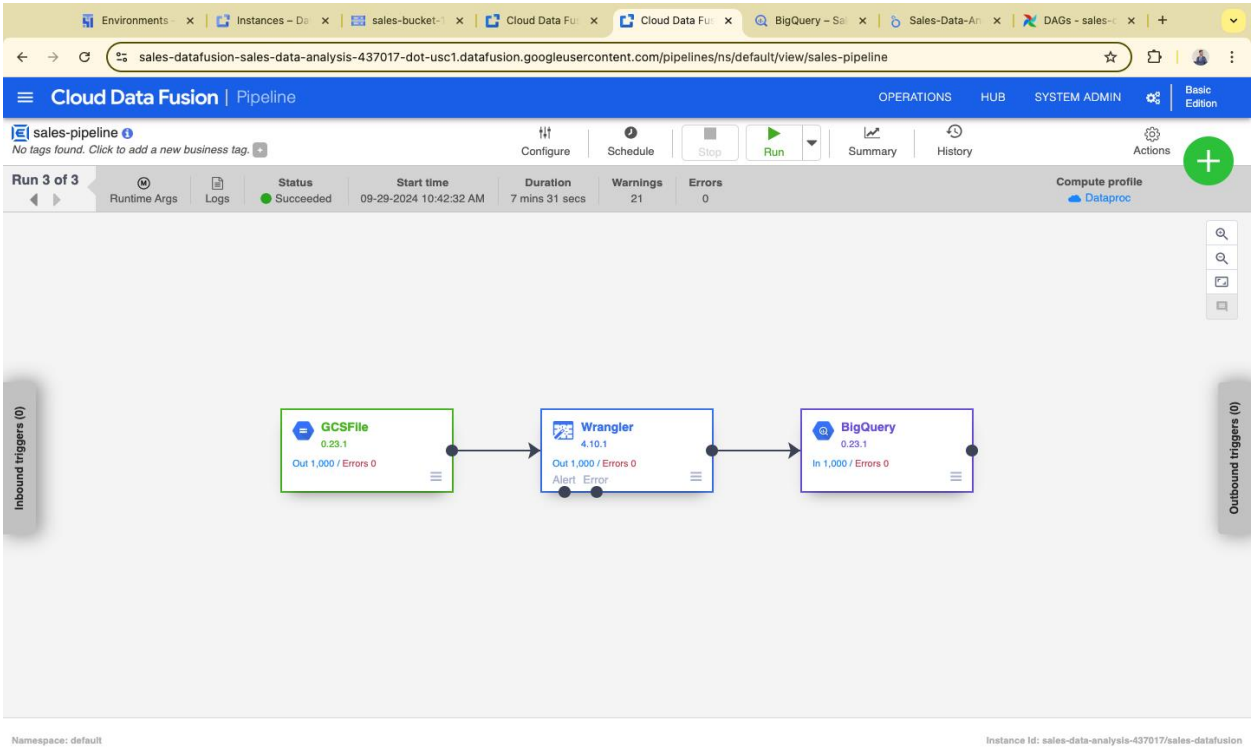


Figure 6.5: Data Fusion Pipeline

To automate the ETL process, create a Batch Pipeline in Cloud Data Fusion. In the Data Fusion interface, click Studio and select Create Pipeline. Add a GCS Source and point it to the file in your bucket (e.g., gs://sales-bucket-1/sales_data.csv). Ensure that you correctly configure the file format (CSV with headers) to prevent issues during transformation.

Step 7: Add a BigQuery Sink

With the data source configured, the next step is to define where the transformed data will be stored. Add a BigQuery Sink to the Data Fusion pipeline and configure it to load the cleaned and transformed data into a BigQuery table. Ensure that the schema is properly mapped, particularly for sensitive fields like the masked phone numbers, to maintain data integrity.

Step 8: Set BigQuery Sink Properties

In this step, configure the BigQuery Sink by specifying the dataset location (e.g., us-central1) and setting up the target dataset (e.g., sales_data_analysis). Choose a table name where the

transformed data will be stored. Ensure that you set the appropriate write disposition (e.g., append or overwrite) to avoid overwriting critical data unless necessary.

Step 9: Go to BigQuery and Create a Dataset

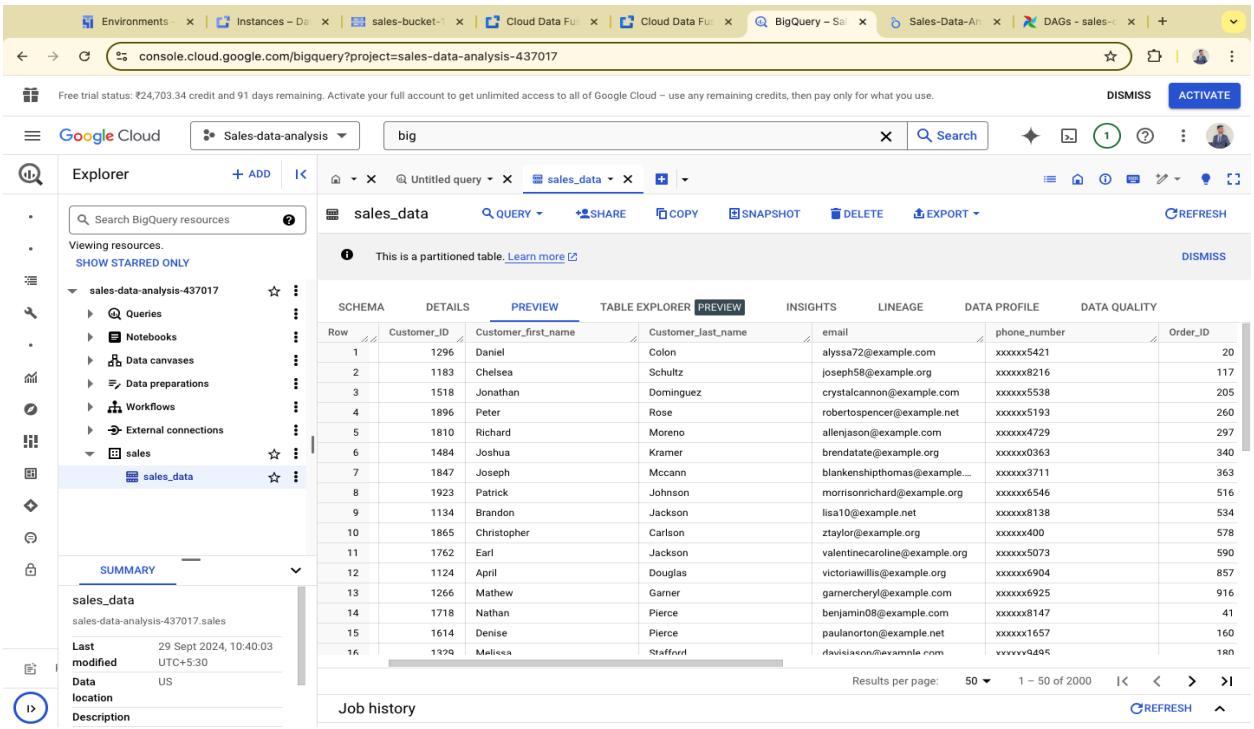


Figure 6.6: Dataset creation at BigQuery

Before running the pipeline, you must first create a BigQuery Dataset where the data will be loaded. Open BigQuery in the Google Cloud Console, click Create Dataset, and provide the necessary details such as the dataset ID (sales_data_analysis). Ensure the dataset is located in the same region as your GCS bucket and Data Fusion instance. Afterward, you can use this dataset as the destination for the transformed data.

Step 10: Complete BigQuery Sink Property Configuration

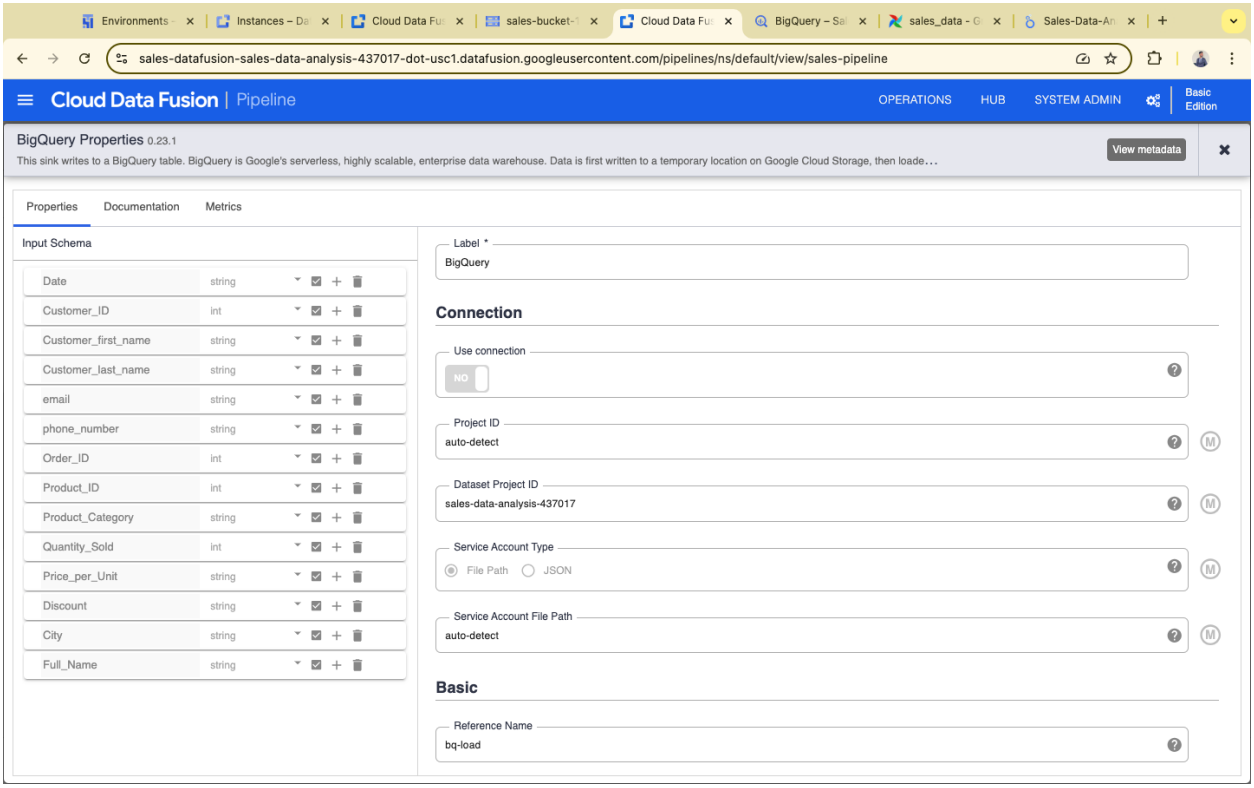


Figure 6.7: BigQuery sink property configuration

After creating the dataset, return to the Data Fusion pipeline and set the dataset you created as the sink for the BigQuery table. Review all the field mappings carefully to ensure that the transformed data, particularly the masked fields, is mapped correctly to the appropriate columns in BigQuery.

Step 11: Run the Data Fusion Pipeline

After setting the pipeline, trigger a run by clicking Run in the Data Fusion interface. Monitor the job logs for any errors or issues that may arise during the execution. Once the pipeline completes, verify that the transformed data has been successfully loaded into BigQuery by running queries to check for data integrity and accuracy.

Step 13: Visualize the Data in Looker Studio

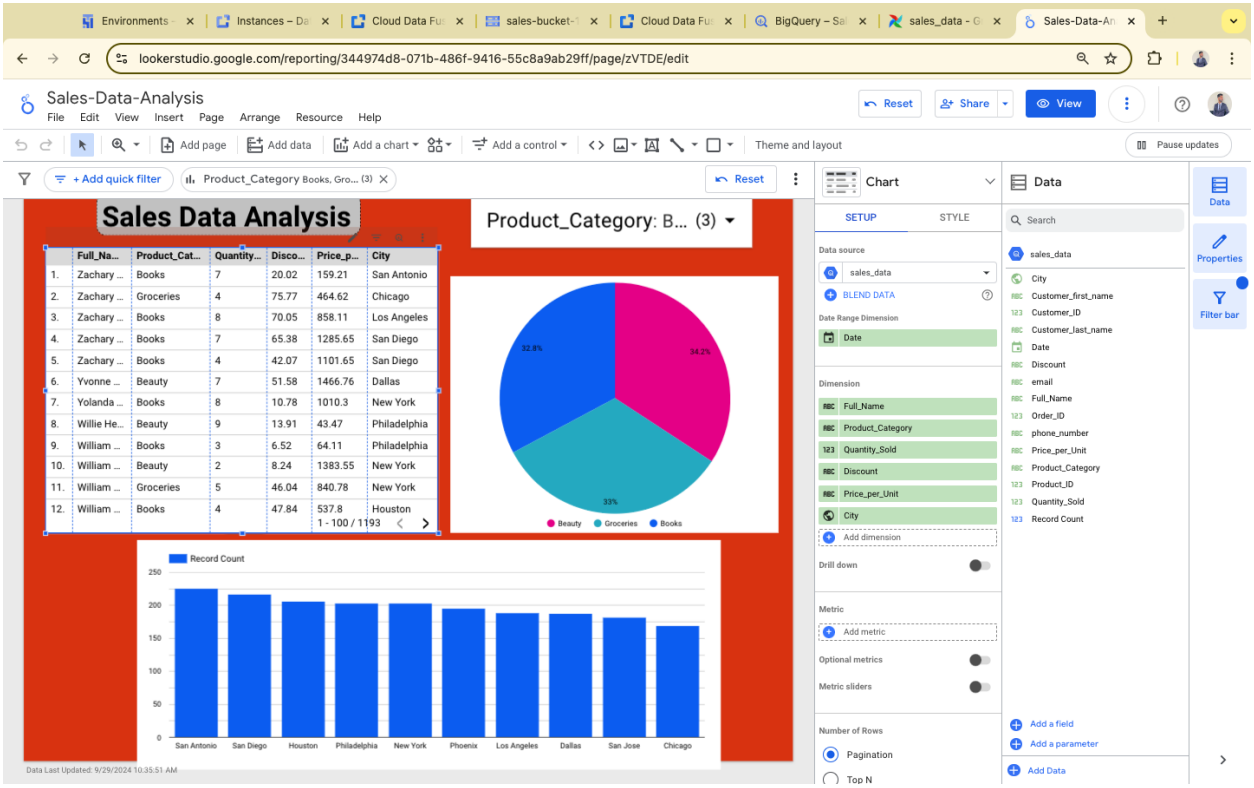


Figure 6.8: Data Visualize in Looker Studio

After the data is loaded into BigQuery, visualize the results using Looker Studio. Open Looker Studio and create a new report. Connect the report to the BigQuery dataset (sales_data_analysis) created earlier. Design the dashboard by adding relevant charts and graphs to visualize sales data and performance metrics. Ensure that all sensitive data remains masked as per your transformation rules.

Step 14: Automate the Pipeline using Airflow (Composer)

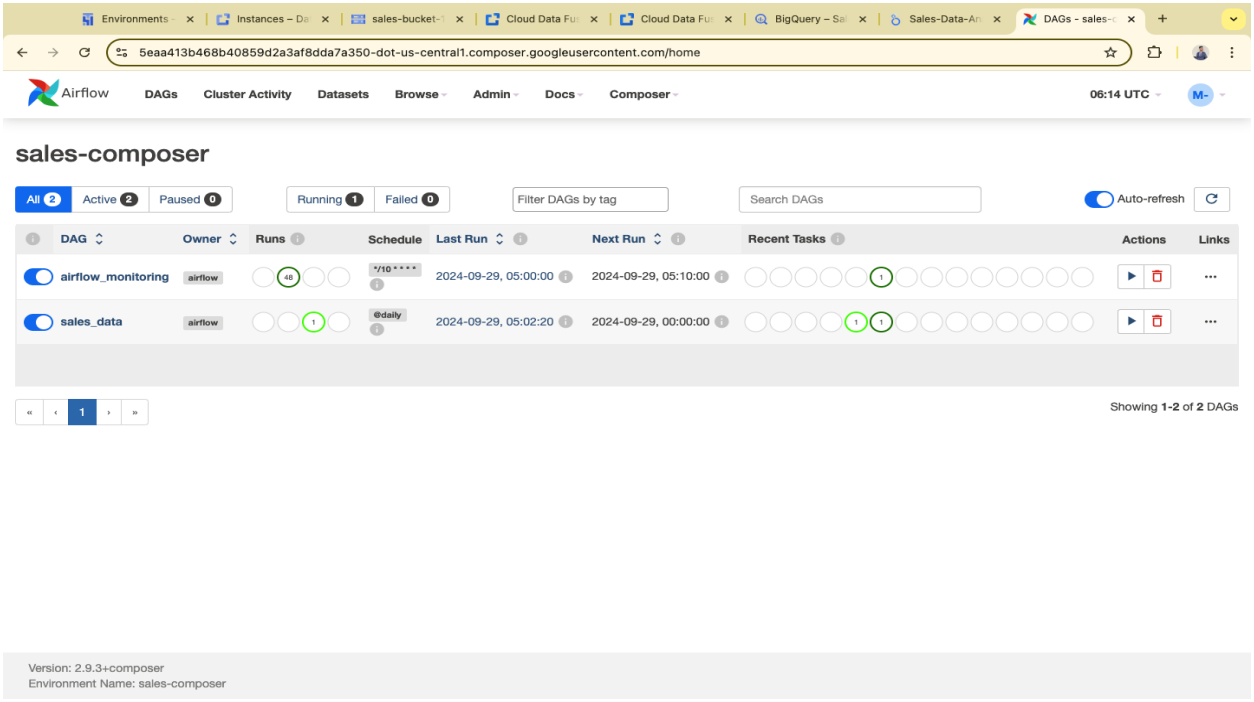


Figure 6.9: Pipeline using Airflow

To automate the entire process, use Airflow within Cloud Composer. Start by creating a folder (e.g., scripts) in your Composer environment and upload the Python script (extract.py) that generates the sales data. This will enable Airflow to run the extraction script automatically.

Step 15: Write the Airflow DAG

In the same Cloud Composer environment, write a DAG (dag.py) that orchestrates the entire ETL pipeline. The DAG should include tasks to run the Python extraction script and trigger the Data Fusion pipeline. Once the DAG is written, upload it to the Composer environment to automate the pipeline execution.

Step 16: Trigger the Airflow DAG

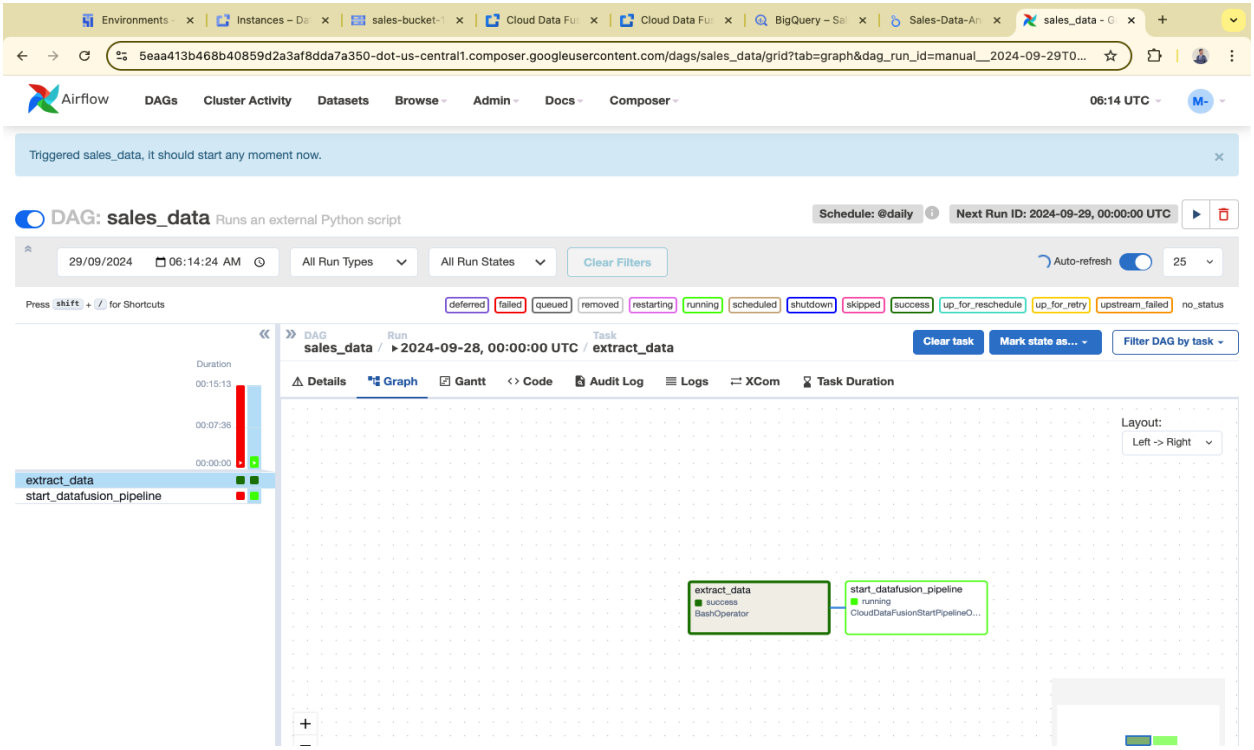


Figure 6.10: Airflow DAG

Finally, navigate to the DAGs tab in Cloud Composer and locate the sales_data DAG. Trigger it manually to verify that all tasks are executed as expected. The pipeline will now execute on the defined schedule (e.g., daily) to ensure automated extraction, transformation, and loading of sales data into BigQuery.

6.2. Use Cases for Cloud Implementation

The cloud implementation is ideal for:

Large-Scale Data Processing: Organizations handling vast datasets, such as retail or e-commerce businesses, benefit from cloud scalability, allowing seamless storage and processing of large volumes of sales data.

Real-Time Sales Analytics: Businesses that require real-time insights can leverage cloud services like BigQuery and Looker Studio to monitor sales trends and make quick, informed decisions based on live data.

Automated and Scheduled Workflows: Cloud Composer enables automated ETL workflows, ensuring continuous data processing without manual intervention, making it ideal for daily or real-time updates.

Data Security and Compliance: Industries handling sensitive information (e.g., healthcare, finance) benefit from the data masking and encryption features of Data Fusion, ensuring regulatory compliance.

Collaboration and Accessibility: Teams across different locations can access and collaborate on data in real time, thanks to the cloud's centralized storage and interactive dashboards.

Cost-Efficiency: The pay-as-you-go pricing model allows businesses to scale without heavy upfront infrastructure costs, making it cost-effective for growing organizations.

7. Non-Cloud Implementation (PySpark and Matplotlib)

In addition to the cloud-based pipeline, a non-cloud implementation was developed to provide an alternative for organizations with smaller-scale data or limited access to cloud resources. This local solution utilizes PySpark for data processing and Matplotlib for data visualization. The goal of this implementation is to demonstrate how the same sales data can be handled and visualized locally, offering a comparison between cloud-based scalability and the limitations of local environments.

7.1. Data Processing with PySpark

In the non-cloud implementation, PySpark—a powerful tool for large-scale data processing—is used to perform the Extract, Transform, Load (ETL) operations locally. PySpark enables distributed data processing on a local machine, making it ideal for handling moderately large datasets without requiring cloud services. The PySpark script performs the following tasks:

Data Extraction: The sales data, stored in a local CSV file, is loaded into a PySpark DataFrame.

Data Transformation: Using PySpark's built-in transformation functions, the data is grouped by product category, with sales quantities and other metrics calculated. Additionally, custom transformations can be applied, such as filtering out unnecessary data or calculating total sales revenue per category.

Data Loading: After transformations, the processed data is saved locally as a new CSV file for further analysis or reporting.

While PySpark is an efficient tool for local data processing, it is limited by the resources of the local machine (such as CPU, memory, and disk space). This makes it less scalable compared to cloud platforms like Google Cloud Data Fusion, which can automatically adjust resources based on workload. However, for small to medium datasets, PySpark provides a viable alternative for organizations that may not need the extensive infrastructure provided by cloud services.

7.2. Data Visualization with Matplotlib

For visualizing the processed data, the **Matplotlib** library is used. Matplotlib is a widely used Python library for creating static, two-dimensional plots, such as bar charts, pie charts, and line graphs. In this implementation, the sales data—specifically the distribution of orders across different product categories—is visualized using a **pie chart**. The steps include:

Data Preparation: The grouped data from PySpark is converted into a Pandas DataFrame to simplify the visualization process.

Visualization: A pie chart is created using Matplotlib to visualize the proportion of sales per product category. The pie chart visually demonstrates the distribution of sales, allowing stakeholders to see which product categories generate the most sales.

Static Output: The resulting visualization is saved as a static image (e.g., PNG format), which can be included in reports or presentations.

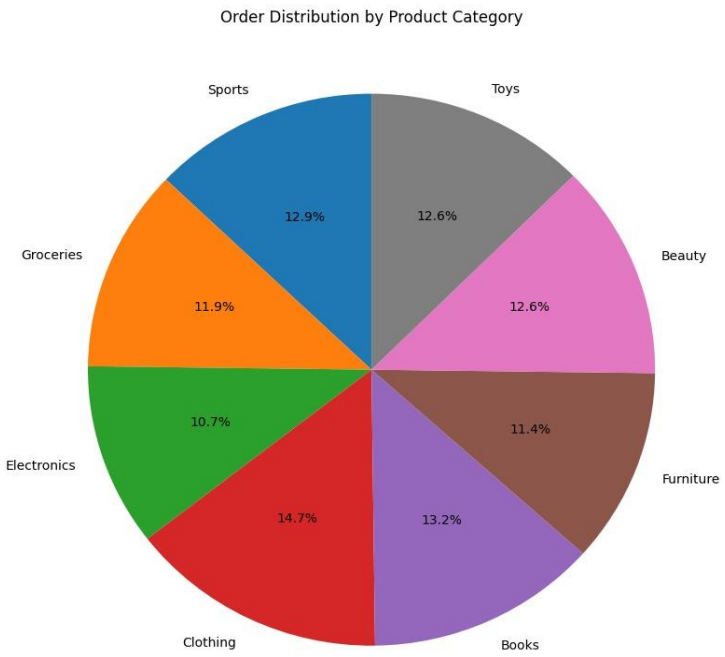


Figure 7.1: Data visualization using Matplotlib

While Matplotlib is easy to use and highly customizable for creating basic visualizations, it lacks the interactivity and real-time update capabilities of cloud-based tools like **Looker Studio**. Matplotlib is well-suited for scenarios where static visualizations are sufficient, but it cannot offer the dynamic, shareable dashboards that cloud-based solutions provide.

7.3. Limitations of Non-Cloud Implementation

Although the non-cloud solution using PySpark and Matplotlib is efficient for small-scale projects, it comes with certain limitations:

Scalability: PySpark, when run locally, is constrained by the hardware of the local machine. Processing larger datasets can lead to performance bottlenecks, making it impractical for big data scenarios.

Automation: Unlike cloud platforms that can automate and orchestrate complex workflows, the non-cloud implementation requires manual execution, which can be time-consuming and prone to errors if not properly managed.

Interactivity: Matplotlib provides static visualizations, which are useful for reports but do not allow for interactive exploration of data. In contrast, tools like Looker Studio enable dynamic, real-time dashboards that can be shared across an organization.

7.4. Use Cases for Non-Cloud Implementation

The non-cloud implementation is ideal for:

Development and Testing: It provides a local environment where data engineers and analysts can develop and test data transformations before deploying them to a cloud environment.

Small-Scale Projects: Organizations with small datasets or those in early-stage development may benefit from the simplicity and cost-effectiveness of a non-cloud solution.

Offline Data Processing: In cases where internet access is limited or unavailable, the non-cloud implementation can still perform data processing and visualization.

Overall, the non-cloud implementation serves as a useful comparison to the cloud-based solution, demonstrating that while it can effectively handle smaller projects, it lacks the scalability,

automation, and interactivity that cloud platforms offer. By integrating both cloud and non-cloud implementations into this project, a comprehensive understanding of the strengths and limitations of each approach is provided.

8. Comparison: Cloud Platform vs Non-Cloud Implementation

The cloud-based implementation using Google Cloud Platform and the non-cloud implementation using PySpark and Matplotlib offer two distinct approaches to building an ETL pipeline for sales data processing and visualization. Each has its own strengths and limitations, depending on the project's size, resource availability, and long-term scalability needs. Below is a detailed comparison of both implementations based on several critical factors:

i. Scalability

- **Cloud Platform**: The cloud-based solution is highly scalable, making it ideal for businesses handling large datasets. Services like **Google Cloud Storage** and **BigQuery** allow seamless handling of vast amounts of data, automatically scaling up as data volume grows. Cloud resources dynamically allocate processing power and storage, meaning businesses do not need to worry about infrastructure limitations.

- **Non-Cloud Implementation**: The non-cloud solution, which uses PySpark for processing, is limited by the hardware and resources available locally. As the dataset grows, performance can degrade significantly due to hardware constraints such as CPU, memory, and storage. For smaller datasets, this approach is effective, but it lacks the flexibility to scale to large datasets without significant performance degradation.

ii. Automation and Workflow Management

- **Cloud Platform**: With **Cloud Composer (Apache Airflow)**, the entire ETL pipeline is automated. Airflow manages the orchestration of tasks, ensuring that data extraction, transformation, and loading occur automatically based on a schedule or trigger. This minimizes the need for manual intervention and ensures the pipeline runs reliably, providing up-to-date insights.

- **Non-Cloud Implementation**: The non-cloud implementation lacks the automation capabilities of the cloud platform. Data engineers must manually run scripts to extract, transform, and load data. While this can work for small projects or one-off tasks, it is not

suitable for continuous, scheduled data processing without setting up complex local task schedulers like cron jobs, which still lack the robustness of cloud automation tools.

iii. Data Security and Privacy

- **Cloud Platform**: Google Cloud services come with built-in security features, including encryption at rest and in transit, IAM (Identity and Access Management) policies, and VPC (Virtual Private Cloud) networking. Additionally, **Cloud Data Fusion** supports data masking and transformation, ensuring sensitive information like customer data is securely processed before it reaches BigQuery. This makes it highly suitable for industries with strict data compliance requirements, such as healthcare and finance.

- **Non-Cloud Implementation**: Security in a non-cloud implementation is entirely dependent on local infrastructure and policies. While PySpark can handle data masking through manual coding, it lacks the built-in security features and compliance certifications that cloud platforms offer. Storing sensitive data locally also increases the risk of data breaches, especially if adequate security measures are not in place.

iv. Real-Time Data Processing and Analytics

- **Cloud Platform**: The cloud-based solution supports real-time data processing and analytics. **BigQuery** enables real-time querying of large datasets, providing instantaneous results. Additionally, **Looker Studio** allows users to build interactive dashboards that update in real-time, enabling stakeholders to make data-driven decisions quickly.

- **Non-Cloud Implementation**: The non-cloud solution, while effective for batch processing, lacks real-time processing capabilities. Data needs to be manually processed in batches, and visualizations with **Matplotlib** are static, meaning they don't update automatically as new data becomes available. This is a limitation when real-time insights are required, making it less suitable for dynamic business environments.

v. Cost Efficiency

- **Cloud Platform**: The cloud implementation follows a **pay-as-you-go** pricing model. Businesses only pay for the resources they use, making it cost-effective, especially for

growing organizations that need to scale up gradually. While there may be ongoing operational costs, these are often offset by the efficiency gains from automation, scalability, and real-time processing capabilities.

- **Non-Cloud Implementation**: The non-cloud approach incurs no direct infrastructure costs beyond existing local hardware, making it an appealing choice for small projects or organizations with tight budgets. However, as the dataset and processing needs grow, the cost of upgrading local hardware and maintaining infrastructure may become prohibitive. Furthermore, the lack of automation means higher human resource costs for manual operations.

vi. Data Visualization

- **Cloud Platform**: The cloud-based solution integrates seamlessly with **Looker Studio**, providing dynamic, real-time dashboards that allow stakeholders to interact with the data. These dashboards can be customized and shared across teams, making it ideal for collaborative environments where data insights need to be accessed by multiple users.

- **Non-Cloud Implementation**: The non-cloud solution relies on **Matplotlib** for data visualization, which provides static images such as pie charts or bar graphs. While Matplotlib is highly customizable for generating visualizations, it does not offer interactivity or real-time updates. As a result, it is limited to one-time data reporting and is not suited for continuous, collaborative environments where live updates are crucial.

vii. Maintenance and Upkeep

- **Cloud Platform**: Maintenance in the cloud is minimal. Google Cloud services like **BigQuery**, **Cloud Storage**, and **Cloud Data Fusion** are fully managed, meaning the cloud provider handles infrastructure updates, scaling, and security patches. This significantly reduces the burden on IT teams and ensures the system is always running efficiently with minimal downtime.

- **Non-Cloud Implementation**: Local implementations require significant maintenance efforts, including hardware upkeep, software updates, and performance monitoring. As

the system grows, maintaining the infrastructure can become complex and resource-intensive. Additionally, any issues that arise in the local environment must be resolved manually, which could lead to increased downtime and reduced productivity.

viii. Collaboration and Accessibility

- **Cloud Platform**: Cloud platforms are inherently designed for collaboration. Multiple users can access, query, and visualize data simultaneously using tools like **BigQuery** and **Looker Studio**. These platforms are accessible from anywhere, allowing teams across different regions to work together in real time.

- **Non-Cloud Implementation**: In contrast, non-cloud solutions are limited to the local environment, meaning access is restricted to individuals who can physically connect to the machine. Sharing data or results requires manual exporting and distributing of files, which can lead to version control issues and inefficient workflows.

The cloud-based implementation using Google Cloud Platform offers significant advantages in terms of scalability, automation, security, and real-time processing, making it an ideal choice for businesses dealing with large datasets or requiring real-time insights. It is also well-suited for collaborative environments where multiple teams need access to data. On the other hand, the non-cloud implementation using PySpark and Matplotlib is a cost-effective solution for smaller projects or in cases where internet connectivity is limited. However, it lacks the scalability, automation, and interactive features that cloud services provide, making it less suitable for larger or dynamic projects. Ultimately, the choice between cloud and non-cloud implementations depends on the organization’s data size, processing needs, budget, and the level of automation required.

9. Conclusion

This project successfully demonstrates the development of a comprehensive sales data pipeline using both cloud-based and non-cloud solutions. The cloud implementation, powered by Google Cloud services such as **Cloud Composer**, **Data Fusion**, **BigQuery**, and **Looker Studio**, provides a scalable, automated, and secure platform for handling large-scale data. It offers real-time data processing and visualization capabilities, enabling organizations to gain valuable, actionable insights from their sales data while ensuring data privacy through masking and encryption.

On the other hand, the non-cloud implementation using **PySpark** and **Matplotlib** offers a cost-effective and straightforward solution for small-scale projects or early-stage development, where internet access or cloud infrastructure might be limited. While it lacks the scalability, automation, and real-time capabilities of the cloud-based solution, it serves as a useful alternative for testing and development or offline data processing.

By comparing both approaches, this project highlights the flexibility and adaptability of modern data pipelines to cater to various business needs, ranging from small, localized solutions to enterprise-level, cloud-driven architectures. Ultimately, the choice between cloud and non-cloud solutions depends on the organization's data size, processing requirements, and long-term scalability needs. The cloud implementation proves to be the ideal choice for organizations seeking to harness the power of real-time data analytics and automation, while the non-cloud solution remains suitable for specific use cases that require simplicity and cost-efficiency.

References

[1] R. Gandhi, S. Khurana, and H. Manchanda, “ETL Data Pipeline to Analyze Scraped Data,” in *Decision Intelligence*, B. K. Murthy, B. V. R. Reddy, N. Hasteer, and J.-P. Van Belle, Eds., Singapore: Springer Nature Singapore, 2023, pp. 379–388.

[2] M. H. Ali, M. S. Hosain, and M. A. Hossain, “Big Data analysis using BigQuery on cloud computing platform,” *Australian JofEng Inno Tech*, vol. 3, no. 1, pp. 1–9, 2021.

[3] S. Shukla, “Developing Pragmatic Data Pipelines using Apache Airflow on Google Cloud Platform,” *International Journal of Computer Sciences and Engineering Open Access Research Paper*, vol. 10, no. 8, 2022, doi: 10.26438/ijcse/v10i8.18.

[4] Y. K. Gupta and S. Kumari, “A Study of Big Data Analytics using Apache Spark with Python and Scala,” in *2020 3rd International Conference on Intelligent Sustainable Systems (ICISS)*, 2020, pp. 471–478. doi: 10.1109/ICISS49785.2020.9315863.

[5] C. Pahl, H. Xiong, and R. Walshe, “A Comparison of On-Premise to Cloud Migration Approaches,” in *Service-Oriented and Cloud Computing*, K.-K. Lau, W. Lamersdorf, and E. Pimentel, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 212–226.

Appendix

Git Repository Link : https://github.com/sachinsingh2156/ETL_Pipeline-SDE-Mini_Project.git

Project Demonstration Link: https://drive.google.com/file/d/1PC4C521mAyP2gNUaZKhHBQrHhmP-l14L/view?usp=sharing