**Sales Data Generation and Uploading to Google Cloud Storage (GCS)**

**extract.py**

This Python script generates synthetic sales data for analysis purposes and uploads the generated data to Google Cloud Storage (GCS). It is important to note that, while real-time sales data from an actual business can be used in production scenarios, this script generates **synthetic data** as a placeholder to simulate sales data for development and testing purposes. This data can later be replaced with actual business sales data once available.

**Key Components of the Script**

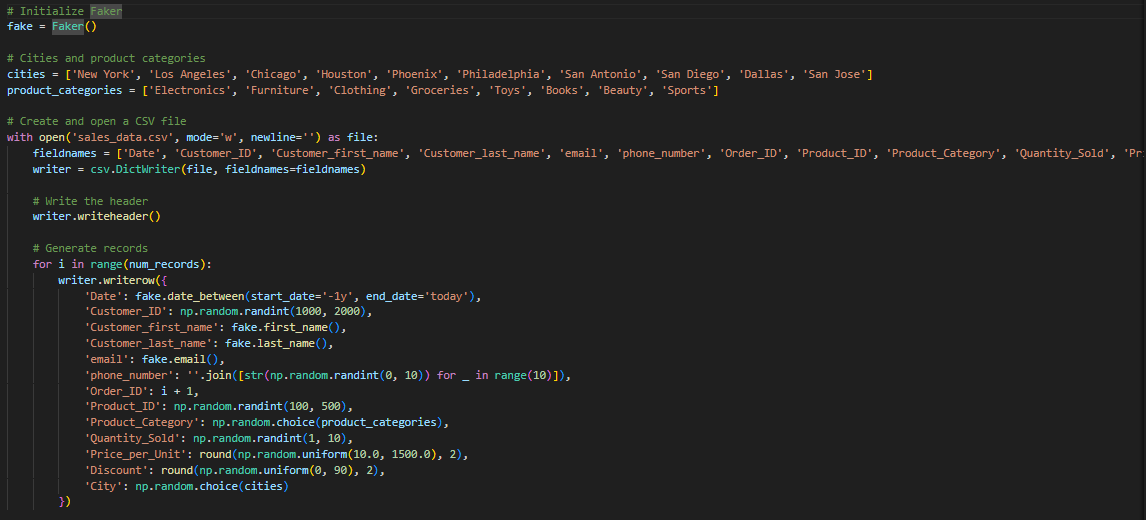
**1. Libraries Used**

**A screen shot of a computer

Description automatically generated**

* **Faker:** This library is utilized to generate realistic synthetic data, such as names, emails, phone numbers, and dates.
* **Google Cloud Storage (GCS):** This library facilitates interactions with GCS, allowing the script to upload files to the cloud.
* **csv:** This module is used for writing data into a CSV file.
* **numpy and pandas:** While primarily known for data analysis, numpy is used here for generating random numbers efficiently.

**2. Data Generation Process**

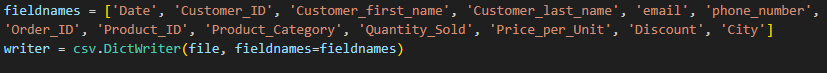
****

The script generates **1,000 synthetic sales records**. For each record, several data fields are created, including customer information, product details, transaction data, and geographic locations. Here's a breakdown of the process:

* **Customer Information:** Includes randomly generated customer IDs, first and last names, emails, and phone numbers.
* **Product Information:** Each record includes a product ID, product category (e.g., electronics, clothing), quantity sold, and price per unit.
* **Sales Details:** A sales record includes an order ID, discount percentage, and the city where the sale occurred. The dates for the sales are randomly chosen within the past year.

The cities and product categories are hardcoded, but in a real-world scenario, these could be retrieved dynamically from a business database.

**3. Data Fields**

****

Each synthetic sales record includes the following fields:

* **Date:** The sale date, randomly chosen from the last year.
* **Customer\_ID:** A unique identifier for the customer.
* **Customer\_first\_name/Customer\_last\_name:** The customer's first and last names.
* **email:** The customer's email address.
* **phone\_number:** The customer's phone number.
* **Order\_ID:** A unique identifier for the sales order.
* **Product\_ID:** A unique identifier for the product.
* **Product\_Category:** The category the product belongs to (e.g., Electronics, Groceries).
* **Quantity\_Sold:** The number of units sold in this transaction.
* **Price\_per\_Unit:** The price per unit of the product.
* **Discount:** Any applicable discount for the product.
* **City:** The city where the sale occurred.

**4. Uploading to Google Cloud Storage**

**A screen shot of a computer

Description automatically generated**

Once the sales data is generated and written to a CSV file (sales\_data.csv), the script uses Google Cloud Storage (GCS) to upload the file to a specified GCS bucket. The function upload\_to\_gcs handles the uploading process:

* **bucket\_name:** The name of the GCS bucket where the file will be uploaded.
* **source\_file\_name:** The name of the local file to be uploaded (in this case, sales\_data.csv).
* **destination\_blob\_name:** The name to be given to the file once uploaded to GCS.

**5. Usage Instructions**

**Setting Up Google Cloud**

1. **Install Google Cloud SDK:** Ensure that Google Cloud SDK is installed and properly configured on your machine. You must authenticate your credentials using gcloud auth login.
2. **Set up a GCS Bucket:** Create a GCS bucket (or use an existing one) to store the generated CSV file.

**6. Application in Real-World Scenarios**

While this script generates **synthetic data**, it is designed to simulate real-world sales data. In actual use, businesses could connect the data extraction step to real-time sales systems, such as databases, APIs, or data lakes, to replace the synthetic data with actual sales data. Once real data is integrated, businesses can conduct meaningful analyses, including customer behavior, sales trends, and product performance.

For now, this synthetic data serves as a useful placeholder for building and testing the **data pipeline**, allowing teams to refine and optimize their system before live data becomes available.

This script provides a starting point for creating a data pipeline involving data generation, processing, and storage in Google Cloud Storage. Although real-time business data is not currently available, this synthetic data allows you to prepare and test your workflows. Later, the same structure can be adapted to real-world data, supporting comprehensive sales analysis, data-driven decision-making, and strategic planning.

**Airflow DAG for Sales Data Pipeline Orchestration**

**dag.py**

This documentation outlines the steps involved in orchestrating a sales data pipeline using **Apache Airflow**. The DAG (Directed Acyclic Graph) defined in this script is responsible for running a Python script that extracts sales data and subsequently triggers a Cloud Data Fusion pipeline to transform and load the data.

**Key Components**

**1. Libraries and Imports**

**A screen shot of a computer

Description automatically generated**

* **datetime:** Provides date and time functionalities, used to define the start date and time intervals in the DAG.
* **timedelta:** Used for specifying time delays (e.g., retry delays) within the DAG's configuration.
* **DAG (Directed Acyclic Graph):** The fundamental concept in Airflow, representing the workflow and task dependencies.
* **BashOperator:** Executes bash commands or scripts as part of an Airflow task.
* **CloudDataFusionStartPipelineOperator:** Used to trigger a Cloud Data Fusion pipeline from Airflow. This operator is provided by Google Cloud.

**2. DAG Configuration**

**A computer screen shot of a program code

Description automatically generated**

The DAG is defined with default parameters that include the following:

* **owner:** Specifies the owner of the DAG (set to 'airflow' by default).
* **start\_date:** The date from which the DAG should start running. In this case, it is set to **September 29, 2024**.
* **depends\_on\_past:** If set to True, the current task will run only after the previous one is successful. Here, it is False to allow independent task execution.
* **email:** Specifies an email address where notifications should be sent in case of failures or retries. (The email address is 'm24cse033@iitj.ac.in').
* **retries:** Defines how many times a task should be retried upon failure. Set to 1 in this case.
* **retry\_delay:** Specifies the delay before retrying a failed task. Set to 5 minutes using timedelta(minutes=5).

**3. DAG Schedule and Catchup**

**A computer screen shot of a program

Description automatically generated**

* **schedule\_interval:** The DAG is scheduled to run **daily** using the interval '@daily'. This means that the pipeline will execute once every day.
* **catchup:** This is set to False, which ensures that the DAG only runs from the defined start\_date onwards and does not attempt to run for previous untriggered dates.

**4. Task 1: Running the Data Extraction Script**

**A computer screen shot of code

Description automatically generated**

* **BashOperator:** The first task, run\_script\_task, uses the BashOperator to run a Python script located at /home/airflow/gcs/dags/scripts/extract.py.
* The purpose of this task is to extract the sales data (likely through interaction with a database, API, or other data source) and prepare it for the next stages of the pipeline.

**5. Task 2: Triggering the Cloud Data Fusion Pipeline**

* **CloudDataFusionStartPipelineOperator:** After the data extraction script runs, the DAG triggers a Cloud Data Fusion pipeline using this operator.
  + The pipeline is located in the **us-central1** region.
  + The Cloud Data Fusion pipeline is named **sales-pipeline**, and it is executed on the Cloud Data Fusion instance **sales-datafusion**.

The operator sends a request to start the data pipeline in Cloud Data Fusion, where the data transformation and loading processes occur.

**6. Task Dependencies**

The workflow follows a linear sequence:

* **run\_script\_task → start\_pipeline:** The DAG ensures that the Cloud Data Fusion pipeline will only start once the data extraction script has successfully executed.

**7. DAG Usage Instructions**

**Steps to Use This DAG:**

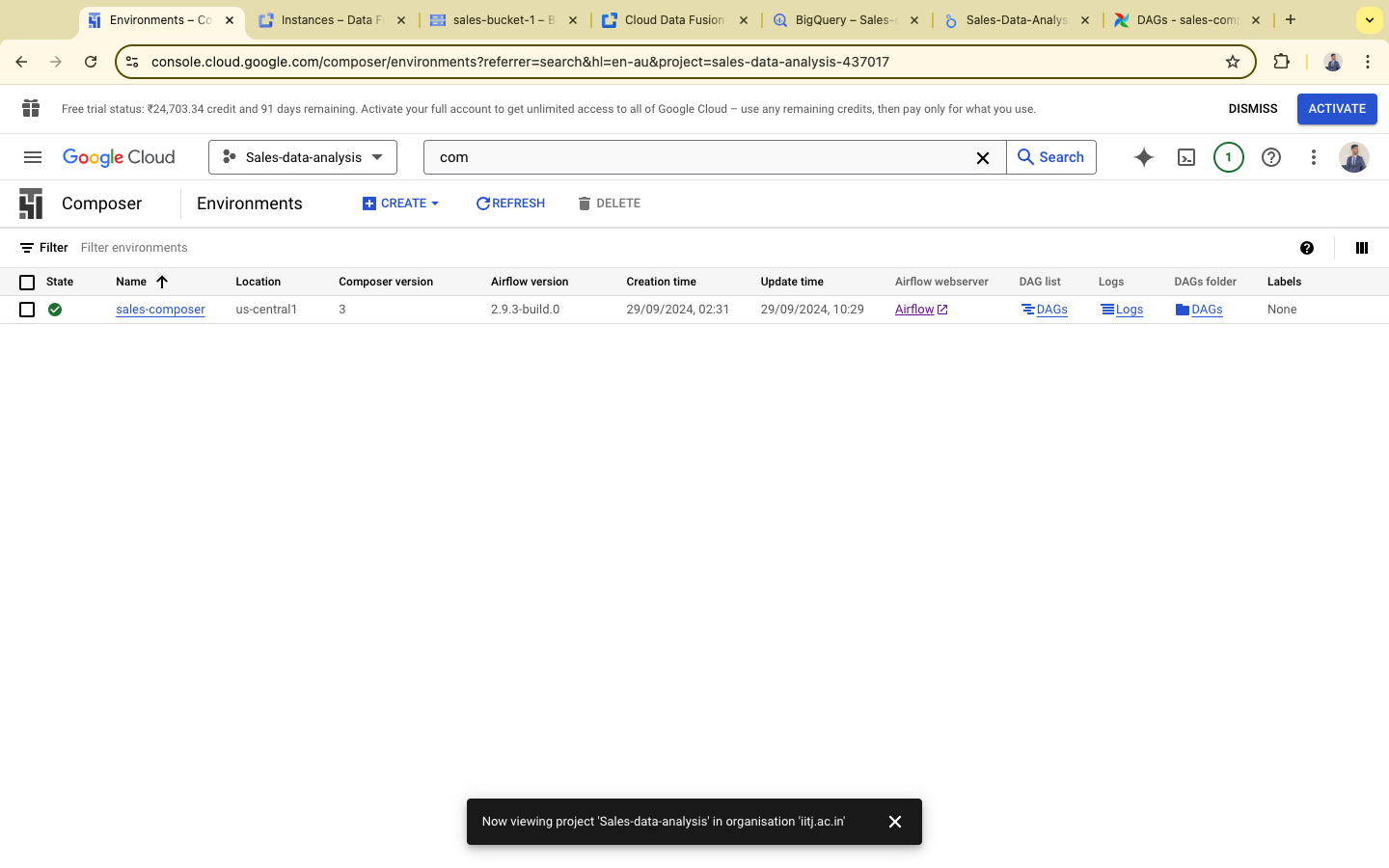
1. **Place Python Script in GCS:**
   * Ensure that the Python extraction script (extract.py) is placed in the appropriate directory on Google Cloud Storage (GCS). The path used in this DAG is /home/airflow/gcs/dags/scripts/extract.py.
2. **Create Cloud Data Fusion Pipeline:**
   * Ensure that a Cloud Data Fusion instance (sales-datafusion) and pipeline (sales-pipeline) are correctly set up. This pipeline should handle the transformation and loading of sales data into Google BigQuery or another destination.
3. **Run the DAG:**
   * Once the script and Data Fusion pipeline are in place, the Airflow scheduler will automatically run the DAG based on the schedule (@daily), extracting the sales data and triggering the Cloud Data Fusion pipeline daily.

This Airflow DAG automates the orchestration of a sales data pipeline by running a Python data extraction script and triggering a Cloud Data Fusion pipeline. The workflow is designed to ensure that data flows smoothly and efficiently through extraction, transformation, and loading stages. By utilizing Airflow's scheduling and monitoring features, the pipeline can run reliably and be easily maintained over time.

**Steps**

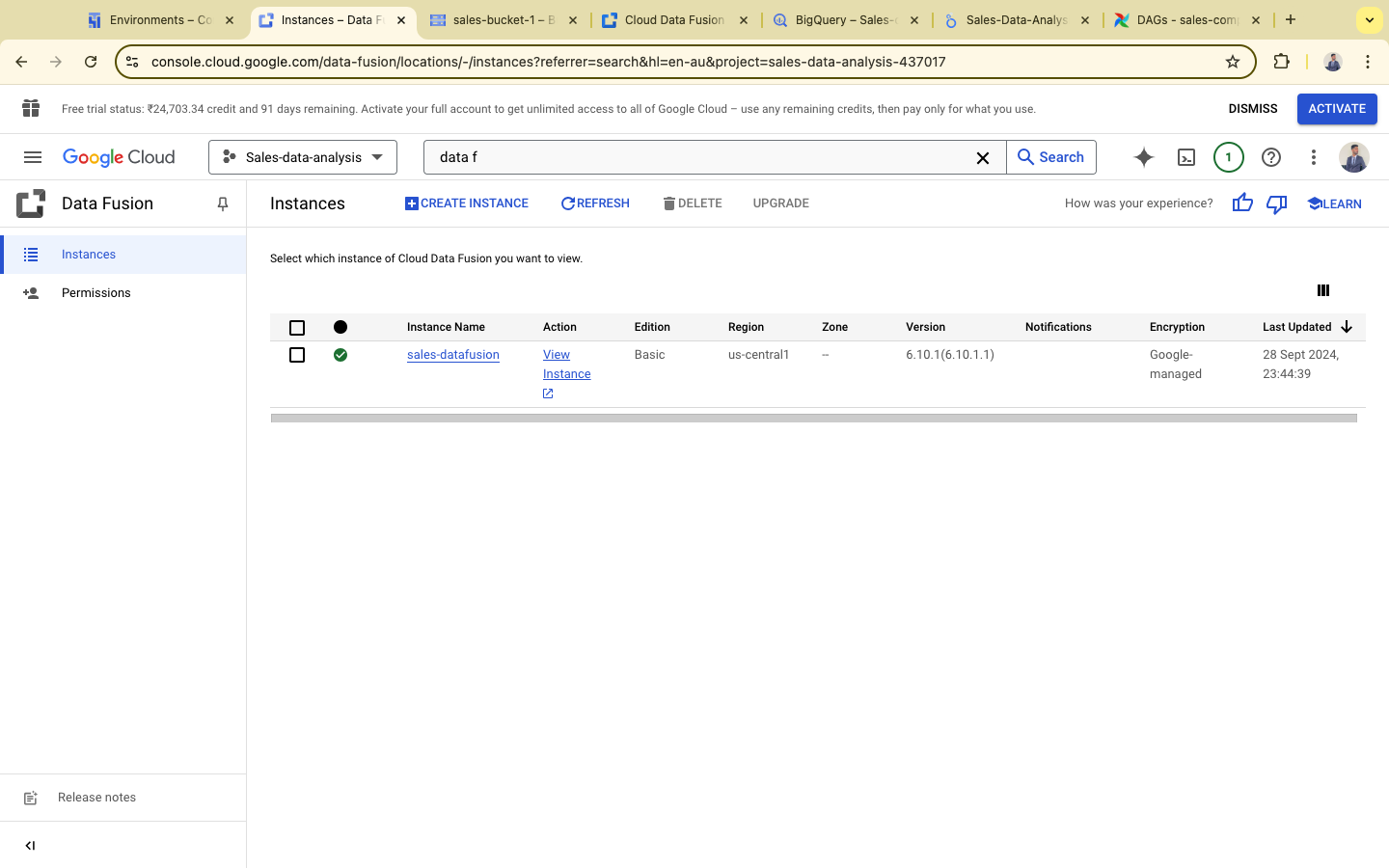
**Step-by-Step Guide for Sales Data ETL Pipeline Project**

**Step 1: Create a Cloud Composer Environment**

****

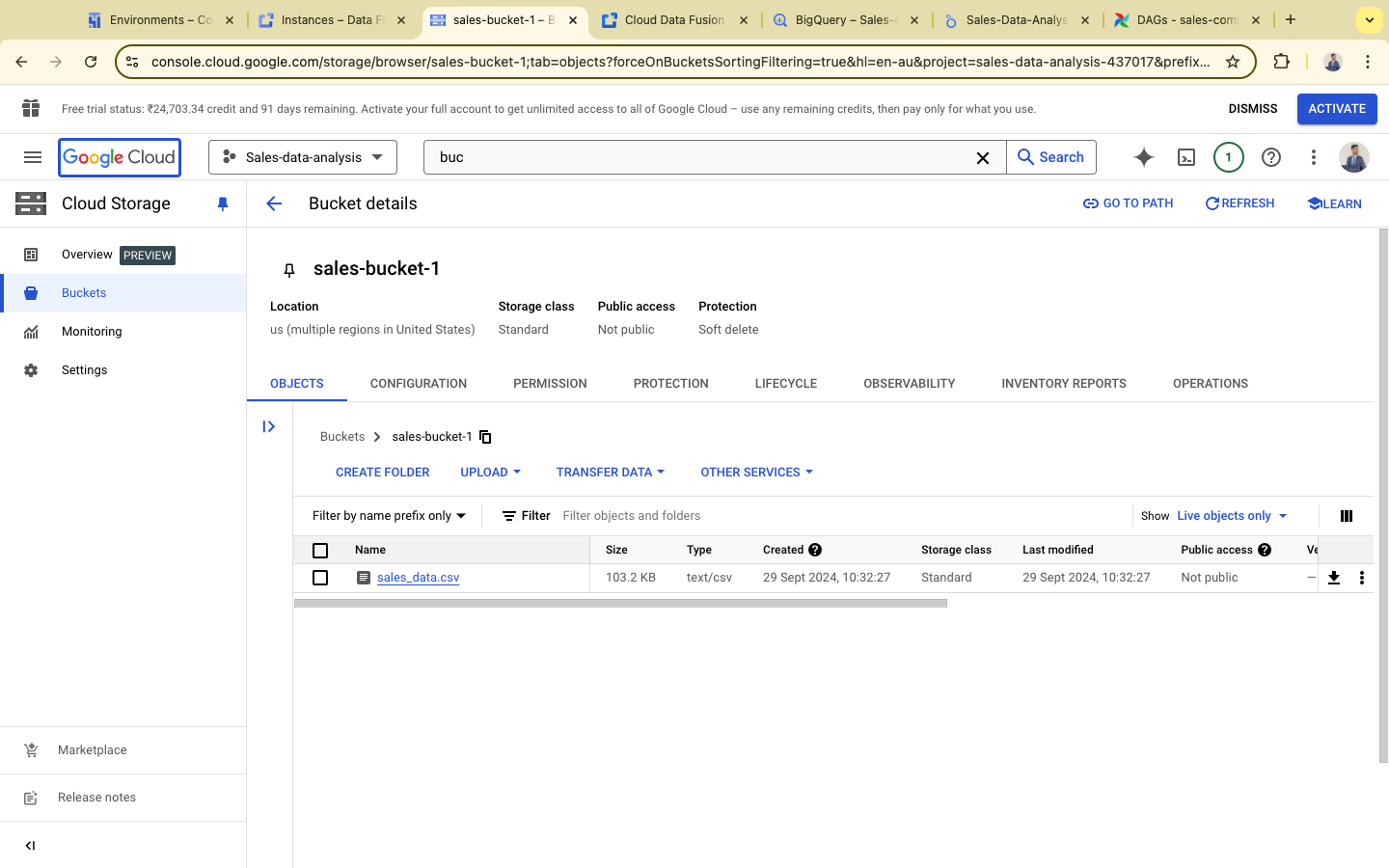
1. Open the **Google Cloud Console**.
2. Navigate to **Cloud Composer** from the navigation menu.
3. Click on **Create environment**.
4. Set your environment name and choose the **location** (e.g., us-central1).
5. Configure settings such as **Node count** and **Machine type** based on the scale of your DAG tasks.
6. Click **Create** and wait for the environment to be provisioned (this might take a few minutes).

**Step 2: Create a Cloud Data Fusion Instance**

****

1. From the **Google Cloud Console**, search for **Cloud Data Fusion**.
2. Click **Create Instance**.
3. Select the appropriate **region** (e.g., us-central1), and provide a unique instance name (e.g., sales-datafusion).
4. Choose **Basic** or **Enterprise** edition based on your needs.
5. Click **Create**. Once the instance is ready, you'll be able to start creating pipelines.

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

****

1. In the **Google Cloud Console**, go to **Cloud Storage** and click **Create bucket**.
2. Provide a globally unique name for your bucket (e.g., sales-bucket-1).
3. Choose a **region** matching your Data Fusion instance and Composer environment.
4. Select appropriate **storage class** (e.g., Standard).
5. Click **Create** to finish setting up your bucket.

**Step 4: Run Python Code and Check the Bucket**

1. Ensure your Python script for generating sales data (extract.py) is ready.
2. Use your local machine or an instance (e.g., Cloud Shell) to run the code and generate the CSV file.
3. Upload the generated sales\_data.csv to the GCS bucket (sales-bucket-1).

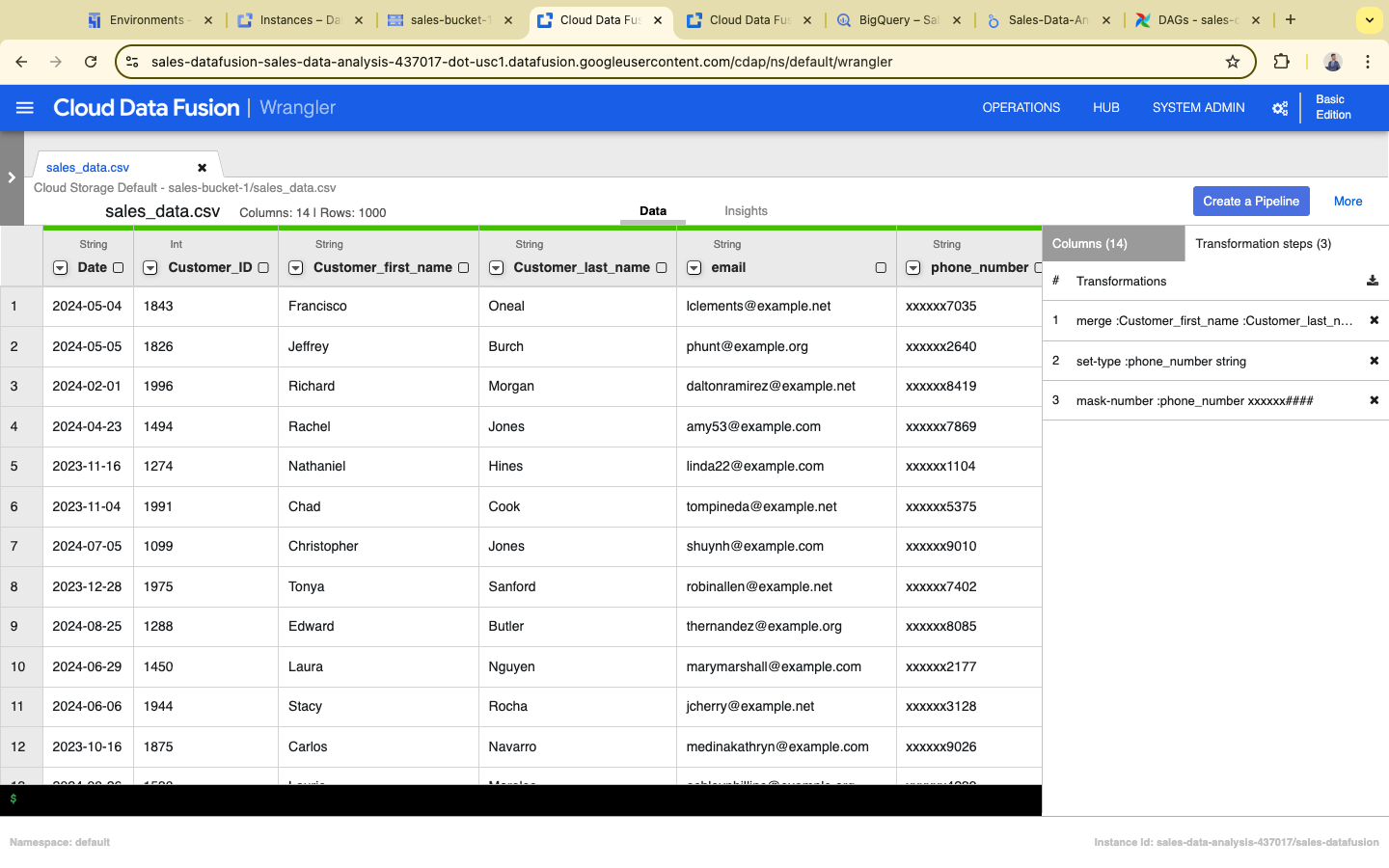
bash

Copy code

gsutil cp sales\_data.csv gs://sales-bucket-1/sales\_data.csv

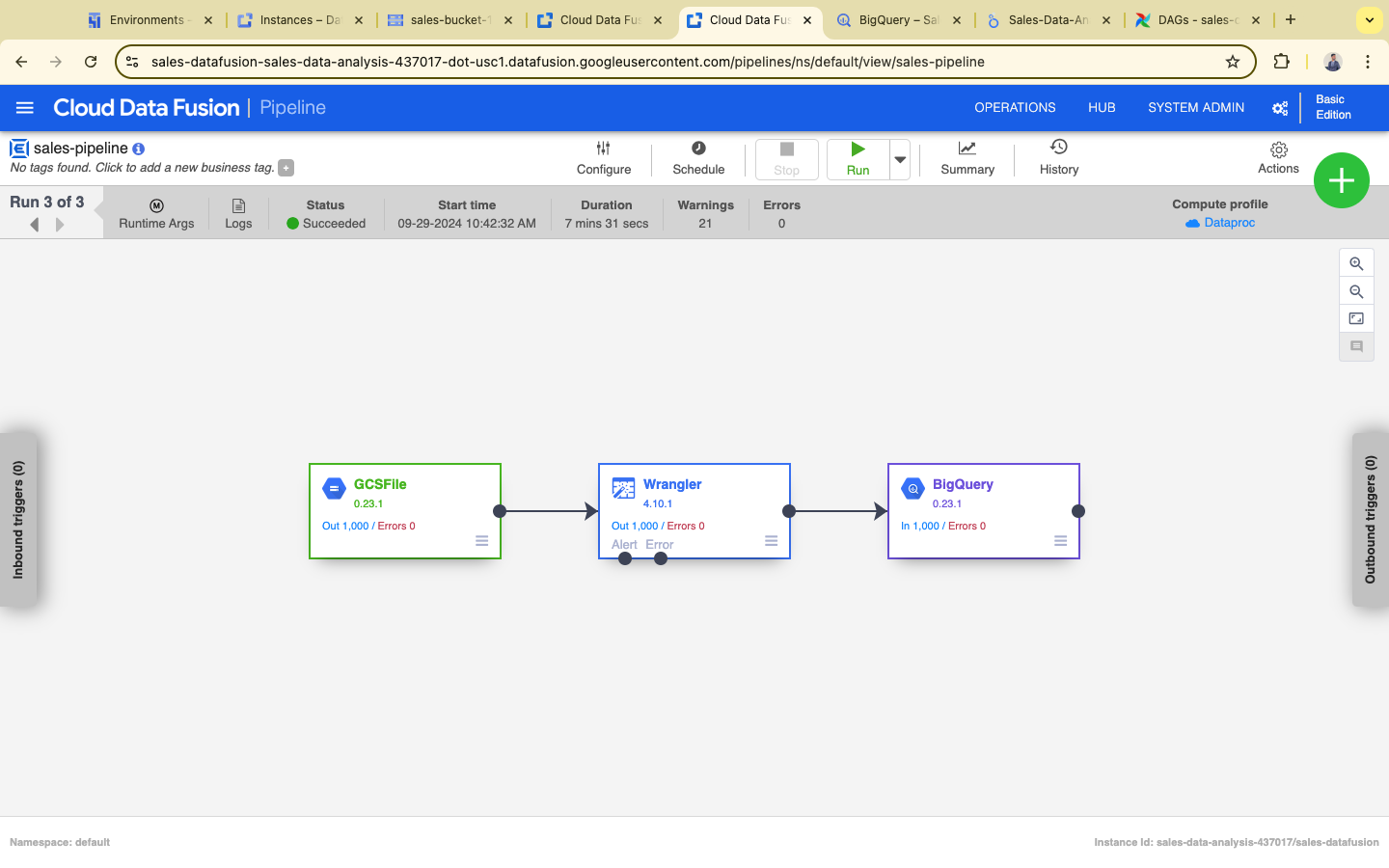
1. Verify that the file is successfully uploaded by checking the **Cloud Storage** bucket.

**Step 5: Perform Data Transformation (Masking Personal Information)**

****

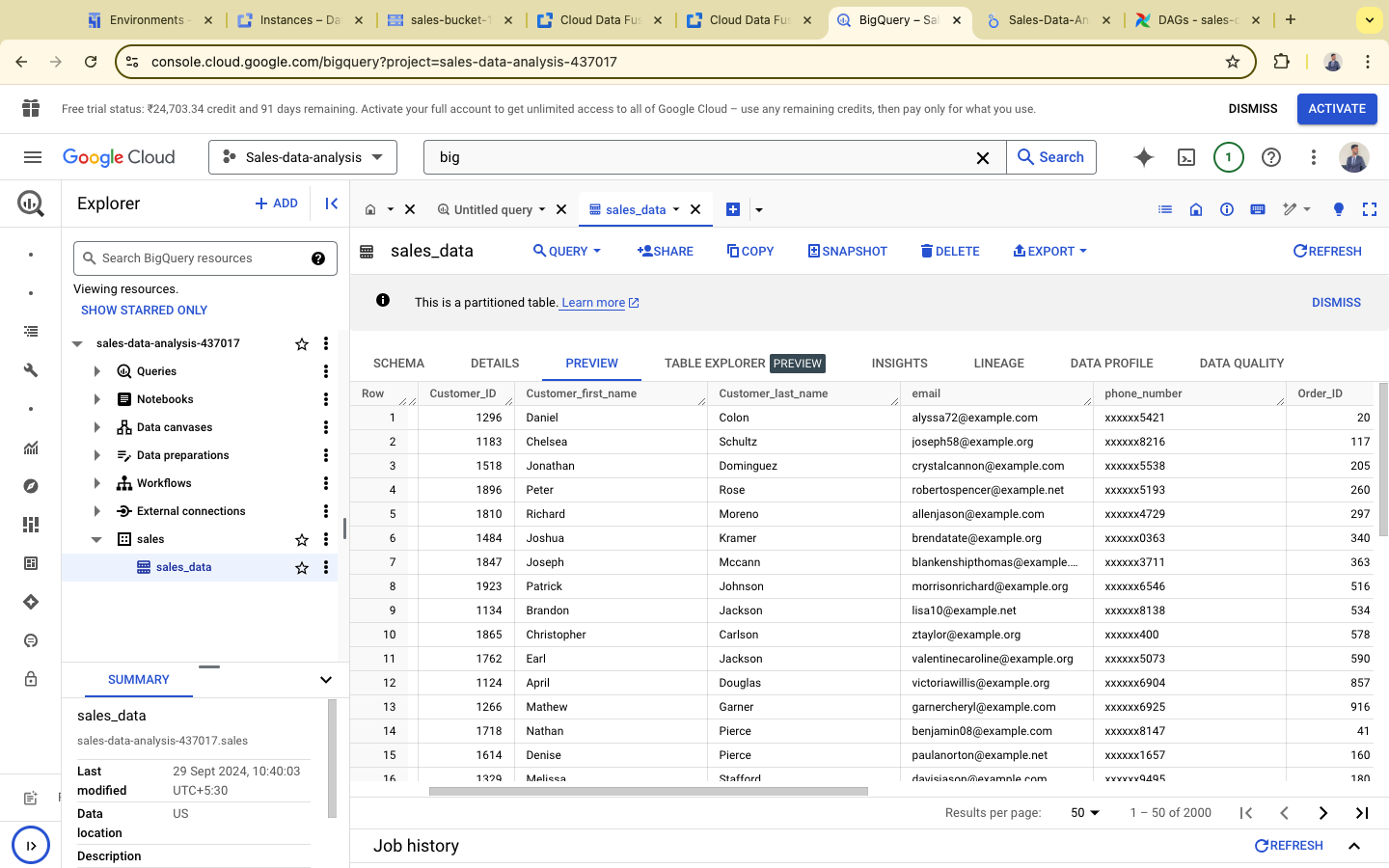
1. After adding the GCS source in **Cloud Data Fusion**, add a **Data Transformation** step.
2. Apply transformations such as:
   * **Masking the phone number** (e.g., replace with XXXX-XXXX).
   * **Hiding personal information** like email addresses or other sensitive data.
3. Use built-in **Data Preparation** transformations or write custom scripts to handle these tasks.

**Step 6: Create a Data Fusion Pipeline**

****

1. Open **Cloud Data Fusion** and navigate to your instance.
2. Select **Studio** to create a new pipeline.
3. Choose **Batch Pipeline**.
4. Set up the **GCS Source**:
   * In the source step, point to the CSV file in the GCS bucket (gs://sales-bucket-1/sales\_data.csv).
   * Ensure you configure the file format correctly (CSV with headers).

**Step 7: Add a BigQuery Sink**

****

1. In your Data Fusion pipeline, add a **BigQuery Sink** as the destination.
2. Select the **BigQuery table** where the transformed data will be stored.
3. Configure the table schema, ensuring that sensitive fields (e.g., masked phone number) are correctly mapped.

**Step 8: Set BigQuery Sink Properties**

1. Set the **location** (e.g., us-central1) and **dataset** for your BigQuery instance.
2. Specify the table name where the data will be stored.
3. Ensure you set the appropriate **write disposition** (e.g., append or overwrite) based on your needs.

**Step 9: Go to BigQuery and Create a Dataset**

1. Navigate to **BigQuery** in the Google Cloud Console.
2. Click **Create Dataset**, and provide the necessary details:
   * Dataset ID (e.g., sales\_data\_analysis).
   * Set the **location** to match your GCS bucket and Data Fusion instance.
3. Create the dataset.

**Step 10: Complete BigQuery Sink Property Configuration**

1. Return to your Data Fusion pipeline and set the dataset you created (sales\_data\_analysis) as the sink for your BigQuery table.
2. Review the field mappings to ensure that all transformations, especially the masked data, are set correctly.

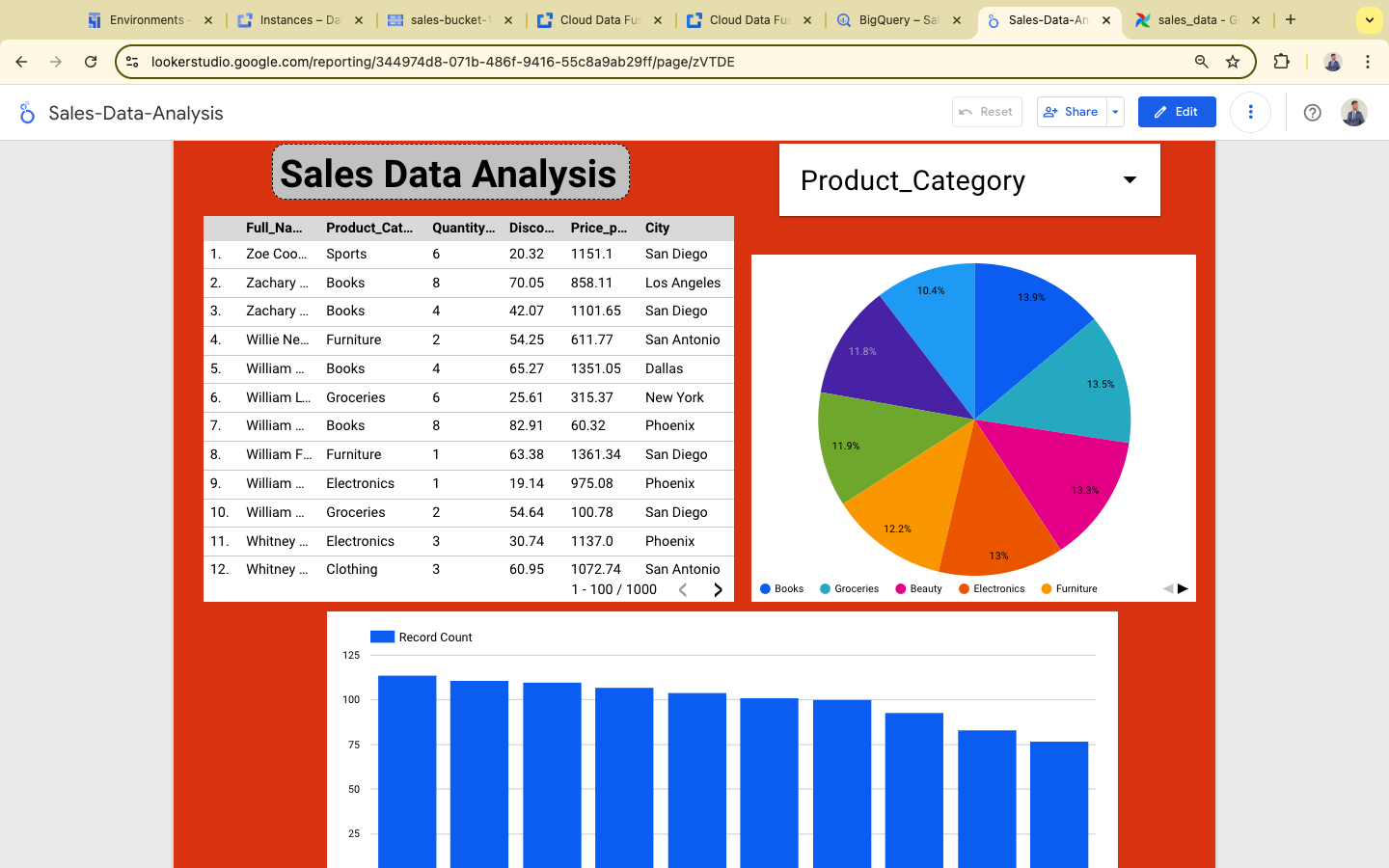
**Step 11: Deploy the Data Fusion Pipeline**

1. Once the pipeline is configured and all transformations are correctly set, click **Deploy** to finalize the pipeline.
2. Review the configuration for any errors or missing elements.

**Step 12: Run the Data Fusion Pipeline**

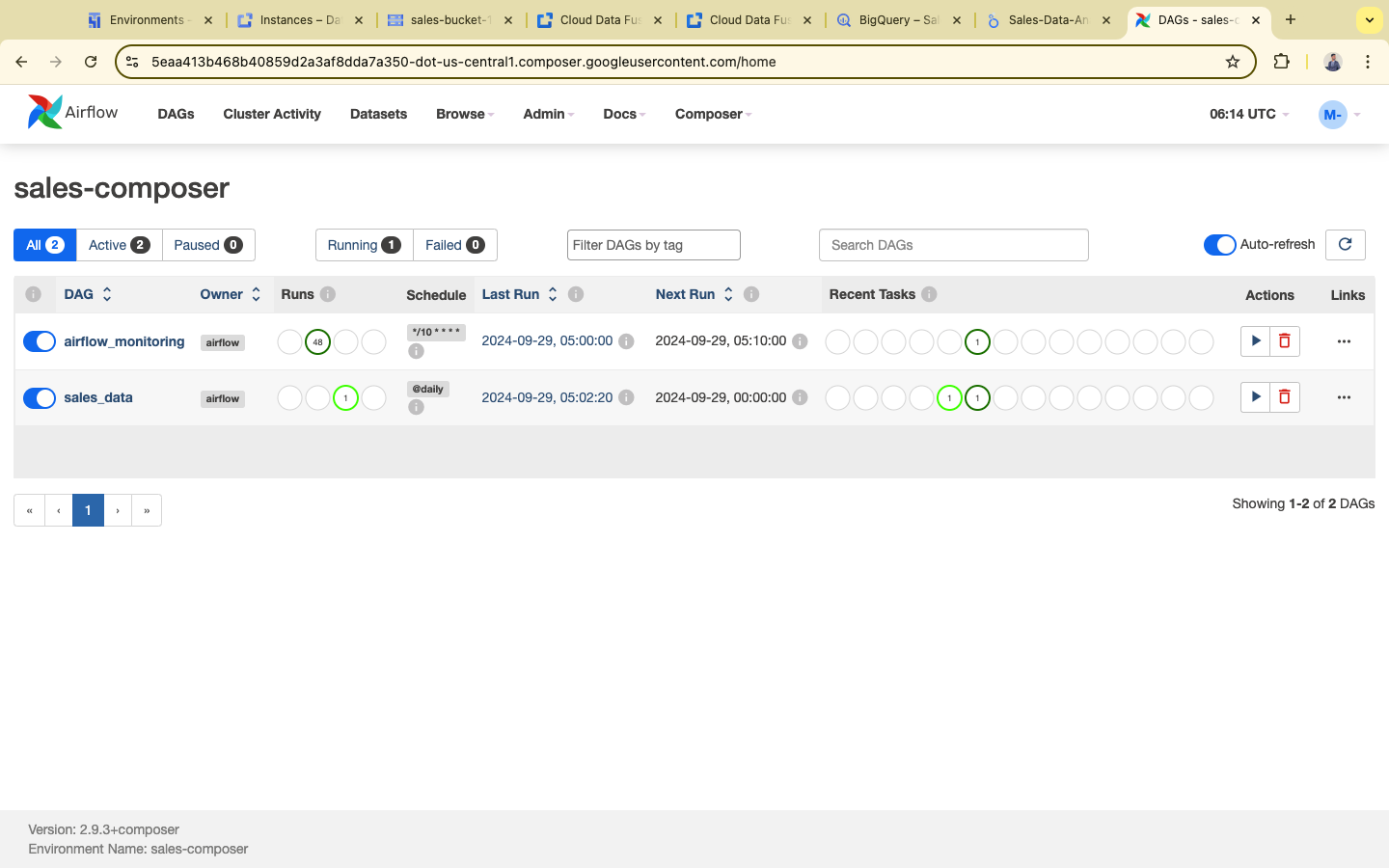
1. After deploying the pipeline, click **Run** to start the batch pipeline.
2. Monitor the job logs for any errors during execution.
3. Once complete, check BigQuery to verify that the transformed data has been loaded successfully.

**Step 13: Visualize the Data in Looker Studio**

****

1. Open **Looker Studio** (formerly Google Data Studio).
2. Create a new report and connect it to the BigQuery dataset (sales\_data\_analysis).
3. Design your dashboard by adding relevant charts and graphs to visualize the sales data.
4. Ensure sensitive data remains hidden or masked as per your transformations.

**Step 14: Automate the Pipeline using Airflow (Composer)**

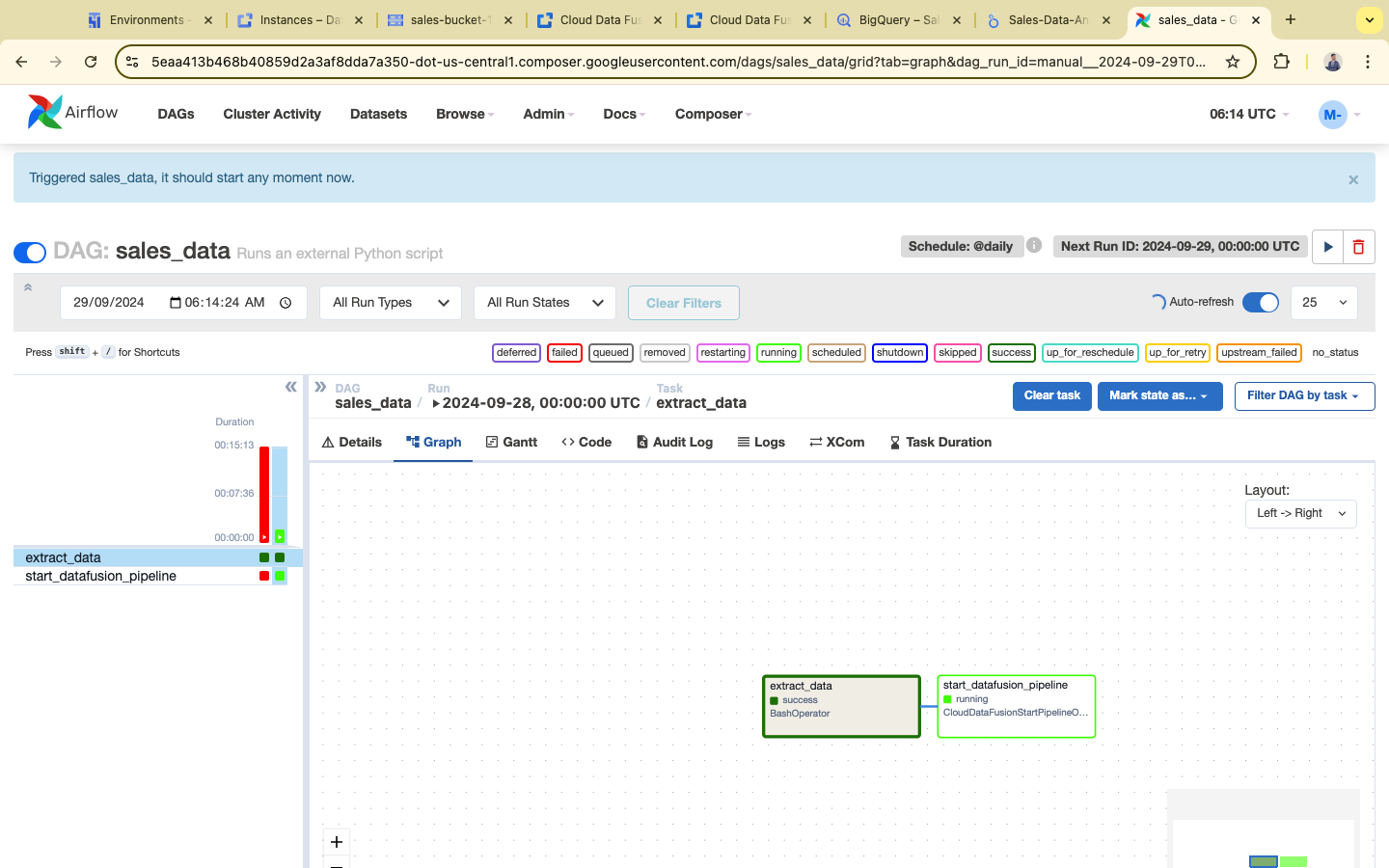
****

1. Open **Cloud Composer** and navigate to your environment.
2. Create a **scripts** folder within your Composer environment by uploading your extract.py file.
3. Upload this file into /home/airflow/gcs/dags/scripts/extract.py.

**Step 15: Write the Airflow DAG**

1. In the same **Cloud Composer** environment, write a DAG (dag.py) that orchestrates the pipeline:
   * Run the Python script (extract.py).
   * Trigger the Data Fusion pipeline (as described in the earlier Airflow code).
2. Save this DAG inside the **DAGs folder** in your Composer environment.

**Step 16: Trigger the Airflow DAG**

****

1. In **Cloud Composer**, navigate to the **DAGs** tab.
2. Find the DAG (sales\_data) you uploaded and trigger it manually to verify that everything works as expected.
3. The DAG will execute daily according to the schedule you set.