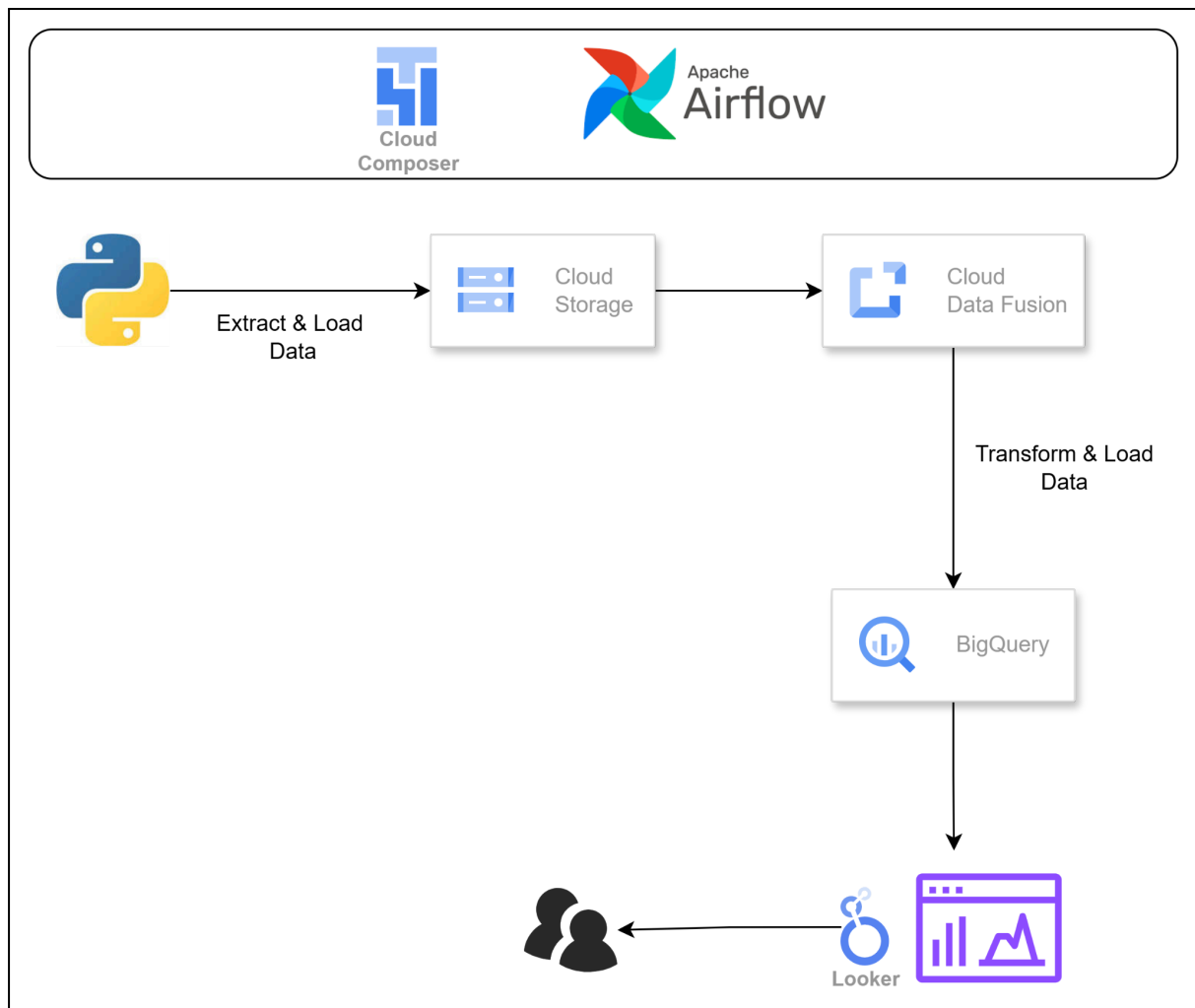


# End-to-End ETL Pipeline + Automation + Analytics



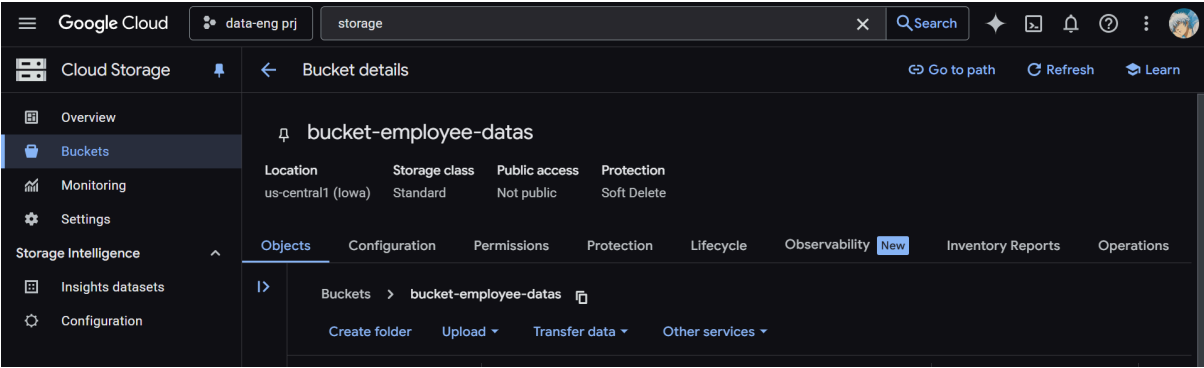
## Stage 1 — Data Generation & Upload to GCS

### 7.1 Description

A Python script uses the **Faker** library to create synthetic employee records (e.g., id, name, email, phone, DOB, ssn, address, department, salary). Generated data is written as a CSV and uploaded to a Google Cloud Storage bucket named **bucket-employee-datas**.

### 7.2 Steps

1. Create a GCS bucket (Console or `gsutil mgs://bucket-employee-datas`).

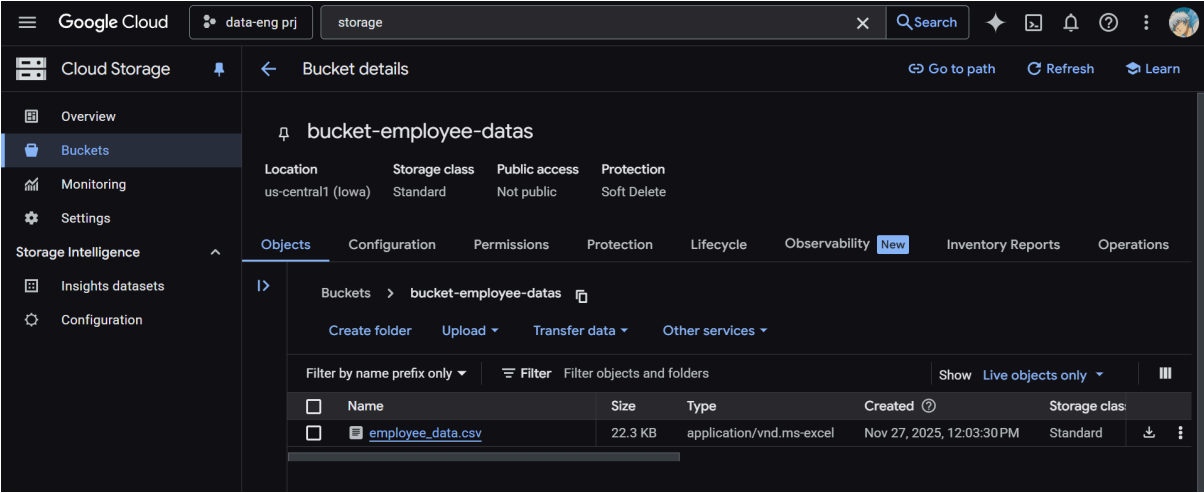


[Snapshot: Created bucket (bucket-employee-datas)]

2. Create Python script (see Appendix) that:

- Generates `NUM_EMPLOYEES` records (e.g., 150).
- Saves as `employee_data.csv`.
- Uploads CSV to GCS using `google.cloud.storage`.

3. Run script locally or from a VM. Confirm CSV uploaded to `gs://bucket-employee-datas/employee_data.csv`.



[Snapshot: CSV in GCS bucket]

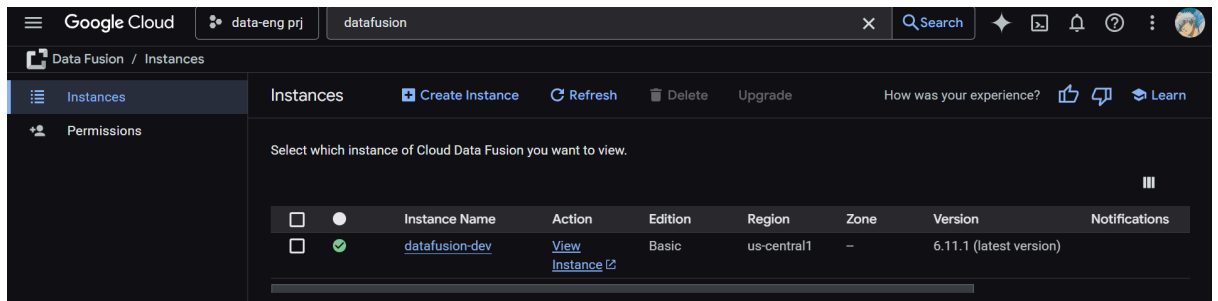
# Stage 2 — Cloud Data Fusion: Wrangler & Transformations

## 8.1 Description

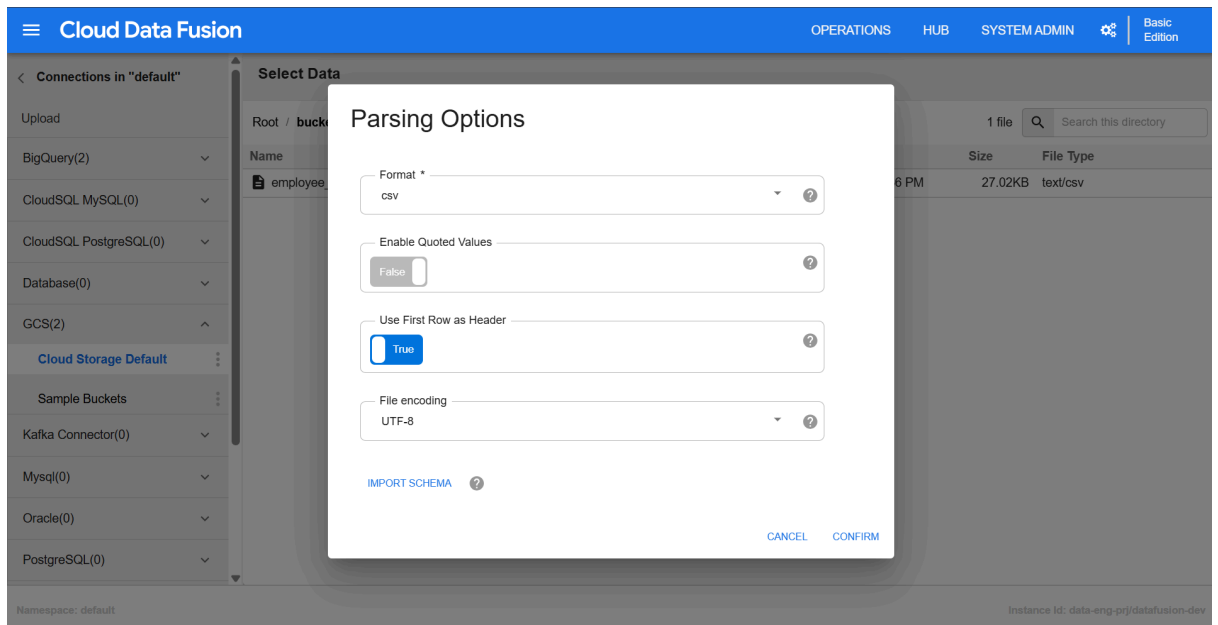
Use Cloud Data Fusion Wrangler to import the CSV, run data cleaning and transformations, then build a Batch pipeline to sync the transformed data to BigQuery.

## 8.2 Steps (Wrangler)

1. Open your Data Fusion instance → **View Instance** → **Wrangler**.



2. Choose the CSV file from Cloud Storage as source.



[Snapshot: Wrangler source selection]

### 3. Transformation operations performed:

- **Join first and last name** to create **full\_name** (select both columns → Join).

The screenshot shows the Cloud Data Fusion Wrangler interface. A table named 'employee\_data.csv' is displayed with columns: employee\_id, first\_name, last\_name, email, phone\_number, and date\_of\_birth. A context menu is open over the 'first\_name' and 'last\_name' columns, and the 'Join two columns' option is selected. A dialog box is shown with 'Comma' as the delimiter and 'full\_names' as the new column name. The 'Join' button is highlighted.

#	Name	Completion
<input type="checkbox"/>	1 employee_id	100%
<input type="checkbox"/>	2 first_name	100%
<input type="checkbox"/>	3 last_name	100%
<input type="checkbox"/>	4 email	100%
<input type="checkbox"/>	5 phone_number	100%
<input type="checkbox"/>	6 date_of_birth	100%
<input type="checkbox"/>	7 ssn	100%
<input type="checkbox"/>	8 address	100%
<input type="checkbox"/>	9 department	100%
<input type="checkbox"/>	10 salary	100%
<input type="checkbox"/>	11 password	100%
<input checked="" type="checkbox"/>	12 full_name	100%

- **Mask salary**: select salary column → mask → choose custom mask → select characters to mask → Apply.

The screenshot shows the Cloud Data Fusion Wrangler interface with the 'Mask' operation applied to the 'salary' column. A 'Mask Data' dialog box is open, showing the selected characters to mask. The 'Apply' button is highlighted.

#	Name	Completion
<input type="checkbox"/>	1 employee_id	100%
<input type="checkbox"/>	2 first_name	100%
<input type="checkbox"/>	3 last_name	100%
<input type="checkbox"/>	4 email	100%
<input type="checkbox"/>	5 phone_number	100%
<input type="checkbox"/>	6 date_of_birth	100%
<input type="checkbox"/>	7 ssn	100%
<input type="checkbox"/>	8 address	100%
<input type="checkbox"/>	9 department	100%
<input type="checkbox"/>	10 salary	100%
<input type="checkbox"/>	11 password	100%
<input type="checkbox"/>	12 full_name	100%

- **Password encoding** (applied similarly as masking/encoding).

- Use the **transform step** side-click to inspect and undo transformations if necessary.

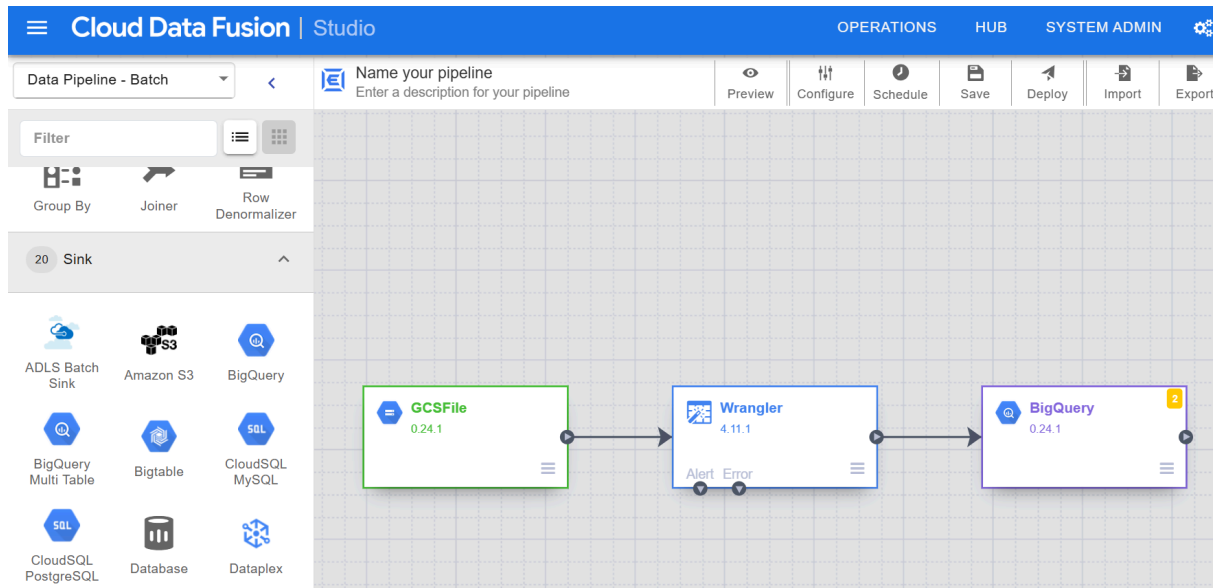
Columns (14)		Transformation steps (4)	
		#	Transformations
		1	merge :first_name :last_name :full_name ,
		2	mask-number :salary xxxxxx
		3	encode base64 :password
		4	encode hex :password

## 8.3 Build & Deploy Pipeline

1. Click **Create Pipeline** → *Batch Pipeline*.

The screenshot shows the Cloud Data Fusion Studio interface. The top navigation bar includes 'Cloud Data Fusion | Studio', 'OPERATIONS', 'HUB', 'SYSTEM ADMIN', and 'Basic Edition'. The main workspace is titled 'Name your pipeline' with a description field. On the left, a 'Filter' section shows '21 Source' options, including ADLS Batch Source, Amazon S3, BigQuery, Bigtable, CloudSQL MySQL, CloudSQL PostgreSQL, Database, Dataplex, Datastore, Excel, FTP (Deprecated), File, Kafka, and Microsoft. The main canvas displays a pipeline with two steps: 'GCSFile 0.24.1' and 'Wrangler 4.11.1'. The Wrangler step has an 'Alert Error' option. The bottom status bar shows 'Namespace: default' and 'Instance id: data-eng-prj/datafusion-dev'.

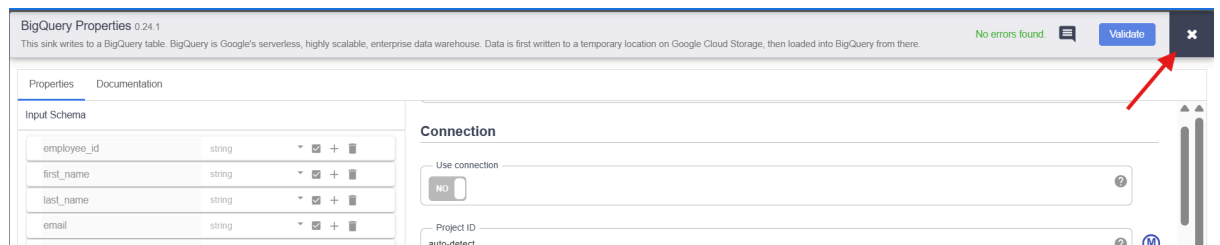
2. Add BigQuery sink (under Sync → BigQuery).



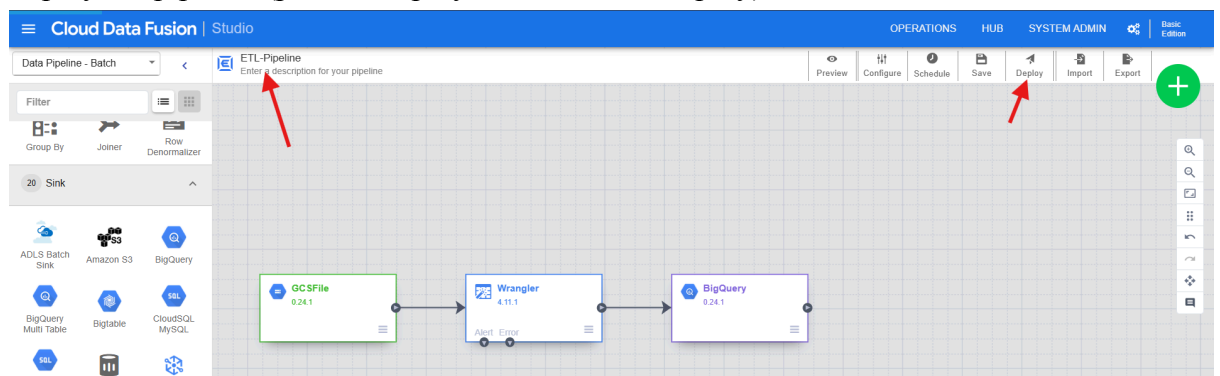
### 3. Configure BigQuery sink properties:

- Dataset Project ID: **data-eng-prj**
- Reference Name: **bq-load**
- Dataset: **employee**
- Table: **emp\_data**

### 4. Validate pipeline (top → Validate). Ensure green checks (no errors).



### 5. Deploy the pipeline (provide deployment name → Deploy).



### 6. Monitor logs (Advanced logs) until pipeline succeeds.

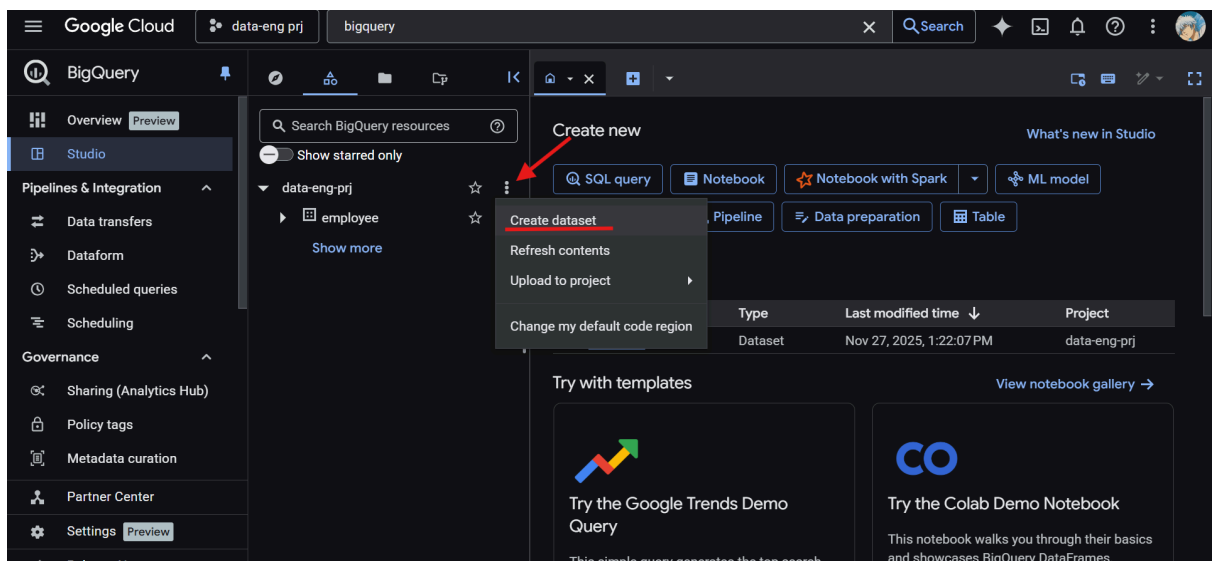
# Stage 3 — BigQuery Setup & Data Load

## 9.1 Description

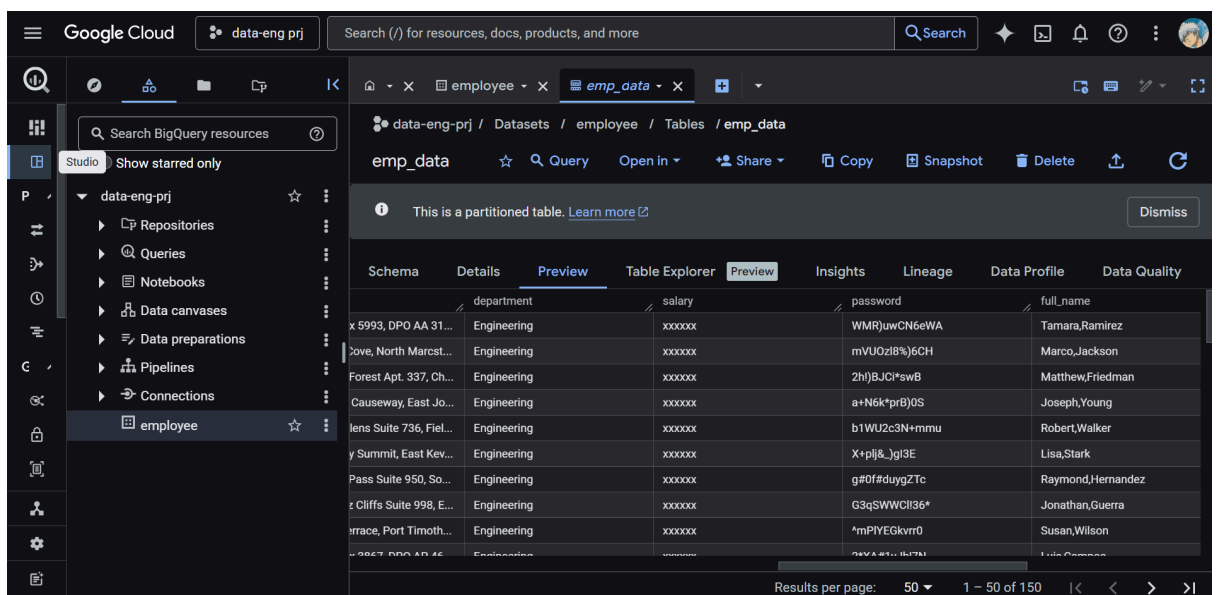
Configure dataset and table in BigQuery then verify the ETL load.

## 9.2 Steps

1. In BigQuery console, create dataset **employee** if not existing.



2. Confirm table **emp\_data** is created by Data Fusion pipeline or create schema manually.
3. After pipeline run, verify **emp\_data** contains the transformed and masked records.



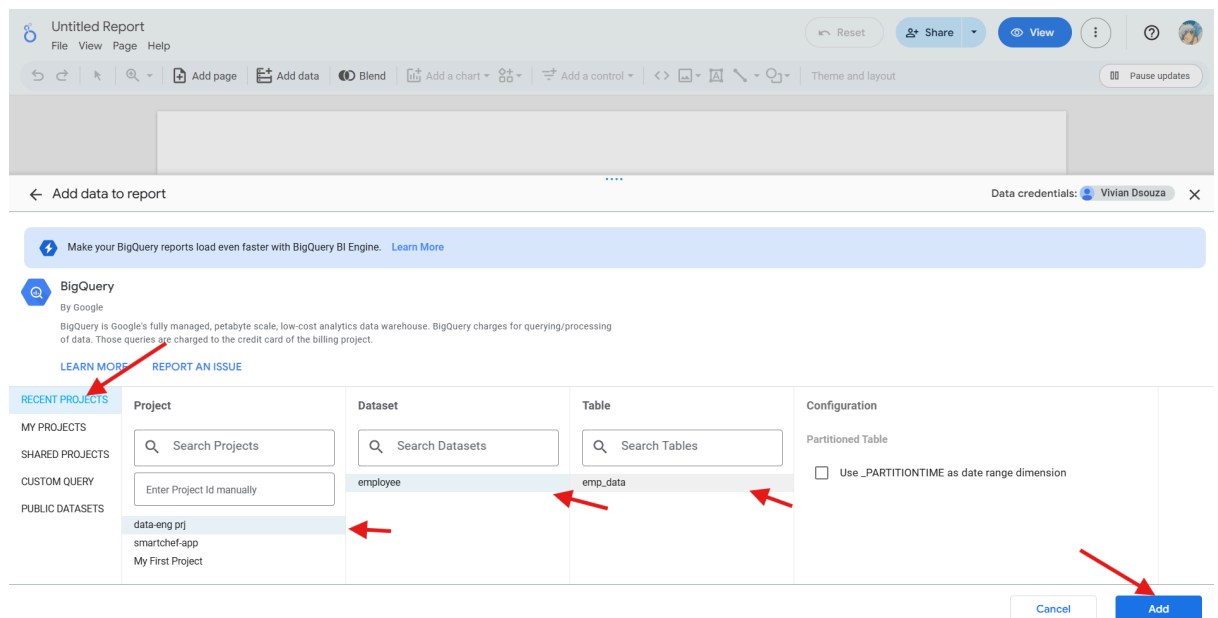
# Stage 4 — Visualization (Looker Studio)

## 10.1 Description

Connect BigQuery dataset to Looker Studio and create a dashboard to visualize employee data.

## 10.2 Steps

1. Open Looker Studio (Data Studio) → Create → Blank Report.
2. Add Data → BigQuery connector → Select recent project **data-eng-prj**.
3. Choose dataset **employee** → table **emp\_data**.





4. Build visualization: tables, charts, and KPIs (e.g., headcount by department, salary distribution—note masked values).

The screenshot shows the 'Untitled Report' interface. The main canvas displays a table with employee data. The 'Table properties' panel on the right is open, showing the 'Setup' tab. The 'Data source' is set to 'emp\_data'. The 'Dimension' section lists fields: employee\_id, full\_name, email, department, and salary. The 'Metric' section is empty. The 'Filter' section is also empty. The 'Record Count' is 123.

	employee_id	full_name	email	department
1.	492107	Lisa Stark	stephanee40@exam...	Engineering
2.	822640	Susan Wilson	sotodaniel@exampl...	Engineering
3.	118439	Luis Campos	amy52@example.org	Engineering
4.	647276	Mark Nelson	uncoors@example.net	Engineering
5.	885262	Janet Strong	christophermth@e...	Engineering
6.	333811	Michael Parrish	laurarivera@exampl...	Engineering
7.	21185	Thomas Carlson	kellywilliam@exam...	Engineering
8.	14634	Steven Johnson	uwhite@example.net	Engineering
9.	28925	Cristina Stout	jackgross@example...	Engineering
10.	736977	Jacqueline Alexander	angelaharris@exam...	Engineering
11.	541528	Jack Greene	dayam@example.net	Engineering
12.	765032	Michael Martinez	tamarahill@example...	Engineering
13.	358276	Gabriel Payne	stephanetorres@ex...	Engineering
14.	561389	Sarah McDaniel	lopezjason@exampl...	Finance
15.	649654	Logan Williamson	adam68@example.net	Finance
16.	757340	Andrea Thomas	michaelanderson@...	Finance
17.	618384	Brian Gonzalez	jennifer02@exampl...	Finance
18.	895	Daniel Gordon	eperez@example.net	Finance
19.	282837	Thomas Brewer	isa53@example.org	Finance
20.	556801	Joseph Henson	eric05@example.com	Finance
21.	323691	Joseph Martinez	davidlores@exampl...	Finance
22.	926413	Michele Hall	brianmth@example...	Finance
23.	469100	Steven Moore	johnfisher@example...	Finance

5. Arrange dashboard visuals and add titles, filters, and date controls as needed.

The screenshot shows the 'Employee Dashboard' in the 'Untitled Report' interface. The dashboard has a title 'Employee Dashboard' and a 'Dropdown Control' for 'department'. The main content area contains a table of employee data, a bar chart titled 'Full Name' showing counts for each department, and a pie chart showing the percentage distribution of employees by department. The 'Data' panel on the right is open, showing the 'emp\_data' source and various fields. The 'Record Count' is 123.

employee_id	full_name	email	department
1	Lisa Stark	stephanee40@example.net	Engineering
2	Susan Wilson	sotodaniel@example...	Engineering
3	Luis Campos	amy52@example.org	Engineering
4	Mark Nelson	uncoors@example.net	Engineering
5	Janet Strong	christophermth@e...	Engineering
6	Michael Parrish	laurarivera@example...	Engineering
7	Thomas Carlson	kellywilliam@exam...	Engineering
8	Steven Johnson	uwhite@example.net	Engineering
9	Cristina Stout	jackgross@example...	Engineering
10	Jacqueline Alexander	angelaharris@exam...	Engineering
11	Jack Greene	dayam@example.net	Engineering
12	Michael Martinez	tamarahill@example...	Engineering
13	Gabriel Payne	stephanetorres@ex...	Engineering
14	Sarah McDaniel	lopezjason@example...	Finance
15	Logan Williamson	adam68@example.net	Finance
16	Andrea Thomas	michaelanderson@...	Finance
17	Brian Gonzalez	jennifer02@example...	Finance
18	Daniel Gordon	eperez@example.net	Finance
19	Thomas Brewer	isa53@example.org	Finance
20	Joseph Henson	eric05@example.com	Finance

Bar Chart: Full Name

Department	Count
Finance	10
Sales	10
IT Support	10
HR	10
Engineering	10
Marketing	10

Pie Chart: Department Distribution

Department	Percentage
Finance	16.8%
Sales	16.8%
IT Support	16.8%
HR	16.8%
Engineering	16.8%
Marketing	16.8%

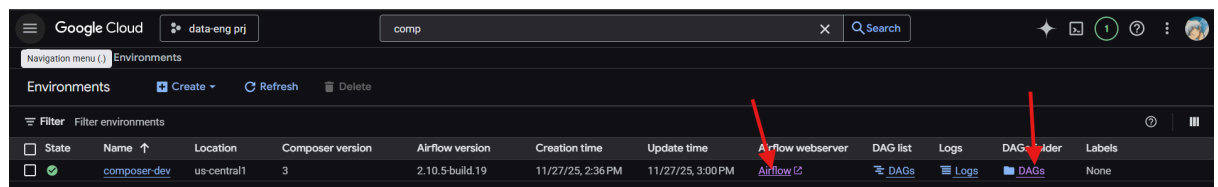
# Stage 5 — Automation with Cloud Composer (Airflow)

## 11.1 Description

Automate the ETL pipeline by creating a Cloud Composer environment and an Airflow DAG that runs the extraction script and triggers the Cloud Data Fusion pipeline.

## 11.2 Composer Setup & IAM Fix

1. Create Cloud Composer environment.



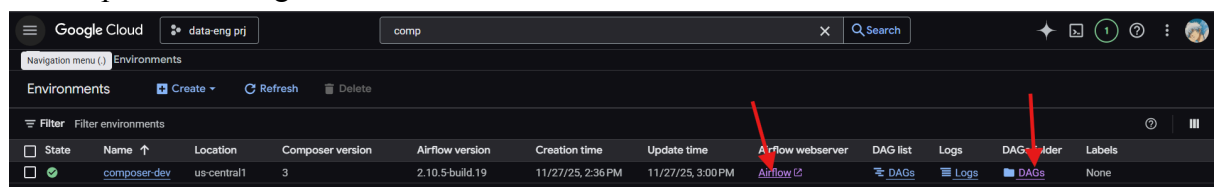
If Composer shows an error about missing IAM roles (example message):

The issue may be caused by missing IAM roles in the following Service Accounts: [service-1047842664348@cloudcomposer-accounts.iam.gserviceaccount.com](mailto:service-1047842664348@cloudcomposer-accounts.iam.gserviceaccount.com) is missing role `roles/composer.ServiceAgentV2Ext`

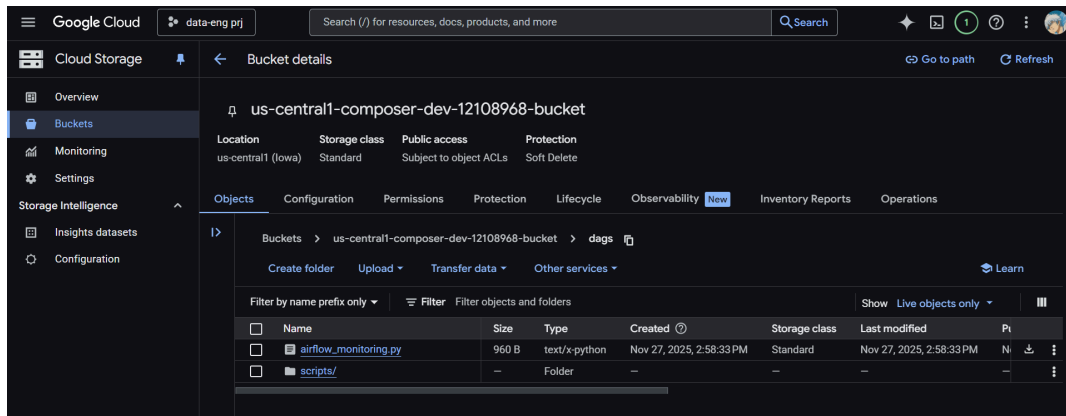
2. **Fix:** Go to IAM & Admin → locate the service account → Edit principal → Add role **Cloud Composer Service Agent v2 Extensions** → Save.

## 11.3 Prepare DAG & Scripts

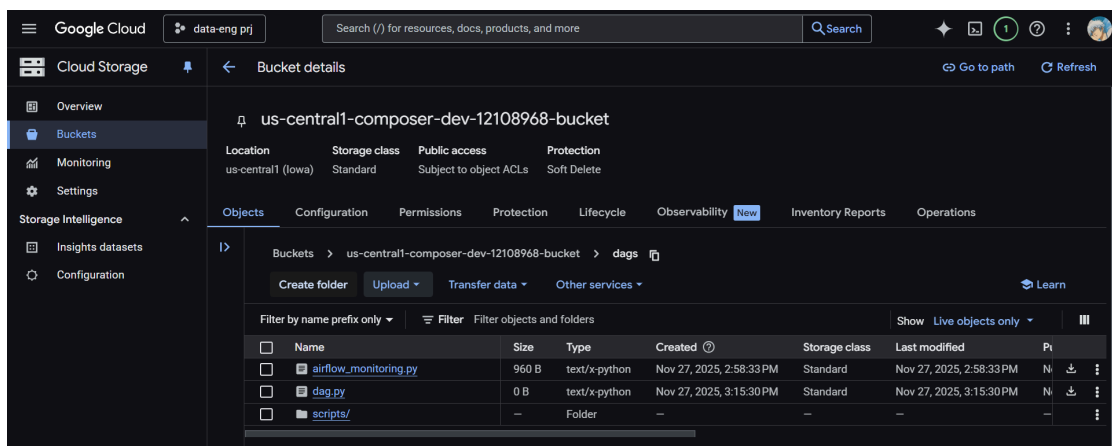
1. In Composer → navigate to the DAGs bucket.



2. Create `scripts/` folder and upload `extract.py` (the extractor script used earlier).

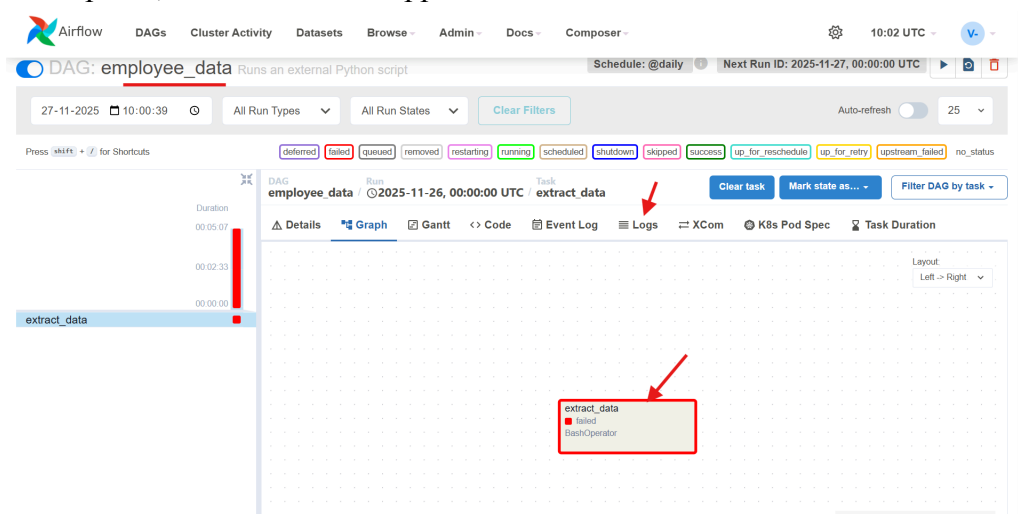


3. Upload `dag.py` (see Appendix for code). The DAG does:



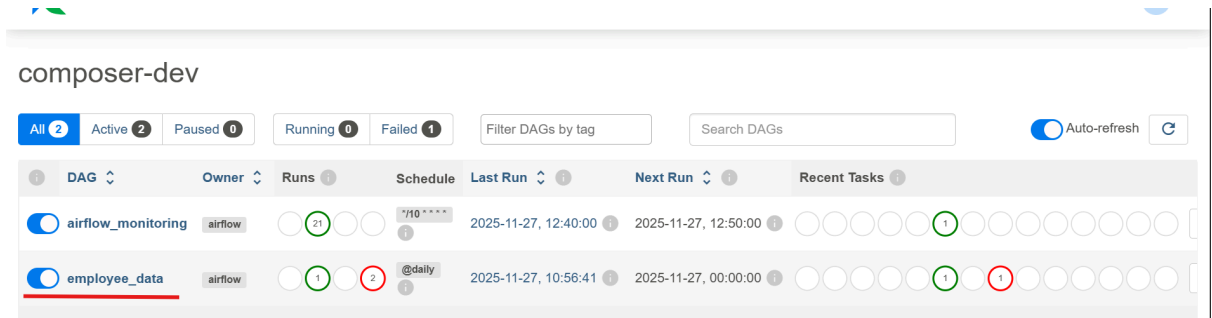
- Task 1: BashOperator → run `extract.py`.
- Task 2: CloudDataFusionStartPipelineOperator → start Data Fusion pipeline (e.g., `etl-pipeline`).
- Set DAG schedule to `@daily` and catchup = False.

4. After upload, the DAG should appear in the Airflow UI.

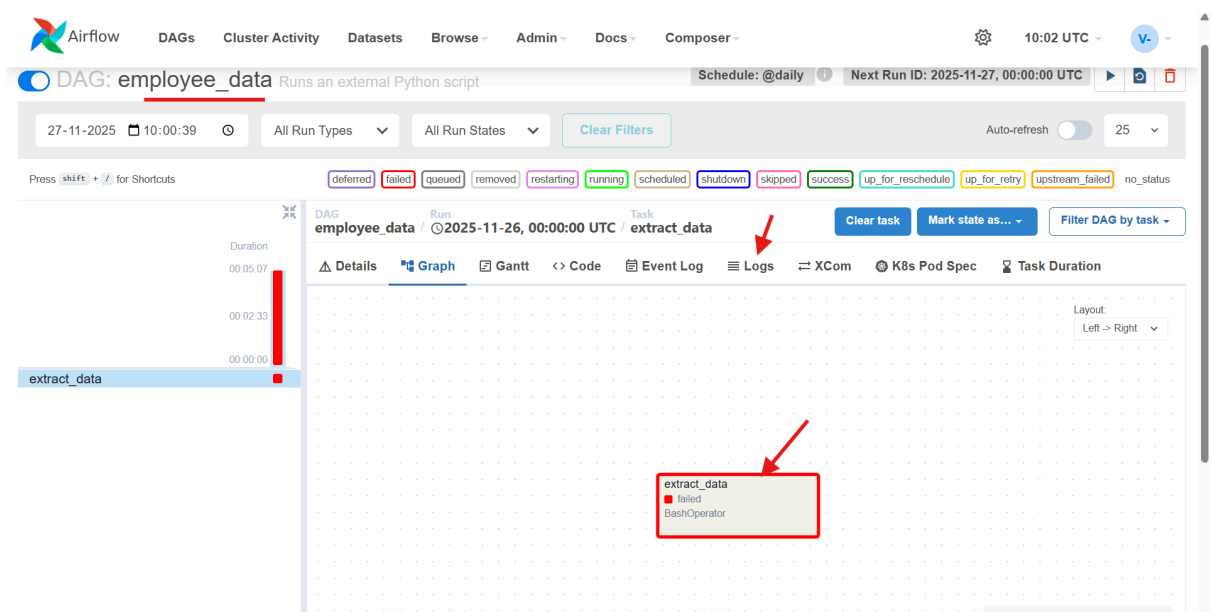


## 11.4 Running & Observing DAG

1. Trigger the DAG manually or wait for scheduled run.



2. If a task fails (yellow/red), click into the task → **View logs** to inspect stack trace and error.



## Composer Error Handling & Fixes

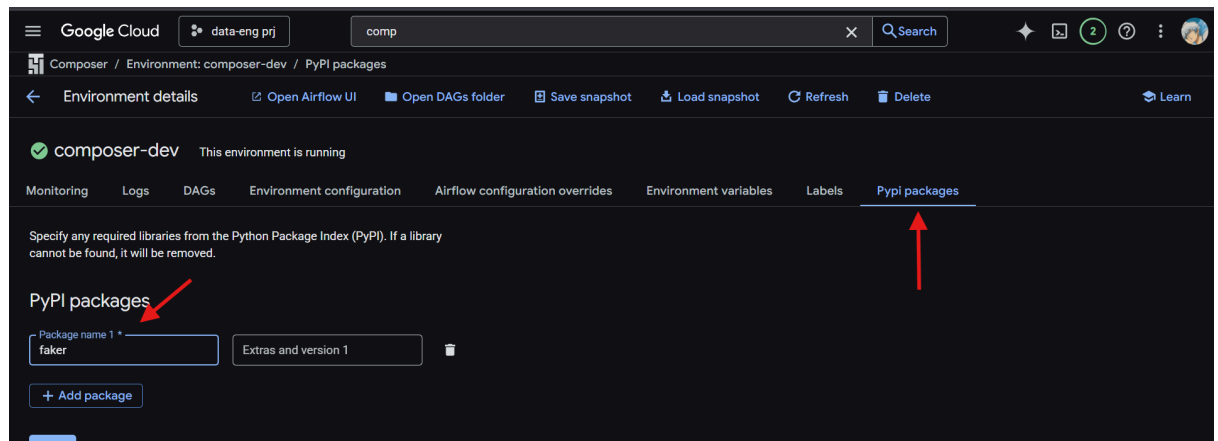
During the execution of the Airflow DAG inside Cloud Composer, one of the tasks failed. To identify the root cause of the failure, we first opened the Airflow UI and navigated to the task logs:

Based on the error message in the logs, the issue was related to missing Python dependencies required by the script. To fix this:

---

## Fix 1 — Install Missing Python Packages in Cloud Composer

1. Go to **Cloud Composer** in the GCP Console.
2. Open your Composer **instance**.
3. Navigate to the **PyPI Packages** tab.
4. Click **Add Package** and install the missing package(s).



5. Save and wait for the environment to update.  
(Place snapshot here)

---

## Fix 2 — Clear Incorrect Files from Composer DAG Bucket

While Composer updates, we also clean up an issue in the DAGs bucket:

1. Go to **Cloud Storage** → open the DAGs bucket for the Composer environment.
2. Navigate to:  
**Bucket** → **bucket-employee-datas**
3. Delete **only the incorrectly created table file** inside this folder.
4. Do *not* delete the entire **bucket-employee-datas** bucket—only the problematic file.

---

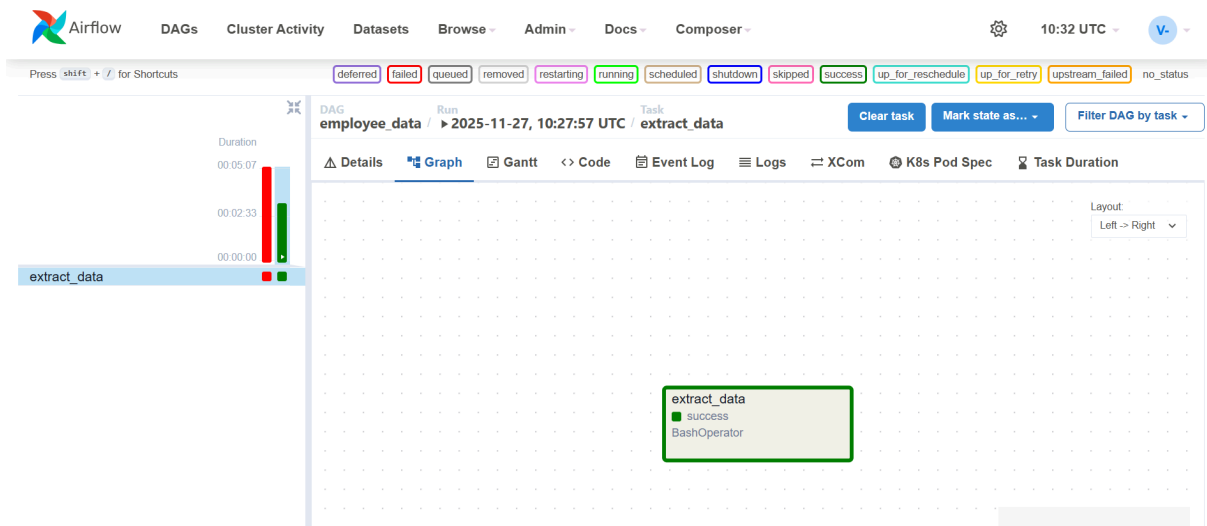
## Re-running the DAG

- Go back to Airflow.

- Trigger the DAG again.

This time, the DAG runs successfully:

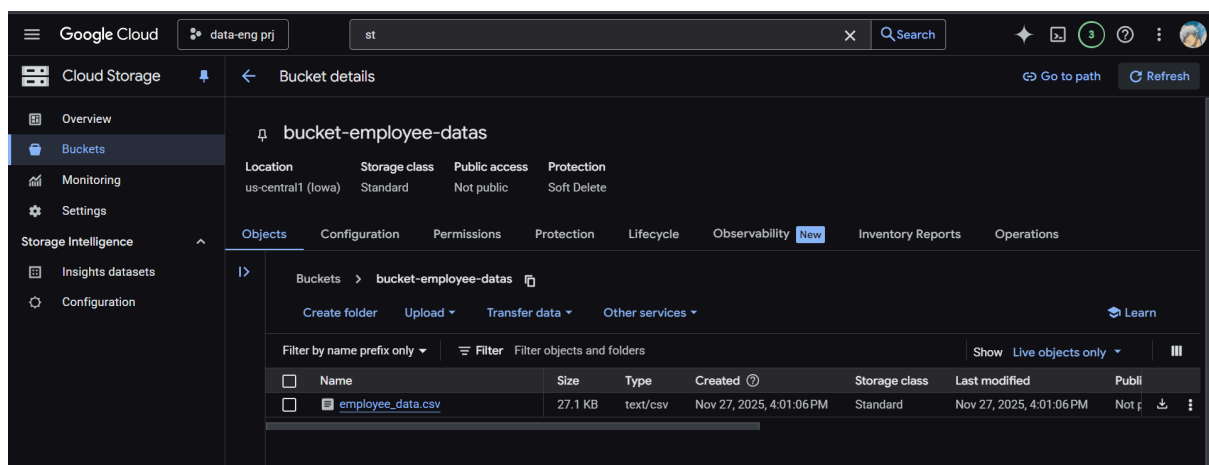
“When we rerun the Airflow DAG we get:”



## Verification — Data Successfully Loaded

After the DAG executes, we confirm the new data has been successfully generated and uploaded:

“And now we see the data is loaded into the bucket.”



# Starting the Data Fusion Pipeline from Airflow

To trigger the Cloud Data Fusion pipeline directly from Airflow, we reference the official Airflow documentation:

<https://airflow.apache.org/docs/apache-airflow-providers-google/stable/operators/cloud/datafusion.html>

We use the operator below in the DAG:

```
start_pipeline = CloudDataFusionStartPipelineOperator(  
    location=LOCATION,  
    pipeline_name=PIPELINE_NAME,  
    instance_name=INSTANCE_NAME,  
    pipeline_timeout=1000,  
    task_id="start_pipeline",  
)
```

This operator ensures that once the extract script completes, Airflow triggers the Data Fusion ETL pipeline automatically.

# Final End-to-End Workflow Verification

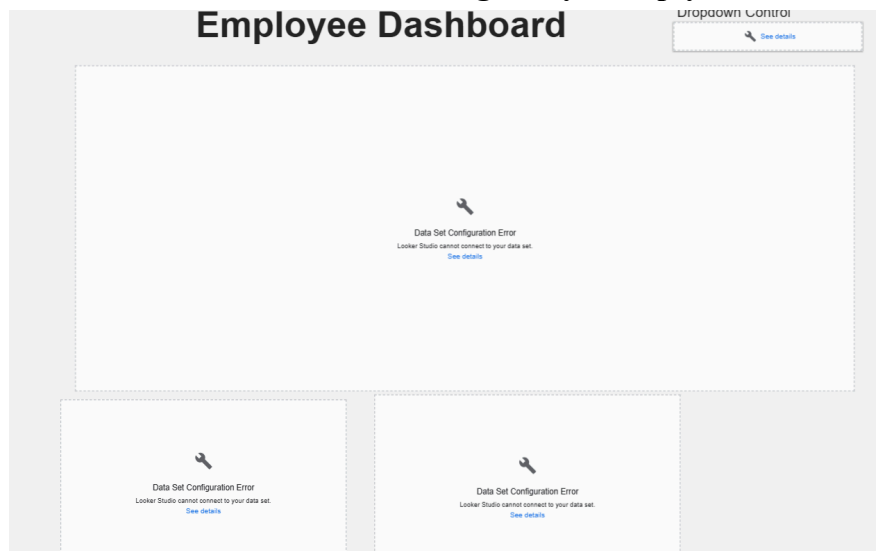
To fully validate the automated pipeline, we perform a complete end-to-end test by clearing the existing data, triggering the DAG, and verifying updates in each system: BigQuery → GCS → Data Fusion → Looker Studio.

## 1. Clearing the BigQuery Table

Before testing the automation flow, we delete the existing table contents in BigQuery:

- Delete the **emp\_data** table (or truncate it).
- This ensures that Looker Studio will show **no data** when refreshed.

**“We will then delete the emp\_data from BigQuery, and when we check Looker Studio, it shows no data since BigQuery is empty.”**



---

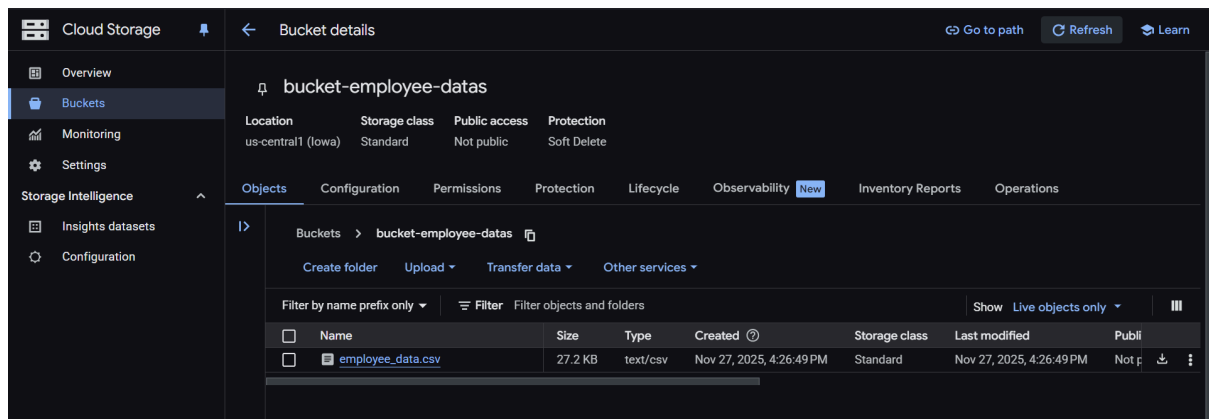
## 2. Running the Airflow DAG

Next, we trigger the Airflow DAG from Cloud Composer:

- Airflow runs the extract script.
- New synthetic employee data is generated.
- A fresh CSV file is uploaded into the GCS bucket.



**“We then run the Airflow DAG, which creates the data in the bucket.”**



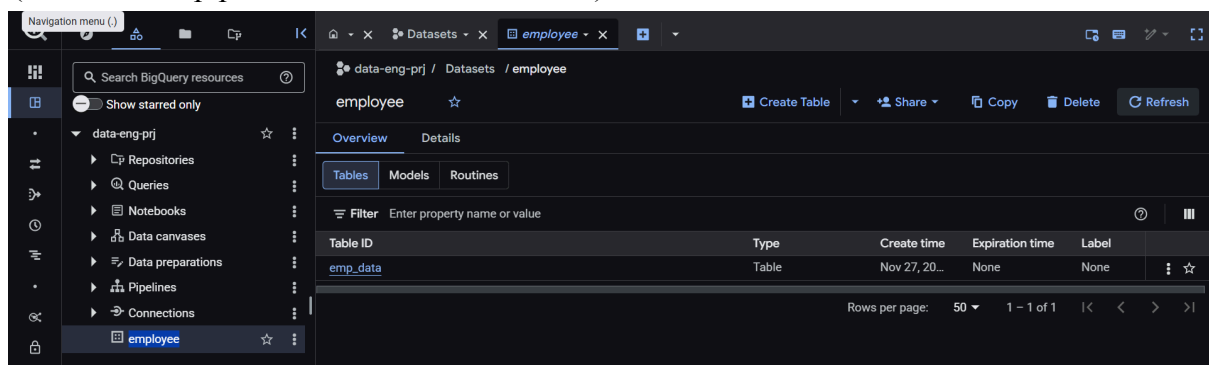
### 3. Cloud Data Fusion Pipeline Execution

After the extract script completes, Airflow triggers the Data Fusion pipeline:

- Data Fusion retrieves the CSV from the GCS bucket.
- Transformations (join, masking, encoding) are applied.
- The processed data is pushed into BigQuery.

**“Then we check Data Fusion where the pipeline is triggered, and the transformed data is loaded into BigQuery.”**

(Data Fusion pipeline run status = succeeded)



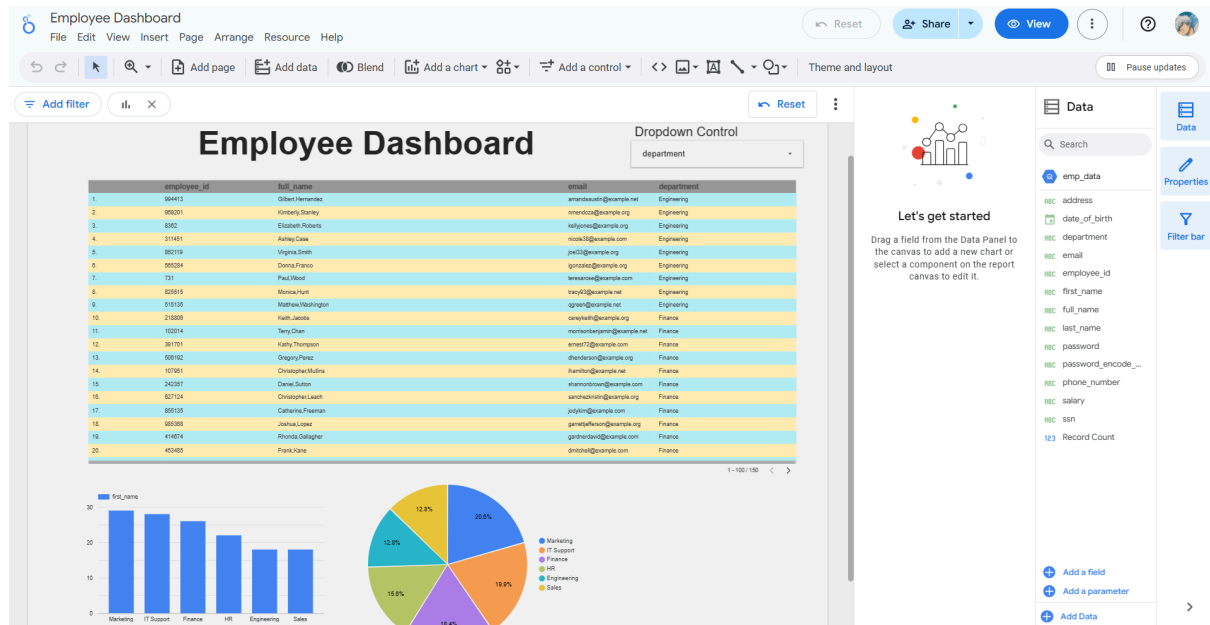
### 4. Refreshing Looker Studio Dashboard

Finally, we move to Looker Studio and refresh the data source:

- Looker Studio re-pulls data from BigQuery.

- Since the pipeline successfully loaded the new records, the dashboard instantly updates.

**“Then we go to Looker Studio and refresh, and now we get the updated dashboard with the newly loaded data.”**



# Code & Commands

## 16.1 Synthetic Data Generator & GCS Upload (**employee\_generator.py**)

```
from faker import Faker
import csv
import random
from google.cloud import storage
import os
import hashlib # Only needed if using hashed passwords option

# -----
# CONFIGURATION
# -----

NUM_EMPLOYEES = 150
OUTPUT_FILE = "employee_data.csv"
GCS_BUCKET_NAME = "bucket-employee-datas"
GCS_DESTINATION_BLOB = "employee_data.csv"

fake = Faker()

DEPARTMENTS = ["HR", "Finance", "Engineering", "Sales", "Marketing", "IT Support"]

def generate_employee():
    # Generate a strong random password
    password = fake.password(
        length=12,
        special_chars=True,
        digits=True,
        upper_case=True,
        lower_case=True
    )

    # Clean address to avoid CSV / BigQuery issues
    address = fake.address().replace('\n', ',').replace('"', '')

    return {
        "employee_id": fake.unique.random_number(digits=6),
        "first_name": fake.first_name(),
        "last_name": fake.last_name(),
        "email": fake.email(),
```

```

        "phone_number": fake.phone_number(),
        "date_of_birth": fake.date_of_birth(minimum_age=20,
maximum_age=65).isoformat(),
        "ssn": fake.ssn(),
        "address": address,
        "department": random.choice(DEPARTMENTS),
        "salary": random.randint(35000, 150000),
        "password": password
    }

def generate_employee_data(num_records):
    return [generate_employee() for _ in range(num_records)]

def save_to_csv(data, filename):
    with open(filename, mode="w", newline="", encoding="utf-8") as file:
        writer = csv.DictWriter(
            file,
            fieldnames=data[0].keys(),
            quoting=csv.QUOTE_ALL    # Critical fix for cleaner parsing
        )
        writer.writeheader()
        writer.writerows(data)

# -----
# UPLOAD TO GCS
# -----

def upload_to_gcs(bucket_name, source_file, destination_blob):
    client = storage.Client()
    bucket = client.get_bucket(bucket_name)
    blob = bucket.blob(destination_blob)
    blob.upload_from_filename(source_file)
    print(f"File uploaded to gs://{bucket_name}/{destination_blob}")

# -----
# MAIN EXECUTION
# -----

if __name__ == "__main__":
    employees = generate_employee_data(NUM_EMPLOYEES)
    save_to_csv(employees, OUTPUT_FILE)
    print(f"Generated {NUM_EMPLOYEES} employee records")
    print(f"CSV saved locally as: {OUTPUT_FILE}")

    upload_to_gcs(GCS_BUCKET_NAME, OUTPUT_FILE, GCS_DESTINATION_BLOB)

```

## 16.2 Airflow DAG (**dag.py**)

```
from datetime import datetime, timedelta
from airflow import DAG
from airflow.operators.bash_operator import BashOperator
from airflow.utils.dates import days_ago
from airflow.providers.google.cloud.operators.datafusion import
CloudDataFusionStartPipelineOperator

default_args = {
    'owner': 'airflow',
    'start_date': datetime(2023, 12, 18),
    'depends_on_past': False,
    'email': ['vishal.bulbule@techtrapture.com'],
    'email_on_failure': False,
    'email_on_retry': False,
    'retries': 1,
    'retry_delay': timedelta(minutes=5),
}

dag = DAG(
    'employee_data',
    default_args=default_args,
    description='Runs an external Python script and triggers Data Fusion
ETL',
    schedule_interval='@daily',
    catchup=False
)

with dag:
    # Task 1: Run extraction script to generate & upload CSV
    run_script_task = BashOperator(
        task_id='extract_data',
        bash_command='python /home/airflow/gcs/dags/scripts/extract.py',
    )

    # Task 2: Trigger Cloud Data Fusion pipeline
    start_pipeline = CloudDataFusionStartPipelineOperator(
        location="us-central1",
        pipeline_name="ETL-Pipeline",
        instance_name="datafusion-dev",
        task_id="start_datafusion_pipeline",
    )

    # DAG Task Order
    run_script_task >> start_pipeline
```