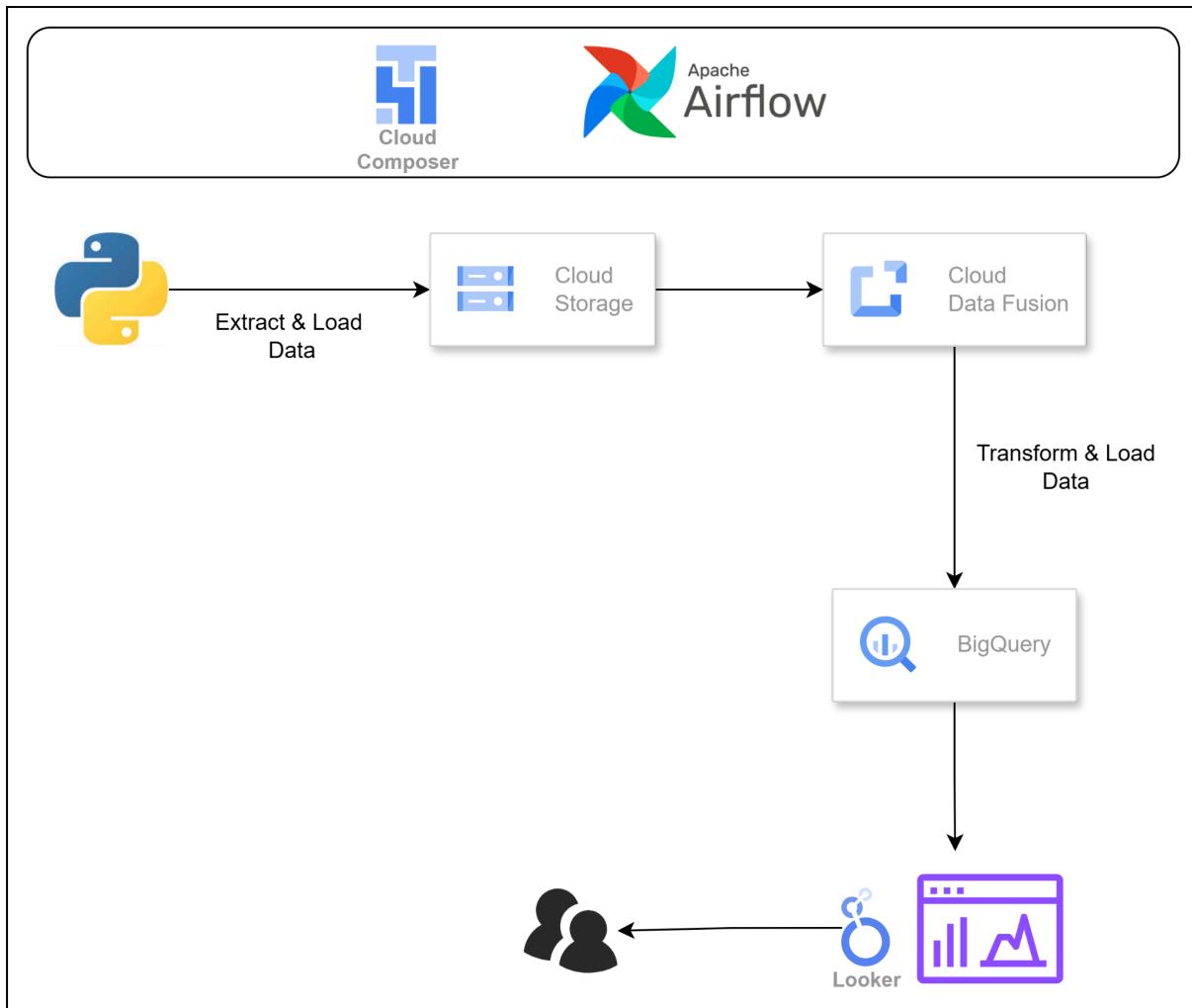


# End-to-End ETL Pipeline + Automation + Analytics



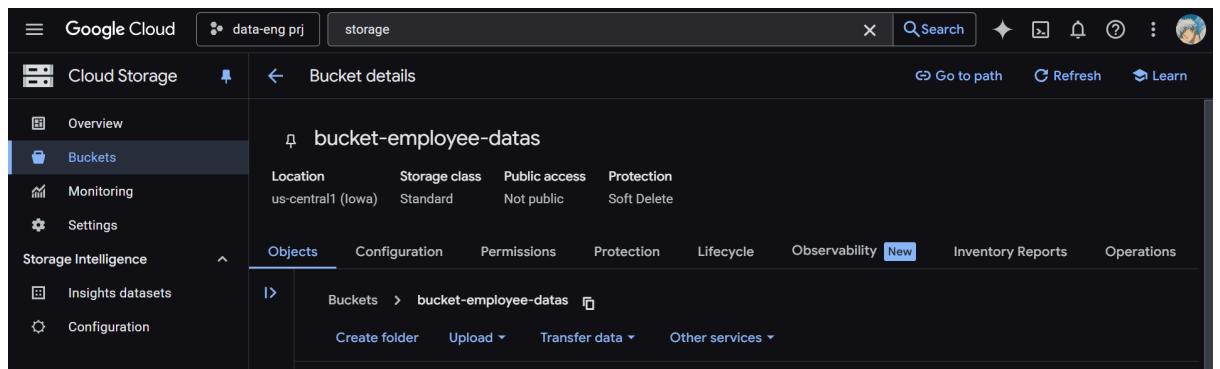
## Stage 1 — Data Generation & Upload to GCS

### 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`).

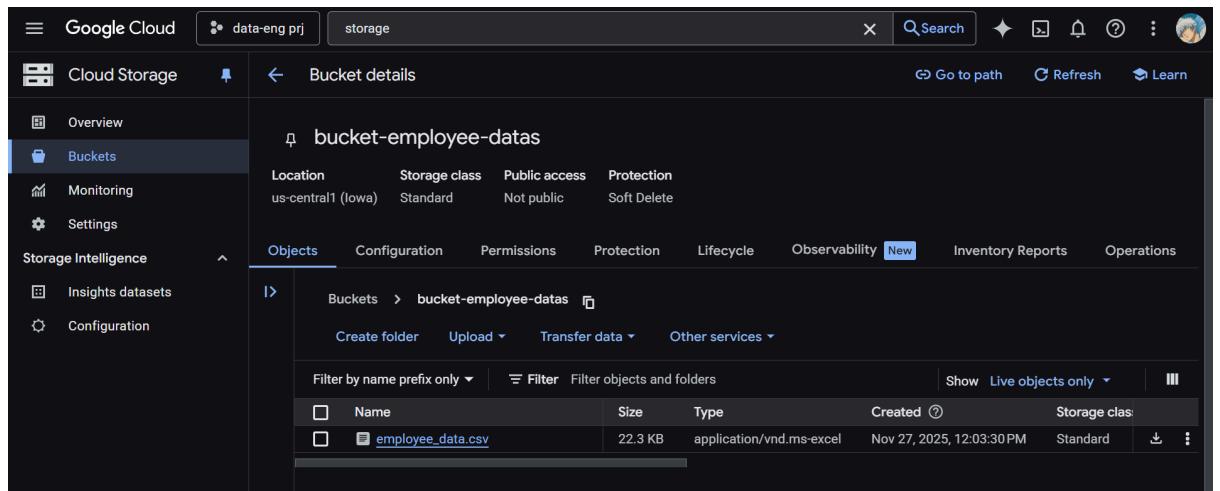


[Snapshot: Created bucket (bucket-employee-datas)]

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]

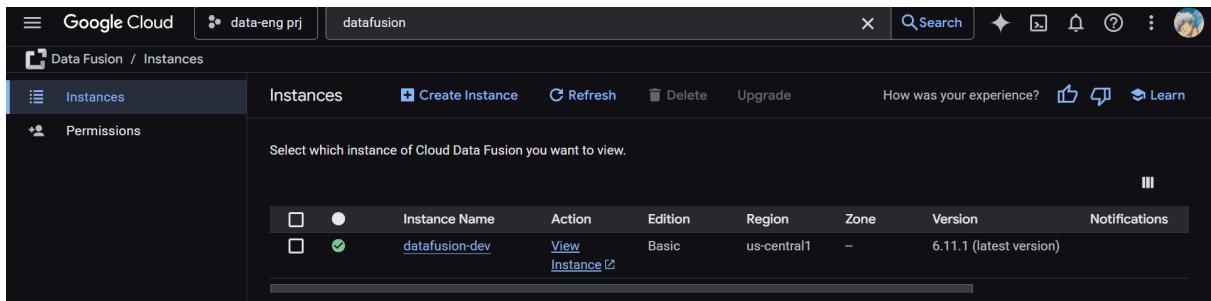
# Stage 2 — Cloud Data Fusion: Wrangler & Transformations

## 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.

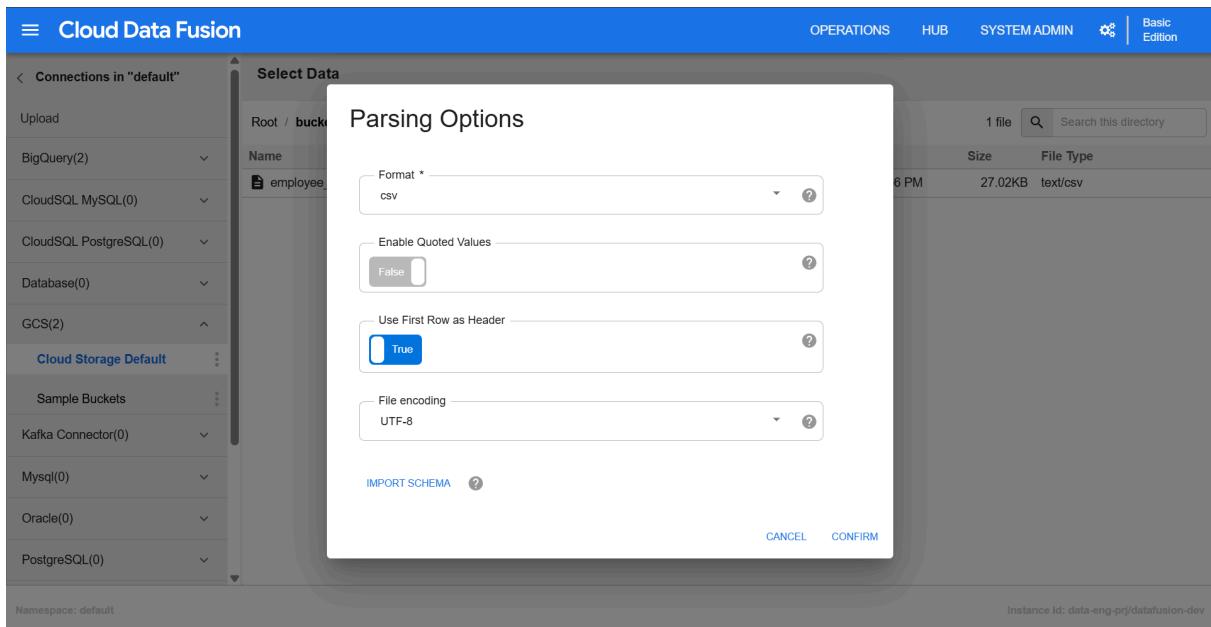
## 8.2 Steps (Wrangler)

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



The screenshot shows the Google Cloud Data Fusion Instances page. The 'Wrangler' tab is selected in the top navigation bar. On the left, there's a sidebar with 'Instances' and 'Permissions'. The main area displays a table of instances. One instance, 'datafusion-dev', is selected and highlighted with a green checkmark. The table columns include 'Instance Name', 'Action', 'Edition', 'Region', 'Zone', 'Version', and 'Notifications'. Below the table, there's a message: 'Select which instance of Cloud Data Fusion you want to view.'

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



The screenshot shows the Cloud Data Fusion Wrangler interface. A 'Parsing Options' dialog box is open over a list of connections. The dialog contains settings for importing a CSV file: 'Format' set to 'csv', 'Enable Quoted Values' set to 'False', 'Use First Row as Header' set to 'True', and 'File encoding' set to 'UTF-8'. At the bottom of the dialog are 'IMPORT SCHEMA' and 'CANCEL' buttons, with a 'CONFIRM' button visible at the very bottom right. The background shows a list of connections including 'BigQuery(2)', 'CloudSQL MySQL(0)', 'CloudSQL PostgreSQL(0)', 'Database(0)', 'GCS(2)', and 'Cloud Storage Default'. The 'Cloud Storage Default' connection is expanded, showing 'Sample Buckets' and 'Kafka Connector(0)'. The status bar at the bottom right indicates 'Instance id: data-eng-prj/datafusion-dev'.

[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 modal window titled 'Join' is open over a table named 'employee\_data.csv'. The table has 12 columns and 150 rows. The columns are: employee\_id, first\_name, last\_name, email, phone\_number, date\_of\_birth, ssn, address, department, salary, password, and full\_name. The 'full\_name' column is currently empty. The 'Transformation steps' pane on the right shows the mapping from 'first\_name' and 'last\_name' to 'full\_name'. Both 'first\_name' and 'last\_name' have a completion rate of 100%. The 'full\_name' column also has a completion rate of 100%.

This screenshot shows the same interface after the join operation. The 'full\_name' column now contains the concatenated values of 'first\_name' and 'last\_name' for each row. The transformation steps pane shows that the 'full\_name' column now has a 100% completion rate, while 'first\_name' and 'last\_name' are no longer listed.

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

The screenshot shows the 'Mask' operation applied to the 'salary' column. A modal window titled 'Mask' is open, showing the selected characters to be masked (the entire salary value). The 'Transformation steps' pane on the right shows the 'salary' column with a 100% completion rate, indicating that the masking operation has been successfully applied.

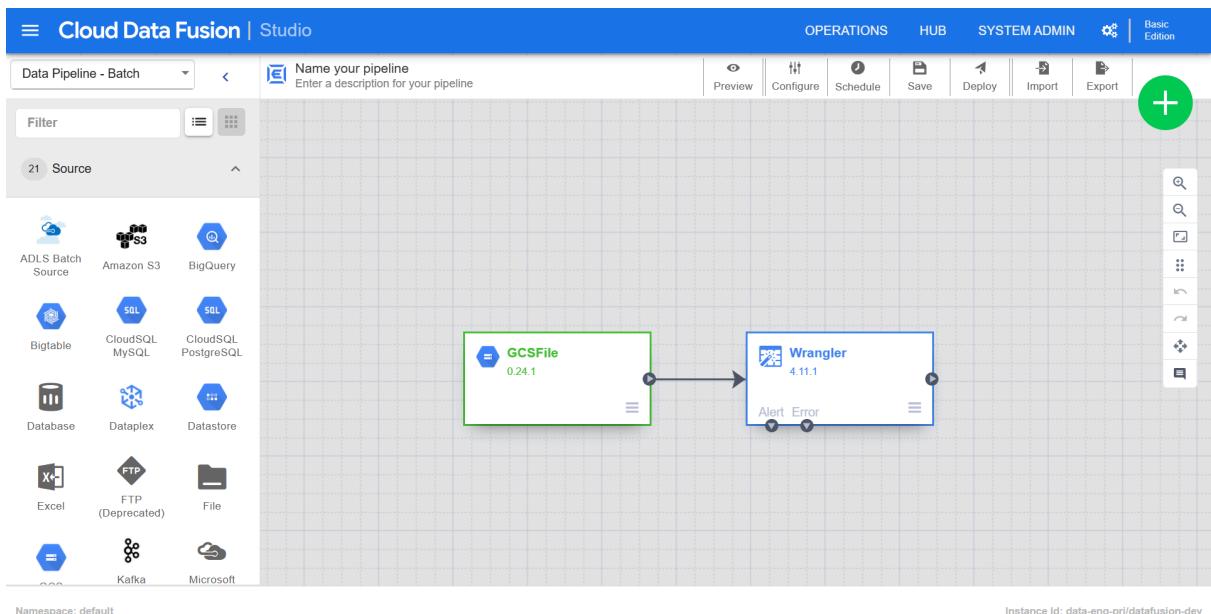
- **Password encoding** (applied similarly as masking/encoding).

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

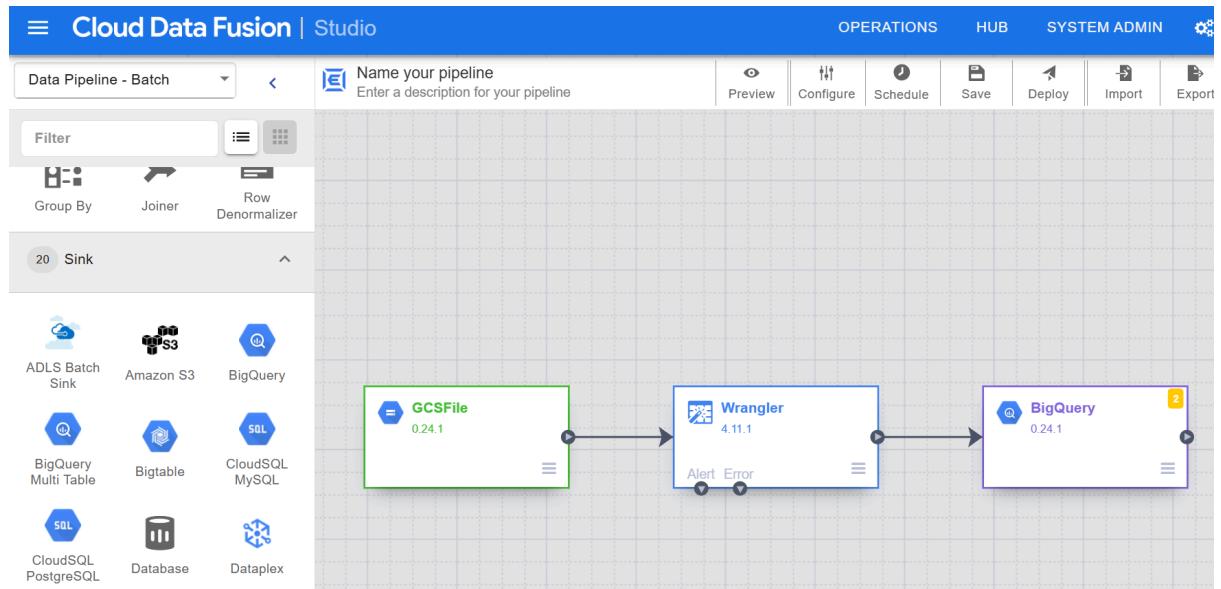
Columns (14)		Transformation steps (4)
<input type="checkbox"/> password		
V01SKXV3Q		1 merge :first_name :last_name :full_name , <span style="color: red;">×</span>
bVZVT3psOr		2 mask-number :salary xxxxxxx <span style="color: red;">×</span>
MmghKUJKC		3 encode base64 :password <span style="color: red;">×</span>
Nm1Gb0VH		4 encode hex :password <span style="color: red;">×</span>
UCHXUFBOT		
QGwoZOluZt		
MDIXRE84b:		
M0AzYkizZy:		
JUQ3OUFm		

## 8.3 Build & Deploy Pipeline

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



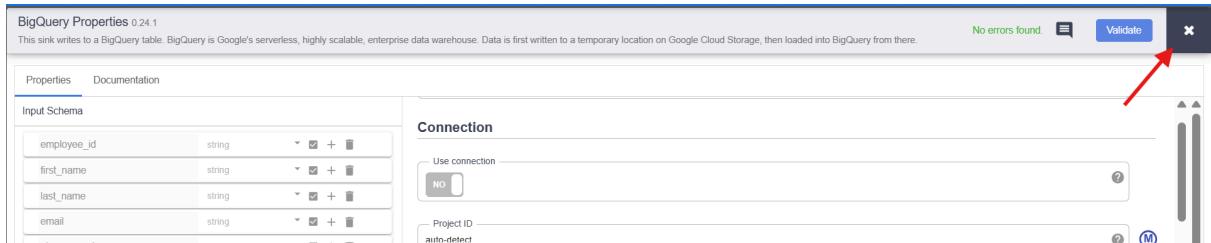
2. Add BigQuery sink (under Sync → BigQuery).



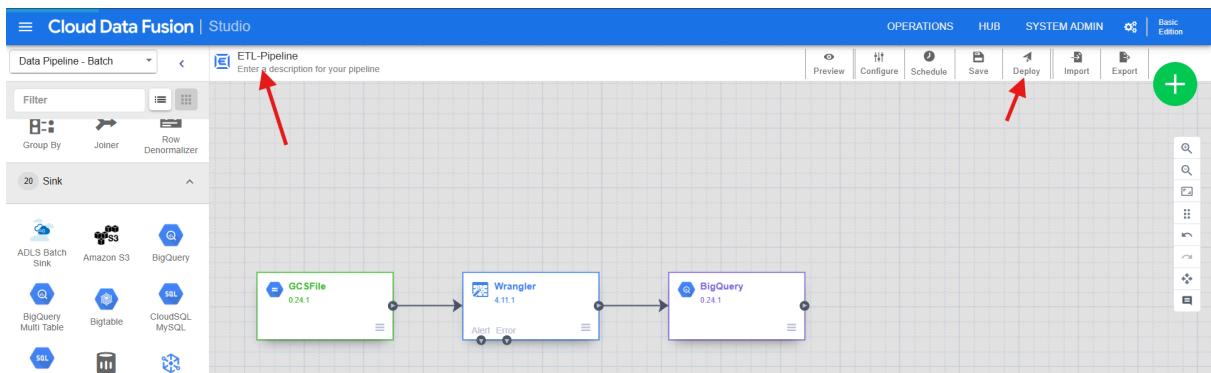
### 3. Configure BigQuery sink properties:

- 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.

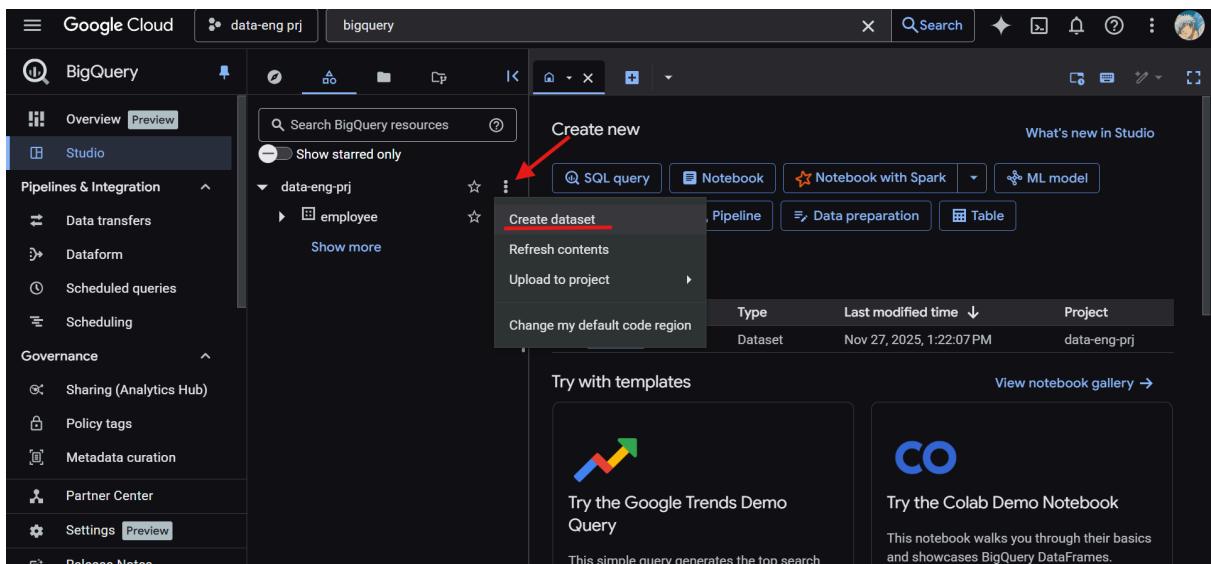
# Stage 3 — BigQuery Setup & Data Load

## 9.1 Description

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

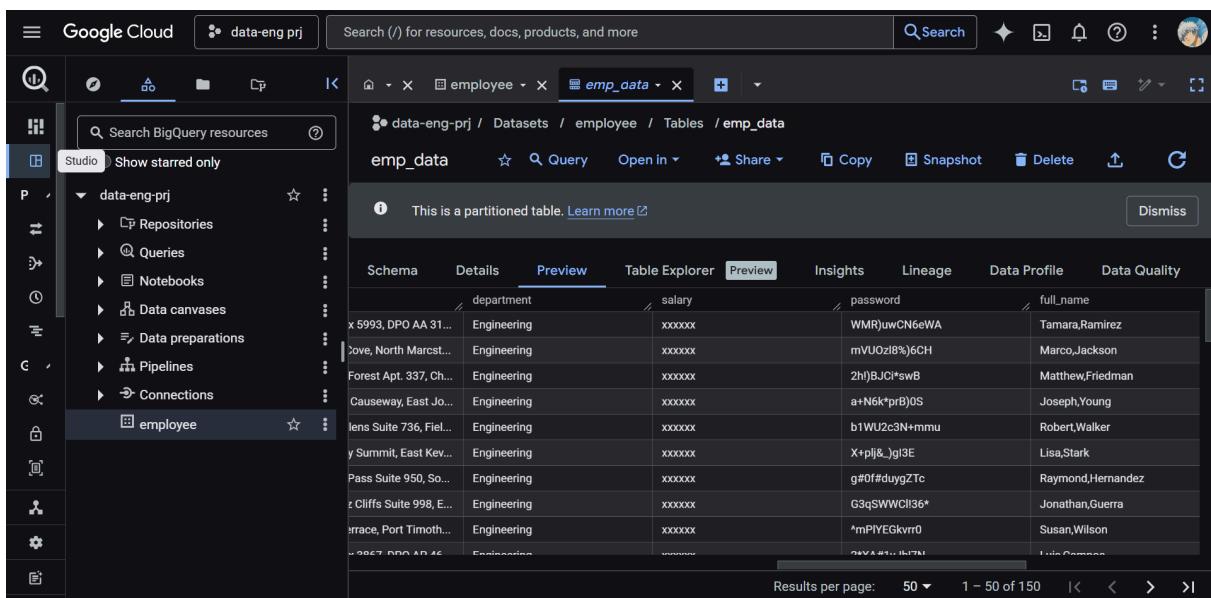
## 9.2 Steps

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



The screenshot shows the Google Cloud BigQuery Studio interface. On the left, the sidebar includes sections for Pipelines & Integration (Data transfers, Dataform, Scheduled queries, Scheduling), Governance (Sharing, Policy tags, Metadata curation, Partner Center, Settings), and a Release Notes section. The main area has tabs for Overview and Preview. A search bar at the top right says "Search BigQuery resources". Below it, there's a "Create new" dropdown menu with options like SQL query, Notebook, Notebook with Spark, ML model, Create dataset, Pipeline, Data preparation, and Table. The "Create dataset" option is highlighted with a red arrow. To the right of the dropdown, there's a "What's new in Studio" section. At the bottom, there are two cards: "Try with templates" (Google Trends Demo Query) and "View notebook gallery" (Colab Demo Notebook).

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.



The screenshot shows the Google Cloud BigQuery Studio interface. The sidebar is similar to the previous screenshot. The main area shows a dataset named "employee" under the "data-eng-prj" project. Inside the "employee" dataset, there is a table named "emp\_data". The table is described as a partitioned table. The "Preview" tab is selected, showing a table with columns: department, salary, password, and full\_name. The data rows show various employee entries with their department, salary (xxxxxx), password (long masked values), and full name. The bottom of the screen shows navigation controls for results per page (50), and back/forward arrows.

department	salary	password	full_name
x 5993, DPO AA 31...	Engineering	xxxxxx	WMRJuwCN6eWA
Cove, North Marsti...	Engineering	xxxxxx	mVUOzI8%6CH
Forest Apt. 337, Ch...	Engineering	xxxxxx	2hJBjCl*swB
Causeway, East Jo...	Engineering	xxxxxx	a+N6k*prB)OS
Lens Suite 736, Fiel...	Engineering	xxxxxx	b1WU2c3N+mmu
y Summit, East Kev...	Engineering	xxxxxx	X+pIj&_gj3E
Pass Suite 950, So...	Engineering	xxxxxx	g#0f#uygZTc
z Cliffs Suite 998, E...	Engineering	xxxxxx	G3qSWWC!l56*
terrace, Port Timoth...	Engineering	xxxxxx	^mPIYEGkvrr0
w 997 DPO AD 16	Engineering	xxxxxx	Susan,Wilson

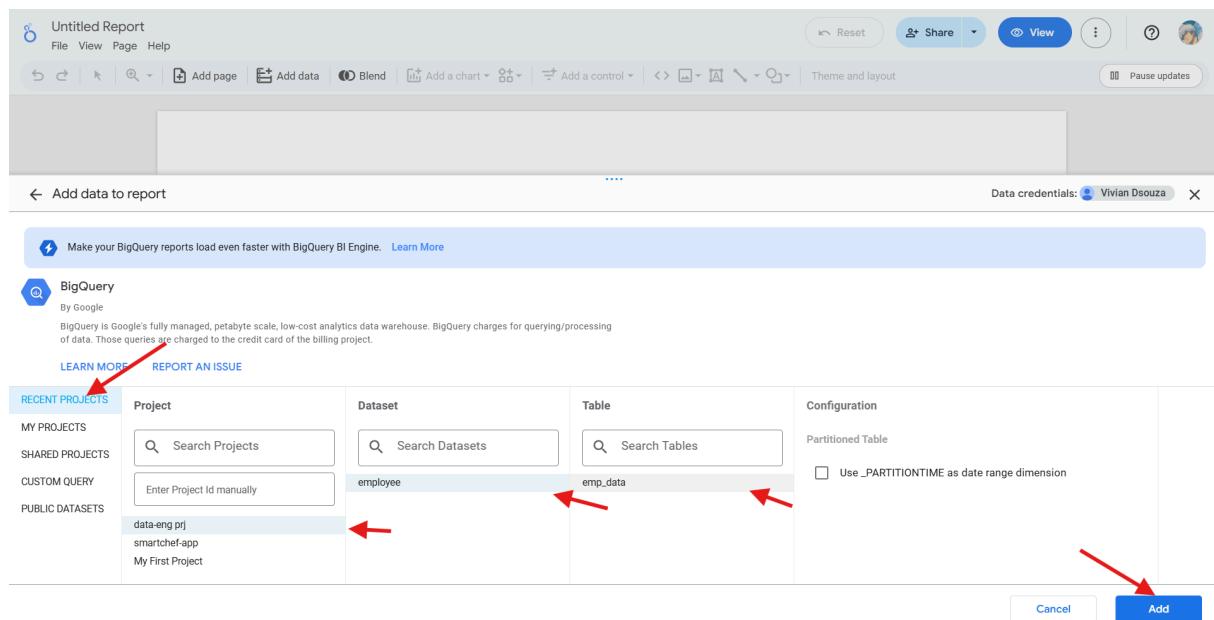
# Stage 4 — Visualization (Looker Studio)

## 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 Tableau desktop application. On the left, there is a data grid titled "emp\_data" containing employee information such as employee\_id, full\_name, email, and department. On the right, the "Table properties" pane is open, specifically the "Setup" tab under "Chart types". A red arrow points from the "Dimension" section to the "employee\_id" field. Another red arrow points from the "Dimension" section to the "full\_name" field. A third red arrow points from the "Metric" section to the "Record Count" field. The "Data" pane on the far right lists various fields like address, date\_of\_birth, department, etc.

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

The screenshot shows a completed Tableau dashboard titled "Employee Dashboard". The dashboard features a "Dropdown Control" for "department" at the top. Below it is a table visual showing employee details. To the right of the table is a bar chart titled "first\_name" with categories Finance, Sales, IT Support, HR, Engineering, and Marketing. To the right of the bar chart is a pie chart showing department distribution with segments for Finance (16.5%), Sales (13%), Marketing (12.1%), IT Support (12.2%), HR (10.8%), and Engineering (10.2%). The "Data" pane on the right side of the interface lists fields such as address, date\_of\_birth, department, email, first\_name, full\_name, last\_name, password, password\_encode\_base64, phone\_number, salary, and ssn.

# 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.

The screenshot shows the Google Cloud Platform interface for managing environments. In the top navigation bar, 'Google Cloud' and 'data-eng pri' are visible. Below the navigation bar, there's a search bar with 'comp' typed into it. The main area displays a table of environments. The columns include State, Name (sorted by name), Location, Composer version, Airflow version, Creation time, Update time, Airflow webserver, DAG list, Logs, DAGs, DAGsolder, and Labels. Two red arrows point to the 'Airflow' and 'DAGs' column headers respectively. The 'composer-dev' environment is listed with a green status icon, 'us-central1' location, '3' Composer version, '2.10.5-build.19' Airflow version, and creation and update times of '11/27/25, 2:36 PM'. The 'Airflow' and 'DAGs' columns show small icons.

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 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.

This screenshot is identical to the one above, showing the Cloud Composer environments list. It includes the same table structure and the same environment entry for 'composer-dev'. Red arrows point to the 'Airflow' and 'DAGs' column headers.

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

The screenshot shows the Google Cloud Storage interface for the 'data-eng-prj' project. The left sidebar shows 'Cloud Storage' with 'Buckets' selected. The main area displays the 'us-central1-composer-dev-12108968-bucket'. Under the 'Objects' tab, there is a list of files and folders in the 'dags' directory. One file, 'airflow\_monitoring.py', is listed with a size of 960 B, type text/x-python, and creation date Nov 27, 2025, 2:58:33 PM. Below it is a folder named 'scripts/'. The 'scripts/' folder is highlighted with a red box.

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

This screenshot shows the same Google Cloud Storage interface after the 'dag.py' file has been uploaded. The 'dags' directory now contains three entries: 'airflow\_monitoring.py', 'dag.py', and the 'scripts/' folder. The 'dag.py' file is listed with a size of 960 B, type text/x-python, and creation date Nov 27, 2025, 2:58:33 PM. The 'scripts/' folder is also present.

- 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.

The screenshot shows the Airflow UI for the 'employee\_data' DAG. The DAG was scheduled to run daily at 00:00 UTC on November 26, 2025. The task 'extract\_data' is shown as failed, indicated by a red box and an arrow. The task duration is listed as 0.00:07. The 'Logs' tab is highlighted with a red box and an arrow, showing the error message: 'extract\_data [failed] BashOperator'.

## 11.4 Running & Observing DAG

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

The screenshot shows the Airflow Cloud Composer Dev UI. At the top, there are filters for 'All' (2), 'Active' (2), 'Paused' (0), 'Running' (1), and 'Failed' (1). Below the filters is a search bar for 'Search DAGs'. On the right, there are 'Auto-refresh' and 'C' buttons. The main area displays two DAGs: 'airflow\_monitoring' and 'employee\_data'. 'airflow\_monitoring' has 21 successful runs (green circles) and 1 failed run (red circle). 'employee\_data' has 1 successful run (green circle), 2 failed runs (red circles), and 1 @daily scheduled run. The 'employee\_data' DAG is currently selected.

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

The screenshot shows the Airflow UI for the 'employee\_data' DAG. The task 'extract\_data' is selected. The 'Logs' tab is highlighted with a red arrow. The log output shows a single entry: 'extract\_data failed BashOperator'. A red box highlights this error message.

## Composer Error Handling & Fixes

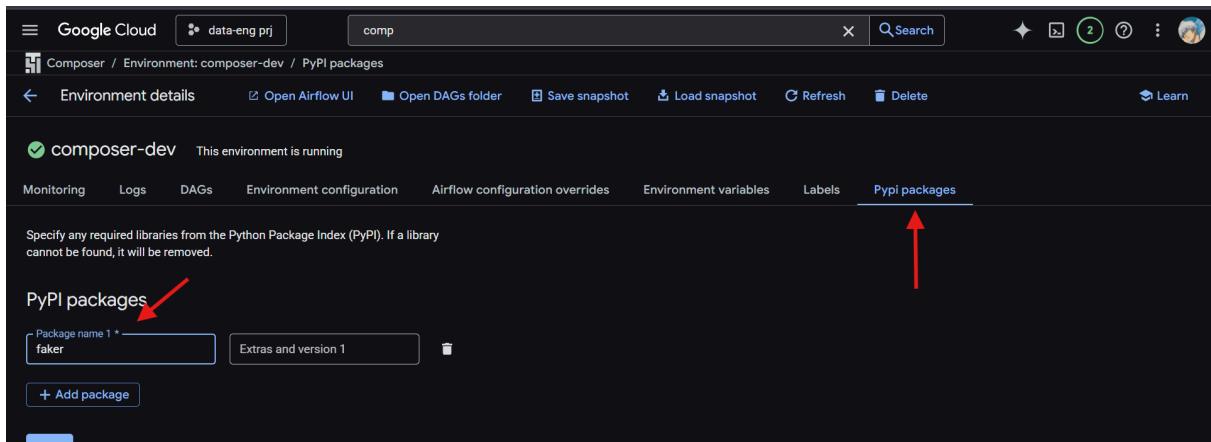
During the execution of the Airflow DAG inside Cloud Composer, one of the tasks failed. To identify the root cause of the failure, we first opened the Airflow UI and navigated to the task logs:

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.

---

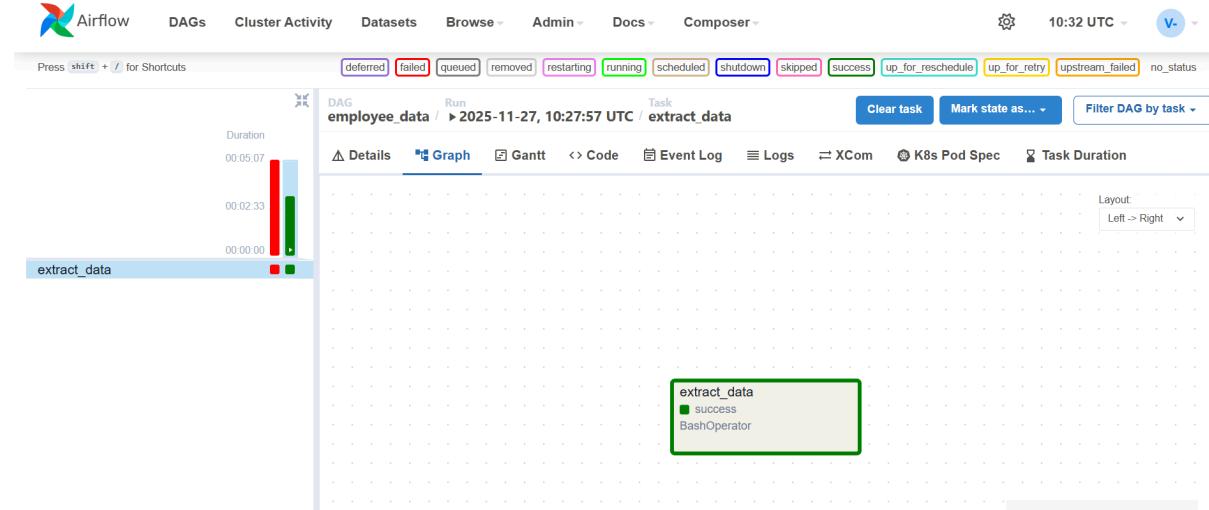
## Re-running the DAG

- 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.”**

The screenshot shows the Google Cloud Storage interface with the following details:

- Bucket:** bucket-employee-data
- Location:** us-central1 (Iowa)
- Storage class:** Standard
- Public access:** Not public
- Protection:** Soft Delete
- Objects:**
  - Name:** employee\_data.csv
  - Size:** 27.1 KB
  - Type:** text/csv
  - Created:** Nov 27, 2025, 4:01:06 PM
  - Storage class:** Standard
  - Last modified:** Nov 27, 2025, 4:01:06 PM
  - Public:** Not p

# 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

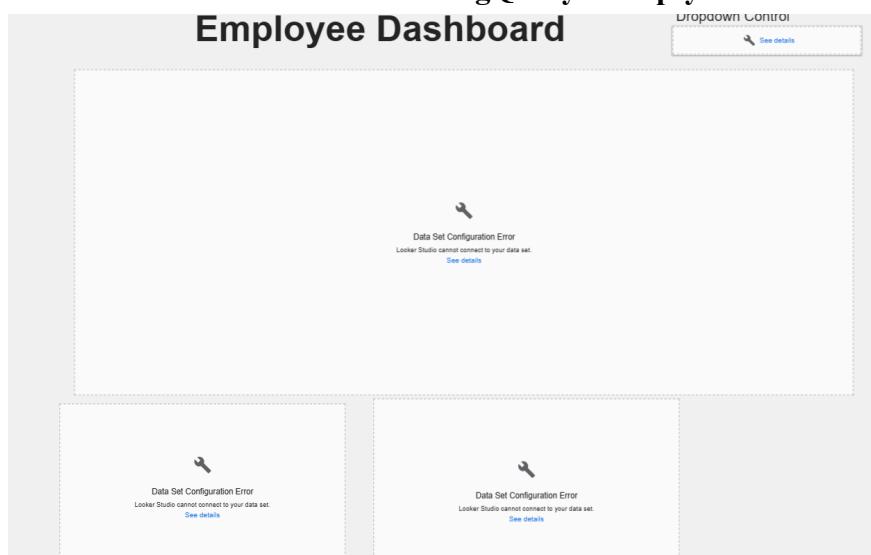
To fully validate the automated pipeline, we perform a complete end-to-end test by clearing the existing data, triggering the DAG, and verifying updates in each system: BigQuery → GCS → Data Fusion → Looker Studio.

## 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.”**

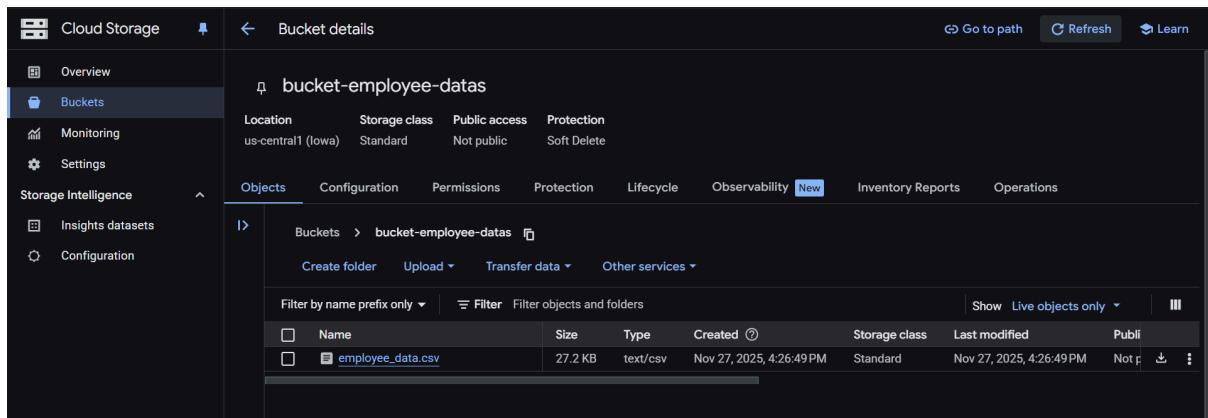


## 2. Running the Airflow DAG

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.”**



The screenshot shows the 'Bucket details' page for 'bucket-employee-data'. The left sidebar has 'Buckets' selected. The main area shows a single object named 'employee\_data.csv' with a size of 27.6 KB, type text/csv, and creation date Nov 27, 2025, 4:26:49 PM.

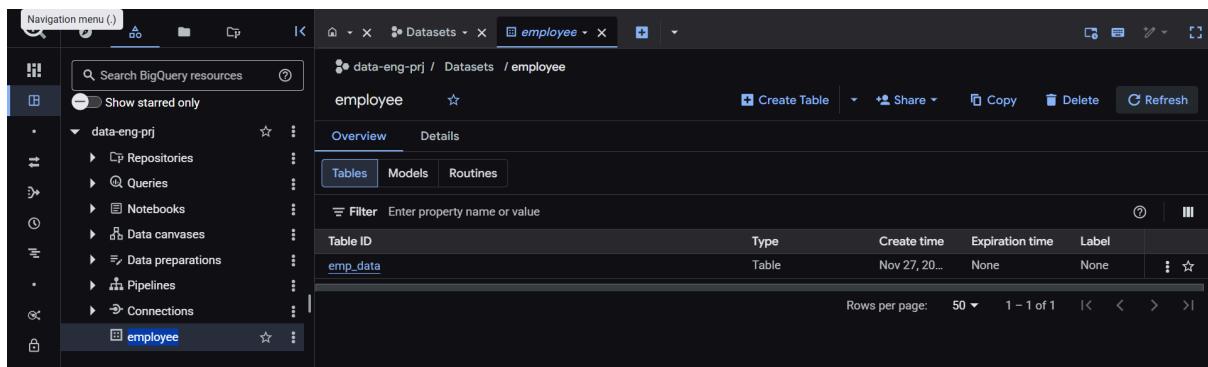
### 3. Cloud Data Fusion Pipeline Execution

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)



The screenshot shows the BigQuery UI with the 'data-eng-prj' project selected. The 'Tables' tab is active, showing a single table named 'emp\_data' with Type 'Table', Create time 'Nov 27, 2025', and Expiration time 'None'.

### 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.”**

The screenshot displays the Looker Studio interface with the following components:

- Employee Dashboard:** The title of the dashboard.
- Table:** Shows a list of employees with columns for employee\_id, full\_name, email, and department. The department column values are all 'Engineering'.
- Bar Chart:** A bar chart titled 'first\_name' showing counts for Marketing, IT Support, Finance, HR, Engineering, and Sales. The counts are approximately 30, 28, 25, 22, 18, and 19 respectively.
- Pie Chart:** A pie chart showing the distribution of employees by department. The segments are: Marketing (20.5%), Sales (18.4%), HR (15.6%), IT Support (12.8%), Finance (12.8%), and Engineering (18.9%).
- Dropdown Control:** A dropdown labeled 'Dropdown Control' set to 'department'.
- Data Panel:** A sidebar on the right containing a 'Let's get started' message and a list of data fields:
  - emp\_data
  - address
  - date\_of\_birth
  - department
  - email
  - employee\_id
  - first\_name
  - full\_name
  - last\_name
  - password
  - password\_encode\_...
  - phone\_number
  - salary
  - ssn
  - Record Count
- Tool Buttons:** Buttons for Add filter, Share, View, and Pause updates.
- Bottom Buttons:** Buttons for Add a field, Add a parameter, and Add 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
```