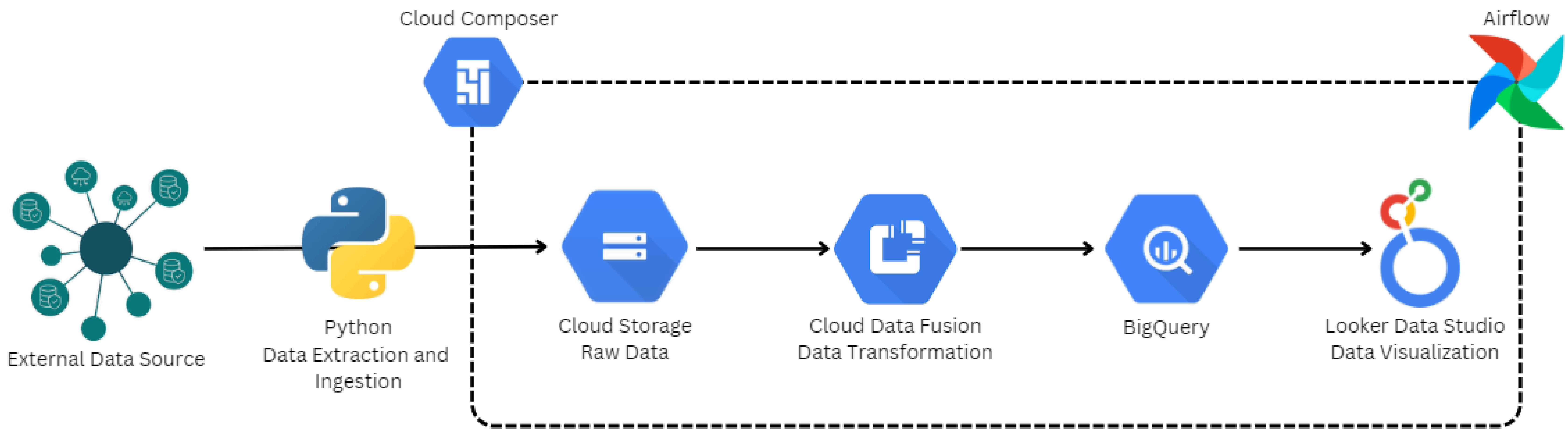
ETL Pipeline and Reporting in Google Cloud



Problem Statement:

You are tasked with designing and implementing a comprehensive data pipeline to manage employee data from multiple sources, ensuring that sensitive information is handled securely and appropriately masked. This data will then be loaded into Google BigQuery, where it will be available for secure visualization on a web-based dashboard.

Requirements

Data Extraction:

- Extract employee data from various sources, including databases, CSV files, and APIs.
- Ensure the process is efficient, scalable, and can handle data from multiple formats and sources.

Data Masking:

- Identify and mask sensitive information within the employee data, such as social security numbers, salary details, and personal contact information.
- Implement robust data masking techniques to maintain data privacy and security.

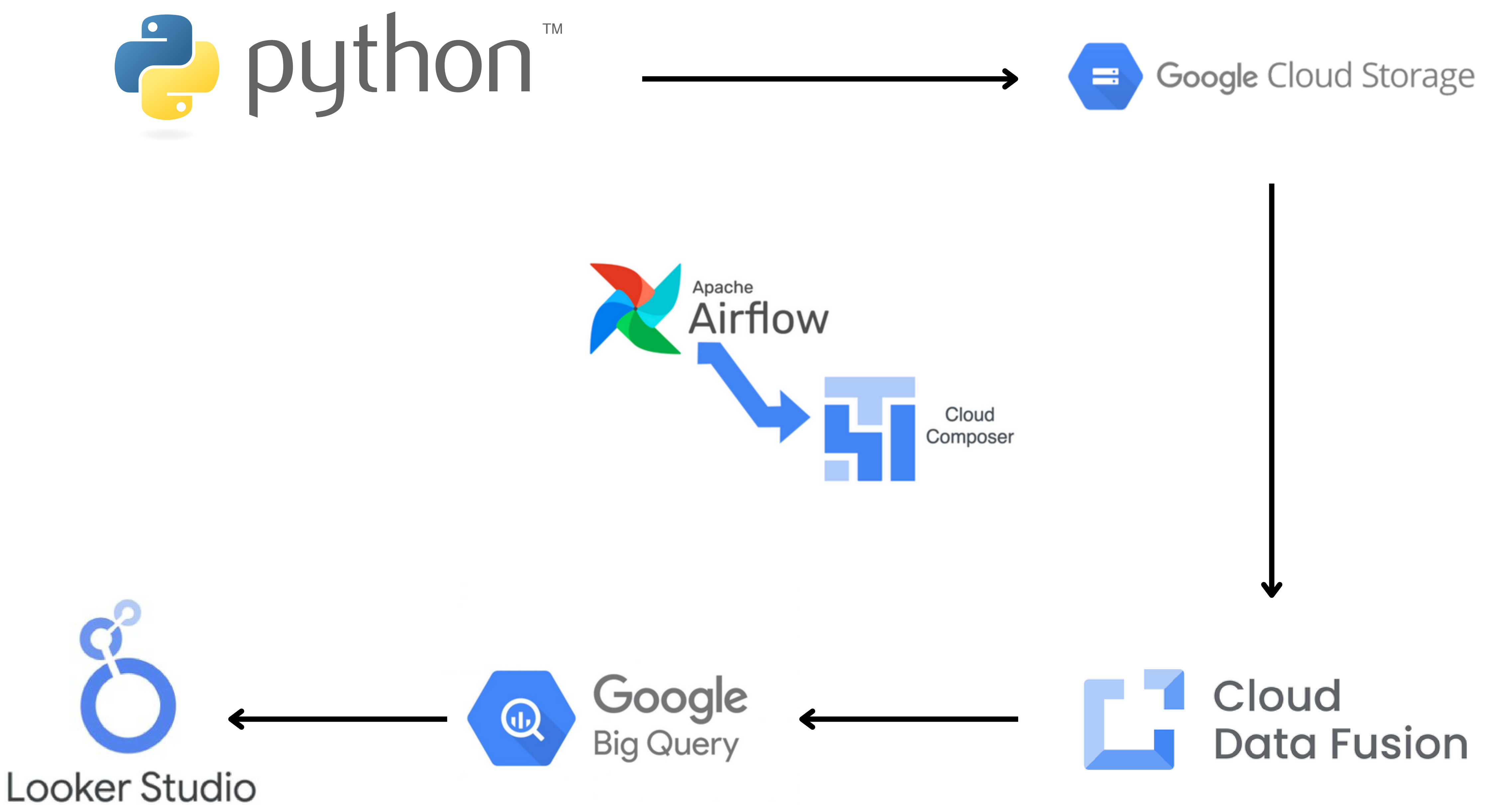
Data Loading into BigQuery:

- Design and implement a secure process to load the masked employee data into Google BigQuery.
- Ensure the data loading process is optimized for performance, reliability, and scalability.

Dashboard Visualization:

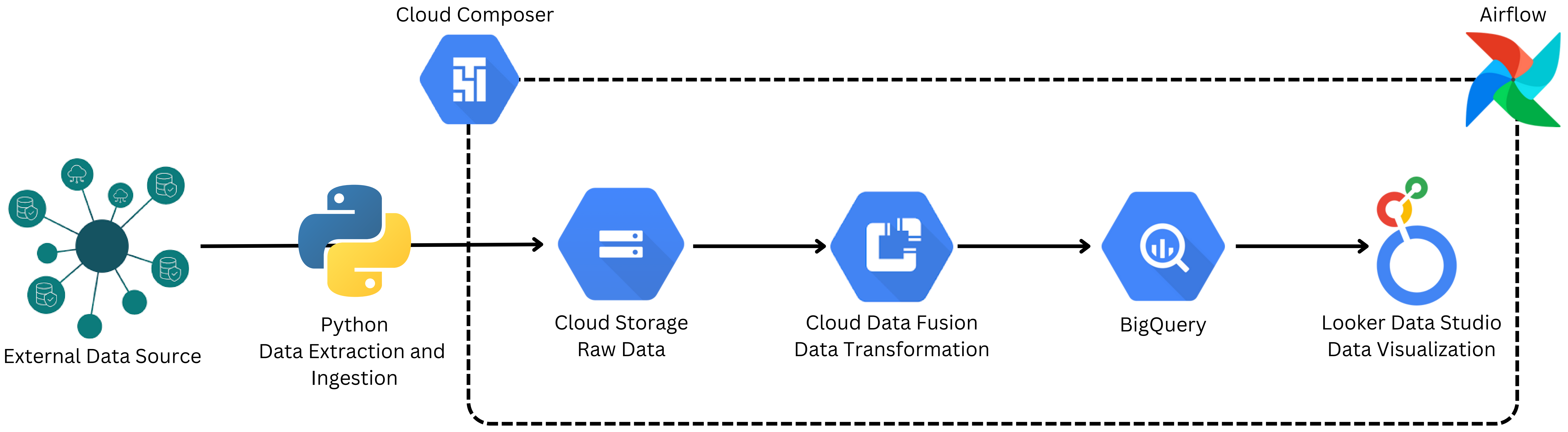
- Develop a user-friendly, web-based dashboard using visualization tools such as Google Data Studio, Tableau, or custom dashboards.
- Ensure that the dashboard provides clear, secure, and insightful visualizations of the employee data.

TechStack



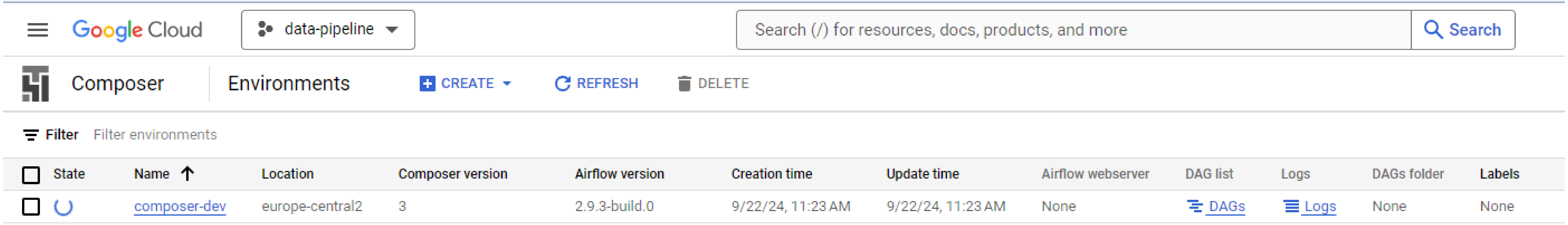
1. Python will be used for Data Extraction
2. We will use google cloud storage for storing the RAW data
3. Cloud data fusion as a data pipeline where we will be doing data transformation
4. Then we will load the data in BigQuery and visualize the data in looker studio.
5. Finally all the above process will be orchestrating on cloud composer using Airflow.
6. At the end we will visualize it in the looker studio

Pipeline: From Extraction to Visualization



Above pipeline shows data flows from external data sources into a system for extraction and ingestion using Python. The ingested raw data is stored in Google Cloud Storage, from where it is processed and transformed using Cloud Data Fusion. The transformed data is then loaded into BigQuery for data warehousing and further analysis. Finally, the data is visualized using Looker Data Studio (Google Data Studio). The entire pipeline is orchestrated by Cloud Composer, utilizing Apache Airflow to automate and schedule tasks.

Creating a Composer

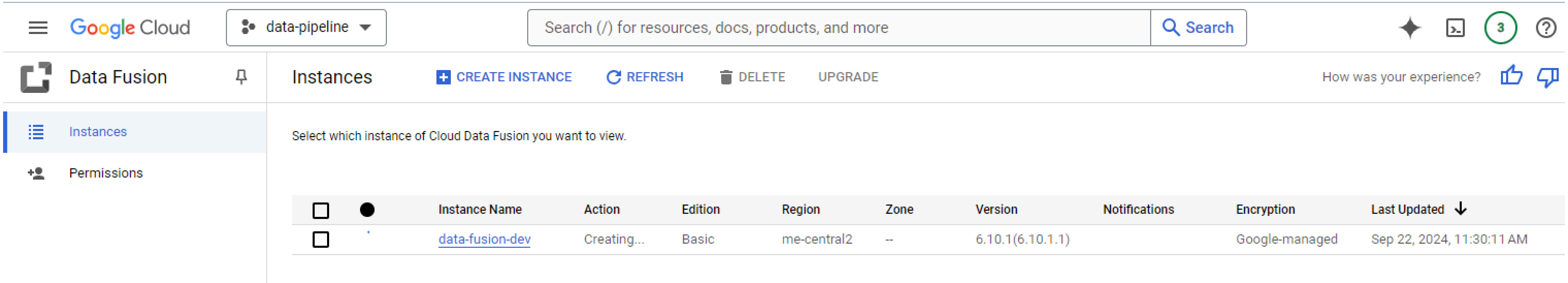


Here I have created cloud composer ***'composer-dev'*** which helps to create managed Airflow environments quickly and use Airflow-native tools

Google Cloud Composer is a workflow orchestration service that helps users:

- **Create, schedule, and monitor workflows:** Build, schedule, and monitor workflows across clouds and on-premises data centers
- **Integrate with other Google products:** Connect with other Google Cloud products like BigQuery, Dataflow, and Cloud Storage
- **Use Airflow-native tools:** Use Airflow's web interface and command-line tools
- **Automate complex workflows:** Streamline and automate workflows for data processing, machine learning, DevOps automation, and business process management

Creating a Data Fusion

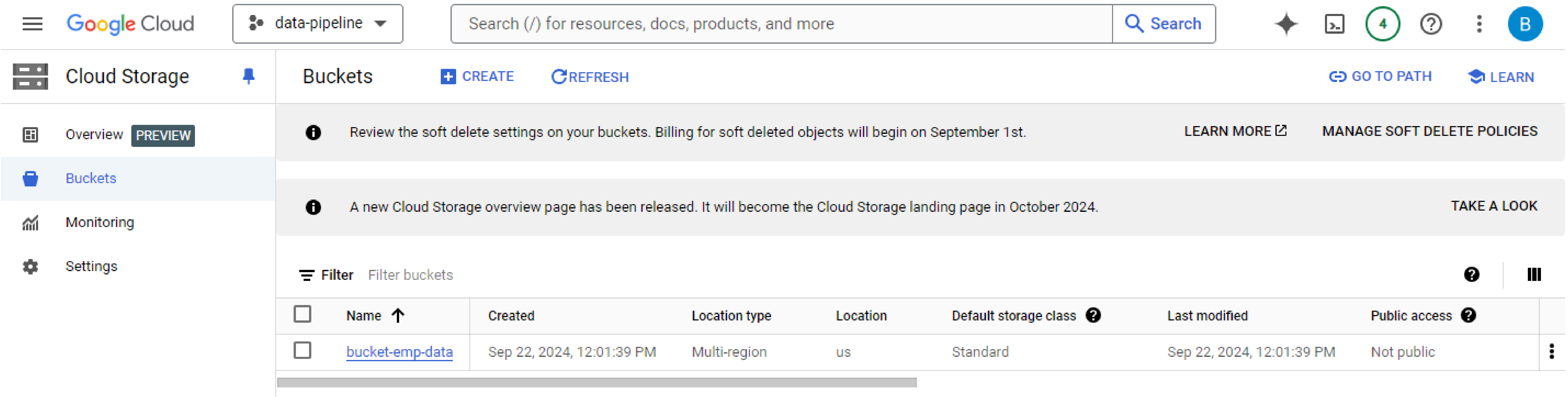


Created a data fusion ***'data-fusion-dev'***

Google Cloud Data Fusion is a cloud-native data integration service that helps users build and manage data pipelines:

- Data integration: Users can connect to various data sources, transform the data, and transfer it to other systems.
- Data lakes: Users can build scalable, distributed data lakes by integrating data from on-premises platforms.
- Data warehouses: Users can create data warehouses in BigQuery.
- Data visualization: Users can visualize data in Google Data Studio dashboards.

Creating a Bucket



Created bucket ***'bucket-emp-data'*** to store employee data.

Google Cloud Storage is a service that allows users to store and retrieve data in Google Cloud. It offers a variety of storage options, including:

- Standard storage: For data that is accessed frequently, such as mobile apps, websites, and streaming videos
- Nearline storage: A low-cost, durable storage option for data that is accessed infrequently
- Coldline storage: A very low-cost, durable storage option for data that is accessed infrequently
- Archive storage: A low-cost, durable storage service for data archiving, disaster recovery, and online backup

Creating a Python Script

```python
import csv
from faker import Faker
import random
import string
from google.cloud import storage

# Specify number of employees to generate
num_employees = 100

# Create Faker instance
fake = Faker()

# Define the character set for the password
password_characters = string.ascii_letters + string.digits + 'm'

# Generate employee data and save it to a CSV file
with open('employee_data.csv', mode='w', newline='') as file:
    fieldnames = ['first_name', 'last_name', 'job_title', 'department', 'email', 'address', 'phone_number', 'salary', 'password']
    writer = csv.DictWriter(file, fieldnames=fieldnames)

    writer.writeheader()
    for _ in range(num_employees):
        writer.writerow({
            "first_name": fake.first_name(),
            "last_name": fake.last_name(),
            "job_title": fake.job(),
            "department": fake.job(),  # Generate department-like data using the job() method
            "email": fake.email(),
            "address": fake.city(),
            "phone_number": fake.phone_number(),
            "salary": fake.random_number(digits=5),  # Generate a random 5-digit salary
            "password": ''.join(random.choice(password_characters) for _ in range(8))  # Generate an 8-character password with 'm'
        })

# Upload the CSV file to a GCS bucket
def upload_to_gcs(bucket_name, source_file_name, destination_blob_name):
    storage_client = storage.Client()
    bucket = storage_client.bucket(bucket_name)
    blob = bucket.blob(destination_blob_name)

    blob.upload_from_filename(source_file_name)

    print(f'File {source_file_name} uploaded to {destination_blob_name} in {bucket_name}.')

# Set your GCS bucket name and destination file name
bucket_name = 'bucket-emp-data'
source_file_name = 'employee_data.csv'
destination_blob_name = 'employee_data.csv'

# Upload the CSV file to GCS
upload_to_gcs(bucket_name, source_file_name, destination_blob_name)
```

Created a python script .

Data Generation:

- Uses the Faker library to generate synthetic employee data, such as first name, last name, job title, department, email, address, phone number, and a random salary for 100 employees.
- Generates an 8-character password for each employee using a combination of letters, digits, and the character 'm'.

CSV File Creation:

- Opens a file named employee_data.csv and writes the employee data in CSV format, including headers for each field (e.g., first name, last name, job title).

Google Cloud Storage (GCS) Setup:

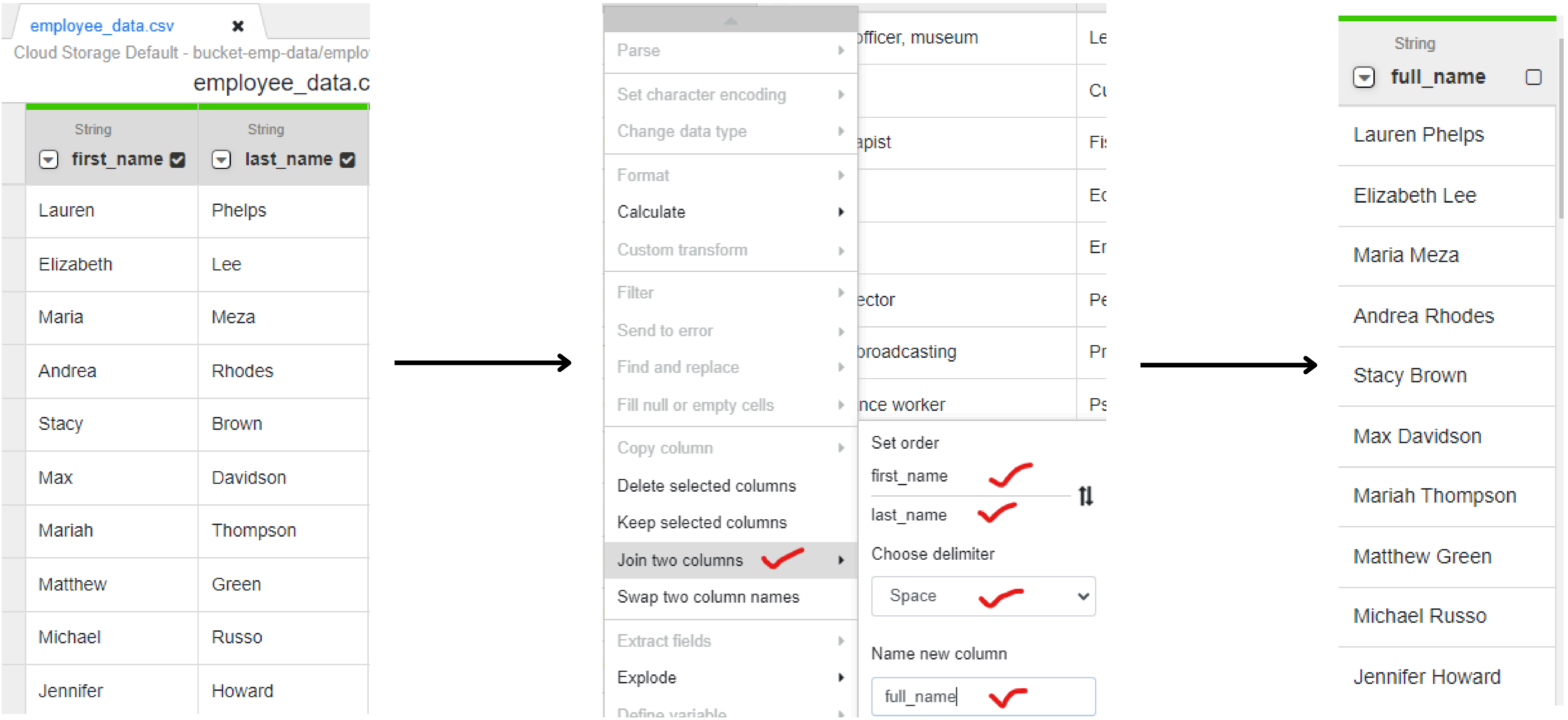
- Uses the google.cloud.storage library to create a client to connect to a Google Cloud Storage bucket.

Uploading the CSV File:

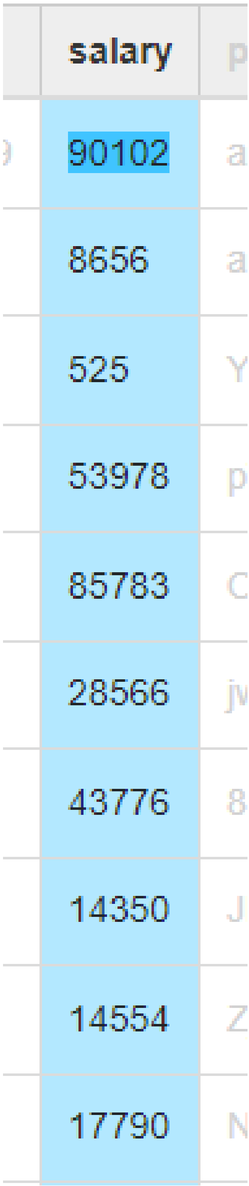
- Defines a function upload_to_gcs to upload the generated CSV file (employee_data.csv) to a specified GCS bucket (bucket-emp-data).
- Execution:
- Calls the function to upload the file, and upon success, prints a message confirming that the file has been uploaded to the desired bucket location in GCS.

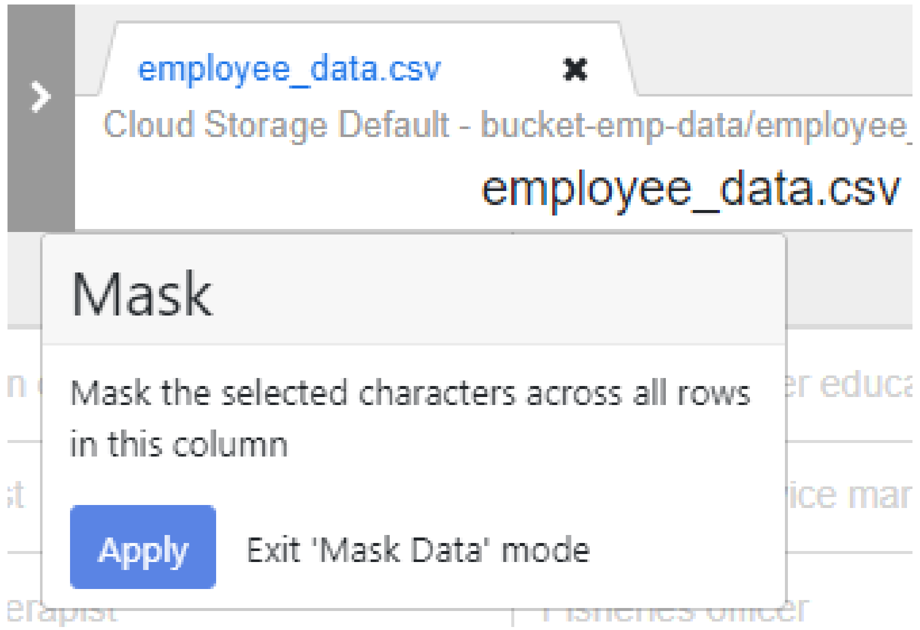
Data Fusion - Wrangler (Clean and Transform data)

Joining first_name and last_name as full_name

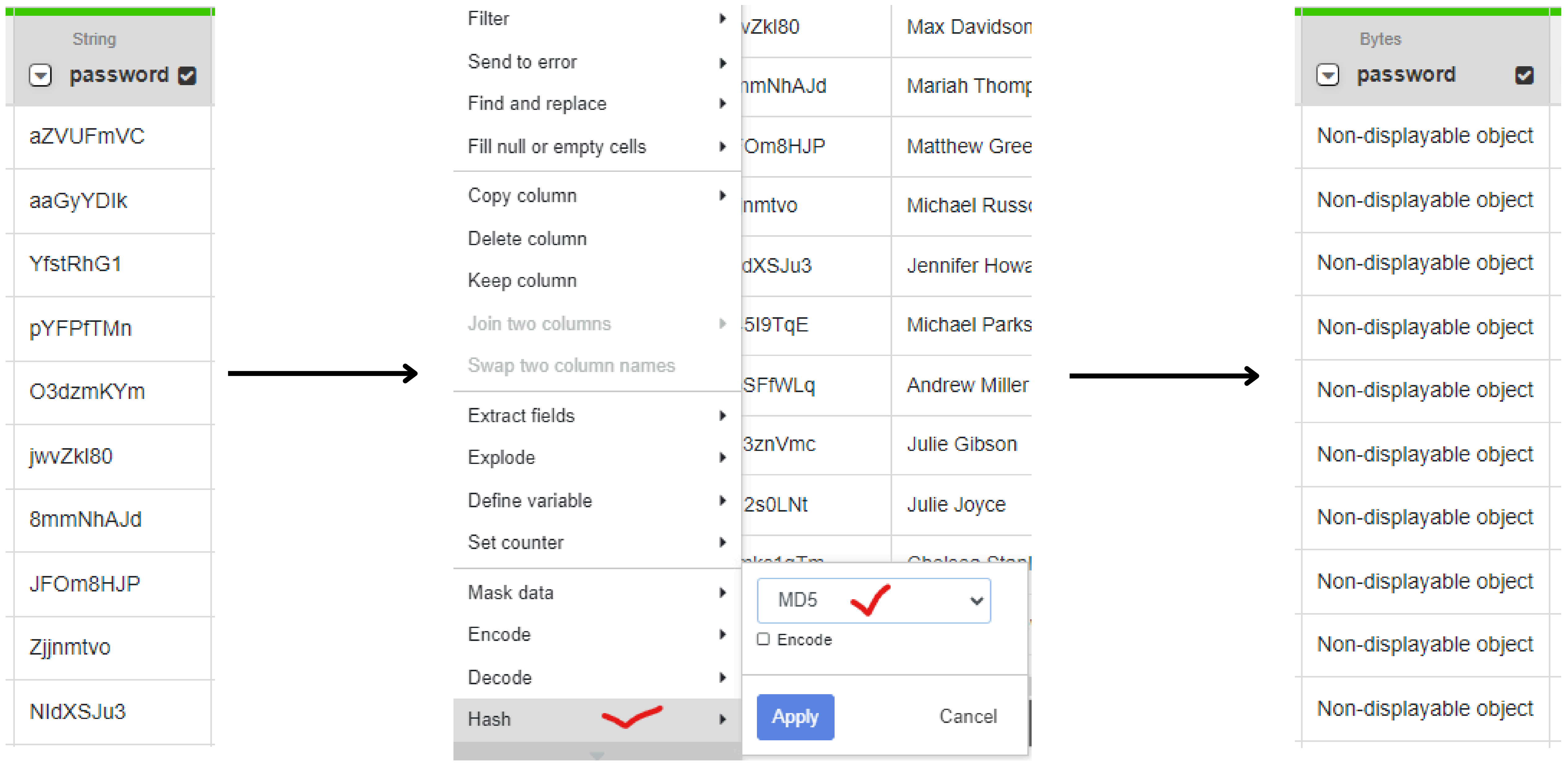


Masking the salary column

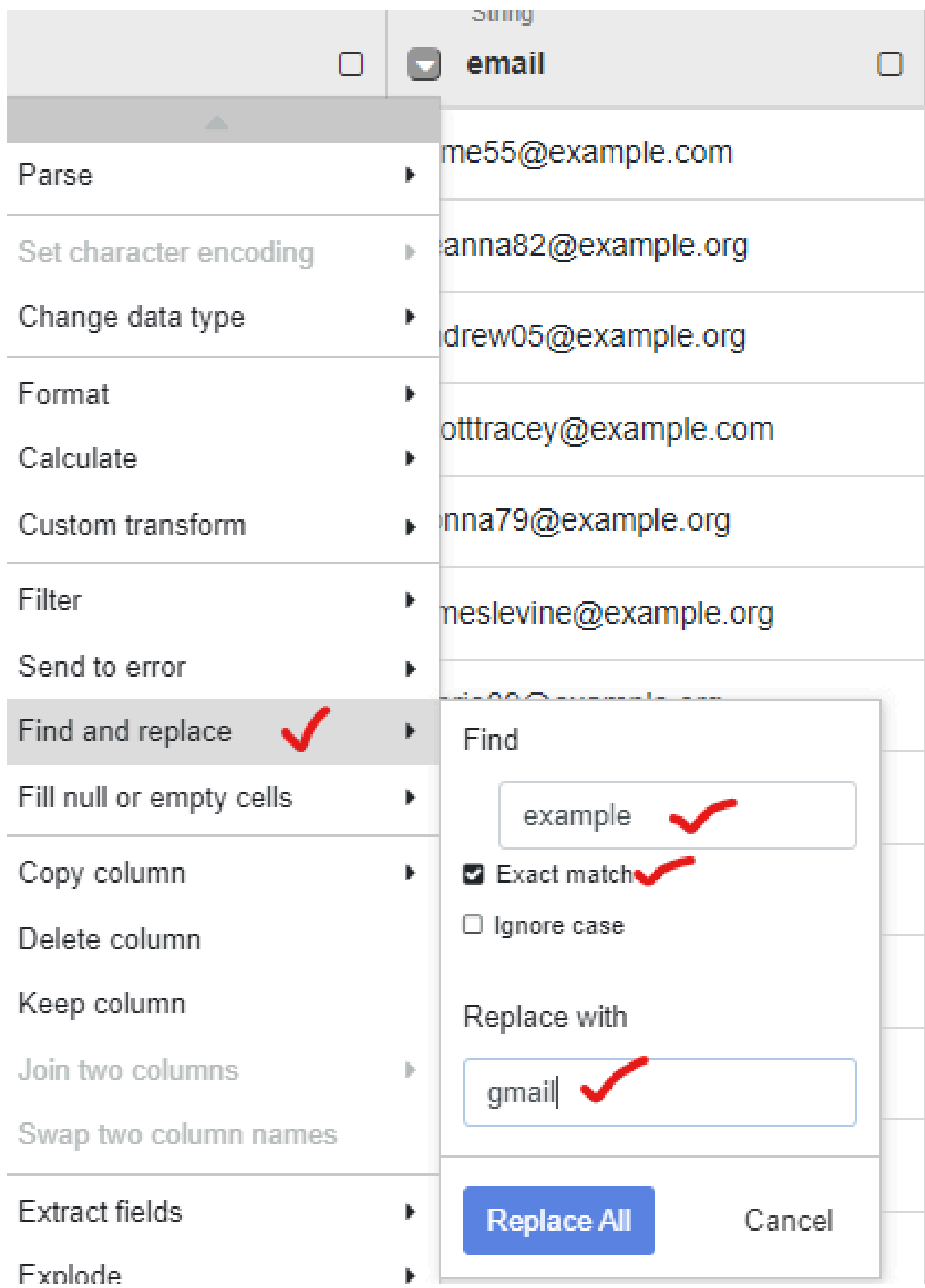


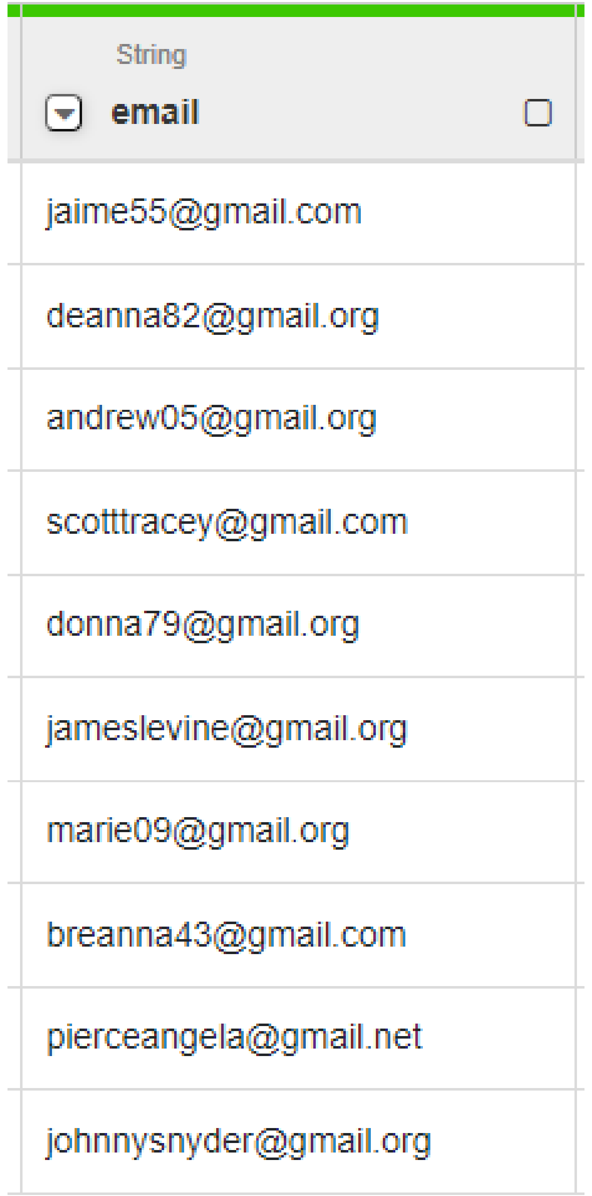


Hashing the password column

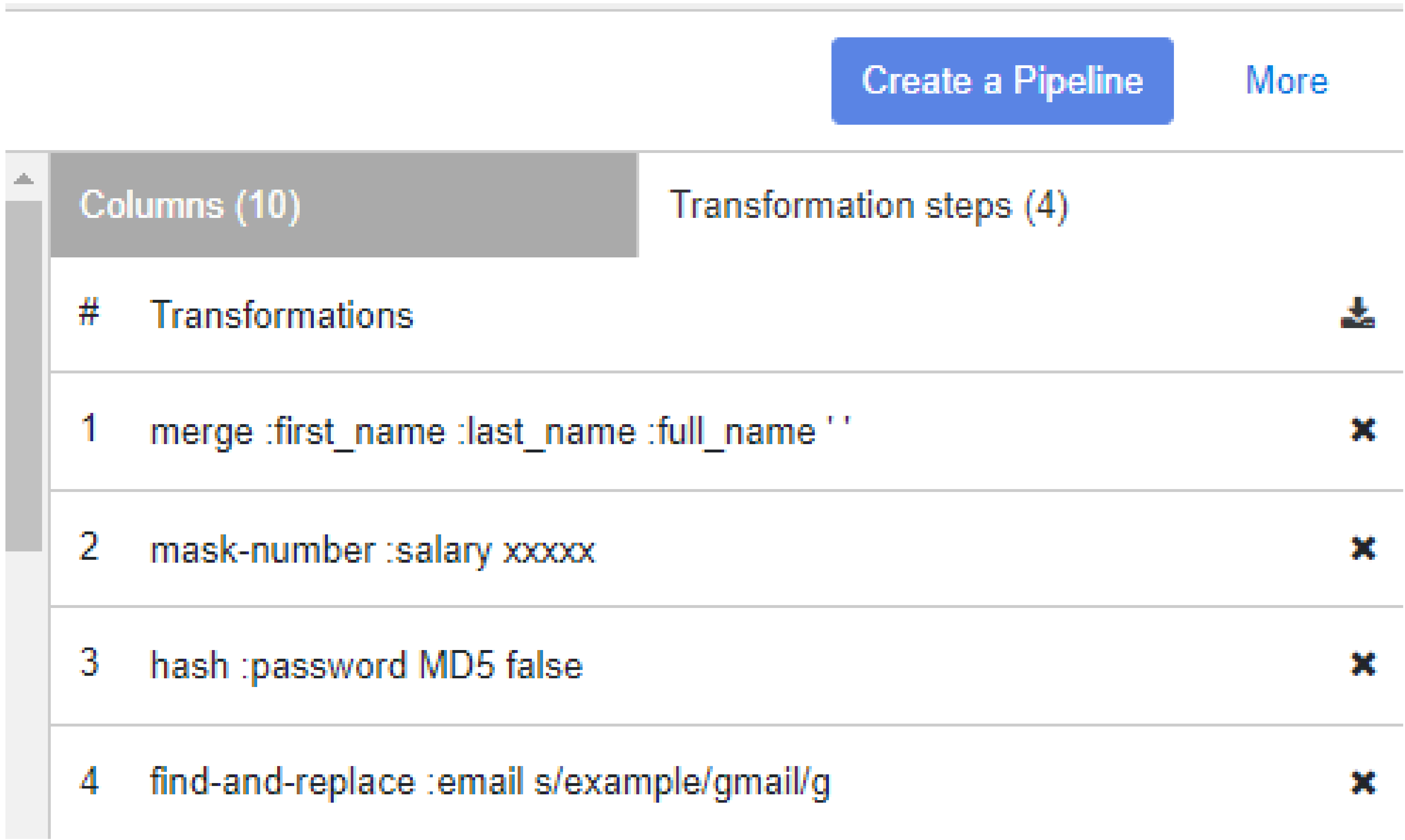


cleaning and replacing email column

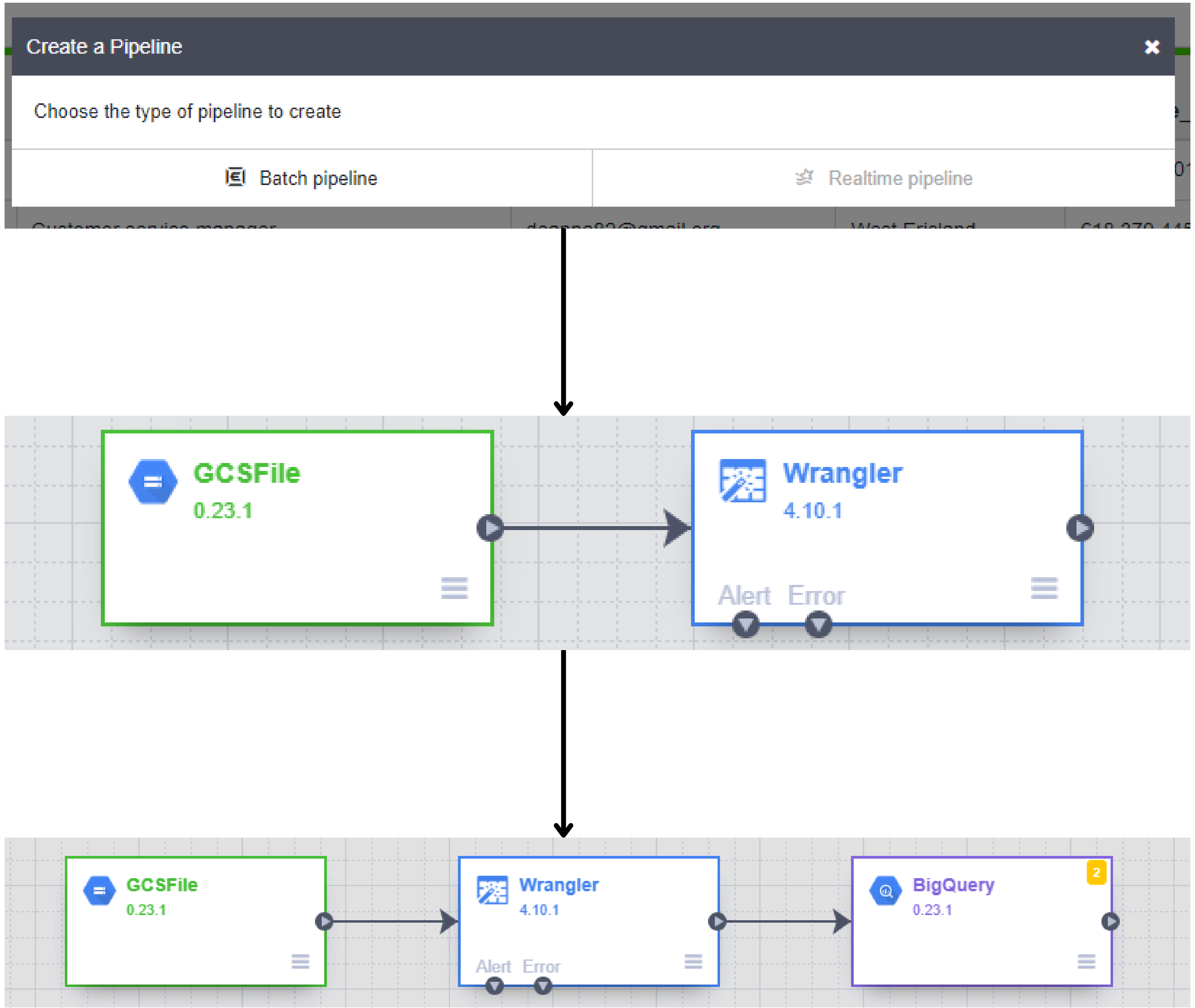




Transformation steps



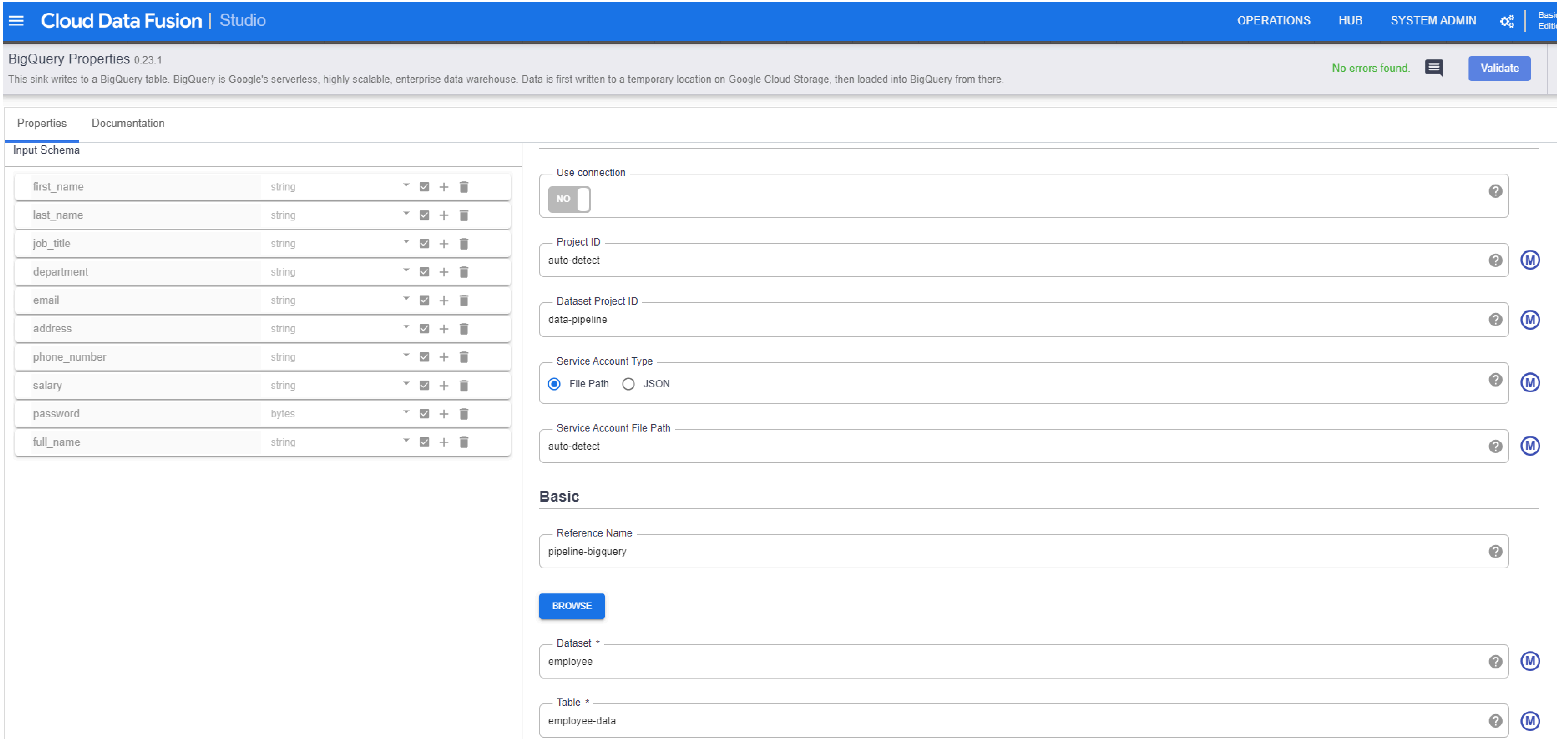
Creating Batch Pipeline



A batch pipeline is a system that processes large amounts of data in batches or at scheduled intervals. It's a structured and automated process that's often used for non-time-sensitive tasks and complex data processing.

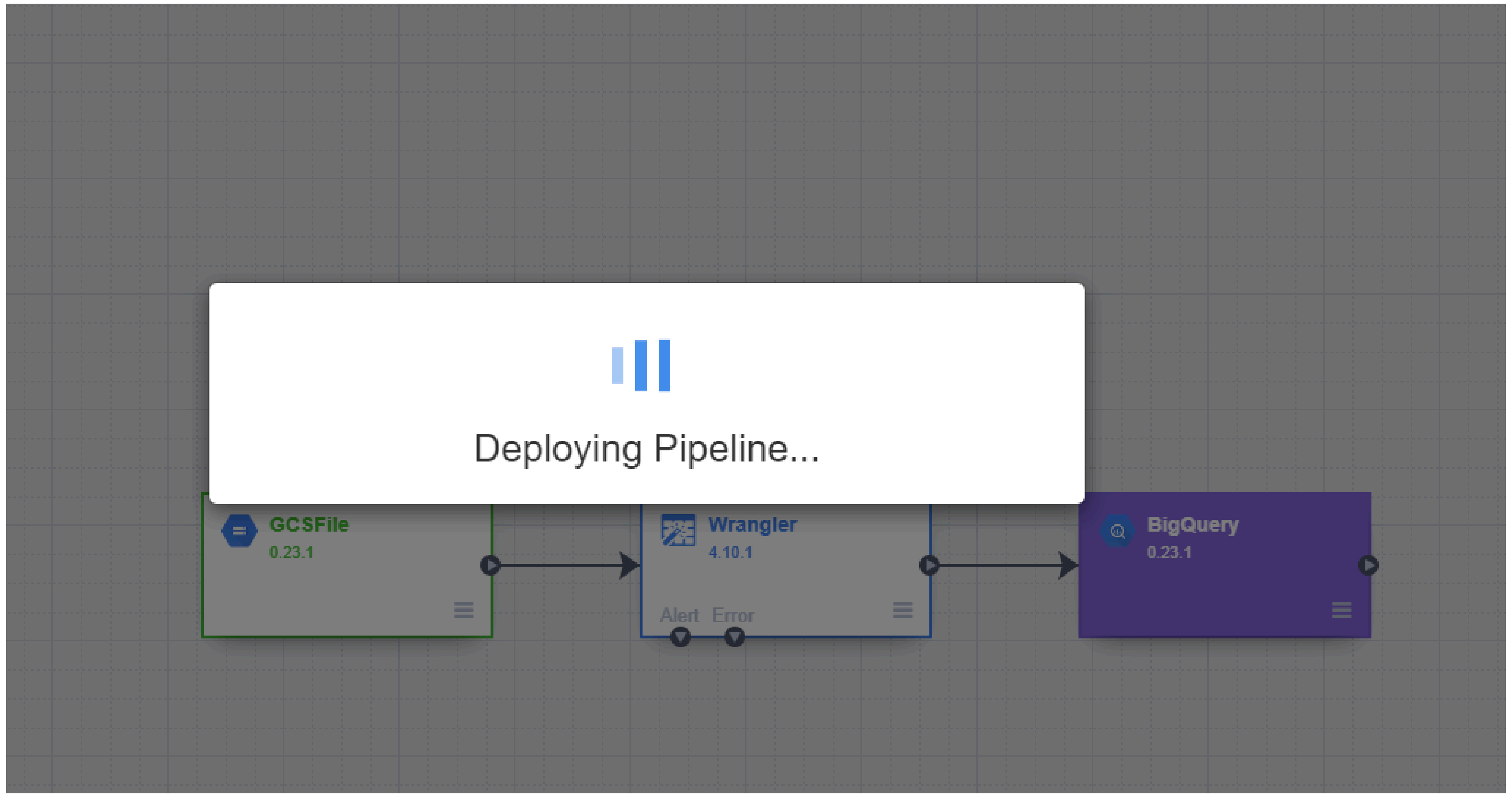
At final, dumping the data into bigquery after transformation for analysis.

Setting up BigQuery services

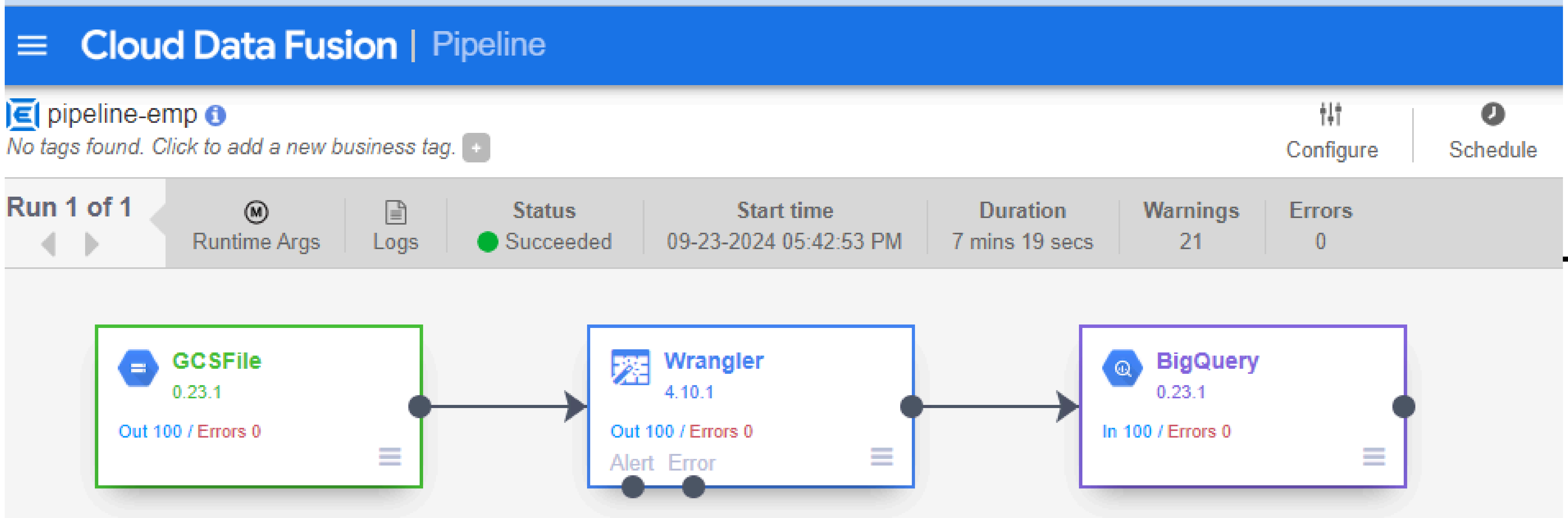


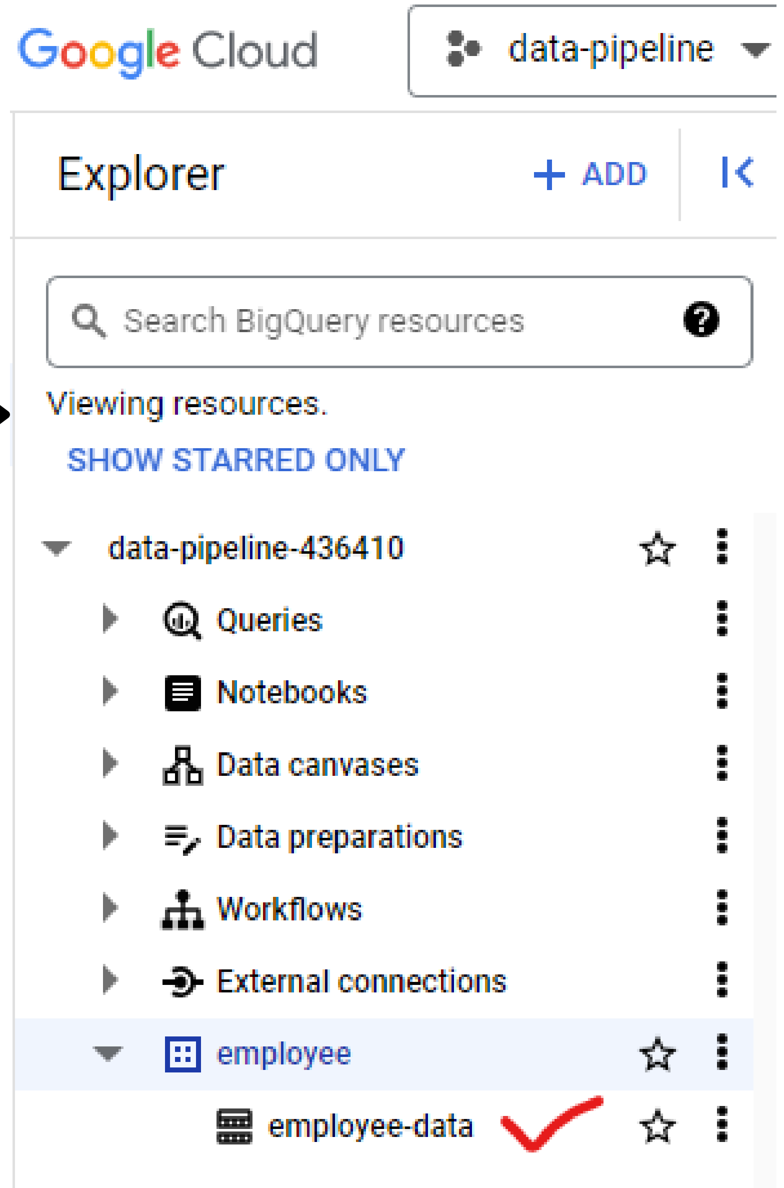
Deploying Pipeline

After successfully creating pipeline and setting up my bigquery its time to deploy the data into the bigquery services for further analysis.

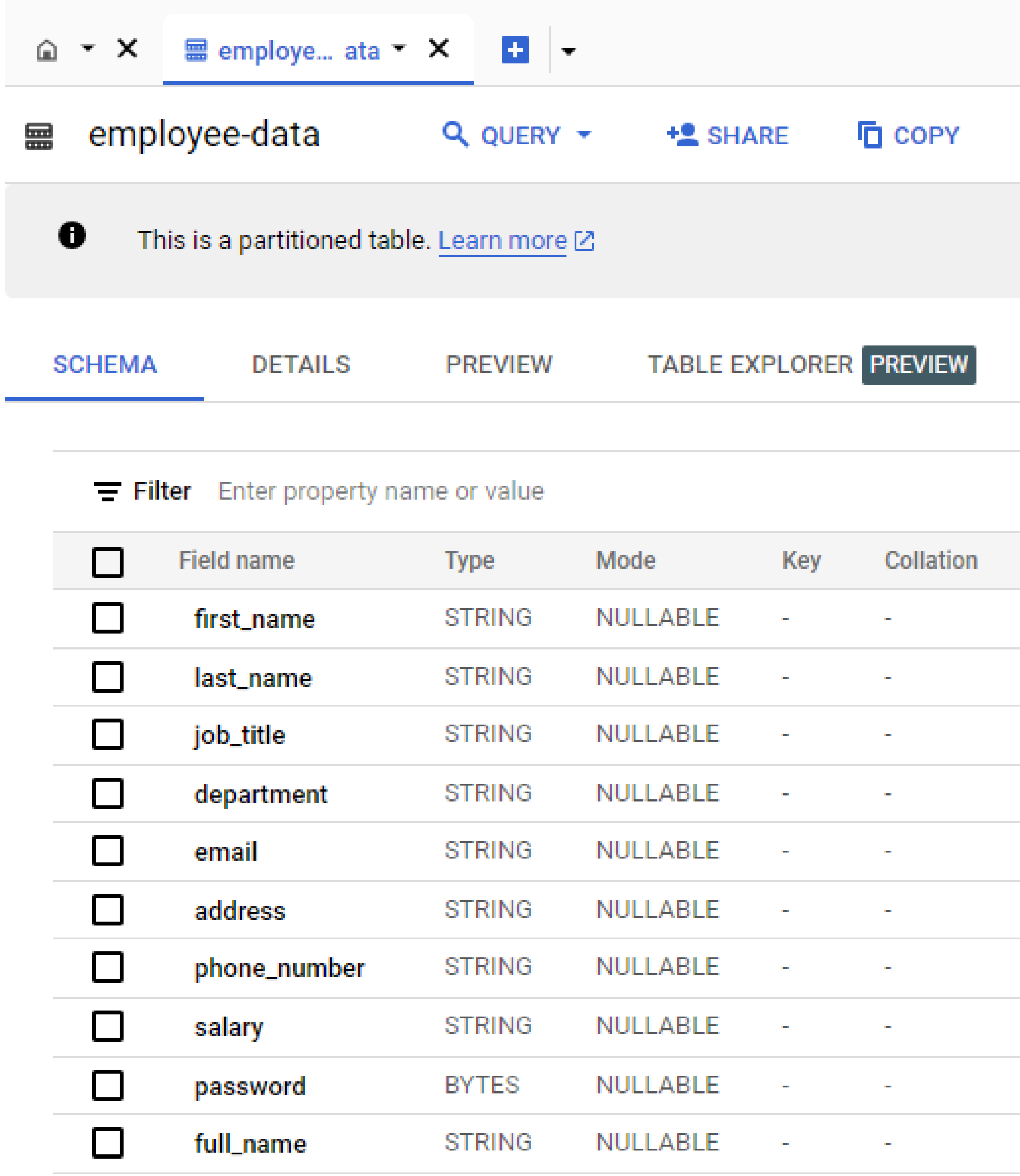


Data sinking into bigquery





As we can see employee-data is deployed into the bigquery services.



Verifying Schema

Cross checking the deployed data into schema where I can check for columns with respective data type and data mode.

BigQuery to validate data transformation

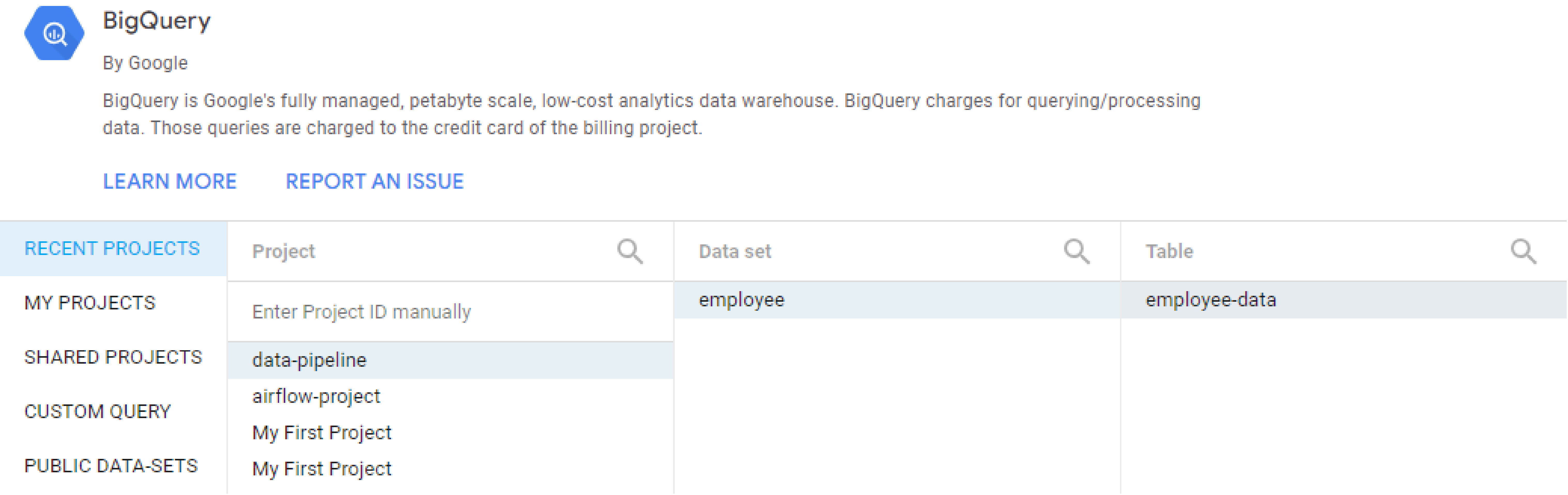
```
SELECT * FROM `data-pipeline-436410.employee.employee-data` ;
```

email	address	phone_number	salary	password	full_name
ashleyrice@gmail.com	Jeremymouth	564.456.5562x119	xxxxx	hHbhmezgvmVQF3RFweHEtQ==	Ann Barnett
jameslevine@gmail.org	North Leslie	853.670.2963x040	xxxxx	OaqRSx8xRaeyVRX5h3AOMg==	Max Davidson
jessica17@gmail.com	West Alexander	(205)603-7777	xxxxx	ZDGLg+8bfUe3RUhfZBZEVA==	Alex Johnson
mbrooks@gmail.org	Lake John	346.229.5141	xxxxx	oU59az+E9Nh10ZPkCgKkCg==	Cory Hall
shelia43@gmail.net	North Jennifer	772.962.8097	xxxxx	GQOW8ajP5Vz2+tlMmVEWmQ...	Drew Alexander
psoto@gmail.org	Lake Carmenville	001-507-812-5297x276	xxxxx	cXVrpOZ14Ee2ktmedGksPQ==	Eric Kirk
sjohnson@gmail.com	Connorview	001-616-713-4005x954	xxxxx	9VpfuRvwt4EofdTSjaaTUQ==	Eric Guzman
enichols@gmail.com	Robinchester	(380)594-9493	xxxxx	jl/S1g3HaUJ7Njt42vxAfA==	John Tyler
howardtran@gmail.org	Lake Danielburgh	+1-208-865-3818x1697	xxxxx	IE9AoBbyMj8Zq959fPnFNg==	Kyle Moran

In data fusion services I have transformed the data, here we can validate it.

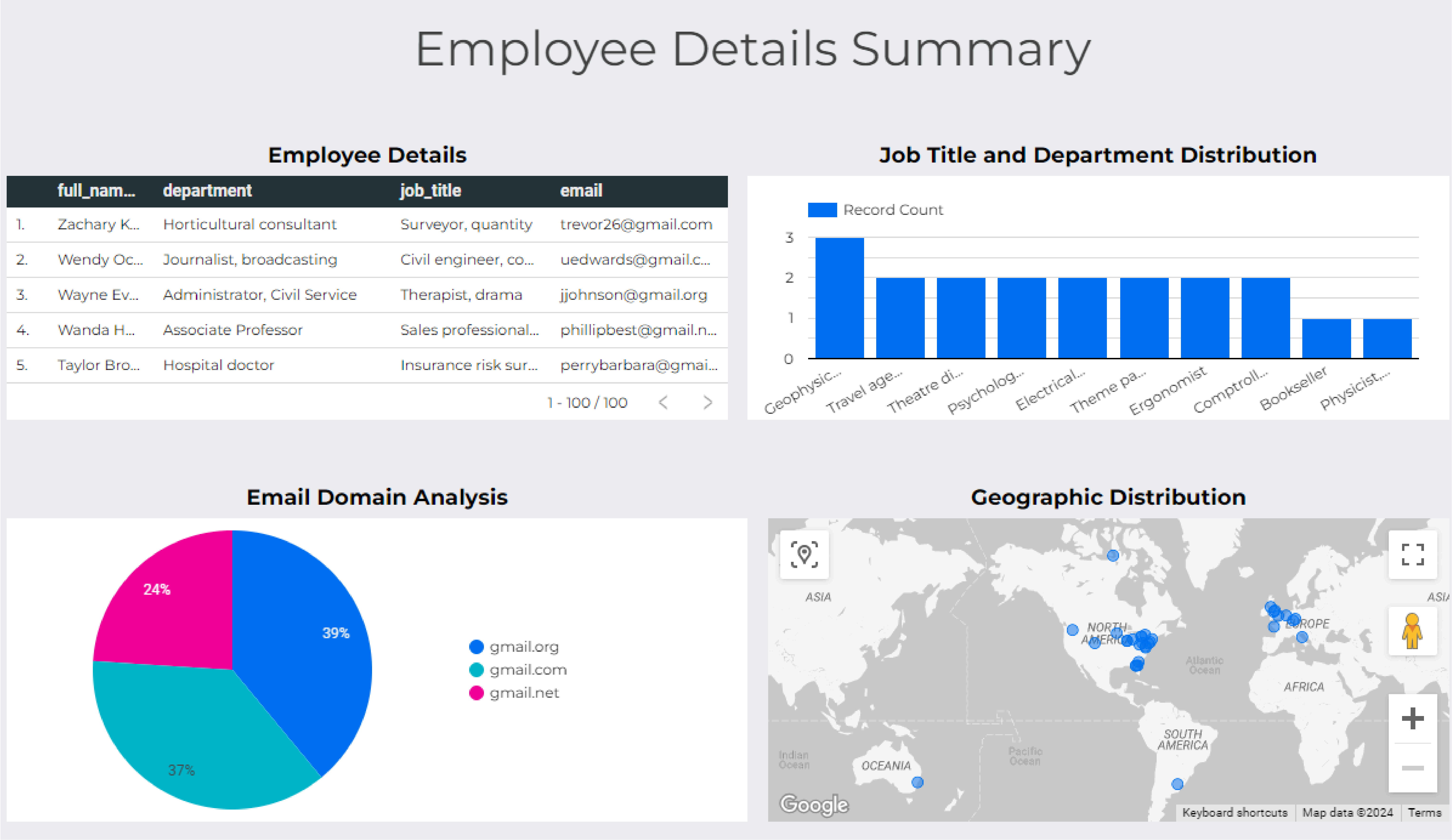
1. We replace the hostname to gmail.com from example.com which python faker library have created
2. We masked the salary column
3. We used hashing in password column to hide the real password.
4. At last, we combined first and last name to the full name

Data Visualization using Looker Studio

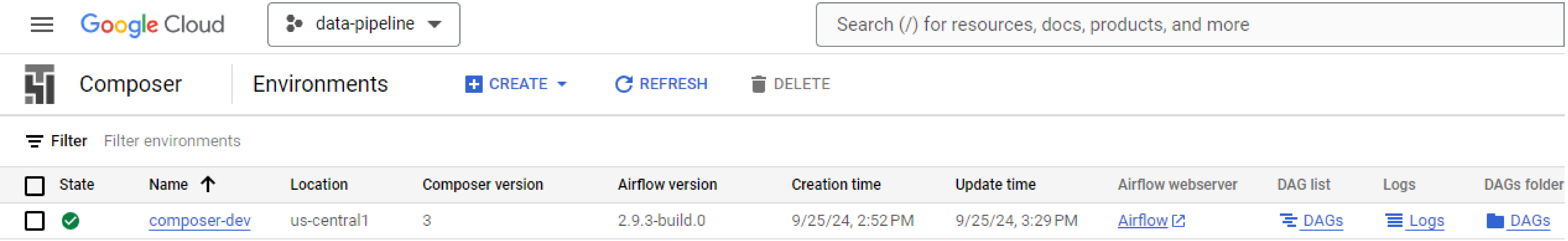


After validating the transformation we imported data into looker studio for visualization.

Visualization



Cloud Composer and Dag



Google Cloud Composer is a fully managed workflow orchestration service built on Apache Airflow, designed to automate and manage complex data pipelines. Within Composer, a DAG (Directed Acyclic Graph) defines the structure and sequence of tasks that constitute a data pipeline. The Composer DAG orchestrates the execution of various tasks, such as data extraction, transformation, loading, and triggering other services like Google Cloud Data Fusion pipelines. By specifying dependencies and scheduling within the DAG, Composer ensures that tasks run in the correct order and at the appropriate times. This automation allows for reliable, repeatable, and scalable data workflows, minimizing manual intervention and reducing the potential for errors.

Composer DAGs are responsible for pipeline automation because they provide a programmable and flexible framework to manage task dependencies, handle retries and failures, monitor workflow execution, and integrate seamlessly with other cloud services, thereby streamlining the entire data processing lifecycle.

Dag Script

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

default_args = {
    'owner': 'airflow',
    'start_date': days_ago(1),  # Use a past date or `days_ago`
    'depends_on_past': False,
    'email': ['bidhanpanta123@gmail.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 a DataFusion pipeline',
          schedule_interval='@daily',
          catchup=False)

with dag:
    run_script_task = BashOperator(
        task_id='data_extraction',
        bash_command='python /home/airflow/gcs/dags/script/data_extraction.py',
    )

    start_pipeline = CloudDataFusionStartPipelineOperator(
    location="europe-west2",
    pipeline_name="employee-pipeline",
    instance_name="data-fusion-dev",
    task_id="start_pipeline",
    )

    run_script_task >> start_pipeline
```

Created DAG script:

Importing Libraries:

- The script imports necessary libraries from datetime, airflow, and airflow.operators to define the DAG and handle tasks.
- The BashOperator is used to run external scripts, and CloudDataFusionStartPipelineOperator is used to trigger Google Cloud Data Fusion pipelines.

Default Arguments (default_args):

- Defines standard configurations for the DAG, such as the owner (airflow), the start date (set to a day ago), email notifications (disabled), retry attempts (1 retry with a 5-minute delay), and preventing the DAG from depending on past runs.

DAG Creation:

- The DAG is named 'employee_data' and is scheduled to run daily (@daily) without backfilling past runs (catchup=False).

Task 1: Running an External Python Script:

- The first task, run_script_task, uses the BashOperator to execute a Python script (data_extraction.py) located in the DAGs folder. This script presumably handles the employee data extraction process.

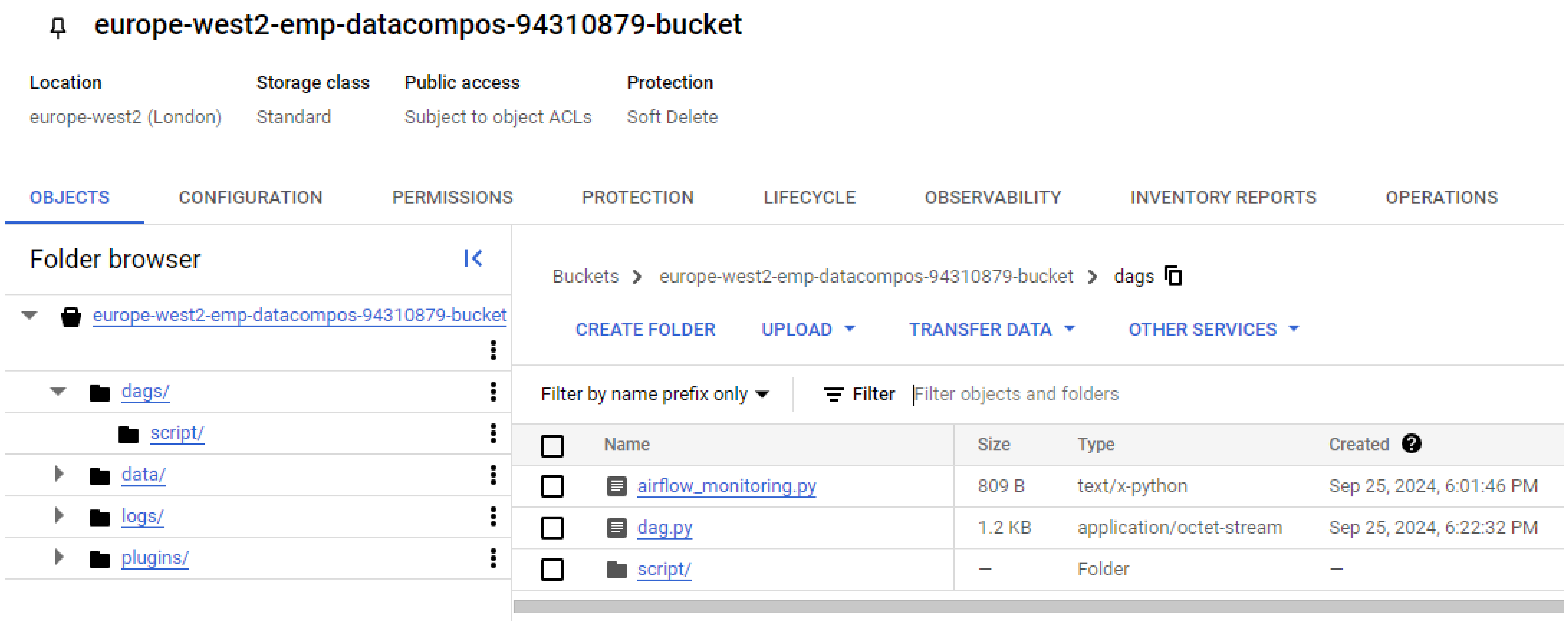
Task 2: Starting a Data Fusion Pipeline:

- The second task, start_pipeline, uses CloudDataFusionStartPipelineOperator to trigger a Google Cloud Data Fusion pipeline named "employee-pipeline". The pipeline runs in the region europe-west2 within the Data Fusion instance "data-fusion-dev".

Task Dependency:

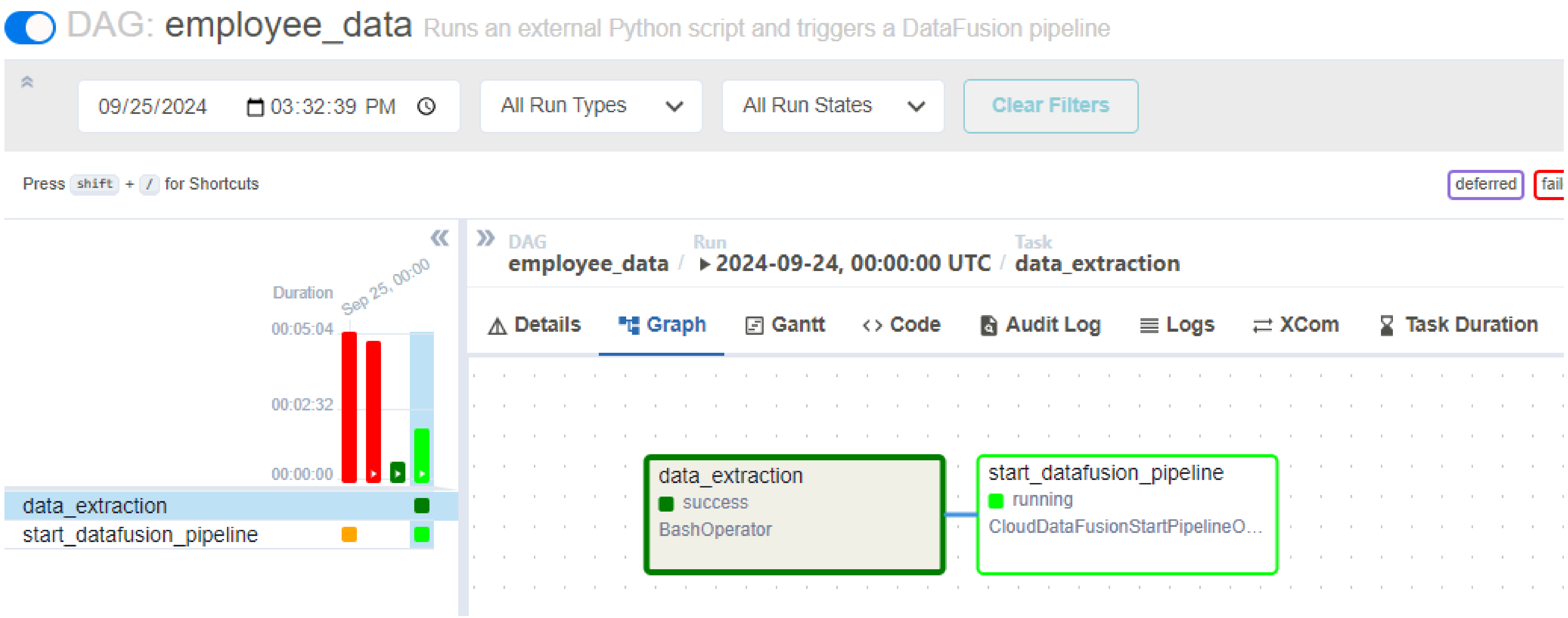
- The script ensures that the Python data extraction script runs first, and upon its successful completion, the Data Fusion pipeline is triggered (run_script_task >> start_pipeline).

Cloud Composer and Dag



After creating DAG we upload it to composer dag folder ***'dag.py'*** and we upload employee python code inside the script folder then it creates the automation pipeline inside the apache airflow ***'employee_data'*** where we can monitor our pipeline based on scheduled time in our case it is triggered daily.

Apache Airflow



Apache Airflow is an open-source platform used to create, schedule, and monitor complex workflows or data pipelines. It enables users to define workflows as code using Directed Acyclic Graphs (DAGs), which specify the sequence and dependencies of tasks. It also offers built-in features for scheduling, error handling, logging, and monitoring through an intuitive web interface. By automating data processing, machine learning, and other repetitive tasks, Airflow helps streamline operations and ensures scalable, reliable workflow management. In above figure we can see successfully transmission of data to the bucket and then to the bigquery and then to the looker studio through pipeline automation.

Thank You

By: Bidhan Pant