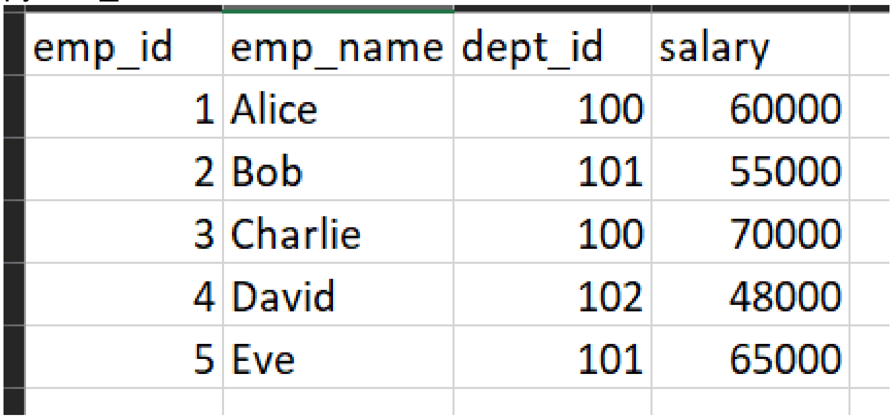
# Employees Batch Data Processing

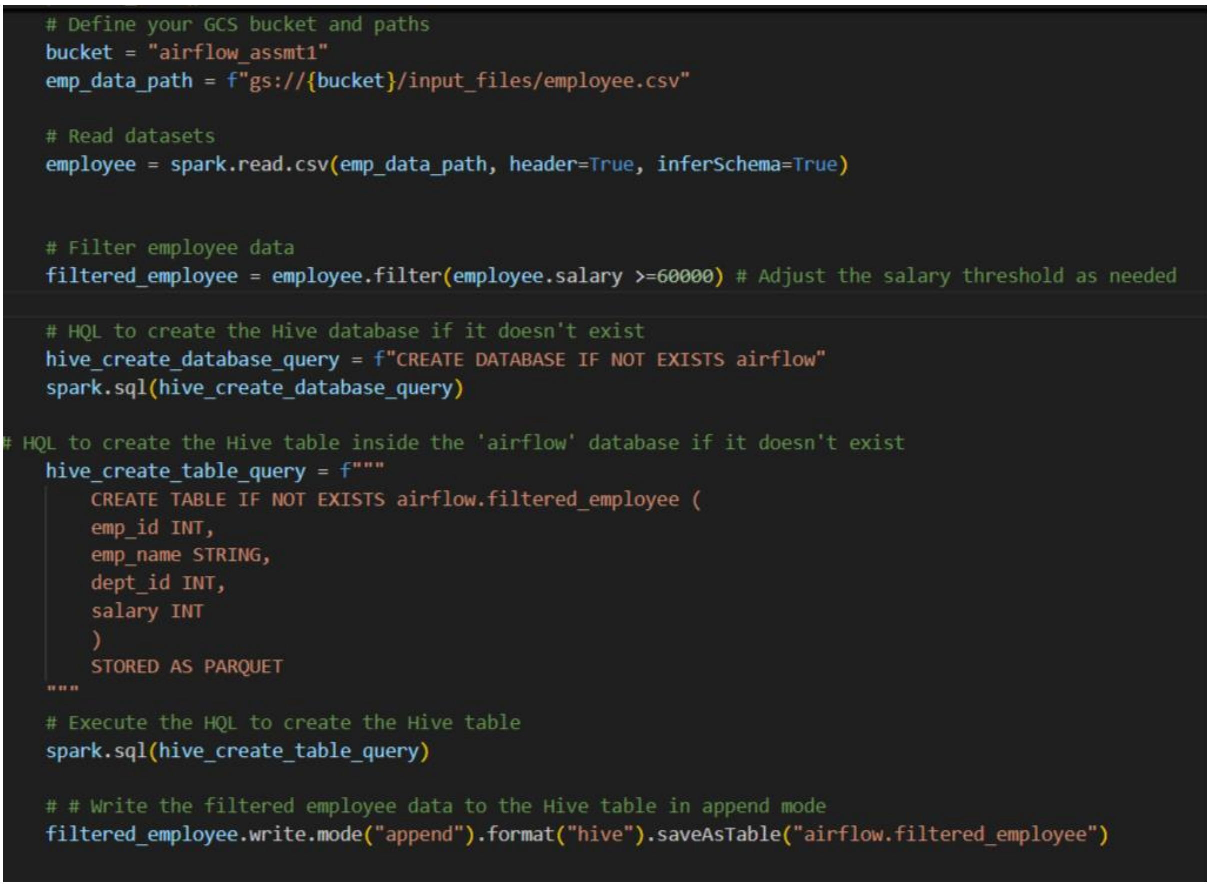
1. automated a workflow using Apache Airflow to process daily incoming CSV files from a GCP bucket using a Dataproc PySpark job and saving the transformed data into a Hive table.

2. Used GCP services- DataProc/GCS and Airflow to schedule the dag job.

3. created a bucket called 'airflow_assmt1' and placed employee.csv under input_files folder. This file will be picked up by the spark job which is a part of the DAG. Also made sure to place the pyspark job 'employee_batch.py' in the python_file folder in the same GCS bucket.

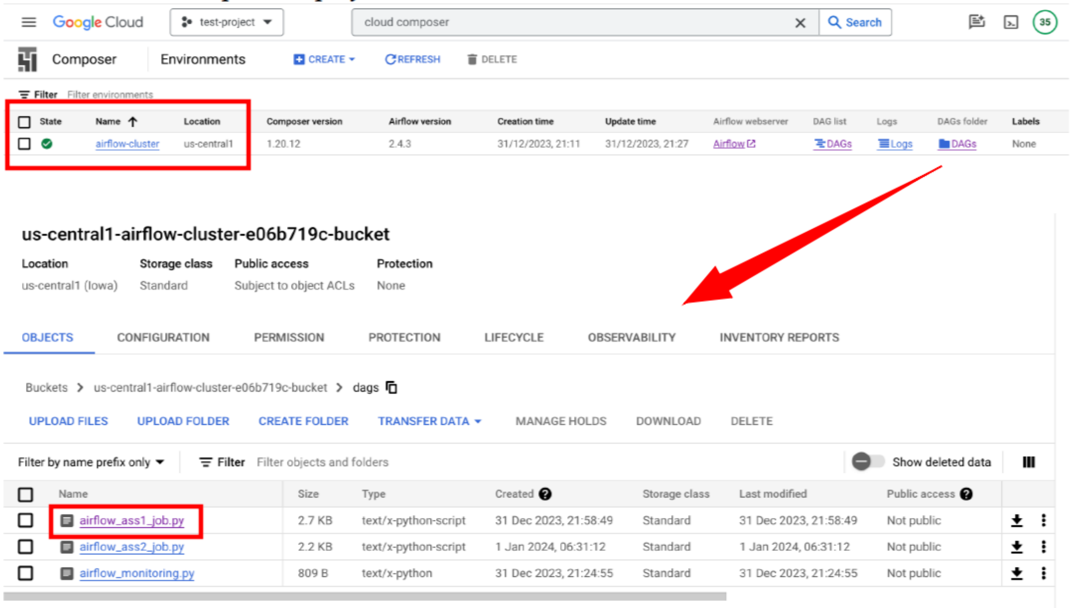


| emp_id | emp_name | dept_id | salary |
|---|---|---|---|
| 1 | Alice | 100 | 60000 |
| 2 | Bob | 101 | 55000 |
| 3 | Charlie | 100 | 70000 |
| 4 | David | 102 | 48000 |
| 5 | Eve | 101 | 65000 |



```
    # Define your GCS bucket and paths
    bucket = "airflow_assmt1"
    emp_data_path = f"gs://{bucket}/input_files/employee.csv"

    # Read datasets
    employee = spark.read.csv(emp_data_path, header=True, inferSchema=True)


    # Filter employee data
    filtered_employee = employee.filter(employee.salary >=60000) # Adjust the salary threshold as needed

    # HQL to create the Hive database if it doesn't exist
    hive_create_database_query = f"CREATE DATABASE IF NOT EXISTS airflow"
    spark.sql(hive_create_database_query)

# HQL to create the Hive table inside the 'airflow' database if it doesn't exist
    hive_create_table_query = f"""
        CREATE TABLE IF NOT EXISTS airflow.filtered_employee (
        emp_id INT,
        emp_name STRING,
        dept_id INT,
        salary INT
        )
        STORED AS PARQUET
    """

    # Execute the HQL to create the Hive table
    spark.sql(hive_create_table_query)

    # # Write the filtered employee data to the Hive table in append mode
    filtered_employee.write.mode("append").format("hive").saveAsTable("airflow.filtered_employee")
```

4. I created an airflow cluster and then proceeded to place the ‘airflow_ass1_job.py’ file in the DAG list so that it can be picked up by airflow.



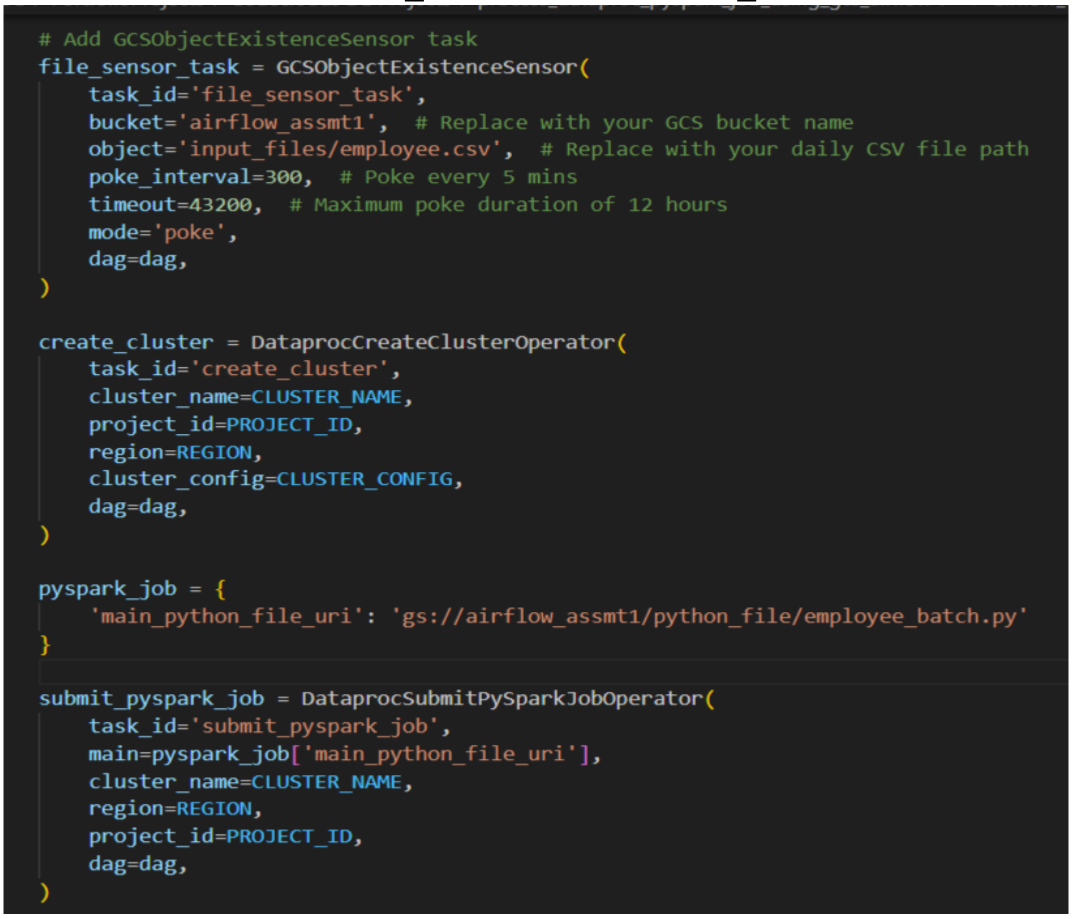
| State | Name | Location | Composer version | Airflow version | Creation time | Update time | Airflow webserver | DAG list | Logs | DAGs folder | Labels |
|---|---|---|---|---|---|---|---|---|---|---|---|
| ✓ | airflow-cluster | us-central1 | 1.20.12 | 2.4.3 | 31/12/2023, 21:11 | 31/12/2023, 21:27 | Airflow | DAGs | Logs | DAGs | None |

us-central1-airflow-cluster-e06b719c-bucket

| Location | Storage class | Public access | Protection |
|---|---|---|---|
| us-central1 (Iowa) | Standard | Subject to object ACLs | None |

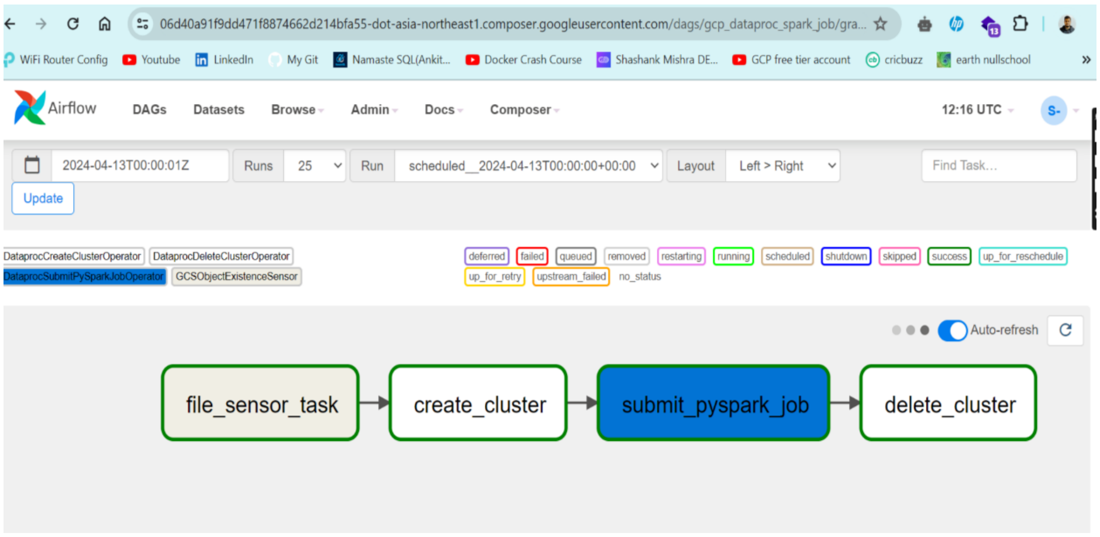
| Name | Size | Type | Created | Storage class | Last modified | Public access |
|---|---|---|---|---|---|---|
| airflow_ass1_job.py | 2.7 KB | text/x-python-script | 31 Dec 2023, 21:58:49 | Standard | 31 Dec 2023, 21:58:49 | Not public |
| airflow_ass2_job.py | 2.2 KB | text/x-python-script | 1 Jan 2024, 06:31:12 | Standard | 1 Jan 2024, 06:31:12 | Not public |
| airflow_monitoring.py | 809 B | text/x-python | 31 Dec 2023, 21:24:55 | Standard | 31 Dec 2023, 21:24:55 | Not public |

5. Now the various stages can be seen in the DAG. A file sensor checks every 5 mins in the input_file location, once it detects employee.csv a new cluster was created followed by which the spark job was launched which then filters employees with salary >=60,000 and then placing the output in another GCS location in the bucket airflow_assmt1 under the hive_data folder.

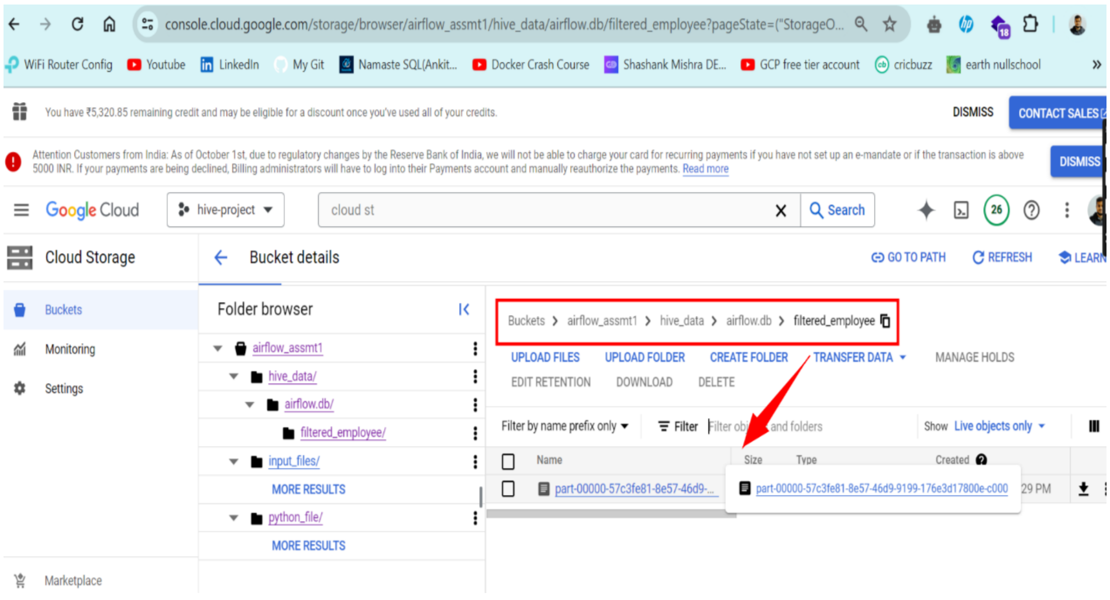


```python
# Add GCSObjectExistenceSensor task
file_sensor_task = GCSObjectExistenceSensor(
    task_id='file_sensor_task',
    bucket='airflow_assmt1',  # Replace with your GCS bucket name
    object='input_files/employee.csv',  # Replace with your daily CSV file path
    poke_interval=300,  # Poke every 5 mins
    timeout=43200,  # Maximum poke duration of 12 hours
    mode='poke',
    dag=dag,
)

create_cluster = DataprocCreateClusterOperator(
    task_id='create_cluster',
    cluster_name=CLUSTER_NAME,
    project_id=PROJECT_ID,
    region=REGION,
    cluster_config=CLUSTER_CONFIG,
    dag=dag,
)

pyspark_job = {
    'main_python_file_uri': 'gs://airflow_assmt1/python_file/employee_batch.py'
}

submit_pyspark_job = DataprocSubmitPySparkJobOperator(
    task_id='submit_pyspark_job',
    main=pyspark_job['main_python_file_uri'],
    cluster_name=CLUSTER_NAME,
    region=REGION,
    project_id=PROJECT_ID,
    dag=dag,
)
```

6. Then we were able to see that all stages have run successfully followed by deletion of the cluster as well.



7. We can see the resultant data saved in a parquet file format saved in hive_data folder (over which we were able to build external hive tables to query the data)


You have ₹5,320.85 remaining credit and may be eligible for a discount once you've used all of your credits.

Attention Customers from India: As of October 1st, due to regulatory changes by the Reserve Bank of India, we will not be able to charge your card for recurring payments if you have not set up an e-mandate or if the transaction is above 5000 INR. If your payments are being declined, Billing administrators will have to log into their Payments account and manually reauthorize the payments. Read more

Buckets > airflow_assmt1 > hive_data > airflow.db > filtered_employee

part-00000-57c3fe81-8e57-46d9-9199-176e3d17800e-c000

Challenges:
1. Issues with the spark job being picked up since we had to define a location to save the output.
2. We can't put the output on the local of the cluster since with the deletion phase of the cluster this data would disappear as well. Hence it made sense to place the output in a GCS bucket