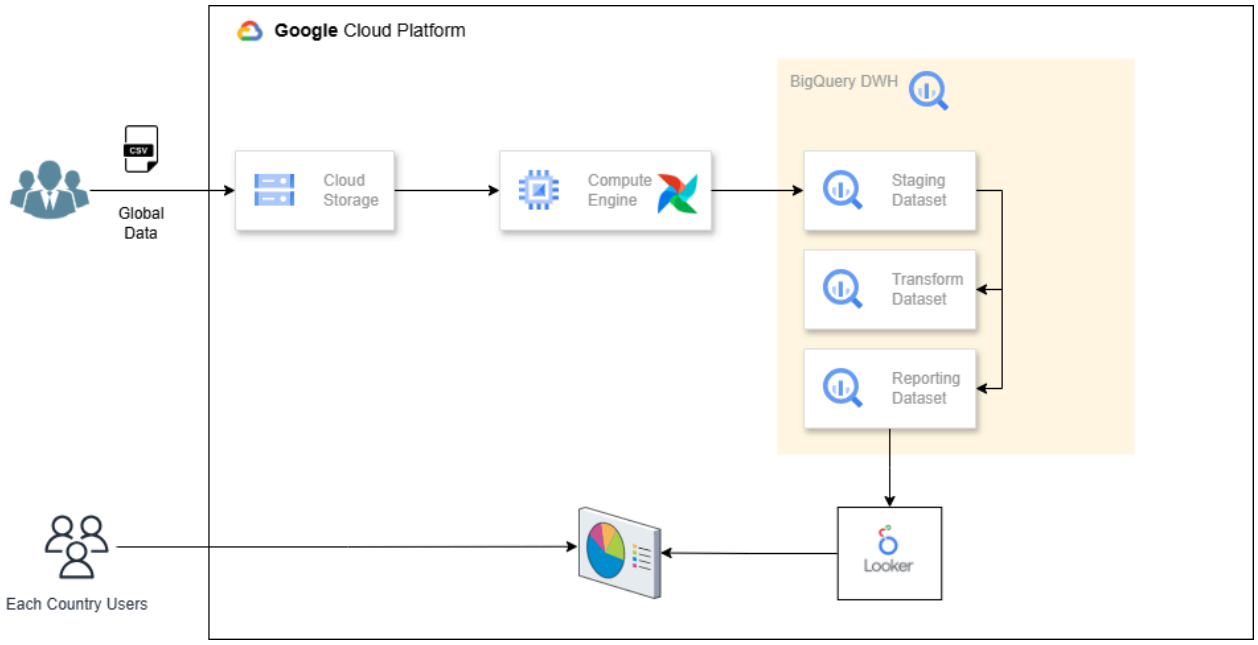
Setup Process: Airflow, Google Cloud, Bigquery

Architecture of this project is as below:



Listed below the entire step-by-step process that I followed in order to complete this project:

– Open Google Cloud Console, Create a new project. Go to IAM & Admin; grant Owner Role to the project. Next, go to APIs & Services→Credentials; If you don't have a service account, click on the create credentials and create a new service account. If you already have a service account listed in the APIs & Services→Credentials; open it and go to Keys→Add key→Create new JSON key. This will download the key to your system downloads folder.

– Go to Compute Engine, Create a VM Instance (In Machine Configuration - Default; OS and Storage - Default; Data Protection - Default; Networking - Allow HTTP Traffic in Firewall; Observability - Default; Security - Allow Full Access To All Cloud APIs in Access Scopes) — Done. Next, go to the IAM & Admin; you can see the Compute Engine Service Account. Grant Owner Role to it.

– After the VM is Up and Ready, Click the SSH. It should run. If not, Open the Network interface details and check in the Default network→Firewalls→vpc-firewall-rules→**default-allow-ssh** (this allow-ssh rule should be present. If not add this firewall rule). This should run the SSH.

– In the SSH, Run the below commands one by one:-

→ sudo apt install python3 python3-pip python3-venv

→ python3 -m venv ~/airflow-env

→ source airflow-env/bin/activate (this will start your airflow virtual environment)
→ pip3 install apache-airflow[gcp]
→ nohup airflow standalone > airflow.log (this will start the airflow)
****

→ pip install apache-airflow-providers-google (Run this if you are getting Import Error (ModuleNotFoundError: No module named 'airflow.providers.google') while importing the DAG that is created in the SSH into the airflow)

→ pip install "apache-airflow-providers-google==10.12.0" --force-reinstall (Use this to Downgrade to a stable version of the Google provider if you are getting serialization error like “TypeError: cannot serialize object of type <class 'type'>” when your triggered DAG is failed. Then restart the airflow using nohup command and try triggering the DAG again. This might resolve the serialization error.)

→ If you are encountering import error (AttributeError: cffi library '_openssl' has no function, constant or global variable named 'Cryptography_HAS_MEM_FUNCTIONS') while importing the DAG to airflow; it is related to a compatibility issue between the cryptography library and your version of OpenSSL or cffi. For that, run (source ~/airflow_venv/bin/activate), next (pip uninstall cryptography cffi pyopenssl -y), next (pip install cryptography==41.0.3 cffi==1.15.1 pyOpenSSL==23.2.0), You can also run if needed (pip install --upgrade pip setuptools wheel) and restart the airflow using nohup command and reload the airflow. This should remove the import error and successfully import the DAG to airflow.
****

– Now, go to google cloud instance, copy the External IP of your instance, open a new tab and enter http://*paste_your_external_ip_here*:8080 (this will open the airflow). If it doesn't, go to Default Network→Firewalls→vpc-firewall-rules→default-allow-http (edit the TCP Port and enter 8080 and save it). This should open the airflow with the external ip now.

– Go to SSH; Type and run (ls), then (cd airflow), then (ls) now you can see the password file that is automatically generated and saved in this directory. Type and run (cat password_file_name), you can see the user ID and password to login to airflow. Save it in your notepad. Use this user ID and password to login to airflow.

– To remove the example DAGs in airflow; In SSH, go to the airflow directory. In that, there will be an airflow.cfg file. Open this airflow configuration file using the command (nano airflow.cfg), and set the (load_examples = False) and save it. Now restart the airflow using the nohup command and reload the airflow page again. This should remove all the example DAGs.

– IN SSH, Click on Upload file; then upload the downloaded JSON key. This key will be stored in the base directory. You can type and run “cd”; it lets you go to the main/base directory, then “ls” to list all the folders/files like airflow, airflow-env, airflow.log, and also your uploaded JSON key file; Create a new directory with the name “keys” by using (mkdir -p ~/keys) then move the uploaded JSON key to the keys folder and rename the JSON key file with your desired name by

using (mv ~/*your_uploaded_json_file_name.json* ~/keys/*new_name.json*). Then activate the airflow using “source ~/airflow_venv/bin/activate”. Next run the below code:

****

```
airflow connections add 'google_cloud_default' \
  --conn-type 'google_cloud_platform' \
  --conn-extra '{
    "extra__google_cloud_platform__key_path": "/home/venuanupati/keys/my-gcp-key.json",
    "extra__google_cloud_platform__project": "your-gcp-project-id"
  }'
```

****

Replace "your-gcp-project-id" with your actual GCP project ID. You can find your actual GCP project ID by running the command “gcloud config get-value project”.

Lastly, verify if the connection was added by running “airflow connections get google_cloud_default”.

If the project value is still set to the placeholder like "extra__google_cloud_platform__project": "your-gcp-project-id" then run the below command and next verify the connection again.

****

```
airflow connections delete google_cloud_default
airflow connections add 'google_cloud_default' \
  --conn-type 'google_cloud_platform' \
  --conn-extra '{
    "extra__google_cloud_platform__key_path": "/home/venuanupati/keys/my-gcp-key.json",
    "extra__google_cloud_platform__project": "airflowand-bigquery"
  }'
```

****

If you missed the entire step that is above mentioned, you might get an error (AirflowNotFoundException: The conn_id `google_cloud_default` isn't defined) when your triggered DAG fails.

– Go to Google Cloud and create a Storage Bucket using “Cloud Storage→Create Bucket”; give a name to the bucket and click create. (*Now our target is to load the data into bigquery i.e. pick the data from the CSV file that we will upload here in this bucket and load into bigquery*)

– Go to BigQuery and under the current project; Create a new dataset and name it as staging_dataset. (*So, this dataset will be used as the destination for us to store the data that we will pick from the cloud bucket and store it here in this dataset*)

– In SSH, go to airflow directory, create a new directory called “dags” using (mkdir dags) and in that dags directory, create a new file and name it dag1.py using (nano dag1.py)

– Now open the VS Code or Jupiter notebook or any other python coding environment to build our first DAG. (For this, I have created a gcstobq.py file. In this code, under the DAG definition; I have given the dag_id. Under the Task; I have given a task_id, cloud storage bucket name that we have created, original file name that we are going to upload it into the bucket, destination as the data set ID that we have created in the bigquery plus the table name *(I have given as "airflowand-bigquery.staging_dataset.raw_global_data". If the table name is not given, it will automatically create a table. But in my case, I have given the table name i.e. raw_global_data*) So, this DAG only load data from the CSV file that is in the cloud bucket to bigquery)

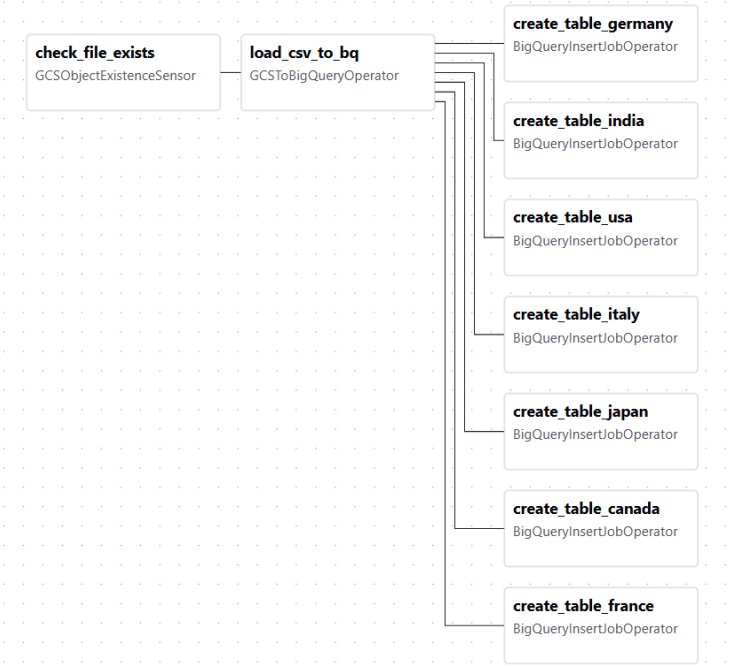
– In the SSH, go to the dags folder that we have created; Open the dag1.py file using "nano dag1.py" and paste the code that we have developed in the VS Code and save it. Go to the airflow page and we can see our DAG will be successfully imported there if there are no issues. Now Trigger the DAG, this should run and fail by throwing the following error (ERROR - Task failed with exception: source="task", NotFound: 404 Not found: URI gs://global_health_bkt/global_health_data.csv). This is expected, because we still have not uploaded the CSV file into the bucket.

– Next, our task is to make the DAG not fail if there is no file in the bucket. The DAG should wait for the file until it is uploaded/exist in the bucket without getting failed. For that we have to create one sensor/task before the load data from bucket to bigquery task; so that it will check for the existence of our file in the bucket. If the file is present in the bucket, then only it will allow the load data from the bucket to bigquery task to be executed and start loading the data. For this, I have built a code with the file name "gcstobq_sensor.py" in the VS Code. In this code, in addition to the previous code, I have added another task to check the existence of the CSV file object in the bucket, then only it will allow the execution of the load to bigquery task. Otherwise, it will wait until the file is loaded/exists in the bucket. So there are two tasks in this DAG now i.e. to check if the file exists, and then to load the data into bigquery. In the SSH, create a new file with the name "dag2.py" in the dags folder. Open it and paste the code there and save. Now reload the airflow tab to see the new DAG imported over there. Open to see the graph that there are two tasks in it as shown in the below image. Trigger this DAG and it will run as per the tasks created.

– Now Upload the CSV file in the bucket and trigger the DAG. This will successfully check the existence of the file in the bucket and run the next task to save the data into the staging_dataset.

– Created a new dataset with the name "transformed_dataset". Also, I created another code with the name "transformation.py" and placed this code in the dags folder with the name "dag3.py". In this code, in addition to the previous two tasks related code, I have added another task to create country specific tables. I have picked 7 countries, and made sure that the data

related to those countries are picked and saved into the “transformed_dataset” as a separate table for each country. The graph in the airflow for this DAG looks as in the below image.



– Building a report on the complete table will involve in lot of cost and time. So, to reduce the complexity, I have picked the first 5 columns for the report by creating Views on top of it so that I can have only the data that is required for the looker report. I have created another dataset with the name “reporting_dataset” in order to only have the required data for the reporting purpose. I have created a new file with the name “final_view.py”. In addition to the previous code, I have added two more tasks i.e. one task will create the views with only the selected 5 columns by only getting the data for which the Vaccines Treatment = ‘false’. And the other task is a dummy task which will check if all the tasks are completed and it will just mark the task and the DAG as complete. Next, I have created a new DAG file with the name “dag4.py” in the dags folder and saved the code in it and triggered this DAG in the airflow. The graph from this DAG is shown in the below image. It successfully ran and created views as expected.

– Next, I have connected this BigQuery to the looker studio and created a basic report by selecting the India_table view. Below is the snapshot of the report.