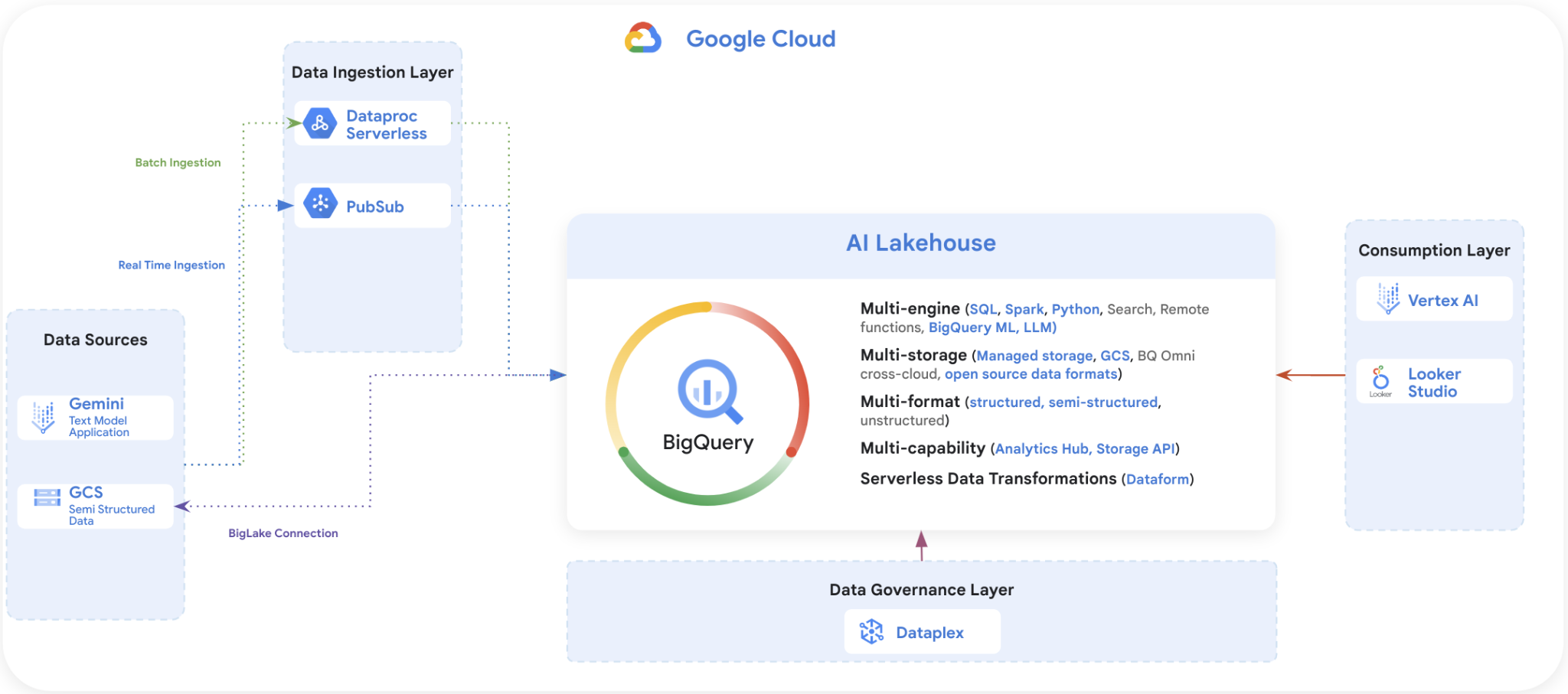
**Data & AI Boot-Kon Event**

| **Title: Data Ingestion with Dataproc , BigLake & PubSub** | | **Goal of the lab**   * Ingest fraudulent and non fraudulent transactions dataset into BigQuery using three methods: * BigLake with data stored in Google Cloud Storage (GCS) * Batch Ingestion into BigQuery using Dataproc Serverless * Near real-time ingestion into BigQuery using Cloud PubSub |
| --- | --- | --- |
| **Author**: Wissem Khlifi | **Date**: 2024-04-01 | **Estimated Completion Time**: 45 Minutes |
|  | | |

CAUTION:

This lab is for educational purposes only and should be used with caution in production environments. Google Cloud Platform (GCP) products are changing frequently, and screenshots and instructions might become inaccurate over time. Always refer to the latest GCP documentation for the most up-to-date information.

# **Architecture Diagram:**



# **[LAB] BigLake Object Tables**

<https://cloud.google.com/bigquery/docs/biglake-intro>

### **Enable API (you can skip this step if you completed LAB 1)**

Ensure the necessary APIs (BigQuery API, Vertex AI API, Pubsub API, BigQuery Connection API, Dataform API, Secret Manager API) are [enabled](https://console.cloud.google.com/flows/enableapi?apiid=storage-component.googleapis.com%2Cpubsub.googleapis.com%2Cbigquery.googleapis.com%2Cbigqueryconnection.googleapis.com%2Caiplatform.googleapis.com&_ga=2.132962701.243207769.1688884437-279425947.1688884437)

### **Required roles (you can skip this step if you completed LAB 1)**

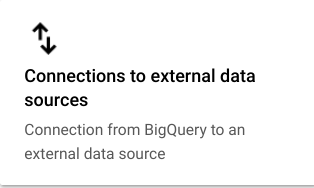
To create a BigLake table, you need the following BigQuery Identity and Access Management (IAM) permissions:

* bigquery.tables.create
* bigquery.connections.delegate

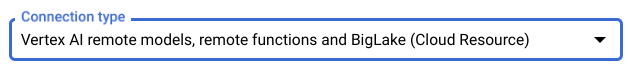
The BigQuery Data Editor and connection admin predefined Identity and Access Management role includes these permissions. If you are not a principal in this role, ask your administrator to grant you access or to create the BigLake table for you.

### **Add BigLake connection**

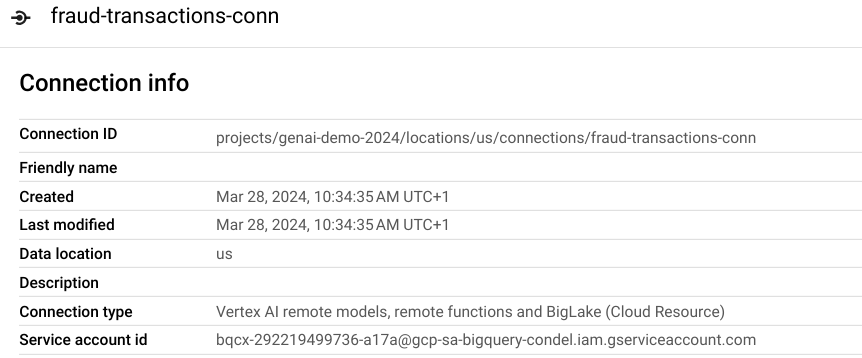
1. From Bigquery Studio, Click “+ ADD” to add data.
2. Choose “Connections to external data sources”



1. Select the Vertex AI models , remote functions and BigLake connection type.

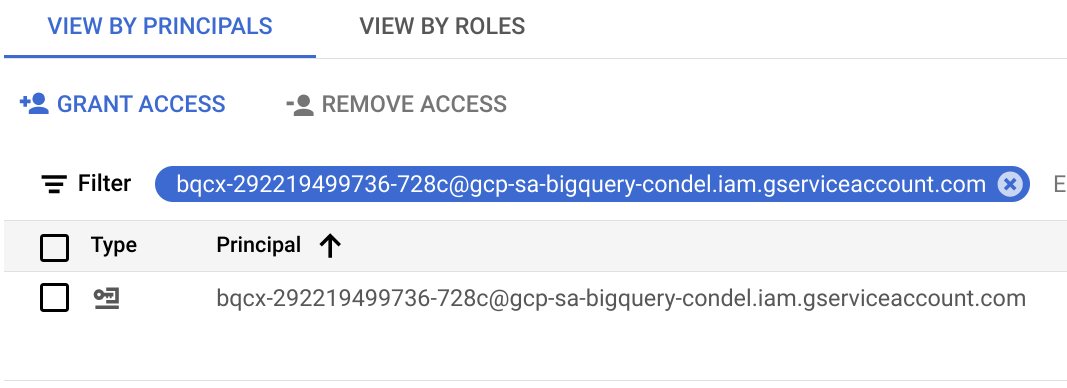


1. Enter the connection ID as “fraud-transactions-conn” and choose the US multi-region location.
2. Click “CREATE CONNECTION”
3. Open the connection details for the newly added external connection “fraud-transactions-conn”

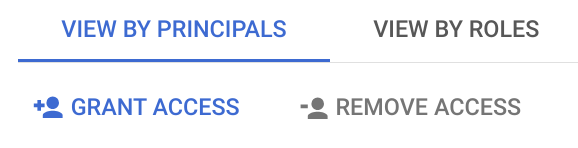


1. Note the service account ID and grant it Storage Object Viewer role:.

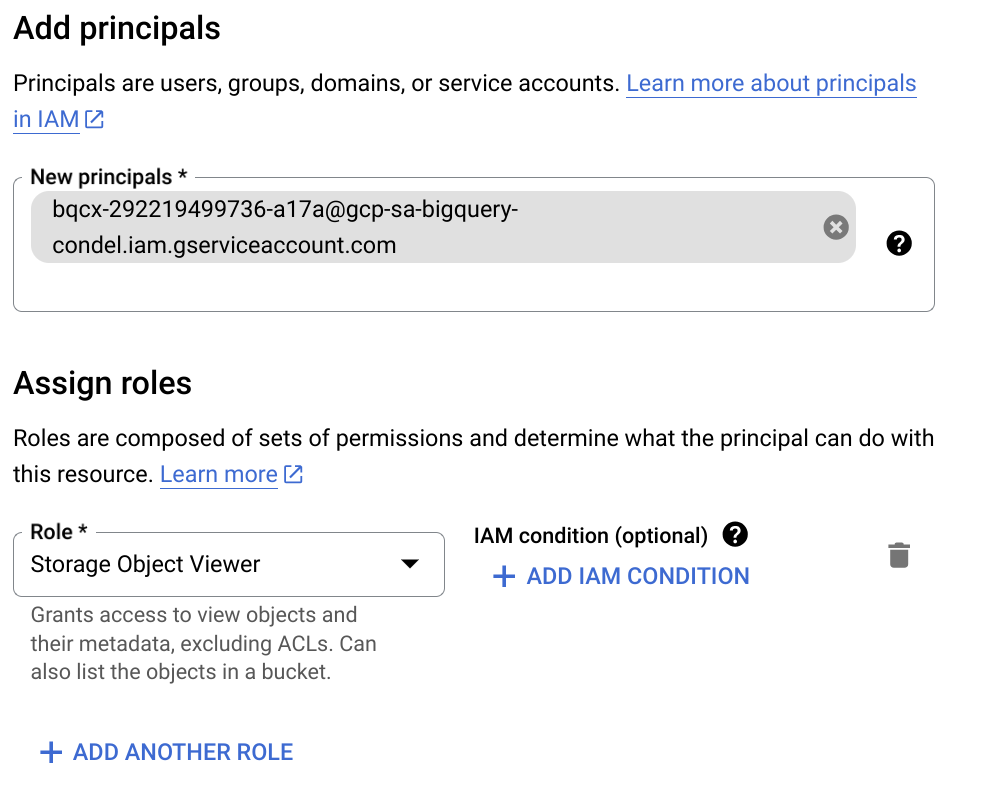
* Go to IAM & Admin
* Filter the Service account



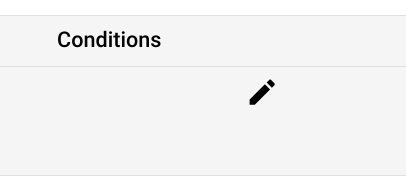
* If you can’t find the service account ID, add it as a principal: Follow the steps below.
  + First click on “Grant Access”



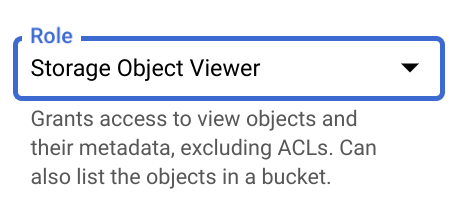
* + Copy and paste the service account id into the “new principals” input box. (make sure you paste the complete service account email). Add the Storage Object Viewer Role and click on Save.



* If you find the service account id, then click on Edit Principal



* Click on Add Role and add the Storage Object Viewer role, then click on Save.



1. From Cloud Shell, create a dataset named ml\_datasets in the US multi-region.

| ***Linux command line : Create a BigQuery dataset: call it ml\_datasets in US multi region*** |
| --- |
| *DATASET\_NAME="ml\_datasets"*  *bq --location=US mk -d \*  *--description "Fraudulent and Non Fraudulent transactions BigQuery dataset" \*  *$DATASET\_NAME* |

>>> you can ignore the warning ; warnings.warn("urllib3 ({}) or chardet ({})/charset\_normalizer ({}) doesn't match a supported "

### Go to BigQuery , check that the dataset has been created successfully (you may need to click “refresh contents” from the 3-dot menu for the project in the Explorer).

### 

### Click on the “+” icon on the right end of the tabs in the workspace to open a new SQL Query;



### Create BigLake tables on non partitioned parquet data on GCS;

*Copy the following commands and click on *

* *Replace your-project-id with your current project ID at 3 places!*

| ***BigQuery SQL : Create BigLake table on non partitioned parquet data on GCS*** |
| --- |
| *CREATE OR REPLACE EXTERNAL TABLE*  *`your-project-id.ml\_datasets.ulb\_fraud\_detection\_blake`*  *WITH CONNECTION `us.fraud-transactions-conn` OPTIONS (*  *format ="PARQUET",*  *uris = ['gs://your-project-id-bucket/data-ingestion/parquet/ulb\_fraud\_detection/\*'],*  *max\_staleness=INTERVAL 30 MINUTE,*  *metadata\_cache\_mode="AUTOMATIC");*  *# check the results from BigQuery*  *SELECT \* FROM `your-project-id.ml\_datasets.ulb\_fraud\_detection\_blake` LIMIT 1000;* |

# **[LAB] Batch data ingestion into BigQuery using Dataproc**

### **Serverless Dataproc**

Importing data from Google Cloud Storage (GCS) in Parquet format into BigQuery using PySpark requires a sequence of steps, including setup and code implementation. Here’s a comprehensive guide:

Running PySpark jobs on Google Cloud Dataproc Serverless simplifies the process by abstracting the management of the Spark environment. Here’s how to set everything up to import Parquet files from Google Cloud Storage (GCS) to BigQuery using Dataproc Serverless.

#### **Prerequisites: (you can skip this step if you completed LAB 1)**

* Google Cloud Storage (GCS): Ensure your Parquet files are stored in a GCS bucket.
* BigQuery Dataset & Table: Create or have a BigQuery dataset and table where you want to load the data.
* Google Cloud SDK: Install the Google Cloud SDK if you intend to use the command line. For simplicity we will use Cloud shell.
* Enable APIs: Enable the BigQuery, Cloud Storage, and Dataproc Serverless APIs for your project.
* Download BigQuery Connector for Spark: To read from BigQuery, you'll need the BigQuery Connector for Spark. This should be included in your Spark job's dependencies.
  + Download Spark 3.5 Jar version : <https://github.com/GoogleCloudDataproc/spark-bigquery-connector>
  + Upload to GCS: *gs://${BUCKET\_NAME}*/jar/

#### **IAM Roles: (you can skip this step if you completed LAB 1)**

Ensure your Google Cloud user account or service account has the following roles:

* Dataproc Editor (roles/dataproc.editor): Allows for the creation and running of Dataproc Serverless jobs.
* BigQuery Data Editor (roles/bigquery.dataEditor): Allows for creating tables and inserting data into BigQuery.
* Storage Object Viewer (roles/storage.objectViewer): Allows reading data from the specified GCS bucket.

#### **Set Up a Dataproc Serverless Spark Batch:**

1. Create an empty BigQuery table ;

*Replace your-project-id with your current project ID, and run this query:*

| ***BigQuery SQL : Create BigQuery table for parquet data*** |
| --- |
| *CREATE OR REPLACE TABLE `your-project-id.ml\_datasets.ulb\_fraud\_detection\_parquet` AS*  *SELECT \* from `your-project-id.ml\_datasets.ulb\_fraud\_detection\_blake` where 1=2;* |

1. Download and upload parquet files to GCS **(you can skip this step if you completed LAB 1).**
2. In Cloud shell, go to directory ; **cd *$HOME/bootkon-h2-2024/data-ingestion/src***
3. Prepare the **import\_parquet\_to\_bigquery.py** PySpark Script:

Adapt your script to read from GCS and write to BigQuery. Ensure you specify the GCS path to your Parquet files and the target BigQuery table.

* *Replace your-project-id with your current project ID (in 3 locations)*

| ***PySpark Script : Import parquet data into BigQuery table.*** |
| --- |
| *from pyspark.sql import SparkSession*  *# GCP Project ID*  *project\_id = "****your-project-id****"*  *# GCS Path to Parquet Files*  *gcs\_parquet\_path = "gs://****your-project-id****-bucket/data-ingestion/parquet/ulb\_fraud\_detection/"*  *# BigQuery Dataset and Table Name*  *bq\_dataset\_name = "ml\_datasets"*  *bq\_table\_name = "ulb\_fraud\_detection\_parquet"*  *# GCS Bucket for temporary storage*  *temporary\_gcs\_bucket = "****your-project-id****-bucket"*  *# Create a SparkSession*  *spark = SparkSession.builder\*  *.appName("bigquery\_to\_gcs\_parquet")\*  *.getOrCreate()*  *# Read Parquet Files from GCS*  *df = spark.read.parquet(gcs\_parquet\_path)*  *# Write DataFrame to BigQuery*  *df.write.format("bigquery") \*  *.option("table", f"{project\_id}:{bq\_dataset\_name}.{bq\_table\_name}") \*  *.option("temporaryGcsBucket", temporary\_gcs\_bucket) \*  *.mode("overwrite") \*  *.save()*  *spark.stop()* |

1. Submit a Dataproc Serverless Batch Job:

Use the Google Cloud Console or the gcloud command-line tool to submit your PySpark job. Here is an example gcloud command (run it from cloud shell):

* *Replace your-project-id with your current project ID (in 3 locations) and your-region with your GCP region, for example ; us-central1*

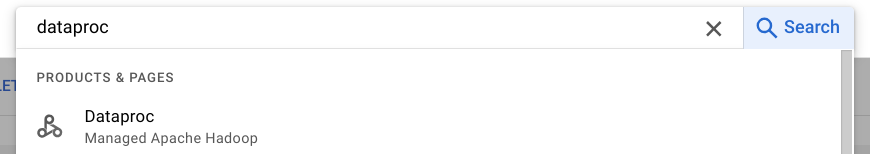
| ***Linux command line : Submit spark job to Dataproc*** |
| --- |
| *gcloud dataproc batches submit pyspark import\_parquet\_to\_bigquery.py \*  *--project=****your-project-id*** *\*  *--region=your-region \*  *--deps-bucket=gs://****your-project-id****-bucket \*  *--jars=gs://****your-project-id****-bucket/data-ingestion/jar/spark-3.3-bigquery-0.37.0.jar* |

This command specifies:

* The location of your PySpark script in GCS.
* The project and region to run in.
* A GCS bucket (--deps-bucket) for Dataproc to use for job dependencies.
* A JAR file (--jars) containing the BigQuery connector for Spark.

1. Monitor the job execution progress in Dataproc;

From the search write : dataproc



1. Click on Batches and monitor the execution log from the console



1. Once the dataproc job finishes , check the results in BigQuery table

# **[LAB](Near-)Real time data ingestion into BigQuery using PUSUB**

### **Prerequisite : Enable API (you can skip this step if you completed LAB 1)**

* Make sure all the necessary APIs (BigQuery API, Vertex AI API, Pubsub API, BigQuery Connection API, Dataform API, Secret Manager API) are [enabled](https://console.cloud.google.com/flows/enableapi?apiid=storage-component.googleapis.com%2Cpubsub.googleapis.com%2Cbigquery.googleapis.com%2Cbigqueryconnection.googleapis.com%2Caiplatform.googleapis.com&_ga=2.132962701.243207769.1688884437-279425947.1688884437)

### ***Step 1: Create PUBSUB Topic with schema and BQ subscription***

1. Create an empty BQ table

| ***BigQuery SQL : Create BigQuery Table for streaming ingestion***   * *Replace your-project-id with your current project ID* |
| --- |
| *CREATE OR REPLACE TABLE `****your-project-id****.ml\_datasets.ulb\_fraud\_detection`*  *(*  *Time FLOAT64 ,*  *V1 FLOAT64 ,*  *V2 FLOAT64 ,*  *V3 FLOAT64 ,*  *V4 FLOAT64 ,*  *V5 FLOAT64 ,*  *V6 FLOAT64 ,*  *V7 FLOAT64 ,*  *V8 FLOAT64 ,*  *V9 FLOAT64 ,*  *V10 FLOAT64 ,*  *V11 FLOAT64 ,*  *V12 FLOAT64 ,*  *V13 FLOAT64 ,*  *V14 FLOAT64 ,*  *V15 FLOAT64 ,*  *V16 FLOAT64 ,*  *V17 FLOAT64 ,*  *V18 FLOAT64 ,*  *V19 FLOAT64 ,*  *V20 FLOAT64 ,*  *V21 FLOAT64 ,*  *V22 FLOAT64 ,*  *V23 FLOAT64 ,*  *V24 FLOAT64 ,*  *V25 FLOAT64 ,*  *V26 FLOAT64 ,*  *V27 FLOAT64 ,*  *V28 FLOAT64 ,*  *Amount FLOAT64 ,*  *Class INTEGER,*  *Feedback String*  *);* |

1. You can find the PUBSUB schema definition ***my\_avro\_fraud\_detection\_schema.json*** file in $HOME/bootkon-h2-2024/data-ingestion/src directory

| ***JSON : PubSub Schema Definition***  ***(This is the content of the schema definition file, for you information ONLY)*** |
| --- |
| *{*  *"type": "record",*  *"name": "Avro",*  *"fields": [*  *{"name": "Time","type": "float"},*  *{"name": "V1","type": "float"},*  *{"name": "V2","type": "float"},*  *{"name": "V3","type": "float"},*  *{"name": "V4","type": "float"},*  *{"name": "V5","type": "float"},*  *{"name": "V6","type": "float"},*  *{"name": "V7","type": "float"},*  *{"name": "V8","type": "float"},*  *{"name": "V9","type": "float"},*  *{"name": "V10","type": "float"},*  *{"name": "V11","type": "float"},*  *{"name": "V12","type": "float"},*  *{"name": "V13","type": "float"},*  *{"name": "V14","type": "float"},*  *{"name": "V15","type": "float"},*  *{"name": "V16","type": "float"},*  *{"name": "V17","type": "float"},*  *{"name": "V18","type": "float"},*  *{"name": "V19","type": "float"},*  *{"name": "V20","type": "float"},*  *{"name": "V21","type": "float"},*  *{"name": "V22","type": "float"},*  *{"name": "V23","type": "float"},*  *{"name": "V24","type": "float"},*  *{"name": "V25","type": "float"},*  *{"name": "V26","type": "float"},*  *{"name": "V27","type": "float"},*  *{"name": "V28","type": "float"},*  *{"name": "Amount","type": "float"},*  *{"name": "Class","type": "int"} ,*  *{"name": "Feedback","type": "string"}*  *]*  *}* |

1. Create the PubSub Schema Using gcloud

| ***Linux command line : Create PubSub Schema*** |
| --- |
| *cd* $HOME/bootkon-h2-2024/data-ingestion/src  *gcloud pubsub schemas create my\_fraud\_detection\_schema \*  *--project=$PROJECT\_ID \*  *--type=AVRO \*  *--definition-file=my\_avro\_fraud\_detection\_schema.json*  *Note: Make sure $PROJECT\_ID is set correctly.* |

1. Create the Pub/Sub Topic:

| ***Linux command line : Create PubSub Topic*** |
| --- |
| *gcloud pubsub topics create my\_fraud\_detection-topic \*  *--project=$PROJECT\_ID \*  *--schema=my\_fraud\_detection\_schema \*  *--message-encoding=BINARY*  *Note: Make sure $PROJECT\_ID is set correctly.* |

1. In order to grant The Pub/Sub service account in IAM needs the following BigQuery roles:

| * roles/bigquery.dataEditor * roles/bigquery.jobUser |
| --- |

* First, Find Your Pub/Sub Service Account:

| ***Linux command line : Find out the Pub/Sub service account email address*** |
| --- |
| *export PROJECT\_ID=****your\_project\_id***  *export PROJECT\_NUM=$(gcloud projects describe ${PROJECT\_ID} --format="value(projectNumber)")*  *echo $PROJECT\_NUM*  *export PUBSUBSVCACCT=service-$PROJECT\_NUM@gcp-sa-pubsub.iam.gserviceaccount.com*  *echo $PUBSUBSVCACCT* |

* Then, Grant Permissions (if not already granted), see below commands:

| ***Linux command line : Grant privileges to the Pub/Sub service account*** |
| --- |
| *gcloud projects add-iam-policy-binding $PROJECT\_ID --member=serviceAccount:$PUBSUBSVCACCT --role=roles/bigquery.dataEditor*  *gcloud projects add-iam-policy-binding $PROJECT\_ID --member=serviceAccount:$PUBSUBSVCACCT --role=roles/bigquery.jobUser* |

1. Create the Pub/Sub BQ subscription:

| ***Linux command line : Create a PUBSUB to BQ Subscription*** |
| --- |
| *gcloud pubsub subscriptions create my\_fraud\_detection-subscription \*  *--project=$PROJECT\_ID \*  *--topic=my\_fraud\_detection-topic \*  *--bigquery-table=$PROJECT\_ID.ml\_datasets.ulb\_fraud\_detection \*  *--use-topic-schema*  *Note: Make sure $PROJECT\_ID is set correctly.* |

### ***Step 2: Ingest data into BQ through PUBSUB***

1. Create Virtual env:

| ***Linux command line : Create a local virtual environment*** |
| --- |
| *cd $HOME*  *python3 -m venv hack*  *source hack/bin/activate* |

1. Install library requirements

Navigate to the root directory of the cloned repository, for example , bootkon-h2-2024. You find requirements.txt file. By using the requirements file you will be able to install the following packages ;

| * *google-cloud-aiplatform* * *google-api-python-client* * *google-cloud* * *google-cloud-bigquery* * *google-cloud-bigquery-storage* * *google-cloud-pubsub* * *google-cloud-logging* |
| --- |

| ***Linux command line : Install the required packages, Run the following commands within your virtual environment called “hack”*** |
| --- |
| *cd $HOME*  *cd bootkon-h2-2024/*  *pip install -r requirements.txt* |

1. **METHOD 1:** find the import\_csv\_to\_bigquery\_1.py under $HOME/bootkon-h2-2024/data-ingestion/src directory

*In the script:*

* *Replace your-project-id with your project\_id (in 1 location) (leave the double quotes “ unchanged)*
* *Replace your-project-id with your project\_id in “bucket\_name” (leave “-bucket” at the end) (in 1 location) (leave the double quotes “ unchanged)*
* *Comment out the Line (just add* ***#*** *at the beginning): 'os.environ['GOOGLE\_APPLICATION\_CREDENTIALS'] = <service key json location>/service-key.json'*

| ***Python Script : Import data into BigQuery in near real time [Method 1]*** |
| --- |
| *import io*  *import csv*  *import json*  *import avro.schema*  *from avro.io import BinaryEncoder, DatumWriter*  *from google.cloud import pubsub\_v1*  *from google.cloud import storage*  *import os*  *# Set Google Cloud credentials and project details*  *os.environ['GOOGLE\_APPLICATION\_CREDENTIALS'] = '<service key json location>/service-key.json'*  *project\_id = "****your-project-id****"*  *topic\_id = "my\_fraud\_detection-topic"*  *bucket\_name = "****your-project-id****-bucket"*  *csv\_folder\_path = "data-ingestion/csv/ulb\_fraud\_detection/"*  *schema\_file\_path = "data-ingestion/src/my\_avro\_fraud\_detection\_schema.json"*  *# Initialize Cloud Storage client and get the bucket*  *storage\_client = storage.Client()*  *bucket = storage\_client.bucket(bucket\_name)*  *# Load the AVRO schema from GCS*  *blob = bucket.blob(schema\_file\_path)*  *schema\_json = json.loads(blob.download\_as\_text())*  *avro\_schema = avro.schema.parse(json.dumps(schema\_json))*  *# Pub/Sub client initialization*  *publisher = pubsub\_v1.PublisherClient()*  *topic\_path = publisher.topic\_path(project\_id, topic\_id)*  *def publish\_avro\_record(record):*  *bytes\_io = io.BytesIO()*  *writer = DatumWriter(avro\_schema)*  *encoder = BinaryEncoder(bytes\_io)*  *writer.write(record, encoder)*  *future = publisher.publish(topic\_path, bytes\_io.getvalue())*  *return future.result()*  *def process\_csv\_blob(blob):*  *temp\_file\_path = "/tmp/tempfile.csv"*  *blob.download\_to\_filename(temp\_file\_path)*  *with open(temp\_file\_path, mode='r', encoding='utf-8') as csv\_file:*  *csv\_reader = csv.reader(csv\_file)*  *for row in csv\_reader:*  *feedback = row[-1]*  *record = {*  *"Time": float(row[0]),*  *"V1": float(row[1]),*  *"V2": float(row[2]),*  *"V3": float(row[3]),*  *"V4": float(row[4]),*  *"V5": float(row[5]),*  *"V6": float(row[6]),*  *"V7": float(row[7]),*  *"V8": float(row[8]),*  *"V9": float(row[9]),*  *"V10": float(row[10]),*  *"V11": float(row[11]),*  *"V12": float(row[12]),*  *"V13": float(row[13]),*  *"V14": float(row[14]),*  *"V15": float(row[15]),*  *"V16": float(row[16]),*  *"V17": float(row[17]),*  *"V18": float(row[18]),*  *"V19": float(row[19]),*  *"V20": float(row[20]),*  *"V21": float(row[21]),*  *"V22": float(row[22]),*  *"V23": float(row[23]),*  *"V24": float(row[24]),*  *"V25": float(row[25]),*  *"V26": float(row[26]),*  *"V27": float(row[27]),*  *"V28": float(row[28]),*  *"Amount": float(row[29]),*  *"Class": int(row[30]),*  *"Feedback": feedback*  *}*  *message\_id = publish\_avro\_record(record)*  *print(f"Published message with ID: {message\_id}")*  *# Process all CSV files in the folder*  *blobs = storage\_client.list\_blobs(bucket, prefix=csv\_folder\_path)*  *for blob in blobs:*  *if blob.name.endswith('.csv'):*  *process\_csv\_blob(blob)* |

1. **(you can skip this step if you completed LAB 1)** Ensure that your project compute engine service account has access to Dataproc worker , BigQuery Data editor, BigQuery Job user , PUBSUB and GCS bucket.
2. Run the Python Job

| ***Linux command line : Execute the script*** |
| --- |
| *time python data-ingestion/src/import\_csv\_to\_bigquery\_1.py*  ***Note: Make sure you run the command within your virtual environment called “hack”.*** |

1. Notice the output of the command execution.
2. After 5 minutes do interrupt the execution of the script, (perform a ***ctrl + c*** command). This method would have taken approximately between ***40 - 60*** minutes.
3. Check that there are some rows inserted into the ulb\_fraud\_detection table.

| ***BigQuery SQL : Check there are some rows inserted into ulb\_fraud\_detection table*** |
| --- |
| *select \* from `****your\_project\_id****.ml\_datasets.ulb\_fraud\_detection` ;* |

1. **Important:** Recreate the*`your\_project\_id.ml\_datasets.ulb\_fraud\_detection`* table.

| ***BigQuery SQL : Drop and Recreate BigQuery table***  *`****your-project-id****.ml\_datasets.ulb\_fraud\_detection`* |
| --- |
| *DROP TABLE `****your-project-id****.ml\_datasets.ulb\_fraud\_detection` ;*  *CREATE OR REPLACE TABLE `****your project id****.ml\_datasets.ulb\_fraud\_detection`*  *(*  *Time FLOAT64 ,*  *V1 FLOAT64 ,*  *V2 FLOAT64 ,*  *V3 FLOAT64 ,*  *V4 FLOAT64 ,*  *V5 FLOAT64 ,*  *V6 FLOAT64 ,*  *V7 FLOAT64 ,*  *V8 FLOAT64 ,*  *V9 FLOAT64 ,*  *V10 FLOAT64 ,*  *V11 FLOAT64 ,*  *V12 FLOAT64 ,*  *V13 FLOAT64 ,*  *V14 FLOAT64 ,*  *V15 FLOAT64 ,*  *V16 FLOAT64 ,*  *V17 FLOAT64 ,*  *V18 FLOAT64 ,*  *V19 FLOAT64 ,*  *V20 FLOAT64 ,*  *V21 FLOAT64 ,*  *V22 FLOAT64 ,*  *V23 FLOAT64 ,*  *V24 FLOAT64 ,*  *V25 FLOAT64 ,*  *V26 FLOAT64 ,*  *V27 FLOAT64 ,*  *V28 FLOAT64 ,*  *Amount FLOAT64 ,*  *Class INTEGER,*  *Feedback String*  *);* |

1. **METHOD 2 :** find the import\_csv\_to\_bigquery\_2.py under ./bootkon-h2-2024/data-ingestion/src directory

*In the script:*

* *Replace your-project-id with your project\_id (in 1 location) (leave the double quotes “ unchanged)*
* *Replace, in bucket\_name your-project-id with your project\_id (leave the “-bucket” suffix) (in 1 location) (leave the double quotes “ unchanged)*
* *Comment out the Line (just add # at the beginning): the os.environ['GOOGLE\_APPLICATION\_CREDENTIALS'] = '<service key json location>/service-key.json'*

| ***Python Script : Import data into BigQuery in near real time [Method 2]*** |
| --- |
| *import io*  *import csv*  *import json*  *import avro.schema*  *from avro.io import BinaryEncoder, DatumWriter*  *from google.cloud import pubsub\_v1*  *from google.cloud import storage*  *import os*  *# Set Google Cloud credentials and project details*  *os.environ['GOOGLE\_APPLICATION\_CREDENTIALS'] = '<service key json location>/service-key.json'*  *project\_id = "****your-project-id****"*  *topic\_id = "my\_fraud\_detection-topic"*  *bucket\_name = "****your-project-id****-bucket"*  *csv\_folder\_path = "data-ingestion/csv/ulb\_fraud\_detection/"*  *schema\_file\_path = "data-ingestion/src/my\_avro\_fraud\_detection\_schema.json"*  *# Initialize Cloud Storage client*  *storage\_client = storage.Client()*  *bucket = storage\_client.bucket(bucket\_name)*  *# Load the AVRO schema from GCS*  *blob = bucket.blob(schema\_file\_path)*  *schema\_json = json.loads(blob.download\_as\_text())*  *avro\_schema = avro.schema.parse(json.dumps(schema\_json))*  *# Pub/Sub client initialization with batch settings*  *batch\_settings = pubsub\_v1.types.BatchSettings(*  *max\_bytes=1024 \* 1024, # One megabyte*  *max\_latency=1, # One second*  *max\_messages=100 # 100 messages*  *)*  *publisher = pubsub\_v1.PublisherClient(batch\_settings=batch\_settings)*  *topic\_path = publisher.topic\_path(project\_id, topic\_id)*  *def publish\_avro\_record(records\_batch):*  *"""*  *Encodes records to AVRO format and publishes them to the specified Pub/Sub topic.*  *"""*  *futures = []*  *for record in records\_batch:*  *# Serialize data*  *bytes\_io = io.BytesIO()*  *writer = DatumWriter(avro\_schema)*  *encoder = BinaryEncoder(bytes\_io)*  *writer.write(record, encoder)*  *# Publish data*  *future = publisher.publish(topic\_path, bytes\_io.getvalue())*  *futures.append(future)*  *return futures*  *def process\_csv\_blob(blob):*  *"""*  *Reads a CSV file from GCS, encodes rows to AVRO, and publishes in batches to Pub/Sub.*  *"""*  *temp\_file\_path = "/tmp/tempfile.csv"*  *blob.download\_to\_filename(temp\_file\_path)*  *records\_batch = []*  *with open(temp\_file\_path, mode='r', encoding='utf-8') as csv\_file:*  *csv\_reader = csv.reader(csv\_file)*  *for row in csv\_reader:*  *feedback = row[-1]*  *record = {*  *"Time": float(row[0]),*  *"V1": float(row[1]),*  *"V2": float(row[2]),*  *"V3": float(row[3]),*  *"V4": float(row[4]),*  *"V5": float(row[5]),*  *"V6": float(row[6]),*  *"V7": float(row[7]),*  *"V8": float(row[8]),*  *"V9": float(row[9]),*  *"V10": float(row[10]),*  *"V11": float(row[11]),*  *"V12": float(row[12]),*  *"V13": float(row[13]),*  *"V14": float(row[14]),*  *"V15": float(row[15]),*  *"V16": float(row[16]),*  *"V17": float(row[17]),*  *"V18": float(row[18]),*  *"V19": float(row[19]),*  *"V20": float(row[20]),*  *"V21": float(row[21]),*  *"V22": float(row[22]),*  *"V23": float(row[23]),*  *"V24": float(row[24]),*  *"V25": float(row[25]),*  *"V26": float(row[26]),*  *"V27": float(row[27]),*  *"V28": float(row[28]),*  *"Amount": float(row[29]),*  *"Class": int(row[30]),*  *"Feedback": feedback*  *}*  *records\_batch.append(record)*  *if len(records\_batch) >= 100:*  *publish\_avro\_record(records\_batch)*  *records\_batch = []*  *if records\_batch:*  *publish\_avro\_record(records\_batch)*  *# List and process all CSV files*  *blobs = storage\_client.list\_blobs(bucket, prefix=csv\_folder\_path)*  *for blob in blobs:*  *if blob.name.endswith('.csv'):*  *process\_csv\_blob(blob)* |

1. **(you can skip this step if you completed LAB 1)** Ensure that your project compute engine service account has access to Dataproc worker , BigQuery Data editor, BigQuery Job user , PUBSUB and GCS bucket.
2. Run the Python Job

Notice the time it takes to run. Notice the differences between import\_csv\_to\_bigquery\_1.py and import\_csv\_to\_bigquery\_2.py execution time. The execution of METHOD 1 would have taken approximately between ***40 - 60 minutes***.

| ***Linux command line : Execute the script*** |
| --- |
| *time python data-ingestion/src/import\_csv\_to\_bigquery\_2.py*  ***Note: Make sure you run the command within your virtual environment called “hack”.*** |

1. Check Bigquery row count.

| ***BigQuery SQL : Table Count*** |
| --- |
| *select count(\*) from `****your\_project\_id****.ml\_datasets.ulb\_fraud\_detection` ;*  ***Note: the query should return about 260,249 records.*** |

*[TASK] Take up to 5 minutes and discuss within your group the key differences between Method 1 and Method 2. Focus specifically on how the execution time and overall efficiency compare between the two methods.*

**🥳🥳Congratulations on completing Lab 2!**

**You can now move on to Lab 3 for further practice. 🥳🥳**