Title: Data Ingestion

Goal of the lab

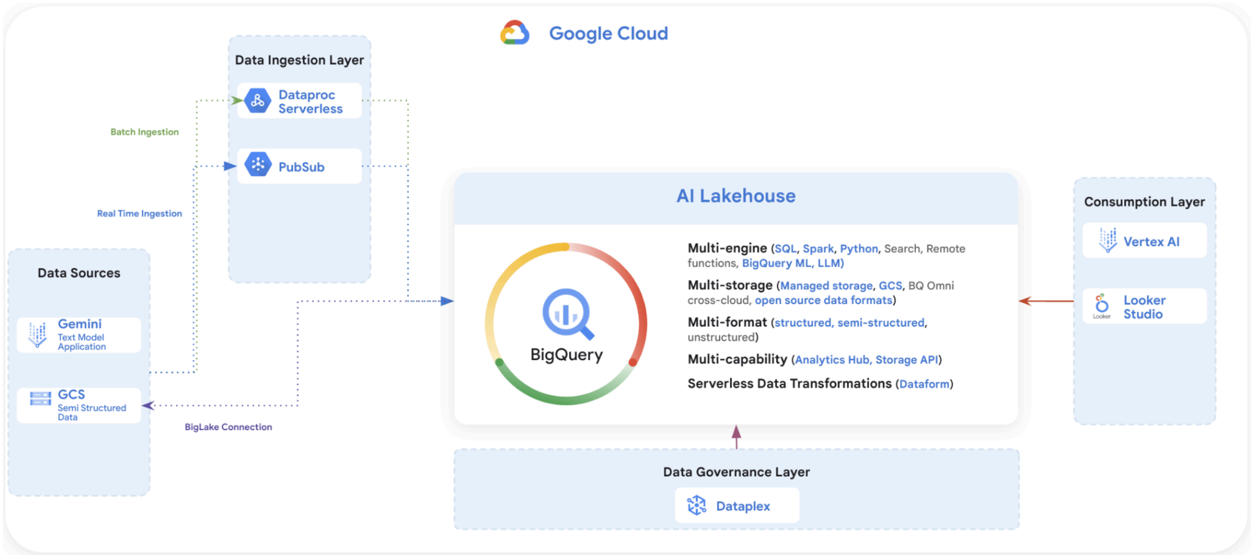
- Ingest fraudulent and non fraudulent transactions dataset into BigQuery using three methods:
  - Using BigLake with data stored in Google Cloud Storage (GCS)
  - [OPTIONAL] Batch Ingestion into BigQuery using Dataproc Serverless
  - [OPTIONAL] Near real-time ingestion into BigQuery using Cloud PubSub

Author: Wissem Khlifi, refactored by Daniel Holgate **Date**: 2024-11-11

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:


Google Cloud
Data Ingestion Layer
Dataproc Serverless
PubSub
Batch Ingestion
Real Time Ingestion
Data Sources
Gemini
Text Model Application
GCS
Semi Structured Data
BigLake Connection
AI Lakehouse
BigQuery
Multi-engine (SQL, Spark, Python, Search, Remote functions, BigQuery ML, LLM)
Multi-storage (Managed storage, GCS, BQ Omni cross-cloud, open source data formats)
Multi-format (structured, semi-structured, unstructured)
Multi-capability (Analytics Hub, Storage API)
Serverless Data Transformations (Dataform)
Consumption Layer
Vertex AI
Looker Studio
Data Governance Layer
Dataplex

[LAB] Load data via BigLake External Tables

BigLake compliments Google Cloud's BigQuery data warehouse to allow us to easily query both structured and unstructured data which can be stored in Google Cloud Storage (GCS). In this lab we will take some data in Parquet-format which is in GCS, build an external table to reference the data and then query it using BigQuery. This data will also be used in later labs.
You can find more information about BigLake here: [Introduction to BigLake external tables]

Set up

Note: you can skip the first two set up steps (Enable APIs and Required Roles) and go directly to "Add BigLake connection" if you ran the script to set up your environment:

Enable APIs

Make sure the following APIs are enabled in your project:

- BigQuery API
- Vertex AI API
- Pubsub API
- BigQuery Connection API
- Dataform API
- Secret Manager API

Required Roles

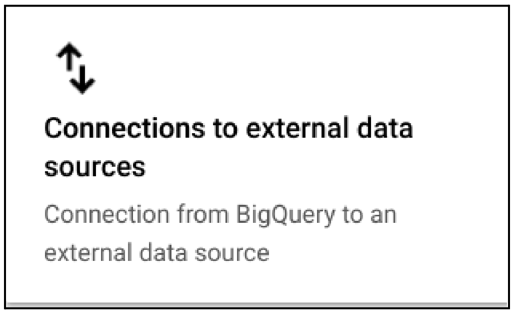
To create a BigLake table, you need the following BigQuery Identity and Access Management (IAM) permissions:

- bigquery.tables.create
- bigquery.connections.delegate

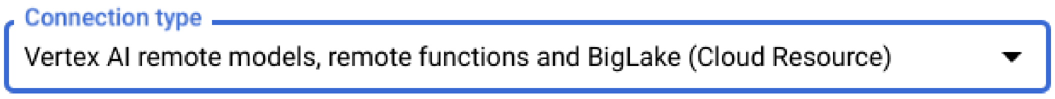
The BigQuery Data Editor and connection admin predefined Identity and Access Management role includes these permissions. If you are not a principal in this role, ask your administrator to grant you access or to create the BigLake table for you.

Add BigLake connection

1. In the Google Cloud console navigate to **Bigquery Studio** and click the **+ ADD** button
2. Choose "***Connections to external data sources***"

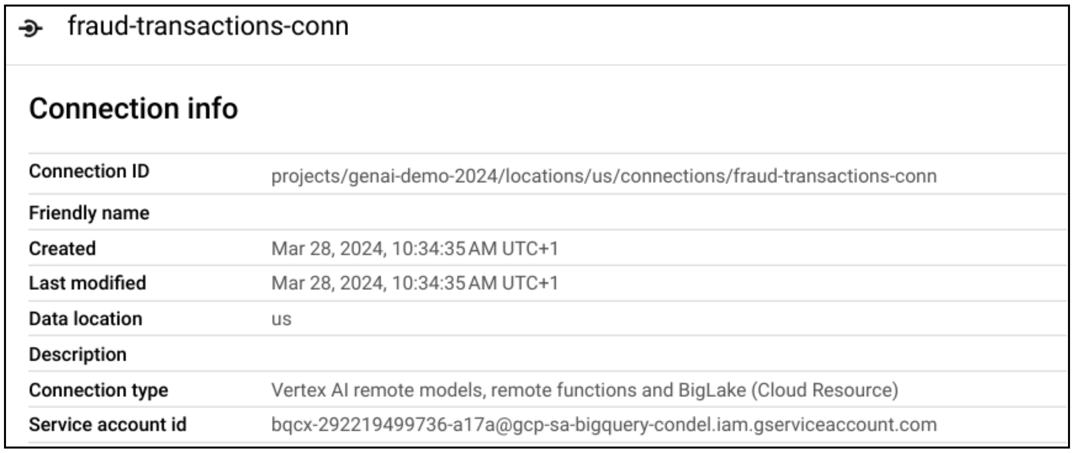

Connections to external data sources
Connection from BigQuery to an external data source

3. As connection type select ***Vertex AI remote models , remote functions and BigLake (Cloud Resources)***


Connection type

Vertex AI remote models, remote functions and BigLake (Cloud Resource)

4. Enter the connection ID as "fraud-transactions-conn" and choose the US multi-region location.
5. Click "CREATE CONNECTION"
6. Open the connection details for the newly added external connection "fraud-transactions-conn"

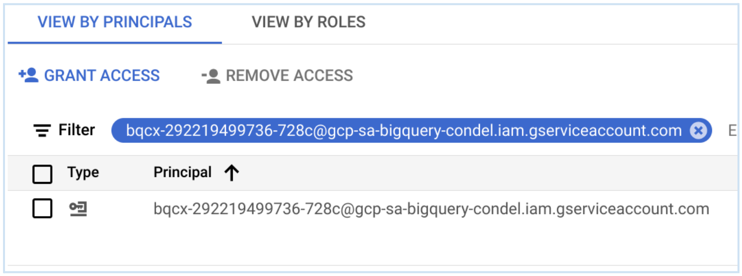

fraud-transactions-conn

Connection info

Connection ID	projects/genai-demo-2024/locations/us/connections/fraud-transactions-conn
Friendly name	
Created	Mar 28, 2024, 10:34:35 AM UTC+1
Last modified	Mar 28, 2024, 10:34:35 AM UTC+1
Data location	us
Description	
Connection type	Vertex AI remote models, remote functions and BigLake (Cloud Resource)
Service account id	bqcx-292219499736-a17a@gcp-sa-bigquery-condel.iam.gserviceaccount.com

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

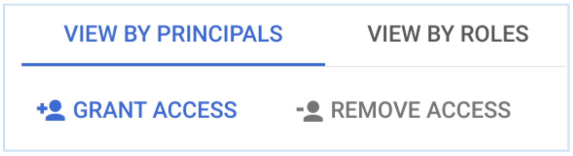
- Go to IAM & Admin
- Filter the Service account



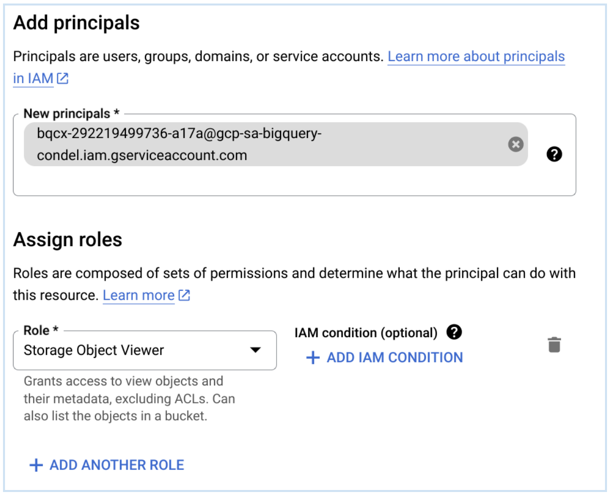
bqcx-292219499736-728c@gcp-sa-bigquery-condel.iam.gserviceaccount.com

Type	Principal
	bqcx-292219499736-728c@gcp-sa-bigquery-condel.iam.gserviceaccount.com

- 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.



Add principals

Principals are users, groups, domains, or service accounts. Learn more about principals in IAM

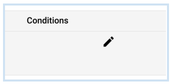
New principals *
bqcx-292219499736-a17a@gcp-sa-bigquery-condel.iam.gserviceaccount.com

Assign roles

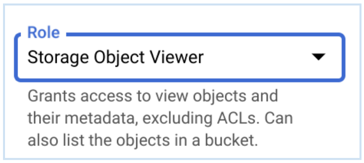
Roles are composed of sets of permissions and determine what the principal can do with this resource. Learn more

Role *
Storage Object Viewer
Grants access to view objects and their metadata, excluding ACLs. Can also list the objects in a bucket.

- 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.



Role
Storage Object Viewer
Grants access to view objects and their metadata, excluding ACLs. Can also list the objects in a bucket.

8. From Cloud Shell, create a dataset named **ml_datasets** in the US multi-region.

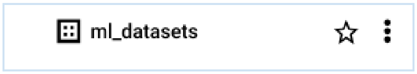
Linux command line : Create a BigQuery dataset: call it ml_datasets in US multi region

```
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 "

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



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

11. 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;
```

[OPTIONAL LAB] Batch data ingestion into BigQuery using Dataproc

Google Cloud Dataproc is a fully managed and scalable service for running Apache Hadoop, Apache Spark, Apache Flink, Presto, and 30+ open source tools and frameworks. Dataproc allows data to be loaded and also transformed or pre-processed as it is brought in.

Serverless Dataproc

In this lab the Paquet-format files are imported from Google Cloud Storage into BigQuery using PySpark. The label details the steps, including setup and code implementation.

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;
```

2. Download and upload parquet files to GCS **(you can skip this step if you completed LAB 1).**
3. In Cloud shell, go to directory ; **cd *$HOME/bootkon-h2-2024/data-ingestion/src***
4. 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) (leave the double quotes " unchanged)*

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()
```

5. 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*

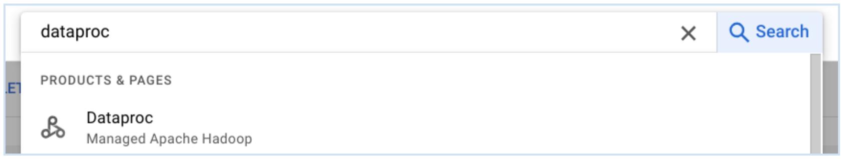
Linux command line : Submit spark job to Dataproc

```
gcloud dataproc batches submit pyspark import_parquet_to_bigquery.py \
   --project=your-project-id \
    --region=your-region \
   --deps-bucket=gs://your-project-id-bucket \
```

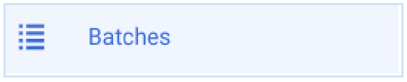
This command specifies:

- 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.

6. Monitor the job execution progress in Dataproc;
   From the search write : dataproc



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



8. After the Dataproc job completes, confirm that data has been loaded into the BigQuery table (replace your-project-id with your project ID). You should see over 200,000 records, but the exact count isn't critical.

   Use the following query to check:

   SELECT count(*) FROM `your-project-id.ml_datasets.ulb_fraud_detection_parquet`

[OPTIONAL 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

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
);
```

2. 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"}

  ]
}
```

3. 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.

4. 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.

5. 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
```

6. 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
```

2. 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
```

3. **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)
```

4. **(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.
5. Run the Python Job

Linux command line : Execute the script

```
time python $HOME/bootkon-h2-2024/data-ingestion/src/import_csv_to_bigquery_1.py
```

Note: Make sure you run the command within your virtual environment called "hack".

6. Notice the output of the command execution.
7. After 4 or 5 minutes do interrupt the execution of the script, (perform a **ctrl + c** command). This method would have taken approximately between ***40 - 60*** minutes.
8. 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` ;
```

9. **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
);
```

10. **METHOD 2 :** find the import_csv_to_bigquery_2.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, 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)
```

11. **(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.
12. 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 (execution should take between 1-2 minutes)

```
time python  $HOME/bootkon-h2-2024/data-ingestion/src/import_csv_to_bigquery_2.py
```

Note: Make sure you run the command within your virtual environment called "hack".

13. Check Bigquery row count.

BigQuery SQL : Table Count

```
select count(*) from  `your_project_id.ml_datasets.ulb_fraud_detection` ;
```

Note: The query should return a few hundred thousand records, but the exact count may fluctuate as Pub/Sub continues loading data. The exact count isn't critical. You don't need to wait for the entire ingestion to finish; feel free to proceed with Lab 3 while the data loads in the background.

[TASK] Take up to a couple (2) of 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. 🥳🥳