NY taxi data engineering proyect with GCP

Tools that we are going to use:

- Python
- Jupyter Notebooks
- Mage (open source data pipeline tool for transforming and integrating data) alternativa a Airflow
- GCP tools like Google Cloud Storage, Compute Engine, Big Query and Looker
- Lucid to create a diagram of the tables

Basic concepts

- [Fact table]:
  - contains quantitative measures or metrics that are used for analysis
  - tipically contains foreign keys that link to dimension tables
  - contains collumns that have [high cardinallity] **and change frequently**
    a table that has a high variety of unique values. eg: user_id
  - contains columns that are not useful for analysis by themselves, but necessary to calculate metrics
- [Dimension table]:
  - contains columns that describe attributes of the data being analyzed
  - tipically contains primary keys that link to fact tables
  - contains columns that have [low cardinality] **and don't change frequently**
    a table that has a limited variety of unique values. eg: gender (can be men, women, nb)
  - contains columns that can be used for grouping or filtering data for analysis

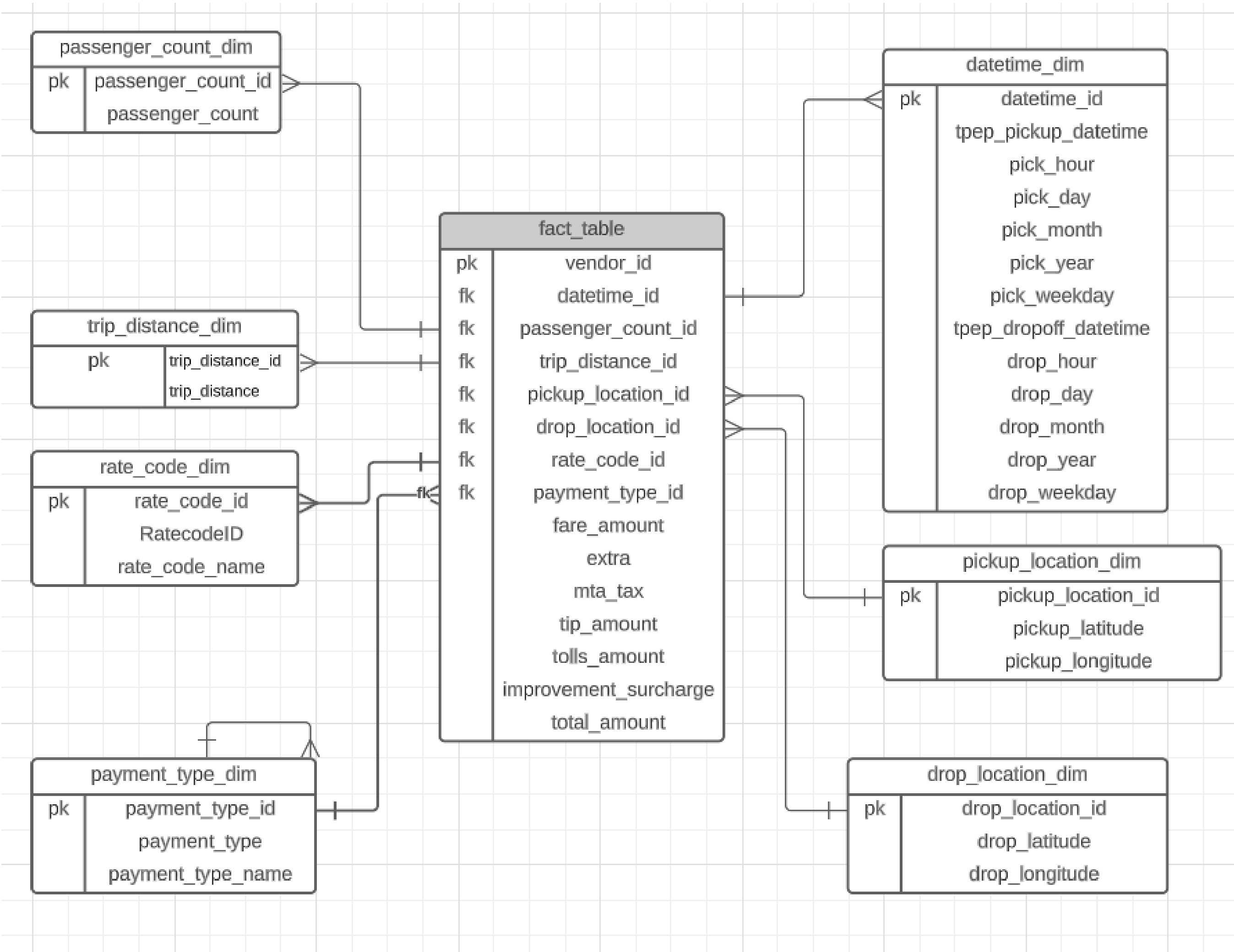
1.- Download and load data into Jupyter

Download parquet data in https://www.nyc.gov/site/tlc/about/tlc-trip-record-data.page and transform it into csv with:

`df = pd.read_parquet("C:\Users\ruben\Desktop\data engineering\projects\tlc_nyc\yellow_tripdata_2022-01.parquet")

`df.to_csv("C:\\Users\\ruben\\Desktop\\data engineering\\projects\\tlc_nyc\\yellow_22-01.csv", index=False)`

1.1.- Create the fact and dim tables diagram with LucidCharts



1.2.- Writing the transformations in jupyter notebook

Creamos el código de las transfomaciones necesarias para pasar los datos en bruto a la configuración fact table / dimension tables con pyspark.
El código quedaría algo así:

```
from pyspark.sql.functions import when
from pyspark.sql.functions import year, month, dayofmonth, dayofweek, date_format, hour
from pyspark.sql.functions import monotonically_increasing_id
from pyspark.sql.functions import year, month, dayofmonth, dayofweek, date_format, hour
from pyspark.sql.functions import monotonically_increasing_id
from pyspark.sql.functions import col


df = spark.read.parquet("C:\\Users\\ruben\\Desktop\\data
engineering\\projects\\tlc_nyc\\pyspark\\yellow_tripdata_2022-01.parquet")

sample_df = df.sample(fraction=1.0, withReplacement=False, seed=42)
sample_df = sample_df.withColumn('trip_id', monotonically_increasing_id())
```

```
datetime_dim = sample_df[['tpep_pickup_datetime','tpep_dropoff_datetime']]

#Creamos un índice que nos valdrá de identificador.
datetime_dim = datetime_dim.withColumn("datetime_id", monotonically_increasing_id())

#Dividimos el timestamp de pick y drop en sus respectivo hora, dia, mes, año y día de la semana
creando columnas con .withColumn
datetime_dim = datetime_dim.withColumn("pick_hour", hour(datetime_dim["tpep_pickup_datetime"]))
datetime_dim = datetime_dim.withColumn("pick_day",
dayofmonth(datetime_dim["tpep_pickup_datetime"]))
datetime_dim = datetime_dim.withColumn("pick_month",
month(datetime_dim["tpep_pickup_datetime"]))
datetime_dim = datetime_dim.withColumn("pick_year", year(datetime_dim["tpep_pickup_datetime"]))
datetime_dim = datetime_dim.withColumn("pick_weekday",
dayofweek(datetime_dim["tpep_pickup_datetime"]))

datetime_dim = datetime_dim.withColumn("drop_hour",
hour(datetime_dim["tpep_dropoff_datetime"]))
datetime_dim = datetime_dim.withColumn("drop_day",
dayofmonth(datetime_dim["tpep_dropoff_datetime"]))
datetime_dim = datetime_dim.withColumn("drop_month",
month(datetime_dim["tpep_dropoff_datetime"]))
datetime_dim = datetime_dim.withColumn("drop_year",
year(datetime_dim["tpep_dropoff_datetime"]))
datetime_dim = datetime_dim.withColumn("drop_weekday",
dayofweek(datetime_dim["tpep_dropoff_datetime"]))

# Reordenamos el dataset
datetime_dim = datetime_dim.select(*[['datetime_id','tpep_pickup_datetime', 'pick_hour',
'pick_day', 'pick_month', 'pick_year', 'pick_weekday',
                             'tpep_dropoff_datetime', 'drop_hour', 'drop_day', 'drop_month',
'drop_year', 'drop_weekday']] )

passenger_count_dim = sample_df[['passenger_count']]
passenger_count_dim = passenger_count_dim.withColumn("passenger_count_id",
monotonically_increasing_id())
passenger_count_dim = passenger_count_dim.select(*[['passenger_count_id', 'passenger_count']])

trip_distance_dim = sample_df[['trip_distance']]
trip_distance_dim = trip_distance_dim.withColumn("trip_distance_id",
monotonically_increasing_id())
trip_distance_dim = trip_distance_dim.select(*[['trip_distance_id', 'trip_distance']])

rate_code_type = {
    1 :"Standard rate",
    2 :"JFK",
```

```
    3 :"Newark",
    4 :"Nassau or Westchester",
    5 :"Negotiated fare",
    6:"Group ride"
}

rate_code_dim = sample_df[['RatecodeID']]
rate_code_dim = rate_code_dim.withColumn("rate_code_id", monotonically_increasing_id())

# ¿Por qué esto no funciona? Que forma tengo de hacerlo más eficaz?
# for code, name in rate_code_type.items():
#     rate_code_dim = rate_code_dim.withColumn("rate_code_name", when(col("RatecodeID") ==
code, name))

rate_code_dim = rate_code_dim.withColumn("rate_code_name", when(col("RatecodeID") == 1,
rate_code_type[1])
    .when(col("RatecodeID") == 2, rate_code_type[2])
    .when(col("RatecodeID") == 3, rate_code_type[3])
    .when(col("RatecodeID") == 4, rate_code_type[4])
    .when(col("RatecodeID") == 5, rate_code_type[5])
    .when(col("RatecodeID") == 6, rate_code_type[6])
    .otherwise("Unknown"))

rate_code_dim = rate_code_dim.select(*[['rate_code_id', 'RatecodeID', 'rate_code_name']])

payment_type_name = {
    1:"Credit card",
    2:"Cash",
    3:"No charge",
    4:"Dispute",
    5:"Unknown",
    6:"Voided trip"
}

payment_type_dim = sample_df[['payment_type']]
payment_type_dim = payment_type_dim.withColumn("payment_type_id",
monotonically_increasing_id())


payment_type_dim = payment_type_dim.withColumn("payment_type_name", when(col("payment_type") ==
1, rate_code_type[1])
    .when(col("payment_type") == 2, payment_type_name[2])
    .when(col("payment_type") == 3, payment_type_name[3])
    .when(col("payment_type") == 4, payment_type_name[4])
    .when(col("payment_type") == 5, payment_type_name[5])
    .when(col("payment_type") == 6, payment_type_name[6])
```

```
    .otherwise("Unknown"))


payment_type_dim = payment_type_dim.select(*[['payment_type_id', 'payment_type',
'payment_type_name']])

taxi_zone = spark.read.csv("C:\\Users\\ruben\\Desktop\\data
engineering\\projects\\tlc_nyc\\datasets\\taxi_zone.csv", header = True)

pickup_location_dim = sample_df[['PULocationID']]
pickup_location_dim = pickup_location_dim.withColumn('pickup_location_id',
monotonically_increasing_id())
pickup_location_dim = pickup_location_dim.join(taxi_zone, pickup_location_dim["PULocationID"]
== taxi_zone["LocationID"], "left")
pickup_location_dim = pickup_location_dim.select(*[['pickup_location_id', 'PULocationID',
'Borough', 'Zone', 'service_zone']])

pickup_location_dim = pickup_location_dim.withColumnRenamed("Borough", "Borough_pickup") \
    .withColumnRenamed("Zone", "Zone_pickup") \
    .withColumnRenamed("service_zone", "service_zone_pickup")

drop_location_dim = sample_df[['DOLocationID']]
drop_location_dim = drop_location_dim.withColumn('drop_location_id',
monotonically_increasing_id())
drop_location_dim = drop_location_dim.join(taxi_zone, drop_location_dim["DOLocationID"] ==
taxi_zone["LocationID"], "left")
drop_location_dim = drop_location_dim.select(*[['drop_location_id', 'DOLocationID', 'Borough',
'Zone', 'service_zone']])

drop_location_dim = drop_location_dim.withColumnRenamed("Borough", "Borough_drop") \
    .withColumnRenamed("Zone", "Zone_drop") \
    .withColumnRenamed("service_zone", "service_zone_drop")

# Realizar la unión de DataFrames con diferentes columnas de unión
fact_table = sample_df.join(passenger_count_dim, sample_df["trip_id"] ==
passenger_count_dim["passenger_count_id"], "inner") \
    .join(trip_distance_dim, sample_df["trip_id"] == trip_distance_dim["trip_distance_id"],
"inner") \
    .join(rate_code_dim, sample_df["trip_id"] == rate_code_dim["rate_code_id"], "inner") \
    .join(datetime_dim, sample_df["trip_id"] == datetime_dim["datetime_id"], "inner") \
    .join(payment_type_dim, sample_df["trip_id"] == payment_type_dim["payment_type_id"],
"inner") \
    .join(pickup_location_dim, sample_df["trip_id"] ==
pickup_location_dim["pickup_location_id"], "inner") \
    .join(drop_location_dim, sample_df["trip_id"] == drop_location_dim["drop_location_id"],
"inner")
```

```
fact_table = fact_table.select(*[['trip_id','VendorID', 'datetime_id', 'passenger_count_id',
               'trip_distance_id', 'rate_code_id', 'store_and_fwd_flag', 'pickup_location_id',
'drop_location_id',
               'payment_type_id', 'fare_amount', 'extra', 'mta_tax', 'tip_amount',
'tolls_amount',
               'improvement_surcharge', 'airport_fee', 'congestion_surcharge',
'total_amount']])
fact_table.show(10)
```

Google Cloud Platform

Una vez creado la template de trasformación en un notebook, vamos a modificarla en un archivo .py para que coja los datos de google cloud storage los procese y los mande a BigQuery.
El código del archivo .py configurado para usarse en un cluster de Dataproc es el siguiente:

```
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("EjemploSpark").getOrCreate()

df = spark.read.parquet("gs://batch_taxis/yellow_tripdata_2022-01.parquet")
from pyspark.sql.functions import when
from pyspark.sql.functions import year, month, dayofmonth, dayofweek, date_format, hour
from pyspark.sql.functions import monotonically_increasing_id
from pyspark.sql.functions import year, month, dayofmonth, dayofweek, date_format, hour
from pyspark.sql.functions import monotonically_increasing_id
from pyspark.sql.functions import col



sample_df = df.sample(fraction=1.0, withReplacement=False, seed=42)
sample_df = sample_df.withColumn('trip_id', monotonically_increasing_id())

datetime_dim = sample_df[['tpep_pickup_datetime','tpep_dropoff_datetime']]

#Creamos un Ãndice que nos valdrÃ¡ de identificador.
datetime_dim = datetime_dim.withColumn("datetime_id", monotonically_increasing_id())

#Dividimos el timestamp de pick y drop en sus respectivo hora, dia, mes, aÃ±o y dÃa de la
semana creando columnas con .withColumn
datetime_dim = datetime_dim.withColumn("pick_hour", hour(datetime_dim["tpep_pickup_datetime"]))
datetime_dim = datetime_dim.withColumn("pick_day",
dayofmonth(datetime_dim["tpep_pickup_datetime"]))
datetime_dim = datetime_dim.withColumn("pick_month",
month(datetime_dim["tpep_pickup_datetime"]))
datetime_dim = datetime_dim.withColumn("pick_year", year(datetime_dim["tpep_pickup_datetime"]))
```

```
datetime_dim = datetime_dim.withColumn("pick_weekday",
dayofweek(datetime_dim["tpep_pickup_datetime"]))

datetime_dim = datetime_dim.withColumn("drop_hour",
hour(datetime_dim["tpep_dropoff_datetime"]))
datetime_dim = datetime_dim.withColumn("drop_day",
dayofmonth(datetime_dim["tpep_dropoff_datetime"]))
datetime_dim = datetime_dim.withColumn("drop_month",
month(datetime_dim["tpep_dropoff_datetime"]))
datetime_dim = datetime_dim.withColumn("drop_year",
year(datetime_dim["tpep_dropoff_datetime"]))
datetime_dim = datetime_dim.withColumn("drop_weekday",
dayofweek(datetime_dim["tpep_dropoff_datetime"]))

# Reordenamos el dataset
datetime_dim = datetime_dim.select(*[['datetime_id','tpep_pickup_datetime', 'pick_hour',
'pick_day', 'pick_month', 'pick_year', 'pick_weekday',
                             'tpep_dropoff_datetime', 'drop_hour', 'drop_day', 'drop_month',
'drop_year', 'drop_weekday']] )

passenger_count_dim = sample_df[['passenger_count']]
passenger_count_dim = passenger_count_dim.withColumn("passenger_count_id",
monotonically_increasing_id())
passenger_count_dim = passenger_count_dim.select(*[['passenger_count_id', 'passenger_count']])

trip_distance_dim = sample_df[['trip_distance']]
trip_distance_dim = trip_distance_dim.withColumn("trip_distance_id",
monotonically_increasing_id())
trip_distance_dim = trip_distance_dim.select(*[['trip_distance_id', 'trip_distance']])

rate_code_type = {
    1 :"Standard rate",
    2 :"JFK",
    3 :"Newark",
    4 :"Nassau or Westchester",
    5 :"Negotiated fare",
    6:"Group ride"
}

rate_code_dim = sample_df[['RatecodeID']]
rate_code_dim = rate_code_dim.withColumn("rate_code_id", monotonically_increasing_id())

# Â¿Por quÃ© esto no funciona? Que forma tengo de hacerlo mÃ¡s eficaz?
# for code, name in rate_code_type.items():
#     rate_code_dim = rate_code_dim.withColumn("rate_code_name", when(col("RatecodeID") ==
code, name))
```

```
rate_code_dim = rate_code_dim.withColumn("rate_code_name", when(col("RatecodeID") == 1,
rate_code_type[1])
    .when(col("RatecodeID") == 2, rate_code_type[2])
    .when(col("RatecodeID") == 3, rate_code_type[3])
    .when(col("RatecodeID") == 4, rate_code_type[4])
    .when(col("RatecodeID") == 5, rate_code_type[5])
    .when(col("RatecodeID") == 6, rate_code_type[6])
    .otherwise("Unknown"))

rate_code_dim = rate_code_dim.select(*[['rate_code_id', 'RatecodeID', 'rate_code_name']])

payment_type_name = {
    1:"Credit card",
    2:"Cash",
    3:"No charge",
    4:"Dispute",
    5:"Unknown",
    6:"Voided trip"
}

payment_type_dim = sample_df[['payment_type']]
payment_type_dim = payment_type_dim.withColumn("payment_type_id",
monotonically_increasing_id())


payment_type_dim = payment_type_dim.withColumn("payment_type_name", when(col("payment_type") ==
1, rate_code_type[1])
    .when(col("payment_type") == 2, payment_type_name[2])
    .when(col("payment_type") == 3, payment_type_name[3])
    .when(col("payment_type") == 4, payment_type_name[4])
    .when(col("payment_type") == 5, payment_type_name[5])
    .when(col("payment_type") == 6, payment_type_name[6])
    .otherwise("Unknown"))


payment_type_dim = payment_type_dim.select(*[['payment_type_id', 'payment_type',
'payment_type_name']])

taxi_zone = spark.read.csv("gs://batch_taxis/taxi_zone.csv", header = True)


pickup_location_dim = sample_df[['PULocationID']]
pickup_location_dim = pickup_location_dim.withColumn('pickup_location_id',
monotonically_increasing_id())
pickup_location_dim = pickup_location_dim.join(taxi_zone, pickup_location_dim["PULocationID"]
```

```
== taxi_zone["LocationID"], "left")
pickup_location_dim = pickup_location_dim.select(*[['pickup_location_id', 'PULocationID',
'Borough', 'Zone', 'service_zone']])

pickup_location_dim = pickup_location_dim.withColumnRenamed("Borough", "Borough_pickup") \
    .withColumnRenamed("Zone", "Zone_pickup") \
    .withColumnRenamed("service_zone", "service_zone_pickup")

drop_location_dim = sample_df[['DOLocationID']]
drop_location_dim = drop_location_dim.withColumn('drop_location_id',
monotonically_increasing_id())
drop_location_dim = drop_location_dim.join(taxi_zone, drop_location_dim["DOLocationID"] ==
taxi_zone["LocationID"], "left")
drop_location_dim = drop_location_dim.select(*[['drop_location_id', 'DOLocationID', 'Borough',
'Zone', 'service_zone']])

drop_location_dim = drop_location_dim.withColumnRenamed("Borough", "Borough_drop") \
    .withColumnRenamed("Zone", "Zone_drop") \
    .withColumnRenamed("service_zone", "service_zone_drop")

# Realizar la uniÃ³n de DataFrames con diferentes columnas de uniÃ³n
fact_table = sample_df.join(passenger_count_dim, sample_df["trip_id"] ==
passenger_count_dim["passenger_count_id"], "inner") \
    .join(trip_distance_dim, sample_df["trip_id"] == trip_distance_dim["trip_distance_id"],
"inner") \
    .join(rate_code_dim, sample_df["trip_id"] == rate_code_dim["rate_code_id"], "inner") \
    .join(datetime_dim, sample_df["trip_id"] == datetime_dim["datetime_id"], "inner") \
    .join(payment_type_dim, sample_df["trip_id"] == payment_type_dim["payment_type_id"],
"inner") \
    .join(pickup_location_dim, sample_df["trip_id"] ==
pickup_location_dim["pickup_location_id"], "inner") \
    .join(drop_location_dim, sample_df["trip_id"] == drop_location_dim["drop_location_id"],
"inner")

fact_table = fact_table.select(*[['trip_id','VendorID', 'datetime_id', 'passenger_count_id',
               'trip_distance_id', 'rate_code_id', 'store_and_fwd_flag', 'pickup_location_id',
'drop_location_id',
               'payment_type_id', 'fare_amount', 'extra', 'mta_tax', 'tip_amount',
'tolls_amount',
               'improvement_surcharge', 'airport_fee', 'congestion_surcharge',
'total_amount']])
fact_table.show(10)
fact_table.printSchema()

tablesnames = ['passenger_count_dim', 'trip_distance_dim','rate_code_dim','datetime_dim',
'payment_type_dim', 'pickup_location_dim', 'drop_location_dim']
```

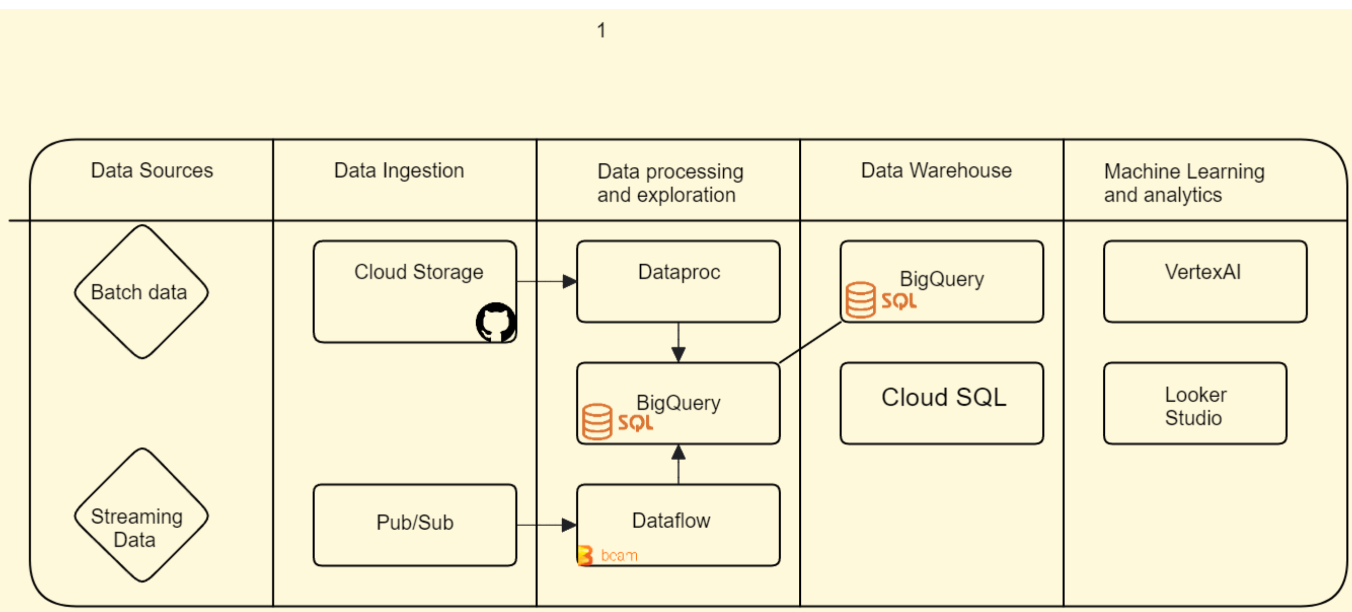
```
# fact_table.write.csv("gs://batch_taxis/data001.csv")
# fact_table.write.format("bigquery") \
#     .option("table", "data_taxi_batch.clean_data") \
#     .save()
bucket = "batch_taxis"
spark.conf.set('temporaryGcsBucket', bucket)

tablesnames = [fact_table, passenger_count_dim, trip_distance_dim,rate_code_dim,datetime_dim,
payment_type_dim, pickup_location_dim, drop_location_dim]
rutasnames = ['data_taxi_batch.fact_table', 'data_taxi_batch.passenger', '
data_taxi_batch.tripdistance', 'data_taxi_batch.ratecode', 'data_taxi_batch.datetime',
'data_taxi_batch.paymenttype', 'data_taxi_batch.pickuploc', 'data_taxi_batch.droploc']


for tablasaux, rutasaux in zip(tablesnames, rutasnames):
    tablasaux.write.format('bigquery') \
    .option('table', rutasaux) \
    .option('mode', 'overwrite') \
    .save()

spark.stop()
```

Arquitectura Batch Pipeline en GCP



Dataproc

Submit job en Dataproc

Lo primero que vamos a hacer es descargar el archivo .py que guardo en un repositorio de github para guardar las actualizaciones del código desde allí.

```
git -C ~ clone https://github.com/Rubnserrano/dataengportfolio

cd dataengportoflio

gsutil cp tlc_new.py gs://batch_taxis/
```

Ahora mandamos el job a un cluster de dataproc

```
gcloud dataproc jobs submit pyspark gs://batch_taxis/tlc_new.py     --cluster=prueba123241
--region=us-central1
```

BigQuery

Una vez el trabajo ha finalizado, en tendremos las tablas del diagrama E-R. Uniremos todas las tablas según las claves para crear una tabla de analíticas.

Query para unir todos los datos

`` `CREATE OR REPLACE TABLE` taxi-project-405909.data_taxi_batch.analitycs` AS (
SELECT
f.trip_id,
d.tpep_pickup_datetime,
d.tpep_dropoff_datetime,
p.passenger_count,
t.trip_distance,
r.rate_code_name,
pay.payment_type_name,
f.fare_amount,
f.extra,
f.mta_tax,
f.tip_amount,
f.tolls_amount,
f.improvement_surcharge,
f.total_amount,
pick.Borough_pickup,
pick.Zone_pickup,
k.Borough_drop,
k.Zone_drop,

FROM

`taxi-project-405909.data_taxi_batch.fact_table` f

JOIN `taxi-project-405909.data_taxi_batch.datetime` d ON f.datetime_id=d.datetime_id
JOIN `taxi-project-405909.data_taxi_batch.passenger` p ON p.passenger_count_id=f.passenger_count_id
JOIN `taxi-project-405909.data_taxi_batch.tripdistance` t ON t.trip_distance_id=f.trip_distance_id
JOIN `taxi-project-405909.data_taxi_batch.ratecode` r ON r.rate_code_id=f.rate_code_id
JOIN `taxi-project-405909.data_taxi_batch.pickuploc` pick ON pick.pickup_location_id=f.pickup_location_id
JOIN `taxi-project-405909.data_taxi_batch.droploc` k ON k.drop_location_id=f.drop_location_id
JOIN `taxi-project-405909.data_taxi_batch.paymenttype` pay ON pay.payment_type_id=f.payment_type_id)```

Con el 100% de los datos de 2022 en local tarda 18s.

Looker Studio

Report para datos Enero del 2022