

# Using Spark and BigQuery to Process Aviation Data

Module 7 - ST 517: Data Analytics I | Oregon State University

Paul J Anderson & Rachel Hughes

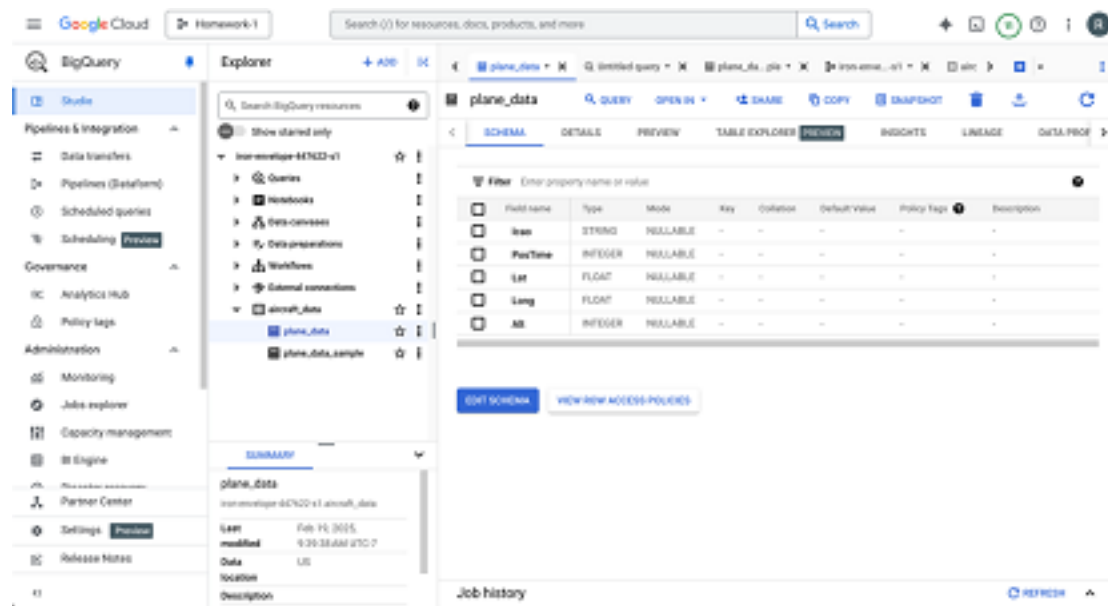
2025-02-21

## 1. Abstract

This project is designed to explore the power of Google Cloud Platform services, especially using Dataproc to run Spark jobs on managed clusters. Using the provided dataset of airplane traffic, the ultimate goal of the project is to determine the total amount of distance traveled by all airplanes that flew during the given day.

## 2. Obtain

The project imports a complete 24-hour dataset of airplane traffic for January 15, 2018 into BigQuery. Previous classwork involved scrubbing the data in DataPrep to get a standardized JSON format and to eliminate excess data that was not needed for the project. As documented previously, attempts to import into BigQuery failed. Instead, a .csv file was provide and uploaded into BigQuery. This data was then used for the rest of the project. The schema for the imported data is below – the data is stored in BigQuery table `plane_data`.

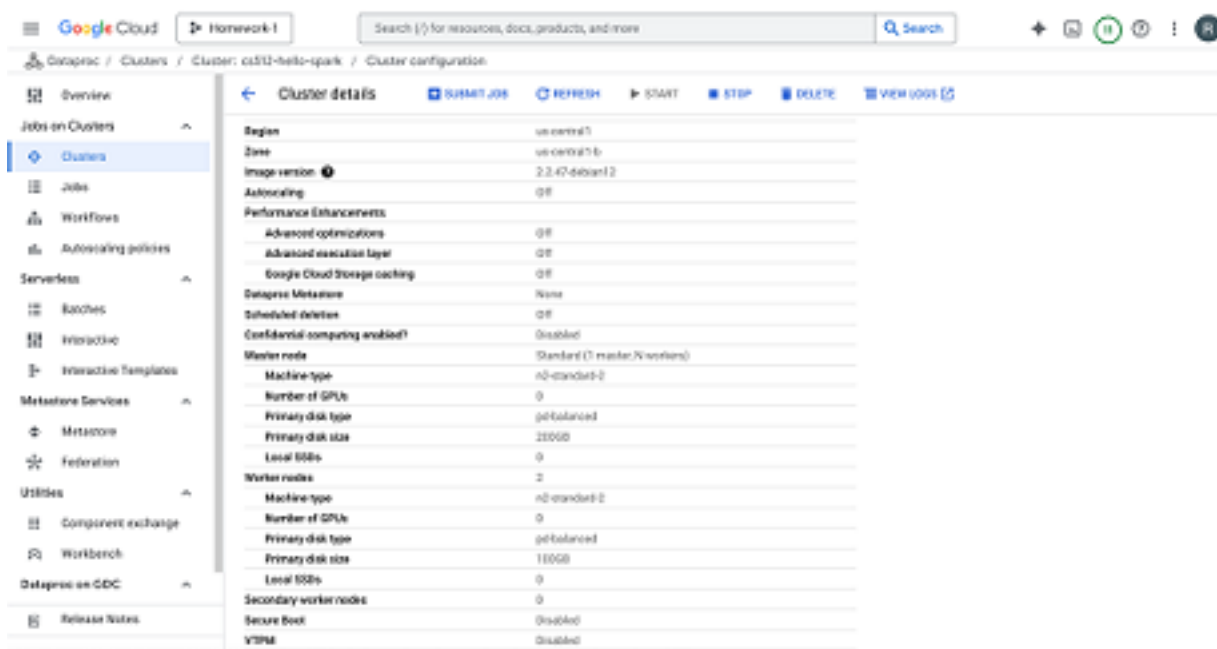


The screenshot shows the Google Cloud BigQuery console interface. The left sidebar contains navigation links for BigQuery, Explorer, and various management tools. The main panel displays the 'plane\_data' table schema. The schema table lists the following fields:

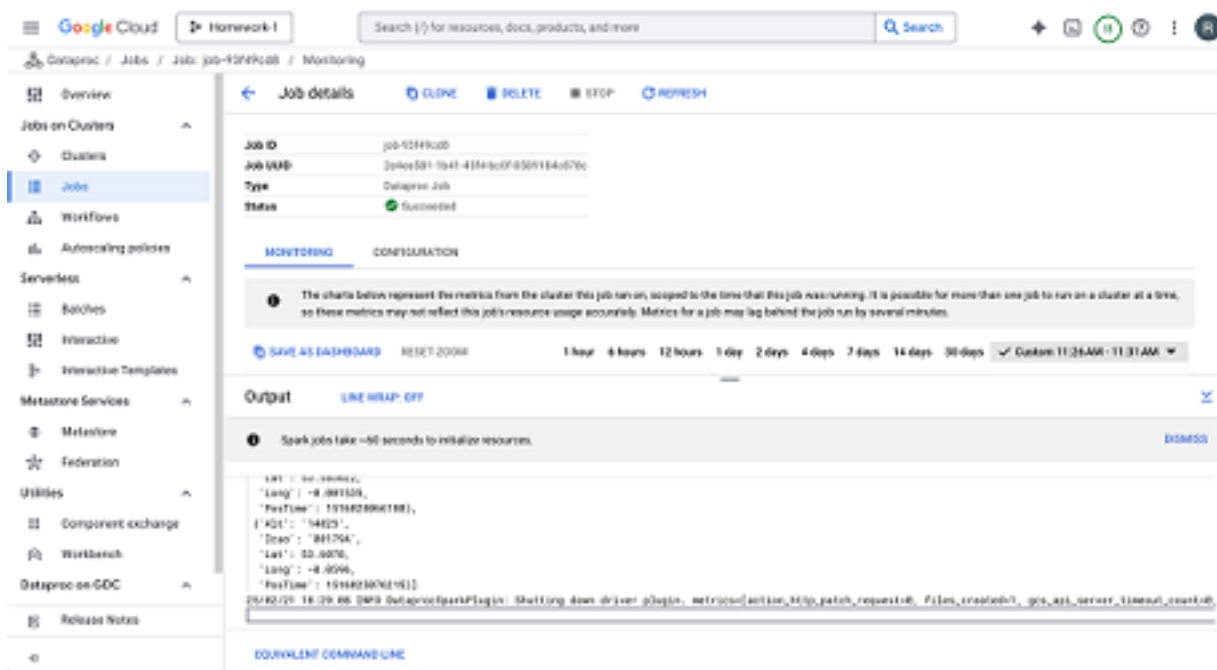
| Field name | Type    | Mode     | Key | Collation | Default Value | Policy Tags | Description |
|------------|---------|----------|-----|-----------|---------------|-------------|-------------|
| icao       | STRING  | NULLABLE | --  | --        | --            | --          | --          |
| PosTime    | INTEGER | NULLABLE | --  | --        | --            | --          | --          |
| Lat        | FLOAT   | NULLABLE | --  | --        | --            | --          | --          |
| Long       | FLOAT   | NULLABLE | --  | --        | --            | --          | --          |
| Alt        | INTEGER | NULLABLE | --  | --        | --            | --          | --          |

Below the schema table, there are buttons for 'EDIT SCHEMA' and 'VIEW NEW ACCESS POLICIES'. At the bottom, a 'Job history' section is partially visible.

With this dataset imported, the next step in this project was to set up the cluster for analyzing the data. The cluster was set up with the following parameters



This cluster was then used in the next steps, to test a PySpark job to extract 5 samples of the data.



### 3. Scrub

The project began by setting up a cluster of Spark parallelizations within Google Dataproc for the scrub step. The cluster was configured with 2 manager vCPUs, each with 8 GB of RAM and 166 GB of storage. Additionally, we set up 2 worker nodes, each with 2 vCPUs, 8 GB of RAM, and 83 GB of storage.

We then processed the data (previously prepared in an earlier module) in Google Dataprep, ensuring the fields Icao, Lat, Long, and PosTime were correctly formatted. Once this was validated, the data was uploaded to Google BigQuery.

Next, we focused on the Python script (see included block). To start, we imported various libraries: pyspark,

SparkSession (from pyspark.sql), pprint, json, StructType, FloatType, LongType, StringType, StructField (from pyspark.sql.types), Window, radians, cos, sin, asin, sqrt (from math), and functions like lead, udf, struct, and col (from pyspark.sql.functions). For later stages, we incorporated the Haversine function and conversions from strings to integers and floats.

We initiated a Spark context for configuration and set up the cluster to pull data from BigQuery. To ensure the correct dataset was used, we updated the project settings via MapReduce for the project, dataset, and table IDs. From the imported table, we pulled the values and returned them as JSON object strings. These strings were then mapped to JSON objects, and values were converted to integers and floats using the To\_numb function.

Next, we created a schema for our DataFrame, which included the relevant fields: Icao, Lat, Long, and PosTime. We partitioned the DataFrame into six groups and returned the first five as a sample for proof of functionality. Afterward, we deleted the temporary files created during the process.

The .py file was then saved and uploaded to our project directory in Google Cloud Storage, alongside the data file.

Finally, we used Google Dataproc to run the files in parallel on the cluster. Critical setup parameters included the main Python file (which accessed the data file in BigQuery), the JAR file, the region (us-east1), and the cluster.

```
# Oregon State University
# CS 512 - hello_Spark
# Date: 2025/02/21
# Author: Paul J Anderson & Rachel Hughes - Starter code provided by Justin Wolford

import pyspark
from pyspark.sql import SparkSession
import pprint
import json
from pyspark.sql.types import StructType, FloatType, LongType, StringType, StructField
from pyspark.sql import Window
from math import radians, cos, sin, asin, sqrt
from pyspark.sql.functions import lead, udf, struct, col

### haversine distance
def haversine(lon1, lat1, lon2, lat2):
    """
    Calculate the great circle distance between two points
    on the earth (specified in decimal degrees)
    """
    # convert decimal degrees to radians
    lon1, lat1, lon2, lat2 = map(radians, [lon1, lat1, lon2, lat2])
    # haversine formula
    dlon = lon2 - lon1
    dlat = lat2 - lat1
    a = sin(dlat/2)**2 + cos(lat1) * cos(lat2) * sin(dlon/2)**2
    c = 2 * asin(sqrt(a))
    r = 6371 # Radius of earth in kilometers. Use 3956 for miles
    return float(c * r)

def To_numb(x):
    x['PosTime'] = int(x['PosTime'])
    # x['FSeen'] = int(x['FSeen'])
    x['Lat'] = float(x['Lat'])
```

```

    x['Long1'] = float(x['Long1'])
    return x

sc = pyspark.SparkContext()

#PACKAGE_EXTENSIONS= ('gs://hadoop-lib/bigquery/bigquery-connector-hadoop2-latest.jar')

bucket = sc._jsc.hadoopConfiguration().get('fs.gs.system.bucket')
project = sc._jsc.hadoopConfiguration().get('fs.gs.project.id')
input_directory = 'gs://{}/hadoop/tmp/bigquery/pyspark_input'.format(bucket)
output_directory = 'gs://{}/pyspark_demo_output'.format(bucket)

spark = SparkSession \
    .builder \
    .master('yarn') \
    .appName('flights') \
    .getOrCreate()

#update with your project specific settings
conf={
    'mapred.bq.project.id':project,
    'mapred.bq.gcs.bucket':bucket,
    'mapred.bq.temp.gcs.path':input_directory,
    'mapred.bq.input.project.id': 'cs512-447721',
    'mapred.bq.input.dataset.id': 'aircraft_data',
    'mapred.bq.input.table.id': 'plane_data',
}

## pull table from big query
table_data = sc.newAPIHadoopRDD(
    'com.google.cloud.hadoop.io.bigquery.JsonTextBigQueryInputFormat',
    'org.apache.hadoop.io.LongWritable',
    'com.google.gson.JsonObject',
    conf = conf)

## convert table to a json like object, turn PosTime and Fseen back into numbers
vals = table_data.values()
pprint.pprint(vals.take(5)) #added to help debug whether table was loaded
vals = vals.map(lambda line: json.loads(line))
vals = vals.map(To_num)

##schema
schema = StructType([
    # StructField('FSeen', LongType(), True),
    StructField("Icao", StringType(), True),
    StructField("Lat", FloatType(), True),
    StructField("Long1", FloatType(), True),
    StructField("PosTime", LongType(), True)])

## create a dataframe object
df1 = spark.createDataFrame(vals, schema= schema)

df1.repartition(6)

```

```
pprint.pprint(vals.take(5))
```

```
## deletes the temporary files
input_path = sc._jvm.org.apache.hadoop.fs.Path(input_directory)
input_path.getFileSystem(sc._jsc.hadoopConfiguration()).delete(input_path, True)
```

#### 4. Explore

This week's analysis did not require exploration, but the test output of five samples of data is given here as proof of completion.

Output:

```
25/02/21 18:28:58 INFO FileInputFormat: Total input files to process : 24
```

```
['{"Icao": "000E28", "PosTime": "1516025251286", "Lat": 53.9995, "Long": 0.0612, "Alt": "30036"}',
 '{"Icao": "001003", "PosTime": "1516023228314", "Lat": 0, "Long": 0, "Alt": "1049097"}',
 '{"Icao": "001003", "PosTime": "1516039338859", "Lat": 0, "Long": 0}',
 '{"Icao": "00179A", "PosTime": "1516028066180", "Lat": 53.583022, "Long": -0.001535, "Alt": "2750"}',
 '{"Icao": "00179A", "PosTime": "1516025076215", "Lat": 53.6078, "Long": -0.0596, "Alt": "14025"}']

[{'Alt': '30036',
 'Icao': '000E28',
 'Lat': 53.9995,
 'Long': 0.0612,
 'PosTime': 1516025251286},
 {'Alt': '1049097',
 'Icao': '001003',
 'Lat': 0.0,
 'Long': 0.0,
 'PosTime': 1516023228314},
 {'Icao': '001003', 'Lat': 0.0, 'Long': 0.0, 'PosTime': 1516039338859},
 {'Alt': '2750',
 'Icao': '00179A',
 'Lat': 53.583022,
 'Long': -0.001535,
```

```
'PosTime': 1516028066180},  
  
{ 'Alt': '14025',  
  
'Icao': '00179A',  
  
'Lat': 53.6078,  
  
'Long': -0.0596,  
  
'PosTime': 1516025076215}]
```

## 5. Model

Modeling was not part of this work.

## 6. iNterpret

Interpretation was not covered in this work.

## Obstacles Encountered in Work

Much of the assignment went smoothly. We were able to set up the clusters without any issues and work through the provided examples. However, we encountered several obstacles:

A. A 404 error occurred when trying to use the renamed `plane_data_parsed`. We modified the `.py` file to reference `plane_data` instead, which resolved the issue.

B. An `IllegalArgumentError` occurred due to a format mismatch, as my file was a CSV rather than a BigQuery file. This was not resolved easily. We iterated through several code modifications to import the CSV, but none were successful. After reviewing the documentation and lectures, we realized that the code chunk for pulling the table from BigQuery was intended to pull a BigQuery file and convert it to JSON objects. We reverted the code to its original version before any modifications.

C. We received a “Table is already external” error pointing to `table_data = sc.newAPIHadoopRDD`. This led us to believe the issue was with the dataset schema. Upon investigation, the dataset metadata was significantly different, indicating the schema had not been properly configured. We went through the upload process to BigQuery again with the assistance of the TA. This newly structured dataset now followed the typical configuration used by other students and partners.

D. The same error from step C occurred again. To resolve it, we deleted the Hadoop folder in Google Cloud Storage and reran the job. This time, the job executed as expected and produced the desired output, as seen above.

## Distribution of Work

Both Rachel and Paul completed all steps of the process. We consulted on difficulties along the way. The final report was split out into sections with Rachel and Paul completing different parts of the write-up.

## References

Wolford, J. (2025). Spark. Oregon State University. Retrieved February 21, 2025, from [https://canvas.oregonstate.edu/courses/1988535/assignments/9919261?module\\_item\\_id=25112253](https://canvas.oregonstate.edu/courses/1988535/assignments/9919261?module_item_id=25112253)

ADS-B Exchange. (2025). What is the Exchange? ADS-B Exchange. Retrieved February 21, 2025, from <https://www.adsbexchange.com/what-is-the-exchange/> Apache Software Foundation. (2025). Apache Spark™ 3.5.4 documentation. Retrieved February 21, 2025, from <https://spark.apache.org/docs/3.5.4/>

## **Coding Sources**

Wolford, J. (2025). Spark. Oregon State University. Retrieved February 21, 2025, from [https://canvas.oregonstate.edu/courses/1988535/assignments/9919261?module\\_item\\_id=25112253](https://canvas.oregonstate.edu/courses/1988535/assignments/9919261?module_item_id=25112253)

Apache Software Foundation. (2025). Apache Spark™ 3.5.4 documentation. Retrieved February 21, 2025, from <https://spark.apache.org/docs/3.5.4/>