**DATAPROC == EARTHQUAKE PROJECT**

Project files

* Dataflow
* historical.py
* daily.py
* utility.py
* configuration.py
* test.py
* Dataproc
  + - historical.py
    - daily.py
    - utility.py
    - configuration.py
    - test.py
* Requirement.txt

python==3.9.13  
pyspark==3.3.2  
apache-airflow[gcp]  
apache-beam[gcp]  
google-cloud-storage  
google-cloud-bigquery  
numpy  
pandas

* Project\_details\_pdf

1. **Task 1: Read Data from an API and Load it into a GCS Bucket**

* **Objective**: Fetch data from a specified API and upload it to a Google Cloud Storage (GCS) bucket in its original JSON format.

**Steps**:

* **Read JSON Data: Use the requests library to read JSON data from the API.**
* **Upload to GCS: After retrieving the JSON data, use the google-cloud-storage client library to upload the data to a specified GCS location in JSON format.**
* **Workflow Diagram**

**Fetch JSON Data from API using requests library0**

**Data Retrieved**

**in JSON Format**

**Upload Data to GCS Bucket using GCS Client**

**Start**

* **Optimization technique**
* **Task 2: Read Data from GCS Bucket and Flatten the Existing JSON Format**

**Objective**: Fetch data from a Google Cloud Storage (GCS) bucket and flatten the existing JSON into the following format:

{"mag": 0.89,

"place": "6 km NW of The Geysers, CA",

"time": 1729308248850,

"updated": 1729308343908,

"tz": null,

"url": "https://earthquake.usgs.gov/earthquakes/eventpage/nc75076006",

"detail": "https://earthquake.usgs.gov/earthquakes/feed/v1.0/detail/nc75076006.geojson",

"felt": null,

"cdi": null,

"mmi": null,

"alert": null,

"status": "automatic",

"tsunami": 0,

"sig": 12,

"net": "nc",

"code": "75076006",

"ids": ",nc75076006,",

"sources": ",nc,",

"types": ",nearby-cities,origin,phase-data,",

"nst": 9,

"dmin": 0.01303,

"rms": 0.02,

"gap": 77,

"magType": "md",

"type": "earthquake",

"title": "M 0.9 - 6 km NW of The Geysers, CA",

"longtitude":-122.813163757324,

"latitude":38.8125,

"depth": 3.25999999046326}

* **Read data from the GCS bucket using the Python google-cloud-storage client library.**
* **Extract the features key dictionary from the data, which contains the actual information.**
* **Flatten the dictionary structure using the dictionary keys to achieve the required format.**

**Workflow Diagram**

**Fatten the data with required column**

**Read data from GCS bucket**

**Extract feature key from dictionary**

**Optimization technique**

* **Task 3: Perform Transformations, and Upload to GCS and BigQuery**

**Objective:** After flattening the data, perform the required transformations on the DataFrame, and then upload the data to both Google Cloud Storage (GCS) in Parquet format and to a BigQuery table.

* **Create a DataFrame with the flattened data and define the required schema.**
* **Perform transformations on the DataFrame:**
  + **Convert the date and updated columns (in epoch time format) into timestamps.**
  + **Extract the region from the place column.**
  + **Add an insert\_date column at the end of the DataFrame to indicate the time of insertion.**
* **Write the transformed DataFrame to GCS in Parquet format.**
* **Write the transformed DataFrame to BigQuery using a JDBC connection (with the appropriate JAR connection file).**

**Workflow Diagram**

**Create**

**dataframe**

**Write to biqquery table**

**Parquet upload to GCS**

**Transformation as per required**

**Optimization technique**

**Note:**

**For historical loads, we can perform the operation manually. After that, we can automate the process for daily data loads using Composer, Workflow Templates Dataproc, or Cloud Scheduler.**