# Documentation for Earthquake Data Analysis Using Dataflow

This documentation provides an overview of the data pipeline for processing and transforming earthquake data using Apache Beam and Google Cloud Platform (GCP) services. The pipeline fetches data from a URL, processes and transforms it, writes it to Google Cloud Storage (GCS) in different formats, and finally loads the data into BigQuery for analysis.

**Project Files:**

1. dataflow\_load\_historical\_data.py
2. dynamic\_methods.py
3. schema\_data.py
4. util.py

### Libraries Used:

* **apache\_beam**: Used for distributed data processing and building the pipeline.
* **google.cloud.storage**: For performing operations related to Google Cloud Storage (GCS), such as reading and writing files to GCS.
* **google.cloud.bigquery**: For interacting with BigQuery to load data after transformation.
* **pytest**: For writing unit tests to validate the correctness of transformations and pipeline steps.
* **json**: For parsing and handling JSON data.
* **datetime**: For handling date/time operations and formatting strings (e.g., for dynamic date-based file naming).
* **logging**: For logging the pipeline's processing steps, errors, and other useful information.

### ****Pipeline Overview****

The pipeline follows a series of stages, each performing specific transformations and actions on the data. The stages include:

1. Fetching data from a remote API (USGS Earthquake Feed).
2. Writing the fetched data to GCS in JSON format.
3. Processing and transforming the data (feature extraction, flattening, and time conversion).
4. Writing the transformed data to GCS as Parquet format.
5. Loading the Parquet data from GCS into BigQuery for further analysis.

### ****Pipeline Flow****

#### ****Step 1: Fetching Data from URL****

* **Objective**: Fetch earthquake data from a remote API endpoint.
* **Process**:
  + The pipeline initiates an HTTP request to fetch earthquake data from the USGS Earthquake Feed (GeoJSON format).
  + The FetchDataFromUrl method is responsible for performing the HTTP GET request and processing the response.
  + The result is output to the pipeline, where errors are captured and handled separately from the successful fetch.

**Optimization Techniques**:

* + The API fetch operation is executed asynchronously, and error handling ensures only successful data is passed to the next steps of the pipeline.

#### ****Step 2: Writing Data to GCS as JSON****

* **Objective**: Store the raw earthquake data in JSON format in Google Cloud Storage (GCS).
* **Process**:
  + The data fetched in Step 1 is written to GCS using WriteToText in JSON format.
  + Data is written without sharding and with a single output file for simplicity.

**Optimization Techniques**:

* + The pipeline avoids unnecessary file splitting by specifying a single shard (num\_shards=1), ensuring a single output file is created.
  + File naming and suffixing are controlled to ensure clean, consistent output.

#### ****Step 3: Data Processing and Transformation****

* **Objective**: Process and transform the earthquake data for downstream analysis.
* **Process**:
  + The data stored as JSON is read from GCS.
  + Several transformations are applied:
    1. **Extract Features**: Feature extraction is performed using ExtractFeatures.
    2. **Flatten Data**: Complex nested JSON structures are flattened using FlattenJSON.
    3. **Unix to IST**: Timestamps are converted from Unix format to Indian Standard Time (IST) using UnixToIst.
    4. **Adding Area Field**: Geographic data is enriched with the area field.
    5. **Adding Ingestion Date**: A timestamp for when the data was ingested is added using IngestionDate.

**Optimization Techniques**:

* + Data transformations are modularized into individual steps, ensuring better maintainability and parallel execution of each step.
  + By splitting the transformations into separate stages, the pipeline can scale better across distributed environments like GCP Dataproc.

#### ****Step 4: Writing Transformed Data to GCS as Parquet****

* **Objective**: Store the transformed data in Parquet format in GCS for efficient storage and querying.
* **Process**:
  + The transformed data is written to GCS as a Parquet file.
  + Parquet is a columnar format that is optimized for analytical queries and minimizes storage costs.
  + A schema for the Parquet file is defined using SchemaConverter to ensure consistency in the output data structure.

**Optimization Techniques**:

* + Parquet format is used for its efficiency in terms of both storage and read performance in BigQuery.
  + Data is written with a single shard (num\_shards=1) to avoid unnecessary shuffling of data.

#### ****Step 5: Writing Parquet Data from GCS to BigQuery****

* **Objective**: Load the Parquet data from GCS into BigQuery for analysis and reporting.
* **Process**:
  + The Parquet data stored in GCS is read into the pipeline.
  + The data is then written into a BigQuery table, creating the table if it doesn’t exist and overwriting the existing data (if any) using WRITE\_TRUNCATE.
  + The schema for the BigQuery table is defined using SchemaConverter.

**Optimization Techniques**:

* + **WRITE\_TRUNCATE** ensures that the table is replaced with the most up-to-date dataset.
  + The use of Parquet data ensures the data is efficiently processed in BigQuery.
  + The schema is predefined and dynamically managed, ensuring that the data written to BigQuery is well-structured and consistent.

### ****Pipeline Execution Process****

1. **Set Pipeline Options**:
   * Initialize the PipelineOptions and GoogleCloudOptions to define the Google Cloud project, region, and temporary locations used in the pipeline.
2. **Fetch Data from API**:
   * Fetch the earthquake data from the USGS API.
   * Capture successful and error results for further processing.
3. **Write Data to GCS as JSON**:
   * Store the raw JSON data in GCS for subsequent stages.
4. **Process and Transform Data**:
   * Read the raw JSON data from GCS and apply various transformations (flattening, timestamp conversion, feature extraction, etc.).
5. **Write Transformed Data to GCS as Parquet**:
   * Write the transformed data in Parquet format back to GCS.
6. **Write Data to BigQuery**:
   * Load the Parquet data into BigQuery for analysis.

### ****Error Handling and Logging****

* **Error Handling**: Errors during the fetch operation are captured separately, ensuring that only successful data is processed further.
* **Logging**: Logging at each stage provides insights into the pipeline’s progress, helping with debugging and tracking execution.

### ****Additional Notes****

* **Bucket Creation**: The create\_bucket function ensures that the GCS bucket exists before the pipeline starts. If the bucket doesn't already exist, it will be created.
* **Date Management**: The pipeline uses the current date (datetime.now().strftime('%Y%m%d')) to dynamically name files and organize data by date.