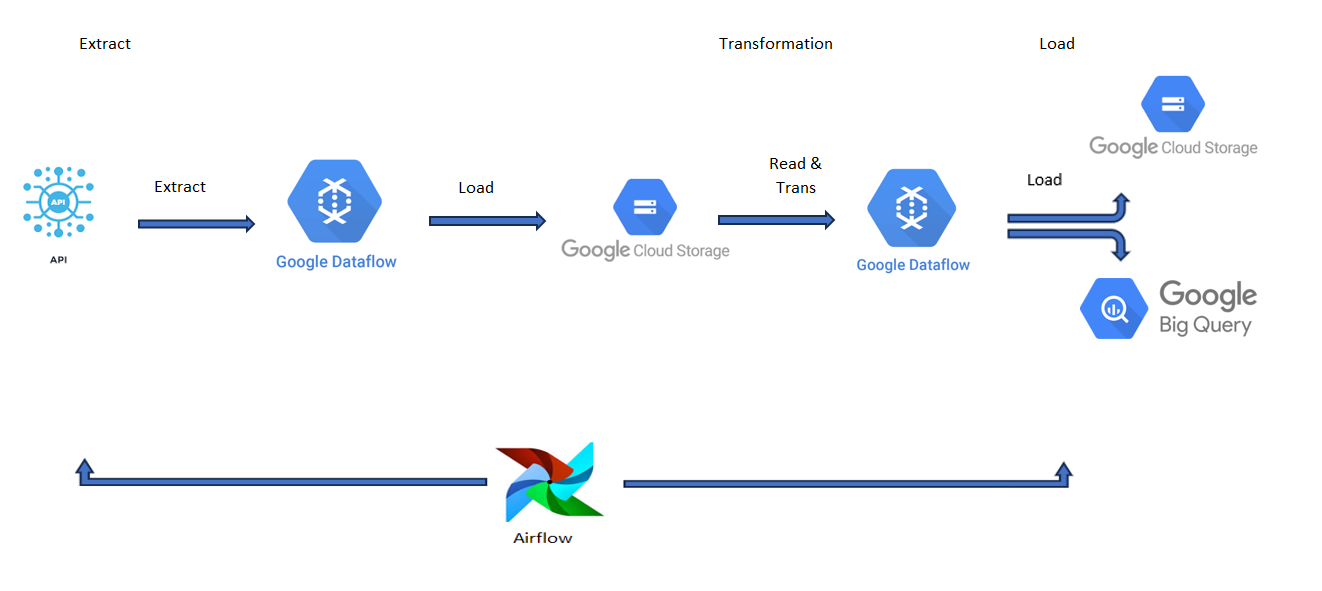
𝗘𝗮𝗿𝘁𝗵𝗾𝘂𝗮𝗸𝗲 𝗗𝗮𝘁𝗮 𝗨𝗦𝗚𝗦 API Data Using

DataFlow

Mohini Zurange

**Architecture:**

****

**Task1** **: Extract data from earthquake API and stored it in GCS**

* **Description**: create dataflow pipeline for extract all data from Given earthquake API and as it stores in GCS bucket.
* **Flow Diagram:**

**Extract data from API**

**(ParDO class use response lib)**

**Write the Pcollection in gcs bucket by passing explicit schema**

**P** Collection**(dict)**

**Create data flow pipeline P1**

**Api URL**

* **Inputs**: API URI
* **Output:** Pcollection in dict format and write in GCS bucket in parquet format***('gs://earthquake\_analysis\_buck/dataflow/landing/{cur\_timestamp}/earthquake'***)

|  |  |  |
| --- | --- | --- |
| **Task Name** | **Description** | **Transformation Details** |
| **StartPipeline** | Initializes the pipeline with a dummy element | beam.Create([None]) is used as a placeholder to start the pipeline without actual data at this step. |
| **Extract Data from API** | Retrieves data from an API endpoint | beam.ParDo(ExtractDataFormAPI(), api\_url) sends a request to the specified api\_url, parses the response, and processes the API data for further pipeline steps. |
| **Extracted Data Write to GCS** | Writes extracted data to GCS in Parquet format | beam.io.WriteToParquet(landing\_location, schema=raw\_parquet\_schema) writes the data to the specified GCS location in Parquet format, using the defined schema and specified file names. |

**Input data format:**

{"type":"FeatureCollection","metadata":{"generated":1730283889000,"url":"https://earthquake.usgs.gov/earthquakes/feed/v1.0/summary/all\_month.geojson","title":"USGS All Earthquakes, Past Month","status":200,"api":"1.14.1","count":8530},"features":[{"type":"Feature","properties":{"mag":1.36,"place":"10 km E of Ojai, CA","time":1730283707570,"updated":1730283833587,"tz":null,"url":"https://earthquake.usgs.gov/earthquakes/eventpage/ci40966712","detail":"https://earthquake.usgs.gov/earthquakes/feed/v1.0/detail/ci40966712.geojson","felt":null,"cdi":null,"mmi":null,"alert":null,"status":"automatic","tsunami":0,"sig":28,"net":"ci","code":"40966712","ids":",ci40966712,","sources":",ci,","types":",nearby-cities,origin,phase-data,scitech-link,","nst":16,"dmin":0.02225,"rms":0.18,"gap":51,"magType":"ml","type":"earthquake","title":"M 1.4 - 10 km E of Ojai, CA"},"geometry":{"type":"Point","coordinates":[-119.1324997,34.4588318,8.54]},"id":"ci40966712"},

{"type":"Feature","properties":{"mag":1.3,"place":"5 km SE of Ontario, CA","time":1730283480840,"updated":1730283611971,"tz":null,"url":"https://earthquake.usgs.gov/earthquakes/eventpage/ci40966704","detail":"https://earthquake.usgs.gov/earthquakes/feed/v1.0/detail/ci40966704.geojson","felt":null,"cdi":null,"mmi":null,"alert":null,"status":"automatic","tsunami":0,"sig":26,"net":"ci","code":"40966704","ids":",ci40966704,","sources":",ci,","types":",nearby-cities,origin,phase-data,scitech-link,","nst":36,"dmin":0.02056,"rms":0.11,"gap":60,"magType":"ml","type":"earthquake","title":"M 1.3 - 5 km SE of Ontario, CA"},"geometry":{"type":"Point","coordinates":[-117.5936661,34.0159988,5.06]},"id":"ci40966704"},

{"type":"Feature","properties":{"mag":2.4,"place":"6 km SE of Houston, Alaska","time":1730281770240,"updated":1730282198448,"tz":null,"url":"https://earthquake.usgs.gov/earthquakes/eventpage/ak024dyx0yzk","detail":"https://earthquake.usgs.gov/earthquakes/feed/v1.0/detail/ak024dyx0yzk.geojson","felt":1,"cdi":2,"mmi":null,"alert":null,"status":"automatic","tsunami":0,"sig":89,"net":"ak","code":"024dyx0yzk","ids":",ak024dyx0yzk,","sources":",ak,","types":",dyfi,origin,phase-data,","nst":null,"dmin":null,"rms":0.36,"gap":null,"magType":"ml","type":"earthquake","title":"M 2.4 - 6 km SE of Houston, Alaska"},"geometry":{"type":"Point","coordinates":[-149.7428,61.586,39]},"id":"ak024dyx0yzk"},

.

.}

**Above data write in gcp bucket but in parquet file for that we need to pass explicit schema:**

**raw\_parquet\_schema** = pyarrow.schema([  
 ('type', pyarrow.string()), # Type of the GeoJSON (FeatureCollection)  
 ('metadata', pyarrow.struct([  
 ('generated', pyarrow.int64()),  
 ('url', pyarrow.string()),  
 ('title', pyarrow.string()),  
 ('status', pyarrow.int64()),  
 ('api', pyarrow.string()),  
 ('count', pyarrow.int64())  
 ])), # End of 'metadata' struct  
 ('features', pyarrow.list\_(  
 pyarrow.struct([  
 ('type', pyarrow.string()), # Type of the feature  
 ('properties', pyarrow.struct([  
 ('mag', pyarrow.float64()), # Magnitude  
 ('place', pyarrow.string()), # Location  
 ('time', pyarrow.int64()), # Timestamp  
 ('updated', pyarrow.int64()), # Update timestamp  
 ('tz', pyarrow.int64()), # Time zone  
 ('url', pyarrow.string()), # URL for the event  
 ('detail', pyarrow.string()), # Detail URL  
 ('felt', pyarrow.float64()), # Felt reports  
 ('cdi', pyarrow.float64()), # Community Internet Intensity  
 ('mmi', pyarrow.float64()), # Maximum Intensity  
 ('alert', pyarrow.string()), # Alert level  
 ('status', pyarrow.string()), # Event status  
 ('tsunami', pyarrow.int64()), # Tsunami indicator  
 ('sig', pyarrow.int64()), # Significance  
 ('net', pyarrow.string()), # Network  
 ('code', pyarrow.string()), # Event code  
 ('ids', pyarrow.string()), # IDs associated with the event  
 ('sources', pyarrow.string()), # Sources  
 ('types', pyarrow.string()), # Types of data  
 ('nst', pyarrow.int64()), # Number of stations  
 ('dmin', pyarrow.float64()), # Minimum distance to event  
 ('rms', pyarrow.float64()), # Root Mean Square  
 ('gap', pyarrow.int64()), # Gap  
 ('magType', pyarrow.string()), # Magnitude type  
 ('type', pyarrow.string()), # Type of earthquake  
 ('title', pyarrow.string()) # Title of the event  
 ])), # End of 'properties' struct  
 ('geometry', pyarrow.struct([  
 ('type', pyarrow.string()), # Type of geometry  
 ('coordinates', pyarrow.list\_(pyarrow.float64())) # Coordinates  
 ])), # End of 'geometry' struct  
 ('id', pyarrow.string()) # ID of the feature  
 ]) # End of feature struct  
 )) # End of features list  
])

**Task 2** **: Create new pipeline and Read raw data from GCS and select specific col and apply transformations**

* **Description**: create separate pipeline for read the data from gcs and did flattening and apply the transformations
* **Flow diagram:**

**Create Dataflow pipeline P2**

**Read File from gcs bucket**

**Flatten the data**

Appy transformations

|  |  |  |
| --- | --- | --- |
| **Task Name** | **Description** | **Transformation Details** |
| Read data from GCS landing location | Reads the Parquet data from the specified GCS location( ***'gs://earthquake\_analysis\_buck/dataflow/landing/{cur\_timestamp}/earthquake'***  ) | Uses beam.io.ReadFromParquet |
| Fetch required data and flatten it | Extract needed fields and flattens nested data | ExtractRequiredDataAndFlatten via ParDo, selecting fields and handling nested structures |
| Apply transformations on flattened data | Applies additional transformations to the flattened data | ApplyTransformation via ParDo, which could include calculations, string manipulations, filtering, etc. |

* **Task information**

**Flatten the data:** select only specific col

**Input data for flatten:**

{"type":"FeatureCollection","metadata":{"generated":1730283889000,"url":"https://earthquake.usgs.gov/earthquakes/feed/v1.0/summary/all\_month.geojson","title":"USGS All Earthquakes, Past Month","status":200,"api":"1.14.1","count":8530},"features":[{"type":"Feature","properties":{"mag":1.36,"place":"10 km E of Ojai, CA","time":1730283707570,"updated":1730283833587,"tz":null,"url":"https://earthquake.usgs.gov/earthquakes/eventpage/ci40966712","detail":"https://earthquake.usgs.gov/earthquakes/feed/v1.0/detail/ci40966712.geojson","felt":null,"cdi":null,"mmi":null,"alert":null,"status":"automatic","tsunami":0,"sig":28,"net":"ci","code":"40966712","ids":",ci40966712,","sources":",ci,","types":",nearby-cities,origin,phase-data,scitech-link,","nst":16,"dmin":0.02225,"rms":0.18,"gap":51,"magType":"ml","type":"earthquake","title":"M 1.4 - 10 km E of Ojai, CA"},"geometry":{"type":"Point","coordinates":[-119.1324997,34.4588318,8.54]},"id":"ci40966712"},

**After flatten:**

**{'mag': 5.9, 'place': '111 km E of Petropavlovsk-Kamchatsky, Russia', 'time': 1727698918571, 'updated': 1729946934210, 'tz': None, 'url': 'https://earthquake.usgs.gov/earthquakes/eventpage/us7000nh82', 'detail': 'https://earthquake.usgs.gov/earthquakes/feed/v1.0/detail/us7000nh82.geojson', 'felt': None, 'cdi': None, 'mmi': 4.868, 'alert': 'green', 'status': 'reviewed', 'tsunami': 0, 'sig': 536, 'net': 'us', 'code': '7000nh82', 'ids': ',us7000nh82,', 'sources': ',us,', 'types': ',losspager,moment-tensor,origin,phase-data,shakemap,', 'nst': 103, 'dmin': 0.989, 'rms': 1.46, 'gap': 76, 'magType': 'mww', 'type': 'earthquake', 'title': 'M 5.9 - 111 km E of Petropavlovsk-Kamchatsky, Russia', 'geometry': {'longitude': 160.2866, 'latitude': 53.0601, 'depth': 30.0}}**

**Transformation details**:

1)Convert(time,update column) UNIX timestamps( in milliseconds )to timestamp(Convert milliseconds to seconds and then to readable timestamp)  
2) Generate column “area” - based on existing “place” column,

3) Add insert date column

**Input data for transformation:**

**{'mag': 5.9, 'place': '111 km E of Petropavlovsk-Kamchatsky, Russia', 'time': 1727698918571, 'updated': 1729946934210, 'tz': None, 'url': 'https://earthquake.usgs.gov/earthquakes/eventpage/us7000nh82', 'detail': 'https://earthquake.usgs.gov/earthquakes/feed/v1.0/detail/us7000nh82.geojson', 'felt': None, 'cdi': None, 'mmi': 4.868, 'alert': 'green', 'status': 'reviewed', 'tsunami': 0, 'sig': 536, 'net': 'us', 'code': '7000nh82', 'ids': ',us7000nh82,', 'sources': ',us,', 'types': ',losspager,moment-tensor,origin,phase-data,shakemap,', 'nst': 103, 'dmin': 0.989, 'rms': 1.46, 'gap': 76, 'magType': 'mww', 'type': 'earthquake', 'title': 'M 5.9 - 111 km E of Petropavlovsk-Kamchatsky, Russia', 'geometry': {'longitude': 160.2866, 'latitude': 53.0601, 'depth': 30.0}}……..**

**After transformations data:**

**{'mag': 5.9, 'place': '111 km E of Petropavlovsk-Kamchatsky, Russia', 'time': '2024-09-30 12:21:58', 'updated': '2024-10-26 12:48:54', 'tz': None, 'url': 'https://earthquake.usgs.gov/earthquakes/eventpage/us7000nh82', 'detail': 'https://earthquake.usgs.gov/earthquakes/feed/v1.0/detail/us7000nh82.geojson', 'felt': None, 'cdi': None, 'mmi': 4.868, 'alert': 'green', 'status': 'reviewed', 'tsunami': 0, 'sig': 536, 'net': 'us', 'code': '7000nh82', 'ids': ',us7000nh82,', 'sources': ',us,', 'types': ',losspager,moment-tensor,origin,phase-data,shakemap,', 'nst': 103, 'dmin': 0.989, 'rms': 1.46, 'gap': 76, 'magType': 'mww', 'type': 'earthquake', 'title': 'M 5.9 - 111 km E of Petropavlovsk-Kamchatsky, Russia', 'area': 'Petropavlovsk-Kamchatsky, Russia', 'geometry': {'longitude': 160.2866, 'latitude': 53.0601, 'depth': 30.0}, 'insert\_date': '2024-10-30 12:19:42'}........**

**Task 3: Clean Data load to silver layer (GCS bucket) and big query(database)**

* **Description:** Writes cleaned data to the GCS silver layer in Parquet format and to BigQuery for analytics and reporting
* **Flow Diagram:**

Clean\_data\_schema

Clean data Pcollection

Write in Silver layer (GCS Bucket)

Write in Biq Query

Big query schema

**Task Information:**

|  |  |  |
| --- | --- | --- |
| **Task Name** | **Description** | **Transformation Details** |
| **Write Cleaned Data to GCS Silver** | Stores cleaned data in the GCS silver layer  ***('gs://earthquake\_analysis\_buck/dataflow/silver/{cur\_timestamp}/flatten\_earthquake\_data'***) | beam.io.WriteToParquet to write to silver\_gcs\_location with clean\_data\_parquet\_schema and .parquet file suffix |
| **Write Cleaned Data to BigQuery** | Loads cleaned data into BigQuery  ***('spark-learning-43150.earthquake\_db.dataflow\_earthquake\_data')*** | beam.io.WriteToBigQuery with output\_db, bq\_schema, WRITE\_APPEND, and CREATE\_IF\_NEEDED dispositions |

**Explicit Schema used:**

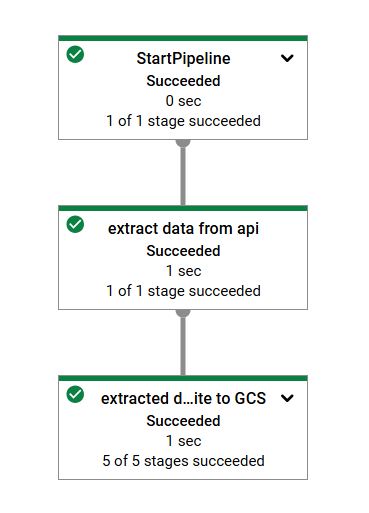
1. **Raw\_parquet\_schema**

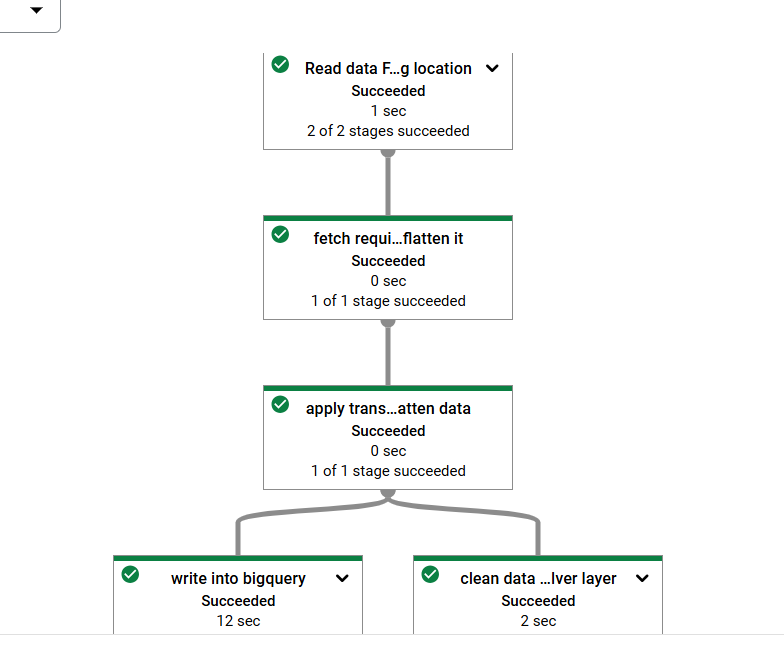
**raw\_parquet\_schema =** pyarrow.schema([  
 ('type', pyarrow.string()), # Type of the GeoJSON (FeatureCollection)  
 ('metadata', pyarrow.struct([  
 ('generated', pyarrow.int64()),  
 ('url', pyarrow.string()),  
 ('title', pyarrow.string()),  
 ('status', pyarrow.int64()),  
 ('api', pyarrow.string()),  
 ('count', pyarrow.int64())  
 ])), # End of 'metadata' struct  
 ('features', pyarrow.list\_(  
 pyarrow.struct([  
 ('type', pyarrow.string()), # Type of the feature  
 ('properties', pyarrow.struct([  
 ('mag', pyarrow.float64()), # Magnitude  
 ('place', pyarrow.string()), # Location  
 ('time', pyarrow.int64()), # Timestamp  
 ('updated', pyarrow.int64()), # Update timestamp  
 ('tz', pyarrow.int64()), # Time zone  
 ('url', pyarrow.string()), # URL for the event  
 ('detail', pyarrow.string()), # Detail URL  
 ('felt', pyarrow.float64()), # Felt reports  
 ('cdi', pyarrow.float64()), # Community Internet Intensity  
 ('mmi', pyarrow.float64()), # Maximum Intensity  
 ('alert', pyarrow.string()), # Alert level  
 ('status', pyarrow.string()), # Event status  
 ('tsunami', pyarrow.int64()), # Tsunami indicator  
 ('sig', pyarrow.int64()), # Significance  
 ('net', pyarrow.string()), # Network  
 ('code', pyarrow.string()), # Event code  
 ('ids', pyarrow.string()), # IDs associated with the event  
 ('sources', pyarrow.string()), # Sources  
 ('types', pyarrow.string()), # Types of data  
 ('nst', pyarrow.int64()), # Number of stations  
 ('dmin', pyarrow.float64()), # Minimum distance to event  
 ('rms', pyarrow.float64()), # Root Mean Square  
 ('gap', pyarrow.int64()), # Gap  
 ('magType', pyarrow.string()), # Magnitude type  
 ('type', pyarrow.string()), # Type of earthquake  
 ('title', pyarrow.string()) # Title of the event  
 ])), # End of 'properties' struct  
 ('geometry', pyarrow.struct([  
 ('type', pyarrow.string()), # Type of geometry  
 ('coordinates', pyarrow.list\_(pyarrow.float64())) # Coordinates  
 ])), # End of 'geometry' struct  
 ('id', pyarrow.string()) # ID of the feature  
 ]) # End of feature struct  
 )) # End of features list  
]) # End of schema  
 **2- clean\_data\_parquet\_schema**

**clean\_data\_parquet\_schema** = pyarrow.schema([  
 ('mag', pyarrow.float64()),  
 ('place', pyarrow.string()),  
 ('time', pyarrow.string()), # Timestamp in seconds  
 ('updated', pyarrow.string()), # Timestamp in seconds  
 ('tz', pyarrow.string()),  
 ('url', pyarrow.string()),  
 ('detail', pyarrow.string()),  
 ('felt', pyarrow.float64()),  
 ('cdi', pyarrow.float64()),  
 ('mmi', pyarrow.float64()),  
 ('alert', pyarrow.string()),  
 ('status', pyarrow.string()),  
 ('tsunami', pyarrow.int64()),  
 ('sig', pyarrow.int64()),  
 ('net', pyarrow.string()),  
 ('code', pyarrow.string()),  
 ('ids', pyarrow.string()),  
 ('sources', pyarrow.string()),  
 ('types', pyarrow.string()),  
 ('nst', pyarrow.int64()),  
 ('dmin', pyarrow.float64()),  
 ('rms', pyarrow.float64()),  
 ('gap', pyarrow.int64()),  
 ('magType', pyarrow.string()),  
 ('type', pyarrow.string()),  
 ('title', pyarrow.string()),  
 ('area', pyarrow.string()),  
 ('geometry', pyarrow.struct([  
 ('longitude', pyarrow.float64()),  
 ('latitude', pyarrow.float64()),  
 ('depth', pyarrow.float64())  
 ])),  
 ('insert\_date', pyarrow.string()) # Timestamp in seconds  
])

**3-Schema for big query  
bq\_schema** = {  
 "fields": [  
 {"name": "mag", "type": "FLOAT"},  
 {"name": "place", "type": "STRING"},  
 {"name": "time", "type": "TIMESTAMP"},  
 {"name": "updated", "type": "TIMESTAMP"},  
 {"name": "tz", "type": "STRING"},  
 {"name": "url", "type": "STRING"},  
 {"name": "detail", "type": "STRING"},  
 {"name": "felt", "type": "FLOAT"},  
 {"name": "cdi", "type": "FLOAT"},  
 {"name": "mmi", "type": "FLOAT"},  
 {"name": "alert", "type": "STRING"},  
 {"name": "status", "type": "STRING"},  
 {"name": "tsunami", "type": "INTEGER"},  
 {"name": "sig", "type": "INTEGER"},  
 {"name": "net", "type": "STRING"},  
 {"name": "code", "type": "STRING"},  
 {"name": "ids", "type": "STRING"},  
 {"name": "sources", "type": "STRING"},  
 {"name": "types", "type": "STRING"},  
 {"name": "nst", "type": "INTEGER"},  
 {"name": "dmin", "type": "FLOAT"},  
 {"name": "rms", "type": "FLOAT"},  
 {"name": "gap", "type": "INTEGER"},  
 {"name": "magType", "type": "STRING"},  
 {"name": "type", "type": "STRING"},  
 {"name": "title", "type": "STRING"},  
 {"name": "area", "type": "STRING"},  
 {"name": "geometry", "type": "RECORD", "fields": [  
 {"name": "longitude", "type": "FLOAT"},  
 {"name": "latitude", "type": "FLOAT"},  
 {"name": "depth", "type": "FLOAT"}  
 ]},  
 {"name": "insert\_date", "type": "TIMESTAMP"}  
 ]  
}

**DATAFLOW JOB :**

****

****

**##Audit table:  
Purpose:**

The audit\_event function logs the details of a pipeline task or job into a BigQuery audit table. It helps track and monitor the execution of tasks.

**Parameters:**

* bigquery\_audit\_tbl (str): BigQuery table name where the audit records are stored.
* job\_id (datetime): Unique identifier for the job or pipeline run.
* pipeline\_nm (str): Name of the pipeline.
* start\_time (datetime): Task start time.
* task\_name (str): Name of the task.
* end\_time (datetime): Task end time.
* status (str): Status of the task (e.g., "SUCCESS" or "FAILED").
* error\_msg (str, optional): Error message if the task failed.

Working:

* **Before execution**: We log the task's start time (start\_time) and task name.
* **Actual task run:** The task executes its logic (e.g., reading data, transforming data).
* **After execution**: We log the task's status (success or failure) and capture the end time (end\_time).
* **Send all details to Audit\_event function**: to insert the entry to audit\_log table