

# Data Engineering Technical Challenge Solution

## Scenarios:

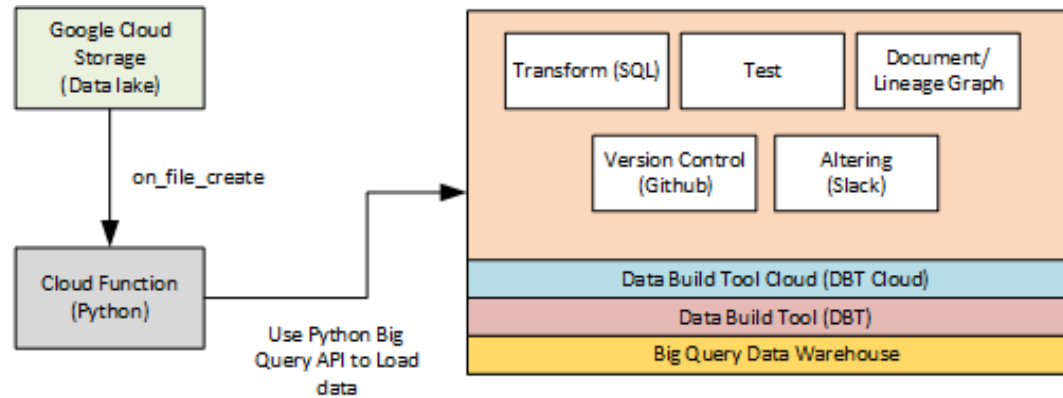
1. Load events logs file data to tables and **incorporate dimensional modelling concepts**
2. Create Snapshot table as per given business rules

## Tools/Languages Used

Below are the tools/services used in solution design. AWS equivalents are mentioned wherever available

Google Cloud Service Used	Purpose	Possible AWS Equivalent
Google Cloud Storage (Google's Data Lake)	To stage events log file	AWS S3 Bucket
Google Cloud Function written in Python	To process staged files in data lake	AWS Lambda
Google Big Query (Data warehouse)	To store data	AWS Redshift
DBT (Data Build Tool)	To engineer with data	N/A
GitHub	To version control DBT SQL models and automation with DBT Cloud	N/A
DBT Cloud	To automate the solution	N/A

## Dataflow



## Assumptions:

- Some background process (Cron Job/3rd party process) will sync the daily event logs file in data lake (Google Cloud Storage/Redshift)

## Attribute Definitions

Below are the column definitions, their possible values and inferred data types based on sample values

Column	Description	Possible Values	Inferred data type based on values
event_id	A unique event identifier.	Any integer/string value	STRING
event_type	Describes the professional's activity on the platform	Created_account, became_able_to_propose, Became_unable_to_propose, proposed	STRING
professional_id	A unique professional identifier.	Any integer/string value	STRING
created_at	Timestamp recorded for the event.	2021-03-10 23:59:59	DATETIME

meta_data	Additional information associated with the <b>proposed</b> event type{service_id}_{service_name_nl}_{service_name_en}_{lead_fee}	127_binnenhuis-ontwerp_interior-design_3.4	STRING
-----------	---	--	--------

## Business Rules Applied for Availability Snapshot Table

Following business rules are applied to create availability snapshot table,

- Date range of data in snapshot table is between minimum event time and 2021-03-10
- Professionals are considered as Active if they perform '**became\_able\_to\_propose**' event and will be considered as In Active if they perform '**became\_not\_able\_to\_propose**' event
- If professionals perform both '**became\_able\_to\_propose**' and '**became\_not\_able\_to\_propose**' in a single day at different time intervals then their active/in active status would be decided based on the very first event they performed on that specific day.

## Solution Design Overview:

### 1. Cloud Function:

- Cloud function is written in python and following libraries are used to read logs from data lake and load it into Big Query.
  - Storage API - to make cloud function interact with data lake and read the file
  - Pandas - to hold the logs data temporarily
  - Big Query API - to make cloud function interact with Big Query and load data into its table
- Pandas data frame is used to read the data from file and header row is skipped
- With Pandas data frame, an Audit column is added which is not part of the events file. Its is added for Audit purposes
- Lastly, Big Query API is called to load data frame into Big Query table

```

1  from google.cloud import bigquery
2  from google.cloud import storage
3  import pandas as pd
4  from datetime import datetime
5  import pytz
6
7  project_id='poetic-genius-315513'
8  dataset_id='events_information'
9  table_id='raw_event_logs'
10
11 def load_events_data_to_bq(event, context):
12     file_name = 'gs://data-files-bucket/' + str(event['name'])
13     print(str(event['name']))
14     df = pd.read_csv(file_name, header=None, skiprows=1, dtype=str, sep=',') # reading all CSV columns into one data frame column
15     df.columns = ['EventLogEntry'] # assigning a unique label to one column name
16     df['AuditCreatedDatetime'] = datetime.now(pytz.timezone('America/Chicago')).strftime('%Y-%m-%d %H:%M:%S') # audit column (not part of data file)
17     print(df.tail(5))
18
19     client_bq = bigquery.Client(project=project_id)
20     dataset_ref = client_bq.dataset(dataset_id)
21     table_ref = dataset_ref.table(table_id)
22     job_config = bigquery.LoadJobConfig()
23     job_config.autodetect = False
24
25     # stage table will be truncated before loading new data
26     job_config.write_disposition = bigquery.WriteDisposition.WRITE_APPEND
27     load_job = client_bq.load_table_from_dataframe(df, table_ref, job_config=job_config) # calling BQ API to load data frame data into BQ
28     load_job.result()
29
30     # retrieving number of rows loaded into staging table
31     table = client_bq.get_table(project_id + '.' + dataset_id + '.' + table_id)
32     print("Loaded {} rows and {} columns to {}".format(table.num_rows, len(table.schema), table_id))
33

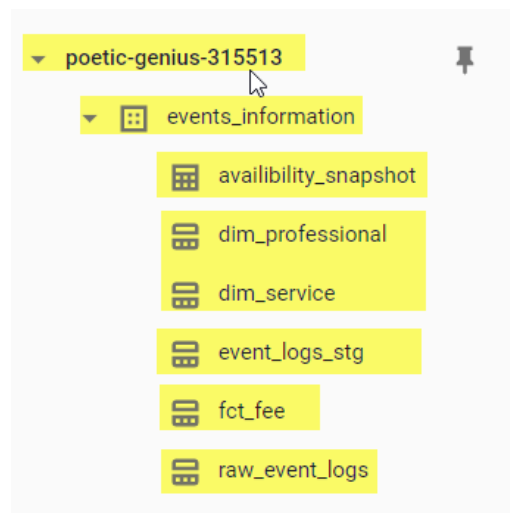
```

## 2. Big Query:

### a. Identified Dimensions & Facts

Following dimensions and facts are identified,

- **dim\_professional** - Dimension
- **dim\_service** - Dimension
- **fct\_fee** - Fact
- **availibilty\_snapshot** - snapshot table as per business rules
- **raw\_event\_logs** - Raw table in which cloud function loads the data)
- **event\_logs\_stg** - Serves as base table for DBT models



## event\_logs\_stg

**i** This is a partitioned table. [Learn more](#)

Field name	Type	Mode
EventID	STRING	NULLABLE
EventType	STRING	NULLABLE
ProfessionalID	STRING	NULLABLE
CreatedAt	STRING	NULLABLE
Metadata	STRING	NULLABLE
AuditCreatedDatetime	DATETIME	NULLABLE

## dim\_professional

**i** This is a partitioned table. [Learn more](#)


Field name	Type
ProfessionalID	STRING
AuditCreatedDateTime	DATETIME

## dim\_service

**i** This is a partitioned table. [Learn more](#)

Field name	Type
ServiceID	STRING
ServiceNameDutch	STRING
ServiceNameEnglish	STRING
AuditCreatedDateTime	DATETIME

fct\_fee

 This is a partitioned table. [Learn more](#)

[Schema](#) [Details](#) [Preview](#)

Field name	Type
ProfessionalID	STRING
ServiceID	STRING
EventID	STRING
EventType	STRING
CreatedAt	STRING
LeadFee	FLOAT
AuditCreatedDatetime	DATETIME

availability\_snapshot

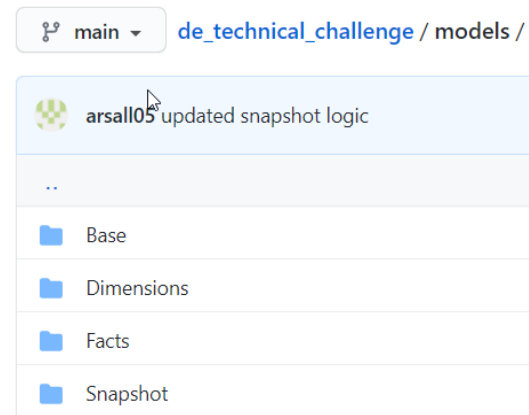
[Schema](#) [Details](#) [Preview](#)

Field name	Type
CreatedAt	DATE
ActiveProfessionals	INTEGER

### 3. Data Build Tool:

- a. Data build tool is used to incorporate following analytics engineering practices in solution design,
  - i. Code Reusability & easy to debug code
  - ii. Version Controlling
  - iii. Data Quality Tests
  - iv. Have rich data docs generated for each run
- b. DBT Project is divided into 4 sub directories with in models directory,
  - i. **Dimensions Folder** - It will contain all of the dimension models
  - ii. **Facts Folder** - It will contain all of the fact models

- iii. **Snapshot Folder** - It will contain models related to the snapshot. Separate folder is created for this because business need can be changed to take some other snapshot based on certain business rules
- iv. **Base Folder** - It will contain all of the base models which would serve as basis of all of the dimensions/facts/snapshots



c. DBT Materialization Used

- i. Incremental materialization is used for Dimensions and Fact.
  - 1. DBT **config()** function is used to define all the table level details at one place in the SQL model
  - 2. **unique\_key** parameter is defined as grain column
  - 3. **sort** parameter is used to define the key on which data would be sorted & stored in table
  - 4. **partition\_by** parameter is defined to reduce the amount of data processing and ultimately save on cost. This would be super helpful for product analysts because they often want to analyze the data for specific dates

34 lines (24 sloc)   793 Bytes		
<pre> 1  {{ 2    config( 3      materialized='incremental', 4      unique_key = 'ProfessionalID', 5      sort='ProfessionalID', 6      partition_by={ 7        "field": "AuditCreatedDateTime", 8        "data_type": "datetime", 9        "granularity": "day" 10     } 11   ) 12 }}</pre>	<pre> {{   config(     materialized='incremental',     unique_key = 'ServiceID',     sort='ServiceID',     partition_by={       "field": "AuditCreatedDateTime",       "data_type": "datetime",       "granularity": "day"     }   ) }}</pre>	<pre> {{   config(     materialized='incremental',     sort=['ProfessionalID', 'ServiceID', 'EventID'],     partition_by={       "field": "AuditCreatedDateTime",       "data_type": "datetime",       "granularity": "day"     }   ) }}</pre>

d. Dbt\_project.yaml and profiles.yaml file

```

name: 'werkspot_technical_challenge'
version: '1.0.0'
config-version: 2

profile: 'werkspot_technical_challenge'

source-paths: ["models"]
analysis-paths: ["analysis"]
test-paths: ["tests"]
data-paths: ["data"]
macro-paths: ["macros"]
snapshot-paths: ["snapshots"]

target-path: "target" # directory which will store compiled SQL files
clean-targets: # directories to be removed by `dbt clean`
  - "target"
  - "dbt_modules"
```



```
workspot_technical_challenge:  
  target: dev  
  outputs:  
    dev:  
      type: bigquery  
      method: oauth  
      project: poetic-genius-315513  
      dataset: events_information  
      threads: 1  
      timeout_seconds: 300  
      location: US  
      priority: interactive
```

e. DBT Lineage Graph Generated After DBT Cloud Run



f. DBT Cloud Example Run

<b>Success</b> RUN RESULT	Kicked off from UI by arsalan.mehmood05@gmail.com TRIGGER	#5790f3 COMMIT SHA	<a href="#">View Documentation &gt;</a> ARTIFACTS	<a href="#">DE Technical Challenge Daily Job &gt;</a> JOB	<a href="#">Development Environment &gt;</a> ENVIRONMENT
------------------------------	--	-----------------------	--	--	---

## Details

Timing

Artifacts

1 minute, 7 seconds ago  
RUN TRIGGERED

27 seconds  
TIME IN QUEUE

5 seconds ago  
COMPLETED

1 minute, 2 seconds  
COMPLETED AFTER

## Run Steps



**Clone Git Repository**

SUCCESS - 00:00:00



**Create Profile from Connection Bigquery**

SUCCESS - 00:00:00



**Invoke dbt with `dbt deps`**

SUCCESS - 00:00:00



**Invoke dbt with `dbt run --models event\_logs\_stg`**

SUCCESS - 00:00:03



**Invoke dbt with `dbt run --models dim\_professional dim\_service fct\_fee`**

SUCCESS - 00:00:08



**Invoke dbt with `dbt run --models availability\_snapshot`**

SUCCESS - 00:00:05



**Invoke dbt with `dbt docs generate`**

SUCCESS - 00:00:05

## Potential Improvement Points:

- Data Quality Tests can be applied (Schema tests/customized tests)
- Cloud Function can be made more dynamic by passing parameter values as environment variables
- DBT profiles.yaml file can be enhanced to have separate configurations for dev and Prod environments

## Other Details:

### 1. Attached Zip File Folder Details:

- a. De\_technical\_challenge - DBT Project
- b. Cloud Function - load\_events\_data\_to\_bq - Cloud Function developed in Python
- c.

### 2. Trial DBT Cloud Account Credentials:

Email: [arsalan.mehmood05@gmail.com](mailto:arsalan.mehmood05@gmail.com)

Password: tLXM53n#\_Uys8.P

### 3. Loom Screen Recording

to give quick walk through of solution: <https://www.loom.com/share/152e1b570b3a485ab31f933d34d60eac>