Data Engineering Technical Challenge Solution

Scenarios:

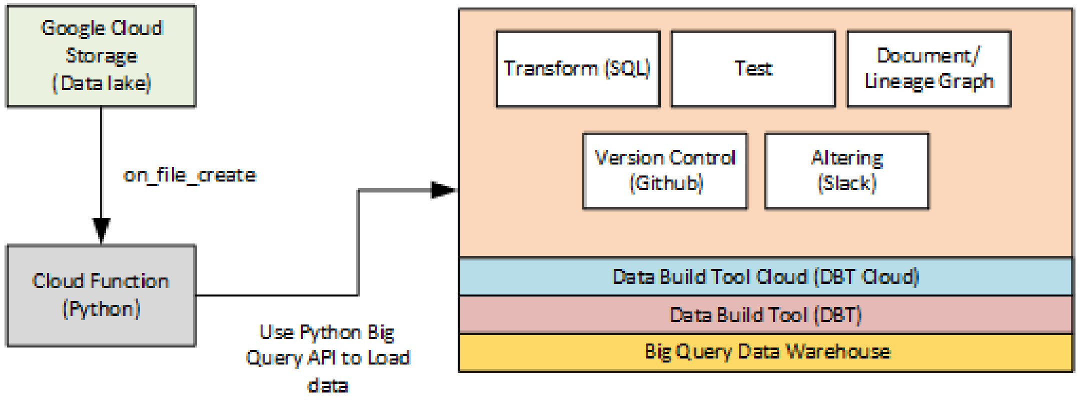
1. Load events logs file data to tables and **incorporate dimensional modelling concepts**
2. Create Snapshot table as per given business rules

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 **proposed** event type{service_id}_{service_name_nl]_{service_name_en}_{lead_fee}	**127_binnenhuis-ontwerp_interior-design_3.4**	STRING

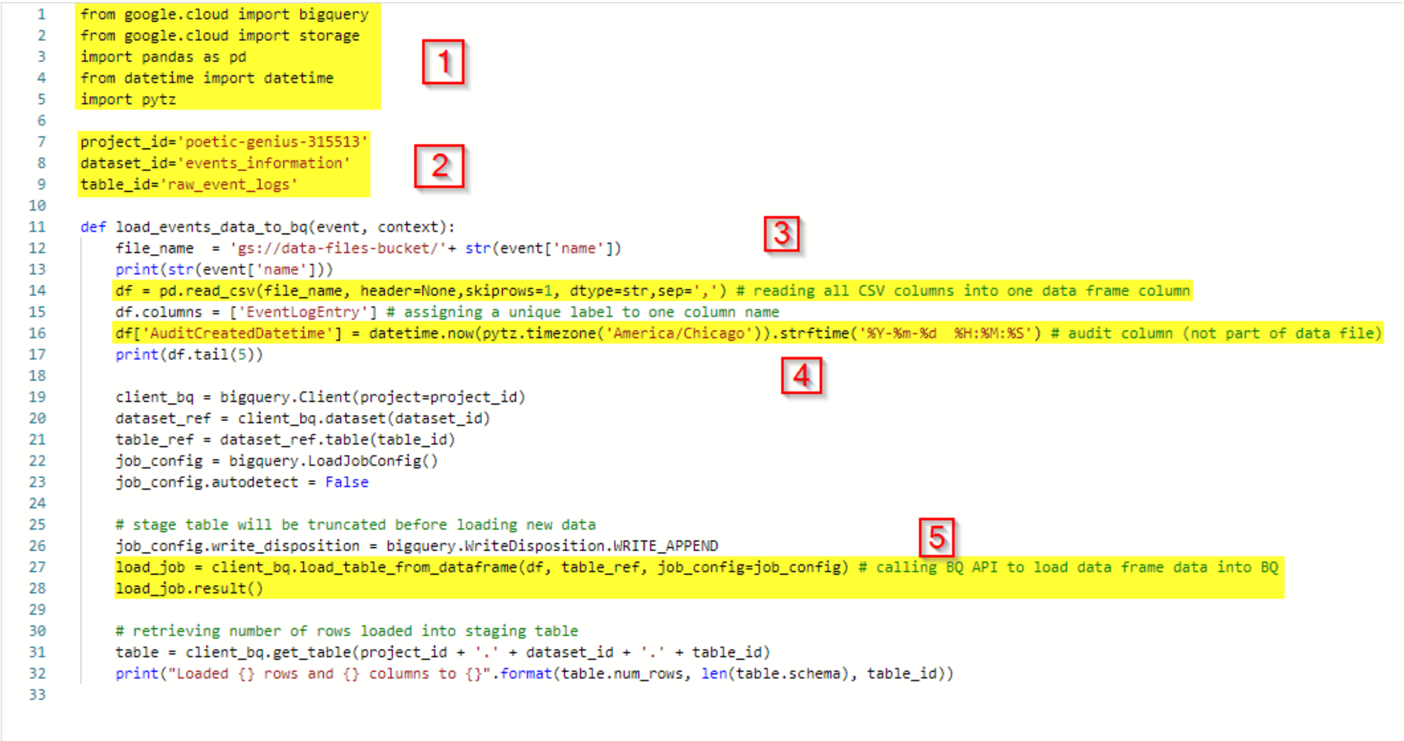
Business Rules Applied for Availability Snapshot Table

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



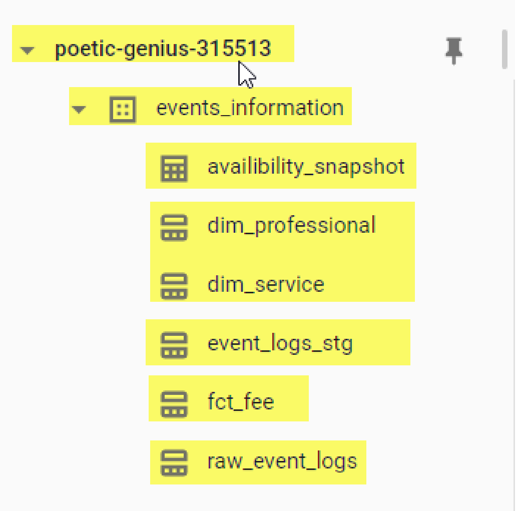
```python
from google.cloud import bigquery
from google.cloud import storage
import pandas as pd
from datetime import datetime
import pytz

project_id='poetic-genius-315513'
dataset_id='events_information'
table_id='raw_event_logs'

def load_events_data_to_bq(event, context):
    file_name  = 'gs://data-files-bucket/'+ str(event['name'])
    print(str(event['name']))
    df = pd.read_csv(file_name, header=None,skiprows=1, dtype=str,sep=',') # reading all CSV columns into one data frame column
    df.columns = ['EventLogEntry'] # assigning a unique label to one column name
    df['AuditCreatedDatetime'] = datetime.now(pytz.timezone('America/Chicago')).strftime('%Y-%m-%d  %H:%M:%S') # audit column (not part of data file)
    print(df.tail(5))

    client_bq = bigquery.Client(project=project_id)
    dataset_ref = client_bq.dataset(dataset_id)
    table_ref = dataset_ref.table(table_id)
    job_config = bigquery.LoadJobConfig()
    job_config.autodetect = False

    # stage table will be truncated before loading new data
    job_config.write_disposition = bigquery.WriteDisposition.WRITE_APPEND
    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
    load_job.result()

    # retrieving number of rows loaded into staging table
    table = client_bq.get_table(project_id + '.' + dataset_id + '.' + table_id)
    print("Loaded {} rows and {} columns to {}".format(table.num_rows, len(table.schema), table_id))
```

2. Big Query:

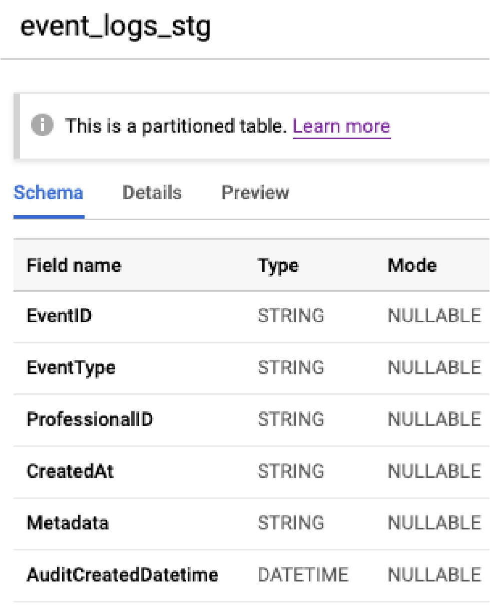
a. Identified Dimensions & Facts

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



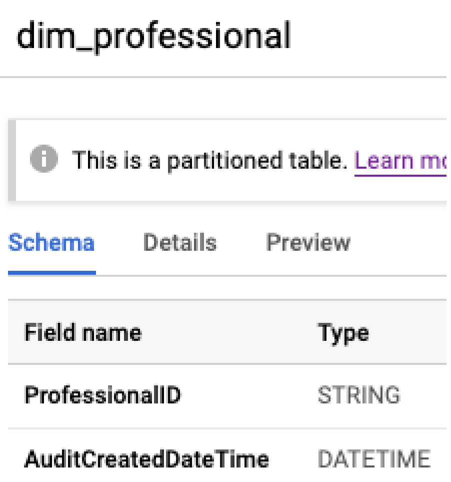
- poetic-genius-315513
  - events_information
    - availibility_snapshot
    - dim_professional
    - dim_service
    - event_logs_stg
    - fct_fee
    - raw_event_logs



event_logs_stg

This is a partitioned table. Learn more

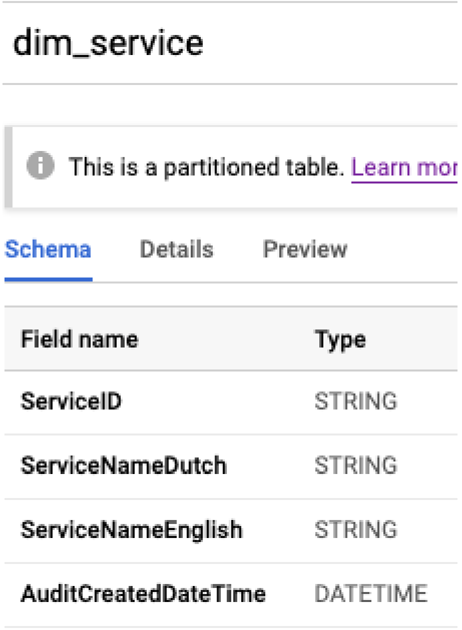
Field name	Type	Mode
EventID	STRING	NULLABLE
EventType	STRING	NULLABLE
ProfessionalID	STRING	NULLABLE
CreatedAt	STRING	NULLABLE
Metadata	STRING	NULLABLE
AuditCreatedDatetime	DATETIME	NULLABLE



dim_professional

This is a partitioned table. Learn mo

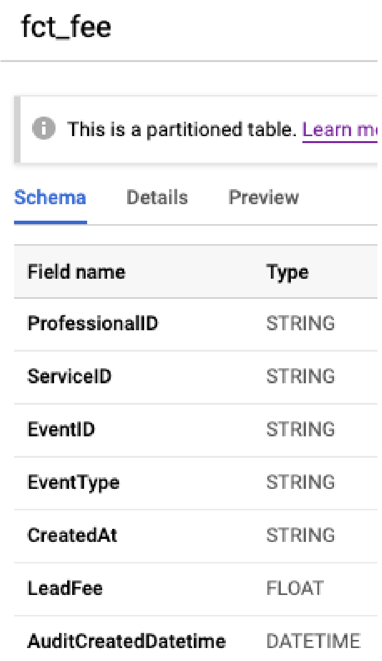
Field name	Type
ProfessionalID	STRING
AuditCreatedDateTime	DATETIME



dim_service

This is a partitioned table. Learn mor

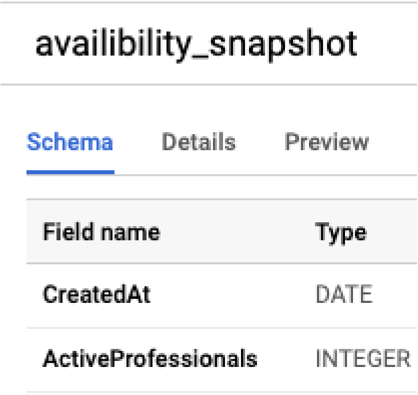
Field name	Type
ServiceID	STRING
ServiceNameDutch	STRING
ServiceNameEnglish	STRING
AuditCreatedDateTime	DATETIME



fct_fee

This is a partitioned table. Learn m

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



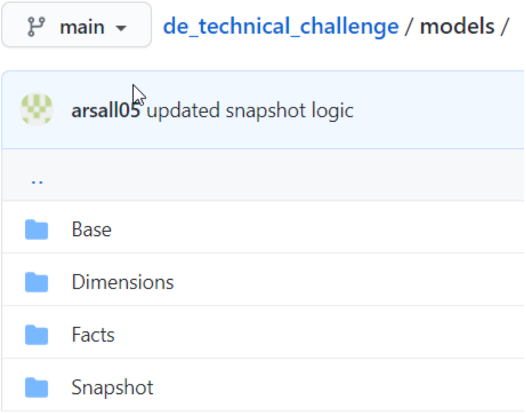
availibility_snapshot

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

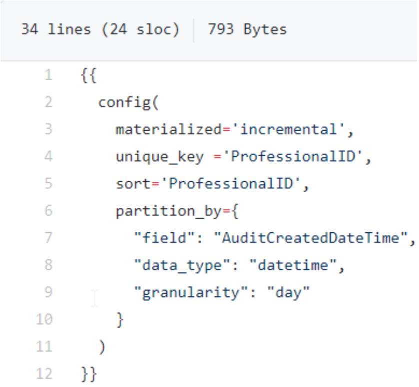

main
de_technical_challenge / models /
arsall05 updated snapshot logic
..
Base
Dimensions
Facts
Snapshot

c. DBT Materialization Used

i. Incremental materialization is used for Dimensions and Fact.

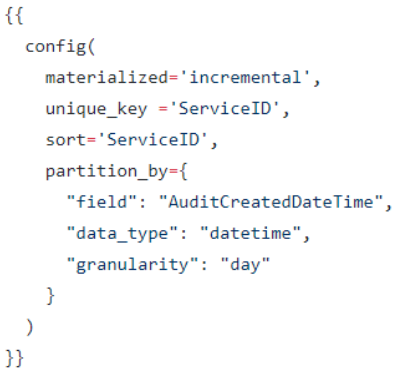
1. DBT **config()** function is used to define all the table level details at one place in the SQL model
2. **unique_key** parameter is defined as grain column
3. **sort** parameter is used to define the key on which data would be sorted & stored in table
4. **partition_by** parameter is defined to reduce 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
5. {{this}} and is_incremental() DBT macro is used for picking incremental data

```
-- this will only be applied on an incremental run & will filter data early
-- {{this}} will give last run date which can then be used to pick CDC records daily
{% if is_incremental() %}
  where AuditCreatedDatetime > (select max(AuditCreatedDatetime) from {{ this }})
{% endif %}
```

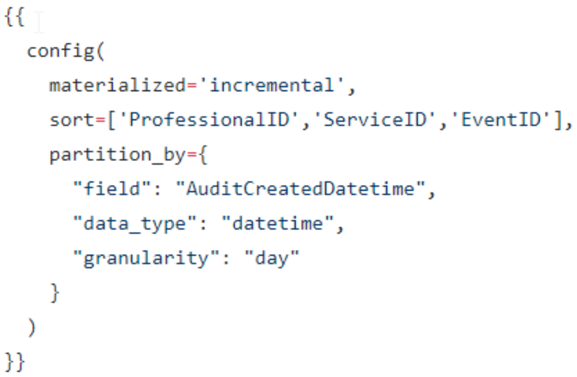


34 lines (24 sloc) | 793 Bytes

```
{{
  config(
    materialized='incremental',
    unique_key ='ProfessionalID',
    sort='ProfessionalID',
    partition_by={
      "field": "AuditCreatedDateTime",
      "data_type": "datetime",
      "granularity": "day"
    }
  )
}}
```

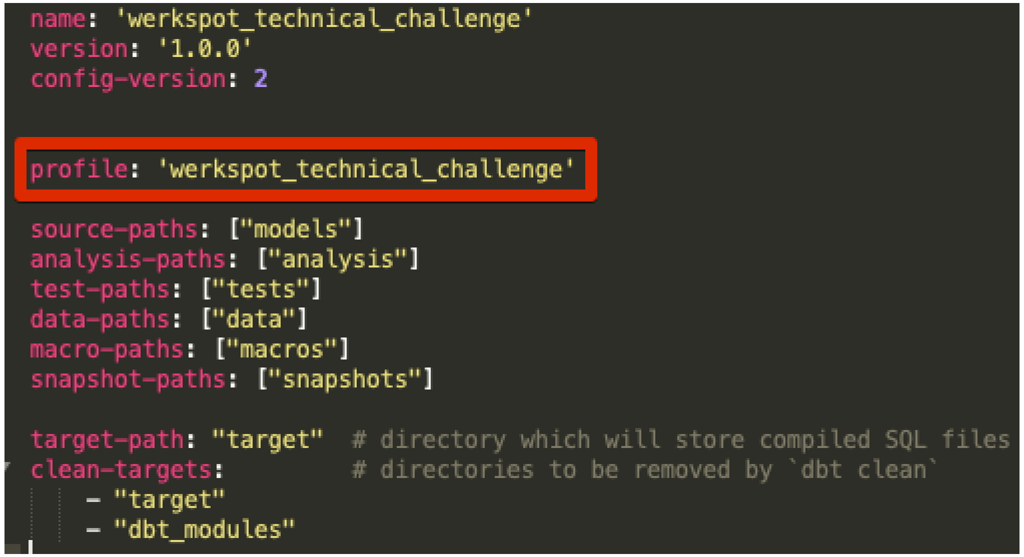


```
{{
  config(
    materialized='incremental',
    unique_key ='ServiceID',
    sort='ServiceID',
    partition_by={
      "field": "AuditCreatedDateTime",
      "data_type": "datetime",
      "granularity": "day"
    }
  )
}}
```

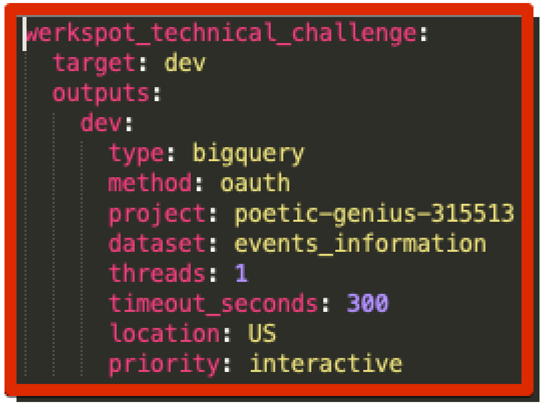


```
{{
  config(
    materialized='incremental',
    sort=['ProfessionalID','ServiceID','EventID'],
    partition_by={
      "field": "AuditCreatedDatetime",
      "data_type": "datetime",
      "granularity": "day"
    }
  )
}}
```

d. Dbt_project.yaml and profiles.yaml file

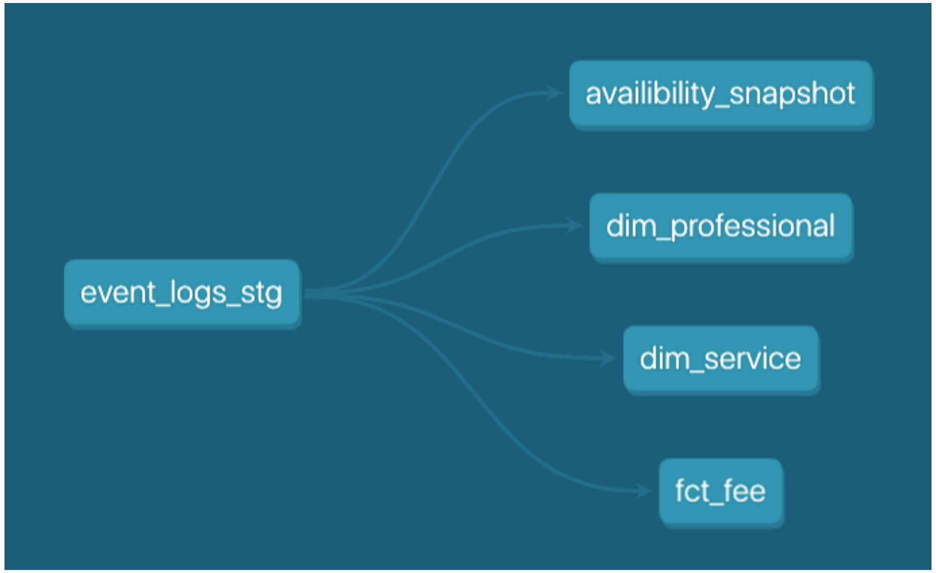


```
name: 'werkspot_technical_challenge'
version: '1.0.0'
config-version: 2


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"
```

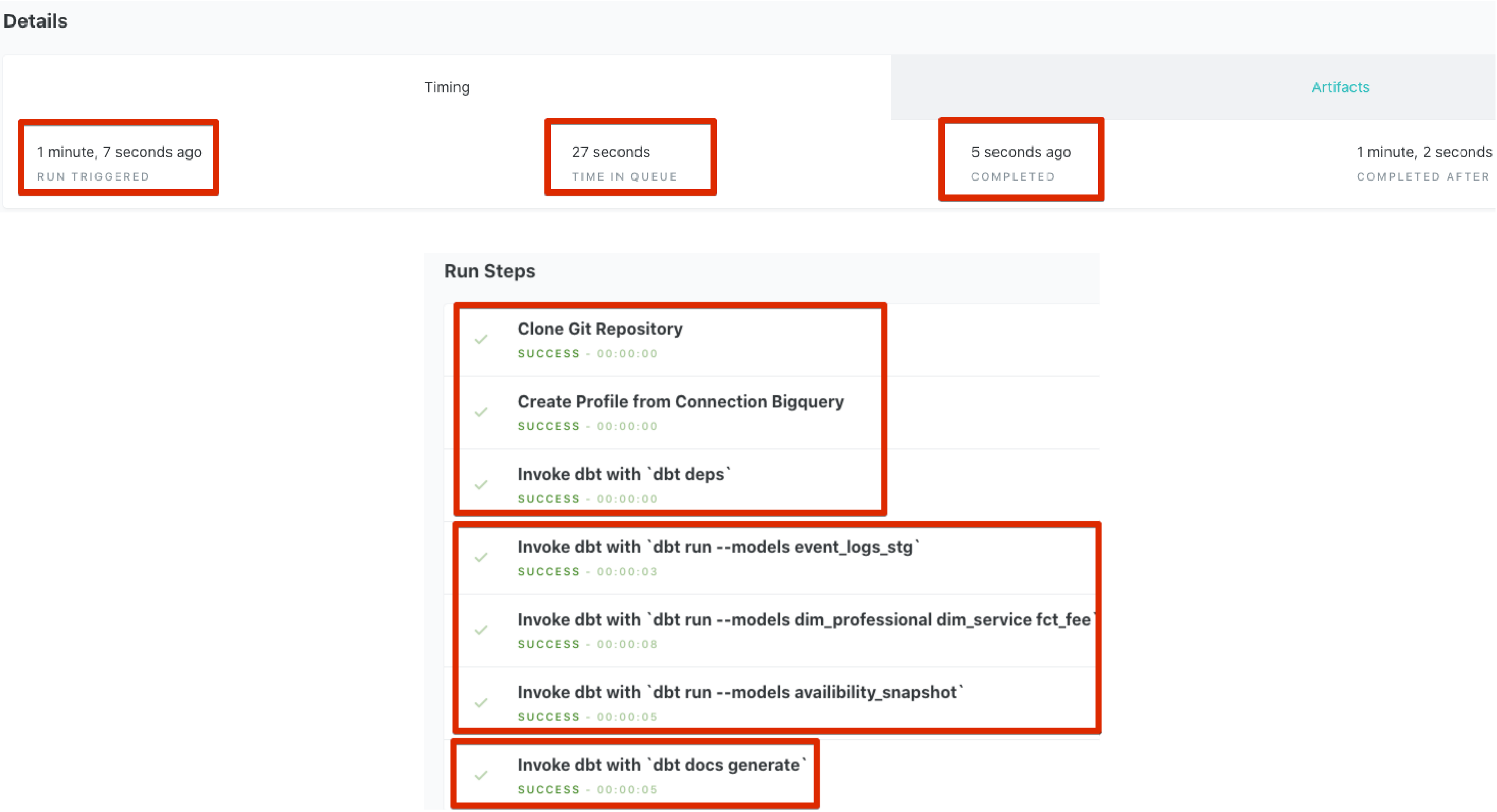


```
werkspot_technical_challenge:
  target: dev
  outputs:
    dev:
      type: bigquery
      method: oauth
      project: poetic-genius-315513
      dataset: events_information
      threads: 1
      timeout_seconds: 300
      location: US
      priority: interactive
```

e. **DBT Lineage Graph Generated After DBT Cloud Run**



f. DBT Cloud Example Run (Steps/Details)



Details

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 availibility_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

    Password: tLXM53n#_Uys8.P

3. **Loom Screen Recording**

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