# Architect Batch ETL Pipeline with BigQuery & Composer

In this document, I will share an overview of:

- a scalable data architecture for structured data using data integration and orchestration services
- detailed solution design for easy to scale ingestion using Cloud Composer

Code Repository: https://github.com/databasedecision/nyctaxi

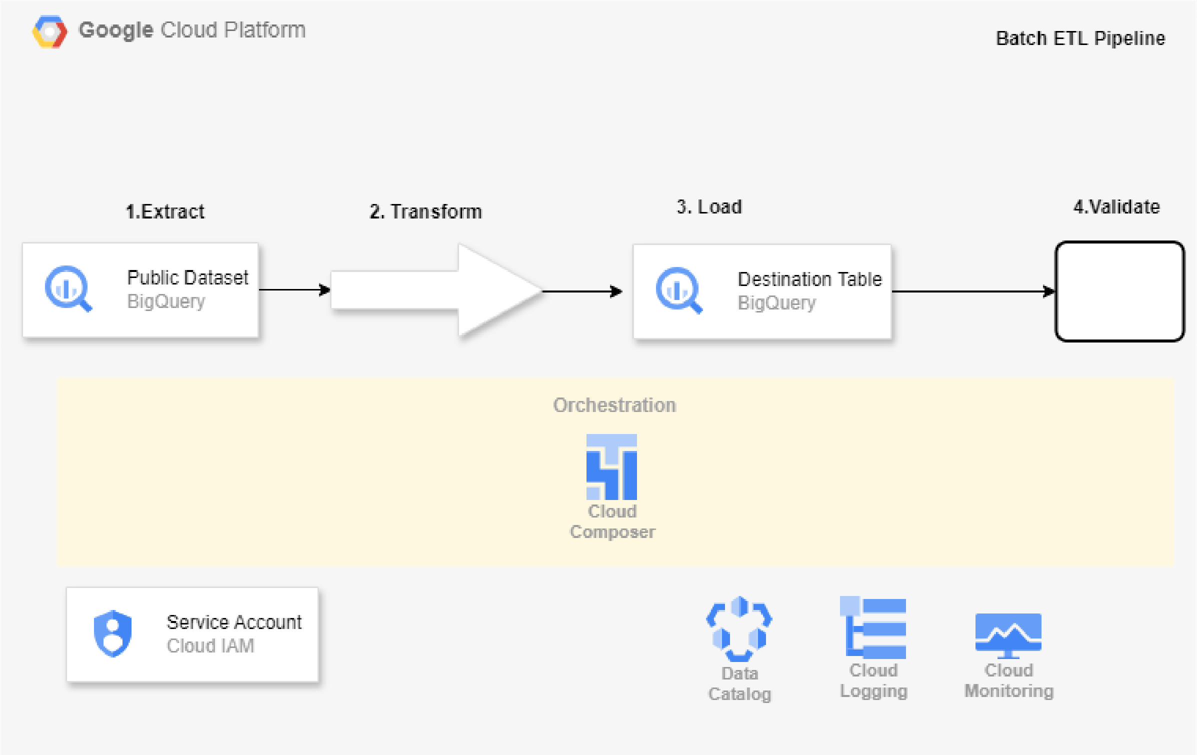
## Key requirements of the use case

There are a few broad requirements that form the premise for this architecture.

1. Leverage mentioned tools(BigQuery & Cloud Composer/Airflow) in task outlined
2. Ingest from public dataset(NYC Taxi)
3. Support complex dependency management in job orchestration, not just for the ingestion jobs, but also custom transformation & validation tasks.
4. Design for a lean code base and configuration driven ingestion pipelines
5. Enable data discoverability while still ensuring appropriate access controls

## Solution Architecture

Architecture designed for the data lake to meet above requirements in shown below. The key GCP services involved in this architecture include services for data integration, storage, orchestration and data discovery.



### Considerations for tool selection

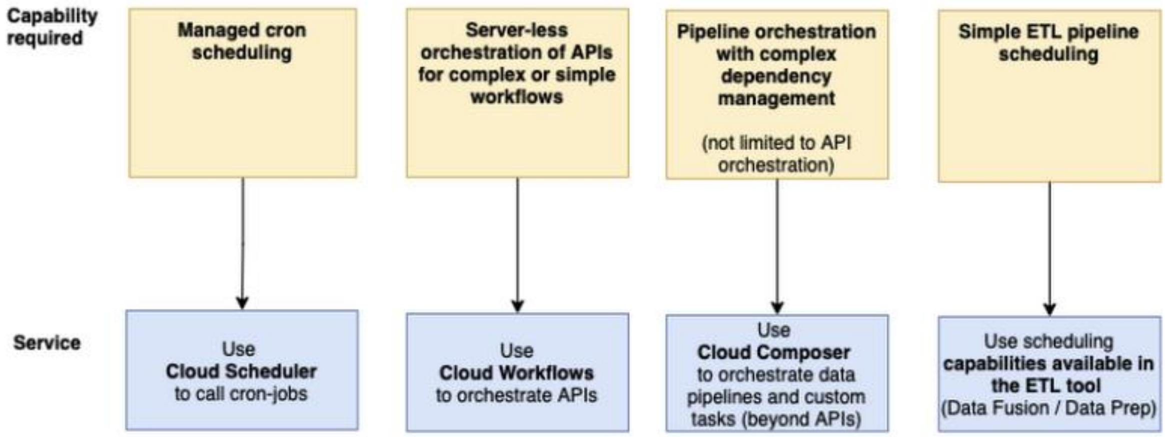
GCP provides a comprehensive set of data and analytics services. There are multiple service options available for each capability and the choice of service requires architects and designers to consider a few aspects that apply to their unique scenarios.

In the following sections, I have described some considerations that architects and designers should make during the selection of different types of services for the architecture, and the rationale behind my final selections for each type of service.

**There are multiple ways to design the architecture with different service combinations and what is described here is just one of the ways. Depending on your unique requirements, priorities and considerations, there are other ways to architect a data lake on GCP.**

### Orchestration

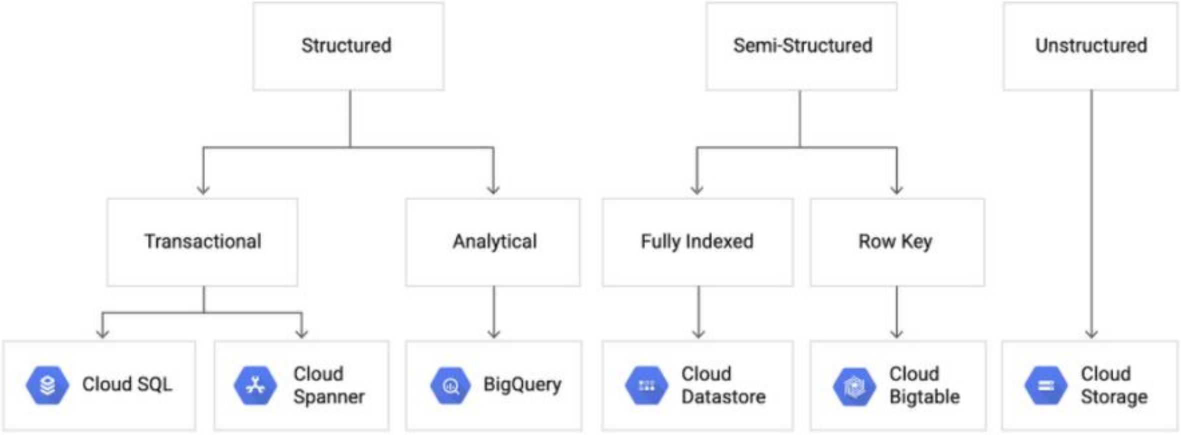
The tree below shows the considerations involved in selecting an orchestration service on GCP.


Capability required
Managed cron scheduling
Server-less orchestration of APIs for complex or simple workflows
Pipeline orchestration with complex dependency management
(not limited to API orchestration)
Simple ETL pipeline scheduling
Service
Use Cloud Scheduler to call cron-jobs
Use Cloud Workflows to orchestrate APIs
Use Cloud Composer to orchestrate data pipelines and custom tasks (beyond APIs)
Use scheduling capabilities available in the ETL tool (Data Fusion / Data Prep)

My use case requires managing complex dependencies such as converging and diverging execution control and data integration. Also, UI capability to access operational information such as historical runs and logs, and the ability to restart workflows from the point of failure was important. Owing to these requirements, Cloud Composeris selected as the orchestration service.

### Data lake storage

Storage layer for the data lake needs to consider the nature of the data being ingested and the purpose it will be used for. The image below provides a decision tree for storage service selection based on these considerations.



Since the source data is a public BQ dataset and we aim to address the solution architecture for structured data which will be used for analytical use cases, GCP BigQuery was selected as the storage service/database for this data lake solution.

### Design approach

The solution design described here provides a framework to ingest a large number of source objects through the use of simple configurations. Once the framework is developed, adding new sources / objects to the data lake ingestion only requires adding new configurations for the new source.

### Design components

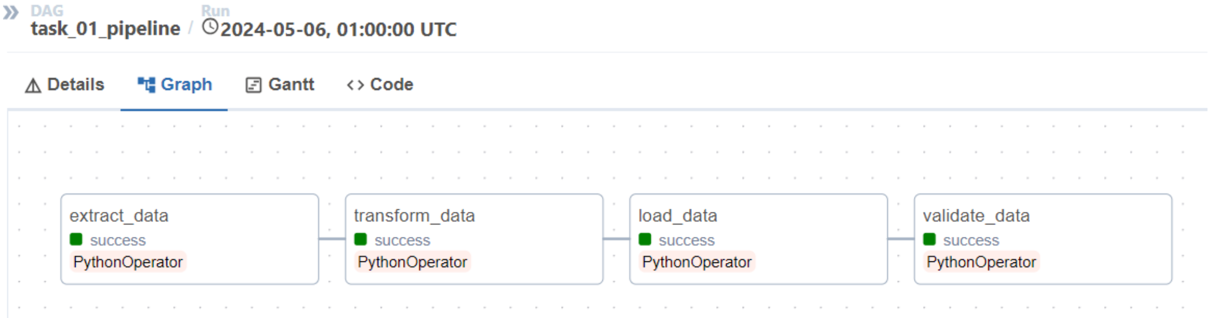
The solution design comprises 4 broad components.

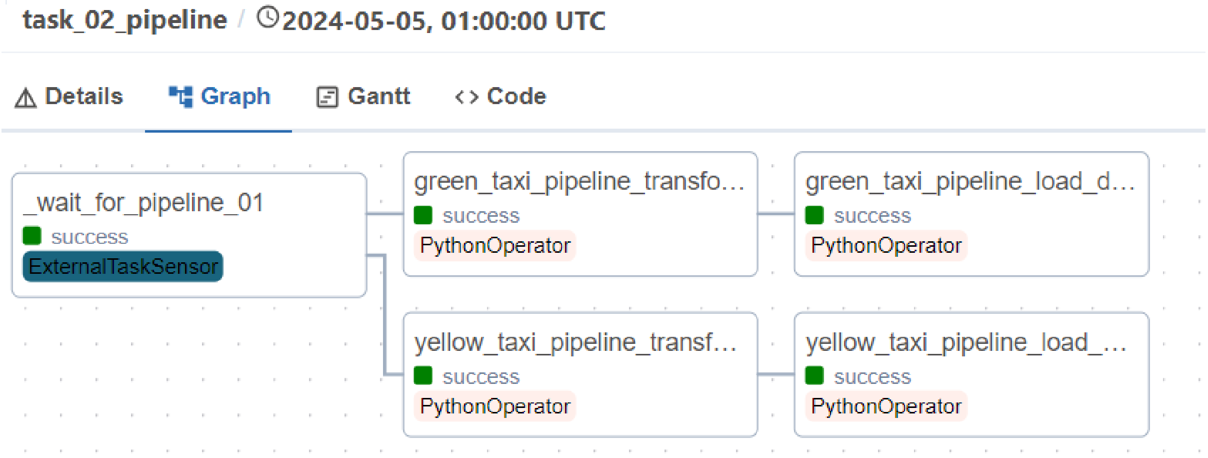
- Custom transformation and validation tasks (Reusable modules)
- Metadata yaml file configuration to keep dynamisms
- Configurations to provide inputs to reusable components and tasks
- Composer DAGs to execute the custom tasks and to call pipelines based on configurations

Let me start with a high level view of the Composer DAG that orchestrates all the parts of the solution, and then provide insight into the different pieces of the solution in the following sections.

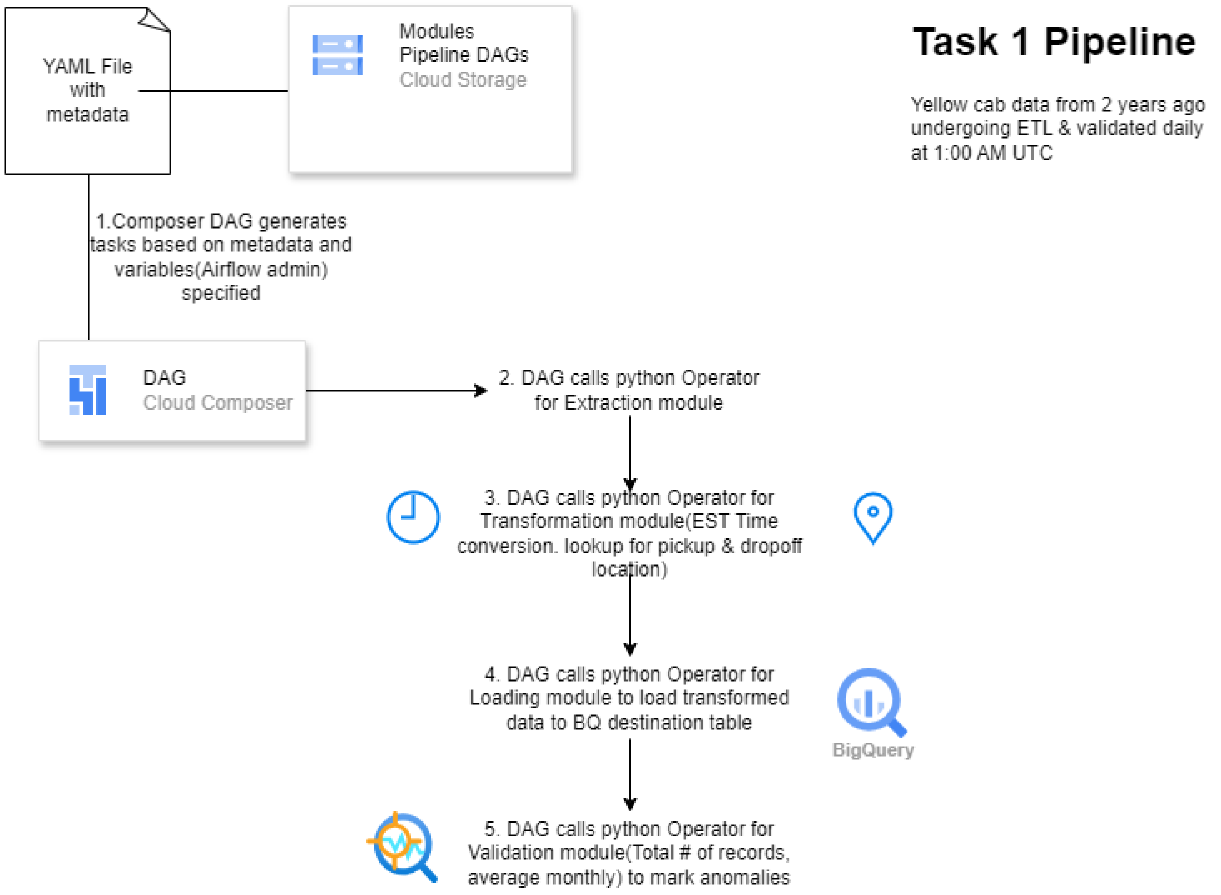
### Composer DAG structure

The Composer DAG is the workflow orchestrator. In this framework, It will follow the flows outlines in the 2 pipelines shown in the images below.

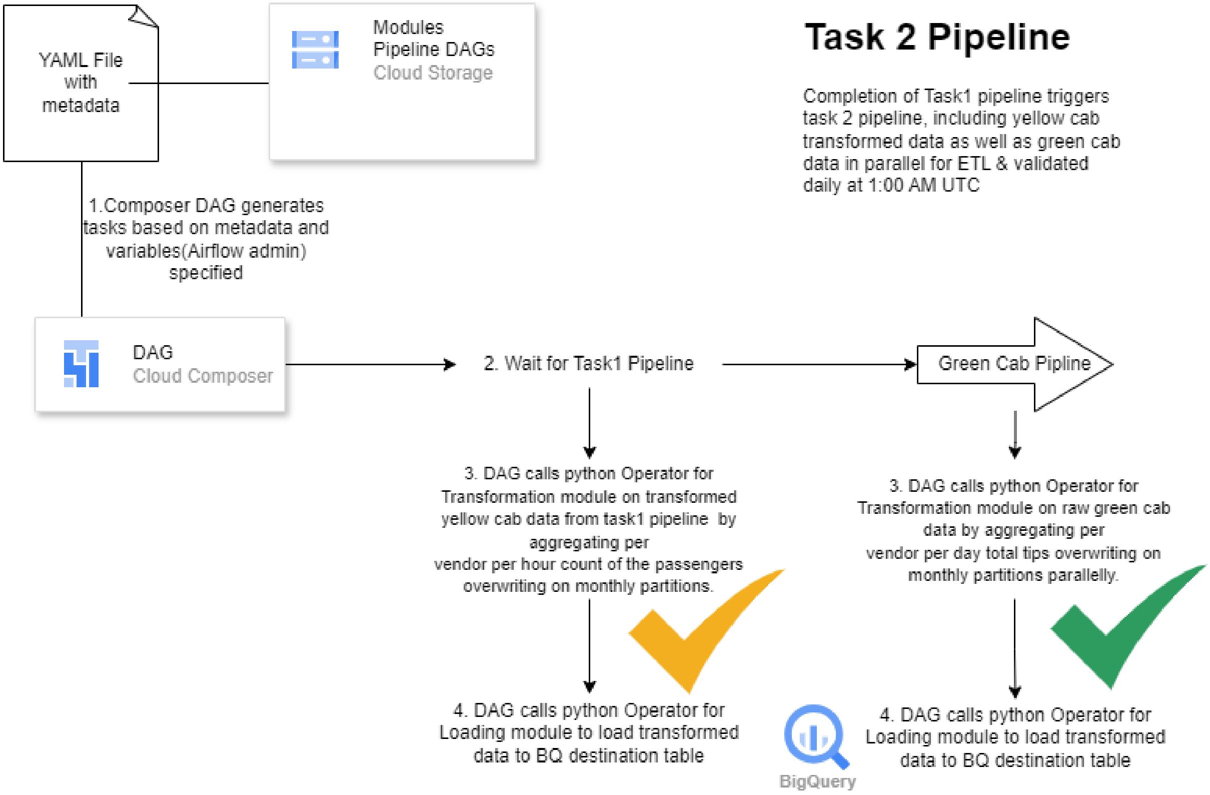




The DAG reads the yaml files (detailed in the next section) for details such as source, target, transformation and validation details, and passes that information to tasks. The image below describes the overall flow.



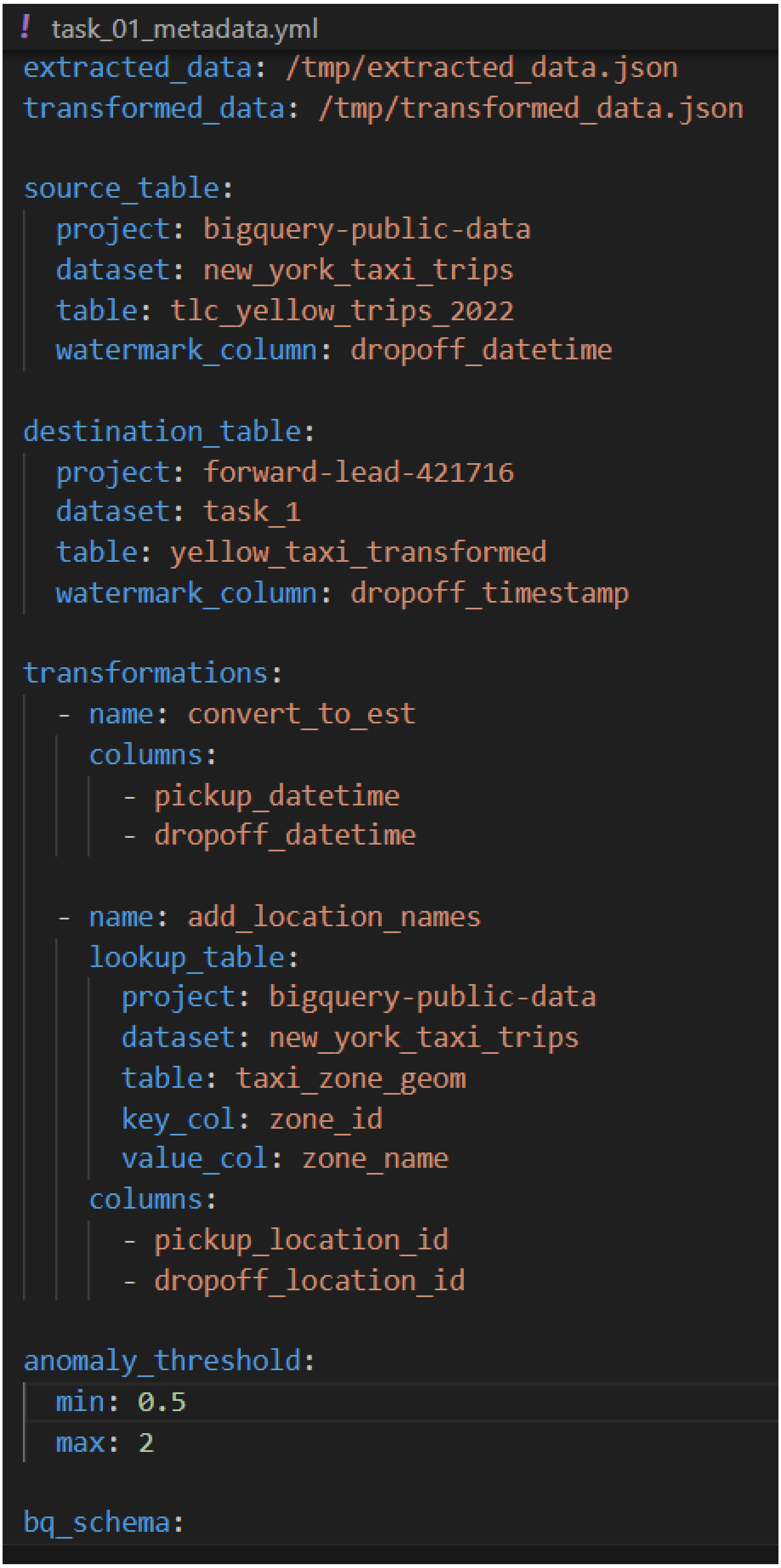
YAML File with metadata
Modules
Pipeline DAGs
Cloud Storage
Task 1 Pipeline
Yellow cab data from 2 years ago undergoing ETL & validated daily at 1:00 AM UTC
1.Composer DAG generates tasks based on metadata and variables(Airflow admin) specified
DAG
Cloud Composer
2. DAG calls python Operator for Extraction module
3. DAG calls python Operator for Transformation module(EST Time conversion. lookup for pickup & dropoff location)
4. DAG calls python Operator for Loading module to load transformed data to BQ destination table
BigQuery
5. DAG calls python Operator for Validation module(Total # of records, average monthly) to mark anomalies


YAML File with metadata
Modules Pipeline DAGs
Cloud Storage
Task 2 Pipeline
Completion of Task1 pipeline triggers task 2 pipeline, including yellow cab transformed data as well as green cab data in parallel for ETL & validated daily at 1:00 AM UTC
1.Composer DAG generates tasks based on metadata and variables(Airflow admin) specified
DAG
Cloud Composer
2. Wait for Task1 Pipeline
Green Cab Pipline
3. DAG calls python Operator for Transformation module on transformed yellow cab data from task1 pipeline by aggregating per vendor per hour count of the passengers overwriting on monthly partitions.
3. DAG calls python Operator for Transformation module on raw green cab data by aggregating per vendor per day total tips overwriting on monthly partitions parallelly.
4. DAG calls python Operator for Loading module to load transformed data to BQ destination table
BigQuery
4. DAG calls python Operator for Loading module to load transformed data to BQ destination table

**Ingestion configuration**

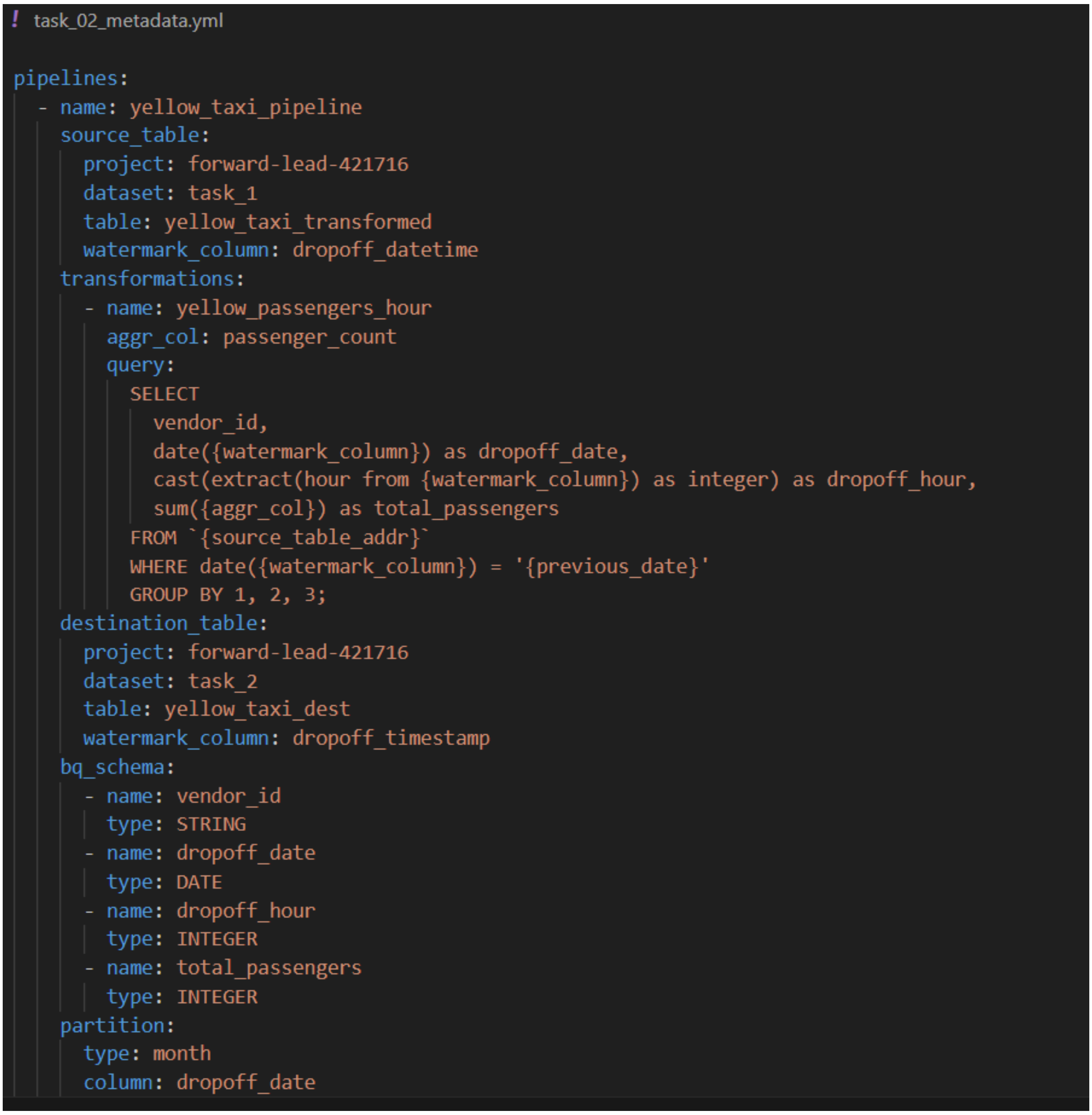
Configuration in this solution is maintained for 3 different levels of information – admin variable, DAG and modules within the DAG. These configurations will live in the Composer environment along with your DAG code .

- **Metadata Configuration from admin variable** for information such as GCP project ID, GCS bucket information, source, destination

- 

task_01_metadata.yml

```yaml
extracted_data: /tmp/extracted_data.json
transformed_data: /tmp/transformed_data.json

source_table:
  project: bigquery-public-data
  dataset: new_york_taxi_trips
  table: tlc_yellow_trips_2022
  watermark_column: dropoff_datetime

destination_table:
  project: forward-lead-421716
  dataset: task_1
  table: yellow_taxi_transformed
  watermark_column: dropoff_timestamp

transformations:
  - name: convert_to_est
    columns:
      - pickup_datetime
      - dropoff_datetime

  - name: add_location_names
    lookup_table:
      project: bigquery-public-data
      dataset: new_york_taxi_trips
      table: taxi_zone_geom
      key_col: zone_id
      value_col: zone_name
    columns:
      - pickup_location_id
      - dropoff_location_id

anomaly_threshold:
  min: 0.5
  max: 2

bq_schema:
```



! task_02_metadata.yml

```yaml
pipelines:
  - name: yellow_taxi_pipeline
    source_table:
      project: forward-lead-421716
      dataset: task_1
      table: yellow_taxi_transformed
      watermark_column: dropoff_datetime
    transformations:
      - name: yellow_passengers_hour
        aggr_col: passenger_count
        query:
          SELECT
            vendor_id,
            date({watermark_column}) as dropoff_date,
            cast(extract(hour from {watermark_column}) as integer) as dropoff_hour,
            sum({aggr_col}) as total_passengers
          FROM `{source_table_addr}`
          WHERE date({watermark_column}) = '{previous_date}'
          GROUP BY 1, 2, 3;
    destination_table:
      project: forward-lead-421716
      dataset: task_2
      table: yellow_taxi_dest
      watermark_column: dropoff_timestamp
    bq_schema:
      - name: vendor_id
        type: STRING
      - name: dropoff_date
        type: DATE
      - name: dropoff_hour
        type: INTEGER
      - name: total_passengers
        type: INTEGER
    partition:
      type: month
      column: dropoff_date
```

- DAG

- **Task Configuration** to specify inputs for the pipeline, for instance the source, the delimiter and pipeline to be triggered.

**Dynamic DAG generation based on configuration**

The solution would comprise two DAGs.

- The main orchestrator DAG that will read the configuration and perform some initial tasks, and trigger child DAGs (henceforth referred to as the worker DAG) based on Task Configuration.
- The worker DAG that actually performs the tasks in the integration process flow and loads data for the configuration provided to it by the orchestrator DAG. An instance of this DAG will be automatically triggered by the orchestrator DAG.

**Key takeaways**

The solution design described above provides a framework to ingest data from a hybrid ecosystem into the data lake. It does so by making use of simple configurations to provide details about the environment, sources and targets, to be executed. Extending the data lake to add more sources is easy and only requires configuration to be added for the new source objects.

Set up of project following creation of google cloud account with free credits

1. **Set Up Google Cloud Service Account:**

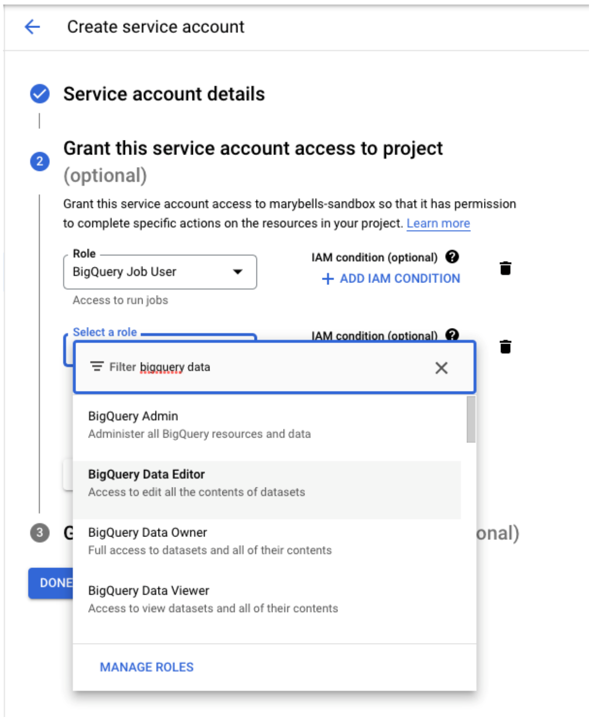
**The Set-up**

1. **Create a separate Google Service Account in each of your** These projects will require the following roles:

    1. **BigQuery JobUser** to issue queries.

    2. **BigQuery Data Editor** for the use of persistent derived tables (PDTs).

    3. *Additional roles required:* Composer Administrator, Composer Worker, Compute Admin, Storage Admin



Service account details

2 Grant this service account access to project (optional)

Grant this service account access to marybells-sandbox so that it has permission to complete specific actions on the resources in your project. Learn more

Role: BigQuery Job User

Access to run jobs

IAM condition (optional)

Select a role

Filter bigquery data

BigQuery Admin
Administer all BigQuery resources and data

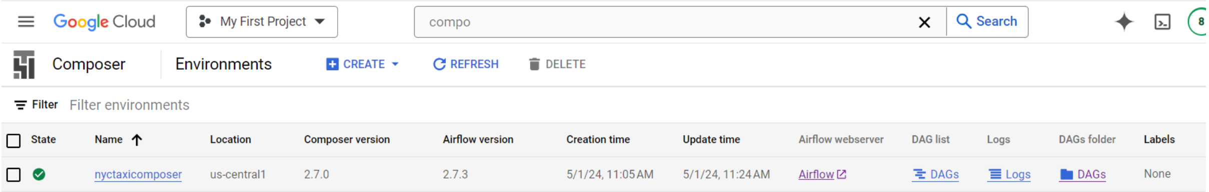
BigQuery Data Editor
Access to edit all the contents of datasets

BigQuery Data Owner
Full access to datasets and all of their contents

BigQuery Data Viewer
Access to view datasets and all of their contents

3 G onal)

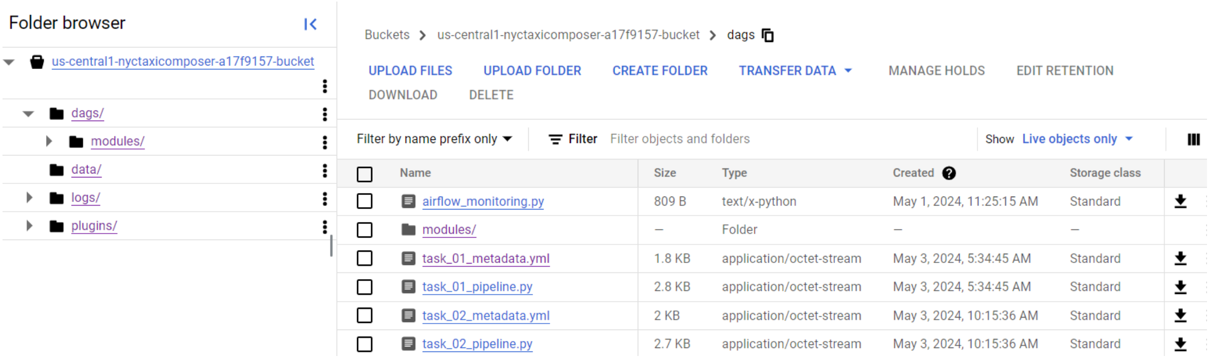
2. Spin up Composer 2 environment with autoscaling



| State | Name ↑ | Location | Composer version | Airflow version | Creation time | Update time | Airflow webserver | DAG list | Logs | DAGs folder | Labels |
|---|---|---|---|---|---|---|---|---|---|---|---|
| ✓ | nyctaxicomposer | us-central1 | 2.7.0 | 2.7.3 | 5/1/24, 11:05 AM | 5/1/24, 11:24 AM | Airflow | DAGs | Logs | DAGs | None |

3. Upload pipeline and metadata yaml files into GCS bucket and modules for ETL & validation in nested within DAG folder

4.



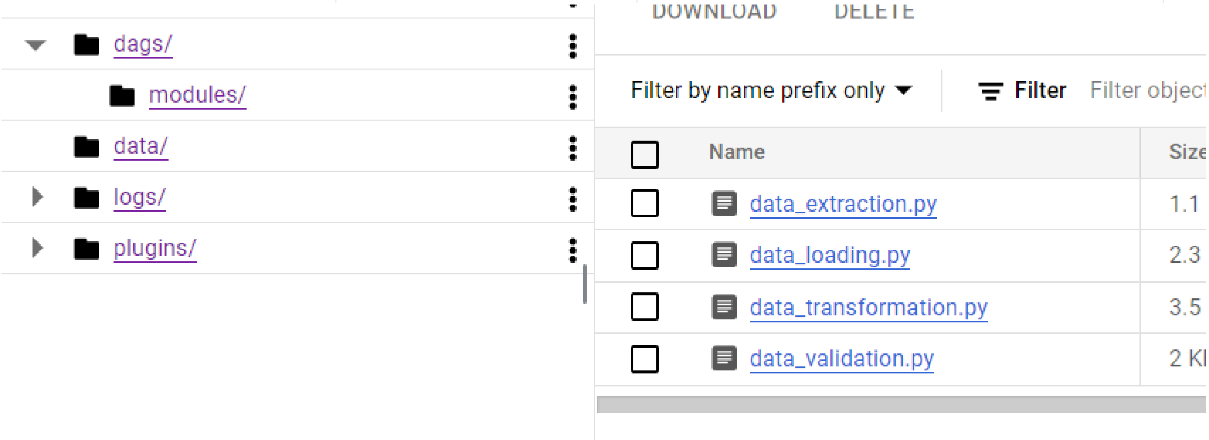
Folder browser

us-central1-nyctaxicomposer-a17f9157-bucket

- dags/
  - modules/
  - data/
  - logs/
  - plugins/

Buckets > us-central1-nyctaxicomposer-a17f9157-bucket > dags

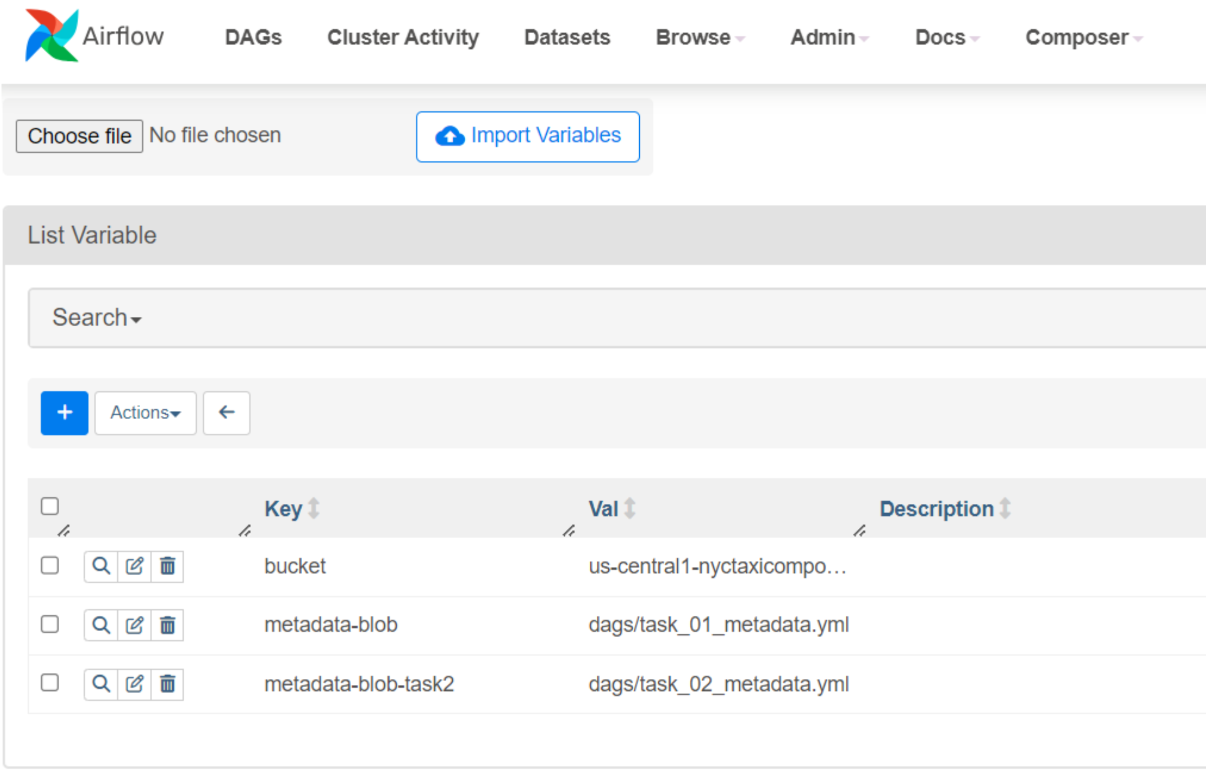
| ☐ | Name | Size | Type | Created | Storage class |
|---|---|---|---|---|---|
| ☐ | airflow_monitoring.py | 809 B | text/x-python | May 1, 2024, 11:25:15 AM | Standard |
| ☐ | modules/ | — | Folder | — | — |
| ☐ | task_01_metadata.yml | 1.8 KB | application/octet-stream | May 3, 2024, 5:34:45 AM | Standard |
| ☐ | task_01_pipeline.py | 2.8 KB | application/octet-stream | May 3, 2024, 5:34:45 AM | Standard |
| ☐ | task_02_metadata.yml | 2 KB | application/octet-stream | May 3, 2024, 10:15:36 AM | Standard |
| ☐ | task_02_pipeline.py | 2.7 KB | application/octet-stream | May 3, 2024, 10:15:36 AM | Standard |



- dags/
  - modules/
  - data/
  - logs/
  - plugins/

| ☐ | Name | Size |
|---|---|---|
| ☐ | data_extraction.py | 1.1 |
| ☐ | data_loading.py | 2.3 |
| ☐ | data_transformation.py | 3.5 |
| ☐ | data_validation.py | 2 KI |

5. Specify as below within admin -> variable



List Variable

| ☐ | | Key | Val | Description |
|---|---|---|---|---|
| ☐ | | bucket | us-central1-nyctaxicompo… | |
| ☐ | | metadata-blob | dags/task_01_metadata.yml | |
| ☐ | | metadata-blob-task2 | dags/task_02_metadata.yml | |

Future Iterations

1. Data Catalog for data discovery
2. Cloud logging and Cloud monitoring observability and health of applications

3. IAM roles at dataset/table level to control access to BQ transformed data

Cost Effective Alternatives:

- Google Workflows are a reasonable low cost alternative to composer for simple pipelines.
- The cloud function can execute automatically when the file arrives (file completion trigger). The function can execute data flow asynchronously and DataFlow will update BigQuery
- Set up an event in Cloud Logging that signals that the load jobs are done. Then that triggers either Pub/Sub or cloud function and then you run the query. One should be able to get this event into Pub/Sub using a log sink and use that as a trigger for your next cloud function.