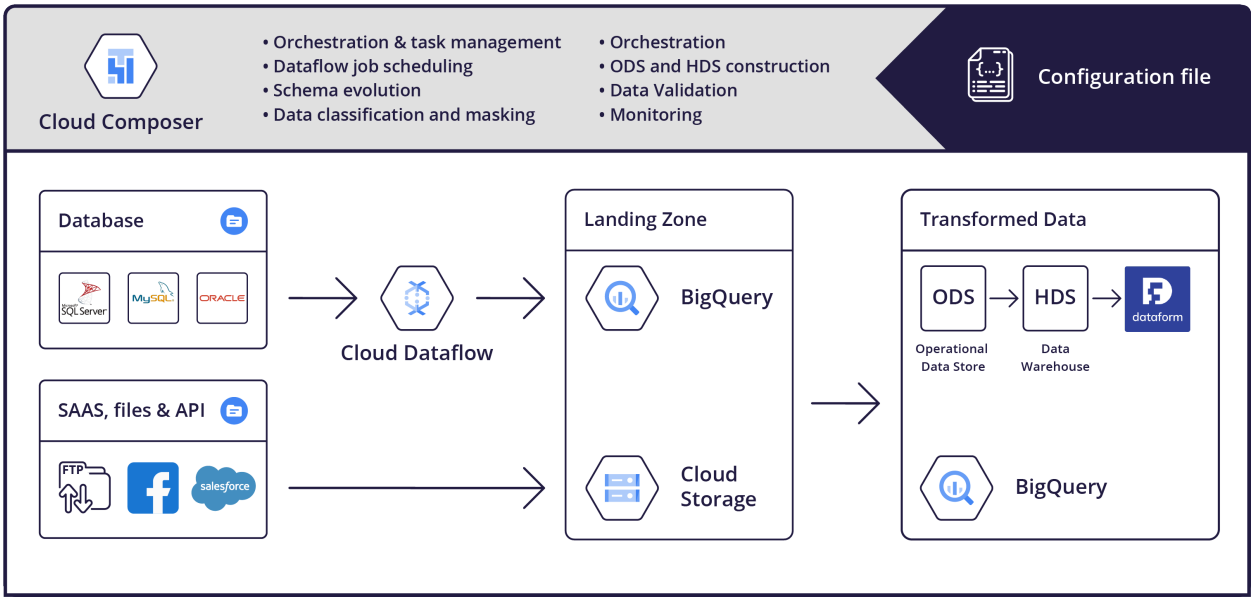
GCP Airflow Foundations

Release 0.2.6

Badal

Oct 26, 2022

CONTENTS:



Airflow is an awesome open source orchestration framework that is the go-to for building data ingestion pipelines on GCP (using Composer - a hosted AIrflow service). However, most companies using it face the same set of problems:

- **Learning curve**: Airflow requires python knowledge and has some gotchas that take time to learn. Further, writing Python DAGs for every single table that needs to get ingested becomes cumbersome. Most companies end up building utilities for creating DAGs out of configuration files to simplify DAG creation and to allow non-developers to configure ingestion
- **Datalake and data pipelines design best practices**: Airflow only provides the building blocks, users are still required to understand and implement the nuances of building a proper ingestion pipelines for the data lake/data warehouse platform they are using
- **Core reusability and best practice enforcement across the enterprise**: Usually each team maintains its own Airflow source code and deployment so sharing and enforcing best practices and tooling is hard

We have written an opinionated yet flexible ingestion framework for building an ingestion pipeline into data warehouse in BigQuery that supports the following features:

- **Zero-cod**e, config file based ingestion - anybody can start ingesting from the growing number of sources by just providing a simple configuration file. Zero python or Airflow knowledge is required.
- **Modular and extendable** - The core of the framework is a lightweight library. Ingestion sources are added as plugins. Adding a new source can be done by extending the provided base classes.
- **Opinionated automatic creation of ODS (Operational Data Store) and HDS (Historical Data Store)** in BigQuery while enforcing best practices such as schema migration, data quality validation, idempotency, partitioning, etc.
- **Dataflow job** support for ingesting large datasets from SQL sources and deploying jobs into a specific network or shared VPC.
- Support of **advanced Airflow features** for job prioritization such as slots and priorities.
- Integration with **GCP data services** such as DLP and Data Catalog [work in progress].
- **Well tested** - We maintain a rich suite of both unit and integration tests.

CHAPTER

ONE

QUICK START

1.1 Installing from PyPI

Install with `pip install 'gcp-airflow-foundations'`

1.2 Generating DAGs

In the Airflow's `dags_folder` create a new Python module (e.g. `parse_dags.py`), which would parse the DAGs from the YAML configuration files:

```
from gcp_airflow_foundations.parse_dags import DagParser

parser = DagParser()

parsed_dags = parser.parse_dags()

if parsed_dags:
    globals().update(parsed_dags)
```

The YAML files are loaded as dictionaries and then converted to data classes using the open-source dacite Python library. Each of the data classes used have their own validators to ensure that the parameters selected by the user are valid. For instance, an error will be raised if the ingestion schedule and the partition time of a snapshot HDS table are not compatible with each other.

1.3 Prerequisites

1.3.1 Running on Google Cloud

- An active Google Cloud with a Cloud Composer environment. The minimal Airflow version required is 2.0.2.
- Enable Cloud Composer, Cloud Storage, and BigQuery APIs
- Optional step: setup a CI/CD pipeline for your Cloud Composer environmental that installs the dependencies from PyPI and syncs your DAGs.

1.3.2 Running with Docker

If you deploy Airflow from a Docker image then you can add GCP Airflow Foundations to the dependencies of your Docker image.

1.4 Airflow Connections

Airflow connections are used to store credentials to communicate with external systems, such as APIs of third-party data sources. Depending on the data sources you are ingesting from you will need to set up the required connections. You can do so either through the Admin menu in the Airflow UI of your Cloud Composer instance, or by using Secret Manager. If you opt for the latter, make sure to follow some additional steps that are required.

CHAPTER

TWO

DATA SOURCES AND SINKS

2.1 Data Sources

2.1.1 Currently Available Sources

gcp-airflow-foundations supports ingesting data from the following sources:

- Google Cloud Storage (including loading from Parquet)
- SFTP
- Oracle (using Dataflow)
- MySQL (using Dataflow)
- Salesforce
- Facebook Ads

2.1.2 Sources in the Making

- Google Ads
- Snapchat
- The Trade Desk
- LinkedIn Marketing
- TikTok
- Twitter
- Amazon DSP
- CM360 & DV360
- Spotify Ads
- Pinterest

2.2 Data Sinks

gcp-airflow-foundations currently supports ingesting data only to BigQuery.

2.3 Data Transformation

gcp-airflow-foundations is an ingestion framework (EL part of ELT), but it supports triggering commoen transformation framweworks post ingestion

- Dataform
- dbt (work in progress)

Transormations can be scheduled to run using post ingestion task dependecies

CHAPTER

THREE

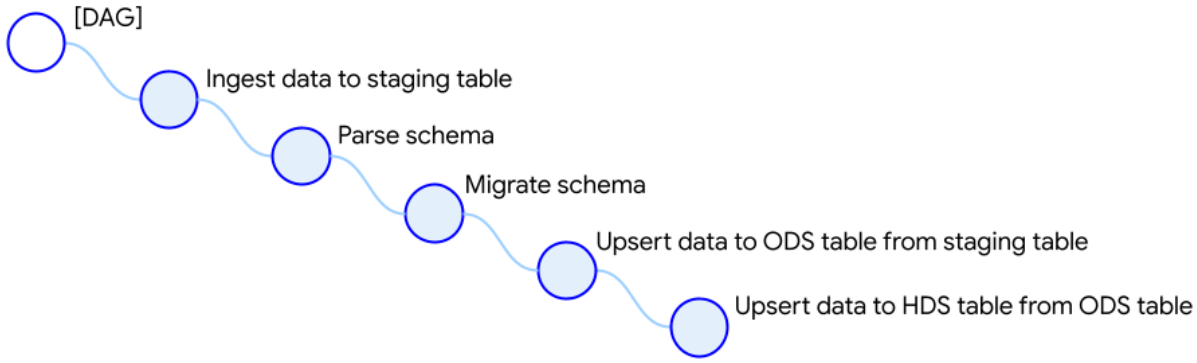
EXTRACTING DATA

3.1 1. Overview

The building blocks of the data ingestion DAG are pre-defined in GCP Airflow Foundations, such that the user only needs to configure the parameters of the data ingestion. The data ingestion follows a set of default steps:

- The first step in the ingestion pipeline is to extract the data from a source and load them to a landing table in BigQuery.
- The schema of the landing table is parsed and compared with the schema of the destination tables.
- If any schema changes are detected, these are migrated to the destination table.
- Finally, the data are upserted to the destination tables.

This can be visualized in the tree diagram bellow:



3.2 2. Config Based DAG Generation

GCP Airflow Foundations support the dynamic generation of ETL/ELT DAGs from simple, user-provided configuation files written in YAML. At minimum, the user declares in the configuration file the derised ingestion mode and the type of the data source, along with the required source tables to be ingested. Optionally, additional parameters can be provided, such us metadata naming and column mapping between the source and destination tables, among others. GCP Airflow Foundations will parse the information declared in the YAML file to generate the building blocks necessary for generating the DAGs for the desired data ingestion.

For a detailed description and data type of each configuration field, please refer to *gcp_airflow_foundations.base_class.source_config.SourceConfig* and *gcp_airflow_foundations.base_class.source_table_config.SourceTableConfig* for the ingestion source and tables respectively.

An example of a simple configuration file to extract marketing data from Facebook:

```yaml
source:
    name: facebook_campaigns_ingestion
    source_type: FACEBOOK
    ingest_schedule: "@daily"
    start_date: "2021-01-01"
    dataset_data_name: facebook
    landing_zone_options:
        landing_zone_dataset: landing_zone
    facebook_options:
        account_lookup_scope: full
        fields: [
            "account_id",
            "campaign_id",
            "impressions",
            "spend",
            "reach",
            "clicks"]
        level: campaign
        time_increment: "1"
tables:
    - table_name: campaign_insights
        surrogate_keys: ["account_id", "campaign_id", "date_start"]
        ingestion_type: INCREMENTAL
        facebook_table_config:
            breakdowns: null
            action_breakdowns: ["action_type"]
    - table_name: campaign_insights_platform_placement
        surrogate_keys: ["account_id", "campaign_id", "date_start", "publisher_platform",
↪ "platform_position"]
        ingestion_type: INCREMENTAL
        facebook_table_config:
            breakdowns:  ["publisher_platform", "platform_position"]
            action_breakdowns: ["action_type"]
```

3.3 3. Replication Scheduling

In the `ingest_schedule` field you can select the ingestion schedule for Airflow. GCP Airflow Foundations currently support hourly, daily, weekly, and monthly intervals.

3.4 4. Source Selection

To declare the data source for an ingestion, you only need to provide an alias for your souce in the `name` field, as well as define the `source_type`. The latter is an enumaration-type field.

For every data source you will be extracting data from, you need to configurate the corresponding Airflow Connection in Airflow's GUI by providing the required credentials.

3.5 5. Ingestion Type

GCP Airflow Foundations support two methods for extracting data out of the source system:

- Full Table Ingestion: All rows of a table (including new, updated, and existing) are extracteed during every ingestion
- Incremental Ingesteion: Only rows that have been added or updated since the last ingestion job are extracted. As an example an *updated_at* column can be useed to identify records that have been updated since a specified time, and then only replicate those records

The ingestion type must be declared in the `ingestion_type` field for each table. Note that you can select a different ingestion type for each table, and some sources support only full table ingestion.

3.6 6. Table Selection

The next step after having selected a data source, is to select the individual tables you need to extract data from. The `tables` field is a list-type field, whose entries are single tables. Start by giving an alias in the `table_name` field. Next, in the `surrogate_keys` field you need to set the columns that will be used as a key to select unique records. These are usually record identifier fields, as well as breakdown dimension fields (e.g. date, geography fields, etc.).

3.6.1 6.1 Configuring an Operational Data Store (ODS)

By default, the destination table will be an Operational Data Store (ODS). An Operational Data Store (ODS) is a table that provides a snapshot of the latest data for operational reporting. As newer records become available, the ODS continuously overwrites older data with either full or incremental data ingestions. With full ingestions, the entire ODS is replaced with the updated data, whereas with incremental ingestions the new data is upserted into the target table

The ODS table will include four metadata columns for each table row:

Table 1: ODS Metadata Columns

Key	Default Name	Description
hash_column_name	metadata_row_hash	The entire row hash
primary_key_hash_column_name	metadata_primary_key_hash	The hash of the primary keys
ingestion_time_column_name	metadata_inserted_at	The ingestion time
update_time_column_name	metadata_updated_at	The update time

Optionally, the user can override the default metadata column names for each table by providing the `ods_config.ods_metadata` field. For example:

```
tables:
    - table_name: campaign_insights
        surrogate_keys: ["account_id", "campaign_id", "date_start"]
        ingestion_type: INCREMENTAL
        facebook_table_config:
            breakdowns: null
            action_breakdowns: ["action_type"]
            column_mapping:
                date_start: date
        ods_config:
            ods_metadata:
```

(continues on next page)

(continued from previous page)

```
            hash_column_name: metadata_row_hash
            primary_key_hash_column_name: metadata_primary_key_hash
            ingestion_time_column_name: metadata_inserted_at
            update_time_column_name: metadata_updated_at
```

3.6.2 6.2 Configuring a Historical Data Store (HDS)

In addition to an ODS destination table, the data can also be ingested in a Historical Data Store (HDS) table. To implement an HDS table, the user can select between a Slowly Changing Diemension Type 2 (SCD2) and a snapshot.

6.2.1 Slowly Changing Diemension Type 2 (SCD2)

In SCD2, a new row is inserted for each change to an existing record in the corresponding target table, as well as for entirely new records. Each record row has metadata timestamp columns that indicate the time of insertion, update, and expiration.

6.2.2 Snapshot

With snapshots, a new partition is appended to the target table at each ETL schedule. Therefore, the target table comprises a collection of snapshots where each partition contains the full dimension at a point in time.

6.2.3 Comparison of SCD2 and Snapshotting

- Even though the SCD2 approach is more computationally efficient, it is also more difficult to maintain and reproduce. Snapshot tables, on the other hand, do not require complex transformations.
- Snapshot tables result in significantly larger tables (since all data is replicated every day) which can result in higher storage costs. However using properly partitioned BigQuery tables mitigates this - partitioned older than 90 days (if they have not been edited) are automatically moved to Big Query long term storage.
- Querying data from a specific day or time ranges is cheaper when using properly partitioned snapshot tables since BigQuery will scan the data only in the appropriate partitions. While doing the same query on SCD2 tables will result in a full table scan.
- Snapshot tables are more intuitive to work with - querying data from a specific date can use the exact same SQL queries used for ODS with the simple addition of filter cluse for that day. While SCD2 requires more complex logic using the created_at and expired_at columns.
- Snapshot tables follow the “functional data engineering ” principle. Most importantly operations are idempotent such that re-running ingestion for the same day will not result in data duplicates or corrupt data.
- Expiring old data is easier with snapshot tables.

6.2.4 Ingesting Data to an HDS Table

To configure an HDS ingestion, the user has to declare the HDS type in under each table. For example:

```
tables:
    - table_name: campaign_insights
        surrogate_keys: ["account_id", "campaign_id", "date_start"]
        ingestion_type: INCREMENTAL
        facebook_table_config:
            breakdowns: null
            action_breakdowns: ["action_type"]
            column_mapping:
                date_start: date
        hds_config:
            hds_table_type: SNAPSHOT
            hds_table_time_partitioning: DAY
            hds_metadata:
                eff_start_time_column_name: metadata_created_at
                eff_end_time_column_name: metadata_expired_at
                hash_column_name: metadata_row_hash
```

Note that the `hds_metadata` field is optional. If not provided the default column names will be used. Also note that the `hds_table_time_partitioning` is only needed for snapshot-type HDS tables, in which case it must match the ingestion schedule.

The HDS table will include four metadata columns for each table row:

Table 2: HDS Metadata Columns

Key	Default Name	Description
hash_column_name	metadata_row_hash	The entire row hash
eff_end_time_column_name	metadata_expired_at	The expiration time (if any)
eff_start_time_column_name	metadata_created_at	The ingestion time

3.7 7. Landing and Destination Datasets

The ingested data will first be stored in a temporary, landing table in BigQuery. The dataset name of the landing tables must be provided in the `landing_zone_options.landing_zone_dataset` field. From the landing dataset, the data are upserted in the destination tables. The destination dataset can be selected in the `dataset_data_name`.

Note: The landing tables are deleted after ingestion.

3.8 8. Column Mapping

Both ODS and HDS ingestions support column mapping and schema migration. When a data field in the data source is desired to have a different name in the destination table, then, the `column_mapping` field can be declared. This is a map-type field, whose keys are the names of columns as they appear in the data source, and the keys are the corresponding names that these columns should have in the destination table.

For example:

```
tables:
    - table_name: campaign_insights
        surrogate_keys: ["account_id", "campaign_id", "date_start"]
        ingestion_type: INCREMENTAL
        facebook_table_config:
            breakdowns: null
            action_breakdowns: ["action_type"]
            column_mapping:
                date_start: date
```

In this example, the `date_start` field extracted from Facebook's API will be mapped to the `date` field in the destination tables.

CHAPTER

FOUR

FEATURES

GCP Airflow Foundations offer a suite of additional features that respond to common data ingestion pitfalls.

4.1 1. Schema Migration

When ingesting from relational database tables and - to a lesser extend - from third party APIs, the source schema might evolve over time. GCP Airflow Foundations will detect such changes after loading the data in a staging table, and update the destination table's schema accordingly. Most schema modifications that are currently supported by BigQuery are also supported here, including:

- Changing a column's data type using the current conversion rules in Standard SQL.
- Relaxing a column's mode from *REQUIRED* to *NULLABLE*

Furthermore, a table is also created in BigQuery to log all the schema migration operations for auditing purposes. The audit table stores information on the table and dataset name, the timestamp of the schema migration, the columns affected, and the type of schema change that was performed.

4.2 2. Post-ingestion Task Dependencies

The data that are ingested are often needed in downstream analytic workflows. These can be orchestrated in the same Airflow instance by utilizing `gcp_airflow_foundations.operators.airflow.external_task.TableIngestionSensor`. From your Python module with the DAG that depends on a table ingestion, you can create a task that waits for the completion of the ingestion. For example:

```
from gcp_airflow_foundations.operators.airflow.external_task import TableIngestionSensor

EXTERNAL_SOURCE_TABLES = {
    'data_source_X':[r"^ABC.*"],
    'data_source_Y':[r".*"]
}

sensor = TableIngestionSensor(
    task_id='table_ingestion_sensor',
    external_source_tables=EXTERNAL_SOURCE_TABLES,
    dag=dag
)
```

The `external_source_tables` argument of `gcp_airflow_foundations.operators.airflow.external_task.TableIngestionSensor` is a dictionary. Each key of the dictionary is a data source and the

value is a list, whose elements are regex expressions that will be matched to the tables under that source. For instance, in the example above, the sensor's state will transition to *success* once 1) the tables of *data_source_X* that start with "ABC" and 2) all tables of *data_source_Y* are ingested.

4.3 3. SQL Workflows with Dataform

Dataform is a framework used to manage data transformation workflows and SQL stored procedures in BigQuery. GCP Airflow Foundations provides a Dataform Operator, such that Dataform runs can be orchestrated in Airflow.

4.3.1 3.1 Invoking Dataform in an External DAG

The Dataform Operator can be used alongside the post-ingestion Operator in your downstream DAG for cases when the data transformation is dependent on the table ingestion DAGs. For example:

```
from gcp_airflow_foundations.operators.airflow.external_task import TableIngestionSensor
from gcp_airflow_foundations.operators.api.operators.dataform_operator import␣
↪DataformOperator

from airflow.models.dag import DAG

EXTERNAL_SOURCE_TABLES = {
    'data_source':['table_to_wait_for']
}

with DAG(
    dag_id="dataform",
    schedule_interval="@daily"
) as dag:

    sensor = TableIngestionSensor(
        task_id='table_ingestion_sensor',
        external_source_tables=EXTERNAL_SOURCE_TABLES,
        dag=dag
    )

    dataform = DataformOperator(
        task_id='dataform_transformation',
        environment='production',
        schedule='dataform_schedule_name',
        dag=dag
    )

    sensor >> dataform
```

4.4 4. Data Processing with Dataflow

GCP Airflow Framework supports ingesting data to BigQuery from relational databases, including Oracle and MySQL, using Dataflow jobs. Dataflow is used to ingest the data into a landing zone in BigQuery. Then the data in landing zone is ingested into the ODS and HDS using the common functionality of the framework. ods.

An example configuration file for migrating Oracle tables to BigQuery using Dataflow can be found here: *Oracle*.

4.5 4.1 Configuring a DataFlow Job

Several prerequisites are required for the ingestion:

A driver .jar file must be available in a bucket on Cloud Storage. For example, for Oracle ingestion an ojdbc{#}.jar file should be provided A path to the compiled Dataflow template must be available in a bucket on Cloud Storage. The JdbcToBigQuery.java template and instructions to compile it are provided by GCP at this repository. Another folder in Cloud Storage should be reserved for the Dataflow job temporary write directory.

A simple way to configure this would be to create a dedicated GCS bucket per source ingestion, with subfolders for the template file, driver and temporary storage. The rest of the configuration are specified in the jdbc config class, and include jdbc credentials, dataflow job parameters, etc.

4.6 4.2 Schema Auto-Detection

As an optional first step in the ingestion the Dataflow job queries the relational database metadata table to retrieve all the tables and their schema. The schema is then compared against the target table schema, and proper schema evolution rules are followed for schema migration schema_migration. This will occur if the configuration parameter "ingest_metadata" is set to True. If the configuration parameter "ingest_metadata" is set to False, however, then a BigQuery table with the same content should be created manually, and the configuration parameter "bq_schema_table" should point to it.

4.7 4.3 Deployment into subnetworks

In many use-cases, Composer is deployed in a network that doesn't have direct access to the source databases. For security purposes it is desirable to allow access to databases only from specific subnetworks. Instead of having to deploy a separate Composer cluster in each subnetwork, Dataflow jobs are configurable to run from separate subnetworks for each source. The source configuration parameter "subnetwork" should be specified as the full subnetwork name in the form: regions/{region}/subnetworks/{subnetwork}.

4.8 4.4 Work In Progress

- Auto ingestion of all tables in a database based on inclusion and exclusion rules (e.g. based on regular expressions, by schema in Oracle).
- Dataflow job partitioning of large table ingestions

CHAPTER

FIVE

DATA LOST PREVENTION

GCP Airflow Foundations provides a blueprint for deploying a secure Data Lake, with abiltiy to discover, classify, and protect your most sensitive data

5.1 1. Landing Zone

Ingestion from all sources reuses a common pattern of first loading the data into and landing_zone in BigQuery and then upserting the data into the ODS and HDS. All data in the landing zone is deleted after a configurable period of time (7 days by default).

Further all intermideiate GCS files are also deleted after the ingestion is finished

5.2 2. DLP integration and column level securiry

As part of the ingestion configuration in Airflow, DLP scan can be enabled for each table - it will run the first time the table is ingest + on a configurable schedule (once a month, etc.) During each scheduled run, Airflow will

1. Run a DLP inspection job on the ODS and HDS tables - it scans the table to detect sensitive data based on a pre-configured template which is a set of InfoType (pre created rule for detecting common sensitive data)
2. Read the results of the job, and if sensitive data is detected in a column, it will apply pre-configured policy tags on the column
3. Policy tag enforcement (column level security) can be enabled for a specific taxonomy (set of tags)

A sample configuration file to enable DLP integration as part of ingestion can be found : here

5.3 3. VPC service control

While outside of the scope of this framework, we recommend deploying Composer as part of a secure service permienter to mitigate the risk of data exfiltration. For more details please see these instructions

CHAPTER

SIX

DEVELOPING CUSTOM DATA SOURCES

GCP Airflow Foundations can be readily expanded to ingest data from APIs that are not built-in. This is possible by creating a class that inherits from the abstract class `gcp_airflow_foundations.source_class.source.DagBuilder` and implements the abstract method `get_bq_ingestion_task`, which returns the Airflow task that ingests data from the external API to a BigQuery staging table. You may have to provide your own Airflow Operators for your data source, if one is not available by the Airflow community.

For example, implemented bellow for Google Ads using the custom `GoogleAdsQueryToBqOperator` Operator:

```
from data_sources.google_ads.operators.ads import GoogleAdsQueryToBqOperator
from data_sources.google_ads.config.google_ads_config import ResourceType
from data_sources.google_ads.config.google_ads_config import GoogleAdsTableConfig

from airflow.models.dag import DAG

from gcp_airflow_foundations.source_class.source import DagBuilder


class GoogleAdstoBQDagBuilder(DagBuilder):

    source_type = "GOOGLE_ADS"

    def get_bq_ingestion_task(self, dag, table_config):

        data_source = self.config.source

        google_ads_config = GoogleAdsTableConfig(**table_config.extra_options['google_ads
↪'])

        client_ids = data_source.extra_options['manager_accounts']

        query_operator = GoogleAdsQueryToBqOperator(
            task_id="google_ads_to_bq",
            client_ids=client_ids,
            manager_accounts=data_source.extra_options['manager_accounts'],
            resource_type=google_ads_config.resource_type,
            project_id=data_source.gcp_project,
            dateset_id=data_source.landing_zone_options.landing_zone_dataset,
            table_id=table_config.landing_zone_table_name_override,
            api_version = "v8",
            dag=dag
        )
```

(continues on next page)

(continued from previous page)

```
        return query_operator
```

Note: You will need to import your implementation of `gcp_airflow_foundations.source_class.source.DagBuilder` in your `parse_dags.py` module inside your `dags_folder`.

CHAPTER

SEVEN

SAMPLE CONFIGURATION FILES

7.1 Oracle

GCP Airflow Foundations supports data warehouse migration from an Oracle database to BigQuery using Dataflow.

For a detailed description and data type of each configuration field, please refer to *gcp_airflow_foundations.base_class.dataflow_job_config.DataflowJobConfig*.

```
source:
    name: CSG
    source_type: ORACLE
    ingest_schedule: "@daily"
    start_date: "2021-01-01"
    extra_options:
        dataflow_job_config:
            system_name: CSG
            region: us-central1
            bq_load_temp_directory: <GCS directory for loading temporary Dataflow files>
            template_path: <GCS path to Dataflow template>
            jdbc_driver_class: oracle.jdbc.driver.OracleDriver
            jdbc_jar_path: <the GCS path to the driver .jar file>
            jdbc_url: <a valid JDBC url for connecting to the database>
            jdbc_user: <the database username>
            jdbc_pass_secret_name: <the database password>
            kms_key_path: <the KMS key path for encrypting/decrypting JDBC credentials>
            sql_casts: {"DATE": "to_char(COLUMN, 'yyyy-mm-dd') as COLUMN"}
            table_type_casts: {}
            bq_schema_table: ALL_TAB_COLUMNS
            database_owner: <owner of the tables to query (query scope)>
    location: US
    dataset_data_name: oracle
    connection: google_cloud_default
    landing_zone_options:
        landing_zone_dataset: staging_zone
tables:
    - table_name: oracle_table
        ingestion_type: FULL
        surrogate_keys: []
        hds_config:
        hds_table_type: SNAPSHOT
        hds_table_time_partitioning: DAY
```

7.2 Salesforce

For a detailed description and data type of each configuration field, please refer to *gcp_airflow_foundations.base_class.salesforce_ingestion_config.SalesforceIngestionConfig*.

```yaml
source:
    name: salesforce
    source_type: SALESFORCE
    ingest_schedule: "@daily"
    start_date: "2021-01-01"
    extra_options:
        gcs_bucket: data-lake-bucket
    location: US
    dataset_data_name: salesforce
    landing_zone_options:
        landing_zone_dataset: landing_zone
tables:
    - table_name: Opportunity
        ingestion_type: FULL
        surrogate_keys: []
        hds_config:
        hds_table_type: SNAPSHOT
        hds_table_time_partitioning: DAY
        extra_options:
            sf_config:
                ingest_all_columns: False
                fields_to_omit: []
                field_names: ["Id", "OwnerId", "Name", "Amount", "StageName"]
                api_table_name: Opportunity
    - table_name: Account
        ingestion_type: FULL
        surrogate_keys: []
        hds_config:
        hds_table_type: SNAPSHOT
        hds_table_time_partitioning: DAY
        extra_options:
            sf_config:
                ingest_all_columns: False
                fields_to_omit: []
                field_names: ["Id","Name"]
                api_table_name: Account
```

CHAPTER

EIGHT

API REFERENCE

8.1 Base Class

8.2 Configuration Data Classes

gcp_airflow_foundations.base_class.source_config.SourceConfig(...)	Source configuration data class.
gcp_airflow_foundations.base_class.source_table_config.SourceTableConfig(...)	Table configuration data class.
gcp_airflow_foundations.base_class.salesforce_ingestion_config.SalesforceIngestionConfig(...)	Salesforce configuration class.
gcp_airflow_foundations.base_class.dataflow_job_config.DataflowJobConfig(...)	Dataflow configuration class.

class gcp_airflow_foundations.base_class.source_config.**SourceConfig**(*name: str, source_type: str, ingest_schedule: str, external_dag_id: ~typing.Optional[str], gcp_project: str, dataset_data_name: str, dataset_hds_override: ~typing.Optional[str], extra_options: ~typing.Optional[dict], landing_zone_options: ~gcp_airflow_foundations.base_class.landing_ acceptable_delay_minutes: int, notification_emails: ~typing.List[str], owner: str, partition_expiration: ~typing.Optional[int], dag_args: ~typing.Optional[dict], location: str, start_date: str, schema_options: ~gcp_airflow_foundations.base_class.schema_ = SchemaOptionsConfig(schema_source_type=<SchemaSourceType 'AUTO'>, schema_object_template=None), facebook_options: ~typing.Optional[~gcp_airflow_foundations.base_ = None, full_ingestion_options: ~gcp_airflow_foundations.base_class.source_ = FullIngestionConfig(ingest_all_tables=False, ingestion_name='', dag_creation_mode='TABLE', regex_table_pattern='ANY'), catchup: bool = True, start_date_tz: str = 'EST', ods_suffix: str = '', hds_suffix: str = '', dagrun_timeout_mins: int = 1440, version: int = 1, sla_mins: int = 900, dlp_config: ~typing.Optional[~gcp_airflow_foundations.base_ = None, num_retries: int = 3, email_on_retry: bool = False, email_on_failure: bool = True, connection: str = 'google_cloud_default'*)

Source configuration data class.

name

Name of source

Type
str

`source_type`

Source type selection. See SourceType class

Type
str

`ingest_schedule`

Ingestion schedule. Currently only supporting @hourly, @daily, @weekly, and @monthly

Type
str

`gcp_project`

Google Cloud Platform project ID

Type
str

`dataset_data_name`

Target dataset name

Type
str

`extra_options`

Google Cloud Storage bucket and objects for source data if loading from GCS

Type
Optional[dict]

`landing_zone_options`

Staging dataset name

Type
gcp_airflow_foundations.base_class.landing_zone_config.LandingZoneConfig

`acceptable_delay_minutes`

Delay minutes limit

Type
int

`notification_emails`

Email address for notification emails

Type
List[str]

`owner`

Airflow user owning the DAG

Type
str

`partition_expiration`

Expiration time for HDS Snapshot partitions in days.

Type
Optional[int]

facebook_options

Extra options for ingesting data from Facebook Marketing API.

Type
Optional[gcp_airflow_foundations.base_class.facebook_config.FacebookConfig]

catchup

Run all dag runs since start_date. https://airflow.apache.org/docs/apache-airflow/stable/dag-run.html#catchup

Type
bool

dag_args

Optional dictionary of parameters to be passed as keyword arguments to the ingestion DAG. Refer to `airflow.models.dag.DAG` for the available parameters.

Type
Optional[dict]

location

BigQuery job location.

Type
str

start_date

Start date for DAG

Type
str

start_date_tz

Timezone

Type
str

ods_suffix

Suffix for ODS tables. Defaults to empty string.

Type
str

hds_suffix

Suffix for HDS tables. Defaults to empty string.

Type
str

version

The Dag version. Can be incremented if logic changes

Type
int

sla_mins

Service Level Agreement (SLA) timeout minutes. This is is an expectation for the maximum time a Task should take.

Type
int

`num_retries`

Number of retries for the DAG before failing - https://airflow.apache.org/docs/apache-airflow/stable/tutorial.html

Type
int

`email_on_retry`

Whether the DAG should email on retries

Type
bool

`email_on_failure`

Whether the DAG should email on failure

Type
bool

`connection`

Aiflow Google Cloud Platform connection

Type
str

class gcp_airflow_foundations.base_class.source_table_config.**SourceTableConfig**(*table_name: str, ingestion_type: ~gcp_airflow_foundations.enum land- ing_zone_table_name_override ~typ- ing.Optional[str], dest_table_override: ~typ- ing.Optional[str], surro- gate_keys: ~typ- ing.List[str], col- umn_mapping: ~typ- ing.Optional[dict], clus- ter_fields: ~typ- ing.Optional[~typing.List[str]], col- umn_casting: ~typ- ing.Optional[dict], new_column_udfs: ~typ- ing.Optional[dict], hds_config: ~typ- ing.Optional[~gcp_airflow_fou start_date: ~typ- ing.Optional[str], ex- tra_options: dict = <factory>, face- book_table_config: ~typ- ing.Optional[~gcp_airflow_fou = Facebook- TableCon- fig(api_object=<ApiObject.INS 'IN- SIGHTS'>, break- downs=[], ac- tion_breakdowns=[]), start_date_tz: ~typ- ing.Optional[str] = 'US', ods_config: ~typ- ing.Optional[~gcp_airflow_fou*

Table configuration data class.

`table_name`

Table name. Used for Dag Id.

Type
str

`ingestion_type`

FULL or INCREMENTAL.

Type
gcp_airflow_foundations.enums.ingestion_type.IngestionType

`landing_zone_table_name_override`

Optional staging zone table name.

Type
Optional[str]

`dest_table_override`

Optional target table name. If None, use table_name instead.

Type
Optional[str]

`surrogate_keys`

Keys used to identify unique records when merging into ODS.

Type
List[str]

`column_mapping`

Mapping used to rename columns.

Type
Optional[dict]

`cluster_fields`

The fields used for clustering. BigQuery supports clustering for both partitioned and non-partitioned tables.

Type
Optional[List[str]]

`column_casting`

Mapping used to cast columns into a specific data type. Note column name uses that of the landing zone table.

Type
Optional[dict]

`ods_config`

ODS table configuration. See `gcp_airflow_foundations.base_class.ods_table_config.OdsTableConfig`.

Type
Optional[gcp_airflow_foundations.base_class.ods_table_config.OdsTableConfig]

hds_config

HDS table configuration. See gcp_airflow_foundations.base_class.hds_table_config.HdsTableConfig.

Type
Optional[gcp_airflow_foundations.base_class.hds_table_config.HdsTableConfig]

facebook_table_config

Extra options for ingesting data from the Facebook API.

Type
Optional[gcp_airflow_foundations.base_class.facebook_table_config.FacebookTableConfig]

extra_options

Field for storing additional configuration options.

Type
dict

start_date

Start date override for DAG

Type
Optional[str]

start_date_tz

Timezone

Type
Optional[str]

version

The Dag version for the table. Can be incremented if logic changes.

Type
int

catchup

Passed to a dag [see doc](https://airflow.apache.org/docs/apache-airflow/stable/dag-run.html#catchup). Defaults to True. May want to change it to False if Dag version is changed, and we don't want to rerun past dags.

Type
bool

class gcp_airflow_foundations.base_class.salesforce_ingestion_config.**SalesforceIngestionConfig**(*api_table_*
str,
in-
gest_all_co
bool,
fields_to_o
Op-
tional[List[
field_name
Op-
tional[List[

Salesforce configuration class.

ingest_all_columns

SELECT * the Salesforce object if true

Type
bool

fields_to_omit

a list of object fields to omit from ingestion

Type
Optional[List[str]]

field_names

an explicit list of fields to ingest

Type
Optional[List[str]]

class gcp_airflow_foundations.base_class.dataflow_job_config.**DataflowJobConfig**(*system_name: str*, *project: str*, *region: str*, *subnetwork: str*, *bq_load_temp_directory: str*, *template_path: str*, *jdbc_driver_class: str*, *jdbc_jar_path: str*, *jdbc_url: str*, *jdbc_user: str*, *jdbc_pass_secret_name: str*, *kms_key_path: str*, *sql_casts: Optional[dict]*, *bq_schema_table: str*, *database_owner: str*, *connection_pool: str*, *max_retry_delay: int = 60*)

Dataflow configuration class.

project

the GCP project in which the Dataflow job runs

Type
str

region

the region in which the Dataflow job should run

Type
str

subnetwork

the specific subnetwork in which the Dataflow job should run

Type
str

bq_load_temp_directory

GCS directory for loading temporary Dataflow files

Type
str

template_path

GCS path to Dataflow template

Type
str

jdbc_driver_class

the name of the JDBC driver class to use (e.g. oracle.jdbc.driver.OracleDriver)

Type
str

jdbc_jar_path

the GCS path to the driver .jar file

Type
str

jdbc_url

a valid JDBC url for connecting to the database

Type
str

jdbc_user

the database username

Type
str

jdbc_pass_secret_name

the secret name of the database password

Type
str

kms_key_path

the KMS key path for encrypting/decrypting JDBC credentials

Type
str

`sql_casts`

a dictionary of sql casts to use when querying the source DB

Type
Optional[dict]

`database_owner`

owner of the tables to query (query scope)

Type
str

INDEX