Cloud Composer Framework Documentation

Introduction

Cloud Composer: It is a **fully managed** workflow orchestration service provided by Google Cloud Platform (GCP). Fully managed means - we do not need to worry about setting up servers, configuring software, or monitoring infrastructure.

Analogy:

Workflows are defined as Directed Acyclic Graphs (DAGs), representing a sequence of tasks with dependencies.

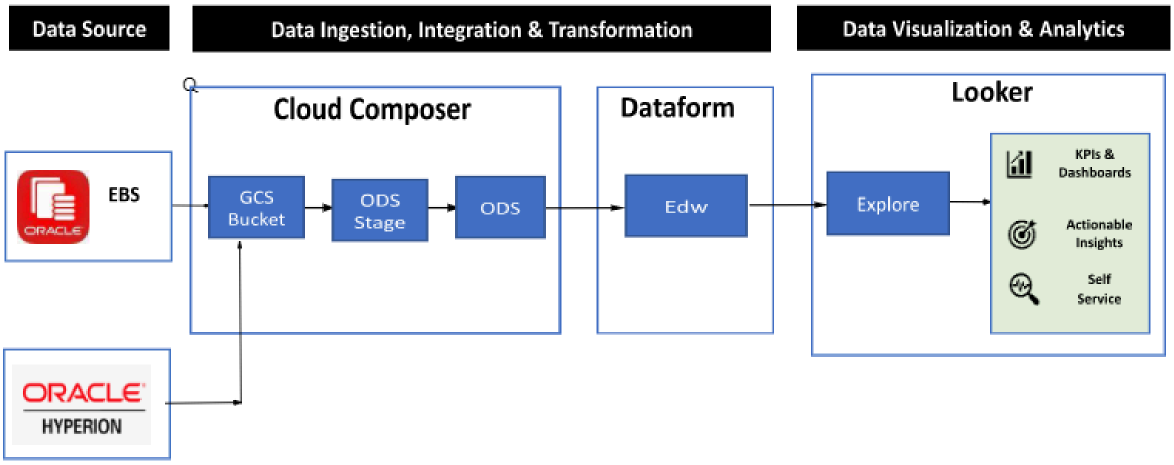
Key Benefits of Cloud Composer:

- Automation: Automate repetitive tasks.
- Flexibility
- Reliability
- Cost Efficiency

Organizations have and continue to use Cloud Composer to streamline their data workflows and achieve business targets.

Airflow: Apache Airflow is an open-source platform for workflow orchestration, which forms the core of Cloud Composer's functionality.

A traditional use is shown in below diagram :



As shown in the previous diagram, Cloud Composer simplifies data integration between various sources and Big Query, making data readily available for analysis. Traditional development approaches often involve complex and time-consuming DAG (Directed Acyclic Graph) management within Cloud Composer, which can be a bottleneck.

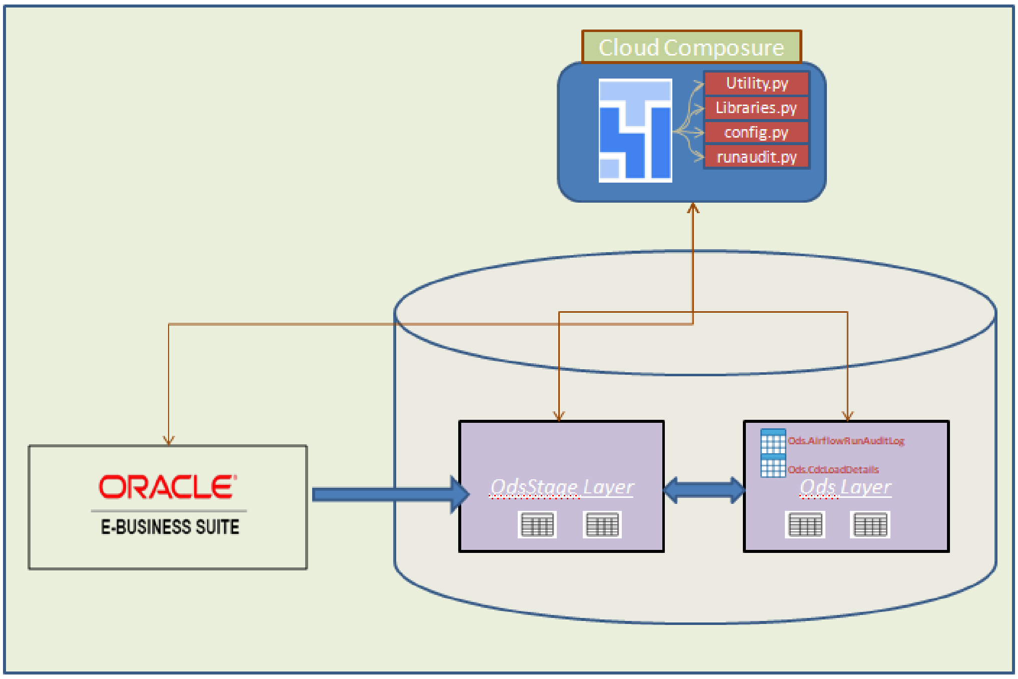
Considering a scenario where you need data directly loaded from its source to GCP and readily available for analysis, the ideal solution should be:

- **Ready to Use:** Easy to implement without extensive technical knowledge.
- **Scalable:** Adapts to accommodate growing data volumes.
- **Easy to Manage:** Requires minimal ongoing maintenance.
- **Extensible:** Allows for future modifications and integrations.
- **Standardized:** Maintains consistency with organizational best practices.

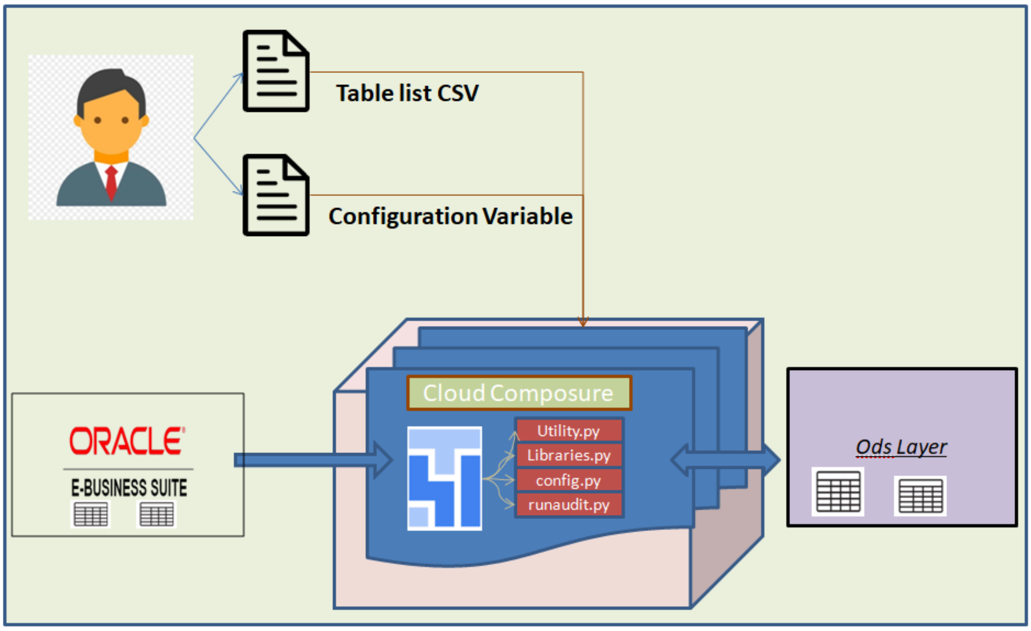
Cloud Composer Framework Documentation

Architecture

The following diagram illustrates how Cloud Composer infrastructure can be leveraged to achieve these goals while maintaining a user-friendly interface



Here data needed in the ODS Layer without any complexity. **This is where our solution comes in!** We eliminate complexity in the ODS layer by offering a user-friendly, 'black box' product. Anyone with minimal computer knowledge can manage and handle it. Our solution takes care of the complex technical aspects, transforming them into a simple, manual flow that we've designed and developed. The following diagram illustrates the user experience after implementing our solution.



Cloud Composer Framework Documentation

Key Benefit & Features

- **Reusability/Generalization**: The code is built with reusable classes (Classes- Utils, Config) which can be utilized for different purposes in future, also it is extendable (if needed small code blocks could be added for new functionality). Both source table details and config parameters are passed to the main code dynamically, instead of hardcoding values anywhere, hence making the code dynamic.

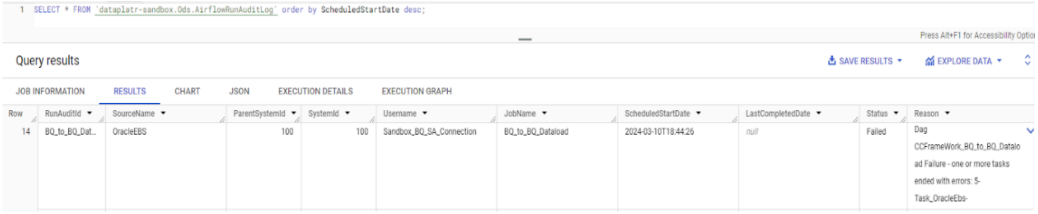
  Class Utils: This class contains below reusable methods:

  - **read_json**: read config_variables.json file and return configs in map value format.
  - **print_configs**: Prints config variables in given arguments
  - **get_SourceExtractSql**: Generate Source Data Extraction SQL based on Incremental flag and CDC Column values
  - **get_SourceTablePrimaryKeySql**: Generate temp table based on primary keys mentioned for different source tables in Source List file.
  - **get_DynamicLoadSql**: If load is not incremental then Generate Insert SQL else generate Update or insert SQL based on comparison between source and target dataset values.
  - **get_CdcLoadDate**: Get CDC Load Date value for specified parameters from CDCLoadDetails table.
  - **get_max_last_update_date**: Get max value of CDC Update date for specified Bigquery table.
  - **update_cdc_load_status**: Update CdcLoadDetails table with Load execution status for a given table.
  - **get_UpsertCDCLoadDetails**: Insert/Update records in the CdcLoadDetails table.
  - **get_UpdateInactiveInd**: Update inactive indicator, date and reason in target table.

  Class Config : This class contains reusable methods to get DAG Id, name, Config path, config variable path, source table list.

- **Ready to use code (lesser development time):** Infrastructure details are captured in a single Configuration file(config_variables.json) stored in the Configurations folder, which can be easily modified. Source table details are captured in a separate file (SourceTableList.csv).
- **Plug and Play approach**: Code follows basic schema/ETL layer structure - ODS Stage→ODS→Edw; if the project follows a similar structure, they can just plug in this code and utilize it directly.
- **Monitoring** : Info Logs captured after every executable so users can verify intermediate outcomes and point out any intermediate issues quickly. Also, error handling is done throughout the code.

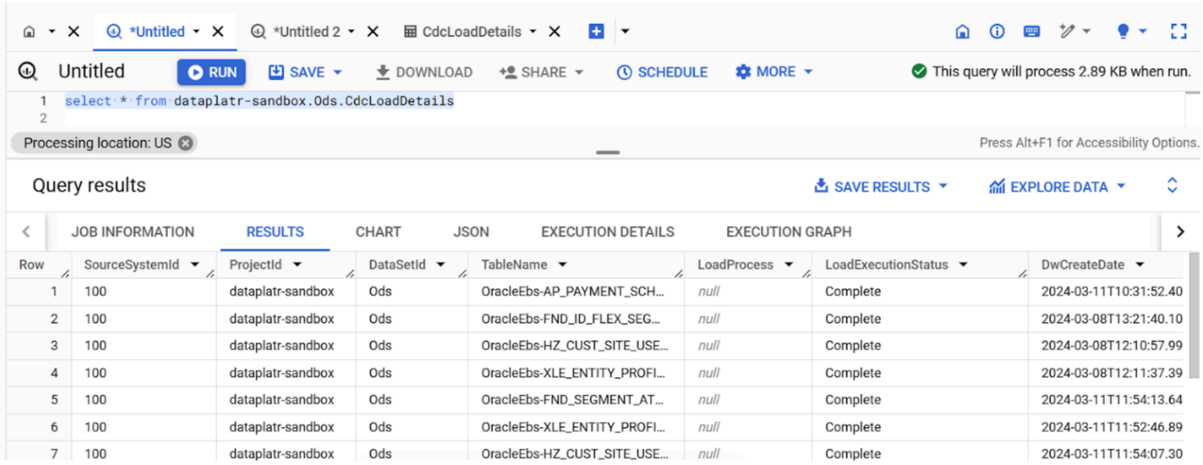
  The below table shows the Airflow error/success logs of the DAG. (Table name – AirflowRunAuditLog)



```
SELECT * FROM `dataplatr-sandbox.Ods.AirflowRunAuditLog` order by ScheduledStartDate desc;
```

Row	RunAuditId	SourceName	ParentSystemId	SystemId	Username	JobName	ScheduledStartDate	LastCompletedDate	Status	Reason
14	BQ_to_BQ_Dat...	OracleEBS	100	100	Sandbox_BQ_SA_Connection	BQ_to_BQ_Dataload	2024-03-10T18:44:26	null	Failed	Dag CCFrameWork_BQ_to_BQ_Dataload Failure - one or more tasks ended with errors: 5-Task_OracleEbs-

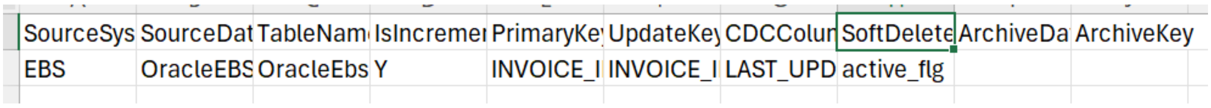
The below table holds the information about the last load dates of a table which will be used for the Change Data Capture. (Table name – CdcLoadDetails)



```
select * from dataplatr-sandbox.Ods.CdcLoadDetails
```

Row	SourceSystemId	ProjectId	DataSetId	TableName	LoadProcess	LoadExecutionStatus	DwCreateDate
1	100	dataplatr-sandbox	Ods	OracleEbs-AP_PAYMENT_SCH...	null	Complete	2024-03-11T10:31:52.40
2	100	dataplatr-sandbox	Ods	OracleEbs-FND_ID_FLEX_SEG...	null	Complete	2024-03-08T13:21:40.10
3	100	dataplatr-sandbox	Ods	OracleEbs-HZ_CUST_SITE_USE...	null	Complete	2024-03-08T12:10:57.99
4	100	dataplatr-sandbox	Ods	OracleEbs-XLE_ENTITY_PROFI...	null	Complete	2024-03-08T12:11:37.39
5	100	dataplatr-sandbox	Ods	OracleEbs-FND_SEGMENT_AT...	null	Complete	2024-03-11T11:54:13.64
6	100	dataplatr-sandbox	Ods	OracleEbs-XLE_ENTITY_PROFI...	null	Complete	2024-03-11T11:52:46.89
7	100	dataplatr-sandbox	Ods	OracleEbs-HZ_CUST_SITE_USE...	null	Complete	2024-03-11T11:54:07.30

- **Custom Demand Execution control** : In the SourceTableList.csv file, users can easily add/delete which tables to load as well as specify the Primary Key, update key, Incremental flag, CDC Column, Soft delete column parameters.



SourceSys	SourceDat	TableNam	IsIncremer	PrimaryKe	UpdateKey	CDCColun	SoftDelete	ArchiveDa	ArchiveKey
EBS	OracleEBS	OracleEbs	Y	INVOICE_I	INVOICE_I	LAST_UPD	active_flg		

- **Readability/Agility** : Function definitions/comments are added before each block of code so it is easy to navigate through the code and update/modify it.

- **Scalability**: Our approach is providing the flexibility to scale the model horizontally (add more tasks/functionalities) and/or vertically based on the user requirement. i.e. we can add huge source tables as part of batch processing and the tasks would run in parallel. Currently, we have successfully executed the code for 14 tables, and more could be added if needed.

Cloud Composer Framework Documentation

Use Cases

In the following, we are showing a use case which takes the source as Big Query and target as Big Query.

Decoding the DAG code:

The DAG code sets up a data pipeline DAG in Airflow to orchestrate a series of tasks for extracting, transforming, and loading data between different data sources and destinations (e.g., Google Cloud Storage, Big Query).
It also includes mechanisms for error handling, logging, and audit tracking to ensure reliability and traceability of the data pipeline.
The modularized DAG aims at making the ETL/ELT experience as smooth and intervention free as possible.In this direction, all the functionalities are divided among different modules and these modules are accessed through objects in the main DAG code.

- It starts with importing standard Python libraries (os, sys, json, csv) as well as specific functions from the pyspark.sql.functions module.

- For the second step, it retrieves the absolute path of the current Python script file (__file__) as: filePath = os.path.abspath(__file__)

- Then it adds the path to the "CommonPattern" folder to the Python system path. This folder contains utility functions or modules used by the script.

- Importing Libraries and modules: Here it imports custom modules (config, utils, runaudit) and specific classes or functions from these modules.

- Configuration Setup: This section creates instances of configuration and utility objects (Config, Utils, RunAudit) using the classes imported earlier.

- Extraction of configuration Variables: Here, the code reads configuration variables from a
- JSON file(config_variables.json) using a method (read_json) defined in the Utils custom class.

- Instantiating Configuration variables: assigning values to various configuration variables
- based on the data read from the JSON file.

- DAG Initialization: Initializes the DAG with specific parameters such as DAG ID, start date,
- default arguments, schedule interval, concurrency, etc.

- DAG Task Definition: Tasks thus defined are as follows:

  Start: A dummy task representing the start of the workflow.

  Task_Print_config_variables: This task prints configuration variables.
  It is defined to call a Python function utilsObj.printconfigs .

  DataFrame creation:Creation of a pandas dataframe to hold the contents of a file named SourceTableList.csv as shown in fig:1.

  Iterating through the CSV file: The script reads the CSV file containing source table configurations and iterates through each row of the Data Frame created from the CSV file. For each row, it performs a series of operations to extract configuration details and execute tasks (from (iv) to (viii))accordingly.

Generation of variables required for Error / Success Logging:
Generates an audit ID as : runauditid = f"{dagName}-{formatted_start}
and creates a dictionary containing audit information for logging purposes as below:

iAudit_dict = { ... }
The dictionary includes various key-value pairs representing different attributes of the audit log. The dictionary will be used for logging audit information related to the execution of the DAG, providing details such as the start time, associated datasets, system IDs etc.

Task_InsertRunAudit: This task is responsible for executing the insert_audit_log method of the runAuditObj object.

oracle_to_gcs_task: It uses the Airflow operator "OracleToGCSOperator" to extract the data from Oracle source to GCS bucket using SQL query (ExtractSql).

gcs_to_bq_task: It uses the Airflow operator "GCSToBigQueryOperator" to migrate data from gcs bucket to target BQ table with the specified configurations.

Source_to_OdsStage_TempTable_task: A task to create a temporary table in the staging area to store the PKs of all the tables being loaded so as to implement Hard delete logic.

OdsStage_to_Ods_task: A task to merge data from the staging table to the final ODS table.

Upsert_Cdc_task: A task to upsert data into the CDC (Change Data Capture) load details table. This CDC Load Details table is maintained to implement CDC for the further delta loads.In this task, this table is populated with max(last update date) of the Ods as the CDCUpdateDate that gives the most recent time the ODS table was updated.

update_inactive_indicator_task: A task to update the inactive indicator in the ODS table.

Task_Archive: Bash command to move data from extracted path to archive path once data load is completed.

Task_Trigger_DataformDag: A task designed to initiate the transfer of data from ODS to EDW using Dataform.

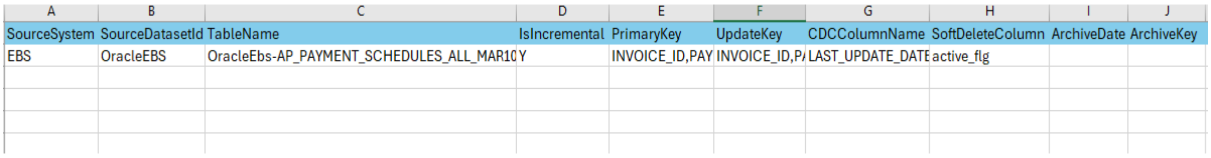
Cloud Composer Framework Documentation

Task Flow

Start >> Task_Print_config_variables >> Task_InsertRunAudit >> Task_GetRunAudit >> oracle_to_gcs_task >> gcs_to_bq_task >> Source_to_OdsStage_PrimaryKeyData >> OdsStage_to_Ods >> Upsert_Cdc >> Update_Inactive_Indicator >> Task_Archive >> Task_Trigger_DataformDag >> End

Source Table List

The file contains the following fields:



	A	B	C	D	E	F	G	H	I	J
1	SourceSystem	SourceDatasetId	TableName	IsIncremental	PrimaryKey	UpdateKey	CDCColumnName	SoftDeleteColumn	ArchiveDate	ArchiveKey
2	EBS	OracleEBS	OracleEbs-AP_PAYMENT_SCHEDULES_ALL_MAR10	Y	INVOICE_ID,PAY	INVOICE_ID,P/	LAST_UPDATE_DATE	active_flg		

Fig:1 SourceTableList.csv

- **SourceSystem:** Type of source system
- **SourceDatasetId:** Source schema / datasetId / catalog – based on the source type.
- **TableName:** The list of tables to be processed
- **IsIncremental:** Whether the table should get Incremental / Full Loads
- **PrimaryKey**: Primary key of each table
- **UpdateKey**: Update keys, if it is different from primary keys. (This is a feature which can be used in future)
- **CdcColumnName:** The column in source used for CDC logic
- **SoftDeleteColumn:** Represents a column which signifies the DELETE status of a record.
- **Archive Date:** Used as a minimum limit date, while updating the HARD DELETED records.
- **Archive Key:** Represents the column name of having Archive Date.

Utilities Functionalities

Dynamic Extract SQL: The dynamic extract sql that loads data from source to stage table is generated through a function 'get_SourceExtractSql' from utility class. Here, it produces the query based on the source system type, whether the IsIncremental flag is 'Y' or 'N' and the presence of CDC column as illustrated below:

- If IsIncremental is 'Y' and CDCColumnName is not None: This condition selects all columns from the specified table where the CDC column i.e., LastUpdateDate is greater than the provided CDC update date (in Cdc Load Details table).
- If IsIncremental is 'Y' and CDCColumnName is None: This condition selects all columns from the specified table without any CDC column filter.
- If IsIncremental is 'N' and CDCColumnName is None: This condition selects all columns from the specified table without any CDC column filter.
- If IsIncremental is 'N' (Full load) and CDCColumnName is not None: This condition selects all columns from the specified table where the CDC column i.e., LastUpdateDate is greater than the provided CDC update date.
  Note:In the case of full load, the Cdc Update date is the initial extract date.

Dynamic Merge SQL: The dynamic merge sql that merges the data from OdsStage to Ods table is generated through a method 'get_DynamicLoadSql' from utils class.
The method is able to gather information about the columns and their data types in the source table, as well as the data types of the primary key columns specifically.

This information can then be used to dynamically generate SQL statements for loading data into a target table, ensuring compatibility with the source schema.

Update Inactive Indicator: Here, the code updates the source hard deleted records in the Ods table through a method ‘get_UpdateInactiveIndSql’ in utils class. This function generates a SQL query that updates records in a target table based on certain conditions, including the presence of corresponding records in the temporary table (created to store the PK of all the tables) from the source dataset and optional archival conditions.

A comparison is carried out between the PK of the temporary table and the Ods table.
If the Pk of the table goes missing in the temp table, it updates the InactiveInd flag as Y and vice versa.

Archive Dates Consideration: During the process of updating the 'InactiveInd' flag to 'Y' in the ODS table, it is imperative to consider archive dates and archive key columns contained within the CSV file. Specifically, the 'InactiveInd' flag will be updated for records meeting the condition of having dates later than the specified archive dates.

Error Logging: To Log Error / Success information, the code calls a method ‘insert_audit_log’ from run_audit_module. The module has the following functions:

- insert_audit_log function,
- update_audit_info function,
- get_runid_timeframe function

These functions work together within an Airflow DAG to insert and update audit logs in Google BigQuery, providing logging functionality for any errors encountered during the process. An email gets generated as per the status of the task whether.