# **Import Fusion PVOs to Big Query – Automation: Step-by-Step process.**

This document explains the step-by-step process to create a new batch to sync fusion PVOs to BigQuery.

1. **Meta data Entries to FUSION\_TO\_BQ\_PVO\_DTL table:**

***Preparing Development Environment:***

The development environment must be set up in developer’s local machine in order to execute the code to prepare and insert entries into meta table.

1. Create a clone of the git repository.

<https://github.com/equinix-enterpriseapps/analytics-finance-gcp>

1. Create a new folder fusion\_automate/fusbq in local machine.
2. Copy the contents of the folder /analytics-finance-gcp/fusbq/fusion-cloud-composer/dags from the GIT Repo into the new folder fusion\_automate/fusbq.
3. Create a new folder fusion\_automate/config in local machine.
4. Copy the /analytics-finance-gcp/fusbq/fusion-cloud-composer/config/dev/fusion\_to\_bq\_sync\_config.jsoninto the folder fusion\_automate/config.
5. Create a virtual environment for the project and activate it.
6. Install below python packages in the virtual environment.

pip install pandas

pip install google-cloud-storage

pip install google-cloud-bigquery

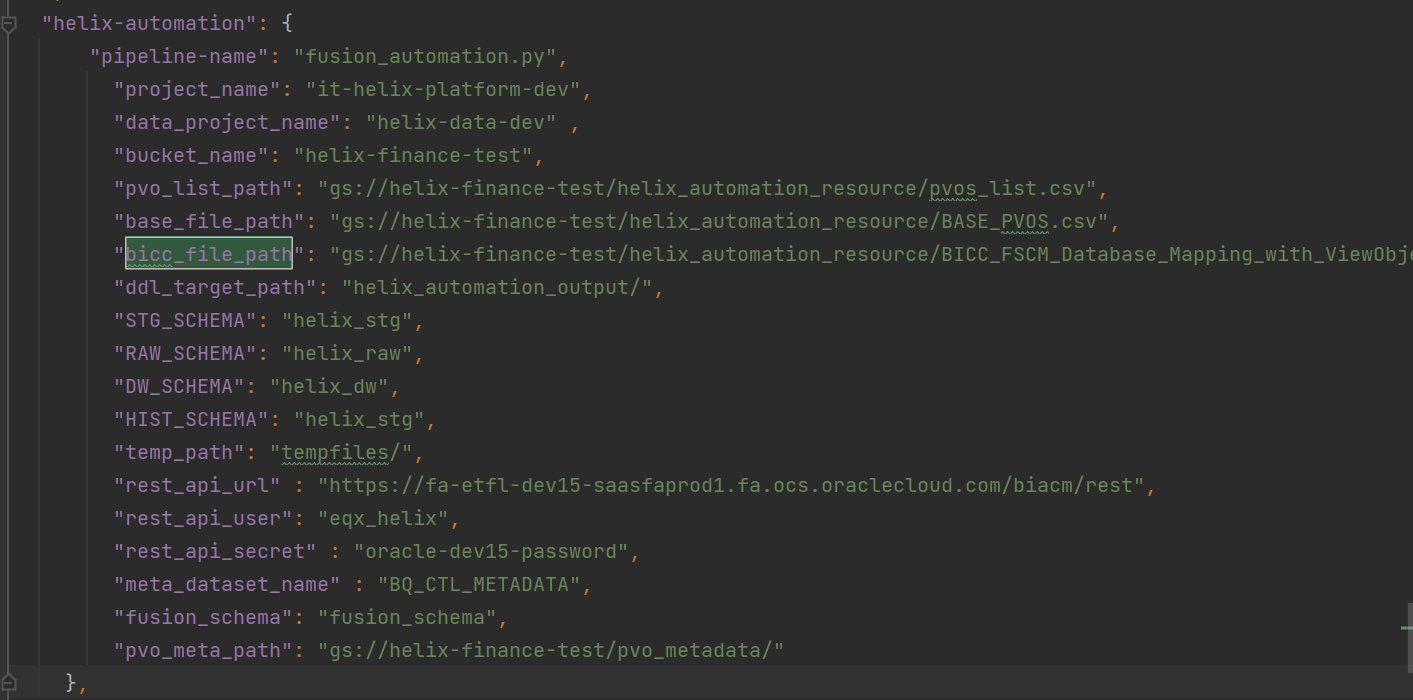
pip install fsspec

pip install gcsfs

pip install apache-airflow

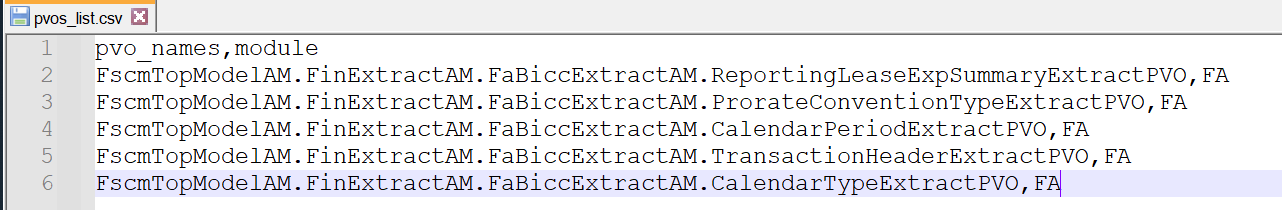
pip install apache-airflow-providers-google

1. The config/fusion\_to\_bq\_sync\_config.json file contains “helix-automation” node which having below properties which are all used for automation process :



|  |  |
| --- | --- |
| Property Name | Description |
| pipeline-name | name of the main python file which creates the DAG for automation. |
| project\_name | GPC Project name/id in which the composer environment and GCS buckets reside. |
| data\_project\_name | GCP Project name/id in which Big query data tables reside. |
| bucket\_name | Google Cloud Storage (GCS) bucket name, which is used for storing DDL files generated by automation process. |
| pvo\_list\_path | GCS path where, list of PVOs stored as a .csv file to create entries to meta table. |
| base\_file\_path | GCS path where, information about each PVO generated by automation code will be generated for review by user, same file will be loaded in to the meta table FUSION\_TO\_BQ\_PVO\_DTL. |
| bicc\_file\_path | GCS path where Oracle provided file uploaded as csv file. This file contains list of PVOs/Columns/Fusion tables details, which is used to derive BigQuery table names and RAW layer column names. |
| ddl\_target\_path | GCS directory where DDL scripts / JSON files created by automation process will be stored. |
| STG\_SCHEMA | Bigquery dataset name where STAGE layer tables to be created. |
| RAW\_SCHEMA | Bigquery dataset name where RAW layer tables to be created. |
| DW\_SCHEMA | Bigquery dataset name where DW layer tables to be created. |
| HIST\_SCHEMA | Bigquery dataset name where HISTORY / Datalake layer tables to be created. |
| temp\_path | GCS Path, which has the DDLs / JSON files generated by the automation process in the latest run. (ddl\_target\_path propery will have DDLs/JSON files generated across all runs) |
| rest\_api\_url | BICC Rest Api URL, which is used to get the meta data details on PVOs. |
| rest\_api\_user | Service account user name, using which BICC Rest API can be accessed. |
| rest\_api\_secret | Key value of Google Cloud Secret manager, which hold the actual password of the BICC Rest API user. |
| meta\_dataset\_name | BigQuery dataset name, in which the meta tables FUSION\_TO\_BQ\_PVO\_DTL reside. |
| fusion\_schema | GCS Path, where JSON files generated by the automation process will be uploaded. These JSON files have the BQ table schema, which will be used at the time of data load process. |
| pvo\_meta\_path | GCS Path, where meta details about the PVOs stored as .csv files.  (manually downloaded from BICC) This property can be used if any access issues with BICC Rest API. |
|  |  |

1. Create a .csv file in the below format with the list of PVOs (Public View Objects) to be import into BigQuery.



1. Upload this file to the GCS path property **pvo\_list\_path.**
2. Ensure the Oracle provided file is available in **bicc\_file\_path.**
3. Now execute the **fusbq.base\_file\_template\_creation.py** file with below command:

**python -m fusbq.base\_file\_template\_creation generate**

* This operation reads the csv file uploaded in pvo\_list\_path.
* Reads the csv file available in bicc\_file\_path.
* Lookups the PVO name in the bicc file and filter the primary key columns, and gets the Fusion table name.
* Based on Fusion table name, derives names STG, RAW, HIST layers and BQ Procedures.
* Derived PRIMARY\_KEY & INCREMENTAL\_KEY for each PVO.
* Exports all these details as a .csv file named as BASE\_PVOS.csv to the path property **base\_file\_path.**

1. Now download the **BASE\_PVOS.csv** file from GCS and analyze manually if the details generated are correct. There could be chances of having multiple fusion table for a single PVOs. In such cases, we must decide on which table name should be used for that PVO.
2. Add a new Column **TABLE\_ID** as the first column in the file which should have the below format:

**module\_name+’\_’+series\_num**

Ex : FA\_01 / AR\_01.

Please ensure that, the table\_id created here is not already exist in the **PVO\_TO\_BQ\_PVO\_DTL.**

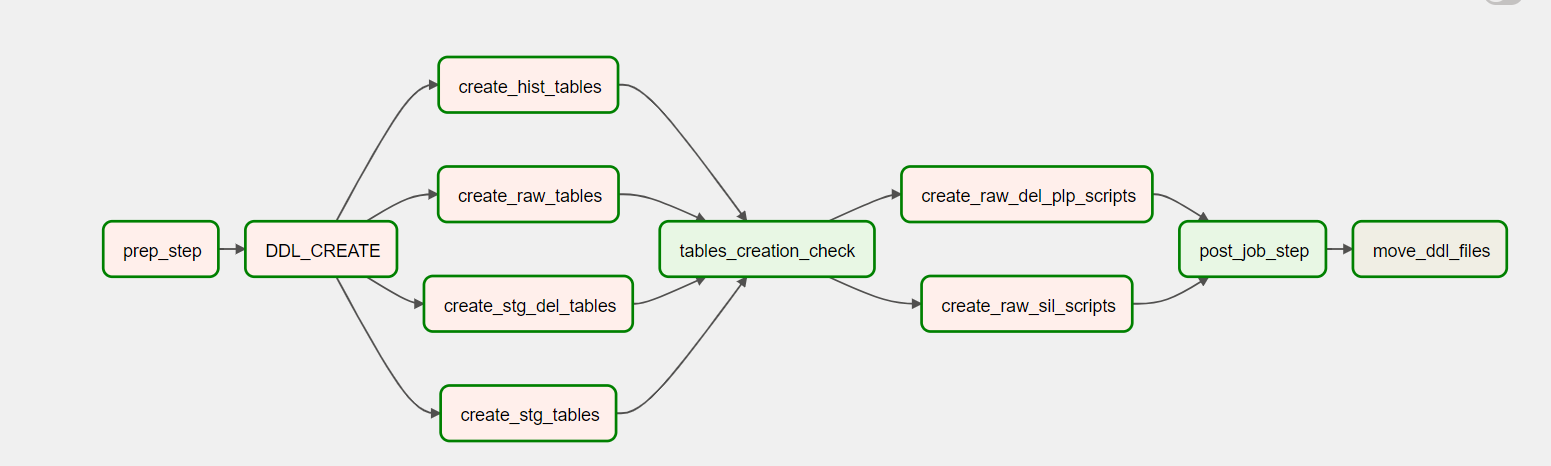
1. Once the file is validated / corrected, please upload the same to **base\_file\_path.** (Replace the existing file).
2. Now execute the **fusbq.base\_file\_template\_creation.py** file with below command:

**python -m fusbq.base\_file\_template\_creation insert**

* This will insert the data in the BASE\_PVOS.csv file to the BigQuery table **FUSION\_TO\_BQ\_PVO\_DTL.**

1. Please validate the records inserted in BQ.
2. **Executing the Automation DAG:**

Dev Url: <https://0b3a46718a044735a45196b29a2a3c01-dot-us-west1.composer.googleusercontent.com/dags/fusion_automation/graph>



Below operations done by the fusion\_automation DAG:

* **PREP\_STEP**: This task deletes any files available in the GCS folder **temp\_path**.
* **DDL\_CREATE:**

1. This task reads data in BQ\_CTL\_METADATA.FUSION\_TO\_BQ\_PVO\_DTL where **RUN\_DDL\_FLAG = TRUE**
2. Loops through each record, and creates DDL SQL Scripts for STAGE , RAW, STAGE\_DEL, HISTORY layers.
3. RAW & HISTORY table column names are derived based on below logic:

Lookup the PVO View attributes (column names) with the **bicc\_file\_path** to get the fusion Table level column name.

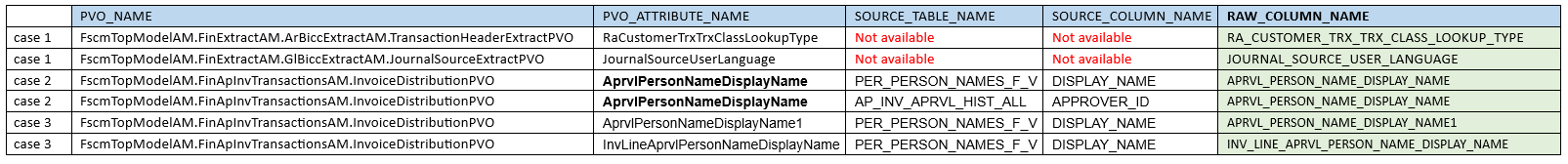
If the PVO view attribute not available in the above BICC excel

**or**

Multiple PVO attributes found for the same PVO in the file

**or**

Multiple PVO View attributes having same Fusion table column name, then we are going with below logic:



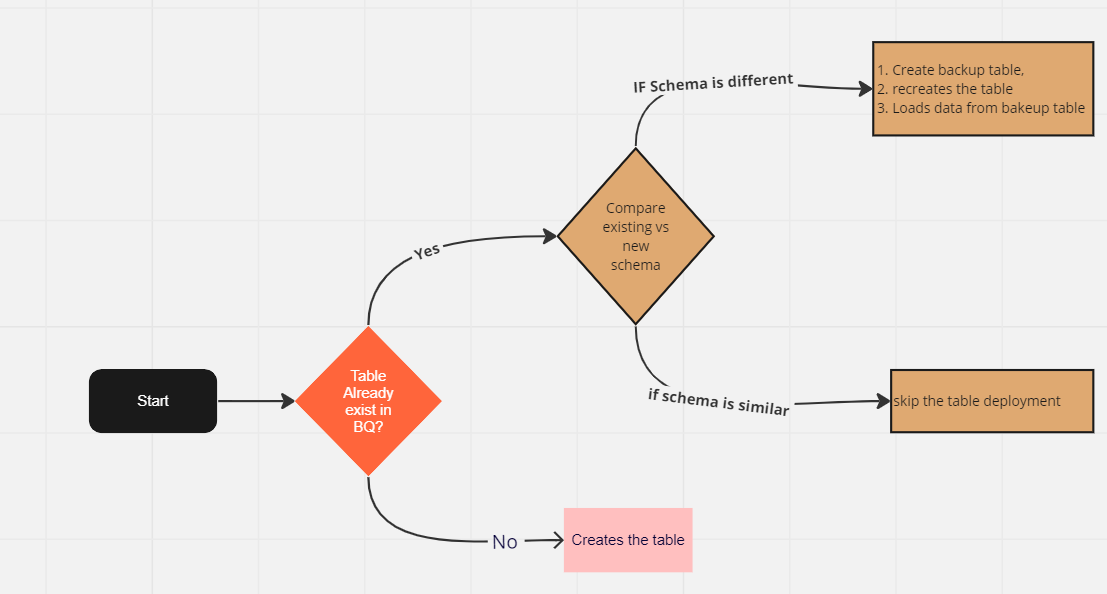
1. Creates JSON file, which represents BigQuery schema of STAGE, STAGE\_DEL, RAW & HISTORY tables.
2. All these scripts created are uploaded to the GCS folder **temp\_path**.

* **CREATE\_HIST\_TABLES:**

1. This step Reads JSON schema files in **temp\_path/JSON\_HIST** GCS folder.
2. Deploys the table as per below logic.

* **CREATE\_RAW\_TABLES:**

1. This step Reads JSON schema files in **temp\_path/JSON\_RAW** GCS folder.
2. Deploys the table as per below logic.



* **CREATE\_STG\_DEL\_TABLES:**

1. This step Reads DDL SQL files in **temp\_path/STG\_DEL** GCS folder.
2. Deploys the files to BigQuery.

* **CREATE\_STG\_TABLES:**

1. This step Reads DDL SQL files in **temp\_path/STG\_DEL** GCS folder.
2. Deploys the table as per above logic.

* **TABLE\_CREATION\_CHECK**

1. This task succeeds only if all upstream task are in success status.

* **POST\_JOB\_STEP**

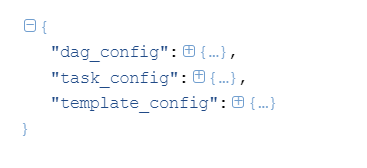
1. Updates the meta table BQ\_CTL\_METADATA.FUSION\_TO\_BQ\_PVO\_DTL table where RUN\_DDL\_FLAG=TRUE as FALSE.

* **MOVE\_DDL\_FILES**

1. Copy all files in the **temp\_path** GCS directory to **ddl\_target\_path.**
2. **Create Data Load DAG**
3. Take a copy of the attached sample JSON file:



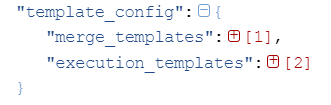
1. This JSON file structure is used to create DAGs as per Platform Teams’s Orchestration framework.
2. This JSON file contains 3 main Nodes:



1. The dag\_config node should be provided with DAG specific information, as below:



1. The template\_config node specifies the TASK operations supported by Platform Teams’ Orchestration Framework.



1. merge\_template supports, below task\_types:

* MERGE\_UPDATE
* MERGE\_APPEND
* MERGE\_TRUNCATE
* UPDATE\_SD\_ALL\_PK
* UPDATE\_SD\_DEL\_PK

Execution\_template supports below task\_types:

* DATAFLOW
* SQL
* AUTHENTICATED\_CLOUD\_FUNCTION
* TRIGGER\_DAG
* GCS\_TO\_BQ

1. In Fusion Ingestion, we are using Task Types highlighted in Yellow.

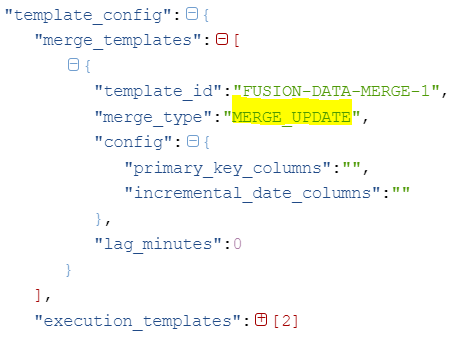
DATAFLOW -> To bring files from Fusion UCM to GCS

GCS\_TO\_BQ -> To load CSV files to BQ Stage layer.

MERGE\_UPDATE -> To merge data in STG layer to DL & RAW layer.

1. These task types configured in the file as below:

Merge Template:



Execution Template:



1. task\_config:

task\_config node consists list of all tasks and dependencies between them.

task\_config can be generated through automation code by executing below commands.

To generate task\_configs for data\_ingestion DAG run with the command **data\_tasks\_json.**

By using **--module** option we can specify, for tables under which module task-config must be generated. Below command will create task\_config for all tables belongs to the module **GL.**

**python -m fusbq.base\_file\_template\_creation data\_tasks\_json --module GL**

If task\_config has to be generated for specific tables, **--tables** option can be used as below.

**python -m fusbq.base\_file\_template\_creation data\_tasks\_json --tables “GL\_TEST01,GL\_TEST02”**

To generate task\_configs for soft\_delete DAG run with **del\_tasks\_json** command.

**python -m fusbq.base\_file\_template\_creation del\_tasks\_json --module GL**

**python -m fusbq.base\_file\_template\_creation del\_tasks\_json --tables “GL\_TEST01,GL\_TEST02”**

1. Once task\_config part is generated, it will be stored under **ddl\_target\_path** as TASK\_LISTS.json.
2. Utilize this file to create config json file and deploy it to composer.

------------------------------------------------------------------------------------------------------------------------------------------