

# Google BigQuery Artifact Extraction Prerequisites

# Contents

- 1. Introduction .....3
- 2. Artifact Extraction.....3
  - 2.1 DDL and Routine Scripts.....3
  - 2.2 Query Execution Logs .....6
    - 2.2.1 Using EXPORT DATA Command .....7
    - 2.2.2 Using BigQuery Query Editor .....8
  - 2.3 Other Scripts and Artifacts .....9
- 3. Getting Help.....10

## 1. Introduction

LeapLogic's Assessment profiles existing inventory, identifies complexity, performs dependency analysis, and provides recommendations for migration to other data platforms.

## 2. Artifact Extraction

LeapLogic requires certain artifacts to perform workload assessment. As a prerequisite, you need to have an admin user service account to fetch the required data. You can copy them from your GIT instance or export from the BigQuery repository where all the artifacts such as DDL scripts, stored procedures, functions, query execution logs, DML scripts, and other database objects are stored. LeapLogic needs all these artifacts in the form of .sql files.

### 2.1 DDL and Routine Scripts

Use the Python script (attached below) that utilizes the GCP BigQuery Python API to extract the DDL scripts. To use the BigQuery Python API, you need to install specific Python packages. Use the following command to install the required packages:

```
pip install google-cloud-bigquery
pip install pandas
pip install db-dtypes
```

Below is the Python script that extracts tables, views, routines, etc. from BigQuery. The script is also attached as a text file.

```
from google.cloud import bigquery
import os
import pandas as pd
import csv
import warnings
warnings.filterwarnings("ignore")

# Function to get DDL for a given table
def get_table_ddl(dataset_id, output_folder):
    try:
        output_folder = os.path.join(output_folder, dataset_id)
        os.makedirs(output_folder, exist_ok=True)
        query = f"""
        SELECT ddl
        FROM `{project_id}.{dataset_id}.INFORMATION_SCHEMA.TABLES`
        WHERE table_type = 'BASE TABLE'
        """
        ddls = client.query(query).to_dataframe()
        if not ddls.empty:
            ddls.to_csv(os.path.join(output_folder,
dataset_id+'_tables_ddl_export.sql'), index=False, mode='a', header=False,
quoting=csv.QUOTE_NONE, escapechar=' ')
        except Exception as e:
            raise
```

```

# Function to get DDL for a given view
def get_view_ddl(dataset_id, output_folder):
    try:
        output_folder = os.path.join(output_folder, dataset_id)
        os.makedirs(output_folder, exist_ok=True)
        query = f"""
        SELECT ddl
        FROM `{project_id}`.{dataset_id}.INFORMATION_SCHEMA.TABLES`
        WHERE table_type = 'VIEW'
        """
        ddls = client.query(query).to_dataframe()
        if not ddls.empty:
            ddls.to_csv(os.path.join(output_folder,
dataset_id+'_views_ddl_export.sql'), index=False, mode='a', header=False,
quoting=csv.QUOTE_NONE, escapechar=' ')
        except Exception as e:
            raise

def get_mv_ddl(dataset_id, output_folder):
    try:
        output_folder = os.path.join(output_folder, dataset_id)
        os.makedirs(output_folder, exist_ok=True)
        query = f"""
        SELECT ddl
        FROM `{project_id}`.{dataset_id}.INFORMATION_SCHEMA.TABLES`
        WHERE table_type = 'MATERIALIZED VIEW'
        """
        ddls = client.query(query).to_dataframe()
        if not ddls.empty:
            ddls.to_csv(os.path.join(output_folder,
dataset_id+'_materialized_views_ddl_export.sql'), index=False, mode='a',
header=False, quoting=csv.QUOTE_NONE, escapechar=' ')
        except Exception as e:
            raise

# Function to get DDL for a given procedure
def get_procedure_ddl(dataset_id, output_folder):
    try:
        output_folder = os.path.join(output_folder, dataset_id)
        os.makedirs(output_folder, exist_ok=True)
        query = f"""
        SELECT ddl
        FROM `{project_id}`.{dataset_id}.INFORMATION_SCHEMA.ROUTINES`
        WHERE routine_type = 'PROCEDURE'
        """
        ddls = client.query(query).to_dataframe()
        if not ddls.empty:
            ddls.to_csv(os.path.join(output_folder,
dataset_id+'_procedure_ddl_export.sql'), index=False, mode='a', header=False,
quoting=csv.QUOTE_NONE, escapechar=' ')
        except Exception as e:
            raise

# Function to get DDL for a given function
def get_function_ddl(dataset_id, output_folder):
    try:
        output_folder = os.path.join(output_folder, dataset_id)
        os.makedirs(output_folder, exist_ok=True)
        query = f"""

```

```

        SELECT ddl
        FROM `{project_id}`.{dataset_id}.INFORMATION_SCHEMA.ROUTINES`
        WHERE routine_type = 'FUNCTION'
        """

        ddls = client.query(query).to_dataframe()
        if not ddls.empty:
            ddls.to_csv(os.path.join(output_folder,
dataset_id+'_udf_ddl_export.sql'), index=False, mode='a', header=False,
quoting=csv.QUOTE_NONE, escapechar=' ')
        except Exception as e:
            raise

# Function to get DDL for a given function
def get_table_function_ddl(dataset_id, output_folder):
    try:
        output_folder = os.path.join(output_folder, dataset_id)
        os.makedirs(output_folder, exist_ok=True)
        query = f"""
        SELECT ddl
        FROM `{project_id}`.{dataset_id}.INFORMATION_SCHEMA.ROUTINES`
        WHERE routine_type = 'TABLE FUNCTION'
        """

        ddls = client.query(query).to_dataframe()
        if not ddls.empty:
            ddls.to_csv(os.path.join(output_folder,
dataset_id+'_table_function_ddl_export.sql'), index=False, mode='a',
header=False, quoting=csv.QUOTE_NONE, escapechar=' ')
        except Exception as e:
            raise

# Export DDLs
def export_ddl(datasets, output_folder):
    try:
        # Iterate over all datasets
        for dataset in datasets:
            dataset_id = dataset
            print(f"Processing dataset: {dataset_id}")

            # Export table DDLs
            get_table_ddl(dataset_id, output_folder)

            # Export view DDLs
            get_view_ddl(dataset_id, output_folder)

            # Export mv DDLs
            get_mv_ddl(dataset_id, output_folder)

            # Export procedure DDLs
            get_procedure_ddl(dataset_id, output_folder)

            # Export udf DDLs
            get_function_ddl(dataset_id, output_folder)

            # Export table function DDLs
            get_table_function_ddl(dataset_id, output_folder)

    except Exception as e:
        raise

```

```

if __name__ == "__main__":
    try:
        # Provide Google Cloud credentials: Service Account JSON file path
        os.environ["GOOGLE_APPLICATION_CREDENTIALS"] =
r"/path/to/service_account.json"

        # Specify your GCP project ID
        project_id = 'your-gcp-project-id'

        # Directory to store DDL files
        output_dir = r"/path/to/bigquery ddls"

        # Dataset names for DDL export. Keep empty list [] for all datasets
        datasets = ['dataset1', 'dataset2']

        # Initialize a BigQuery client
        client = bigquery.Client()

        # Get a list of all datasets in the project
        if not datasets:
            datasets = [dataset.dataset_id for dataset in
client.list_datasets(project=project_id)]

        # export ddls
        export_ddl(datasets, output_dir)

        print("Export completed.")

    except Exception as e:
        print('Exception while executing the ddl export script:' + str(e))

```

Replace the variable values highlighted in **Yellow** with the actual values.

### Python Script:



bigquery\_ddl\_export.txt

**Script output:** The Python script will create a separate folder for each dataset and export DDLs, including tables, views, routines, etc.

## 2.2 Query Execution Logs

For extracting the query execution logs there are two options available. The first approach uses the EXPORT DATA command to directly export query execution log to a GCS bucket. The second requires you to run the query directly in the BigQuery editor and export the output as a CSV file.

**Note:** We recommend using the EXPORT DATA command, especially when the file size is substantial.

## 2.2.1 Using EXPORT DATA Command

The EXPORT DATA command generates the required data file to a GCS bucket. Please see the prerequisites below.

To export the required query execution logs, use the following EXPORT DATA command. Please do remember to change the project-id, dataset and start date CAST(start\_time AS date) between '2024-01-01' and '2024-05-31' and GCS path where the files are expected to be generated. All the required inputs are marked in **Yellow** in the below script.

```
-- Create temp table to persist query log data
CREATE TABLE `project-id.dataset.sample_query_log`
OPTIONS (
  expiration_timestamp=TIMESTAMP_ADD(CURRENT_TIMESTAMP(), INTERVAL 1 DAY)
) AS
SELECT concat('IDWWM', '~',
ifnull(user_email, 'NA'), '~',
'app_u', '~',
'client_u', '~',
ifnull(start_time, '9999-01-01 00:00:00'), '~',
round(ifnull(AmpIO, 0)), '~',
ifnull(total_bytes_processed, 0), '~',
ifnull(creation_time, '9999-01-01 00:00:00'), '~',
ifnull(start_time, '9999-01-01 00:00:00'), '~',
ifnull(start_time, '9999-01-01 00:00:00'), '~',
0, '~',
CASE WHEN TB11.Query_ID IS NULL THEN jb.job_id ELSE TB11.Query_ID END, '~',
(max(ifnull(AmpIO, 0)) OVER (
  ORDER BY NULL)), '~',
(max(ifnull(total_bytes_processed, 0)) OVER (ORDER BY NULL)) , '~',
(SUM(ifnull(AmpIO, 0)) OVER (ORDER BY NULL)), '~',
(SUM(ifnull(total_bytes_processed, 0)) OVER (ORDER BY NULL)) , '~',
IFNULL(total_slot_ms, 0) / 1000, '~',
CASE WHEN LEFT(TB11.dataset_id, 7) = '_script' THEN 'Not Present' ELSE
IFNULL(TB11.dataset_id, 'Not Present') END, '~',
0, '~',
COALESCE (statement_type, ''), '~',
'', '~',
'', '~',
COALESCE (query, ''), '~',
COALESCE (job_type, 'NA'), '~',
COALESCE (TB11.tables_used, ''), '~',
ifnull(priority, ''), '~',
COALESCE (dml_statistics.inserted_row_count, 0) + COALESCE
(dml_statistics.deleted_row_count, 0) + COALESCE
(dml_statistics.updated_row_count, 0)) as query_log
FROM `region-us`.INFORMATION_SCHEMA.JOBS AS JB
LEFT JOIN (
SELECT DISTINCT JOB_ID, query_info.query_hashes.normalized_literals AS
query_id, r.dataset_id, string_agg(DISTINCT r.table_id ORDER BY r.table_id) AS
tables_used
FROM `region-us`.INFORMATION_SCHEMA.JOBS
LEFT JOIN UNNEST (referenced_tables) AS r
GROUP BY 1, 2, 3
) AS TB11
ON (JB.job_id = TB11.JOB_ID)
LEFT JOIN (SELECT job_id, round(sum(j.compute_ratio_avg) +
sum(j.compute_ratio_max) + sum(j.compute_ms_avg) + sum(j.compute_ms_max)
+ sum(read_ratio_avg) + sum(read_ms_avg) + sum(read_ratio_max)
```

```

+ sum(read_ms_max) + sum(write_ratio_avg) + sum(write_ms_avg) +
sum(write_ratio_max) + sum(write_ms_max)) AS AmpIO
FROM `region-us`.INFORMATION_SCHEMA.JOBS, unnest(job_stages) AS j
GROUP BY job_id) AS TB12
ON (JB.JOB_ID = TB12.JOB_ID)
WHERE error_result.message IS NULL AND state = 'DONE'
AND CAST(start_time AS date) > '2024-01-01' AND CAST(start_time AS date) <
'2024-05-31'
;

-- Export Data from temp table
EXPORT DATA OPTIONS(
  uri='gs://path/to/gcs/*.gzip',
  format='CSV',
  overwrite=true,
  header=false,
  compression=GZIP,
  field_delimiter='~') AS
select * from `project-id.dataset.sample_query_log`
;

```

## 2.2.2 Using BigQuery Query Editor

To export the required query execution logs, refer to the below script. Please change the project-id, dataset and start date CAST(start\_time AS date) between '2024-01-01' and '2024-05-31' as required.

We recommend exporting the query logs for a three months' timeframe. Next, execute the script given below using the BigQuery Query Editor UI and download the output after execution.

```

SELECT concat('IDWWM', '~',
ifnull(user_email, 'NA'), '~',
'app_u', '~',
'client_u', '~',
ifnull(start_time, '9999-01-01 00:00:00'), '~',
round(ifnull(AmpIO, 0)), '~',
ifnull(total_bytes_processed, 0), '~',
ifnull(creation_time, '9999-01-01 00:00:00'), '~',
ifnull(start_time, '9999-01-01 00:00:00'), '~',
ifnull(start_time, '9999-01-01 00:00:00'), '~',
0, '~',
CASE WHEN TB11.Query_ID IS NULL THEN jb.job_id ELSE TB11.Query_ID END, '~',
(max(ifnull(AmpIO, 0)) OVER(
  ORDER BY NULL)), '~',
(max(ifnull(total_bytes_processed, 0)) OVER (ORDER BY NULL )) , '~',
(SUM(ifnull(AmpIO, 0)) OVER (ORDER BY NULL)), '~',
(SUM(ifnull(total_bytes_processed, 0)) OVER (ORDER BY NULL )), '~',
IFNULL(total_slot_ms, 0) / 1000, '~',
CASE WHEN LEFT(TB11.dataset_id, 7) = '_script' THEN 'Not Present' ELSE
IFNULL(TB11.dataset_id, 'Not Present') END, '~',
0, '~',
COALESCE (statement_type, ''), '~',
'', '~',
'', '~',
COALESCE (query, ''), '~',
COALESCE (job_type, 'NA'), '~',
COALESCE (TB11.tables_used, ''), '~',
ifnull(priority, ''), '~',
COALESCE (dml_statistics.inserted_row_count, 0 )+ COALESCE
(dml_statistics.deleted_row_count, 0) + COALESCE
(dml_statistics.updated_row_count, 0)) as query_log

```



```

FROM `region-us`.INFORMATION_SCHEMA.JOBS AS JB
LEFT JOIN (
SELECT DISTINCT JOB_ID, query_info.query_hashes.normalized_literals AS
query_id, r.dataset_id, string_agg(DISTINCT r.table_id ORDER BY r.table_id) AS
tables_used
FROM `region-us`.INFORMATION_SCHEMA.JOBS
LEFT JOIN UNNEST (referenced_tables) AS r
GROUP BY 1, 2,3
) AS TB11
ON (JB.job_id = TB11.JOB_ID)
LEFT JOIN (SELECT job_id, round(sum(j.compute_ratio_avg) +
sum(j.compute_ratio_max) + sum(j.compute_ms_avg) + sum(j.compute_ms_max)
+ sum(read_ratio_avg) + sum(read_ms_avg) + sum(read_ratio_max)
+ sum(read_ms_max) + sum(write_ratio_avg) + sum(write_ms_avg) +
sum(write_ratio_max) + sum(write_ms_max)) AS AmpIO
FROM `region-us`.INFORMATION_SCHEMA.JOBS, unnest(job_stages) AS j
GROUP BY job_id) AS TB12
ON (JB.JOB_ID = TB12.JOB_ID)
WHERE error_result.message IS NULL AND state = 'DONE'
AND CAST(start_time AS date) > '2024-01-01' AND CAST(start_time AS date) <
'2024-05-31'
;

```

## 2.3 Other Scripts and Artifacts

Copy any other scripts such as DML scripts etc. from your GIT and share them with the LeapLogic team to produce more extensive insights.

### 3. Getting Help

Contact LeapLogic technical support at [info@leaplogic.io](mailto:info@leaplogic.io)