BigQuery Permission Mapper User Guide

Table of Contents

Useful information

Status: Staging
Last updated: 2021-12-03

Summary

BigQuery Permission Mapper aids in data warehouse migration from Teradata to BigQuery by allowing users to map existing Teradata user permissions to equivalent Google Cloud Identity and Access Management (Cloud IAM) permissions and roles. The existing Teradata databases commonly contain multiple user-defined roles that combine access permissions and capture common data access patterns. This tool is helpful in identifying the proper mapping between Teradata and BigQuery, removing the need for manual IAM setup by generating JSON files or Terraform scripts.

The following functionalities are provided in the Alpha feature:

- Export the necessary information from the source database to a spreadsheet (CSV) in a standardized format.
- Allow users to change the names of the BigQuery resources being mapped.
    - E.g. Teradata database name: `database1` → BigQuery dataset name: `dataset1`
- Map the existing Teradata users and roles to GCP users and custom and predefined roles.
- Create IAM commands as Terraform scripts or JSON files to create the custom groups, assign permissions, and bind to BigQuery resources.
- Highlight potential mapping issues like invalid permissions, invalid dataset names, and invalid user names.

Limitations

- The tool currently only supports Teradata permissions.
- The tool does not automatically extract permissions from Teradata.
- The tool does not currently support multi-org users.
- The tool does not map column or row level permissions.
- The tool will not migrate Teradata service accounts.
- The tool will not migrate your datasets or tables.
- Each script will only be for one GCP project, but the user may run the script multiple times for each project they want to migrate.

Getting Started with BigQuery Permission Mapper Tool

Overview

Every data warehouse (DWH) migration to BigQuery on Google Cloud includes the need to migrate DWH users and their permissions to the respective ones in GCP. This mapping process is a complex and time consuming process given the fact that admin and security teams would need to analyze, compare, and match hundreds to thousands of Teradata permissions to BigQuery IAM permissions. Due to the error prone nature of this task, this mapper tool will automate this step in the customer's migration journey by automatically map permissions when possible and report on cases when the manual mapping is required. Terraform scripts will be created for the end user to run to create the necessary custom groups, assign permissions, and bind to BigQuery resources.

The mapper tool is a directory of scripts which relies on the accuracy of input files provided by the end user. The scripts and reports generated will be available at the local directory for the user to review and use. The steps to generate the BigQuery IAM permission as JSON files or Terraform scripts are:

1. Unpack the BigQuery Permission Mapper tool and set up a directory for the tool to dump files to
2. Execute the provided SQL query to retrieve and export Teradata permissions from Teradata
3. Run the mapper tool and verify the resources and permissions that need to be migrated
4. Generate and review the JSON files or Terraform scripts, and a CSV report containing the mapping summary

Note: The data sources (datasets and tables) must already be in BigQuery before using this tool since this requires the data to exist to perform bindings.

Steps

1. Install and setup the mapper tool

1. Download or clone the tool from the github URL to your local execution directory. For example, clone it into `/td2bq-permissions-mapper` folder.
2. Use Python 3.8 or later and Install package dependencies with pip:

```
pip install -r requirements.txt
```

3. Create a directory in your local environment where the output files from this tool will be dumped. For example, we create `/Users/michua/Projects/td2bq_mapper_output` folder. You will need to provide the full path of this directory as input when running the tool in step 3 and 4. You also might be required to modify the output files in this directory as described in the following sections.

2. Retrieve and export Teradata permissions from Teradata

1. In your Teradata environment, run the SQL query found in the directory file `/td2bq-permissions-mapper/td2bq_mapper/data/get_teradata_perms.sql` to retrieve the Teradata permissions required as input to the mapper tool.

Modify the last two rows in the query if you only need to export a subset of Teradata users or roles:

```
AND    username  IN ()
AND    rolename IN ()
```

2. Export the query results as a CSV file. For example, we name the file "***td_permissions.csv***", and save it in the folder created in step 1.3. Keep column names as they are listed in **get_teradata_perms.sql**. The file path to this would be:

```
/Users/michua/Projects/td2bq_mapper_output/td_permissions.csv
```

3. Run the mapper tool: verify resources to migrate

Now that you have the required files, we can start using the mapper tool.

1. Go to the ***td2bq-permissions-mapper*** directory on the command line and enter this command:

```
./td2bq.py validate --td_acl_file TD_ACL_FILE --change_folder CHANGE_FOLDER [--log LOG]
[--overwrite]
```

Parameter	Details	Example
--td_acl_file	Path to the CSV file that contains Teradata permissions from step 2.2	`--td_acl_file "/Users/michua/Projects/td2bq_mapper_output/td_permissions.csv"`
--change_folder	Full path to the directory where the output files to be modified will be dumped (created in Step 1)	`--change_folder "/Users/michua/Projects/td2bq_mapper_output"`

--log	Full path to the output log file. Default log file `/td2bq-permissions-mapper/logs/td2bq.log`	`--log "/td2bq-permissions-mapper/logs/td2bq.log"`
--overwrite	Overwrite the existing output files in CHANGE_FOLDER	`--overwrite`

This will generate the four following CSV files in the local directory as specified by --change_folder:

File	Details	File columns
report_users.csv	Teradata users to migrate to GCP users	• Teradata_user • GCP_user
report_groups.csv	Teradata roles to migrate and place into GCP user groups	• Teradata_role • GCP_user_group • Rename_GCP_user_group
report_datasets.csv	Teradata datasets to migrate to BigQuery databases	• Teradata_database • BigQuery_dataset
report_objects.csv	Teradata objects to migrate to BigQuery objects	• Teradata_object • BigQuery_object

These files list GCP users, user groups, and BigQuery resources which the ***td_permissions.csv*** file contains. These files allow you to change the respective names for the GCP resources that the mapper is about to create in the next substeps. You are able to suggest names for GCP users, datasets, tables, user groups, and custom roles, and/or omit generating scripts for some resources.

Note that not all Teradata permissions can be matched to BigQuery IAM. During the validation step, the mapper reports such permission in ***invalid_acls.csv*** file. Four output files mentioned above contain only those Teradata roles, users, and objects that have direct mapping to BigQuery IAM for all their permissions.

Action needed by admin: review ***invalid_acls.csv*** and either change the IAM permissions directly before using the mapper or ignore n/a permissions.

Do NOT modify the file names or move these 4 files from this directory as it will be used in the next steps.

2. Review the 4 CSV files in --change_folder, make any necessary changes:
    a. Provide your input for the following columns which would be empty initially (see the detailed examples below):
        i. `GCP_user` in ***report_users.csv***
        ii. `BigQuery_dataset` in ***report_datasets.csv***
        iii. `BigQuery_object` in ***report_objects.csv***
        iv. `Rename_GCP_user_group` in ***report_groups.csv***
    b. Do not modify the other columns!
3. Save the files.

Actions you can take for each file

report_users.csv

Action on GCP_user column	Result	Example (below)
Leave `GCP_user` empty	`Teradata_user` will not be migrated	Row 1
Modify `GCP_user` to be the same as `Teradata_user`	`Teradata_user` will be migrated as `GCP_user`	Row 2
Modify `GCP_user` to be different than `Teradata_user`	`Teradata_user` will be migrated as `GCP_user`	Row 3
Modify `GCP_user` to be the same as another `Teradata_user`	Combines 2 separate `Teradata_user` and migrates as 1 `GCP_user`	Row 4, 5

Example

	Teradata_user	GCP_user
1	john@google.com	
2	kim@google.com	kim@google.com
3	adams@google.com	george@google.com
4	sal@google.com	sal@google.com
5	nick@google.com	sal@google.com

Things to be aware of

- If you choose to modify `GCP_user`, you must provide the full username and domain name, e.g. user@google.com.
- Do not modify `Teradata_user`

report_groups.csv - Match a Teradata role to a GCP user group

Action on Rename_GCP_user_group column	Result	Example (below)
Leave `Rename_GCP_user_group` empty	`Teradata_role` will not be migrated	Row 1
Modify `Rename_GCP_user_group` to be the same as `GCP_user_group`	`Teradata_role` will be added to `GCP_user_group`	Row 2
Modify `Rename_GCP_user_group` to be different than `GCP_user_group`	`Teradata_role` will be added to `Rename_GCP_user_group`	Row 3
Modify `Rename_GCP_user_group` to be different as another `Teradata_role` with the same `GCP_user_group`	Leaves 2 of the same `Teradata_role` as separate GCP user groups	Row 3, 4

Example

	Teradata_role	GCP_user_group	Rename_GCP_user_group
1	reader	group1@{%domain.com%}	
2	writer	group2@{%domain.com%}	writer@google.com
3	manager	group3@{%domain.com%}	reviewer@google.com
4	manager2	group3@{%domain.com%}	manager@google.com
5	manager3	group3@{%domain.com%}	manager@google.com

`GCP_user_group` will initially be populated with text in the format of `groupx@{%domain.com%}` with `x` being the unique group incremented starting from 1, and `{%domain.com%}` acts as a placeholder for your domain address.

Things to be aware of

- Two or more `GCP_user_group` entries might be the same if the `Teradata_role` permissions are the same. The tool will attempt to group them together, but you can choose to keep them as distinct groups by renaming them as two different groups (e.g. Row 3 and 4). Name them the same if you want to group them together as one group (e.g. Row 4 and 5).
- You cannot modify `Rename_GCP_user_group` entry to be the same as another `GCP_user_group` if `GCP_user_group` does not match (e.g. combining reader and writer roles into the same user group).
- Do not modify `GCP_user_group`

report_datasets.csv

Action on BigQuery_dataset column	Result	Example (below)
Leave `BigQuery_dataset` empty	`Teradata_database` will not be migrated	Row 1
Modify `BigQuery_dataset` to be the same as `Teradata_database`	`Teradata_database` will be migrated as `BigQuery_dataset`	Row 2
Modify `BigQuery_dataset` to be different than `Teradata_database`	`Teradata_database` will be migrated as `BigQuery_dataset`	Row 3

Example

	Teradata_database	BigQuery_dataset
1	database1	
2	database2	database2
3	database3	database4

Things to be aware of

- Modifying `BigQuery_dataset` is just a name change of the `Teradata_database`, the contents will not change.
- Naming conventions
  - Dataset names are case-sensitive: `mydataset` and `MyDataset` can coexist in the same project.
  - Dataset names cannot contain spaces or special characters such as -, &, @, or %.
- Do not name two different datasets the same.

report_objects.csv

Action on BigQuery_object column	Result	Example (below)
Leave `BigQuery_object` empty	`Teradata_object` will not be migrated	Row 1
Modify `BigQuery_object` to be the same as `Teradata_object`	`Teradata_object` will be migrated as `BigQuery_object`	Row 2
Modify `BigQuery_object` to be different than `Teradata_object`	`Teradata_object` will be migrated as `BigQuery_object`	Row 3

Example

	Teradata_object	BigQuery_object
1	database1.table1	
2	database1.table2	table2
3	database2.table2	table4

Things to be aware of

- Modifying `BigQuery_object` is just a name change of the `Teradata_object`, the contents will not change.
- You do not need to provide the dataset name of the object (e.g. `dataset.table1`). Dataset names are pulled from ***report_datasets.csv***.
- Do not name two different objects the same.

4. Run the mapper tool

BQ Permission Mapper outputs BigQuery IAM as a CSV report and also generates Terraform scripts or JSON files for gcloud CLI. Decide which output you want and run the mapper as described below.

4.1. Generate Terraform scripts

After ensuring that everything that needs to be migrated is included, run the following command to generate Terraform scripts:

```
./td2bq.py generate_tf --td_acl_file TD_ACL_FILE --change_folder CHANGE_FOLDER
--project_ID PROJECT_ID --customer_ID CUSTOMER_ID [--log LOG] [--overwrite]
```

Parameter	Details	Example
--td_acl_file	Path to the CSV file that contains Teradata permissions from Step 2.2	`--td_acl_file "/Users/michua/Projects/td2bq_mapper_output/td_permissions.csv"`
--change_folder	Full path to the directory where the modified files are (report_*.csv from Step 3) and where generated	`--change_folder "/Users/michua/Projects/td2bq_mapper_output"`

	Terraform files will be dumped	
--project_ID	The GCP project ID. It is used in generated Terraform files.	`--project_ID "my-sample-project-191923"`
--customer_ID	The customer ID (find it here). It is used in generated Terraform files.	`--customer_ID "C02dfkjhd"`
--log	Full path to the output log file. Default log file `/td2bq-permissions-mapper/logs/td2bq.log`	`--log "/td2bq-permissions-mapper/logs/td2bq.log"`
--overwrite	Force overwriting the existing files in CHANGE_FOLDER	`--overwrite`

If there are mappings errors with any of the input files in --change_folder when this command is run, the mapper will create an ***invalid_reports.csv*** file in --change_folder, populate it with the errors encountered, and stop the process. You must review the invalid file and modify those before retrying.

Action needed by admin: review ***invalid_reports.csv*** and fix reported errors in output files (report_*.csv from Step 3).

Additionally, if the mapper encounters Teradata permissions that cannot be mapped to BigQuery IAM, those permissions will be excluded from further processing and reported into ***unmapped_acls.csv***. This report is similar to ***invalid_acls.csv*** generated during the validation step as described in Step 2. It serves as guardrails if the user skips the validation step before generating JSON files.

Action needed by admin: review ***unmapped_acls.csv*** and either change the IAM permissions directly before using the mapper or ignore n/a permissions.

4.2. Generate JSON files for gcloud CLI

You can use gcloud CLI to create and manage Google Cloud resources. BQ Permission Mapper generates JSON files capturing BigQuery objects and IAM permissions which you can use as the input for gcloud CLI.

After ensuring that everything that needs to be migrated is included, run the following command to generate JSON files:

```
./td2bq.py generate_json --td_acl_file TD_ACL_FILE --change_folder CHANGE_FOLDER
--project_ID PROJECT_ID [--log LOG] [--overwrite]
```

Parameter	Details	Example
--td_acl_file	Path to the CSV file that contains Teradata permissions from Step 2.2	`--td_acl_file "/Users/michua/Projects/td2bq_mapper_output/td_permissions.csv"`

--change_folder	Full path to the directory where the modified files are (report_*.csv from Step 3) and where generated JSON files will be dumped	`--change_folder "/Users/michua/Projects/td2bq_mapper_output"`
--project_ID	The GCP project ID. It is used in generated JSON files.	`--project_ID "my-sample-project-191923"`
--log	Full path to the output log file. Default log file `/td2bq-permissions-mapper/logs/td2bq.log`	`--log "/td2bq-permissions-mapper/logs/td2bq.log"`
--overwrite	Force overwriting the existing files in CHANGE_FOLDER	`--overwrite`

If there are mappings errors with any of the input files in --change_folder when this command is run, the mapper will create an ***invalid_reports.csv*** file in --change_folder, populate it with the errors encountered, and stop the process. You must review the invalid file and modify those before retrying.

Action needed by admin: review ***invalid_reports.csv*** and fix reported errors in output files (report_*.csv from Step 3).

Additionally, if the mapper encounters Teradata permissions that cannot be mapped to BigQuery IAM, those permissions will be excluded from further processing and reported into ***unmapped_acls.csv***. This report is similar to ***invalid_acls.csv*** generated during the validation step as described in Step 2. It serves as guardrails if the user skips the validation step before generating JSON files.

Action needed by admin: review ***unmapped_acls.csv*** and either change the IAM permissions directly before using the mapper or ignore n/a permissions.

Output Information

CSV files to review

After each run BQ Permission Mapper generates a CSV report containing TD2BQ permission map. Review the following files generated in the --change_folder directory:

1. ***td2bq_mapping.csv***: CSV file containing the TD2BQ mapping.
    a. Note: modifying this file will not impact the Terraform or JSON scripts. This file is generated only as a reference. If something is incorrectly mapped, please check the input files that you provided to the mapper and error reports if any was produced.
2. ***invalid_reports.csv***: CSV file containing invalid records from user provided input files described in Step 3.
3. ***unmapped_acls.csv***: CSV file containing invalid Teradata roles and/or users due to permissions grouping that can't be mapped to GCP.

Terraform scripts generated

When you use *generate_tf* option and the tool finishes running, the ***terraform_generated*** folder located in the tool directory will contain:

1. ***iam_custom_roles.tf.json***: BigQuery custom roles with its respective IAM permissions
2. ***iam_invalid_custom_roles.tf.json***: BigQuery custom roles that had ARCs which did not map to a respective IAM permission
    a. Action needed by admin: review the roles in this file and either change the IAM permissions directly before using the file alongside ***iam_custom_roles.tf.json*** or ignore it and only use ***iam_custom_roles.tf.json***
3. ***iam_datasets_roles_binding.tf.json***: BigQuery resources (databases or objects) with its respective BigQuery roles and users
    a. Action needed by admin: review the roles assigned for each dataset in the file since they may contain invalid roles. If the admin decides to not implement invalid roles (above), those roles need to be deleted from the respective dataset before usage.
4. ***Iam_email_list_schema_editor_org_com.json***: users emails
5. ***iam_groups.tf.json***: GCP user groups to create

These scripts are generated in this order:

1. IAM groups
2. IAM custom roles
3. IAM custom roles to users/group and BQ datasets/tables bindings

The file paths for these scripts are

```
/[local path]/td2bq-permissions-mapper/terraform_generated/[script name]
```

You are able to review and then run these scripts to create BigQuery datasets, roles, grant permissions, and bind resources.

Open the appropriate project in the GCP console. Activate the Cloud Shell from the upper right hand corner and open the Cloud Shell Editor. Create a new folder and import the above JSON files into the folder.

Run the following commands to apply the terraform to the environment.

```
terraform init
terraform apply
```

JSON files generated

When you use *generate_json* and the tool finishes running, the ***json_generated*** folder located in --change_folder directory will contain:

1. ***custom_roles:*** folder containing BigQuery custom roles each in a separate file and its respective IAM permissions. For each JSON file in *json_generated/custom_roles,* run the *gcloud iam roles create* command to create a new custom role. For example, create a custom role from generated *td2bq_mapper_bqcustom1.json*:

    ```
    gcloud iam roles create td2bq_mapper_bqcustom1 \
    --project="my-sample-project-191923" \
    --file=./json_generated/custom_roles/td2bq_mapper_bqcustom1.json
    ```

    Keep custom role names the same as the file names in *json_generated/custom_roles*.

2. ***datasets***: folder containing BigQuery datasets with its respective BigQuery roles and users. Presently gcloud CLI supports a three-step approach to assign permissions to datasets in BigQuery. For each JSON file in *json_generated/datasets*:
    - retrieve the current information about a dataset
    - add new permissions

- update the dataset information

For example, retrieve the current information about *db2* dataset:

```
bq show --format=prettyjson my-sample-project-191923:db2 > ./db2_current.json
```

Then edit *db2_current.json* and add to the "*access*" section permissions from *json_generated/datasets/db2.json* which has been generated by the mapper. Update the BigQuery dataset using modified *db2_current.json* as following:

```
bq update --source ./db2_current.json my-sample-project-191923:db2
```

3. ***objects***: folder containing BigQuery objects with its respective BigQuery roles and users
   You can use bq CLI and a tree-step process to assign permissions to tables and views in BigQuery using JSON files. For each JSON file in *json_generated/tables*:
   - retrieve the current table or view policies
   - add new policies
   - update table of view policies

   For example, retrieve current policies for table db2.t2:

```
bq get-iam-policy --format=prettyjson \
my-sample-project-191923:db2.t2 > ./db2.t2_current.json
```

   Then edit *db2.t2_current.json* and add to the "*bindings*" section policies from *json_generated/tables/db2.t2.json* which has been generated by the mapper. Update BigQuery policies using modified *db2.t2_current.json* as following:

```
bq set-iam-policy my-sample-project-191923:db2.t2 ./db2.t2_current.json
```

Presently gcloud CLI supports identity groups and memberships. However, it doesn't accept JSON files as input. You can check ***td2bq_mapping.csv*** to see the user groups and memberships and use gcloud CLI, or Google Workspace Admin web page, or SDK to manage those.

These scripts are generated in this order:

1. BigQuery custom roles
2. BigQuery datasets and tables bindings

You are able to review and then run these files to create BigQuery datasets, roles, grant permissions, and bind resources in Cloud Shell.

Pricing

There will be no charge for the BigQuery Permission Mapper tool.

Frequently Asked Questions

1. Why are the Terraform scripts empty after running the tool?
   a. If the script files are empty, it means that nothing was migrated due to leaving any/all of the four input files' rename column from step 3 blank. Please review and ensure that everything is filled out.
2. Do I have to clear the invalid_acls.csv or unmapped_acls.csv files after fixing the errors presented in those?
   a. No, you can directly rerun the commands after fixing the errors presented. The invalid_acls.csv and unmapped_acls.csv file contents will not impact the Terraform or JSON scripts generated.
3. How do I migrate column and row level permissions from Teradata to BigQuery?

a. The column and row level permissions mapping functions are not currently available in the tool due to the complexity of mapping between Teradata privilege data to Cloud Data Catalog taxonomy and BigQuery table schema definition. This feature is being analyzed and planned for a future version.
b. To extract the column and row level permissions from Teradata, follow the following steps.
    i. 16.20 - Column-Level Privileges - Teradata Database
    ii. 16.10 - Teradata Row Level Security Privileges - Teradata Database
c. To map the column and row level permissions to BigQuery, follow the following steps.
    i. Create column level permission in BigQuery
    ii. Create row level permission in BigQuery

4. Why did running my Terraform script generate an error?
    a. Make sure that you did not modify the Teradata permissions CSV input file. A common issue is that Teradata will not assign a role to a resource if it's not in the resource hierarchy. For example, assigning a role with dataset level permissions to a table.

Potential error messages when running terraform apply

```
Error when reading or editing CloudIdentityGroup "groups/[id]": Get
"https://cloudidentity.googleapis.com/v1/groups/[id]?alt=json": dial tcp [tcp]:443:
connect: cannot assign requested address
```

➔ This is due to the connection being lost on Cloud Shell. Reconnect and run `terraform apply` again.