BigQuery

Not Like the Others

By Daran J. Johnson

Introduction

With BigQuery, you can store data securely with high performance and nearly unlimited growth. It is perfect for data analysis.

As an analyst, you work with data. The data may be stored somewhere…acceptable. Sometimes it's not. Sometimes it might just be good enough to use a text file or excel document. But a lot of times that's very impractical for a whole host of reasons.

If the data is large, it will take a long time to load and query. If you want to share reproducible results, it may be better if the data is remotely located, somewhere that is accessible through code. Also, the data should only be accessible to those that should have access to it (security!).

Data Storage Options

This is generally solved through using a database, like MySQL, SQL Server or PostgreSQL. However, that requires that you have a server available and that there is a secure (read: encrypted) connection to it. That is not always practical. I have often used a remote MySQL or SQL Server as a solution (sometimes local, but…reproducibility). It has not always been as secure as I would like and scalability could sometimes (read: often) be an issue. Additionally, for data to be accessed efficiently, tables need to be optimized through indexes and server configuration.

If you are looking for data storage that is secure, fast, scalable and self-optimizing, BigQuery might just be the solution you need. BigQuery is part of the Google Cloud Platform (GCP). Once loaded, you can access data stored in BigQuery securely. In fact, there is no other option. It is hugely scalable as well — practically limitless. It works a lot like a database, but most of the optimization is taken care of by Google. You only need to worry about how you want to structure and query the data.

BigQuery Overview

BigQuery is a serverless data warehouse that stores and serves data based on SQL queries. It has continued to evolve over the years. BigQuery separates out the compute and storage tasks. This allows for it to query data that is outside of BigQuery storage,

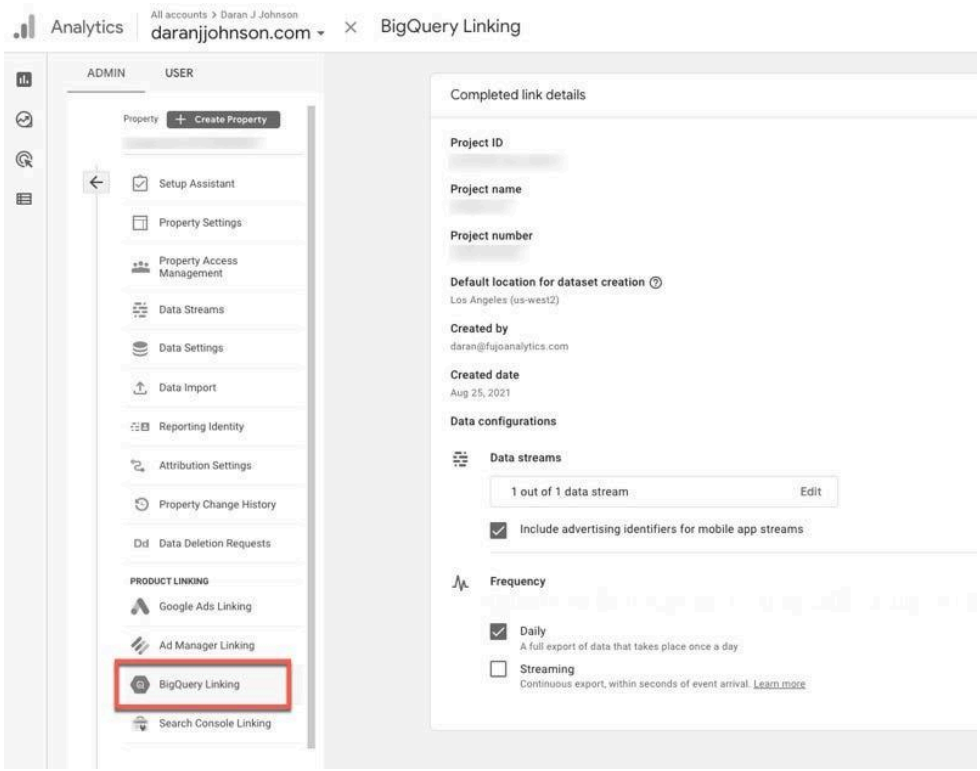
such as a csv file stored in Google Cloud Storage (although the queries would run faster if the data is loaded into BigQuery).

Yes, you can actually query data that is not even in BigQuery — true separation of physical storage and logic. The data still needs to reside in GCP and be queryable.

BigQuery Dataflow

Data can be streamed or batched into BigQuery. Google Dataflow can be used to create an Apache Bean Pipeline for ETL and Cloud Pub/Sub can be used to ingest streaming data (both can be found under Big Data in the GCP left sidebar).

You can also use Cloud Shell and the bq command-line tool (bq --location=us mk --reservation --project_id=project reservation_name). Full disclosure - the last two sentences were so I could sound like a Data Engineer. Data can be manually loaded through the GCP interface, as well. Data from Google Analytics 4 can be pushed directly into BigQuery from the GA admin screen (as can be seen below - it is so, so easy).



BigQuery can then be connected directly to reporting platforms like Looker, Data Studio, and Tableau because it is a fully-featured SQL engine. It can also do machine learning through the use of BigQuery ML, AutoML tables and the Google AI platform. Yeah, it's pretty awesome.

BigQuery Infrastructure

BigQuery separates compute and storage tasks. For compute, BigQuery sits on top of Borg (a cluster management system, but sounds like an alien species coming to assimilate all of human-kind) and Dremel (a query processing engine that sounds like…I think of a drum — drumming the data in/out of storage). Dremel uses a distributed computing process to speed up queries.

Compute resources are denominated in terms of slots (roughly half a CPU), which are added and removed dynamically. They can be reserved using the BigQuery Reservation API. If you don't reserve them, your queries could be slower. However, I think that by the time you need that kind of processing power, you'll already know what you're doing. So, never mind.

BigQuery Storage

Data in BigQuery is stored in Colossus (Google's distributed file system) in Capacitor storage (just think capacity and I think you'll be fine remembering this one). Capacitor is a columnar storage format that allows BigQuery to access compressed data, without uncompressing it, on the fly.

Storing the data in columnar format minimizes traffic, compared to row storage, by only scanning those columns passed by the query. It can also achieve a higher compression ratio since all the data in a given column will be the same data type. Capacitor also stores BigQuery data as tables, not files. This allows for key data features, such as ACID (Atomicity, Consistency, Isolation, and Durability) properties.

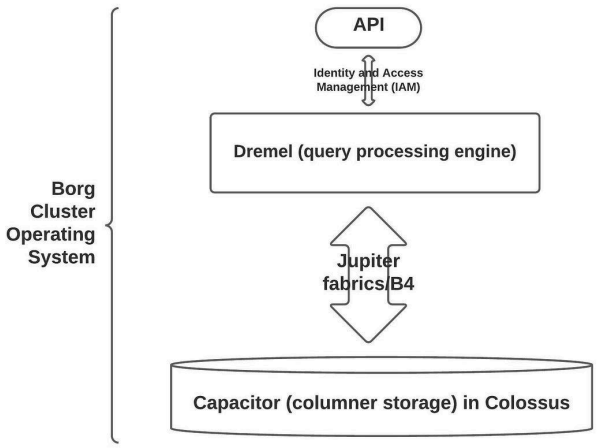
Vectors, Attributes, Dataframes, and Columns

I think this is a good time to also reflect that columns are attributes (in databases) or vectors (in R), and a dataframe is just a collection of vectors. A well-defined vector is one that only allows values within a predetermined set. For example, the quarters of a year — there are only the numbers 1, 2, 3, or 4.

The vector should then only allow one of those numbers and nothing else. If you create a table that allows a column (vector) such as this to have anything else, you will eventually have issues with your data.

BigQuery Network

To manage all the data throughput, Google uses a network system called Jupiter fabrics (great name, since Jupiter is the largest planet in the solar system). Jupiter fabrics can deliver more than 1 petabyte/sec of total bisection bandwidth. It optimizes the use of resources by spreading work over a large network with many small switches. This is connected through the B4 networking infrastructure, which allows dynamic allocation of bandwidth.



GCP & BigQuery have a number of additional features/applications/platforms that make working with BigQuery easier, flexible and extensible. Here are a few (most):

- Cloud Logging & Cloud Monitoring: Review who and when data is access and jobs run.
- Cloud Dataproc: Read/Transform/Write data from Apache Spark.
- Federated Queries: Query data held in Google Cloud Storage, Cloud SQL, BigTable, Spanner, or Google Drive.
- Google Cloud Data Loss Prevention API: Manage sensitive data and mask PII.
- AutoML: Create machine learning models directly in BigQuery with SQL.
- Cloud Data Catalog: Lists data used throughout the account.
- Cloud Pub/Sub: Ingest streaming data.
- Cloud Dataflow: ETL & streaming queries.
- Cloud AI Platform: Allows streamlined ML workflows, training, prediction, and version management of advanced ML models.
- Cloud Scheduler & Cloud Functions: Schedule BigQuery queries to be run.
- Cloud Data Fusion: Drag/drop data integration tools.
- Cloud Composer: Allows orchestration of BigQuery jobs along with tasks that need to be performed in Cloud Dataflow.
- IAM & Google Virtual Private Cloud (VPC): Security.

Getting Started Using BigQuery

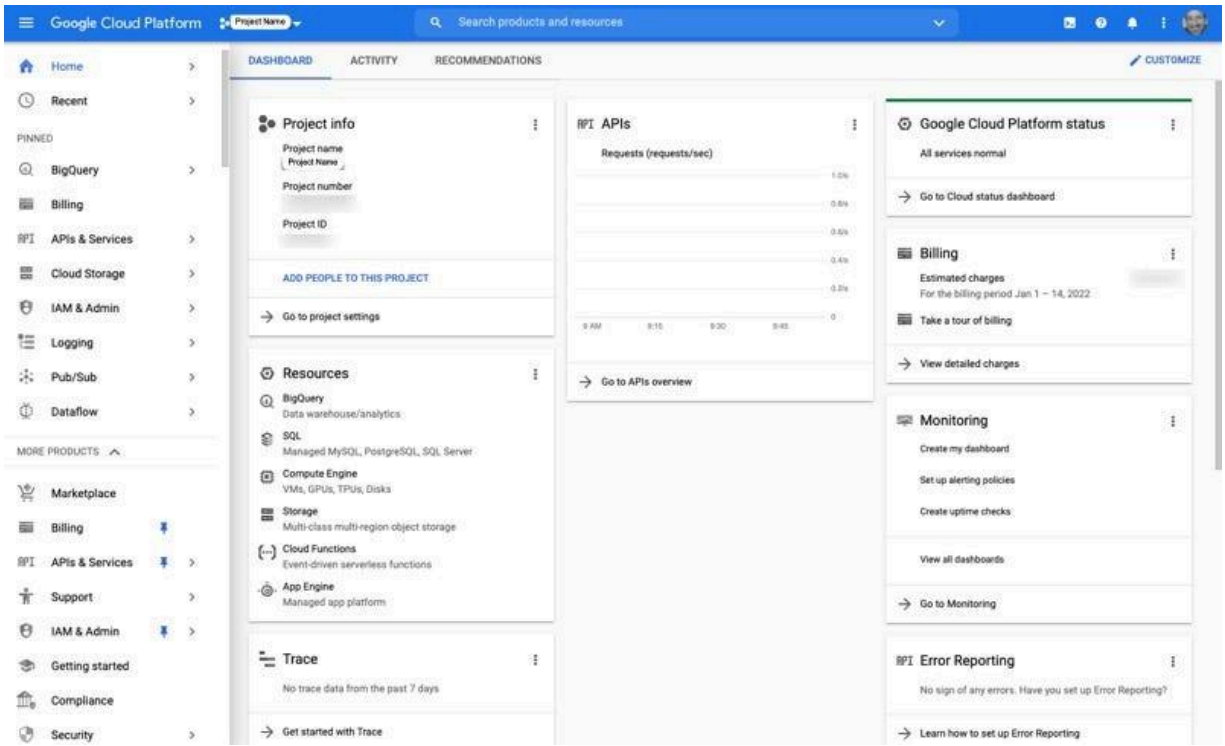
If you don't have a GCP account, you'll need to create one. Just go to https://cloud.google.com/ and sign up. Enter your personal/company information as well as payment information. You can go to https://cloud.google.com/bigquery/pricing for pricing information. It is very reasonable and I like that it scales — the more data/queries the higher the cost. There are also tools to help you manage costs.

Once you have set up your GCP account and created a project, you can go to your GCP Home for that project. A single project might be fine, but it might make sense to create multiple projects. For example, projects could be split by department, clients, or business projects.

GCP Home

The Home dashboard is made up of cards, like Project info, Resources, Billing, etc. You can customize the dashboard as well. There are also Activity & Recommendation tabs at the top. On the left-hand side there is a navigation menu with the following sections: Compute, Serverless, Storage, Databases, Application Integration, Networking, Operations, CI/CD, Tools, Big Data, Artificial Intelligence, Other Google Products, and Partner Solutions.

You can also pin resources, which displays them at the top of the navigation. In this image, I have pinned BigQuery, Billing, APIs & Services, Cloud Storage, IAM & Admin, Logging, Pub/Sub & Dataflow, so they are at the top of the navigation.



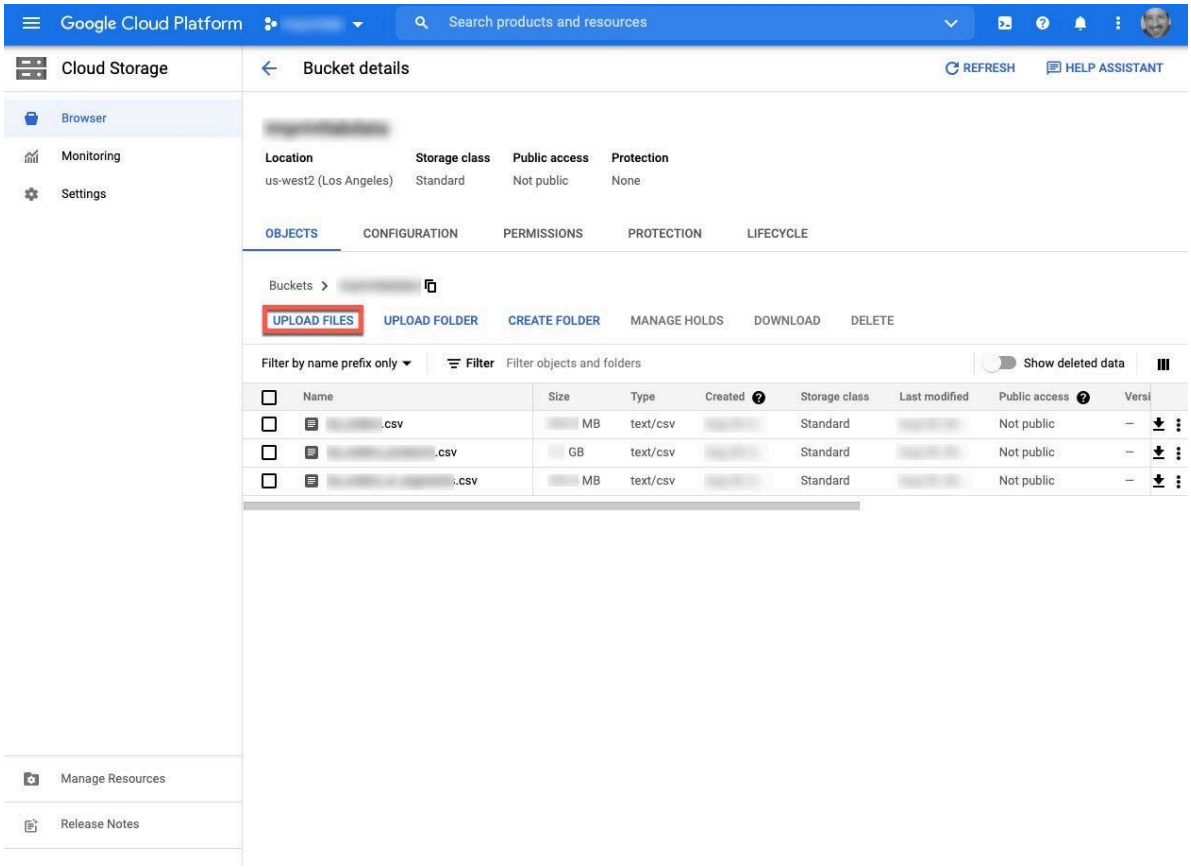
To work with data in BigQuery, you'll first need to load the data.

Loading Data

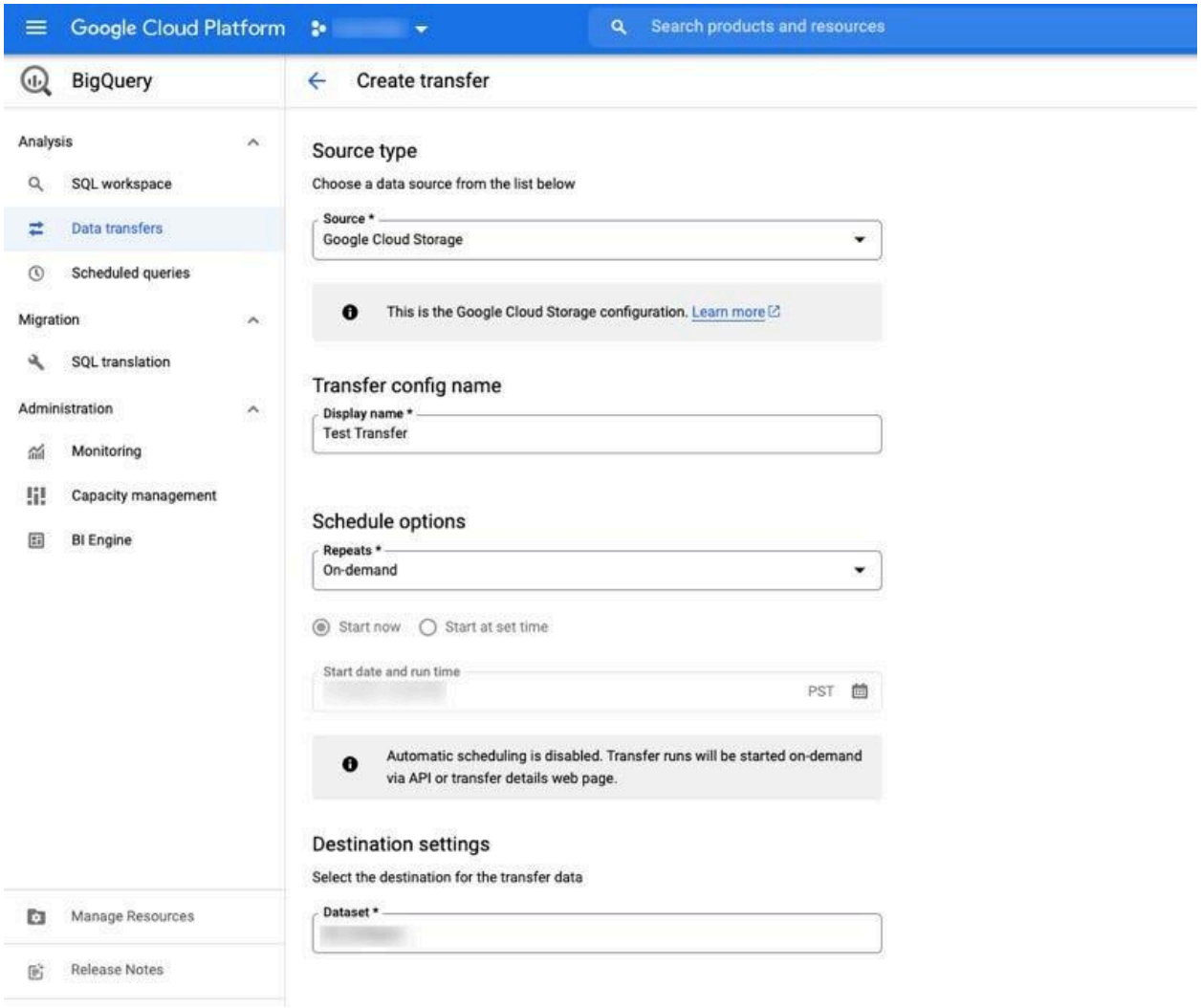
There are a number of ways to load data into BigQuery. If you are using Google Analytics 4, you can push data directly from GA4 to BigQuery (as shown above). With a text file, you can load it into BigQuery through Google Cloud Storage. To do that, navigate to Cloud Storage and create a Bucket. Click on the Bucket and you will see the following options:

- UPLOAD FILES
- UPLOAD FOLDER
- CREATE FOLDER

Once you have uploaded a few files, your screen should look like the following.

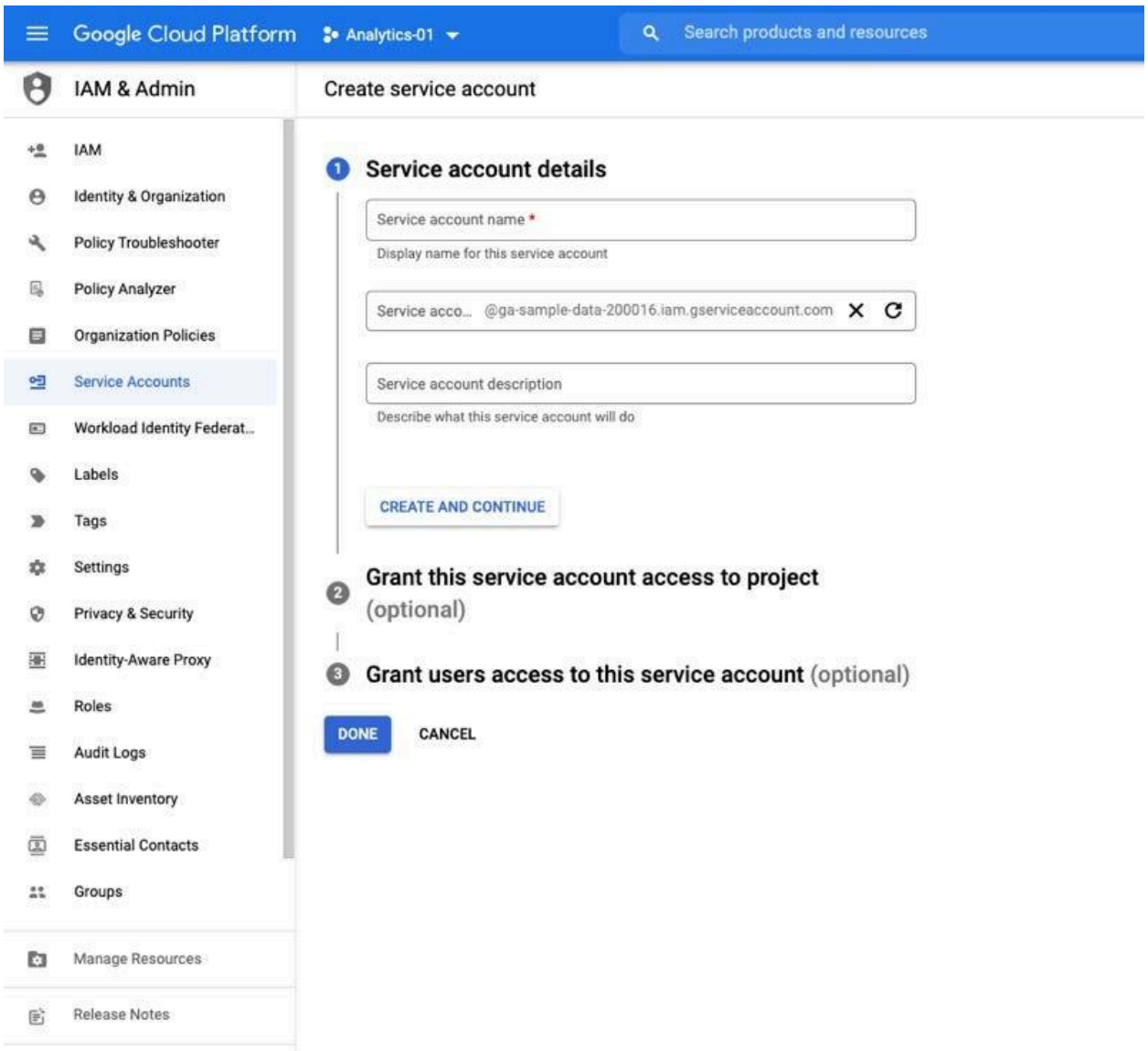


To then get the data into BigQuery, navigate to BigQuery and select Data Transfers. Create a new data transfer with the source as Google Cloud Storage. Name your transfer and use On-demand for Repeats, if this is a one-time only upload. You'll need to select the destination for the dataset. There are a few other options you'll need to select as well. It may take a few times to get it just right.

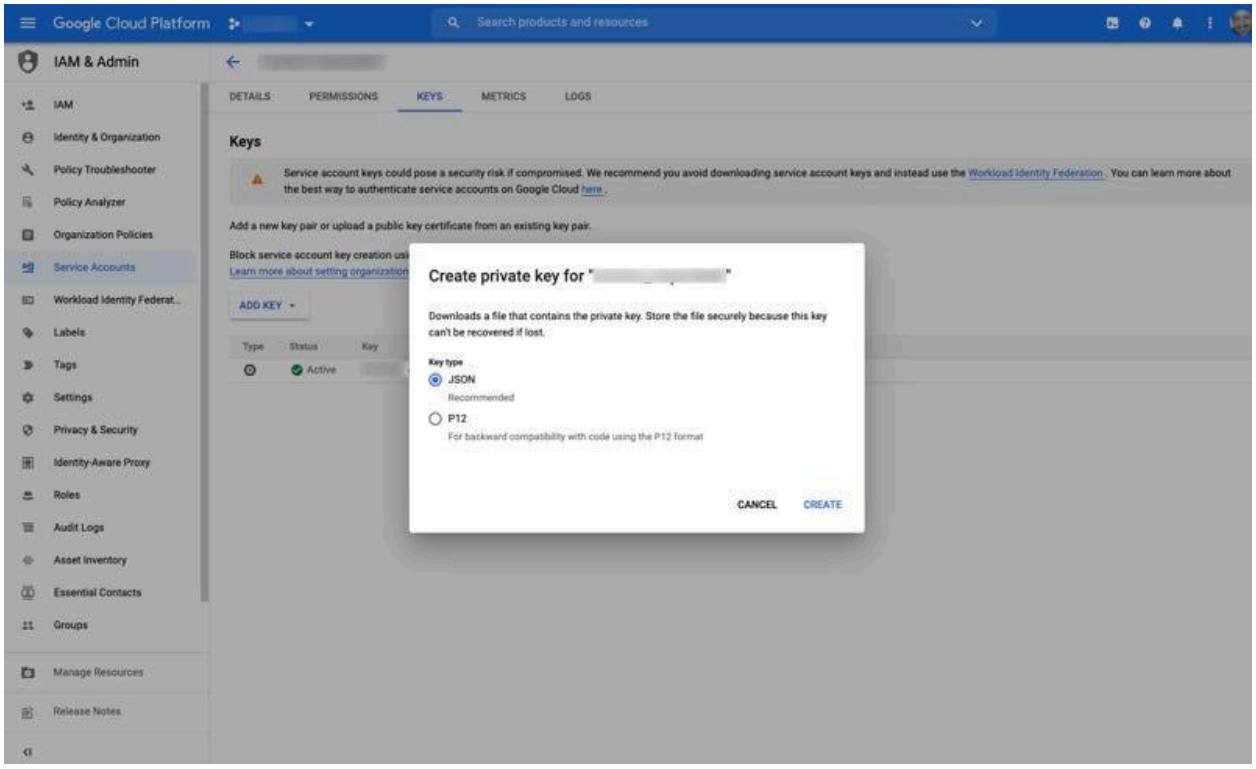


Connecting to R

One way to access BigQuery data is to create a service account and use a JSON key. Click on IAM & Admin, then navigate to Service Accounts. Click on CREATE SERVICE ACCOUNT. Your screen should look something like the screen below.



Click on the service account, then click on KEYS, then click on ADD KEY. You should see something like the screen below.



Select JSON for the type of key. It will automatically download to your local machine. This is a sensitive file, so keep it safe.

In R, reference the JSON file when connecting to BigQuery. In the code below, I have created a function, sqlQuery(), that will be used to pass SQL queries through to BigQuery and retrieve the resulting dataset. The JSON file mentioned above is named your_project_dataset.json below.

Python

```
# Install & load the package to connect to BQ.

install.packages('bigrquery')

library(bigrquery)

# Declare the path to the BQ JSON authorization file.

bigrquery::bq_auth(path = "your_project_dataset.json")
```

```
# Create a function to open a connection to BQ and send the query
# then close the connection on exit.
sqlQuery <- function (query) {
 bq_conn <- DBI::dbConnect(bigrquery::bigquery(),
                           project = "project",
                           dataset = "dataset")
 # Close connection after function call exits.
 on.exit(DBI::dbDisconnect(bq_conn))
 # Send query to BQ and save result set as a dataframe.
 result <- as.data.frame(DBI::dbGetQuery(bq_conn, query))
 # Return the dataframe to the caller.
 return(result)
}
```

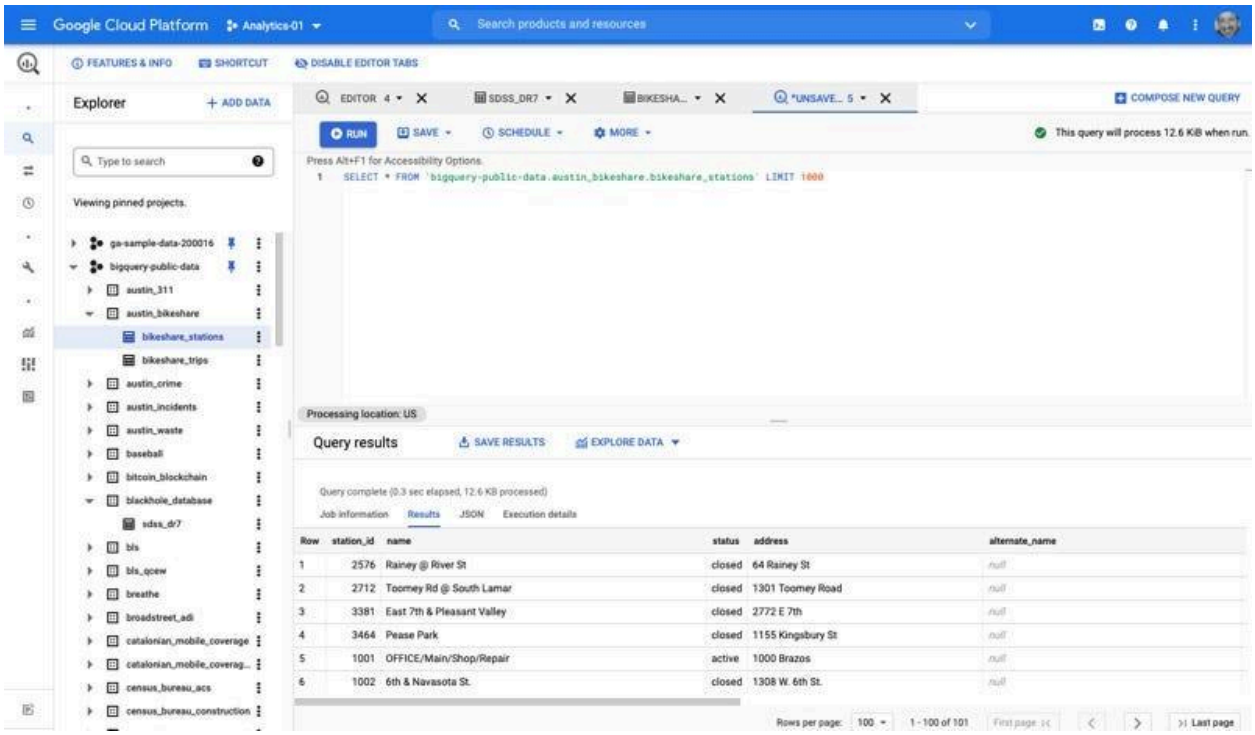
Retrieving Data

BigQuery uses the standard SQL dialect (in most cases — for ML (wait, what?! Machine Learning…yup!), for example, it uses its own SQL dialect) to query the data and send the dataset back to the caller (R, in this case). In the past, Dremel used its own dialect of SQL, now called legacy SQL, which can still be used today.

There are two sections of a dataset — the project id and the dataset. For example, if you wanted to query the Austin Bikeshare Stations in the BigQuery public data, you would reference the austin_bikeshare from bigquery-public-data like so:

```
SQL
SELECT * FROM 'bigquery-public-data.austin_bikeshare'
```

And the results will look like this:



If you were to do this query inside R, it would look like this:

```python
# sqlQuery will take care of opening & closing the connetion to BQ.

bike_share_stations <- sqlQuery('SELECT * FROM
`bigquery-public-data.austin_bikeshare.bikeshare_stations` LIMIT 1000')
```

One final thing to mention. BigQuery uses arrays. This can be cool or a huge pain. Let's say you are working with GA4 data and you only want to select the event_params.key column. That column is part of an array. So, to select it, you cannot write:

```
SQL

SELECT event_params.key

FROM    ga-sample-data-200016.analytics_252049252.events_20220113

LIMIT     1000.
```

You have to write:

```
SQL

SELECT event_params[SAFE_OFFSET(0)].key

FROM    ga-sample-data-200016.analytics_252049252.events_20220113

LIMIT     1000
```

Also, each day is nested as another dataset. So, to access all days, you have to use an asterisk at the end of the data source, like this:

```
SQL

SELECT event_params[SAFE_OFFSET(0)].key

FROM    ga-sample-data-200016.analytics_252049252.events_*

LIMIT     1000
```

Conclusion

That should get you started using BigQuery with R. Once you get used to uploading the data and how to manage the GCP process, you'll probably want to find out what else GCP can provide you. There's a lot, including ML through SQL (wait, what?! I know! … I'm still going to probably use R for that, but Wow!)

There's still so much more to cover. I'm sure I'm going to be writing a lot more about BigQuery.