

Connecting to Google Cloud Platform

Maria C. D'Angelo, PhD

Senior Data Scientist, Delphia Inc

Google Cloud Platform

The screenshot shows the Google Cloud Platform dashboard for the project "My First Project".

- Project info:** Project name: My First Project, Project ID: rational-photon-239401, Project number: 826624791147. Action: Go to project settings.
- API APIs:** Requests (requests/sec) chart. Message: No data is available for the selected time frame. Action: Go to APIs overview.
- Google Cloud Platform status:** All services normal. Action: Go to Cloud status dashboard.
- Billing:** Estimated charges CAD \$0.00 for the billing period May 1 – 4, 2019. Action: View detailed charges.
- Error Reporting:** No sign of any errors. Have you set up Error Reporting? Action: Learn how to set up Error Reporting.
- News:** (partial view)

Follow along at: <https://github.com/MCDAngelo/connectGCP>

Google Cloud Platform

Today I'll be focusing two parts of Google Cloud Platform that I've used to access data: Google Cloud Storage and BigQuery

Google Cloud Storage

- Online file storage service for storing and accessing data on Google Cloud Platform
- Can be accessed from R using the `googleCloudStorageR` package

BigQuery

- Set of tables that enables interactive analysis of massively large datasets working in conjunction with Google Storage
- Can be accessed from R using the `bigrquery` package

Google Cloud Storage

The screenshot shows the Google Cloud Platform Storage Browser. On the left, there's a sidebar with options: Browser (selected), Transfer, Transfer Appliance, and Settings. The main area has tabs for Browser, CREATE BUCKET, REFRESH, and DELETE. A search bar is at the top right. Below it, a table lists buckets. One bucket is shown in detail: **mcd_test_bucket**, Multi-Regional, European Union (multiple regions in the European Union), Per object, None, Bucket policy & ACLs (Off). There's also a 'Columns' dropdown.

This screenshot shows the 'Bucket details' page for the **mcd_test_bucket**. The sidebar on the left is identical to the previous screenshot. The main content area shows the bucket name and navigation links: Objects (selected), Overview, Permissions, and Bucket Lock. It includes buttons for Upload files, Upload folder, Create folder, Manage holds, and Delete. A filter bar is present. Below, a table lists objects: **activity.csv** (342.61 KB, text/csv, Multi-Regional, 5/8/19, Not public, Google-managed key, Retention expiration date: -, Holds: None) and **clapping-hands-sign.gif** (70.91 KB, image/gif, Multi-Regional, 5/8/19, Not public, Google-managed key, Retention expiration date: -, Holds: None).

Google Cloud Storage

Important terms

Project

- Organizes all your Google Cloud Platform resources.
- Name is shown at the top of the window (e.g. "My First Project" in the previous slide).
- If you click on the project name, you'll see the corresponding **project ID** (used later in the presentation).

Bucket

- Similar to a folder or directory in your file system
- Can be nested

Object

- A file (e.g. **.csv, .Rda, .gif**)

The `googleCloudStorageR` Package

1. Getting set up

- How to authenticate

2. What's in my project?

- Listing & inspecting buckets

3. Saving data, models, scripts

- Creating buckets & uploading objects

4. I'd like some data, please!

- Downloading objects

Getting set up

Background: I volunteered to do this talk because of how hard I found it to get started with `googleCloudStorageR` 😐

I initially found authentication confusing because two different authentication files are mentioned in the documentation.

1. Credentials for auto-authentication (`gcs-auto-auth.json`)
2. Client ID for authenticating an application (`client-id.json`). Sets up authentication for an app (e.g. a Shiny App)

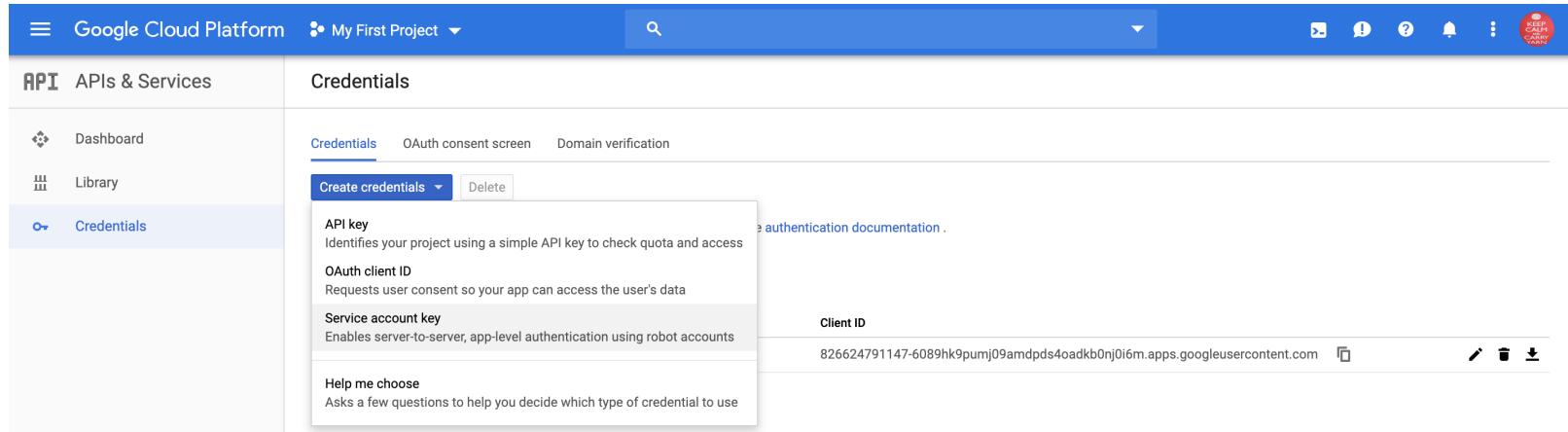
You only need credentials for auto-authentication for local R sessions (*this was not obvious to me*).

I'll focus on how to set up the credentials for auto-authentication in this talk, but have slides at the end on setting up a Client ID.

More helpful details on the two files can be found in the discussion of [issue #94](#) in `googleCloudStorageR`.

🔑 Authentication in googleCloudStorageR

To get started, you need a **service account key**.



The screenshot shows the Google Cloud Platform interface with the navigation bar "Google Cloud Platform" and "My First Project". The left sidebar is titled "API APIs & Services" and has three options: "Dashboard", "Library", and "Credentials", which is currently selected. The main content area is titled "Credentials" and includes tabs for "Credentials", "OAuth consent screen", and "Domain verification". A sub-menu under "Create credentials" shows four options: "API key", "OAuth client ID", "Service account key", and "Help me choose". The "API key" option is highlighted. Below it, there is a brief description: "Identifies your project using a simple API key to check quota and access authentication documentation." To the right, a specific credential entry is shown for an "API key": "Client ID" is listed as "826624791147-6089hk9pumj09amdpds4oadkb0nj0i6m.apps.googleusercontent.com" with a copy icon, a edit icon, a trash icon, and a download icon.

🔑 Authentication in googleCloudStorageR

To create a new service account, you'll need to give the service account a name and permissions.

Warning: I selected Storage Object Admin for my personal account - this may be dangerous.

- Check with your friendly neighbourhood dev-ops/security person for help with this step.

Select the **JSON** key type and when you click on **create** you'll be prompted to download **.json** file.

- I named my file **gcs-auto-auth.json**.

The screenshot shows the 'Create service account key' interface in the Google Cloud Platform. On the left, there's a form for creating a new service account, including fields for 'Service account name' and 'Service account ID'. Below these, under 'Key type', 'JSON' is selected as the recommended option. At the bottom of the form are 'Create' and 'Cancel' buttons. To the right, a 'Selected' dropdown menu is open, showing various roles. The 'Storage Object Admin' role is highlighted with a tooltip: 'Full control of GCS objects.' Other visible roles include Datastore, Error Reporting, IAM, Logging, Monitoring, Organization Policy, Resource Manager, Roles, Service Accounts, Service Management, Service Usage, Stackdriver, Stackdriver Debugger, and Storage.

Authentication in googleCloudStorageR

Once you have your credentials file, set the environment variable **GCS_AUTH_FILE** to the file location.

- TIL: if you set this variable before you load the library, then **gcs_auth()** will be called when the **googleCloudStorageR** library is loaded.

```
# Authenticate service account
Sys.setenv("GCS_AUTH_FILE" = "gcs-auto-auth.json")
library(googleCloudStorageR)

## Setting scopes to https://www.googleapis.com/auth/devstorage.full_control
## Successfully authenticated via gcs-auto-auth.json

# gcs_auth() # no longer needed :)
```

We're in!



- What's in my project? *Listing & inspecting buckets*
- Saving data, models, scripts *Creating buckets & uploading objects*
- I'd like some data, please! *Downloading objects*



Listing Buckets

- **gcs_list_buckets**: will return a data frame with the buckets in your project
- Each bucket will have a name, storage class, location, and time-stamp the bucket was last updated

```
## get your project ID by clicking on the name of it on GCP
my_project <- "rational-photon-239401"
buckets <- gcs_list_buckets(my_project)
buckets

##           name   storageClass location          updated
## 1 mcd_test_bucket MULTI_REGIONAL    EU 2019-05-08 17:57:12
```



Inspecting Buckets

- `gcs_get_bucket`: will return meta data for a bucket

```
buckets$name[[1]]  
## [1] "mcd_test_bucket"  
  
gcs_get_bucket(buckets$name[[1]])  
  
## ==Google Cloud Storage Bucket==  
## Bucket: mcd_test_bucket  
## Project Number: 826624791147  
## Location: EU  
## Class: MULTI_REGIONAL  
## Created: 2019-05-05 17:02:55  
## Updated: 2019-05-08 17:57:12  
## Meta-generation: 4  
## eTag: CAQ=
```



Creating Buckets

- `gcs_create_bucket()`: will create a bucket in your project

```
new_bucket_name <- "rladies_example"
gcs_create_bucket(new_bucket_name, projectId = my_project, location = "EU")

## 2019-05-09 16:23:03 -- Bucket created successfully:rladies_example in EU

## ==Google Cloud Storage Bucket==
## Bucket:          rladies_example
## Project Number:  826624791147
## Location:        EU
## Class:           MULTI_REGIONAL
## Created:         2019-05-09 20:23:10
## Updated:         2019-05-09 20:23:10
## Versioning:      FALSE
## Meta-generation: 1
## eTag:            CAE=

gcs_list_buckets(my_project)

##             name  storageClass location          updated
## 1 mcd_test_bucket  MULTI_REGIONAL    EU 2019-05-08 17:57:12
## 2 rladies_example  MULTI_REGIONAL    EU 2019-05-09 20:23:10
```



Uploading Objects to Buckets

- `gcs_upload()`: will upload file to the specified bucket
- Can specify a file or data frame. Data frames are converted to csv by default

```
buckets <- gcs_list_buckets(my_project)
gcs_upload(iris, bucket = buckets$name[[2]])

## 2019-05-09 16:23:04 -- File size detected as 4.7 Kb

## ==Google Cloud Storage Object==
## Name: iris.csv
## Type: text/csv
## Size: 4.7 Kb
## Media URL: https://www.googleapis.com/download/storage/v1/b/rladies_example/o/iris.csv?ge=
## Download URL: https://storage.cloud.google.com/rladies_example/iris.csv
## Public Download URL: https://storage.googleapis.com/rladies_example/iris.csv
## Bucket: rladies_example
## ID: rladies_example/iris.csv/1557433392904290
## MD5 Hash: iFpB0GOnkjro0OhATZKRow==
## Class: MULTI_REGIONAL
## Created: 2019-05-09 20:23:12
## Updated: 2019-05-09 20:23:12
## Generation: 1557433392904290
## Meta Generation: 1
## eTag: COLwlMGjj+ICEAE=
## crc32c: c+bZLQ==
```



Listing Objects

- `gcs_list_objects()`: will list objects in a specified bucket

```
(objects <- gcs_list_objects(bucket = buckets$name[2]))  
## id  
## 1 rladies_example/iris.csv/1557433392904290  
## selfLink  
## 1 https://www.googleapis.com/storage/v1/b/rladies_example/o/iris.csv  
## name bucket generation metageneration contentType  
## 1 iris.csv rladies_example 1557433392904290 1 text/csv  
## timeCreated updated storageClass  
## 1 2019-05-09 20:23:12 2019-05-09 20:23:12 MULTI_REGIONAL  
## timeStorageClassUpdated size md5Hash  
## 1 2019-05-09T20:23:12.903Z 4.7 Kb iFpB0GOnkjro0OhATZKRow==  
##  
## 1 https://www.googleapis.com/download/storage/v1/b/rladies_example/o/iris.csv?generation=1557433392904290  
## crc32c etag componentCount contentLanguage  
## 1 c+bZLQ== COLwlMGjj+ICEAE= NA NA
```



Downloading Objects

- **gcs_get_object()**: Will download an object directly into the R session (*watch out for large files!*)
- Can save the object directly to disk with **saveToDisk = TRUE**

```
gcs_eg_df <- gcs_get_object(object = objects$name[[1]],
                                bucket = buckets$name[[2]])
head(gcs_eg_df)

## # A tibble: 6 x 6
##       X1 Sepal.Length Sepal.Width Petal.Length Petal.Width Species
##   <dbl>      <dbl>      <dbl>      <dbl>      <dbl> <chr>
## 1     1          5.1        3.5        1.4        0.2 setosa
## 2     2          4.9        3          1.4        0.2 setosa
## 3     3          4.7        3.2        1.3        0.2 setosa
## 4     4          4.6        3.1        1.5        0.2 setosa
## 5     5          5          3.6        1.4        0.2 setosa
## 6     6          5.4        3.9        1.7        0.4 setosa
```

🔥 Deleting Objects and Buckets

(cleaning up - use with caution)

- `gcs_delete_object()`: will delete an object in a specified bucket
- `gcs_delete_bucket()`: will delete a bucket in your project

Note: you cannot delete buckets that contain objects

```
if(new_bucket_name %in% buckets$name){  
  obj_to_delete <- gcs_list_objects(bucket = new_bucket_name)  
  gcs_delete_object(object_name = obj_to_delete$name[[1]],  
                    bucket = new_bucket_name)  
  gcs_delete_bucket(new_bucket_name)  
}  
  
## 2019-05-09 16:23:07 -- Bucket rladies_example deleted successfully.  
## [1] TRUE
```

Google BigQuery

The screenshot shows the Google BigQuery web interface. On the left, there is a sidebar with navigation links: Query history, Saved queries, Job history, Transfers, Scheduled queries, BI Engine, Resources, and ADD DATA. Below these, it shows a dataset named "rational-photon-239401" containing a "test_dataset" which has a "mtcars" table selected. The main area is the "Query editor" where a single row labeled "1" is visible. Below the editor are buttons for Run, Save query, Save view, Schedule query, and More. To the right of the editor, there are buttons for QUERY TABLE, COPY TABLE, DELETE TABLE, and EXPORT. The "mtcars" table schema is displayed in a table with columns: Field name, Type, Mode, and Description. The schema is as follows:

Field name	Type	Mode	Description
mpg	FLOAT	NULLABLE	
cyl	FLOAT	NULLABLE	
disp	FLOAT	NULLABLE	
hp	FLOAT	NULLABLE	
drat	FLOAT	NULLABLE	
wt	FLOAT	NULLABLE	
qsec	FLOAT	NULLABLE	

Google BigQuery

Important terms

Project ✓

Dataset

Table



lions and tigers and bears, oh my!

Google BigQuery

Important terms

Project

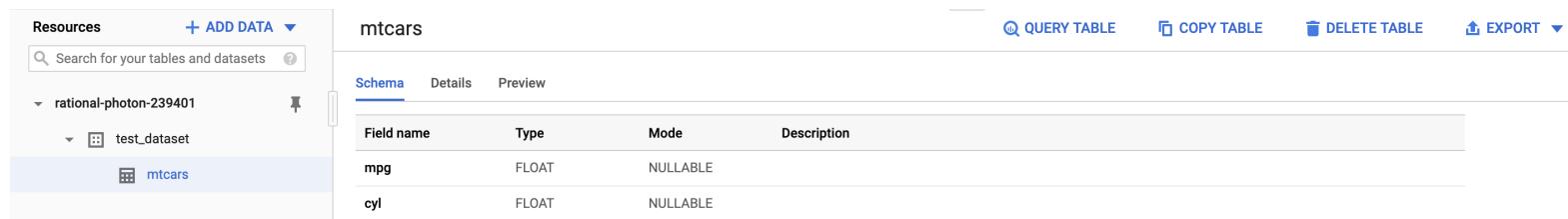
- Way to organize your Google Cloud Platform resources. In this example, my project is called **rational-photon-239401**.

Dataset

- Like buckets or folders, a way to organize tables. In this example, there is one dataset called **test_dataset**.

Table

- A table of data. In this example, it is called **mtcars**.



The screenshot shows the Google BigQuery web interface. On the left, the navigation pane displays the project 'rational-photon-239401' and the dataset 'test_dataset', with the table 'mtcars' selected. The main area is titled 'mtcars' and shows the table schema. The schema table has columns for 'Field name', 'Type', 'Mode', and 'Description'. Two rows are listed: 'mpg' (FLOAT, NULLABLE) and 'cyl' (FLOAT, NULLABLE). At the top right, there are buttons for 'QUERY TABLE', 'COPY TABLE', 'DELETE TABLE', and 'EXPORT'.

Field name	Type	Mode	Description
mpg	FLOAT	NULLABLE	
cyl	FLOAT	NULLABLE	

bigrquery Package

1. Getting set up

- How to authenticate

2. Sharing is caring

- Creating and uploading tables

3. I'd like some data, please!

- Querying and downloading tables
 - Low-level API
 - **DBI** interface
 - **dplyr** interface



Authentication with `bigrquery`

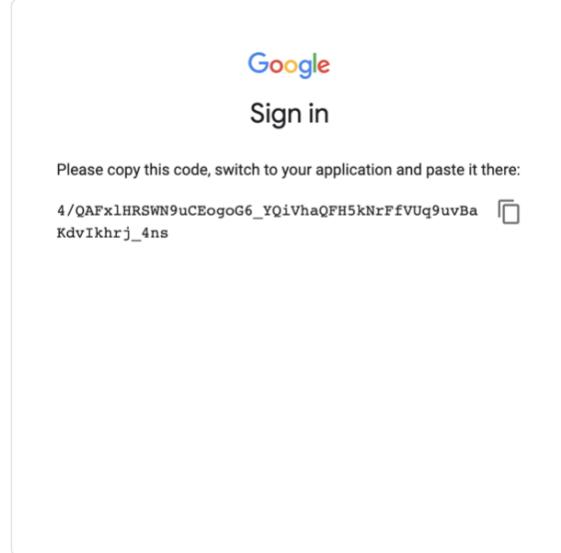
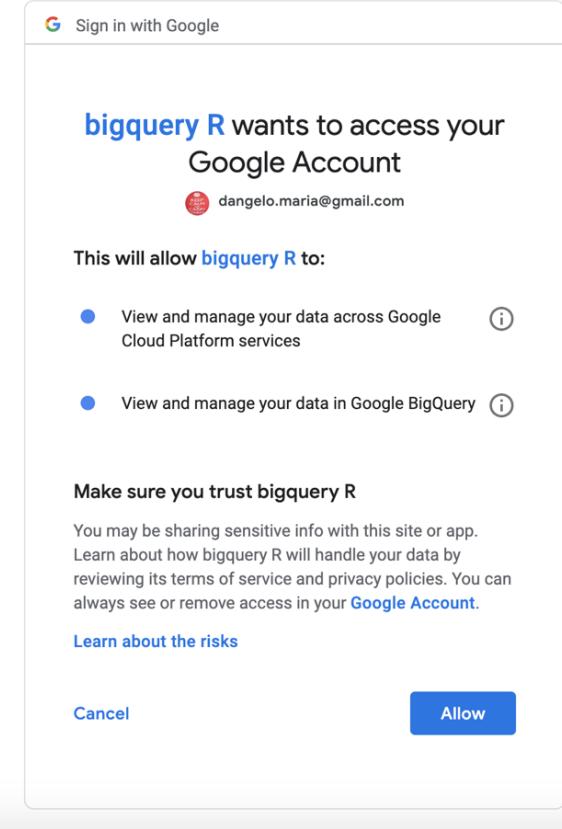
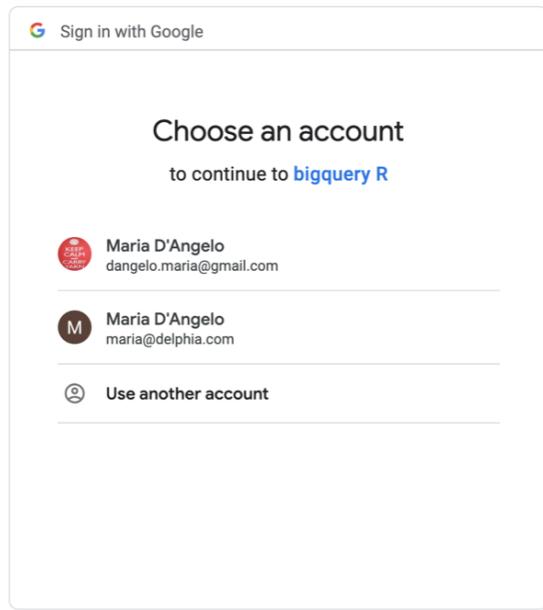
- When you run your first command with `bigrquery`, you will be prompted to authenticate.
- Note: to use `bigrquery` in a non-interactive session (i.e., when knitting a document), you will need to keep a local file to cache the access credentials. (Select 1 below)

```
Use a local file ('.httr-oauth'), to cache OAuth access credentials between R sessions?
```

1: Yes
2: No

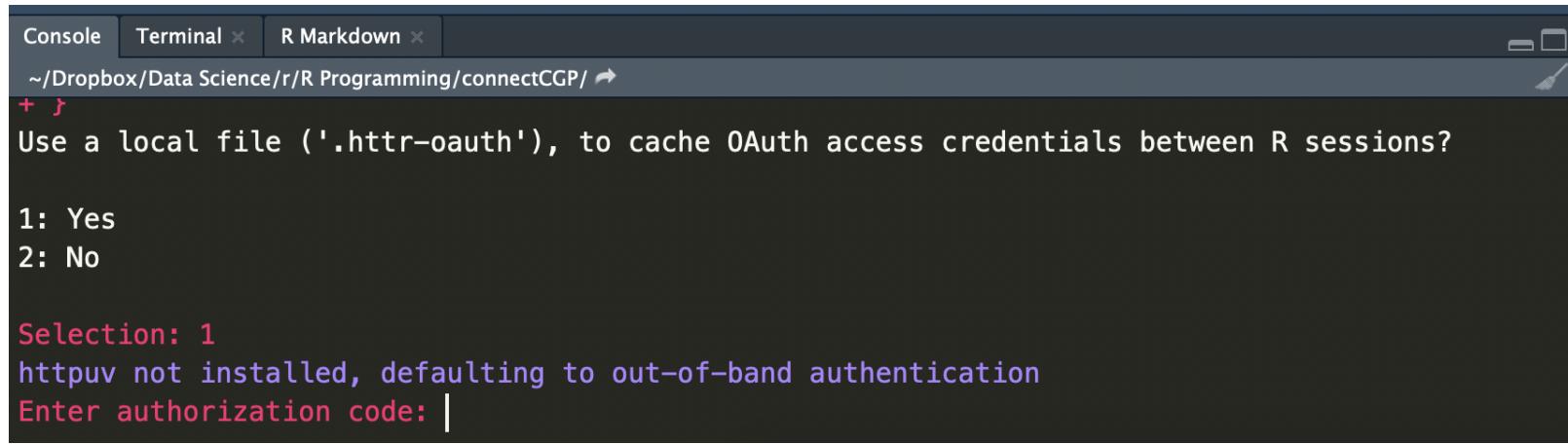
Selection:

🔒 Authentication with **bigrquery**



🔑 Authentication with `bigrquery`

Paste the authentication code you just copied into R.



The screenshot shows an RStudio interface with three tabs: Console, Terminal, and R Markdown. The Console tab is active, displaying the following text:

```
~/Dropbox/Data Science/r/R Programming/connectCGP/ ↵
+ r
Use a local file ('.httr-oauth'), to cache OAuth access credentials between R sessions?

1: Yes
2: No

Selection: 1
httpuv not installed, defaulting to out-of-band authentication
Enter authorization code: |
```

🔑 Authentication with **bigrquery**

You'll now have a `.httr-oauth` file in your directory.

```
doge ➤ ...r/R Programming/connectCGP ➤ master ✓ ➤ ls -la
total 936
drwxr-xr-x@ 10 mariadangelo  staff      320  4 May 11:54 .
drwxr-xr-x@ 14 mariadangelo  staff     448 28 Apr 15:09 ..
drwxr-xr-x@  4 mariadangelo  staff     128 28 Apr 15:09 .Rproj.user
drwxr-xr-x@ 12 mariadangelo  staff     384  4 May 12:00 .git
-rw-r--r--@  1 mariadangelo  staff      61  4 May 11:59 .gitignore
-rw-----@  1 mariadangelo  staff    4658  4 May 11:56 .httr-oauth
-rw-r--r--@  1 mariadangelo  staff    2443  4 May 11:54 connectCGP.Rmd
-rw-r--r--@  1 mariadangelo  staff     205  4 May 11:53 connectCGP.Rproj
-rw-r--r--@  1 mariadangelo  staff   419849 4 May 11:30 connectCGP.html
drwxr-xr-x@ 12 mariadangelo  staff     384  4 May 11:55 images
12:00:12
```



Authentication with **bigrquery**

The `.httr-oauth` file is automatically added to your `.gitignore` file

```
more .gitignore  
## .Rproj.user  
## .Rhistory  
## .RData  
## .Ruserdata  
## .DS_Store  
## .httr-oauth  
## gcs-auto-auth.json  
## client-id.json
```

What's Next?



- Sharing is caring *Creating and uploading tables*
- I'd like some data, please! *Querying and downloading tables*



Getting your ducks in a row

- `bq_table()`: set up information for `bq_table` object

```
library(bigrquery)
my_project <- "rational-photon-239401"
my_dataset <- "test_dataset"
my_table <- "ToothGrowth"

#Set up BQ info
my_bq_table_info <- bq_table(project = my_project,
                                dataset = my_dataset,
                                table = my_table)
```

🔨 Creating & Uploading Tables

- **bq_table_create()**: Creates the table
- **bq_table_upload()**: Uploads data; fields will create the schema from the data frame
You might get an error if fields is not set

```
bq_table_create(my_bq_table_info)
## <bq_table> rational-photon-239401.test_dataset.ToothGrowth

bq_table_upload(x = my_bq_table_info,
                 values = ToothGrowth,
                 fields = as_bq_fields(ToothGrowth))

#Check if table exists
bq_table_exists(my_bq_table_info)
## [1] TRUE
```

↔ Translating to BQ Schema

- `as_bq_fields()`: Convert df schema information into BQ fields

```
str(ToothGrowth)

## 'data.frame':   60 obs. of  3 variables:
## $ len : num  4.2 11.5 7.3 5.8 6.4 10 11.2 11.2 5.2 7 ...
## $ supp: Factor w/ 2 levels "OJ","VC": 2 2 2 2 2 2 2 2 2 2 ...
## $ dose: num  0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 ...

as_bq_fields(ToothGrowth)

## <bq_fields>
##   len <FLOAT>
##   supp <STRING>
##   dose <FLOAT>
```



Downloading Tables

- **bq_project_query()**: Submits a query, waits for it to complete, and returns a **bq_table**
- **bq_table_download()**: Retrieves rows from a **bq_table** object, can specify **page_size** for retrieval in chunks (useful for large tables, default is 10,000 rows)

```
library(glue)
sql <- glue("SELECT * FROM `my_project}.{my_dataset}.{my_table}`")

tb <- bq_project_query(my_project, sql)
bq_eg_df <- bq_table_download(tb)

head(bq_eg_df)

## # A tibble: 6 x 3
##       len supp   dose
##     <dbl> <chr> <dbl>
## 1     4.2 VC    0.5
## 2    11.5 VC    0.5
## 3     7.3 VC    0.5
## 4     5.8 VC    0.5
## 5     6.4 VC    0.5
## 6    10.0 VC    0.5
```

DBI Interface

- **DBI::dbConnect()**: connect to BigQuery like any other database, only need to supply your project name
- **DBI::dbGetQuery()**: using the connection, execute the query provided
- **DBI::dbReadTable(), DBI::dbWriteTable()**: Read and write tables

```
sql <- glue("SELECT * FROM `my_project}.{my_dataset}.{my_table}`")  
  
con <- DBI::dbConnect(bigrquery(), project = my_project)  
dbi_eg_df <- DBI::dbGetQuery(con, sql)  
head(dbi_eg_df)  
  
## # A tibble: 6 x 3  
##       len supp dose  
##   <dbl> <chr> <dbl>  
## 1     4.2 VC    0.5  
## 2    11.5 VC    0.5  
## 3     7.3 VC    0.5  
## 4     5.8 VC    0.5  
## 5     6.4 VC    0.5  
## 6    10.0 VC    0.5  
  
DBI::dbDisconnect(con)
```

dplyr Interface

- **tbl()**: create a table from a data source (in our case: BigQuery)
- **dbplyr** is used to access BigQuery tables as if they are in-memory data frames, automatically converts dplyr code into SQL
- Note: This does not actually execute the query, **nrow()** gives **NA** and the class is a lazy connection

```
library(dplyr); library(dbplyr)
con <- DBI::dbConnect(bigrquery::bigrquery(), project = my_project)
dplyr_eg <- tbl(con, glue("{my_dataset}.{my_table}"))
dplyr_query <- dplyr_eg %>% group_by(supp, dose) %>% count()
nrow(dplyr_query)

## [1] NA

class(dplyr_query)

## [1] "tbl_BigQueryConnection" "tbl_dbi"
## [3] "tbl_sql"                 "tbl_lazy"
## [5] "tbl"
```

- Helpful vignette from **dbplyr** can be found [here](#)

dplyr Interface

- **collect()**: retrieves data into a local tibble
- We can now see the number of rows and that the output is a **data.frame**

```
dplyr_eg_df <- collect(dplyr_query)
nrow(dplyr_eg_df)

## [1] 6

class(dplyr_eg_df)

## [1] "grouped_df"   "tbl_df"        "tbl"          "data.frame"

head(dplyr_eg_df)

## # A tibble: 6 x 3
## Groups:   supp, dose [6]
#>   supp     dose     n
#>   <chr>    <dbl>   <int>
#> 1 OJ        2       10
#> 2 VC        2       10
#> 3 VC        1       10
#> 4 OJ        0.5      10
#> 5 VC        0.5      10
#> 6 OJ        1       10

DBI::dbDisconnect(con)
```

🔥 Deleting Tables

(cleaning up - use with caution)

- **bq_table_exists()**: checks to see if table exists, returns **TRUE** or **FALSE**
- **bq_table_delete()**: deletes table

```
if (bq_table_exists(my_bq_table_info)){  
    bq_table_delete(my_bq_table_info)  
}
```

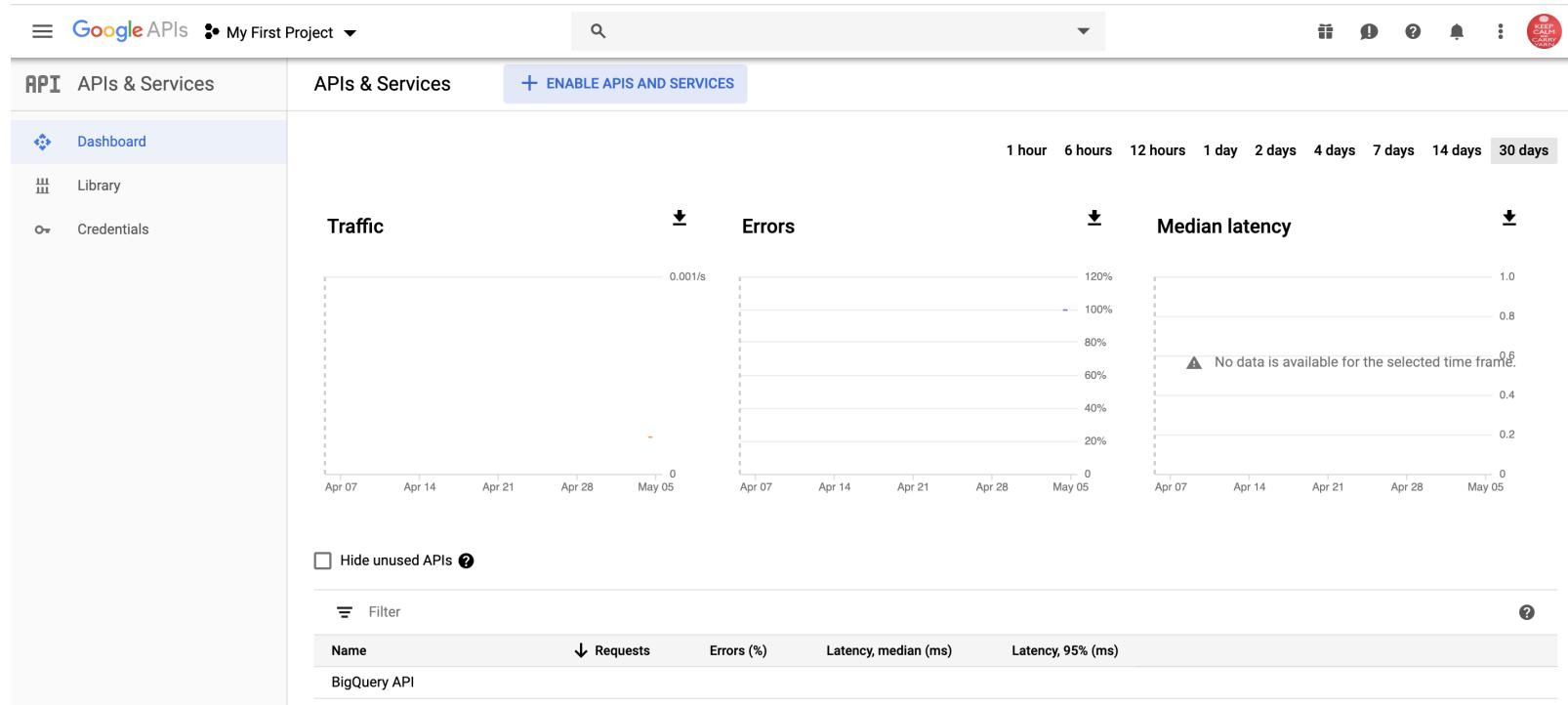
Thank you!



Any Questions?

🐦 Connect with me on Twitter!
[@mariacdangelo](https://twitter.com/mariacdangelo)

App Authentication with googleCloudStorageR



App Authentication with googleCloudStorageR

The screenshot shows the Google Cloud Platform API & Services interface. The left sidebar has 'API' selected under 'APIs & Services'. The main area is titled 'Credentials' with tabs for 'Credentials', 'OAuth consent screen', and 'Domain verification'. The 'Credentials' tab is active. A modal window is open over the main content, titled 'APIs Credentials'. It contains instructions about needing credentials to access APIs and links to enable APIs and view documentation. Below this, a 'Create credentials' button is highlighted in blue. A dropdown menu from this button lists four options: 'API key', 'OAuth client ID', 'Service account key', and 'Help me choose'. Each option has a brief description.

Option	Description
API key	Identifies your project using a simple API key to check quota and access
OAuth client ID	Requests user consent so your app can access the user's data
Service account key	Enables server-to-server, app-level authentication using robot accounts
Help me choose	Asks a few questions to help you decide which type of credential to use

App Authentication with googleCloudStorage

≡ Google APIs • My First Project ▾

🔍

Gift Chat Help Bell More

API APIs & Services Credentials

Dashboard Library Credentials

Credentials OAuth consent screen Domain verification

Before your users authenticate, this consent screen will allow them to choose whether they want to grant access to their private data, as well as give them a link to your terms of service and privacy policy. This page configures the consent screen for all applications in this project.

Verification status ⓘ Needs verification

Application name ⓘ The name of the app asking for consent RLadies_Test

Application logo ⓘ An image on the consent screen that will help users recognize your app Local file for upload Browse

Support email ⓘ Shown on the consent screen for user support

About the consent screen

The consent screen tells your users who is requesting access to their data and what kind of data you're asking to access.

OAuth verification

To protect you and your users, your consent screen and application may need to be verified by Google. Verification is required if your app is marked as Public and at least one of the following is true:

- Your app uses a sensitive and/or restricted scope
- Your app displays an icon on its OAuth consent screen
- Your app has a large number of authorized domains
- You have made changes to a previously-verified OAuth consent screen

The verification process may take up to several weeks, and you will receive email updates as it progresses. [Learn more about verification](#)

App Authentication with googleCloudStorage

The screenshot shows the Google Cloud Platform API console interface. At the top, there's a navigation bar with 'Google APIs' and 'My First Project'. Below it, a search bar and a toolbar with various icons. A breadcrumb trail says 'Create OAuth client ID'. The main content area has a heading about OAuth 2.0 client IDs, followed by a section for selecting an application type (Web application, Android, Chrome App, iOS, Other), where 'Other' is selected. A 'Name' field contains 'RLadies_GCS'. At the bottom are 'Create' and 'Cancel' buttons.

For applications that use the OAuth 2.0 protocol to call Google APIs, you can use an OAuth 2.0 client ID to generate an access token. The token contains a unique identifier. See [Setting up OAuth 2.0](#) for more information.

Application type

Web application

Android [Learn more](#)

Chrome App [Learn more](#)

iOS [Learn more](#)

Other

Name ?

RLadies_GCS

Create Cancel

App Authentication with googleCloudStorageR

Create a client ID and download .json file (icon on far right) I've named it **client-id.json**

The screenshot shows the Google Cloud Platform API Credentials page. The left sidebar has 'API' selected under 'APIs & Services'. The main area is titled 'Credentials' and shows three tabs: 'Credentials' (selected), 'OAuth consent screen', and 'Domain verification'. A 'Create credentials' button is visible. Below the tabs, a note says 'Create credentials to access your enabled APIs. For more information, see the [authentication documentation](#)'. The 'OAuth 2.0 client IDs' section lists one item:

Name	Creation date	Type	Client ID
RLadies_GCS	May 5, 2019	Other	[REDACTED].apps.googleusercontent.com

On the far right of the table are edit, delete, and download icons.

App Authentication with `googleCloudStorageR`

To authenticate the client ID run the following code:

```
scope = "https://www.googleapis.com/auth/cloud-platform"

# Authenticate ClientID
googleAuthR::gar_set_client("client-id.json", scopes = scope)
```

App Authentication with googleCloudStorageR

