**HU Extension School E-90 Cloud Computing**

# Final Project Topic: Google BigQuery

**Hanjiao Qiu**

## Problem

1) Query massive dataset using Google BigQuery. Search and sort the popular CDN (Content Delivery Network) providers, top sites to have content security policy to prevent XSS and explore other web behaviors. Understand the web performance from the HTTP Archive data since the last two months. 2) Do real-time system log analysis, execute queries and visualize the log data through interacting with the BigQuery in Google sheets. This problem will examine the super fast performance, high reliability and security and low-cost in big data analysis of Google BigQuery.

## Description

Google BigQuery provides three ways to access it: Web UI, bq command-line tool and BigQuery REST API using a variety of client libraries such as Java, Python. It also enables a variety of third-party tools to interact with it, such as visualizing or loading the data. This project details the three ways to analyze the large scale HTTP Archive data using BigQuery. It also demonstrates the data loading and visualization by interacting with Fluentd and Google sheets.

## Why BigQuery?

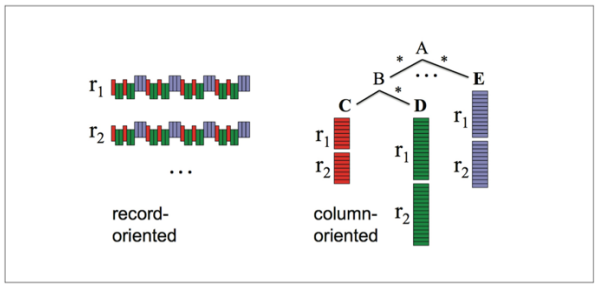
Available Big Data Solutions & Issues

- Hadoop & Hive
    - Pros
        - High Scalable and Distributed Computing
        - Storage (HDFS) optimized for high throughput
    - Cons
        - Security disabled by default
        - MapReduce is batch based, hence no real time operations
        - Costly to maintain
- Amazon Redshift
    - Pros
        - Highly Scalable, talks of handing Petabytes
        - Elastic set of resources to return result sets
        - Almost 10x fast as compared to Hadoop
    - Cons
        - High costs of Data Migration and integration
        - Operations/Maintenance cost may shoot up

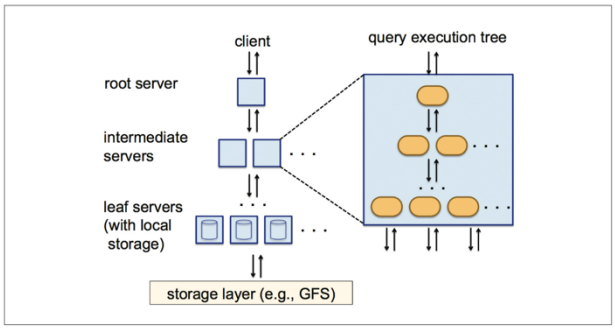
## BigQuery Benefits

1) Analytics as a service, uses SQL-style query and can be accessed by RESTful API.
2) Reasonably fast, process multi-terabyte datasets in seconds.
3) Scalable, reliable and secure.

4) It has a variety of use cases:
   - Log Analysis. Making sense of computer generated records.
   - Retailer. Using data to forecast product sales.
   - Ads Targeting. Targeting proper customer sections.
   - Sensor Data. Collect and visualize ambient data.
   - Data Mashup. Query terabytes of heterogeneous data.
5) BigQuery is based on Dremel, a scalable, interactive ad-hoc query system for analysis of read-only nested data. BigQuery used Columnar storage & multi-level execution trees to achieve interactive performance for queries against multi-terabytes datasets. The query is processed by thousands of servers in a multi-level execution tree structure, with the final results aggregated at the root.



*Columnar storage of Dremel*



*Tree architecture of Dremel*

## Dataset

HTTP Archive data (http://httparchive.org/), also available through BigQury. (200+ GB, csv)
Google BigQuery public data sample. (50+ GB, csv)
Real-time log generated from Google Compute Engine Instance (78 MB, csv)

## Operating System

Mac OS, Ubuntu 14.01

## Software

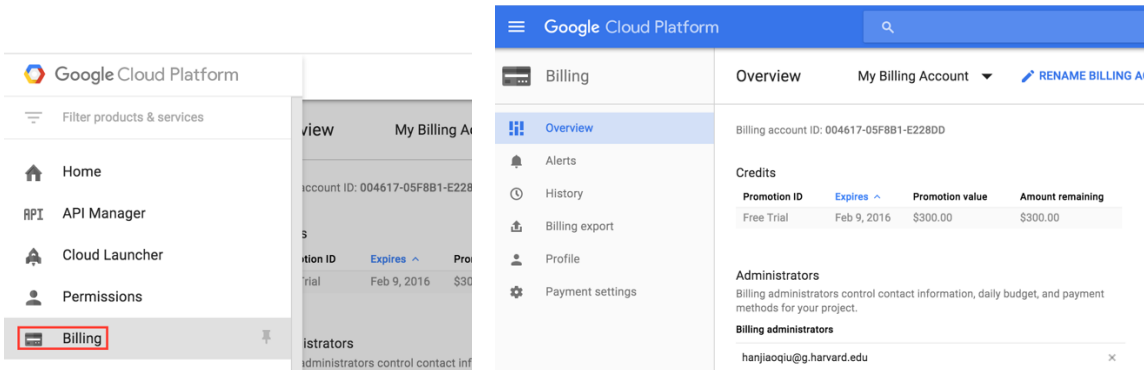
Google Cloud SDK, Google BigQuery API Client Library for Java, Java version 1.8.0.

## Detailed Steps

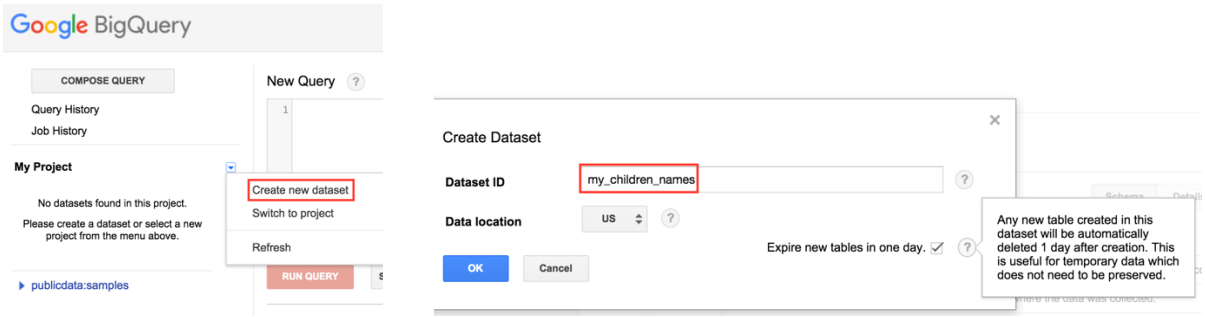
1) Use the BigQuery Web UI to analyze the data.

Reference: https://cloud.google.com/bigquery/web-ui-quickstart

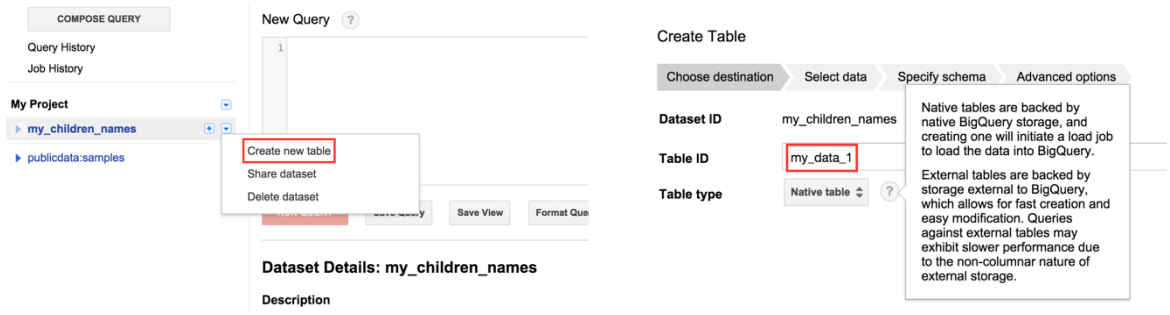
1.1. Go to the Google Cloud Platform console, accept the terms of service and create a new project. Before loading the data, sign up for the free tier and set up billing in the console.




1.2. Open the BigQuery web UI. (https://bigquery.cloud.google.com/queries/) Create a dataset to hold the data. Click the down narrow icon next to the project name in the navigation, and then click "Create new dataset".

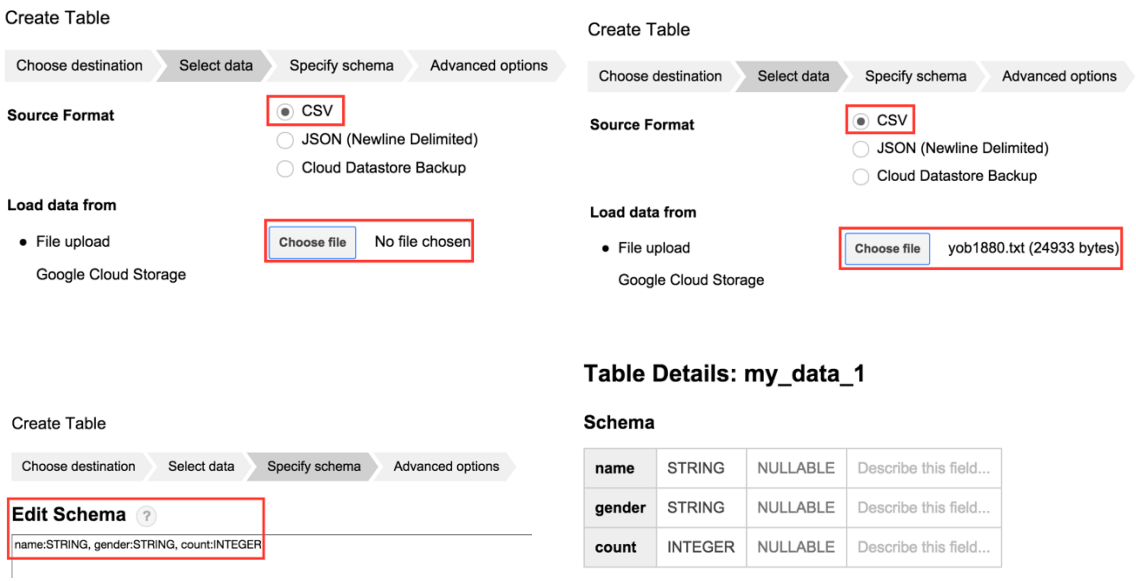



1.3. Download some example data about popular baby names from US Social Security Administration. Hover on the dataset ID we just created, click "Create new table".

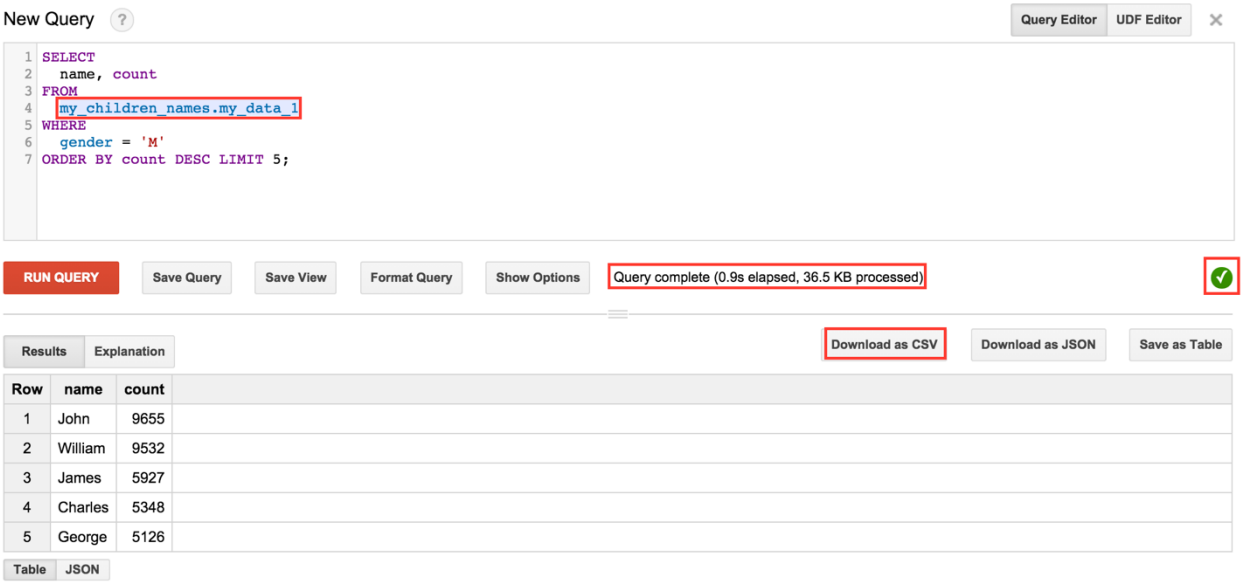



Input the table ID. Click the "Next" button, then "Choose file". Upload a sample file "yob1880.txt". Then click "Next". In "Edit Schema", click "Edit as text", type in "name:string, gender:string, count:string". Click the "Submit" button.

Once you complete the above steps, BigQuery creates a table and loads data into it. When you click on the table, you can see the "schema" shows there.

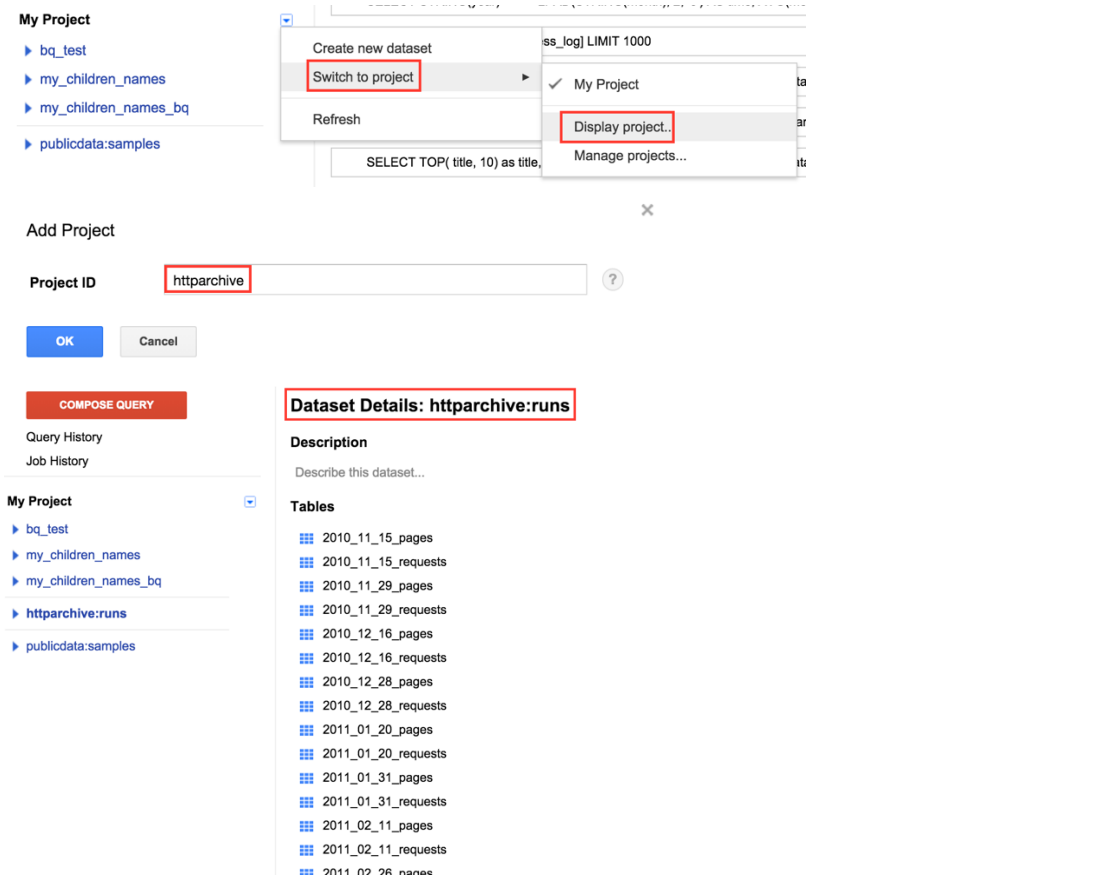



1.4. Click the "Compose Query" button, run queries against the data we just uploaded.

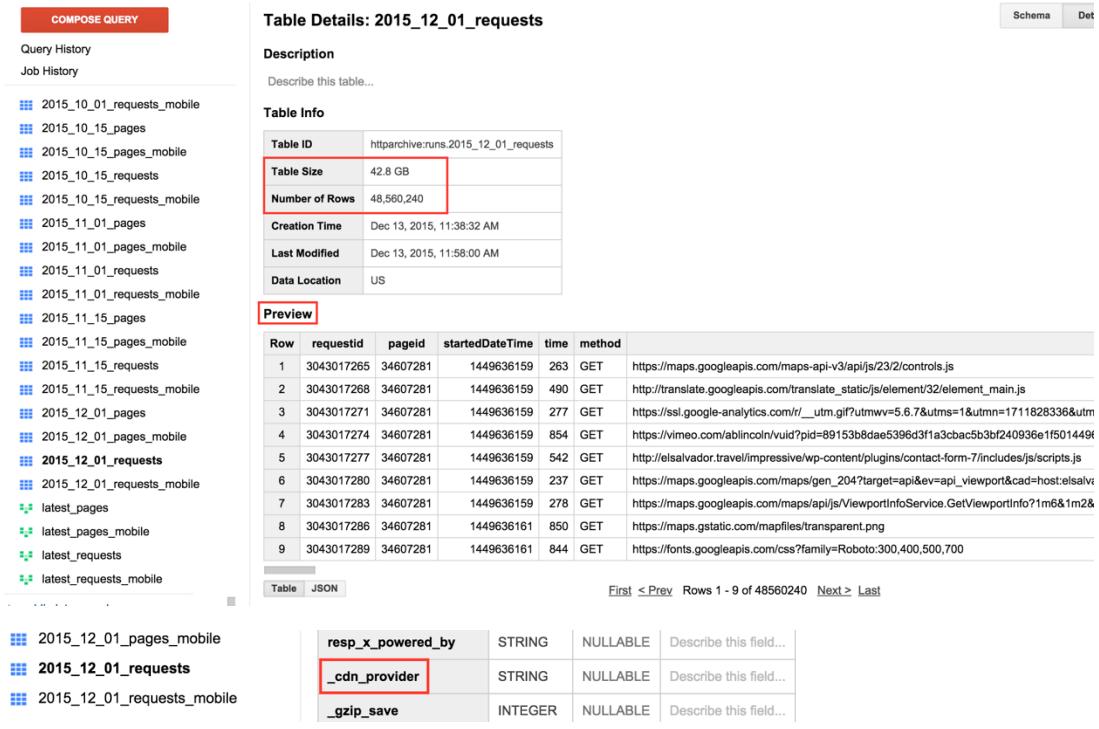



1.5. Load in another data set called "HTTPArchive" (http://httparchive.org/index.php). The HTTP Archive tracks how the web is built. It uses distributed web crawler to extract web data twice a month. We can analyze the trends in web technology, interesting statics and website performance through the data. We will solve our problems using this dataset!

Load in this data set by click the down narrow icon next to "My Project". Go to "Switch to project -> Display project -> Add Project", enter "httparchive". The dataset "httparchive:runs" will show on the left under "My Project".




We can see that the dataset is very huge!! Preview the table details. Since it has a column named "_cdn_provider" , we can analyze the CDN providers through this dataset.



Let's measure how many requests go through CDNs and the popularity of different CDNs.

CDNUsage.sql

```
SELECT cdn, num, ROUND(ratio*100) as percent FROM (
  SELECT cdn, COUNT(cdn) as num, RATIO_TO_REPORT(num) OVER() ratio FROM (
    SELECT CASE
      WHEN _cdn_provider IN ('')
        THEN 'None'
        ELSE 'CDN'
      END as cdn
     FROM httparchive:runs.2015_12_01_requests,
          httparchive:runs.2015_11_15_requests,
          httparchive:runs.2015_11_01_requests,
          httparchive:runs.2015_10_15_requests,
          httparchive:runs.2015_12_01_requests_mobile,
          httparchive:runs.2015_11_15_requests_mobile,
          httparchive:runs.2015_11_01_requests_mobile,
          httparchive:runs.2015_10_15_requests_mobile
   ) GROUP BY cdn
) ORDER BY percent DESC
```

Destination Table: Select Table — No table selected
Write Preference: Write if empty / Append to table / Overwrite table
Results Size: Allow Large Results
Results Schema: Flatten Results
Query Caching: Use Cached Results
Query Priority: Interactive / Batch
UDF Source URIs: Edit

RUN QUERY | Save Query | Save View | Format Query | Hide Options | Query complete (2.0s elapsed, 902 MB processed)

Results | Explanation | Download as CSV

| Row | cdn | num | percent |
|---|---|---|---|
| 1 | None | 122432081 | 63.0 |
| 2 | CDN | 72461839 | 37.0 |

The CDN usage is about 37% among all the requests. Note that the total size of these tables is above 200 GB. Since BigQuery stores the data in columns, it will only process the data related to the query. Thus, we only process 902 MB data here. It costs only 2 seconds!!

CDNProvider.sql

```
SELECT provider, ROUND(100*ratio) AS percent, num FROM (
  SELECT
    _cdn_provider AS provider, COUNT(*) AS num, RATIO_TO_REPORT(num) OVER() ratio
  FROM
    httparchive:runs.2015_12_01_requests,
    httparchive:runs.2015_11_15_requests,
    httparchive:runs.2015_11_01_requests,
    httparchive:runs.2015_10_15_requests,
    httparchive:runs.2015_12_01_requests_mobile,
    httparchive:runs.2015_11_15_requests_mobile,
    httparchive:runs.2015_11_01_requests_mobile,
    httparchive:runs.2015_10_15_requests_mobile
  WHERE
    _cdn_provider != ''
  GROUP BY provider )
ORDER BY num DESC
LIMIT 10
```

RUN QUERY | Save Query | Save View | Format Query | Show Options | Query complete (2.6s elapsed, 902 MB processed)

Results | Explanation | Download as CSV

| Row | provider | percent | num |
|---|---|---|---|
| 1 | Google | 40.0 | 28997532 |
| 2 | Akamai | 17.0 | 12217367 |
| 3 | Cloudflare | 13.0 | 9079980 |
| 4 | Facebook | 11.0 | 7704510 |
| 5 | Amazon CloudFront | 5.0 | 3857754 |
| 6 | Edgecast | 4.0 | 2664794 |
| 7 | NetDNA | 2.0 | 1627192 |

Table | JSON

First < Prev Rows 1 - 7 of 10 Next > Last

This query sorts out the top 10 most popular CDN providers. Google is the most popular one and occupies 40% of the market.

RUN QUERY | Save Query | Save View | Format Query | Show Options | Query complete (2.6s elapsed, 902 MB processed)

Results | Explanation | Download as CSV

| Row | provider | percent | num |
|---|---|---|---|
| 1 | Google | 40.0 | 28997532 |
| 2 | Akamai | 17.0 | 12217367 |
| 3 | Cloudflare | 13.0 | 9079980 |
| 4 | Facebook | 11.0 | 7704510 |
| 5 | Amazon CloudFront | 5.0 | 3857754 |
| 6 | Edgecast | 4.0 | 2664794 |
| 7 | NetDNA | 2.0 | 1627192 |
| 8 | Fastly | 2.0 | 1574370 |
| 9 | WordPress Jetpack | 2.0 | 1115876 |
| 10 | Reapleaf | 1.0 | 570609 |

Table | JSON

Here is the query about the hostname ranks using Google as the CDN provider.

HostRankInGoogle.sql

```
SELECT req_host, COUNT(req_host) AS num
FROM
  httparchive:runs.2015_12_01_requests,
  httparchive:runs.2015_11_15_requests,
  httparchive:runs.2015_11_01_requests,
  httparchive:runs.2015_10_15_requests,
  httparchive:runs.2015_12_01_requests_mobile,
  httparchive:runs.2015_11_15_requests_mobile,
  httparchive:runs.2015_11_01_requests_mobile,
  httparchive:runs.2015_10_15_requests_mobile
WHERE
  _cdn_provider = 'Google'
GROUP BY req_host
ORDER BY num DESC
LIMIT 20
```

No Cached Results

RUN QUERY | Save Query | Save View | Format Query | Show Options | Query complete (2.3s elapsed, 4.49 GB processed)

Results | Explanation | Download as CSV

| Row | req_host | num |
|---|---|---|
| 1 | fonts.gstatic.com | 3539718 |
| 2 | www.google-analytics.com | 3072752 |
| 3 | pagead2.googlesyndication.com | 2276876 |
| 4 | googleads.g.doubleclick.net | 2151651 |
| 5 | www.google.com | 1966309 |
| 6 | tpc.googlesyndication.com | 1838237 |
| 7 | apis.google.com | 1796544 |

Table | JSON

First < Prev Rows 1 - 7 of 20 Next > Last

Note that we choose “No Cache Results” here. It only needs 2.3 seconds to process 4.49 GB data for the first time!!

To make the query more interesting, we can check the leaderboard of the sites that enable content security policy, which offers a certain protection against the XSS attack. Google, Akamai, Github are all on the list!!

ContentSecurityPolicy.sql

```
SELECT DOMAIN(url) AS domainname, COUNT(*) AS num
FROM
  httparchive:runs.2015_12_01_requests,
  httparchive:runs.2015_11_15_requests,
  httparchive:runs.2015_11_01_requests,
  httparchive:runs.2015_10_15_requests,
  httparchive:runs.2015_12_01_requests_mobile,
  httparchive:runs.2015_11_15_requests_mobile,
  httparchive:runs.2015_11_01_requests_mobile,
  httparchive:runs.2015_10_15_requests_mobile
WHERE
  LOWER(respOtherHeaders) CONTAINS "content-security-policy"
GROUP BY domainname
ORDER BY num DESC
LIMIT 10
```

RUN QUERY | Save Query | Save View | Format Query | Show Options | Query complete (4.7s elapsed, 34.2 GB processed)

Results | Explanation | Download as CSV

| Row | domainname | num |
|---|---|---|
| 1 | blogger.com | 55434 |
| 2 | atlassbx.com | 21132 |
| 3 | vimeo.com | 15132 |
| 4 | google.com | 8885 |
| 5 | r9cdn.net | 6535 |
| 6 | ok.ru | 6023 |
| 7 | dropboxusercontent.com | 5891 |
| 8 | akamaihd.net | 4823 |
| 9 | cloudfront.net | 3761 |
| 10 | githubusercontent.com | 3565 |

Table | JSON

We can also analyze some interesting statics such as the average individual response size.

AvgResponseSizeByType.sql

```
SELECT format, ROUND(AVG(resp_content_length)/1024) AS size
FROM
  httparchive:runs.2015_12_01_requests,
  httparchive:runs.2015_11_15_requests,
  httparchive:runs.2015_11_01_requests,
  httparchive:runs.2015_10_15_requests,
  httparchive:runs.2015_12_01_requests_mobile,
  httparchive:runs.2015_11_15_requests_mobile,
  httparchive:runs.2015_11_01_requests_mobile,
  httparchive:runs.2015_10_15_requests_mobile
WHERE
  format != ""
GROUP BY format
ORDER BY size DESC
```

No Cached Results

RUN QUERY | Save Query | Save View | Format Query | Show Options | Query complete (2.8s elapsed, 1.65 GB processed)

Results | Explanation | Download as CSV

| Row | format | size |
|---|---|---|
| 1 | flv | 16060.0 |
| 2 | mp4 | 8074.0 |
| 3 | f4v | 2094.0 |
| 4 | flash | 105.0 |
| 5 | jpg | 37.0 |
| 6 | webp | 36.0 |
| 7 | swf | 34.0 |
| 8 | png | 22.0 |
| 9 | gif | 8.0 |
| 10 | svg | 7.0 |
| 11 | ico | 5.0 |

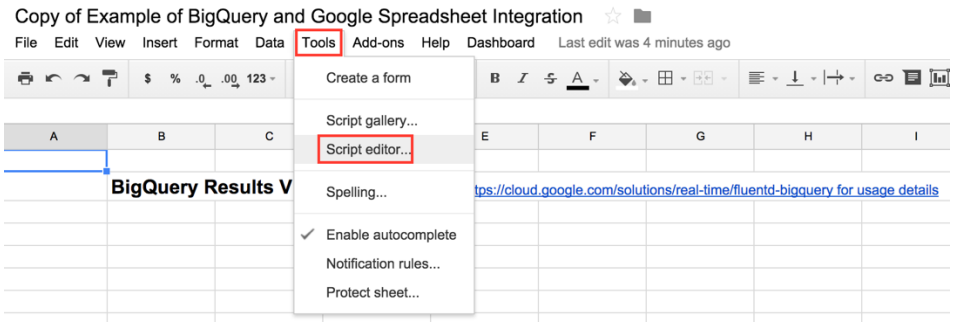
Table | JSON

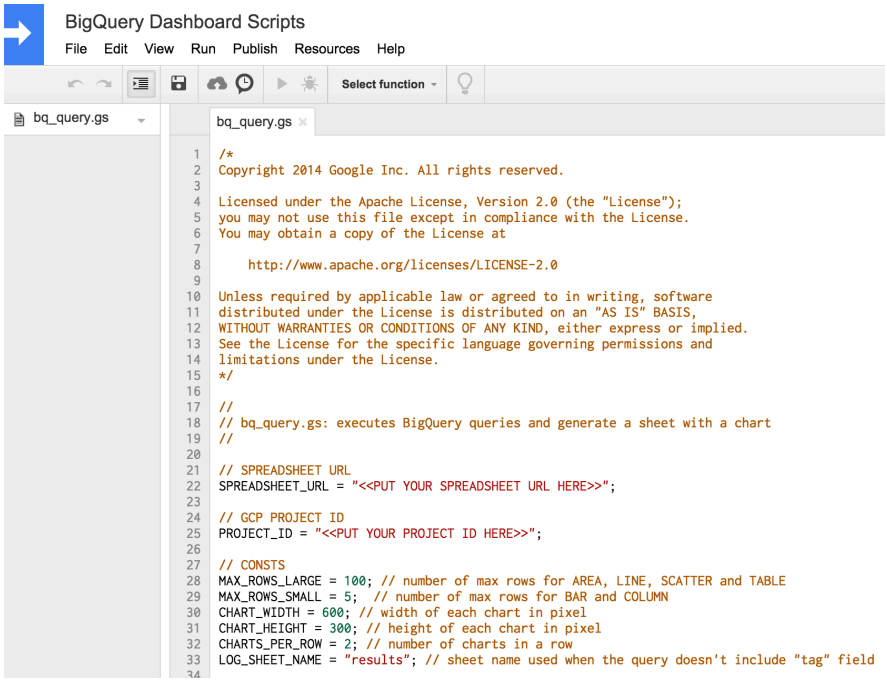
Videos do have the longest content length!

1.6. We can also submit SQL statement to BigQuery from a Google spreadsheet. The spreadsheet contains an App script that executes the BigQuery queries, store the results and visualizes the them in an embedded chart. You can even configure the script and corresponding chart to automatically update at a specific interval.

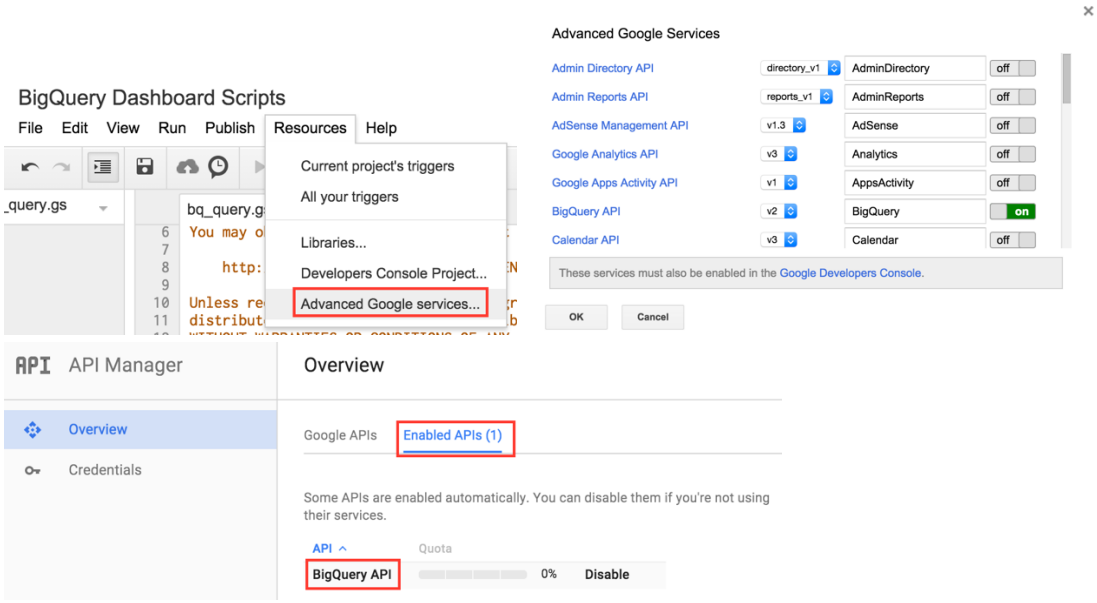
Make a copy of the example of BigQuery and Google Spreadsheet integration: https://docs.google.com/spreadsheets/d/1Xwk2icyXH2DmVIZC33SAs5bs012ZIt0-goyX0dZZu7s/edit

Select "Tools -> Script editor", it will show the script "bq_query.gs". This script will execute BigQuery queries, save query results to a sheet, and generate charts to visualize query results. Replace your spreadsheet url and project ID into the script like below. Also, select "Advanced Google Services" and check the Google BigQuery API is turned on.



```
//
// bq_query.gs: executes BigQuery queries and generate a sheet with a chart
//

// SPREADSHEET URL
SPREADSHEET_URL = "https://docs.google.com/spreadsheets/d/10K57xuoWMPOqAfJLbEh7pqgniGCAGM4Vy1qBCDMU73o/edit#gid=0";

// GCP PROJECT ID
PROJECT_ID = "friendly-path-115605";
```

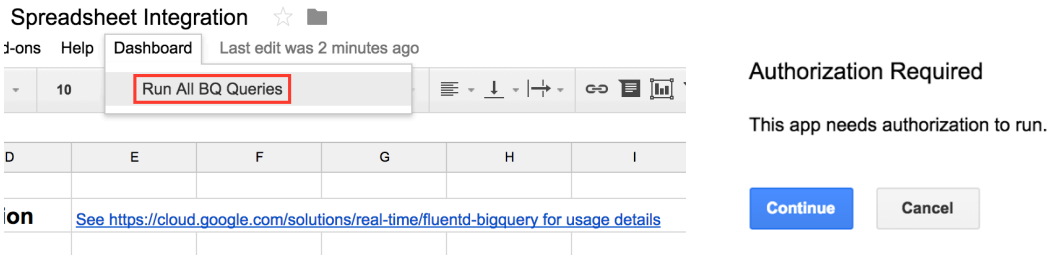



Open the spreadsheet and put in the SQL statements like the example. The first column is the name of the generated result table and chart type. We want to display the popularity of CDN providers in columns, so we use “COLUMN” here. The second column is the querying interval, it will be used to execute the query automatically and regularly. The third column is the real SQL statement. The last one is the finish time of the query.

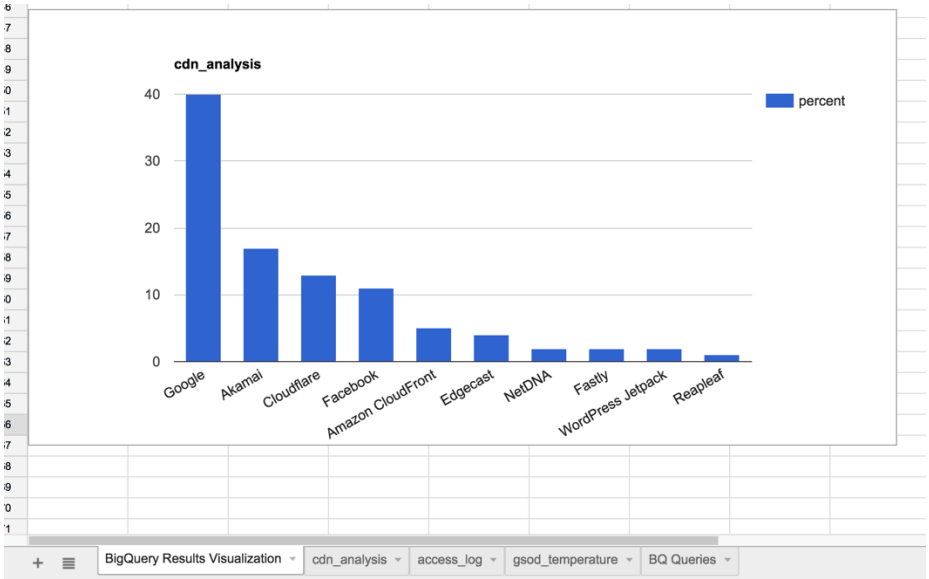
| | | | |
|---|---|---|---|
| cdn_analysis_COLUMN | 0 | SELECT provider, ROUND(100*ratio) AS percent, num FROM (<br>SELECT<br>_cdn_provider AS provider, COUNT(*) AS num, RATIO_TO_REPORT(num) OVER() ratio<br>FROM<br>httparchive:runs.2015_12_01_requests,<br>httparchive:runs.2015_11_15_requests,<br>httparchive:runs.2015_11_01_requests,<br>httparchive:runs.2015_10_15_requests,<br>httparchive:runs.2015_12_01_requests_mobile,<br>httparchive:runs.2015_11_15_requests_mobile,<br>httparchive:runs.2015_11_01_requests_mobile,<br>httparchive:runs.2015_10_15_requests_mobile<br>WHERE<br>_cdn_provider != ''<br>GROUP BY provider )<br>ORDER BY num DESC<br>LIMIT 10 | 12/13 14:09:16 |

```
// determine chart properties from the tag suffixes
var chartType = null;
var startColumn = 1;
var rowSize = MAX_ROWS_LARGE;
var isStacked = false;
var matched = tag.match(/(.*)_(AREA|BAR|COLUMN|LINE|SCATTER|TABLE)(_STACKED)?/);
if (matched) {
  tableSheetName = matched[1];
  isStacked = matched[3] != null;
  if (matched[2] == "AREA") {
    chartType = Charts.ChartType.AREA;
  } else if (matched[2] == "BAR") {
    chartType = Charts.ChartType.BAR;
    rowSize = MAX_ROWS_SMALL;
  } else if (matched[2] == "COLUMN") {
    chartType = Charts.ChartType.COLUMN;
    rowSize = MAX_ROWS_SMALL;
  } else if (matched[2] == "LINE") {
    chartType = Charts.ChartType.LINE;
  } else if (matched[2] == "SCATTER") {
    chartType = Charts.ChartType.SCATTER;
  } else if (matched[2] == "TABLE") {
    chartType = Charts.ChartType.TABLE;
  }
}
```

Select “Dashboard -> Run All BQ Queries”. A dialog box with the message "Authorization Required" appears the first time you run BigQuery. Click Continue and Accept.




After the query finishes executing, you'll see a new sheet named “cdn_analysis”. Open this sheet and check that it contains query results. Open the BigQuery Results Visualization sheet and check that it contains the following embedded column chart of the CDN providers data that was returned by the query. We can do this for all the other queries too.



2) Use the “bq Command-Line Tool” to analyze the data.
Reference: https://cloud.google.com/bigquery/bq-command-line-tool-quickstart

2.1. Activate the BigQuery service with a Google APIs Console project:
https://cloud.google.com/bigquery/sign-up
Install the Google Cloud SDK: https://cloud.google.com/sdk/

```
hqiu@bos-mpdei>> curl https://sdk.cloud.google.com | bash
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   421    0   421    0     0    751      0 --:--:-- --:--:-- --:--:--  3422
Downloading Google Cloud SDK install script: https://dl.google.com/dl/cloudsdk/channels/rapid/install_g
oogle_cloud_sdk.bash
######################################################################## 100.0%
Running install script from: /var/folders/b7/3fvccjfn62b6gd2t2jdsv57r0014lp/T/tmp.XXXXXXXXXX.tQokluat/i
nstall_google_cloud_sdk.bash
which curl
curl -# -f https://dl.google.com/dl/cloudsdk/channels/rapid/google-cloud-sdk.tar.gz
######################################################################## 100.0%
```

```
hqiu@bos-mpdei>> gcloud init
Welcome! This command will take you through the configuration of gcloud.

Your current configuration has been set to: [default]

To continue, you must login. Would you like to login (Y/n)?  Y

Your browser has been opened to visit:

    https://accounts.google.com/o/oauth2/auth?redirect_uri=http%3A%2F%2Flocalhost%3A8085%2F&prompt=sele
ct_account&response_type=code&client_id=32555940559.apps.googleusercontent.com&scope=https%3A%2F%2Fwww.
googleapis.com%2Fauth%2Fuserinfo.email+https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fcloud-platform+https%3
A%2F%2Fwww.googleapis.com%2Fauth%2Fappengine.admin+https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fcompute&ac
cess_type=offline


You are now logged in as: [hanjiaoqiu@g.harvard.edu]

Enter project id you would like to use:  friendly-path-115605
Your current project has been set to: [friendly-path-115605].
```

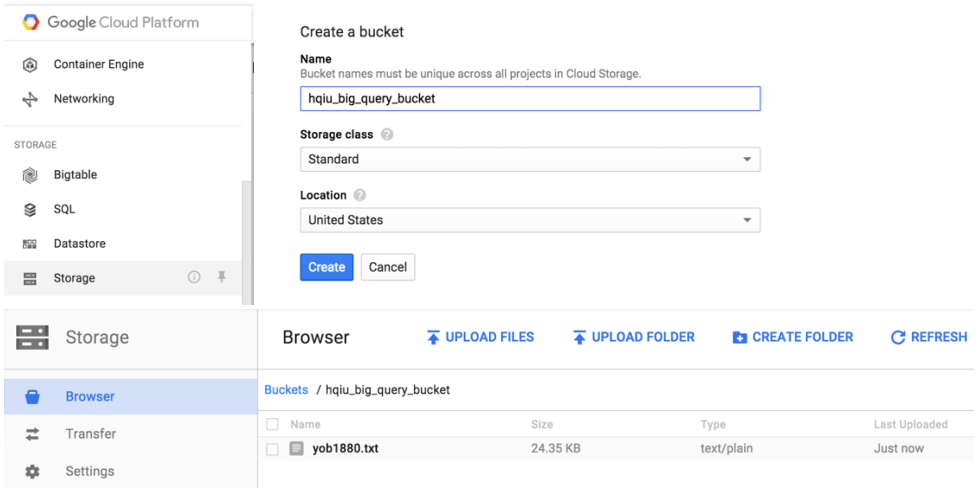
2.2. Create table, schema, load data and execute queries through command-line tool.
First show the available datasets to verify our installation. Also show the schema of the Google public data samples. Then make a simple query against the sample data.

```
hqiu@bos-mpdei>> bq shell
Welcome to BigQuery! (Type help for more information.)
friendly-path-115605> ls
     datasetId
 -------------------
  my_children_names
friendly-path-115605> exit
Goodbye.
```

```
hqiu@bos-mpdei>> bq show publicdata:samples.shakespeare
Table publicdata:samples.shakespeare

   Last modified                Schema                Total Rows   Total Bytes   Expiration
 ----------------- ---------------------------------- ------------ ------------ ------------
  26 Aug 17:43:49   |- word: string (required)         164656       6432064
                    |- word_count: integer (required)
                    |- corpus: string (required)
                    |- corpus_date: integer (required)
```

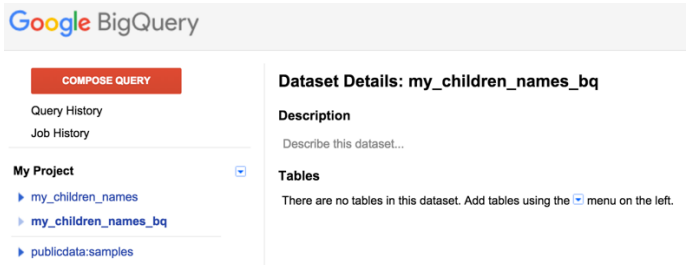
```
hqiu@bos-mpdei>> bq query "SELECT word, COUNT(word) as count FROM publicdata:samples.shakespeare WHERE
word CONTAINS 'raisin' GROUP BY word"
Waiting on bqjob_r38d32b35230ce865_00000151927b78af_1 ... (0s) Current status: DONE
+--------------+-------+
|     word     | count |
+--------------+-------+
| raising      |     5 |
| dispraising  |     2 |
| Praising     |     4 |
| praising     |     7 |
| dispraisingly |     1 |
| raisins      |     1 |
+--------------+-------+
```

2.3. Upload the csv file to the Google Cloud Storage. Load the dataset from Google Storage to BigQuery and query it. Since huge dataset takes long time to upload, I'll just use a small and simple dataset to show that we can create table, load data and make query via CLI. First enable Google Storage for the project. Then go to the "Google Cloud Storage" pane, upload the file to Google Cloud Storage. Like AWS S3, create a bucket.




Create a new dataset, it will also be showed from the console.

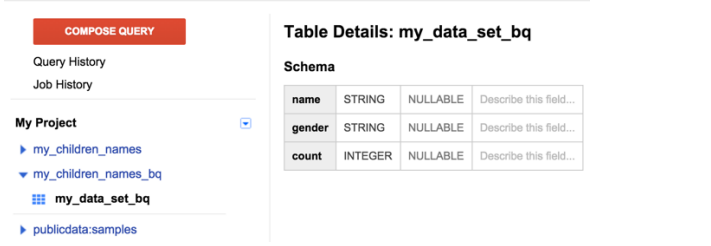
```
hqiu@bos-mpdei>> bq mk my_children_names_bq
Dataset 'friendly-path-115605:my_children_names_bq' successfully created.
hqiu@bos-mpdei>> bq ls
       datasetId
 ----------------------
  my_children_names
  my_children_names_bq
```




Upload the table with schema. We can check the schema by calling "bs show".

```
hqiu@bos-mpdei>> bq load --source_format CSV my_data_set_bq gs://hqiu_big_query_bucket/yob1880.txt name
:string,gender:string,count:integer
BigQuery error in load operation: Cannot determine table described by my_data_set_bq
hqiu@bos-mpdei>> bq load --source_format CSV my_children_names_bq.my_data_set_bq gs://hqiu_big_query_bu
cket/yob1880.txt name:string,gender:string,count:integer
Waiting on bqjob_r539a33511ffd9950_000001519294ae02_1 ... (34s) Current status: DONE
```

```
hqiu@bos-mpdei>> bq ls my_children_names_bq
     tableId       Type
 ---------------- -------
  my_data_set_bq   TABLE
hqiu@bos-mpdei>>
hqiu@bos-mpdei>> bq show my_children_names_bq.my_data_set_bq
Table friendly-path-115605:my_children_names_bq.my_data_set_bq

   Last modified         Schema         Total Rows   Total Bytes   Expiration
 ----------------- -------------------- ------------ ------------- ------------
  11 Dec 14:47:07   |- name: string     2000         37400
                    |- gender: string
                    |- count: integer
```



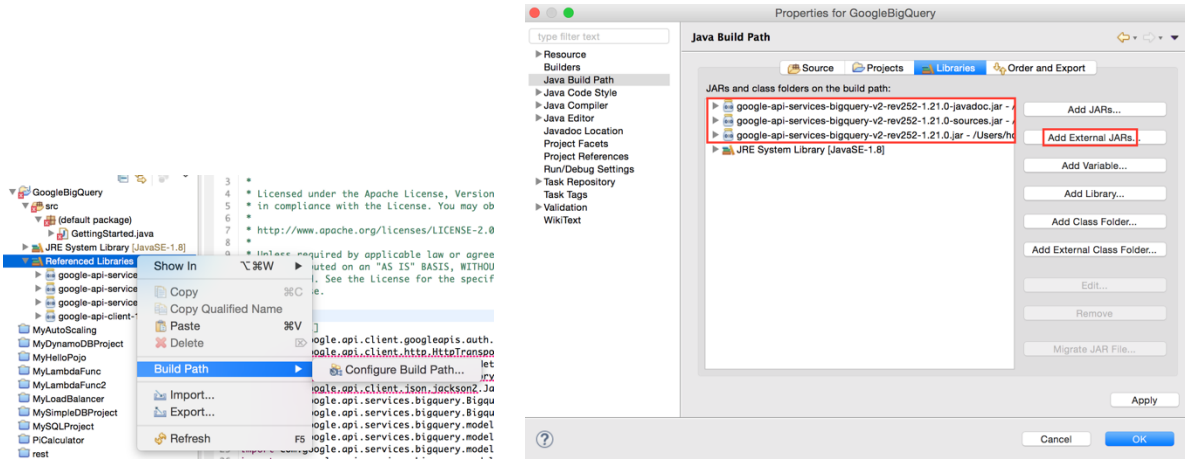

Query against the data.

```
hqiu@bos-mpdei>> bq query "SELECT name,count FROM my_children_names_bq.my_data_set_bq WHERE gender = 'F
' ORDER BY count DESC LIMIT 5"
Waiting on bqjob_r317289c0b8af0424_000001519297591d_1 ... (0s) Current status: DONE
+-----------+-------+
|   name    | count |
+-----------+-------+
| Mary      |  7065 |
| Anna      |  2604 |
| Emma      |  2003 |
| Elizabeth |  1939 |
| Minnie    |  1746 |
+-----------+-------+
```

In this way, we can query any dataset from the “bq command line” tool.

3) Use the “BigQuery API” to analyze the data.
Reference: https://cloud.google.com/bigquery/bigquery-api-quickstart

3.1. Download the Google BigQuery API Client Library for Java. (The BigQuery service provides a REST-based API that can be programmatically accessed using Java, Python, C# and any other language) Download the Client libraries from: https://cloud.google.com/bigquery/client-libraries. Import all the libraries into our project.



3.2. Import data from local file and Google cloud storage, query data and export results in Java.
Java code examples:
https://github.com/GoogleCloudPlatform/java-docs-samples/tree/master/bigquery/src/main/java/com/google/cloud/bigquery/samples

First start with “GettingStarted.java”. Make a simple query against the data “httparchive:runs”.

The main function:

```
        public static void main(String[] args) throws IOException {
                String projectId = "friendly-path-115605";
                System.out.println("projectId: " + projectId + "\n");

                // Create a new Bigquery client authorized via Application Default Credentials.
                Bigquery bigquery = createAuthorizedClient();

                String queryString = "SELECT provider, ROUND(100*ratio) AS percent, num FROM ( \n" +
                                     "  SELECT _cdn_provider AS provider, COUNT(*) AS num,
RATIO_TO_REPORT(num) OVER() ratio \n" +
                                     "  FROM httparchive:runs.2015_12_01_requests, \n" +
                                     "       httparchive:runs.2015_11_15_requests, \n" +
                                     "       httparchive:runs.2015_11_01_requests, \n" +
                                     "       httparchive:runs.2015_10_15_requests, \n" +
                                     "       httparchive:runs.2015_12_01_requests_mobile, \n" +
                                     "       httparchive:runs.2015_11_15_requests_mobile, \n" +
                                     "       httparchive:runs.2015_11_01_requests_mobile, \n" +
                                     "       httparchive:runs.2015_10_15_requests_mobile \n" +
                                     "  WHERE _cdn_provider != '' GROUP BY provider ) ORDER BY num DESC
LIMIT 10";
                System.out.println("Query: \n" + queryString + "\n");

                List<TableRow> rows = executeQuery(queryString, bigquery, projectId);

                printResults(rows);
        }
```

Like AWS, we need to create the credentials for connection first.

```
        public static Bigquery createAuthorizedClient() throws IOException {
                // Create the credential
                HttpTransport transport = new NetHttpTransport();
                JsonFactory jsonFactory = new JacksonFactory();
                GoogleCredential credential = GoogleCredential.getApplicationDefault(transport,
jsonFactory);

                // Depending on the environment that provides the default credentials (e.g. Compute
Engine, App
                // Engine), the credentials may require us to specify the scopes we need explicitly.
                // Check for this case, and inject the Bigquery scope if required.
                if (credential.createScopedRequired()) {
                        credential = credential.createScoped(BigqueryScopes.all());
                }

                return new Bigquery.Builder(transport, jsonFactory, credential)
                                .setApplicationName("Bigquery Samples").build();
        }
```

The real executeQuery() function. It basically creates the query request with the query string, wraps up the query and inserts it into a job list. Then execute the job.

```
        private static List<TableRow> executeQuery(String querySql, Bigquery bigquery, String
projectId) throws IOException {
                QueryResponse query = bigquery.jobs().query(
                                projectId,
                                new QueryRequest().setQuery(querySql))
                                .execute();

                // Execute it
                GetQueryResultsResponse queryResult = bigquery.jobs().getQueryResults(
                                query.getJobReference().getProjectId(),
                                query.getJobReference().getJobId()).execute();

                return queryResult.getRows();
        }
```

The results:

```
<terminated> GettingStarted [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_45.jdk/Contents/Home/bin/java (Dec 13, 2015, 5:16:28 PM)
projectId: friendly-path-115605

Query:
SELECT provider, ROUND(100*ratio) AS percent, num FROM (
  SELECT _cdn_provider AS provider, COUNT(*) AS num, RATIO_TO_REPORT(num) OVER() ratio
  FROM httparchive:runs.2015_12_01_requests,
       httparchive:runs.2015_11_15_requests,
       httparchive:runs.2015_11_01_requests,
       httparchive:runs.2015_10_15_requests,
       httparchive:runs.2015_12_01_requests_mobile,
       httparchive:runs.2015_11_15_requests_mobile,
       httparchive:runs.2015_11_01_requests_mobile,
       httparchive:runs.2015_10_15_requests_mobile
  WHERE _cdn_provider != '' GROUP BY provider ) ORDER BY num DESC LIMIT 10


Query Results:
------------
Google                                  40.0                                    28997532
Akamai                                  17.0                                    12217367
Cloudflare                              13.0                                    9079980
Facebook                                11.0                                    7704510
Amazon CloudFront                       5.0                                     3857754
Edgecast                                4.0                                     2664794
NetDNA                                  2.0                                     1627192
Fastly                                  2.0                                     1574370
WordPress Jetpack                       2.0                                     1115876
Reapleaf                                1.0                                     570609
```

### 3.3. Load data from Google Cloud Storage and local CSV file. (LoadDataCsvSample.java)

Create the job to load data from local CSV file:

```
public static Job loadJobFromCsv(
                final Bigquery bigquery,
                final String csvfile,
                final TableReference table,
                final TableSchema schema) throws IOException {

        FileContent content = new FileContent("application/octet-stream", new File(csvfile));

        JobConfigurationLoad load = new JobConfigurationLoad()
                        .setDestinationTable(table)
                        .setSchema(schema)
                        .setSourceFormat("csv");

        return bigquery.jobs().insert(table.getProjectId(),
                        new Job().setConfiguration(new JobConfiguration().setLoad(load)), content)
                        .execute();
}
```

Create the job to load data from Google Cloud Storage:

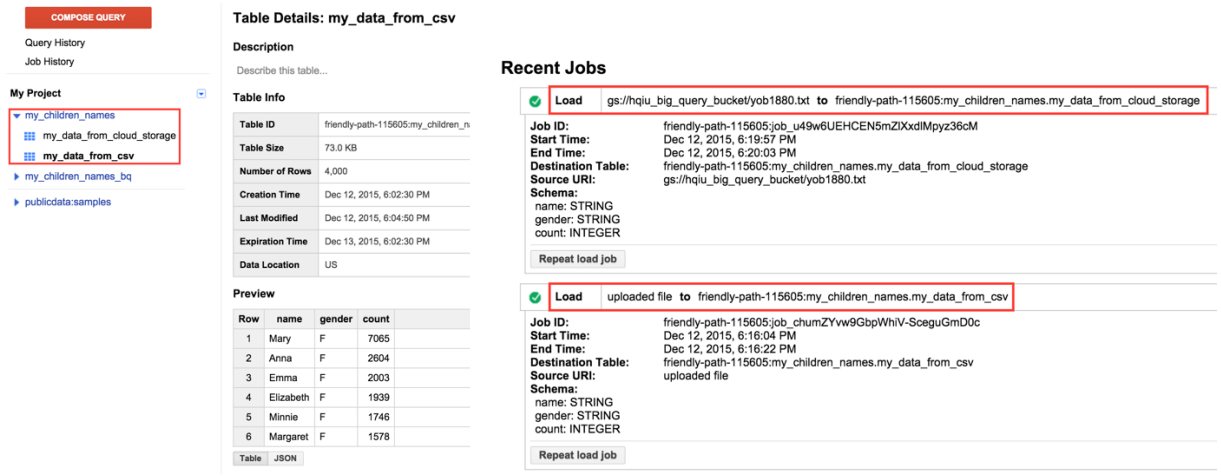
```
public static Job loadJobFromCloudStorage(
                final Bigquery bigquery,
                final String cloudStoragePath,
```

```
                final TableReference table,
                final TableSchema schema) throws IOException {

        JobConfigurationLoad load = new JobConfigurationLoad()
                        .setDestinationTable(table)
                        .setSchema(schema)
                        .setSourceUris(Collections.singletonList(cloudStoragePath));

        return bigquery.jobs().insert(table.getProjectId(),
                        new Job().setConfiguration(new JobConfiguration().setLoad(load)))
                        .execute();
}
```

The example input schema file:

```
[
  {"name": "name", type": "STRING"},
  {"name": "gender", "type": "STRING"},
  {"name": "count","type": "INTEGER"}
]
```

Results from the Google console. In this way, we can upload any dataset and process using Java.



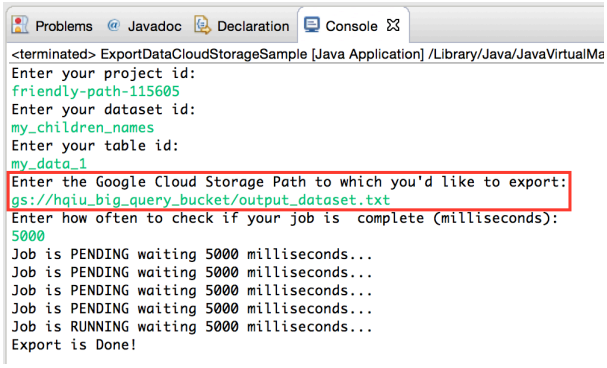

The whole Java code is available from LoadDataCsvSample.java, I just extract the core part here.

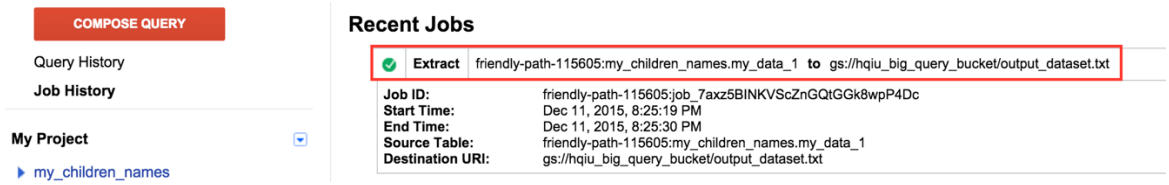
3.4. Export result data to Google Cloud storage. (ExportDataCloudStorageSample.java)

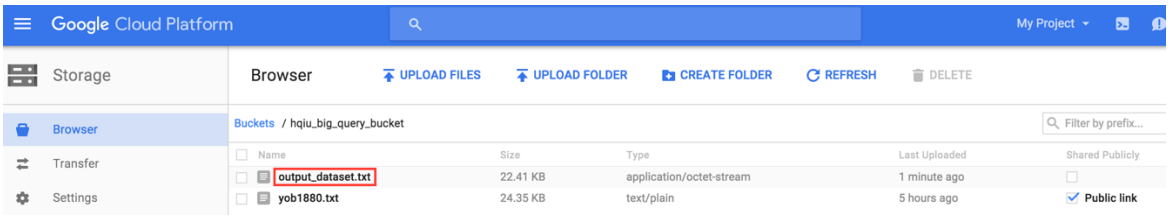
3.2 ~ 3.4 composes the whole dataflow to load data, query data and export data to file.

```
    public static Job extractJob(
                        final Bigquery bigquery,
                        final String cloudStoragePath,
                        final TableReference table) throws IOException {

                JobConfigurationExtract extract = new JobConfigurationExtract()
                                .setSourceTable(table)
                                .setDestinationUri(cloudStoragePath);

                return bigquery.jobs().insert(table.getProjectId(),
                                new Job().setConfiguration(new JobConfiguration().setExtract(extract)))
                                .execute();
        }
```

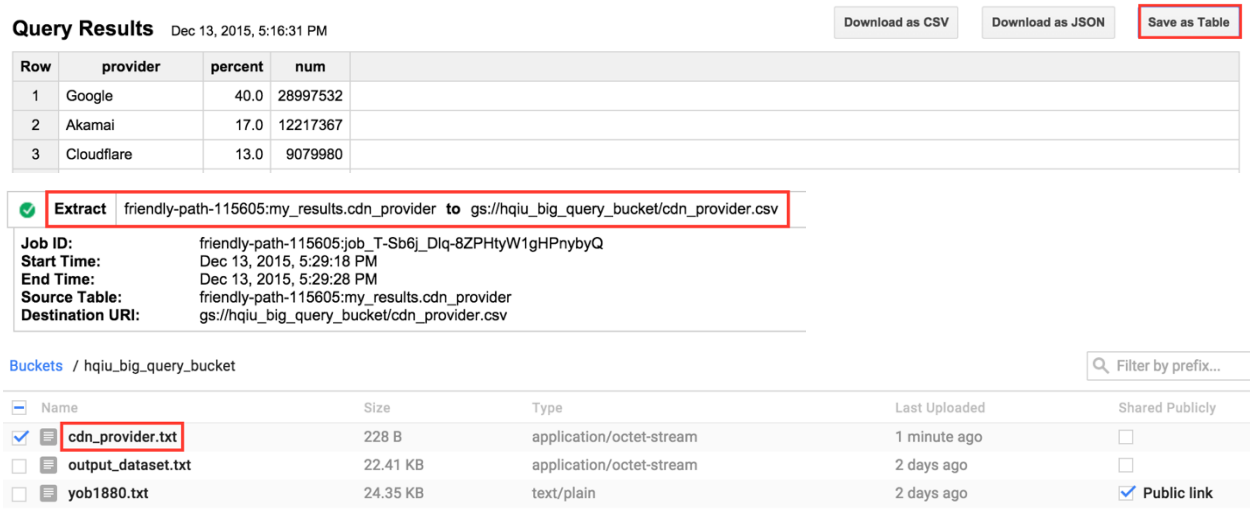
Results from Eclipse console and Google console:




We can use this program to export our previous results into Google Cloud Storage. Just first save it as table and then upload the table directly to the Google Cloud Storage.



3.5. Synchronized query. (SyncQuerySample.java) Synchronous queries are handled by a job, as all queries are, and will appear in the jobs history. The job will continue to run until it finishes, and you will hang on there.

```
public static Iterator<GetQueryResultsResponse> run(final String projectId,
                final String queryString,
                final long waitTime) throws IOException {
        Bigquery bigquery = BigqueryServiceFactory.getService();
        // Wait until query is done with 10 second timeout, at most 5 retries on error
        QueryResponse query = bigquery.jobs().query(
                        projectId,
                        new QueryRequest().setTimeoutMs(waitTime).setQuery(queryString))
                        .execute();

        GetQueryResults getRequest = bigquery.jobs().getQueryResults(
                        query.getJobReference().getProjectId(),
                        query.getJobReference().getJobId());

        return BigqueryUtils.getPages(getRequest);
}
```

3.6. Asynchronized query. (AsyncQuerySample.java) Asynchronous queries are run by calling “bigquery.jobs.insert”. The method returns immediately with a job ID, and you must either request the job status periodically and check status, or call “jobs.getQueryResults”, which will return when the job is complete.

```
public static Job asyncQuery(final Bigquery bigquery,
                final String projectId,
                final String querySql,
                final boolean batch) throws IOException {

        JobConfigurationQuery queryConfig = new JobConfigurationQuery()
                        .setQuery(querySql);

        if (batch) {
                queryConfig.setPriority("BATCH");
        }

        Job job = new Job().setConfiguration(
                        new JobConfiguration().setQuery(queryConfig));

        return bigquery.jobs().insert(projectId, job).execute();
}
```

```
public static Iterator<GetQueryResultsResponse> run(final String projectId,
                final String queryString,
                final boolean batch,
                final long waitTime)
                throws IOException, InterruptedException {

        Bigquery bigquery = BigqueryServiceFactory.getService();

        Job query = asyncQuery(bigquery, projectId, queryString, batch);
        Bigquery.Jobs.Get getRequest = bigquery.jobs().get(
                        projectId, query.getJobReference().getJobId());

        // Poll every waitTime milliseconds, retrying at most retries times if there are errors
        pollJob(getRequest, waitTime);

        GetQueryResults resultsRequest = bigquery.jobs().getQueryResults(
                        projectId, query.getJobReference().getJobId());
```

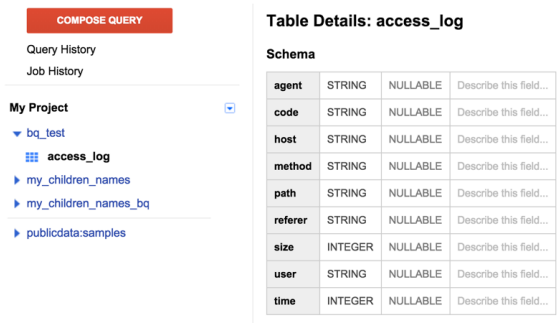
```
            return getPages(resultsRequest);
        }
```

4) Real-time logs analysis using Fluentd and BigQuery.
Reference: https://cloud.google.com/solutions/real-time/fluentd-bigquery

4.1. Create Google Compute Engine instance and load data using Fluentd.
First clone the GitHub repository and create a dataset and bigQuery table.

```
hqiu@bos-mpdei>> git clone https://github.com/GoogleCloudPlatform/bigquery-fluentd-docker-sample
Cloning into 'bigquery-fluentd-docker-sample'...
remote: Counting objects: 24, done.
remote: Total 24 (delta 0), reused 0 (delta 0), pack-reused 24
Unpacking objects: 100% (24/24), done.
Checking connectivity... done.
```

```
hqiu@bos-mpdei>> bq mk bq_test
Dataset 'friendly-path-115605:bq_test' successfully created.
hqiu@bos-mpdei>> cd bigquery-fluentd-docker-sample
hqiu@bos-mpdei>> bq mk -t bq_test.access_log ./schema.json
Table 'friendly-path-115605:bq_test.access_log' successfully created.
```




Create a Google compute Engine instance.

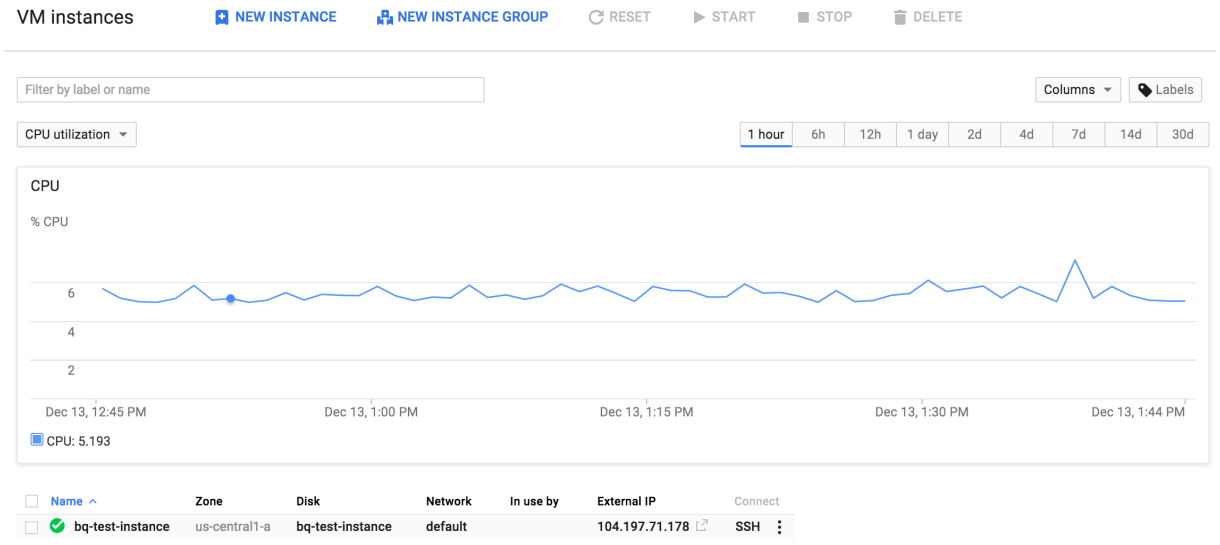
```
hqiu@bos-mpdei>> gcloud compute instances create "bq-test-instance" --zone "us-central1-a" --machine-ty
pe "n1-standard-1" --scopes storage-ro,bigquery --image container-vm
Created [https://www.googleapis.com/compute/v1/projects/friendly-path-115605/zones/us-central1-a/instan
ces/bq-test-instance].
NAME             ZONE          MACHINE_TYPE  PREEMPTIBLE INTERNAL_IP EXTERNAL_IP     STATUS
bq-test-instance us-central1-a n1-standard-1             10.240.0.2  104.197.71.178 RUNNING
```

Run ngnix and Fluentd in a Docker container.

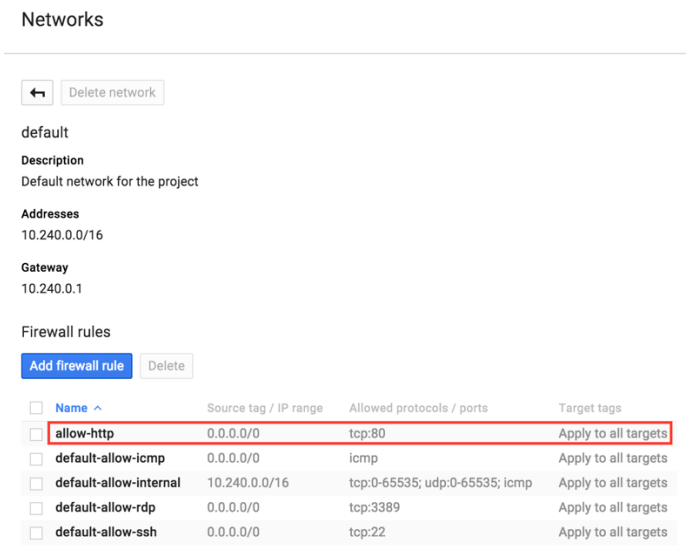
```
hqiu@bos-mpdei>> gcloud compute ssh bq-test-instance --zone=us-central1-a
WARNING: You do not have an SSH key for Google Compute Engine.
WARNING: [/usr/bin/ssh-keygen] will be executed to generate a key.
Generating public/private rsa key pair.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /Users/hqiu/.ssh/google_compute_engine.
Your public key has been saved in /Users/hqiu/.ssh/google_compute_engine.pub.
The key fingerprint is:
28:6a:94:87:5b:a5:ad:67:84:00:0b:7b:ae:22:9a:c6 hqiu@bos-mpdei
The key's randomart image is:
+--[ RSA 2048]----+
|o                |
|.+               |
|o o    .         |
| o + = .         |
|  = * + S        |
| o = +           |
|= + . o          |
|+E   o           |
|+                |
+-----------------+
Updated [https://www.googleapis.com/compute/v1/projects/friendly-path-115605].
Warning: Permanently added '104.197.71.178' (RSA) to the list of known hosts.
Warning: Permanently added '104.197.71.178' (RSA) to the list of known hosts.
Linux bq-test-instance 3.16.0-0.bpo.4-amd64 #1 SMP Debian 3.16.7-ckt11-1+deb8u4~bpo70+1 (2015-09-22) x8
6_64
```

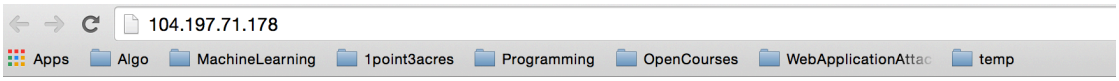
```
hqiu@bq-test-instance:~$ sudo docker run -e GCP_PROJECT="friendly-path-115605" -p 80:80 -t -i -d google
/fluentd-bigquery-sample
Unable to find image 'google/fluentd-bigquery-sample:latest' locally
latest: Pulling from google/fluentd-bigquery-sample
d634beec75db: Pull complete
dce3cd94bfae: Pull complete
ee729b484402: Pull complete
9da7fdb2c125: Pull complete
351d02002a45: Pull complete
0f0209a8846d: Pull complete
f4cd9cea67ba: Pull complete
2d029826ca63: Pull complete
e81e68b9f661: Pull complete
0413fcc94127: Pull complete
cd8019ab928b: Pull complete
66ef38ff6d3e: Pull complete
f2a4da5fe47d: Pull complete
48be486d78ad: Pull complete
0a2d07ba6114: Pull complete
0f7cb9c1d21e: Pull complete
4070a34ea03f: Pull complete
b7d433f9758e: Pull complete
ddbf05223fa5: Pull complete
5221fe752d26: Pull complete
3db64a0fae53: Pull complete
ca46767a38aa: Pull complete
Digest: sha256:332a18b6aa4b13d65b63c6a4040d334ffd97097c25f38939f892226de9ebbe79
Status: Downloaded newer image for google/fluentd-bigquery-sample:latest
f6d881515da221c593e33e1f6cee5d8bbec4df097a80bd2b2c9ba4cac9e01f69
hqiu@bq-test-instance:~$ sudo docker ps -a
CONTAINER ID        IMAGE                             COMMAND                  CREATED             STATU
S                   PORTS                  NAMES
f6d881515da2        google/fluentd-bigquery-sample    "/bin/sh -c '/etc/ini"   11 seconds ago      Up 10
 seconds            0.0.0.0:80->80/tcp     compassionate_banach
```

We can check the VM instances through Google console.




There's a pop up window right beside the External IP (104.197.71.178). Click on it and we can find we couldn't remote login to the instance. Check the Network firewall rules setting. Add a firewall rule to allow http traffic on "tcp:80". Now we can login through the web browser!




**Welcome to nginx!**

Use the Apache Bench tool to generate simulate some load to the nginx server.

```
hqiu@bos-mpdei>> ab -c 20 -n 1000000 http://104.197.71.178/
This is ApacheBench, Version 2.3 <$Revision: 1663405 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 104.197.71.178 (be patient)
Completed 100000 requests
Completed 200000 requests
Completed 300000 requests
Completed 400000 requests
Completed 500000 requests
```

```
Server Software:        nginx/1.1.19
Server Hostname:        104.197.71.178
Server Port:            80

Document Path:          /
Document Length:        151 bytes

Concurrency Level:      20
Time taken for tests:   3639.959 seconds
Complete requests:      987940
Failed requests:        0
Total transferred:      357634280 bytes
HTML transferred:       149178940 bytes
Requests per second:    271.42 [#/sec] (mean)
Time per request:       73.688 [ms] (mean)
Time per request:       3.684 [ms] (mean, across all concurrent requests)
Transfer rate:          95.95 [Kbytes/sec] received
```

**Table Details: access_log**

**Description**

Describe this table...

**Table Info**

| | |
|---|---|
| **Table ID** | friendly-path-115605:bq_test.access_log |
| **Table Size** | 77.3 MB |
| **Number of Rows** | 1,209,295 |
| **Creation Time** | Dec 13, 2015, 11:30:28 AM |
| **Last Modified** | Dec 13, 2015, 3:11:21 PM |
| **Data Location** | US |

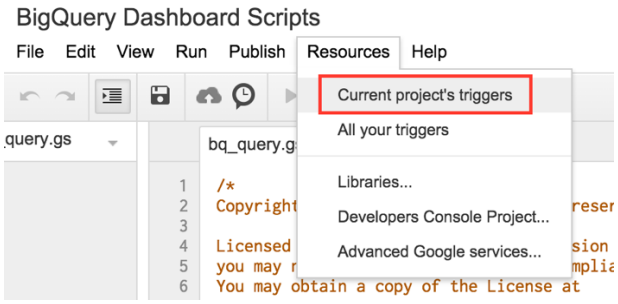
**Preview**

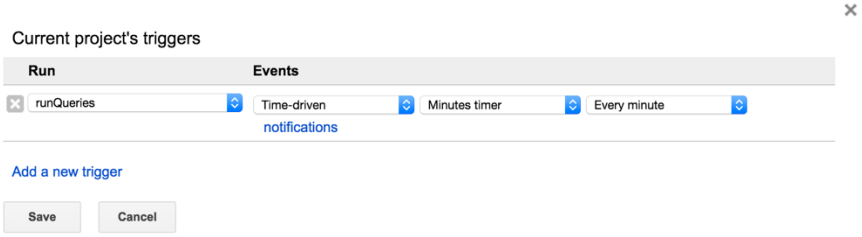
| Row | agent | code | host | method | path | referer | size | user | time |
|---|---|---|---|---|---|---|---|---|---|
| 1 | ApacheBench/2.3 | 200 | 192.54.222.20 | GET | / | - | 151 | - | 1450036802 |
| 2 | ApacheBench/2.3 | 200 | 192.54.222.20 | GET | / | - | 151 | - | 1450036800 |
| 3 | ApacheBench/2.3 | 200 | 192.54.222.20 | GET | / | - | 151 | - | 1450036802 |
| 4 | ApacheBench/2.3 | 200 | 192.54.222.20 | GET | / | - | 151 | - | 1450036800 |
| 5 | ApacheBench/2.3 | 200 | 192.54.222.20 | GET | / | - | 151 | - | 1450036802 |

Table JSON

Check the “access_log” table, we can see that all system logs have been populated to the table!

4.2. Use BigQuery and visualize data in Google sheet.
Put the SQL statement in the spreadsheet and create a project trigger.



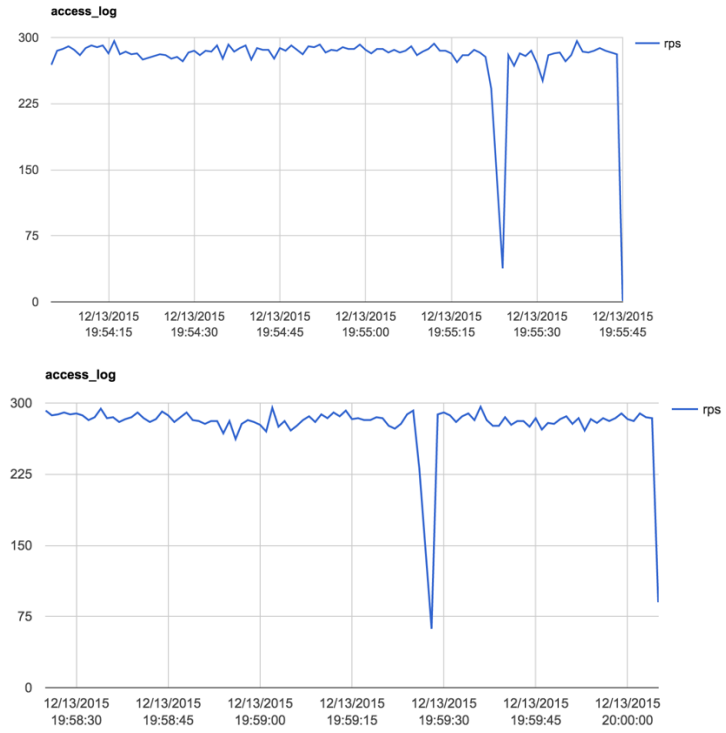




query name

| | A | B | C | D |
|---|---|---|---|---|
| | query name | interval (min) | query | last execution time |
| | gsod_temperature_LINE | 0 | SELECT<br>STRING(year) + '-' + LPAD(STRING(month), 2, '0') AS time,<br>AVG(mean_temp) AS temp_avg,<br>MIN(mean_temp) AS temp_min,<br>MAX(mean_temp) AS temp_max<br>FROM [publicdata:samples.gsod]<br>GROUP BY time ORDER BY time ASC LIMIT 100 | 12/13 11:55:45 |
| | access_log_LINE | 1 | SELECT<br>STRFTIME_UTC_USEC(time * 1000000, "%Y-%m-%d %H:%M:%S") as tstamp,<br>count(*) as rps<br>FROM bq_test.access_log<br>GROUP BY tstamp ORDER BY tstamp DESC; | 12/13 12:09:56 |

Navigate to “Tools > Script editor” and select “Resources > Current project's triggers”. You'll see that no triggers have been set up yet. Click the link to add a trigger. Select "runQueries" from the Run menu, and for Events, select “Time-driven”, “Minutes timer”, and “Every minute”. Click Save. This triggers the script bq_query.gs to run once per minute.

Return to the BigQuery Results Visualization sheet and you'll see the "access_log" graph refresh every minute.




## Summary

We have demonstrated three ways to query massive dataset using Google BigQuery. Google BigQuery enables us to solve the problems super-fast, which queries terabytes of data in seconds. We have many interesting findings such as popular CDN providers and web performance through processing the HTTP Archive data using BigQuery. It also provides great help to process machine generated records such as system log, which is pretty useful. BigQuery is also fully integrated with other services and tools, we should practice and make use of them. We will definitely have more interesting findings!

## References


https://cloud.google.com/bigquery/what-is-bigquery

https://cloud.google.com/bigquery/third-party-tools

https://github.com/GoogleCloudPlatform/java-docs-samples/tree/master/bigquery/src/main/java/com/google/cloud/bigquery/samples

https://developers.google.com/resources/api-libraries/documentation/bigquery/v2/java/latest/

http://httparchive.org/interesting.php

http://bigqueri.es/t/whats-the-popularity-of-different-cdns/477

http://bigqueri.es/t/which-sites-have-content-security-policy-to-prevent-xss/589

## Links to Youtube videos

Short video:
https://youtu.be/LaEz-mcvIKk

Full video:
https://youtu.be/_zN6Vjo29yU