Article
Ron Sweeney · Nov 7, 2019 5m read

Open Exchange

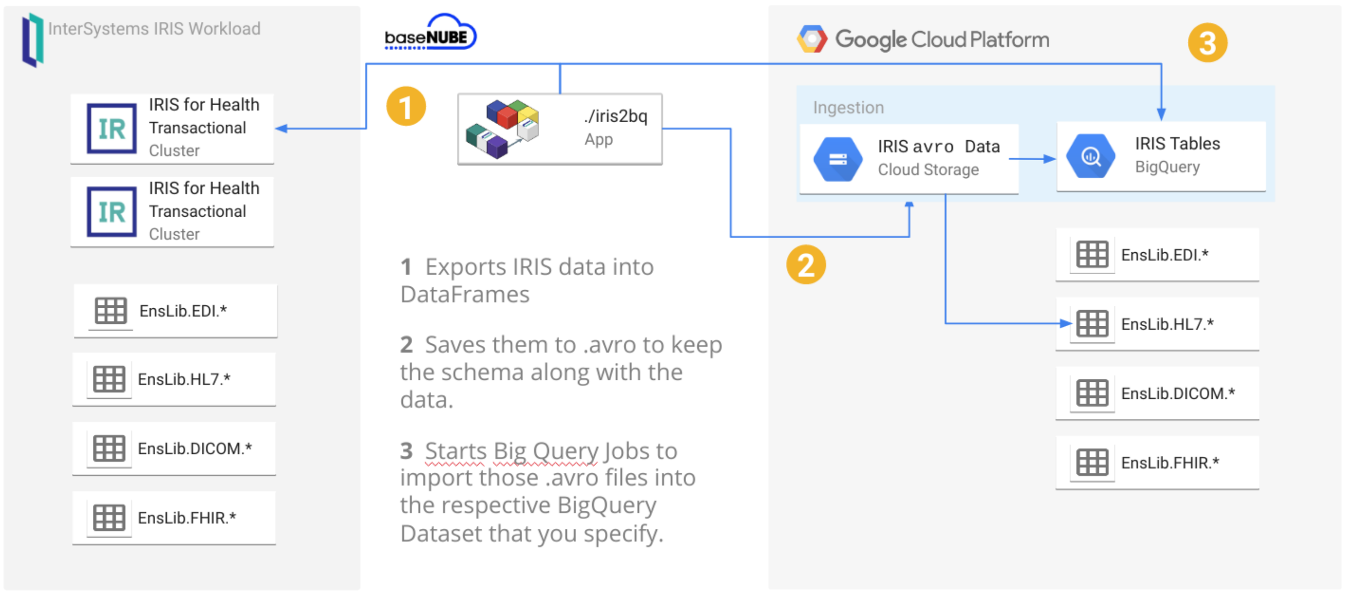
Export InterSystems IRIS Data to BigQuery on Google Cloud Platform

Loading your IRIS Data to your Google Cloud Big Query Data Warehouse and keeping it current can be a hassle with bulky Commercial Third Party Off The Shelf ETL platforms, but made dead simple using the iris2bq utility.

Let's say IRIS is contributing to workload for a Hospital system, routing DICOM images, ingesting HL7 messages, posting FHIR resources, or pushing CCDA's to next provider in a transition of care. Natively, IRIS persists these objects in various stages of the pipeline via the nature of the business processes and anything you included along the way. Lets send that up to Google Big Query to augment and compliment the rest of our Data Warehouse data and ETL (Extract Transform Load) or ELT (Extract Load Transform) to our hearts desire.

A reference architecture diagram may be worth a thousand words, but 3 bullet points may work out a little bit better:

- It exports the data from IRIS into DataFrames
- It saves them into GCS as .avro to keep the schema along the data: this will avoid to specify/create the BigQuery table schema beforehands.
- It starts BigQuery jobs to import those .avro into the respective BigQuery tables you specify.


InterSystems IRIS Workload
IRIS for Health Transactional Cluster
IRIS for Health Transactional Cluster
EnsLib.EDI.*
EnsLib.HL7.*
EnsLib.DICOM.*
EnsLib.FHIR.*
baseNUBE
1
./iris2bq App
1 Exports IRIS data into DataFrames
2 Saves them to .avro to keep the schema along with the data.
3 Starts Big Query Jobs to import those .avro files into the respective BigQuery Dataset that you specify.
Google Cloud Platform
3
2
Ingestion
IRIS avro Data Cloud Storage
IRIS Tables BigQuery
EnsLib.EDI.*
EnsLib.HL7.*
EnsLib.DICOM.*
EnsLib.FHIR.*

Under the hood, iris2bq it is using the Spark framework for the sake of simplicity, but no Hadoop cluster is needed. It is configured as a "local" cluster by default, meaning the application and is running standalone. The tool is meant to be launched on an interval either through cron or something like Airflow.

All you have to do is point it at your IRIS instance, tell it what tables you want to sync to Big Query, then they magically sync to an existing dataset or a creates a new one that you specify.

How To Setup

And if a reference architecture and 3 bullet points didn't do a good job explaining it, maybe actually running it will:

Google Cloud Setup

You can do this anyway you want, here are a few options for you, but all you have to do in GCP is:

- Create a Project
- Enable the API's of Big Query and Cloud Storage
- Create a service account with access to create resources and download the json file.

Using the Google Cloud Console (Easiest)

https://cloud.google.com

Using gcloud (Impress Your Friends):

```
gcloud projects create iris2bq-demo--enable-cloud-apis
```

With Terraform (Coolest):

Create a main.tf file after modifying the values:

```
// Create the GCP Project
resource "google_project" "gcp_project" {
  name = "IRIS 2 Big Query Demo"
  project_id = "iris2bq-demo"
  // You'll need this
  org_id     = "1234567"
}
// Enable the APIS
resource "google_project_services" "gcp_project_apis" {
  project = "iris2bq-demo"
  services   = ["bigquery.googleapis.com", "storage.googleapis.com"]
}
```

Then do a:

```
terraform init
terraform plan
terraform apply
```

IRIS Setup

Lets quickly jam some data into IRIS for a demonostration.

Create a class like so:

```
Class User.People Extends (%Persistent, %Populate)
{
Property ID As %String;
Property FirstName As %String(POPSPEC = "NAME");
Property LastName As %String(POPSPEC = "NAME");
}
```

Then run the populate to generate some data.

```
USER>do ##class(User.People).Populate(10000)
```

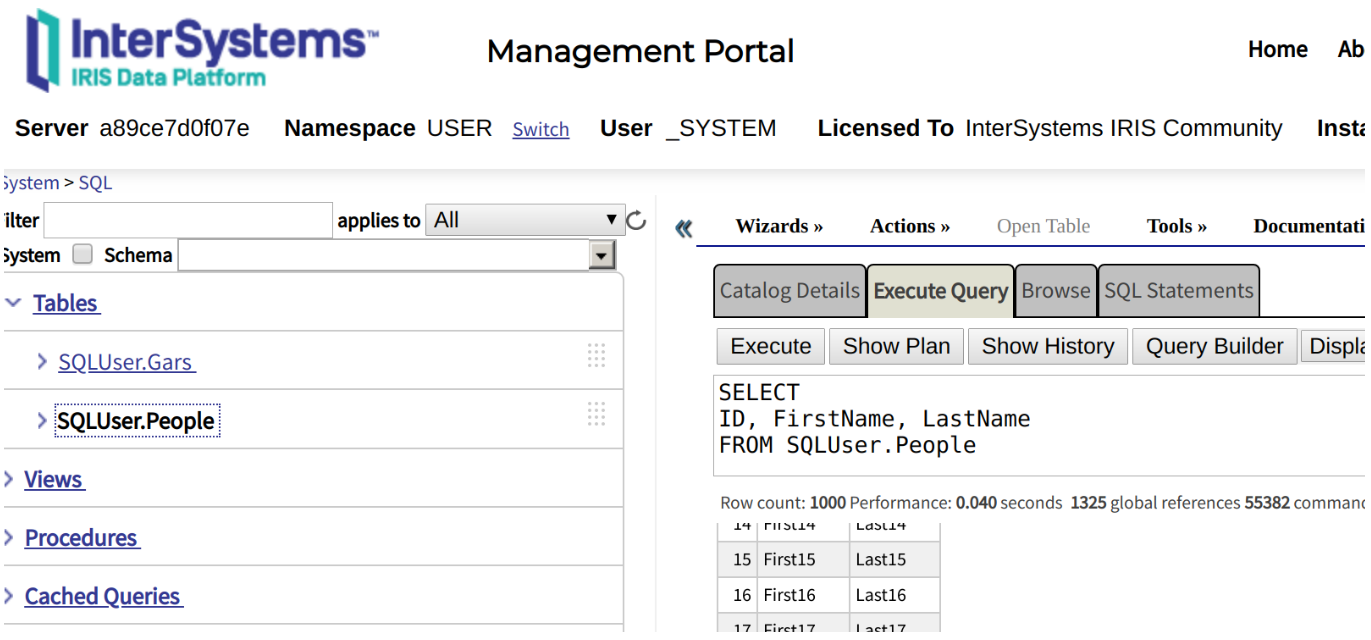
Alternatively, you can grab an irissession, ensure you are in the USER namespace and run the following commands.

```
USER> SET result=$SYSTEM.SQL.Execute("CREATE TABLE People(ID int, FirstName varchar(2
55), LastName varchar(255))")
```

```
USER> for i=1:1:100000 { SET result=$SYSTEM.SQL.Execute("INSERT INTO People VALUES ("
_i_", 'First"_i_"', 'Last"_i_"')") }
```

Both routes will create a table called "People" and insert 100,000 rows.

Either way you to and if everything worked out, you should be able to query for some dummy rows in IRIS. These are the rows we are sending to Big Query.



Server a89ce7d0f07e Namespace USER Switch User _SYSTEM Licensed To InterSystems IRIS Community Insta

ystem > SQL

ilter applies to All

ystem Schema

Tables
- SQLUser.Gars
- SQLUser.People

Views

Procedures

Cached Queries

```
SELECT
ID, FirstName, LastName
FROM SQLUser.People
```

Row count: 1000 Performance: 0.040 seconds 1325 global references 55382 comman

14	First14	Last14
15	First15	Last15
16	First16	Last16
17	First17	Last17

IRIS2BQ Setup

Download the latest release of the utility [iris2bq](), and unzip it. Then cd to the `bin`directory and move over your credentials to the root of this directory and create an application.conf file as below into the same root.

Taking a look at the below configuration file here, you can get an idea of how the utility works.

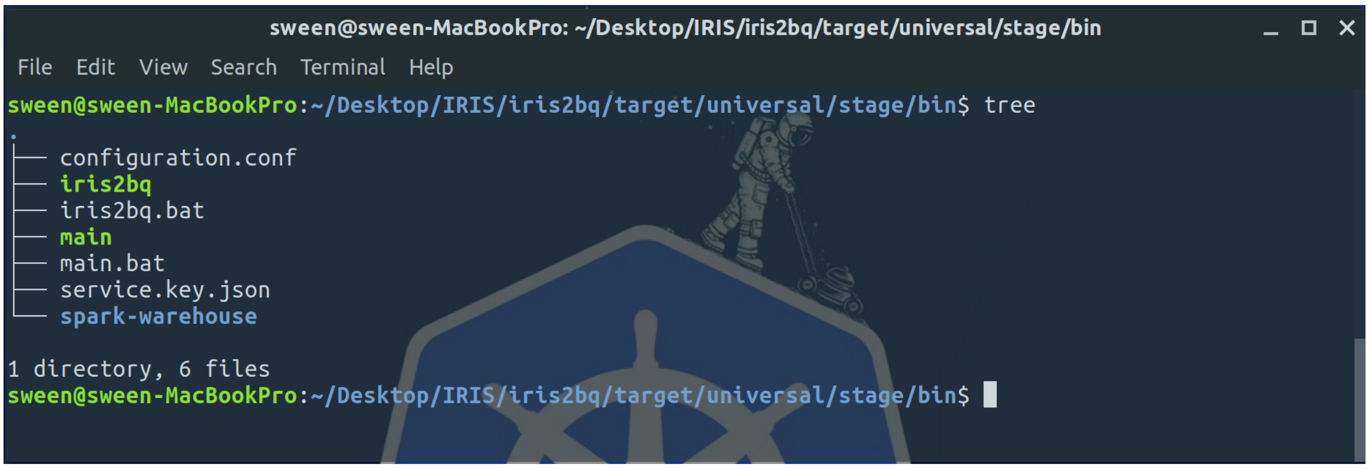
1. Specify a jdbc url and the credentials for the system user.
2. Give it a list of tables that you wan to appear in Big Query.
3. Tell the utility which project to point to, the location of your credentials file.
4. Then tell it a target Big Query Dataset, and a target bucket to write the .avro files to.

Quick note on the GCP block, the dataset and bucket can either exist or not exist as the utility will create those resources for you.

```
jdbc {
  url = "jdbc:IRIS://127.0.0.1:51773/USER"
  user = "_SYSTEM"
  password = "flounder" // the password is flounder
  tables = [ "people" ] //IRIS tables to send to big query
}
gcloud {
  project = "iris2bq-demo"
  service-account-key-path = "service.key.json" //gcp service account
  bq.dataset = "iris2bqdemods" // target bq dataset
  gcs.tmp-bucket = "iris2bqdemobucket" //target storage bucket
}
```

Run

At this point we should be parked at our command prompt in the root of the utility, with a conf file we created and the json credentials file.


sween@sween-MacBookPro: ~/Desktop/IRIS/iris2bq/target/universal/stage/bin
File Edit View Search Terminal Help

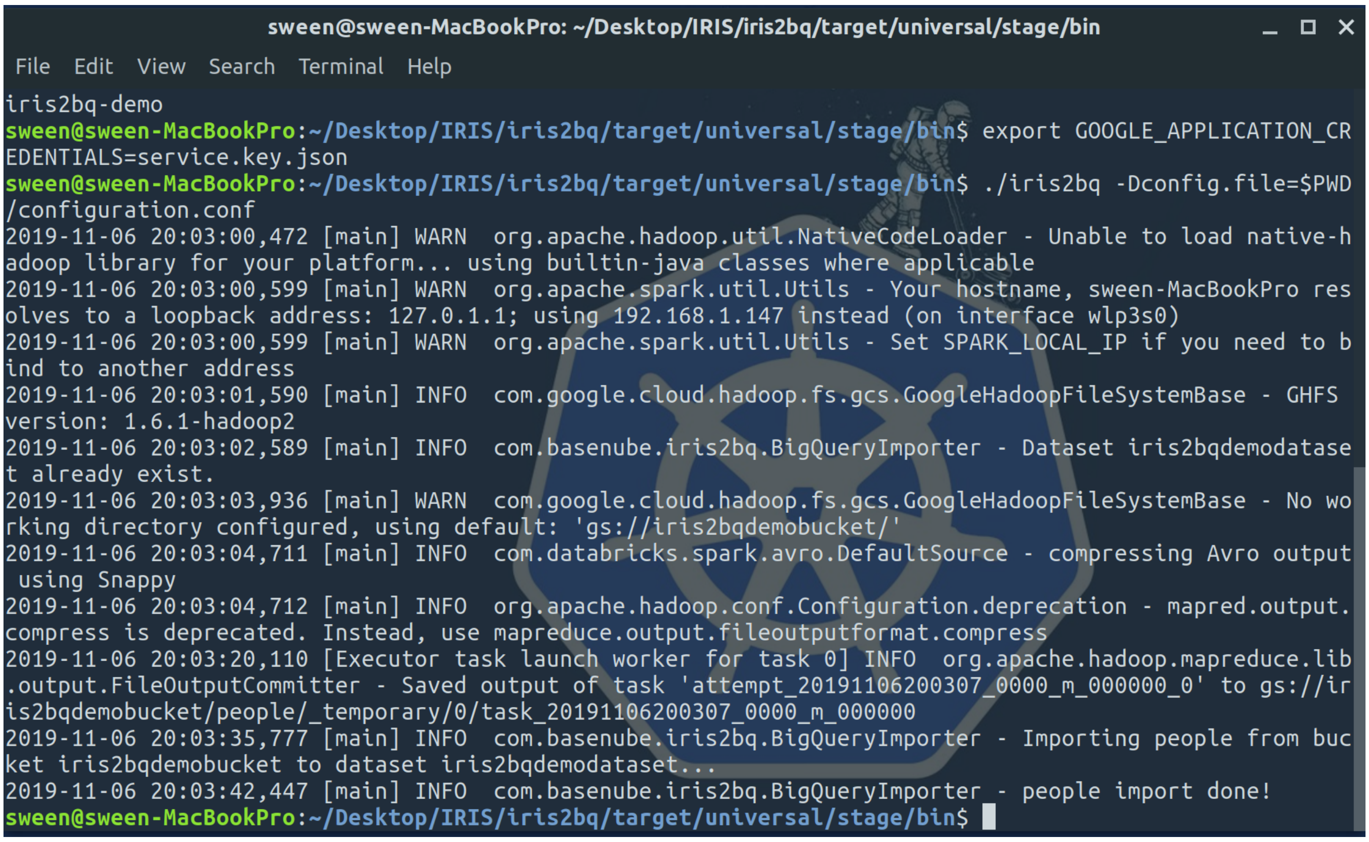
```
sween@sween-MacBookPro:~/Desktop/IRIS/iris2bq/target/universal/stage/bin$ tree
.
├── configuration.conf
├── iris2bq
├── iris2bq.bat
├── main
├── main.bat
├── service.key.json
└── spark-warehouse

1 directory, 6 files
sween@sween-MacBookPro:~/Desktop/IRIS/iris2bq/target/universal/stage/bin$
```

Now that we have all that in place, lets run it and check the result.

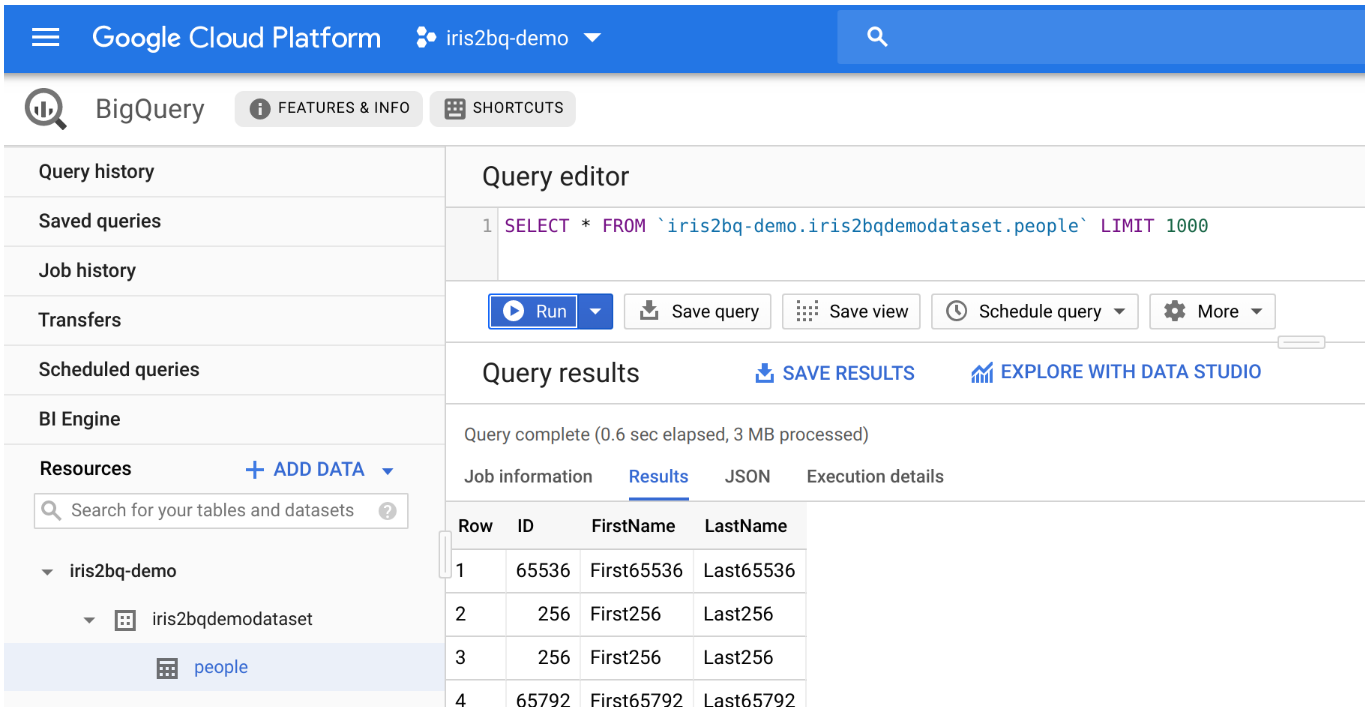
```
$ export GOOGLE_CLOUD_PROJECT=iris2bq-demo
$ exportGOOGLE_APPLICATION_CREDENTIALS=service.key.json
$./iris2bq -Dconfig.file=configuration.conf
```

The output is a tad chatty, but if the import was successful it will state ̀people import done!`


sween@sween-MacBookPro: ~/Desktop/IRIS/iris2bq/target/universal/stage/bin
File Edit View Search Terminal Help

```
iris2bq-demo
sween@sween-MacBookPro:~/Desktop/IRIS/iris2bq/target/universal/stage/bin$ export GOOGLE_APPLICATION_CR
EDENTIALS=service.key.json
sween@sween-MacBookPro:~/Desktop/IRIS/iris2bq/target/universal/stage/bin$ ./iris2bq -Dconfig.file=$PWD
/configuration.conf
2019-11-06 20:03:00,472 [main] WARN  org.apache.hadoop.util.NativeCodeLoader - Unable to load native-h
adoop library for your platform... using builtin-java classes where applicable
2019-11-06 20:03:00,599 [main] WARN  org.apache.spark.util.Utils - Your hostname, sween-MacBookPro res
olves to a loopback address: 127.0.1.1; using 192.168.1.147 instead (on interface wlp3s0)
2019-11-06 20:03:00,599 [main] WARN  org.apache.spark.util.Utils - Set SPARK_LOCAL_IP if you need to b
ind to another address
2019-11-06 20:03:01,590 [main] INFO  com.google.cloud.hadoop.fs.gcs.GoogleHadoopFileSystemBase - GHFS
version: 1.6.1-hadoop2
2019-11-06 20:03:02,589 [main] INFO  com.basenube.iris2bq.BigQueryImporter - Dataset iris2bqdemodatase
t already exist.
2019-11-06 20:03:03,936 [main] WARN  com.google.cloud.hadoop.fs.gcs.GoogleHadoopFileSystemBase - No wo
rking directory configured, using default: 'gs://iris2bqdemobucket/'
2019-11-06 20:03:04,711 [main] INFO  com.databricks.spark.avro.DefaultSource - compressing Avro output
 using Snappy
2019-11-06 20:03:04,712 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - mapred.output.
compress is deprecated. Instead, use mapreduce.output.fileoutputformat.compress
2019-11-06 20:03:20,110 [Executor task launch worker for task 0] INFO  org.apache.hadoop.mapreduce.lib
.output.FileOutputCommitter - Saved output of task 'attempt_20191106200307_0000_m_000000_0' to gs://ir
is2bqdemobucket/people/_temporary/0/task_20191106200307_0000_m_000000
2019-11-06 20:03:35,777 [main] INFO  com.basenube.iris2bq.BigQueryImporter - Importing people from buc
ket iris2bqdemobucket to dataset iris2bqdemodataset...
2019-11-06 20:03:42,447 [main] INFO  com.basenube.iris2bq.BigQueryImporter - people import done!
sween@sween-MacBookPro:~/Desktop/IRIS/iris2bq/target/universal/stage/bin$
```

Lets head over to to Big Query and inspect our work...



```
SELECT * FROM `iris2bq-demo.iris2bqdemodataset.people` LIMIT 1000
```

Query complete (0.6 sec elapsed, 3 MB processed)

Row	ID	FirstName	LastName
1	65536	First65536	Last65536
2	256	First256	Last256
3	256	First256	Last256
4	65792	First65792	Last65792

The baseNUBE team hopes you found this helpful!

Now setup a job to run it on an interval and JOIN all over your IRIS data in Big Query!

#Best Practices #Big Data #Cloud #GCP #integration-required #InterSystems IRIS #InterSystems IRIS for Health
Check the related application on InterSystems Open Exchange

Source URL:https://community.intersystems.com/post/export-intersystems-iris-data-bigquery-google-cloud-platform