📖 **06-retail-store-analytics-bigquery-execution.md**

Retail store analytics using Serverless Spark and Metastore with Google BigQuery

Following are the lab modules:

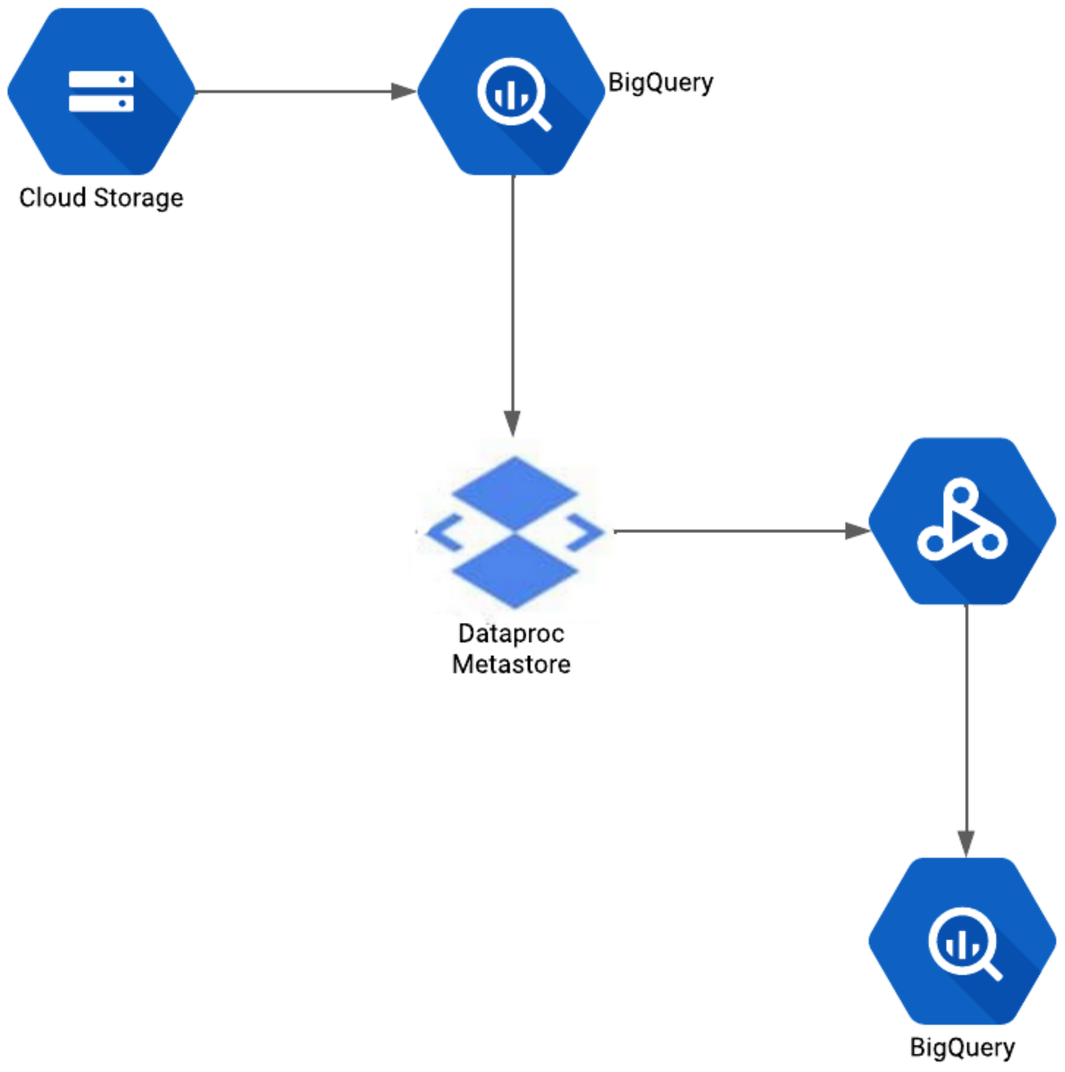
1. Understanding the data

The datasets used for this project are

1. Aisles data.
2. Departments data .
3. Orders data.
4. Products data.
5. Order_products__prior.
6. Order_products__train.

- Aisles: This table includes all aisles. It has a single primary key (aisle_id)
- Departments: This table includes all departments. It has a single primary key (department_id)
- Products: This table includes all products. It has a single primary key (product_id)
- Orders: This table includes all orders, namely prior, train, and test. It has single primary key (order_id).
- Order_products_train: This table includes training orders. It has a composite primary key (order_id and product_id) and indicates whether a product in an order is a reorder or not (through the reordered variable).
- Order_products_prior : This table includes prior orders. It has a composite primary key (order_id and product_id) and indicates whether a product in an order is a reorder or not (through the reordered variable).

2. Solution Architecture



Data Pipeline

The data pipeline involves the following steps:
- Create buckets in GCS
- Create Dataproc and Persistent History Server Cluster
- Copy the raw data files, pyspark and notebook files into GCS
- Create a metastore service in Cloud Dataproc
- Executing the code through Big Query
- Getting the output tables in Google BigQuery and dataproc

3. Declaring Variables

3.1 Set the PROJECT_ID in Cloud Shell

Open Cloud shell or navigate to shell.cloud.google.com
Run the below

```
gcloud config set project $PROJECT_ID
```

3.2 Verify the PROJECT_ID in Cloud Shell

Next, run the following command in cloud shell to ensure that the current project is set correctly:

```
gcloud config get-value project
```

3.3 Declare the variables

Based on the prereqs and checklist, declare the following variables in cloud shell by replacing with your values:

```
PROJECT_ID=$(gcloud config get-value project)       #current GCP project where we are building our use case
REGION=                                             #GCP region where all our resources will be created
SUBNET=                                             #subnet which has private google access enabled
BUCKET_CODE=                                        #GCP bucket where our code, data and model files will be stored
BUCKET_PHS=                                         #bucket where our application logs created in the history server will be stored
HISTORY_SERVER_NAME=                                #name of the history server which will store our application logs
BQ_DATASET_NAME=                                    #BigQuery dataset where all the tables will be stored
UMSA=serverless-spark                               #name of the user managed service account required for the PySpark job executions
SERVICE_ACCOUNT=$UMSA@$PROJECT_ID.iam.gserviceaccount.com
METASTORE_NAME                                      #name of the metastore which will store our schema
NAME=                                               #your name
```

3.4 Update Cloud Shell SDK version

Run the below on cloud shell-

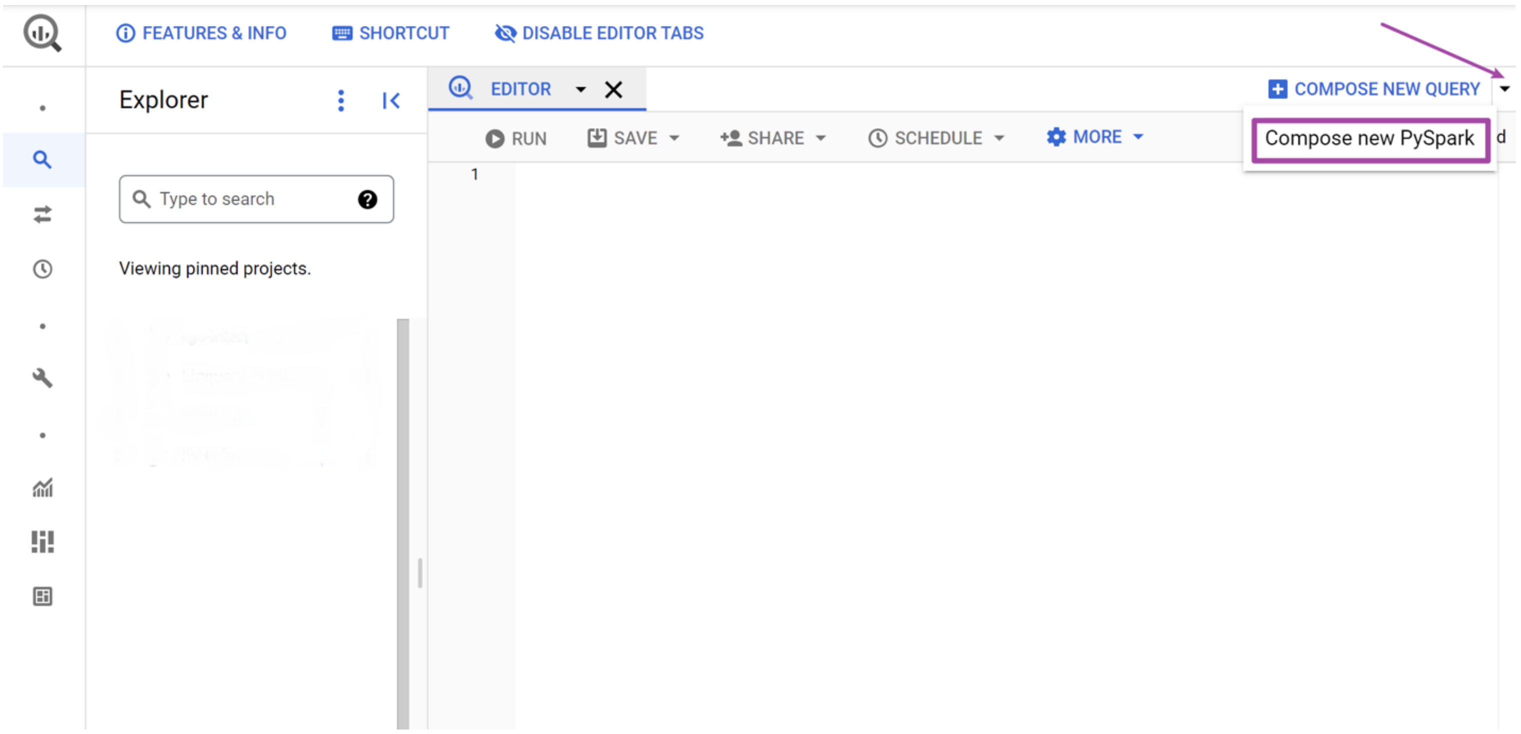
```
gcloud components update
```

4. Running the job on BigQuery

4.1. Navigate to PySpark Console in BigQuery

Open BigQuery Console

Click on the the dropdown to "Compose new query" and click on **"Compose new PySpark"**

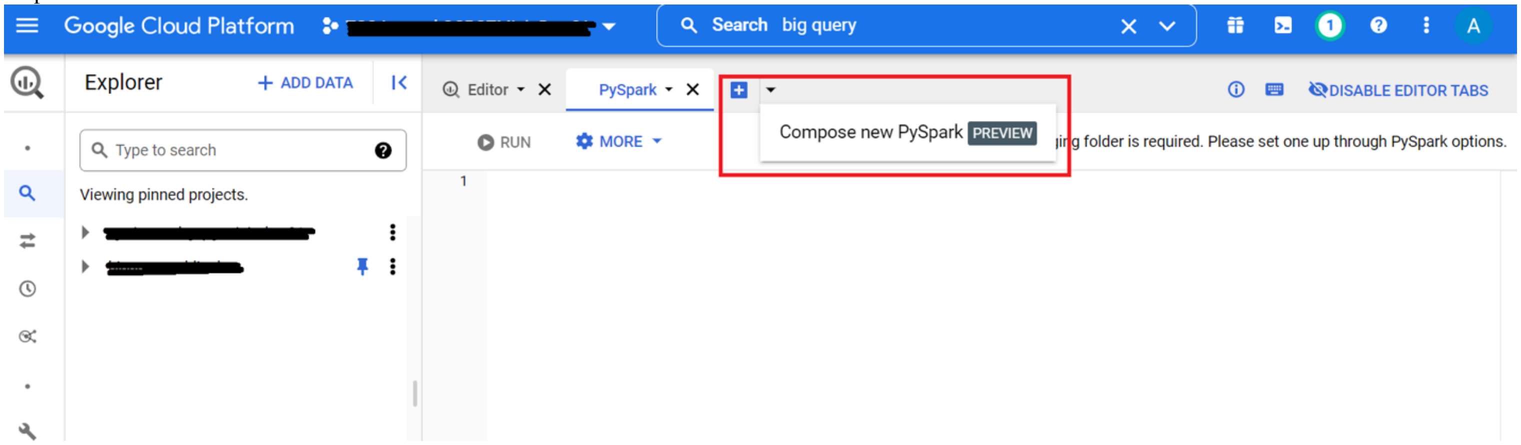


4.2. Provide Configuration for Serverless Spark job

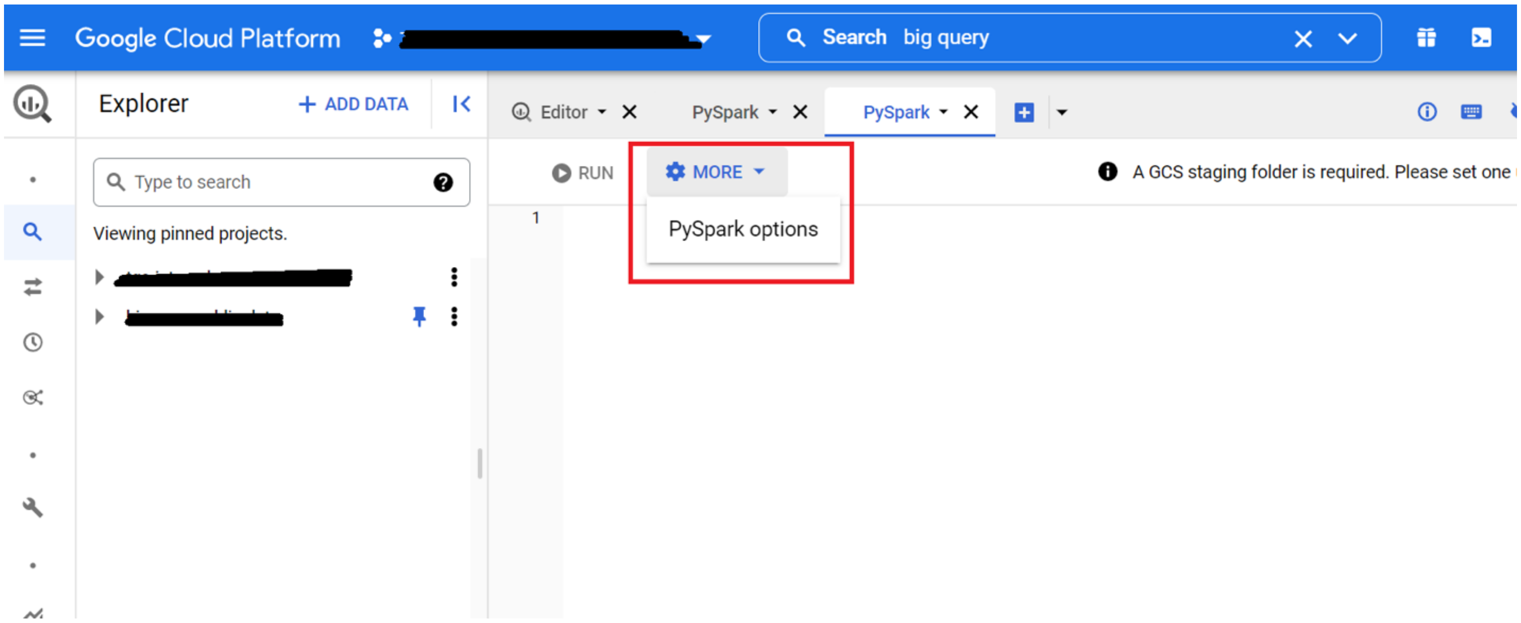
Navigate to Bigquery > Compose new PySpark Query > PySpark Options

How to reach to configure a Serverless Spark job

Step-1

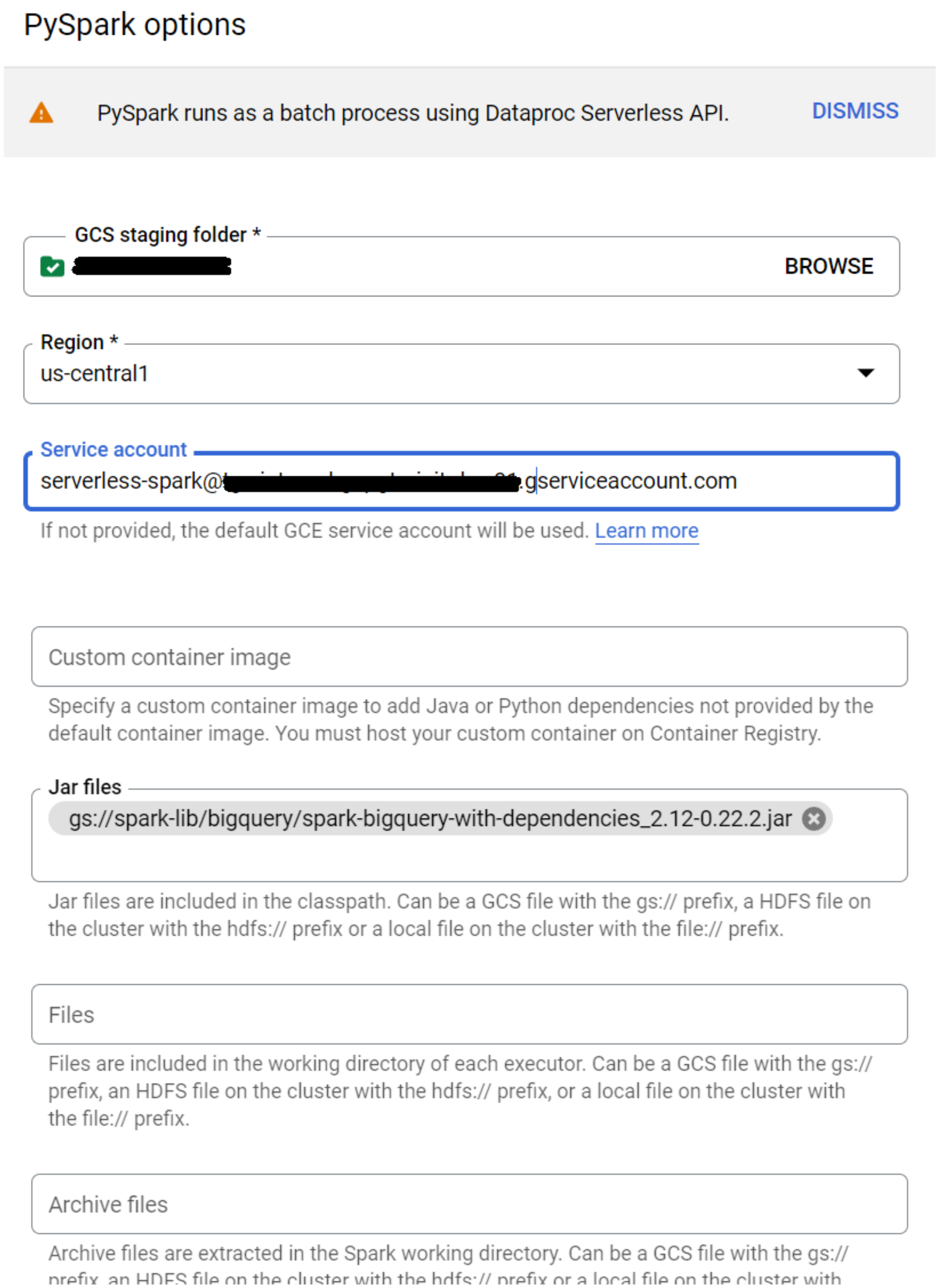


Step-2



Next, fill in the following values in the PySpark creation window :

- **GCS Staging Folder** - <your_bucket_name>
- **Region** - The region name provided by the Admin team
- **Service Account** - <UMSA_NAME>@<PROJECT_ID>.iam.gserviceaccount.com



PySpark options

PySpark runs as a batch process using Dataproc Serverless API. DISMISS

GCS staging folder * [redacted] BROWSE

Region * us-central1

Service account serverless-spark@[redacted].gserviceaccount.com

If not provided, the default GCE service account will be used. Learn more

Custom container image

Specify a custom container image to add Java or Python dependencies not provided by the default container image. You must host your custom container on Container Registry.

Jar files gs://spark-lib/bigquery/spark-bigquery-with-dependencies_2.12-0.22.2.jar

Jar files are included in the classpath. Can be a GCS file with the gs:// prefix, a HDFS file on the cluster with the hdfs:// prefix or a local file on the cluster with the file:// prefix.

Files

Files are included in the working directory of each executor. Can be a GCS file with the gs:// prefix, an HDFS file on the cluster with the hdfs:// prefix, or a local file on the cluster with the file:// prefix.

Archive files

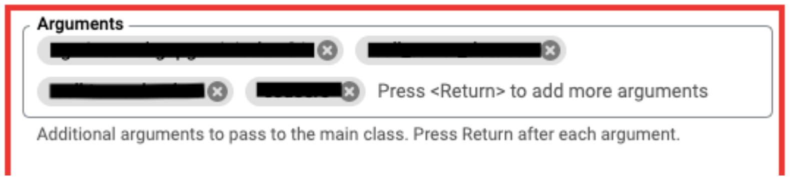
Archive files are extracted in the Spark working directory. Can be a GCS file with the gs:// prefix, an HDFS file on the cluster with the hdfs:// prefix or a local file on the cluster with

- **Arguments** -
  Four Arguments needs to be provided:
  - <your_project_name> #project name
  - <your_bq_dataset_name> #dataset_name
  - <your_code_bucket_name>

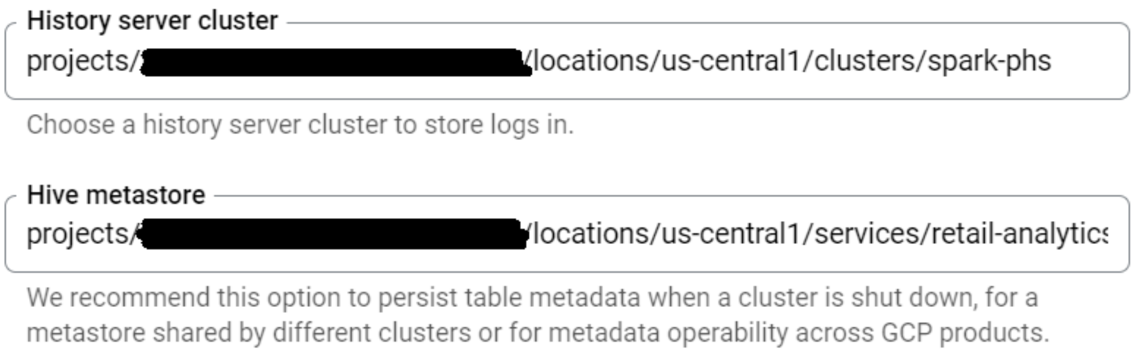
    - <your_name>

  Note: Press RETURN after each argument
  Note: The arguments must be passed in the same order as mentioned as they are extracted in the order they are provided

- **Network Configuration** - select the network and subnetwork with Private Google Access Enabled
- **Hive Metastore** - select the Dataproc metastore created by the Admin
- **History Server Cluster** - projects/<PROJECT_ID>/regions/<REGION_NAME>/clusters/<HISTORY_SERVER_NAME>

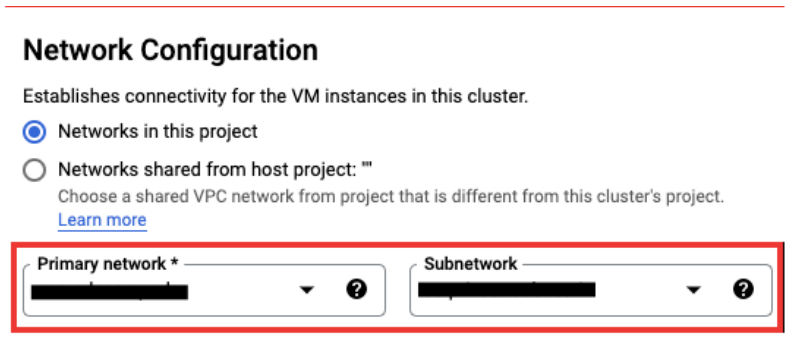


Arguments
[redacted] [redacted] [redacted] [redacted] Press <Return> to add more arguments
Additional arguments to pass to the main class. Press Return after each argument.



History server cluster
projects/[redacted]/locations/us-central1/clusters/spark-phs
Choose a history server cluster to store logs in.

Hive metastore
projects/[redacted]/locations/us-central1/services/retail-analytics
We recommend this option to persist table metadata when a cluster is shut down, for a metastore shared by different clusters or for metadata operability across GCP products.

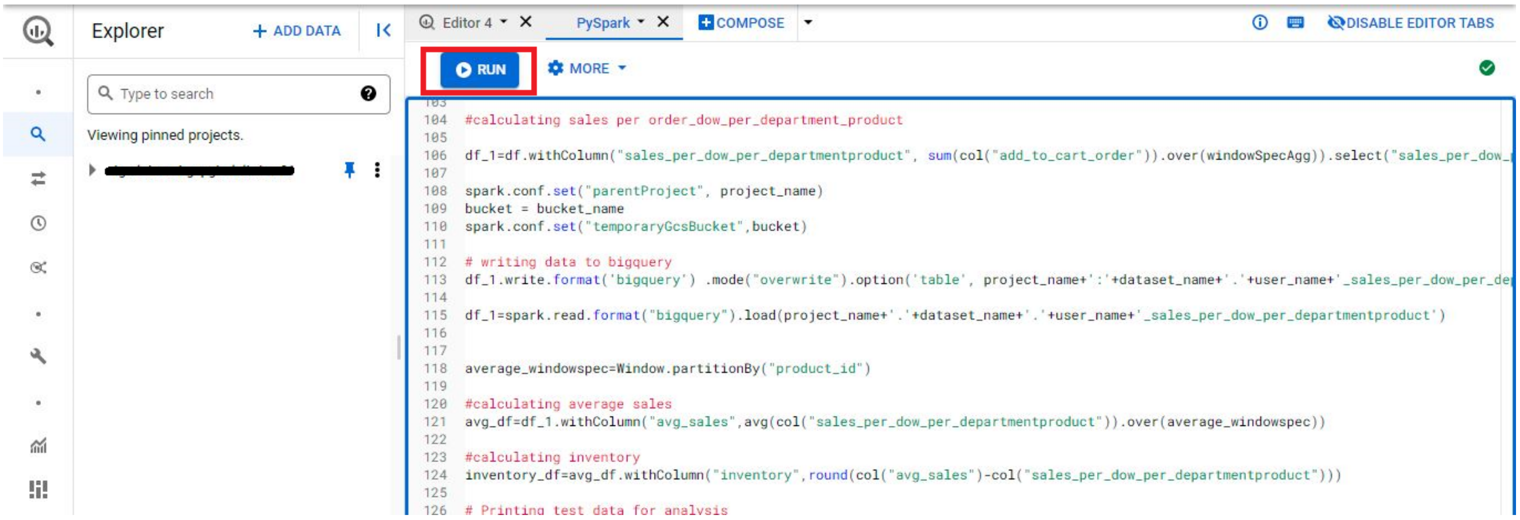


Network Configuration
Establishes connectivity for the VM instances in this cluster.
[✓ Networks in this project]
Networks shared from host project: ""
Choose a shared VPC network from project that is different from this cluster's project.
Learn more
Primary network * [redacted]
Subnetwork [redacted]

Once all the details are in, you can save the session. Once the serverless spark session is created, you can execute your code against this session.

4.3. Submit the BigQuery PySpark Job

Copy the code 00-scripts/retail-store-analytics-bigquery.py in to the big query notebook created. Next, hit the Run button as shown .

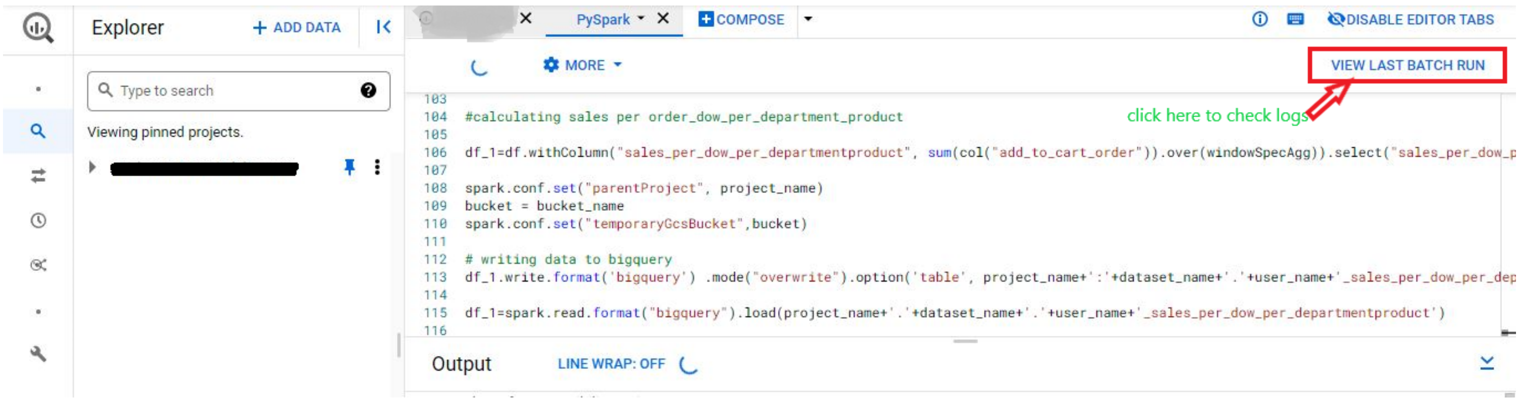


```
#calculating sales per order_dow_per_department_product

df_1=df.withColumn("sales_per_dow_per_departmentproduct", sum(col("add_to_cart_order")).over(windowSpecAgg)).select("sales_per_dow_

spark.conf.set("parentProject", project_name)
bucket = bucket_name
spark.conf.set("temporaryGcsBucket",bucket)

# writing data to bigquery
df_1.write.format('bigquery') .mode("overwrite").option('table', project_name+':'+dataset_name+'.'+user_name+'_sales_per_dow_per_de

df_1=spark.read.format("bigquery").load(project_name+'.'+dataset_name+'.'+user_name+'_sales_per_dow_per_departmentproduct')


average_windowspec=Window.partitionBy("product_id")

#calculating average sales
avg_df=df_1.withColumn("avg_sales",avg(col("sales_per_dow_per_departmentproduct")).over(average_windowspec))

#calculating inventory
inventory_df=avg_df.withColumn("inventory",round(col("avg_sales")-col("sales_per_dow_per_departmentproduct")))

# Printing test data for analysis
```

4.4. Examine the BigQuery Batch Details

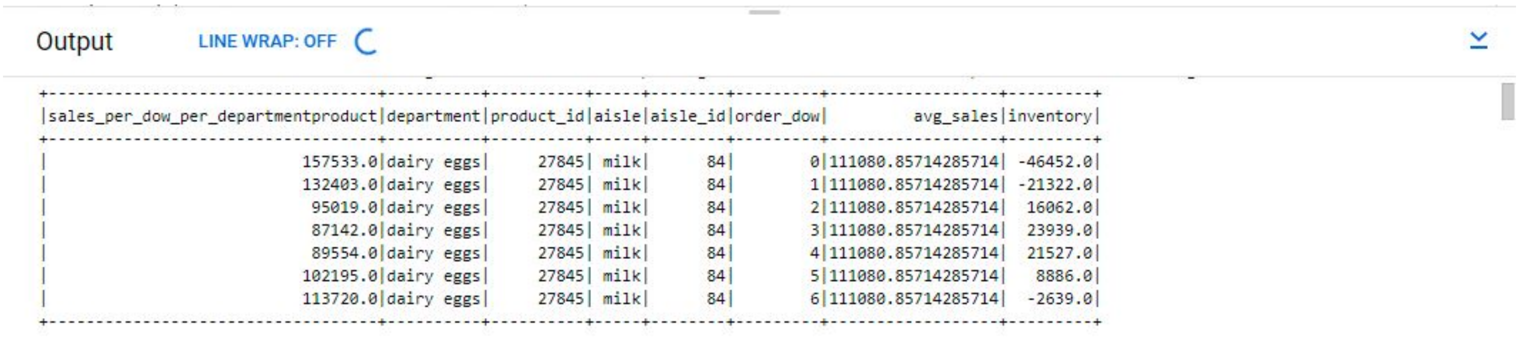
Once you submit the job, you will the see the Batches page populate with the current run.

To navigate, click on **Last Batch Run**, which will open the page with the batch details and you can monitor the execution and output.



```
#calculating sales per order_dow_per_department_product

df_1=df.withColumn("sales_per_dow_per_departmentproduct", sum(col("add_to_cart_order")).over(windowSpecAgg)).select("sales_per_dow_p

spark.conf.set("parentProject", project_name)
bucket = bucket_name
spark.conf.set("temporaryGcsBucket",bucket)

# writing data to bigquery
df_1.write.format('bigquery') .mode("overwrite").option('table', project_name+':'+dataset_name+'.'+user_name+'_sales_per_dow_per_dep

df_1=spark.read.format("bigquery").load(project_name+'.'+dataset_name+'.'+user_name+'_sales_per_dow_per_departmentproduct')
```

The same details are found on the BigQuery Output section as well.

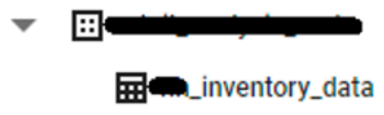


```
+-----------------------------------+----------+----------+-----+--------+---------+------------------+---------+
|sales_per_dow_per_departmentproduct|department|product_id|aisle|aisle_id|order_dow|         avg_sales|inventory|
+-----------------------------------+----------+----------+-----+--------+---------+------------------+---------+
|                           157533.0|dairy eggs|     27845| milk|      84|        0|111080.85714285714| -46452.0|
|                           132403.0|dairy eggs|     27845| milk|      84|        1|111080.85714285714| -21322.0|
|                            95019.0|dairy eggs|     27845| milk|      84|        2|111080.85714285714|  16062.0|
|                            87142.0|dairy eggs|     27845| milk|      84|        3|111080.85714285714|  23939.0|
|                            89554.0|dairy eggs|     27845| milk|      84|        4|111080.85714285714|  21527.0|
|                           102195.0|dairy eggs|     27845| milk|      84|        5|111080.85714285714|   8886.0|
|                           113720.0|dairy eggs|     27845| milk|      84|        6|111080.85714285714|  -2639.0|
+-----------------------------------+----------+----------+-----+--------+---------+------------------+---------+
```

Congratulations on successfully completing the first Serverless Spark Batch on BigQuery. As you can see in the example, no cluster or infrastructure administration was required to perform this batch. Dataproc serverless is responsible for launching the cluster, running the job, scaling it as needed, and cleaning up after the job is completed.

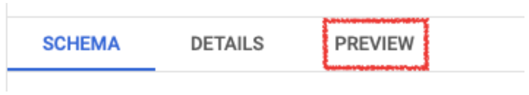
5. BQ output tables

Navigate to BigQuery Console, and check the **<your_BQ_DATASET_NAME>** dataset.
Once the Serverless batches execution is completed, a new new table '<your_name_here>_inventory_data' will be created:


[redacted]
[redacted]_inventory_data

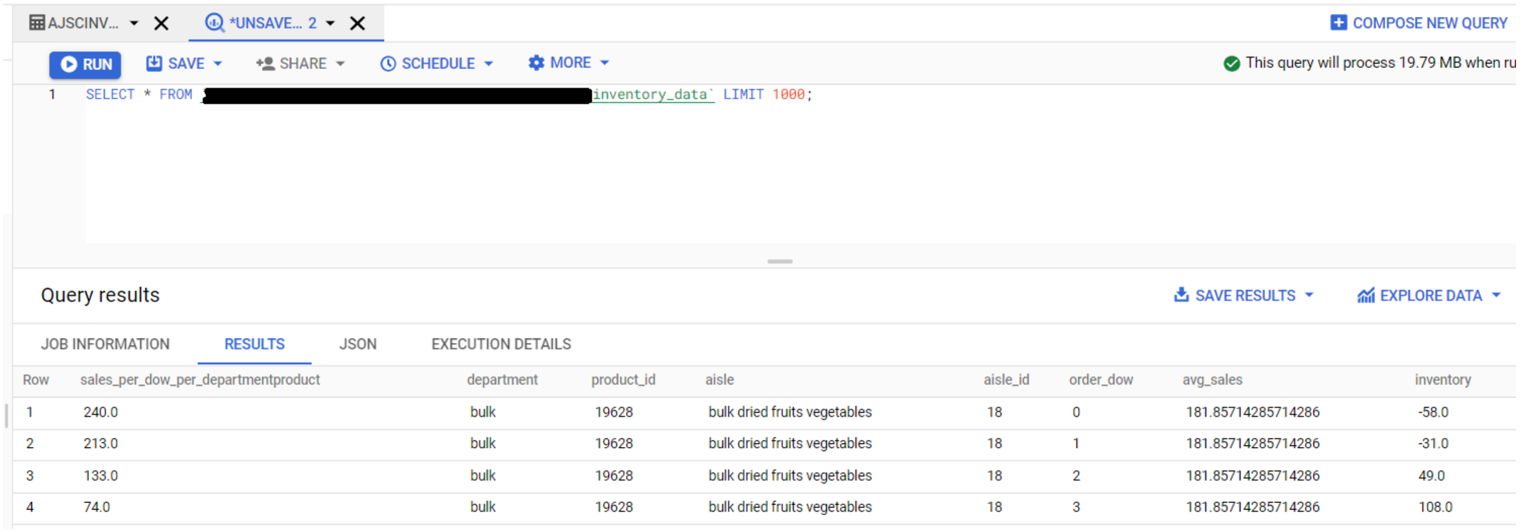
To view the data in these tables -

- Select the table from BigQuery Explorer by navigating 'project_id' **>** 'dataset' **>** 'table_name'
- Click on the **Preview** button to see the data in the table



Note: If the **Preview** button is not visible, run the below queries to view the data. However, these queries will be charged for the full table scan.

```
SELECT * FROM `<project_name>.<dataset_name>.<your_name_here>_inventory_data` LIMIT 1000;
```



```
SELECT * FROM [redacted]inventory_data` LIMIT 1000;
```

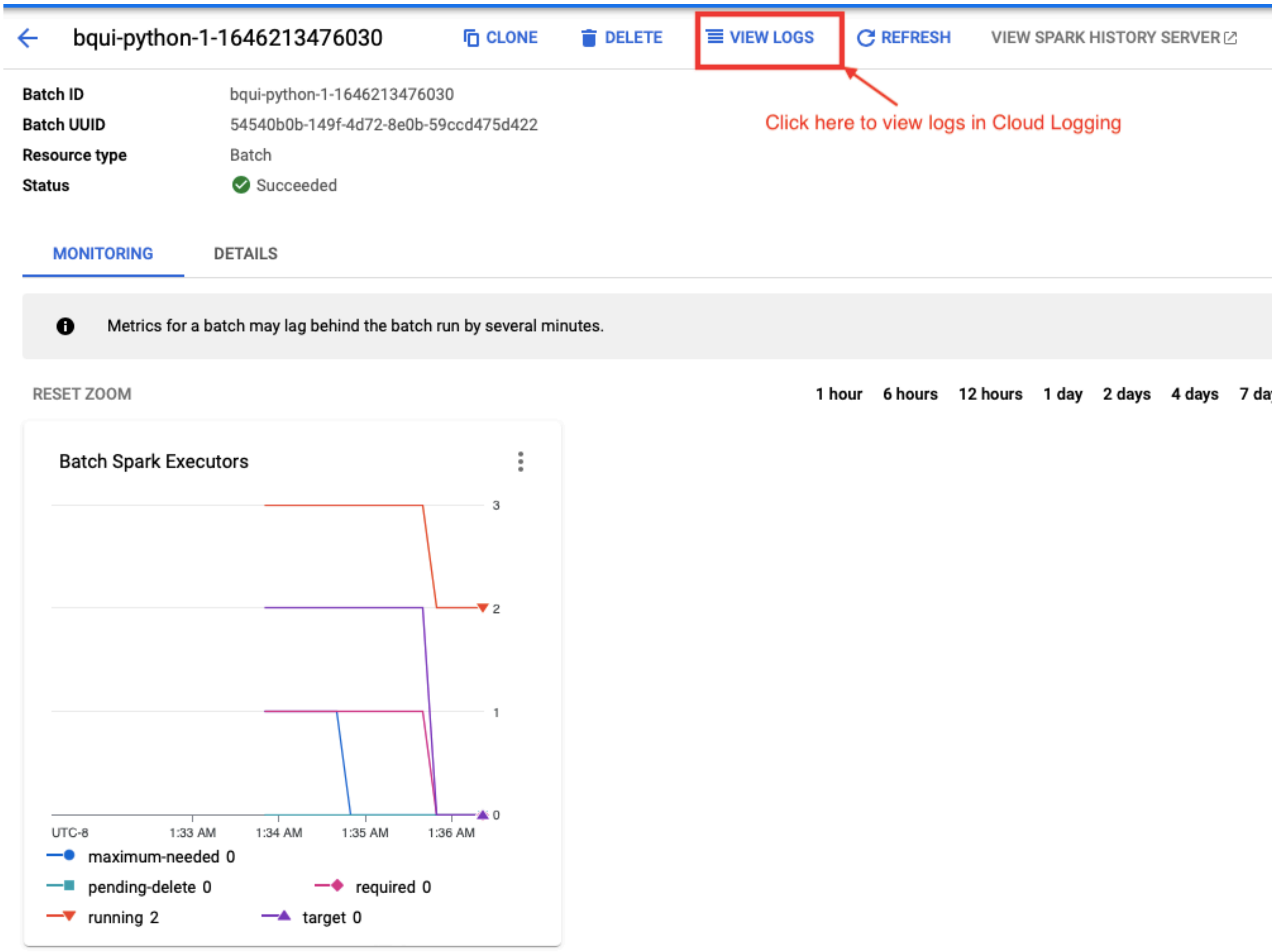
Row	sales_per_dow_per_departmentproduct	department	product_id	aisle	aisle_id	order_dow	avg_sales	inventory
1	240.0	bulk	19628	bulk dried fruits vegetables	18	0	181.85714285714286	-58.0
2	213.0	bulk	19628	bulk dried fruits vegetables	18	1	181.85714285714286	-31.0
3	133.0	bulk	19628	bulk dried fruits vegetables	18	2	181.85714285714286	49.0
4	74.0	bulk	19628	bulk dried fruits vegetables	18	3	181.85714285714286	108.0

Note: Edit all occurrences of <project_name> and <dataset_name> to match the values of the variables PROJECT_ID, and BQ_DATASET_NAME respectively

6. Logging

6.1 Serverless Batch logs

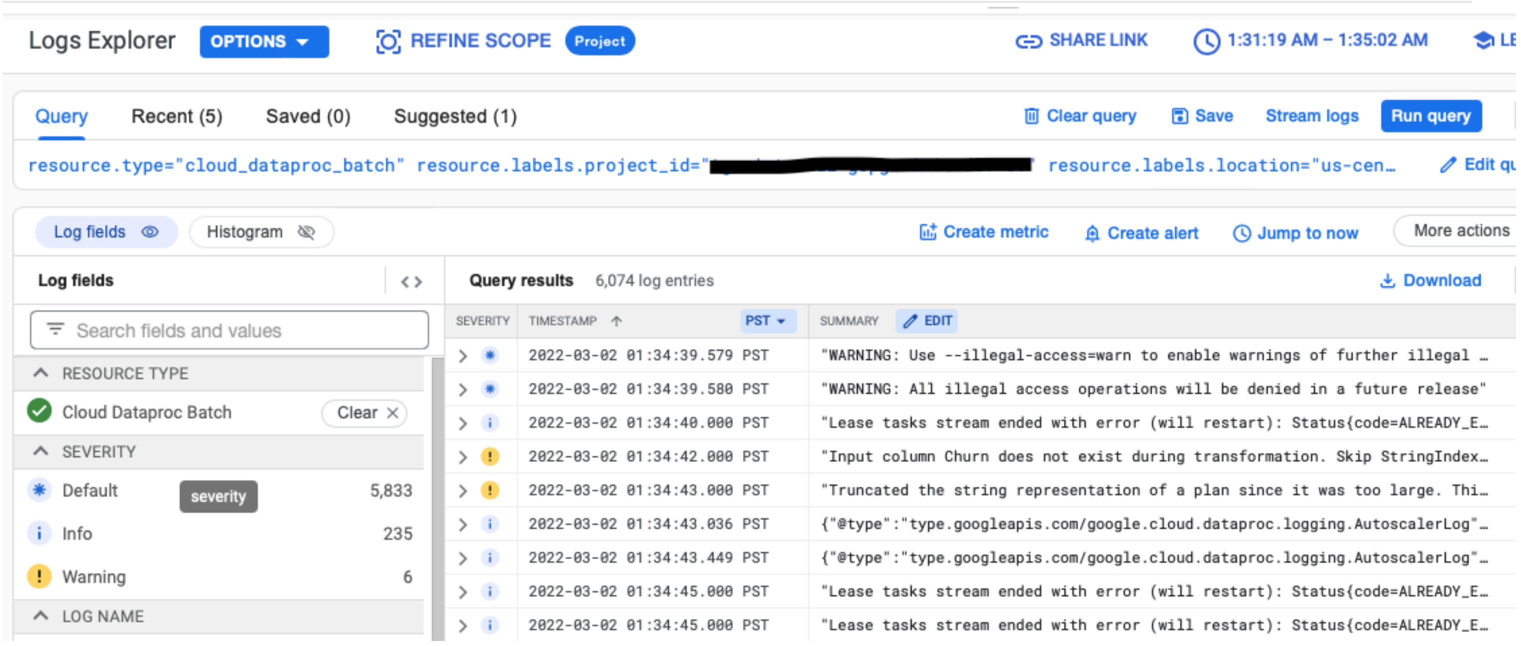
Logs associated with the application can be found in the logging console under **Dataproc > Serverless > Batches > <batch_name>**.
You can also click on “View Logs” button on the Dataproc batches monitoring page to get to the logging page for the specific Spark job.



Batch ID	bqui-python-1-1646213476030
Batch UUID	54540b0b-149f-4d72-8e0b-59ccd475d422
Resource type	Batch
Status	✓ Succeeded

Metrics for a batch may lag behind the batch run by several minutes.

maximum-needed	0
pending-delete	0
required	0
running	2
target	0



```
resource.type="cloud_dataproc_batch" resource.labels.project_id="[redacted]" resource.labels.location="us-cen…
```

Log fields

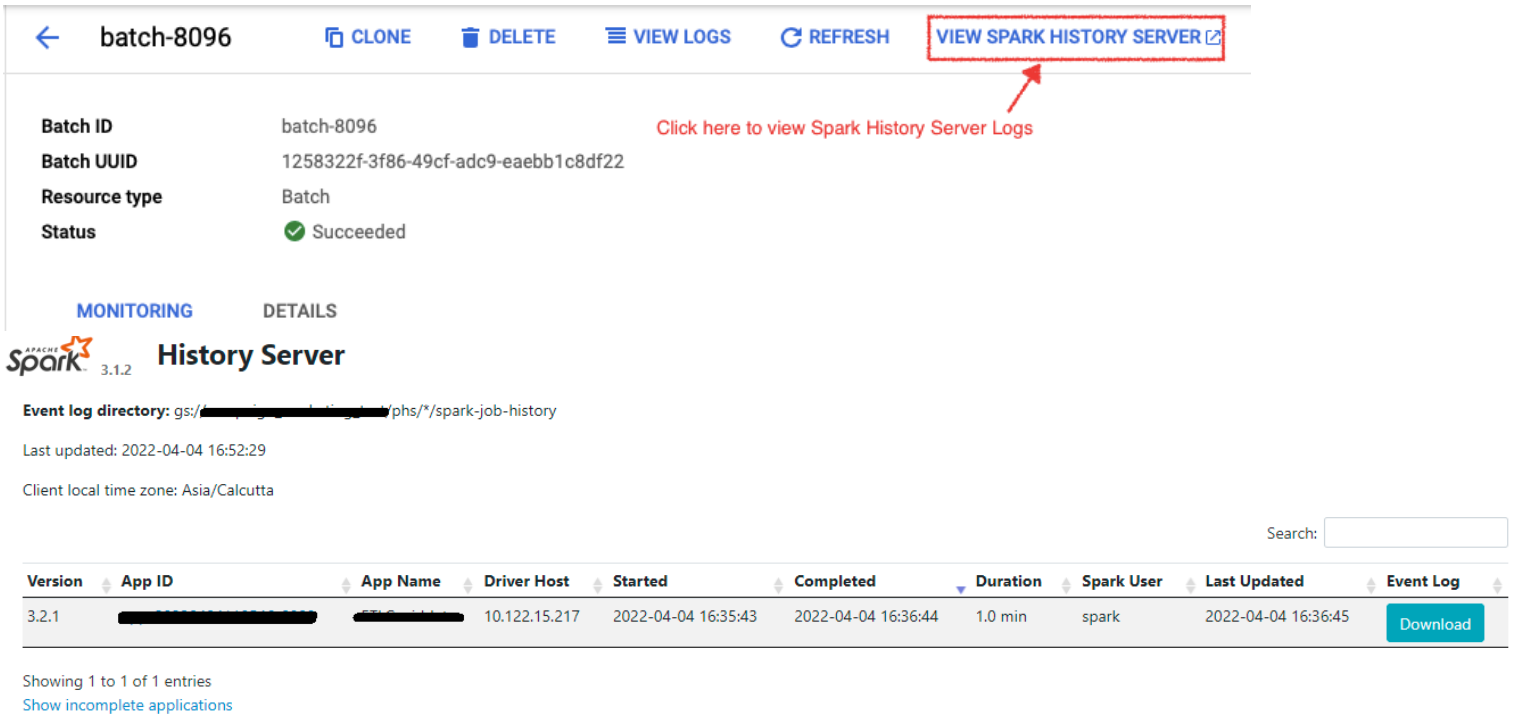
RESOURCE TYPE	
Cloud Dataproc Batch	
SEVERITY	
Default	5,833
Info	235
Warning	6
LOG NAME	

Query results 6,074 log entries

```
SEVERITY  TIMESTAMP                    SUMMARY
          2022-03-02 01:34:39.579 PST  "WARNING: Use --illegal-access=warn to enable warnings of further illegal …
          2022-03-02 01:34:39.580 PST  "WARNING: All illegal access operations will be denied in a future release"
          2022-03-02 01:34:40.000 PST  "Lease tasks stream ended with error (will restart): Status{code=ALREADY_E…
          2022-03-02 01:34:42.000 PST  "Input column Churn does not exist during transformation. Skip StringIndex…
          2022-03-02 01:34:43.000 PST  "Truncated the string representation of a plan since it was too large. Thi…
          2022-03-02 01:34:43.036 PST  {"@type":"type.googleapis.com/google.cloud.dataproc.logging.AutoscalerLog"…
          2022-03-02 01:34:43.449 PST  {"@type":"type.googleapis.com/google.cloud.dataproc.logging.AutoscalerLog"…
          2022-03-02 01:34:45.000 PST  "Lease tasks stream ended with error (will restart): Status{code=ALREADY_E…
          2022-03-02 01:34:45.000 PST  "Lease tasks stream ended with error (will restart): Status{code=ALREADY_E…
```

6.2 Persistent History Server logs

To view the Persistent History server logs, click the 'View History Server' button on the Dataproc batches monitoring page and the logs will be shown as below:


batch-8096
CLONE
DELETE
VIEW LOGS
REFRESH
VIEW SPARK HISTORY SERVER
Click here to view Spark History Server Logs
MONITORING
DETAILS
Spark 3.1.2
History Server
Search:
Download
Show incomplete applications

Batch ID	batch-8096
Batch UUID	1258322f-3f86-49cf-adc9-eaebb1c8df22
Resource type	Batch
Status	Succeeded

Event log directory: gs://[redacted]/phs/*/spark-job-history

Last updated: 2022-04-04 16:52:29

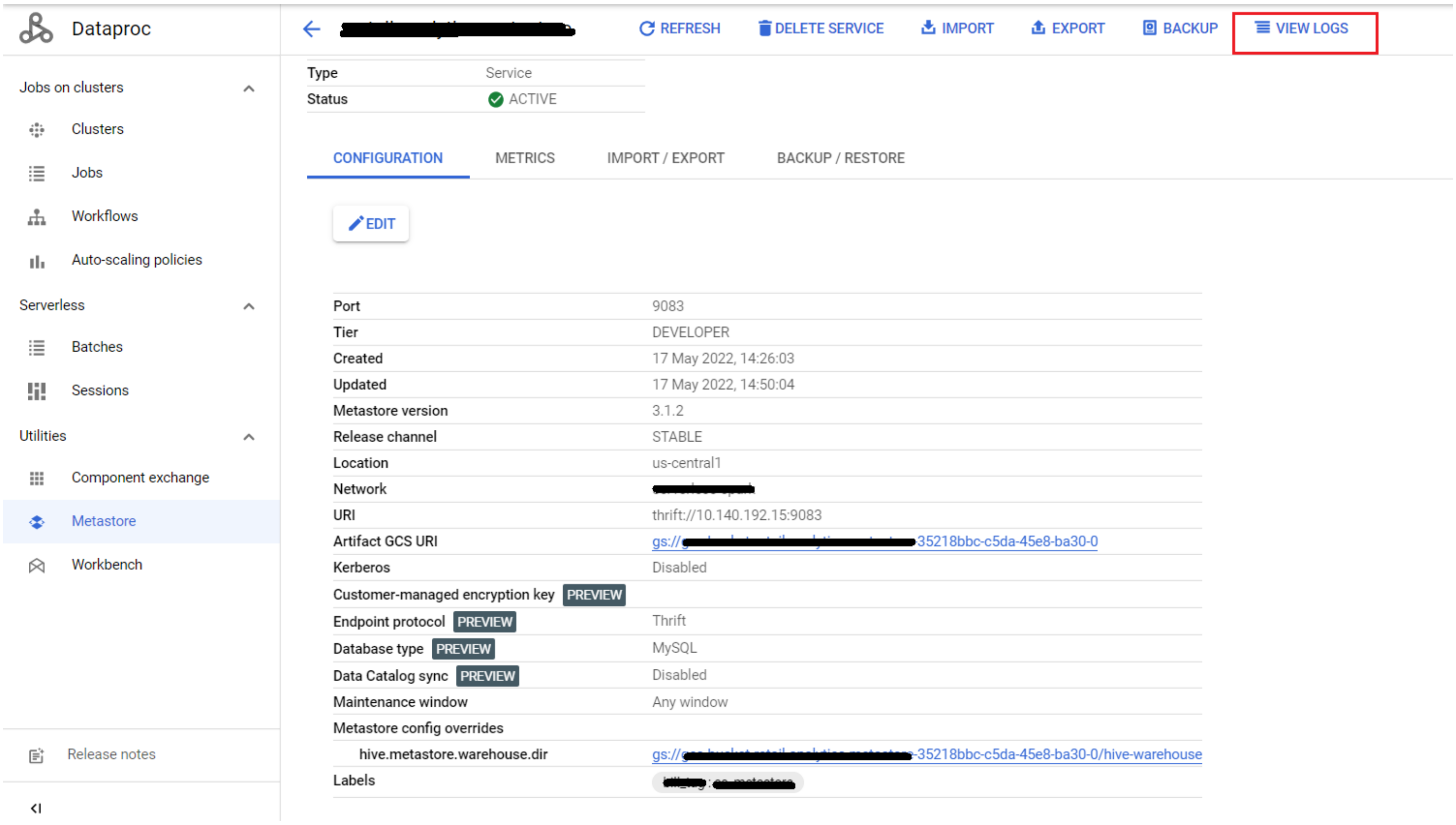
Client local time zone: Asia/Calcutta

Version	App ID	App Name	Driver Host	Started	Completed	Duration	Spark User	Last Updated	Event Log
3.2.1	[redacted]	[redacted]	10.122.15.217	2022-04-04 16:35:43	2022-04-04 16:36:44	1.0 min	spark	2022-04-04 16:36:45	Download

Showing 1 to 1 of 1 entries

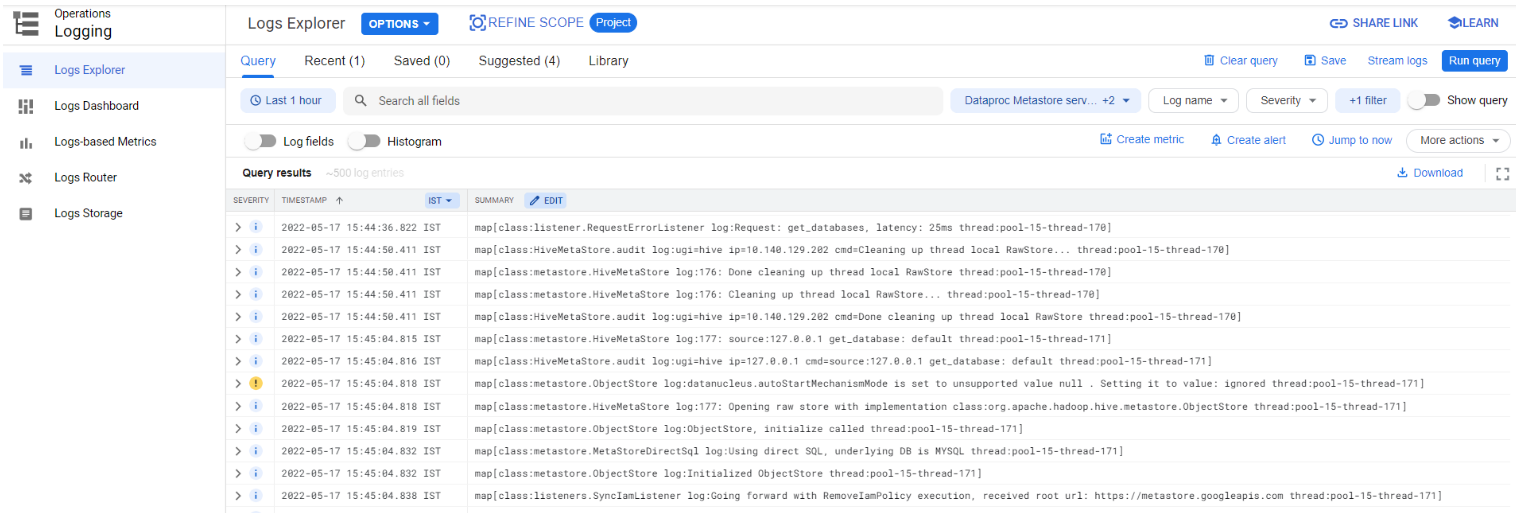
6.3. Metastore logs

To view the metastore logs, click the 'View Logs' button on the metastore page and the logs will be shown as below:



Type	Service
Status	ACTIVE

Port	9083
Tier	DEVELOPER
Created	17 May 2022, 14:26:03
Updated	17 May 2022, 14:50:04
Metastore version	3.1.2
Release channel	STABLE
Location	us-central1
Network	[redacted]
URI	thrift://10.140.192.15:9083
Artifact GCS URI	gs://[redacted]-35218bbc-c5da-45e8-ba30-0
Kerberos	Disabled
Customer-managed encryption key PREVIEW	
Endpoint protocol PREVIEW	Thrift
Database type PREVIEW	MySQL
Data Catalog sync PREVIEW	Disabled
Maintenance window	Any window
Metastore config overrides	
hive.metastore.warehouse.dir	gs://[redacted]-35218bbc-c5da-45e8-ba30-0/hive-warehouse
Labels	[redacted]



```
2022-05-17 15:44:36.822 IST  map[class:listener.RequestErrorListener log:Request: get_databases, latency: 25ms thread:pool-15-thread-170]
2022-05-17 15:44:50.411 IST  map[class:HiveMetaStore.audit log:ugi=hive ip=10.140.129.202 cmd=Cleaning up thread local RawStore... thread:pool-15-thread-170]
2022-05-17 15:44:50.411 IST  map[class:metastore.HiveMetaStore log:176: Done cleaning up thread local RawStore thread:pool-15-thread-170]
2022-05-17 15:44:50.411 IST  map[class:metastore.HiveMetaStore log:176: Cleaning up thread local RawStore... thread:pool-15-thread-170]
2022-05-17 15:44:50.411 IST  map[class:HiveMetaStore.audit log:ugi=hive ip=10.140.129.202 cmd=Done cleaning up thread local RawStore thread:pool-15-thread-170]
2022-05-17 15:45:04.815 IST  map[class:metastore.HiveMetaStore log:177: source:127.0.0.1 get_database: default thread:pool-15-thread-171]
2022-05-17 15:45:04.816 IST  map[class:HiveMetaStore.audit log:ugi=hive ip=127.0.0.1 cmd=source:127.0.0.1 get_database: default thread:pool-15-thread-171]
2022-05-17 15:45:04.818 IST  map[class:metastore.ObjectStore log:datanucleus.autoStartMechanismMode is set to unsupported value null . Setting it to value: ignored thread:pool-15-thread-171]
2022-05-17 15:45:04.818 IST  map[class:metastore.HiveMetaStore log:177: Opening raw store with implementation class:org.apache.hadoop.hive.metastore.ObjectStore thread:pool-15-thread-171]
2022-05-17 15:45:04.819 IST  map[class:metastore.ObjectStore log:ObjectStore, initialize called thread:pool-15-thread-171]
2022-05-17 15:45:04.832 IST  map[class:metastore.MetaStoreDirectSql log:Using direct SQL, underlying DB is MYSQL thread:pool-15-thread-171]
2022-05-17 15:45:04.832 IST  map[class:metastore.ObjectStore log:Initialized ObjectStore thread:pool-15-thread-171]
2022-05-17 15:45:04.838 IST  map[class:listeners.SyncIamListener log:Going forward with RemoveIamPolicy execution, received root url: https://metastore.googleapis.com thread:pool-15-thread-171]
```