

Running Apache Spark jobs on Dataproc

Setting(google cloud shell , cloud Dataproc cluster)

Scenario

You are migrating an existing Spark workload to Cloud Dataproc and then progressively modifying the Spark code to make use of GCP native features and services.

Activate Google Cloud Shell

`gcloud auth list`

`gcloud config list project`

Check project permissions (see Appendix)

Configure and start a Cloud Dataproc cluster

- 1.In the GCP Console, on the **Navigation menu**, in the **Big Data** section, click **Dataproc**.
- 2.Click **Create Cluster**.
- 3.Enter sparktodb for **Cluster Name**.
- 4.In the **Versioning** section, click **Change** and select **2.0 (Debian 10, Hadoop 3.2, Spark 3.1)**.

This version includes Python3 which is required for the sample code used in this lab.

Choose Image Version

STANDARD DATAPROC IMAGE

CUSTOM IMAGE

Cloud Dataproc uses versioned images to bundle the operating system, big data components, and Google Cloud Platform connectors into one package that is deployed on your cluster. [Learn more](#)

- ☒ 2.0 (Debian 10, Hadoop 3.2, Spark 3.1)
First released on 1/22/2021.
- ☐ 2.0 (Ubuntu 18.04 LTS, Hadoop 3.2, Spark 3.1)
First released on 1/22/2021.
- ☐ 1.5 (CentOS 8, Hadoop 2.10, Spark 2.4)
First released on 11/4/2020.
- ☐ 1.5 (Debian 10, Hadoop 2.10, Spark 2.4)
First released on 3/25/2020.
- ☐ 1.5 (Ubuntu 18.04 LTS, Hadoop 2.10, Spark 2.4)
First released on 3/25/2020.
- ☐ 1.4 (Debian 10, Hadoop 2.9, Spark 2.4)
First released on 3/22/2019.
- ☐ 1.4 (Ubuntu 18.04 LTS, Hadoop 2.9, Spark 2.4)
First released on 3/22/2019.
- ☐ 1.3 (Debian 10, Hadoop 2.9, Spark 2.3)
First released on 8/16/2018.
- ☐ 1.3 (Ubuntu 18.04 LTS, Hadoop 2.9, Spark 2.3)
First released on 3/22/2019.

SELECT

CANCEL

5. Click **Select**.

6. In the **Components > Component gateway** section, select **Enable component gateway**.

7. Under **Optional components**, Select **Jupyter Notebook**.

8. Click **Create**.

The cluster should start in a couple of minutes. You can proceed to the next step without waiting for the Cloud Dataproc Cluster to fully deploy

The screenshot shows the 'Create a cluster' wizard in Google Cloud. The left sidebar contains a list of steps: 'Set up cluster' (active), 'Configure nodes (optional)', 'Customize cluster (optional)', and 'Manage security (optional)'. The main area is divided into sections: 'Name' (Cluster Name: sparktddp), 'Location' (Region: us-central1, Zone: us-central1-c), 'Cluster type' (Standard selected), 'Autoscaling' (Policy: None), 'Versioning' (Image Type and Version: 2.0-debian10, Release Date: 1/22/2021), 'Components' (Component Gateway: Enable component gateway checked), and 'Optional components' (Jupyter Notebook checked). Annotations with green text and blue arrows point to specific elements: 'Name cluster' points to the Cluster Name field; 'Select version' points to the Versioning section; 'Enable component gateway' points to the 'Enable component gateway' checkbox; and 'Optional components' points to the 'Optional components' section.

Create a cluster

- Set up cluster
Begin by providing basic information.
- Configure nodes (optional)
Change node compute and storage capabilities.
- Customize cluster (optional)
Add cluster properties, features, and actions.
- Manage security (optional)
Change access, encryption, and security settings.

CREATE CANCEL

Equivalent [REST](#) or [command line](#)

Name
Cluster Name *
sparktddp

Location
Region *
us-central1
Zone *
us-central1-c

Cluster type
☒ Standard (1 master, N workers)
☐ Single Node (1 master, 0 workers)
Provides one node that acts as both master and worker. Good for proof-of-concept or small-scale processing.
☐ High Availability (3 masters, N workers)
Hadoop High Availability mode provides uninterrupted YARN and HDFS operations despite single-node failures or reboots.

Autoscaling
Automates cluster resource management based on an autoscaling policy.
Policy
None

Versioning
Use a custom image to load pre-installed packages. [Learn more](#)

Image Type and Version
2.0-debian10

Release Date
First released on 1/22/2021.
[CHANGE](#)

Components
Component Gateway
☒ Enable component gateway
Provides access to the web interfaces of default and selected optional components on the cluster. [Learn more](#)

Optional components
Select one or multiple components. [Learn more](#)

- ☐ Anaconda
- ☐ Hive WebHCat
- ☒ Jupyter Notebook
- ☐ Zeppelin Notebook
- ☐ Druid
- ☐ Presto
- ☐ ZooKeeper
- ☐ Ranger
- ☐ HBase
- ☐ Flink
- ☐ Docker
- ☐ Solr

Setting(get repository, specify dataproc storage ,copy notebook to jupyter working folder)

Clone the source repository for the lab

In the Cloud Shell you clone the Git repository for the lab and copy the required notebook files to the Cloud Storage bucket used by Cloud Dataproc as the home directory for Jupyter notebooks.

1.To **clone the Git repository** for the lab enter the following command in Cloud Shell:

git -C ~ clone <https://github.com/GoogleCloudPlatform/training-data-analyst>

2. To **locate** the default **Cloud Storage bucket used by Cloud Dataproc** enter the following command in Cloud Shell:

export DP_STORAGE="gs://\$(gcloud dataproc clusters describe sparktodp --region=us-central1 --format=json | jq -r '.config.configBucket')"

3. To **copy** the sample **notebooks** into **the Jupyter working folder** enter the following command in Cloud Shell:

gsutil -m cp ~/training-data-analyst/quests/sparktobq/*.ipynb \$DP_STORAGE/notebooks/jupyter

Log in to the Jupyter Notebook

As soon as the cluster has fully started up you can connect to the Web interfaces. Click the refresh button to check as it may be deployed fully by the time you reach this stage.

1. On the Dataproc Clusters page wait for the cluster to finish starting and then click the name of your cluster to open the **Cluster details** page.

2. Click **Web Interfaces**.

3. Click the **Jupyter** link to open a new Jupyter tab in your browser.

This opens the Jupyter home page. Here you can see the contents of the /notebooks/jupyter directory in Cloud Storage that now includes the sample Jupyter notebooks used in this lab.

4. Under the **Files** tab, click the **GCS** folder and then click **01_spark.ipynb** notebook to open it.

5. Click **Cell** and then **Run All** to run all of the cells in the notebook.

6. Page back up to the top of the notebook and follow as the notebook completes runs each cell and outputs the results below them.

Jupyter Notebook(fetch data, copy to Hadoop, list of data)

The first code cell fetches the source data file, which is an extract from the KDD Cup competition from the Knowledge, Discovery, and Data (KDD) conference in 1999. The data relates to computer intrusion detection events.

```
!wget https://archive.ics.uci.edu/ml/machine-learning-databases/kddcup99-mld/kddcup.data\_10\_percent.gz
```

In the second code cell, the source data is copied to the default (local) Hadoop file system.

```
!hadoop fs -put kddcup* /
```

In the third code cell, the command lists contents of the default directory in the cluster's HDFS file system.

```
!hadoop fs -ls /
```

Jupyter Notebook(Reading in data)

The data are gzipped CSV files. In Spark, these can be read directly using the `textFile` method and then parsed by splitting each row on commas.

The Python Spark code starts in cell In[4]. In this cell Spark SQL is initialized and Spark is used to read in the source data as text and then **returns the first 5 rows**.

```
from pyspark.sql import SparkSession, SQLContext, Row
spark = SparkSession.builder.appName("kdd").getOrCreate()
sc = spark.sparkContext
data_file = "hdfs:///kddcup.data_10_percent.gz"
raw_rdd = sc.textFile(data_file).cache()
raw_rdd.take(5)
```

Jupyter Notebook(clean data)

In cell In [5] each row is split, using , as a delimiter and parsed using a prepared inline schema in the code.

```
csv_rdd = raw_rdd.map(lambda row: row.split(","))
parsed_rdd = csv_rdd.map(lambda r: Row(
    duration=int(r[0]),
    protocol_type=r[1],
    service=r[2],
    flag=r[3],
    src_bytes=int(r[4]),
    dst_bytes=int(r[5]),
    wrong_fragment=int(r[7]),
    urgent=int(r[8]),
    hot=int(r[9]),
    num_failed_logins=int(r[10]),
    num_compromised=int(r[12]),
    su_attempted=r[14],
    num_root=int(r[15]),
    num_file_creations=int(r[16]),
    label=r[-1]
))
parsed_rdd.take(5)
```


Jupyter Notebook(Spark analysis)

In cell In [6] a Spark SQL context is created and a Spark dataframe using that context is created using the parsed input data from the previous stage. Row data can be selected and displayed using the dataframe's .show() method to output a view summarizing a count of selected fields.

```
sqlContext = SQLContext(sc)
df = sqlContext.createDataFrame(parsed_rdd)
connections_by_protocol =
df.groupBy('protocol_type').count().orderBy('count',
ascending=False)
connections_by_protocol.show()
```

protocol_type	count
icmp	283602
Tcp	190065
udp	20354

Jupyter Notebook(SparkSQL to Dataframe)

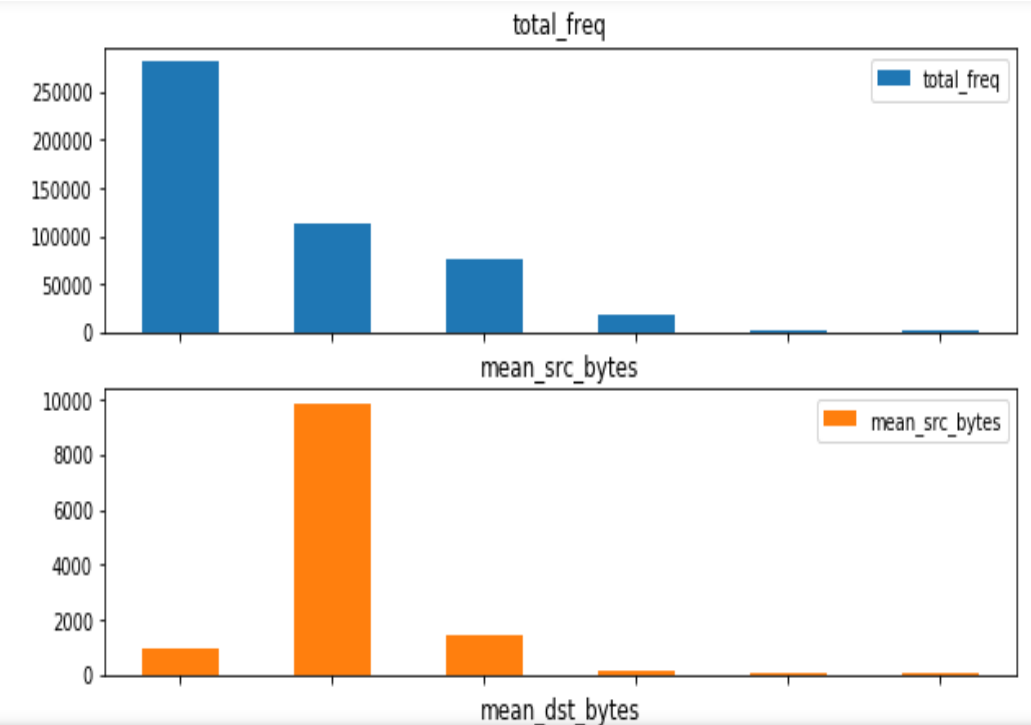
```
df.registerTempTable("connections")
attack_stats = sqlContext.sql("""
SELECT
    protocol_type,
    CASE label
        WHEN 'normal.' THEN 'no attack'
        ELSE 'attack'
    END AS state,
    COUNT(*) as total_freq,
    ROUND(AVG(src_bytes), 2) as mean_src_bytes,
    ROUND(AVG(dst_bytes), 2) as mean_dst_bytes,
df.registerTempTable("connections")
attack_stats = sqlContext.sql("""
SELECT
    protocol_type,
    CASE label
        WHEN 'normal.' THEN 'no attack'
        ELSE 'attack'
    END AS state,
    COUNT(*) as total_freq,
    ROUND(AVG(src_bytes), 2) as mean_src_bytes,
    ROUND(AVG(dst_bytes), 2) as mean_dst_bytes,
    ROUND(AVG(duration), 2) as mean_duration,
    SUM(num_failed_logins) as total_failed_logins,
```

```
    SUM(num_compromised) as total_compromised,
    SUM(num_file_creations) as total_file_creations,
    SUM(su_attempted) as total_root_attempts,
    SUM(num_root) as total_root_acceses
FROM connections
GROUP BY protocol_type, state
ORDER BY 3 DESC
""")
attack_stats.show()
    ROUND(AVG(duration), 2) as
mean_duration,
    SUM(num_failed_logins) as total_failed_logins,
    SUM(num_compromised) as total_compromised,
    SUM(num_file_creations) as total_file_creations,
    SUM(su_attempted) as total_root_attempts,
    SUM(num_root) as total_root_acceses
FROM connections
GROUP BY protocol_type, state
ORDER BY 3 DESC
""")
attack_stats.show()
```

Jupyter Notebook(SparkSQL to Dataframe, viz)

protocol_type	state	Total-freq	Mean_src_bytes
icmp	Attack	282314	932.14
Tcp	Attack	113252	9880.38
tcp	No attack	76813	1439.31
...
...
udp	Attack	1177	27.5

```
%matplotlib inline  
ax = attack_stats.toPandas().plot.bar(x='protocol_type',  
subplots=True, figsize=(10,25))
```



Cloud storage instead of HDFS(copy data to new storage bucket)

Modify Spark jobs to use Cloud Storage instead of HDFS

Taking this original 'Lift & Shift' sample notebook you will now create a copy that decouples the storage requirements for the job from the compute requirements. In this case, all you have to do is replace the Hadoop file system calls with Cloud Storage calls by replacing `hdfs://` storage references with `gs://` references in the code and adjusting folder names as necessary.

You start by using the cloud shell to place a copy of the source data in a new Cloud Storage bucket.

1. In the Cloud Shell **create a new storage bucket** for your source data.

```
export PROJECT_ID=$(gcloud info --format='value(config.project)')
```

```
gsutil mb gs://$PROJECT_ID
```

2. In the Cloud Shell **copy the source data into the bucket**.

```
wget https://archive.ics.uci.edu/ml/machine-learning-databases/kddcup99-mld/kddcup.data_10_percent.gz
```

```
gsutil cp kddcup.data_10_percent.gz gs://$PROJECT_ID/
```

Cloud storage instead of HDFS(change code for reading from google cloud storage)

3.Switch back to the 01_spark Jupyter Notebook tab and **Make a Copy with name De-couple-storage** and close first one ,**delete first three cells** because we work with google cloud storage .

4 .replace code in cell 4 with the following code

```
from pyspark.sql import SparkSession, SQLContext, Row
gcs_bucket='[Your-Bucket-Name]'
spark = SparkSession.builder.appName("kdd").getOrCreate()
sc = spark.sparkContext
data_file = "gs://" + gcs_bucket + "//kddcup.data_10_percent.gz"
raw_rdd = sc.textFile(data_file).cache()
raw_rdd.take(5)
```

Deploy Spark Jobs(Optimize Spark jobs to run on Job specific clusters)

3.Switch back to the 01_spark Jupyter Notebook tab and **Make a Copy with name PySpark-analysis-file** and close first one ,.

4 .insert cell above with following code

```
%%writefile spark_analysis.py
import matplotlib
matplotlib.use('agg')
import argparse
parser = argparse.ArgumentParser()
parser.add_argument("--bucket", help="bucket for input and output")
args = parser.parse_args()
BUCKET = args.bucket
```

Deploy Spark Jobs(Optimize Spark jobs to run on Job specific clusters)

The `%%writefile spark_analysis.py` Jupyter magic command creates a new output file to contain your standalone python script. You will add a variation of this to the remaining cells to append the contents of each cell to the standalone script file.

This code also imports the matplotlib module and explicitly sets the default plotting backend via `matplotlib.use('agg')` so that the plotting code runs outside of a Jupyter notebook.

10. For the remaining cells insert `%%writefile -a spark_analysis.py` at the start of each Python code cell. These are the five cells labelled **In [x]**.

```
%%writefile -a spark_analysis.py
```

For example the next cell should now look as follows.

```
%%writefile -a spark_analysis.py
from pyspark.sql import SparkSession, SQLContext, Row
spark = SparkSession.builder.appName("kdd").getOrCreate()
sc = spark.sparkContext
data_file = "gs://{}/kddcup.data_10_percent.gz".format(BUCKET)
raw_rdd = sc.textFile(data_file).cache()
#raw_rdd.take(5)
```

Deploy Spark Jobs(Optimize Spark jobs to run on Job specific clusters)

11.Repeat this step, inserting `%%writefile -a spark_analysis.py` at the start of each code cell until you reach the end.

12.In the last cell, where the Pandas bar chart is plotted remove the `%matplotlib` inline magic command.

Note: You must remove this inline matplotlib Jupyter magic directive or your script will fail when you run it.

13.Make sure you have selected the last code cell in the notebook then, in the menu bar, click **Insert** and select **Insert Cell Below**.

14.Paste the following code into the new cell.

```
%%writefile -a spark_analysis.py  
ax[0].get_figure().savefig('report.png');
```

15.Add another new cell at the end of the notebook and paste in the following:

```
%%writefile -a spark_analysis.py  
import google.cloud.storage as gcs  
bucket = gcs.Client().get_bucket(BUCKET)  
for blob in bucket.list_blobs(prefix='sparktodb/'):   
    blob.delete()  
bucket.blob('sparktodb/report.png').upload_from_filename('report.png')
```


Test Automation

You now test that the PySpark code runs successfully as a file by calling the local copy from inside the notebook, passing in a parameter to identify the storage bucket you created earlier that stores the input data for this job. The same bucket will be used to store the report data files produced by the script.

1. In the **PySpark-analysis-file** notebook add a **new cell** at the end of the notebook and paste in the following:

```
BUCKET_list = !gcloud info --format='value(config.project)'  
BUCKET=BUCKET_list[0]  
print('Writing to {}'.format(BUCKET))  
!/opt/conda/miniconda3/bin/python spark_analysis.py --bucket=$BUCKET
```

This code assumes that you have followed the earlier instructions and created a Cloud Storage Bucket using your lab Project ID as the Storage Bucket name. If you used a different name modify this code to set the BUCKET variable to the name you used.

2. Add a **new cell** at the end of the notebook and paste in the following:

```
!gsutil ls gs://$BUCKET/sparktodp/**
```

This lists the script output files that have been saved to your Cloud Storage bucket.

Test Automation

3.To save a copy of the Python file to persistent storage, add a **new cell** and paste in the following:

```
!gsutil cp spark_analysis.py gs://$BUCKET/sparktodb/spark_analysis.py
```

4.Run All

```
!gsutil ls gs://$BUCKET/sparktodb/**
```

```
gs://qwiklabs-gcp-00-b0ac3d99ccd1/sparktodb/connections_by_protocol/  
gs://qwiklabs-gcp-00-b0ac3d99ccd1/sparktodb/connections_by_protocol/_SUCCESS  
gs://qwiklabs-gcp-00-b0ac3d99ccd1/sparktodb/connections_by_protocol/part-00000-b1575b34-d708-46d9-b04f-e487b2a90  
59c-c000.csv  
gs://qwiklabs-gcp-00-b0ac3d99ccd1/sparktodb/connections_by_protocol/part-00001-b1575b34-d708-46d9-b04f-e487b2a90  
59c-c000.csv  
gs://qwiklabs-gcp-00-b0ac3d99ccd1/sparktodb/connections_by_protocol/part-00002-b1575b34-d708-46d9-b04f-e487b2a90  
59c-c000.csv  
gs://qwiklabs-gcp-00-b0ac3d99ccd1/sparktodb/report.png
```

```
!gsutil cp spark_analysis.py gs://$BUCKET/sparktodb/spark_analysis.py
```

```
Copying file:///spark_analysis.py [Content-Type=text/x-python]...  
/ [1 files][ 2.8 KiB/ 2.8 KiB]  
Operation completed over 1 objects/2.8 KiB.
```

Test Automation(Run the Analysis Job from Cloud Shell)

1.Switch back to your Cloud Shell and copy the Python script from Cloud Storage so you can run it as a Cloud Dataproc Job.

```
gsutil cp gs://$PROJECT_ID/sparktdp/spark_analysis.py spark_analysis.py
```

2. Create a launch script.

```
nano submit_onejob.sh
```

3. Paste the following into the script:

```
#!/bin/bash
```

```
gcloud dataproc jobs submit pyspark \
```

```
  --cluster sparktdp \
```

```
  --region us-central1 \
```

```
  spark_analysis.py \
```

```
  -- --bucket=$1
```

4.Press **CTRL+X** then **Y** and Enter key to exit and save.

5.Make the script executable:

Launch job

Wait job ,show
output in storage

Delete cluster

6. **Launch** the PySpark Analysis **job**:
./submit_onejob.sh \$PROJECT_ID

7. In the Cloud Console tab navigate to the **Dataproc > Clusters** page if it is not already open.

8. Click **Jobs** , **wait until job finished**.

10. Navigate to your **storage bucket** and note that the output report, **/sparktodb/report.png** has an updated time-stamp indicating that the stand-alone job has completed successfully.

The storage bucket used by this Job for input and output data storage is the bucket that is used just the Project ID as the name.

11. Navigate back to the **Dataproc > Clusters** page.

12. Select the **sparktodb** cluster and click **Delete**. You don't need it any more.


13. Click **CONFIRM**.

14. Close the **Jupyter** tabs in your browser.

appendix

Check project permissions

Before you begin your work on Google Cloud, you need to ensure that your project has the correct permissions within Identity and Access Management (IAM).

1. In the Google Cloud console, on the **Navigation menu** (), click **IAM & Admin > IAM**.
2. Confirm that the default compute Service Account {project-number}-compute@developer.gserviceaccount.com is present and has the editor role assigned. The account prefix is the project number, which you can find on **Navigation menu > Home**.

IAM & Admin

IAM

- Identity & Organization
- Policy Troubleshooter
- Policy Analyzer
- Organization Policies
- Service Accounts
- Workload Identity Federat...
- Labels
- Tags
- Settings
- Privacy & Security
- Identity-Aware Proxy
- Roles
- Audit Logs

IAM

ADD

REMOVE

PERMISSIONS

RECOMMENDATIONS HISTORY

Permissions for project "qwiklabs-gcp-03-e30ac90a32e4"

These permissions affect this project and all of its resources. [Learn more](#)

View By:

PRINCIPALS

ROLES

Filter Enter property name or value

<input type="checkbox"/> Type	Principal	Name	Role	Se
<input type="checkbox"/>	407543585891-compute@developer.gserviceaccount.com	Compute Engine default service account	Editor	
<input type="checkbox"/>	407543585891@cloudbuild.gserviceaccount.com		Cloud Build Service Account	
<input type="checkbox"/>	407543585891@cloudservices.gserviceaccount.com	Google APIs Service Agent	Editor	
<input type="checkbox"/>	admiral@qwiklabs-services-prod.iam.gserviceaccount.com		Owner	
<input type="checkbox"/>	qwiklabs-gcp-03-e30ac90a32e4@qwiklabs-gcp-03-e30ac90a32e4.iam.gserviceaccount.com	Qwiklabs User Service Account	App Engine Admin BigQuery Admin	

If the account is not present in IAM or does not have the editor role, follow the steps below to assign the required role.

- In the Google Cloud console, on the **Navigation menu**, click **Home**.
- Copy the project number (e.g. 729328892908).
- On the **Navigation menu**, click **IAM & Admin > IAM**.
- At the top of the **IAM** page, click **Add**.
- For **New principals**, type:
 - {project-number}-compute@developer.gserviceaccount.com

Replace {project-number} with your project number.

- For **Role**, select **Project (or Basic) > Editor**. Click **Save**.