Leveraging Unstructured Data - Lab 4 : Leverage GCP v1.3

## Overview

In this lab, you will explore Spark using PySpark jobs, use Cloud Storage instead of HDFS, and run a PySpark application from Cloud Storage.

## Objectives

In this lab, you will perform the following tasks:

* Explore Spark using PySpark jobs
* Using Cloud Storage instead of HDFS
* Run a PySpark application from Cloud Storage
* Using Python Pandas to add BigQuery to a Spark application

## Task 1: Prepare the Master Node and the Bucket

### **Connect to the Master Node**

A Dataproc cluster has been prepared for you. You will be performing most of the lab steps from the Master Node of the cluster in an SSH terminal window.

1. In the Console, on the **Navigation menu** (7a91d354499ac9f1.png) click **Dataproc** > **Clusters**.
2. Locate the cluster named **dataproc-cluster**.
3. Click on the name **dataproc-cluster** to go to the Cluster details page.
4. The Cluster details page opens to the **Monitoring** tab. Click on the tab labeled **VM Instances**.
5. On the line for the VM named **dataproc-cluster-m**, you will see that it has the Role of Master and there is an SSH link next to it. Click on **SSH** to open a terminal window to the Master Node.

### **Copy sample files to the Master Node home directory**

The sample files you need have already been archived on the Master Node. You will need to copy them into your user directory with the following command.

1. In the Master Node SSH terminal window.

cd

cp -r /training .

ls

1. You should now have the **/training** directory in your home directory. And it should have files within it.

### **Set the $BUCKET environment variable**

A Cloud Storage bucket has already been created for you. It has the same name as the Project ID. You will create an environment variable to make it easy to reference the bucket from the command line on the Master Node.

1. In the Console, on the **Navigation menu** (7a91d354499ac9f1.png) click **Storage** > **Browser**. Locate the bucket with the same name as the Project ID. Highlight the bucket name and use it to set the environment variable.
2. In the Master Node SSH terminal window, enter:

BUCKET=<bucket-name>

echo $BUCKET

You can use $BUCKET in CLI commands and retrieve the name with **echo $BUCKET**.

## Task 2. The two letter lab

This is called the "two letter lab" because it only involves adding the two letters "gs" to the front of a file reference to tell Spark to use Cloud Storage instead of HDFS.

HDFS is based on a white paper that was published in 2003 on Google File Storage (GFS), an early ancestor of Cloud Storage. Cloud Storage can be used in place of HDFS for great benefit. Dataproc clusters have the connector to Cloud Storage already installed by default.

**Why would you want to use Cloud Storage instead of HDFS?**

* You can shut down the cluster when you are not running jobs. The storage persists even when the cluster is shut down, so you don't have to pay for the cluster just to maintain data in HDFS.
* In some cases Cloud Storage provides better performance than HDFS.
* Cloud Storage does not require the administration overhead of a local file system.

### **Replace HDFS with GS**

1. Place a copy of your sample data file in a Cloud Storage bucket instead of HDFS.
2. In the Master Node terminal window, enter the following gsutil command to copy the sample text files to the Cloud Storage bucket.

gsutil cp /training/road-not-taken.txt gs://$BUCKET

1. In the SSH terminal for the Master Node, use **nano** or **vi** to create the file **wordcount.py**.
2. Copy and paste the following code into the file.

from pyspark.sql import SparkSession

from operator import add

import re

print("Okay Google.")

spark = SparkSession\

.builder\

.appName("CountUniqueWords")\

.getOrCreate()

lines = spark.read.text("/sampledata/road-not-taken.txt").rdd.map(lambda x: x[0])

counts = lines.flatMap(lambda x: x.split(' ')) \

.filter(lambda x: re.sub('[^a-zA-Z]+', '', x)) \

.filter(lambda x: len(x)>1 ) \

.map(lambda x: x.upper()) \

.map(lambda x: (x, 1)) \

.reduceByKey(add) \

.sortByKey()

output = counts.collect()

for (word, count) in output:

print("%s = %i" % (word, count))

spark.stop()

This PySpark application performs word counting on data contained in HDFS. Notice that the application currently expects to find the data in HDFS in a directory called **/sampledata**. You will modify the program to work on the data you just uploaded to Cloud Storage.

1. First, verify that the data file does not exist in HDFS.

hadoop fs -ls /

There is no directory **/sampledata** in HDFS on this cluster.

1. Next, use the Hadoop file system command to view the files through the hadoop connector to Cloud Storage. This verifies that the connector is working and that the file is available in the bucket.

hadoop fs -ls gs://$BUCKET

1. Edit **wordcount.py** in **nano** or **vi**.

Replace this line that refers to the file in HDFS.

lines = spark.read.text("/sampledata/road-not-taken.txt").rdd.map(lambda x: x[0])

With a line the refers to the file in Cloud Storage. Remember to remove "/sampledata" because that directory does not exist. Remember to use the actual bucket name and not the environment variable. The Worker Nodes on the cluster where the program will run do not know the value of the local environment variable on the Master Node.

lines = spark.read.text("gs://<YOUR-BUCKET>/road-not-taken.txt").rdd.map(lambda x: x[0])

1. Run the job.

spark-submit wordcount.py

Observe that the application works although the data is not located in HDFS. HDFS is still being used for temporary files during processing. Input and output data can be located in Cloud Storage. The reason this is possible is because the Google Cloud Network provides super fast data transfer so that off-cluster storage is nearly as fast, and in some cases faster than, on-cluster storage.

Leveraging this GCP feature makes the cluster stateless, because all the persistent data is kept off-cluster. And that means (a) the cluster can be shut down when not in use, solving the Hadoop utilization problem, and (b) a cluster can be created and dedicated to a single job, solving the Hadoop configuration and tuning problem.

## Task 3. Run a PySpark application from Cloud Storage

### **Stage a program in Cloud Storage and run it**

In the previous task, you created a PySpark application in a development environment (on the Master Node). You tested the application using **spark-submit**.

In this task, you will migrate the application from the development environment to a production environment. You will stage the working application file in Cloud Storage. And you will run the production job from Console.

1. In the Master Node terminal, use the following command to copy the tested **wordcount.py** PySpark application to the bucket.

gsutil cp wordcount.py gs://$BUCKET

1. In the Console, on the **Navigation menu** (7a91d354499ac9f1.png) click **Dataproc** > **Clusters**. Take note of the region where the cluster is located. You will need that in the next steps.
2. You will also need the bucket name. You can also retrieve the bucket name from the Master Node terminal by entering the following. Highlight the bucket name and copy it.

echo $BUCKET

You can also find the bucket name in the Console. On the **Navigation menu** (7a91d354499ac9f1.png) click **Storage** > **Browser**.

1. In the Console, on the **Navigation menu** (7a91d354499ac9f1.png) click **Dataproc** > **Jobs**.
2. Click **Submit Job**.
3. Specify the following, and leave the remaining settings as their defaults:

|  |  |
| --- | --- |
| **Property** | **Value**  (type value or select option as specified) |
| **Region** | <your-region> |
| **Cluster** | dataproc-cluster |
| **Job type** | PySpark |
| **Main Python file** | gs://<your bucket>/wordcount.py |

1. Click **Submit**.
2. You can view the progress in the **Dataproc** > **Jobs** page in Console.