**GCP Use Case**

Creating and Running Pyspark/Hive Jobs on DataProc Cluster and automating it using Airflow DAGs

Dataproc is a fully managed and highly scalable service that allows you within other things to set up a cluster and run your Pyspark/Spark/hadoop/SQL jobs on Google cloud platform.

Data Sources: Any Vehicle related data available on Public Datasets / Create a sample dataset.

Took a sample automobile dataset from Kaggle (<https://www.kaggle.com/datasets/fazilbtopal/auto85?resource=download&select=auto.csv>)

Pre-Requisites :

* Install **Gcloud SDK** (Libraries and tools for interacting with Google Cloud products and services) and authorize. Follow the link below and click on Guide Me for step-by-step instructions.

(<https://cloud.google.com/sdk/docs/install-sdk>)

(<https://cloud.google.com/sdk/gcloud/reference/auth/login>)

* Install **Gsutil** (a tool that enables you to access Cloud Storage from the command-line using HTTPS.) Follow the link below for instructions.

(<https://cloud.google.com/storage/docs/gsutil_install>)

* Create a **Storage bucket in your GCP project**.

The data files we will be using are stored in a GCS bucket.

How Spark executes a Job:

* We execute a spark job through **spark submit** command.
* After that we specify the config options, give the Pyspark scripts etc., At this point of time we specify the **deployMode**. It can be either **client mode** or **cluster mode**.
* When the job is submitted, then two processes begin: **driver program** and **executor program**
* **Driver program** is the one which executes the spark job. It runs on one machine.
* **Executor program** are spawned through the driver program. These are executed on different nodes on the cluster. These are the worker nodes in which data processing is done.
* When a user submits the spark job, it goes to the **edge node** which is a gateway to the cluster. Edge node is also a part of the cluster.
* **Client mode** -> The **driver program is spawned on the same machine where we are running the spark submit job**. Meaning it runs on the same edge node and use the resources of the edge node (memory, CPU utilization etc.,)
* The problem in the client mode is if a different user submitted different jobs through client mode, then the edge node gets overloaded and gives an out of memory error because it is running on a single machine and the resources aren’t available. Whenever we get this error the driver program will exit and terminates spark job. So, we always deploy the spark jobs on production using cluster mode.
* **Cluster mode** ->in this mode, **edge node will take the job and will spawn the driver program on any available machine onto the cluster**.This way the resources are used efficiently.

Process for Setting up Dataproc Jobs:

1. We need to install Google SDK to run CLI commands.
2. Our data and the scripts we use to trigger the Pyspark job should be available in a storage bucket on GCS.
3. When we are ready with the code and other dependency files, create a Dataproc cluster on GCP.
4. Submit the Dataproc job. We can do it in three different ways:

a - Using cloud console, b - Using Gcloud, c - Using SSH

In this Demo we are doing it using Google cloud console. For the other options we can get the code on the same page.

Note:

* You can submit any job based on your requirements. (Hadoop/Spark/Pyspark/Hive/SparkSQL/Pig). Based on the job we are executing; additional properties must be specified.

Example: If we are submitting a Pyspark job, in the properties field, we will have to pass **spark.submit.deployMode : client.** The reason we do it in client mode is because we already have a dedicated master in our cluster so we will be using that machine for running the spark driver rather than working machines.

* If your data is from BigQuery you will have to use BigQuery connecter for the Dataproc

Steps to create DataProc Cluster:

1. Enable DataProc API.

2. Create a test Dataproc cluster using **Google Compute Engine** with the following selections: **Region:us-central1, Zone: us-central1-b, Cluster type: Standard, Autoscaling : None, versioning: (1.5 Ubuntu 18.04 LTS, Hadoop 2.10, Spark 2.4)**

The configure nodes we use general purpose : **N1 series with 2 vCPU** for both Master and worker nodes.

A screenshot of a computer

Description automatically generated

3. login to your gcloud using GCL and create a google cloud bucket -> open command prompt and type the following commands

"**gcloud auth login**" -> check your project name. if you want to change your project you can change it in your GCP

"**gsutil mb gs://<projectID>**" -> create a bucket with your projectID

NOTE (You can also create buckets directly inside GCP: "GCP/Cloud Storage/Buckets/Create")

A black screen with white text

Description automatically generated

4. Once you have the code ready, go to **Dataproc jobs -> create job -> give a job name, select your region, cluster**

- **Job type : pyspark**

- **main python file : upload your main.py file to your GCS bucket and then give your file location "gs://<bucket location>/main.py"**

**For Properties key : "spark.submit.deployMode" and for Value : "client" and run the spark job**.

5. If your output is a hive table, you can create a Hive Job to query the table. Go to Dataproc Jobs -> give a job name, give your cluster and

- **job type : Hive**

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

- **For query source type you can either give a SQL script location from GCS or you can select Query text and write a sample query in the text box below and submit the job**

1. Once the jobs are done go to **Dataproc Clusters - DELETE** the cluster and other unnecessary storage buckets.

A screenshot of a computer

Description automatically generated

**- Hive table is created on the Dataproc cluster. When you delete the cluster, the data is deleted as well. To avoid this, we can store the output as a parquet/any other format inside GCS.**

Now let’s try to automate the above process using Apache Airflow.

Airflow:

Apache Airflow is an open-source platform for authoring, scheduling, and monitoring data and computing workflows.

We need Google Cloud Composer to work with Airflow Environment.

Airflow is created on top of Google Kubernetes Engine

Steps to Create Airflow Environment:

1. Go to Google cloud composer.
2. Create Environment -> Composer1
3. Give a name, -> Location : us-central1, Disk size : 30GB as minimum.
4. Leave everything else to Default

It takes a while for this environment to be created. Once it is created, go to environment configuration, and check your **DAGs folder**. This bucket is related to Airflow related files inside GCS.

A screenshot of a computer

Description automatically generated

1. We need to copy the **dag.py** file into this GCS location to trigger the run.
2. Once you upload your file, you can trigger the DAG run from Airflow UI

A screenshot of a computer

Description automatically generated

1. When you trigger the DAG and click on the DAG name, you can see different views.

A screenshot of a computer

Description automatically generated

The current running part is shown with a Green border.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

1. Here my DAG is basically creating a Dataproc cluster and then it triggers the same pyspark job used in the previous example and once done, it deletes the cluster.
2. Since we are running it on Airflow, this process is automated. We can also schedule the DAG script to make the runs as scheduled or repetitive.
3. Once everything is done and verified you can Delete the Airflow environment to avoid costs.

DAG scripts are customizable. We can create different DAGs for different environments like Dev/Staging/Prod and then schedule them however we want.

To copy files from local to GCS bucket on a scheduled basis:

On Windows:

1. Open terminal

2. Create a shell script with the gsutil cp command

"**gsutil cp -r C:/Users/bhanu/Desktop/Miracle/GCP\_Usecase/scripts/source/\* gs://usecase-swift-hope-391116**"

3. To test the shell script, navigate it to your directory where you created shell script and use this command in terminal to trigger "bash backup\_script.sh". If you get an error regarding 'bash' not found, you will have to install gitbash

4. To schedule it on windows go to task scheduler -> Create basic task -> Give a name/description -> For trigger give Daily -> Setup a start date and time -> Action is Start a program -> For the program give the shell script path.

5. This creates an xml file in the background which we can use for scheduling tasks. We can edit the xml file once its created.

6. Once done you can go to GCS bucket and verify if the files are copied

NOTE:

If you want to edit/delete your task, you must find your task. It may be difficult to find your task as there are so many of them. You can search in this location if you cannot find your task. “**C:\WINDOWS\system32\Tasks\taskname.xml**”