

# CSCI 5408

## DATA MANAGEMENT AND WAREHOUSING



## LAB ASSIGNMENT - 5

**Submitted By:** Kenil Shaileshkumar Patel  
(kenil.patel@dal.ca)  
**Banner ID:** B00954251  
**Submitted On:** November 04, 2023

### **Gitlab Repository Link**

[https://git.cs.dal.ca/kenil/csci5408\\_f23\\_b00954251\\_kenil\\_patel/-/tree/main/Lab5](https://git.cs.dal.ca/kenil/csci5408_f23_b00954251_kenil_patel/-/tree/main/Lab5)

## Table of Contents

Sr. No	Title	Page No.
1.	Setting up Apache Spark on GCP instances	3
2.	Spark Program (Python)	5
3.	References	9

# Setting up Apache Spark on GCP instances

Created a Dataproc cluster on computer engine and also modified the nodes

Start your free trial with \$300 in credit. Don't worry – you won't be charged if you run out of credit. [Learn more](#)

DISMISS START FREE

Google Cloud CSCI-5408-F23 dataproc Search

Dataproc

Jobs on clusters

- Clusters
- Jobs
- Workflows
- Auto-scaling policies

Serverless

- Batches
- Interactive

Metastore services

- Metastore
- Federation

Utilities

- Component exchange
- Release notes

Create a Dataproc cluster on Compute Engine

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

CREATE CANCEL

EQUIVALENT COMMAND LINE

Name

Cluster name \*  
weatherapi-cluster

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

Versioning

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

Image type and version  
2.1-debian11

Release date

<https://console.cloud.google.com/?project=csc-5408-f23-402321>

Figure 1: Setting Up the Cluster.

The cluster for the weatherapi is created and it is successfully running.

Start your free trial with \$300 in credit. Don't worry – you won't be charged if you run out of credit. [Learn more](#)

DISMISS START FREE

Google Cloud CSCI-5408-F23 dataproc Search

Dataproc

Clusters

CREATE CLUSTER REFRESH START STOP DELETE REGIONS + 5 RECOMMENDED ALERTS SHOW INFO PANEL

Filter Search clusters, press Enter

	Name	Status	Region	Zone	Total worker nodes	Scheduled deletion	Cloud Storage staging bucket	Created
<input type="checkbox"/>	weatherapi-cluster	Running	us-central1	us-central1-c	2	Off	<a href="#">dataproc-staging-us-central1-806435822560-j3yawd6</a>	3 Nov 2023, 23:16:48

Figure 2: weatherapi-cluster is running.

In the cluster, there are three VM instances in which one master node and other two worker node.

The screenshot displays the Google Cloud Dataproc console interface. The left sidebar shows the navigation menu with 'Clusters' selected. The main panel shows the 'Cluster details' for 'weatherapi-cluster'. The 'VM INSTANCES' tab is active, displaying a table of three instances: one Master node and two Worker nodes. A notification banner at the top suggests using Auto Zone. Below the table, there is an 'EQUIVALENT REST' section.

Start your free trial with \$300 in credit. Don't worry - you won't be charged if you run out of credit. [Learn more](#)

DISMISS START FREE

Google Cloud CSC1-5408-F23 dataproc Search

Dataproc Cluster details SUBMIT JOB REFRESH START STOP DELETE VIEW LOGS

Consider using Auto Zone rather than selecting a zone manually. See <https://cloud.google.com/dataproc/docs/concepts/configuring-clusters/auto-zone> MORE

Name weatherapi-cluster  
Cluster UUID 7e9aa3c5-73f8-4a20-8fa8-572917f70c8c  
Type Dataproc cluster  
Status Running

MONITORING JOBS VM INSTANCES CONFIGURATION WEB INTERFACES

Filter Filter instances

Name	Role	
<a href="#">weatherapi-cluster-m</a>	Master	SSH
<a href="#">weatherapi-cluster-w-0</a>	Worker	
<a href="#">weatherapi-cluster-w-1</a>	Worker	

EQUIVALENT REST

Figure 3: VM instances in the cluster.

## Spark Program (Python)

Write Spark program to filter the response data where the daily “feels\_like” temperature for the next 5-days is less than 15°C during the “day” time. Exclude the current, minutely, and hourly fields.

Below is the Python program which loads the data from the JSON file and processes the data. It filters the data where the daily “feels\_like” temperature for the next 5 days is less than 15°C during the “day” time.

```
weather_lab5.py M X
Lab5 > weather_lab5.py > ...
1 from pyspark.sql import SparkSession
2 from pyspark.sql.functions import col, expr
3
4
5 spark = SparkSession.builder.appName("FilterWeatherData") \
6     .config("spark.hadoop.fs.hdfs.impl", "org.apache.hadoop.fs.LocalFileSystem") \
7     .config("spark.hadoop.mapreduce.framework.name", "local") \
8     .config("spark.sql.warehouse.dir", "file:/tmp") \
9     .getOrCreate()
10
11
12
13 weather_data = spark.read.json("weather.json")
14
15
16 filtered_data = weather_data.select(
17     col("lat"),
18     col("lon"),
19     col("timezone"),
20     col("timezone_offset"),
21     col("daily.dt").alias("dt"),
22     col("daily.temp.day").alias("day_temp"),
23     expr("daily.feels_like.day[0]").alias("day_feels_like"),
24     expr("daily.weather[0].description").alias("description")
25 ).filter(col("day_feels_like") < 15)
26
27
28 filtered_data.write.json("fall_weather.json")
29
30 print("Filtered data saved to fall_weather.json")
31 spark.stop()
32
```

Figure 4: Spark Python Code

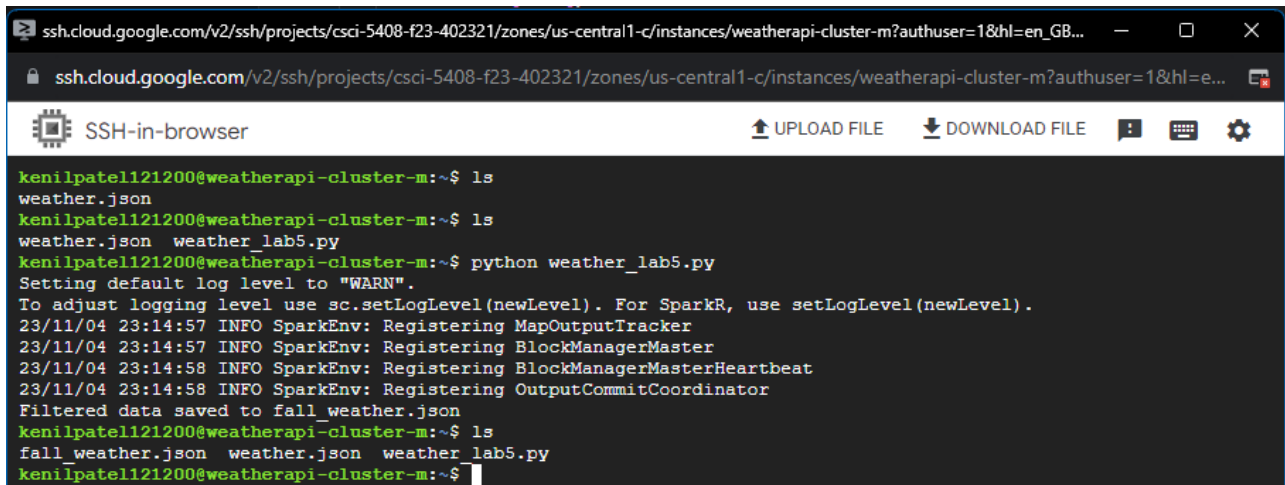
I have uploaded weather.json and weather\_lab5.py file.

A screenshot of a web browser window displaying an SSH-in-browser terminal. The address bar shows the URL: ssh.cloud.google.com/v2/ssh/projects/csci-5408-f23-402321/zones/us-central1-c/instances/weatherapi-cluster-m?authuser=1&hl=en... The browser interface includes a gear icon for settings, an 'SSH-in-browser' title, and buttons for 'UPLOAD FILE' and 'DOWNLOAD FILE'. The terminal window shows the user 'kenilpatel121200' at the 'weatherapi-cluster-m' prompt. The user enters 'ls' and the output is 'weather.json'. Then, the user enters 'ls' again, and the output is 'weather.json weather\_lab5.py'.

```
kenilpatel121200@weatherapi-cluster-m:~$ ls
weather.json
kenilpatel121200@weatherapi-cluster-m:~$ ls
weather.json  weather_lab5.py
kenilpatel121200@weatherapi-cluster-m:~$
```

Figure 5: File Uploading.

I am running the python file and the fall\_weather.json file is generated.

A screenshot of a web browser window displaying an SSH-in-browser terminal. The address bar shows the URL: ssh.cloud.google.com/v2/ssh/projects/csci-5408-f23-402321/zones/us-central1-c/instances/weatherapi-cluster-m?authuser=1&hl=en\_GB... The browser interface includes a gear icon for settings, an 'SSH-in-browser' title, and buttons for 'UPLOAD FILE' and 'DOWNLOAD FILE'. The terminal window shows the user 'kenilpatel121200' at the 'weatherapi-cluster-m' prompt. The user enters 'ls' and the output is 'weather.json'. Then, the user enters 'ls' again, and the output is 'weather.json weather\_lab5.py'. Finally, the user enters 'python weather\_lab5.py'. The output shows SparkEnv logging messages: 'Setting default log level to "WARN".', 'To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).', '23/11/04 23:14:57 INFO SparkEnv: Registering MapOutputTracker', '23/11/04 23:14:57 INFO SparkEnv: Registering BlockManagerMaster', '23/11/04 23:14:58 INFO SparkEnv: Registering BlockManagerMasterHeartbeat', and '23/11/04 23:14:58 INFO SparkEnv: Registering OutputCommitCoordinator'. The final output is 'Filtered data saved to fall\_weather.json'.

```
kenilpatel121200@weatherapi-cluster-m:~$ ls
weather.json
kenilpatel121200@weatherapi-cluster-m:~$ ls
weather.json  weather_lab5.py
kenilpatel121200@weatherapi-cluster-m:~$ python weather_lab5.py
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
23/11/04 23:14:57 INFO SparkEnv: Registering MapOutputTracker
23/11/04 23:14:57 INFO SparkEnv: Registering BlockManagerMaster
23/11/04 23:14:58 INFO SparkEnv: Registering BlockManagerMasterHeartbeat
23/11/04 23:14:58 INFO SparkEnv: Registering OutputCommitCoordinator
Filtered data saved to fall_weather.json
kenilpatel121200@weatherapi-cluster-m:~$ ls
fall_weather.json  weather.json  weather_lab5.py
kenilpatel121200@weatherapi-cluster-m:~$
```

Figure 6: Running the Python file.

Displaying the fall\_weather.json in command prompt.

```
kenilpatell121200@weatherapi-cluster-m:~$ cat fall_weather.json
{
  "lat": 44.6462,
  "lon": -63.5736,
  "timezone": "America/Halifax",
  "timezone_offset": -10800,
  "daily": [
    {
      "dt": 1655827200,
      "sunrise": 1655800138,
      "sunset": 1655856188,
      "moonrise": 1655786400,
      "moonset": 1655829840,
      "moon_phase": 0.75,
      "temp": {
        "day": 12.95,
        "min": 11.01,
        "max": 14.56,
        "night": 12.65,
        "eve": 14.12,
        "morn": 11.99
      },
      "feels_like": {
        "day": 12.68,
        "night": 12.22,
        "eve": 13.68,
        "morn": 11.75
      },
      "pressure": 1021,
      "humidity": 91,
      "dew_point": 11.33,
      "wind_speed": 2.66,
      "wind_deg": 1,
      "wind_gust": 4.9,
      "weather": [
        {
          "id": 500,
          "main": "Rain",
          "description": "light rain",
          "icon": "10d"
        }
      ],
      "clouds": 100,
      "pop": 0.35,
      "rain": 0.73,
      "uvi": 4.11
    }
  ]
}
```

Figure 7: Displaying Filtered Data.

Saved the filtered data i.e. fall\_weather.json in local computer.

```
weather_lab5.py  fall_weather.json  weather.json
{} fall_weather.json > [ ] daily > {} 0 > {} temp
1  {
2    "lat": 44.6462,
3    "lon": -63.5736,
4    "timezone": "America/Halifax",
5    "timezone_offset": -10800,
6    "daily": [
7      {
8        "dt": 1655827200,
9        "sunrise": 1655800138,
10       "sunset": 1655856188,
11       "moonrise": 1655786400,
12       "moonset": 1655829840,
13       "moon_phase": 0.75,
14       "temp": {
15         "day": 12.95,
16         "min": 11.01,
17         "max": 14.56,
18         "night": 12.65,
19         "eve": 14.12,
20         "morn": 11.99
21       },
22       "feels_like": {
23         "day": 12.68,
24         "night": 12.22,
25         "eve": 13.68,
26         "morn": 11.75
27       },
28       "pressure": 1021,
29       "humidity": 91,
30       "dew_point": 11.33,
31       "wind_speed": 2.66,
32       "wind_deg": 1,
33       "wind_gust": 4.9,
34       "weather": [
35         {
36           "id": 500,
37           "main": "Rain",
38           "description": "light rain",
39           "icon": "10d"
40         }
41       ],
42       "clouds": 100,
43       "pop": 0.35,
44       "rain": 0.73,
45       "uvi": 4.11
46     }
47   ]
48 }
```

Figure 8: Filtered Data in the Local



## References

- [1] Mysql.com. [Online]. Available: <https://dev.mysql.com/doc/workbench/en/wb-forward-engineering-live-server.html>. [Accessed: 18-Oct-2023].
- [2] “Google Cloud Platform,” Google.com. [Online]. Available: <https://cloud.google.com/?hl=en>. [Accessed: 18-Oct-2023].
- [3] “Apache Spark™ - Unified Engine for large-scale data analytics,” Apache.org. [Online]. Available: <https://spark.apache.org/>. [Accessed: 04-Nov-2023].
- [4] “Python,” Python.org. [Online]. Available: <https://www.python.org/>. [Accessed: 04-Nov-2023].