Hadoop Multi-Node Cluster Setup with AWS

Jonathan Glasgow

Table of Contents

Build Procedure

Cluster Creation

Building a Hadoop Multi-Node cluster on AWS (Amazon Web Services) can be done using the EMR service. The first step is to go to the EMR Service in the AWS Console and click ‘Create Cluster’. You will then be asked to specify the configuration for the cluster. Enter the configuration as shown in the ‘Configuration’ section and create the cluster. Ensure that you associate the cluster with an EC2 key pair that you have access to.

Data Preparation

While AWS provisions the cluster, you can prepare your data to be stored in the cloud using AWS’s object storage service: S3. You will need to open the S3 service in the AWS console and then click ‘create bucket’. Once you have an S3 bucket, you can upload the CSV files to the service.

Accessing the Cluster

Once the cluster is provisioned, the default security group will block all traffic. To give ourselves access to the cluster, go to the cluster’s homepage, and click on the ‘Security group for Master’ and choose the ‘ElasticMapReduce-master’. Next, click ‘Add rule’ and select ‘SSH’ and ‘My IP’. Now you can SSH into the master node using the command provided on the cluster’s homepage in EMR. This will require having access to the EC2 key pair you created earlier.

Once you are in the cluster, you can perform hdfs queries. For example:

```
hdfs fs -ls
```

In addition, you can execute Spark/MapReduce applications. For example:

```
spark-submit main.py
```

Environment Variables

The location of where Java is in the cluster nodes, which is used when executing Hadoop commands and running Java programs

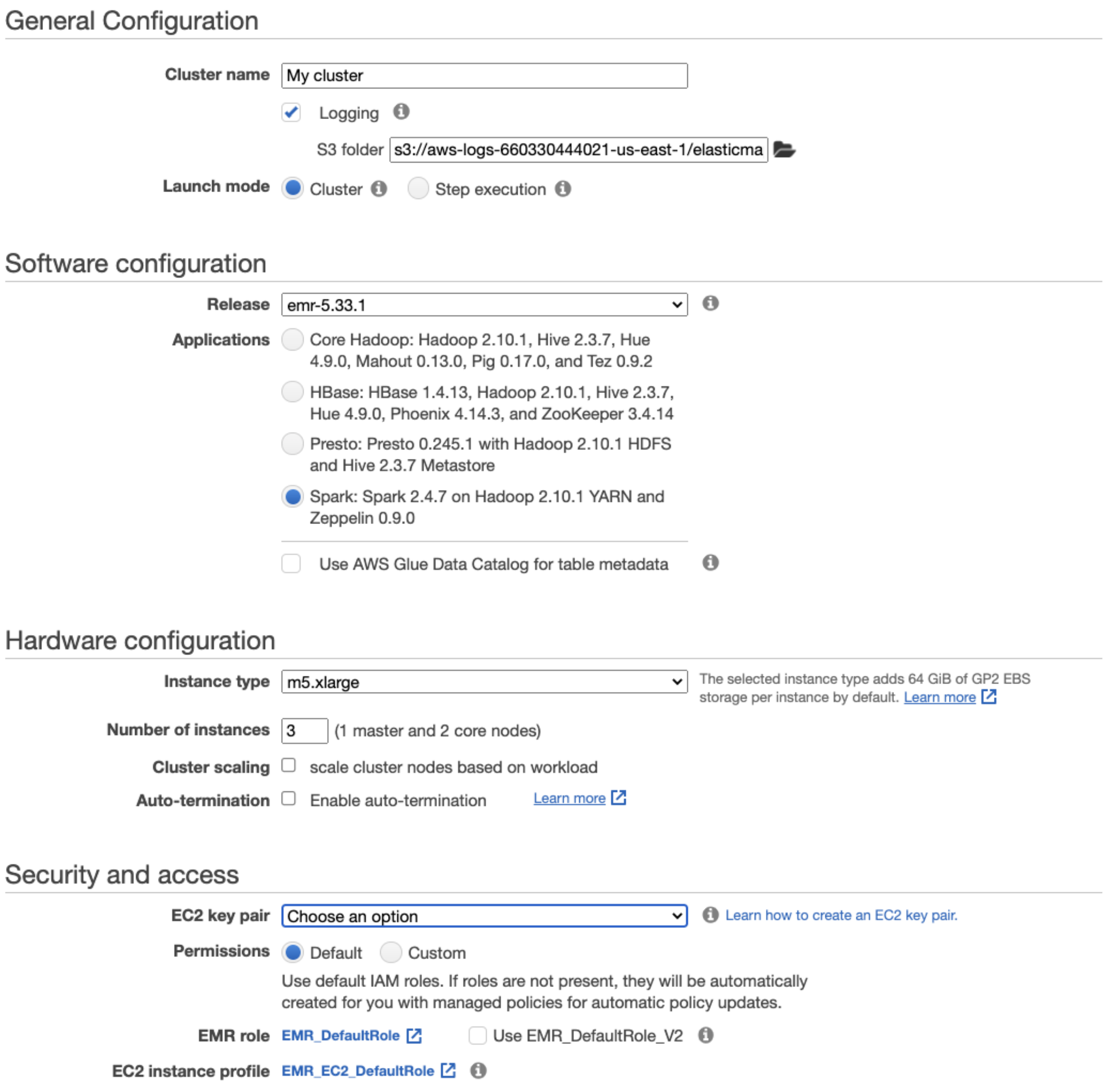
```
JAVA_HOME=/etc/alternatives/jre
```

The home directory of the SSH user:

```
HOME=/home/hadoop
```

Configuration

AWS EMR provides us with a high-level abstraction to configure a Hadoop cluster on their cloud platform. The configuration, as seen below, is set to run Spark 2.4.7 on Hadoop 2.10.1 YARN. The cluster consists of 3 m5.xlarge EC2 instances, 1 of which is the master, and 2 of which are the core nodes.



General Configuration

Cluster name: My cluster

[✓] Logging

S3 folder: s3://aws-logs-660330444021-us-east-1/elasticma

Launch mode: [✓ Cluster] Step execution

Software configuration

Release: emr-5.33.1

Applications:
- Core Hadoop: Hadoop 2.10.1, Hive 2.3.7, Hue 4.9.0, Mahout 0.13.0, Pig 0.17.0, and Tez 0.9.2
- HBase: HBase 1.4.13, Hadoop 2.10.1, Hive 2.3.7, Hue 4.9.0, Phoenix 4.14.3, and ZooKeeper 3.4.14
- Presto: Presto 0.245.1 with Hadoop 2.10.1 HDFS and Hive 2.3.7 Metastore
- [✓ Spark: Spark 2.4.7 on Hadoop 2.10.1 YARN and Zeppelin 0.9.0]

[ ] Use AWS Glue Data Catalog for table metadata

Hardware configuration

Instance type: m5.xlarge — The selected instance type adds 64 GiB of GP2 EBS storage per instance by default. Learn more

Number of instances: 3 (1 master and 2 core nodes)

Cluster scaling: [ ] scale cluster nodes based on workload

Auto-termination: [ ] Enable auto-termination Learn more

Security and access

EC2 key pair: Choose an option — Learn how to create an EC2 key pair.

Permissions: [✓ Default] Custom

Use default IAM roles. If roles are not present, they will be automatically created for you with managed policies for automatic policy updates.

EMR role: EMR_DefaultRole [ ] Use EMR_DefaultRole_V2

EC2 instance profile: EMR_EC2_DefaultRole

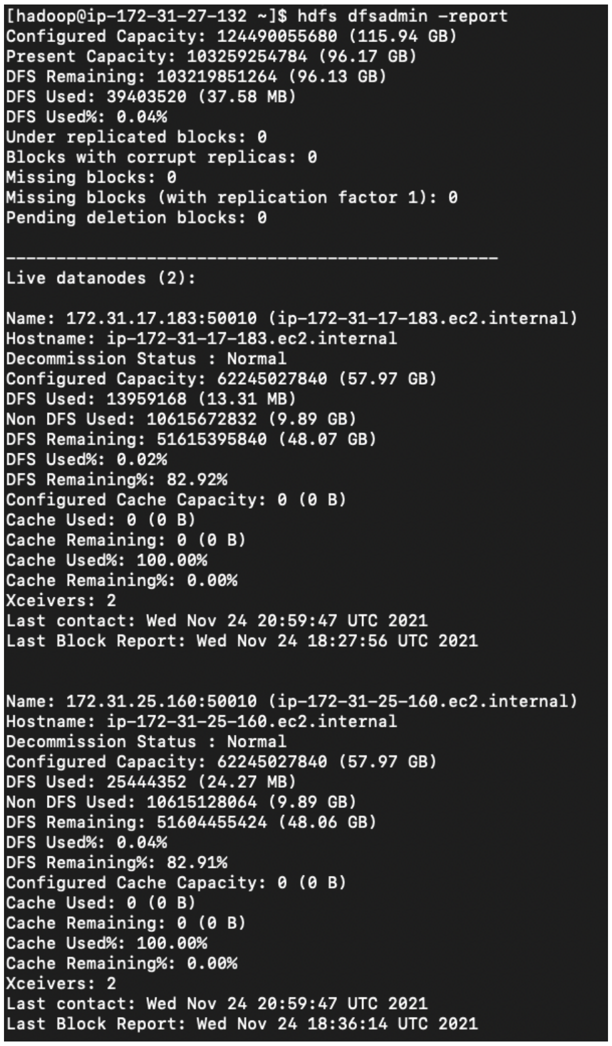
Daemon Summary and Configuration

Amazon EMR handles the setup and configuration of the Hadoop daemons at cluster creation.

NameNode: Stores and maintains metadata for the cluster

Secondary NameNode: Performs housekeeping functions for NameNode

DataNode: Stores data blocks in HDFS on the cluster



```
[hadoop@ip-172-31-27-132 ~]$ hdfs dfsadmin -report
Configured Capacity: 124490055680 (115.94 GB)
Present Capacity: 103259254784 (96.17 GB)
DFS Remaining: 103219851264 (96.13 GB)
DFS Used: 39403520 (37.58 MB)
DFS Used%: 0.04%
Under replicated blocks: 0
Blocks with corrupt replicas: 0
Missing blocks: 0
Missing blocks (with replication factor 1): 0
Pending deletion blocks: 0

-------------------------------------------------
Live datanodes (2):

Name: 172.31.17.183:50010 (ip-172-31-17-183.ec2.internal)
Hostname: ip-172-31-17-183.ec2.internal
Decommission Status : Normal
Configured Capacity: 62245027840 (57.97 GB)
DFS Used: 13959168 (13.31 MB)
Non DFS Used: 10615672832 (9.89 GB)
DFS Remaining: 51615395840 (48.07 GB)
DFS Used%: 0.02%
DFS Remaining%: 82.92%
Configured Cache Capacity: 0 (0 B)
Cache Used: 0 (0 B)
Cache Remaining: 0 (0 B)
Cache Used%: 100.00%
Cache Remaining%: 0.00%
Xceivers: 2
Last contact: Wed Nov 24 20:59:47 UTC 2021
Last Block Report: Wed Nov 24 18:27:56 UTC 2021


Name: 172.31.25.160:50010 (ip-172-31-25-160.ec2.internal)
Hostname: ip-172-31-25-160.ec2.internal
Decommission Status : Normal
Configured Capacity: 62245027840 (57.97 GB)
DFS Used: 25444352 (24.27 MB)
Non DFS Used: 10615128064 (9.89 GB)
DFS Remaining: 51604455424 (48.06 GB)
DFS Used%: 0.04%
DFS Remaining%: 82.91%
Configured Cache Capacity: 0 (0 B)
Cache Used: 0 (0 B)
Cache Remaining: 0 (0 B)
Cache Used%: 100.00%
Cache Remaining%: 0.00%
Xceivers: 2
Last contact: Wed Nov 24 20:59:47 UTC 2021
Last Block Report: Wed Nov 24 18:36:14 UTC 2021
```

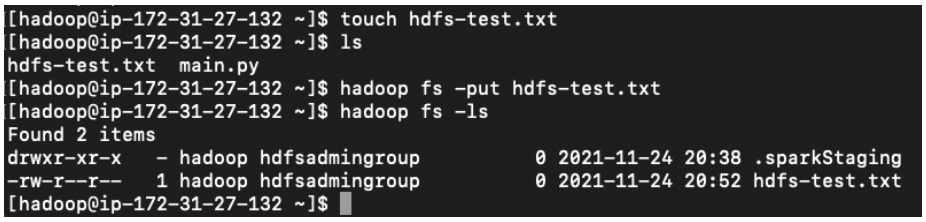
Resource Manager: Manages and allocates resources within the cluster

Node Manager: Executes tasks on the slave nodes

Cluster Test Results

HDFS

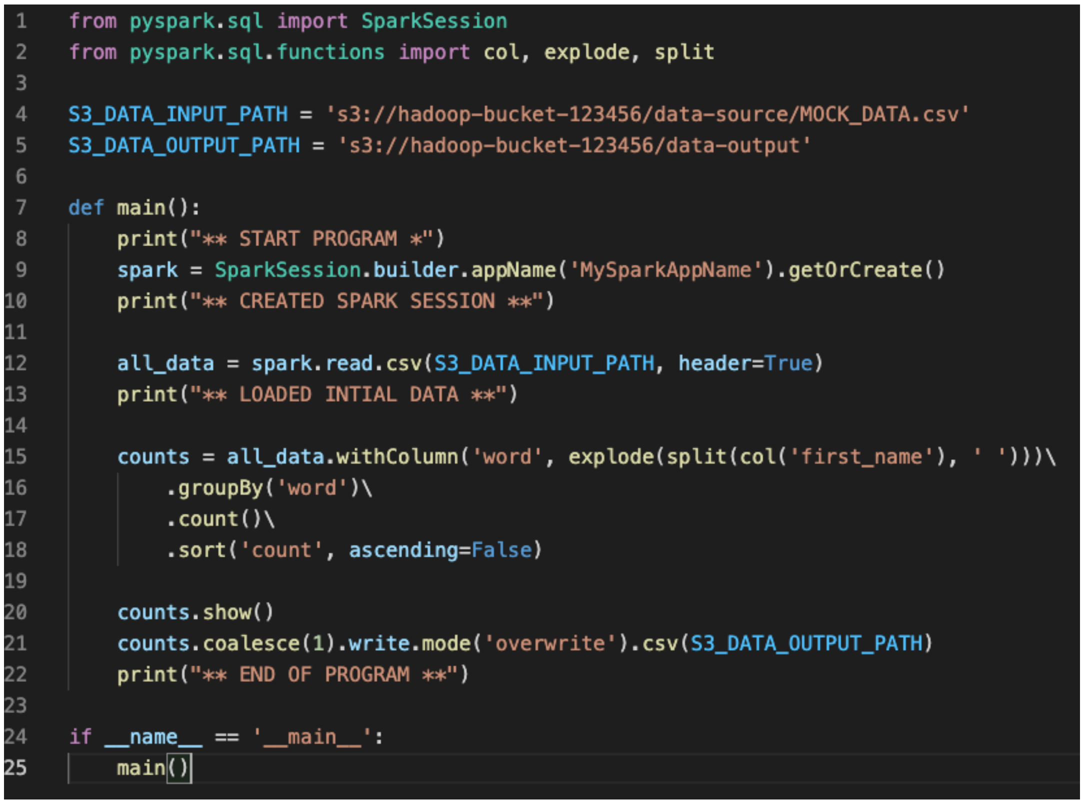
Below is a screenshot from the cluster of a file being created and transferred to HDFS. We also show that the file successfully transferred. This proves that the cluster can run HDFS tasks as needed.



```
[hadoop@ip-172-31-27-132 ~]$ touch hdfs-test.txt
[hadoop@ip-172-31-27-132 ~]$ ls
hdfs-test.txt  main.py
[hadoop@ip-172-31-27-132 ~]$ hadoop fs -put hdfs-test.txt
[hadoop@ip-172-31-27-132 ~]$ hadoop fs -ls
Found 2 items
drwxr-xr-x   - hadoop hdfsadmingroup          0 2021-11-24 20:38 .sparkStaging
-rw-r--r--   1 hadoop hdfsadmingroup          0 2021-11-24 20:52 hdfs-test.txt
[hadoop@ip-172-31-27-132 ~]$
```

MapReduce Application

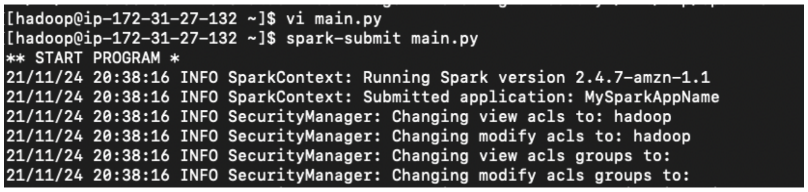
MapReduce is shown below to operate successfully on the cluster. The data set consisted of 1000 rows with randomized information based on first names, last names, emails, genders, and colors. Using this data, you can test the cluster's MapReduce abilities to give us a word count for the variable 'first_name'. Shown below are screenshots of this process and its results in action.



```python
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, explode, split

S3_DATA_INPUT_PATH = 's3://hadoop-bucket-123456/data-source/MOCK_DATA.csv'
S3_DATA_OUTPUT_PATH = 's3://hadoop-bucket-123456/data-output'

def main():
    print("** START PROGRAM *")
    spark = SparkSession.builder.appName('MySparkAppName').getOrCreate()
    print("** CREATED SPARK SESSION **")

    all_data = spark.read.csv(S3_DATA_INPUT_PATH, header=True)
    print("** LOADED INTIAL DATA **")

    counts = all_data.withColumn('word', explode(split(col('first_name'), ' ')))\
        .groupBy('word')\
        .count()\
        .sort('count', ascending=False)

    counts.show()
    counts.coalesce(1).write.mode('overwrite').csv(S3_DATA_OUTPUT_PATH)
    print("** END OF PROGRAM **")

if __name__ == '__main__':
    main()
```

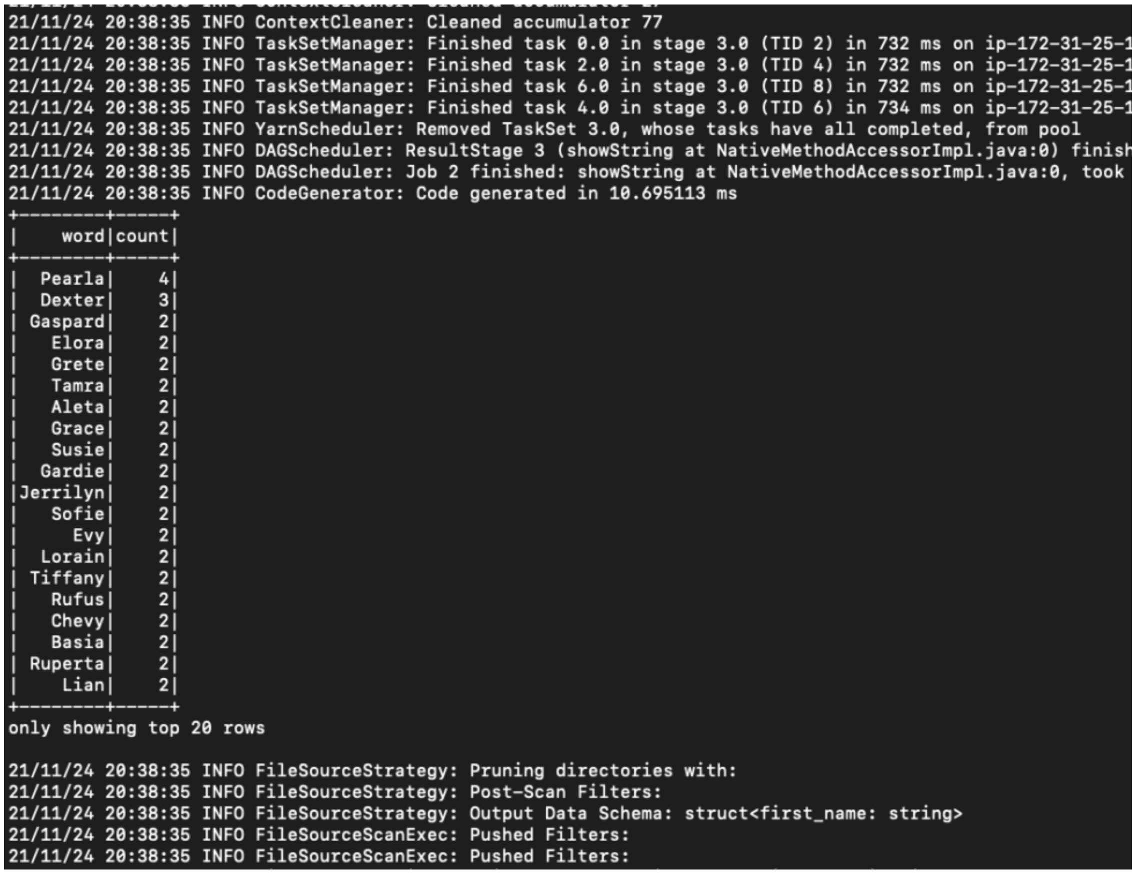
main.py: The program to run the WordCount task.

Starting the Program



```
[hadoop@ip-172-31-27-132 ~]$ vi main.py
[hadoop@ip-172-31-27-132 ~]$ spark-submit main.py
** START PROGRAM *
21/11/24 20:38:16 INFO SparkContext: Running Spark version 2.4.7-amzn-1.1
21/11/24 20:38:16 INFO SparkContext: Submitted application: MySparkAppName
21/11/24 20:38:16 INFO SecurityManager: Changing view acls to: hadoop
21/11/24 20:38:16 INFO SecurityManager: Changing modify acls to: hadoop
21/11/24 20:38:16 INFO SecurityManager: Changing view acls groups to:
21/11/24 20:38:16 INFO SecurityManager: Changing modify acls groups to:
```

Program Output



```
21/11/24 20:38:35 INFO ContextCleaner: Cleaned accumulator 77
21/11/24 20:38:35 INFO TaskSetManager: Finished task 0.0 in stage 3.0 (TID 2) in 732 ms on ip-172-31-25-1
21/11/24 20:38:35 INFO TaskSetManager: Finished task 2.0 in stage 3.0 (TID 4) in 732 ms on ip-172-31-25-1
21/11/24 20:38:35 INFO TaskSetManager: Finished task 6.0 in stage 3.0 (TID 8) in 732 ms on ip-172-31-25-1
21/11/24 20:38:35 INFO TaskSetManager: Finished task 4.0 in stage 3.0 (TID 6) in 734 ms on ip-172-31-25-1
21/11/24 20:38:35 INFO YarnScheduler: Removed TaskSet 3.0, whose tasks have all completed, from pool
21/11/24 20:38:35 INFO DAGScheduler: ResultStage 3 (showString at NativeMethodAccessorImpl.java:0) finish
21/11/24 20:38:35 INFO DAGScheduler: Job 2 finished: showString at NativeMethodAccessorImpl.java:0, took
21/11/24 20:38:35 INFO CodeGenerator: Code generated in 10.695113 ms
+--------+-----+
|    word|count|
+--------+-----+
|  Pearla|    4|
|  Dexter|    3|
| Gaspard|    2|
|   Elora|    2|
|   Grete|    2|
|   Tamra|    2|
|   Aleta|    2|
|   Grace|    2|
|   Susie|    2|
|  Gardie|    2|
|Jerrilyn|    2|
|   Sofie|    2|
|     Evy|    2|
|  Lorain|    2|
| Tiffany|    2|
|   Rufus|    2|
|   Chevy|    2|
|   Basia|    2|
| Ruperta|    2|
|    Lian|    2|
+--------+-----+
only showing top 20 rows

21/11/24 20:38:35 INFO FileSourceStrategy: Pruning directories with:
21/11/24 20:38:35 INFO FileSourceStrategy: Post-Scan Filters:
21/11/24 20:38:35 INFO FileSourceStrategy: Output Data Schema: struct<first_name: string>
21/11/24 20:38:35 INFO FileSourceScanExec: Pushed Filters:
21/11/24 20:38:35 INFO FileSourceScanExec: Pushed Filters:
```

Showing the top 20 rows for word count of first_name.

Results Conclusion

By utilizing AWS and EMR, you successfully created a hadoop cluster that can run HDFS commands, spark, and yarn effectively. This can be verified in the screenshots and the attached files showing the success of the WordCount task.