Big Data Wrangling With Google Books Ngrams

Author: Kartik
Contact: kkakar664@gmail.com
Date: Nov 14, 2023

Table of Contents

Part 1. Introduction

This report outlines the steps taken to analyze the Google Books Ngrams dataset using cloud-based distributed computing tools such as Hadoop, Spark, Hive, and the S3 filesystem. The Google Ngrams dataset, created by Google's research team, represents approximately 4% of all books ever printed, spanning from the 1800s to the 2000s.

The dataset was accessed from a public S3 bucket, processed using Spark on an AWS EMR cluster, and visualized using matplotlib. The report also includes a comparison of Hadoop and Spark as distributed file systems.

Part 2. Steps and Commands

2.1 Spin up a new EMR cluster on AWS for using Spark and EMR notebooks.

Log in to your AWS account via this link: http://aws.amazon.com/console/

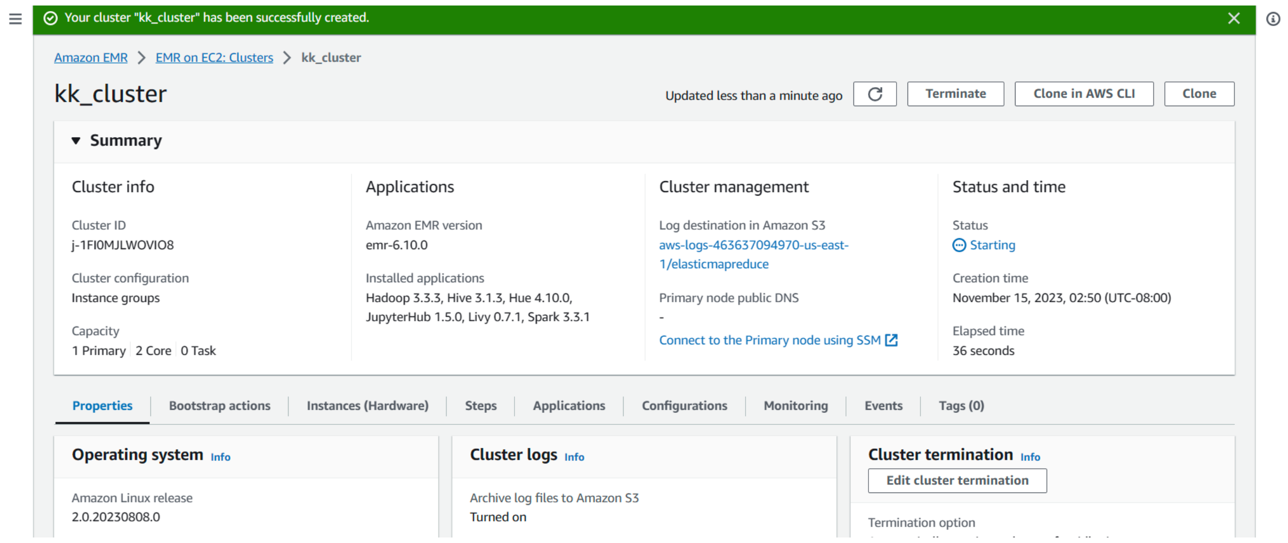
1) Create Cluster: We start by navigating to the EMR panel in AWS and clicking on 'Create Cluster'.

2) Cluster Name and Application Bundle: We give our cluster a name and select 'emr-6.10.0' from the 'Release' dropdown. We then select the Custom application bundle and click the boxes for Hadoop, Hue, JupyterHub, Livy, Hive, and Spark. These are the software applications we want to run on our cluster.
3) Instance Groups: We remove the task instance group and allocate 2 nodes to the core instance group. The core instance group contains the master node and core nodes that run Hadoop Distributed File System (HDFS) and other services.

4) Cluster Termination: We set the cluster to terminate after 4 hours of idle time and turn off termination protection. This helps to manage costs by ensuring that we are not paying for resources that we are not using.

5) Security and Access Management: We select our key pair for secure access to our cluster. In Identity and Access Management, we choose the ‘EMR_DefaultRole’ and ‘EMR_EC2_DefaultRole’. These roles define the permissions for our cluster and the EC2 instances it uses.

6) Create Cluster: Finally, we click the 'Create Cluster' button. The cluster creation process begins, which involves spinning up the EC2 instances and configuring the cluster software. Once the cluster is created, it enters a 'Waiting' state, indicating that it is ready for use.


Your cluster "kk_cluster" has been successfully created.

Amazon EMR > EMR on EC2: Clusters > kk_cluster

kk_cluster

Updated less than a minute ago

Summary

Cluster info	Applications	Cluster management	Status and time
Cluster ID: j-1FI0MJLWOVIO8	Amazon EMR version: emr-6.10.0	Log destination in Amazon S3: aws-logs-463637094970-us-east-1/elasticmapreduce	Status: Starting
Cluster configuration: Instance groups	Installed applications: Hadoop 3.3.3, Hive 3.1.3, Hue 4.10.0, JupyterHub 1.5.0, Livy 0.7.1, Spark 3.3.1	Primary node public DNS: -	Creation time: November 15, 2023, 02:50 (UTC-08:00)
Capacity: 1 Primary, 2 Core, 0 Task		Connect to the Primary node using SSM	Elapsed time: 36 seconds

Operating system Info
Amazon Linux release
2.0.20230808.0

Cluster logs Info
Archive log files to Amazon S3
Turned on

Cluster termination Info
Edit cluster termination
Termination option

Please see the screenshots at the end of the report.

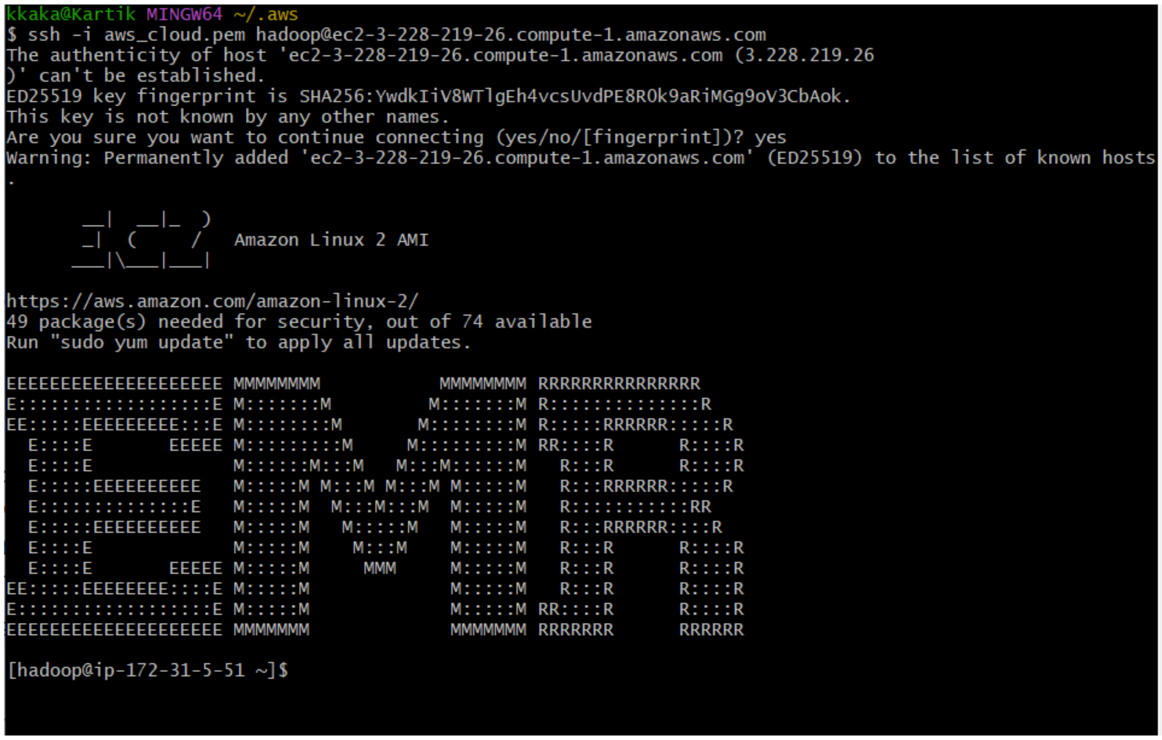
2.2 Connect to the head node of the cluster using SSH.

1) SSH Command: We use the SSH command to establish a secure connection with the head node of the cluster. The command is structured as follows: **ssh -i mykey.pem -L 9995:localhost:9443 hadoop@xxxxxxxxxxxxxx.compute.amazonaws.com**.

- **-i mykey.pem** specifies the private key file (.pem) used for the connection. The file path and name should correspond to your actual .pem key. The command assumes that the current directory contains the .pem key file.
- **-L 9995:localhost:9443** sets up a secure tunnel by forwarding a port (9995) on your local machine to a port (9443) on the remote machine.
- **hadoop@xxxxxxxxxxxxxx.compute.amazonaws.com** is the username and the public DNS of the head node. The 'xxxxx……' should be replaced with your actual 'Primary node public DNS', which can be found on the overview page of your cluster.

2) Successful Connection: If the connection is successful, you will see the EMR ASCII banner in your terminal. This indicates that you are now connected to the head node of the cluster.

3) Bypassing the Security Warning: When you access JupyterHub in your web browser using the forwarded port (hbp://localhost:9995), you may encounter a security warning because the connec.on is not publicly verified. You can bypass this warning to proceed to JupyterHub. The exact steps to bypass the warning depend on your browser. Generally, you would click on "Advanced" and then "Proceed to localhost (unsafe)".

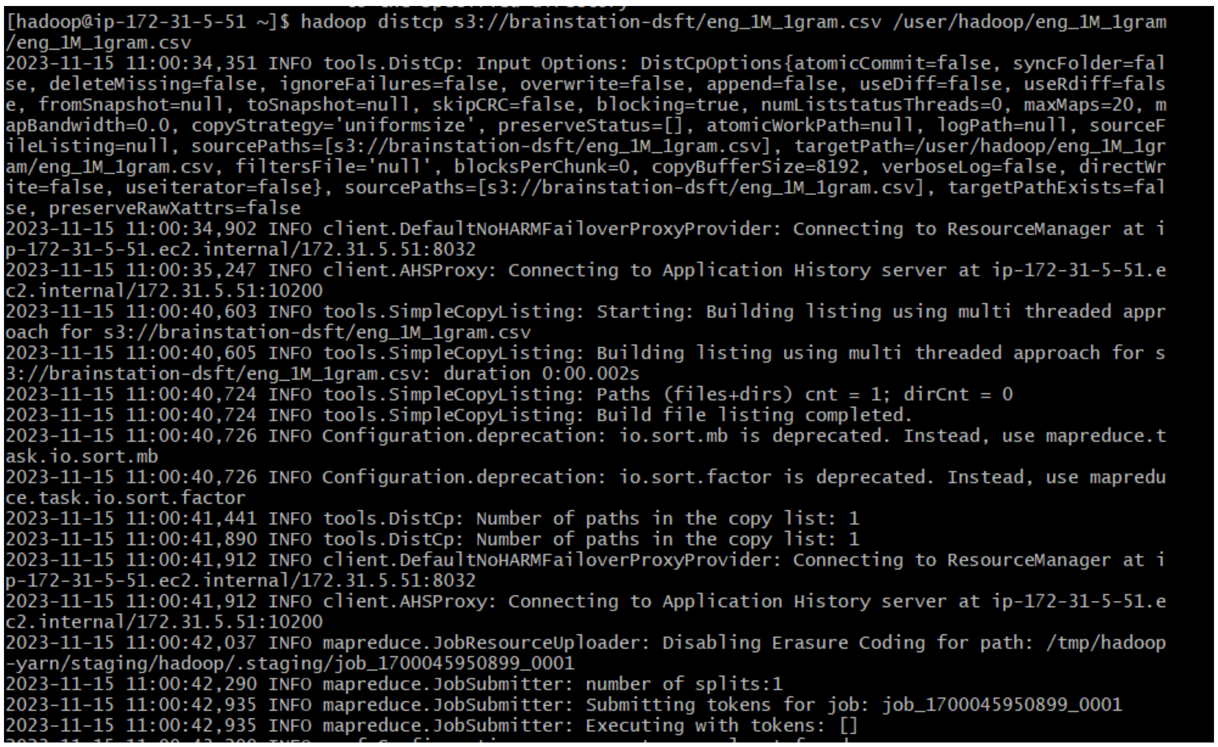


```
kkaka@Kartik MINGW64 ~/.aws
$ ssh -i aws_cloud.pem hadoop@ec2-3-228-219-26.compute-1.amazonaws.com
The authenticity of host 'ec2-3-228-219-26.compute-1.amazonaws.com (3.228.219.26
)' can't be established.
ED25519 key fingerprint is SHA256:YwdkIiV8WTlgEh4vcsUvdPE8ROk9aRiMGg9oV3CbAok.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'ec2-3-228-219-26.compute-1.amazonaws.com' (ED25519) to the list of known hosts
.

       __|  __|_  )
       _|  (     /   Amazon Linux 2 AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-2/
49 package(s) needed for security, out of 74 available
Run "sudo yum update" to apply all updates.

EEEEEEEEEEEEEEEEEEEE MMMMMMMM           MMMMMMMM RRRRRRRRRRRRRRR
E::::::::::::::::::E M:::::::M         M:::::::M R::::::::::::::R
EE:::::EEEEEEEEE:::E M::::::::M       M::::::::M R:::::RRRRRR:::::R
  E::::E       EEEEE M:::::::::M     M:::::::::M RR::::R      R::::R
  E::::E             M::::::M:::M   M:::M::::::M   R:::R      R::::R
  E:::::EEEEEEEEEE   M:::::M M:::M M:::M M:::::M   R:::RRRRRR:::::R
  E::::::::::::::E   M:::::M  M:::M:::M  M:::::M   R:::::::::::RR
  E:::::EEEEEEEEEE   M:::::M   M:::::M   M:::::M   R:::RRRRRR::::R
  E::::E             M:::::M    M:::M    M:::::M   R:::R      R::::R
  E::::E       EEEEE M:::::M     MMM     M:::::M   R:::R      R::::R
EE:::::EEEEEEEE::::E M:::::M             M:::::M   R:::R      R::::R
E::::::::::::::::::E M:::::M             M:::::M RR::::R      R::::R
EEEEEEEEEEEEEEEEEEEE MMMMMMM             MMMMMMM RRRRRRR      RRRRRR

[hadoop@ip-172-31-5-51 ~]$
```

2.3 Copy the data folder from the S3 bucket directly into a directory on the Hadoop File System (HDFS) named /user/hadoop/eng_1M_1gram.

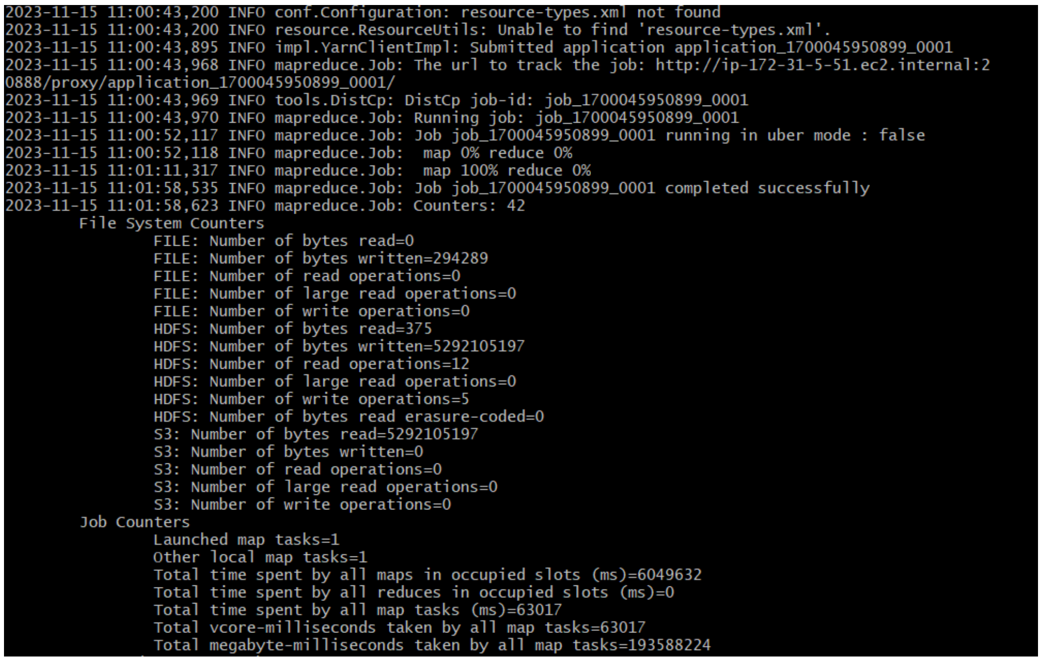
We are copying the data from an S3 bucket directly into a directory on the Hadoop File System (HDFS). Here's a summary of the steps:

1) Hadoop "distCp" Command: We use the Hadoop distCp (Distributed Copy) command to copy the data from the S3 bucket to the HDFS. The command is structured as follows: **hadoop distcp s3://brainsta8on-ds0/eng_1M_1gram.csv/user/hadoop/eng_1M_1gram/eng_1M_1gram.csv**. This command copies the **eng_1M_1gram.csv** file from the S3 bucket **s3://brainsta8on-ds0/** to the directory **/user/hadoop/eng_1M_1gram/** in the HDFS.



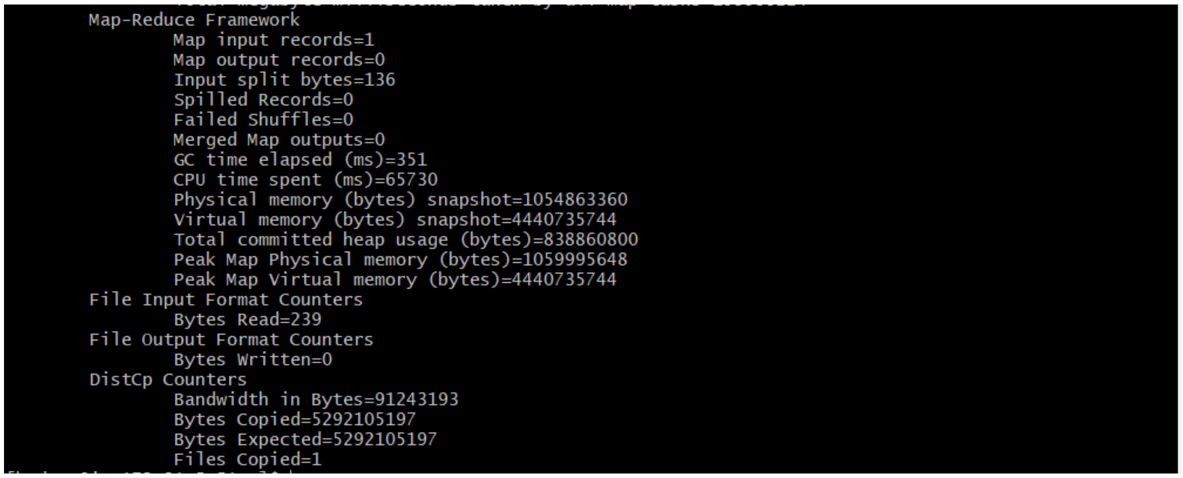
```
[hadoop@ip-172-31-5-51 ~]$ hadoop distcp s3://brainstation-dsft/eng_1M_1gram.csv /user/hadoop/eng_1M_1gram
/eng_1M_1gram.csv
2023-11-15 11:00:34,351 INFO tools.DistCp: Input Options: DistCpOptions{atomicCommit=false, syncFolder=fal
se, deleteMissing=false, ignoreFailures=false, overwrite=false, append=false, useDiff=false, useRdiff=fals
e, fromSnapshot=null, toSnapshot=null, skipCRC=false, blocking=true, numListstatusThreads=0, maxMaps=20, m
apBandwidth=0.0, copyStrategy='uniformsize', preserveStatus=[], atomicWorkPath=null, logPath=null, sourceF
ileListing=null, sourcePaths=[s3://brainstation-dsft/eng_1M_1gram.csv], targetPath=/user/hadoop/eng_1M_1gr
am/eng_1M_1gram.csv, filtersFile='null', blocksPerChunk=0, copyBufferSize=8192, verboseLog=false, directWr
ite=false, useiterator=false}, sourcePaths=[s3://brainstation-dsft/eng_1M_1gram.csv], targetPathExists=fal
se, preserveRawXattrs=false
2023-11-15 11:00:34,902 INFO client.DefaultNoHARMFailoverProxyProvider: Connecting to ResourceManager at i
p-172-31-5-51.ec2.internal/172.31.5.51:8032
2023-11-15 11:00:35,247 INFO client.AHSProxy: Connecting to Application History server at ip-172-31-5-51.e
c2.internal/172.31.5.51:10200
2023-11-15 11:00:40,603 INFO tools.SimpleCopyListing: Starting: Building listing using multi threaded appr
oach for s3://brainstation-dsft/eng_1M_1gram.csv
2023-11-15 11:00:40,605 INFO tools.SimpleCopyListing: Building listing using multi threaded approach for s
3://brainstation-dsft/eng_1M_1gram.csv: duration 0:00.002s
2023-11-15 11:00:40,724 INFO tools.SimpleCopyListing: Paths (files+dirs) cnt = 1; dirCnt = 0
2023-11-15 11:00:40,724 INFO tools.SimpleCopyListing: Build file listing completed.
2023-11-15 11:00:40,726 INFO Configuration.deprecation: io.sort.mb is deprecated. Instead, use mapreduce.t
ask.io.sort.mb
2023-11-15 11:00:40,726 INFO Configuration.deprecation: io.sort.factor is deprecated. Instead, use mapredu
ce.task.io.sort.factor
2023-11-15 11:00:41,441 INFO tools.DistCp: Number of paths in the copy list: 1
2023-11-15 11:00:41,890 INFO tools.DistCp: Number of paths in the copy list: 1
2023-11-15 11:00:41,912 INFO client.DefaultNoHARMFailoverProxyProvider: Connecting to ResourceManager at i
p-172-31-5-51.ec2.internal/172.31.5.51:8032
2023-11-15 11:00:41,912 INFO client.AHSProxy: Connecting to Application History server at ip-172-31-5-51.e
c2.internal/172.31.5.51:10200
2023-11-15 11:00:42,037 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop
-yarn/staging/hadoop/.staging/job_1700045950899_0001
2023-11-15 11:00:42,290 INFO mapreduce.JobSubmitter: number of splits:1
2023-11-15 11:00:42,935 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1700045950899_0001
2023-11-15 11:00:42,935 INFO mapreduce.JobSubmitter: Executing with tokens: []
```

2) Job Execution: After running the 'distcp' command, a MapReduce job is initiated to perform the copy operation. The job progress is displayed in the terminal, including the job ID, the job status, and the percentage of map and reduce tasks completed.



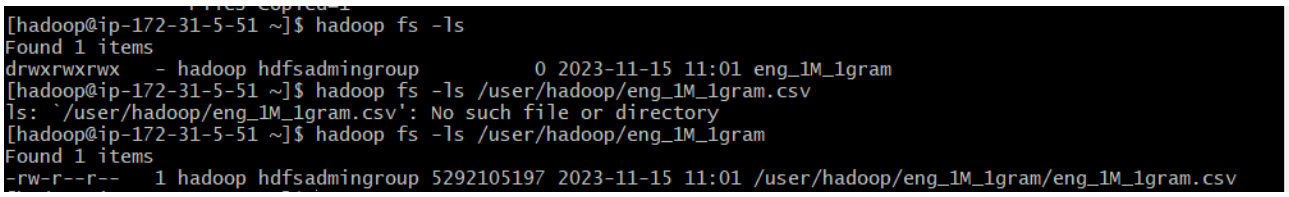
```
2023-11-15 11:00:43,200 INFO conf.Configuration: resource-types.xml not found
2023-11-15 11:00:43,200 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2023-11-15 11:00:43,895 INFO impl.YarnClientImpl: Submitted application application_1700045950899_0001
2023-11-15 11:00:43,968 INFO mapreduce.Job: The url to track the job: http://ip-172-31-5-51.ec2.internal:2
0888/proxy/application_1700045950899_0001/
2023-11-15 11:00:43,969 INFO tools.DistCp: DistCp job-id: job_1700045950899_0001
2023-11-15 11:00:43,970 INFO mapreduce.Job: Running job: job_1700045950899_0001
2023-11-15 11:00:52,117 INFO mapreduce.Job: Job job_1700045950899_0001 running in uber mode : false
2023-11-15 11:00:52,118 INFO mapreduce.Job:  map 0% reduce 0%
2023-11-15 11:01:11,317 INFO mapreduce.Job:  map 100% reduce 0%
2023-11-15 11:01:58,535 INFO mapreduce.Job: Job job_1700045950899_0001 completed successfully
2023-11-15 11:01:58,623 INFO mapreduce.Job: Counters: 42
        File System Counters
                FILE: Number of bytes read=0
                FILE: Number of bytes written=294289
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=375
                HDFS: Number of bytes written=5292105197
                HDFS: Number of read operations=12
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=5
                HDFS: Number of bytes read erasure-coded=0
                S3: Number of bytes read=5292105197
                S3: Number of bytes written=0
                S3: Number of read operations=0
                S3: Number of large read operations=0
                S3: Number of write operations=0
        Job Counters
                Launched map tasks=1
                Other local map tasks=1
                Total time spent by all maps in occupied slots (ms)=6049632
                Total time spent by all reduces in occupied slots (ms)=0
                Total time spent by all map tasks (ms)=63017
                Total vcore-milliseconds taken by all map tasks=63017
                Total megabyte-milliseconds taken by all map tasks=193588224
```

3) Job Completion: Once the job is completed successfully, the terminal displays a summary of the job, including the number of bytes read and written, the number of read and write operations and other job counters.



```
        Map-Reduce Framework
                Map input records=1
                Map output records=0
                Input split bytes=136
                Spilled Records=0
                Failed Shuffles=0
                Merged Map outputs=0
                GC time elapsed (ms)=351
                CPU time spent (ms)=65730
                Physical memory (bytes) snapshot=1054863360
                Virtual memory (bytes) snapshot=4440735744
                Total committed heap usage (bytes)=838860800
                Peak Map Physical memory (bytes)=1059995648
                Peak Map Virtual memory (bytes)=4440735744
        File Input Format Counters
                Bytes Read=239
        File Output Format Counters
                Bytes Written=0
        DistCp Counters
                Bandwidth in Bytes=91243193
                Bytes Copied=5292105197
                Bytes Expected=5292105197
                Files Copied=1
```

4) Data Verification: After the copy operation, we verify the data transfer by listing the contents of the HDFS directory using the '**hadoop fs -ls**' and '**hadoop fs -ls /user/hadoop/eng_1M_1gram'** commands. These commands display the directories and files in the HDFS, confirming that the **eng_1M_1gram.csv** file has been successfully copied to the **/user/hadoop/eng_1M_1gram/** directory.

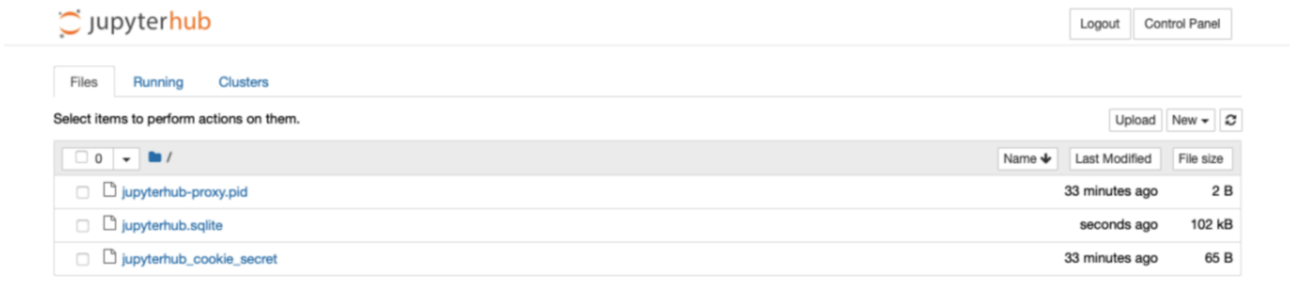


```
[hadoop@ip-172-31-5-51 ~]$ hadoop fs -ls
Found 1 items
drwxrwxrwx   - hadoop hdfsadmingroup          0 2023-11-15 11:01 eng_1M_1gram
[hadoop@ip-172-31-5-51 ~]$ hadoop fs -ls /user/hadoop/eng_1M_1gram.csv
ls: `/user/hadoop/eng_1M_1gram.csv': No such file or directory
[hadoop@ip-172-31-5-51 ~]$ hadoop fs -ls /user/hadoop/eng_1M_1gram
Found 1 items
-rw-r--r--   1 hadoop hdfsadmingroup 5292105197 2023-11-15 11:01 /user/hadoop/eng_1M_1gram/eng_1M_1gram.csv
```

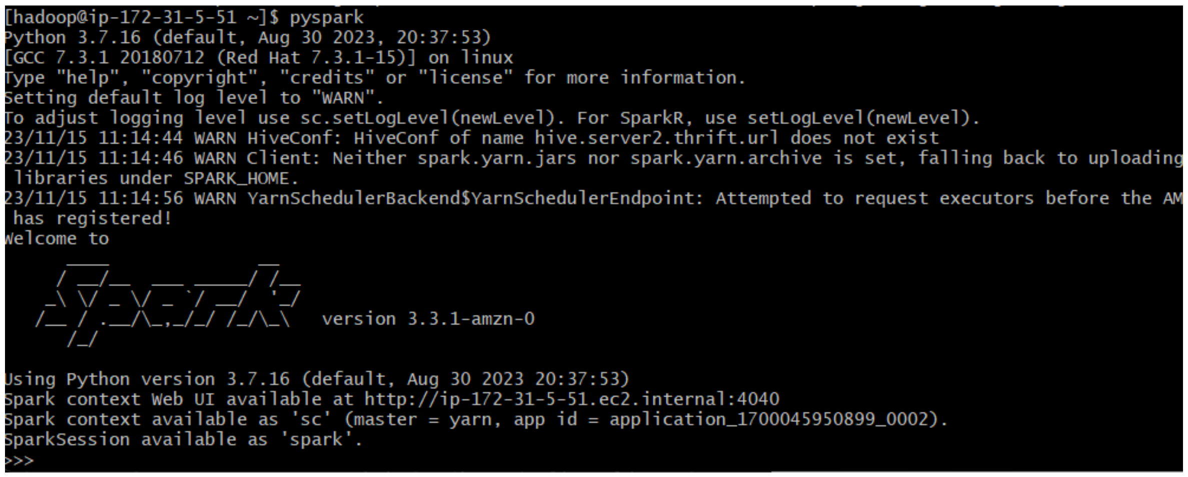
2.4 Using pyspark, read the data you copied into HDFS in Step 3. You may either use Jupyterhub on EMR (the default user and password are jovyan and jupyter) or work from pyspark in the terminal if you prefer. Once you have created a pyspark DataFrame, complete the following steps below:

We are using PySpark to read the data that was previously copied into the Hadoop Distributed File System (HDFS). Here's a summary of the steps:

1) **Launch PySpark**: PySpark is launched either through Jupyterhub on EMR or directly from the terminal.

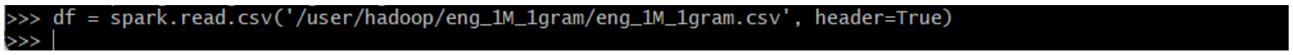


```
jupyterhub-proxy.pid          33 minutes ago    2 B
jupyterhub.sqlite             seconds ago       102 kB
jupyterhub_cookie_secret      33 minutes ago    65 B
```



```
[hadoop@ip-172-31-5-51 ~]$ pyspark
Python 3.7.16 (default, Aug 30 2023, 20:37:53)
[GCC 7.3.1 20180712 (Red Hat 7.3.1-15)] on linux
Type "help", "copyright", "credits" or "license" for more information.
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
23/11/15 11:14:44 WARN HiveConf: HiveConf of name hive.server2.thrift.url does not exist
23/11/15 11:14:46 WARN Client: Neither spark.yarn.jars nor spark.yarn.archive is set, falling back to uploading
 libraries under SPARK_HOME.
23/11/15 11:14:56 WARN YarnSchedulerBackend$YarnSchedulerEndpoint: Attempted to request executors before the AM
 has registered!
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 3.3.1-amzn-0
      /_/

Using Python version 3.7.16 (default, Aug 30 2023 20:37:53)
Spark context Web UI available at http://ip-172-31-5-51.ec2.internal:4040
Spark context available as 'sc' (master = yarn, app id = application_1700045950899_0002).
SparkSession available as 'spark'.
>>>
```

2) **Read Data**: We use the **spark.read.csv** function to read the **eng_1M_1gram.csv** file from the HDFS directory **/user/hadoop/eng_1M_1gram/**. The **header=True** argument is used to indicate that the first row of the CSV file contains the column names. The data is loaded into a Dataframe 'df'.



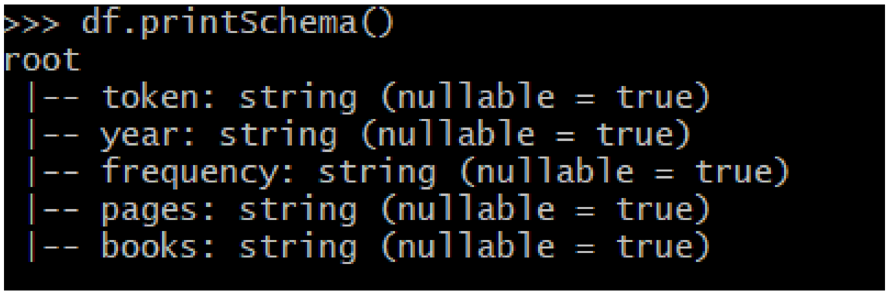
```
>>> df = spark.read.csv('/user/hadoop/eng_1M_1gram/eng_1M_1gram.csv', header=True)
>>> |
```

This step is crucial as it loads the data into a Spark DataFrame, which allows us to perform distributed data processing tasks using Spark's capabilities.

a. Describe the dataset (examples include size, shape, schema) in pyspark

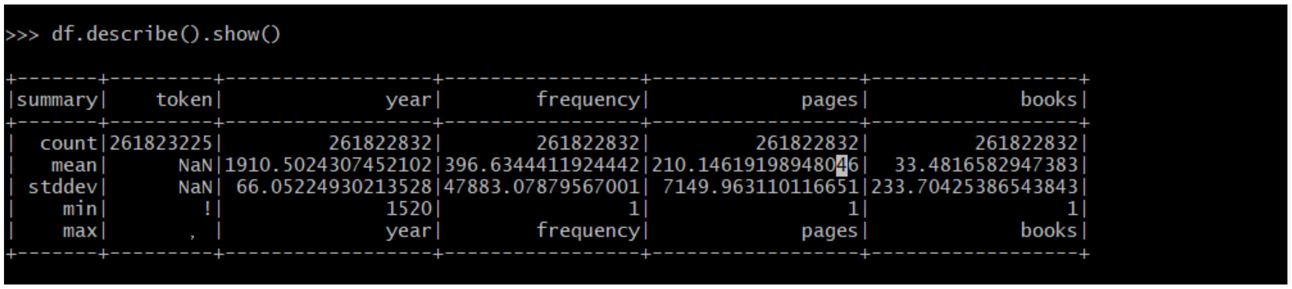
We are exploring the dataset using PySpark's DataFrame API. Here's a summary of the steps:

1) Print Schema: The **df.printSchema()** function is used to print the schema of the DataFrame **df**. The schema includes the column names and their data types. The dataset contains five columns: 'token', 'year', 'frequency', 'pages', and 'books', all of which are of string data type.



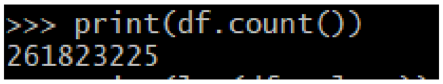
```
>>> df.printSchema()
root
 |-- token: string (nullable = true)
 |-- year: string (nullable = true)
 |-- frequency: string (nullable = true)
 |-- pages: string (nullable = true)
 |-- books: string (nullable = true)
```

2) Descriptive Statistics: The **df.describe().show()** function is used to generate descriptive statistics of the DataFrame. It includes count, mean, standard deviation, min, and max for each column.

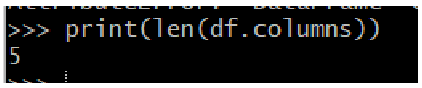


```
>>> df.describe().show()

+-------+---------+------------------+-----------------+------------------+------------------+
|summary|    token|              year|        frequency|             pages|             books|
+-------+---------+------------------+-----------------+------------------+------------------+
|  count|261823225|         261822832|        261822832|         261822832|         261822832|
|   mean|      NaN|1910.5024307452102|396.6344411924442|210.14619198948046| 33.4816582947383|
| stddev|      NaN| 66.05224930213528|47883.07879567001| 7149.963110116651|233.70425386543843|
|    min|        !|              1520|                1|                 1|                 1|
|    max|       , |              year|        frequency|             pages|             books|
+-------+---------+------------------+-----------------+------------------+------------------+
```

3) Count Rows and Columns: The **df.count()** function is used to count the number of rows in the DataFrame, and **len(df.columns)** is used to count the number of columns. The dataset contains 261,823,225 rows and 5 columns.

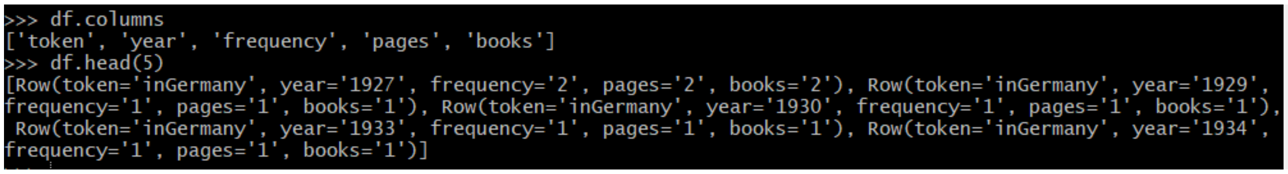


```
>>> print(df.count())
261823225
```



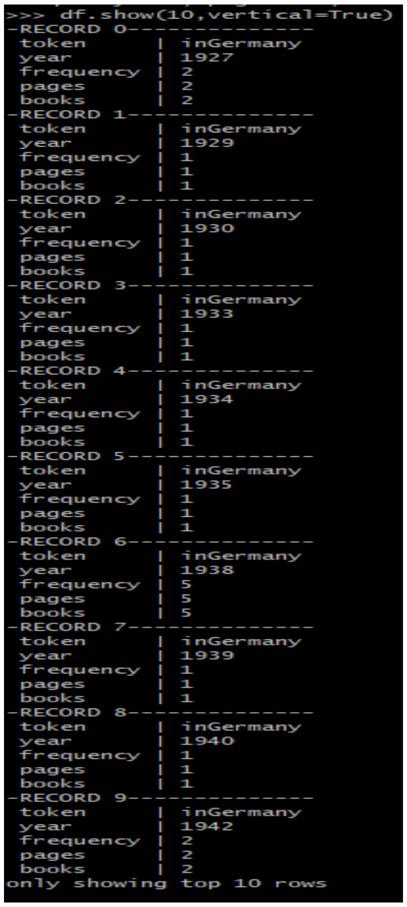
```
>>> print(len(df.columns))
5
```

4) View Column Names: The **df.columns** function is used to view the names of the columns in the DataFrame.
View Top Rows: The **df.head(5)** function is used to view the first 5 rows of the DataFrame.



```
>>> df.columns
['token', 'year', 'frequency', 'pages', 'books']
>>> df.head(5)
[Row(token='inGermany', year='1927', frequency='2', pages='2', books='2'), Row(token='inGermany', year='1929',
frequency='1', pages='1', books='1'), Row(token='inGermany', year='1930', frequency='1', pages='1', books='1'),
 Row(token='inGermany', year='1933', frequency='1', pages='1', books='1'), Row(token='inGermany', year='1934',
frequency='1', pages='1', books='1')]
```

5) Show Rows: The **df.show(10,vertical=True)** function is used to display the first 10 rows of the DataFrame in a vertical format for better readability.



```
>>> df.show(10,vertical=True)
-RECORD 0--------------
 token     | inGermany
 year      | 1927
 frequency | 2
 pages     | 2
 books     | 2
-RECORD 1--------------
 token     | inGermany
 year      | 1929
 frequency | 1
 pages     | 1
 books     | 1
-RECORD 2--------------
 token     | inGermany
 year      | 1930
 frequency | 1
 pages     | 1
 books     | 1
-RECORD 3--------------
 token     | inGermany
 year      | 1933
 frequency | 1
 pages     | 1
 books     | 1
-RECORD 4--------------
 token     | inGermany
 year      | 1934
 frequency | 1
 pages     | 1
 books     | 1
-RECORD 5--------------
 token     | inGermany
 year      | 1935
 frequency | 1
 pages     | 1
 books     | 1
-RECORD 6--------------
 token     | inGermany
 year      | 1938
 frequency | 5
 pages     | 5
 books     | 5
-RECORD 7--------------
 token     | inGermany
 year      | 1939
 frequency | 1
 pages     | 1
 books     | 1
-RECORD 8--------------
 token     | inGermany
 year      | 1940
 frequency | 1
 pages     | 1
 books     | 1
-RECORD 9--------------
 token     | inGermany
 year      | 1942
 frequency | 2
 pages     | 2
 books     | 2
only showing top 10 rows
```

These steps provide a comprehensive overview of the dataset, including its size, structure, and some basic statistics.

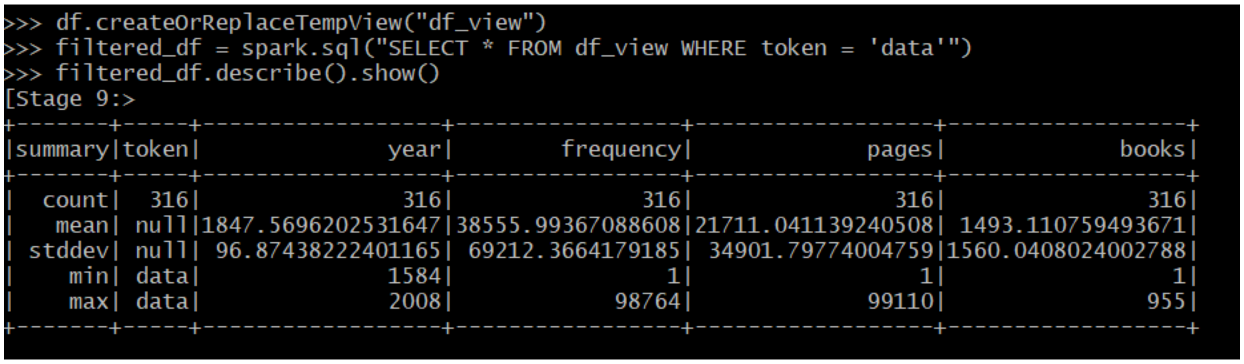
b. Create a new DataFrame from a query using Spark SQL, filtering to include only the rows where the token is "data" and describe the new dataset.

We are creating a new DataFrame by filtering the original DataFrame to include only the rows where the token is "data". Here's a summary of the steps:

1) **Create Temporary View**: The **df.createOrReplaceTempView("df_view")** function is used to create a temporary view named "df_view". This view is a representation of the DataFrame **df** that can be used in SQL queries.

2) **SQL Query**: The **spark.sql("SELECT * FROM df_view WHERE token = 'data'")** function is used to execute a SQL query on the "df_view". The query selects all columns from the view where the token is "data". The result of this query is a new DataFrame, **filtered_df**.

3) **Descriptive Statistics**: The **filtered_df.describe().show()** function is used to generate descriptive statistics of the new DataFrame. It includes count, mean, standard deviation, min, and max for each column.



```
>>> df.createOrReplaceTempView("df_view")
>>> filtered_df = spark.sql("SELECT * FROM df_view WHERE token = 'data'")
>>> filtered_df.describe().show()
[Stage 9:>
+-------+-----+------------------+-----------------+------------------+------------------+
|summary|token|              year|        frequency|             pages|             books|
+-------+-----+------------------+-----------------+------------------+------------------+
|  count|  316|               316|              316|               316|               316|
|   mean| null|1847.5696202531647|38555.99367088608|21711.041139240508| 1493.110759493671|
| stddev| null| 96.87438222401165| 69212.3664179185| 34901.79774004759|1560.0408024002788|
|    min| data|              1584|                1|                 1|                 1|
|    max| data|              2008|            98764|             99110|               955|
+-------+-----+------------------+-----------------+------------------+------------------+
```

The new DataFrame **'filtered_df'**contains 316 rows where the token is "data". The descriptive statistics provide a summary of the 'year', 'frequency', 'pages', and 'books' columns for these rows.

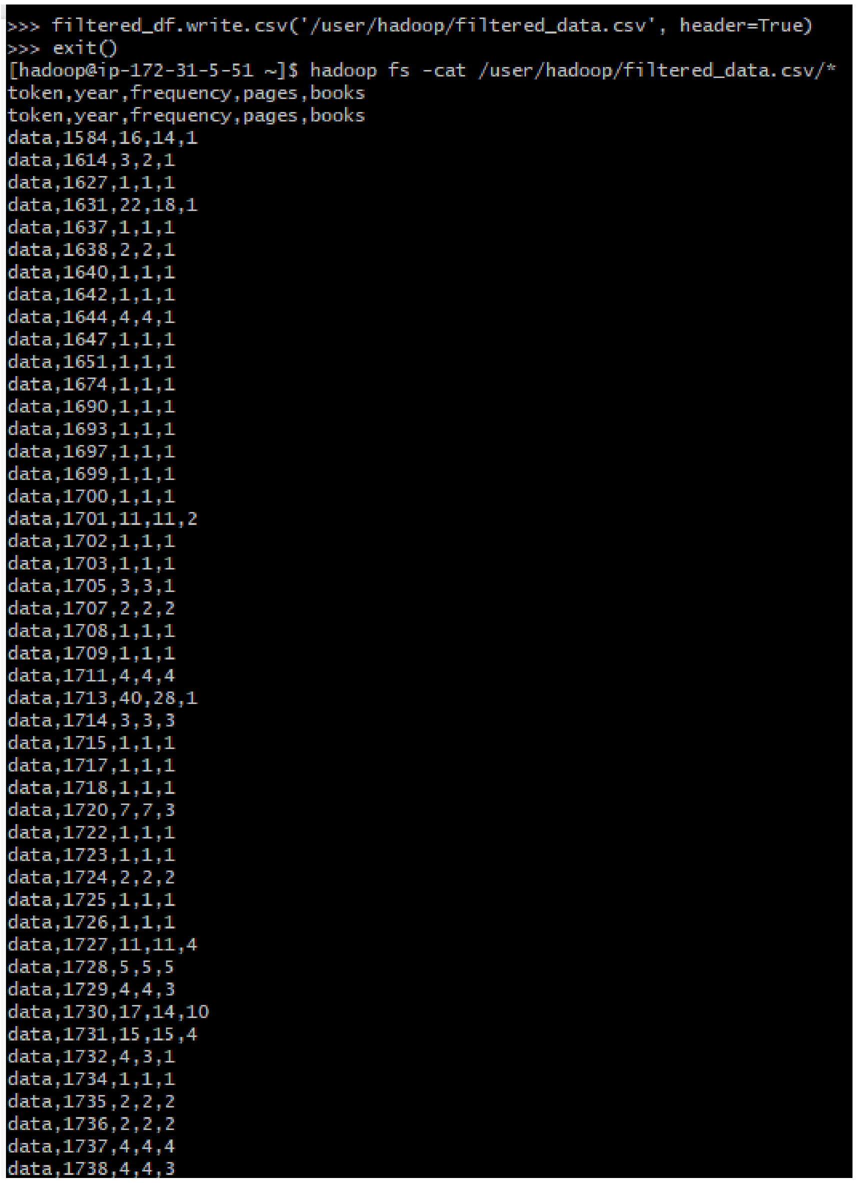
c. Write the filtered data back to a directory in the HDFS from Spark using df.write.csv().
Be sure to pass the header=True parameter and examine the contents of what you've written.

We are writing the filtered DataFrame back to a directory in the Hadoop Distributed File System (HDFS) and examining the contents of the written file. Here's a summary of the steps:

1) **Write to CSV**: The **filtered_df.write.csv('/user/hadoop/filtered_data.csv', header=True)** function is used to write the DataFrame **filtered_df** to a CSV file in the HDFS. The path of the file is '/user/hadoop/filtered_data.csv'. The **header=True** parameter is passed to include the column names in the CSV file.

2) **Exit Spark**: The **exit()** command is used to exit the Spark shell.

3) **Examine Contents**: The **hadoop fs -cat /user/hadoop/filtered_data.csv/*** command is used to concatenate and display the contents of the CSV file in the terminal. This command is executed in the Hadoop shell, not the Spark shell.



```
>>> filtered_df.write.csv('/user/hadoop/filtered_data.csv', header=True)
>>> exit()
[hadoop@ip-172-31-5-51 ~]$ hadoop fs -cat /user/hadoop/filtered_data.csv/*
token,year,frequency,pages,books
token,year,frequency,pages,books
data,1584,16,14,1
data,1614,3,2,1
data,1627,1,1,1
data,1631,22,18,1
data,1637,1,1,1
data,1638,2,2,1
data,1640,1,1,1
data,1642,1,1,1
data,1644,4,4,1
data,1647,1,1,1
data,1651,1,1,1
data,1674,1,1,1
data,1690,1,1,1
data,1693,1,1,1
data,1697,1,1,1
data,1699,1,1,1
data,1700,1,1,1
data,1701,11,11,2
data,1702,1,1,1
data,1703,1,1,1
data,1705,3,3,1
data,1707,2,2,2
data,1708,1,1,1
data,1709,1,1,1
data,1711,4,4,4
data,1713,40,28,1
data,1714,3,3,3
data,1715,1,1,1
data,1717,1,1,1
data,1718,1,1,1
data,1720,7,7,3
data,1722,1,1,1
data,1723,1,1,1
data,1724,2,2,2
data,1725,1,1,1
data,1726,1,1,1
data,1727,11,11,4
data,1728,5,5,5
data,1729,4,4,3
data,1730,17,14,10
data,1731,15,15,4
data,1732,4,3,1
data,1734,1,1,1
data,1735,2,2,2
data,1736,2,2,2
data,1737,4,4,4
data,1738,4,4,3
```

The output shows the contents of the CSV file. Each row corresponds to a record in the DataFrame, and the columns are 'token', 'year', 'frequency', 'pages', and 'books'. The 'token' for all rows is 'data', as per the filter applied in the SQL query.

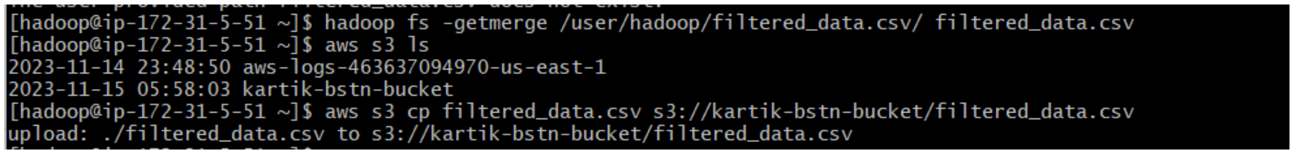
2.5 Collect the contents of the directory into a single file on the local drive of the head node using "getmerge" and move this file into a S3 bucket in your account.

We are collecting the contents of the directory into a single file on the local drive of the head node and moving this file into an S3 bucket. Here's a summary of the steps:

1) **Merge Files**: The **hadoop fs -getmerge /user/hadoop/filtered_data.csv/ filtered_data.csv** command is used to merge all the part files in the '/user/hadoop/filtered_data.csv/' directory into a single file named 'filtered_data.csv' on the local file system of the head node. The 'getmerge' command is useful when the data is split into multiple part files, and we want to consolidate it into a single file.

2) **List S3 Buckets**: The **aws s3 ls** command is used to list all the S3 buckets in the AWS account. In this case, there is a single bucket named **'kartik-bstn-bucket'**.

3) **Copy File to S3**: The **aws s3 cp filtered_data.csv s3://kartik-bstn-bucket/filtered_data.csv** command is used to copy the 'filtered_data.csv' file from the local file system to the 'kartik-bstnbucket' S3 bucket. The 'cp' command is used to copy files between the local file system and S3, or between two S3 buckets.



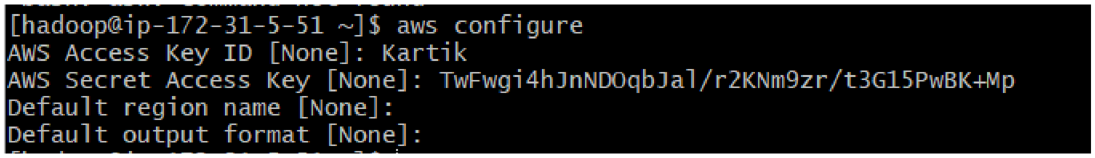
```
[hadoop@ip-172-31-5-51 ~]$ hadoop fs -getmerge /user/hadoop/filtered_data.csv/ filtered_data.csv
[hadoop@ip-172-31-5-51 ~]$ aws s3 ls
2023-11-14 23:48:50 aws-logs-463637094970-us-east-1
2023-11-15 05:58:03 kartik-bstn-bucket
[hadoop@ip-172-31-5-51 ~]$ aws s3 cp filtered_data.csv s3://kartik-bstn-bucket/filtered_data.csv
upload: ./filtered_data.csv to s3://kartik-bstn-bucket/filtered_data.csv
```

The output shows that the 'filtered_data.csv' file has been successfully uploaded to the 'kartik-bstn-bucket' S3 bucket.

2.6 On your local machine (or on AWS outside of Spark) in python, read the CSV data from the S3 folder into a pandas DataFrame (You will have to research how to read data into pandas from S3 buckets). Note You must have first authenticated on your machine using aws configure on the command line to complete this step).

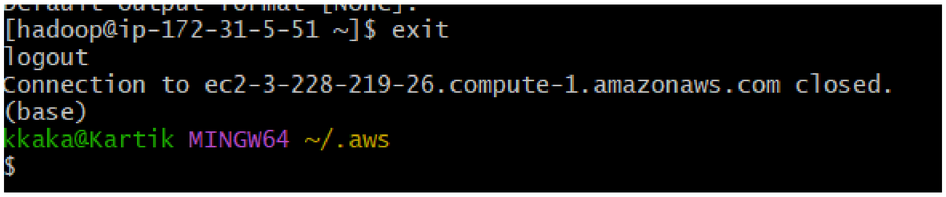
We are preparing to read the CSV data from the S3 bucket into a pandas DataFrame on their local machine. Here's a summary of the steps:

1) AWS Configuration: The **aws configure** command is used to set up AWS credentials and default region on the local machine. This step is necessary to interact with AWS services, including S3. We are prompted to enter their AWS Access Key ID, AWS Secret Access Key, Default region name, and Default output format.



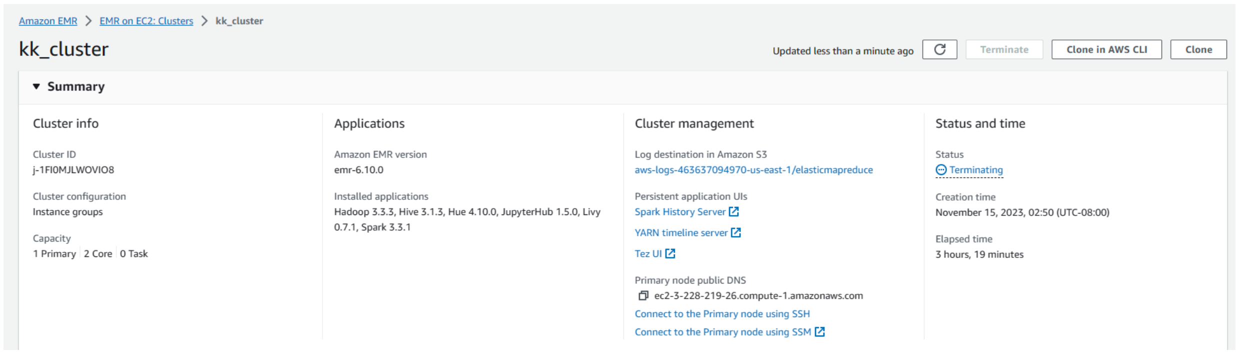
```
[hadoop@ip-172-31-5-51 ~]$ aws configure
AWS Access Key ID [None]: Kartik
AWS Secret Access Key [None]: TwFwgi4hJnNDOqbJal/r2KNm9zr/t3G15PwBK+Mp
Default region name [None]:
Default output format [None]:
```

2) Exit SSH Session: The **exit** command is used to close the SSH session and return to the local terminal.



```
[hadoop@ip-172-31-5-51 ~]$ exit
logout
Connection to ec2-3-228-219-26.compute-1.amazonaws.com closed.
(base)
kkaka@Kartik MINGW64 ~/.aws
$
```

3) Terminate cluster on AWS.



Amazon EMR > EMR on EC2: Clusters > kk_cluster

kk_cluster

Updated less than a minute ago

Summary

Cluster info

Cluster ID
j-1FI0MJLWOVIO8

Cluster configuration
Instance groups

Capacity
1 Primary | 2 Core | 0 Task

Applications

Amazon EMR version
emr-6.10.0

Installed applications
Hadoop 3.3.3, Hive 3.1.3, Hue 4.10.0, JupyterHub 1.5.0, Livy 0.7.1, Spark 3.3.1

Cluster management

Log destination in Amazon S3
aws-logs-463637094970-us-east-1/elasticmapreduce

Persistent application UIs
Spark History Server
YARN timeline server
Tez UI

Primary node public DNS
ec2-3-228-219-26.compute-1.amazonaws.com
Connect to the Primary node using SSH
Connect to the Primary node using SSM

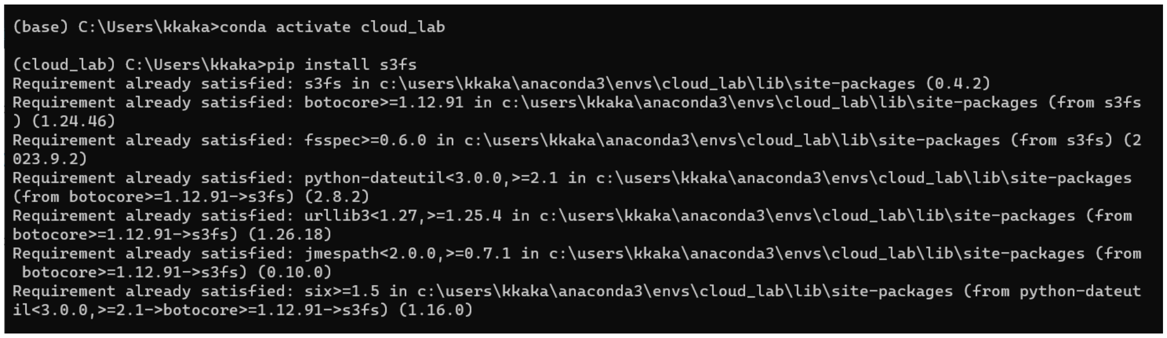
Status and time

Status
Terminating

Creation time
November 15, 2023, 02:50 (UTC-08:00)

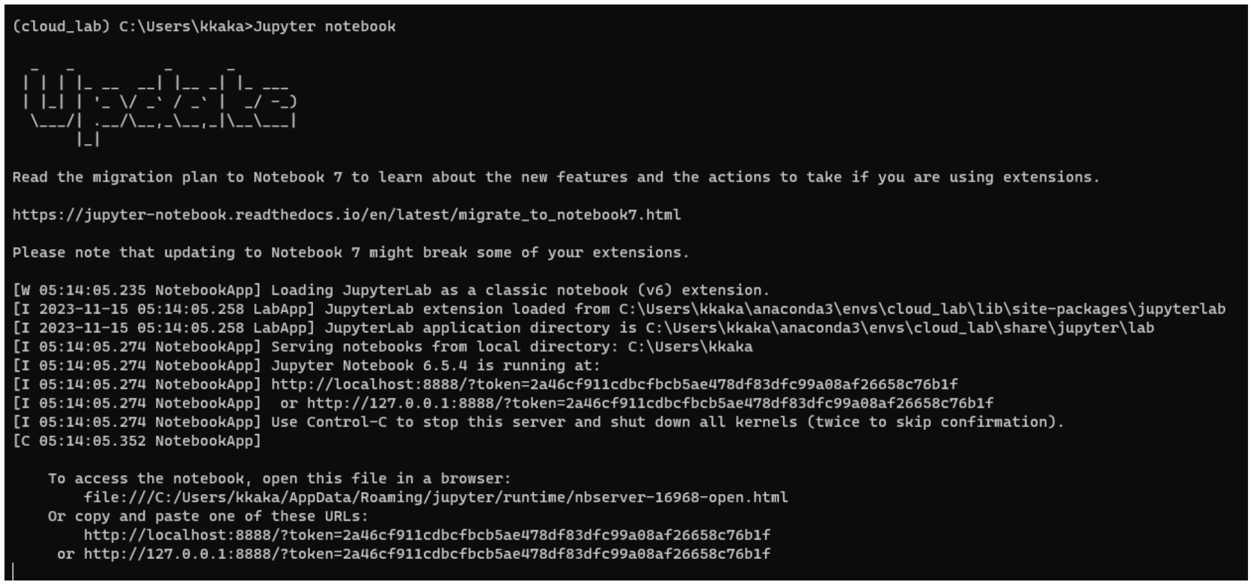
Elapsed time
3 hours, 19 minutes

4) Install s3fs: The **pip install s3fs** command is used to install the **s3fs** Python library if not already installed. This library is used to create a connection between Python and S3, allowing Python to read and write files in S3.



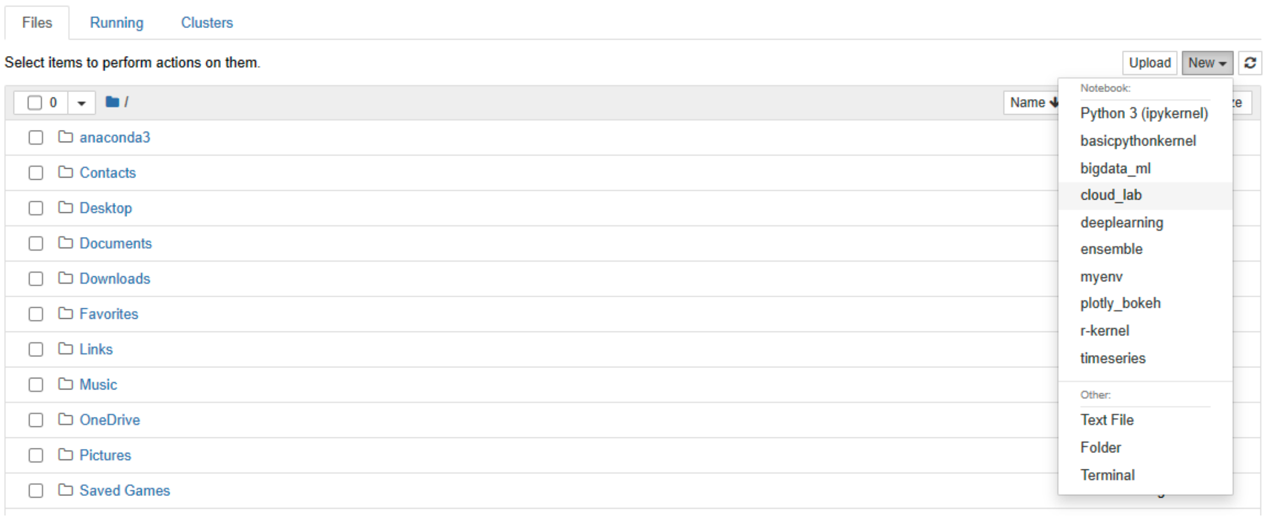
```
(base) C:\Users\kkaka>conda activate cloud_lab

(cloud_lab) C:\Users\kkaka>pip install s3fs
Requirement already satisfied: s3fs in c:\users\kkaka\anaconda3\envs\cloud_lab\lib\site-packages (0.4.2)
Requirement already satisfied: botocore>=1.12.91 in c:\users\kkaka\anaconda3\envs\cloud_lab\lib\site-packages (from s3fs
) (1.24.46)
Requirement already satisfied: fsspec>=0.6.0 in c:\users\kkaka\anaconda3\envs\cloud_lab\lib\site-packages (from s3fs) (2
023.9.2)
Requirement already satisfied: python-dateutil<3.0.0,>=2.1 in c:\users\kkaka\anaconda3\envs\cloud_lab\lib\site-packages
(from botocore>=1.12.91->s3fs) (2.8.2)
Requirement already satisfied: urllib3<1.27,>=1.25.4 in c:\users\kkaka\anaconda3\envs\cloud_lab\lib\site-packages (from
botocore>=1.12.91->s3fs) (1.26.18)
Requirement already satisfied: jmespath<2.0.0,>=0.7.1 in c:\users\kkaka\anaconda3\envs\cloud_lab\lib\site-packages (from
 botocore>=1.12.91->s3fs) (0.10.0)
Requirement already satisfied: six>=1.5 in c:\users\kkaka\anaconda3\envs\cloud_lab\lib\site-packages (from python-dateut
il<3.0.0,>=2.1->botocore>=1.12.91->s3fs) (1.16.0)
```

5) Launch Jupyter Notebook: The **jupyter notebook** command is used to start a Jupyter notebook server on the local machine.



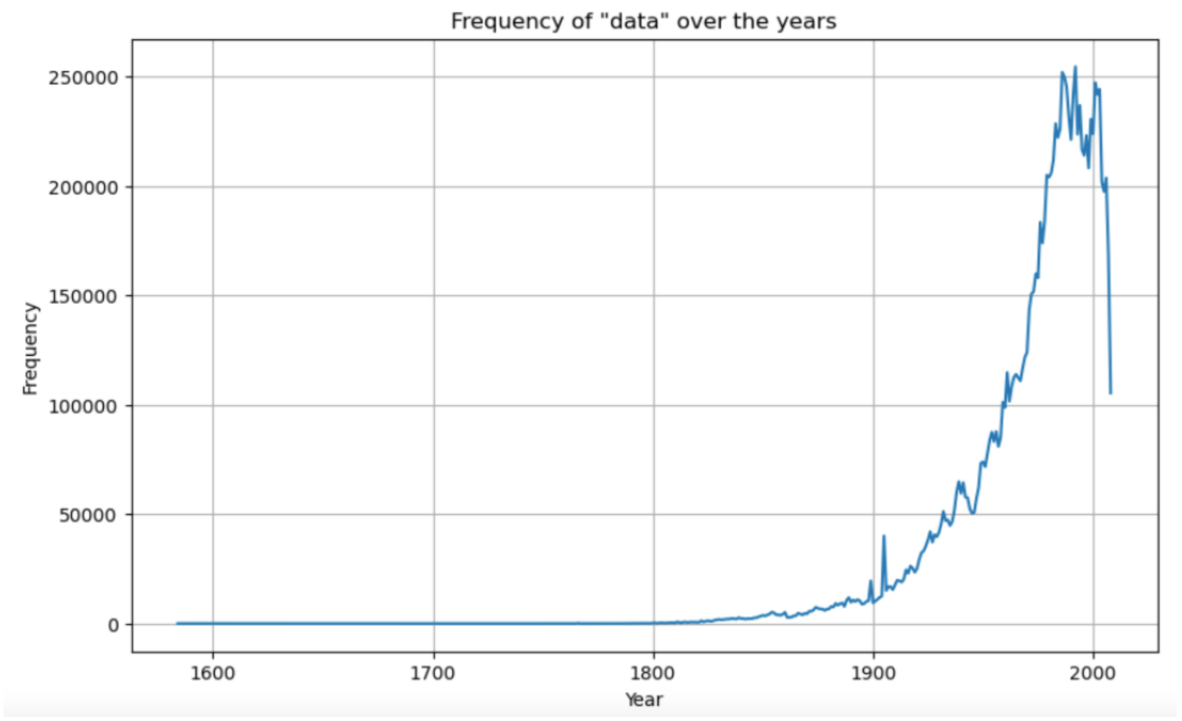
```
(cloud_lab) C:\Users\kkaka>Jupyter notebook

  _   _          _      _
 | | | |_ __  __| |__ _| |_ ___
 | |_| | '_ \/ _` / _` |  _/ -_)
  \___/| .__/\__,_\__,_|\__\___|
       |_|

Read the migration plan to Notebook 7 to learn about the new features and the actions to take if you are using extensions.

https://jupyter-notebook.readthedocs.io/en/latest/migrate_to_notebook7.html

Please note that updating to Notebook 7 might break some of your extensions.

[W 05:14:05.235 NotebookApp] Loading JupyterLab as a classic notebook (v6) extension.
[I 2023-11-15 05:14:05.258 LabApp] JupyterLab extension loaded from C:\Users\kkaka\anaconda3\envs\cloud_lab\lib\site-packages\jupyterlab
[I 2023-11-15 05:14:05.258 LabApp] JupyterLab application directory is C:\Users\kkaka\anaconda3\envs\cloud_lab\share\jupyter\lab
[I 05:14:05.274 NotebookApp] Serving notebooks from local directory: C:\Users\kkaka
[I 05:14:05.274 NotebookApp] Jupyter Notebook 6.5.4 is running at:
[I 05:14:05.274 NotebookApp] http://localhost:8888/?token=2a46cf911cdbcfbcb5ae478df83dfc99a08af26658c76b1f
[I 05:14:05.274 NotebookApp]  or http://127.0.0.1:8888/?token=2a46cf911cdbcfbcb5ae478df83dfc99a08af26658c76b1f
[I 05:14:05.274 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 05:14:05.352 NotebookApp]

    To access the notebook, open this file in a browser:
        file:///C:/Users/kkaka/AppData/Roaming/jupyter/runtime/nbserver-16968-open.html
    Or copy and paste one of these URLs:
        http://localhost:8888/?token=2a46cf911cdbcfbcb5ae478df83dfc99a08af26658c76b1f
     or http://127.0.0.1:8888/?token=2a46cf911cdbcfbcb5ae478df83dfc99a08af26658c76b1f
```

6) We can then **create a new notebook** and write Python code to read the CSV data from the S3 bucket into a pandas DataFrame.



Please refer to the Jupyter notebook for further work.

2.7 Plot the number of occurrence of the token (the frequency column) of data over the years using matplotlib.



The plot shows how the frequency of the token 'data' has changed over the years, it was steadily increasing between 1900s and 2000s, and dropped rapidly after 2000s. The peak in frequency could be associated with significant events, technological advancements, or shifts in language usage during that timeframe. The rise and fall of the frequency could be linked to broader cultural or technological shifts. For example, the increase might be related to the proliferation of data-related technologies, while the decline could coincide with shifts in language usage or changes in the focus of content.

2.8 Compare Hadoop and Spark as distributed file systems.

a. What are the advantages/ differences between Hadoop and Spark? List two advantages for each.

Apache Hadoop and Apache Spark are both big data processing frameworks, but they have some key differences in terms of architecture, performance, and use cases. Here are two advantages for each framework:

1. Hadoop

- Batch Processing: Hadoop is well-suited for batch processing tasks. It excels in processing large volumes of data in a distributed and fault-tolerant manner. Hadoop's MapReduce paradigm allows for the efficient processing of massive datasets by dividing them into smaller chunks and processing them in parallel.

- Mature Ecosystem: Hadoop has a mature and well-established ecosystem with various components such as HDFS (Hadoop Distributed File System), MapReduce, Hive, Pig, and others. This ecosystem provides a comprehensive set of tools for data storage, data processing, and data analysis. Hadoop's ecosystem has been widely adopted and has a large community of users and contributors.

2. Spark

- In-Memory Processing: One of Spark's significant advantages is its ability to perform in-memory processing. Unlike Hadoop MapReduce, which relies heavily on disk-based storage between processing stages, Spark keeps intermediate data in memory. This results in significantly faster data processing as it reduces the need for data to be written to and read from disk, making Spark suitable for iterative algorithms and interactive data analysis.

- Unified Data Processing Engine: Spark provides a unified data processing engine that supports batch processing, interactive queries, streaming analytics, and machine learning within a single framework. This flexibility allows users to perform a wide range of data processing tasks using the same codebase, simplifying development and reducing the need for multiple specialized tools.

b. Explain how the HDFS stores the data.

Hadoop Distributed File System (HDFS) is the distributed file system used by Apache Hadoop for storing and managing large volumes of data across a cluster of commodity hardware. HDFS is designed to provide reliable, scalable, and fault-tolerant storage for big data applications. Here's an explanation of how HDFS stores data:

- **File Write:** When a new file is written to HDFS, the client application contacts the NameNode to request the creation of the file. The NameNode checks if the file already exists and allocates metadata space for the new file.
- **Block Division:** The file is divided into fixed-size blocks. The block size is configurable but is typically several megabytes or even gigabytes in size.
- **Block Replication:** Each block is replicated across multiple DataNodes in the cluster. The replication factor is configurable and determines how many copies of each block are stored. Replication helps in achieving fault tolerance and data availability.
- **DataNode Storage:** The actual data blocks are stored on the local file systems of the DataNodes. Each DataNode independently manages its block replicas.
- **Metadata Management:** The NameNode keeps track of the metadata, including the location and replication factor of each block. It maintains the file namespace, directory structure, and metadata about each file and directory.

Part 3. Conclusion

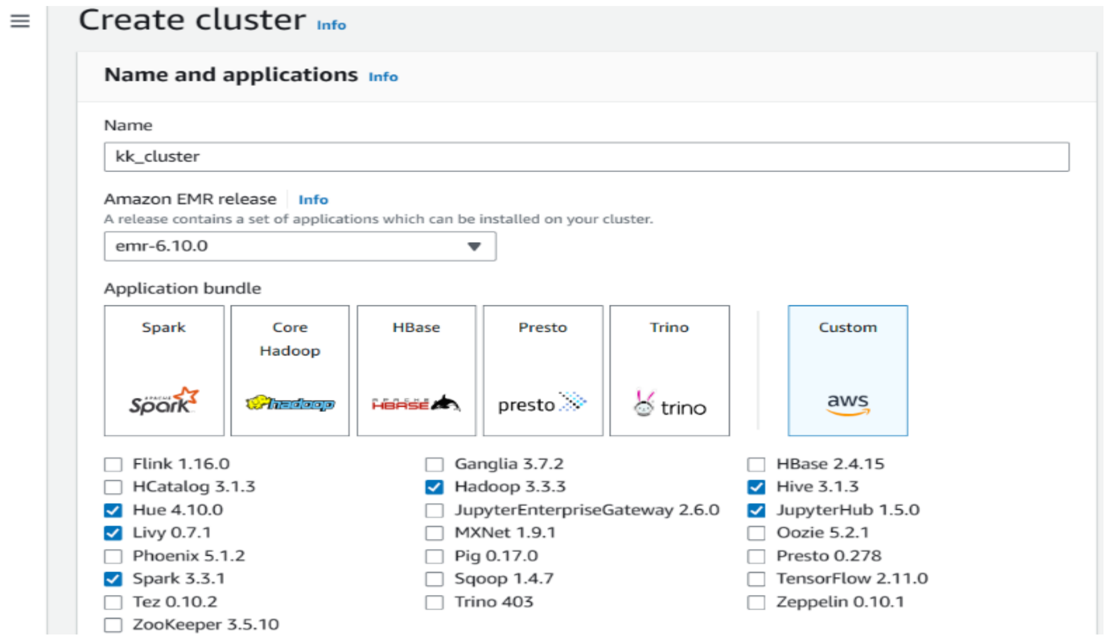
We have successfully completed a comprehensive data analysis workflow involving multiple technologies including Apache Hadoop, AWS S3, Python, pandas, and matplotlib.
The workflow can be summarized as follows:

- **Data Filtering with Hadoop**: We utilized Apache Hadoop to filter a large dataset stored on an AWS EMR cluster. The Hadoop MapReduce job was written in Python and was used to filter out specific tokens from the dataset.
- **Data Transfer to S3**: After filtering the data, we collected the contents of the directory into a single file and moved this file into an S3 bucket in their AWS account using the **aws s3 cp** command.
- **Data Loading with pandas**: We then switched to their local machine and read the CSV data from the S3 bucket into a pandas DataFrame. This was done using the **s3fs** library to create a connection to S3 and then using pandas' **read_csv** function to load the data.
- **Data Visualization with matplotlib**: Finally, we visualized the frequency of the token 'data' over the years using matplotlib. The 'year' and 'frequency' columns were converted to integers and a line plot was created to show the trend over time.

In conclusion, this workflow demonstrates a powerful combination of distributed computing, cloud storage, and data analysis tools to process, analyze, and visualize large datasets. We were able to filter a large dataset using Hadoop, store and retrieve the data using AWS S3, and then analyze and visualize the data using pandas and matplotlib. This workflow is a good example of how different technologies can be combined to handle big data tasks.

Part 2.1 Screenshots

1.



Create cluster Info

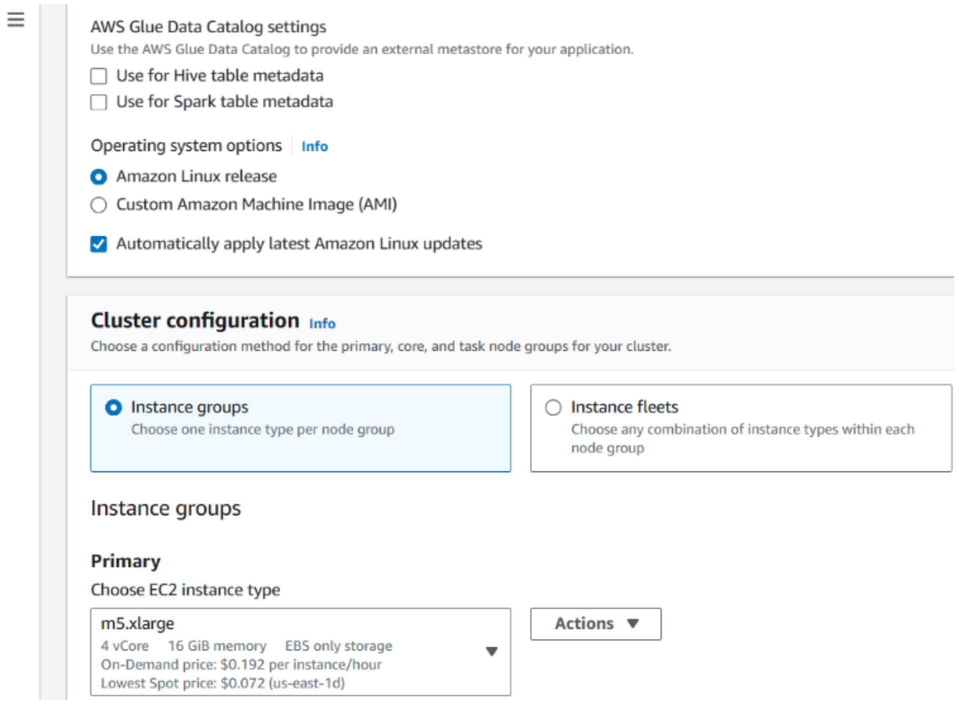
Name and applications Info

Name
kk_cluster

Amazon EMR release Info
A release contains a set of applications which can be installed on your cluster.
emr-6.10.0

Application bundle
Spark | Core Hadoop | HBase | Presto | Trino | Custom

- [ ] Flink 1.16.0
- [ ] HCatalog 3.1.3
- [x] Hue 4.10.0
- [x] Livy 0.7.1
- [ ] Phoenix 5.1.2
- [x] Spark 3.3.1
- [ ] Tez 0.10.2
- [ ] ZooKeeper 3.5.10
- [ ] Ganglia 3.7.2
- [x] Hadoop 3.3.3
- [ ] JupyterEnterpriseGateway 2.6.0
- [ ] MXNet 1.9.1
- [ ] Pig 0.17.0
- [ ] Sqoop 1.4.7
- [ ] Trino 403
- [ ] HBase 2.4.15
- [x] Hive 3.1.3
- [x] JupyterHub 1.5.0
- [ ] Oozie 5.2.1
- [ ] Presto 0.278
- [ ] TensorFlow 2.11.0
- [ ] Zeppelin 0.10.1


AWS Glue Data Catalog settings

Use the AWS Glue Data Catalog to provide an external metastore for your application.

☐ Use for Hive table metadata

☐ Use for Spark table metadata

Operating system options Info

● Amazon Linux release

○ Custom Amazon Machine Image (AMI)

✓ Automatically apply latest Amazon Linux updates

Cluster configuration Info

Choose a configuration method for the primary, core, and task node groups for your cluster.

● Instance groups — Choose one instance type per node group

○ Instance fleets — Choose any combination of instance types within each node group

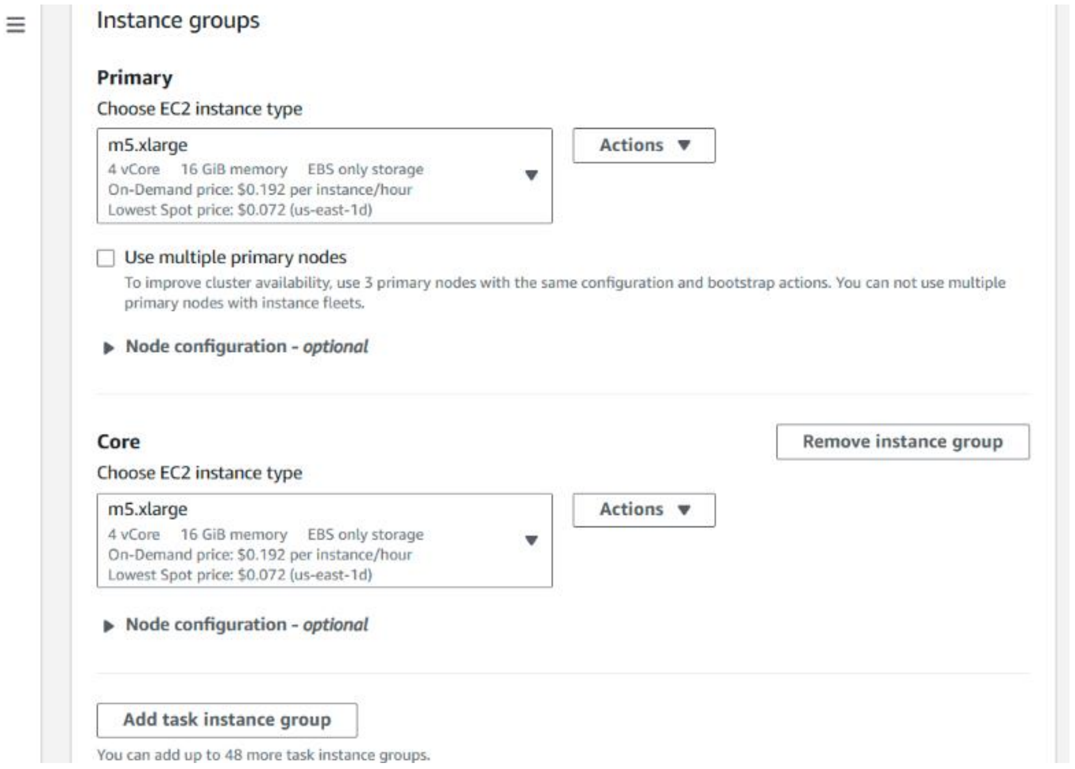
Instance groups

Primary

Choose EC2 instance type

m5.xlarge
4 vCore 16 GiB memory EBS only storage
On-Demand price: $0.192 per instance/hour
Lowest Spot price: $0.072 (us-east-1d)

2.


Instance groups

Primary

Choose EC2 instance type

m5.xlarge
4 vCore 16 GiB memory EBS only storage
On-Demand price: $0.192 per instance/hour
Lowest Spot price: $0.072 (us-east-1d)

☐ Use multiple primary nodes

To improve cluster availability, use 3 primary nodes with the same configuration and bootstrap actions. You can not use multiple primary nodes with instance fleets.

▶ **Node configuration - *optional***

Core

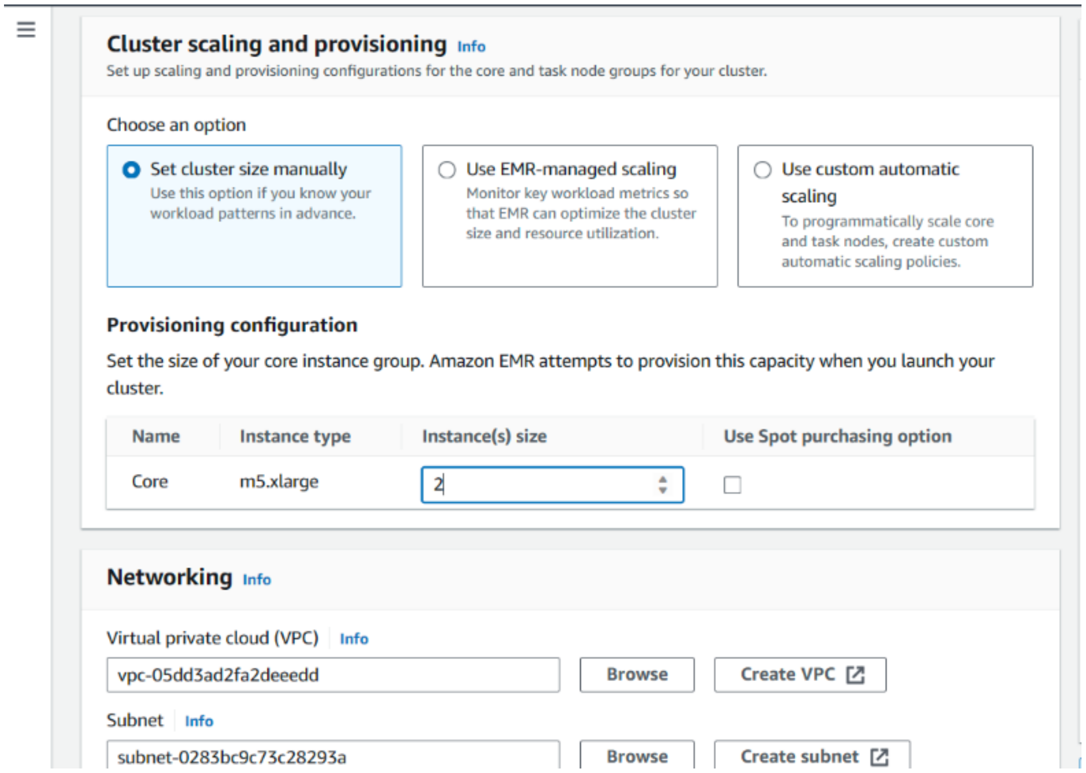
Choose EC2 instance type

m5.xlarge
4 vCore 16 GiB memory EBS only storage
On-Demand price: $0.192 per instance/hour
Lowest Spot price: $0.072 (us-east-1d)

▶ **Node configuration - *optional***

You can add up to 48 more task instance groups.

3.



4.

Cluster scaling and provisioning Info

Set up scaling and provisioning configurations for the core and task node groups for your cluster.

Choose an option

[✓ Set cluster size manually] Use this option if you know your workload patterns in advance.

Use EMR-managed scaling: Monitor key workload metrics so that EMR can optimize the cluster size and resource utilization.

Use custom automatic scaling: To programmatically scale core and task nodes, create custom automatic scaling policies.

Provisioning configuration

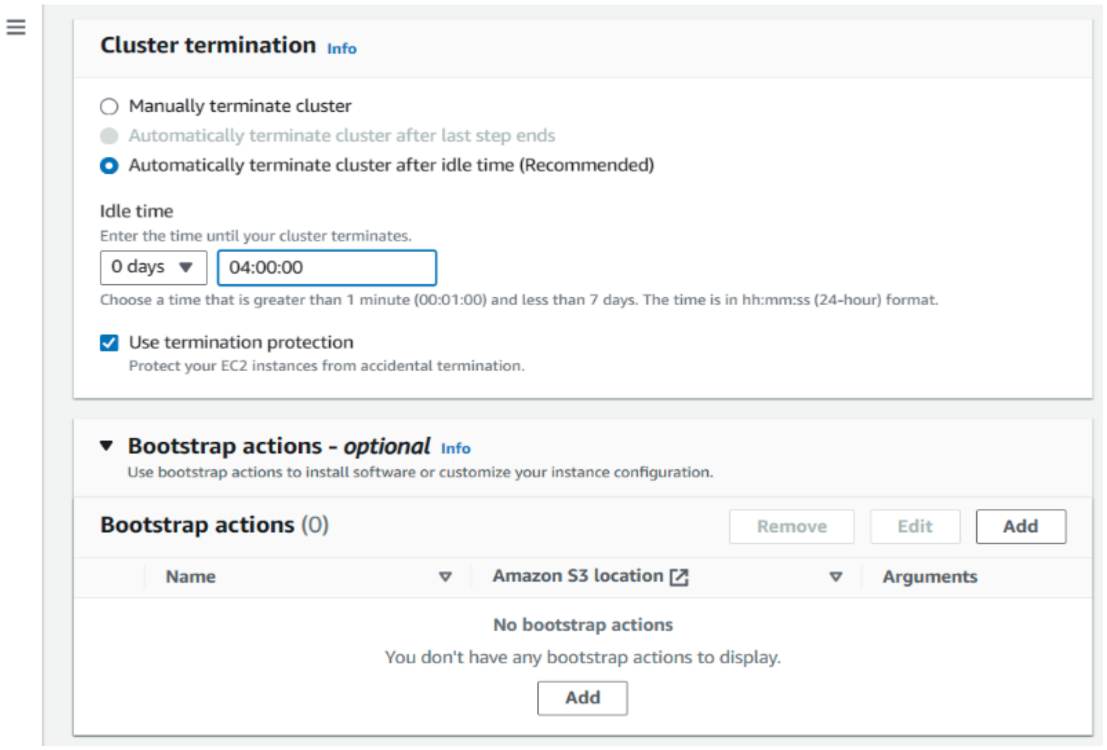
Set the size of your core instance group. Amazon EMR attempts to provision this capacity when you launch your cluster.

Name	Instance type	Instance(s) size	Use Spot purchasing option
Core	m5.xlarge	2	☐

Networking Info

Virtual private cloud (VPC) Info: vpc-05dd3ad2fa2deeedd

Subnet Info: subnet-0283bc9c73c28293a



5.

Cluster termination Info

Manually terminate cluster

Automatically terminate cluster after last step ends

[✓ Automatically terminate cluster after idle time (Recommended)]

Idle time

Enter the time until your cluster terminates.

0 days 04:00:00

Choose a time that is greater than 1 minute (00:01:00) and less than 7 days. The time is in hh:mm:ss (24-hour) format.

☑ Use termination protection

Protect your EC2 instances from accidental termination.

Bootstrap actions - *optional* Info

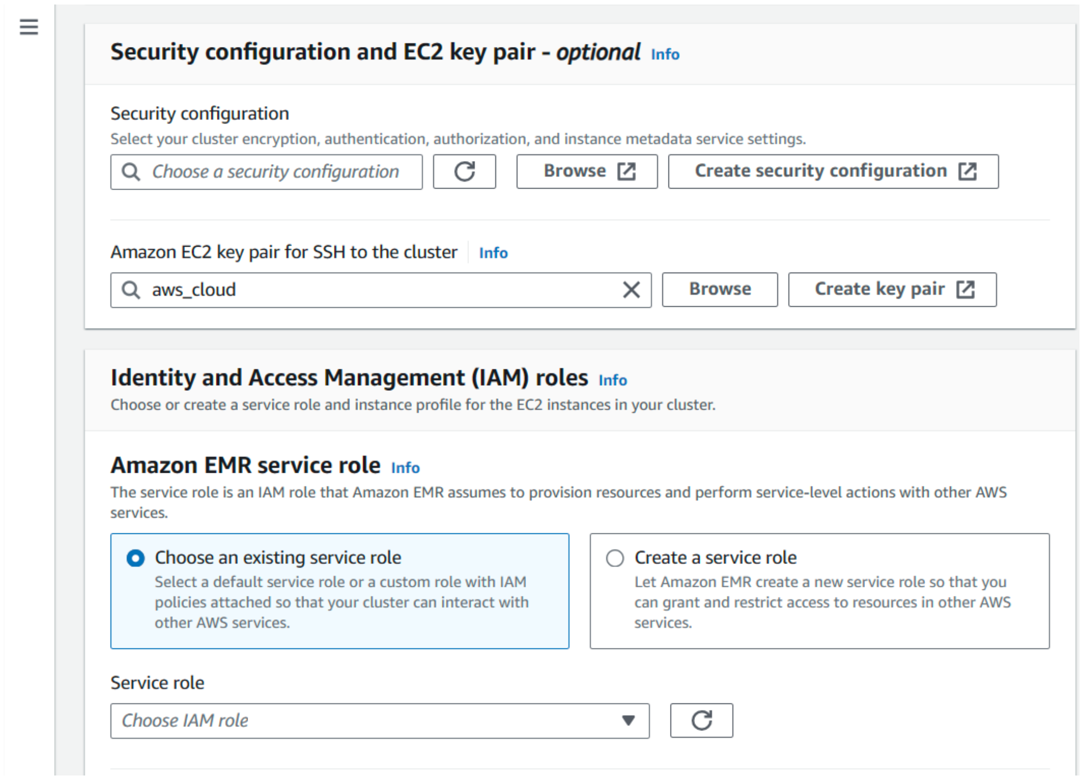
Use bootstrap actions to install software or customize your instance configuration.

Bootstrap actions (0)

Name	Amazon S3 location	Arguments

No bootstrap actions

You don't have any bootstrap actions to display.



Security configuration and EC2 key pair - *optional* Info

Security configuration

Select your cluster encryption, authentication, authorization, and instance metadata service settings.

Choose a security configuration
Browse
Create security configuration

Amazon EC2 key pair for SSH to the cluster Info

aws_cloud

Identity and Access Management (IAM) roles Info

Choose or create a service role and instance profile for the EC2 instances in your cluster.

Amazon EMR service role Info

The service role is an IAM role that Amazon EMR assumes to provision resources and perform service-level actions with other AWS services.

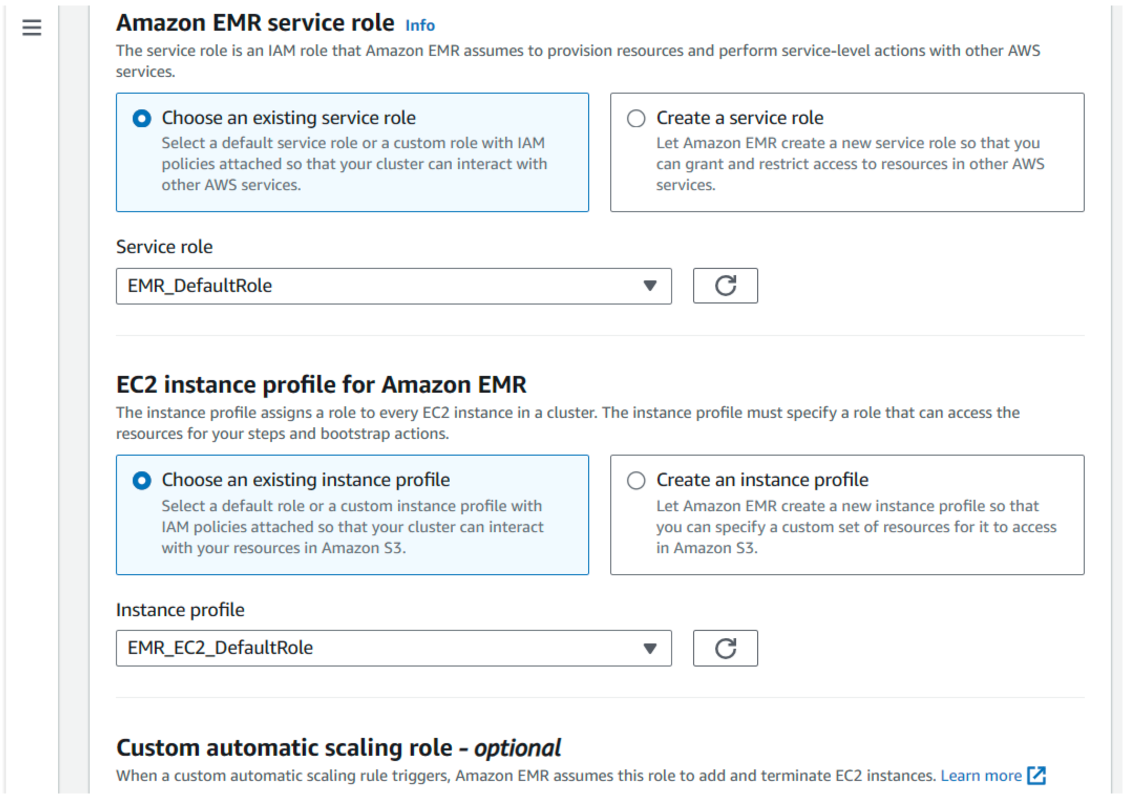
[✓ Choose an existing service role] Select a default service role or a custom role with IAM policies attached so that your cluster can interact with other AWS services.

Create a service role — Let Amazon EMR create a new service role so that you can grant and restrict access to resources in other AWS services.

Service role

Choose IAM role

6.



Amazon EMR service role Info

The service role is an IAM role that Amazon EMR assumes to provision resources and perform service-level actions with other AWS services.

[✓ Choose an existing service role] Select a default service role or a custom role with IAM policies attached so that your cluster can interact with other AWS services.

Create a service role — Let Amazon EMR create a new service role so that you can grant and restrict access to resources in other AWS services.

Service role

EMR_DefaultRole

EC2 instance profile for Amazon EMR

The instance profile assigns a role to every EC2 instance in a cluster. The instance profile must specify a role that can access the resources for your steps and bootstrap actions.

[✓ Choose an existing instance profile] Select a default role or a custom instance profile with IAM policies attached so that your cluster can interact with your resources in Amazon S3.

Create an instance profile — Let Amazon EMR create a new instance profile so that you can specify a custom set of resources for it to access in Amazon S3.

Instance profile

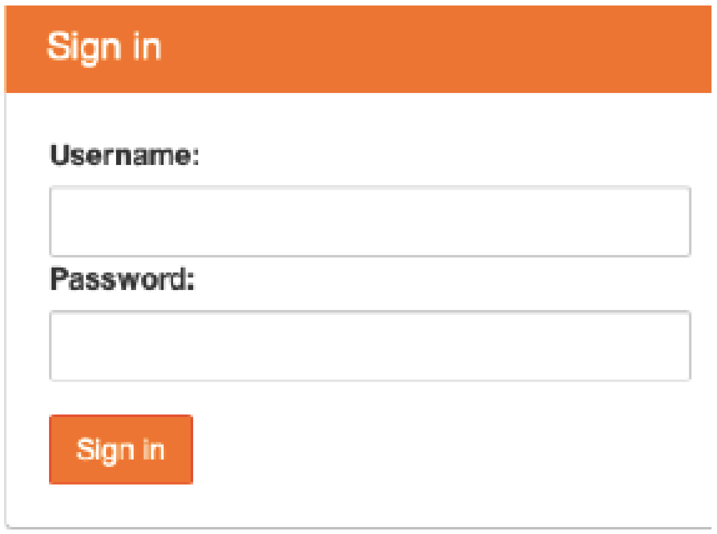
EMR_EC2_DefaultRole

Custom automatic scaling role - *optional*

When a custom automatic scaling rule triggers, Amazon EMR assumes this role to add and terminate EC2 instances. Learn more

7.

8.


Sign in
Username:
Password:
Sign in