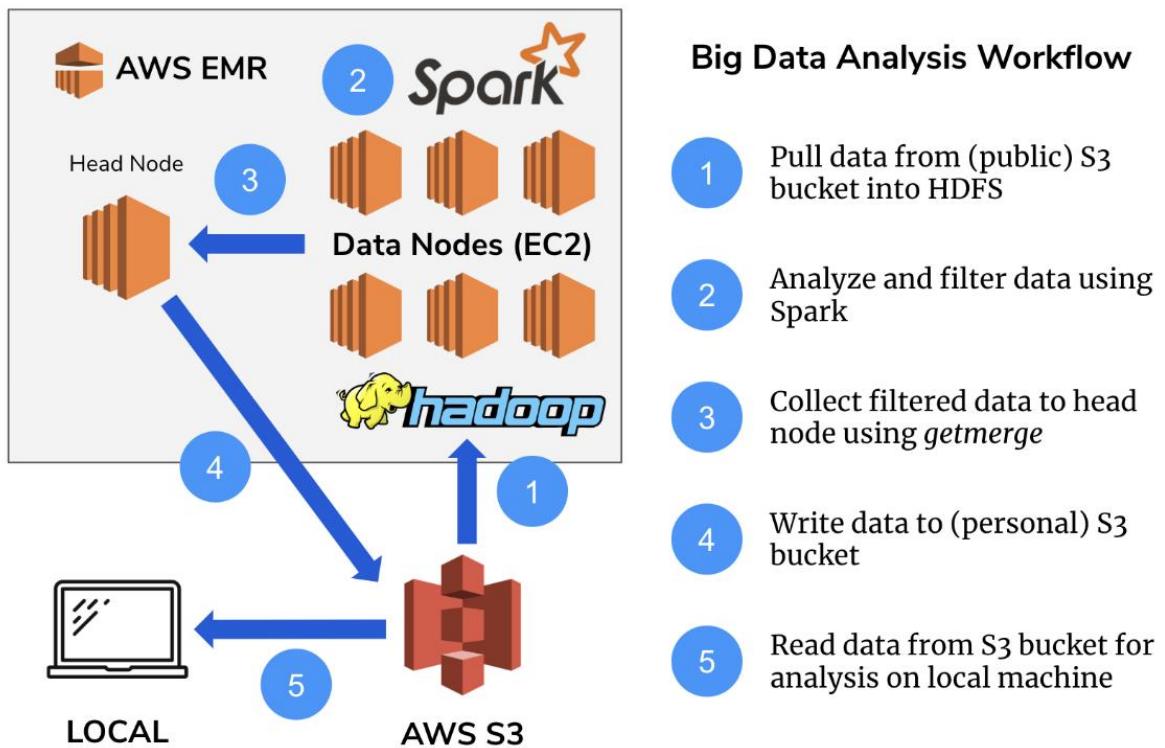


Big Data Wrangling with Google Books Ngrams

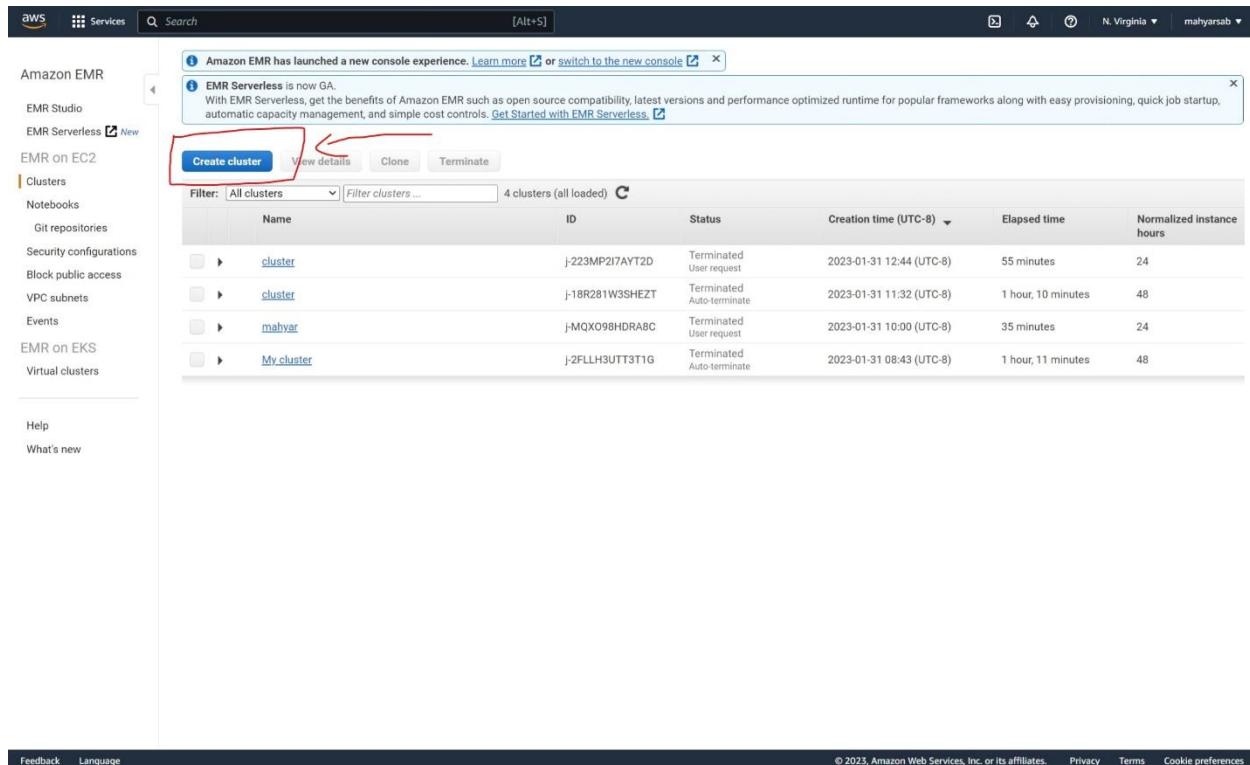
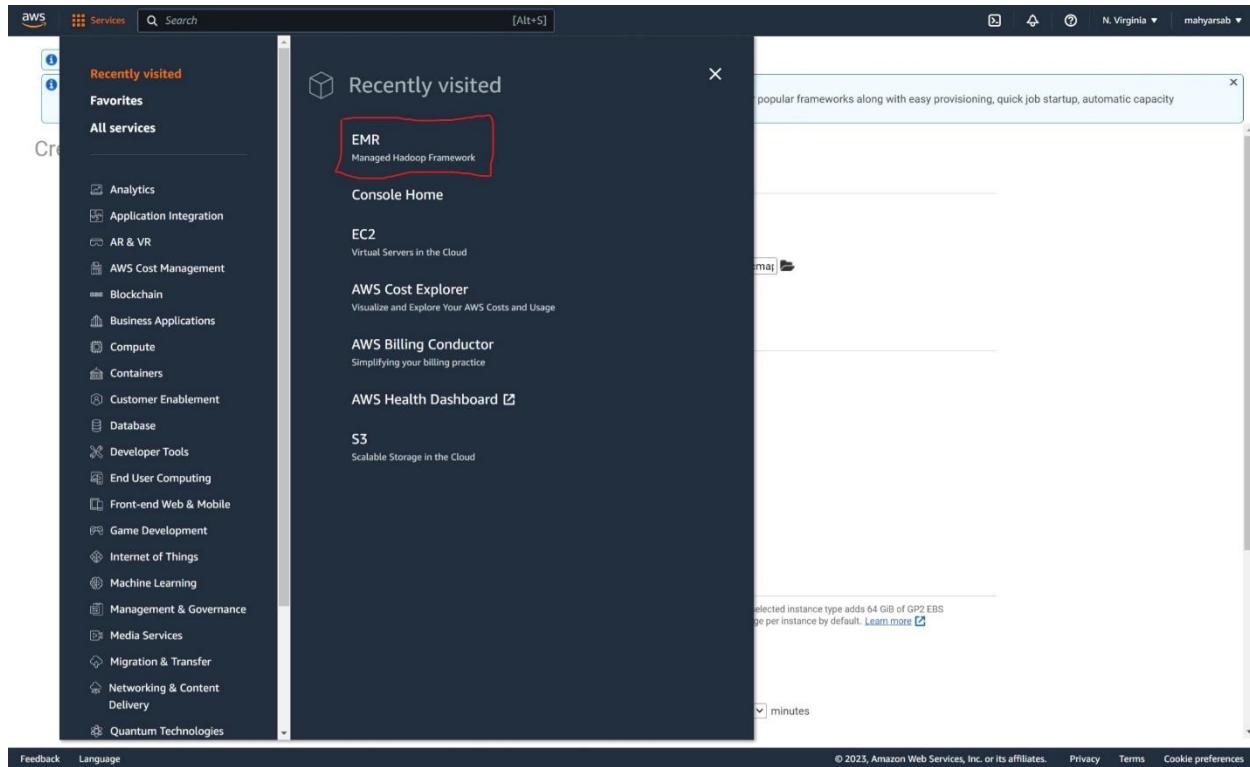
For this deliverable, you will produce a report, as well as a jupyter notebook, which will follow a Big Data analysis workflow. As part of this workflow you will filter and reduce data down to a manageable size, and then do some analysis locally on our machine after extracting data from the Cloud and processing it using Big Data tools. The workflow and steps in the process are illustrated below:

Copyright  BrainStation



Question 1: Spin up a new EMR cluster using the AWS Console. Be sure to include Hadoop, Spark, Hive, Jupyterhub, and Livy for your cluster. **For the release version, make sure to use EMR 6.1.1:**

1. First we will sign in to our amazon aws account and launch an EMR cluster like the steps below:



- Click on create cluster and go to advanced options:

Create Cluster - Quick Options

General Configuration

- Cluster name: My cluster
- Logging: checked
- S3 folder: \$3://aws-log-181777116252-us-east-1/elasticmca/
- Launch mode: Cluster (radio button selected)

Software configuration

- Release: emr-5.36.0
- Applications:
 - Core Hadoop: Hadoop 2.10.1, Hive 2.3.9, Hue 4.10.0, Mahout 0.13.0, Pig 0.17.0, and Tez 0.9.2
 - HBase: HBase 1.4.13, Hadoop 2.10.1, Hive 2.3.9, Hue 4.10.0, Phoenix 4.14.3, and ZooKeeper 3.4.14
 - Presto: Presto 0.267 with Hadoop 2.10.1 HDFS and Hive 2.3.9 Metastore
 - Spark: Spark 2.4.8 on Hadoop 2.10.1 YARN and Zeppelin 0.10.0
- Use AWS Glue Data Catalog for table metadata

Hardware configuration

- Instance type: m5.xlarge
- The selected instance type adds 64 GB 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](#)
- Terminate cluster when it is idle after: 1 hours 0 minutes

Feedback Language © 2023, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

- Choose the abovementioned items in first step:

Create Cluster - Advanced Options

Step 1: Software and Steps

Step 2: Hardware

Step 3: General Cluster Settings

Step 4: Security

Software Configuration

Release: emr-6.1.1

Selected Applications (checked):

- Hadoop 3.2.1
- JupyterHub 1.1.0
- Ganglia 3.7.2
- Hive 3.1.2
- ZooKeeper 3.4.14
- Hue 4.7.1
- Spark 3.0.0

Available Applications (unchecked):

- Zookeeper 3.4.14
- Hive 3.1.2
- Tez 0.9.2
- HBase 2.2.5
- Presto 0.232
- MXNet 1.6.0
- Phoenix 5.0.0
- HCatalog 3.1.2
- ZooKeeper 3.4.14
- Hive 3.1.2
- Tez 0.9.2
- HBase 2.2.5
- Presto 0.232
- MXNet 1.6.0
- Phoenix 5.0.0
- HCatalog 3.1.2
- Livy 0.7.0
- Flink 1.11.0
- Pig 0.17.0
- PrestoSQL 338
- Sqoop 1.4.7
- Oozie 5.2.0
- TensorFlow 2.1.0

Multiple master nodes (optional):
 Use multiple master nodes to improve cluster availability. [Learn more](#)

AWS Glue Data Catalog settings (optional):
 Use for Hive table metadata
 Use for Spark table metadata

Edit software settings:
 Enter configuration Load JSON from S3
 classification-config-file-name.properties=[myKey1=myValue1,myKey2=myValue2]

Steps (optional)

A step is a unit of work you submit to the cluster. For instance, a step might contain one or more Hadoop or Spark jobs. You can also submit additional steps to a cluster after it is running. [Learn more](#)

Concurrency:
 Run multiple steps at the same time to improve cluster utilization

After last step completes:
 Clusters enters waiting state
 Cluster auto-terminates

Feedback Language © 2023, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

- Choose a name for your cluster in step 3:

AWS Services Search [Alt+S] N. Virginia mahyarsab

Amazon EMR has launched a new console experience. Learn more or switch to the new console.

EMR Serverless is now GA. With EMR Serverless, get the benefits of Amazon EMR such as open source compatibility, latest versions and performance optimized runtime for popular frameworks along with easy provisioning, quick job startup, automatic capacity management, and simple cost controls. Get Started with EMR Serverless.

Create Cluster - Advanced Options Go to quick options

Step 1: Software and Steps
Step 2: Hardware
Step 3: General Cluster Settings
Step 4: Security

General Options

Cluster name: mahyar_cluster

Logging: S3 folder: s3://aws-logs-181777116252-us-east-1/elasticmapreduce

Log encryption

Termination protection

Tags

Add a key to create a tag

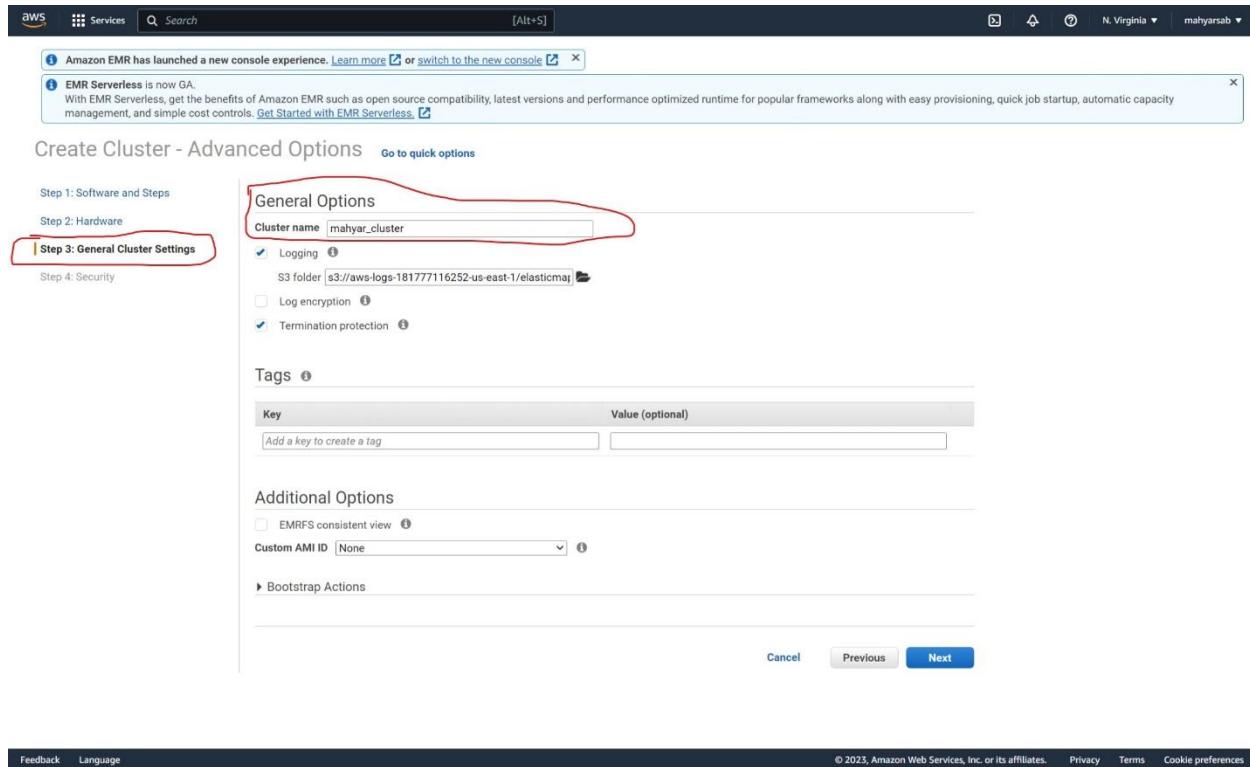
Additional Options

EMRFS consistent view

Custom AMI ID: None

Bootstrap Actions

Cancel Previous Next



- Choose your AWS EC2 key in step4: Security and then click on Create cluster:

AWS Services Search [Alt+S] N. Virginia mahyarsab

Amazon EMR has launched a new console experience. Learn more or switch to the new console.

EMR Serverless is now GA. With EMR Serverless, get the benefits of Amazon EMR such as open source compatibility, latest versions and performance optimized runtime for popular frameworks along with easy provisioning, quick job startup, automatic capacity management, and simple cost controls. Get Started with EMR Serverless.

Create Cluster - Advanced Options Go to quick options

Step 1: Software and Steps
Step 2: Hardware
Step 3: General Cluster Settings
Step 4: Security

Security Options

EC2 key pair: mahyar_aws

Cluster visible to all IAM users in account

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

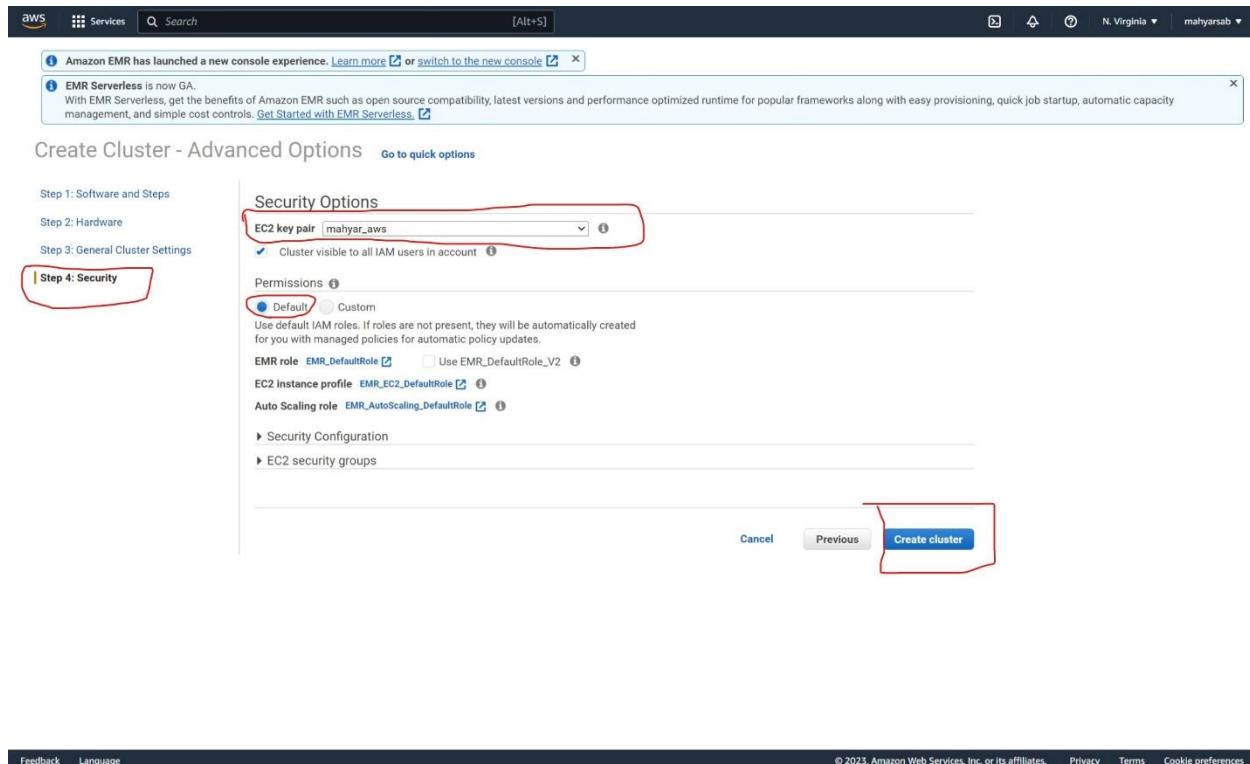
EC2 instance profile: EMR_EC2_DefaultRole

Auto Scaling role: EMR_AutoScaling_DefaultRole

Security Configuration

EC2 security groups

Cancel Previous **Create cluster**



- Then the cluster shows starting status and after a while it will change to Waiting:

Screenshot of the AWS Amazon EMR console showing the cluster configuration for "mahyar_cluster". The cluster status is "Starting". A red box highlights the "Starting" status in the cluster summary.

Configuration details:

- Release label: emr-6.1.1
- Hadoop distribution: Amazon 3.2.1
- Applications: Hive 3.1.2, Hue 4.7.1, Spark 3.0.0, Livy 0.7.0, JupyterHub 1.1.0
- Log URI: s3://aws-logs-181777116252-us-east-1/elastictmapreduce/
- EMRFS consistent view: Disabled
- Custom AMI ID: --

Network and hardware:

- Availability zone: --
- Subnet ID: subnet-00c1730dce8622b05
- Master: Provisioning 1 m5.xlarge
- Core: Provisioning 2 m5.xlarge
- Task: --
- Cluster scaling: Not enabled
- Auto-termination: Not enabled

Security and access:

- Key name: mahyar_aws
- EC2 instance profile: EMR_EC2_DefaultRole
- EMR role: EMR_DefaultRole
- Auto Scaling role: EMR_AutoScaling_DefaultRole
- Visible to all users: All Change
- Security groups for Master: sg-0a0cb250ed3efc7a3 (ElasticMapReduce-master)
- Security groups for Core & sg-07423b4d4e7300d4c (ElasticMapReduce-Task: slave)

Screenshot of the AWS Amazon EMR console showing the cluster configuration for "mahyar_cluster". The cluster status is "Waiting". A red box highlights the "Waiting" status in the cluster summary.

Configuration details:

- Release label: emr-6.1.1
- Hadoop distribution: Amazon 3.2.1
- Applications: Hive 3.1.2, Hue 4.7.1, Spark 3.0.0, Livy 0.7.0, JupyterHub 1.1.0
- Log URI: s3://aws-logs-181777116252-us-east-1/elastictmapreduce/
- EMRFS consistent view: Disabled
- Custom AMI ID: --

Network and hardware:

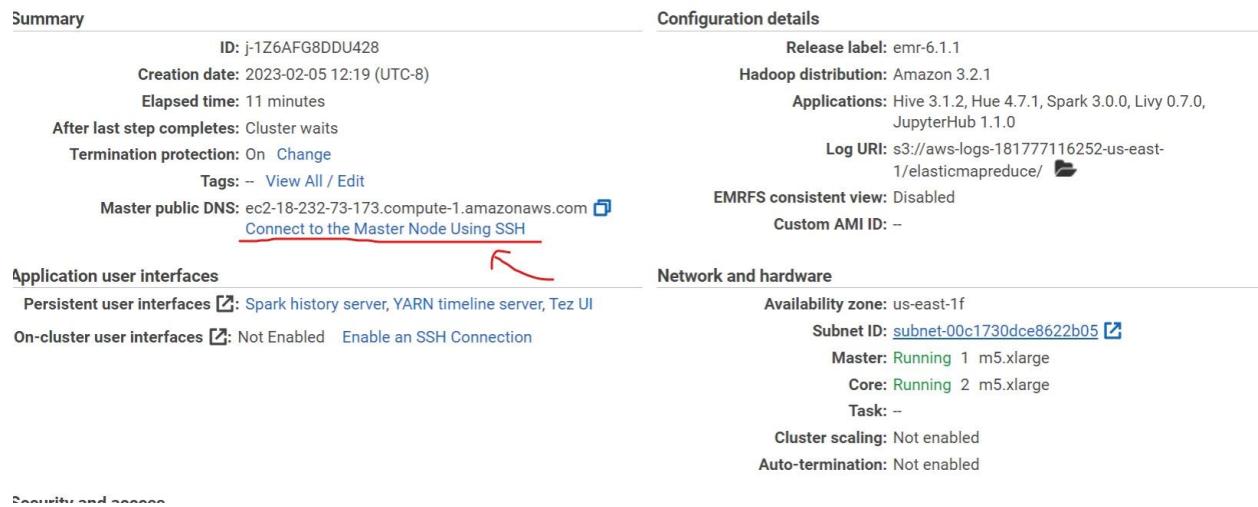
- Availability zone: us-east-1f
- Subnet ID: subnet-00c1730dce8622b05
- Master: Running 1 m5.xlarge
- Core: Running 2 m5.xlarge
- Task: --
- Cluster scaling: Not enabled
- Auto-termination: Not enabled

Security and access:

- Key name: mahyar_aws
- EC2 instance profile: EMR_EC2_DefaultRole
- EMR role: EMR_DefaultRole
- Auto Scaling role: EMR_AutoScaling_DefaultRole
- Visible to all users: All Change
- Security groups for Master: sg-0a0cb250ed3efc7a3 (ElasticMapReduce-master)
- Security groups for Core & sg-07423b4d4e7300d4c (ElasticMapReduce-Task: slave)

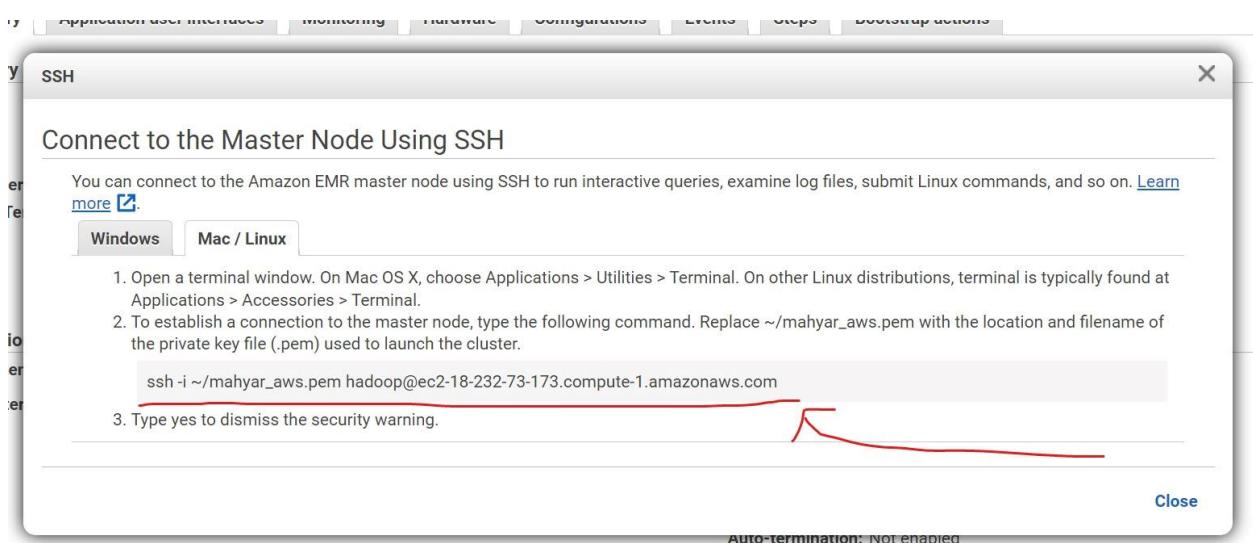
- Now the cluster is ready to use.

Question 2: In this question we should connect to the head node of the cluster using SSH:



The screenshot shows the AWS EMR Cluster Summary page. On the left, there's a 'Summary' section with cluster details like ID, creation date, and elapsed time. Below it is an 'Application user interfaces' section with 'Persistent user interfaces' (Spark history server, YARN timeline server, Tez UI) and 'On-cluster user interfaces' (Not Enabled). A red arrow points from the text 'Click on the mentioned button and copy the underlined part starting with SSH:' to the 'Connect to the Master Node Using SSH' button, which is underlined. On the right, there's a 'Configuration details' section with various configurations like Hadoop distribution, Applications, Log URI, and Network and hardware sections with subnet and master information.

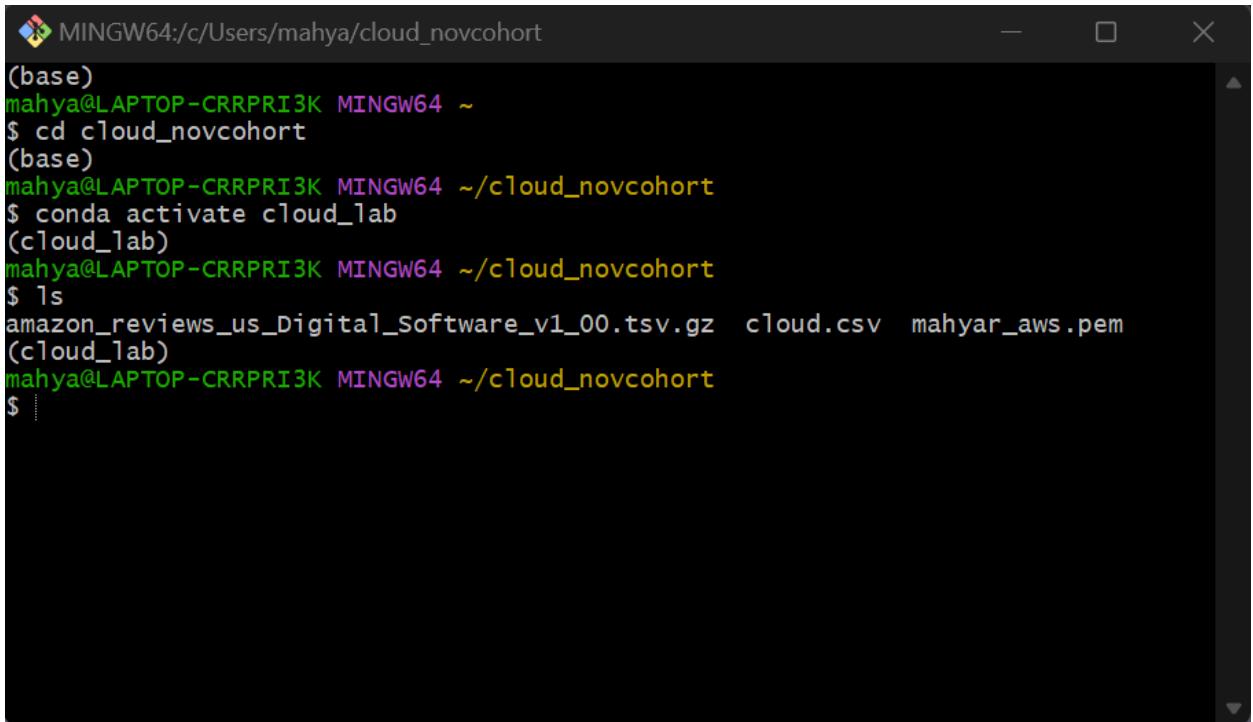
- Click on the mentioned button and copy the underlined part starting with SSH:



The screenshot shows a modal window titled 'SSH' with the sub-section 'Connect to the Master Node Using SSH'. It contains instructions for connecting via SSH, mentioning Mac OS X and Linux. A red arrow points from the text 'Click on the mentioned button and copy the underlined part starting with SSH:' to the 'ssh -i ~/mahyar_aws.pem hadoop@ec2-18-232-73-173.compute-1.amazonaws.com' command, which is underlined. At the bottom of the modal, there's a 'Close' button and a note about auto-termination.

and access

- Now open your terminal and change directory to your cloud folder which contains your pem key which mine here is mahyar_aws.pem :



```
MINGW64:/c/Users/mahya/cloud_novcohort
(base)
mahya@LAPTOP-CRRPRI3K MINGW64 ~
$ cd cloud_novcohort
(base)
mahya@LAPTOP-CRRPRI3K MINGW64 ~/cloud_novcohort
$ conda activate cloud_lab
(cloud_lab)
mahya@LAPTOP-CRRPRI3K MINGW64 ~/cloud_novcohort
$ ls
amazon_reviews_us_Digital_Software_v1_00.tsv.gz  cloud.csv  mahyar_aws.pem
(cloud_lab)
mahya@LAPTOP-CRRPRI3K MINGW64 ~/cloud_novcohort
$ ...
```

- Now paste the copied ssh code from amazon website here in your terminal and delete ~/ from the beginning of the name of your pem key and execute it. Then type yes and you will see a big EMR on your terminal and now you are connected:

```
(base) hadoop@ip-172-31-73-2:~  
mahya@LAPTOP-CRRPRI3K MINGW64 ~  
$ cd cloud_novcohort  
(base)  
mahya@LAPTOP-CRRPRI3K MINGW64 ~/cloud_novcohort  
$ conda activate cloud_lab  
(cloud_lab)  
mahya@LAPTOP-CRRPRI3K MINGW64 ~/cloud_novcohort  
$ ls  
amazon_reviews_us_Digital_Software_v1_00.tsv.gz  cloud.csv  mahyar_aws.pem  
(cloud_lab)  
mahya@LAPTOP-CRRPRI3K MINGW64 ~/cloud_novcohort  
$ ssh -i mahyar_aws.pem hadoop@ec2-18-232-73-173.compute-1.amazonaws.com  
The authenticity of host 'ec2-18-232-73-173.compute-1.amazonaws.com (18.232.73.1  
73)' can't be established.  
ED25519 key fingerprint is SHA256:FstQuTsGDAYTEgyKoGcWcIRk4IeBj2MN7Y/94UsLjiw.  
This key is not known by any other names  
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes ←  
Warning: Permanently added 'ec2-18-232-73-173.compute-1.amazonaws.com' (ED25519)  
to the list of known hosts.  
  
_ _| _ _| /  Amazon Linux 2 AMI  
_ _| \_ _| |  
  
https://aws.amazon.com/amazon-linux-2/  
85 package(s) needed for security, out of 126 available  
Run "sudo yum update" to apply all updates.  
  
EEEEEEEEEEEEEEEEE MMMMMMMM      MMMMMMM RRRRRRRRRRRRRR  
E:::::::EEEEEEEEE::: E M:::::M M:::::M R:::::RRRRRR:::::R  
EE:::::EEEEEEEEE::: E M:::::M M:::::M R:::::RRRRRR:::::R  
   E:::E   EEEEE M:::::M M:::::M R:::::R R:::::R  
   E:::E   M:::::M M:::::M M:::::M R:::::R R:::::R  
   E:::::EEEEEEEEE  M:::::M M:::::M M:::::M R:::::RRRRRR:::::R  
   E:::::EEEEEEEEE  M:::::M M:::::M M:::::M R:::::RRRRRR:::::R  
   E:::::EEEEEEEEE  M:::::M M:::::M M:::::M R:::::RRRRRR:::::R  
   E:::E   EEEEE M:::::M M:::::M R:::::R R:::::R  
   E:::E   M:::::M M:::::M M:::::M R:::::R R:::::R  
EE:::::EEEEEEEEE::: E M:::::M M:::::M R:::::R R:::::R  
E:::::EEEEEEEEE::: E M:::::M M:::::M R:::::R R:::::R  
EEEEEEEEEEEEEEEEE MMMMMMM      MMMMMMR RRRRRRRR RRRRRR  
  
[hadoop@ip-172-31-73-2 ~]$
```

- Now let's execute some Hadoop commands here:

```

hadoop@ip-172-31-73-2:~ 
amazon_reviews_us_Digital_Software_v1_00.tsv.gz  cloud.csv  mahyar_aws.pem
(ccloud_lab)
mahya@LAPTOP-CRRPRI3K MINGW64 ~/cloud_novcohort
$ ssh -i mahyar_aws.pem hadoop@ec2-18-232-73-173.compute-1.amazonaws.com
The authenticity of host 'ec2-18-232-73-173.compute-1.amazonaws.com (18.232.73.1
73)' can't be established.
ED25519 key fingerprint is SHA256:FstQuTsGDAYTEgyKoGcWcIRk4IeBj2MN7Y/94UsLjiw.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'ec2-18-232-73-173.compute-1.amazonaws.com' (ED25519)
to the list of known hosts.

 _|_(_|-_)_
 _\|_|_|
Amazon Linux 2 AMI

https://aws.amazon.com/amazon-linux-2/
85 package(s) needed for security, out of 126 available
Run "sudo yum update" to apply all updates.

EEEEEEEEEEEEEEEEEE MMMMMMM          MMMMMMM RRRRRRRRRRRRRR
E:::::::::::E M:::::::M          M:::::::M R:::::::::::::R
EE:::::E:EEEEEEEEE:::E M:::::::M          M:::::::M R:::::RRRRR::::::R
 E:::::E     EEEE E M:::::::M          M:::::::M RR:::::R    R:::::R
 E:::::E     M:::::::M          M:::::M:::M R:::::R    R:::::R
 E:::::E:EEEEEEEEE E M::::::M          M:::::M M:::::M R:::::RRRRR::::::R
 E:::::::::::E M::::::M          M:::::M:::M M:::::M R:::::RRRRRR::::::R
 E:::::E:EEEEEEEEE M::::::M          M:::::M M:::::M R:::::RRRRR::::::R
 E:::::E     EEEE E M::::::M          M:::::M M:::::M R:::::R    R:::::R
 E:::::E     M::::::M          MMMM M::::::M R:::::R    R:::::R
EE:::::E:EEEEEEEEE:::E M::::::M          M::::::M R:::::R    R:::::R
E:::::::::::E:::::E M::::::M          M::::::M RR:::::R    R:::::R
EEEEEEEEEEEEEEEEEE MMMMMMM          MMMMMMM RRRRRRRR RRRRRR

[hadoop@ip-172-31-73-2 ~]$ hadoop fs -ls
[hadoop@ip-172-31-73-2 ~]$ hadoop fs -ls /user/
Found 8 items
drwxrwxrwx  - hadoop hadoop      0 2023-02-05 20:27 /user/hadoop
drwxr-xr-x  - mapred mapred      0 2023-02-05 20:27 /user/history
drwxrwxrwx  - hdfs hadoop      0 2023-02-05 20:28 /user/hive
drwxrwxrwx  - hue hue          0 2023-02-05 20:27 /user/hue
drwxrwxrwx  - livy livy         0 2023-02-05 20:27 /user/livy
drwxrwxrwx  - oozie oozie        0 2023-02-05 20:27 /user/oozie
drwxrwxrwx  - root hadoop       0 2023-02-05 20:27 /user/root
drwxrwxrwx  - spark spark        0 2023-02-05 20:27 /user/spark
[hadoop@ip-172-31-73-2 ~]$ 

```

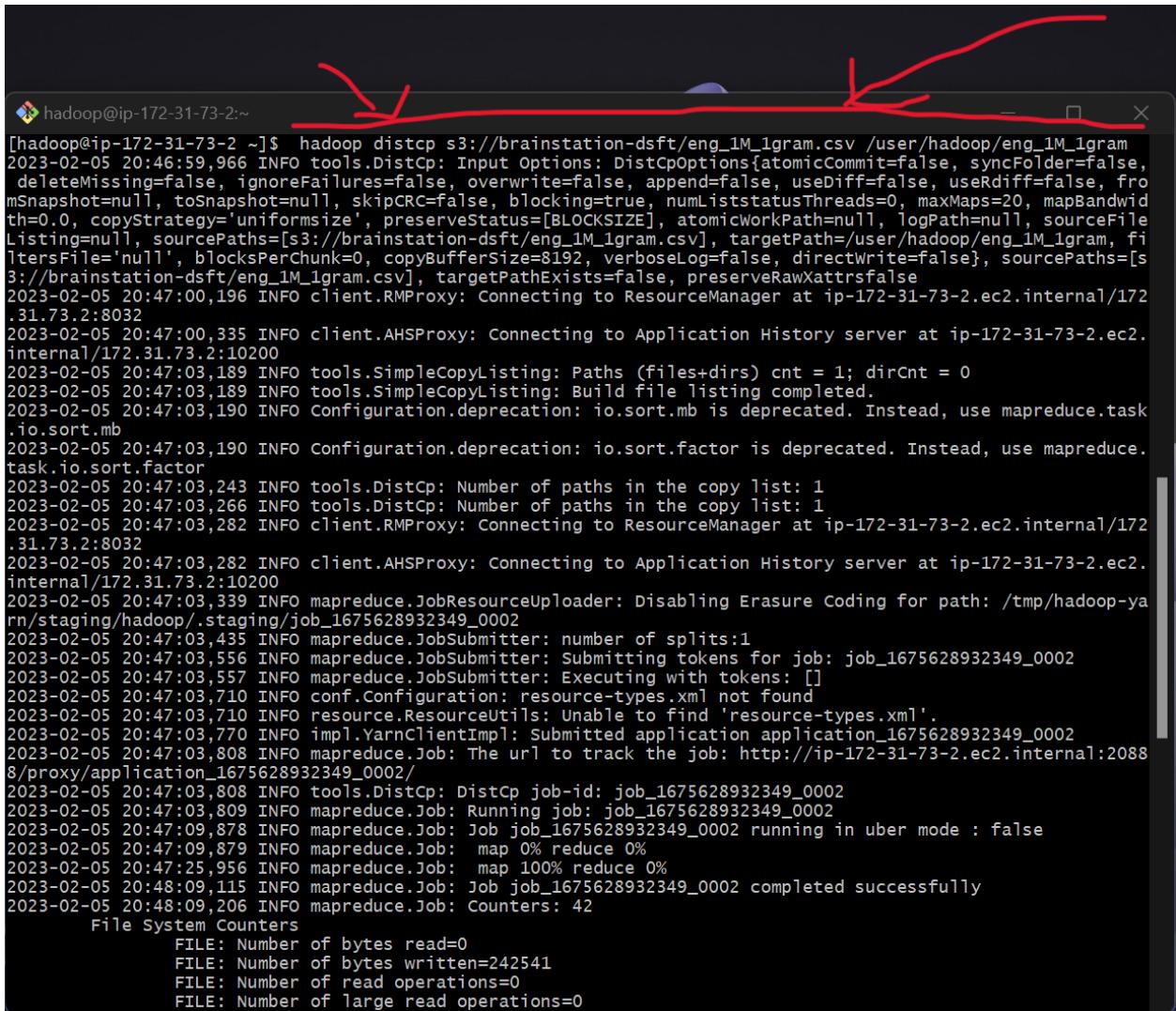
QUESTION 3 : We will copy the data folder from the S3 bucket *directly* into a directory on the Hadoop File System (HDFS) named `/user/hadoop/eng_1M_1gram` using the underlined code below:

- In this question I used ‘`hadoop distcp`’ instead of `cp` for the reason below:

hadoop distcp is a Hadoop command-line tool that can be used to copy large amounts of data between Hadoop clusters, or between a Hadoop cluster and an external storage system such as a local file system or an Amazon S3 bucket.

hadoop distcp uses MapReduce to perform the data transfer, which allows it to scale to handle very large data sets and to recover from failures. The tool also supports various advanced features, such as preserving file attributes, copying only new or updated files, and throttling the data transfer rate.

hadoop distcp is often used for large-scale data migration, data backup and recovery, and data sharing between Hadoop clusters. It is a powerful tool for managing big data, but it can also be complex to use, so it is important to understand the various options and configuration settings available when using **hadoop distcp**.



```
[hadoop@ip-172-31-73-2 ~]$ hadoop distcp s3://brainstation-dsft/eng_1M_1gram.csv /user/hadoop/eng_1M_1gram
2023-02-05 20:46:59,966 INFO tools.DistCp: Input Options: DistCpOptions{atomicCommit=false, syncFolder=false, deleteMissing=false, ignoreFailures=false, overwrite=false, append=false, useDiff=false, useRdiff=false, fromSnapshot=null, toSnapshot=null, skipCRC=false, blocking=true, numListStatusThreads=0, maxMaps=20, mapBandwidth=0.0, copyStrategy='uniformsize', preserveStatus=[BLOCKSIZE], atomicWorkPath=null, logPath=null, sourceFileListing=null, sourcePaths=[s3://brainstation-dsft/eng_1M_1gram.csv], targetPath=/user/hadoop/eng_1M_1gram, filetersFile=null, blocksPerChunk=0, copyBufferSize=8192, verboseLog=false, directWrite=false}, sourcePaths=[s3://brainstation-dsft/eng_1M_1gram.csv], targetPathExists=false, preserveRawXattrs=false
2023-02-05 20:47:00,196 INFO client.RMProxy: Connecting to ResourceManager at ip-172-31-73-2.ec2.internal/172.31.73.2:8032
2023-02-05 20:47:00,335 INFO client.AHSProxy: Connecting to Application History server at ip-172-31-73-2.ec2.internal/172.31.73.2:10200
2023-02-05 20:47:03,189 INFO tools.SimpleCopyListing: Paths (files+dirs) cnt = 1; dirCnt = 0
2023-02-05 20:47:03,189 INFO tools.SimpleCopyListing: Build file listing completed.
2023-02-05 20:47:03,190 INFO Configuration.deprecation: io.sort.mb is deprecated. Instead, use mapreduce.task.io.sort.mb
2023-02-05 20:47:03,190 INFO Configuration.deprecation: io.sort.factor is deprecated. Instead, use mapreduce.task.io.sort.factor
2023-02-05 20:47:03,243 INFO tools.DistCp: Number of paths in the copy list: 1
2023-02-05 20:47:03,266 INFO tools.DistCp: Number of paths in the copy list: 1
2023-02-05 20:47:03,282 INFO client.RMProxy: Connecting to ResourceManager at ip-172-31-73-2.ec2.internal/172.31.73.2:8032
2023-02-05 20:47:03,282 INFO client.AHSProxy: Connecting to Application History server at ip-172-31-73-2.ec2.internal/172.31.73.2:10200
2023-02-05 20:47:03,339 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/hadoop/.staging/job_1675628932349_0002
2023-02-05 20:47:03,435 INFO mapreduce.JobSubmitter: number of splits:1
2023-02-05 20:47:03,556 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1675628932349_0002
2023-02-05 20:47:03,557 INFO mapreduce.JobSubmitter: Executing with tokens: []
2023-02-05 20:47:03,710 INFO conf.Configuration: resource-types.xml not found
2023-02-05 20:47:03,710 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2023-02-05 20:47:03,770 INFO impl.YarnClientImpl: Submitted application application_1675628932349_0002
2023-02-05 20:47:03,808 INFO mapreduce.Job: The url to track the job: http://ip-172-31-73-2.ec2.internal:20888/proxy/application_1675628932349_0002/
2023-02-05 20:47:03,808 INFO tools.DistCp: DistCp job-id: job_1675628932349_0002
2023-02-05 20:47:03,809 INFO mapreduce.Job: Running job: job_1675628932349_0002
2023-02-05 20:47:09,878 INFO mapreduce.Job: Job job_1675628932349_0002 running in uber mode : false
2023-02-05 20:47:09,879 INFO mapreduce.Job: map 0% reduce 0%
2023-02-05 20:47:25,956 INFO mapreduce.Job: map 100% reduce 0%
2023-02-05 20:48:09,115 INFO mapreduce.Job: Job job_1675628932349_0002 completed successfully
2023-02-05 20:48:09,206 INFO mapreduce.Job: Counters: 42
    File System Counters
        FILE: Number of bytes read=0
        FILE: Number of bytes written=242541
        FILE: Number of read operations=0
        FILE: Number of large read operations=0
```

- After that we will see that it will copy and we see the executions below:

```
hadoop@ip-172-31-73-2:~  
2023-02-05 20:47:09,878 INFO mapreduce.Job: Job job_1675628932349_0002 running in uber mode : false  
2023-02-05 20:47:09,879 INFO mapreduce.Job: map 0% reduce 0%  
2023-02-05 20:47:25,956 INFO mapreduce.Job: map 100% reduce 0%  
2023-02-05 20:48:09,115 INFO mapreduce.Job: Job job_1675628932349_0002 completed successfully  
2023-02-05 20:48:09,206 INFO mapreduce.Job: Counters: 42  
File System Counters  
    FILE: Number of bytes read=0  
    FILE: Number of bytes written=242541  
    FILE: Number of read operations=0  
    FILE: Number of large read operations=0  
    FILE: Number of write operations=0  
    HDFS: Number of bytes read=376  
    HDFS: Number of bytes written=5292105197  
    HDFS: Number of read operations=14  
    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)=5484672  
    Total time spent by all reduces in occupied slots (ms)=0  
    Total time spent by all map tasks (ms)=57132  
    Total vcore-milliseconds taken by all map tasks=57132  
    Total megabyte-milliseconds taken by all map tasks=175509504  
Map-Reduce Framework  
    Map input records=1  
    Map output records=0  
    Input split bytes=137  
    Spilled Records=0  
    Failed Shuffles=0  
    Merged Map outputs=0  
    GC time elapsed (ms)=295  
    CPU time spent (ms)=60600  
    Physical memory (bytes) snapshot=1016680448  
    Virtual memory (bytes) snapshot=4435304448  
    Total committed heap usage (bytes)=720896000  
    Peak Map Physical memory (bytes)=1016680448  
    Peak Map Virtual memory (bytes)=4435304448  
File Input Format Counters  
    Bytes Read=239
```

Question 4 : Now by using pyspark, we have to read the data that we copied into HDFS in Step 3. For this I will Launch a jupyter notebook from amazon aws like below:

EMR on EC2

- Clusters
- Notebooks**
- Git repositories
- Security configurations
- Block public access
- VPC subnets
- Events

EMR on EKS

Virtual clusters

Help

What's new

Cluster: mahyar_cluster Waiting Cluster ready to run steps.

Summary Application user interfaces Monitoring Hardware Configurations Events Steps Bootstrap actions

Summary		Configuration details	
ID: j-1Z6AFG8DDU428		Release label: emr-6.1.1	
Creation date: 2023-02-05 12:19 (UTC-8)		Hadoop distribution: Amazon 3.2.1	
Elapsed time: 26 minutes		Applications: Hive 3.1.2, Hue 4.7.1, Spark 3.0.0, Livy 0.7.0, JupyterHub 1.1.0	
After last step completes: Cluster waits		Log URI: s3://aws-logs-181777116252-us-east-1/elasticmapreduce/	
Termination protection: On Change		EMRFS consistent view: Disabled	
Tags: -- View All / Edit		Custom AMI ID: --	
Master public DNS: ec2-18-232-73-173.compute-1.amazonaws.com	Connect to the Master Node Using SSH		

Application user interfaces

Persistent user interfaces: Spark history server, YARN timeline server, Tez UI

On-cluster user interfaces: Not Enabled Enable an SSH Connection

Network and hardware

Availability zone: us-east-1f

Subnet ID: subnet-00c1730dce8622b05

Master: Running 1 m5.xlarge

Core: Running 2 m5.xlarge

Task: -

Cluster scaling: Not enabled

Auto-termination: Not enabled

aws Services Search [Alt+S]

Amazon EMR

EMR Studio

EMR Serverless New

EMR on EC2

Clusters

Notebooks

Git repositories

Security configurations

Block public access

VPC subnets

Events

EMR on EKS

Virtual clusters

Help

What's new

Amazon EMR has launched a new console experience. [Learn more](#) or [switch to the new console](#)

EMR Notebooks will be available as EMR Studio Workspaces in the new console by Feb 08, 2023. You'll still be able to use your existing notebooks here, but on March 10, 2023, we'll turn off the Create notebook button. The Create Workspace button in the new console will replace this functionality. To access or create Workspaces, EMR Notebooks users will need additional IAM role permissions. [Learn more](#)

EMR Serverless is now GA. With EMR Serverless, get the benefits of Amazon EMR such as open source compatibility, latest versions and performance optimized runtime for popular frameworks along with easy provisioning, quick job startup, automatic capacity management, and simple cost controls. [Get Started with EMR Serverless](#)

Notebooks

Use EMR notebooks based on Jupyter to analyze data interactively with live code, narrative text, visualizations, and more. Create and attach notebooks to Amazon EMR clusters running Hadoop, Spark, and Livy. Notebooks run free of charge and are saved in Amazon S3 independently of clusters. Standard billing for clusters and Amazon S3 apply. [Learn more](#)

Create notebook View details Open in JupyterLab Open in Jupyter Start Stop Delete

Filter: All notebooks Filter notebooks... 0 notebooks (all loaded) C

Name	Status	Cluster	Creation time (UTC-8)	Last modified
------	--------	---------	-----------------------	---------------

Feedback Language © 2023, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

- Here we should give the notebook a name then choose our cluster and then put AWS service role on create default role and the click on create notebook:

aws Services Search [Alt+S]

Amazon EMR

EMR Studio

EMR Serverless New

EMR on EC2

Clusters

Notebooks

- Git repositories
- Security configurations
- Block public access
- VPC subnets
- Events

EMR on EKS

Virtual clusters

Help

What's new

Create notebook

Name and configure your notebook

Name your notebook, choose a cluster or create one, and customize configuration options if desired. [Learn more](#)

Notebook name* mahyar_sabouniaghdam_spark_on_EMR_notebook_fo ↗

Names may only contain alphanumeric characters, hyphens (-), or underscores (_).

Description

256 characters max.

Cluster* Choose an existing cluster **Choose** ↗

Create a cluster ↗

Security groups Use default security groups ↗

Choose security groups

AWS service role*

Notebook location* Choose an S3 location where files for this notebook are saved.

Use a location that EMR creates ↗
s3://aws-emr-resources-181777116252-us-east-1/notebooks/

Choose an existing S3 location in us-east-1

▶ **Git repository** Link to a Git repository

▶ **Tags** ↗

Feedback Language © 2023, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Choose a cluster

The listed clusters meet notebook requirements. They are in an EC2-VPC, running EMR 5.18.0 or later, and have Hadoop, Spark, and Livy installed. [Learn more](#)

Filter: 1 cluster (all loaded) **C**

Name ↗	ID	Status
mahyar_cluster	j-1Z6AFG8DDU428	Waiting Cluster ready

Choose security groups

Cancel **Choose cluster**

Screenshot of the AWS Amazon EMR Create notebook page.

Notebook name: mahyar_sabouniaghdam_spark_on_EMR_notebook_for_step4

Description: (empty)

Cluster: Choose an existing cluster (selected) → mahyar_cluster_j-1Z6AFG8DDU428

Security groups: Use default security groups (selected)

AWS service role: Create default role

Notebook location: Use a location that EMR creates (selected) → s3://aws-emr-resources-181777116252-us-east-1/notebooks/

Git repository: Link to a Git repository

Tags: (empty)

Annotations: A red arrow points from the "Notebook name" input field to the "mahyar_sabouniaghdam_spark_on_EMR_notebook_for_step4" text. Another red circle highlights the "Choose an existing cluster" dropdown, and a red arrow points from it to the selected cluster "mahyar_cluster_j-1Z6AFG8DDU428". A third red circle highlights the "Create default role" button, and a red arrow points from it to the "AWS service role" dropdown.

- First our notebook status is starting and after a while it will change to Ready.

Screenshot of the AWS Amazon EMR Notebook details page.

Notebook: mahyar_sabouniaghdam_spark_on_EMR_notebook_for_step4 Starting Starting workspace(notebook). Cluster j-1Z6AFG8DDU428.

Actions: Open in JupyterLab, Open in Jupyter, Stop, Delete

Notebook:

- Notebook ID:** e-4EIS7JSF9QWC5ALKQLD53SFFZ
- Description:** --
- Last modified:** 9 seconds ago
- Last modified by:** ...root
- Created on:** 2023-02-05 13:02 (UTC)
- Created by:** ...root
- Service IAM role:** EMR_Notebooks_DefaultRole
- Notebook tags:** creatorUserid = 181777116252, View All / Edit
- Notebook location:** s3://aws-emr-resources-181777116252-us-east-1/notebooks/

Cluster:

- Cluster:** mahyar_cluster
- Cluster ID:** j-1Z6AFG8DDU428
- Cluster status:** Waiting Cluster ready to run steps.
- Cluster tags:** (empty)
- Step logs:** s3://aws-logs-181777116252-us-east-1/elasticmapreduce/

Git repositories:

The repository can be linked to a notebook once the notebook is ready. Make sure your cluster, service role and security groups have the required settings. Learn more

Repository name: URL Branch Link status Failure reason

Annotations: A red arrow points from the "Starting" status text to the "Waiting Cluster ready to run steps." status text. A red circle highlights the "Cluster ID" field "j-1Z6AFG8DDU428", and a red arrow points from it to the "Cluster status" text.

- Click on open in jupyter and we will continue our work in jupyter notebook:

The screenshot shows the AWS EMR console with a notebook named "mahyar_sabouniaghdam_spark_on_EMR_notebook_for_step4". The "Open in Jupyter" button is highlighted with a red circle. Red arrows point from the "Notebook ID" section (containing "e-4EIS7J5F9QWC5ALKQLD53SFFZ") and the "Cluster" section (containing "Cluster: mahyar_cluster" and "Cluster status: Waiting Cluster ready to run steps.") to the Jupyter interface below.

Notebook: mahyar_sabouniaghdam_spark_on_EMR_notebook_for_step4

Notebook ID: e-4EIS7J5F9QWC5ALKQLD53SFFZ

Description: --

Last modified: 3 seconds ago

Last modified by: ...root

Created on: 2023-02-05 13:02 (UTC-8)

Created by: ...root

Service IAM role: EMR_Notebooks_DefaultRole

Security groups for master instance: sg-025750330e2b5543f

Security groups for notebook instance: sg-0472aa06957c4e6ec

Notebook tags: creatorUserId = 181777116252 View All / Edit

Notebook location: s3://aws-emr-resources-181777116252-us-east-1/notebooks/

Cluster

Cluster: mahyar_cluster

Cluster Id: i-126AFG8DDU42B

Cluster status: Waiting Cluster ready to run steps.

Cluster tags: --

Step logs: s3://aws-logs-181777116252-us-east-1/elasticmapreduce/

Git repositories

The repository can be linked to a notebook once the notebook is ready. Make sure your cluster, service role and security groups have the required settings. Learn more

Feedback Language

© 2023, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

jupyter

Files Running Clusters

Select items to perform actions on them.

Upload New

Name Last Modified File size

mahyar_sabouniaghdam_spark_on_EMR_notebook_for_step4.ipynb 2 minutes ago 72 B

Quit

- Our kernel should be PySpark:

The screenshot shows the Jupyter notebook interface for the same notebook. The "PySpark" kernel is highlighted with a red circle. A red arrow points from the "Notebook ID" section in the AWS interface to the "In []:" input cell in the Jupyter interface.

jupyter mahyar_sabouniaghdam_spark_on_EMR_notebook_for_step4 Last Checkpoint: 4 minutes ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help

In []: ↵

PySpark

Question 4: the whole complete answer for step 4 is on the notebook attached. I will share the pictures here too:

jupyter mahyar_sabouniaghdam_spark_on_EMR_notebook_for_step4 Last Checkpoint: a few seconds ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted | PySpark O

Mahyar Sabouniaghdam BrainStation®

Big Data Wrangling With Google Books Ngrams

In this assignment, we will apply the skills we've learned in the Big Data Fundamentals unit to load, filter, and visualize a large real-world dataset in a cloud-based distributed computing environment using Hadoop, Spark, Hive, and the S3 filesystem. We will prepare a professional report to summarize the findings and be sure to include an appendix with screenshots of the steps completed for Questions 1 and 2.

The Google Ngrams dataset was created by Google's research team by analyzing all of the content in Google Books - these digitized texts represent approximately 4% of all books ever printed, and span a time period from the 1800s into the 2000s.

The dataset is hosted in a public S3 bucket as part of the Amazon S3 Open Data Registry. For this assignment, we have converted the data to CSV and hosted it on a public S3 bucket which may be accessed here: s3://brainstation-dsft/eng_1M_1gram.csv

For this deliverable, we will produce a report, as well as a jupyter notebook, which will follow a Big Data analysis workflow. As part of this workflow we will filter and reduce data down to a manageable size, and then do some analysis locally on our machine after extracting data from the Cloud and processing it using Big Data tools. The workflow and steps in the process are illustrated below:

Copyright © BrainStation

Big Data Analysis Workflow

- 1 Pull data from (public) S3 bucket into HDFS
- 2 Analyze and filter data using Spark
- 3 Connect filtered data to head node using getmerge
- 4 Write data to (personal) S3 bucket
- 5 Read data from S3 bucket for analysis on local machine

1. Up until now we spun up a new EMR cluster using the AWS Console and we included Hadoop, Spark, Hive, Jupyterhub, and Livy for our cluster with the release version of EMR 6.1.1.
2. Then we connected to the head node of the cluster using SSH.
3. Then we copied the data folder from s3://brainstation-dsft/eng_1M_1gram.csv into a directory on the Hadoop File System (HDFS) named /user/hadoop/eng_1M_1gram.
4. Now we are in step 4:

Step 4

In this step by using pyspark, we will read the data we copied into HDFS in Step 3 with a PySpark notebook that we created. Since we have created a pyspark DataFrame and now we are in it, we will complete the following steps below:

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

- First we have to initialize the SparkContext and SparkSession:

```
In [1]: M spark
Starting Spark application
```

jupyter mahyar_sabouniaghdam_spark_on_EMR_notebook_for_step4 Last Checkpoint: a minute ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Trusted | PySpark O

Step 4

In this step by using pyspark, we will read the data we copied into HDFS in Step 3 with a PySpark notebook that we created. Since we have created a pyspark DataFrame and now we are in it, we will complete the following steps below:

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

- First we have to initialize the SparkContext and SparkSession:

```
In [1]: # spark
```

```
Starting Spark application
ID          YARN Application ID  Kind  State  Spark UI  Driver log  Current session?
0  application_1675628932349_0003  pyspark  idle   Link      Link    ✓
```

```
SparkSession available as 'spark'.
```

```
<pyspark.sql.session.SparkSession object at 0x7f09c79955d0>
```

- Now we will read the data:

```
In [3]: # read the data
data = spark.read.csv("/user/hadoop/eng_1M_1gram", header=True)
```

Spark Job Progress

Job [1]: csv at NativeMethodAccessorImpl.java:0

Progress for csv at NativeMethodAccessorImpl.java:0	Job Progress: 1/1 Tasks Complete			
Stage [ID]: name at [source]:[line]	Status	Task Progress	Elapsed Time (seconds)	Failed Task Logs
Stage [1]: csv at NativeMet..java:0	COMPLETE	1/1	7.532	

jupyter mahyar_sabouniaghdam_spark_on_EMR_notebook_for_step4 Last Checkpoint: a minute ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Trusted | PySpark O

4.a

- Now we will describe the dataset (examples include size, shape, schema) in pyspark:

```
In [4]: # Get the shape of the dataset
print("Number of rows:", data.count())
print("Number of columns:", len(data.columns))
```

Spark Job Progress

```
Number of rows: 261823225
Number of columns: 5
```

- Our dataset has 261823225 rows and 5 columns. Let's see the columns our dataset:

```
In [19]: # columns
data.columns
```

```
['token', 'year', 'frequency', 'pages', 'books']
```

- Now let's see the schema of our dataset:

```
In [5]: # Get the schema of the dataset
data.printSchema()
```

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

- Let's see the first 20 rows of the dataset:

jupyter mahyar_sabouniaghdam_spark_on_EMR_notebook_for_step4 Last Checkpoint: a minute ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted PySpark nbdiff

```
data.printSchema()
```

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

- Let's see the first 20 rows of the dataset:

```
In [20]: # First 20 rows
data.show(20)
```

	token	year	frequency	pages	books
1	inGermany	1927	2	2	2
2	inGermany	1929	1	1	1
3	inGermany	1930	1	1	1
4	inGermany	1933	1	1	1
5	inGermany	1934	1	1	1
6	inGermany	1935	1	1	1
7	inGermany	1938	5	5	5
8	inGermany	1939	1	1	1
9	inGermany	1940	1	1	1
10	inGermany	1942	2	2	2
11	inGermany	1941	2	2	2
12	inGermany	1945	2	2	2
13	inGermany	1947	3	3	2
14	inGermany	1948	1	1	1
15	inGermany	1949	1	1	1
16	inGermany	1952	1	1	1
17	inGermany	1956	1	1	1
18	inGermany	1957	2	2	2
19	inGermany	1959	1	1	1
20	inGermany	1960	3	3	3

only showing top 20 rows

4.b

jupyter mahyar_sabouniaghdam_spark_on_EMR_notebook_for_step4 Last Checkpoint: a minute ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted PySpark nbdiff

4.b

- Now we will create a new DataFrame from a query using Spark SQL, filtering to include only the rows where the token is "data" and then we will describe the new dataset:

- Using SQL requires the `CreateOrReplaceTempView` function, which registers the data as a view in the Spark session. We can then query against that view with SQL, with the view name being the table name:

```
In [21]: # Create a view
data.createOrReplaceTempView("ngrams")
```

```
In [22]: # Create a new DataFrame from a query using Spark SQL
filtered_data = spark.sql("SELECT * FROM ngrams WHERE token = 'data'")
```

- Now we created a new DataFrame (`filtered_data`) using Spark SQL and filtered it to include only the rows where the token is "data". Now let's describe the new dataset:

```
In [23]: # Get the shape of the filtered dataset
print("Number of rows:", filtered_data.count())
print("Number of columns:", len(filtered_data.columns))
```

```
Number of rows: 316
Number of columns: 5
```

- Our dataset has 316 rows and 5 columns. Let's see the columns our dataset:

```
In [27]: # columns
filtered_data.columns
```

```
['token', 'year', 'frequency', 'pages', 'books']
```

```
In [28]: # Get the schema of the dataset
```

jupyter mahyar_sabouniaghdam_spark_on_EMR_notebook_for_step4 Last Checkpoint: 2 minutes ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted PySpark nbdiff

• Our dataset has 316 rows and 5 columns. Let's see the columns our dataset.

```
In [27]: # # columns
filtered_data.columns
```

```
[token, 'year', 'frequency', 'pages', 'books']
```

In [28]: # Get the schema of the dataset

```
filtered_data.printSchema()
```

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

• We saw the columns and schema of the filtered dataset again. Let's see the first 20 rows of the filtered dataset:

```
In [29]: # First 20 rows of the filtered dataset
filtered_data.show(20)
```

	token	year	frequency	pages	books
1	data 1584	16	14	1	
2	data 1614	3	2	1	
3	data 1627	1	1	1	
4	data 1631	22	18	1	
5	data 1637	1	1	1	
6	data 1638	2	2	1	
7	data 1640	1	1	1	
8	data 1642	1	1	1	
9	data 1644	4	4	1	
10	data 1647	1	1	1	
11	data 1651	1	1	1	
12	data 1674	1	1	1	
13	data 1698	1	1	1	
14	data 1693	1	1	1	
15	data 1697	1	1	1	
16	data 1699	1	1	1	
17	data 1700	1	1	1	

jupyter mahyar_sabouniaghdam_spark_on_EMR_notebook_for_step4 Last Checkpoint: 2 minutes ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted PySpark nbdiff

	token	year	frequency	pages	books
1	data 1637	1	1	1	
2	data 1638	2	2	1	
3	data 1640	1	1	1	
4	data 1642	1	1	1	
5	data 1644	4	4	1	
6	data 1647	1	1	1	
7	data 1651	1	1	1	
8	data 1674	1	1	1	
9	data 1698	1	1	1	
10	data 1693	1	1	1	
11	data 1697	1	1	1	
12	data 1699	1	1	1	
13	data 1700	1	1	1	

only showing top 20 rows

• Number of rows of our filtered dataset:

```
In [30]: # Number of rows of filtered data
filtered_data.count()
```

```
316
```

• we see that the number of rows of our dataset changed from 261823225 to 316 rows.

4.c

Now let's write the filtered data back to a directory in HDFS from Spark using df.write.csv(). We have to be sure to pass the header=True parameter and examine the contents of what we've written:

```
In [31]: # Write the filtered data back to HDFS
filtered_data.write.csv("/user/hadoop/#filtered_data", header=True)
```

This will write the filtered DataFrame to a directory in HDFS named /user/hadoop/filtered_data with header information.

jupyter mahyar_sabouniaghdam_spark_on_EMR_notebook_for_step4 Last Checkpoint: 3 minutes ago (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Trusted PySpark O

4.c

- Now let's write the filtered data back to a directory in HDFS from Spark using df.write.csv(). We have to be sure to pass the header=True parameter and examine the contents of what we've written:

```
In [31]: # Write the filtered data back to HDFS  
filtered_data.write.csv("/user/hadoop/filtered_data", header=True)
```

- This will write the DataFrame to a directory in HDFS named /user/hadoop/filtered_data with header information.
- To examine the contents of what you've written, you can use the following command in Git Bash:
`hadoop fs -cat /user/hadoop/filtered_data/*`
- We can also use the following command to view the directory structure in HDFS:
`hadoop fs -ls /user/hadoop/filtered_data`
- Our work here is finished so we can stop the spark session:

```
In [ ]: spark.stop()
```

- The step 4 is finished. Now we have to proceed to step 5 to collect the data using merge and then step 6 to create a jupyter notebook on our local machine.

- Now that we wrote the `filtered_data` back to HDFS, let's run some codes on command line to check whether they exist or not:

```

hadoop@ip-172-31-73-2:~$ hadoop fs -ls /user/hadoop/filtered_data
Found 3 items
-rw-r--r-- 1 livy hadoop          0 2023-02-05 22:04 /user/hadoop/filtered_data/_SUCCESS ✓
-rw-r--r-- 1 livy hadoop      33 2023-02-05 22:03 /user/hadoop/filtered_data/part-00000-c5ac7f4e-c869-4f
f9-bd50-d96747ab572a-c000.csv
-rw-r--r-- 1 livy hadoop    7305 2023-02-05 22:04 /user/hadoop/filtered_data/part-00023-c5ac7f4e-c869-4f
f9-bd50-d96747ab572a-c000.csv
[hadoop@ip-172-31-73-2 ~]$ hadoop fs -ls /user/hadoop/
Found 2 items
-rw-r--r-- 1 hadoop hadoop 5292105197 2023-02-05 20:48 /user/hadoop/eng_1M_lgram ✓
drwxr-xr-x  - livy hadoop          0 2023-02-05 22:04 /user/hadoop/filtered_data
[hadoop@ip-172-31-73-2 ~]$ ✓

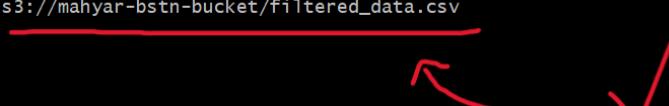
```

- We can see that our filtered_data.csv file is on Hadoop.

Question 5 : In this step we collect the contents of the directory into a single file on the local drive of the head node using getmerge and then we moved filtered file into S3 bucket of my account which is mahyar-bstn-bucket:

- hadoop fs -getmerge is a Hadoop command-line tool that merges multiple files in HDFS (Hadoop Distributed File System) into a single file on the local file system. The merged file will contain the contents of all the input files concatenated together, with a newline character separating each file's contents.

```
hadoop@ip-172-31-73-2:~  
data,1993,223635,111014,4303  
data,1994,236845,110670,4180  
data,1995,216659,108378,4315  
data,1996,213991,108023,4284  
data,1997,223109,110021,4264  
data,1998,208230,104315,4231  
data,1999,230573,108982,4179  
data,2000,223813,107134,4040  
data,2001,247209,115258,3983  
data,2002,241848,109219,3825  
data,2003,244250,105400,3684  
data,2004,201841,93539,3483  
data,2005,197467,88901,3392  
data,2006,203669,92960,3449  
data,2007,168338,78986,3246  
data,2008,105331,47811,2358  
[hadoop@ip-172-31-73-2 ~]$ hadoop fs -ls /user/hadoop/filtered_data  
Found 3 items  
-rw-r--r-- 1 livy hadoop 0 2023-02-05 22:04 /user/hadoop/filtered_data/_SUCCESS  
-rw-r--r-- 1 livy hadoop 33 2023-02-05 22:03 /user/hadoop/filtered_data/part-00000-c5ac7f4e-c869-4f  
f9-bd50-d96747ab572a-c000.csv  
-rw-r--r-- 1 livy hadoop 7305 2023-02-05 22:04 /user/hadoop/filtered_data/part-00023-c5ac7f4e-c869-4f  
f9-bd50-d96747ab572a-c000.csv  
[hadoop@ip-172-31-73-2 ~]$ hadoop fs -ls /user/hadoop/  
Found 2 items  
-rw-r--r-- 1 hadoop hadoop 5292105197 2023-02-05 20:48 /user/hadoop/eng_1M_lgram  
drwxr-xr-x - livy hadoop 0 2023-02-05 22:04 /user/hadoop/filtered_data  
[hadoop@ip-172-31-73-2 ~]$  
[hadoop@ip-172-31-73-2 ~]$ hadoop fs -getmerge /user/hadoop/filtered_data filtered_data.csv  
[hadoop@ip-172-31-73-2 ~]$ aws s3 cp filtered_data.csv s3://mahyar-bstn-bucket  
upload: ./filtered_data.csv to s3://mahyar-bstn-bucket/filtered_data.csv  
[hadoop@ip-172-31-73-2 ~]$  
[hadoop@ip-172-31-73-2 ~]$  
[hadoop@ip-172-31-73-2 ~]$  
[hadoop@ip-172-31-73-2 ~]$  
[hadoop@ip-172-31-73-2 ~]$  
[hadoop@ip-172-31-73-2 ~]$  
[hadoop@ip-172-31-73-2 ~]$ |
```



- After that I am going to type exit in the terminal and terminate the EMR cluster because we don't need it anymore. The filtered file is now on our S3 bucket (our scalable storage on the cloud) and we will continue our work on my local machine.

```
MINGW64:c/Users/mahya/cloud_novcohort
data,1998,208230,104315,4231
data,1999,230573,108982,4179
data,2000,223813,107134,4040
data,2001,247209,115258,3983
data,2002,241848,109219,3825
data,2003,244250,105400,3684
data,2004,201841,93539,3483
data,2005,197467,88901,3392
data,2006,203669,92960,3449
data,2007,168338,78986,3246
data,2008,105331,47811,2358
[hadoop@ip-172-31-73-2 ~]$ hadoop fs -ls /user/hadoop/filtered_data
Found 3 items
-rw-r--r-- 1 livy hadoop          0 2023-02-05 22:04 /user/hadoop/filtered_data/_SUCCESS
-rw-r--r-- 1 livy hadoop      33 2023-02-05 22:03 /user/hadoop/filtered_data/part-00000-c5ac7f4e-c869-4f
f9-bd50-d96747ab572a-c000.csv
-rw-r--r-- 1 livy hadoop    7305 2023-02-05 22:04 /user/hadoop/filtered_data/part-00023-c5ac7f4e-c869-4f
f9-bd50-d96747ab572a-c000.csv
[hadoop@ip-172-31-73-2 ~]$ hadoop fs -ls /user/hadoop/
Found 2 items
-rw-r--r-- 1 hadoop hadoop 5292105197 2023-02-05 20:48 /user/hadoop/eng_1M_igram
drwxr-xr-x - livy hadoop      0 2023-02-05 22:04 /user/hadoop/filtered_data
[hadoop@ip-172-31-73-2 ~]$
[hadoop@ip-172-31-73-2 ~]$ hadoop fs -getmerge /user/hadoop/filtered_data filtered_data.csv
[hadoop@ip-172-31-73-2 ~]$/ aws s3 cp filtered_data.csv s3://mahyar-bstn-bucket
upload: ./filtered_data.csv to s3://mahyar-bstn-bucket/filtered_data.csv
[hadoop@ip-172-31-73-2 ~]$
[hadoop@ip-172-31-73-2 ~]$ exit
logout
Connection to ec2-18-232-73-173.compute-1.amazonaws.com closed.
(Cloud_Lab)
mahya@LAPTOP-CRRPRI3K MINGW64 ~/cloud_novcohort
$
```

Amazon EMR has launched a new console experience. [Learn more](#) or switch to the new console.

EMR Serverless is now GA.
With EMR Serverless, get the benefits of Amazon EMR such as open source compatibility, latest versions and performance optimized runtime for popular frameworks along with easy provisioning, quick job startup, automatic capacity management, and simple cost controls. [Get Started with EMR Serverless](#).

Clusters

Cluster: mahir_cluster Waiting Cluster ready to run steps.

Summary

ID: j-1Z6AFG8DDU428
Creation date: 2023-02-05 12:19 (UTC-8)
Elapsed time: 2 hours, 21 minutes
After last step completes: Cluster waits
Termination protection: On Change
Tags: -- View All / Edit
Master public DNS: ec2-18-232-73-173.compute-1.amazonaws.com Connect to the Master Node Using SSH

Configuration details

Release label: emi-6.1.1
Hadoop distribution: Amazon 3.2.1
Applications: Hive 3.1.2, Hue 4.7.1, Spark 3.0.0, Livy 0.7.0, JupyterHub 1.1.0
Log URI: s3://aws-logs-181777116252-us-east-1/elasticmapreduce/
EMRFS consistent view: Disabled
Custom AMI ID: --

Application user interfaces

Persistent user interfaces: Spark history server, YARN timeline server, Tez UI
On-cluster user interfaces: Not Enabled Enable an SSH Connection

Network and hardware

Availability zone: us-east-1f
Subnet ID: subnet-00c1730dce8622b05
Master: Running 1 m5.xlarge
Core: Running 2 m5.xlarge
Task: --
Cluster scaling: Not enabled
Auto-termination: Not enabled

Security and access

Key name: mahir_aws
EC2 instance profile: EMR_EC2_DefaultRole
EMR role: EMR_DefaultRole
Auto Scaling role: EMR_AutoScaling_DefaultRole
Visible to all users: All Change
Security groups for Master: sg_0a0cb250ed3efc7a3 (ElasticMapReduce-master)
Security groups for Core & sg_07423b4d4e7300d4c (ElasticMapReduce-Task: slave)

Amazon EMR has launched a new console experience. [Learn more](#) or switch to the new console.

EMR Serverless is now GA.
With EMR Serverless, get the benefits of Amazon EMR such as open source compatibility, latest versions and performance optimized runtime for popular frameworks along with easy provisioning, quick job startup, automatic capacity management, and simple cost controls. [Get Started with EMR Serverless](#).

Clusters

Cluster: mahir_cluster Terminated Terminated by user request

Summary

ID: j-1Z6AFG8DDU428
Creation date: 2023-02-05 12:19 (UTC-8)
End date: 2023-02-05 14:45 (UTC-8)
Elapsed time: 2 hours, 25 minutes
After last step completes: Cluster waits
Termination protection: Off
Tags: --
Master public DNS: ec2-18-232-73-173.compute-1.amazonaws.com Connect to the Master Node Using SSH

Configuration details

Release label: emi-6.1.1
Hadoop distribution: Amazon 3.2.1
Applications: Hive 3.1.2, Hue 4.7.1, Spark 3.0.0, Livy 0.7.0, JupyterHub 1.1.0
Log URI: s3://aws-logs-181777116252-us-east-1/elasticmapreduce/
EMRFS consistent view: Disabled
Custom AMI ID: --

Application user interfaces

Persistent user interfaces: Spark history server, YARN timeline server, Tez UI
On-cluster user interfaces: --

Network and hardware

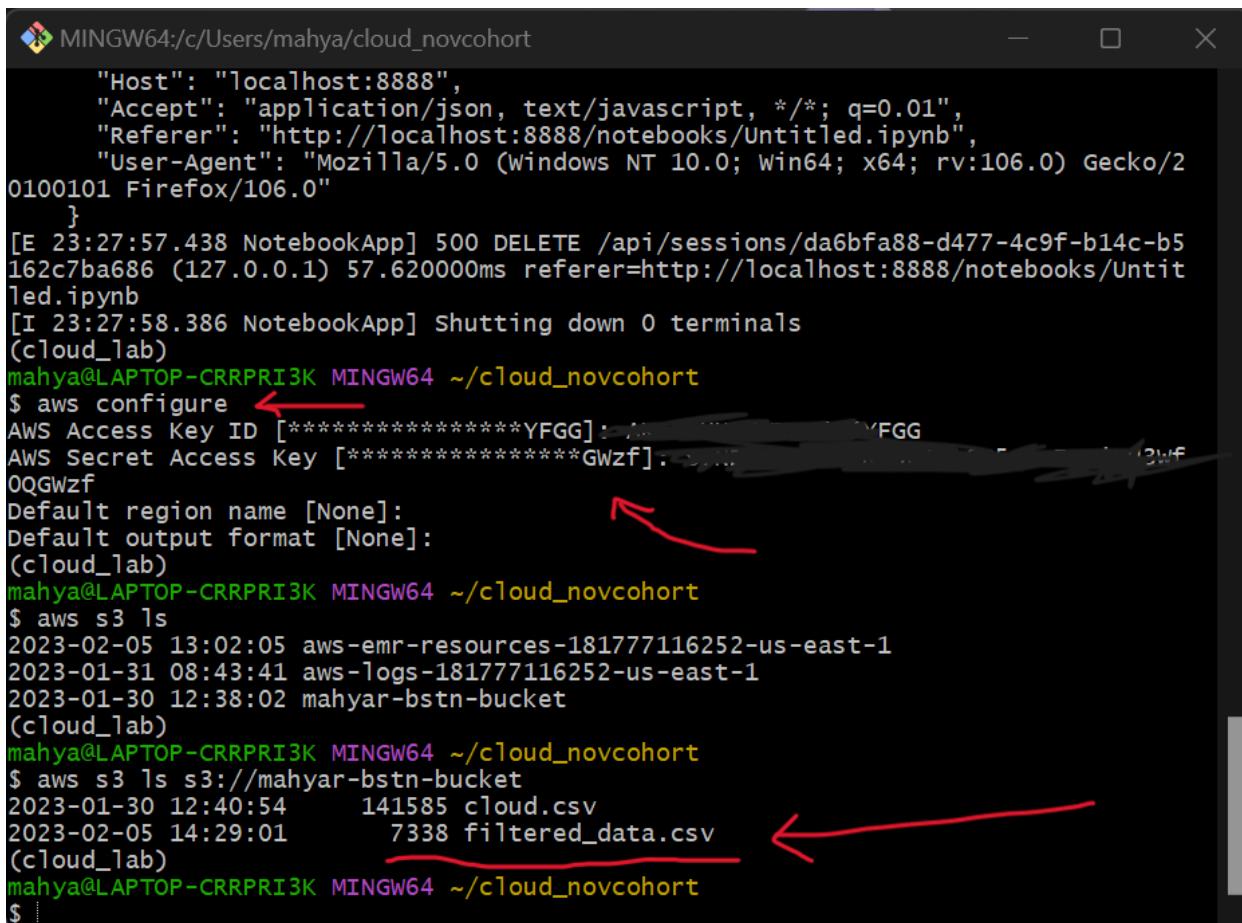
Availability zone: us-east-1f
Subnet ID: subnet-00c1730dce8622b05
Master: Terminated 1 m5.xlarge
Core: Terminated 2 m5.xlarge
Task: --
Cluster scaling: Not enabled
Auto-termination: Not enabled

Security and access

Key name: mahir_aws
EC2 instance profile: EMR_EC2_DefaultRole
EMR role: EMR_DefaultRole
Auto Scaling role: EMR_AutoScaling_DefaultRole
Visible to all users: All Change
Security groups for Master: sg_0a0cb250ed3efc7a3 (ElasticMapReduce-master)
Security groups for Core & sg_07423b4d4e7300d4c (ElasticMapReduce-Task: slave)

- Now our cluster is successfully terminated and let's move to question 6:

Question 6: In this step, on my local machine (or on AWS outside of Spark) in python, I am going to read the CSV data from the S3 folder into a Pandas data frame. But before that, I must have first authenticated on my machine using AWS configure on the command line as below:



The screenshot shows a terminal window titled "MINGW64:/c/Users/mahya/cloud_novcohorts". It displays the following command-line session:

```
Host": "localhost:8888",
"Accept": "application/json, text/javascript, */*; q=0.01",
"Referer": "http://localhost:8888/notebooks/Untitled.ipynb",
"User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:106.0) Gecko/20100101 Firefox/106.0"
}
[E 23:27:57.438 NotebookApp] 500 DELETE /api/sessions/da6bfa88-d477-4c9f-b14c-b5162c7ba686 (127.0.0.1) 57.620000ms referer=http://localhost:8888/notebooks/Untitled.ipynb
[I 23:27:58.386 NotebookApp] Shutting down 0 terminals
(cloud_lab)
mahya@LAPTOP-CRRPRI3K MINGW64 ~/cloud_novcohorts
$ aws configure ←
AWS Access Key ID [*****YFGG]: *****YFGG ←
AWS Secret Access Key [*****GWzf]. ... ←
0QGWzf
Default region name [None]: ←
Default output format [None]: ←
(cloud_lab)
mahya@LAPTOP-CRRPRI3K MINGW64 ~/cloud_novcohorts
$ aws s3 ls
2023-02-05 13:02:05 aws-emr-resources-181777116252-us-east-1
2023-01-31 08:43:41 aws-logs-181777116252-us-east-1
2023-01-30 12:38:02 mahyar-bstn-bucket
(cloud_lab)
mahya@LAPTOP-CRRPRI3K MINGW64 ~/cloud_novcohorts
$ aws s3 ls s3://mahyar-bstn-bucket
2023-01-30 12:40:54      141585 cloud.csv ←
2023-02-05 14:29:01      7338 filtered_data.csv ←
(cloud_lab)
mahya@LAPTOP-CRRPRI3K MINGW64 ~/cloud_novcohorts
$
```

Red arrows highlight the command "aws configure", the AWS Access Key ID and Secret Access Key fields, and the two files listed in the final "aws s3 ls" command.

- Then I ran some codes to check whether the filtered file exists on my S3 bucket. Now we will open the jupyter note book and continue answering questions 6,7,8 there on local computer:

Remaining Questions:

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.
7. Plot the number of occurrences of the token (the frequency column) of data over the years using matplotlib.
8. Compare Hadoop and Spark as distributed file systems:
 - a. What are the advantages/ differences between Hadoop and Spark? List two advantages for each.
 - b. Explain how the HDFS stores the data.

6

- In this step we should first authenticated on our machine using aws configure on the command line.
- Then we will use boto to read the csv data into pandas dataframe:

Using Boto3 to communicate with S3:

- First we will import packages we need:

```
In [1]: # Import packages
1 # Import packages
2
3 import boto3
4 import pandas as pd
5 import matplotlib.pyplot as plt
```

- First we will import packages we need:

```
In [1]: # Import packages
1 # Import packages
2
3 import boto3
4 import pandas as pd
5 import matplotlib.pyplot as plt
```

Let's start with the Client method. We will create a boto3 client object and pass it the AWS service we would like to use, which in this case is S3.

```
In [2]: # Instantiate an S3 client
1 # Instantiate an S3 client
2 s3_client = boto3.client('s3')
3
4 # Check
5 s3_client
```

```
Out[2]: <boto3.core.client.S3 at 0x2632244ebb0>
```

- Recall that this instance is linked to the AWS account authorized on your computer, and if you run this notebook you will see different outputs than what I have here.

```
In [3]: # Send a request to list all my buckets, and assign the response to a variable
1 # Send a request to list all my buckets, and assign the response to a variable
2 response = s3_client.list_buckets()
3
4 # Check
5 response
```

```
Out[3]: {'ResponseMetadata': {'RequestId': 'MQBECBEXBA36ZK0R',
'HostId': '0NY45PAIsx/U01T5C727v7nRIPn600FryeyfRPjXjD0/wID9q88j6/tsM0VeDEPmvE/A887dvhM=',
'HTTPStatusCode': 200,
'HTTPHeaders': {'x-amz-id-2': '0NY45PAIsx/U01T5C727v7nRIPn600FryeyfRPjXjD0/wID9q88j6/tsM0VeDEPmvE/A887dvhM=',
'x-amz-request-id': 'MQBECBEXBA36ZK0R',
'date': 'Mon, 06 Feb 2023 10:05:25 GMT',
'content-type': 'application/xml',
'transfer-encoding': 'chunked',
'server': 'AmazonS3'},
'RetryAttempts': 0},
'Buckets': [{}{'Name': 'aws-emr-resources-181777116252-us-east-1', ...}]}  
...  
...
```

```

    'RetryAttempts': 0},
    'Buckets': [{Name: 'aws-emr-resources-181777116252-us-east-1',
      'CreationDate': datetime.datetime(2023, 2, 5, 21, 2, 5, tzinfo=tzutc()),},
    {'Name': 'aws-logs-181777116252-us-east-1',
      'CreationDate': datetime.datetime(2023, 1, 31, 16, 43, 41, tzinfo=tzutc()),},
    {'Name': 'mahyar-bstn-bucket',
      'CreationDate': datetime.datetime(2023, 1, 30, 20, 38, 2, tzinfo=tzutc())}],
    'Owner': {'DisplayName': 'mahyar_sabooni',
      'ID': '7eb122d16e9fa085284e3de4a147f08b6b4a882422396d0227276f95079a5618'}}]
```

- We can access the keys and values of a dictionary using the `.keys()` and `.values()` methods, respectively:

```
In [4]: 1 # The keys
2
3 response.keys()

Out[4]: dict_keys(['ResponseMetadata', 'Buckets', 'Owner'])

In [5]: 1 # The values
2
3 response.values()

Out[5]: dict_values([{'RequestId': 'MQBECBEXBA36ZK0R', 'HostId': '0NY45PA1sx/U01T5C727v7nRIPn600FryeyfRPjXjD0/wID9q88j6/tsM0VeDEPmvE/A887dvhM=',
  'HTTPStatusCode': 200, 'HTTPHeaders': {'x-amz-id-2': '0NY45PA1sx/U01T5C727v7nRIPn600FryeyfRPjXjD0/wID9q88j6/tsM0VeDEPmvE/A887dvhM=',
    'x-amz-request-id': 'MQBECBEXBA36ZK0R'}, 'date': 'Mon, 06 Feb 2023 10:05:25 GMT', 'content-type': 'application/xml', 'transfer-encoding': 'chunked', 'server': 'AmazonS3'}, 'RetryAttempts': 0}, [{"Name": "aws-emr-resources-181777116252-us-east-1", "CreationDate": datetime.datetime(2023, 2, 5, 21, 2, 5, tzinfo=tzutc())}, {"Name": "aws-logs-181777116252-us-east-1", "CreationDate": datetime.datetime(2023, 1, 31, 16, 43, 41, tzinfo=tzutc())}, {"Name": "mahyar-bstn-bucket", "CreationDate": datetime.datetime(2023, 1, 30, 20, 38, 2, tzinfo=tzutc())}], {'DisplayName': 'mahyar_sabooni', 'ID': '7eb122d16e9fa085284e3de4a147f08b6b4a882422396d0227276f95079a5618'}])
```

- The values aren't easy for us to work with, so let's use the keys instead:

```
In [6]: 1 # What's the output if I index the big dictionary by the 'Buckets' key?
2
3 response['Buckets']

In [6]: 1 # What's the output if I index the big dictionary by the 'Buckets' key?
2
3 response['Buckets']

Out[6]: [{"Name": "aws-emr-resources-181777116252-us-east-1", "CreationDate": datetime.datetime(2023, 2, 5, 21, 2, 5, tzinfo=tzutc())}, {"Name": "aws-logs-181777116252-us-east-1", "CreationDate": datetime.datetime(2023, 1, 31, 16, 43, 41, tzinfo=tzutc())}, {"Name": "mahyar-bstn-bucket", "CreationDate": datetime.datetime(2023, 1, 30, 20, 38, 2, tzinfo=tzutc())}]

In [7]: 1 # Type of the output above
2
3 type(response['Buckets'])

Out[7]: list
```

- Now let's print the names of my buckets:

```
In [9]: 1 # Instantiate a resource service client
2
3 s3_resource = boto3.resource('s3')
4
5 # Iterate over the buckets
6
7 for bucket in s3_resource.buckets.all():
8
9   # Use the `name` method on each bucket to print its name
10  print(bucket.name)

aws-emr-resources-181777116252-us-east-1
aws-logs-181777116252-us-east-1
mahyar-bstn-bucket
```

- Accessing and reading in a specific file:

- Accessing and reading in a specific file:

```
In [10]: 1 # Get the specified file from the specified bucket and assign it to a variable
2
3 s3_object = s3_client.get_object(Bucket='mhyar-bstn-bucket', Key='filtered_data.csv')
4
5 # Check
6
7 s3_object

Out[10]: {'ResponseMetadata': {'RequestId': 'QQ1Z87ZJA41N3FEX',
'HostId': 'pFXqElHUzy1FbNP4PW8ufEFAqh3/JF24yOfyG6LiC7YTQ7ke3Z/KY+9PGoR15aGp10mT64Lytf8=',
'HTTPStatusCode': 200,
'HTTPHeaders': {'x-amz-id-2': 'pFXqElHUzy1FbNP4PW8ufEFAqh3/JF24yOfyG6LiC7YTQ7ke3Z/KY+9PGoR15aGp10mT64Lytf8=',
'x-amz-request-id': 'QQ1Z87ZJA41N3FEX',
'date': 'Mon, 06 Feb 2023 10:11:14 GMT',
'last-modified': 'Sun, 05 Feb 2023 22:29:01 GMT',
'etag': '"fb3cf3d45b1fc32b939cbd13697dbe68"',
'x-amz-server-side-encryption': 'AES256',
'accept-ranges': 'bytes',
'content-type': 'text/csv',
'server': 'AmazonS3',
'content-length': '7338'},
'RetryAttempts': 0,
'AcceptRanges': 'bytes',
'LastModified': datetime.datetime(2023, 2, 5, 22, 29, 1, tzinfo=tzutc()),
'ContentLength': 7338,
'ETag': '"fb3cf3d45b1fc32b939cbd13697dbe68"',
'ContentType': 'text/csv',
'ServerSideEncryption': 'AES256',
'Metadata': {},
'Body': <botocore.response.StreamingBody at 0x26322550820>}

In [11]: 1 # Index the above by the 'Body' key
2
3 s3_object['Body']

Out[11]: <botocore.response.StreamingBody at 0x26322550820>
```

- Now let's read the data:

```
In [12]: 1 # Read the csv into a dataframe; note that we pass in the 'StreamingBody' above
2
3 s3_filtered_data = pd.read_csv(s3_object['Body'] , header = 1)
4
5 # Check
6
7 s3_filtered_data

Out[12]:
   token  year  frequency  pages  books
0     data  1584        16     14      1
1     data  1614         3      2      1
2     data  1627         1      1      1
3     data  1631        22     18      1
4     data  1637         1      1      1
...
311    data  2004  201841  93539    3483
312    data  2005  197467  88901    3392
313    data  2006  203669  92960    3449
314    data  2007  168338  78986    3246
315    data  2008  105331  47811    2358

316 rows × 5 columns
```

- This is one way to read the data with boto3. we can even read the data directly into Pandas with the code below:

Reading data directly into pandas from s3 bucket:

```
In [13]: # Read the csv from the specified S3 path into a dataframe  
1  
2  
3 filtered_data_directly_pandas = pd.read_csv('s3://mahyar-bstn-bucket/filtered_data.csv', header = 1 )  
4  
5 # Check  
6  
7 filtered_data_directly_pandas
```

```
Out[13]:
```

	token	year	frequency	pages	books
0	data	1584	16	14	1
1	data	1614	3	2	1
2	data	1627	1	1	1
3	data	1631	22	18	1
4	data	1637	1	1	1
...
311	data	2004	201841	93539	3483
312	data	2005	197467	88901	3392
313	data	2006	203669	92960	3449
314	data	2007	168338	78986	3246
315	data	2008	105331	47811	2358

316 rows × 5 columns

- Now let's check the shape, info and some basic math calculations of the numeric columns:

```
In [14]: filtered_data_directly_pandas.describe()
```

```
Out[14]:
```

	year	frequency	pages	books
count	316.000000	316.000000	316.000000	316.000000
mean	1847.569620	38555.993671	21711.041139	1493.110759

- Now let's check the shape, info and some basic math calculations of the numeric columns:

```
In [14]: filtered_data_directly_pandas.describe()
```

```
Out[14]:
```

	year	frequency	pages	books
count	316.000000	316.000000	316.000000	316.000000
mean	1847.569620	38555.993671	21711.041139	1493.110759
std	98.874382	69212.366418	34901.797740	1560.040802
min	1584.000000	1.000000	1.000000	1.000000
25%	1771.750000	20.750000	19.000000	11.750000
50%	1850.500000	3004.000000	2729.500000	868.000000
75%	1929.250000	41776.750000	30195.750000	2964.250000
max	2008.000000	254561.000000	122472.000000	4372.000000

```
In [15]: filtered_data_directly_pandas.shape
```

```
Out[15]: (316, 5)
```

- Our data has 316 rows and 5 columns.

```
In [16]: filtered_data_directly_pandas.info()
```

```
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 316 entries, 0 to 315  
Data columns (total 5 columns):  
 #   Column      Non-Null Count  Dtype     
---  --          --          --          --  
 0   token       316 non-null    object    
 1   year        316 non-null    int64    
 2   frequency   316 non-null    int64    
 3   pages       316 non-null    int64    
 4   books       316 non-null    int64  
dtypes: int64(4), object(1)  
memory usage: 12.5+ KB
```

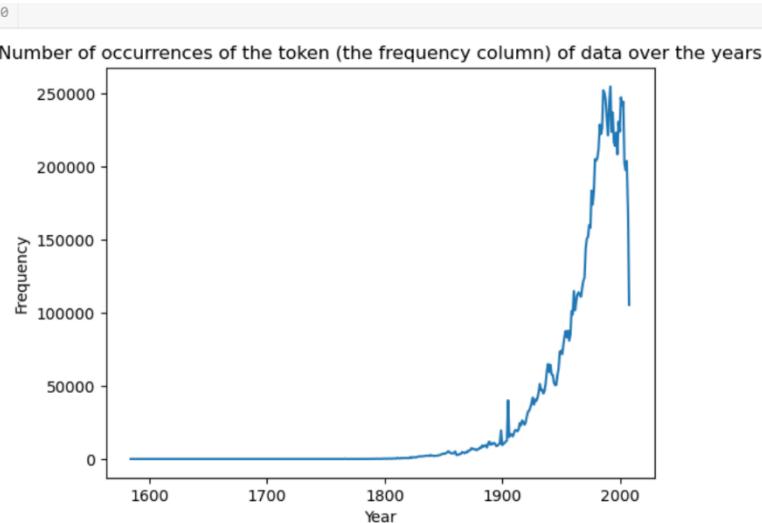
Question 7: I will plot the frequency over year using Matplotlib:

- We have 4 numeric columns and one object column(token). We have no null values.
- In this question we read the data into a pandas dataframe in 2 ways and then we did some basic describe on the dataset.

7

- Now we will plot the number of occurrences of the token (the frequency column) of data over the years using matplotlib:

```
In [18]: 1 import matplotlib.pyplot as plt
2
3 # Group the data by year and calculate the sum of the frequency column
4
5 grouped_data = filtered_data.groupby("year").sum()
6
7 # Plot the sum of the frequency column for each year
8
9 plt.plot(grouped_data.index, grouped_data["frequency"])
10
11 # Add Labels to the x and y axes
12
13 plt.title("Number of occurrences of the token (the frequency column) of data over the years")
14 plt.xlabel("Year")
15 plt.ylabel("Frequency")
16
17 # Show the plot
18
19 plt.show()
20
```



- Question 8: Compare Hadoop and Spark as distributed file systems.

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

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

Hadoop and Spark are both distributed file systems, but they have some differences in terms of their design and functionality:

Advantages of Hadoop:

1. Scalability: Hadoop can scale to handle very large data sets by adding more nodes to the cluster.
2. Cost-effective: Hadoop is an open-source solution, so there are no licensing costs involved, and it can run on commodity hardware.

Advantages of Spark:

1. Speed: Spark is much faster than Hadoop MapReduce for big data processing because it uses in-memory computing and can cache intermediate data in memory for faster access.
2. Ease of use: Spark has a high-level API that makes it easier for developers to write and maintain big data processing code, compared to Hadoop MapReduce, which has a low-level API.

It is worth noting that Hadoop and Spark are complementary technologies, and Spark can be used on top of Hadoop as an alternative to MapReduce for big data processing.

- B. Explain how the HDFS stores the data.

8.b : Explain how the HDFS stores the data.

HDFS (Hadoop Distributed File System) is the primary storage system used by Hadoop. HDFS stores data as large files that are broken down into smaller blocks and distributed across multiple nodes in a Hadoop cluster. The blocks are replicated across multiple nodes in the cluster for increased reliability and to provide fault tolerance in case a node fails.

When a file is written to HDFS, it is divided into blocks and each block is stored on multiple nodes. The blocks are stored on the nodes close to the data, reducing network latency and increasing data transfer speeds. The blocks are also replicated to other nodes in the cluster, providing backup copies of the data in case a node fails.

HDFS uses a master/slave architecture, with a NameNode as the master and DataNodes as the slaves. The NameNode maintains the file system namespace, manages the mapping of blocks to DataNodes, and coordinates access to files stored in HDFS. The DataNodes are responsible for storing the actual data blocks and serving read and write requests from clients.

In summary, HDFS is a highly scalable, fault-tolerant, and distributed file system that provides reliable data storage for Hadoop clusters.

- I attached the filtered_data.csv file to the project folder too in case you want to check it.

Thank You for reading my work!