

Project: Big Data Wrangling With Google Books Ngrams

Author: Zhassulan Yeshmukhametov

Due Date: 26/01/2024

Assignment overview

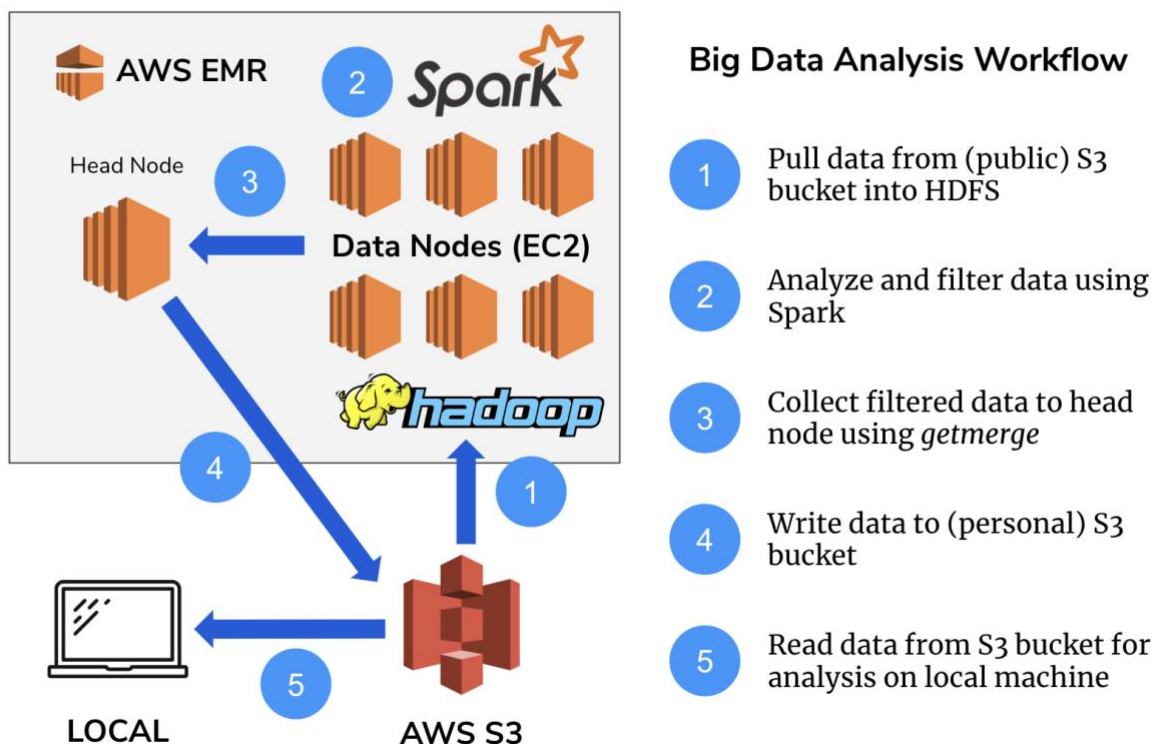
In this assignment, you will apply the skills you've learned in the Big Data Fundamentals unit to load, filter, and visualize a large real-world dataset in a cloud-based distributed computing environment using Hadoop, Spark, Hive, and the S3 filesystem. Prepare a professional report to summarize the 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](https://brainstation-dsft/eng_1M_1gram.csv)

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

Copyright BrainStation



below:

The scope of data processing and analysis is outlined in the questions below. Please document all your steps and commands/ code; there are several format options to submit both the code and a written report.

Assignment questions:

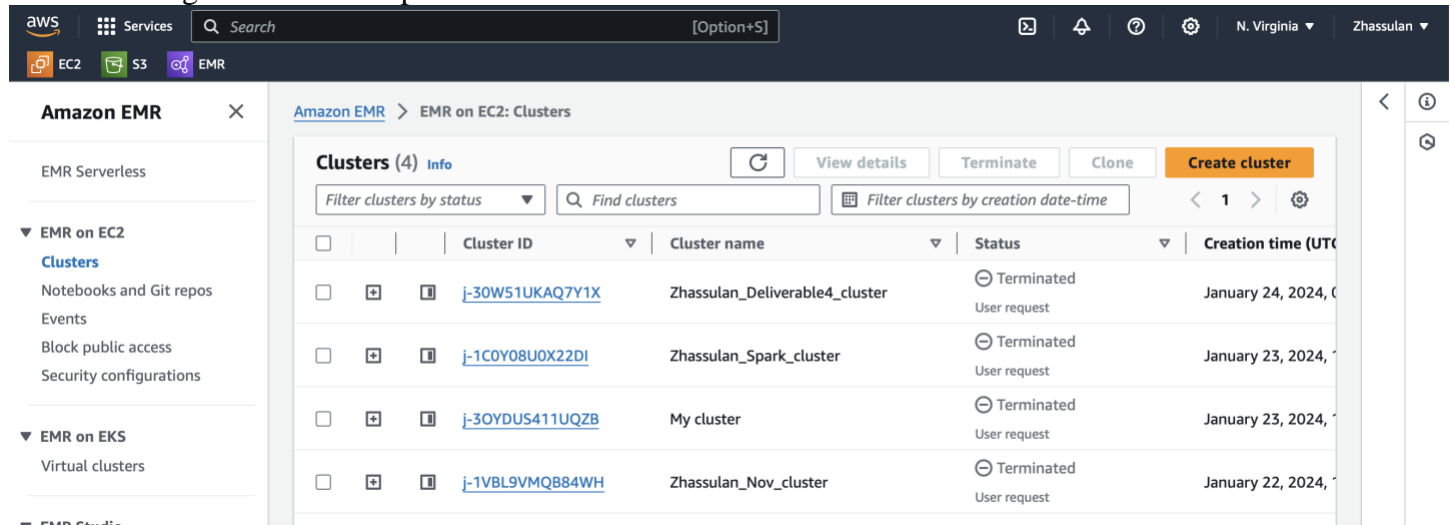
1. Spin up a new EMR cluster on AWS for using Spark and EMR notebooks - **follow the same instructions as for the Spark Lab.**
2. Connect to the head node of the cluster using SSH.
3. Copy the data folder from the S3 bucket *directly* into a directory on the Hadoop File System (HDFS) named `/user/hadoop/eng_1M_1gram`.
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:
 - 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.
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.
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.

Question 1. Spin up a new EMR cluster on AWS for using Spark and EMR notebooks - follow the same instructions as for the Spark Lab.

Create Cluster

First, we need to sign in to the AWS Management Console.

Then we navigate to the EMR panel in AWS and click 'Create Cluster'.

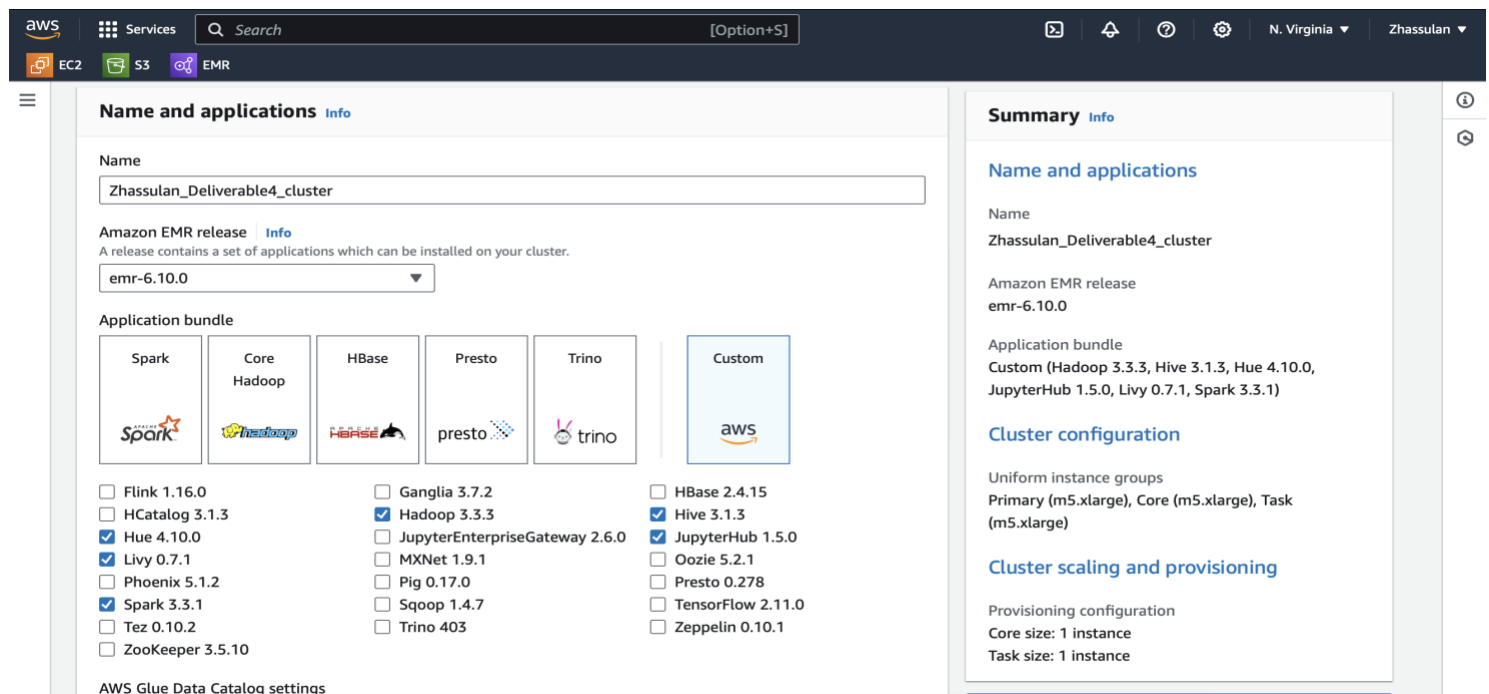


Cluster name and software

In name and application section I named cluster as “Zhasulan_Deliverable4_cluster”

Here we set Amazon EMR release as emr-6.10.0.

Next we choose application bundle. I choose “Custom” and included necessary applications in my bundle.



Instance Groups

WE removing the task instance group, as we no need it and it helps in cost optimization by stopping the billing for those instances. Also we no need additional computation capacity.

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: \$0.078 (us-east-1f)

Remove instance group

Node configuration - optional

Add task instance group
You can add up to 48 more task instance groups.

EBS root volume

EBS root volume applies to the operating systems and applications that you install on the cluster.

Size (GiB)

15
15 - 100 GiB per volume General Purpose SSD (gp2)

Summary Info

Name and applications

Name
Zhassulan_Deliverable4_cluster

Amazon EMR release
emr-6.10.0

Application bundle
Custom (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 configuration

Uniform instance groups
Primary (m5.xlarge), Core (m5.xlarge)

Cluster scaling and provisioning

Provisioning configuration
Core size: 2 instances

Networking

By allocating 2 nodes to the core instance group we are dedicating two virtual machines (instances) to handle the core task within the cluster.

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	<input type="checkbox"/>

Summary Info

Name and applications

Name
Zhassulan_Deliverable4_cluster

Amazon EMR release
emr-6.10.0

Application bundle
Custom (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 configuration

Uniform instance groups
Primary (m5.xlarge), Core (m5.xlarge)

Cluster Termination

I set Idle time to 1 day, to make sure I have enough time to finish Assignment.

And I decided to keep “Use termination protection”, not to terminate cluster accidentally before I done Deliverable4.

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.

1 day 01:00:00

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.

Summary Info

Name and applications

Name
Zhassulan_Deliverable4_cluster

Amazon EMR release
emr-6.10.0

Application bundle
Custom (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 configuration

Uniform instance groups
Primary (m5.xlarge), Core (m5.xlarge)

Cluster scaling and provisioning

Cluster logs

Cluster logs - optional [Info](#)

We automatically archive your log files to Amazon S3. You can specify your own S3 location, or use the default S3 location for Amazon EMR. The default log location is pre-populated in the **Amazon S3 location** field.

☒ Publish cluster-specific logs to Amazon S3

Amazon S3 location

s3://aws-logs-992382709365-us-east-1/elasticmapreduce

×

View [↗](#)

Browse S3

Format: Use s3://bucket/prefix

☐ Encrypt cluster-specific logs

Summary [Info](#)

Name and applications

Name

Zhassulan_Deliverable4_cluster

Amazon EMR release

emr-6.10.0

Application bundle

Custom (Hadoop 3.3.3, Hive 3.1.3, Hue 4.10.0, JupyterHub 1.5.0, Livy 0.7.1, Spark 3.3.1)

Bucket was created automatically here, and I used it for my Deliverable. I could create it myself, how we did during the class.

Create bucket [Info](#)

Buckets are containers for data stored in S3. [Learn more](#) [↗](#)

General configuration

AWS Region

US East (N. Virginia) us-east-1

Bucket type [Info](#)

☒ General purpose
Recommended for most use cases and access patterns. General purpose buckets are the original S3 bucket type. They allow a mix of storage classes that redundantly store objects across multiple Availability Zones.

☐ Directory - New
Recommended for low-latency use cases. These buckets use only the S3 Express One Zone storage class, which provides faster processing of data within a single Availability Zone.

Bucket name [Info](#)

Zhassulan-Deliverable4-bucket

Bucket name must be unique within the global namespace and follow the bucket naming rules. [See rules for bucket naming](#) [↗](#)

Copy settings from existing bucket - optional

Only the bucket settings in the following configuration are copied.

Choose bucket

Format: s3://bucket/prefix

To create a new bucket we need to access S3 in AWS Management Console and click “Create bucket”. Define name for the bucket and choose “AWS region” In bucket we need to create Access key pair. This consists of an Access Key ID and a Secret Access Key. These keys are used to authenticate your requests when making API calls or using AWS SDKs. In Terminal on Mac we need to run `aws configure` and here we will enter our keys.

Services

Search

[Option+S]

EC2

S3

EMR

Access key created

This is the only time that the secret access key can be viewed or downloaded. You cannot recover it later. However, you can create a new access key any time.

```
[hadoop@ip-172-31-73-86 ~]$ aws configure
[AWS Access Key ID [None]: AKIA60DU6NZ23ALR4XWX
AWS Secret Access Key [None]:
```

Security and Access Management

Here I'm using my created keypair for SSH access to the master node of the EMR cluster

Security configuration and EC2 key pair - *optional* [Info](#)

Security configuration

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

Amazon EC2 key pair for SSH to the cluster [Info](#)

By choosing EMR_DefaultRole we grant permission to access AWS services, in our case is Amazon S3. By choosing EMR_EC2_DefaultRole2 it assigned to the EC2 instances that make up our cluster. This role provide necessary permissions for instances to interact with AWS services.

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

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

And we Finish with clicking the “Create cluster” button.

[Amazon EMR](#) > EMR on EC2: Clusters

Clusters (4) [Info](#)

Filter clusters by status

< 1 >

	Cluster ID	Cluster name	Status	Creation time (UTC-08:00)
	j-30W51UKAQ7Y1X	Zhassulan_Deliverable4_cluster	Terminated User request	January 24, 2024, 09:48

P.S. I print screen it after termination

Question 2. Connect to the head node of the cluster using SSH.

I've done through Terminal on Mac.

First, I activated cloud_lab environment, by using **conda activate cloud_lab** command

And connected to head node by using:

ssh -i /path/to/your/keypair.pem hadoop@xxxxxxx.compute.amazonaws.com

```
(base) zhassulan@MacBook-Air-Zasulan ~ % conda activate cloud_lab
(cloud_lab) zhassulan@MacBook-Air-Zasulan ~ % ssh -i Desktop/BrainStation/AWS\ keypair/Zhassulan_Nov_keypair.pem hadoop@ec2-3-235-65-30.compute-1.amazonaws.com
```

The screenshot displays the AWS Management Console interface for an Amazon EMR cluster. The cluster name is 'Zhassulan_Deliverable4_cluster'. The summary section indicates it is an Amazon Linux 2 AMI, emr-6.10.0 version, with Hadoop 3.3.3, Hive 3.1.3, Hue, and JupyterHub 1.5.0 installed. The instance groups section shows 1 Primary, 2 Core, and 0 Task instances. A terminal window is overlaid on the console, showing the SSH command being executed: 'ssh -i Desktop/BrainStation/AWS/keypair/Zhassulan_Nov_keypair.pem hadoop@ec2-3-235-65-30.compute-1.amazonaws.com'. The terminal output shows the SSH connection process, including the warning about the package needed for security, and the successful login to the hadoop user on the ec2-3-235-65-30 instance.

Question 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`.

To do this, we are using command: **hadoop distcp s3://brainstation-dsft/eng_1M_1gram.csv /user/Hadoop/eng_1M_1gram.csv**

```
[hadoop@ip-172-31-73-86 ~]$ hadoop distcp s3://brainstation-dsft/eng_1M_1gram.csv /user/hadoop/eng_1M_1gram.csv
2024-01-24 18:42:32,992 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=[], atomicWorkPath=null, logPath=null, sourceFileListing=null, sourcePaths=[s3://brainstation-dsft/eng_1M_1gram.csv], targetPath=/user/hadoop/eng_1M_1gram.csv, filtersFile='null', blocksPerChunk=0, copyBufferSize=8192, verboseLog=false, directWrite=false, useIterator=false}, sourcePaths=[s3://brainstation-dsft/eng_1M_1gram.csv], targetPaths=[/user/hadoop/eng_1M_1gram.csv]
2024-01-24 18:42:33,259 INFO client.DefaultNoHARMAFailoverProxyProvider: Connecting to ResourceManager at ip-172-31-73-86.ec2.internal/172.31.73.86:8032
2024-01-24 18:42:33,414 INFO client.AHSPProxy: Connecting to Application History server at ip-172-31-73-86.ec2.internal/172.31.73.86:10200
2024-01-24 18:42:36,412 INFO tools.SimpleCopyListing: Starting: Building listing using multi threaded approach for s3://brainstation-dsft/eng_1M_1gram.csv
2024-01-24 18:42:36,414 INFO tools.SimpleCopyListing: Building listing using multi threaded approach for s3://brainstation-dsft/eng_1M_1gram.csv: duration 0:00.002s
2024-01-24 18:42:36,537 INFO tools.SimpleCopyListing: Paths (files+dirs) cnt = 1 ; dirCnt = 0
```

Sanity check by using command: **hadoop fs -ls /user/hadoop**

```
[hadoop@ip-172-31-73-86 ~]$ hadoop fs -ls /user/hadoop
Found 1 items
-rw-r--r--    1 hadoop hdfsadmingroup 5292105197 2024-01-24 18:43 /user/hadoop/en
g_1M_1gram.csv
```

We see that have our csv file successfully copied.

Question 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`).

I choose to work with Jupyterhub on EMR.
And here how I accessed it, by using command below:

```
(cloud_lab) zhassulan@MacBook-Air-Zasulan ~ % ssh -i Desktop/BrainStation/AWS\ keypair/Zhassulan_Nov_keypair.pem -L 9995:localhost:9443 hadoop@ec2-3-235-65-30.compute-1.amazonaws.com
```

Here is a view on Jupyterhub on EMR



[Logout](#)
[Control Panel](#)

[Files](#)
[Running](#)
[Clusters](#)

Select items to perform actions on them.

[Upload](#)
[New ▾](#)
[↻](#)

0 ▾ /		Name ▾	Last Modified	File size
<input type="checkbox"/>	 jupyterhub-proxy.pid		seconds ago	2 B
<input type="checkbox"/>	 jupyterhub.sqlite		seconds ago	102 kB
<input type="checkbox"/>	 jupyterhub_cookie_secret		seconds ago	65 B

Next answers for question 4 will be on Jupyterhub and fail will be attached for submission.

Question 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.

First we check if we done everything right with passing filtered data to HDFS

```
[[hadoop@ip-172-31-73-86 ~]$ hadoop fs -ls /user/hadoop
Found 2 items
-rw-r--r--    1 hadoop hdfsadmingroup 5292105197 2024-01-24 18:43 /user/hadoop/en
g_1M_1gram.csv
drwxr-xr-x    - livy    hdfsadmingroup          0 2024-01-24 19:36 /user/hadoop/fi
ltered_data_csv
[hadoop@ip-172-31-73-86 ~]$
```

After sanity check, we can use **getmerge** and move file into S3 bucket.


```
[hadoop@ip-172-31-73-86 ~]$ hadoop fs -ls /user/hadoop
Found 2 items
-rw-r--r--    1 hadoop hdfsadmingroup 5292105197 2024-01-24 18:43 /user/hadoop/en
g_1M_1gram.csv
drwxr-xr-x   - livy    hdfsadmingroup          0 2024-01-24 20:13 /user/hadoop/fi
ltered_data.csv
[hadoop@ip-172-31-73-86 ~]$ hadoop fs -getmerge /user/hadoop/filtered_data.csv m
erged.csv
[hadoop@ip-172-31-73-86 ~]$
```

Here we **getmerge** our filtered data and saved as **merged.csv**.

We need to run **getmerge** command as our filtered data is batched. AS Data is often processed in parallel by dividing it into smaller chunks or "batches." This is primarily done to enable parallel processing and efficient storage across multiple nodes in a HDFS.

It's important to understand that Hadoop and Spark, by default, output results into multiple files, one per partition or task. This is again for parallelism and distributed processing.

So why we using **getmerge** command to view of the results, especially for visualization or analysis in Jupyter Notebook, it's convenient to merge the results into a single file.

Now we need to file to the S3 bucket.

And we use next command: **aws s3 cp our_file_name.csv s3://bucket-name**

```
[hadoop@ip-172-31-73-86 ~]$ aws s3 cp merged.csv s3://aws-logs-992382709365-us-east-1/merged.csv
upload: ./merged.csv to s3://aws-logs-992382709365-us-east-1/merged.csv
[hadoop@ip-172-31-73-86 ~]$
```

File Name	Size	Storage Class	Created
merged.csv	7.2 KB	Standard	January 24, 2024, 13:58:27 (UTC-08:00)

As we can see, merged.csv successfully moved to bucket.

Question 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).

Question 7. Plot the number of occurrences of the token (the *frequency* column) of **data** over the years using matplotlib.

Answers for question 6 and 7 will be on Jupyter notebook and will be attached as well for submission.

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.
- b. Explain how the HDFS stores the data.

Q8 a.

Apache Hadoop and Apache Spark are tools for handling big data in analytics. Businesses need to process data quickly and at scale for real-time insights. Hadoop allows you to cluster multiple computers to analyze large datasets in parallel more quickly. Spark, a more advanced tool, speeds up analytic queries with in-memory caching and optimized execution. It even incorporates AI and machine learning. Despite their differences, many companies use both Spark and Hadoop together to achieve their data analytics objectives.

Advantages of Hadoop:

Mature Ecosystem:

Hadoop boasts a mature ecosystem with a range of tools such as MapReduce, Hive, and HBase. This established toolkit provides a comprehensive solution for distributed storage and processing, making it suitable for various big data tasks.

Scalability:

Hadoop's HDFS scales horizontally, meaning it can efficiently handle massive amounts of data by distributing it across multiple nodes. This inherent scalability is crucial for accommodating the ever-growing demands of data storage and processing.

Advantages of Spark:

In-Memory Processing:

Spark leverages in-memory processing, allowing data to be stored and processed in the system's memory rather than on disk. This results in significantly faster data access and computation compared to Hadoop's traditional disk-based processing.

Versatility:

Spark stands out for its versatility, supporting diverse workloads. Whether it's batch processing, machine learning, or real-time stream processing, Spark provides a flexible framework that caters to a wide range of data processing needs.

Q8 b.

How HDFS Stores Data:

HDFS stores data by breaking it into blocks (typically 64 MB or 128 MB in size) and distributing these blocks across multiple nodes in a Hadoop cluster. Each block is replicated across different nodes for fault tolerance. This distributed storage approach allows parallel processing of data, as multiple nodes can work on different blocks simultaneously. The master node, called the NameNode, keeps track of the metadata and the locations of these blocks, facilitating efficient data retrieval and fault recovery in case of node failures.

File System Structure:

- The HDFS file system comprises master services, including the NameNode, secondary NameNode, and DataNodes.

Master Services:

- NameNode and Secondary NameNode: These manage HDFS metadata, tracking file and directory information. The secondary NameNode aids the NameNode by periodically merging and compacting log files to reduce the risk of data loss.
- DataNodes: Host the actual data stored in HDFS, responsible for storage, retrieval, and reading/writing of data.

Data Division:

- HDFS breaks data into blocks, typically sized at 64 MB or 128 MB. Each block is considered a unit for storage and processing.

Replication for Fault Tolerance:

- Each data block is replicated across multiple DataNodes to ensure fault tolerance. The default replication factor is usually three, providing redundancy and reliability.

Parallel Processing:

- The distributed storage approach enables parallel processing, allowing multiple nodes to work simultaneously on different blocks. This enhances the overall efficiency of data processing tasks.

Metadata Management:

- The NameNode keeps track of which DataNodes contain the contents of specific files in HDFS. It manages metadata, including file names, permissions, and block locations.

Replica Distribution:

- The NameNode distributes replicas of data blocks across the Hadoop cluster. This ensures that data is stored redundantly across different nodes.

Access Instructions:

- The NameNode instructs users or applications on where to locate the desired information. It directs them to the specific DataNodes hosting the relevant data blocks.

Efficient Retrieval and Fault Recovery:

- The NameNode's management of metadata and block locations facilitates efficient data retrieval. In case of node failures, the replicated nature of data blocks allows for fault recovery without data loss.