Big Data Analysis Workflow with Google Books Ngrams

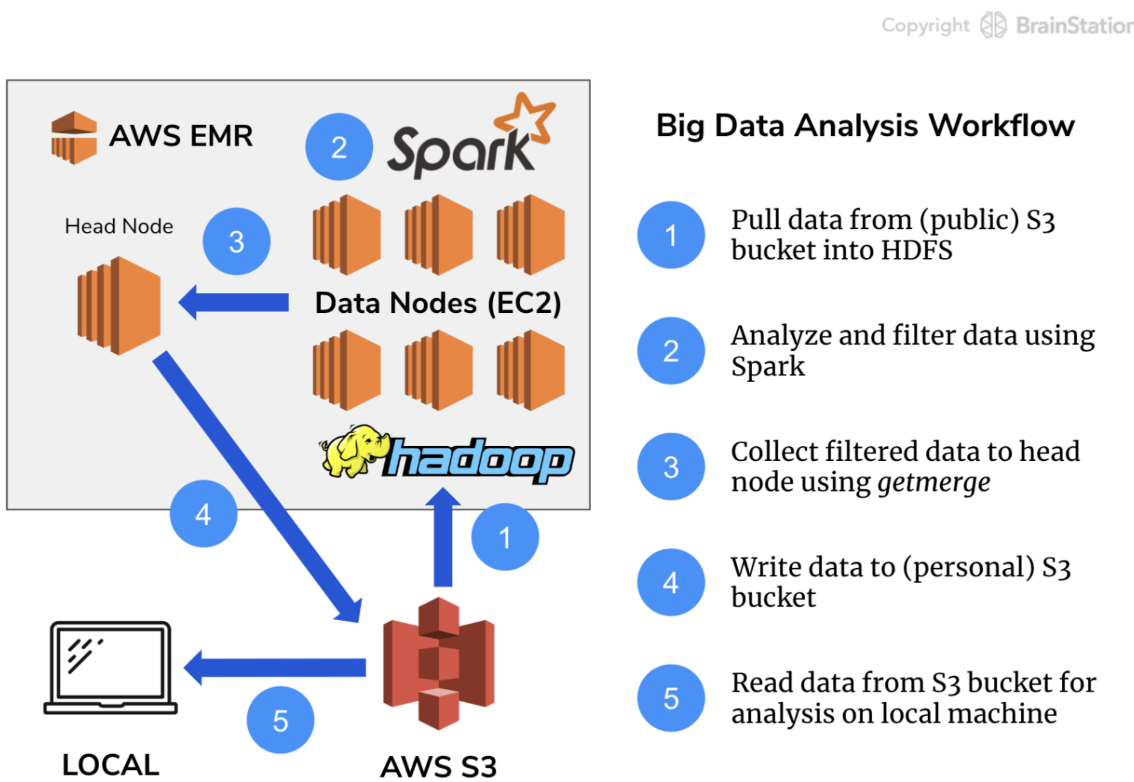
Asli Keser | aslikeser7@gmail.com | LinkedIn | GitHub

In this document, the fundamental skills for working with Big Data such as loading, filtering, and visualizing a large real-world dataset in a cloud-based distributed computing environment using Hadoop, Spark, Hive, and the S3 file system are documented.

The Google Ngrams dataset is used throughout this document 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 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 document, the data is converted to CSV and hosted on a public S3 bucket which may be accessed here: s3://brainstation-dsft/eng_1M_1gram.csv

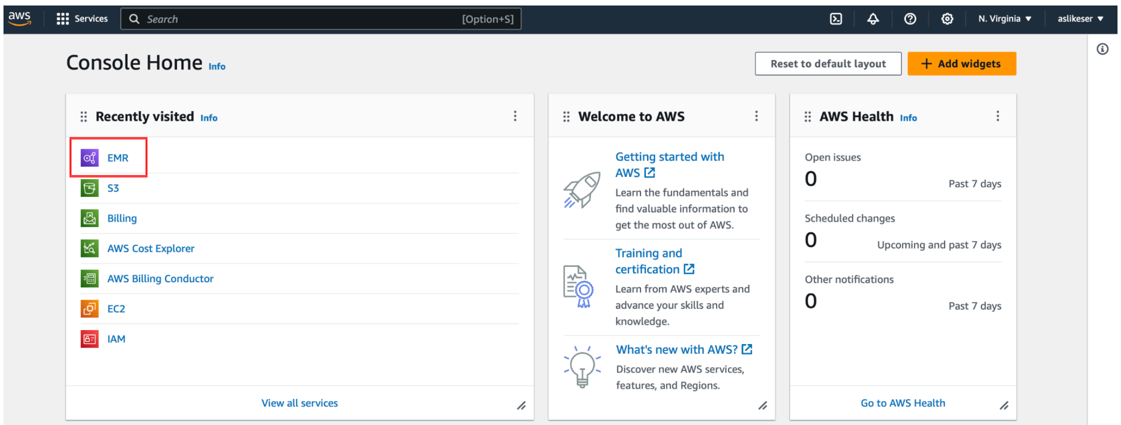
Along with this document, two Jupyter Notebook files will be produced, which will follow a **Big Data analysis workflow**. As part of this workflow, you will **filter and reduce data down to a manageable size** and then **do some analysis locally on our machine after extracting data from the Cloud** and **processing it using Big Data tools**. The workflow and steps in the process are illustrated below:

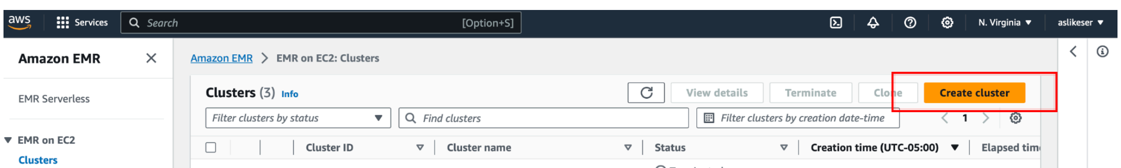


The scope of data processing and analysis is outlined in the questions below. All the steps, commands, and code are documented in detail below.

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

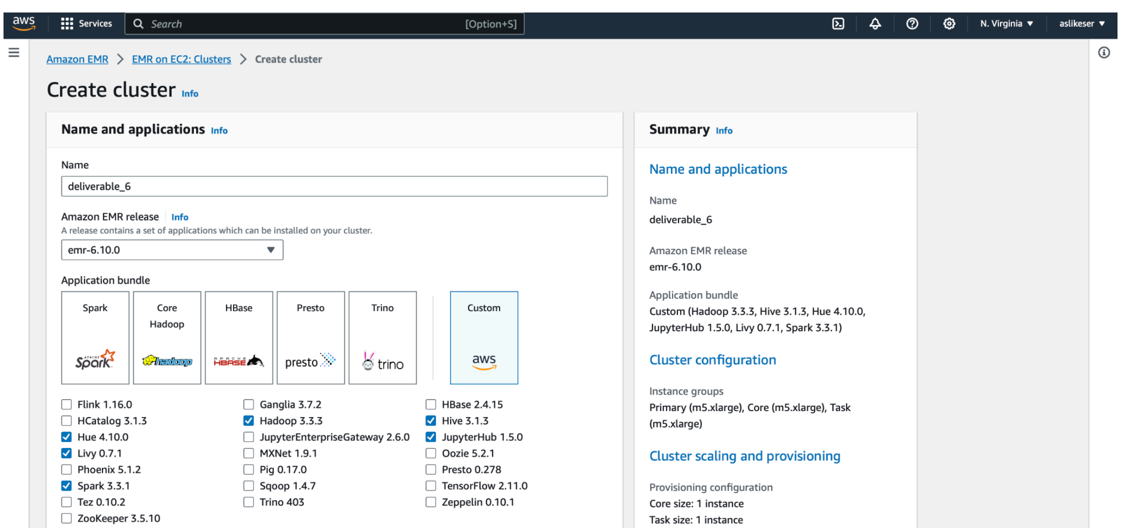
   a. **Create an EMR cluster**





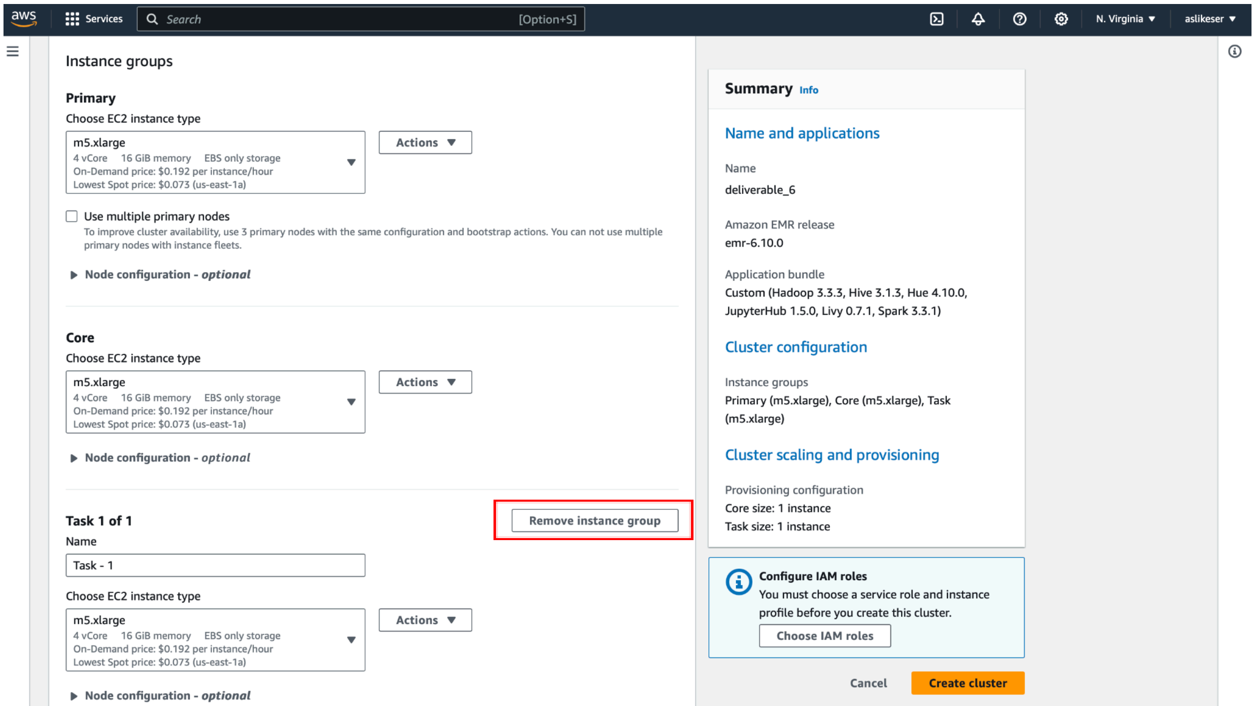
   b. **Cluster name and software**

   Make sure to select the correct EMR release and applications

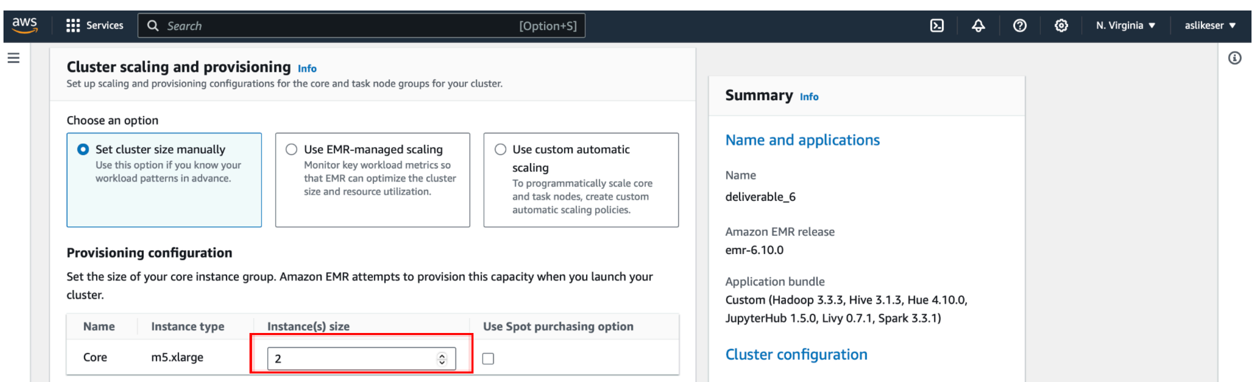


c. Instance Groups

Remove the **task** instance group

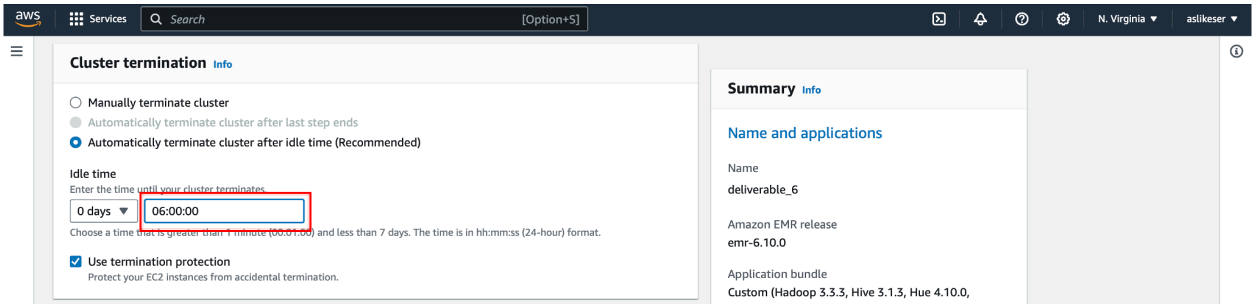


Allocate 2 nodes to the core instance group



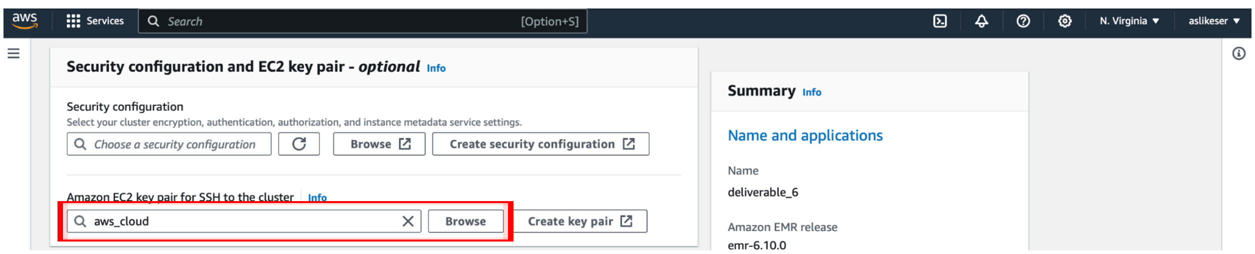
d. Cluster Termination

Set cluster termination to 6 hours.

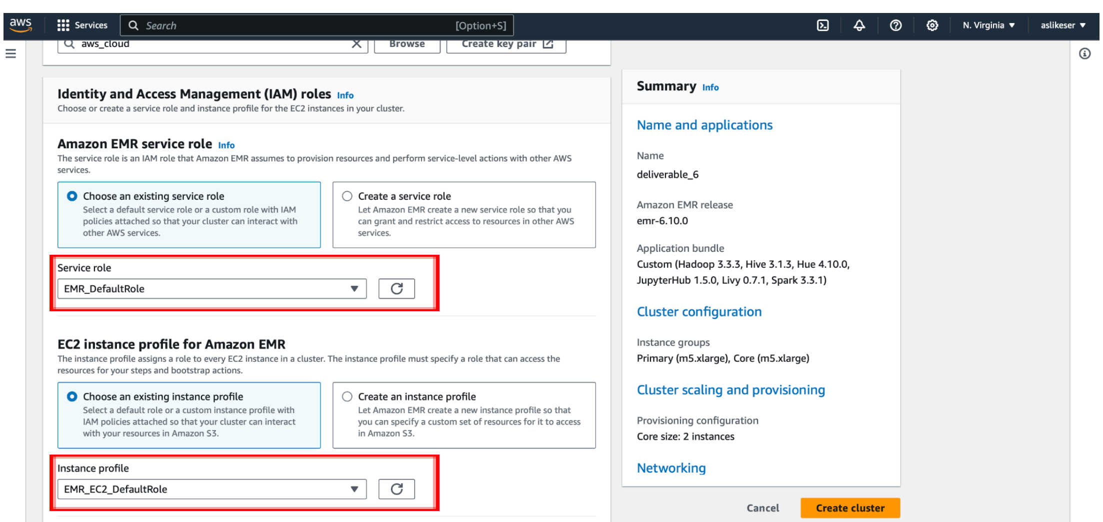


e. Security and Access Management

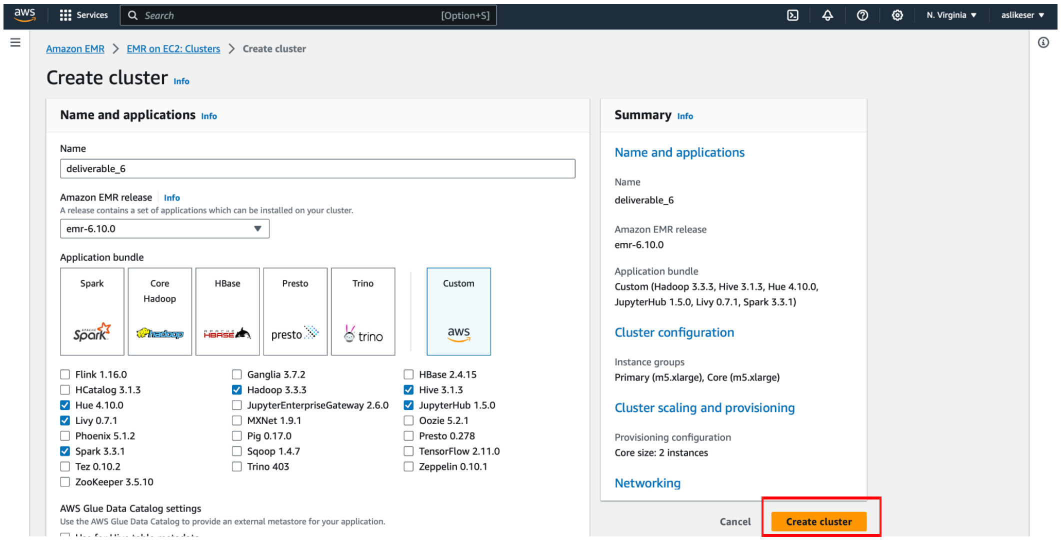
Select the key pair that you created before.



In Identity and Access Management, choose the EMR_DefaultRole and EMR_EC2_DefaultRole.



f. Create Cluster



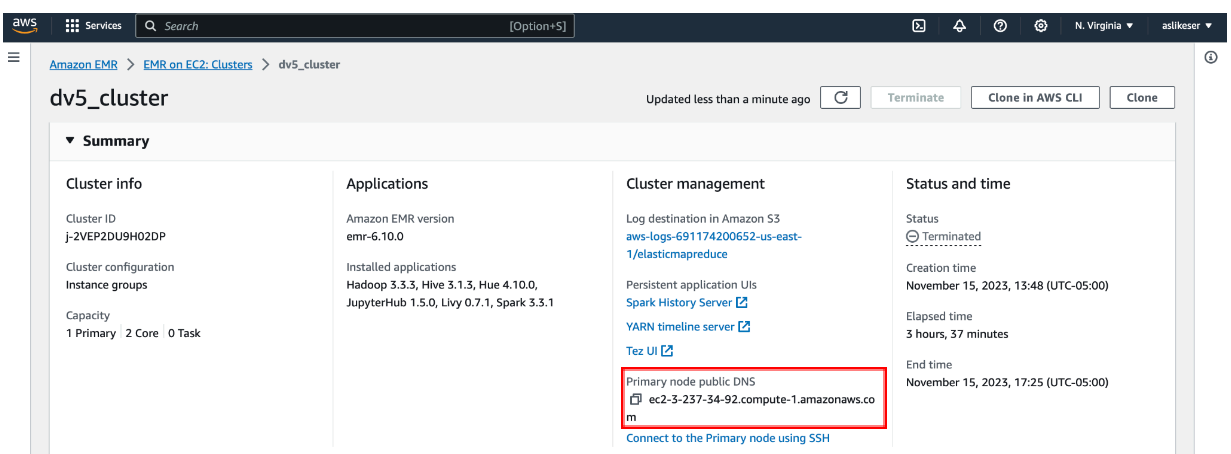
2. Connect to the head node of the cluster using SSH.

Run the below code in your terminal

- the environment you are in should not matter
- Provide your directory to your cloud key

```
ssh -i /Users/Work/Downloads/aws_cloud.pem -L 9995:localhost:9443 hadoop@ec2-3-237-34-92.compute-1.amazonaws.com
```

- 'hadoop@....' part is coming from:



After running the above code successfully, you should be able to see a big EMR sign in your terminal as below.

```
(cloud_lab) Work@Aslis-MBP ~ % ssh -i /Users/Work/Downloads/aws_cloud.pem -L 9995:localhost:9443 hadoop@ec2-3-23
7-34-92.compute-1.amazonaws.com
The authenticity of host 'ec2-3-237-34-92.compute-1.amazonaws.com (3.237.34.92)' can't be established.
ED25519 key fingerprint is SHA256:g3SMvGQvj5XAujhv5yYrQ7JhgOSQc3JMpIl5yF0akcg.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? y
Please type 'yes', 'no' or the fingerprint: yes
Warning: Permanently added 'ec2-3-237-34-92.compute-1.amazonaws.com' (ED25519) to the list of known hosts.

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

https://aws.amazon.com/amazon-linux-2/

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

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.
   - First, make sure you are in HDFS directory '/home/hadoop'
   - If not, run "cd /home/hadoop"
   - Make a directory in the HDFS named 'eng_1M_1gram' by running the below code:

```
mkdir eng_1M_1gram
```

   - check if you successfully put the file into HDFS directory or not by 'ls'
   - You should be able to see the file that you have created.
   - Then, copy the data file from the s3 bucket directly into the HDFS directory that you created by running the below code in your terminal.

```
hadoop distcp s3://brainstation-dsft/eng_1M_1gram.csv
/home/hadoop/eng_1M_1gram/eng_1M_1gram.csv
```

   - Check if you successfully put the file into your HDFS directory by running code below in your terminal.

```
hadoop fs -ls /home/hadoop/eng_1M_1gram
```

- You should be able to see the csv file 'eng_1M_1gram.csv' in the '/home/hadoop/eng_1M_1gram' directory.
- Now, this file is in our hadoop file system meaning that we can use this data in PySpark in the cloud.

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.

If you prefer to use JupyterHub:

- Run the URL below in your internet browser and bypass the security warning.
  https://localhost:9995

- If you are successfully connected, you will be dropped into the JupyterHub login screen. Use below credentials:
  Username: joyvan
  Password: jupyter
- Signing in should forward you to the jupyter page where you can create a new jupyter notebook named "google_ngrams_hadoop" (shared in GitHub)

Within this jupyter notebook, after reading your CSV file from the cloud using PySpark:

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.

See 'google_ngrams_hadoop.ipynb' for the above steps.

- After creating the filtered CSV file and writing it into the cloud 'eng_1M_1gram_filtered.csv', add your filtered CSV file to the same directory as your original file in HDFS by running the below code in your terminal.

```
sudo usermod -a -G hdfsadmingroup livy
```

- Check if you successfully put the file into the HDFS directory by running the code below in your terminal.

```
hadoop fs -ls /home/hadoop/eng_1M_1gram
```

- You should be able to find the two csv files 'eng_1M_1gram.csv' and 'eng_1M_1gram_filtered.csv' in the '/home/hadoop/eng_1M_1gram' directory.

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.

- Collect the contents of filtered data into a single file on the local drive of the head node by running below code in your terminal.

```
hadoop fs -getmerge /home/hadoop/eng_1M_1gram/eng_1M_1gram_filtered.csv eng_1M_1gram_filtered_local.csv
```

- check if you successfully put the file into the HDFS directory by '`ls`'
- You should be able to see the 'eng_1M_1gram_filtered_local.csv' in the local drive of the head node.
- Move this file into an S3 bucket in your account by running the below code in your terminal.

```
aws s3 cp eng_1M_1gram_filtered_local.csv s3://ak-bstn-bucket
```

- You should be able to see this file in your S3 bucket on AWS.

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).

See ‘google_ngrams_local.ipynb’ for this question.

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

See ‘google_ngrams_local.ipynb’ for this 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.

Hadoop and Spark are both frameworks designed for distributed computing.

<u>Differences</u>

1. Required Processing Times

Hadoop is primarily designed for batch processing using the MapReduce programming model. It successfully processes large volumes of data in a distributed manner but lacks performance when it comes to iterative algorithms and interactive processing since all the work is done in the cloud. However, Spark supports both batch processing and iterative algorithms due to its in-memory processing capabilities, making it suitable for iterative workloads, machine learning algorithms, and interactive data analysis compared to Hadoop’s MapReduce.

2. Fault Tolerance

Hadoop deals with fault tolerance through data replication. If a node fails, the data can be retrieved from other nodes. However, Spark implements fault tolerance through lineage information and recomputation. If a partition of data is lost, Spark can recompute it from the original data using lineage information.

Advantages

Hadoop

1. Batch Processing: Hadoop MapReduce is well suited for batch processing tasks and analyzing large datasets in a distributed and fault-tolerant manner. This makes it very convenient for data warehousing.
2. Storage: Hadoop Distributed File System (HDFS) provides a reliable and scalable storage solution. It is designed to store and manage vast amounts of data across multiple nodes in the Hadoop cluster.

Spark

1. In-Memory Processing: One of Spark's key advantages is its ability to perform in-memory processing which allows it to efficiently perform iterative algorithms and interactive processing. Spark can cache intermediate results in memory, reducing the need to read and write to disk between processing stages which significantly speeds up iterative algorithms and interactive data analysis.
2. Ease of Use: Spark provides high-level APIs in Java, Python, R and Scala, making it more accessible to a broader audience. Also,

Spark's built-in libraries for machine learning (MLlib) simplify the development of complex data processing workflows.

b. Explain how the HDFS stores the data.

HDFS breaks down large files into smaller blocks, typically 128 megabytes or 256 megabytes in size. Each block is stored as a separate file on the underlying file system of each node in the Hadoop cluster.

To ensure fault tolerance and data reliability, HDFS replicates each block multiple times across different nodes in the cluster. The default replication factor is 3, meaning each block is stored on 3 different nodes.