Big Data Wrangling With Google Books Ngrams

By: Roman Kovalchuk

Introduction

In this project, we will explore the world of Big Data using Amazon Web Service (AWS) to load, filter and visualize a real world data set in a cloud-based computing environment using Hadoop, Spark, Hive, and the S3 file system. We will be using the "Google Ngrams" dataset, which represents approximately 4% of books ever written between the years 1800 and 2000. The dataset will be pulled from a public S3 bucket, filtered down to a manageable size and analysed locally.

Setting up and connecting to an EMR cluster

We begin by navigating to the EMR section on AWS to start our cluster. The release version will be 6.1.1 and the software configurations include Hadoop, Spark, Jupyterlab, Hive, and Livy. Figure 1 contains the name of the cluster with the emr-6.1.1 release while Figure 2 summarizes the software configurations.

Figure 1: Creating Cluster.

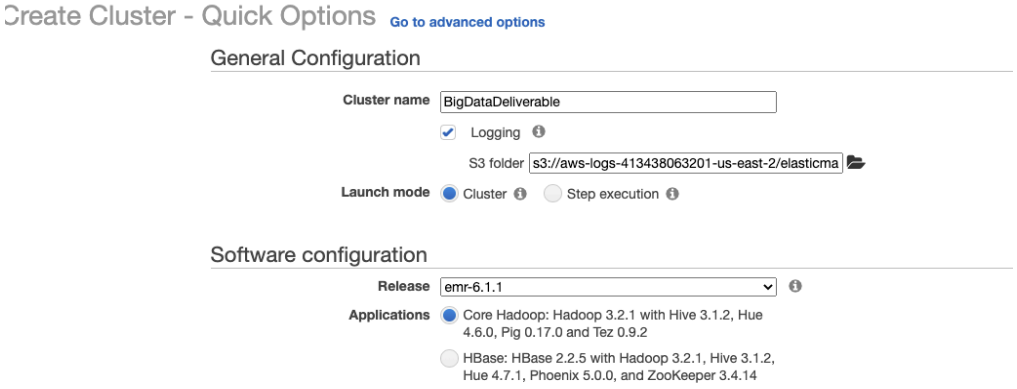
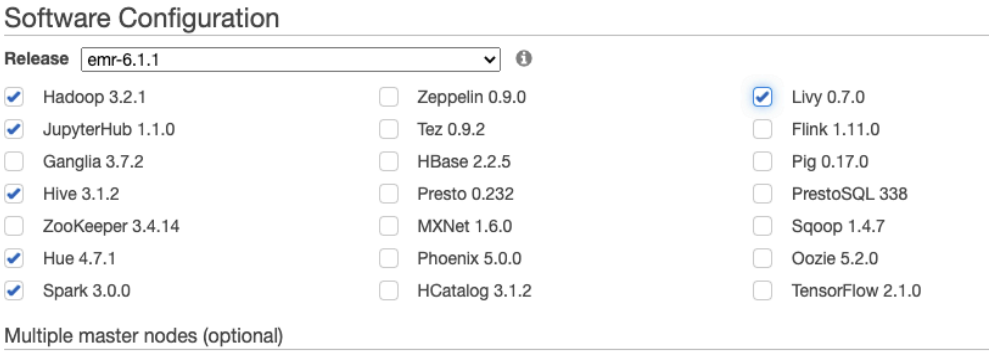


Figure 2: Software Configuration.



Next, we will connect to the head node of the SSH cluster by pairing with an EC2 key and inputting it into terminal. The EC2 key that we will be using is called “aws_cloud” and it’s located in a folder called “aws_lab” on the Desktop. Figure 3 confirms the name of the key while Figure 4 shows the command to input to Terminal. This will connect us to Hadoop.

Figure 3: Selecting the EC2 Key.

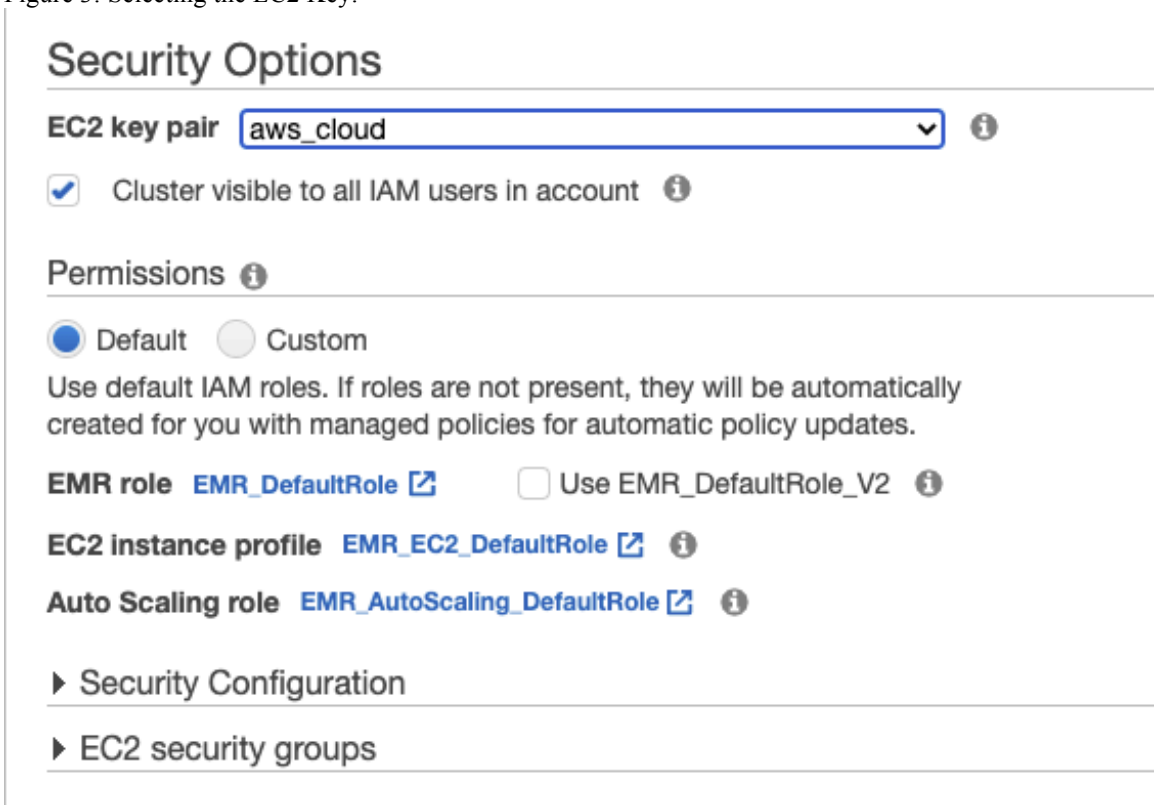
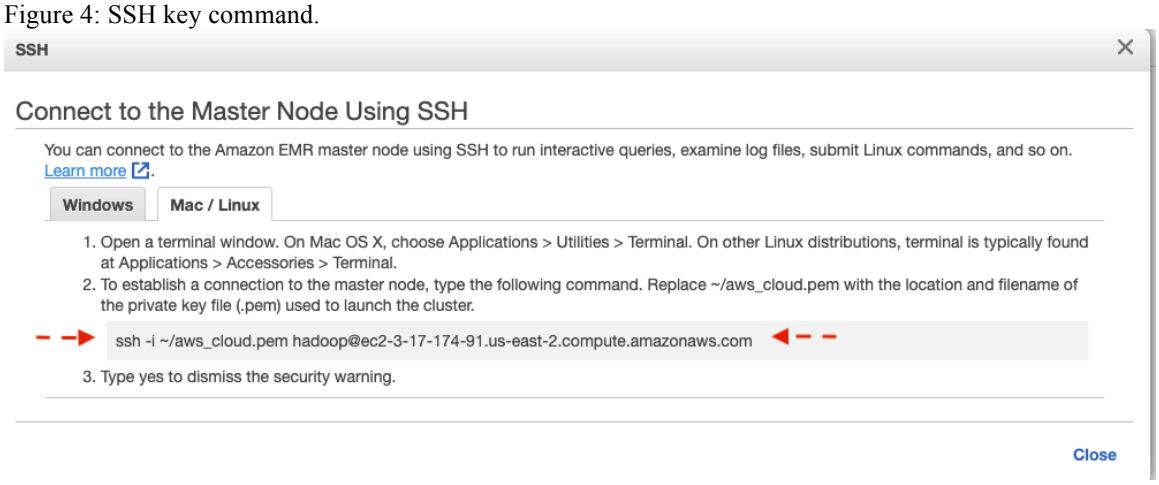
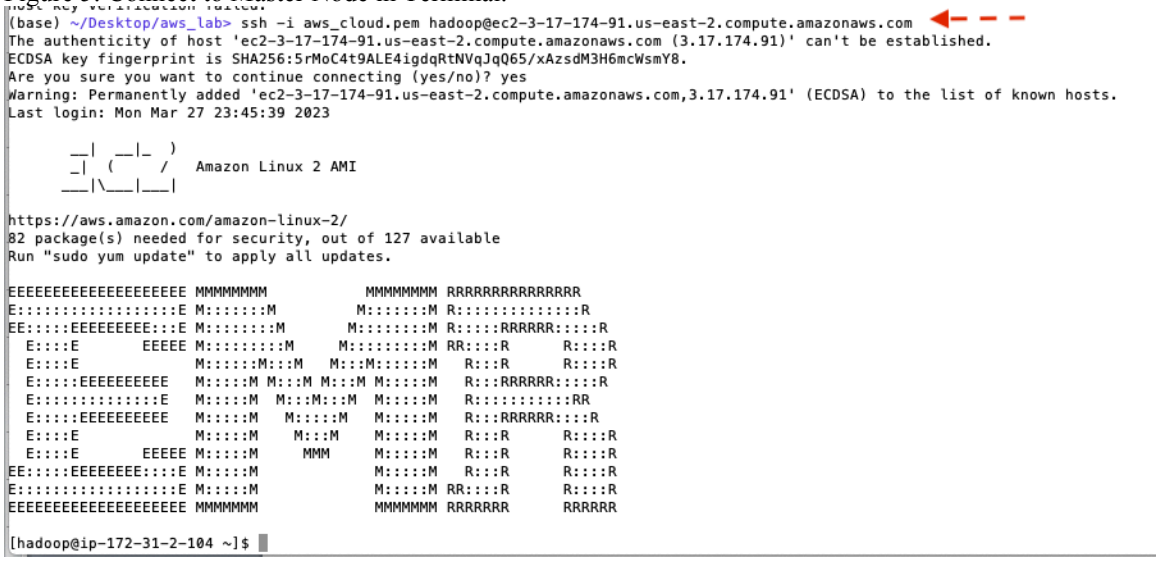


Figure 4: SSH key command.



After copying the SSH address, we will write it into our terminal window. Since we are in the "Desktop/aws_lab" environment, we can remove the "~/" in order to establish a connection. Figure 5 documents this process.

Figure 5: Connect to Master Node in Terminal.

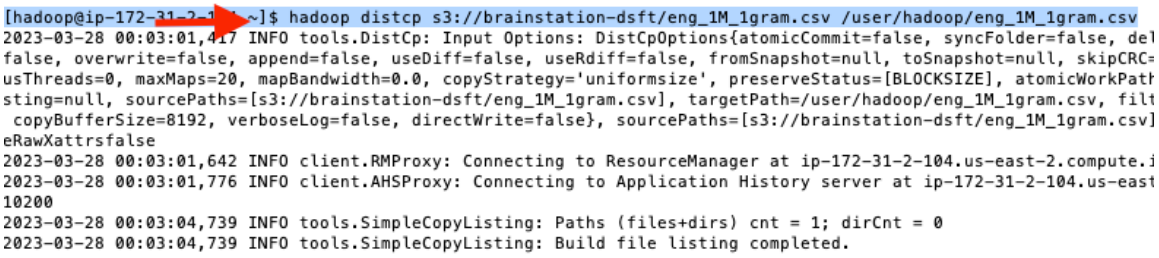


If we do everything correctly, we should see the EMR displayed after running the command. A hadoop@ip- prompt will also appear at the bottom. This confirms our connection to the hadoop framework and allows us to access the S3 bucket.

Accessing the Data

Now we will copy the data folder from the public S3 bucket into a directory on the hadoop framework. In Figure 6, we see the command that allows us to do this. The file is called "eng_1M_1gram" and is in .csv format.

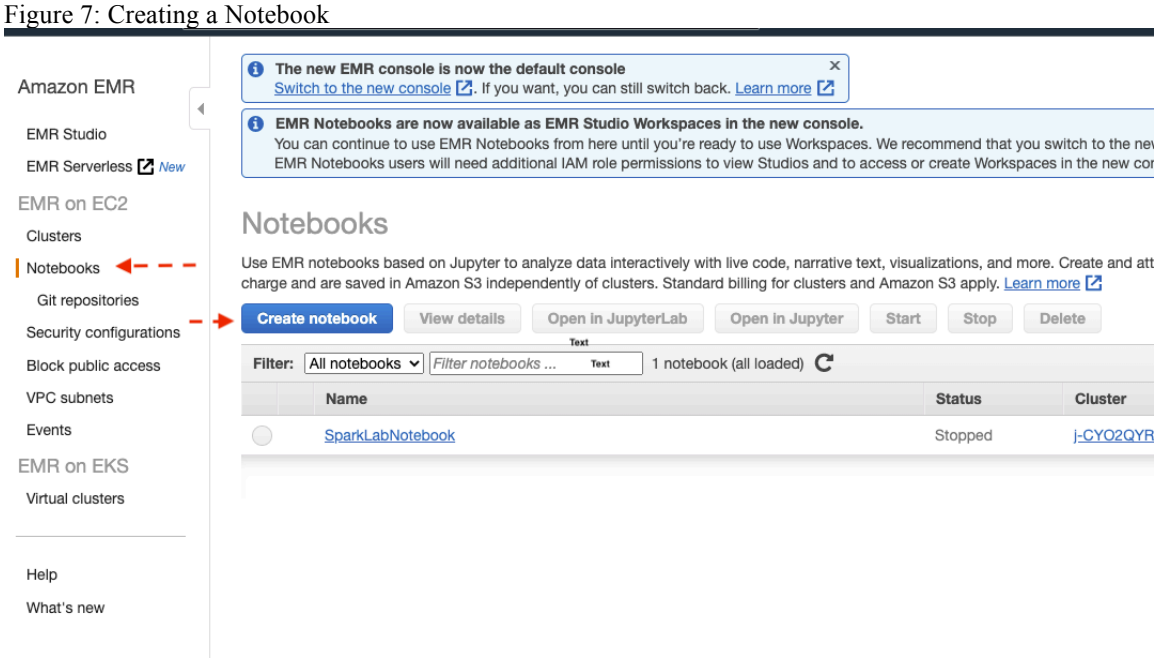
Figure 6: Accessing the data



After this, we navigate to the Notebooks category on the left side of our cluster window and create a notebook that will run this file. The Notebook which we created is called 'SparkLabNotebook" and can be seen in Figure 7. In this Notebook, we start the Spark application with the "spark" command. We need to use the PySpark kernel because it is better suited to working with large datasets, as in this case the dataset that we are working with contains over 260 million rows. After exploring the data, we create a SQL view with the .createOrReplaceTempView command and run a query to filter only the rows that

contain the word “data”. The query is SELECT * FROM eng_1m_1gram WHERE token = “data”, leaving us with 316 rows. Refer to “Notebook_1_Question_4” to see the contents of the notebook.

Figure 7: Creating a Notebook



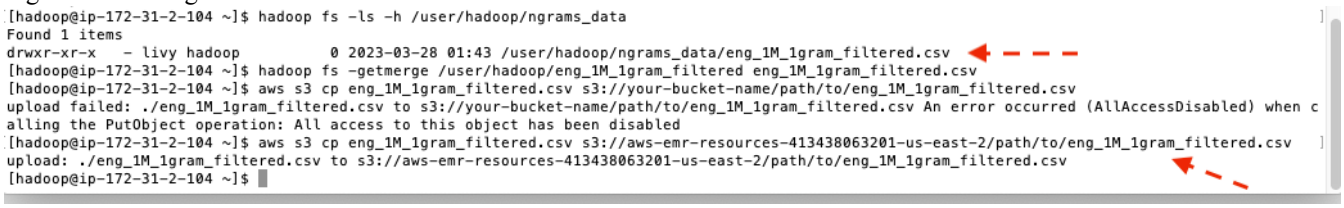
When we save the filtered file in the notebook, we set a path and save it to a folder that we called “ngrams_data”. The complete path is: "hdfs:///user/hadoop/ngrams_data/eng_1M_1gram_filtered.csv". We pass the “header=True” parameter to ensure the header is included in the written file. We can then check to see if the file was saved in our hadoop directory by running the command in Figure 8.

Figure 8: Checking if the file was saved

```
[hadoop@ip-172-31-2-104 ~]$ ssh -i ~/aws_cloud.pem -ND 8157 hadoop@ec2-3-17-174-91.us-east-2.compute.amazonaws.com
Warning: Identity file /home/hadoop/aws_cloud.pem not accessible: No such file or directory.
The authenticity of host 'ec2-3-17-174-91.us-east-2.compute.amazonaws.com (172.31.2.104)' can't be established.
ECDSA key fingerprint is SHA256:5rMoC4t9ALE4igdqRtNVqJqQ65/xAzsdM3H6mcWsmY8.
ECDSA key fingerprint is MD5:87:48:60:4c:c1:cc:1e:ba:05:93:65:7b:71:06:c7:40.
Are you sure you want to continue connecting (yes/no)? no
Host key verification failed.
[hadoop@ip-172-31-2-104 ~]$ ssh -i aws_cloud.pem -ND 8157 hadoop@ec2-3-17-174-91.us-east-2.compute.amazonaws.com
Warning: Identity file aws_cloud.pem not accessible: No such file or directory.
The authenticity of host 'ec2-3-17-174-91.us-east-2.compute.amazonaws.com (172.31.2.104)' can't be established.
ECDSA key fingerprint is SHA256:5rMoC4t9ALE4igdqRtNVqJqQ65/xAzsdM3H6mcWsmY8.
ECDSA key fingerprint is MD5:87:48:60:4c:c1:cc:1e:ba:05:93:65:7b:71:06:c7:40.
Are you sure you want to continue connecting (yes/no)?
Host key verification failed.
[hadoop@ip-172-31-2-104 ~]$ hadoop fs -ls -h /user/hadoop/ngrams_data
Found 1 items
drwxr-xr-x   - livy hadoop          0 2023-03-28 01:43 /user/hadoop/ngrams_data/eng_1M_1gram_filtered.csv
[hadoop@ip-172-31-2-104 ~]$
```

In this case, running the command returns the filtered file inside the folder where we saved it. Now we can use the “-getmerge” command to merge the contents of the directory into a single file on the local drive of the head note. After that we move the merged file into an S3 bucket on our account. The two commands are summarized in Figure 9.

Figure 9. Moving the file into local bucket.



In this case, we moved the filtered file into our s3 bucket titled “aws-emr-resources-…”.

In order to read the file on our local machine, we need to use boto3, which allows us to create AWS resources in our script. To do this, we set up a client to access the S3 bucket and then set the bucket name and file path. Afterwards, we can set a get_object command and then pass the resulting variable into the panda’s “read_csv” command. The complete process can be seen in the “Notebook_2_Question_6_And_7” attached notebook. After further cleaning of the filtered data, we convert the year, frequency, pages, and books columns to float variable and then plot the occurrences of the word “data”. Over the years we see a steep curve that peaks in the late 90s. The increase may be due to the fact that the use of data has become more prevalent in various fields such as science, technology, and business.

Hadoop and Spark as distributed file systems

Hadoop and Spark are both distributed computing frameworks used for processing large amounts of data but they differ in several ways.

Advantages of Hadoop:

- Hadoop can scale to handle large volumes of data and can handle multiple concurrent tasks by leveraging the Hadoop Distributed File System.
- Hadoop runs on commodity hardware and provides a cost-effective solution for storing and processing big data.

Advantages of Spark:

- Spark is known for its speed when processing large amounts of data. It can perform in-memory processing, which means that it can keep data in memory rather than writing to disk. This results in faster processing times.
- Spark provides a simpler API than Hadoop, making it easier for developers to write applications. Spark also supports Java, Python, and Scala which makes it more accessible to a wider range of developers

The Hadoop Distributed File System (HDFS) stores data across multiple machines in a distributed fashion by dividing large files into blocks and distributing them among different machines in the cluster. The blocks are typically 128 or 256 MB in size and are replicated across multiple machines, with each block stored on at least 3 different machines to ensure availability in the event of machine failure. One machine is designated as the NameMode, which maintains metadata about where data is located while other machines are DataNodes, which store the data blocks. When a client wants to read or write a file, it first communicates with the NameMode to obtain the locations of the data blocks. The client then directly accesses the DataNodes that store the blocks, allowing for parallel processing and efficient data retrieval. New data will be automatically split into blocks and distributed across the cluster. If a file is deleted, the blocks are deleted from all of the DataNodes.

Conclusion

In this project, we've loaded, filtered, and visualized a large dataset in a cloud-based distributed environment using Hadoop, Spark, and the S3 filesystem. We filtered a dataset from 360 million rows down to 316 with a single SQL query, moved the file into a bucket and then opened it on our local machine to further visualize the relationship between the frequencies of the word "data" over the years. We compared the advantages of both Hadoop and Spark and summarized why HFDS is a robust file storage system. The screenshots included in the report should help the reader explore the world of Big Data where when it comes to size, the sky is the limit.