Big Data Wrangling With Google Books Ngrams

Using AWS EMR, HDFS, Spark, and S3

Ethan Shelburne
BrainStation: The Digital Learning Company

August 25, 2025

Contents

List of Figures

1 Introduction

1.1 Executive Summary

This project applies a big–data workflow to the Google Books Ngrams corpus using AWS EMR. We provision an EMR cluster (Spark + HDFS + JupyterHub), connect to the primary node via SSH (and an SSH tunnel for JupyterHub), copy the public CSV from S3 into HDFS, and load it in PySpark. We validate the load (schema, row/column counts, quick ranges), register a temporary view, and run Spark SQL to filter the dataset to the token `"data"`. The filtered result is written back to HDFS as CSV with `header=True`, merged to a single local file with `getmerge`, and uploaded to a personal S3 bucket. Finally, on a local machine we read the S3 object into pandas, save a local copy, and visualize trends over multiple year windows (full range vs. consecutive-year subsets), noting peaks in the 1980s/1990s and a decline in the early 2000s. All steps are fully documented with commands and screenshots for reproducibility. Additionally, we compare and contrast Hadoop and Spark in Section 3.

1.2 Dataset Overview

The analysis uses the English 1-gram slice of Google Books Ngrams. Each row represents a single 1-gram token in a given year with four fields: `token` (string), `year` (integer), `frequency` (total occurrences across all books that year), `pages` (distinct pages containing the token), and `books` (distinct volumes containing the token). The series spans from the 1500s through 2008 with sparser coverage in the earlier years. The unfiltered data set includes $261,823,225$ records.

2 Reproducible Steps & Command Log

Note on placeholders. Replace values in `<angle brackets>` with your own environment:

- SSH key path: `<your_key_path>` (e.g., `/c/Users/you/Desktop/.../key.pem` on Git Bash)
- Your S3 bucket/prefix: `s3://<your_bucket>/ngrams/`

2.1 Spin up EMR cluster

Create an EMR 6.x cluster with Hadoop, Spark, and JupyterHub. Screenshots and specifications are documented in Appendix A.

2.2 Connect to the head node (SSH)

Open gitbash, and run the following command with the path to your key, and the public DNS of your EMR primary node (which can be copied from your cluster info on AWS under "Connect to the Primary node using SSH"). This connects you to the head node of your cluster.

```
ssh -i "<your_key_path>" hadoop@ec2-18-191-151-44.us-east-2.compute.amazonaws.com
```

2.3 Create HDFS target directory

Create a target HDFS directory with the following command.

```
hdfs dfs -mkdir -p /user/hadoop/eng_1M_1gram
```

2.4 Copy public CSV from S3 to HDFS

Use the following command to copy the the target CSV from the public S3 bucket into the HDFS directory which was just created.

```
hadoop fs -cp s3://brainstation-dsft/eng_1M_1gram.csv hdfs:///user/hadoop/
    eng_1M_1gram/
```

2.5 Verify HDFS ingest

Confirm the CSV file is now in the HDFS directory by listing the directory's contents, then printing the head of the dataset with the following commands.

```
hdfs dfs -ls -h /user/hadoop/eng_1M_1gram
hdfs dfs -cat /user/hadoop/eng_1M_1gram/eng_1M_1gram.csv | head -n 5
```

2.6 Open a JupyterHub tunnel

After exiting the previous ssh session, use gitbash to open a tunnel to your primary node with the following command.

```
ssh -i "<your_key_path>" \
   -L 9443:localhost:9443 \
   hadoop@ec2-18-191-151-44.us-east-2.compute.amazonaws.com
```

In your browser, open **https://localhost:9443/** and log in with the default credentials (user `jovyan`, password `jupyter`). In a PySpark notebook, read the data from

`hdfs:///user/hadoop/eng_1M_1gram/eng_1M_1gram.csv`,

inspect the structure and schema of the data, filter with `token='data'` via Spark SQL, and write the CSV file to

`hdfs:///user/hadoop/outputs/data_token_csv`

with `header=True`. See all the details of the reading, filtering, and writing process in the notebook `big_data_wrangling_on_cluster.ipynb`.

2.7 Confirm the filtered write in HDFS

Use the following commands to verify the filtered CSV was successfully written back into HDFS (by checking the directory and printing the head of the data).

```
hdfs dfs -ls -h /user/hadoop/outputs
hdfs dfs -cat /user/hadoop/outputs/data_token_csv/part-*.csv | head -n 10
```

2.8 Merge to a single local CSV on the head node

Next, use the following commands to collect the contents of `data_token_csv` into a single file on the local drive of the head node.

```
# Create a local folder
mkdir -p ~/ngrams_outputs

# Merge the directory into one local file
hdfs dfs -getmerge /user/hadoop/outputs/data_token_csv/ ~/ngrams_outputs/data_token.
    csv

# Confirm data was correctly copied to the local drive of the head node
head -n 5 ~/ngrams_outputs/data_token.csv
```

2.9 Upload merged CSV to your S3 bucket

Move the CSV file from the local drive of the head node into a S3 bucket in your account using the following commands.

```
# Upload the CSV to your S3 bucket
aws s3 cp ~/ngrams_outputs/data_token.csv s3://<your_bucket>/ngrams/data_token.csv

# Verify it now exists in your bucket
aws s3 ls s3://<your_bucket>/ngrams/
```

Once, the file has been successfully copied into your bucket, terminate your cluster.

2.10 Local machine: load & plot with pandas

The following commands configure your local AWS CLI credentials and install the Python package needed to read data into pandas from S3 buckets.

```
# Configure AWS CLI credentials
aws configure
# AWS Access Key ID: <your_access_key_id>
# AWS Secret Access Key: <your_secret_access_key>
# Default region name: us-east-2
# Default output format: json

# Install required packages for local analysis
pip install s3fs
```

Next, load the CSV from your S3 bucket and produce the plots in your local notebook; see the notebook `big_data_wrangling_on_local.ipynb` for the full code and outputs.

3 Hadoop, HDFS, and Spark

3.1 Hadoop vs. Spark

Hadoop MapReduce expresses jobs as a sequence of map and reduce phases. Each phase writes intermediate results to disk, which makes it robust but incurs significant resource cost for iterative workloads. Spark uses a DAG (Directed Acyclic Graph) scheduler over Resilient Distributed Datasets (RDDs), which are immutable, partitioned collections of records that enable fault-tolerant parallel operations. Stages are pipelined, and intermediate results are cached in memory (spilling to disk as needed). This design dramatically reduces repeated disk load and lowers end-to-end cost for many workloads.

Both run at cluster scale and integrate with HDFS, S3, and YARN. Hadoop MapReduce remains strong for very large batch ETL where jobs are straightforward and fault tolerance via re-execution is acceptable. Spark is preferred for iterative algorithms (machine learning, network science, queries with many joins), ad-hoc analysis, streaming, and interactive notebooks.

Two advantages of Hadoop (MapReduce).

- **Simplicity and durability for batch.** The disk-centric map/reduce stages are simple, restartable, and well-suited to massive ETL.
- **Mature integration with HDFS/YARN.** Decades of operational experience, strong data locality with HDFS, and a broad set of Hadoop ecosystem tools.

Two advantages of Spark.

- **Performance for iterative/interactive jobs.** In-memory caching and a DAG optimizer expedites iterative jobs (like machine learning algorithms) compared to the disk-bound MapReduce.
- **Unified high-level APIs.** Spark DataFrames/Spark SQL, MLlib, and Structured Streaming provide a single programming model across batch, SQL, ML, streaming, and other big data tasks.

3.2 HDFS as a Storage System

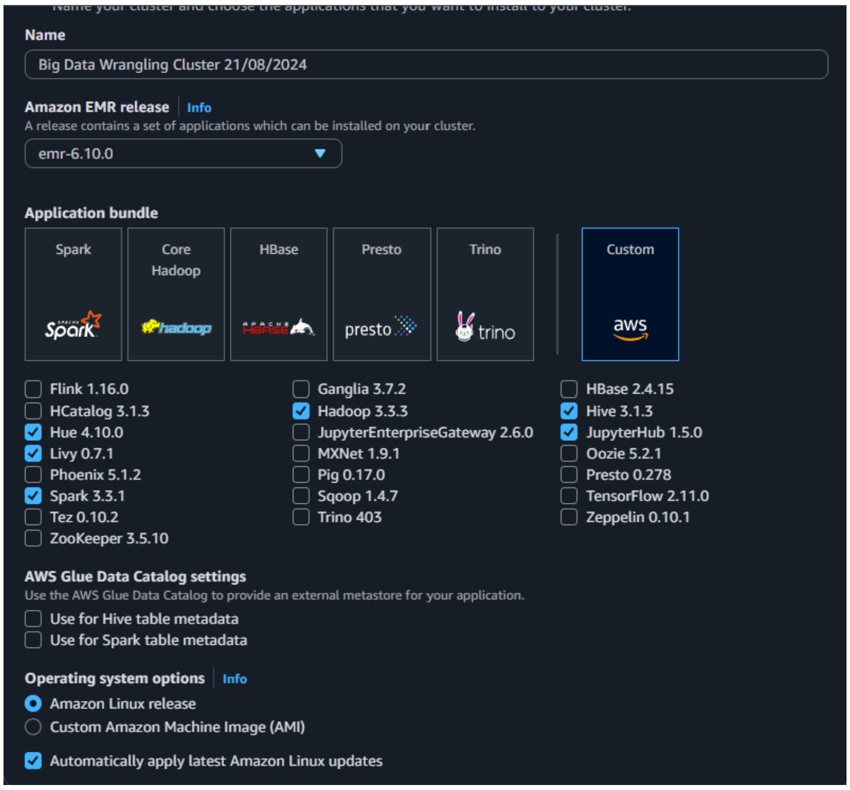
HDFS stores files by splitting them into large fixed-size blocks (128 MB by default in Hadoop 3). Each block is replicated across multiple DataNodes (default replication factor is 3). A central **NameNode** maintains the filesystem namespace and block metadata (which blocks belong to which file, and where those blocks live), while **DataNodes** store the actual block bytes.

To create a file, a client asks the NameNode to allocate a path and block targets. The client then streams data to a replication pipeline of DataNodes; acknowledgements flow back up the pipeline only after each packet is written to each replica. HDFS follows a single-writer, multiple-reader model with write-once (append-supported) semantics, which simplifies consistency and recovery. For reads, the client requests block locations from the NameNode and then fetches the closest replica. This design exploits data locality: compute tasks are scheduled where the data already resides to reduce network expense. DataNodes send block reports to the NameNode. If a DataNode fails or a replica falls below the target replication factor, HDFS automatically re-replicates blocks to healthy nodes. Tools such as the balancer help maintain even distribution and detect inconsistencies.

In summary, HDFS is a distributed filesystem designed to live inside the compute cluster. That tight coupling enables locality-aware scheduling and high throughput for large sequential reads and writes.

A Screenshots for Cluster Setup

Spin up an EMR cluster with emr-6.10.0 and the following custom application bundle.



Name your cluster and choose the applications that you want to install to your cluster.

Name

Big Data Wrangling Cluster 21/08/2024

Amazon EMR release | Info

A release contains a set of applications which can be installed on your cluster.

emr-6.10.0

Application bundle

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

AWS Glue Data Catalog settings

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

☐ Use for Hive table metadata
☐ Use for Spark table metadata

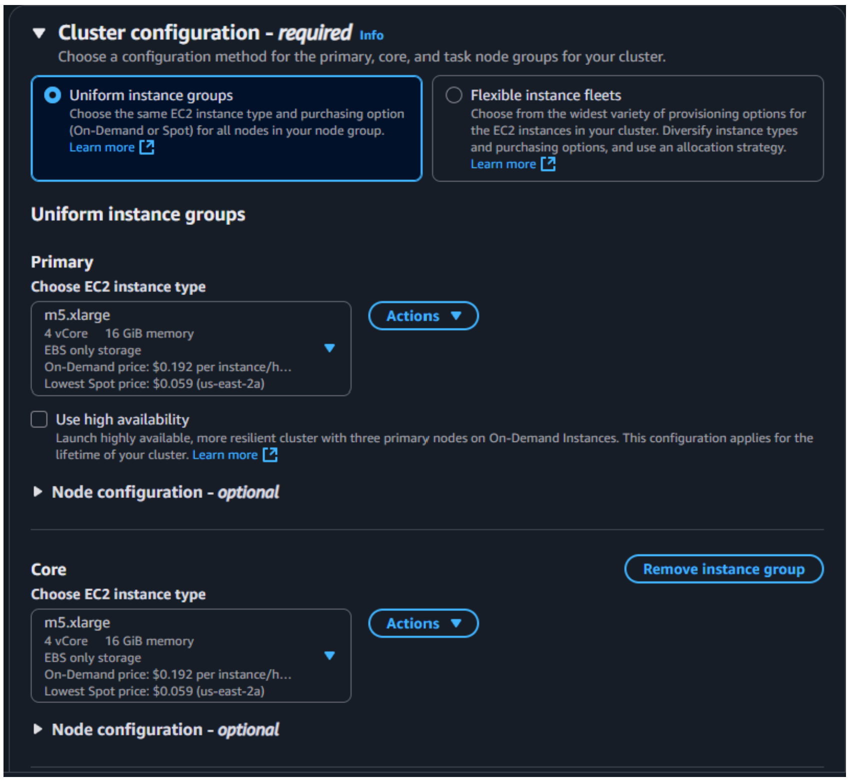
Operating system options | Info

● Amazon Linux release
○ Custom Amazon Machine Image (AMI)

☑ Automatically apply latest Amazon Linux updates

Figure 1: Custom application bundle

Remove the “Task” instance group and set the other instance groups (“Primary” and “Core”) to `m5.xlarge` types.



Cluster configuration - *required* Info

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

● Uniform instance groups
Choose the same EC2 instance type and purchasing option (On-Demand or Spot) for all nodes in your node group.
Learn more

○ Flexible instance fleets
Choose from the widest variety of provisioning options for the EC2 instances in your cluster. Diversify instance types and purchasing options, and use an allocation strategy.
Learn more

Uniform instance groups

Primary

Choose EC2 instance type

m5.xlarge
4 vCore 16 GiB memory
EBS only storage
On-Demand price: $0.192 per instance/h...
Lowest Spot price: $0.059 (us-east-2a)

☐ Use high availability
Launch highly available, more resilient cluster with three primary nodes on On-Demand Instances. This configuration applies for the lifetime of your cluster. Learn more

▶ Node configuration - *optional*

Core

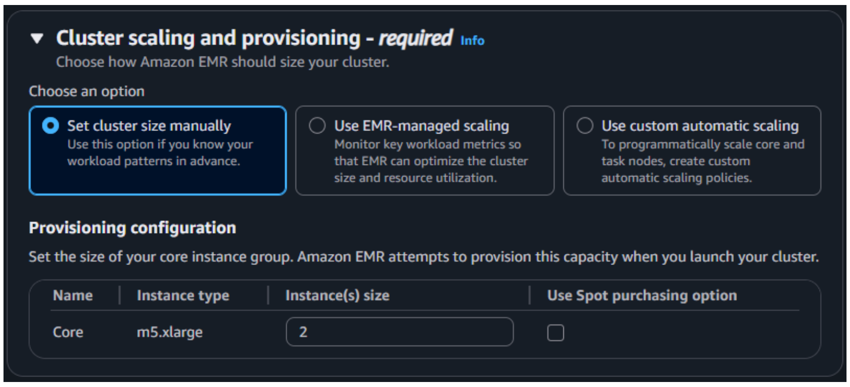
Choose EC2 instance type

m5.xlarge
4 vCore 16 GiB memory
EBS only storage
On-Demand price: $0.192 per instance/h...
Lowest Spot price: $0.059 (us-east-2a)

▶ Node configuration - *optional*

Figure 2: Configure uniform instance groups

Set the cluster size manually so that the "Core" instance count is 2.



Cluster scaling and provisioning - *required* Info

Choose how Amazon EMR should size 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.

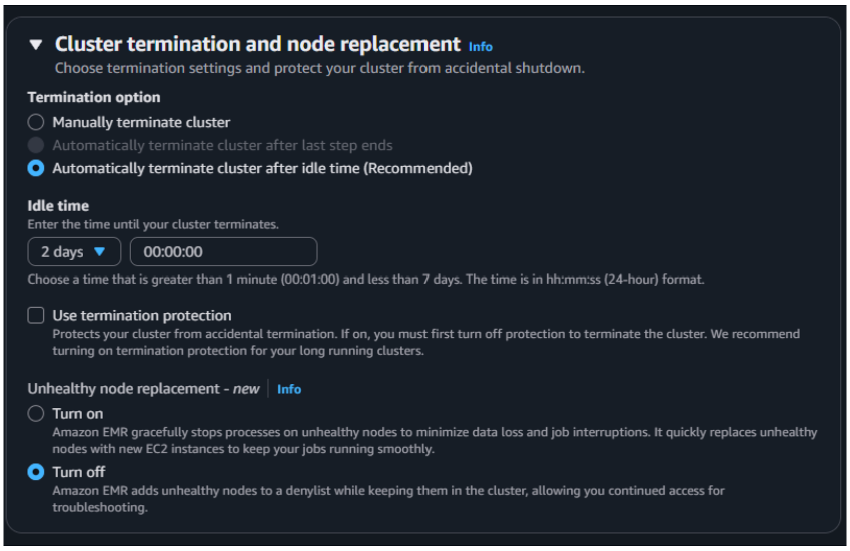
Provisioning configuration

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	☐

Figure 3: Set "Core" instance count to 2

Set auto-termination after 2 days of idle time. We chose to set this metric to 2 days to ensure the cluster would not be accidentally terminated (which could lead to lost work). The steps in Section 2 should take much less time than this so it is essential to manually terminate the cluster when all steps are complete.



Cluster termination and node replacement Info

Choose termination settings and protect your cluster from accidental shutdown.

Termination option

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.

2 days 00:00:00

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

Protects your cluster from accidental termination. If on, you must first turn off protection to terminate the cluster. We recommend turning on termination protection for your long running clusters.

Unhealthy node replacement - *new* Info

Turn on

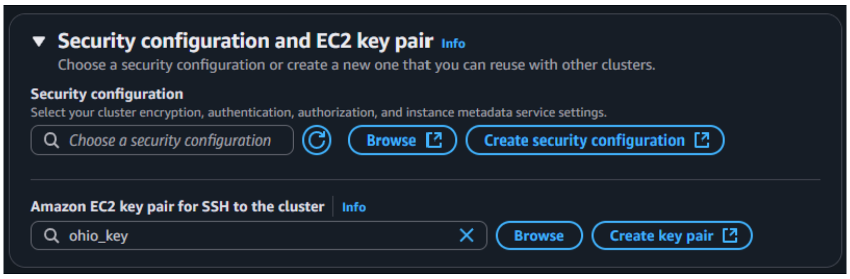
Amazon EMR gracefully stops processes on unhealthy nodes to minimize data loss and job interruptions. It quickly replaces unhealthy nodes with new EC2 instances to keep your jobs running smoothly.

[✓ Turn off]

Amazon EMR adds unhealthy nodes to a denylist while keeping them in the cluster, allowing you continued access for troubleshooting.

Figure 4: Specify cluster termination options

Under the security configuration settings, select your EC2 key pair (ensuring you are in the correct AWS geography). The path to this key pair will be used throughout Section 2.



Security configuration and EC2 key pair Info

Choose a security configuration or create a new one that you can reuse with other clusters.

Security configuration

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

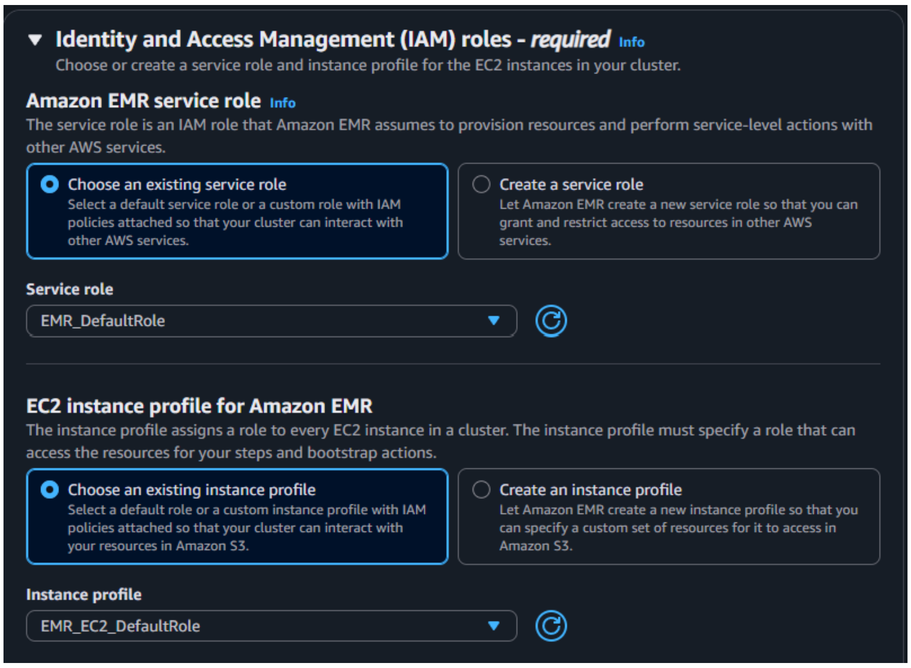
Choose a security configuration

Amazon EC2 key pair for SSH to the cluster Info

ohio_key

Figure 5: Select EC2 key pair

In the “Identity and Access Management” section, select `EMR_DefaultRole` and `EMR_EC2_DefaultRole` under “Amazon EMR service role” and “EC2 instance profile for Amazon EMR” respecitvely.



Identity and Access Management (IAM) roles - *required* 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: EMR_DefaultRole

EC2 instance profile for Amazon EMR

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

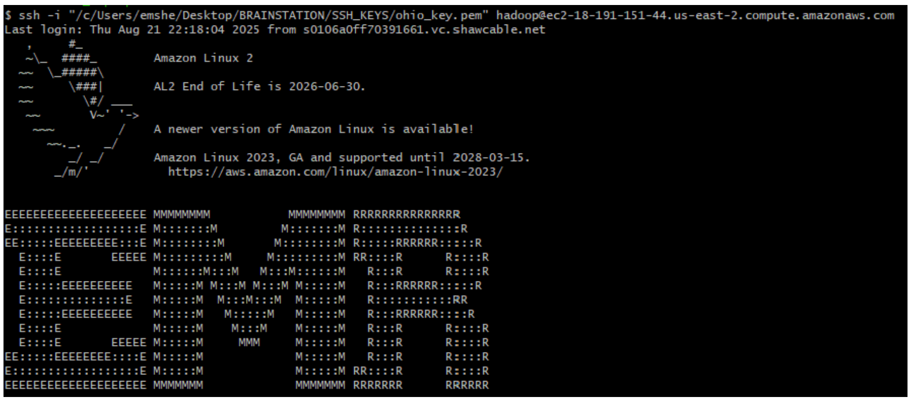
[✓ Choose an existing instance profile] Select a default role or a custom instance profile with IAM policies attached so that your cluster can interact with your resources in Amazon S3.

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

Instance profile: EMR_EC2_DefaultRole

Figure 6: Configure IAM Roles

Click on “Create cluster” to finish the set up. When provisioning is finished and the EMR instance is ready, the cluster state should be “Waiting.” At this point, run the command from Section 2.2 in GitBash to connect to the head node of the cluster and get the following message.



```
$ ssh -i "/c/Users/emshe/Desktop/BRAINSTATION/SSH_KEYS/ohio_key.pem" hadoop@ec2-18-191-151-44.us-east-2.compute.amazonaws.com
Last login: Thu Aug 21 22:18:04 2025 from s0106a0ff70391661.vc.shawcable.net
   ,     #_
   ~\_  ####_        Amazon Linux 2
  ~~  \_#####\
  ~~     \###|       AL2 End of Life is 2026-06-30.
  ~~       \#/ ___
   ~~       V~' '->
    ~~~         /    A newer version of Amazon Linux is available!
      ~~._.   _/
         _/ _/       Amazon Linux 2023, GA and supported until 2028-03-15.
       _/m/'           https://aws.amazon.com/linux/amazon-linux-2023/

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
```

Figure 7: SSH to primary node