Big Data Wrangling With Google Books Ngrams

Benedikt Middelstaedt - 21/06/2024 - BrainStation Data Science & AI Bootcamp

Contents

Introduction 2
EMR-Cluster Setup 2
Connecting to the Head node 7
We have successfully established a connection to the head node in the EMR! 10
Copying the data 10
Working with the data 11
Access to the edited data 11
Comparison Hadoop and Spark 12
Appendix 13

Introduction

The following is a step-by-step guide to working with large data sets in an AWS environment. The source data are the Google Ngrams data records. After setting up the AWS environment, these are filtered according to certain parameters and converted into a format that allows you to work with this data on a local device. The basic approach is as follows:

1. setup of the AWS environment
2. establishing a connection with the AWS environment
3. access to the Ngrams data set
4. filtering of the Ngrams data set
5. storage of the filtered data set
6. processing of the filtered data set

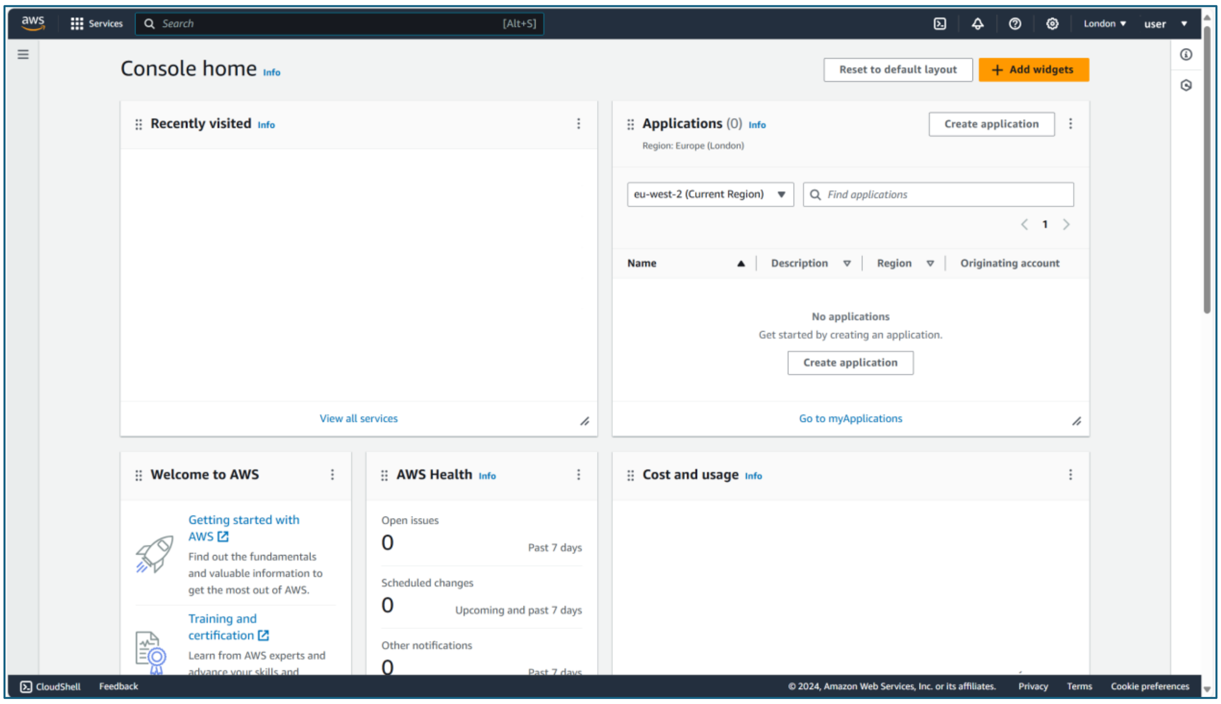
The work is aimed at an interested beginner who has no prior knowledge of big data but has a general understanding of data science.

EMR-Cluster Setup

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

Firstly, the user must have an Amazon Web Service (AWS) account. Creating one is free of charge and can be done within 5 minutes.

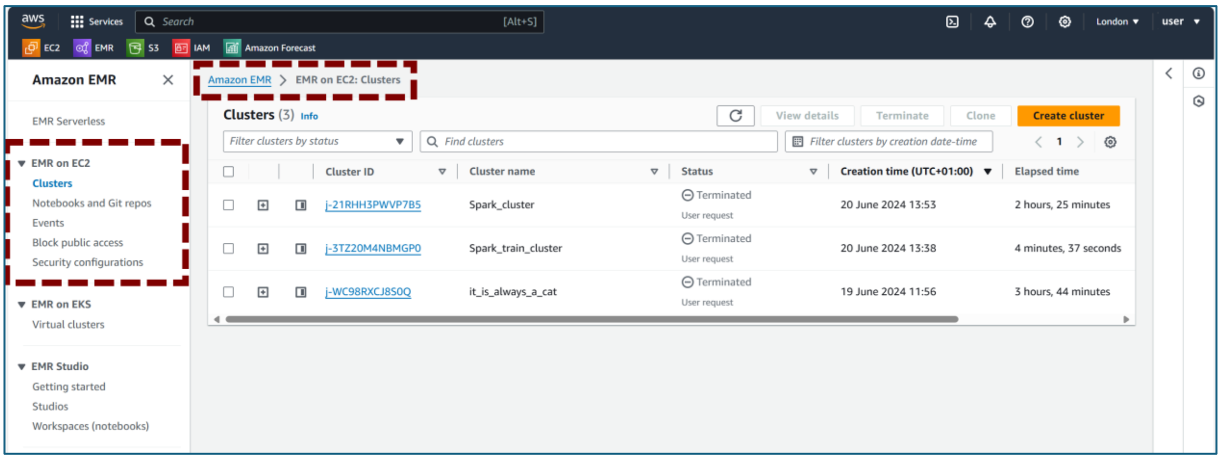
After logging into the AWS console for the first time, the following screen appears.



This console provides a quick overview of all running applications, the most recently opened tabs and a cost overview. The corresponding data centre region can be selected in the top right-hand corner. This setting has a global effect on the following steps and should therefore be selected at the beginning according to the user's requirements.

> **Important Note:** The setup of the AWS environment is **region-dependen**t. An environment in a region can only be used successfully in this region. It is not possible to change the region retrospectively and must therefore be taken into account from the outset.

Using the search bar in the top left-hand corner, we navigate within the AWS environment to the functionalities we require. In this step-by-step instruction, we will retrieve and filter the Ngrams data set within an Elastic Map reduce (EMR) cluster. Therefore, we navigate to EMR via the search bar.



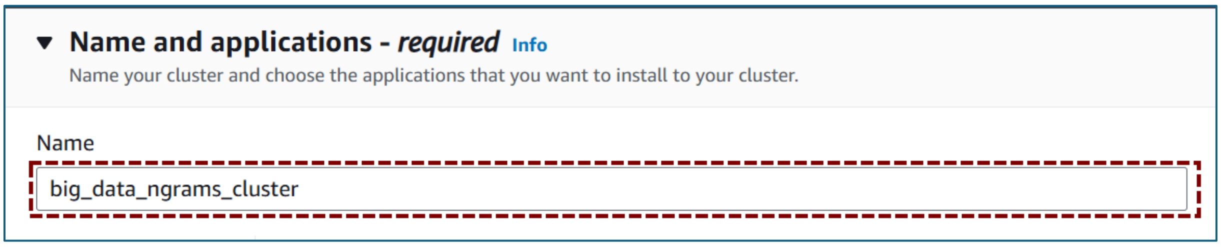
Cluster ID	Cluster name	Status	Creation time (UTC+01:00)	Elapsed time
j-21RHH3PWVP7B5	Spark_cluster	Terminated User request	20 June 2024 13:53	2 hours, 25 minutes
j-3TZ20M4NBMGP0	Spark_train_cluster	Terminated User request	20 June 2024 13:38	4 minutes, 37 seconds
j-WC98RXCJ8S0Q	it_is_always_a_cat	Terminated User request	19 June 2024 11:56	3 hours, 44 minutes

We are now in the Amazon EMR. If we are not yet in the EMR on EC2, we navigate to it. An indication of where we are is outlined in red in the image. The page shows an overview of the status of current and past clusters and should be empty when setting one up for the first time.

To create a cluster we click on **Create cluster** . A new window will open. This is the last opportunity to specify the region of the cluster. We check in the top right-hand corner whether the region is still the desired region.

We now see the setup side of the EMR cluster. The name and applications are selected at the top:

- We choose a name for our cluster



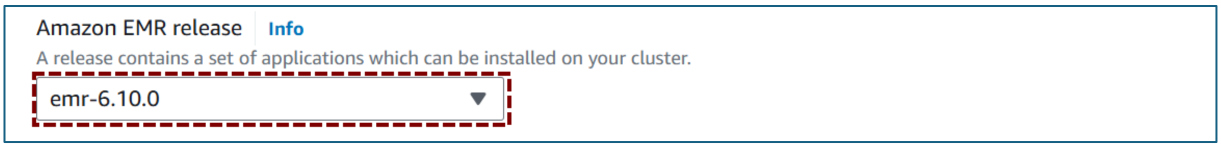
▼ **Name and applications - *required*** Info

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

Name

big_data_ngrams_cluster

- We choose a release EMR in the dropdown: **emr-6.10.0**

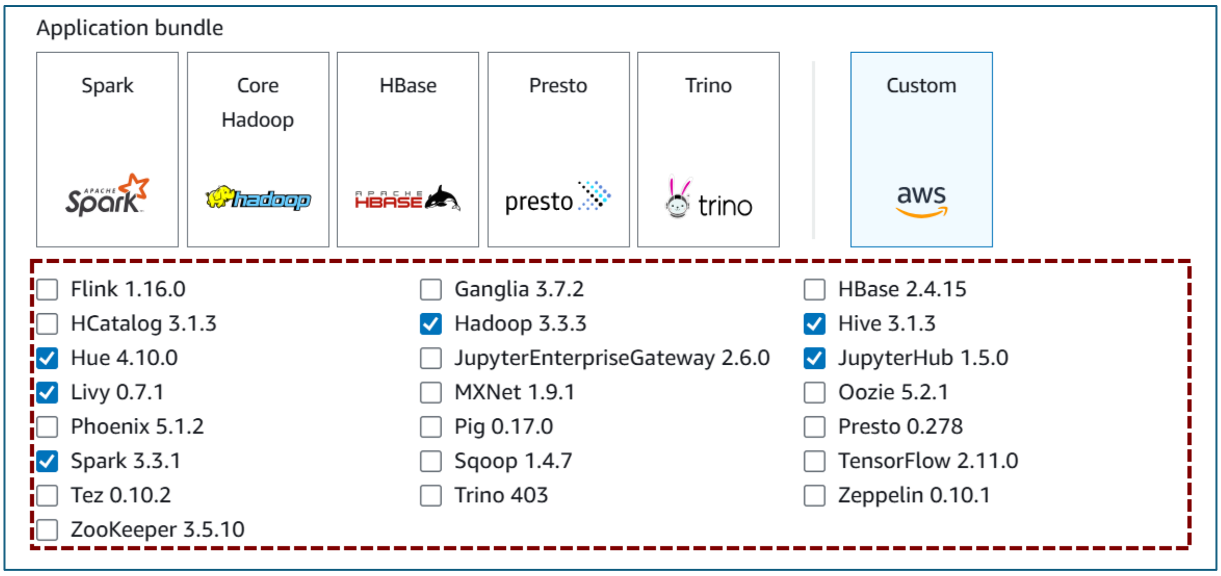


Amazon EMR release Info

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

emr-6.10.0 ▼

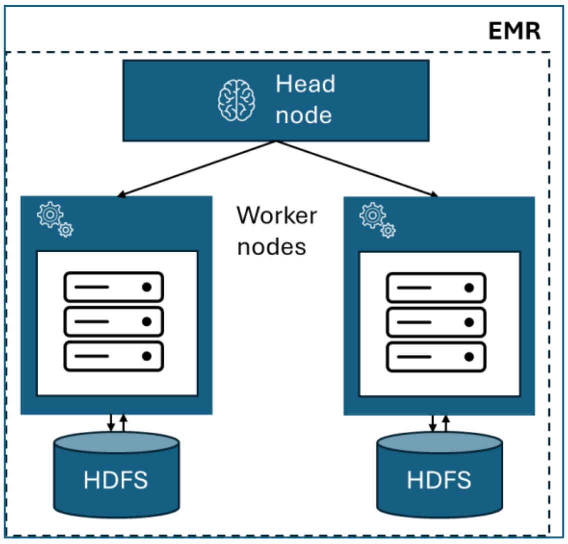
- We choose the application bundle by ticking or unticking the wanted or unwanted applications



☐ 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		

we leave the **other settings in name and applications at the default settings** .

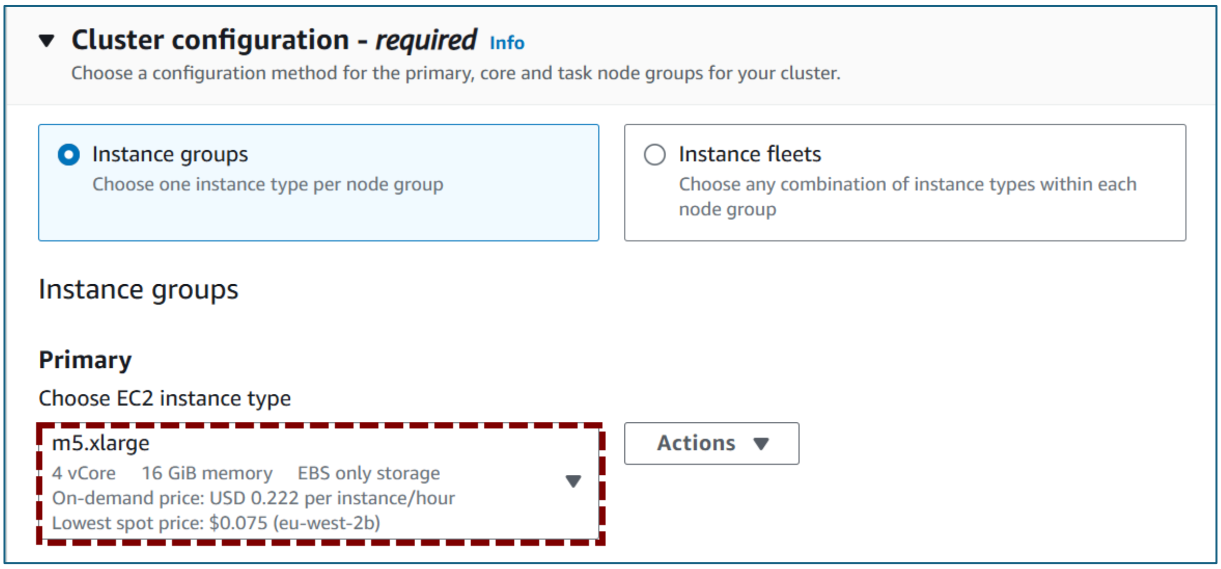
Further below, we are asked to configure the cluster. This is about the architecture of an EMR cluster. Such a cluster consists of a head node and several working nodes, which in turn consist of virtual servers (Elastic Compute Cloud, EC2), set up according to our needs.



In the cluster configuration setting, precisely this architecture of the nodes is determined. A special feature of Spark is that, contrary to the classic Hadoop architecture, the head node not only controls the nodes, but is also used for tasks.

We therefore design the head node (in AWS primary) with the same parameters as the working nodes.

- Primary: m5.xlarge


▼ Cluster configuration - *required* Info

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

● Instance groups
Choose one instance type per node group

○ Instance fleets
Choose any combination of instance types within each node group

Instance groups

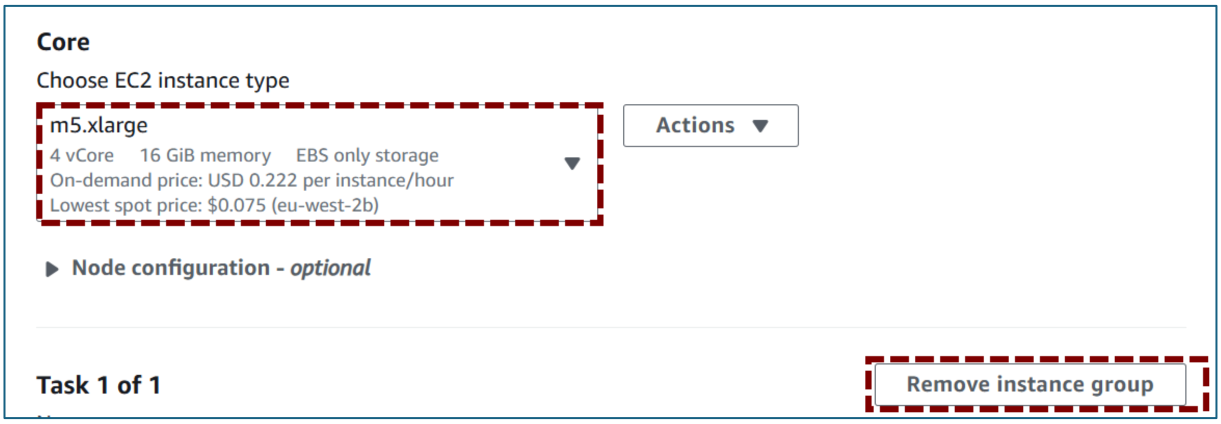
Primary

Choose EC2 instance type

m5.xlarge
4 vCore 16 GiB memory EBS only storage
On-demand price: USD 0.222 per instance/hour
Lowest spot price: $0.075 (eu-west-2b)

The working nodes are called Environment Core in AWs, we create them with the same parameters as the working node:

- Core: m5.xlarge
- Remove the insance groups


Core

Choose EC2 instance type

m5.xlarge
4 vCore 16 GiB memory EBS only storage
On-demand price: USD 0.222 per instance/hour
Lowest spot price: $0.075 (eu-west-2b)

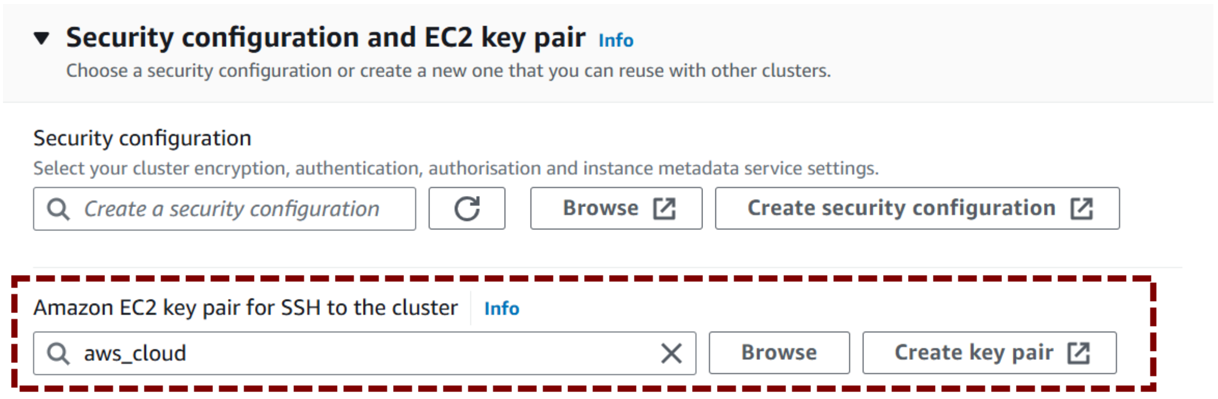
Under the cluster scaling and provisioning section, we specify the number of cores under provisioning configuration:

- Instance(s) size: 2

Important Note: In the **section Cluster termination and node replacement**, it is recommended to automatically terminate the cluster after a specified time in order to avoid unnecessary costs. The idle time is defined for this purpose

In the Security configuration and ec2 key pair section, we need to define a key pair for the secure connection to this environment.

To do this, the SSH key is selected in the Security configuration and EC2 key pair section. This is a user-specific and unique key combination that makes the user recognisable to the server and enables a connection via an SSH tunnel.


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, authorisation and instance metadata service settings.

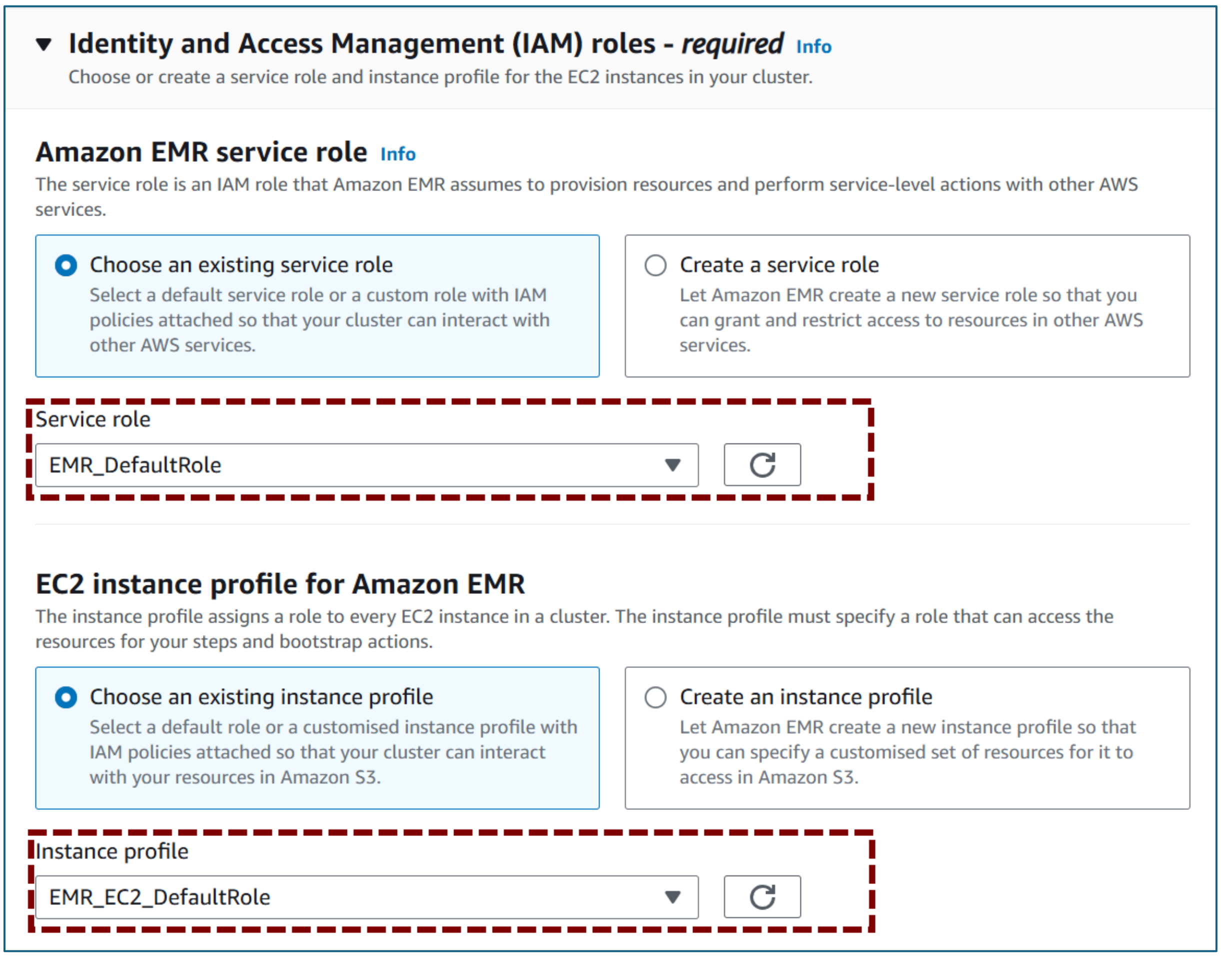
Create a security configuration | Browse | Create security configuration

Amazon EC2 key pair for SSH to the cluster Info

aws_cloud | Browse | Create key pair

- We choose the key.pem file

In the Identity and Access Management (IAM) roles section, we select the corresponding roles for the service and the EC2 instance.


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 customised 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 customised set of resources for it to access in Amazon S3.

Instance profile

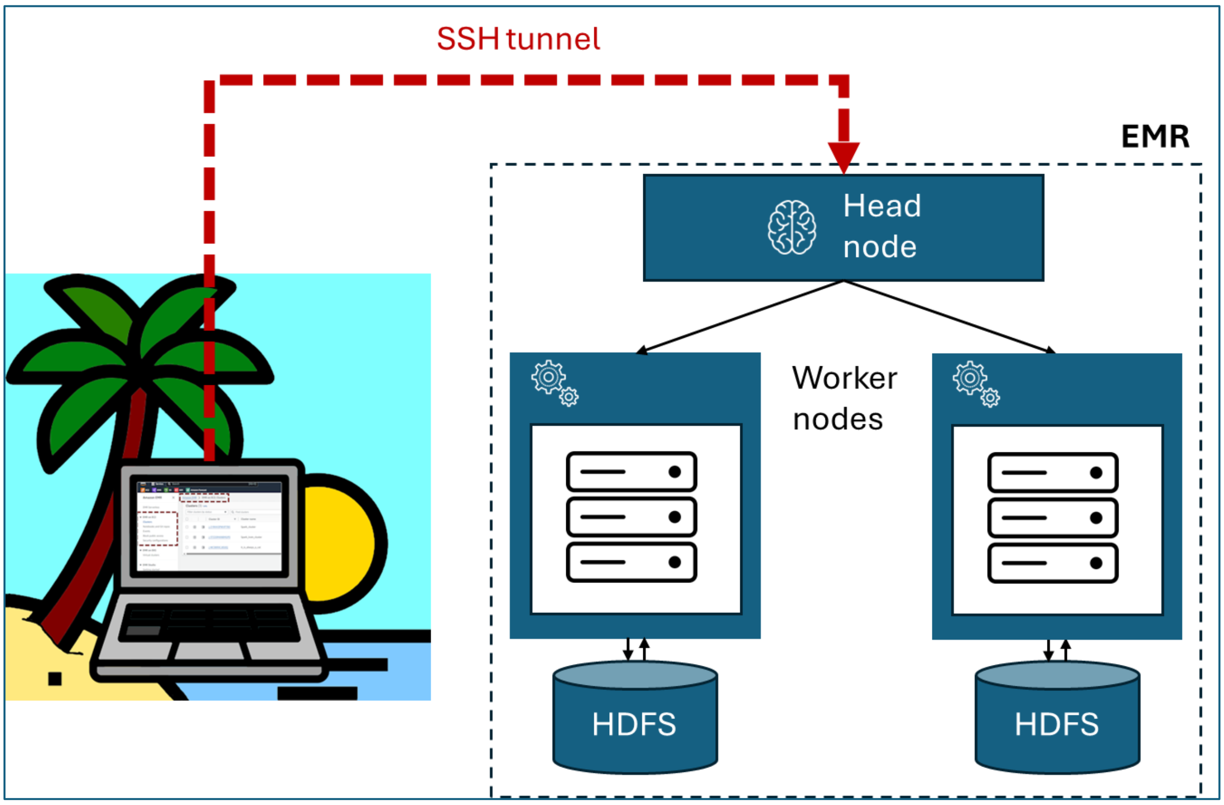
EMR_EC2_DefaultRole

If an SSH key has not yet been created, instructions for creating one can be found in the appendix

Finally we create the cluster by clicking on: Create cluster

The cluster now has the status "**Starting**". A service from AWS is now running in the background, which creates the EMR according to our specifications with the two working nodes EC2 servers and the head node. This process takes a few minutes. While this is happening, we now take a systematic look at the connection between our local laptop and the AWS EMR.

Connecting to the Head node



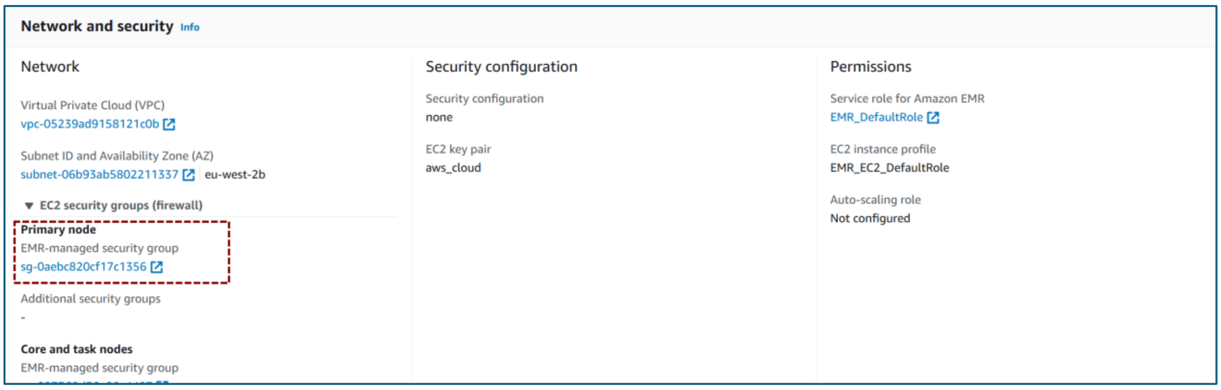
We have now created the EMR, which is set up in the background, according to our specifications. Now, as soon as the EMR is running, we will establish a connection between our local device and the head node (AWS syntax: Primary Node) via an SSH tunnel. We have therefore informed the server of our identity with the SSH key pair during creation and can therefore establish a secure connection. We mirror the head node on our local computer, so to speak. To do this, we have to establish the connection via SSH encryption.

To do this, we have to convert the Key.pem file created in the Windows system into a form that can be processed by Windows using PuTTY. This is because Windows users are not able to set the authorisation to Read only by the user in accordance with Amazon's requirements when creating the pem file. We therefore have to take a diversion via PuTTY.

Important Note: The user must have an existing access key and authenticate themselves within the bash environment using the **aws configure** command

After we have downloaded Putty, we open PuTTYgen. Via 'Load' we navigate to our **.pem file**. We open it in PuTTYgen and save the private key with "**Save private key**" as a **.ppk file**

Nun sollte sich das EMR im Hintergrund aufgebaut haben. Wir navigieren in AWS in dieses und gehen unter Network and secruity auf EC2 secruity groups (firewall) auf den Link unter Primary node – wir werden jetzt eine SSH Verbindung von unserem lokalen Gerät auf den Head Node ermöglichen.


Network and security Info

Network

Virtual Private Cloud (VPC)
vpc-05239ad9158121c0b

Subnet ID and Availability Zone (AZ)
subnet-06b93ab5802211337 eu-west-2b

EC2 security groups (firewall)

Primary node
EMR-managed security group
sg-0aebc820cf17c1356

Additional security groups
-

Core and task nodes
EMR-managed security group

Security configuration

Security configuration
none

EC2 key pair
aws_cloud

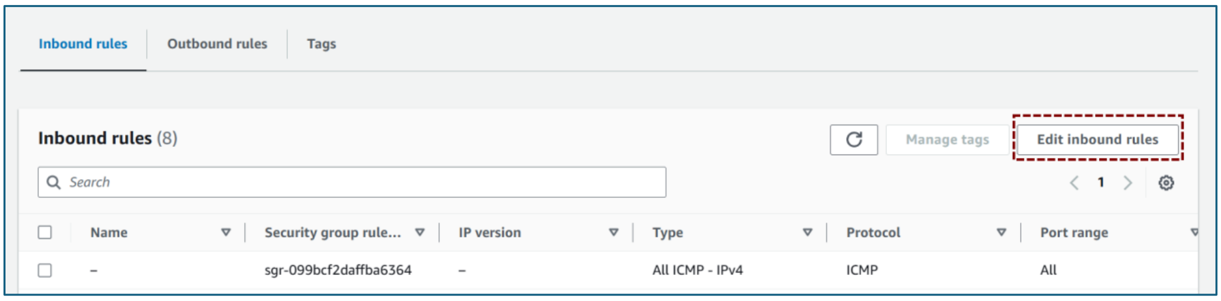
Permissions

Service role for Amazon EMR
EMR_DefaultRole

EC2 instance profile
EMR_EC2_DefaultRole

Auto-scaling role
Not configured

A window opens in which the inbound and outbound rules are listed. To enable an SSH connection, these must be enabled for both inbound and outbound. To do this, click on the **Edit inbound or outbound rules button**.



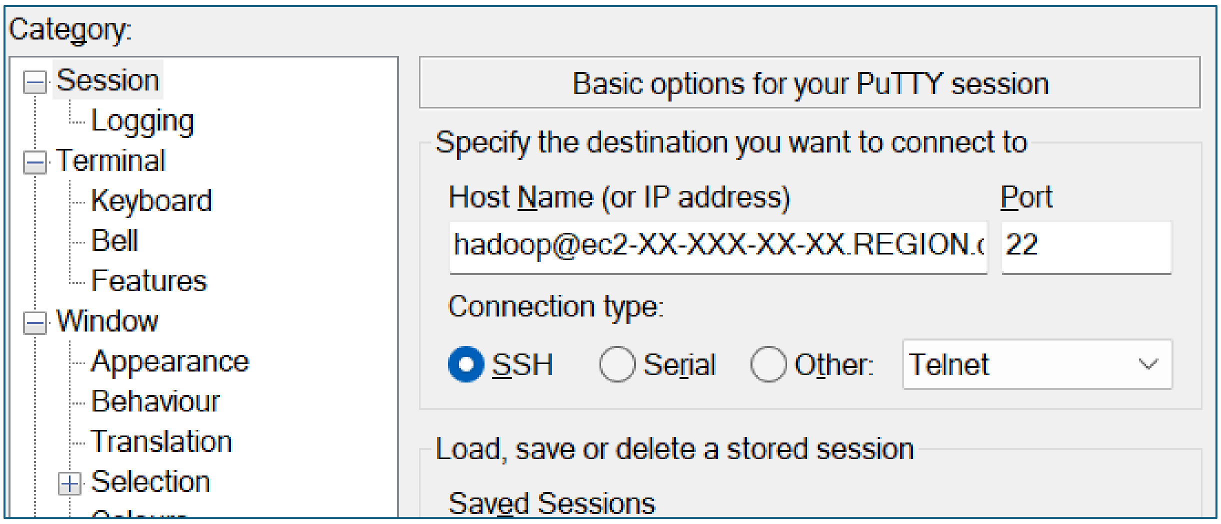
Inbound rules (8)

	Name	Security group rule...	IP version	Type	Protocol	Port range
☐	–	sgr-099bcf2daffba6364	–	All ICMP - IPv4	ICMP	All

To do this, click on **Add rule** at the bottom and set the **type to SSH** and the **source to Anywhere-IPv4**, then click **on Save rules** and repeat the same step for the inbound and outbound rule.

After both the inbound SSH and the outbound SSH connection are allowed, we open PuTTY. Within the EMR, we navigate to Connect to the primary node using SSH and go to Windows.

We copy the host name hadoop@ec2-XX-XXX-XX-XX.REGION.compute.amazonaws.com and paste it into PuTTY:


Category:

- Session
  - Logging
- Terminal
  - Keyboard
  - Bell
  - Features
- Window
  - Appearance
  - Behaviour
  - Translation
  - Selection

Basic options for your PuTTY session

Specify the destination you want to connect to

Host Name (or IP address): hadoop@ec2-XX-XXX-XX-XX.REGION.c

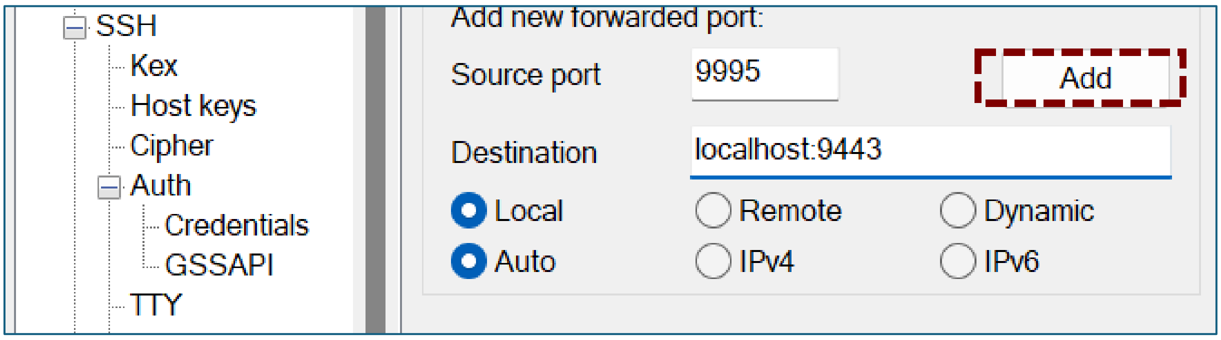
Port: 22

Connection type: [✓ SSH] Serial Other: Telnet

Load, save or delete a stored session

Saved Sessions

We then navigate to SSH - Auth - Credentials under Connections on the left and go to Browse under Private key file for authentication. We now select the .ppk file we have just created from the .pem file. We then navigate to the right under SSH - Tunnels and create the tunnel. In Source port - our local device - we enter port 9995 and under Destination: localhost 9443.


- SSH
  - Kex
  - Host keys
  - Cipher
  - Auth
    - Credentials
    - GSSAPI
  - TTY

Add new forwarded port:

Source port: 9995

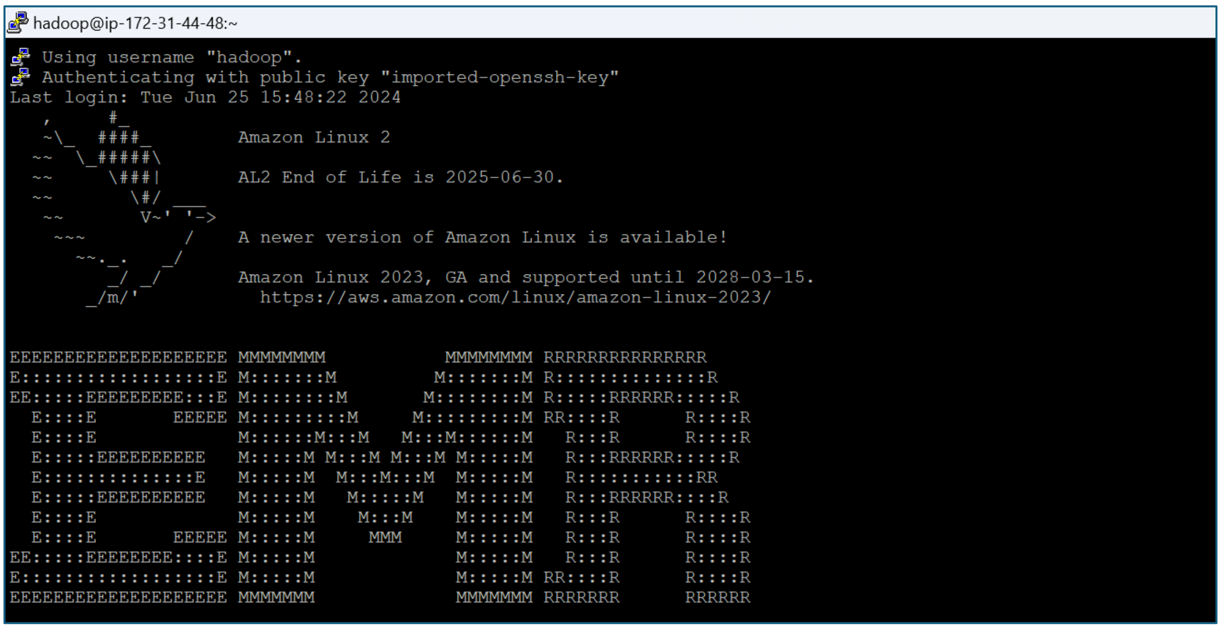
Destination: localhost:9443

[✓ Local] Remote Dynamic

[✓ Auto] IPv4 IPv6

Then click on Add.

The PuTTY console opens, we confirm any warnings and allow the connection. The following screen should now appear:

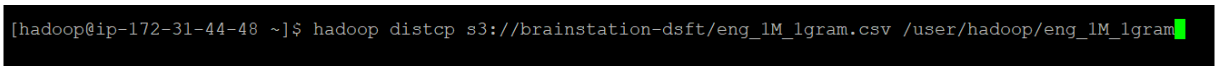

```
hadoop@ip-172-31-44-48:~
Using username "hadoop".
Authenticating with public key "imported-openssh-key"
Last login: Tue Jun 25 15:48:22 2024
   ,     #_
   ~\_  ####_        Amazon Linux 2
  ~~  \_#####\
  ~~     \###|       AL2 End of Life is 2025-06-30.
  ~~       \#/ ___
   ~~       V~' '->
    ~~~         /    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
```

We have successfully established a connection to the head node in the EMR!

Copying the data

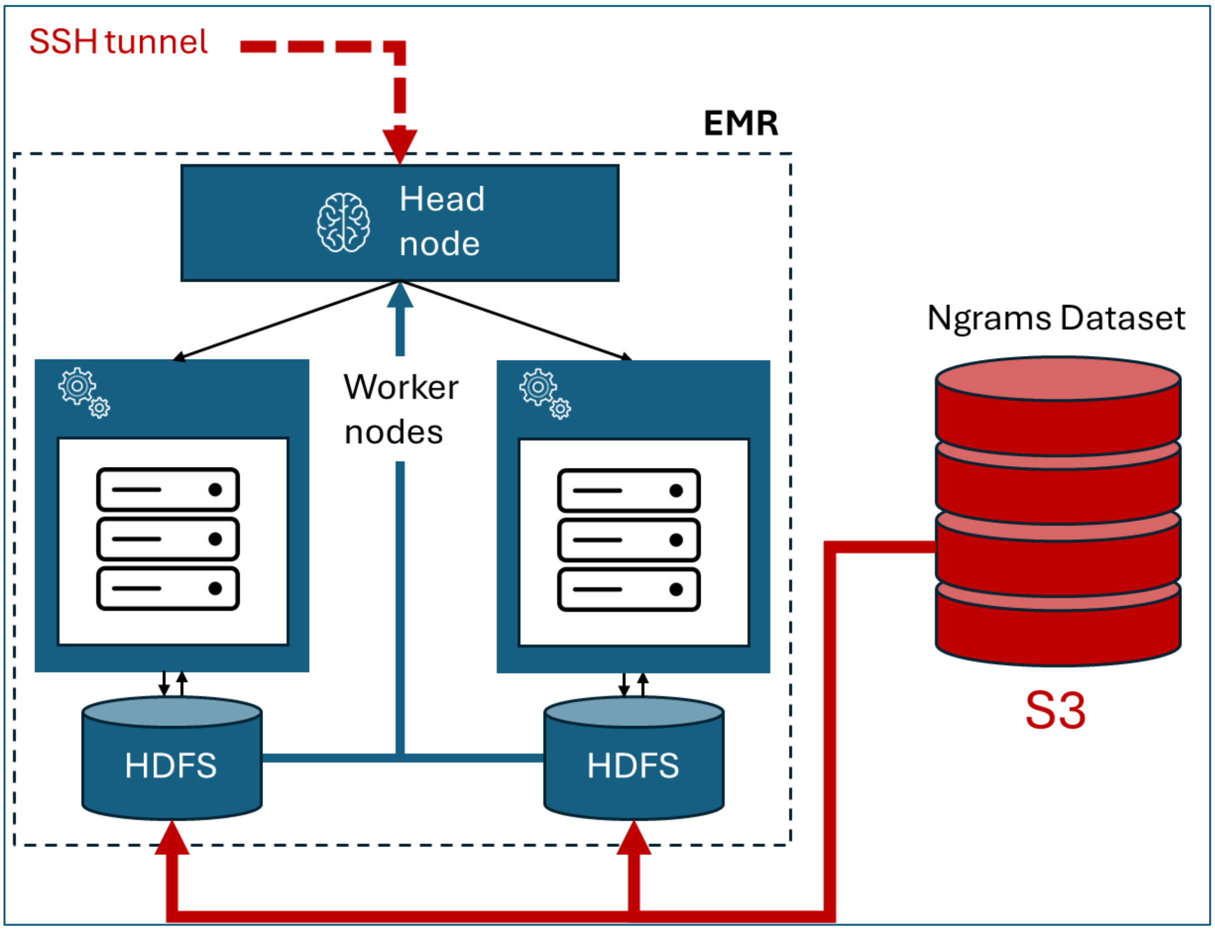
To load the data in the public S3 bucket into our EMR, we use the console in PuTTY. We use the following command:

Hadoop distcp [link to the data in the public S3 bucket] [destination within the HDFS storage inside the EMR]



```
[hadoop@ip-172-31-44-48 ~]$ hadoop distcp s3://brainstation-dsft/eng_1M_1gram.csv /user/hadoop/eng_1M_1gram
```

The data is now loaded from the S3 bucket into our EMR environment. As soon as this is complete, we can access it from within the EMR. To do this, we can control the process of the data from our local device via the SSH tunnel in the head node.



The computing operations are controlled by the head node and processed within the Spark environment. This has certain advantages over the Hadoop method, for more information see Comparison Hadoop and Spark

Working with the data

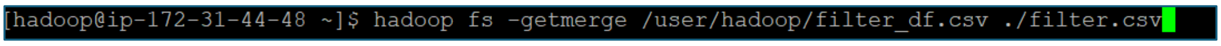
Once we have the data in our HDFS storage in the EMR, we open the Jupyter environment in the head node via the Internet browser at **https://localhost:9995**. Any warnings can be ignored and you can still navigate to the page. (Note, this depends on the browser, but is usually accessible under advanced - proceed to ...).

A login window appears. We log in with the username **jovyan** and the password **jupyter**.

We are now in the Jupyter environment within the head node. In order to filter the data from the Ngrams, we will enter the commands in a notebook here. The corresponding notebook can also be found in the appendix.

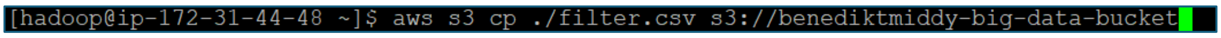
Access to the edited data

After we have processed the data within the Jupyter environment according to our requirements and saved it as a file, we now need to make it available outside our EMR, as this should be terminated after the work has been completed. To do this, we execute the following command in the PuTTY console.



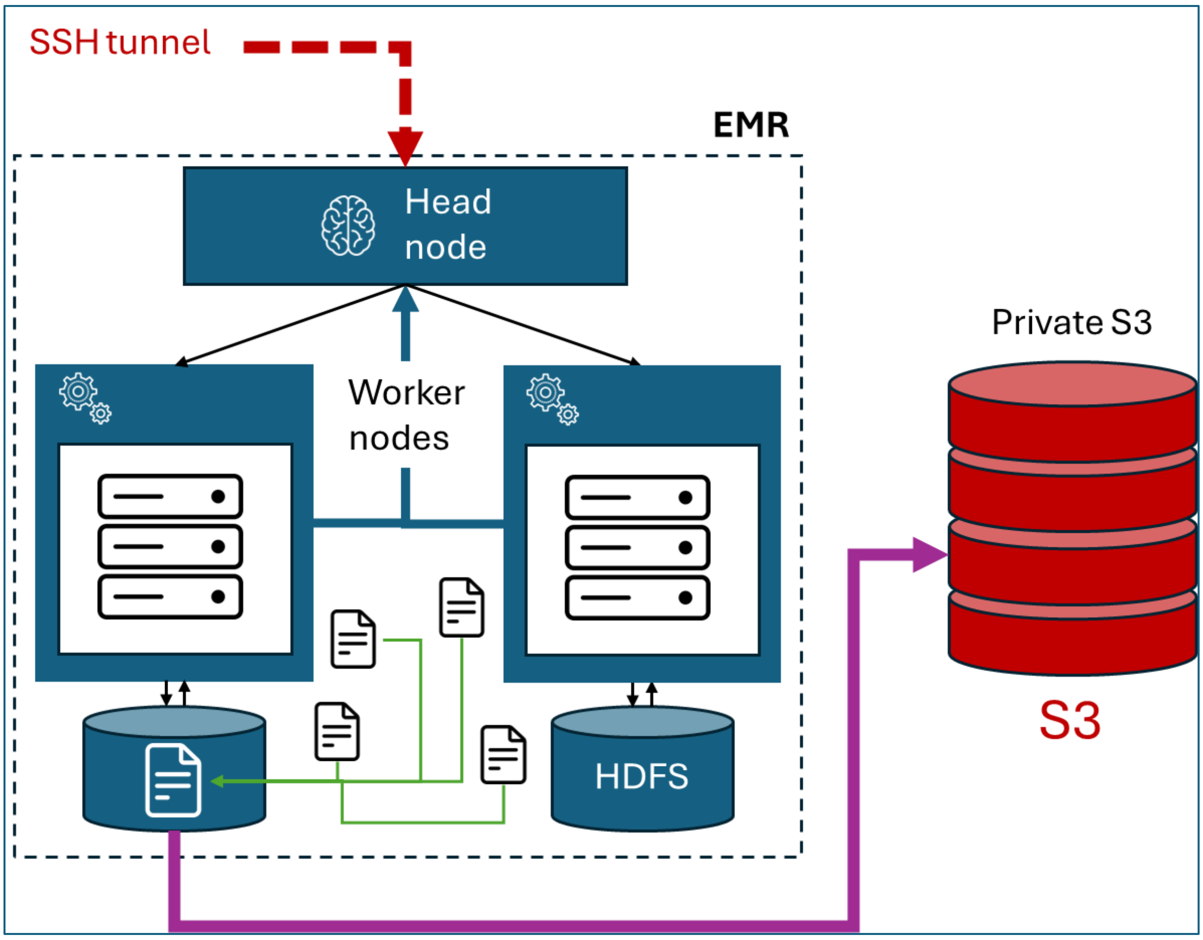
```
[hadoop@ip-172-31-44-48 ~]$ hadoop fs -getmerge /user/hadoop/filter_df.csv ./filter.csv
```

The file processed and saved within the Jupyter environment is now saved as a complete file within the EMR. This step is necessary because, in terms of efficiency, the file was split between the working nodes within the EMR in order to carry out the work in parallel. The compiled file is now located in our EMR environment. It can be extracted from this environment with the command



```
[hadoop@ip-172-31-44-48 ~]$ aws s3 cp ./filter.csv s3://benediktmiddy-big-data-bucket
```

And copied to an S3 environment to save them.

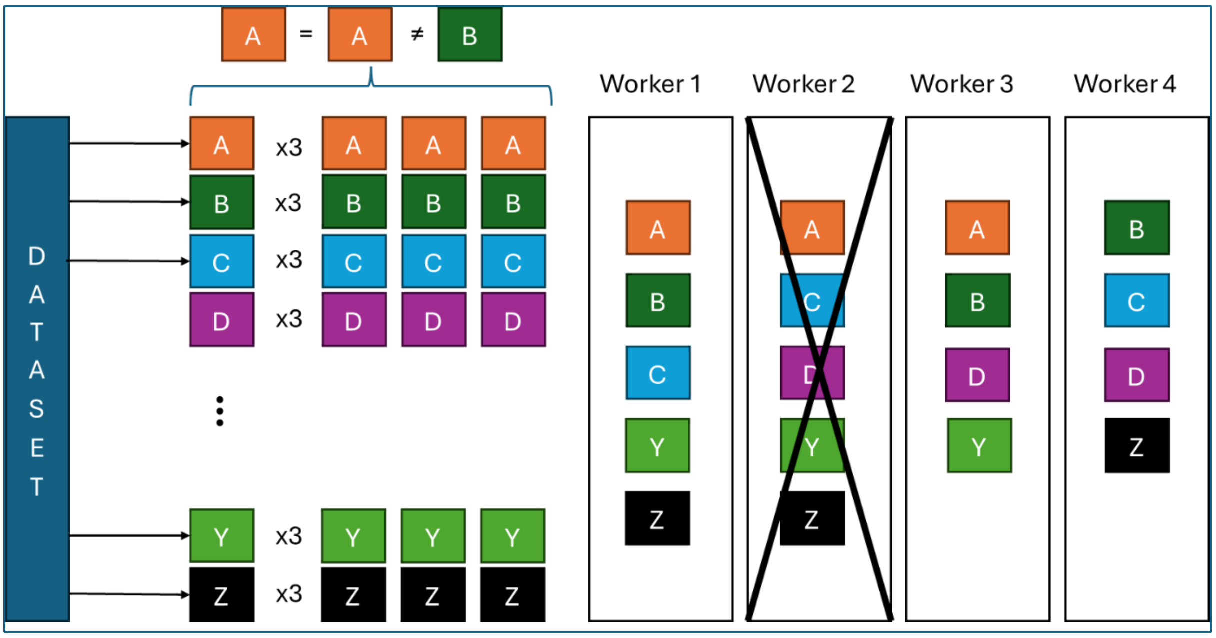


The EMR is then terminated via the AWS website and the tunnel is closed.

Comparison Hadoop and Spark

Hadoop	Spark
Horizontal scaling, the parallel operation of several machines and the even **or most resource-efficient distribution of task packages among them**	Compared to Hadoop, Spark **has a large number of APIs** such as SQL, MLLib and GraphX
More cost-effective solution, as less memory is required	**Operation is faster**, as calculation results remain **temporarily stored in the working memory** and are only transferred to the long-term memory at the end
Machines are distributed locally and therefore have a higher level of redundancy, which is achieved through replicated data distribution. The data set is divided into blocks. These blocks are replicated and distributed across the machines in the horizontal network → see HDFS storage	Machines are distributed locally and therefore have a higher level of redundancy, which is achieved through replicated data distribution.
This method of **operation is slower in relation to Spark**, as a slow read from the long-term memory alternates with arithmetic operations and subsequent long-term writing to the memory.	Spark is **significantly more resource intensive as valuable memory is required** in the worker and head node and therefore a Spark framework is significantly more expensive

HDFS data storage:

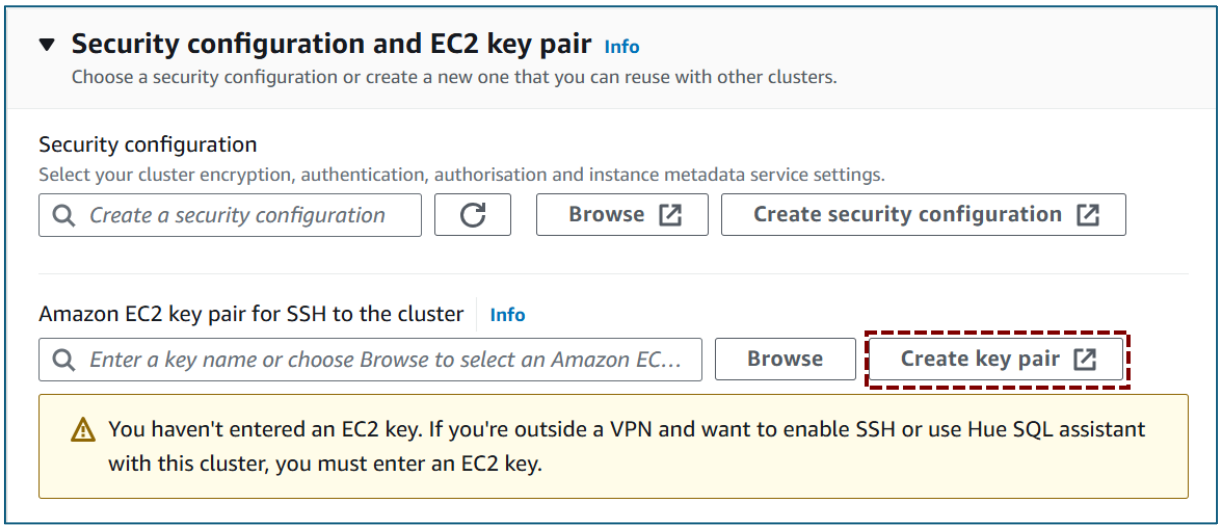


An HDFS memory works by dividing a data set evenly into individual blocks, typically 64 MB or 128 MB in size. The buzzer of blocks A to Z in the example contains the entire data set. These blocks are replicated three times in the above example, i.e. the copies of A are identical to A. These copies are now distributed evenly across all working nodes. The head node is responsible for administration. If a worker fails, in the example of worker 2, the complete data record is still minimum backed up twice. This redundancy is the great advantage of HDFS storage.

Appendix

SSH-key creation and role creation

If no EC2 key pair is available, we press CTRL+ Create key pair button, a new window should open.



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, authorisation and instance metadata service settings.

Create a security configuration
Browse
Create security configuration

Amazon EC2 key pair for SSH to the cluster Info

Enter a key name or choose Browse to select an Amazon EC…
Browse
Create key pair

You haven't entered an EC2 key. If you're outside a VPN and want to enable SSH or use Hue SQL assistant with this cluster, you must enter an EC2 key.

We give the key to be created a name, key pair type as RSA and select **.pem** as the file format. Now we navigate in the same window via the search bar to the Identity and Access Management area, simply enter IAM in the search field. In the tab on the left, we go to Roles. After the new key creation, the roles should not yet be displayed there.

- EMR_DefaultRole
- EMR_EC2_DefaultRole

... be available.

We go to Create role. As a use case, we first search for the use case: EMR for the EMR_DefaultRole and select EMR below. In the next window after Next, the AmazonElasticMapReduceRole or for EMR_EC2_DefaultRole - EMR - EMR Role for EC2 - AmazonElasticMapReduceforEc2Role is now displayed by default. We give the roles a name; the naming convention shown above is recommended. We create a total of 2 roles, with the EMR_EC2_DefaultRole EMR Role for EC2 after selecting the use case. Both roles have now appeared in the Roles overview. We now first select the EMR_DefaultRole by clicking on it. Under Permissions we go to Add permissions - Attach policies. We search for AmazonS3FullAccsess and add it. We repeat the same step for EMR_EC2_DefaultRole:

- EMR_DefaultRole
  - AmazonElasticMapReduceRole
  - AmazonS3FullAccess
- EMR_EC2_DefaultRole
  - AmazonElasticMapReduceforEC2Role
  - AmazonS3FullAccess

Now we go back to the creation of our EMR and should find the two roles after selecting our SSH key.

Attachments

- Big_data_assignment.ipynb – contains the filtering and work inside the EMR
- Big_data_notebook.ipynb – contains the work with the output of the EMR loaded from an S3 bucket.