AWS EMR (Amazon-Elastic-MapReduce)

1. Youtube videos

AWS EMR Cluster Create using AWS Console | Submitting Spark Jobs in AWS EMR Cluster:

https://www.youtube.com/watch?v=XRGveXDutKM

AWS EMR Big Data Processing with Spark and Hadoop | Python, PySpark, Step by Step Instructions:

https://www.youtube.com/watch?v=a_3Md9nV2Mk

How to process big data workloads with spark on AWS EMR:

https://www.youtube.com/watch?v=6OEwdJbnDY8

https://github.com/Primus-Learning/emr-spark-job-to-process-data/tree/main

Running an Amazon EMR Cluster in the AWS Academy Data Engineering Sandbox:

https://www.youtube.com/watch?v=radrkdkUI0U

AWS EMR videos by Dr. Sian Lun: https://www.youtube.com/@sianlun/videos

2. ¿What is AWS EMR?

https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-what-is-emr.html

AWS EMR is a platform managed on AWS that allows us to execute **Big Data** jobs with the **Hadoop** ecosystem as a **distributed processing engine**

Use Amazon Elastic Compute Cloud (Amazon **EC2**) instances to run the clusters with the open source services that we need, such as **Apache Spark** or **Apache Hive**

EMR has **HDFS** as the storage layer for the cluster

Also, it allows us to decouple computing from storage using the **S3** service to store data and logs without limit

You can choose between several versions that determine the open source stack that is deployed in the cluster

Includes Hadoop, Hive, Tez, Flink, Hue, Spark, Oozie, Pig and HBase among others

The system is also integrated with other AWS services and provides us with **notebooks** to run code on the cluster

Jupyter Lab can be used using **Apache Livy**

3. AWS EMR Architecture

Master: Master nodes must be up and running to service the cluster.

Several can be deployed to have high availability.

By default, they host services like Hive Metastore.

Core: These nodes are responsible for storing data in HDFS and executing jobs, they can also be scaled.

Task: These nodes do not store data so they can be added and deleted without risk of data loss.

They are used to add processing capacity to the cluster.

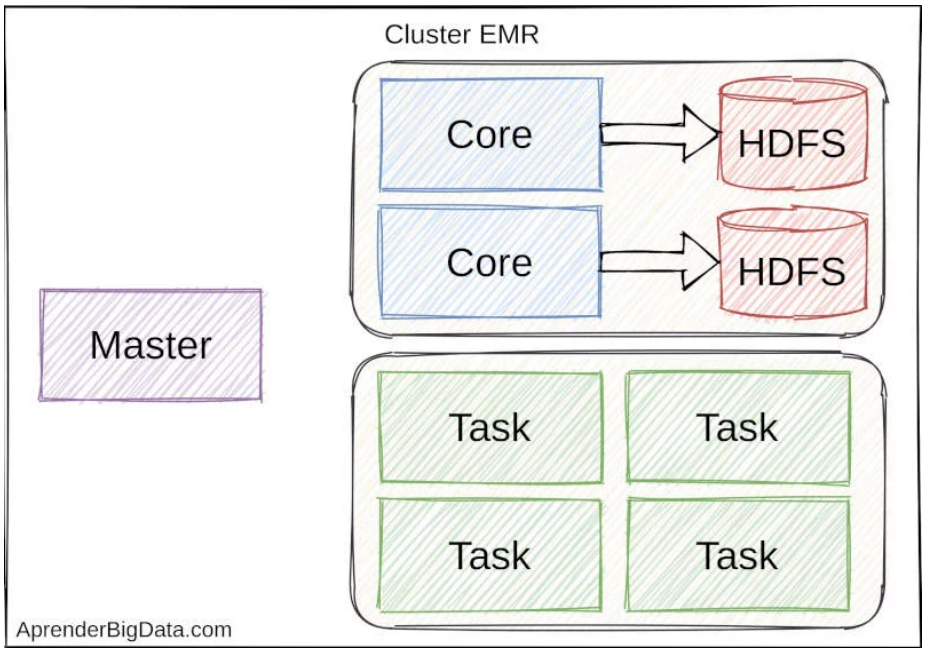

Figure 1: image

Among the deployment options that EMR has, we can choose to pay per use, based on time or save on costs by using reserved instances, savings plans or AWS spot instances.

4. Scaling AWS EMR

Depending on the workloads we want to run, we can deploy **specific clusters** for the duration of our work or have a **permanent cluster** with high availability and auto-scaling based on demand.

The first case is recommended for specific and lighter jobs.

EMR deployments can be scaled **automatically** or **manually** by setting core node limits.

To scale the cluster, **utilization metrics** will be used, taking data replicas into account.

In the case of streaming jobs with **Spark Streaming** we must carefully analyze the cluster's ability to scale with the volume.

It is possible that the cluster can add capacity automatically but when the volume of work decreases again **it will not be able to reduce the number of nodes**, increasing costs considerably.

5. Storing in AWS EMR

We must understand that EMR provides two forms of storage:

EMRFS: This file system is based on the **S3** service. It has the ability to decouple cluster computing from storage.

HDFS: Requires a **dedicated cluster**. We must configure a replication factor for the Core nodes and take it into account for correct sizing.

EMR always needs **HDFS**, so at least one Core type node will be needed.

Both modes are supported, and we can persist our data to whatever storage we need.

Also, we can use **s3DistCp** to copy data between them.

In jobs where many read operations are not performed, we can use EMRFS with S3 to optimize costs.

In jobs with a lot of iterative reads (for example **Machine Learning**), we will benefit more from a system like **HDFS**.

6. AWS EMR Big Data Applications

Let me break down some of the main capabilities of each:

Flink 1.17.1: Apache Flink is a stream processing framework for big data processing and analytics. It supports both batch and stream processing with low-latency and high-throughput.

Ganglia 3.7.2: Ganglia is a scalable and distributed monitoring system for high-performance computing systems. It is often used to monitor clusters and grids.

HBase 2.4.17: Apache HBase is a distributed, scalable, and NoSQL database built on top of Hadoop. It provides real-time read and write access to large datasets.

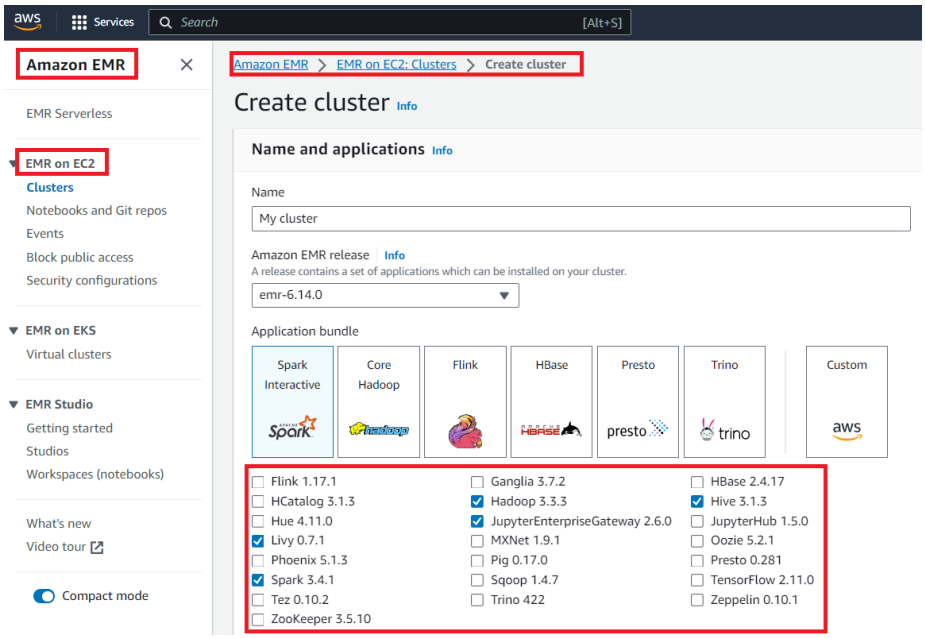

Figure 2: image

HCatalog 3.1.3: HCatalog is a table and storage management service for Hadoop that enables users to share and access data across Pig, Hive, and MapReduce.

Hadoop 3.3.3: Apache Hadoop is an open-source framework for distributed storage and processing of large data sets using a cluster of commodity hardware.

Hive 3.1.3: Apache Hive is a data warehousing and SQL-like query language for large datasets stored in Hadoop distributed file systems.

Hue 4.11.0: Hue is a web-based interface for analyzing data with Hadoop. It provides a user-friendly interface for various Hadoop ecosystem components.

JupyterEnterpriseGateway 2.6.0 and JupyterHub 1.5.0: JupyterHub is a multi-user server for Jupyter notebooks, and Enterprise Gateway enables Jupyter notebooks to interact with big data clusters.

Livy 0.7.1: Apache Livy is a REST service for interacting with Spark from anywhere. It allows remote applications to submit Spark jobs, query status, and retrieve results.

MXNet 1.9.1 and TensorFlow 2.11.0: These are deep learning frameworks. MXNet and TensorFlow are used for developing and training machine learning models.

Oozie 5.2.1: Apache Oozie is a workflow scheduler for Hadoop jobs. It allows

users to define, manage, and schedule data workflows.

Phoenix 5.1.3: Apache Phoenix is a SQL skin over HBase, providing a relational database layer for HBase.

Pig 0.17.0: Apache Pig is a high-level scripting language for analyzing large datasets on Hadoop. It simplifies the development of complex data processing tasks.

Presto 0.281 and Trino 422: Presto (now known as Trino) is a distributed SQL query engine optimized for ad-hoc analysis. It allows querying data where it resides.

Spark 3.4.1: Apache Spark is a fast and general-purpose cluster computing system for big data processing. It supports in-memory processing and various data processing tasks.

Sqoop 1.4.7*: Apache Sqoop is a tool for efficiently transferring bulk data between Hadoop and structured datastores such as relational databases.

Tez 0.10.2: Apache Tez is an extensible framework for building high-performance batch and interactive data processing applications.

Zeppelin 0.10.1: Apache Zeppelin is a web-based notebook that enables interactive data analytics with a variety of interpreters, including for Spark, Hive, and more.

ZooKeeper 3.5.10: Apache ZooKeeper is a distributed coordination service used for maintaining configuration information, naming, providing distributed synchronization, and group services.

These tools collectively form a powerful ecosystem for big data processing, analytics, and machine learning.

They enable various tasks such as data storage, processing, monitoring, and analysis in large-scale distributed systems.

7. ¿How to create a new AWS EMR Cluster?

7.1. First we set the AWS EMR Cluster name

Figure 3: image

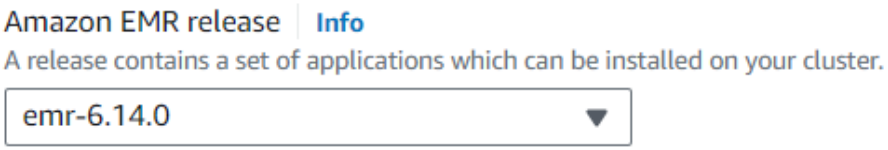

Figure 4: image

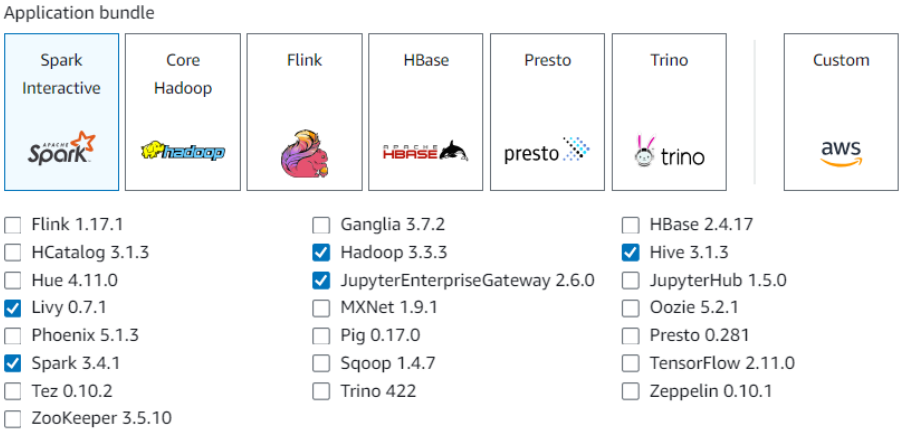

Figure 5: image

7.2. We select the AWS EMR version

7.3. We select the applications to be installed in the AWS EMR cluster

7.4. We select the operating systems for the EC2 integrating the cluster

We leave the default option, to install Linux as operating system

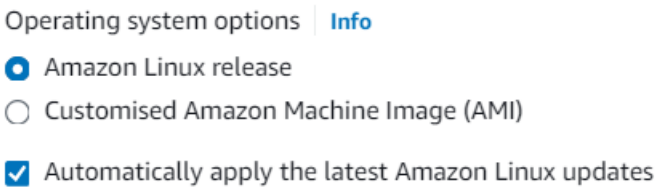

Figure 6: image

7.5. We select the EC2 instances types

We select the option **One instance type per Nodes Group**

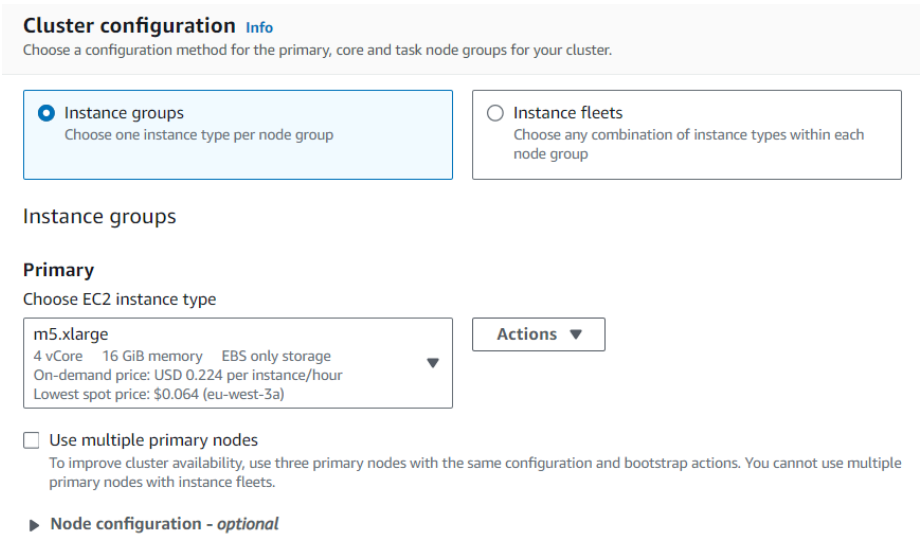

Figure 7: image

7.6. Scaling the cluster

We select the option Manual Scaling

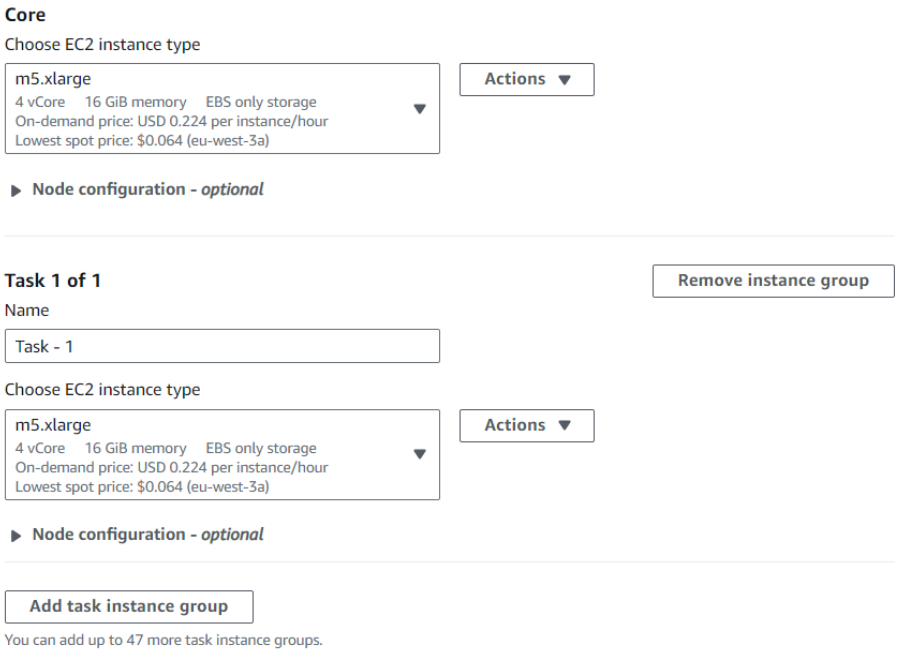

Figure 8: image

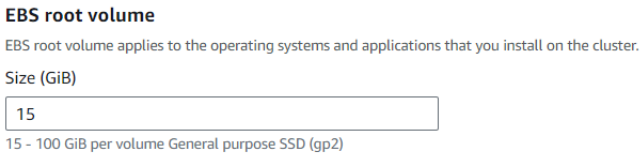

Figure 9: image

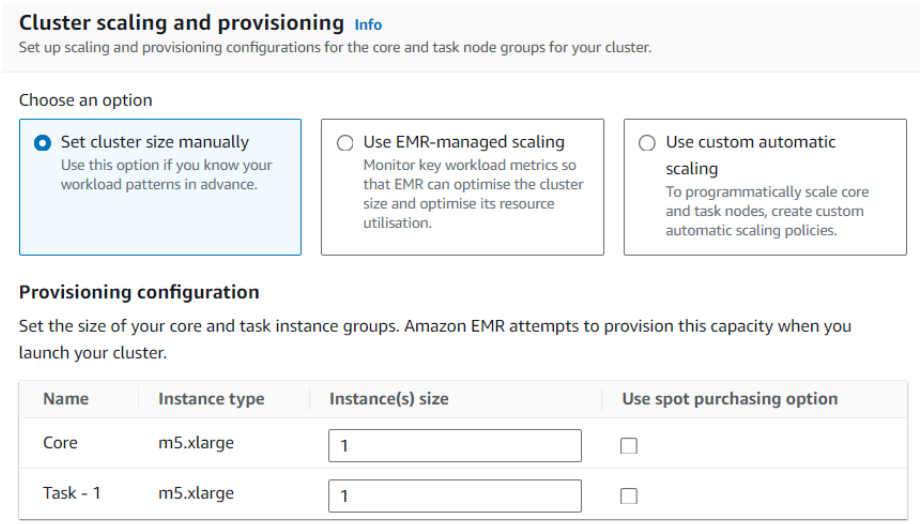

Figure 10: image

7.7. For your subnet to communicate with external sources

As previous stFor your subnet to communicate with external sources:

a) you need to add a default route (0.0.0.0/0) pointing to either an Internet Gateway (if it's a public subnet)

b) or a NAT Gateway (if it's a private subnet).

Here's what you can do:

a) Internet Gateway (for public subnet):

- Go to the AWS VPC Console.
- Navigate to the "Internet Gateways" section.
- Create an Internet Gateway if you haven't already.
- Attach the Internet Gateway to your VPC.
- Go back to the route table (rtb-04c42678a04496623) for subnet-0c773b8ac5250a86a and add a route:
- Destination: 0.0.0.0/0
- Target: The Internet Gateway you just attached.

b) NAT Gateway (for private subnet):

Go to the AWS VPC Console.

Navigate to the "NAT Gateways" section.

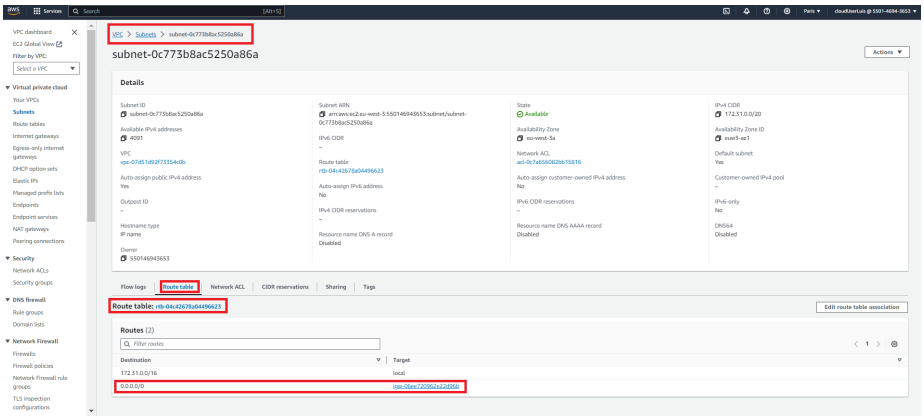

Figure 11: image

Create a NAT Gateway if you haven't already.

Make sure the NAT Gateway is in a public subnet.

Go back to the route table (rtb-04c42678a04496623) for subnet-0c773b8ac5250a86a and add a route:

Destination: 0.0.0.0/0

Target: The NAT Gateway you just created.

After making these changes, your subnet should have a route to external sources,

7.8. VPC and Subnet

We select the Default VPC in our AWS account.

Regarding the Subnet, we select one of the subnets in the Default VPC.

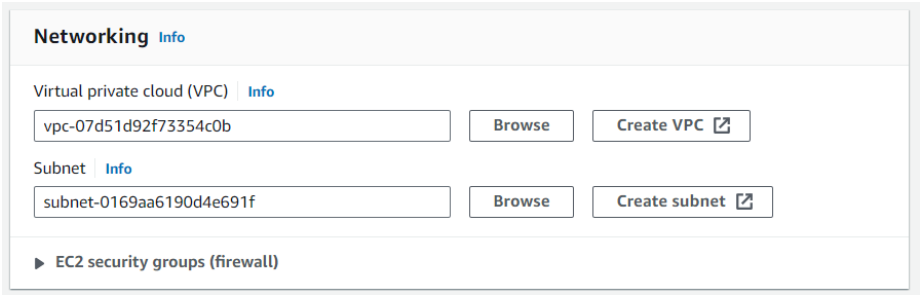

Figure 12: image

You can see the available VPC and Subnets

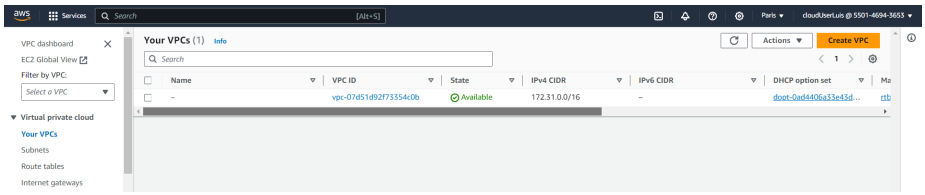

Figure 13: image

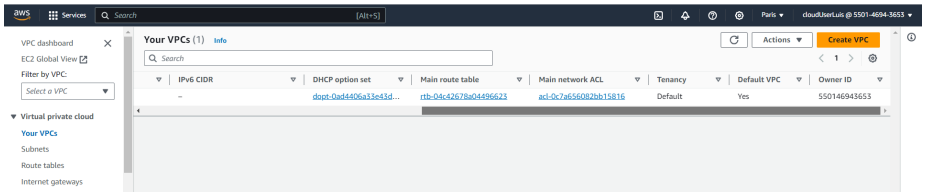

Figure 14: image

7.9. AWS EMR Termination

We select Manual Termination

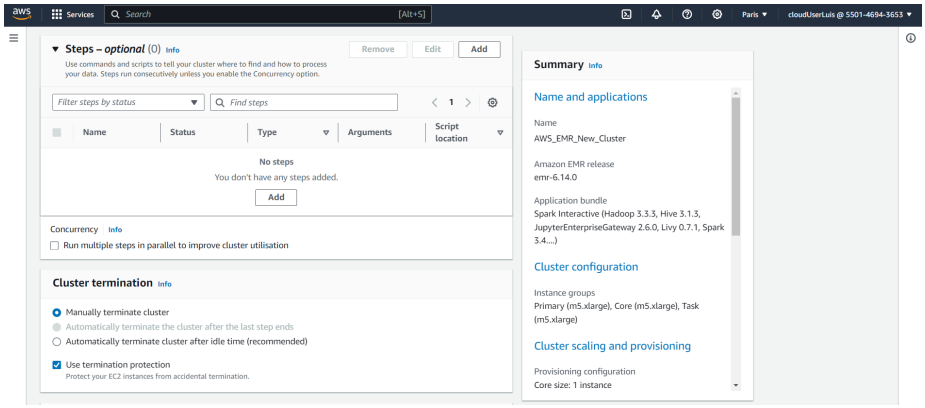

Figure 15: image

7.10. "Bootstrap actions" and "Cluster Logs"

We leave the default values

7.11. "Tags" y "Edit software settings"

We leave the default values

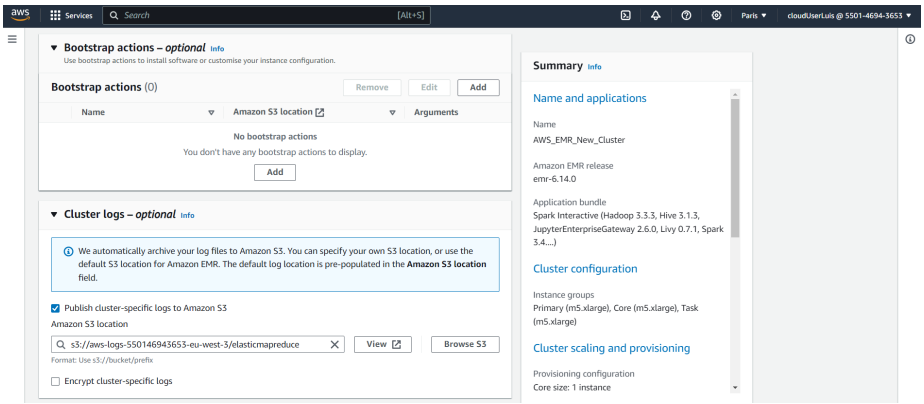

Figure 16: image

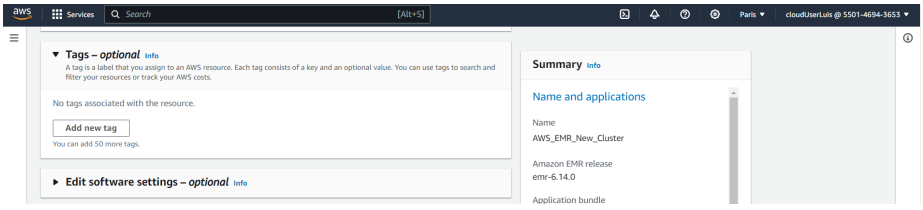

Figure 17: image

7.12. Security configuration and EC2 key pair

As previous step we have to create a **Key-Pair** and download the **ppk** file in our hard disk

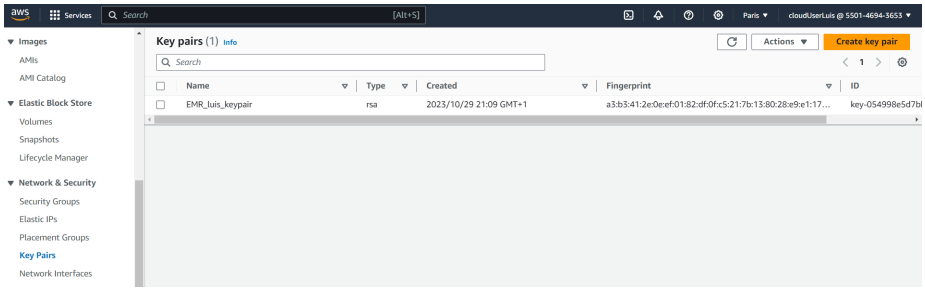

Figure 18: image

After we press the **"Create key pair"** button, and automativally the **ppk** file is downloaded in our computer

Once we created the **ppk** file we load it for configuring the security option in the AWS EMR cluster

7.13. IAM (Identity and Access Management)

We create the "Service Role" and the "Instance Profile" for EC2 instances

7.13.1. Creating the "Service Role" Le asignamos al nuevo role el nombre "EMR_DefaultRole", y pulsamos el botón crear:

7.13.2. Creating the "Instance Profile"

7.13.3. Identity and Access Management (IAM) roles

7.14. Push the "Create cluster" buttton

7.15. Edit the Security Group in the Master node

Press the button "Edit Inbound Rules"

Add another rule in the **SSH** protocol and **port 22** to access from **My IP** and press the **Save Rules** button

Or access with **SSH** protocol and **port 22** from **every IP address** and the press the **Save Rules** button

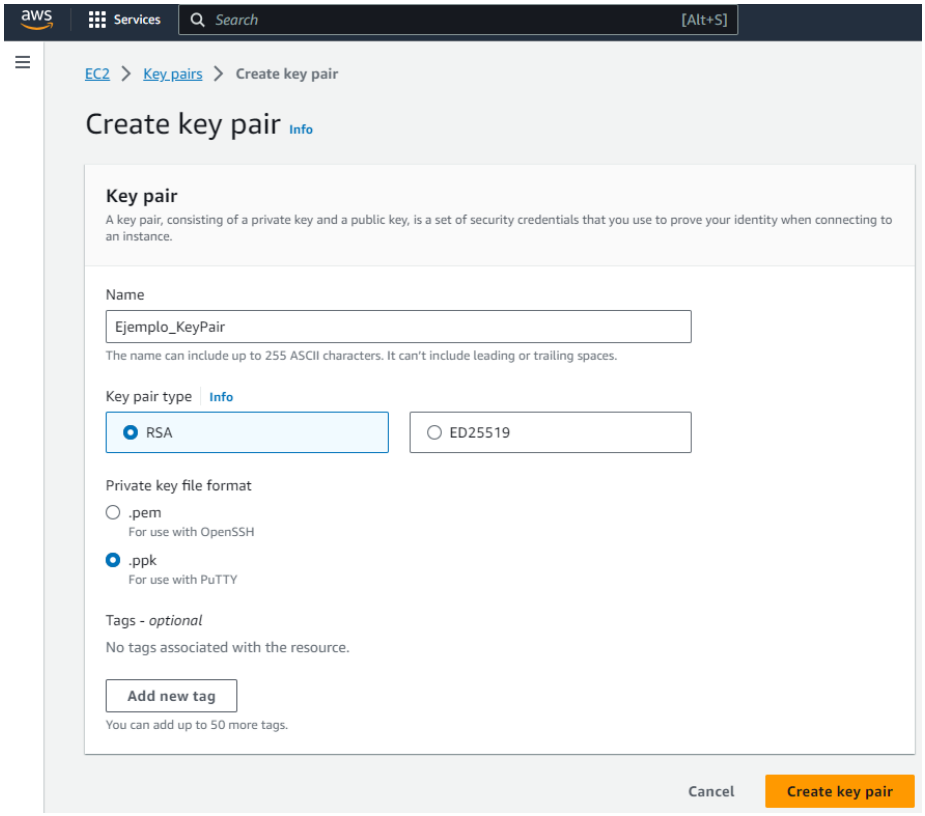

Figure 19: image

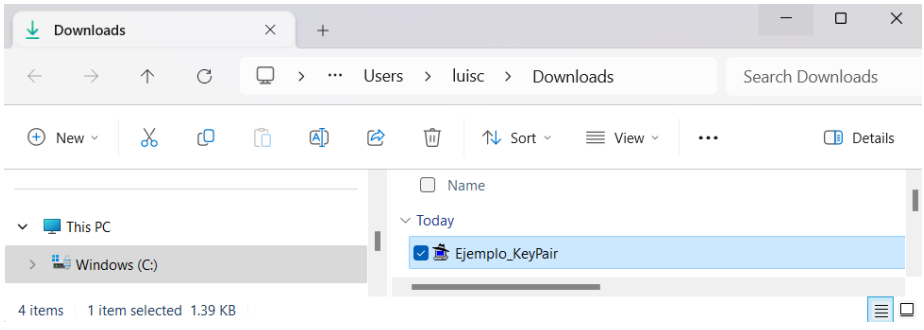

Figure 20: image

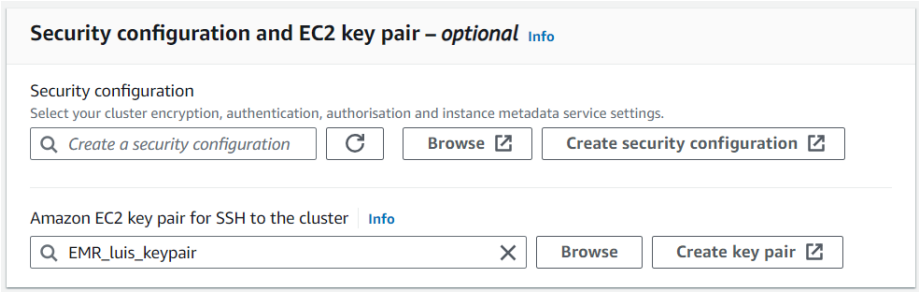

Figure 21: image

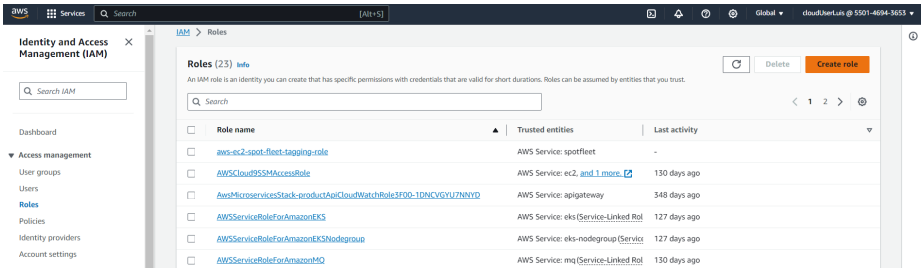

Figure 22: image

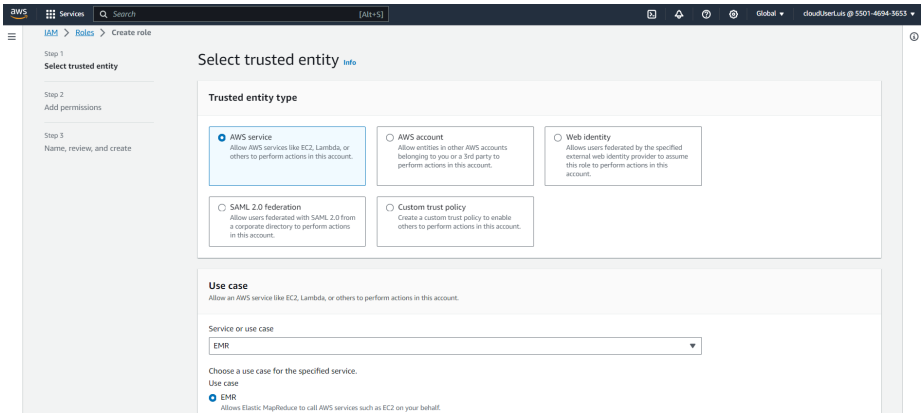

Figure 23: image

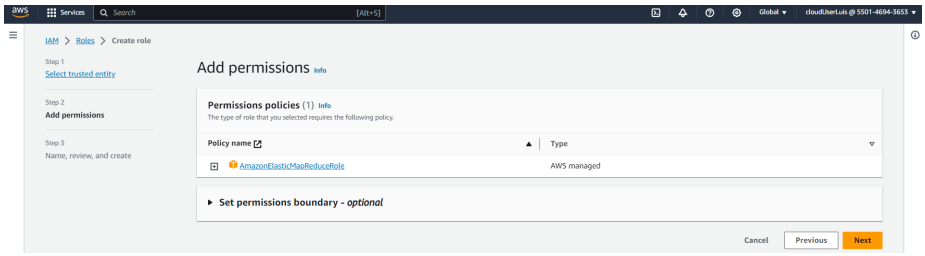

Figure 24: image

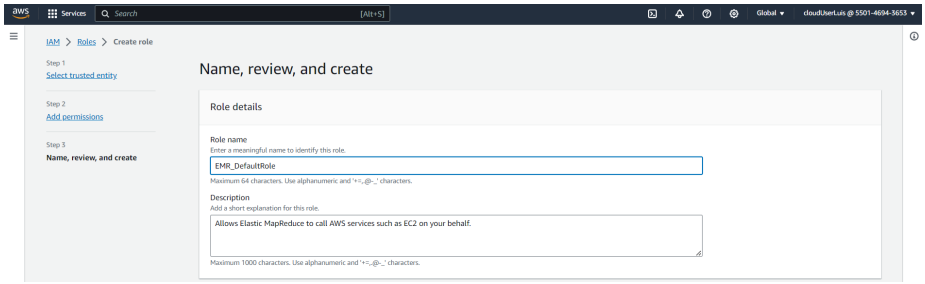

Figure 25: image

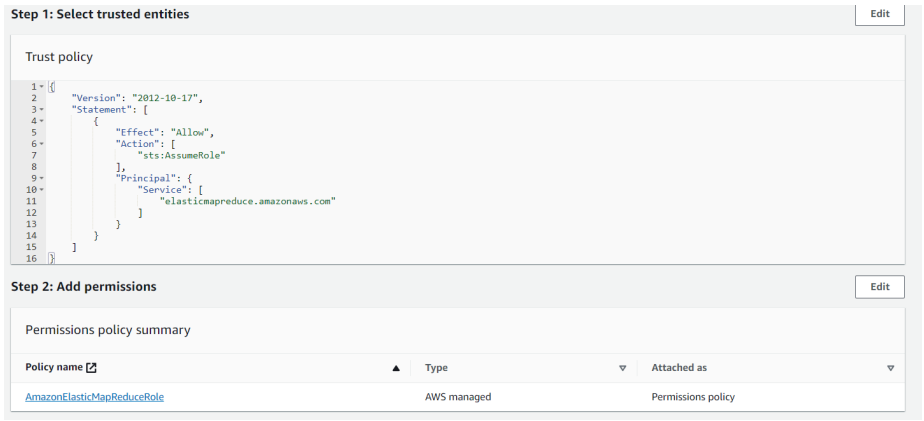

Figure 26: image

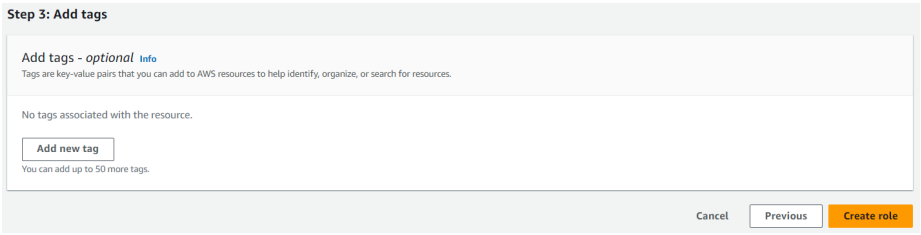

Figure 27: image

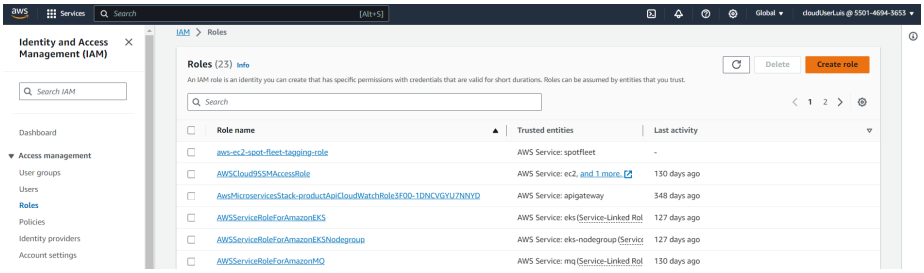

Figure 28: image

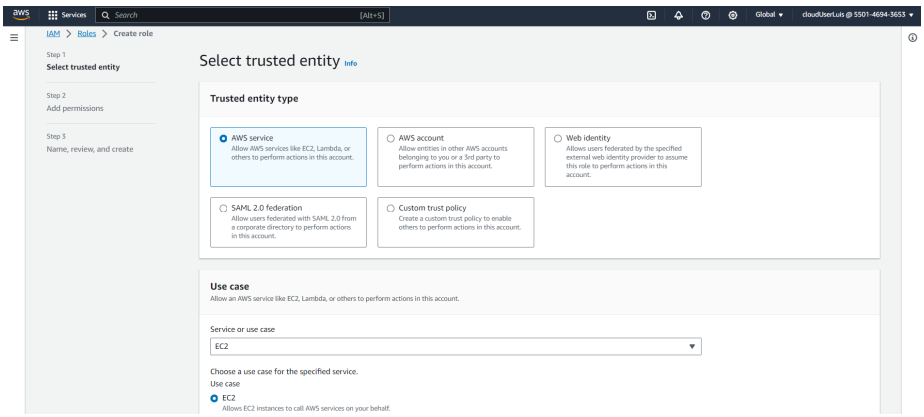

Figure 29: image

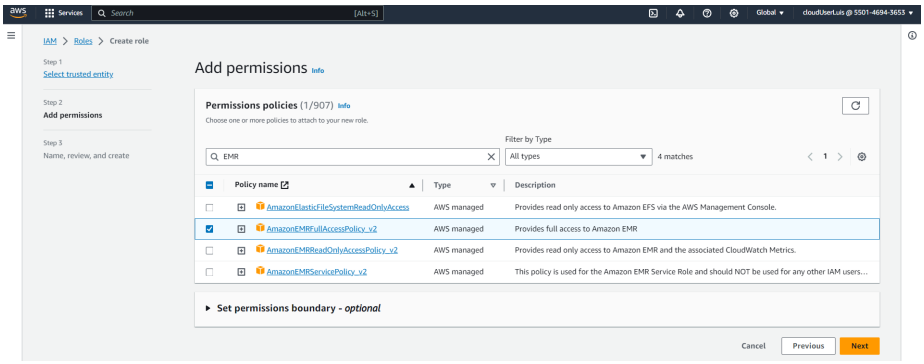

Figure 30: image

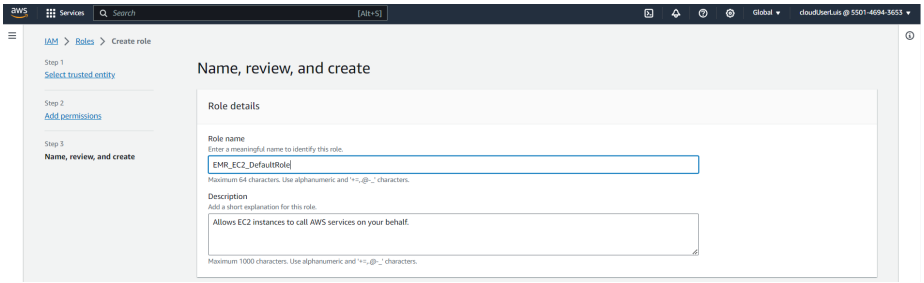

Figure 31: image

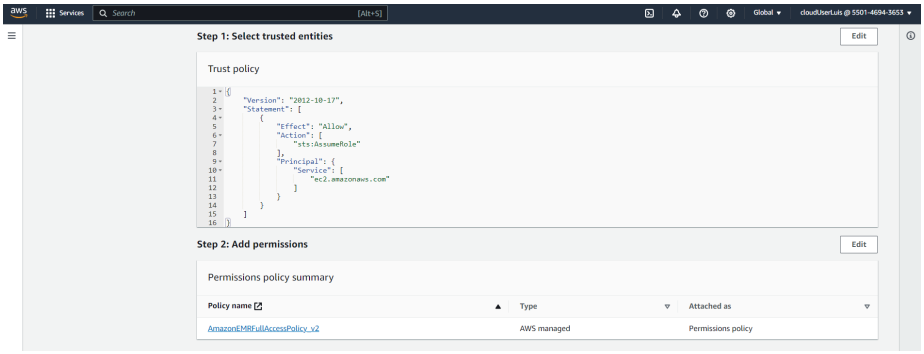

Figure 32: image

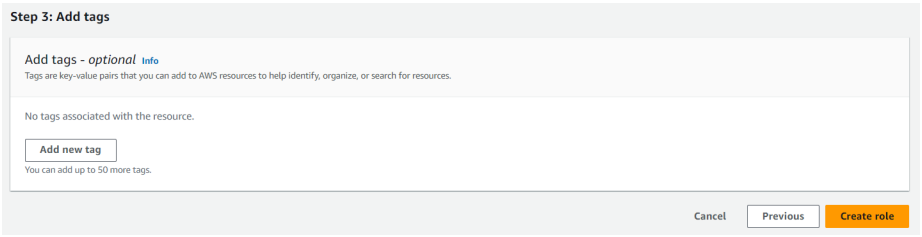

Figure 33: image

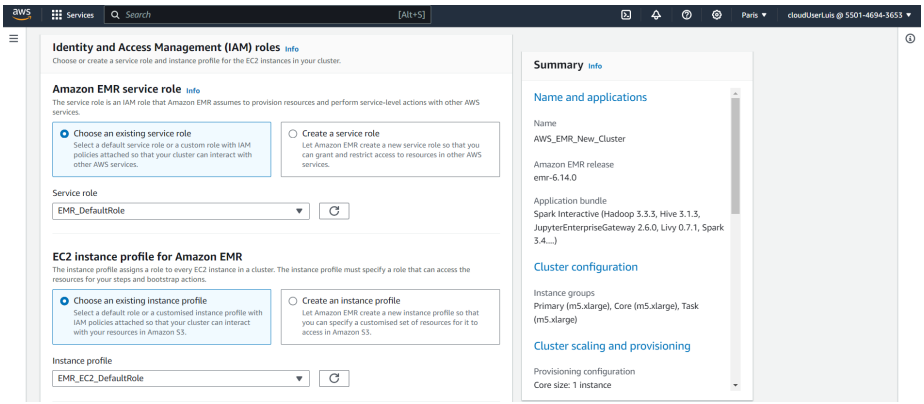

Figure 34: image

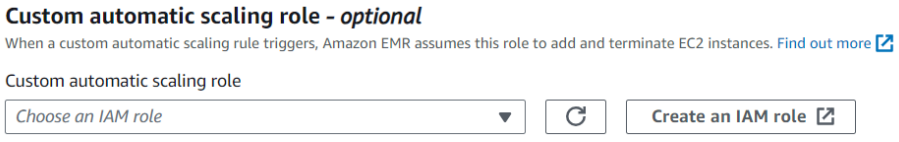

Figure 35: image

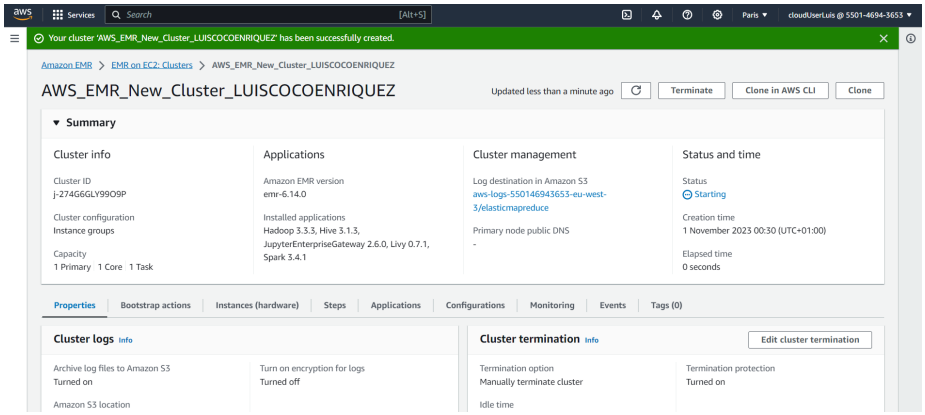

Figure 36: image

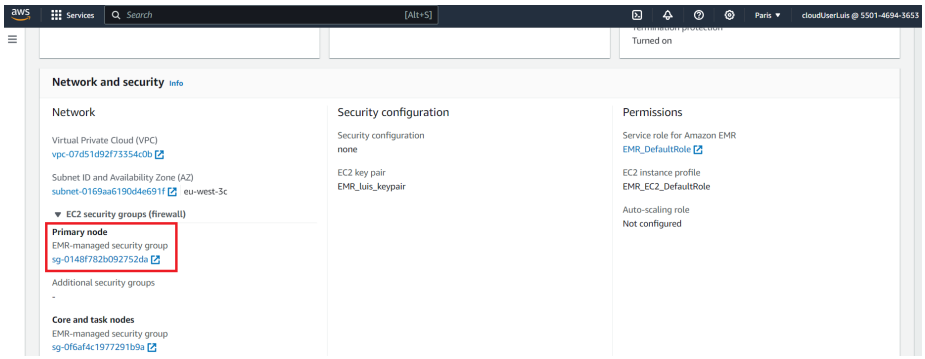

Figure 37: image

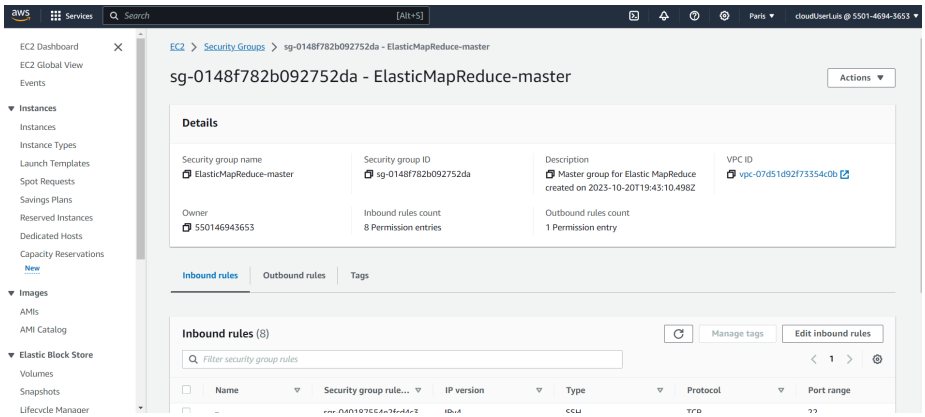

Figure 38: image

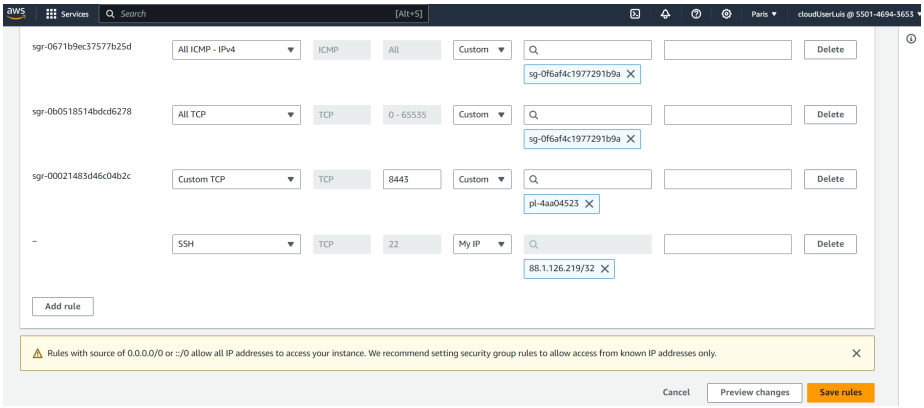

Figure 39: image

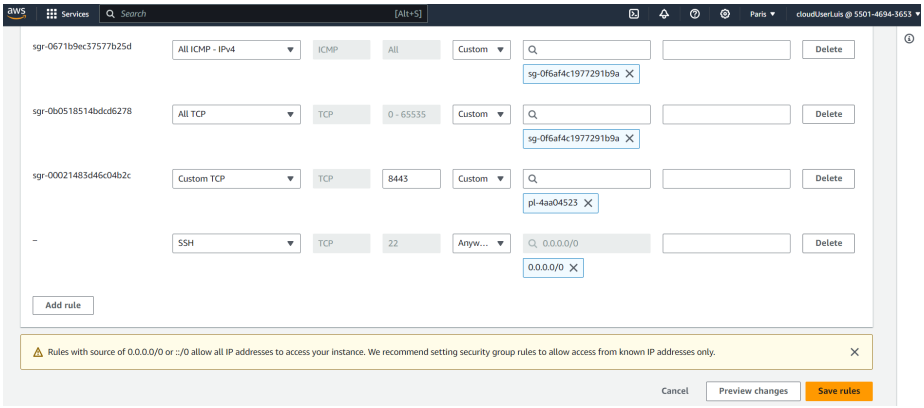

Figure 40: image

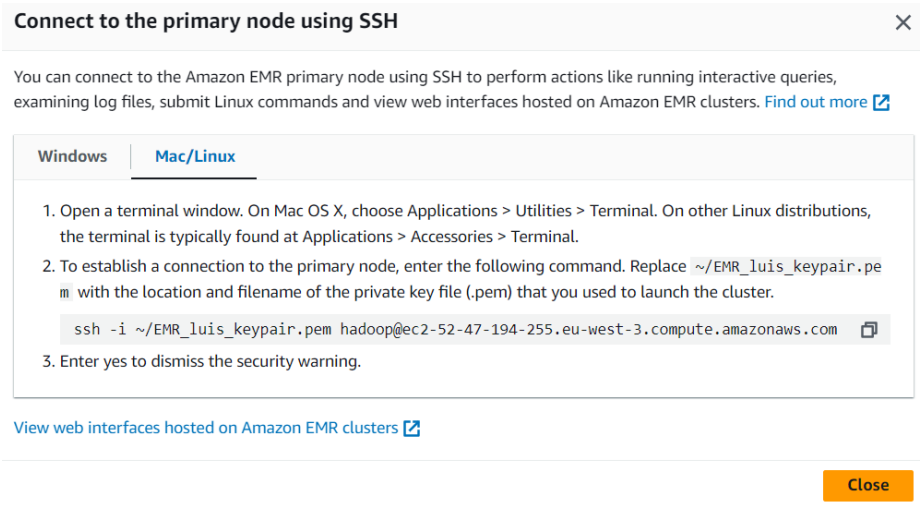

Figure 41: image

7.16. Connect to the "Primary node" using SSH

Tenemos dos opciones conexión en Windows

Or connect with Mac or Linux

8. AWS EMR - Configuración de Putty para conexión con el Nodo Primario en Windows

First we select the **Session** and we set the Host name "hadoop@ec2-52-47-194-255.eu-west-3.compute.amazonaws.com", the protocol SSH and the port 22

We select the menu option **SSH->Auth->Credentials** and we upload the **ppk** file, previously generated when creating the Key-Pair in EC2 AWS

We set the Session name and press the **Save** button

Finally we press the **open** button to connect to the AWS EMR Primary Node

The first time we connect appears the following message. Press the **Accept** button

After that the following console screen is shown (Master node console). We confirm we accessed with the **hadoop** user and we authenticated with the **Key-Pair**

Then we can execute the **sudo yum update** command to update the installed packages in the Linux console

```
sudo yum update
```

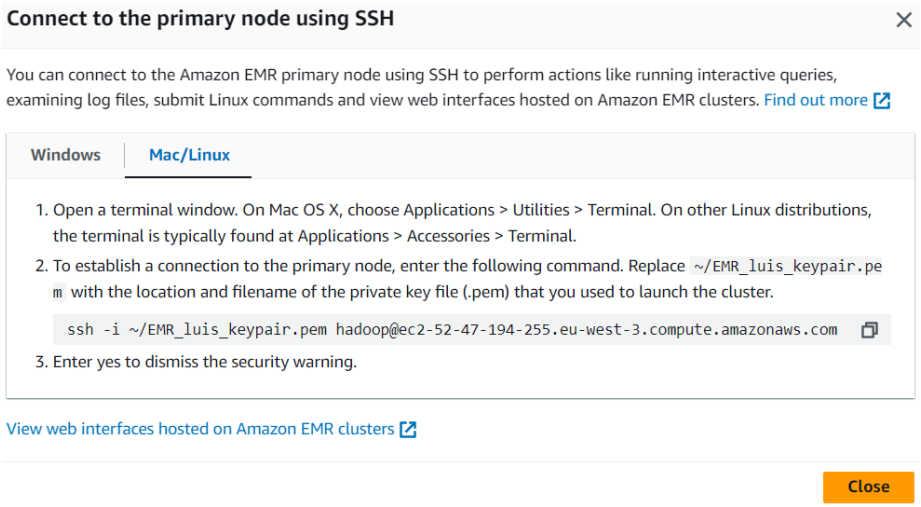

Figure 42: image

This command is used on Linux systems, specifically those using the yum package manager. Let's break it down:

sudo: This stands for "superuser do" and is used to execute commands with elevated privileges.

It's often required for system-level operations.

yum: This is the package manager used by Red Hat-based Linux distributions, such as Fedora and CentOS.

It's used for installing, updating, and removing packages on the system.

update: This is the specific command you're giving to yum. When you run sudo yum update, you're telling yum to update all installed packages to their latest versions.

It checks the repositories configured on your system for newer versions of packages and installs them.

So, in summary, sudo yum update is a command to update all the software packages on your system to the latest available versions.

It's a good practice to run this command periodically to ensure that your system is up to date with the latest security patches and feature updates.

9. Run the Scala and Spark commands in AWS EMR cluster

First we see the "spark-submit" command version

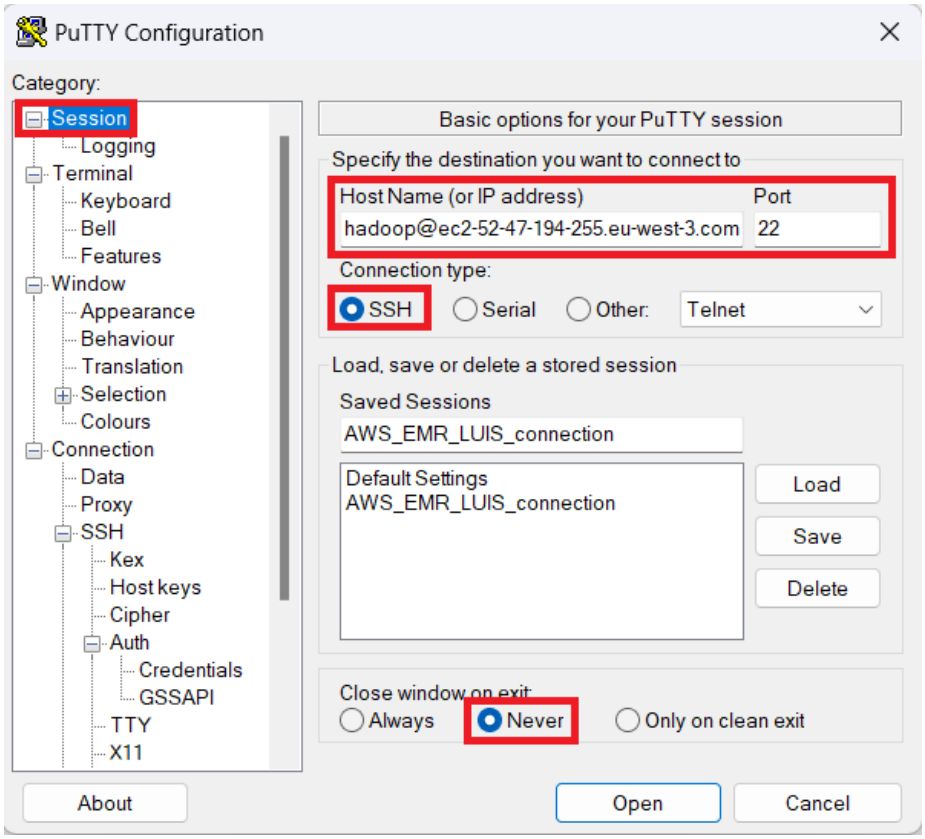

Figure 43: image

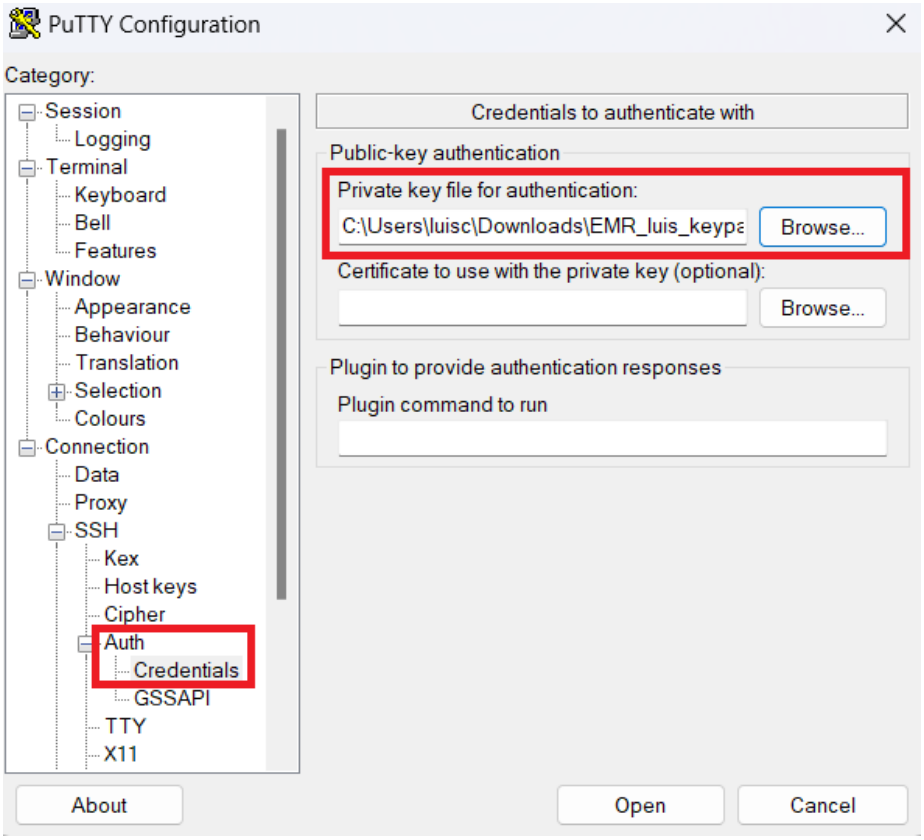

Figure 44: image

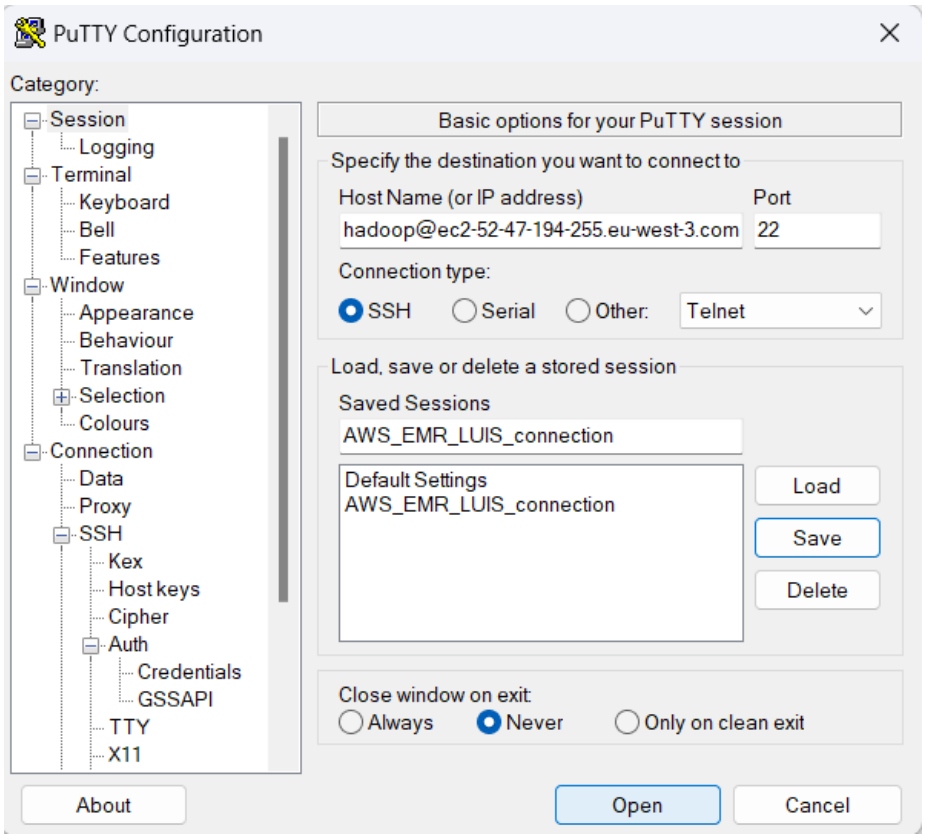

Figure 45: image

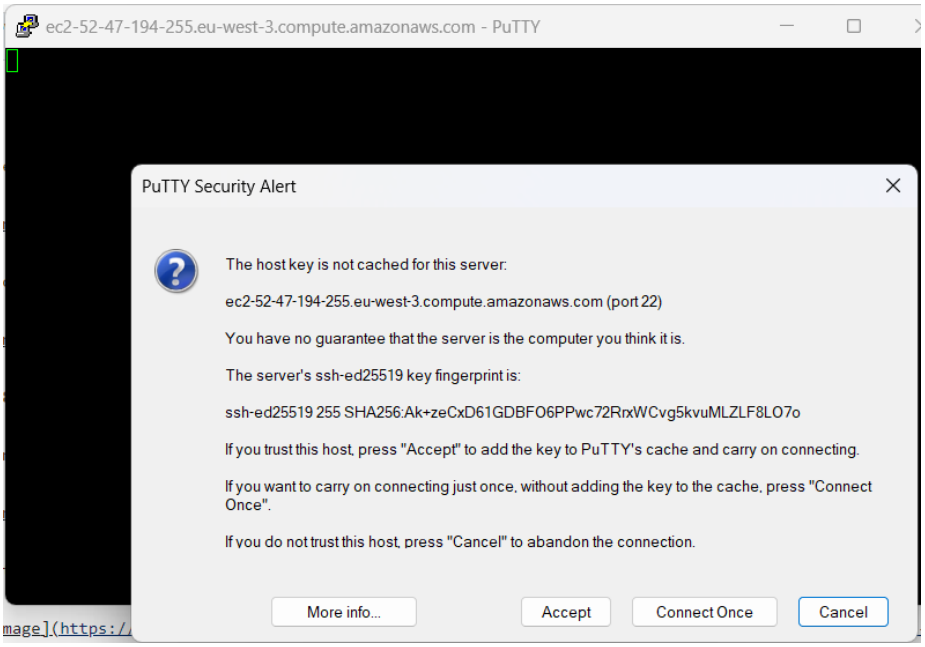

Figure 46: image

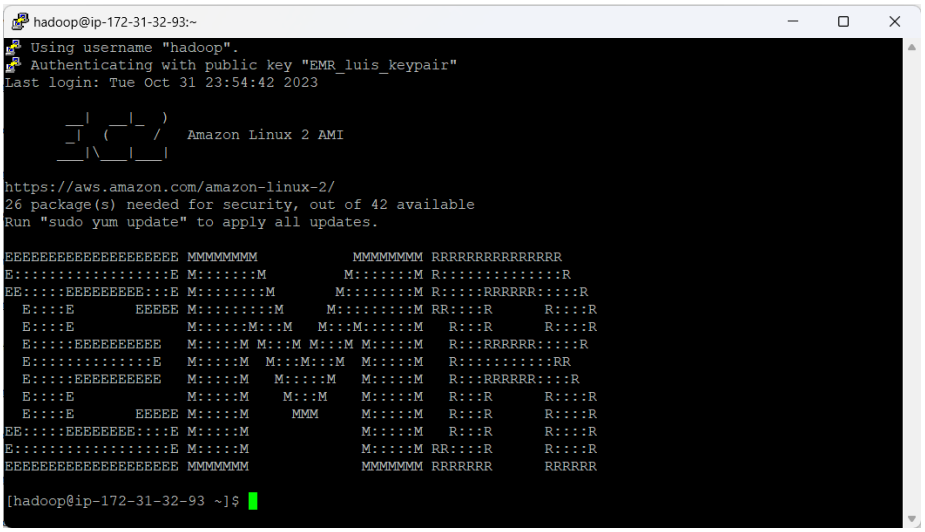

Figure 47: image

```
spark-submit --version
```

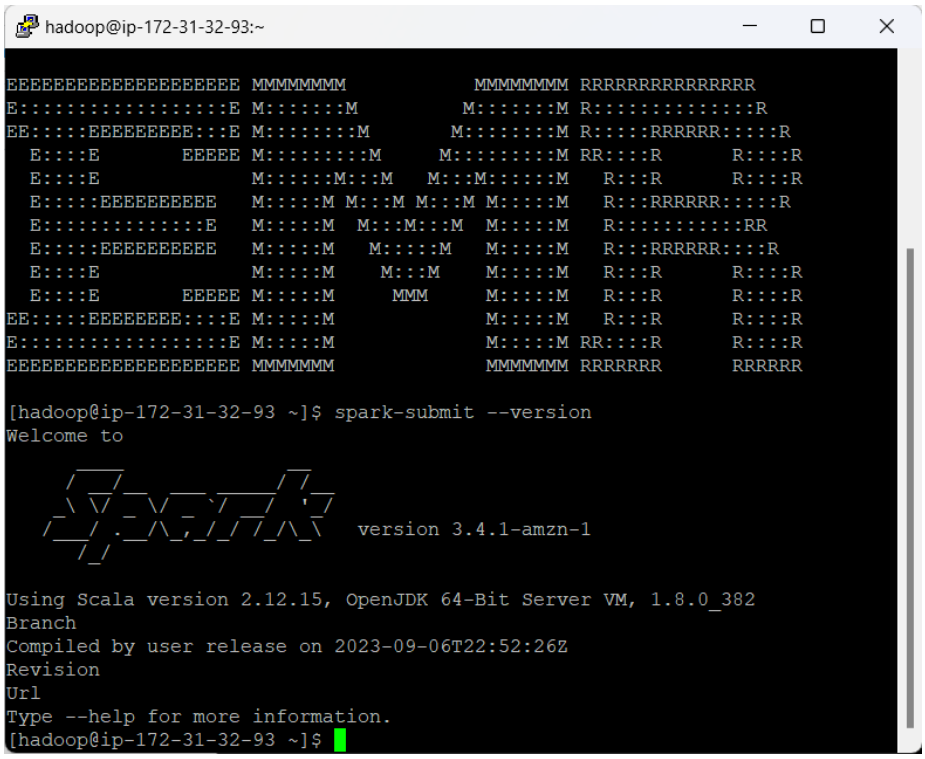

Figure 48: image

Then we run the command “spark-shell” command

```
spark-shell
```

10. Create AWS EMR Cluster Using AWS CLI and Submit job

Create an IAM user

Configure and install AWS CLI

Create an EMR cluster using AWS CLI

Create Pyspark code or Job

Submit job using AWS CLI

Submit job using Primary node

Check output in s3 bucket

Figure 49: image

NOTA IMPORTANTE! see the ZIP file in this Github repo with a real example

Youtube video: https://www.youtube.com/watch?v=XsWnW7-8IGQ

These are the steps to follow in order to **create and run an AWS EMR cluster uring AWS CLI**:

10.1 Create new IAM user

10.2. Create Access Key ID and Access Key

We create an Access Key ID and Access Key for using later during AWS CLI configuration.

10.3. Install and configure AWS CLI

Type the command:

```
aws configure
```

Enter AWS Access Key ID and access key

And Region : eu-west-3

After configuring AWS CLI you can test it running the commands:

List all IAM user

```
aws iam list-users
```

List all buckets in s3

```
aws s3 ls
```

- To see the AWS CLI version, we run the command

```
aws --version
```

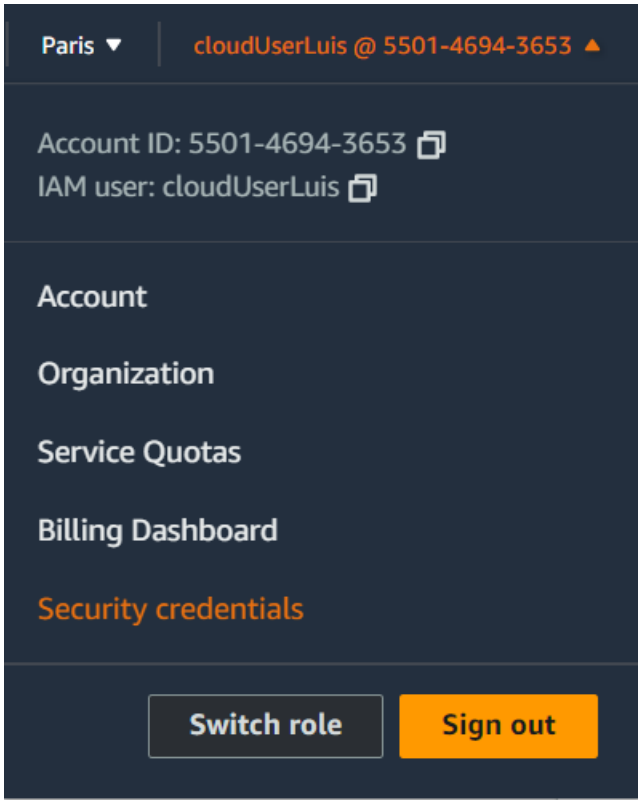

Figure 50: image

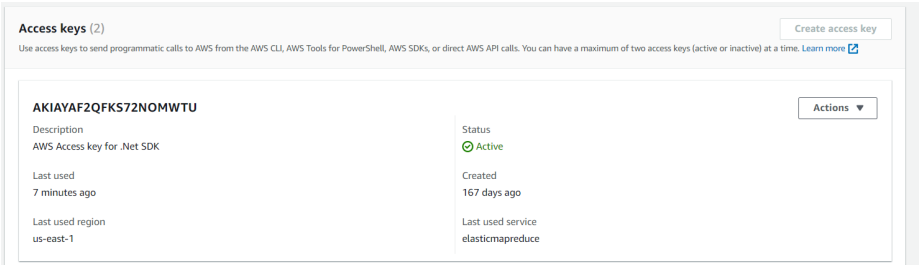

Figure 51: image

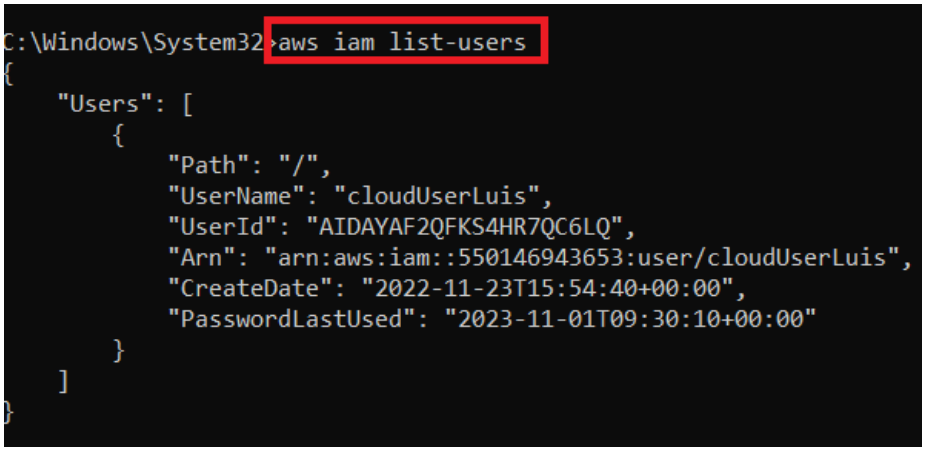

Figure 52: image

Figure 53: image

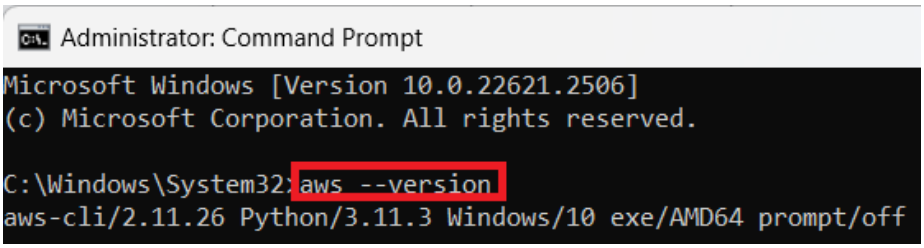

Figure 54: image

10.4. Create a Key-Pair in AWS EC2 (See section 7.2)

10.5. Create a S3 bucket

Create a AWS S3 bucket “myemrproject” and inside several folders: “input”, “logs”, “scripts”

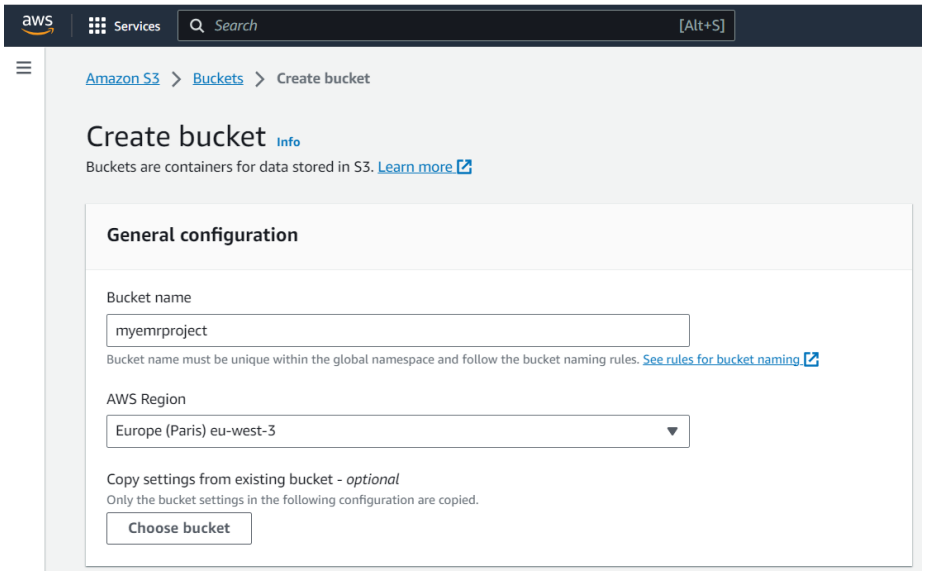

Figure 55: image

Upload to the “input” folder the input data “product_data.csv”

Upload to the “scripts” folder the application source code “mypyspark-script_1.py”

10.6. Create AWS EMR cluster

Create an **AWS EMR cluster** (previously create the AIM roles: EMR_DefaultRole and EMR_EC2_DefaultRole)

```
aws emr create-cluster --name MyEMRCluster --use-default-roles --release-label emr-6.11.0 --
```

This is a command-line interface (CLI) command for AWS Elastic MapReduce (EMR) to create a new EMR cluster.

This command essentially creates an EMR cluster named “MyEMRCluster” with one m5.xlarge instance, running **Spark** and **Hadoop** applications, using default roles, and storing logs in the specified S3 bucket.

aws emr create-cluster: This is the main command to create an EMR cluster using the AWS CLI.

–**name MyEMRCluster**: This sets the name of the EMR cluster to "MyEMRCluster."

–**use-default-roles**: This option specifies that the command should use the default roles for EMR, which are necessary for managing and running EMR clusters.

–**release-label emr-6.11.0**: Specifies the EMR release version to use. In this case, it's version 6.11.0.

–**instance-count 1**: Sets the number of EC2 instances in the cluster to 1.

–**instance-type m5.xlarge**: Specifies the type of EC2 instances to use in the cluster. In this case, it's m5.xlarge.

–**applications Name=Spark Name=Hadoop**: Specifies the applications to be installed on the EMR cluster. In this case, it's Apache Spark and Apache Hadoop.

–**ec2-attributes SubnetIds=subnet-0169aa6190d4e691f,KeyName=EMR_luis_keypair**: Specifies EC2 instance attributes. It specifies the subnet ID where the instances should be launched and the key pair to use for SSH access.

–**log-uri s3://aws-logs-550146943653-eu-west-3/elasticmapreduce**: Specifies the Amazon S3 URI where the EMR logs should be stored. EMR logs contain information about the cluster's execution.

10.7. List and describe the AWS EMR clusters

Now we can list the AWS EMR cluster running this command

```
aws emr list-clusters
```

Using the **ClusterId** to check status and information of the cluster

```
aws emr describe-cluster --cluster-id ClusterId
```

10.8 Excute job

This command is adding a **Spark job** as a step to an existing **EMR cluster**, specifying details such as the **ClusterId**, **Spark job configuration**, and the location of the **Spark script on S3**

```
aws emr add-steps
--cluster-id ClusterId
--steps Type=Spark,Name="MySparkJob",ActionOnFailure=CONTINUE,Args=[--deploy-mode,cluster,--
```

aws emr add-steps: This is the AWS CLI command to add steps to an EMR cluster. It allows you to submit Spark applications or other types of steps to be executed on the EMR cluster.

–**cluster-id ClusterId**: This specifies the unique identifier of the EMR cluster (j-GAVB3ZN07CUB) to which you want to add the Spark job.

–steps Type=Spark,Name="MySparkJob",ActionOnFailure=CONTINUE,Args=[–deploy-mode,cluster,–master,yarn,–conf,spark.yarn.submit.waitAppCompletion=true,s3://myen
This part defines the details of the Spark job you want to submit as a step to the EMR cluster.

Type=Spark: Indicates that the step is a Spark application.

Name="MySparkJob": Specifies a name for the Spark job, in this case, "MySparkJob".

ActionOnFailure=CONTINUE: Specifies the action to take if the step fails. In this case, it continues to the next step even if this one fails.

Args=[...]: Specifies the arguments to pass to the Spark application.

Here's a breakdown of the arguments:

–deploy-mode,cluster: Sets the deploy mode of the Spark application to "cluster," meaning it will run on the EMR cluster.

–master,yarn: Specifies that **YARN** should be used as the cluster manager for Spark.

–conf,spark.yarn.submit.waitAppCompletion=true: Configures Spark to wait for the application to complete before submitting the next step. This is useful for job chaining

s3://myemrproject/scripts/mypysparkscript_1.py: Specifies the location of the Spark script (mypysparkscript_1.py) on Amazon S3. This is the script that will be executed as part of the Spark job

10.9. Submit Spark job to a Spark cluster(Primary node)

Submit Spark job or task directly to a Spark cluster(Primary node)

spark-submit is a command-line tool provided by Apache Spark to submit Spark applications or jobs directly to a Spark cluster

To submit a job we run this command:

```
spark-submit --master yarn ./mypysparkscript_1.py
```

This command is using **spark-submit**, a tool provided by Apache Spark for submitting Spark applications

spark-submit: This is the command to submit a Spark application

–master yarn: This option specifies the cluster manager to use. In this case, it's set to YARN, which is one of the cluster managers supported by Apache Spark. YARN (Yet Another Resource Negotiator) is often used in Hadoop clusters to manage resources and schedule tasks

./mypysparkscript_1.py: This is the path to the Python script (mypysparkscript_1.py) that you want to submit as a Spark application.

The ./ indicates that the script is in the current directory

This is the application source code **mypysparkscript_1.py**:

```
from pyspark.sql import SparkSession

if __name__ == "__main__":
    # Initialize SparkSession
    spark = SparkSession.builder.appName("MyPySparkJob").getOrCreate()

    try:
        # Your PySpark code here
        # Specify the input file path
        input_file = 's3://myemrproject/input/product_data.csv'
        df = spark.read.csv(input_file)
        df.show()
        df.write.option("header", "true").mode("overwrite").parquet("s3://myemrproject/outpu

        # Stop SparkSession
        spark.stop()

    except Exception as e:
        # Handle any exceptions or errors
        print("Error occurred: ", str(e))
        spark.stop()
```

VERY IMPORTANT NOTE:

How to copy the script file "mypysparkscript_1.py" to the AWS EMR Master node when I want to execute this command

spark-submit –master yarn ./mypysparkscript_1.py

You can use the aws emr **cp** command to copy your script file to the master node of your EMR cluster before running spark-submit

```
aws emr cp mypysparkscript_1.py s3://your-s3-bucket/path/to/scripts/
```

Replace **your-s3-bucket** with the name of your **S3** bucket and adjust the path as needed.

This command will upload your script to the specified S3 location.

After uploading, you can use spark-submit with the script file on the master node:

```
spark-submit --master yarn s3://your-s3-bucket/path/to/scripts/mypysparkscript_1.py
```

This assumes that your EMR cluster has the necessary **permissions** to access the S3 bucket.

If your script is in a different local directory, you can replace **s3://your-s3-bucket/path/to/scripts/** with the appropriate local path

10.10. Create an IAM role for an EMR cluster with the necessary permissions to access an S3 bucket

To create an IAM role for an EMR cluster with the necessary permissions to access an S3 bucket, you can follow these general steps:

- Sign in to the **AWS Management Console**:
- Log in to your AWS account and navigate to the IAM Console
- Create a new IAM Role:
- In the IAM Console, click on "**Roles**" in the left navigation pane
- Click the "**Create role**" button
- Choose a use case:
- Select "**AWS service**" as the type of trusted entity
- Choose "**EMR**" from the list of use cases

Attach permissions policies:

In the permissions policies step, attach policies that grant the necessary permissions

At a minimum, you should attach the **AmazonS3FullAccess** policy to allow full access to S3

You can create a custom policy with the specific permissions required if you want more fine-grained control

Give your role a meaningful name and description

Click "**Create role**" to finish the process

Note the **Role ARN**:

After creating the role, note the Role ARN as you will use this when launching your EMR cluster

Launch EMR Cluster with IAM Role:

When creating your EMR cluster, either through the AWS Management Console or using the AWS CLI, specify the IAM role you created in the "Edit software settings" section

Here's an example CLI command to create a cluster with a specified IAM role:

```
aws emr create-cluster --name "MyEMRCluster" --release-label emr-6.4.0 \
--applications Name=Spark Name=Hadoop \
--ec2-attributes KeyName=my-key-pair \
```

```
--instance-type m5.xlarge --instance-count 3 \
--use-default-roles --ec2-attributes KeyName=my-key-pair \
--service-role arn:aws:iam::123456789012:role/YourEMRRole
```

Replace "**YourEMRRole**" with the name of the IAM role you created

This IAM role will have the necessary permissions to access the S3 bucket specified in your Spark job or EMR steps

10.11. Terminate the AWS EMR Cluster

When finishing your job run this command to terminate your AWS EMR cluster:

```
aws emr terminate-clusters --cluster-id ClusterId
```

10.12. What is the difference between "aws emr add-steps" and "spark-submit" commands

aws emr add-steps is a command used to add a step to an Amazon EMR (Elastic MapReduce) cluster

It's a higher-level command provided by AWS to interact with their EMR service

The example you've given is using this command to add a Spark step to an EMR cluster

The step configuration includes details like the script location, Spark configuration, and other parameters

spark-submit is a Spark-specific command used to submit a Spark application to a cluster

In summary, the **aws emr add-steps** command is more focused on interacting with EMR clusters and adding steps to them, while **spark-submit** is specifically for submitting Spark applications to a cluster

The **aws emr add-steps** command might involve additional AWS-specific configurations and could be part of a larger EMR workflow

May I use "spark-submit" without previously using "aws emr add-steps"? Absolutely! **spark-submit** is a standalone command and doesn't require the use of aws emr add-steps before using it

You can submit Spark applications directly using **spark-submit** without adding steps to an EMR cluster using the AWS EMR API

Here's a breakdown:

Use **aws emr add-steps** when you want to programmatically add steps (like Spark jobs) to an existing EMR cluster through the AWS EMR API

This is useful when you want to automate job submissions as part of a larger workflow

Use spark-submit when you just want to submit a Spark application to a cluster manually. This is a more direct way of submitting Spark jobs and is often used for testing and interactive development

So, you can choose the approach that best fits your use case

If you're automating job submissions to an EMR cluster, **aws emr add-steps** might be more suitable. If you just want to run a Spark job on an EMR cluster manually, you can use **spark-submit** directly