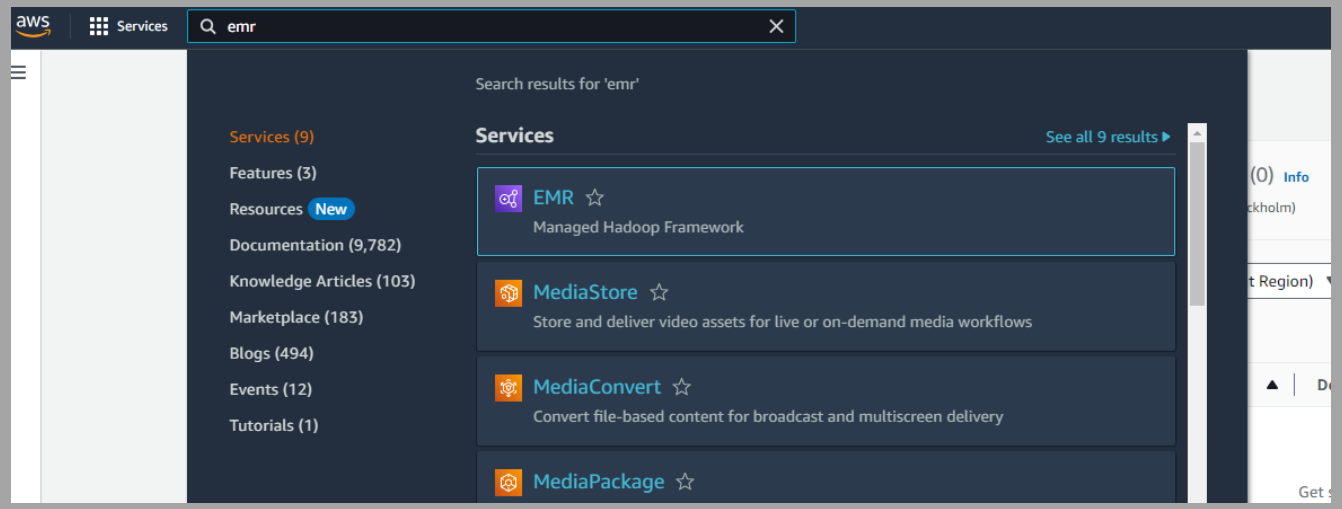
AWS **E**lastic**M**ap**R**educe Data Processing

Scala

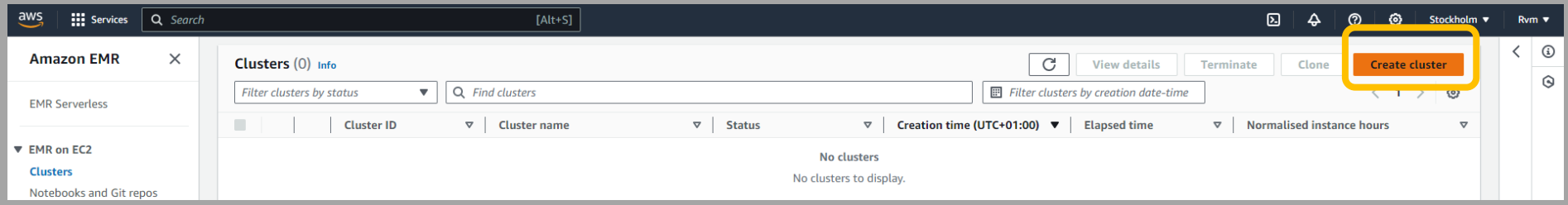
Create an EMR cluster

January 2024

1. Find and Access to EMR service

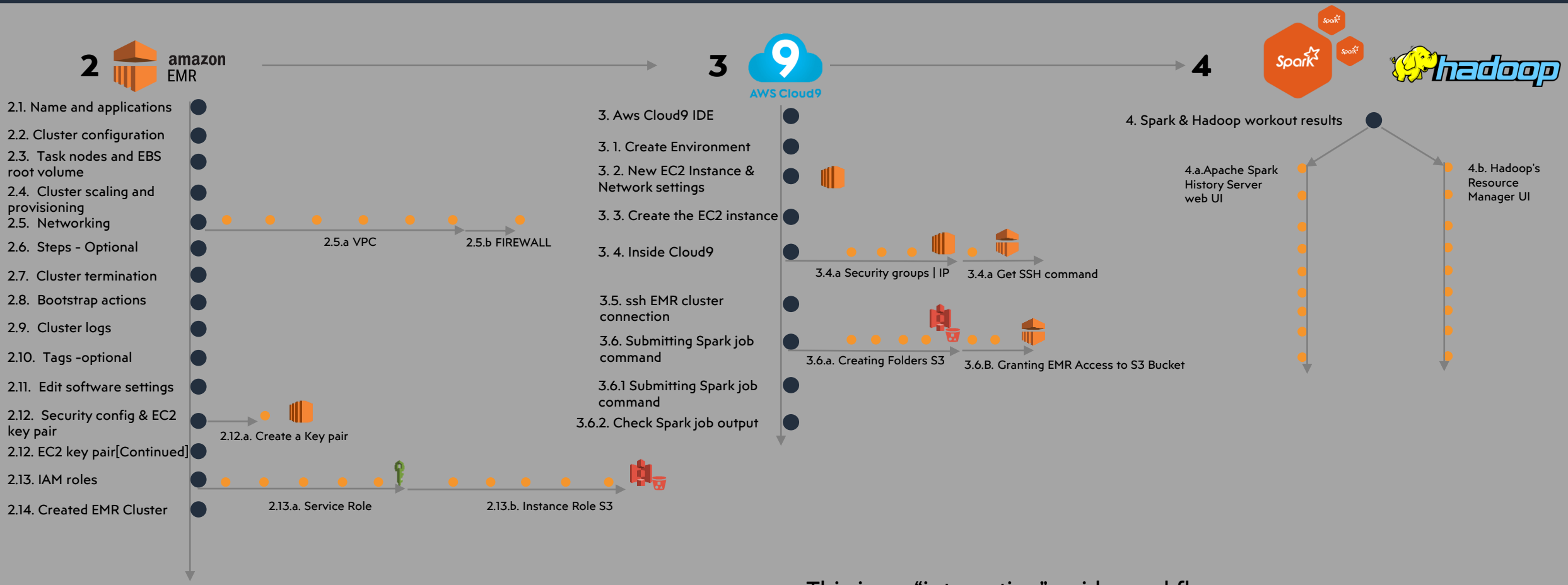


2. Create the EMR Cluster



Guide workflow

January 2024

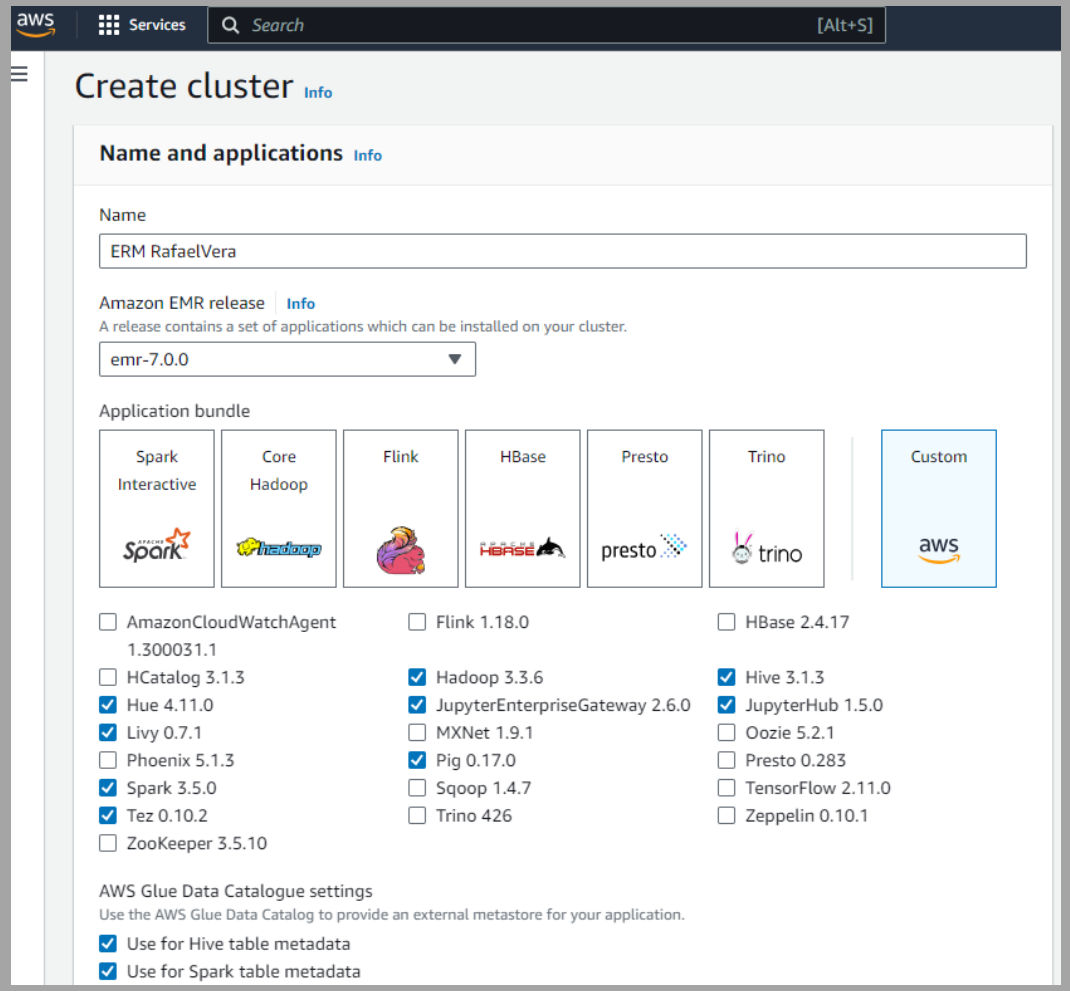


This is an “interactive” guide workflow.
By clicking each step you will be redirected there. You can come back and forth here clicking on the aws logo on the upper right corner in every slide

Create an EMR cluster

January 2024

2.1. Name and applications



EMR Release:
emr-7.0.0 - A managed cluster platform that simplifies running big data frameworks.

Custom Applications:
Enables a personalized selection of applications and versions for project-specific needs.

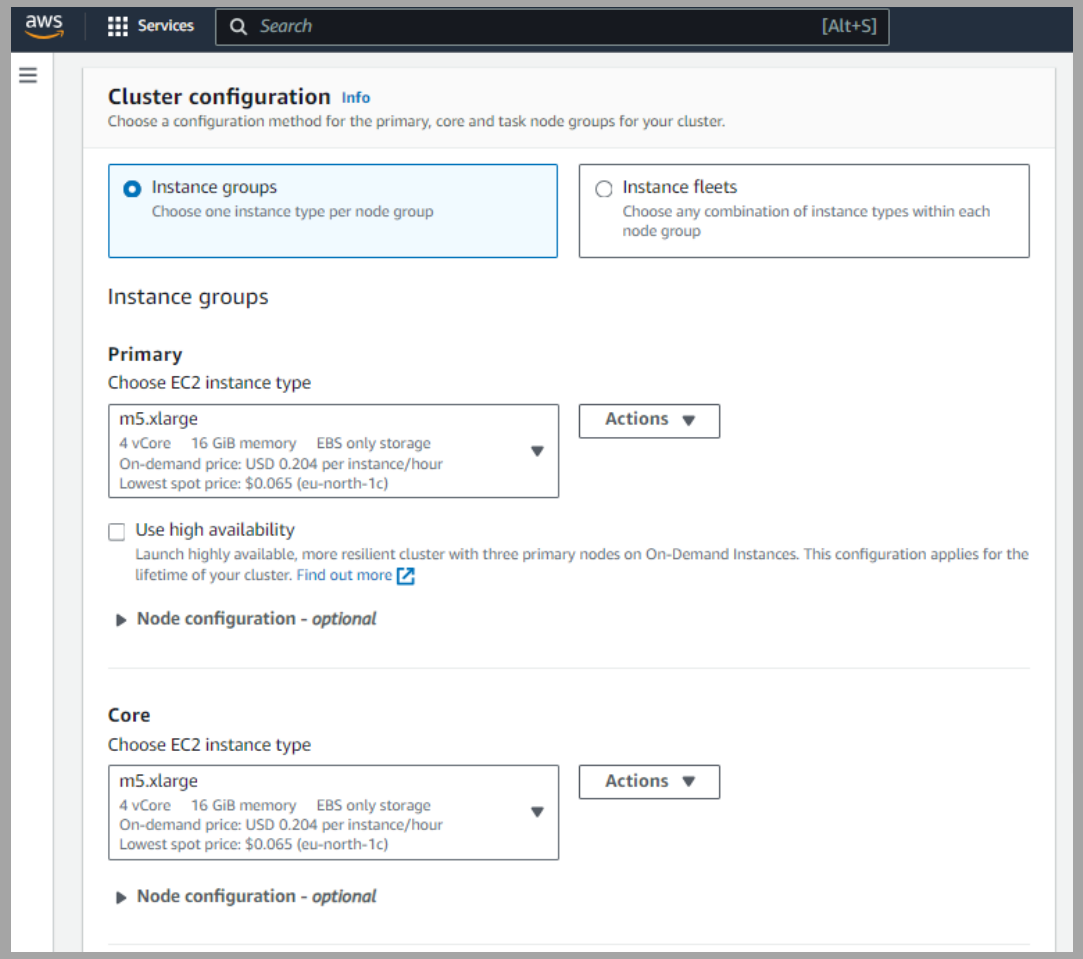
AWS Glue Data Catalogue:
Utilized for storing Hive and Spark table metadata, serving as a central metadata repository.

Application Bundle:
Custom package including Hadoop 3.3.6, Hive 3.1.3, Hue 4.11.0, and Jupyter interfaces.

Create an EMR cluster

January 2024

2.2. Cluster configuration



This section allows to choose the configuration method for the primary, core, and task node groups of the cluster.

Instance Groups:
Where one selects a single EC2 instance type for the primary and core node groups.

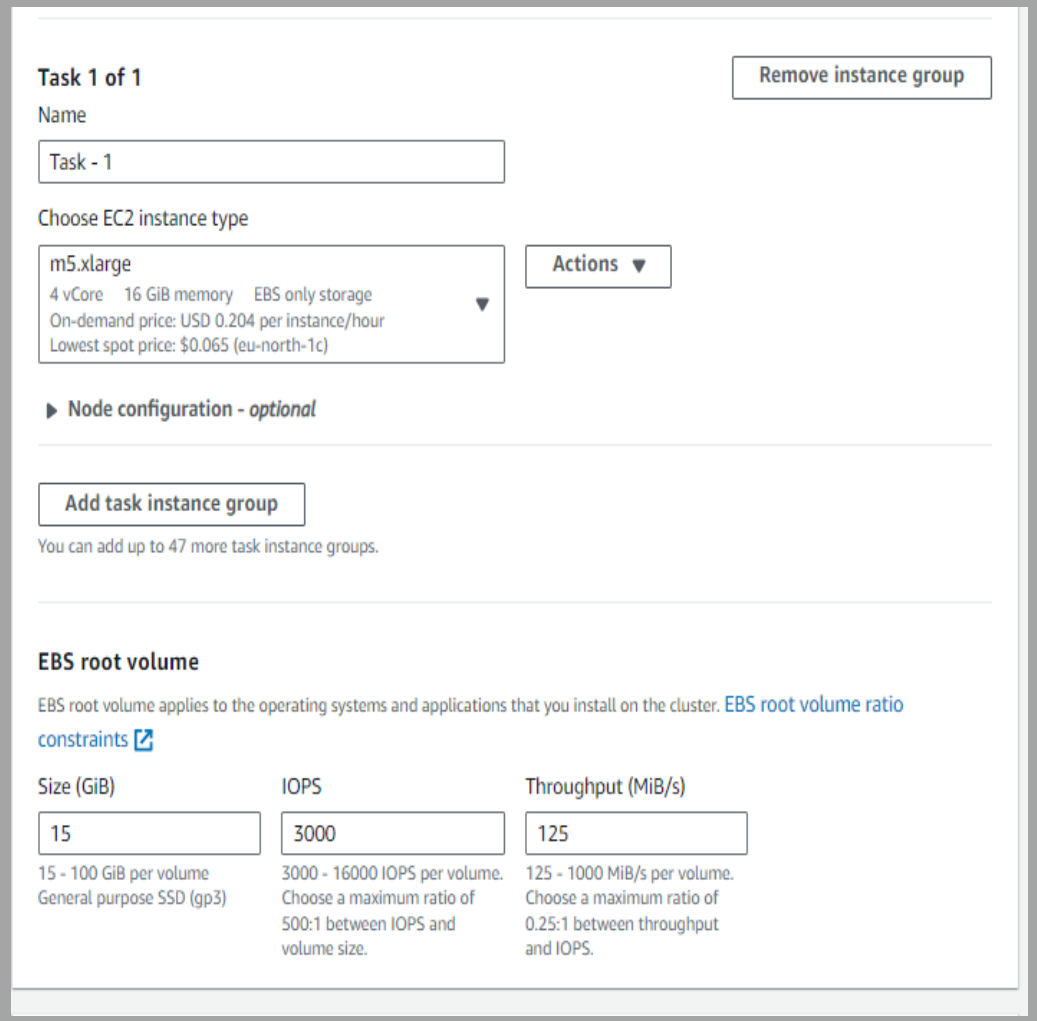
Primary:
m5.xlarge. Use high availability, which means launching a more resilient cluster with three primary nodes on On-Demand Instances.

Core:
Same config as Primary group.

Create an EMR cluster

January 2024

2.3. Task nodes and EBS root volume



Task nodes:
are utilized to boost computational power for parallel data processing without holding persistent data.

They offer scalability to handle fluctuating workloads, cost efficiency via cheaper spot instances, flexibility in instance selection, and have a non-disruptive nature as their removal doesn't affect the cluster's stored data, making them suitable for temporary spot usage.

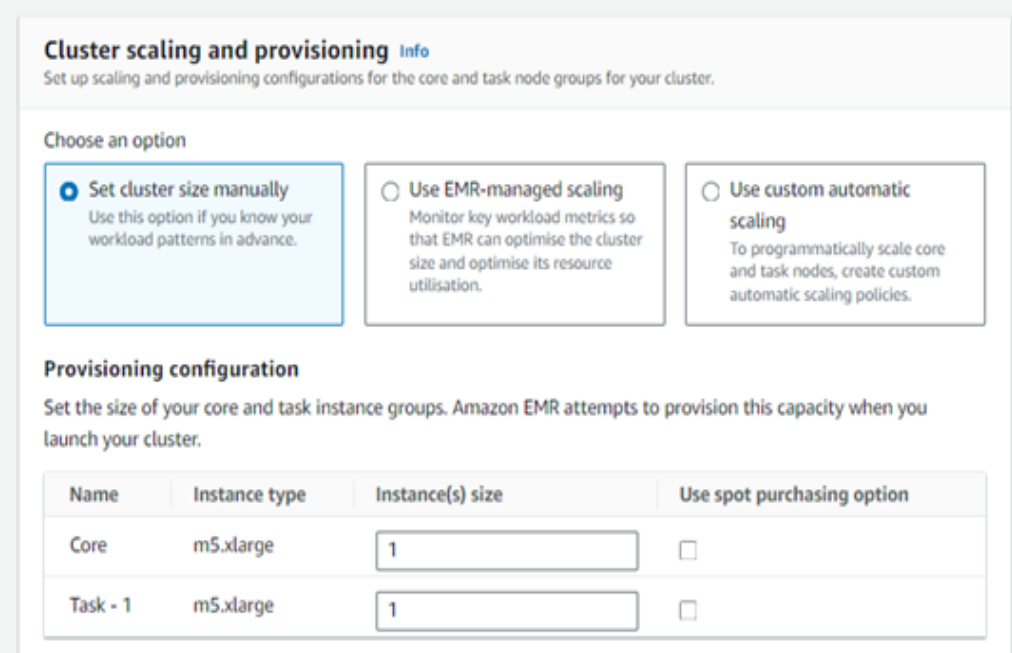
Elastic Block Store (EBS) root volumen:
act as the primary storage for the operating system and applications on the cluster nodes.

Size: indicating the storage capacity of the root volume.
IOPS: Input/Output Operations Per Second, a measure of the volume's performance. How many read/write operations the volume can handle.
Throughput: the rate at which data can be read from or written to the storage volume.

Create an EMR cluster

January 2024

2.4. Cluster scaling and provisioning



Scaling option:
Set cluster size manually:
when you know your workload patterns in advance and want to specify the cluster size yourself.

Provisioning configuration:
allows you to set the size of your core and task instance groups, which Amazon EMR will attempt to provision when launching your cluster.

Instance Type:
Both the 'Core' and 'Task - 1' groups are set to use m5.xlarge instances.

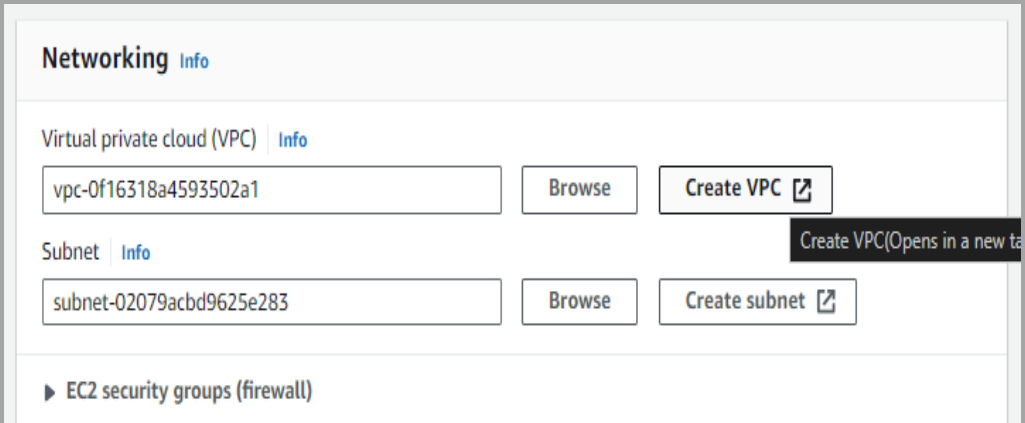
Instance(s) Size: Indicates the number of instances for each group.

Use spot purchasing option:
is not checked in the image, allows the option to purchase spot instances for cost savings.

Create an EMR cluster

January 2024

2.5. Networking



Virtual private cloud (VPC)
is a logically isolated section of the AWS Cloud where you can launch AWS resources in a virtual network that you define.
VPC ID is a unique identifier for a specific VPC.

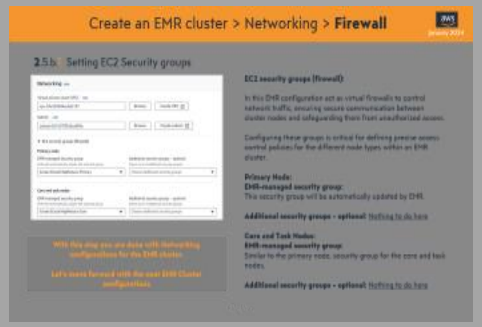
Subnet:
is a range of IP addresses in the VPC.

Setting the VPC and subnet for an EMR cluster is crucial as it determines the virtual network isolation and dictates the IP range and availability zone, ensuring secure and optimized network traffic for the cluster's computational resources within the AWS ecosystem

EC2 Security Groups (firewall):
configure security groups for the service.
Security groups in AWS act as a virtual firewall for instances to control inbound and outbound traffic

1st. Create VPC (a)

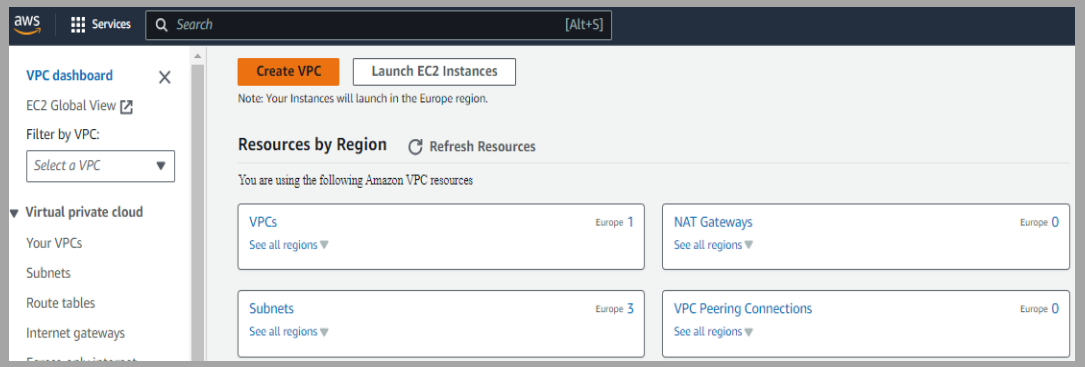
2nd. Create firewall (b)



Create an EMR cluster > Networking > **VPC**

January 2024

2.5.a.1 Create VPC



AWS VPC dashboard from the AWS Management Console:

This dashboard is the central location for managing and configuring Amazon Virtual Private Clouds (VPCs).

It is the starting point for creating and managing network resources that will be associated with an EMR cluster.

Create VPC Button:
for initiating the creation of a new VPC. **Click on it only**

Launch EC2 Instances:
you can launch new EC2 instances.

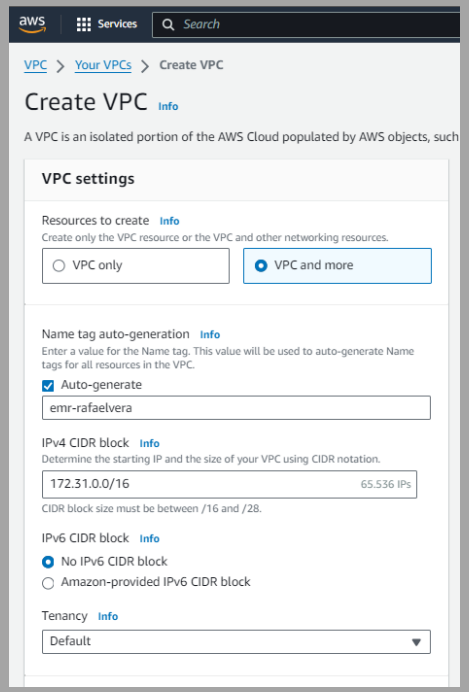
Resources by Region:
Summary of the current resources. The count of VPCs and subnets in the Europe 1 region, and the count of NAT Gateways and VPC Peering Connections.

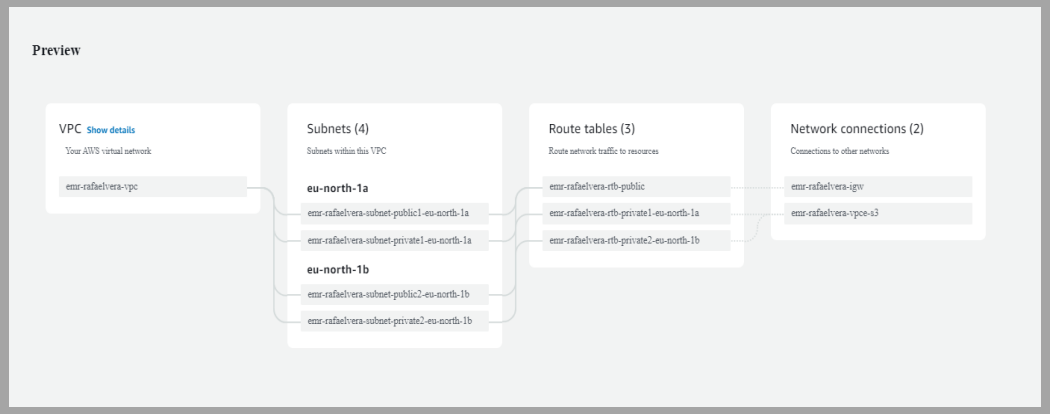
Filter by VPC:
dropdown to filter resources by specific VPCs.

Create an EMR cluster > Networking > **VPC**

January 2024

2.5.a.2 New VPC Settings





Name tag:
Enter a name tag for the new VPC. It helps in identifying the VPC within the AWS environment.

IPv4 CIDR block:
Defines the IP address range of the VPC. **Fill it with the IP on the image.**

IPv6 CIDR block:
choose an Amazon-provided IPv6 CIDR block or not assign one.

Pv4 and IPv6 CIDR blocks specify the IP address ranges for networks, with IPv4 being mandatory for current networks and IPv6 addressing the growing need for more IP addresses.

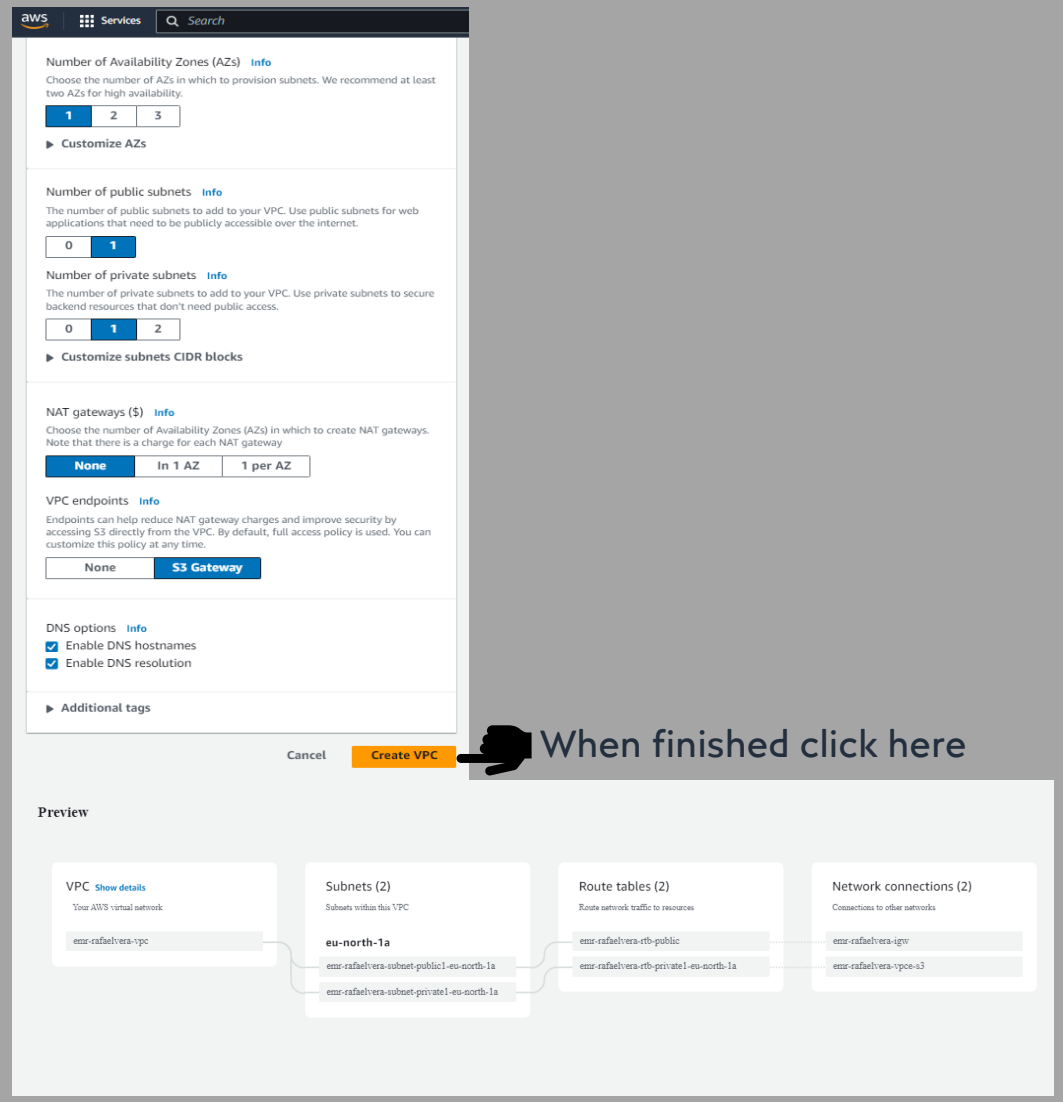
Tenancy:
Choose tenancy of instances launched in the VPC.
"Default" tenancy means instances run on shared hardware.

Preview Pane:
displays the impending creation of a VPC, subnets split between public and private across availability zones, route tables for network routing, and network connections for potential internet or VPC peering. (**this will change in the next step**)

Create an EMR cluster > Networking > **VPC**

January 2024

2.5.a.3 New VPC Settings [Continued]



Number of Availability Zones (AZs) :
Selection of how many Availability Zones to use within the VPC for high availability. (**1**)

Customize AZs: Nothing to do here.

Number of public subnets (**1**) and **Number of private subnets** (**1**)

Customize Subnets CIDR Blocks: Nothing to do here.

NAT Gateway(s):
to be created along with the desired AZ for deployment. **NONE**

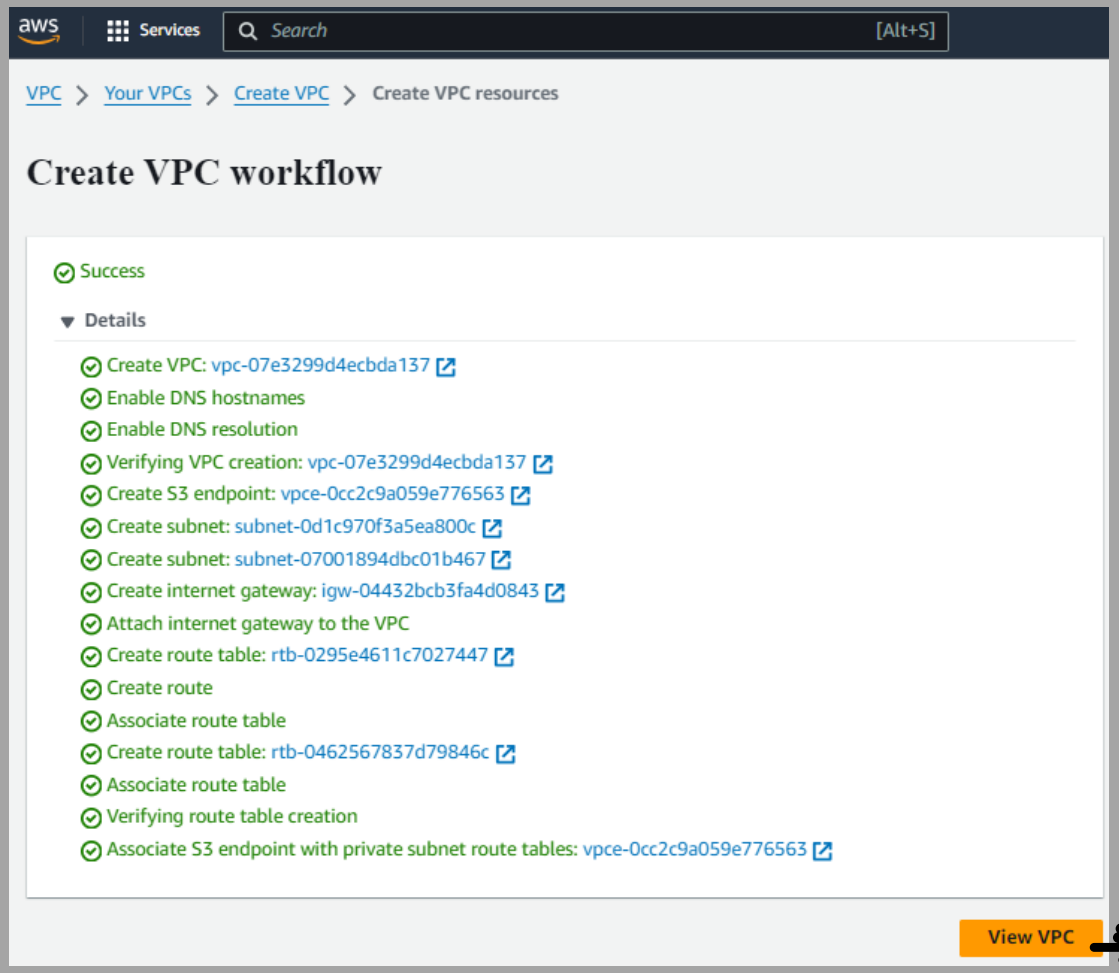
VPC Endpoints creation:
Allow private connections to AWS services. **S3 Gateaway**

DNS Options:
enable DNS hostnames and DNS resolution within the VPC.

Additional Tags: Nothing to do here.

Preview Pane:
two subnets designated as public and private across different availability zones, two route tables for network routing, and two network connections (external/internal)

2.5.a.4 VPC Create workflow



Create VPC workflow" summary:

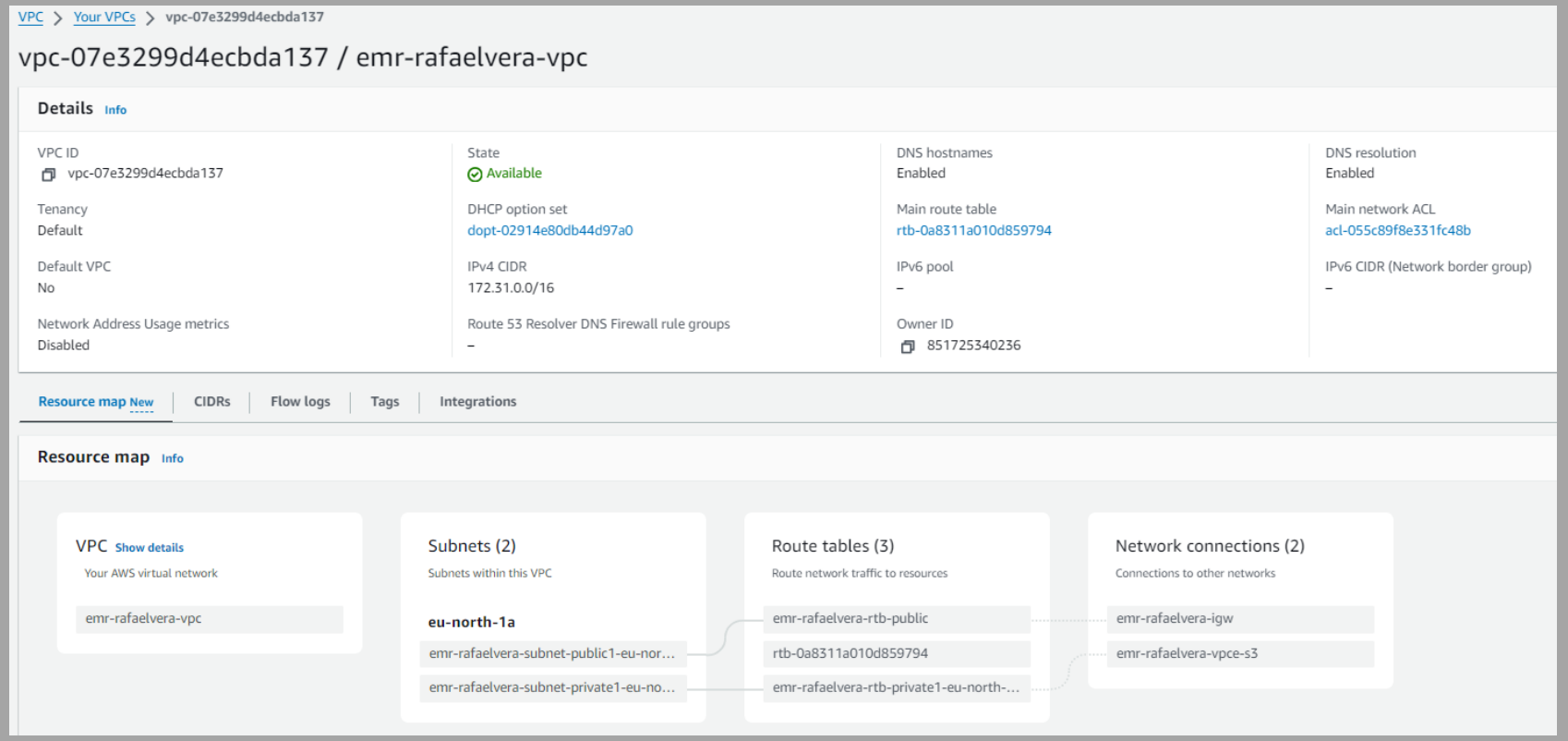
The list confirms the steps taken to set up a VPC with necessary components for network connectivity, access to AWS services via endpoints, and proper routing.
At the bottom, there's a **"View VPC"** button, to inspect the newly created VPC configuration.

Create VPC: A VPC has been created with a specific ID.
Enable DNS hostnames: has been enabled.
Enable DNS resolution: has been activated.
Verifying VPC creation: The creation of the VPC has been verified.
Create S3 endpoint: has been established within the VPC.
Create subnets: Two subnets have been created.
Create internet gateway: has been created and attached to the VPC.
Create route table: have been created and associated with the VPC.
Associate S3 endpoint with private subnet route tables: has been associated.

Create an EMR cluster > Networking > **VPC**

January 2024

2.5.a.5 Created VPC Overview



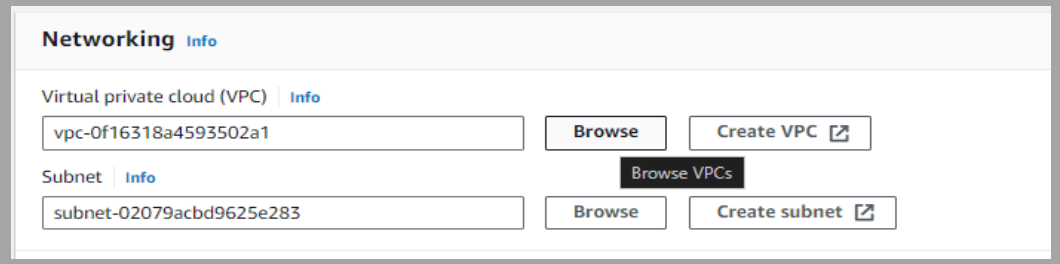
This overview allows to verify the components of the VPC are set up as intended, and to navigate to further configuration details for each component.

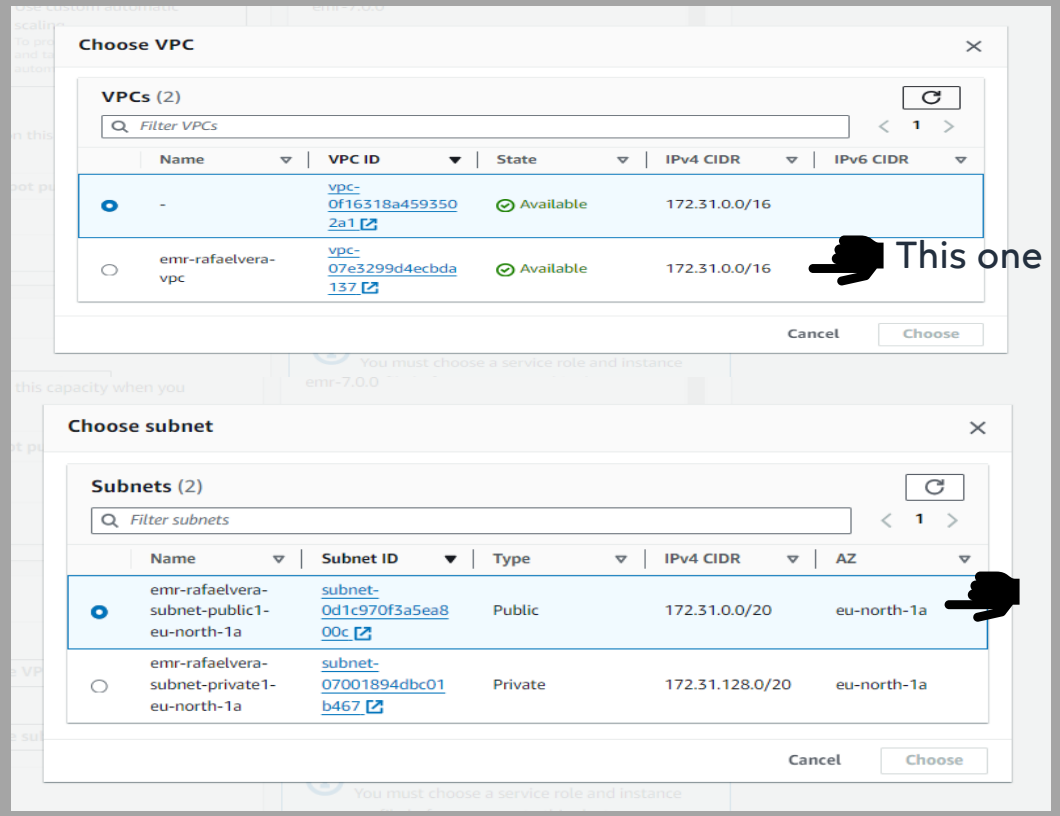
Great! The VPC is now ready to be used with our EMR cluster within the defined network.
Come back to the EMR Cluster Settings > Networking to continue configuring it

Create an EMR cluster

January 2024

2.5. Networking (VPC created) [Continued]





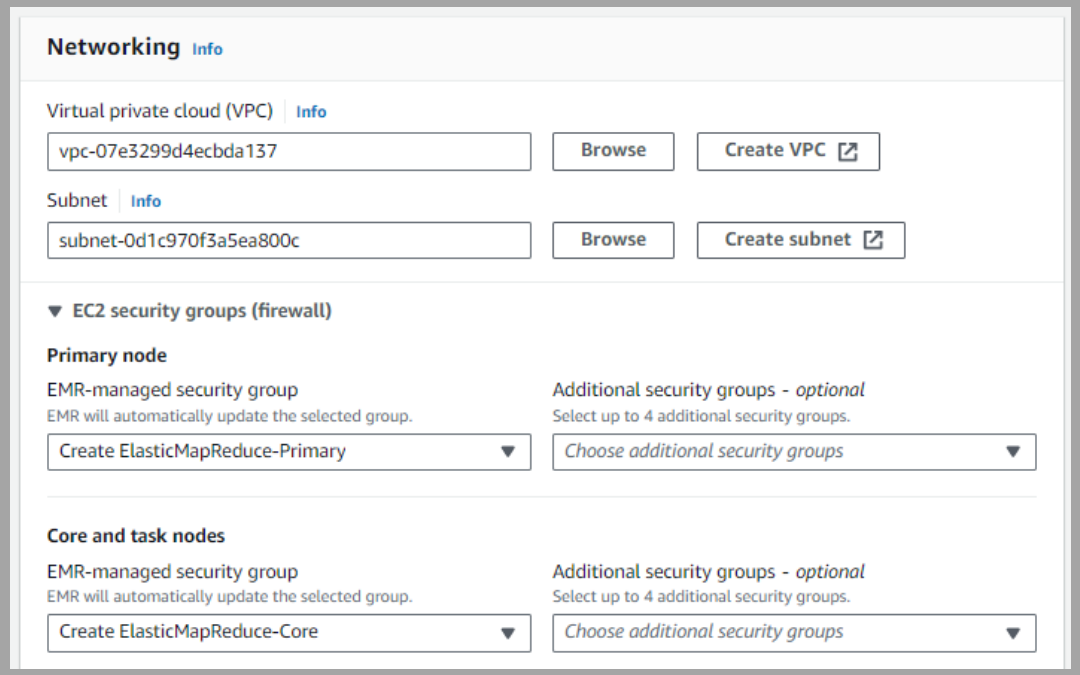
Browse VPC:

1. Select the VPC recently created.

2. Select the public subnet.

2nd part of Networking configuration

2.5.b.1 Setting EC2 Security groups



With this step you are done with Networking configurations for the EMR cluster.

Let's move forward with the next EMR Cluster configurations

EC2 security groups (firewall):

In this EMR configuration act as virtual firewalls to control network traffic, ensuring secure communication between cluster nodes and safeguarding them from unauthorized access.

Configuring these groups is critical for defining precise access control policies for the different node types within an EMR cluster.

Primary Node:
EMR-managed security group:
This security group will be automatically updated by EMR.

Additional security groups - optional: Nothing to do here

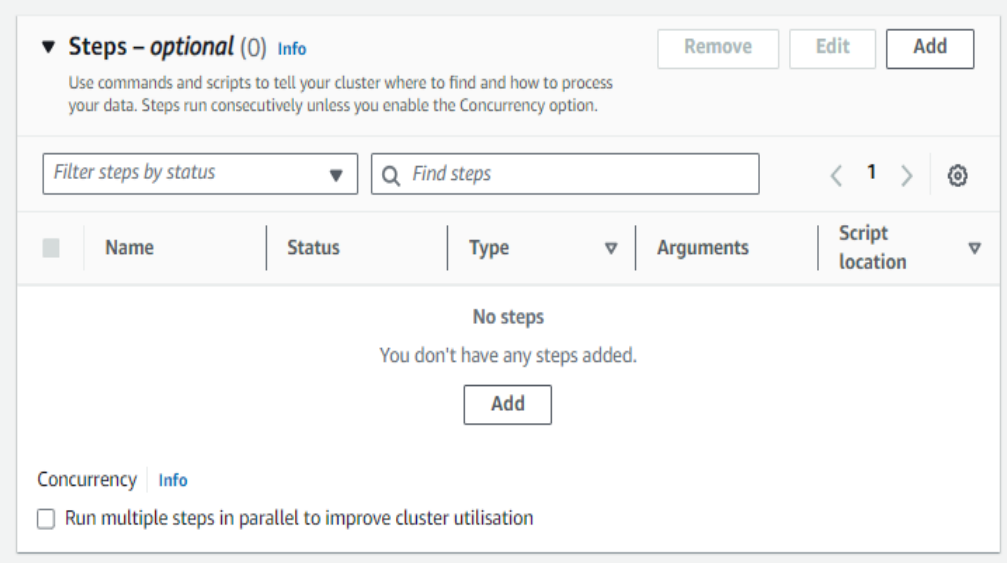
Core and Task Nodes:
EMR-managed security group:
Similar to the primary node, security group for the core and task nodes.

Additional security groups - optional: Nothing to do here

Create an EMR cluster

January 2024

2.6. Steps - Optional



Nothing to do here

This section is critical for automating data processing tasks within an EMR cluster, allowing for the scheduling of specific data processing activities that the cluster should perform.

Steps are used for defining commands and scripts to direct the cluster on where to find data and how to process it

Steps List:
List of steps available.

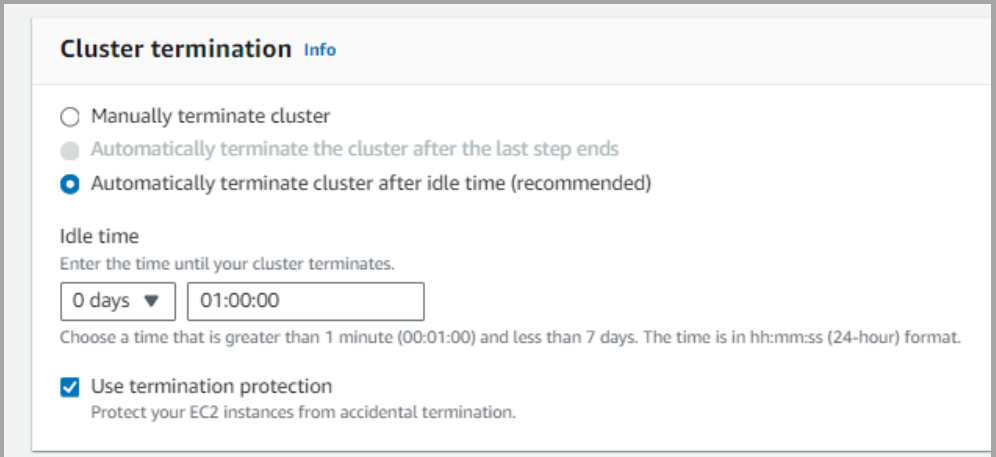
Add Step:
There is an "Add" button, suggesting that the user can add new steps to the cluster.

Concurrency:
By default, the steps defined for processing data in an EMR cluster are executed one after another, in the order they were added.

"Run multiple steps in parallel to improve cluster utilisation," if checked, would allow multiple steps to be executed at the same time, which can lead to more efficient use of the cluster's resources and potentially reduce the overall time taken for data processing.

Create an EMR cluster

2.7. Cluster termination



These settings are designed to help manage costs and prevent accidental termination of the cluster, which can be particularly important in a pay-as-you-go cloud environment where running clusters incur charges.

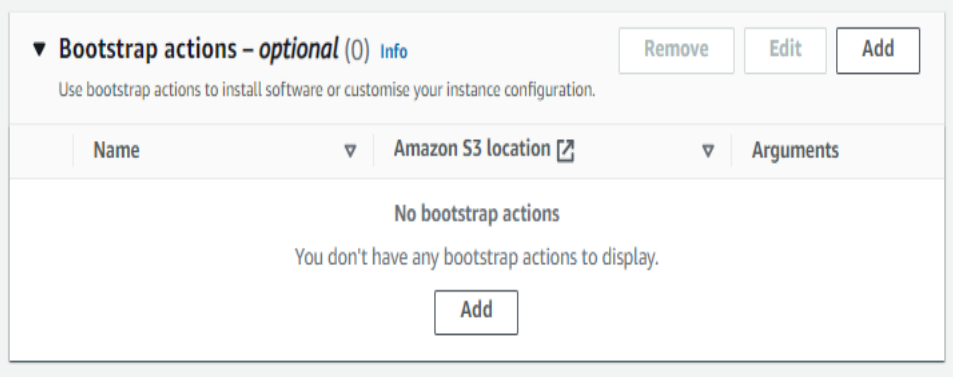
Automatically Terminate After Idle Time (Recommended):
The recommended option where the cluster will automatically terminate after a specified period of inactivity.

Idle Time:
You can specify the duration of inactivity after which the cluster should be terminated.

Use Termination Protection:
Is a feature that safeguards EC2 instances within an AWS EMR cluster from being unintentionally shut down, ensuring that your cluster continues to run until you decide to terminate it explicitly.

Create an EMR cluster

2.8. Bootstrap actions - Optional



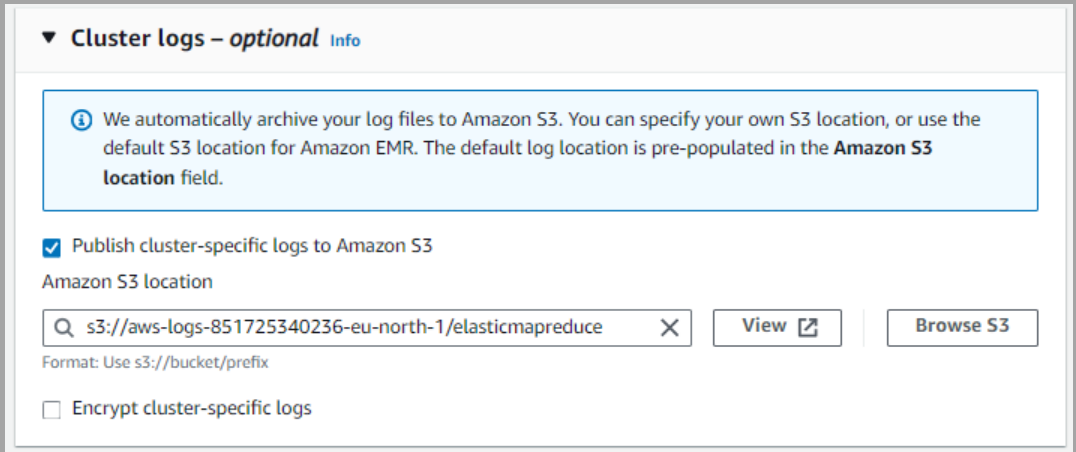
Nothing to do here

Bootstrap actions are used to install additional software or to customize the configuration of cluster instances during their initialization phase. They are a powerful feature for users who need to run custom setup scripts or install additional software packages that are not included in the EMR base image.

Create an EMR cluster

January 2024

2.9. Cluster logs - Optional



These settings are important for monitoring and debugging EMR clusters, as logs provide valuable information about the execution of cluster jobs and the performance of the cluster itself. The ability to encrypt logs adds an additional layer of security for sensitive data.

Automatic Archiving:
User can specify archiving the log files into their own S3 bucket or use the default S3 location pre-populated for Amazon EMR.

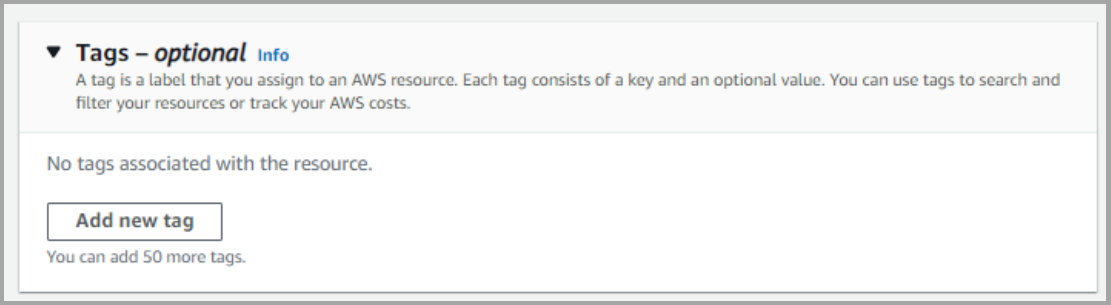
Publish Cluster-specific Logs:
Logs specific to the cluster will be published to Amazon S3.

Amazon S3 Location:
Shows the S3 bucket where logs will be stored.

Create an EMR cluster

January 2024

2.10. Tags - Optional



Nothing to do here

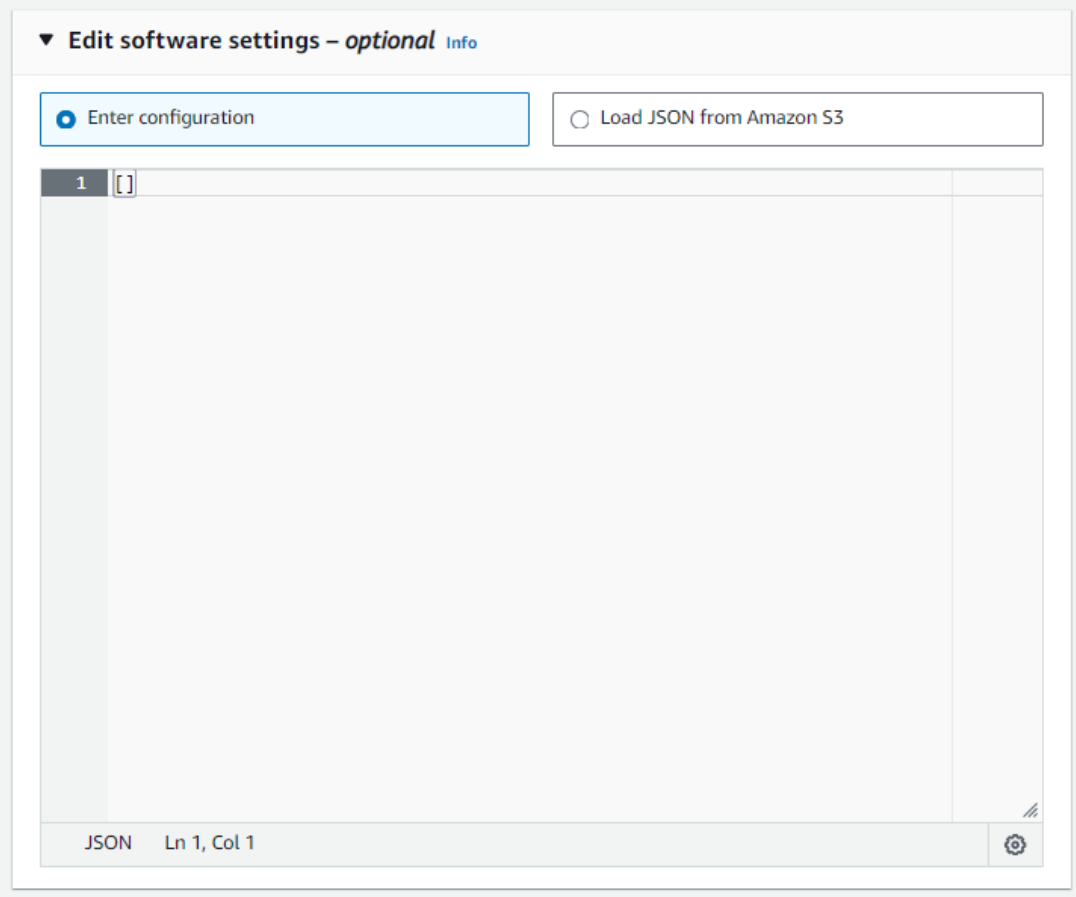
Tags are useful for organizing and managing AWS resources, especially for cost allocation and reporting in larger environments with multiple resources and services.

A TAG is a label assigned to an AWS resource consisting of a key and an optional value. Tags are used for searching and filtering resources or tracking AWS costs.

Create an EMR cluster

January 2024

2.11. Edit software settings - Optional



Nothing to do here

This interface allows users to specify detailed configurations for software on their EMR cluster, such as setting properties for Hadoop, Spark, and other applications within the cluster.

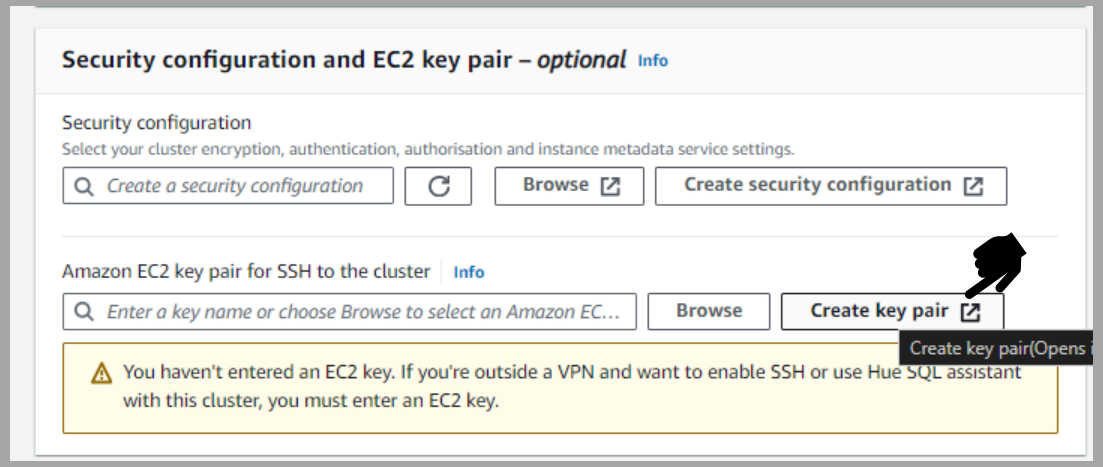
Users can fine-tune the behavior of their EMR cluster software using these configurations.

You can either enter the software configuration settings manually or you can load it from a JSON file stored in an Amazon S3 bucket.

Create an EMR cluster

January 2024

2.12. Security config and EC2 key pair- Optional

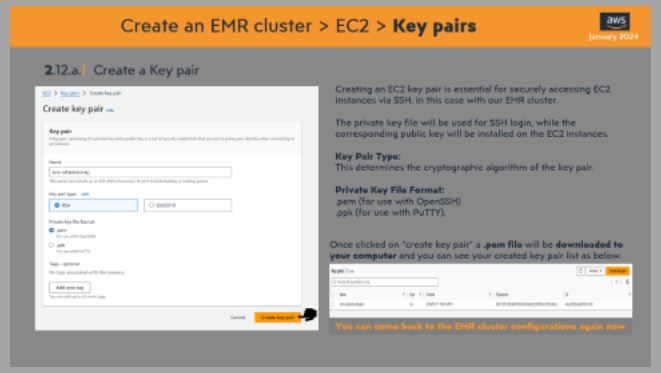


EC2 key pair:
is crucial for securely accessing cluster instances via SSH, which is important for administration tasks, such as troubleshooting or managing the software on the instances. Without this, remote access to the cluster's nodes would not be possible.

Security Configuration: Nothing to do here
Determines the cluster's encryption, authentication, authorization, and instance metadata service settings.

Amazon EC2 Key Pair for SSH to the Cluster:
Hit on create key pair and start creating it.

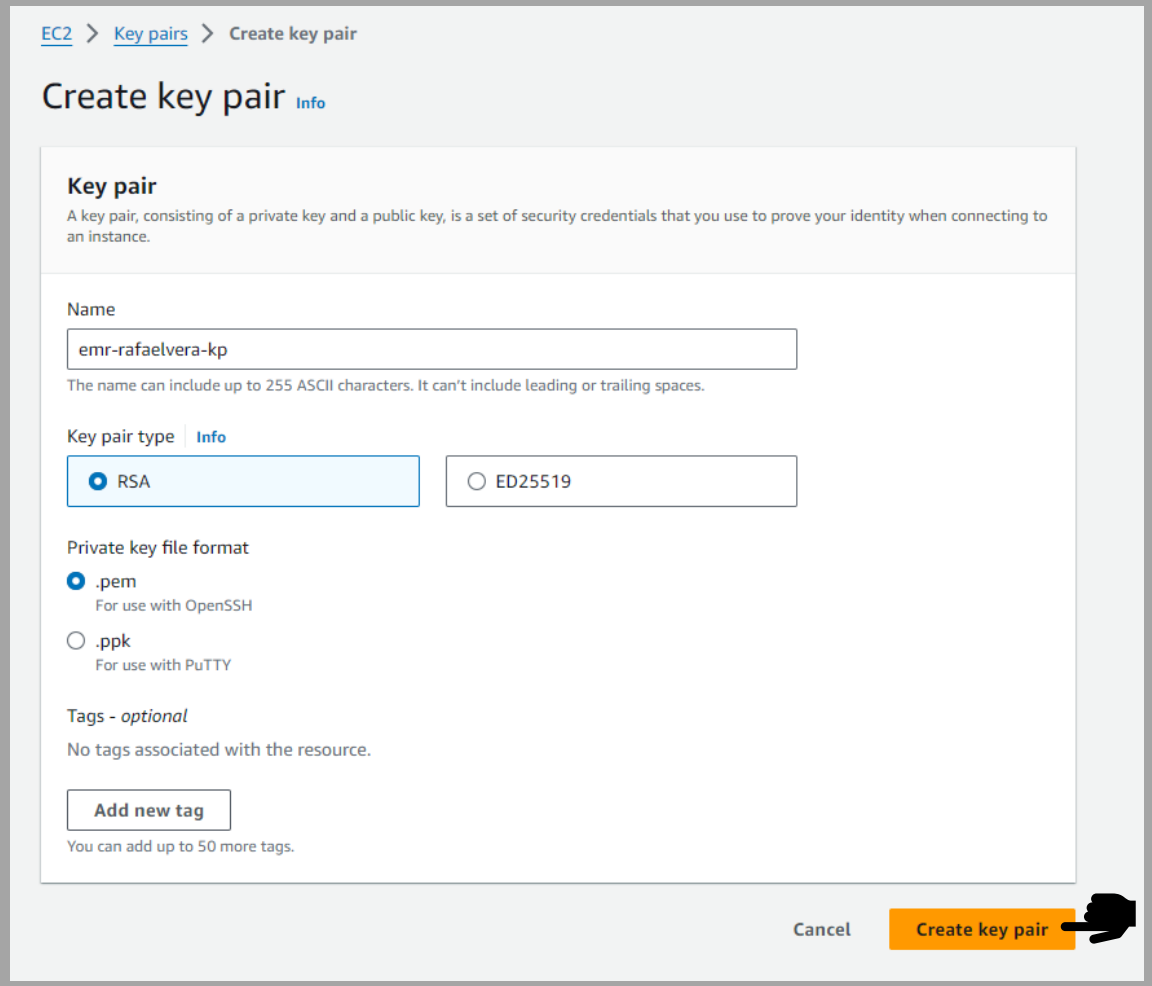
Create Key pairs (a)



Create an EMR cluster > EC2 > **Key pairs**

January 2024

2.12.a.1 Create a Key pair



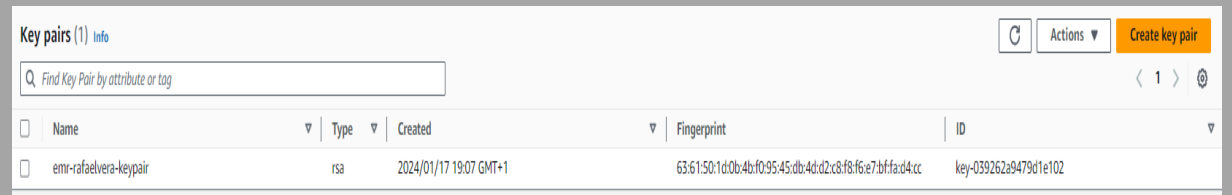
Creating an EC2 key pair is essential for securely accessing EC2 instances via SSH, in this case with our EMR cluster.

The private key file will be used for SSH login, while the corresponding public key will be installed on the EC2 instances.

Key Pair Type:
This determines the cryptographic algorithm of the key pair.

Private Key File Format:
.pem (for use with OpenSSH)
.ppk (for use with PuTTY).

Once clicked on "create key pair" a **.pem file** will be **downloaded to your computer** and you can see your created key pair list as below.

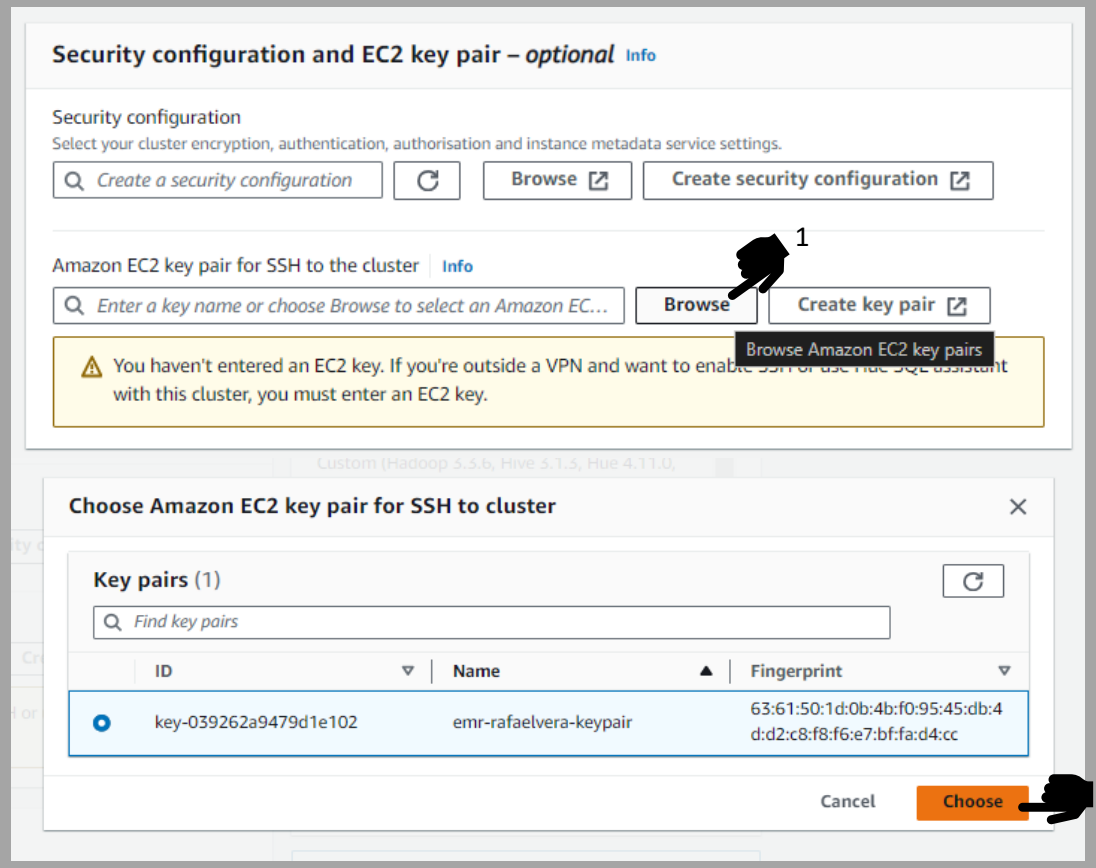


You can come back to the EMR cluster configurations again now

Create an EMR cluster

January 2024

2.12. EC2 key pair- Optional [Continued]



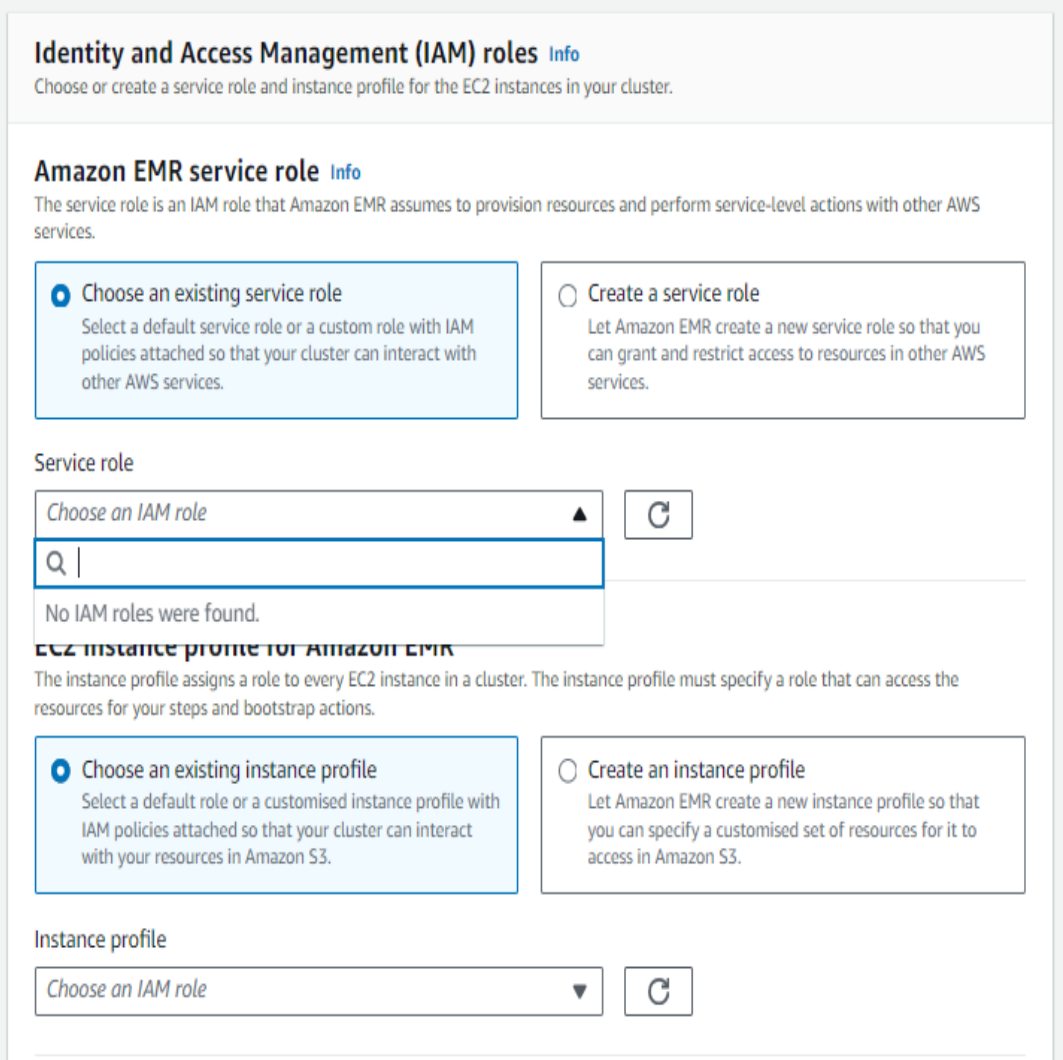
Amazon EC2 Key Pair for SSH to the Cluster:

1. Hit on browse key pair and **select** the recently created Key Pair.

2. **Click on Choose.**

Create an EMR cluster

2.13. IAM roles



Amazon EMR Service Role:
IAM roles and instance profiles are critical for securing EMR clusters and controlling access to AWS resources.
They ensure that the cluster has only the necessary permissions to operate effectively and securely, following the principle of least privilege.

EC2 Instance Profile for Amazon EMR:
assigns a role to EC2 instances in the cluster that allows them to access AWS resources necessary for cluster steps and bootstrap actions

The first time you see this configuration will show as the image on the left and you aren't able to select any existing Service role, **so you must "Create a service role" first.**

To start with it you need to follow the next steps across the **aws IAM service** to create both a Service role and Instance profile.

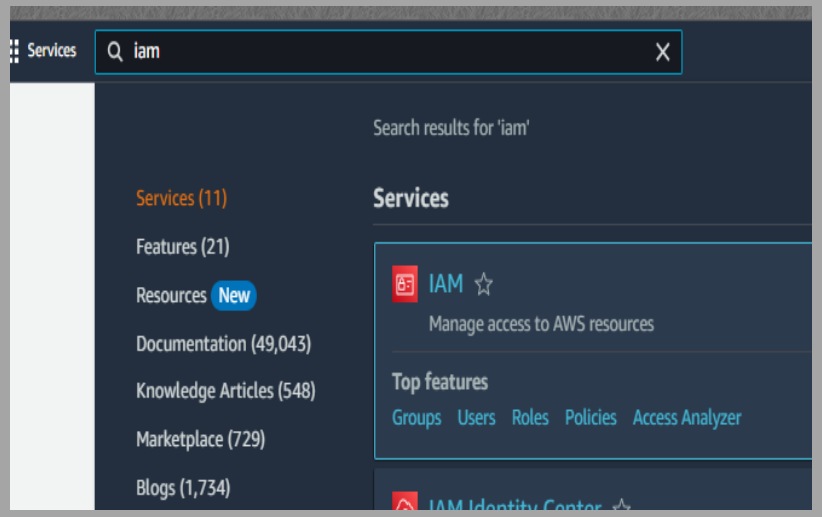
Create Service rol (a)

Create Instance Profile rol (b)

Create an EMR cluster > IAM > **Roles**

January 2024

2.13.a.1 Create Service role

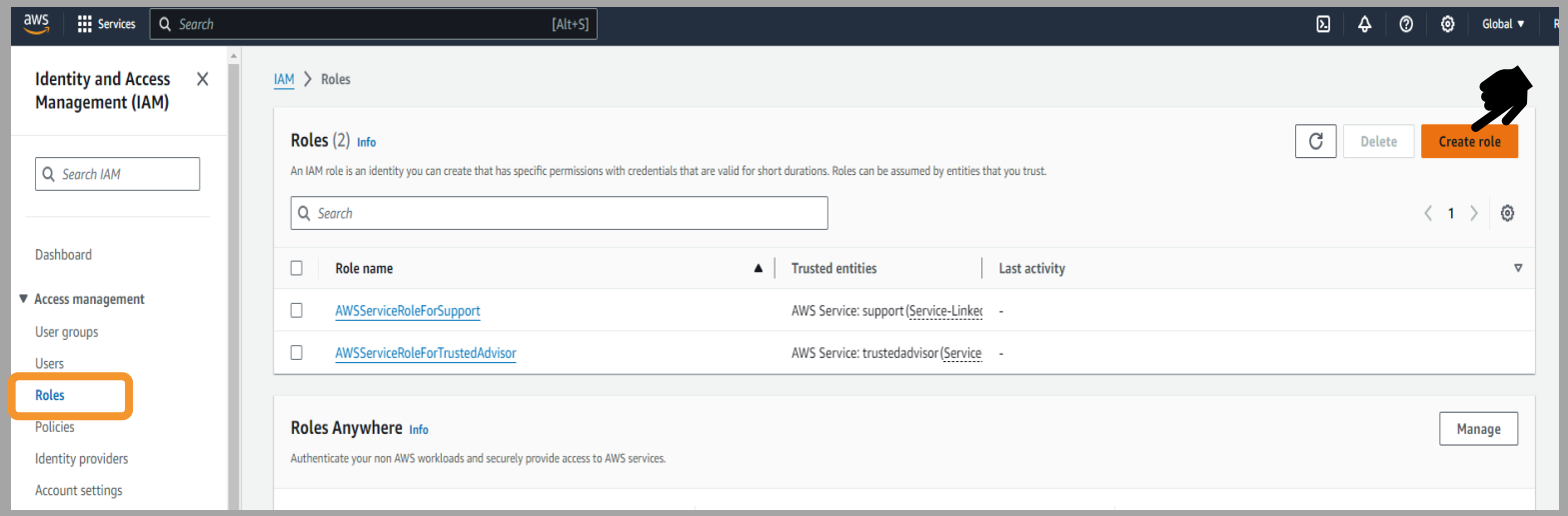


Open a new browser Tab and in aws console search for IAM service.

You will be redirected to the AWS Identity and Access Management (IAM) console, and there **click on Roles in the left menu.**

Then you will se the **Roles Dashboard:**

IAM Roles List:
AWSServiceRoleForSupport: **(autogenerated)**
AWSServiceRoleForTrustedAdvisor: **(autogenerated)**



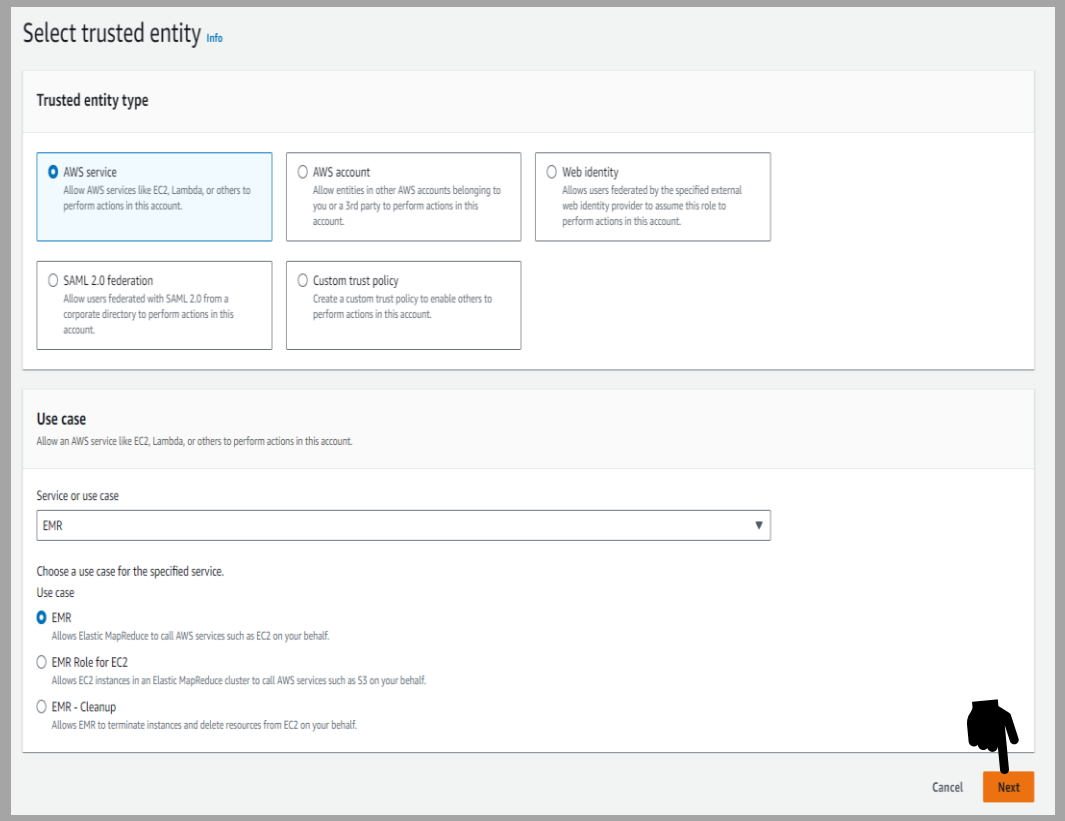
Trusted Entities:
Indicates what AWS service the role is linked to, confirming that these are service-linked roles.

To START creating the role, click on: "Create role".

Create an EMR cluster > IAM > **Roles**

January 2024

2.13.a.2 Create Service role



This interface defines the permissions and scope of an IAM role, ensuring that it is tailored to the needs of the EMR service

Trusted Entity Type:
AWS service:
indicating that the role is intended for AWS services such as EC2 to perform actions within the user's AWS account.

AWS account: For allowing entities in other AWS accounts to perform actions in the user's account.

Web identity: For use with identity providers that are federated through web identity (like Google, Facebook, or Amazon).

SAML 2.0 federation: For federation with SAML 2.0 from a corporate directory.

Custom trust policy: For creating a custom trust policy.

Use case:
EMR: Allows Elastic MapReduce to call AWS services such as EC2 on your behalf.

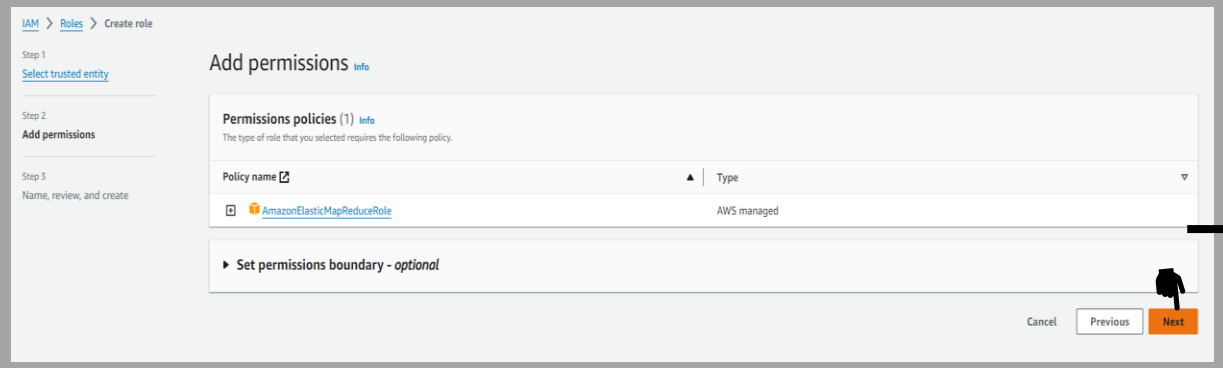
EMR Role for EC2: Allows EC2 instances in an Elastic MapReduce cluster to call AWS services on your behalf.

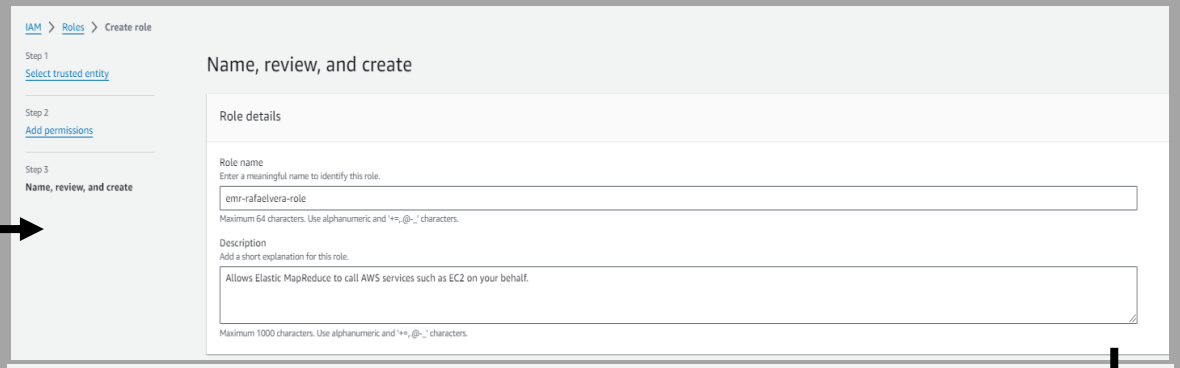
EMR - Cleanup: Allows EMR to terminate instances and delete resources from EC2 on your behalf.

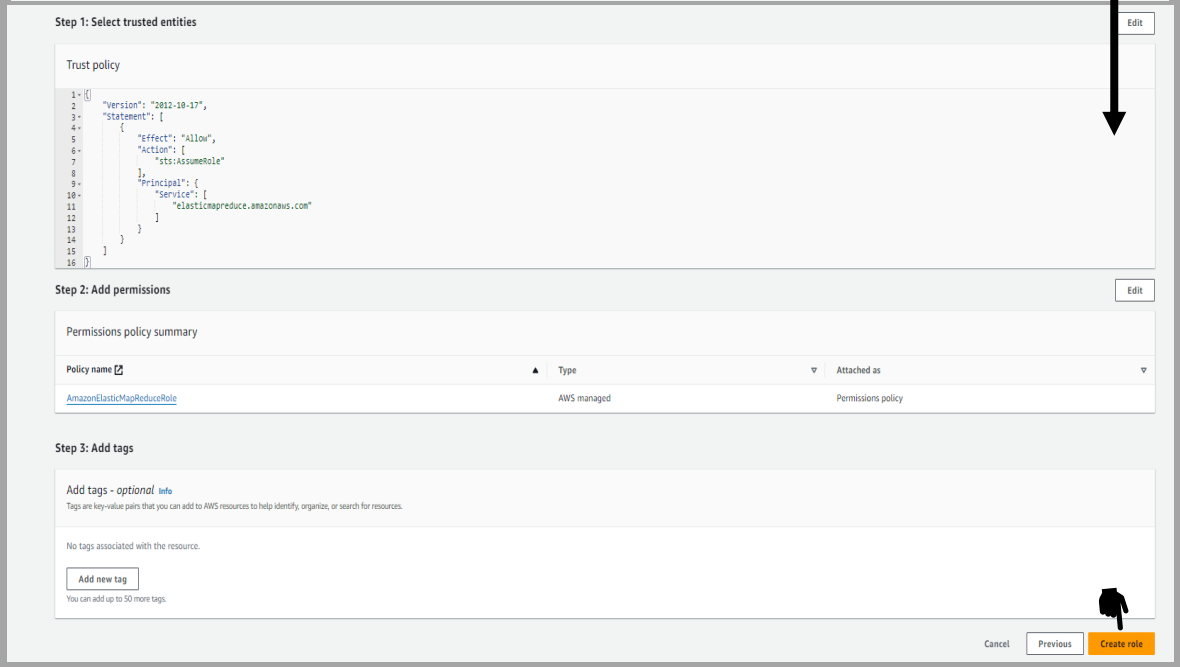
Create an EMR cluster > IAM > **Roles**

January 2024

2.13.a.3 Create Service role







Step 1: Select Trusted entity
Made on the slide before.

Step 2: Add permissions
Involves attaching policies that grant the IAM role the necessary permissions to function correctly within AWS EMR.

AmazonElasticMapReduceRole is an AWS managed policy designed to grant the necessary permissions for an Amazon EMR role.

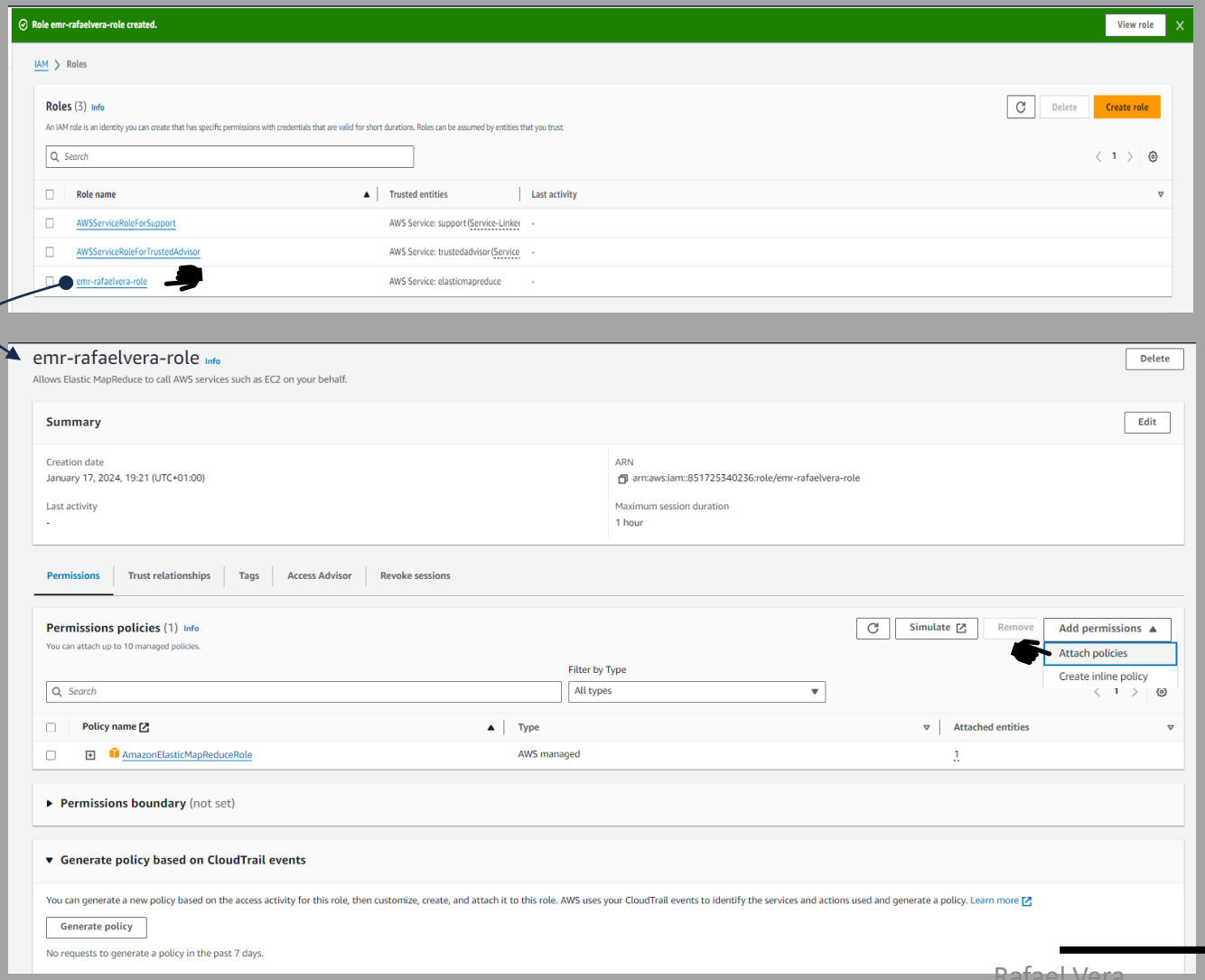
Step 3: Name, review and create
Role name: is used to identify the role within AWS.
The description: context for what the role is intended to do.
Add tags: prompts to add metadata tags to the IAM role for identification, organization, or search purposes.

Review of role creation showing 3 main steps:
Just click Create role

Create an EMR cluster > IAM > **Roles**

January 2024

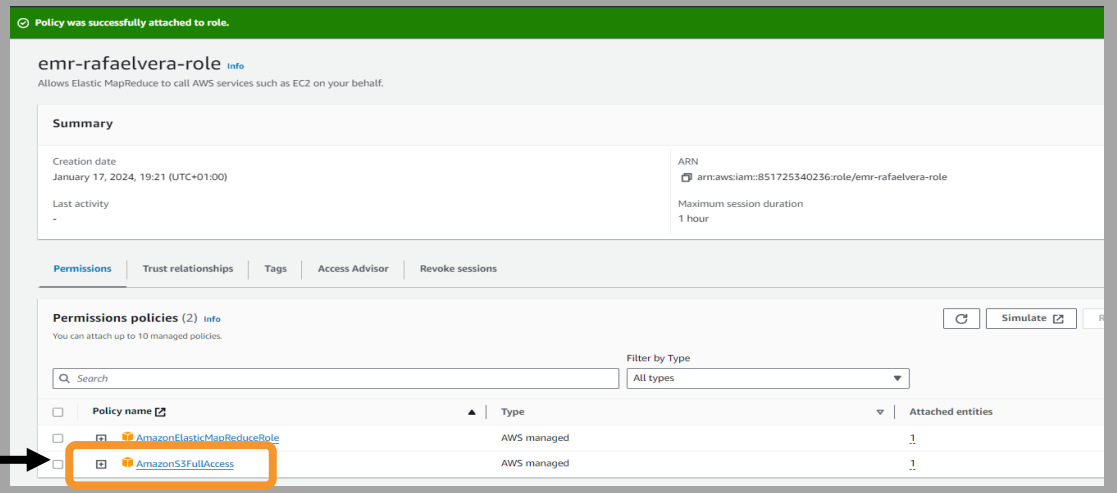
2.13.a.4 Create Service role - Created



The **role** is already **created** and appears **listed** on our roles console.

We need **add an additional Permission policy to this role** in order to having access to the S3 Buckets service, which will be needed afterwards.

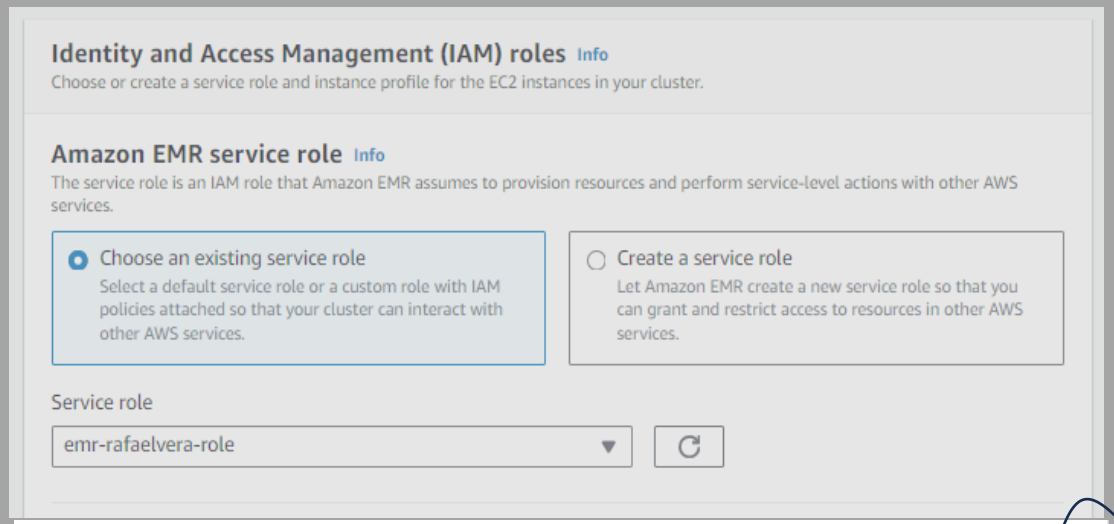
1st. Click on the role name as shown on the image.
2nd. On the Role configuration screen select **attach policies**.
3rd. Search for **AmazonS3 full access.**
4th. Add the new permission policy.
5th. See the **two inlisted Permission policies.**
6th. Go back to the EMR Cluster configurations



Create an EMR cluster

January 2024

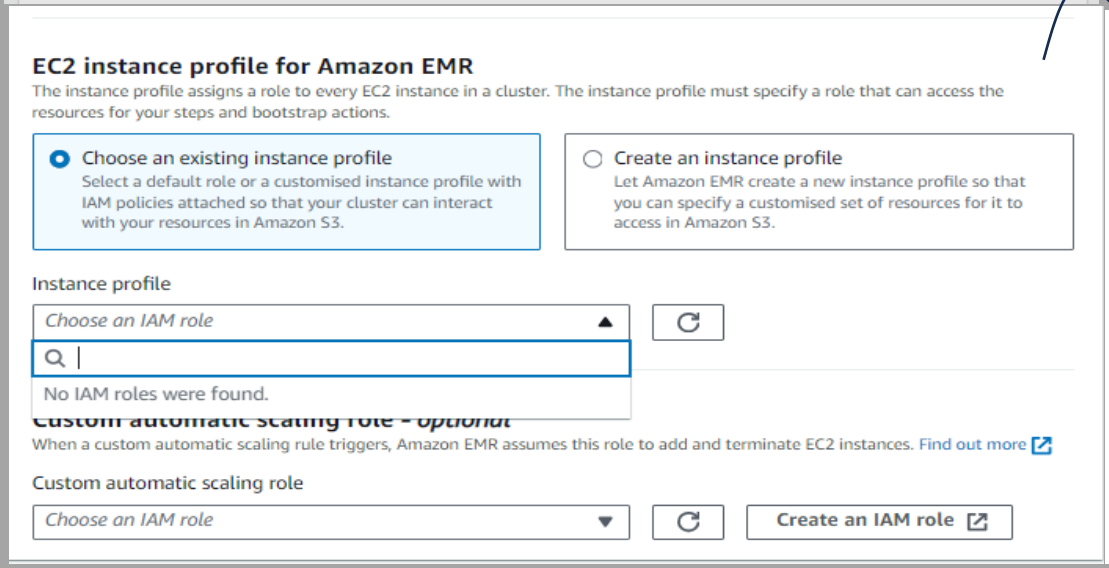
2.13. IAM roles [continued]

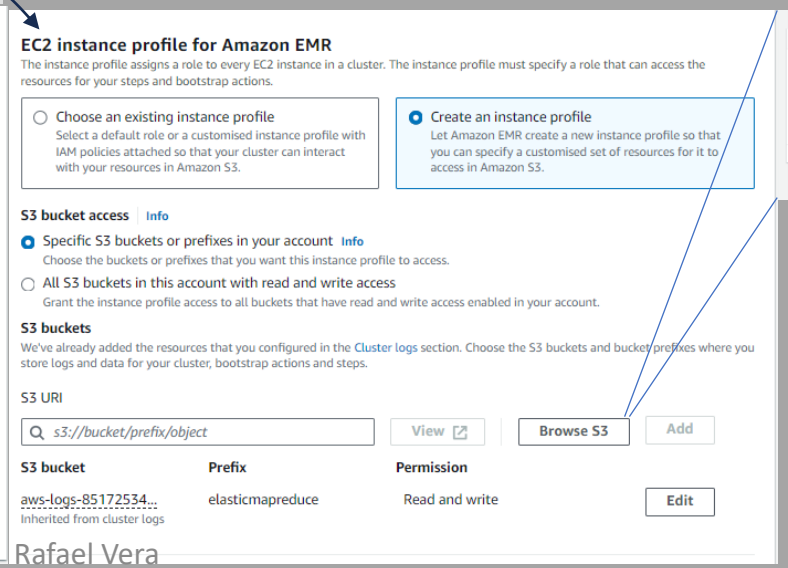


Amazon EMR Service Role:
Back to here,
1st. Select the Service role already created from the dropdown list.

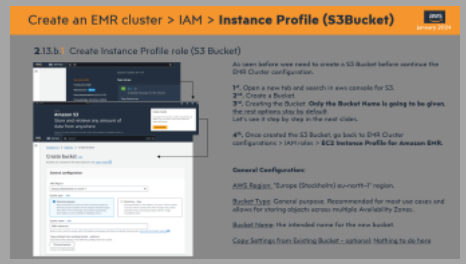
EC2 Instance Profile for Amazon EMR:
Defines access scope of the EMR cluster's EC2 instances to S3 resources (data storage and operations performed by the cluster).

2nd. Select "Create an instance profile"
3rd. Click on Browse S3 and we haven't S3 buckets instances.
4th. Follow the next guide steps -> Create instance profile (b)





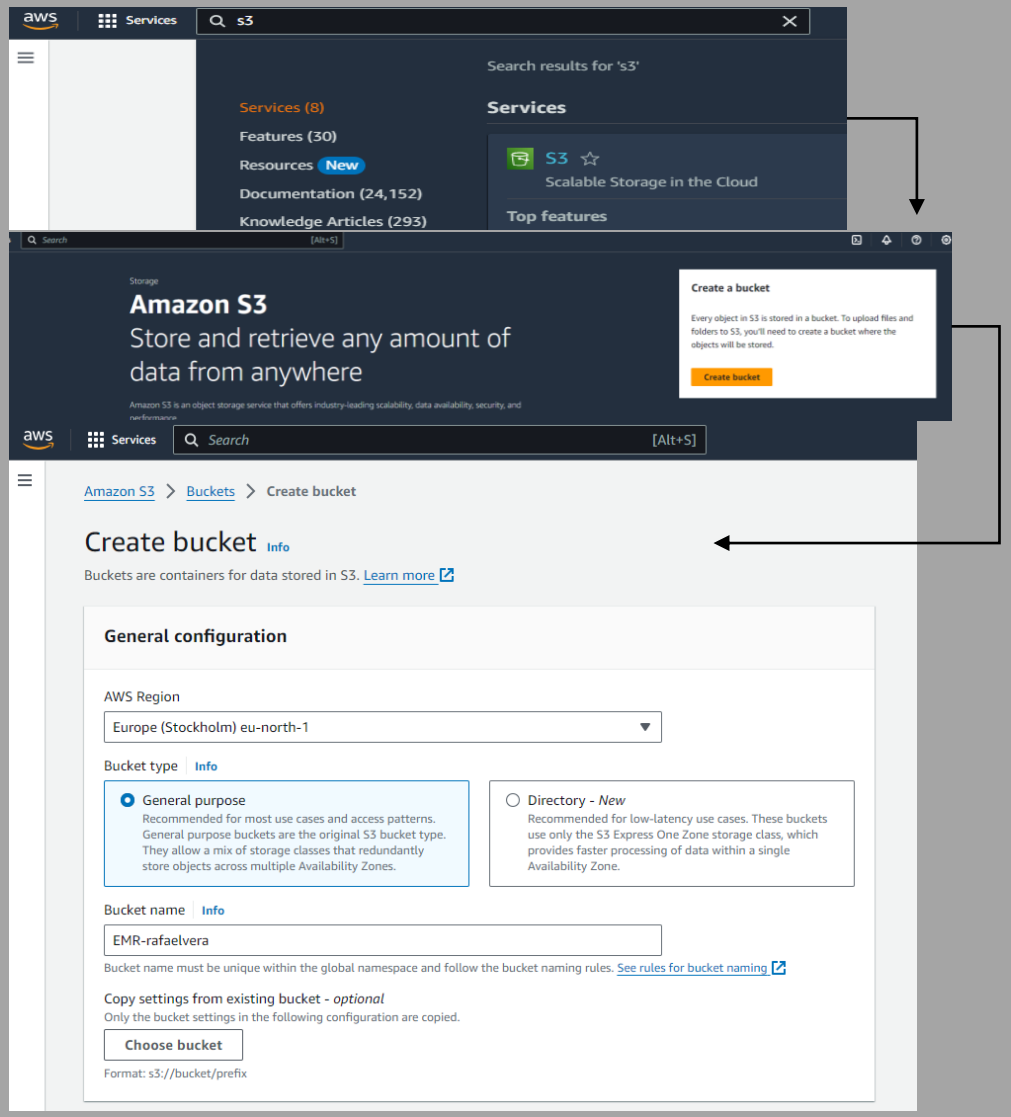
Create Instance Profile (b)



Create an EMR cluster > IAM > **Instance Profile (S3Bucket)**

January 2024

2.13.b.1 Create Instance Profile role (S3 Bucket)



As seen before wee need to create a S3 Bucket before continue the EMR Cluster configuration.

1st. Open a new tab and search in aws console for S3.
2nd. Create a Bucket.
3rd. Creating the Bucket. **Only the Bucket Name is going to be given**, the rest options stay by default.
Let's see it step by step in the next slides.

4th. Once created the S3 Bucket, go back to EMR Cluster configurations > IAM roles > **EC2 Instance Profile for Amazon EMR.**

General Configuration:

AWS Region: "Europe (Stockholm) eu-north-1" region.

Bucket Type: General purpose. Recommended for most use cases and allows for storing objects across multiple Availability Zones.

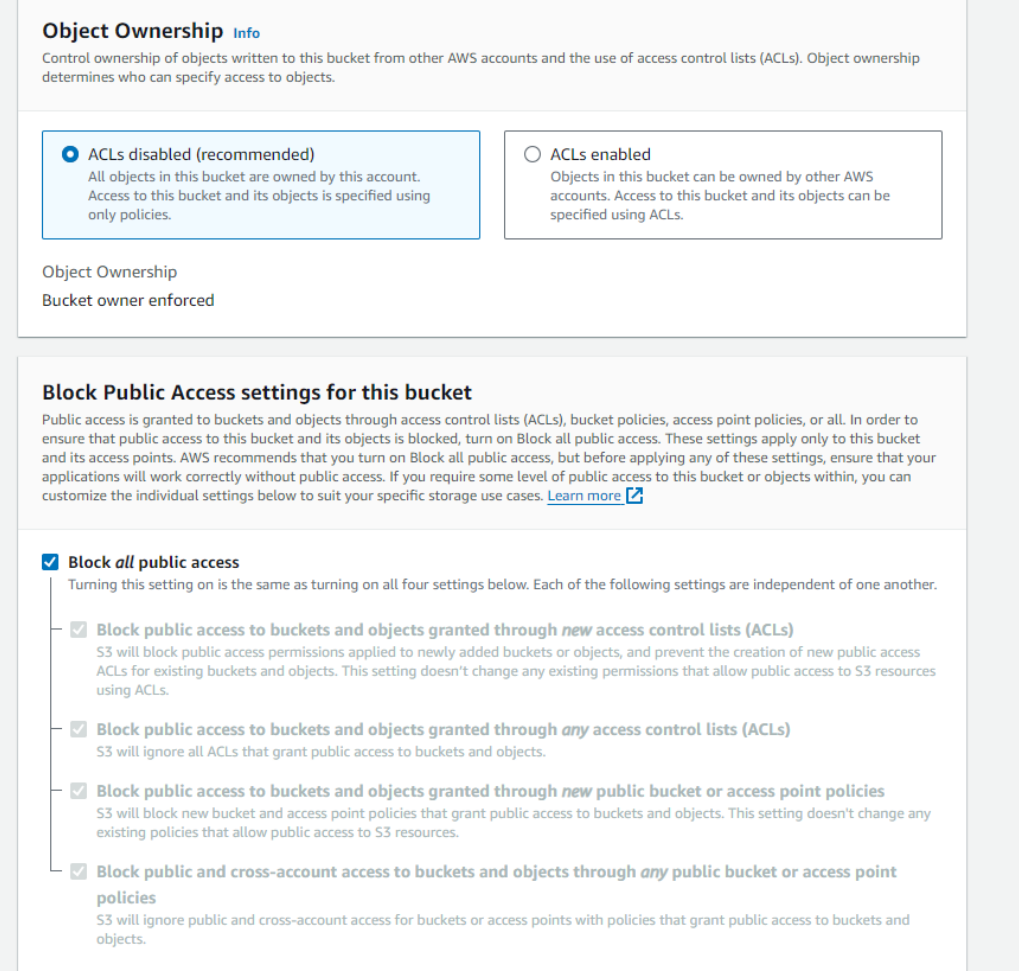
Bucket Name: the intended name for the new bucket.

Copy Settings from Existing Bucket - optional: Nothing to do here

Create an EMR cluster > IAM > **Instance Profile (S3Bucket)**

January 2024

2.13.b.2 Create Instance Profile role (S3 Bucket)



These settings maintain the security and privacy of the data stored in the S3 bucket. By enforcing bucket ownership and blocking public access, the bucket owner can prevent unauthorized access and data leaks.

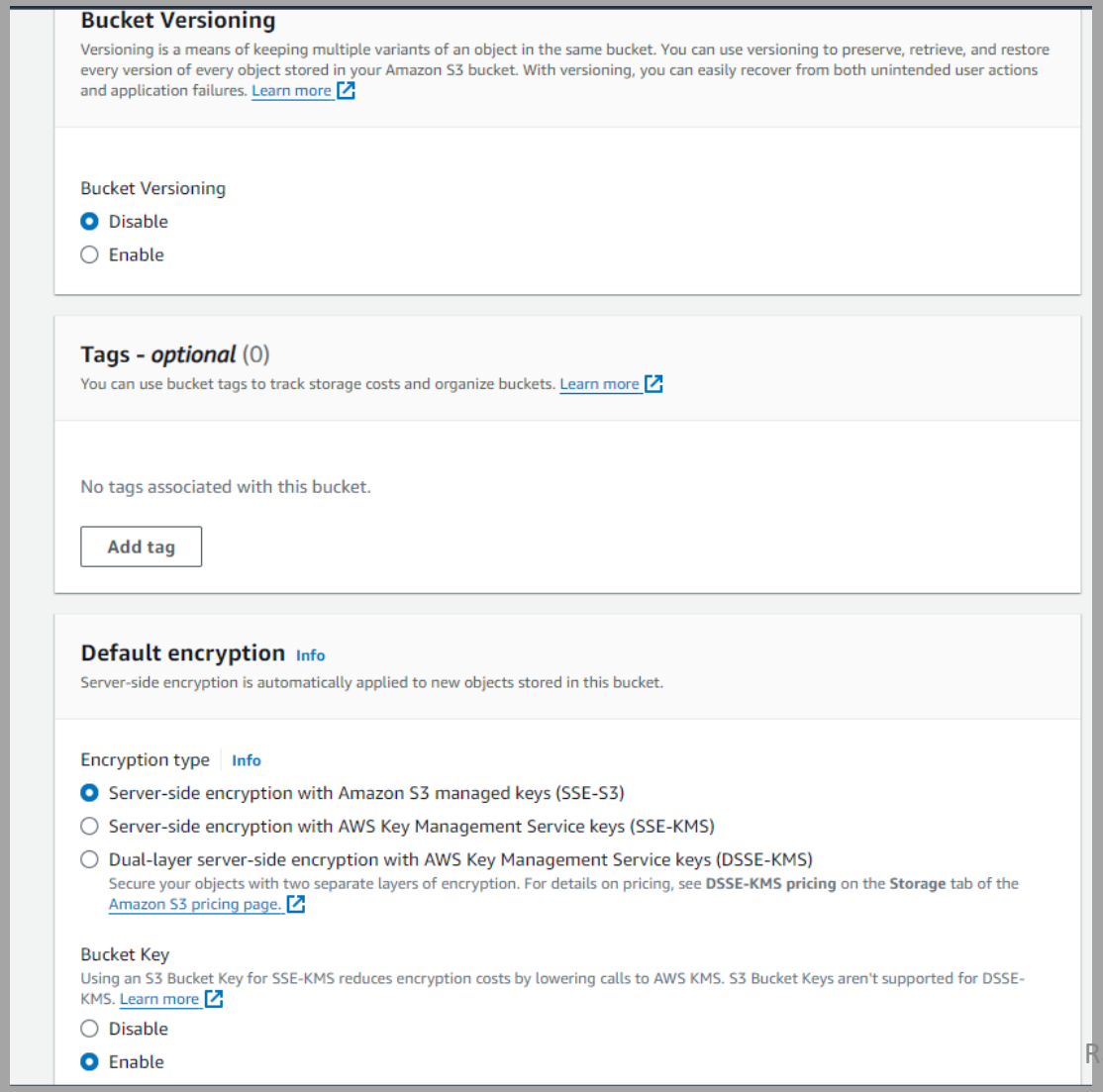
ACLs disabled (recommended): Indicates that access control lists (ACLs) are disabled, and all objects in the bucket are owned by the current account. Access to the bucket and its objects is managed through policies.

The "Bucket owner enforced" setting is highlighted meaning that the bucket owner will retain ownership of all uploaded objects.

Create an EMR cluster > IAM > **Instance Profile (S3Bucket)**

January 2024

2.13.b.3 Create Instance Profile role (S3 Bucket)



These settings maintain data integrity and security within an S3 bucket. Versioning helps with data recovery, tags provide organization and cost management, and encryption ensures data confidentiality and security.

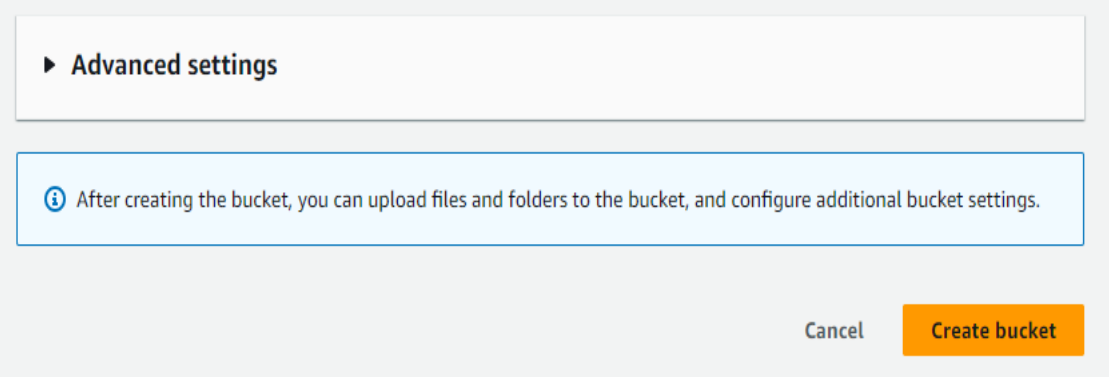
Tags – Optional: Nothing to do here

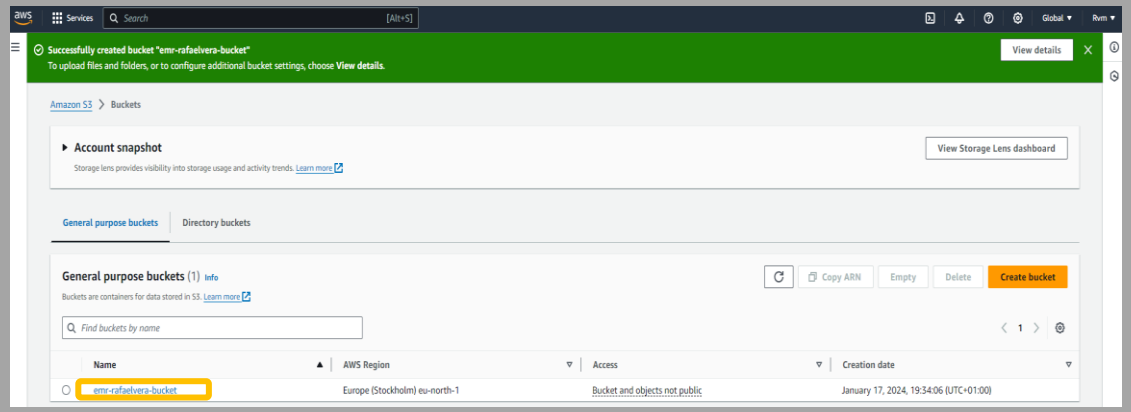
Default Encryption:
Describes the server-side encryption automatically applied to new objects stored in this bucket.
"Server-side encryption with Amazon S3 managed keys (SSE-S3)": This option means objects will be encrypted with keys managed by S3.

Create an EMR cluster > IAM > **Instance Profile (S3Bucket)**

January 2024

2.13.b.4 Create Instance Profile role (S3 Bucket)





Advanced settings: Nothing to do here

Click on "**CREATE BUCKET**"

Amazon S3 console within the AWS Management Console: showing an overview of S3 buckets.

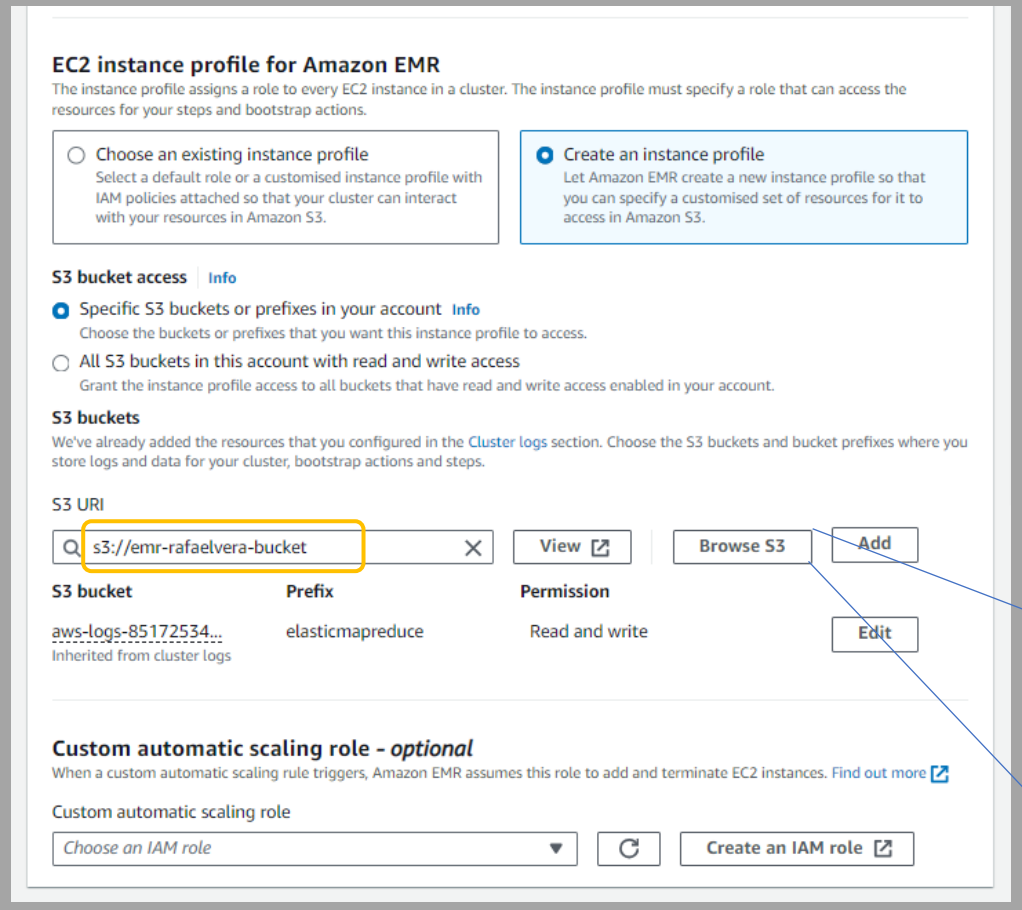
Bucket Information:
Bucket name: "emr-rafaelvera-bucket".
AWS Region: "Europe (Stockholm) eu-north-1".
Creation Date: January 17, 2024, 19:34:06 UTC+00:00".

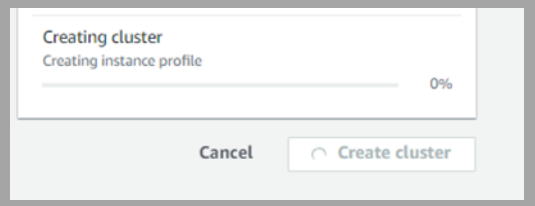
We are ready to come back to the EMR Configuration again and select this bucket in there.

Create an EMR cluster

January 2024

2.13. IAM roles [continued]



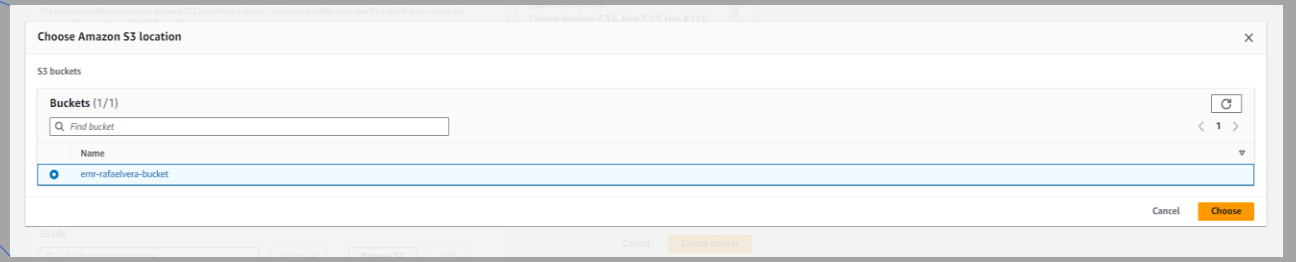


EC2 Instance Profile for Amazon EMR > S3 Bucket

This configuration settings define how the EMR cluster will interact with with S3 for data storage and scaling policies for efficient resource management.

Just click on "**BROWSE S3**" to find the recently created S3Bucket out, and "**CHOOSE**" it.

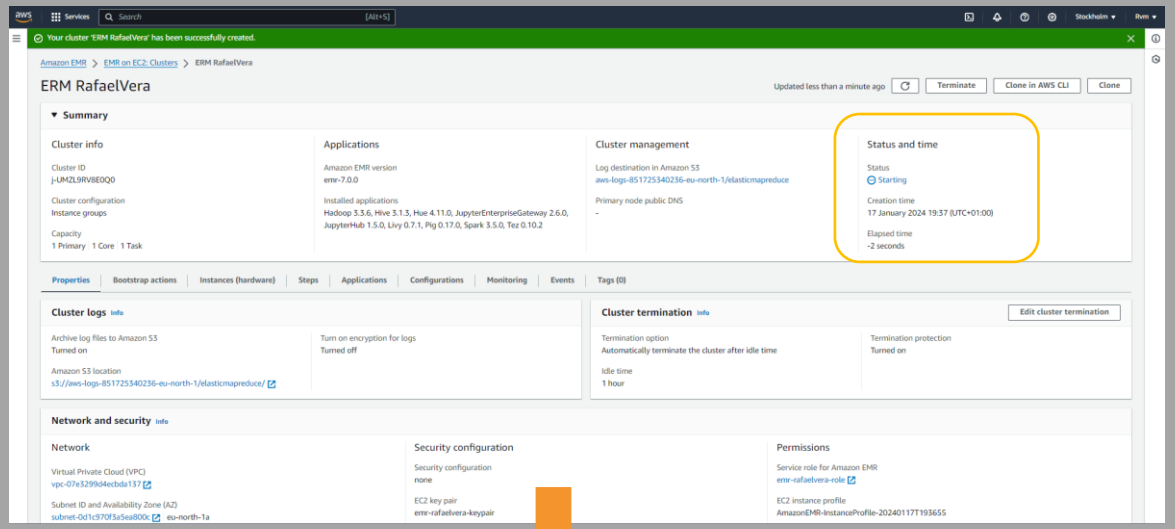
After that, we are done.
Hit the botton "**CREATE CLUSTER**" to start creating the EMR Cluster instance and wait for while until being redirected to next screen.

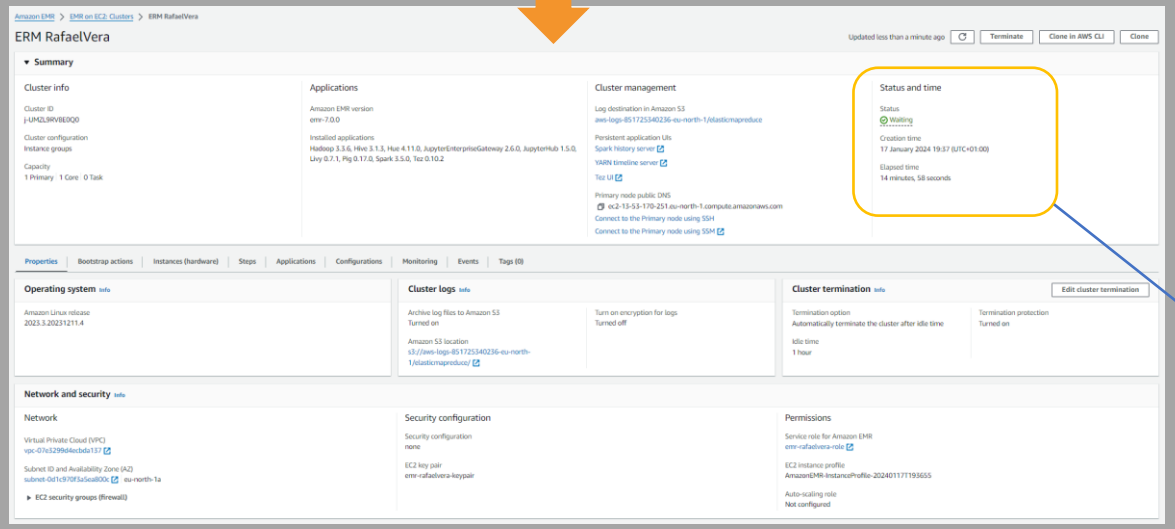


Create an EMR cluster

January 2024

2.14. Created EMR Cluster



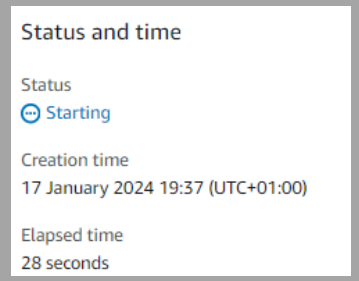


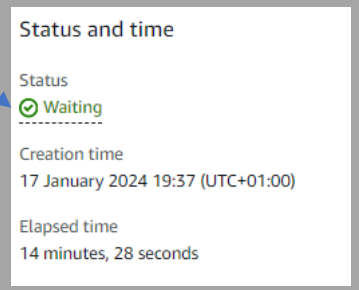
AWS Management Console view of the summary page for the Amazon EMR (Elastic MapReduce) cluster:

This summary page provides a comprehensive view of the cluster's setup, including configuration, network and security settings, software applications, and management settings. Allows therefore an easy monitoring and management of the EMR cluster.

Just wait for Status and Time switch from Starting to Waiting **(aprox 15 min)**

Grab a cup of coffee, you are doing it great!

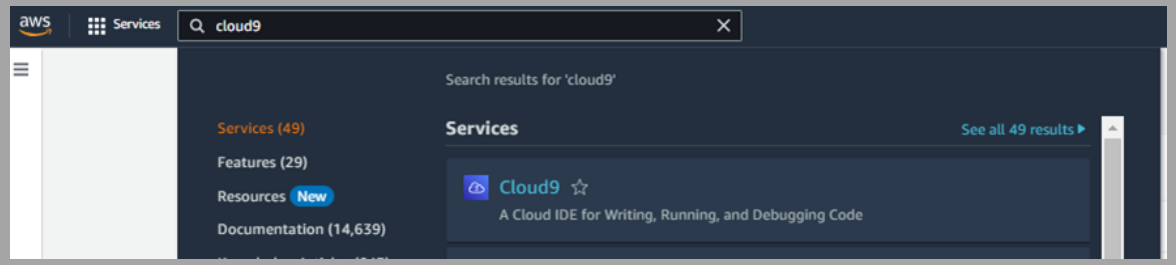




EMR Cluster ready to use.

Move forward to next guide section

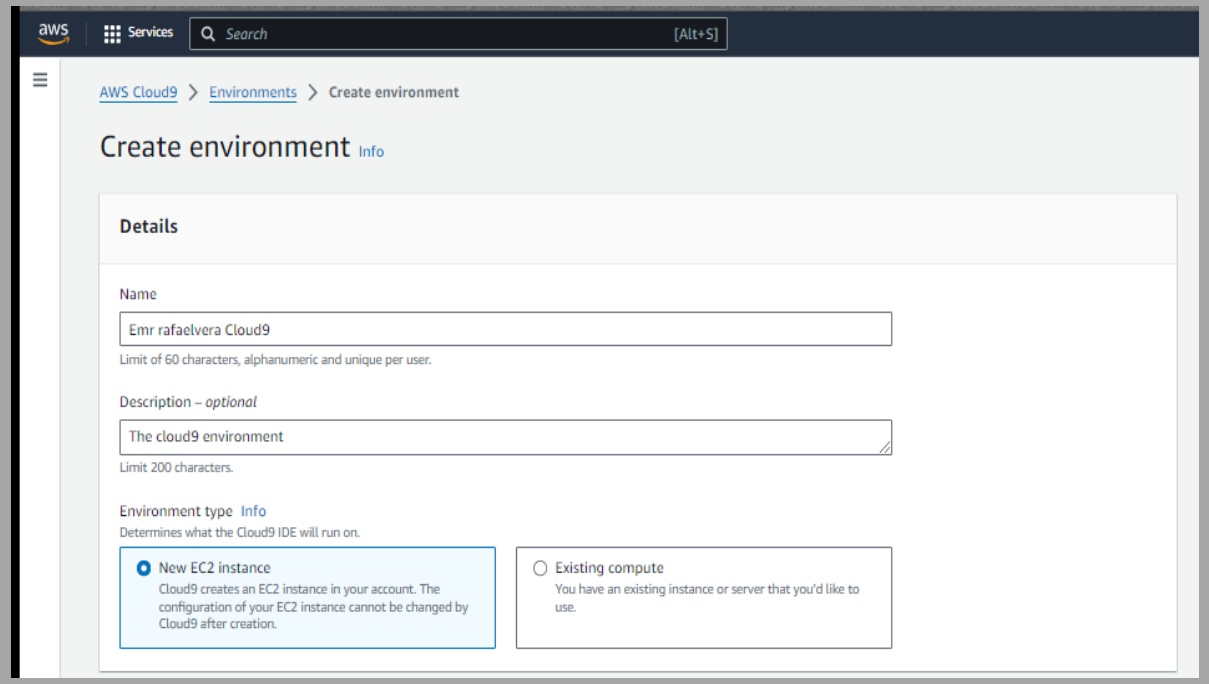
3. Aws Cloud9 IDE



We are instantiating the AWS Cloud9 service and use it as our IDE to run the Data processing script (python) within our EMR cluster.

1. Open a new Browser tab for aws services
2. Search for cloud 9 and click on it
3. You will be landed on aws Cloud9 service homepage
4. Click on create environment
5. Follow the next steps to configure the cloud9 instance environment

3. 1. Create Environment



AWS Cloud9 > Environments > Create Environment

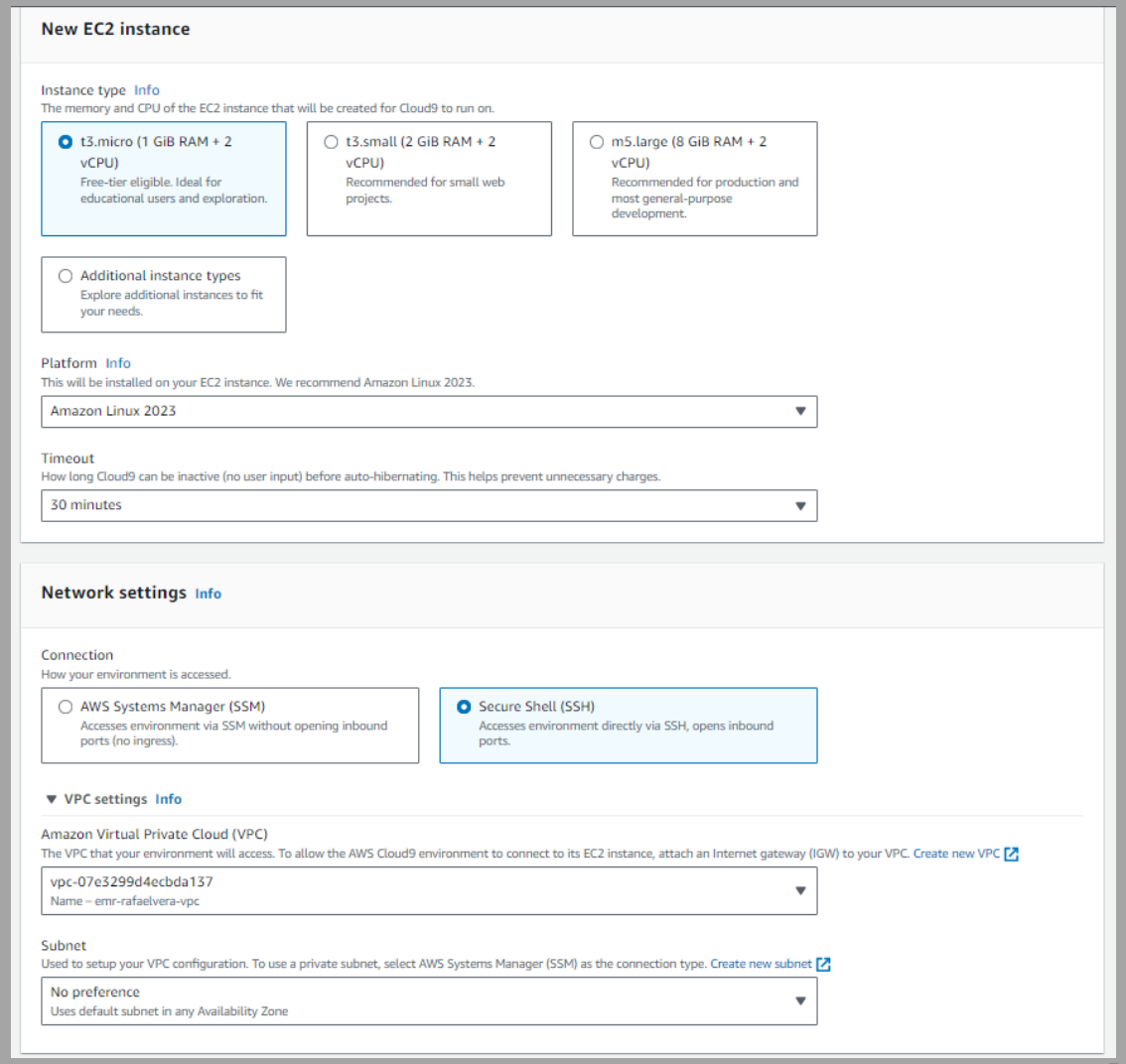
This interface guides us through the setup of the new Cloud9 IDE environment, with options to configure the underlying compute resources. It simplifies setting up a development workspace in the cloud, integrated with AWS services.

Environment Type:

New EC2 Instance**:** Selected by default. Cloud9 will create a new EC2 instance for the environment.

Existing Compute: If selected, this would allow the user to connect the Cloud9 environment to an existing instance or server.

3. 2. New EC2 Instance & Network settings



New EC2 instance:
will host the AWS Cloud9 environment.

Instance Type:
t3.micro (1 GiB RAM + 2 vCPU): free-tier eligible.

Platform:
set to "Amazon Linux 2023", indicates the operating system that will be installed on the EC2 instance.

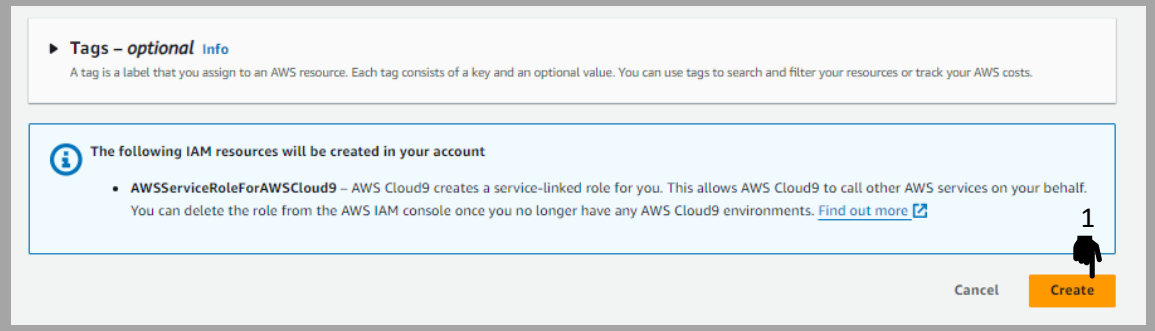
Timeout:
Set the duration for how long the Cloud9 environment can be inactive before auto-hibernating, to prevent unnecessary charges.

Network Settings:
Secure Shell (SSH): Direct access to coud9 environment via SSH, which opens inbound ports.

VPC Settings:
Select the VPC created in steps before.
In my case emr-rafaelvera-vpc.

Subnet:
no specific preference for a subnet, and the default subnet in any Availability Zone will be used.

3. 3. Create the EC2 instance

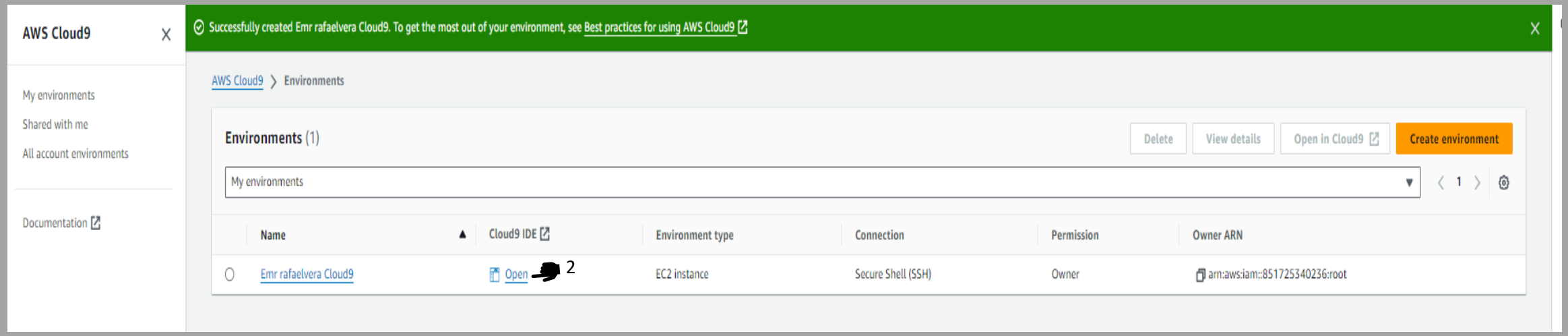


Tags – optional: Nothing to do here

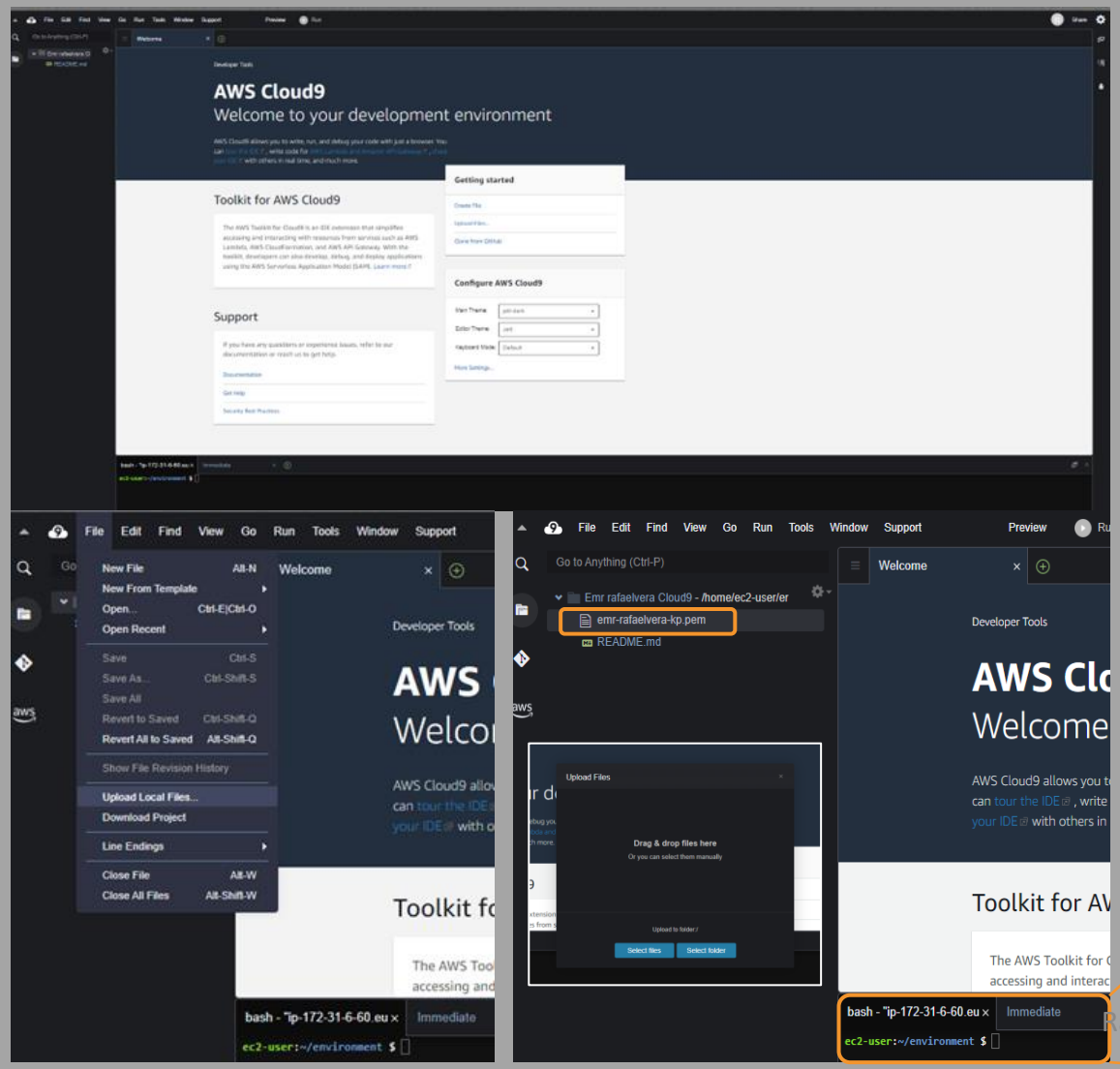
1. Click on **CREATE** and you will be redirected to **Aws cloud9 console**

This Aws cloud9 console lists all the created environments. In this case shows main information from the recently environment instance created.

2. Click on **OPEN** to see how Cloud9 looks like



3. 4. Inside Cloud9



AWS Cloud9 integrated development environment (IDE): provides users with a powerful cloud-based environment for software development, including access to a pre-configured editor, terminal, and essential AWS tools and resources.

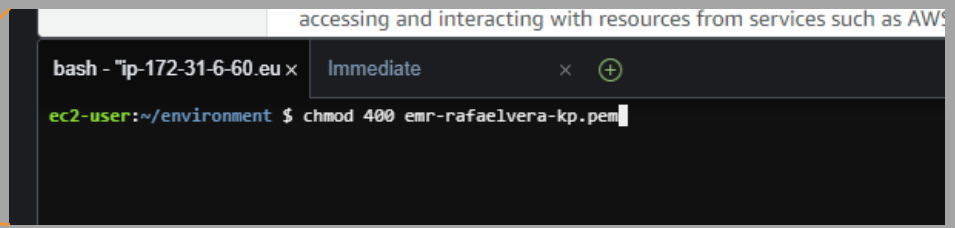
1. **Upload the Keypairs** generated during EMR Cluster. This .pem file was automaticaly downloaded locally as well. **Reminder here**
2. **Run this command:** change the file name.
   chmod 400 emr-rafaelvera-kp.pem

This command change the permissions to be read-only by the file's owner.
The 400 permission setting is used for SSH private key files ensuring they are kept secure and aren't accessible by other users.

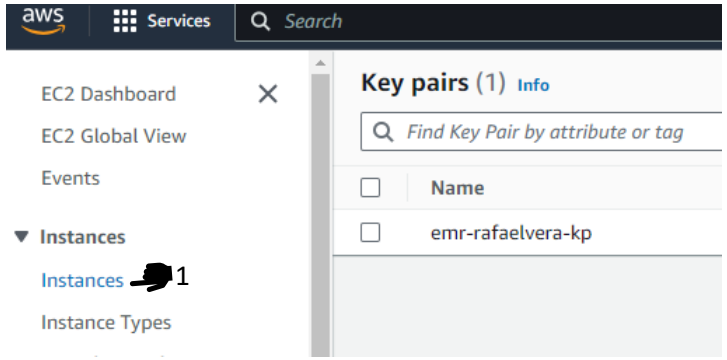
Before any further, we need to:

a) Create inbounds Rules in the EMR Instance allowing cloud9 connection

b) Get the SSH command from EMR to be able to connect with it.

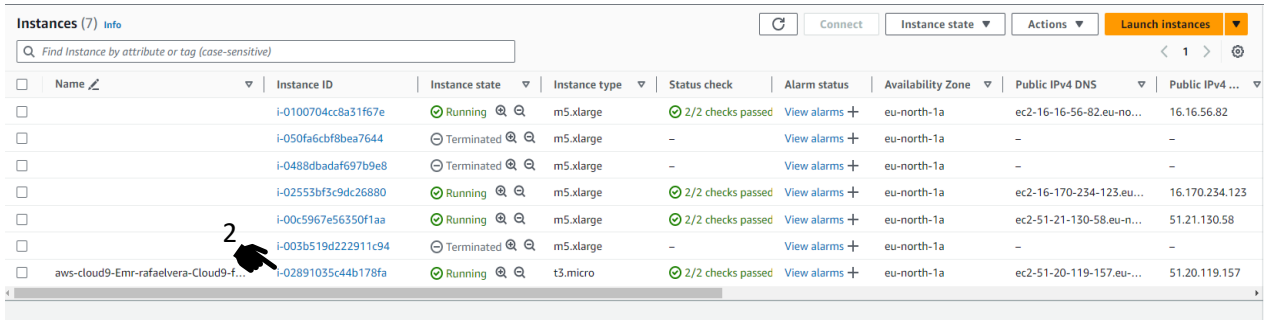


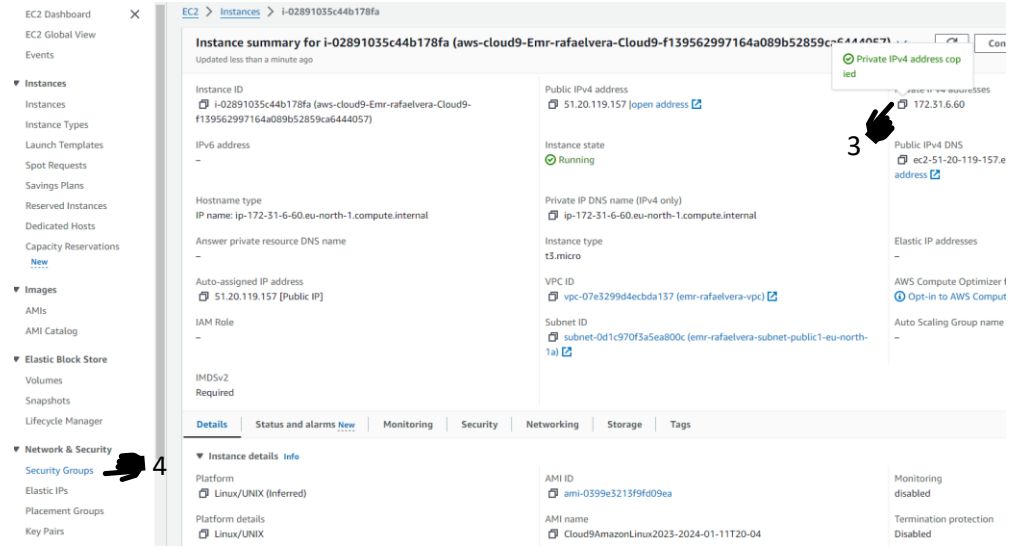
3.4.a.1. Get Aws Cloud9 instance IP



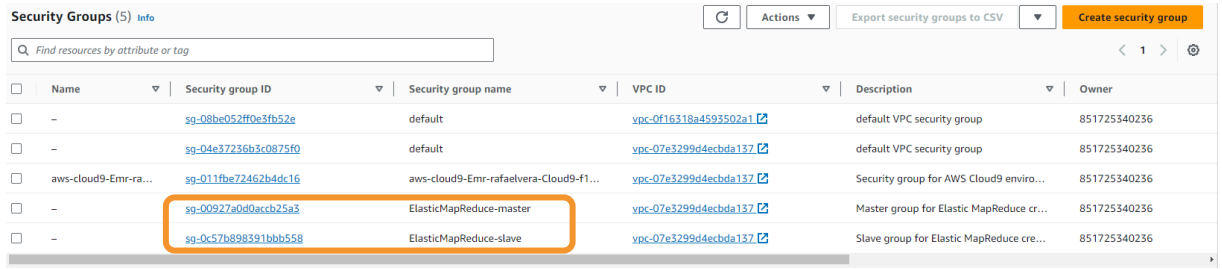
We need to get the AWS Cloud9 instance IP for having access from Cloud9 IDE to the EMR Cluster through an inbound rule.

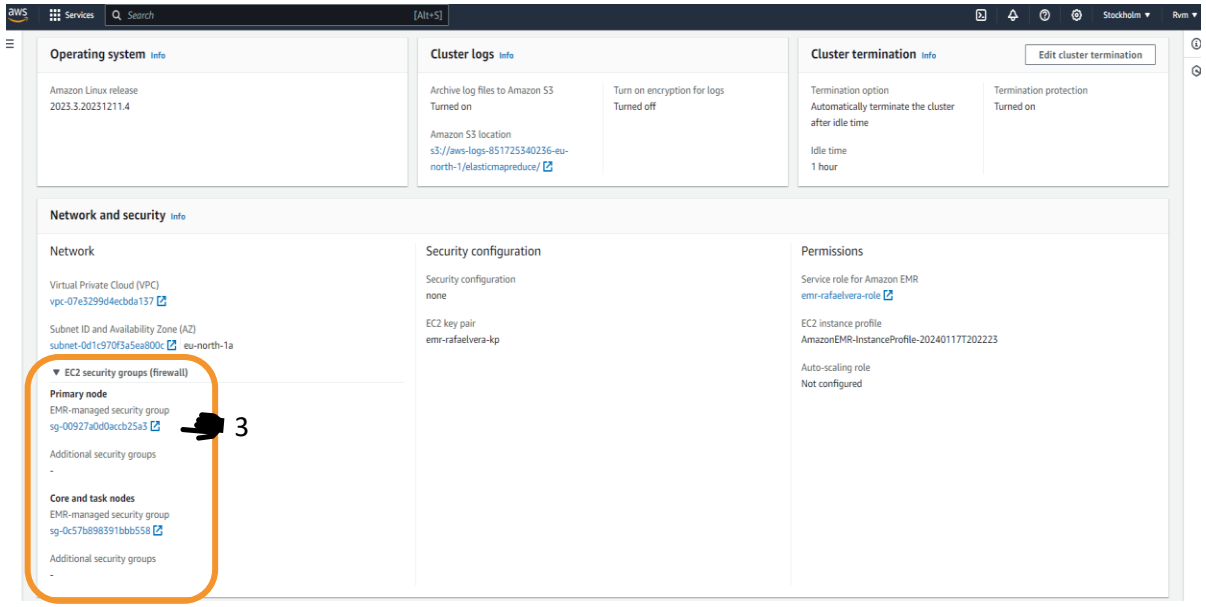
1. Select any of the already opened tabs and click on “**instances**”.
2. Search in the list the aswcloud9 instance and click on its “**instance ID**”. You will be redirected to this instance console.
3. Copy the instance **IPv4.**
4. Click on “**Security groups**”





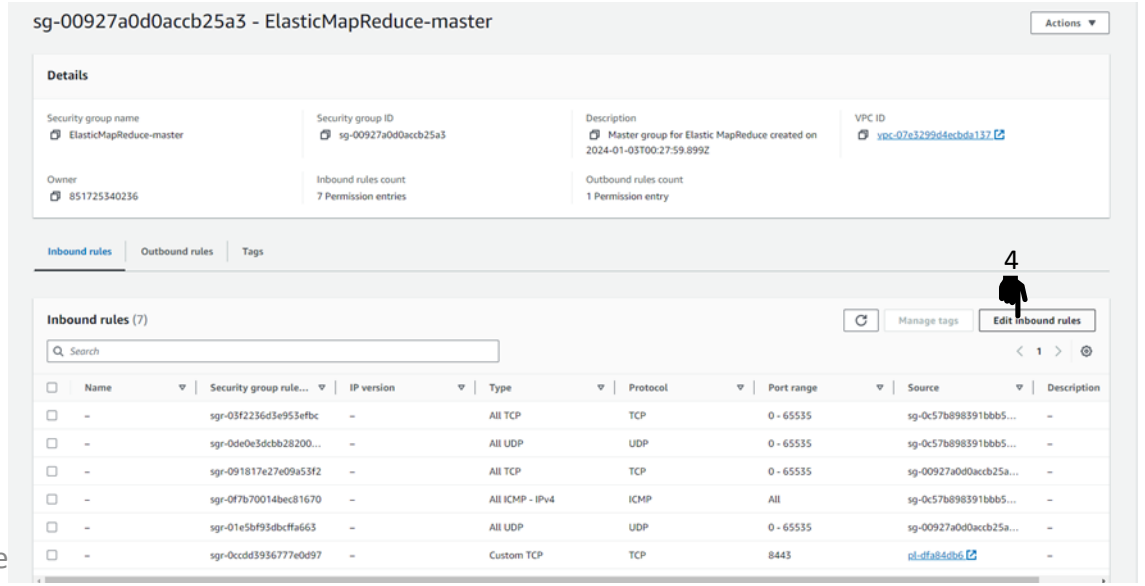
3.4.a.2. Security groups and Inbound rules





Security Groups:
This console lists all the security groups active in the account. And can be matched with those implemented on the EMR cluster.

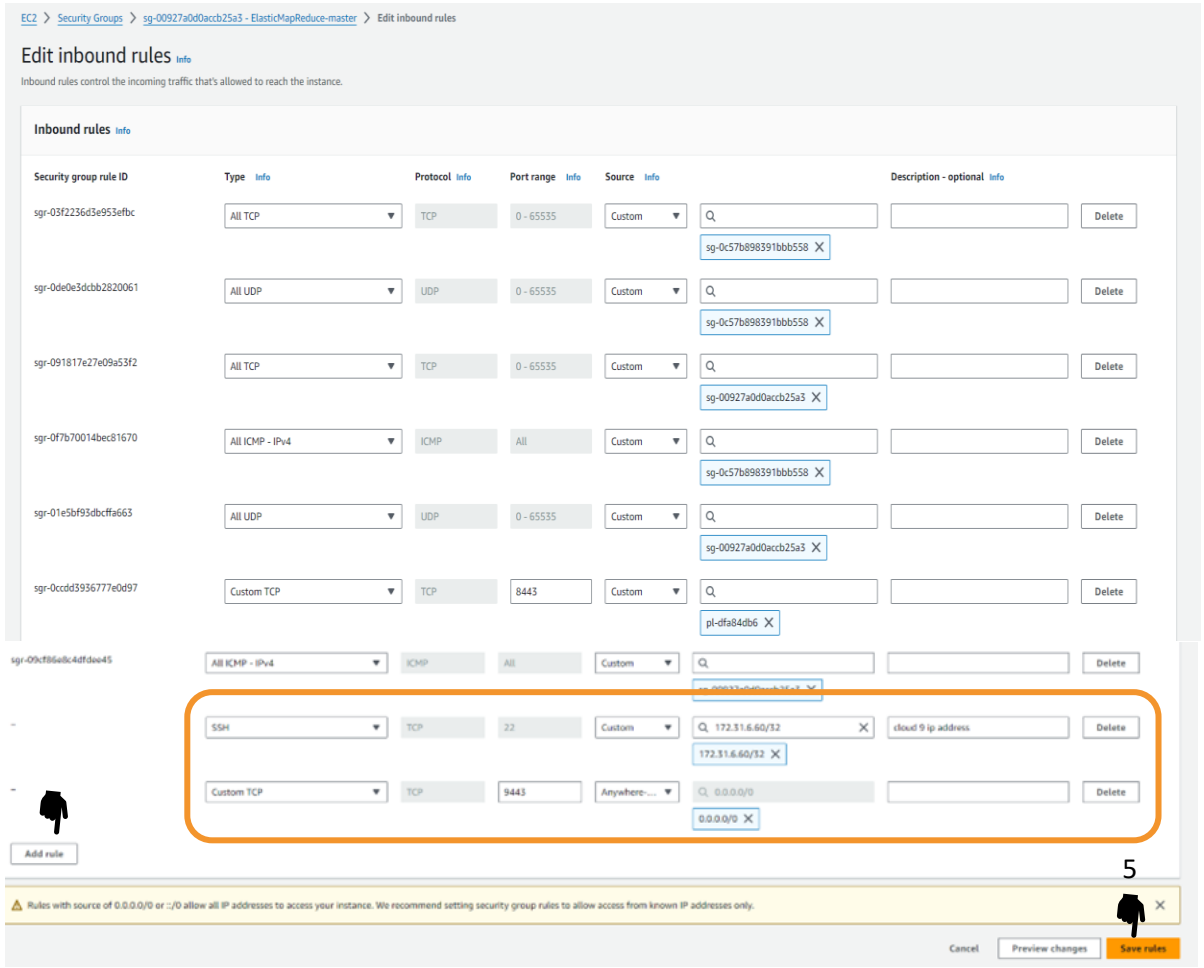
1. Go to the EMR Cluster console browser tab
2. Check the security groups implemented
3. Click on EMR Manager security group (**sg-00927a0d0accb25a3 – ElasticMapReduce-master**)
4. Click on “**Edit inbound rules**”



EMR cluster > Security groups > Create inbounds rules for cloud9

January 2024

3.4.a.3 Inbound rules

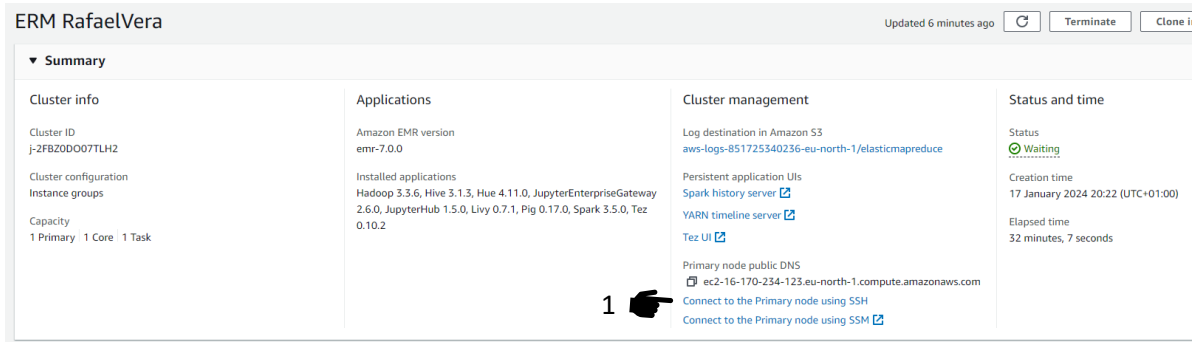


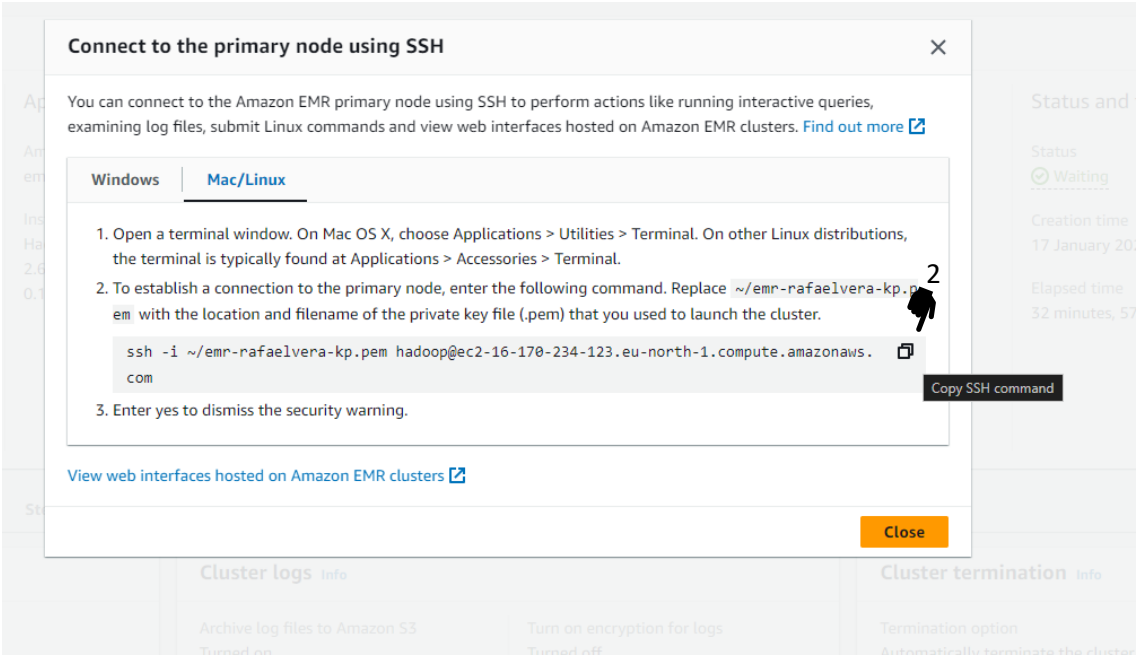
Edit inbound rules section of the AWS EC2 Security Group of sg-00927a0d0accb25a3 – ElasticMapReduce-master

This configuration defines the network access policies for the instances in this Security Group, specifying what kind of traffic is allowed to reach them.

1. Hit "**Add rule**" button
2. Add **SSH** rule as in the image. The IP is the **IPv4 copied before + /32** from the Cloud9 instance.
3. Hit "**Add rule**" button
4. Add Custom TCP rule as in the image. Allowing all addresses
5. Click on "**Save rules**"
6. Once done, lets continue with **section b** – get the ssh command from the EMR cluster to be pasted on Cloud9 IDE.

3.4.b.1 Get the SSH command



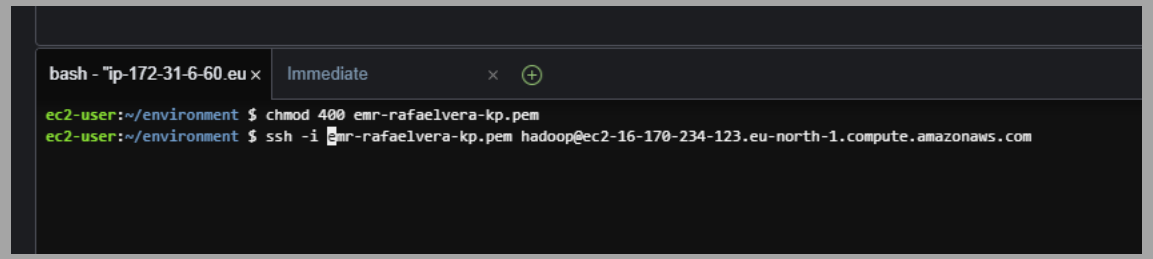


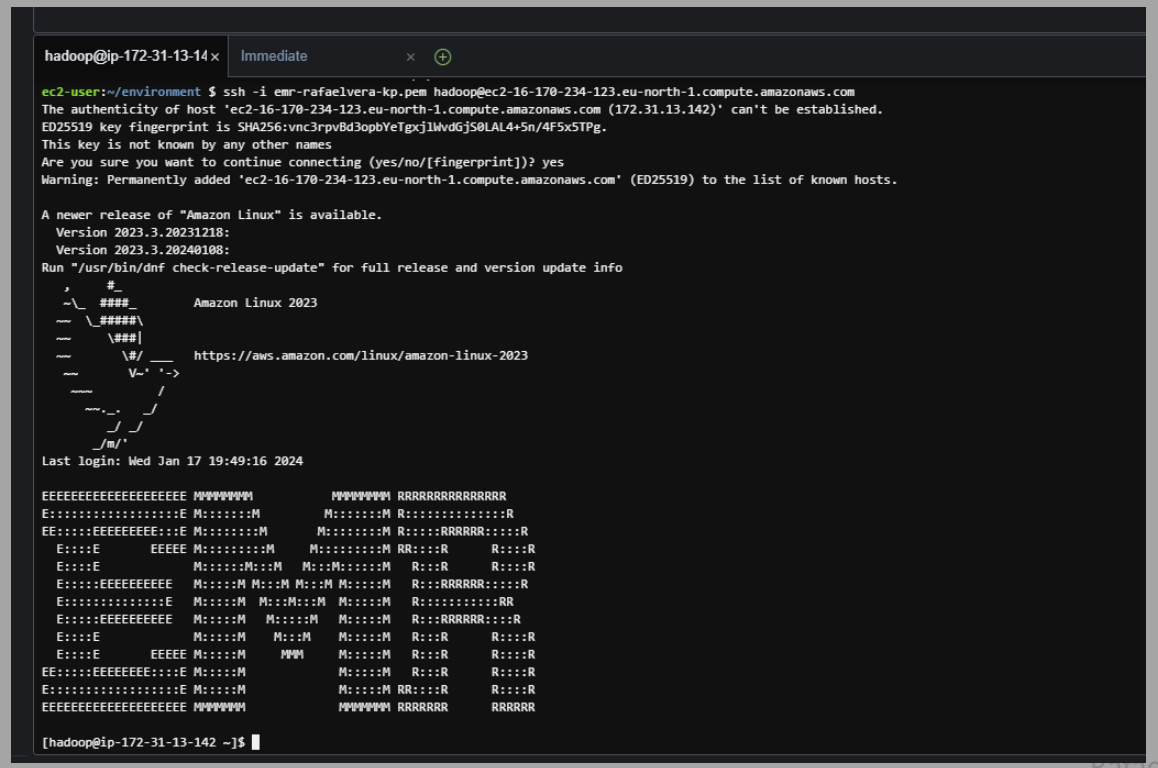
BACK to the EMR Cluster Summary Console:

1. Click on **"Connect to the primary node using SSH"** and a popups will be shown
2. Copy **the ssh command**

With this ssh command copied, now **you must go to AWS Cloud9 IDE again**

3.5. ssh EMR cluster connection





AWS Cloud9 integrated development environment (IDE):
Back here again we continue with the next command which ables to connect cloud9 IDE with the EMR cluster instance.

1. **Paste** de copied ssh command
2. Watch how the connection success

We continue next giving the spark job command to trigger the distributed processing. But we are not yet done, first we need to complete a couple steps.

3.6. Submitting Spark job command

```
       \#/ ___   https://aws.amazon.com/linux/amazon-linux-2023
      V~' '->
       /
     _/ _/
   _/m/'
Last login: Wed Jan 17 19:49:16 2024

EEEEEEEEEEEEEEEEEEE MMMMMMM           MMMMMMM RRRRRRRRRRRRRR
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
EEEEEEEEEEEEEEEEEEE MMMMMMM             MMMMMMM RRRRRRR      RRRRRR

[hadoop@ip-172-31-13-142 ~]$ nano spark-etl.scala
[hadoop@ip-172-31-13-142 ~]$ nano spark-etl.py
[hadoop@ip-172-31-13-142 ~]$ nano spark-etl.py
[hadoop@ip-172-31-13-142 ~]$ spark-submit spark-etl.py
```

Now we need to submit the spark job command with this structure:

spark-submit {Script.py} {S3 input folder / data} {S3 output folder/}

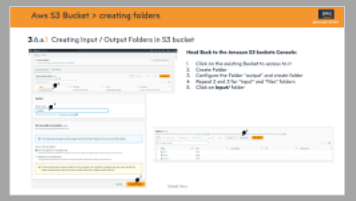
As you can see, we need this 3 components. So go ahead:

1. Run **nano spark-etl.py** to create the file within the EMR Instance
2. **Paste** the python code inside the nano window. (you can find it in my Github repository)
3. Save and close nano.
4. We have one of the three command components:

spark-submit spark-etl.py {S3 input folder / data} {S3 output folder/}

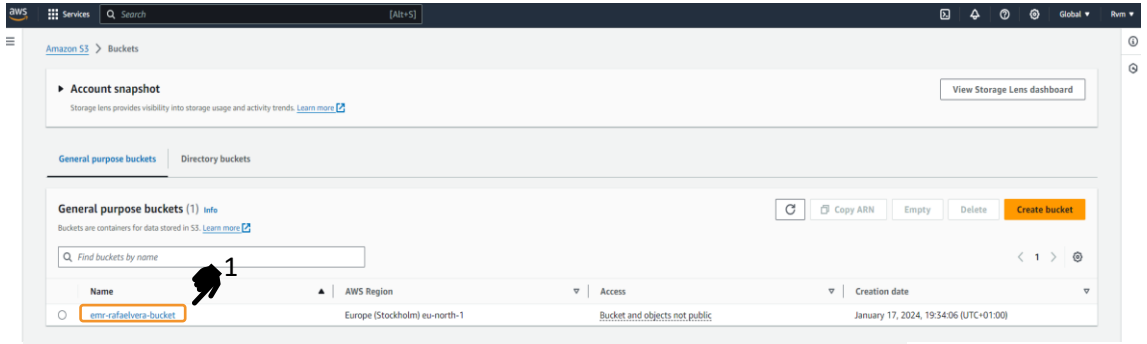
We need to create the folders for our S3 Bucket instance needed to submit spark job and grant EMR access to the S3 bucket.

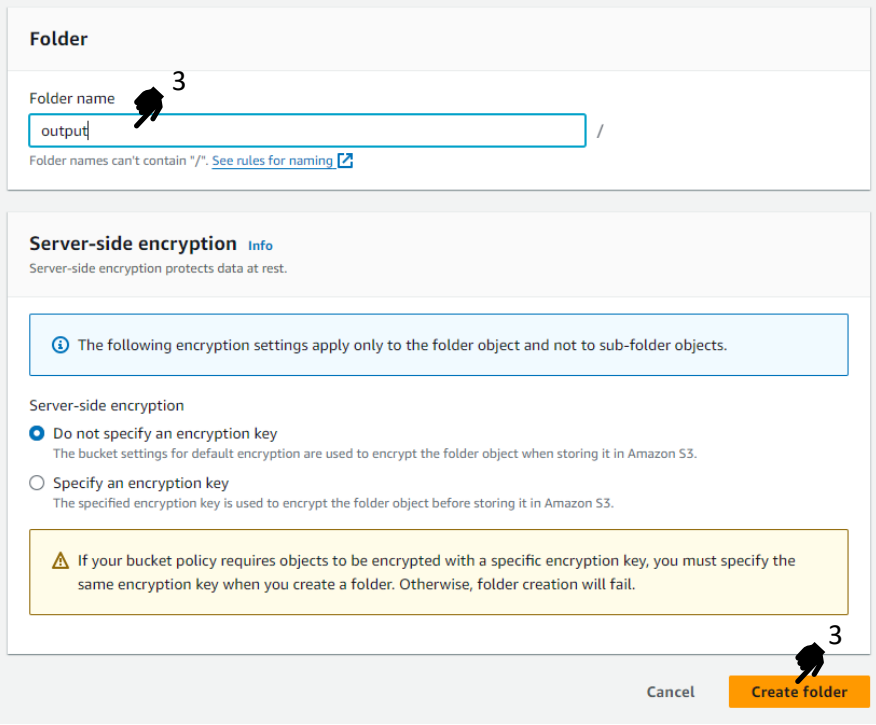
5. Create folders in S3 bucket (a)



6. Grant EMR access to S3 bucket (b)

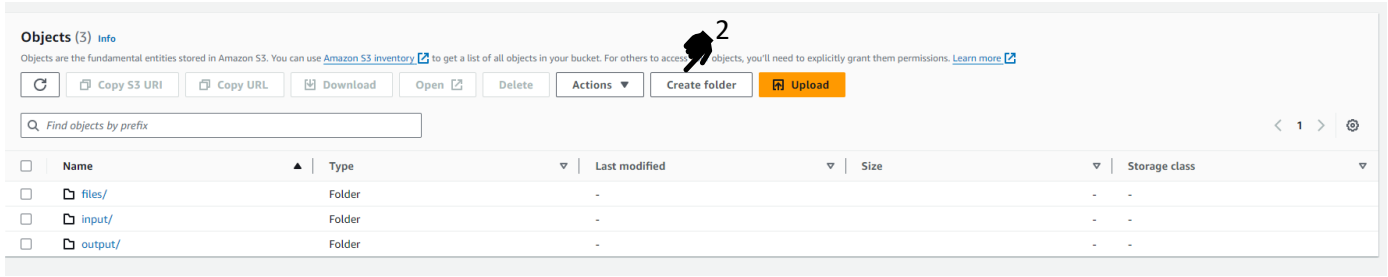
3.6.a.1 Creating Input / Output Folders in S3 bucket



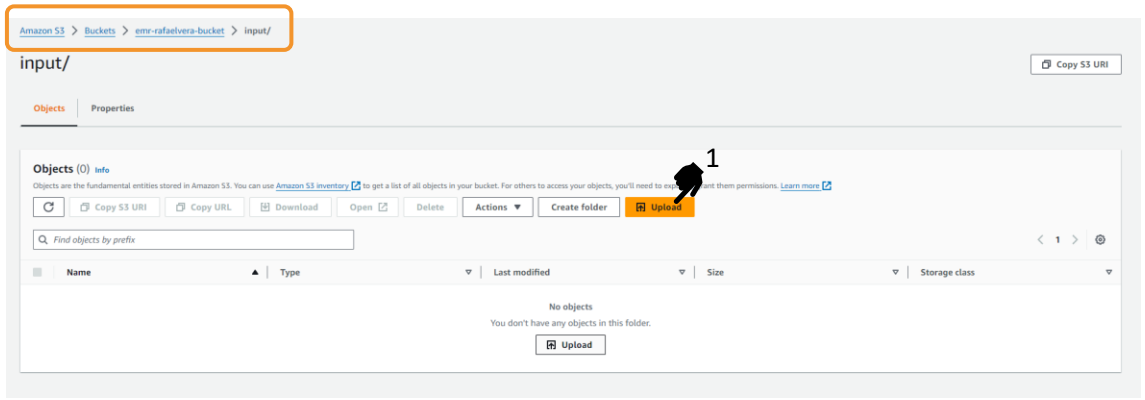


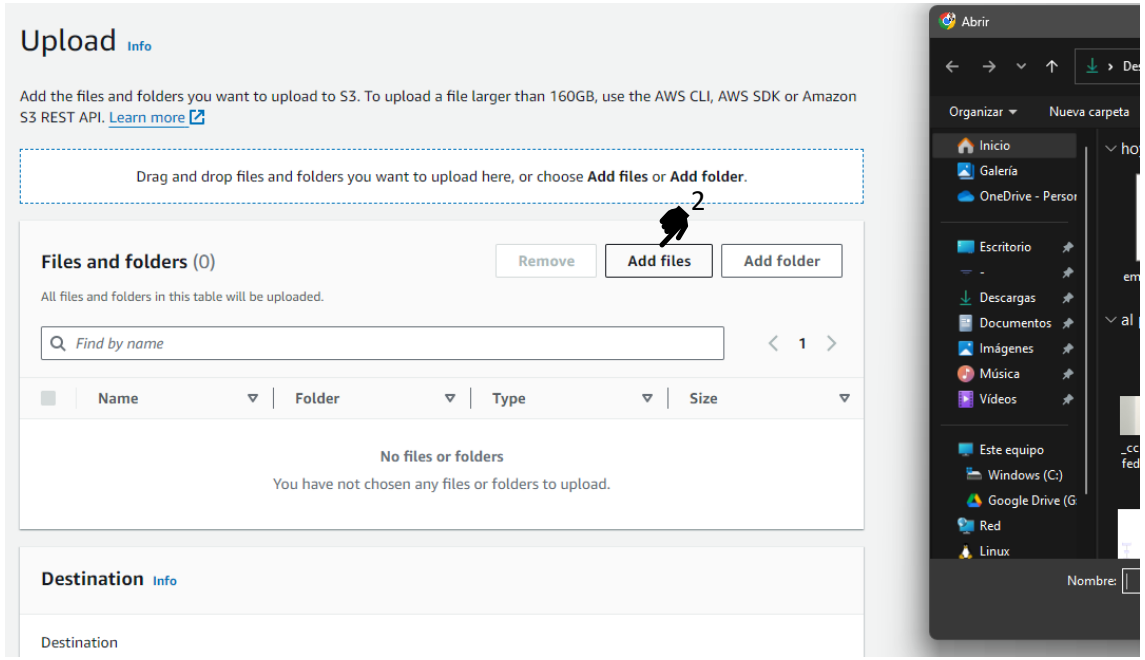
Head Back to the Amazon S3 buckets Console:

1. Click on the existing Bucket to access to it
2. Create Folder
3. Configure the Folder “output” and create folder
4. Repeat 2 and 3 for “input” and “files” folders
5. Click on **input/** folder



3.6.a.2 Uploading dataset to input folder

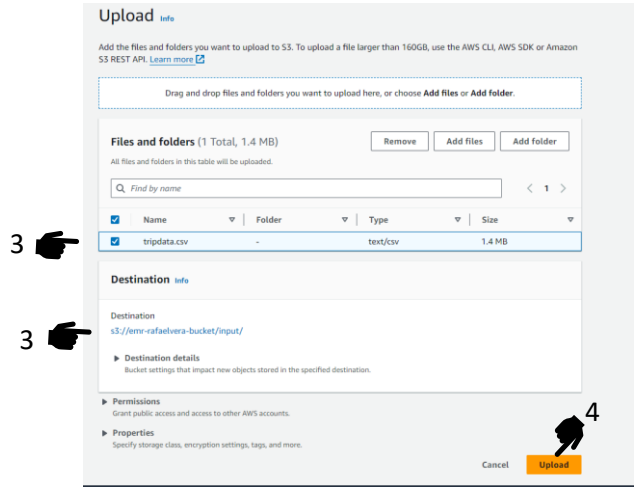




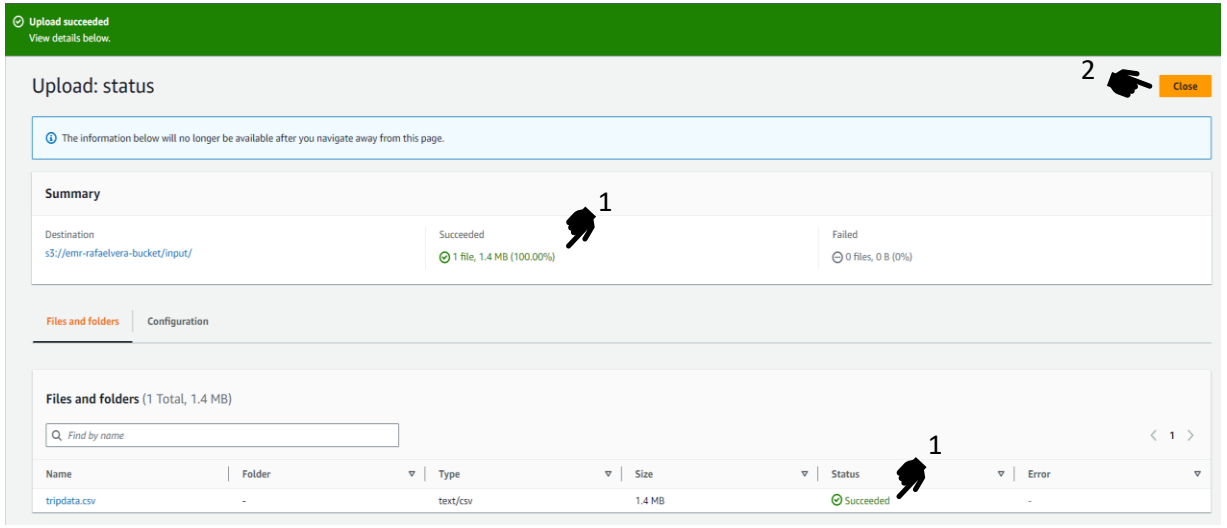
Amazon S3 > Buckets > emr-rafaelvera-bucket > input/

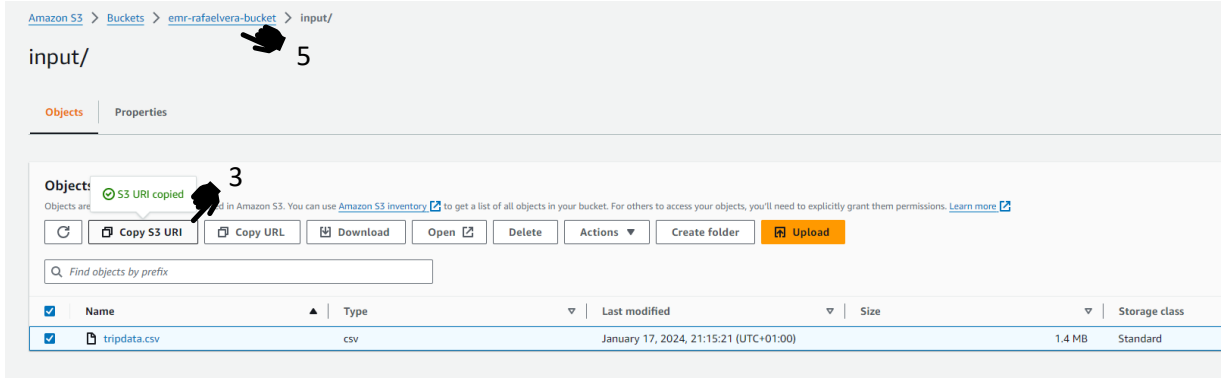
In this bucket folder we need to upload the dataset (.csv) to be processed

1. Click on **Upload**
2. Click on **Add files** and select the .csv from your PC. (.csv available in my Github repository)
3. See the file to be uploaded in the list and destination are correct.
4. Click on Upload



3.6.a.3 Successfully uploaded dataset to input folder





Amazon S3 > Buckets > emr-rafaelvera-bucket > input/

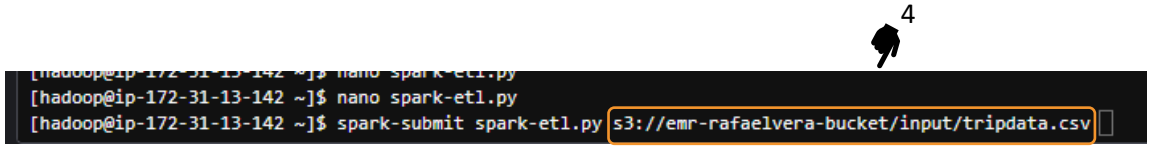
1. **Confirm** the tripdata.csv was successfully uploaded
2. Hit on "**Close**"
3. **Copy S3 URI**

Remember that we are here to load the dataset to be processed by the python script launched with the spark job.
And we still building the spark job command.

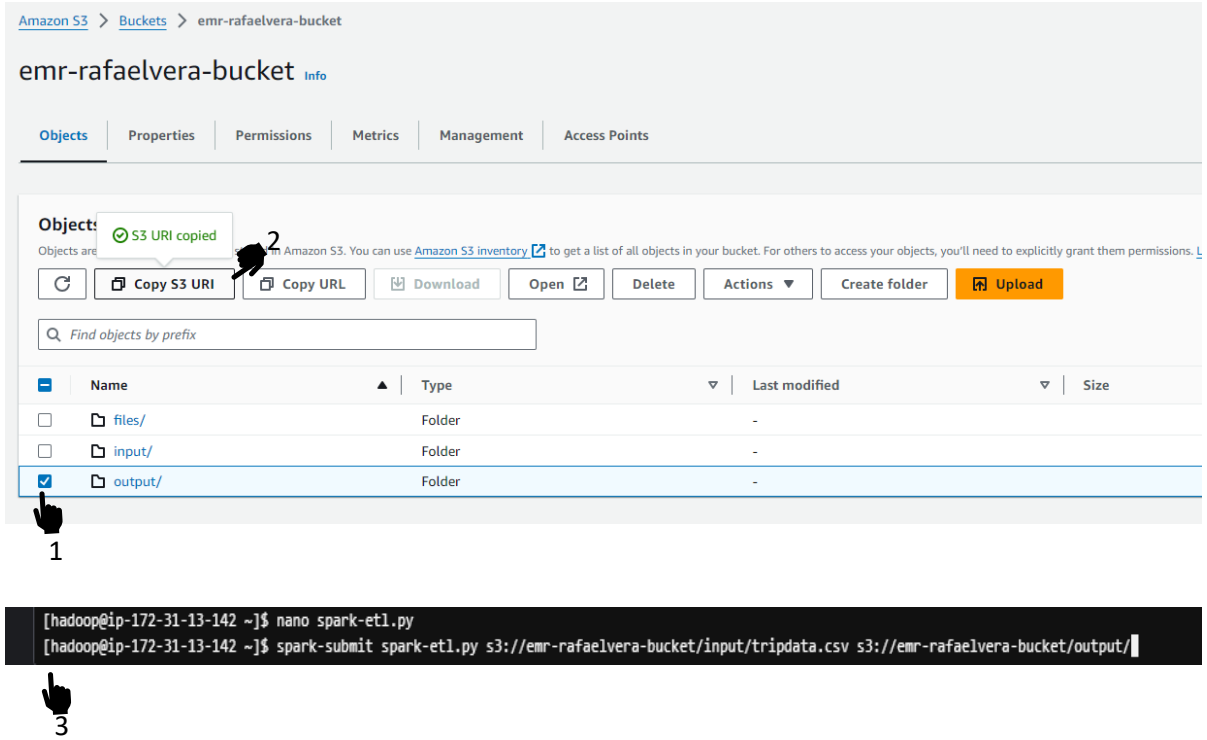
4. Go shortly to Cloud9 IDE and paste the S3 URI as the second command component.

spark-submit spark-etl.py s3://emr-rafaelvera-bucket/input/tripdata.csv {S3 output folder/}

5. Go back to this Bucket main console



3.6.a.4 Getting S3 URI from Output folder



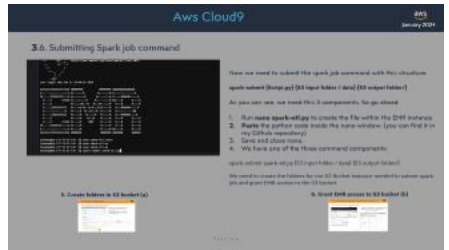
Amazon S3 > Buckets > emr-rafaelvera-bucket

1. **Select output/ folder**
2. **Copy S3 URI**
3. **Go shortly to Cloud9 IDE and paste the S3 URI** as the third command component.

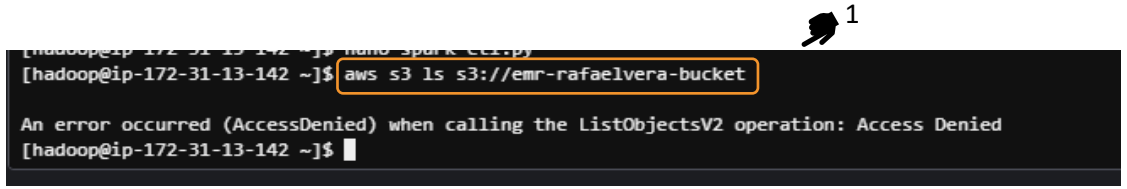
Remember that we are here to load the dataset to be processed by the python script launched with the spark job.
And we still building the spark job command.

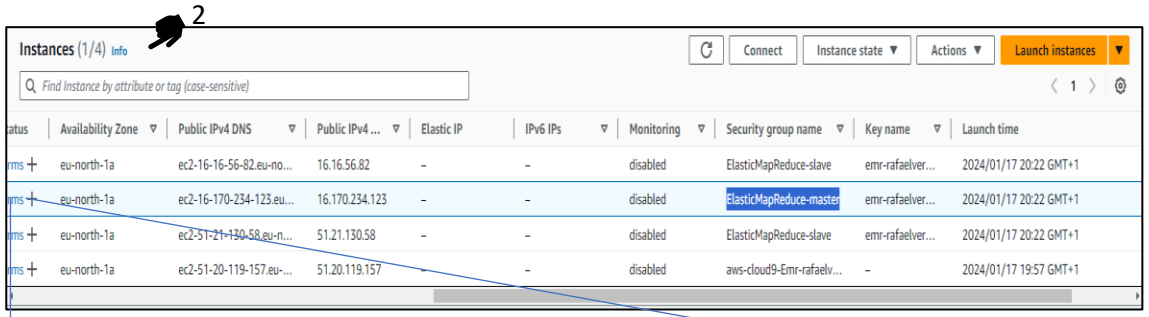
spark-submit spark-etl.py s3://emr-rafaelvera-bucket/input/tripdata.csv s3://emr-rafaelvera-bucket/output/

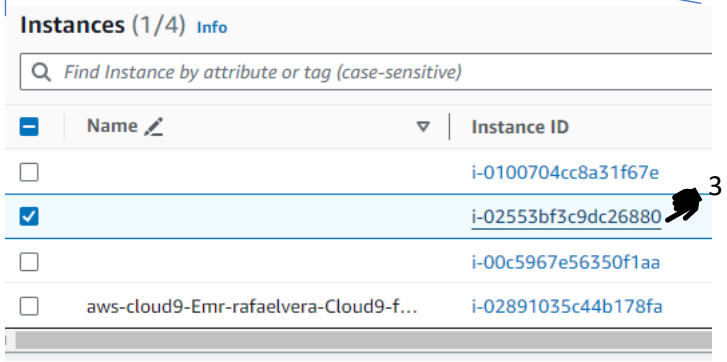
4. We have the **spark submit command complete. Not run it yet**
5. Copy the command handy. We'll add something to it later
6. **Before running this command**, we need to grant EMR access to this S3 Bucket folders. **Follow Section b**



3.6.b.1 Granting EMR Access to S3 Bucket folders



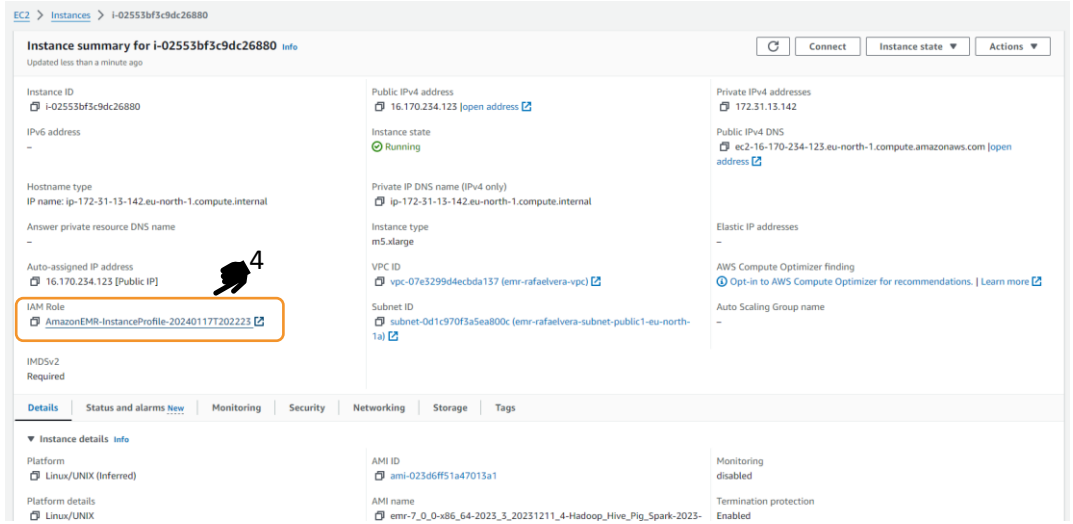




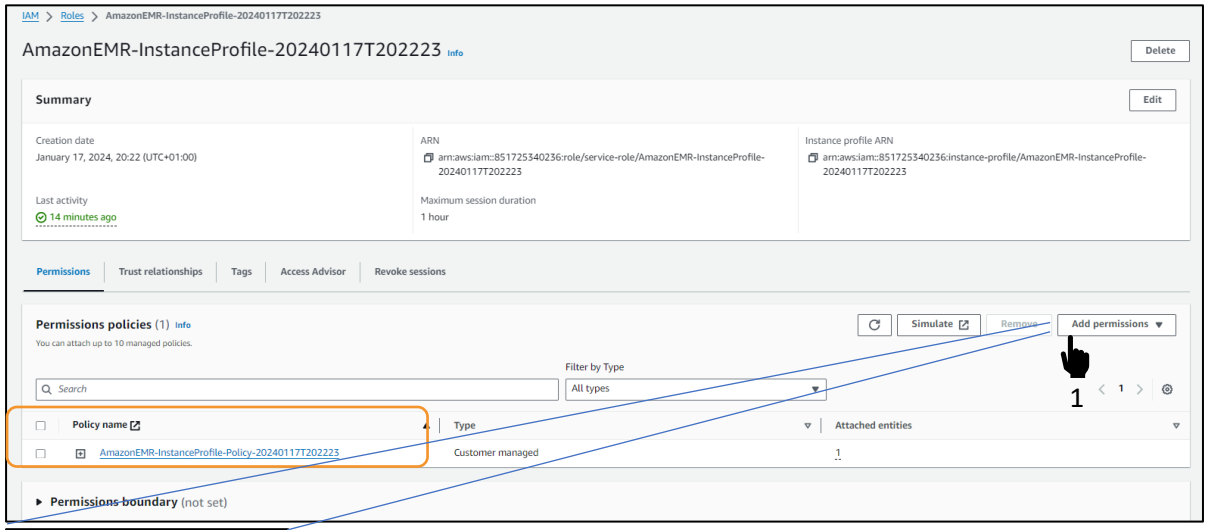
1. **Go shortly to Cloud9 IDE and try to list the content of the S3 Bucket:** (EMR is not yet granted to access S3 Bucket)

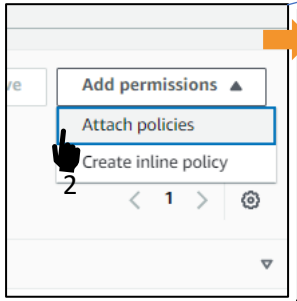
   aws s3 ls s3://emr-rafaelvera-bucket

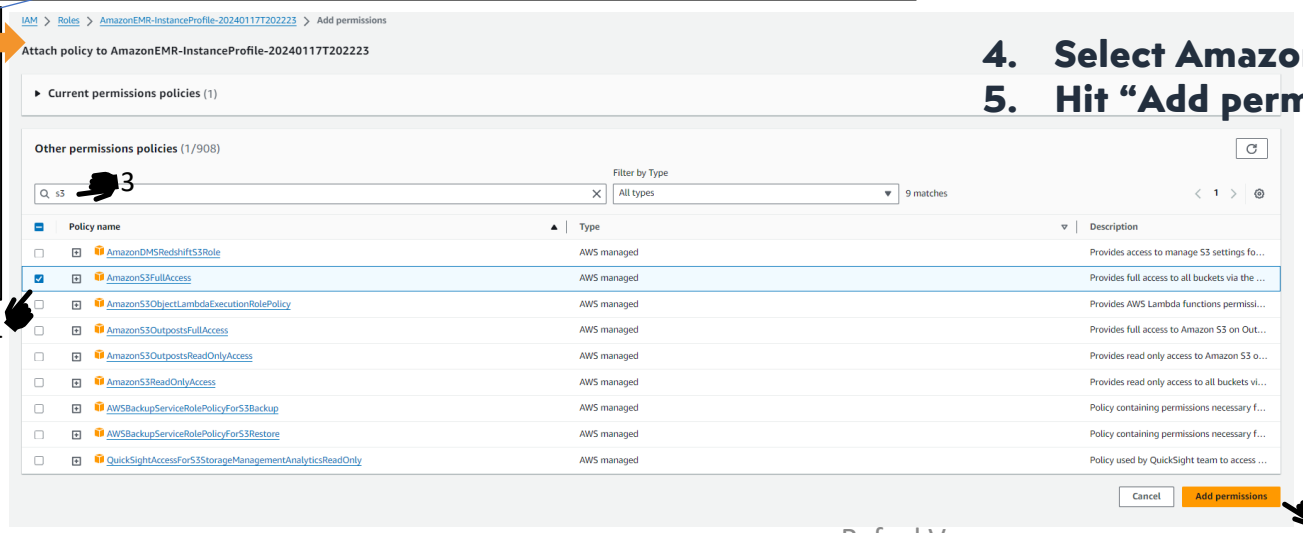
2. Select your Browser tab in which is the **EC2 Instances console.**
3. **Open** the **"ElasticMapReduce-master"** instance by clicking on the Instance ID
4. Inside the instance: **click on IAM Role**



3.6.b.2 Granting EMR Access to S3 Bucket folders







IAM > Roles > Amazon EMR-instanceProfile-20240117T202223

We are in the summary page of IAM role for our EMR instance.

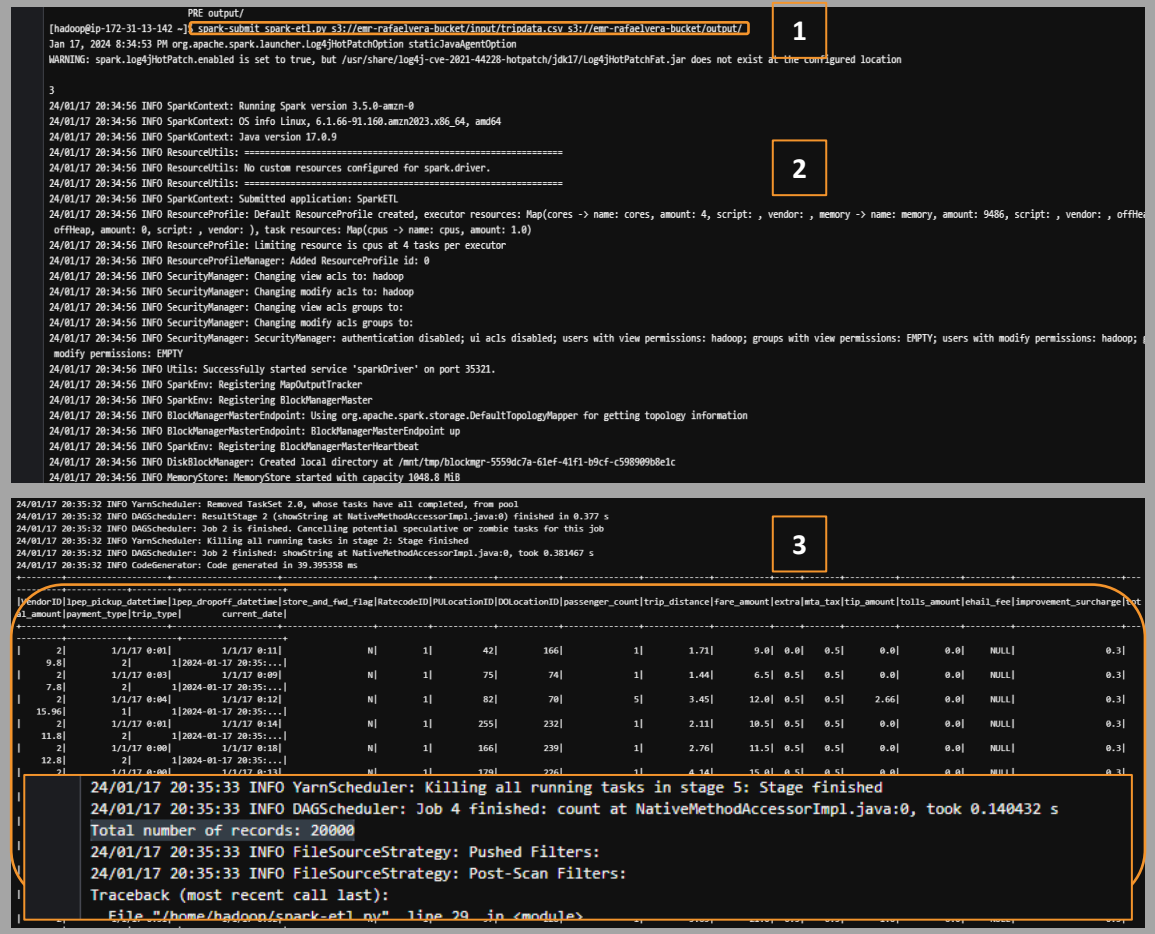
Under **the Permissions tab**, there is one customer-managed policy attached to this role, with the policy name being **AmazonEMR-InstanceProfile-Policy-20240117T202225.**

1. **Click on “Add Permissions”**
2. **Click on “Attach policies”**
3. **Search S3**
4. **Select AmazonS3FullAccess (*not a great practice)**
5. **Hit “Add permissions”**

6. Go to Cloud9 and run the S3 content listing command:

```
[hadoop@ip-172-31-13-142 ~]$ aws s3 ls s3://emr-rafaelvera-bucket
                           PRE files/
                           PRE input/
                           PRE output/
[hadoop@ip-172-31-13-142 ~]$
```

3.6.1 Submitting Spark job command



Create folders in S3 bucket (a) **> DONE**
Now EMR is able to execute the job pointing to the S3 bucket folders passed into the spark-job command.

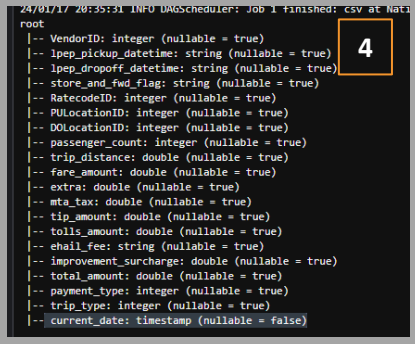
Grant EMR access to S3 bucket (b) > **DONE**
EMR is able to read and write files within the S3 Buckets

1. **Run the spark job. Add a subfolder into "output" to avoid overwriting issues (I'd mistaken by the first try)**

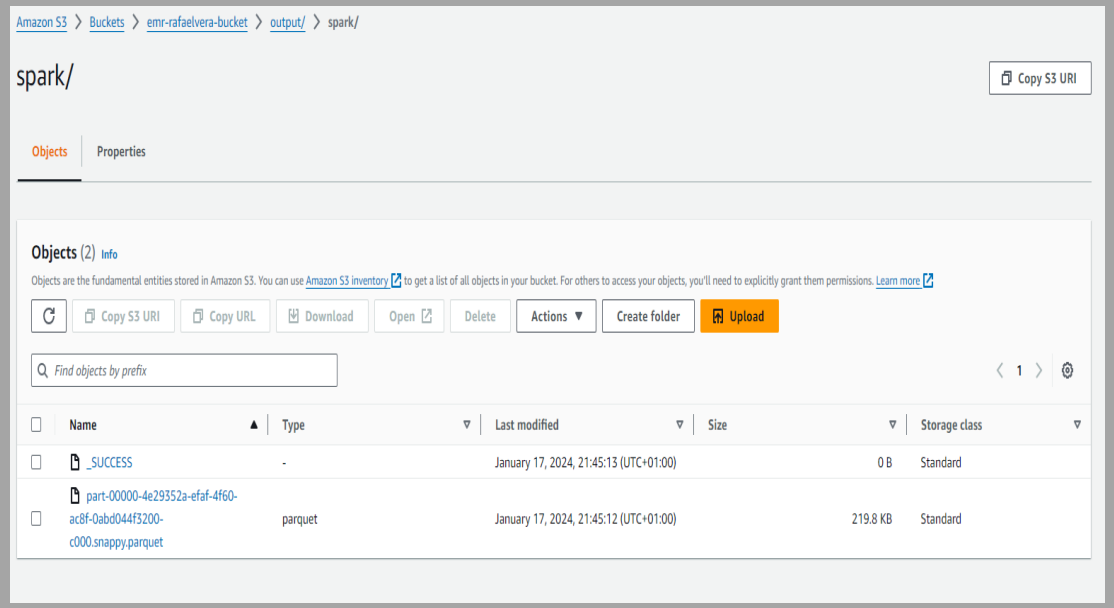
spark-submit spark-etl.py s3://emr-rafaelvera-bucket/input/tripdata.csv s3://emr-rafaelvera-bucket/output/**spark**

2. The distributed data processing started.
3. The processed dataset is printed
4. Dataset schema showing the **new added column**

Our spark-etl.py is a simple example of transformation, consisting in adding a new column "Current_date" (timestamp)



3.6.2 Check Submited Spark job output



Amazon S3 > Buckets > emr-rafaelvera-bucket > output/ > spark/

1. Go to the S3 Bucket folder “spark2” and see the outputted files after running the Spark Job

Two objects listed:

_SUCCESS - Marker file that indicates that the Spark job has been completed successfully. The file has a size of 0 bytes, which is typical for a _SUCCESS file, as it is empty and serves only as an indicator.

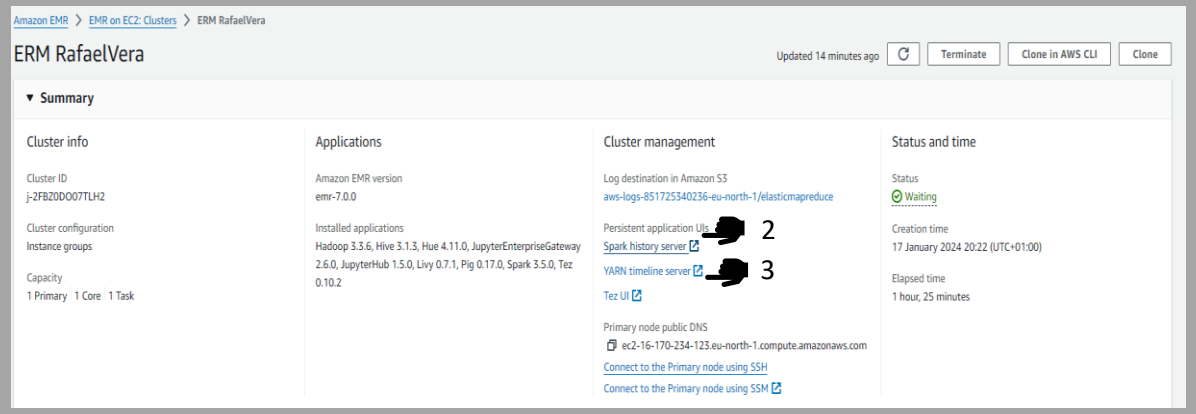
part-00000-4e29352a-efaf-4f60-a8cf-00bad443f200-c000.snappy.parquet - Data file in Parquet format that has been compressed using the Snappy algorithm.

Parquet is a columnar storage file format that is commonly used in the Hadoop ecosystem, and Snappy is a compression and decompression library that aims for very high speeds and reasonable compression. The file size is 219.8 KB.

Spark & Hadoop workout results

January 2024

4. Check Submited Spark job output

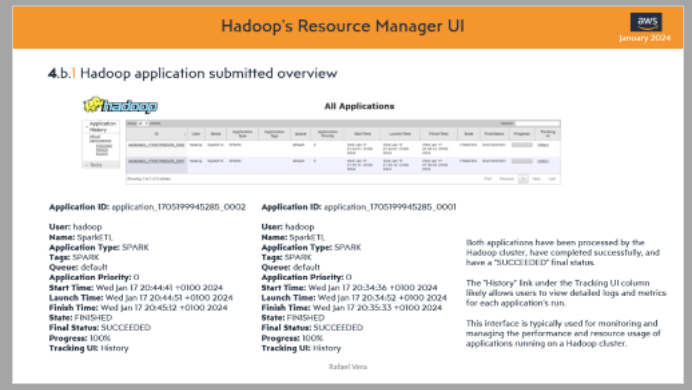


Amazon EMR > EMR on EC2 cluster > EMR RafaelVera

1. Come back again to the EMR cluster console

2. Check results in Spark console (a)

3. Check results in Hadoop console (b)



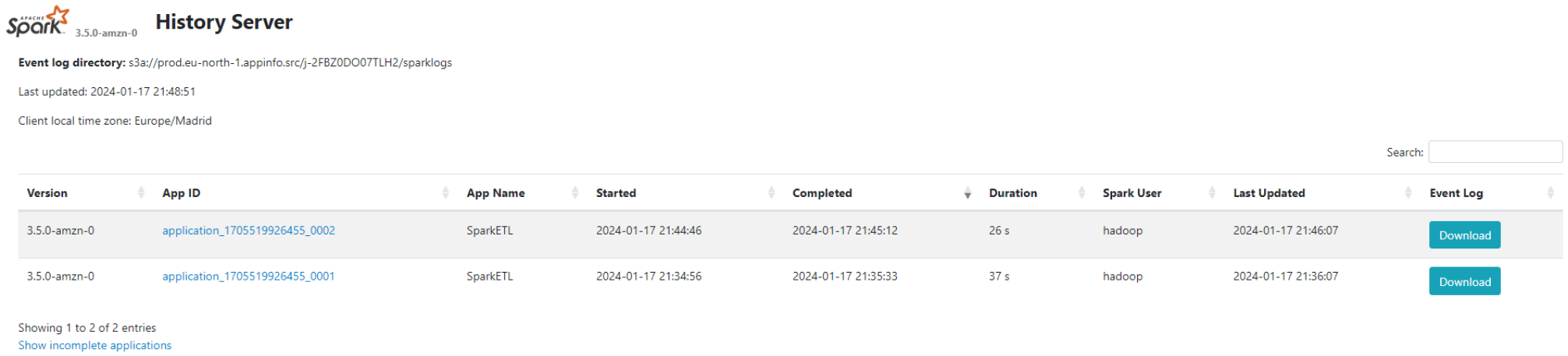
4.a.1 Apache Spark History Server web UI

This History Server UI is used to review and diagnose the performance of Spark applications after they have run

Lists two completed Spark applications :

the first application did not save the output correctly, as I mention before, because I run the Spark Job command ended on /output/ instead of /spark. This is due to since the job is run, the spark process generates the output folder passed within the command by default. So, if the folder is created in S3 Bucket manually, the process runs correctly but is not able to overwrite the folder. Such overwrite permission must be set explicit to this (I didn't do it) or simply create a subfolder from the job command.

The second submission did save the output correctly as shown before.



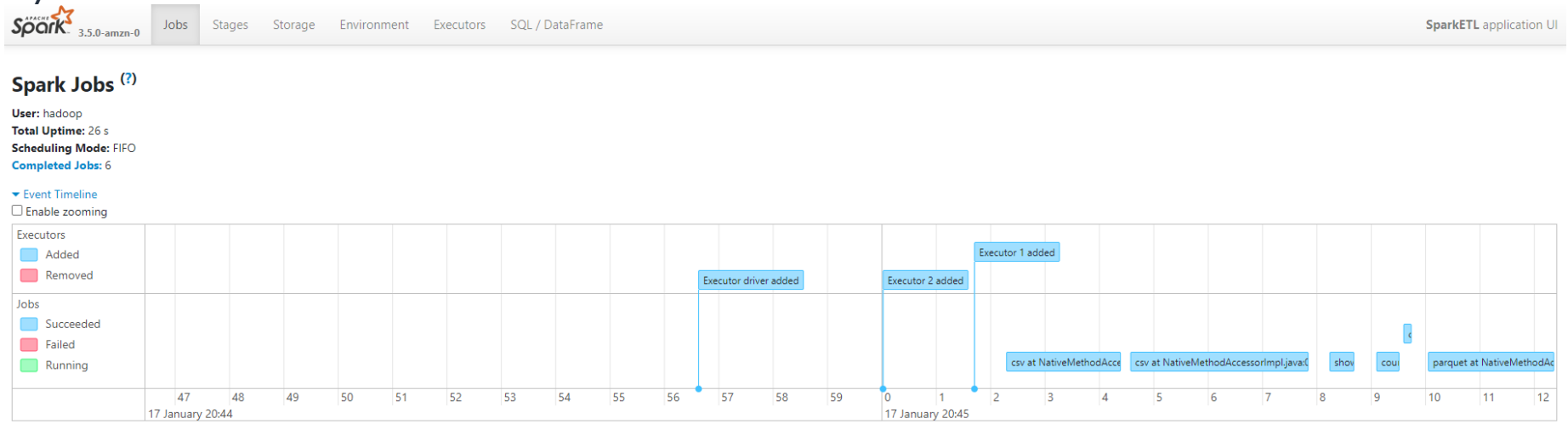
4.a.2 Event timeline for a Spark job

The timeline provides a graphical representation of events that occurred during the execution of the job.

key points:
user running the job : Hadoop. The total runtime of the job: 26 seconds. The scheduling mode: FIFO (First In, First Out)

Timeline itself:
17 January 2024 at seconds mark 52 and 54, two events labeled "Executor added" occur, (resources were allocated for the job). Following that, there are several stages marked as "stage 0," "stage 1," etc., which represent different stages of the Spark job. These are likely to be stages of computation that Spark has broken the job into for execution.

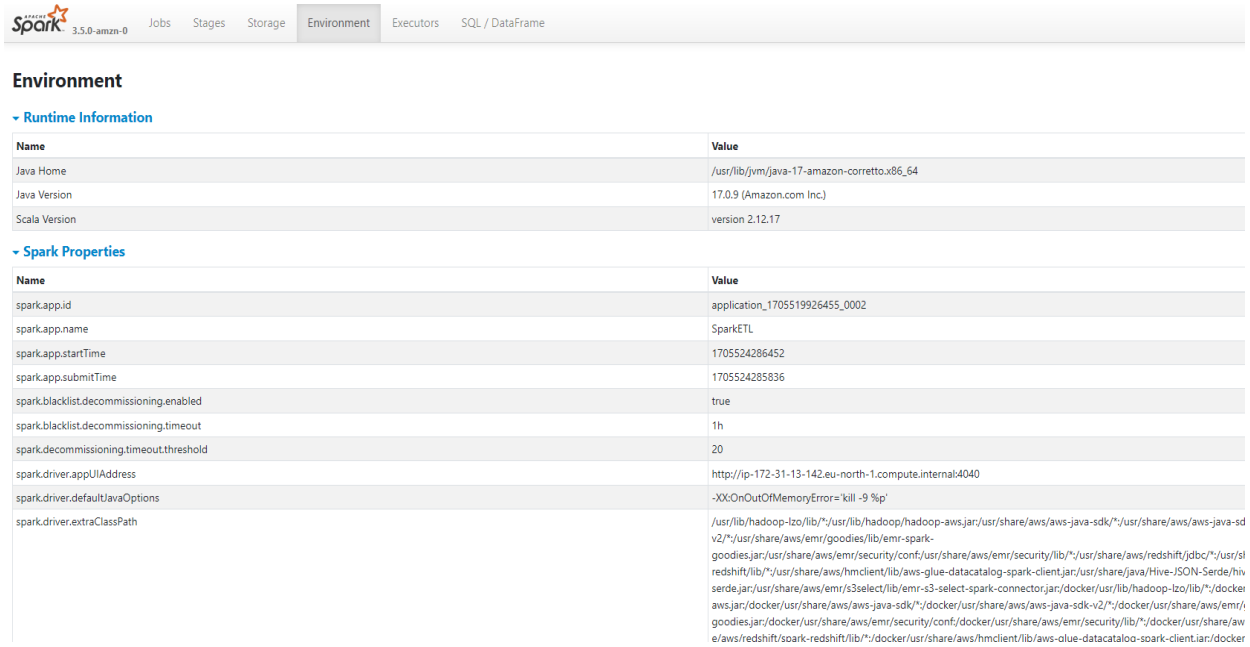
On the far right, there are two blue circles, one labeled "csv at NativeMethodAccessorImpl.java" and another labeled "parquet at NativeMethodAccessorImpl.java." These likely indicate the completion of tasks that involve reading or writing data in CSV and Parquet formats, respectively.



Apache Spark History Server web UI | Job results

January 2024

4.a.4 Environment



Amazon Corretto is a performance-optimized, long-term support OpenJDK distribution by Amazon, suitable for Java applications such as Apache Spark.

It is open-source and multi-platform compatible. Proper configuration of Spark properties, including application naming, memory allocation, and dynamic resource management, is crucial for optimal performance and stability. Corretto's use in Spark can notably impact performance.

Shows various runtime information and Spark properties set for the application.

Runtime Information:

Java Home: /usr/lib/jvm/java-17-amazon-corretto.x86_64, (Amazon Corretto 17 as the Java environment)
Java Version: Version 17.0.9
Scala Version: 2.12.17.

Spark Properties:
spark.app.id and spark.app.name are shown, with name SparkETL.
spark.executorEnv.JAVA_HOME and spark.yarn.appMasterEnv.JAVA_HOME Amazon Corretto Java 17 path .
Other properties such as spark.driver.memory, spark.executor.memory, spark.executor.cores, spark.dynamicAllocation.enabled, spark.eventLog.enabled, spark.master, spark.sql.warehouse.dir, spark.blacklist.decommissioning.enabled, spark.shuffle.service.enabled, and spark.ui.filters are listed, each influencing various aspects of Spark application performance, resource allocation, and user interface configuration.

4.a.5 Spark properties extended

spark.driver.memory (2048M) and spark.executor.memory (9486M): These properties define the amount of memory allocated to the Spark driver and executors, respectively. They are crucial for the performance and stability of Spark applications, especially when processing large volumes of data.

spark.executor.cores (4) and spark.emr.default.executor.cores (4): These configure the number of CPU cores to be allocated to each executor. They are important for application parallelism and performance.

spark.dynamicAllocation.enabled (true): This property, when enabled, allows Spark to automatically adjust the number of executors used by the application based on the workload. It is useful for optimizing resource usage.

spark.eventLog.enabled (true) and spark.eventLog.dir (hdfs:///var/log/spark/apps): These properties enable and define the location for Spark event logs, which are essential for debugging and performance analysis of the application.

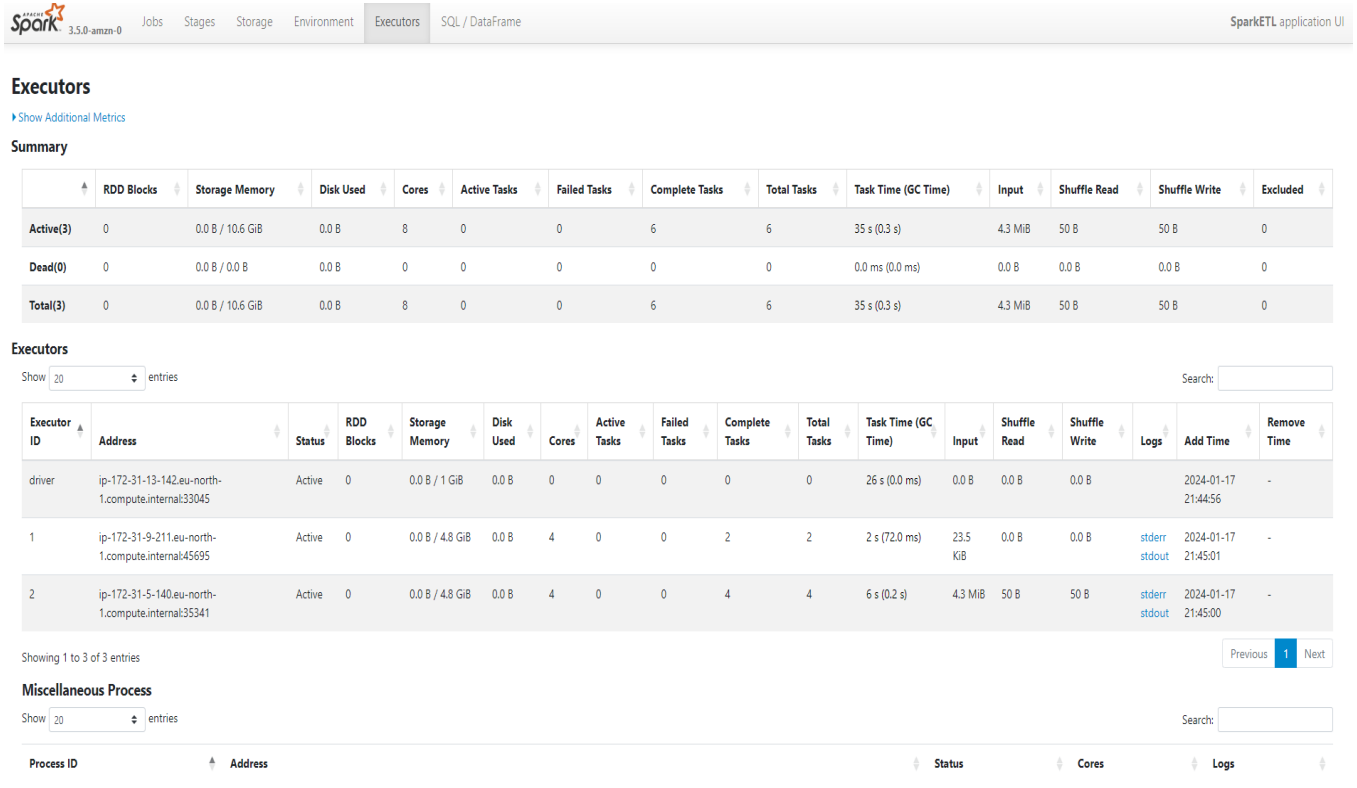
spark.master (yarn): Indicates the cluster manager used for running the application. In this case, it is YARN, which is common in Hadoop environments.

spark.sql.warehouse.dir (hdfs:///user/spark/warehouse): This is the default location for storing table data when working with Spark SQL and DataFrames.

spark.blacklist.decommissioning.enabled (true) and spark.blacklist.decommissioning.timeout (1h): These properties are related to the management of unreliable or unstable resources in the cluster.

spark.shuffle.service.enabled (true): This setting is important for the performance of shuffle operations in Spark, especially in large-scale applications.

4.a.6 Executors



Rafael Vera

Displays the metrics and status of Spark executors for the running application.

Executors Summary: Shows an overview with one active driver and two active executors, indicating no failed tasks.

Memory Usage: The driver has 1 GB of storage memory, and each executor has 4.8 GB allocated, with no disk used by any.

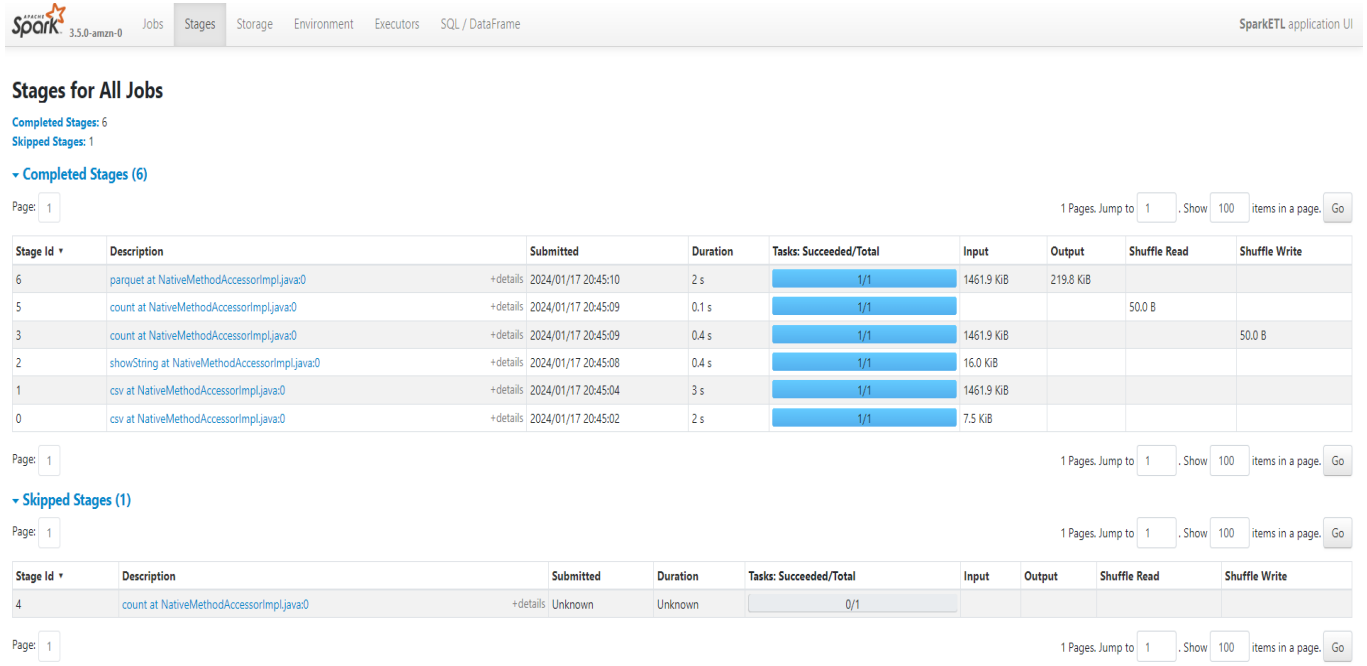
Tasks: The driver has not completed any tasks, which is typical, as the driver coordinates the executors. Executors have completed 6 tasks.

Task Time: The total task time is 35 s, with individual task times listed for each executor.

Shuffle Read/Write: minimal shuffle activity, indicating the tasks did not involve significant data shuffling or it's an initial stage of processing.

Logs: Each executor has a stdout and stderr log available for review.

4.a.7 Stages



The Spark application has processed several stages involving reading from and writing to CSV and Parquet files, and performing count operations

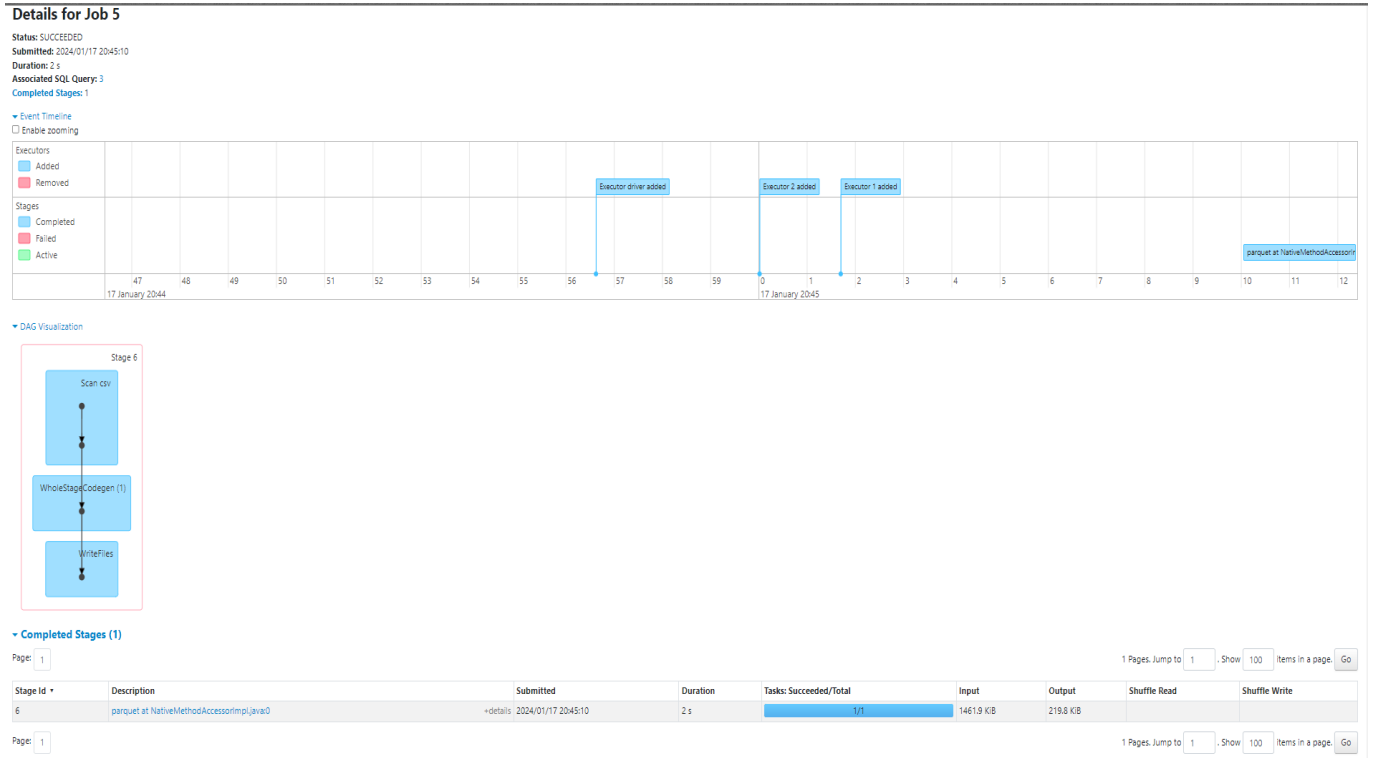
Completed Stages (6):

Stage 6: Involved an operation with Parquet files, took 2.5 seconds, processed 163.9 KB of input and produced 219.8 KB of output.
Stage 5: Involved a count operation, took 0.1 seconds, processed 163.9 KB of input, and no output is listed.
Stage 3: Another count operation, took 0.4 seconds, same input size as Stage 5 and no output.
Stage 2: Show operation, took 0.4 seconds, with 160 KB input and 7.5 KB output.
Stage 1: CSV operation, took 3.5 seconds, with 163.9 KB input and no output listed.
Stage 0: CSV operation, took 2 seconds, with 7.5 KB input and no output listed.
Each stage had one task that succeeded.

Skipped Stages (1):

Stage 4: A count operation that was skipped, therefore no duration, task, input, or output information is available.

4.a.8 Stages | Details for Job 5



Job ID: 5
Status: SUCCEEDED
Submission Time: 2024/01/17 20:45:10
Duration: The job took a total of 2.5 seconds to run.
Stages: 1 stage is completed, and 0 stages are skipped.
Active Stages: 0, Completed Stages: 1
Skipped Stages: 0, Failed Stages: 0

In the Event Timeline, the following events are marked:
Executors are added at three different points in time.
A stage is marked as completed.

The DAG Visualization (Directed Acyclic Graph) of the job's stages, with one stage, labeled "Stage 1", indicating the flow of tasks within the job.

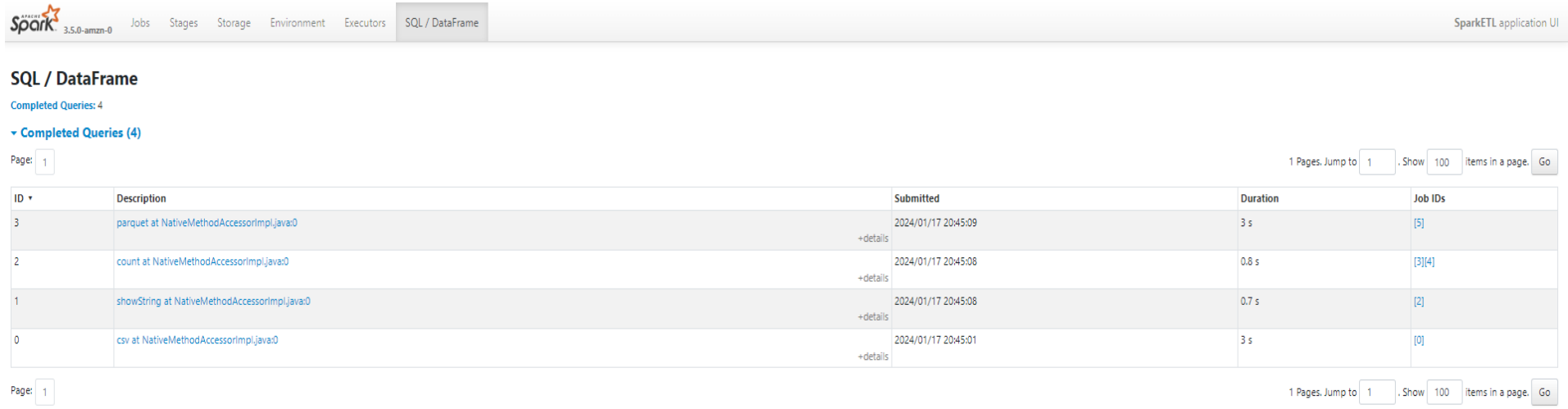
The Completed Stages section lists:

Stage ID 6: Described as "parquet at NativeMethodAccessorImpl.java", submitted on January 17, 2024, and took 2.5 seconds to complete. The input size was 163.9 KB, and the output size was 219.8 KB. There was 1 task for this stage, and it succeeded.

Apache Spark History Server web UI | Job results

aws
January 2024

4.a.9 SQL / DataFrame

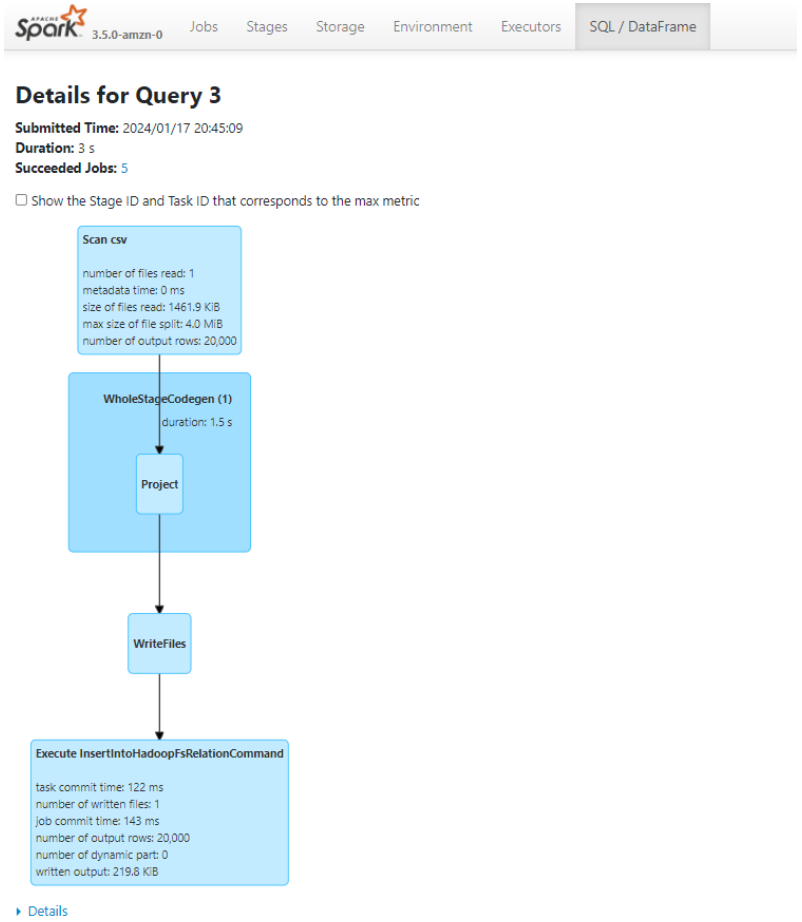


List of completed queries for a Spark application
This tab is useful for performance tuning and debugging of Spark SQL queries, as it provides a clear mapping from high-level DataFrame operations to the underlying jobs executed by Spark.

There are four completed queries listed: ID (0 to 3).
Each query has a description that includes a method invocation, likely indicating a DataFrame operation. These descriptions involve operations such as 'save' or 'count', and they reference a Java method, likely where the DataFrame operation is triggered.

Submitted: submission times for all queries.
Duration: how long each query took to complete.
Job ID: links each query to the corresponding Spark job, providing a way to trace the SQL/DataFrame operations back to the broader context of the Spark job they are part of.

4.a.10 SQL / DataFrame | Details for Query 3



Continue to section (b) Hadoop

Query execution plan is visualized as a DAG:
Scan csv: read 1 file with metadata time of 0 ms, and the size of files read was 1461.9 KB. The maximum size of file split was 40 MB, and the number of output rows was 20,000.

WholeStageCodegen (1): A performance optimization feature in Spark SQL that compiles parts of the query to bytecode. This stage had a duration of 1.5 seconds.

Project: A transformation operation that is likely manipulating or transforming the data in some way. In our case, adding a new column to the dataframe with current date (datetime)

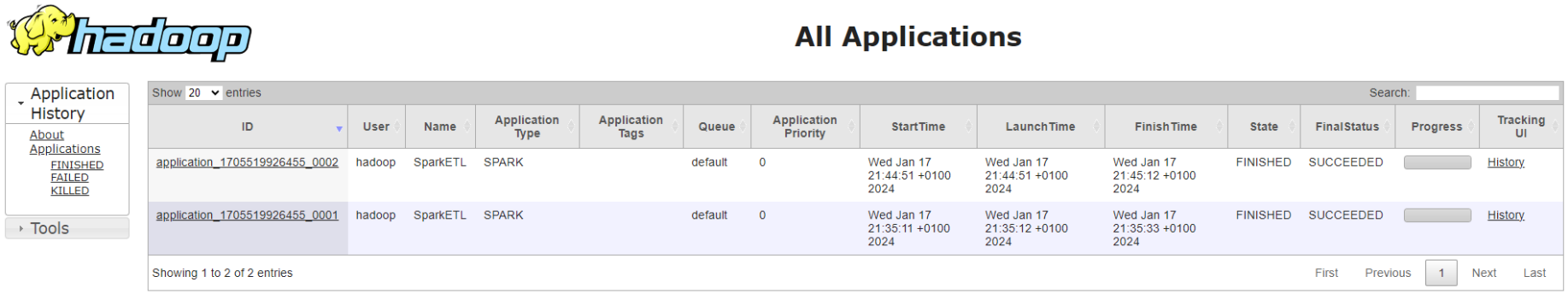
WriteFiles: This operation is responsible for writing the output of the transformation to storage.

Execute InsertIntoHadoopFsRelationCommand: This command indicates an action to insert data into a Hadoop FileSystem (HDFS) relation (table or file), and it involved**:**

Task commit time: 122 ms, **Number of written files:** 1
Job commit time: 143 ms**, Number of output rows:** 20,000
Number of dynamic partitions: 0, **Written output:** 219.8 KB

The diagram visually represents the execution plan of the query, illustrating the flow from reading a CSV file, processing the data (projecting), and then writing the results to the filesystem. The metrics provide insights into the efficiency of the operation, including the speed of reading, processing, and writing the data.

4.b.1 Hadoop application submitted overview



Application ID: application_1705199945285_0002

User: hadoop
Name: SparkETL
Application Type: SPARK
Tags: SPARK
Queue: default
Application Priority: 0
Start Time: Wed Jan 17 20:44:41 +0100 2024
Launch Time: Wed Jan 17 20:44:51 +0100 2024
Finish Time: Wed Jan 17 20:45:12 +0100 2024
State: FINISHED
Final Status: SUCCEEDED
Progress: 100%
Tracking UI: History

Application ID: application_1705199945285_0001

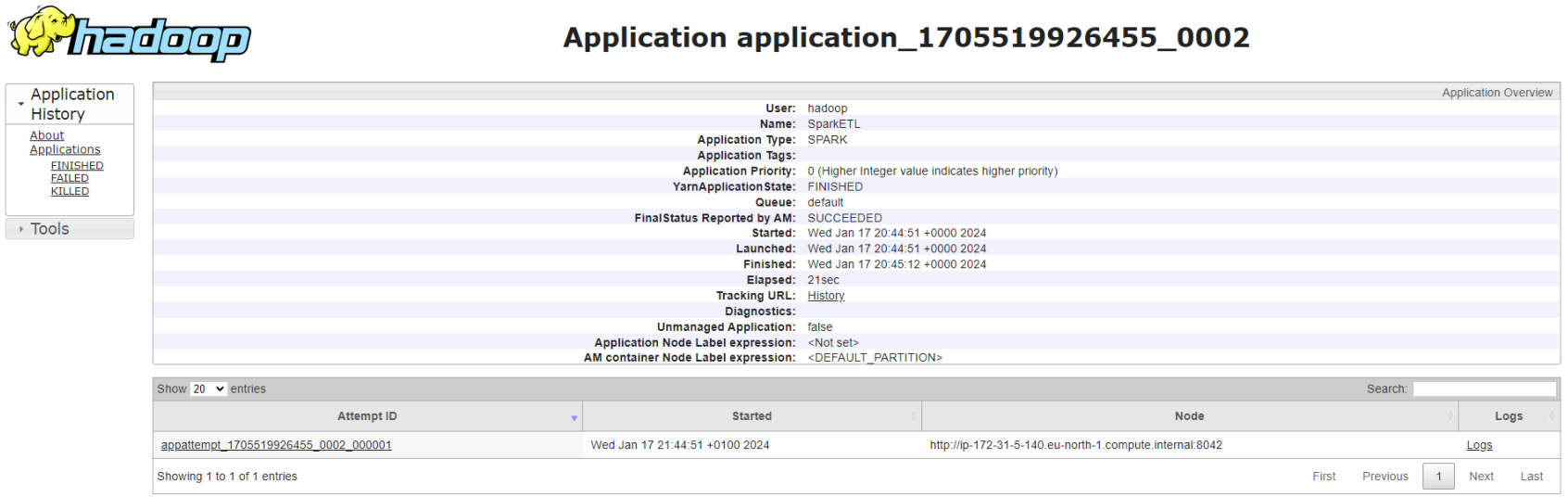
User: hadoop
Name: SparkETL
Application Type: SPARK
Tags: SPARK
Queue: default
Application Priority: 0
Start Time: Wed Jan 17 20:34:36 +0100 2024
Launch Time: Wed Jan 17 20:34:52 +0100 2024
Finish Time: Wed Jan 17 20:35:33 +0100 2024
State: FINISHED
Final Status: SUCCEEDED
Progress: 100%
Tracking UI: History

Both applications have been processed by the Hadoop cluster, have completed successfully, and have a "SUCCEEDED" final status.

The "History" link under the Tracking UI column likely allows users to view detailed logs and metrics for each application's run.

This interface is typically used for monitoring and managing the performance and resource usage of applications running on a Hadoop cluster.

4.b.2 Hadoop processed Spark application details

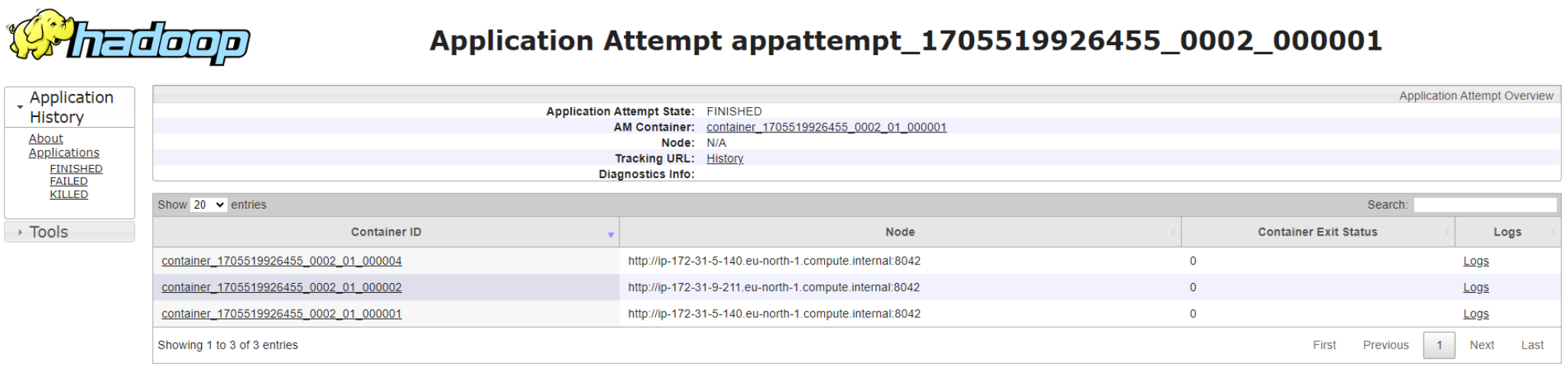


Below is a table with the following columns: Attempt ID, Started, Node, and Logs.

Only one attempt is listed:
Attempt ID: application_1705519926455_0002_000001
Started: Wed Jan 17 20:44:51 +0100 2024
Node: (a URL indicating the node's address within the cluster)
Logs: There's a link to view the logs, which would provide detailed information about the application's execution.

The application named SparkETL was submitted by the user "hadoop" and successfully completed within 21 seconds. The application ran in the default queue with the lowest priority and did not have any specific node labeling for resource allocation. The "Tracking UI" points to the "History" which means detailed historical data about the application's execution can be accessed for further analysis.

4.b.3 Application attempt within the Hadoop ecosystem

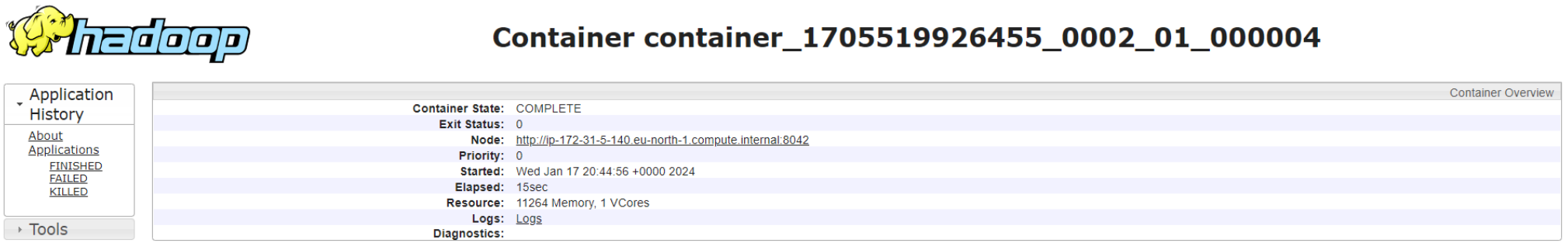


Below there is a table with the following columns:

Container ID: Three different container IDs are listed (container_1705519926455_0002_01_000001, container_1705519926455_0002_01_000002, and container_1705519926455_0002_01_000003).
Node: Lists the HTTP addresses of the nodes where these containers were located, which are internal addresses within the Hadoop cluster.
Container Exit Status: All three containers have an exit status of '0', which conventionally means that they exited without error.
Logs: Each container has a link to its respective logs, which would provide detailed runtime information for each container.

This information is typically used to monitor the status and health of specific application attempts and to debug if there were any issues during the execution of these application attempts. The successful exit status for all containers and the "FINISHED" state of the application attempt indicate that this particular run of the application was successful.

4.b.4 Details for a specific container associated with a Hadoop application



Node: The URL provided is the address where the container was running within the Hadoop cluster.
Elapsed: The time the container was running, which is 15 seconds.
Resources: The resources allocated to this container are 1,024 MB of memory and 1 vCore (virtual CPU).

This information is used to audit and troubleshoot the performance of specific containers within Hadoop applications, ensuring that they are executing as expected and utilizing the allocated resources effectively. The successful exit status and the "COMPLETE" state indicate that this container ran its task successfully.