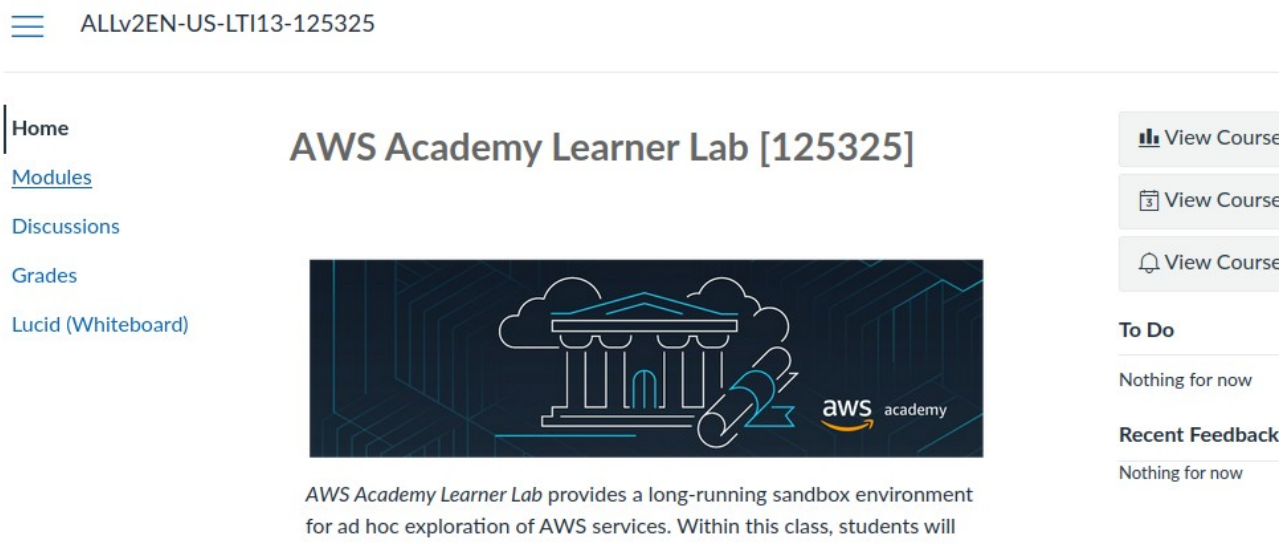
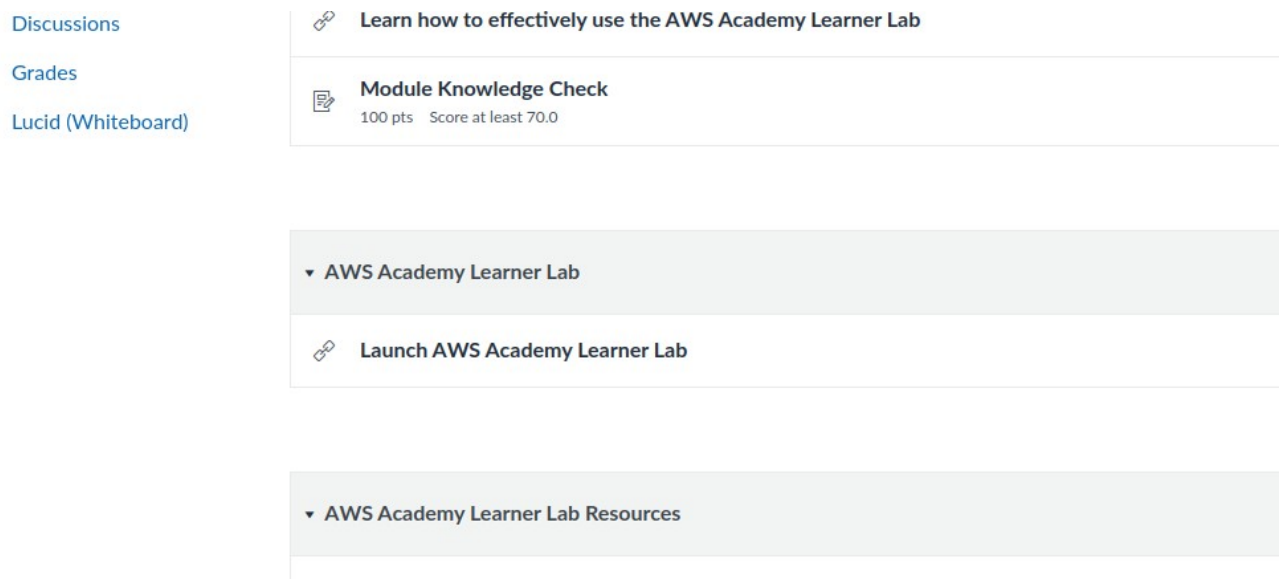
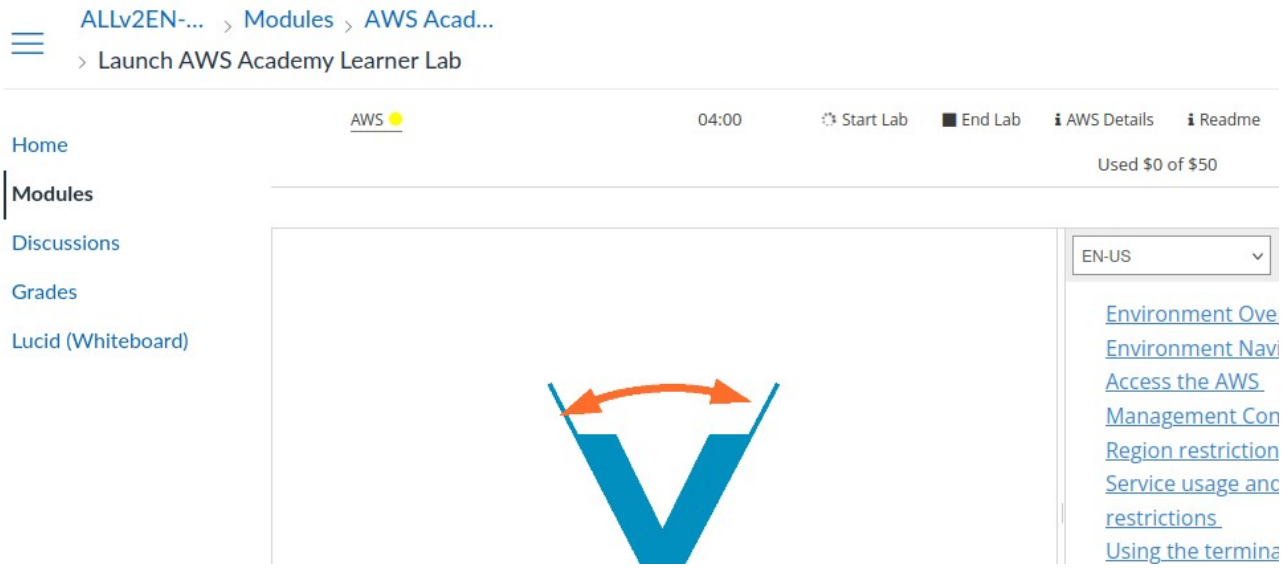
1. In the AWS Academy Learned Lab click on Modules



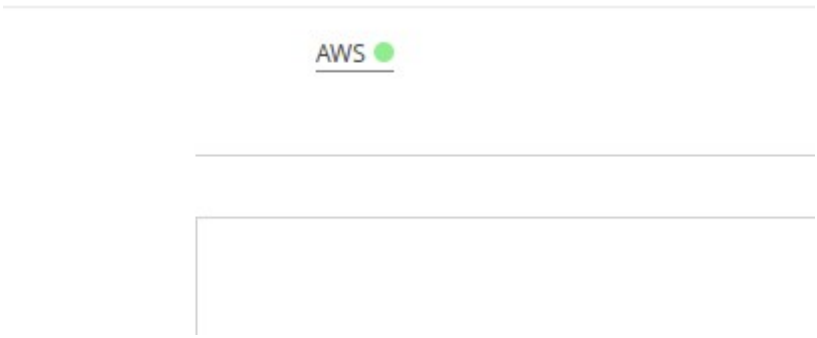
2. Scroll down and click on Launch AWS Academy Learner Lab



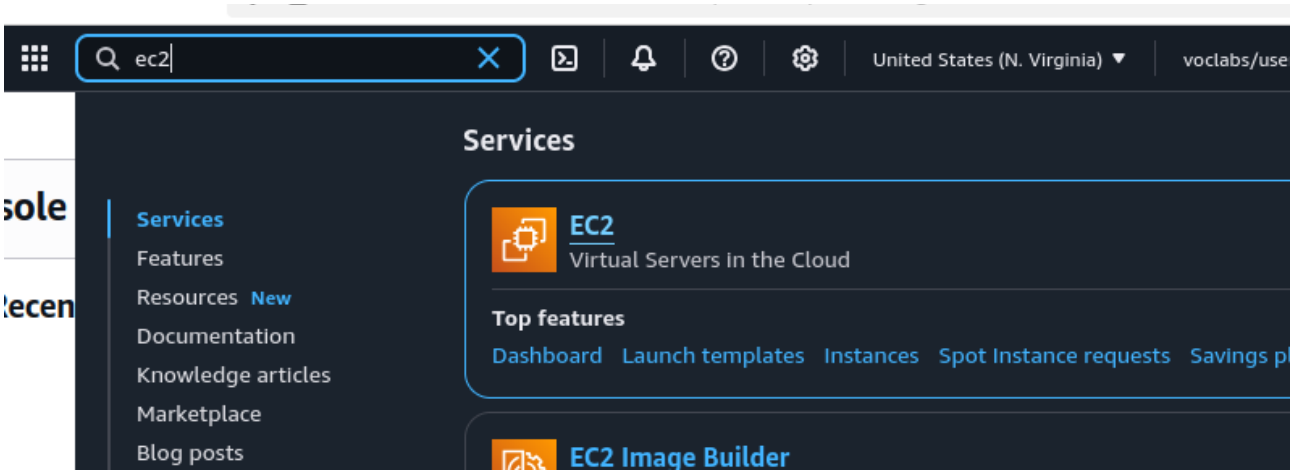
3. Click on Start Lab and wait until the circle next to AWS turns green



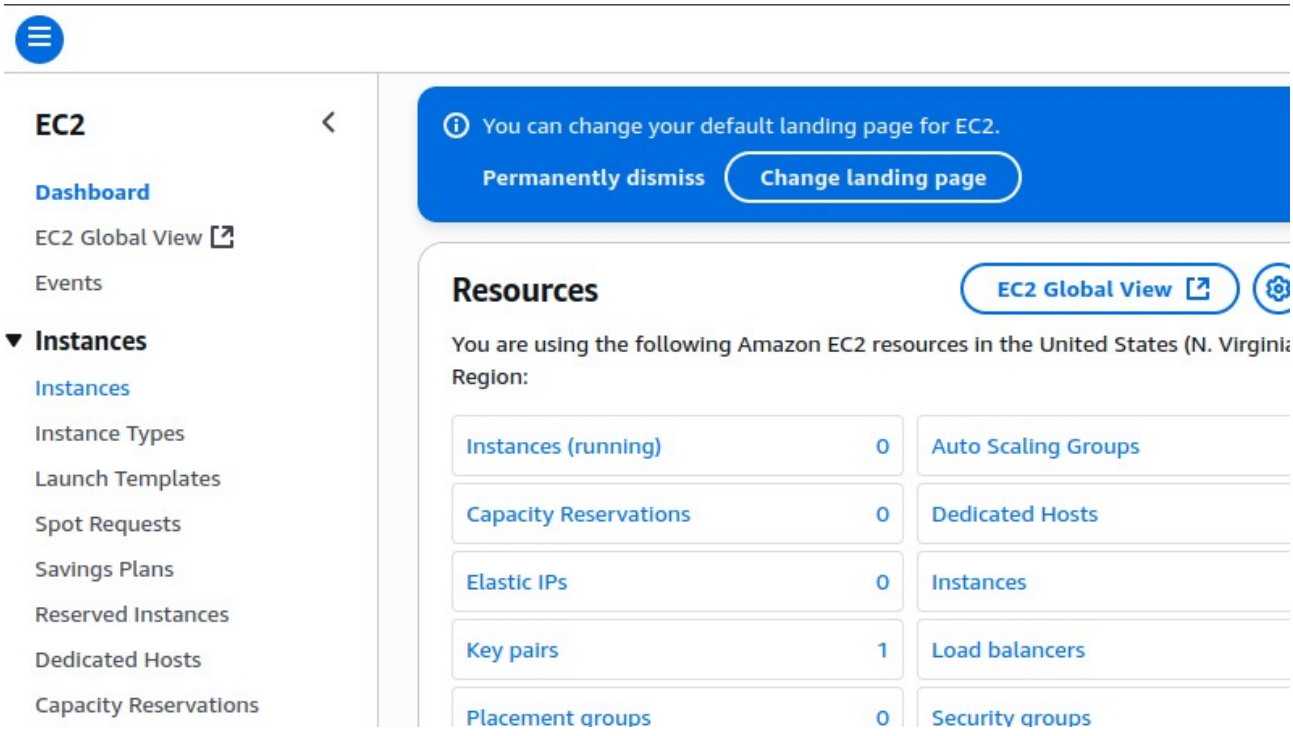
4. Click on AWS after it turns green



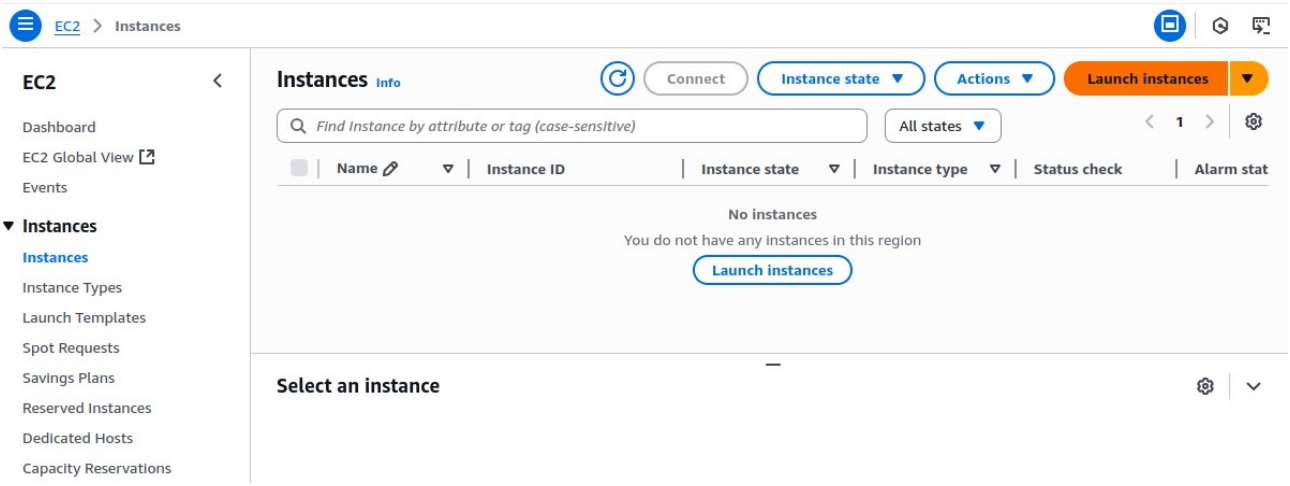
5. It will open the AWS console in the browser. In the search bar search for ec2 and click the EC2 from result tab



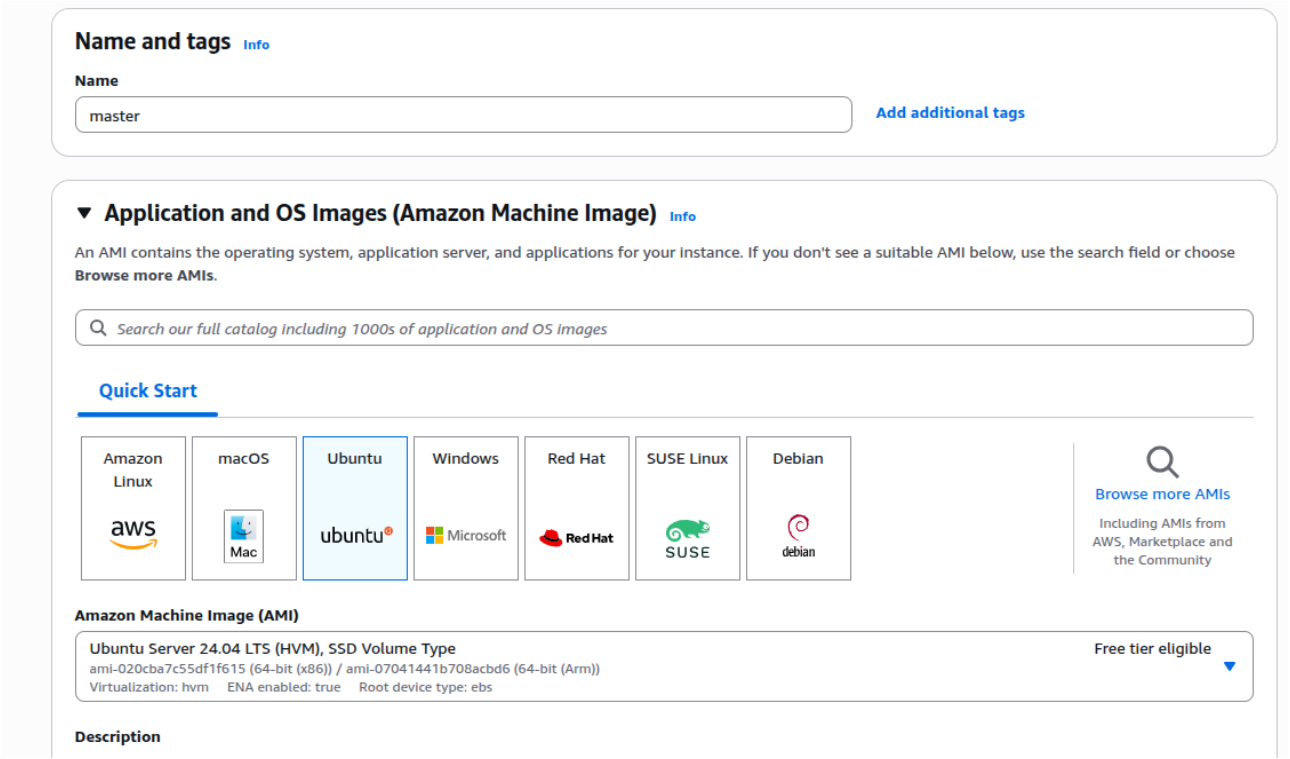
In the EC2 page in the left hand side you will see a tab for Instances. Click on the Instances link below it



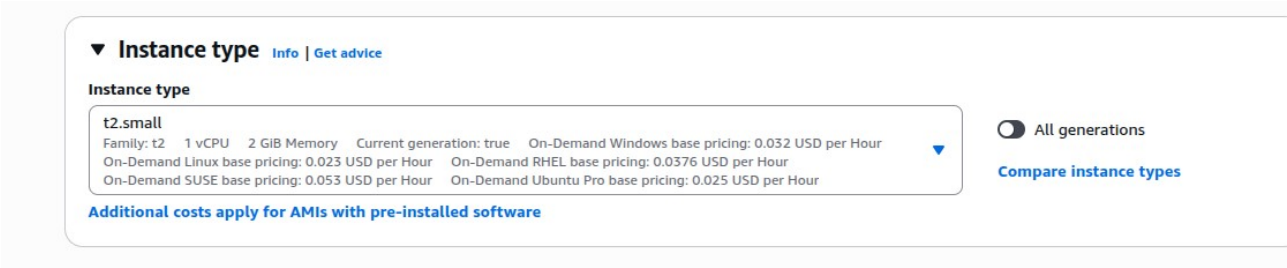
6. In the Instances page. Currently you do not have any instances created. Click the Orange Launch Instances button



7. Set the name as master to recognize it properly. Then select Ubuntu for Os Image:

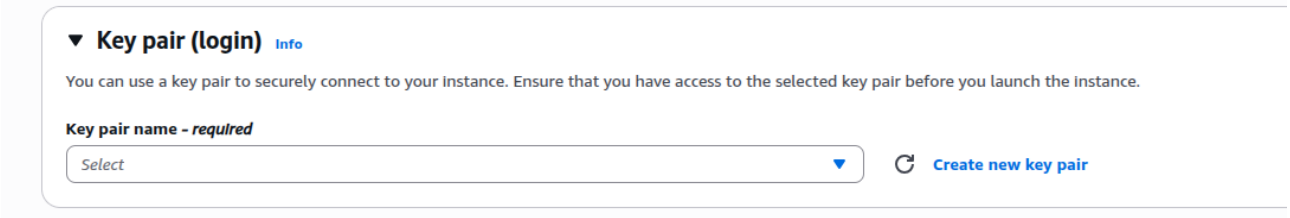


For Instance type select t2.small.



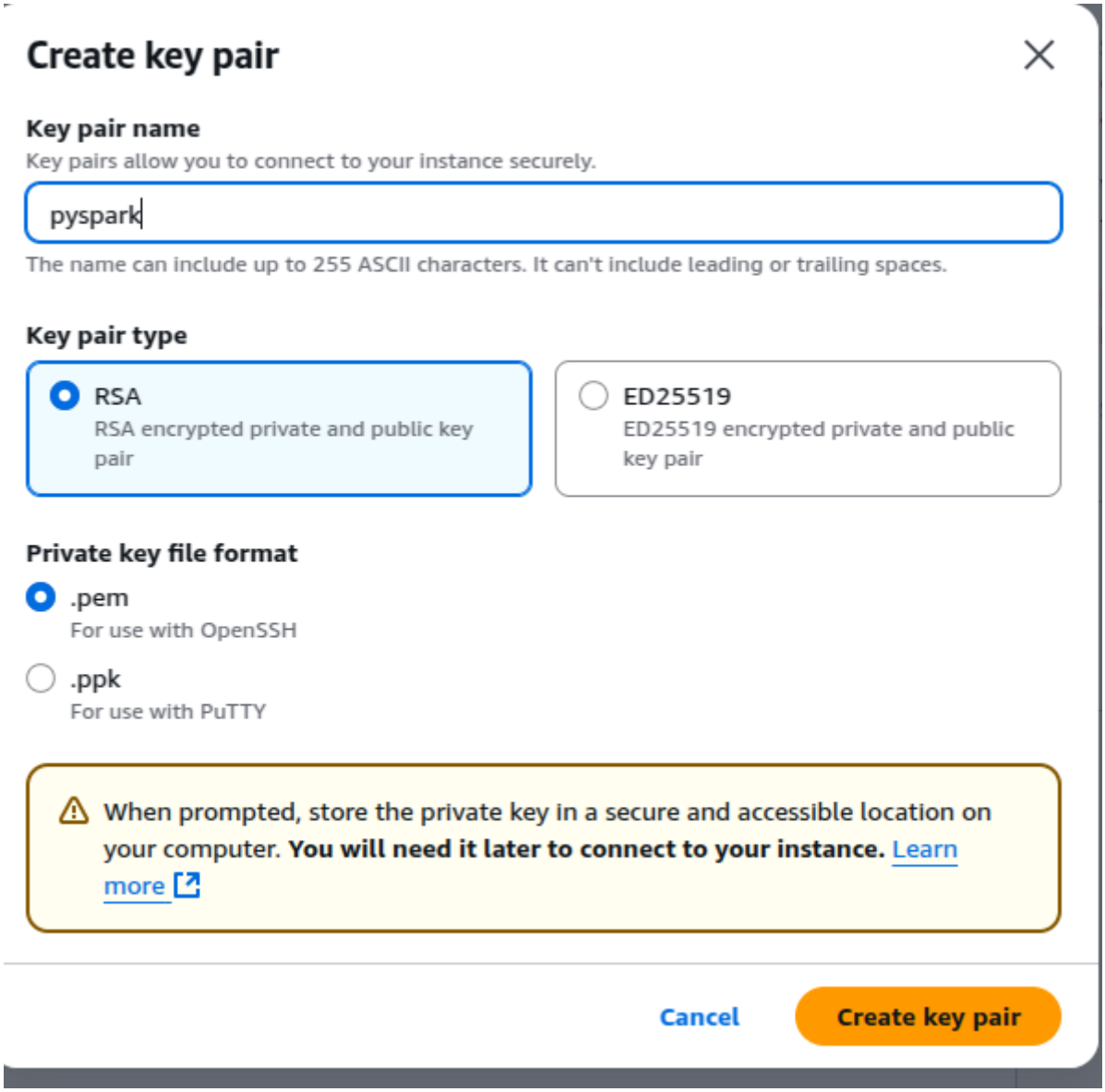
Notice that as per our need we can select instance of any size. Since this is for testing we are using small instance

Click on Create new key pair. You can also choose the default one but is good practice to create new key for different systems.

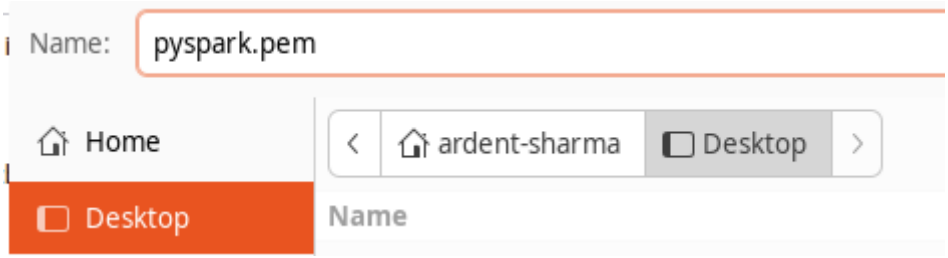


Give it a name. Select RSA for Key pair type and the format as .pem as we will be using openssh for access.

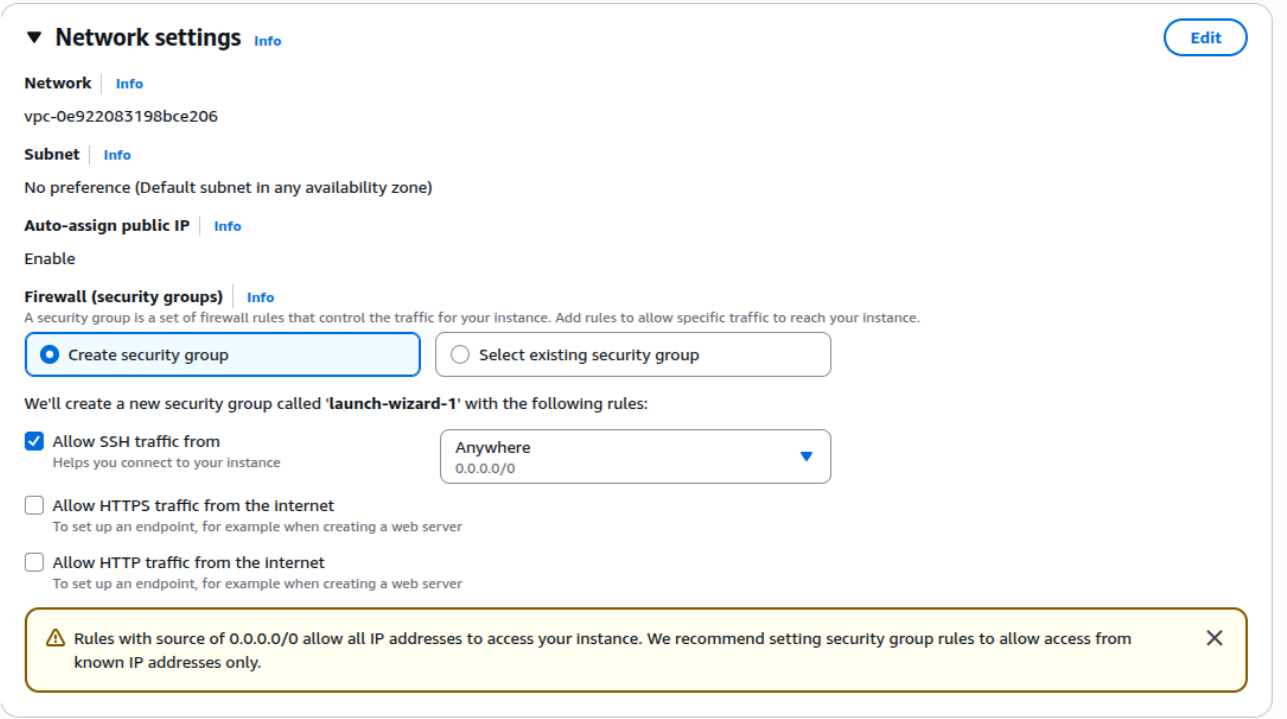
Then click on Create Key Pair



It should automatically download the .pem key. Make sure you store it in safe place. I am saving it in Desktop. **It is very essential that you download and save this key as it is what we will be using in future to access our instance.**

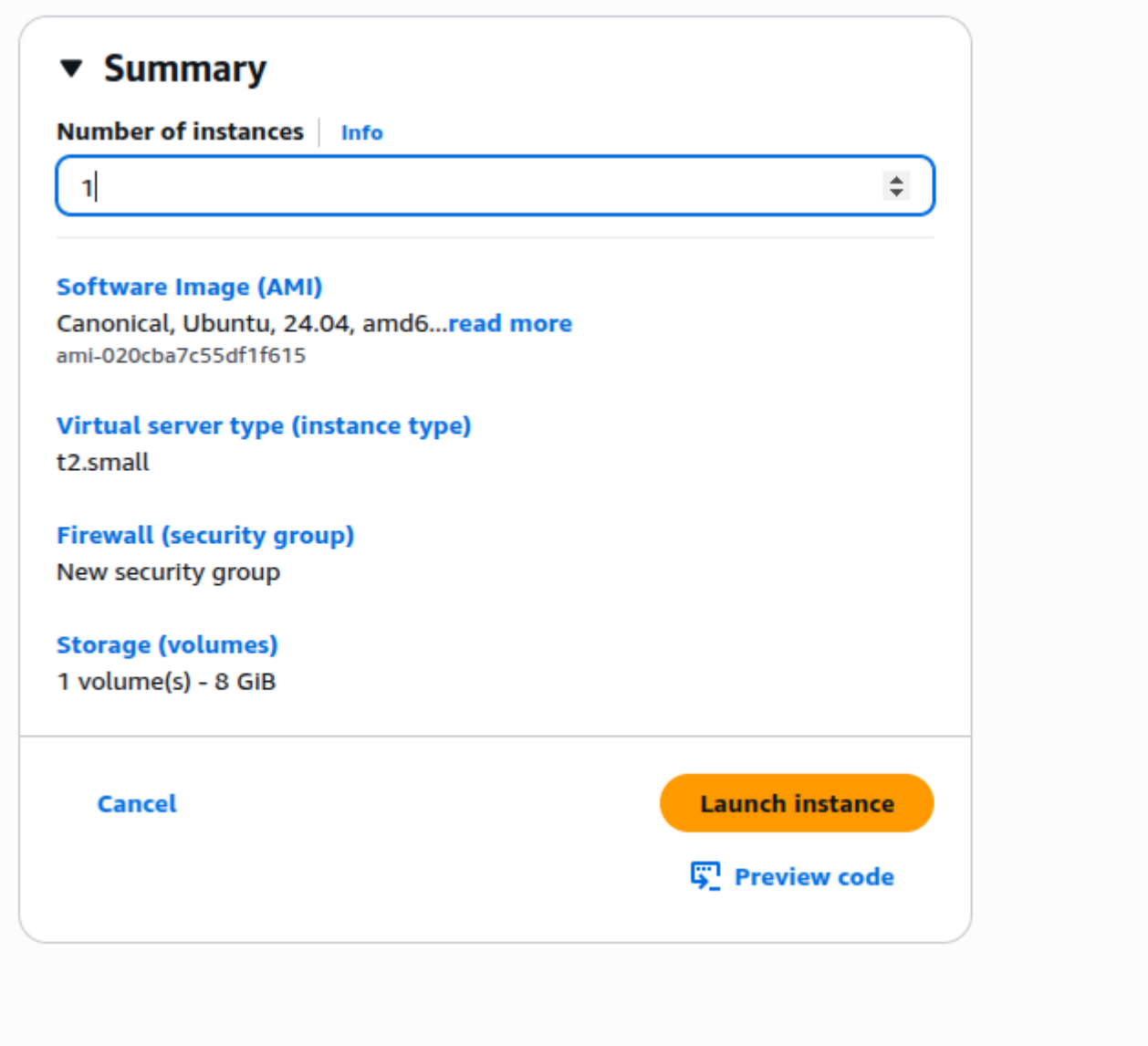


Leave the network settings to default. This is not recommended for production environment. But since we are testing and also not an network/security expert lets leave it to default. Remember the name of the security group being created by default ‘launch-wizard-1’ as well be using this for slaves instances as well.



Leave all the other settings to default as well.

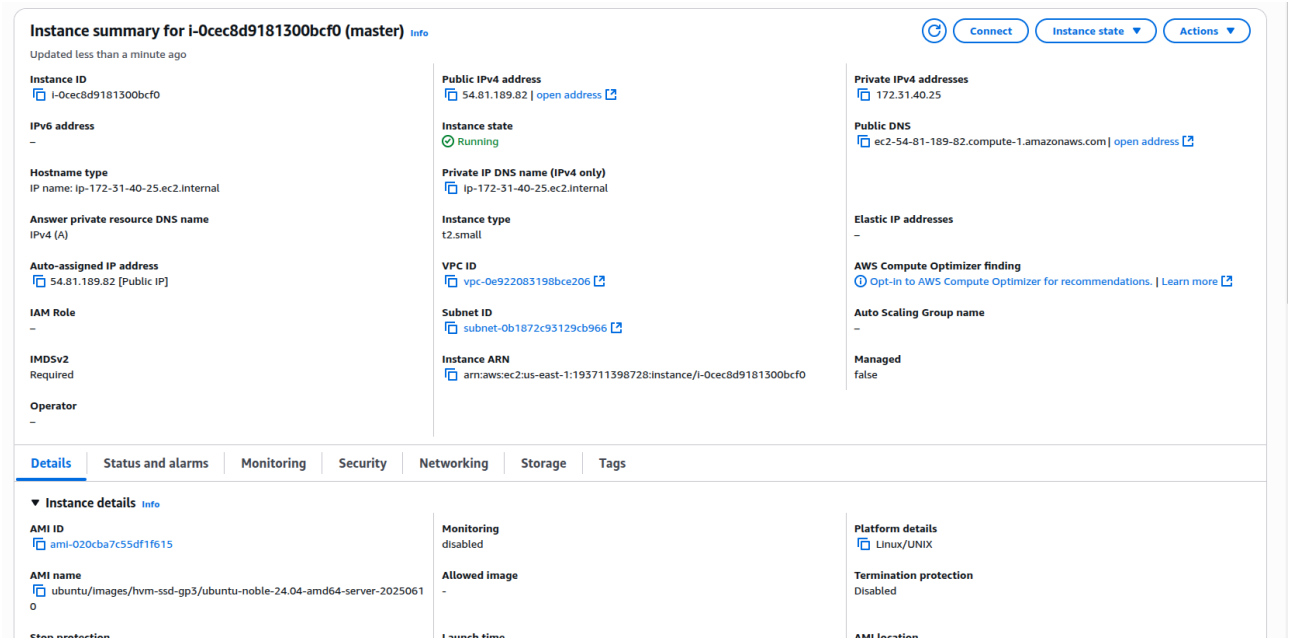
Finally, in the summary section verify everything is as expected and click Launch Instance.



You will get a success screen.

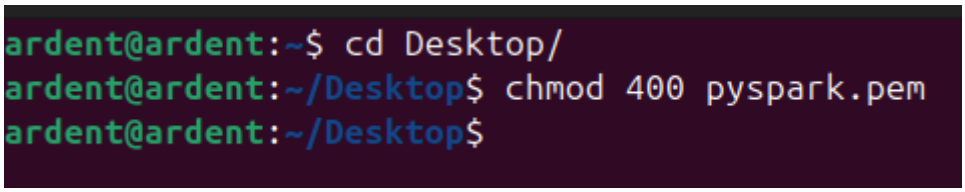
Click on the instance id starting from i-….. You will be redirected to the instance summary page. You can also access this page from previous route EC2 > Instances

Now let us connect to the newly created instance from our terminal using ssh and configure pyspark in that instance.



\

8. Open terminal. Change directory to where you downloaded the .pem file and change its permission to read-only for owner and no access for other users. This is for security purpose.



9. Connect to the instance using following command. Make sure you use your own pem file and public ipv4 address. Click yes when prompted to register the fingerprint.

ssh -i pyspark.pem ubuntu@54.81.189.82

Here,
ssh is the ssh-server tool we are using
-i is the flag for identify_file or your security file. After this file you need to pass either the absolute or relative path to the location of .pem file created for the particular instance we are trying to connect to
ubuntu is the default username provided by aws
54.81.189.82 is the Public IPv4 address of the instance

```
ardent@ardent:~/Desktop$ ssh -i pyspark.pem ubuntu@54.81.189.82
The authenticity of host '54.81.189.82 (54.81.189.82)' can't be established.
ED25519 key fingerprint is SHA256:bwa9r6liAHDGN9bjfyIF5pVskhIFJrPU0hfcuHyLzhU.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '54.81.189.82' (ED25519) to the list of known hosts.
Welcome to Ubuntu 24.04.2 LTS (GNU/Linux 6.8.0-1029-aws x86_64)
```

This should be your final output

```
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

ubuntu@ip-172-31-40-25:~$
```

This means you are accessing the terminal of the instance you just created in aws. All the commands you run from now on we will be executed on that instance. Let us setup this instance to run pyspark as a master/driver

```
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

ubuntu@ip-172-31-40-25:~$ sudo apt update
Hit:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble InRelease
Get:2 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates InRelease [12
6 kB]
Get:3 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-backports InRelease [
```

Run the first two command we always run on a linux system after setup.

1. sudo apt update
2. sudo apt upgrade Press Y when prompted


```
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

ubuntu@ip-172-31-40-25:~$ sudo apt update
Hit:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble InRelease
Get:2 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates InRelease [12
6 kB]
Get:3 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-backports InRelease [
```

```
Reading state information... Done
98 packages can be upgraded. Run 'apt list --upgradable' to see them.
ubuntu@ip-172-31-40-25:~$ sudo apt upgrade
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
Calculating upgrade... Done
The following NEW packages will be installed:
  linux-aws-6.14-headers-6.14.0-1010 linux-aws-6.14-tools-6.14.0-1010
```

Download spark 3.3.1 from apache archive directory using command line tool wget or curl. I am using wget.

```
 ubuntu @ session #2: apt[1559], sshd[1059,1185]
 ubuntu @ user manager service: systemd[1078]
ubuntu@ip-172-31-40-25:~$ wget https://archive.apache.org/dist/spark/spark-3.3.1
/spark-3.3.1-bin-hadoop3.tgz
--2025-08-02 05:39:35--  https://archive.apache.org/dist/spark/spark-3.3.1/spark
-3.3.1-bin-hadoop3.tgz
Resolving archive.apache.org (archive.apache.org)... 65.108.204.189, 2a01:4f9:1a
:a084::2
Connecting to archive.apache.org (archive.apache.org)|65.108.204.189|:443... con
nected.
HTTP request sent, awaiting response... 200 OK
Length: 299350810 (285M) [application/x-gzip]
Saving to: ‘spark-3.3.1-bin-hadoop3.tgz’

n-hadoop3.tgz         5%[>                    ]  16.94M   340KB/s    eta 13m 16s
```

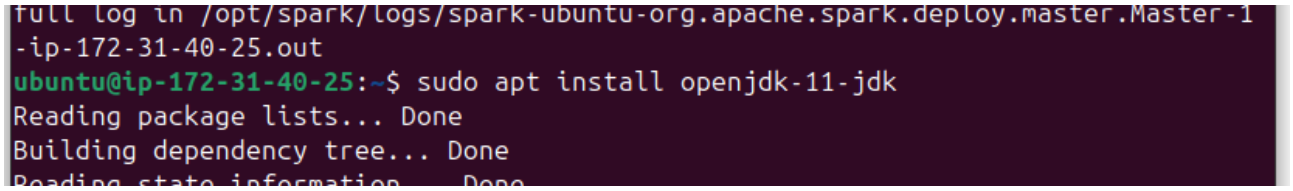
wget https://archive.apache.org/dist/spark/spark-3.3.1/spark-3.3.1-bin-hadoop3.tgz

Wait for a while as it might take some time depending on the internet connection. After the download has completed extract the compressed files using following command

```
ubuntu@ip-172-31-40-25:~$ ls
spark-3.3.1-bin-hadoop3.tgz
ubuntu@ip-172-31-40-25:~$ tar -xvzf spark-3.3.1-bin-hadoop3.tgz
spark-3.3.1-bin-hadoop3/
spark-3.3.1-bin-hadoop3/LICENSE
```

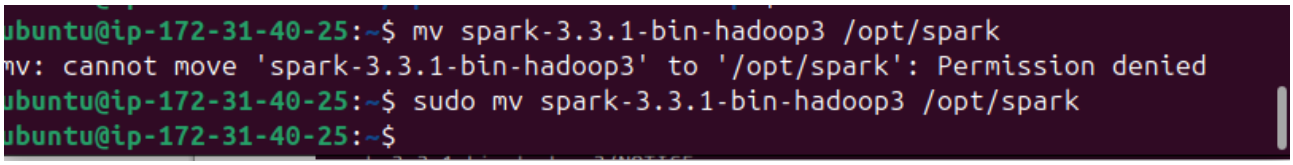
tar -xvzf spark-3.3.1-bin-hadoop3.tgz

Let us install java as spark requires java. We will be using the below command to install. Notice how the steps are similar to what we did to setup our own system and to setup a new server instance.



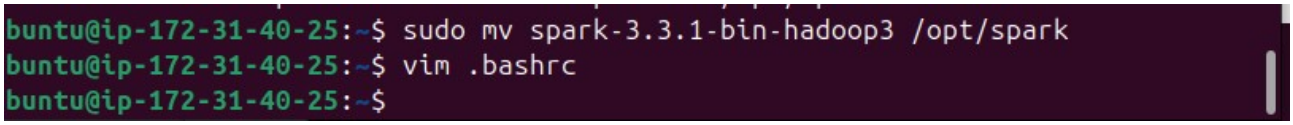
sudo apt install openjdk-11-jdk

Now, Let us move the extracted directory to /opt/spark . This is the standard directory for spark. Use the below command. Make sure you use sudo before mv since only root user is allowed access to root directories



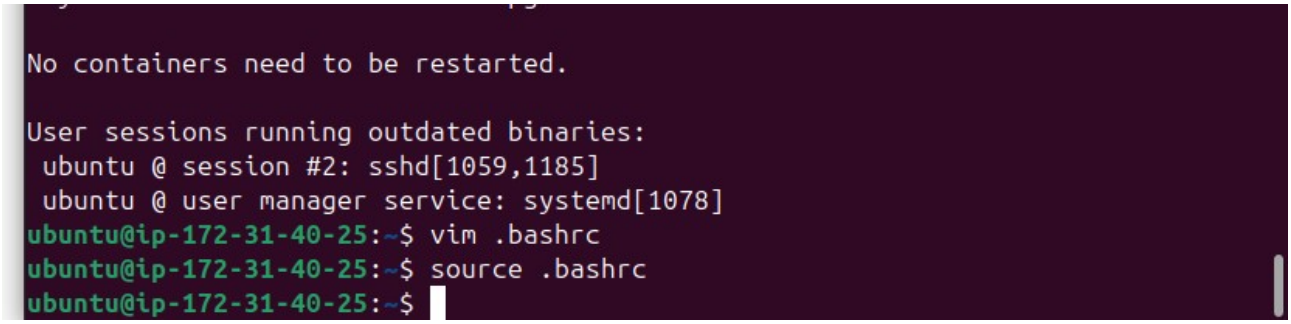
sudo mv spark-3.3.1-bin-hadoop3 /opt/spark

Now let us update the bashrc file to provide information regarding spark directory. Open the .bashrc file using either nano or vim. I will be using vim. Whether you use nano or vim. Make sure you know how to make changes and how to save and exit properly



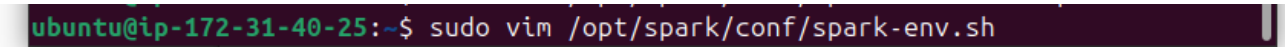
export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64
export SPARK_HOME=/opt/spark
export PATH=$PATH:$SPARK_HOME/bin:$SPARK_HOME/sbin
export PYSPARK_PYTHON=/usr/bin/python3

Since the terminal does not know that the .bashrc has been modified we need to apply/source the changes ourselves using below command. This should not be new to you anymore. If it is spend more time exploring the terminal in your linux system.



source .bashrc

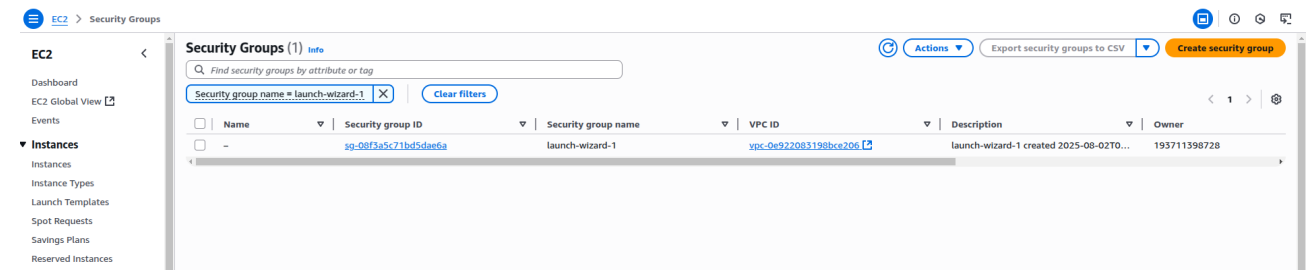
Let us modify the spark-env file to provide ip of the master. Use the below command to create and open the file in vim. As before you can use any other terminal based text editors. Make sure you use sudo to open the file. Also if you cd into the conf directory you will notice that there is a template file for spark-env which you can look at your own leisure to get idea on what kind of configuration can be done.



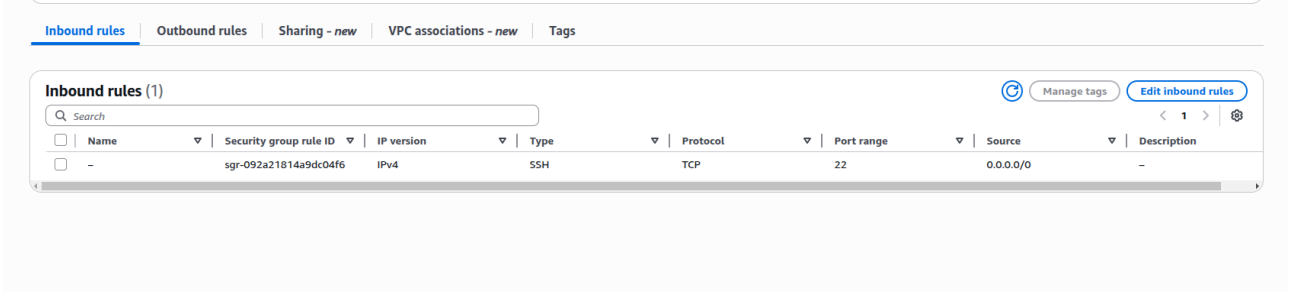
Add the private ip of the master instance. And assign it to SPARK_MASTER_HOST. Make sure you use the same name and there is no spaces before and after the assignment operator. The syntax is same for all bash files.

Before we run our spark master server. We need to make one small change we overlook initially. In order to view whether the master is running or not from other system. We need to allow http/https (techincally only http should suffice but lets give all access available) access to that system. If you remember initially we only provided ssh accessing

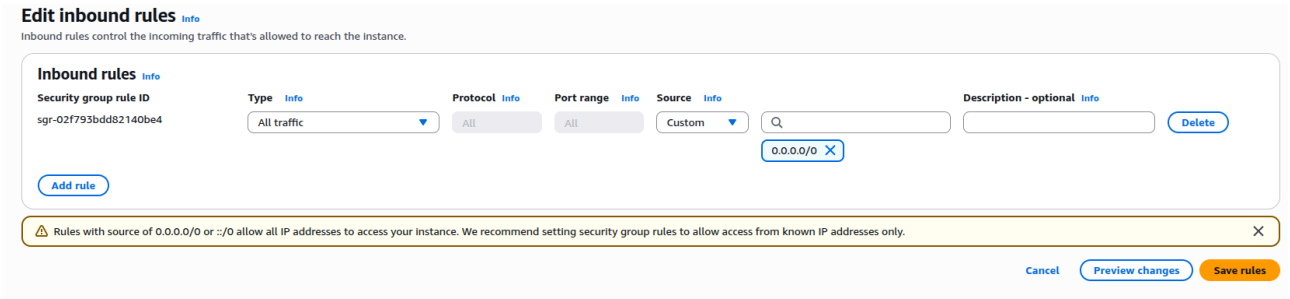
Go to EC2> Security Groups and open the launch-wizard-1 security group by clicking the link under security group id



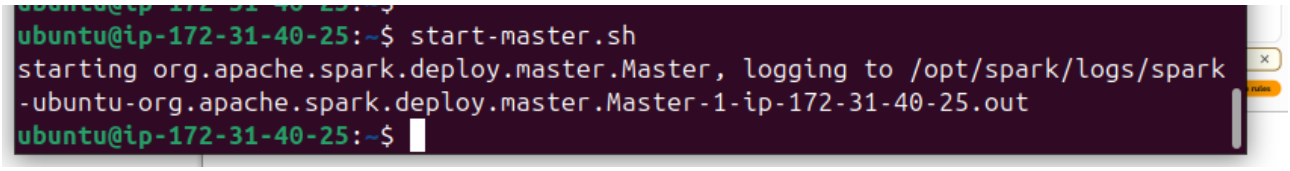
Click on Edit inbound rules



Remove the existing rule and add new rule For All Traffics and range 0.0.0.0/0. This essentially means any type access from anywhere. This setup is security hazard waiting to happen and is not recommended at all. We are only doing it for ease.



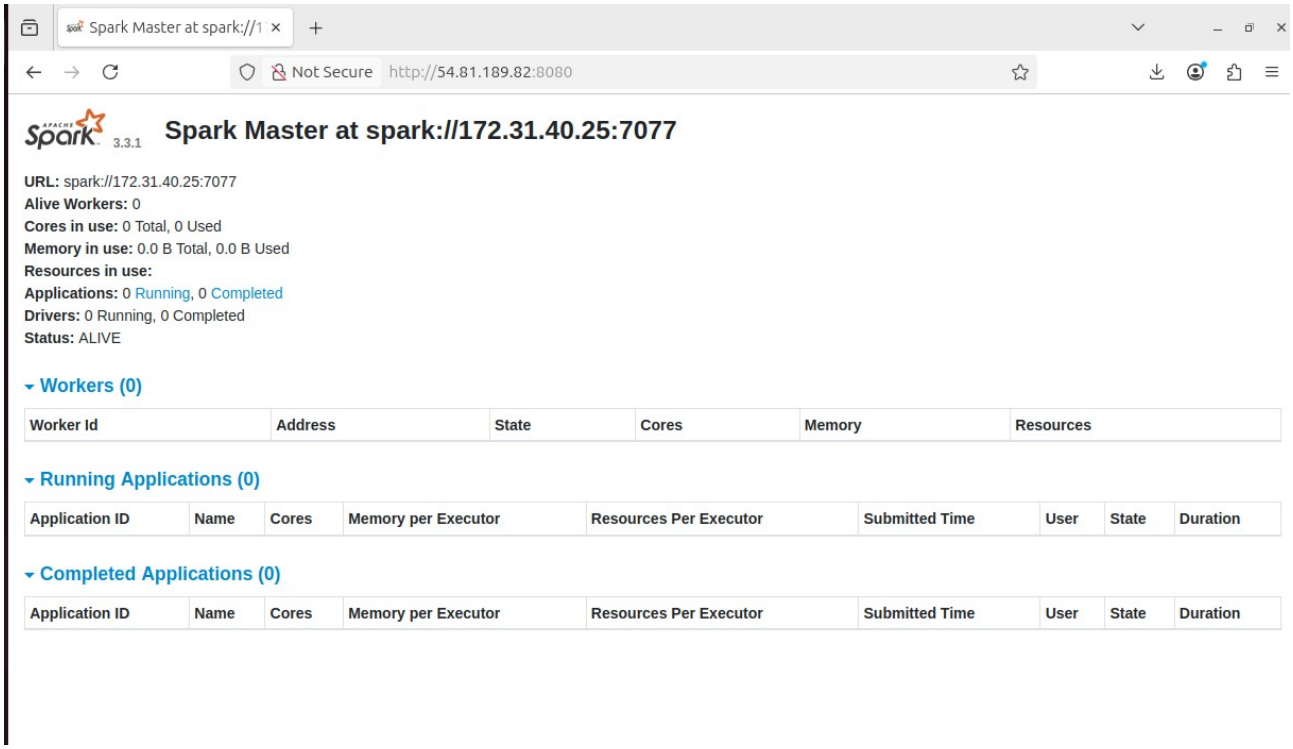
It is finally time to start the master-server. Use the below command:



start-master.sh

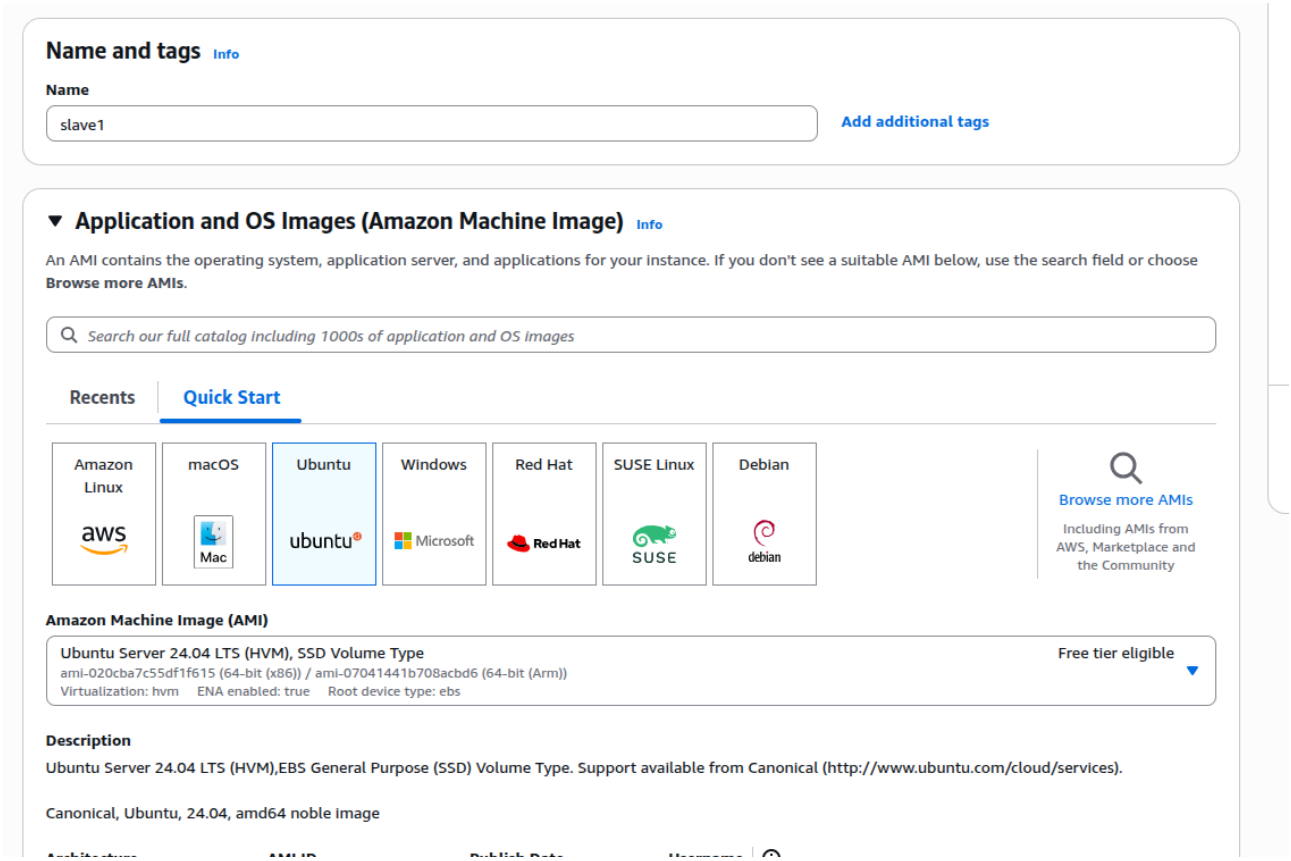
If you check this script is available in /opt/spark/sbin which we have setup in our .bashrc file's PATH variable hence we can run it from anywhere.

Now if you go to the browser and use your public_ip:8080 you should be able to see the below view. Also notice that the actual spark master is running at private_ip:7077 . Port 8080 provides the webui

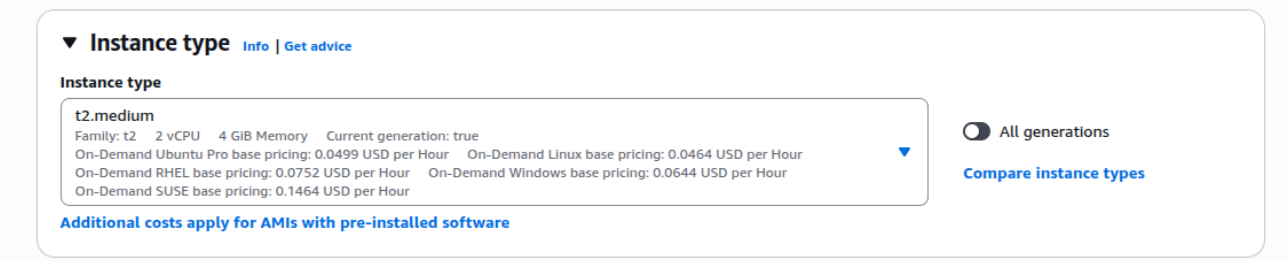


Now, it is time to spin up two new instances that will be the slave/executor. The initial setup is identical to that of master. Everything is exactly the same. So you should be able to do it. Just make sure you provide relevant name to them such as slave1 and slave2. Use the same key we generate earlier and the same security group we created and modify earlier for ease of use.

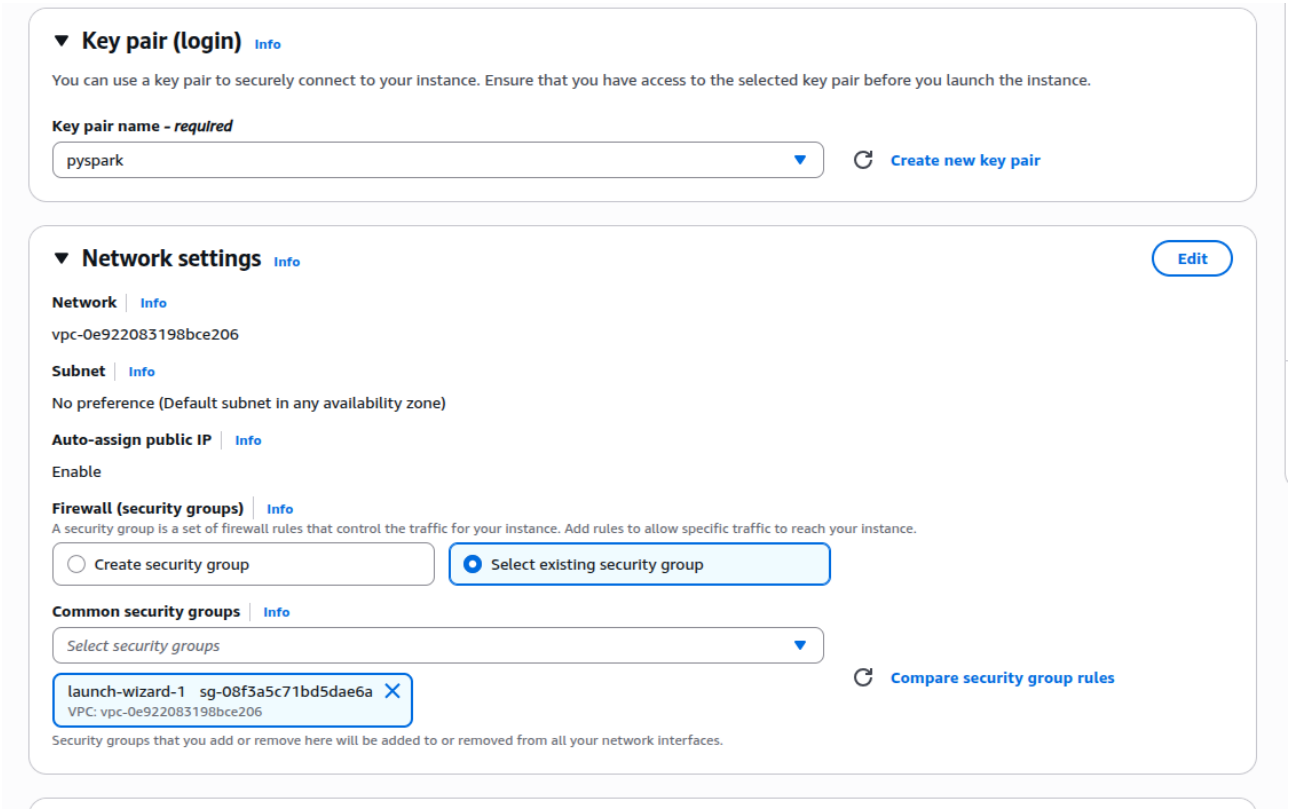
Since the setup is same I will just provide images for reference. If you need detail go above.



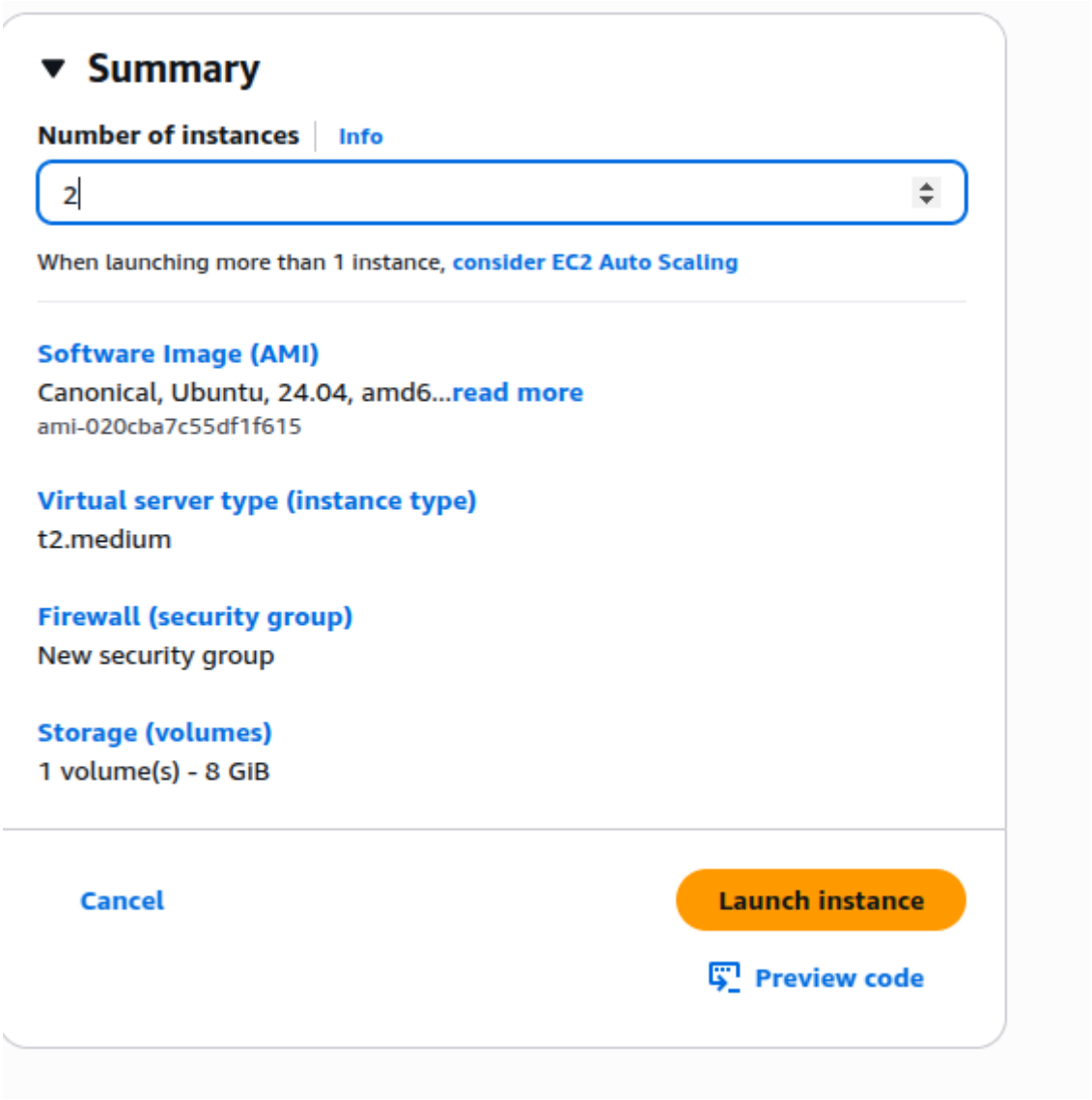
For instance type select t2.medium. Since these our executor we are choosing slightly better instance. There are several instance to choose from in aws. Some are memory optimized, some are compute optimized, some general purpose and so on. Note that we are not considering any of that currently. Since pyspark is memory heavy. A memory optimized instance for executor/slave would be a better choice. Driver/master , however, could benefit for better compute and so on.



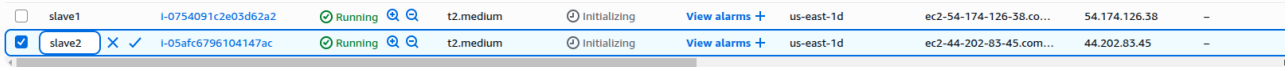
Notice we selected previous created key and existing network security group



For ease of use in the summary section select number of instance to 2 and click launch instnace



You will be redirected to success page. Go to EC2>Instances page. You will see the two newly created instances. Change the name of one of them from slave1 to slave2 for easy recognition



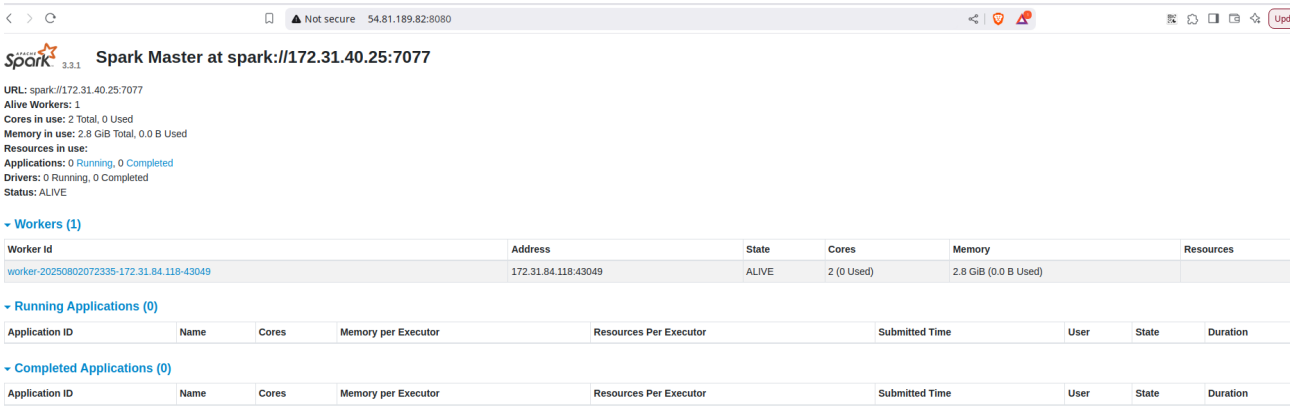
Open two new terminal in your system. Go to directory with .pem file, connect to two new instance using ssh and configure them. I won't be going in detail as initial configuration part is same. The configuration is exactly the same except for the executing part. Master will be started using command start-master.sh and Slave will be started using start-worker.sh master_ip

After following exactly the same process except for the security part as we are using the same group which has already been configured for ease. We will run the following command:

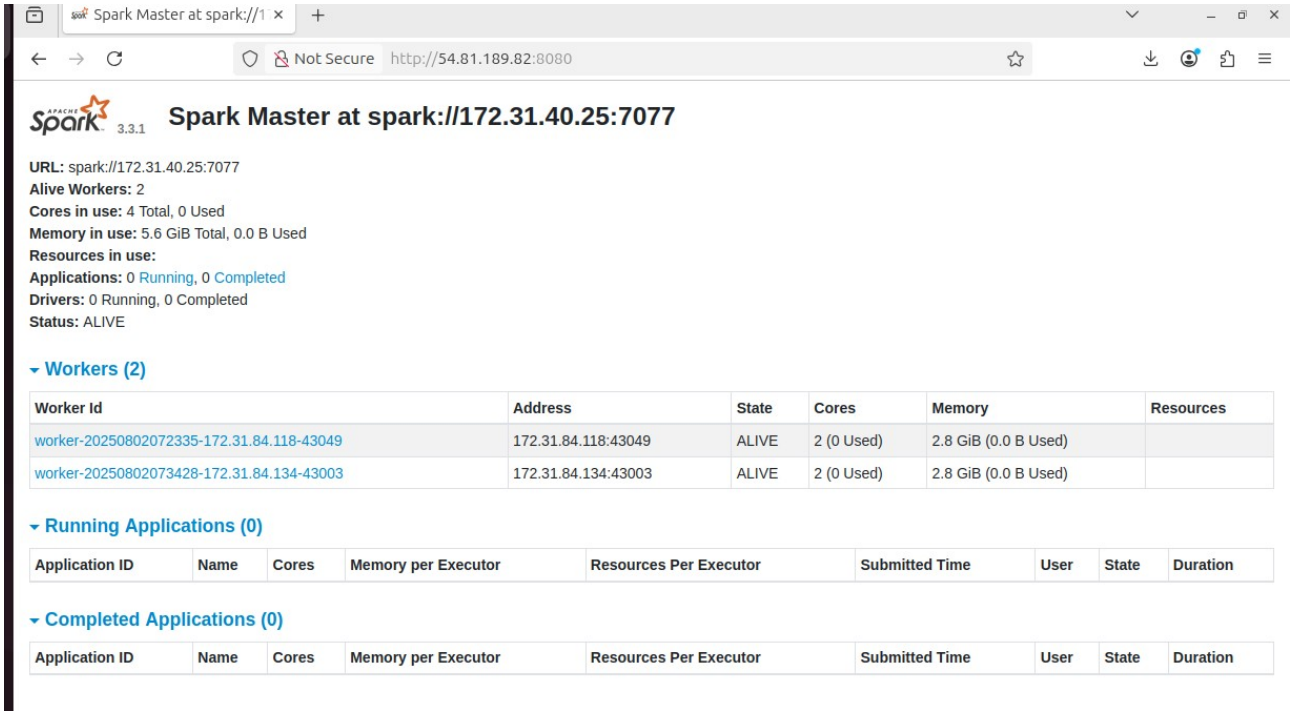
```
ubuntu@ip-172-31-84-118:~$ start-worker.sh spark://172.31.40.25:7077
starting org.apache.spark.deploy.worker.Worker, logging to /opt/spark/logs/spark-ubuntu-org.apache.spark.deploy.worker.Worker-1-ip-172-31-84-118.out
ubuntu@ip-172-31-84-118:~$
```

start-worker.sh spark://<master node IP address>:7077

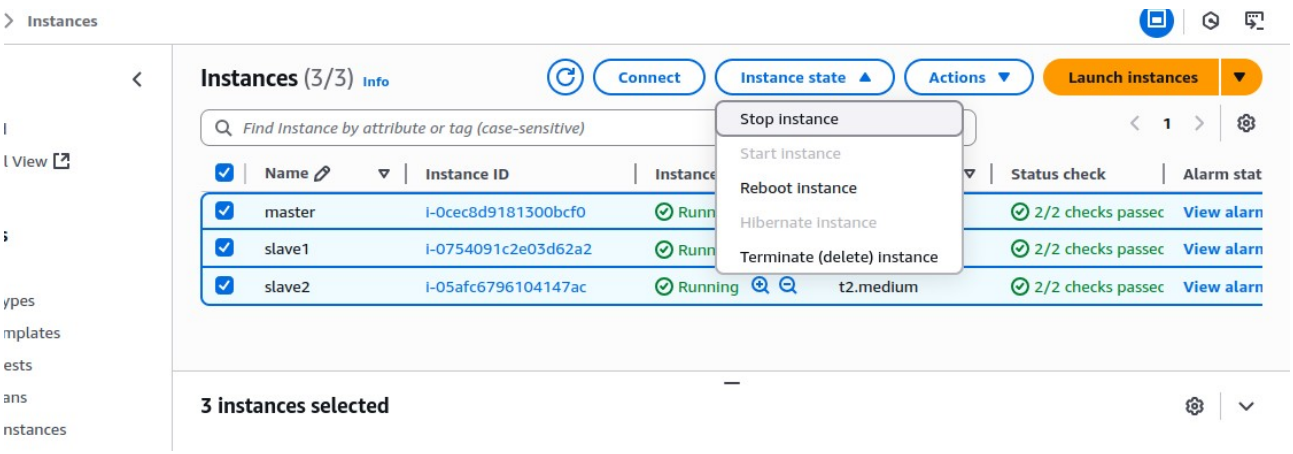
Now, if you go to the web ui and refresh you should be able to see worker added in Workers Section



Configure the next worker/slave/executor as well and then refresh the web ui and it should be reflected as well.

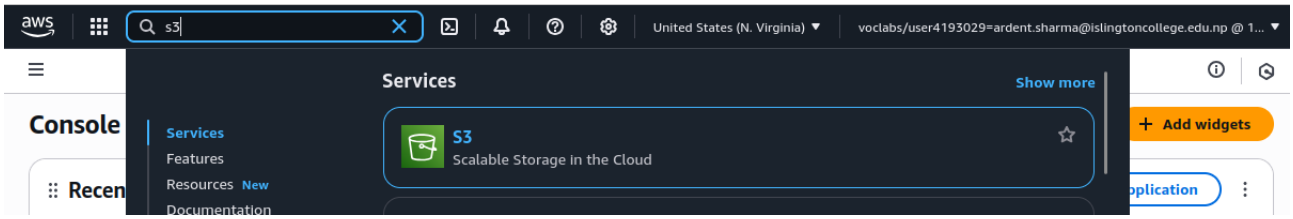


We finally have our cluster setup. Our master is running and connected to two slave nodes. Unless you are doing below steps parallel to ec2 instances cluster setup. It is wise to first stop the instances so that the credit does not accumulate

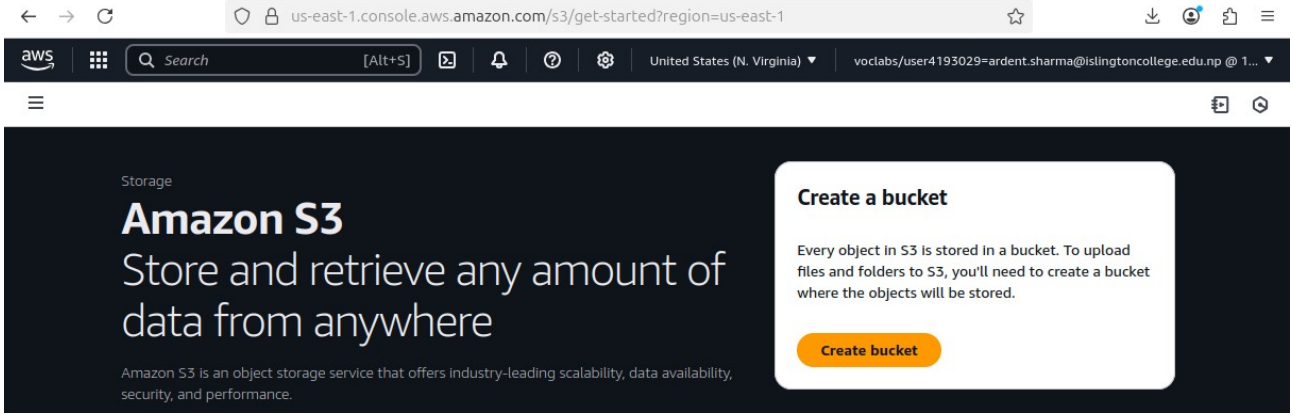


Let us setup our data storage in AWS. We will be using S3 for this task. We will only be running the transform and load section of the code in our cluster. So let us create a bucket in s3 and store the output of our extract stage their. I encourage students to later implement the extract section as well by themselves

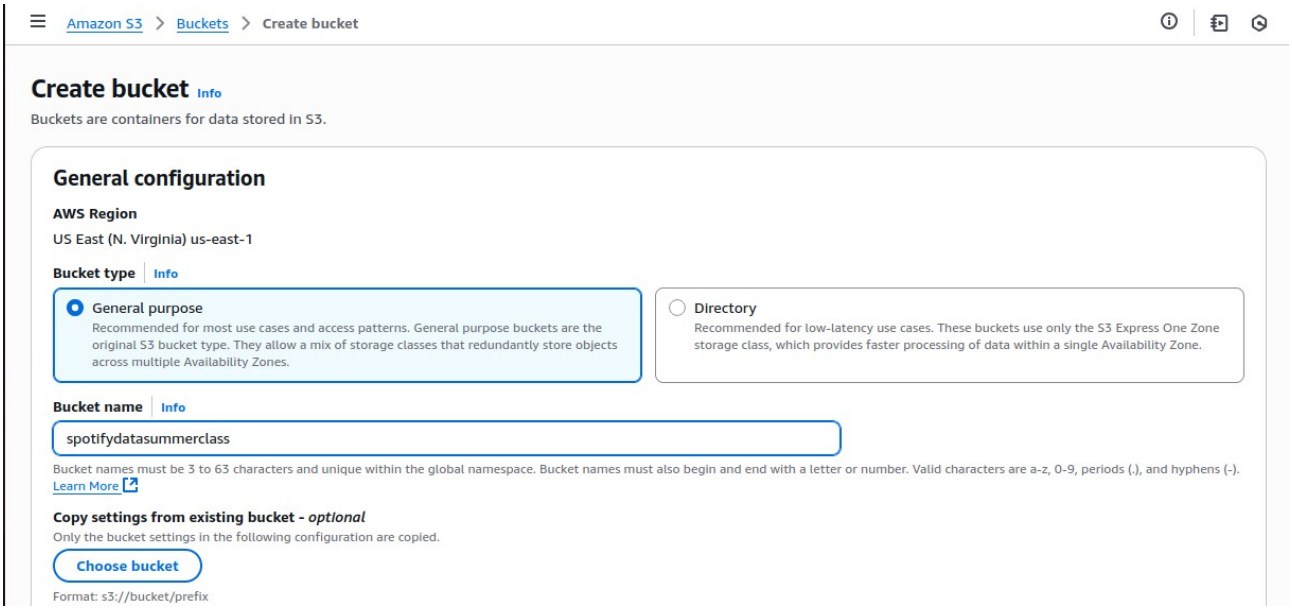
10. Go to search bar and search S3. Click the result



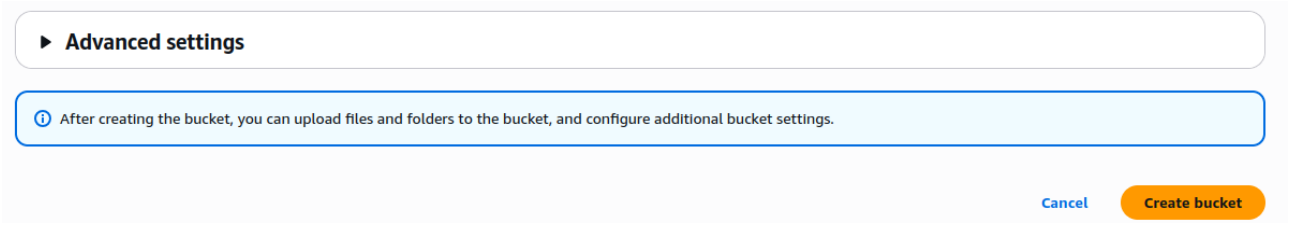
In the Amazon S3 home page click Create bucket.



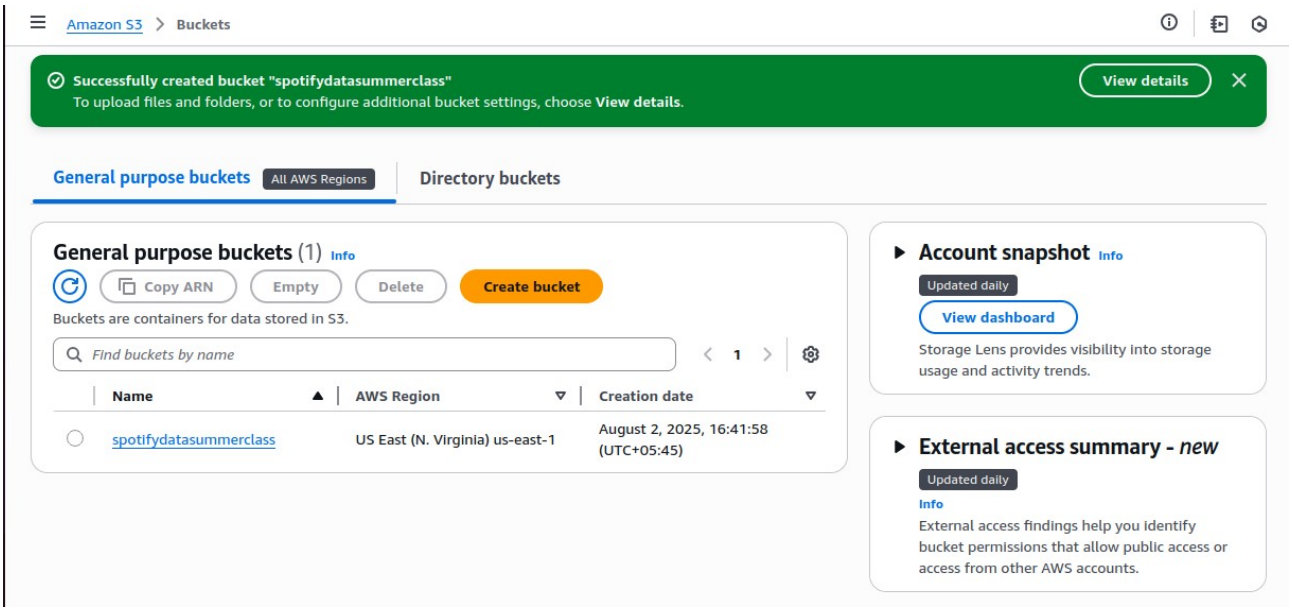
Give it a name. Make sure the name is unique. It needs to be unique across all users in a particular region. Use your name or something unique. Remember the name you give it will be important to access the files later



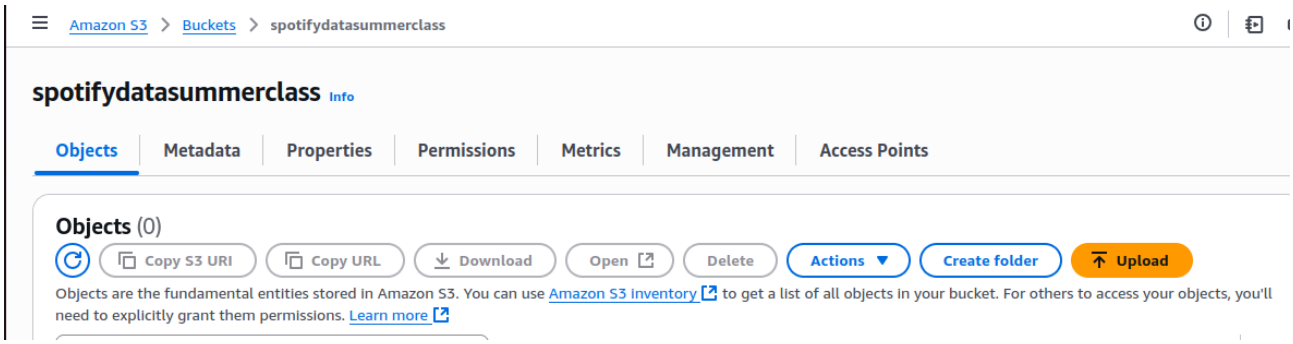
Scroll to the bottom. Keep everything else as default. Click Create Bucket



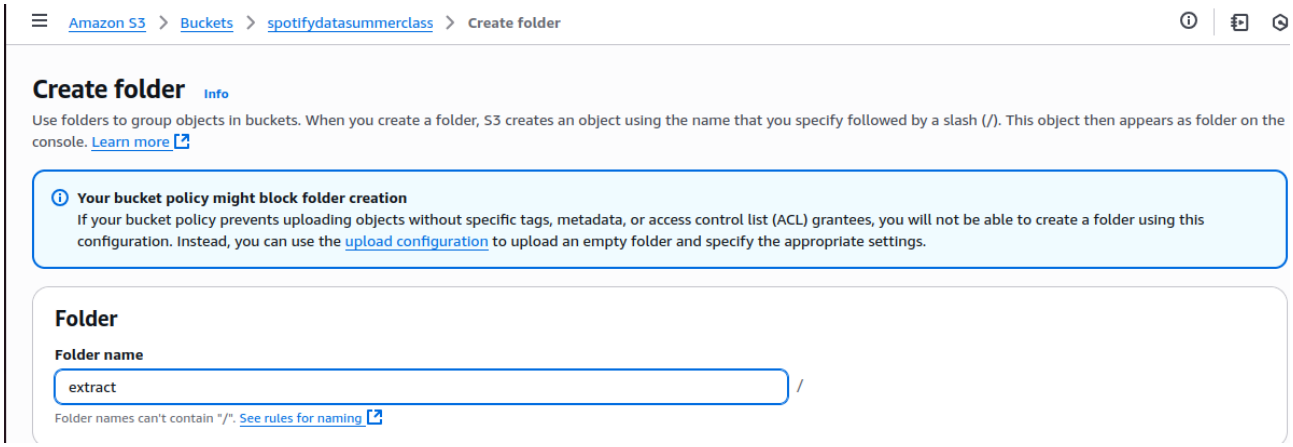
Click on the bucket name to open the bucket



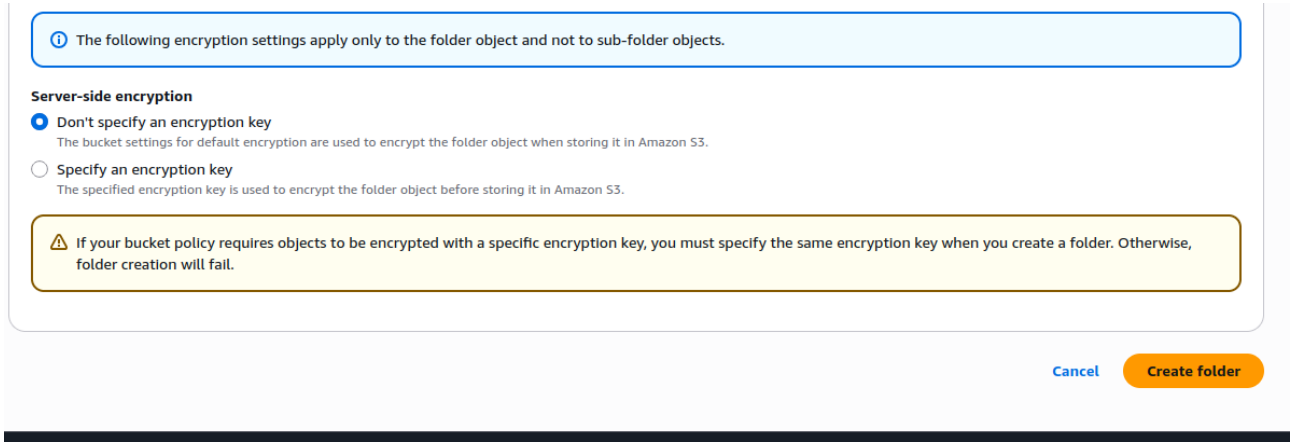
Click on Create Folder to create new folder inside the bucket



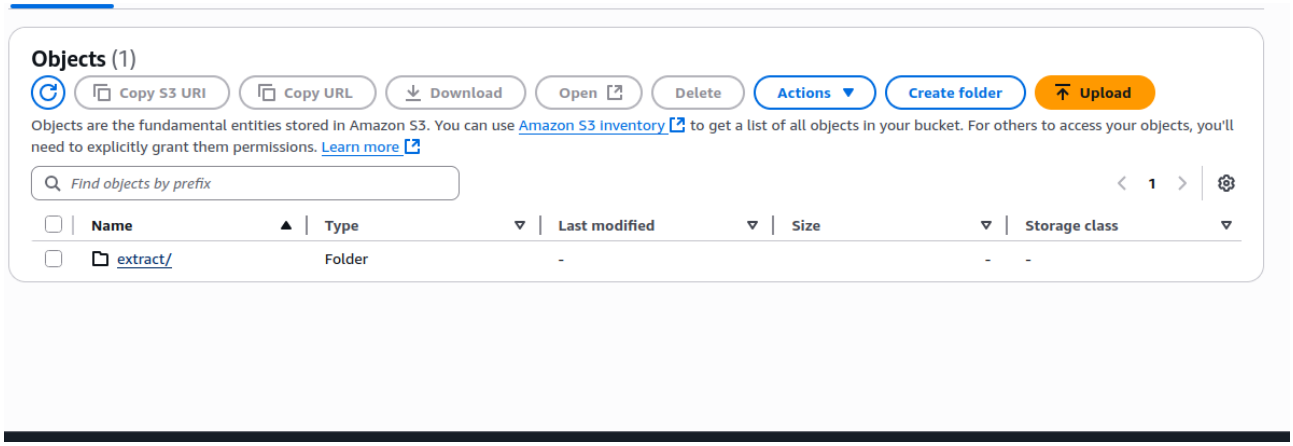
Give name to bucket as extract. This is where we will upload the output of our extract



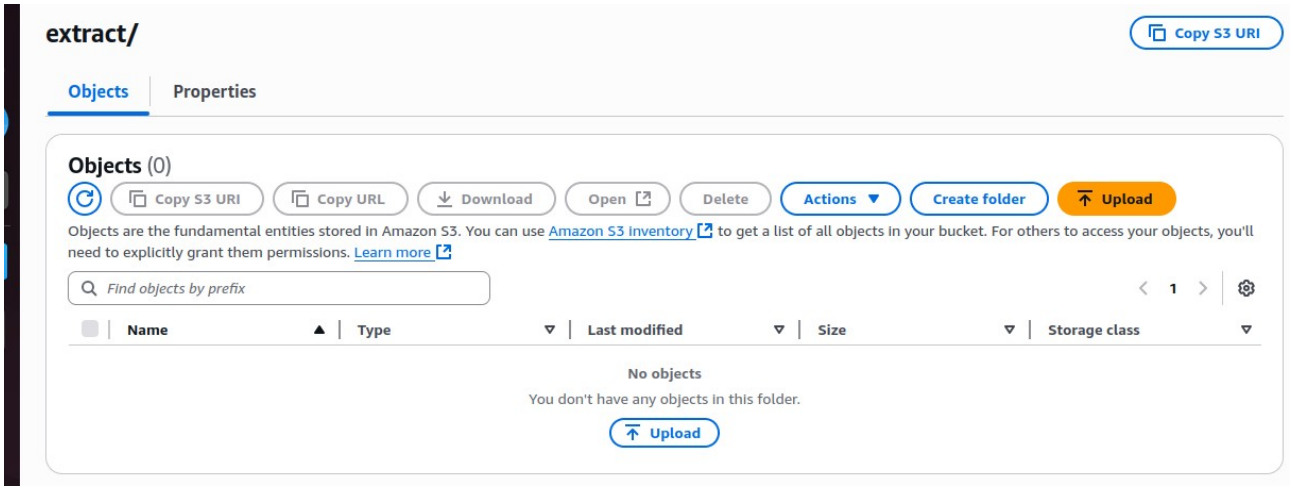
Leave everything else to default and click on Create Folder



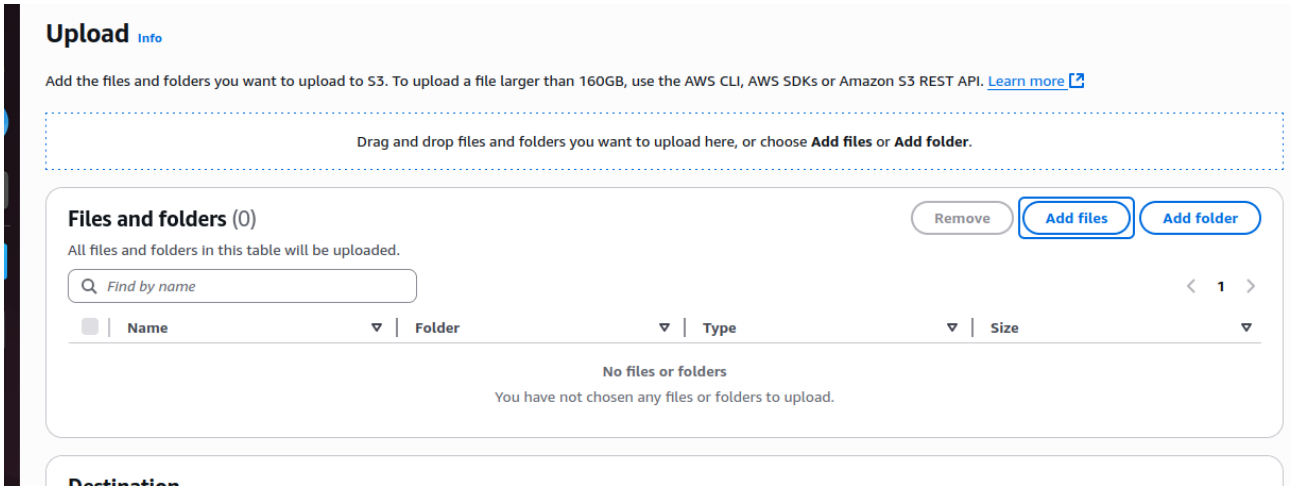
Open the newly created Folder by clicking on it



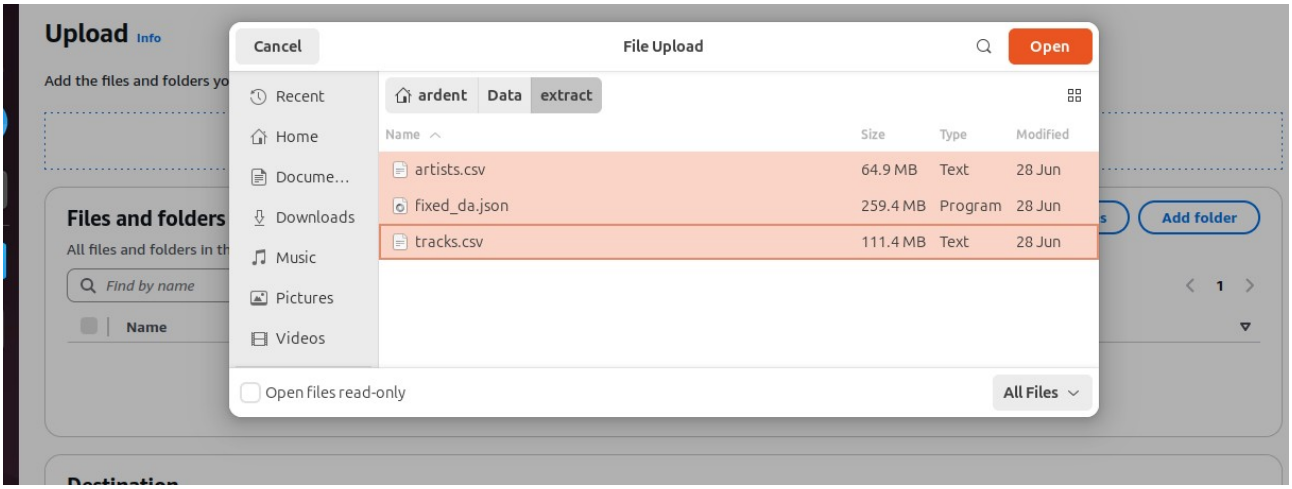
Click on the Upload button to upload our files



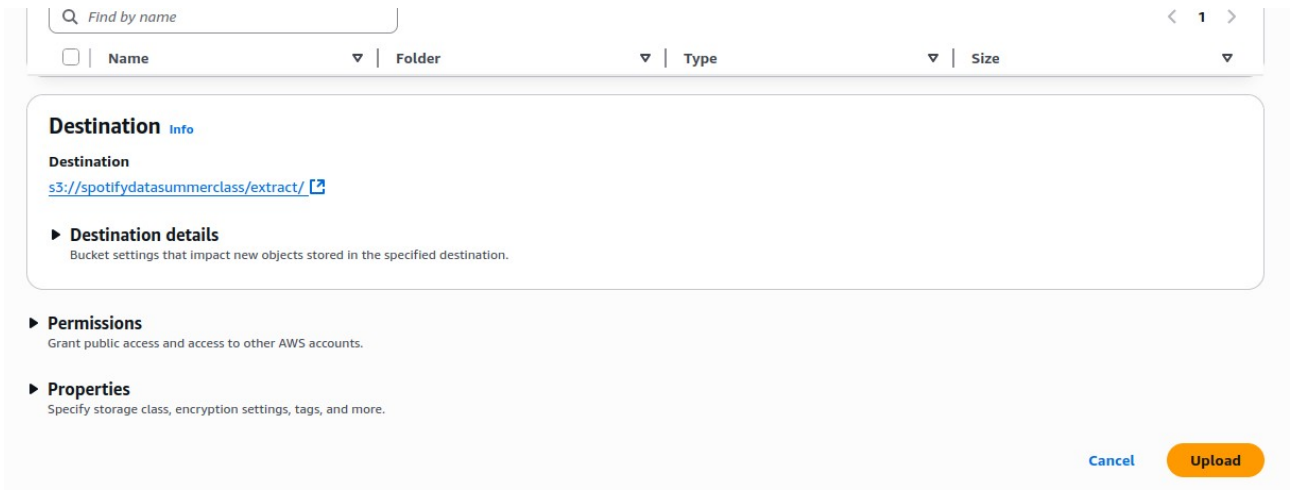
Click on Add Files to upload files



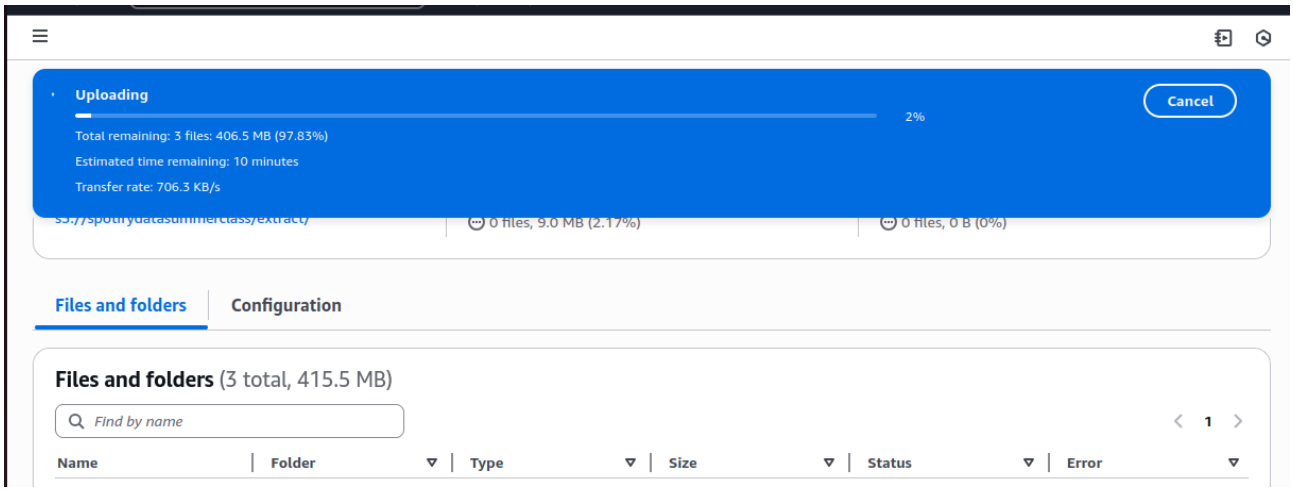
Browse to the extract directory where you have extracted your files. Select all 3 files and Upload them

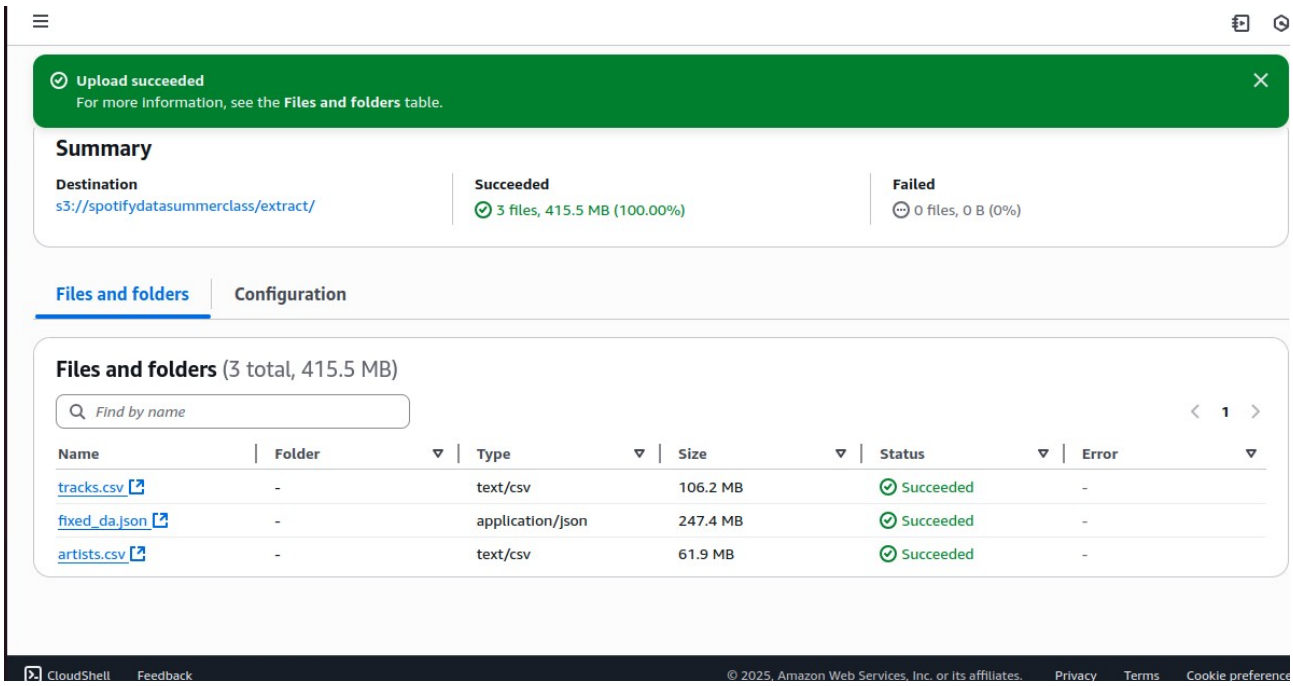


Scroll down and click the upload button. Note the Destination url this is important as it will be our input directory for our transform code

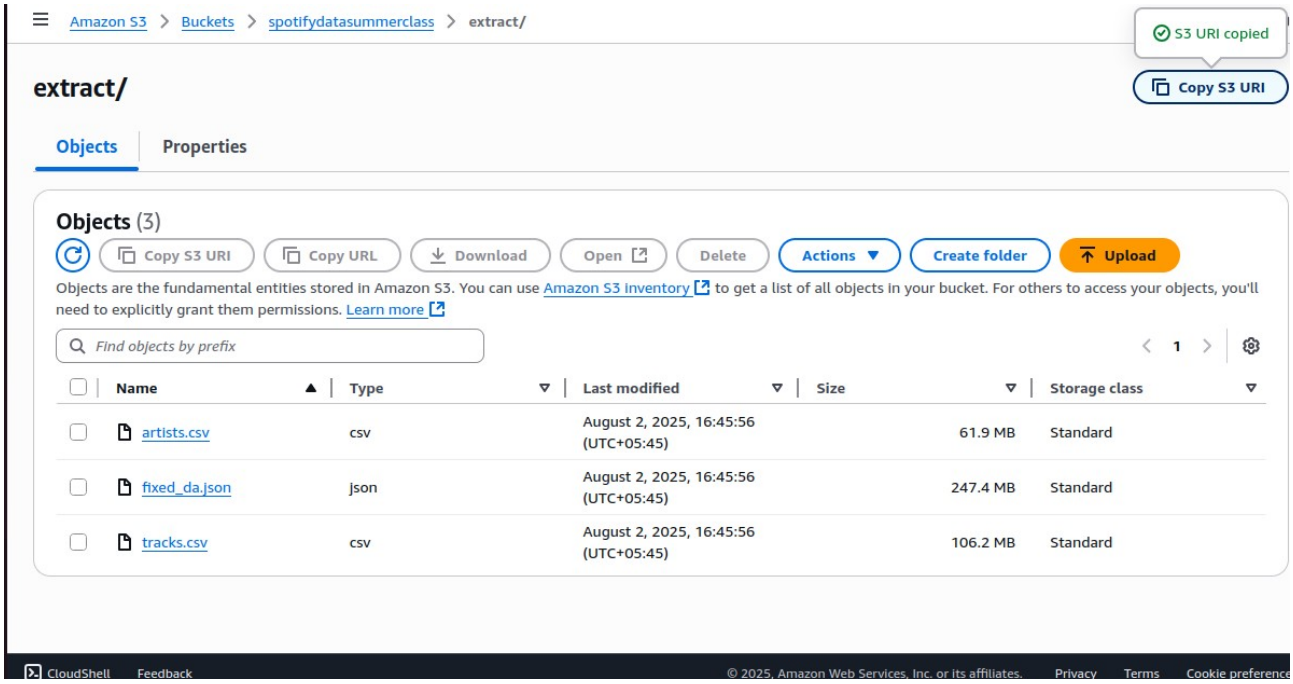


It might take some time depending on the internet. Wait for a while





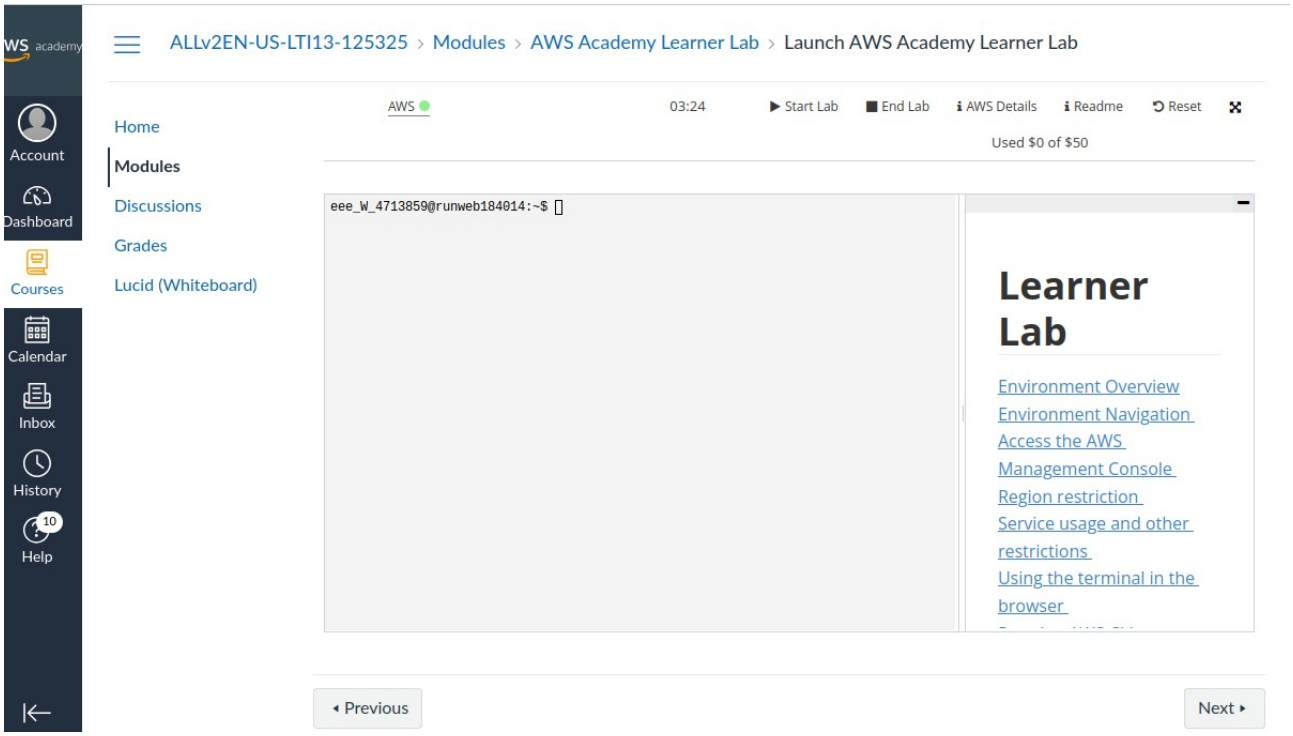
When it has completed click on Destination url and you will be taken to the relevant folder of bucket.



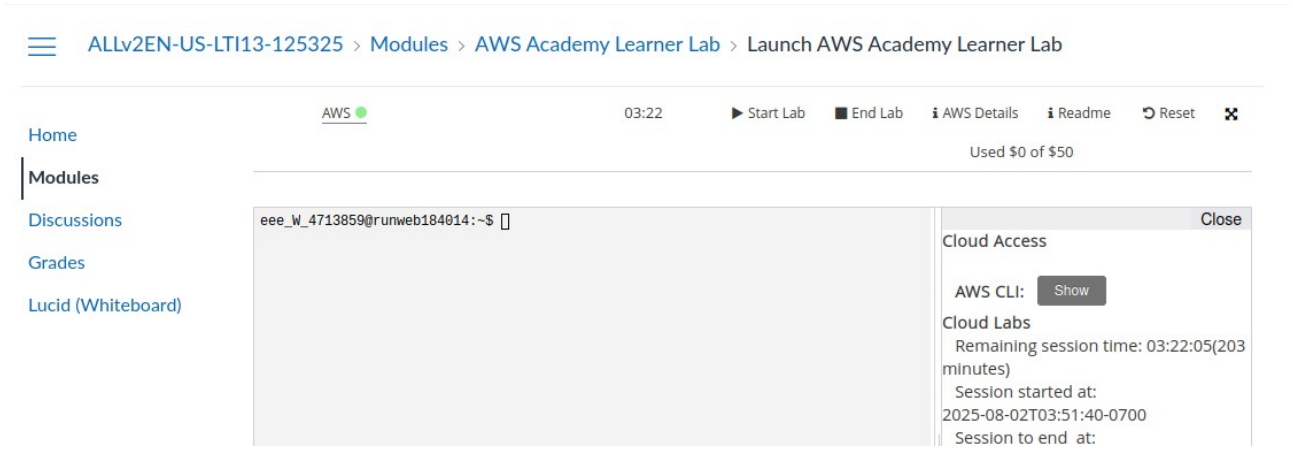
Click Copy s3 Uri available at the top: your uri should be something like this:
s3://spotifydatasummerclass/extract/
s3://The name of your bucket/folder
For pyspark however, we will need to pass an extra a after s3 i.e.,
s3a://The name of your bucket/folder
This behavior can be changed from config but we will let it be for now. Remember this as we will need this later

While it is uploading let us setup Access keys using AWS IAM which is user and identify management module of AWS. Generally we would have to do this ourselves (sysadmin/devops) but in our learner lab the access keys have already been provided by default.

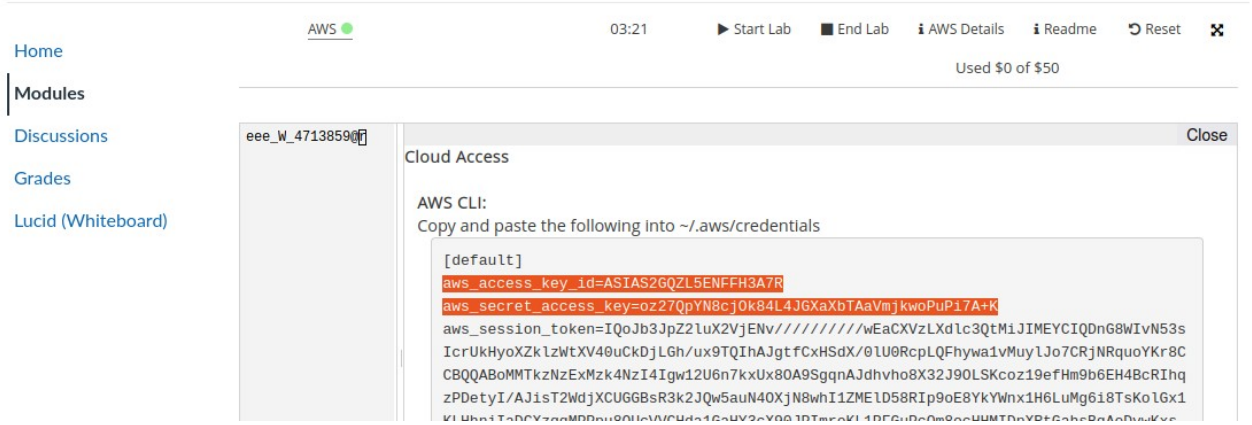
Go to the initial lab page



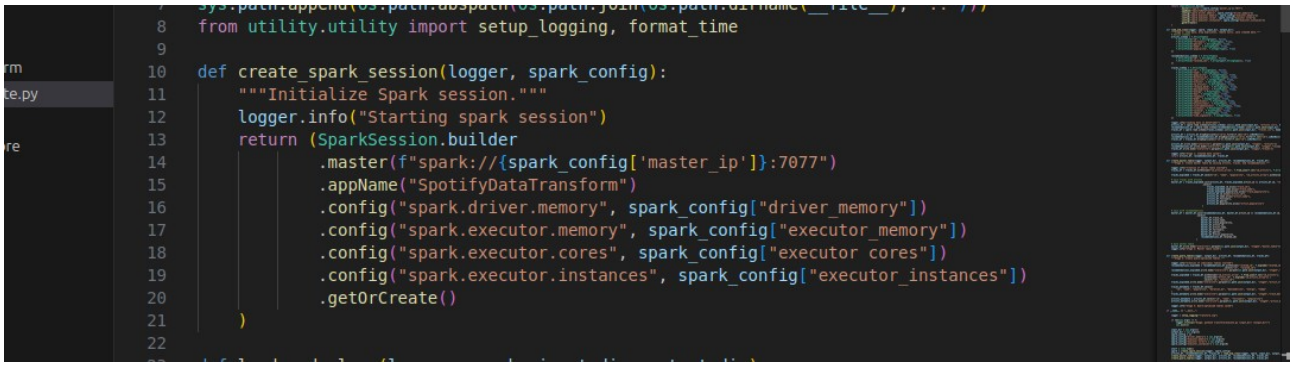
Click the aws detail button



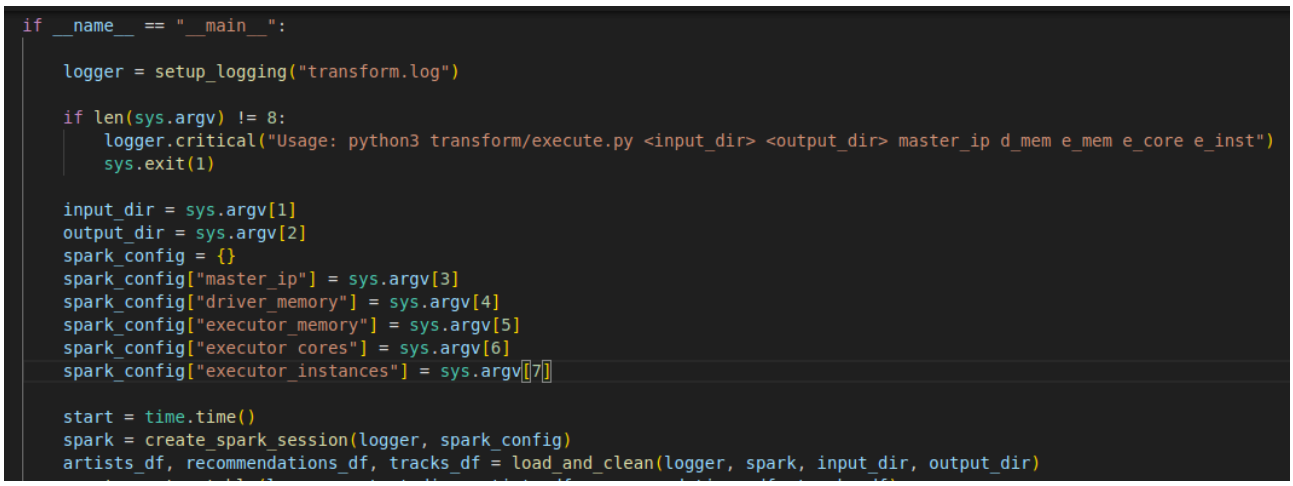
Click the Show button right next to AWS CLI. Copy the aws_access_key_id, aws_secret_access_key, aws_session_token and save it somewhere. Or you can visit this page when you need it later for setup



Let us make changes to our code to make it more dyanamic: In the create_spark_session of both transform and load and a new parameter for spark_config



We will be getting them as argument at the start



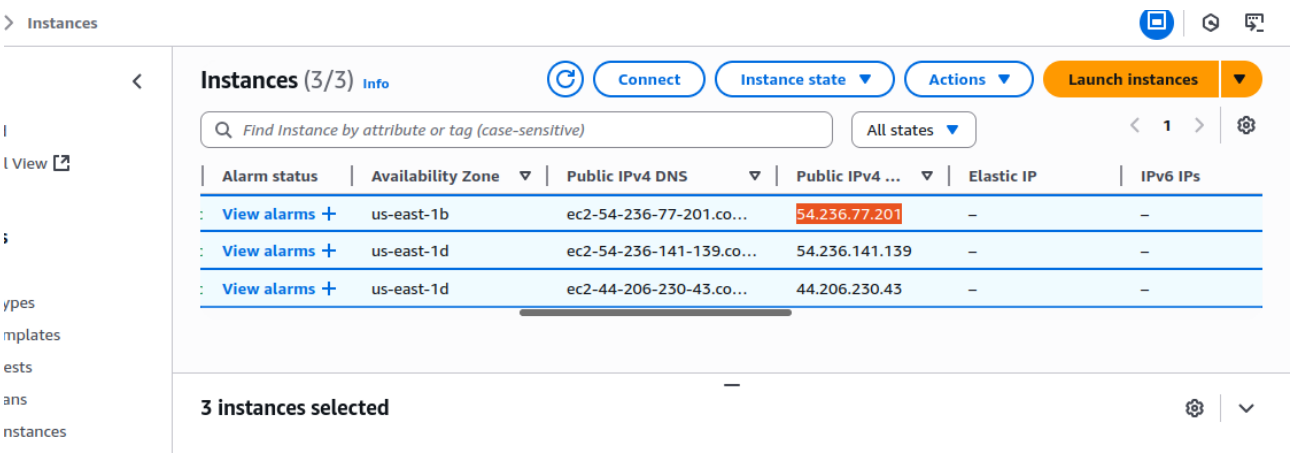
Do the same for load module. Additionally, for load module make sure postgres username, postgres password and postgres host all are also being passed as arguments

```python
if __name__ == "__main__":

    logger = setup_logging("load.log")

    if len(sys.argv) != 9:
        logger.error("Usage: python load/execute.py <input_dir> <pg_un> <pg_pw> pg_host d_mem e_mem e_core e_inst")
        sys.exit(1)

    input_dir = sys.argv[1]
    pg_un = sys.argv[2]
    pg_pw = sys.argv[3]
    pg_host = sys.argv[4]

    spark_config = {}
    spark_config["master_ip"] = sys.argv[5]
    spark_config["driver_memory"] = sys.argv[6]
    spark_config["executor_memory"] = sys.argv[7]
    spark_config["executor cores"] = sys.argv[8]
    spark_config["executor_instances"] = sys.argv[9]

    logger.info("Load stage started")
    start = time.time()

    spark = create_spark_session(logger, spark_config)
    create_postgres_tables(logger, pg_un, pg_pw, pg_host)
```

Push the changes to git. Note here I did not need to do git add as I have not added anything new but it is matter of habit and it does no harm

```
(venv) ardent@ardent:~/Workspace/etl$ git add .
(venv) ardent@ardent:~/Workspace/etl$ git commit -m "chore: logger message changes"
[master dcac62e] chore: logger message changes
 2 files changed, 2 insertions(+), 2 deletions(-)
(venv) ardent@ardent:~/Workspace/etl$ git push origin master
Enumerating objects: 11, done.
Counting objects: 100% (11/11), done.
Delta compression using up to 8 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (6/6), 568 bytes | 568.00 KiB/s, done.
Total 6 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To github.com:neotheobserver/spark-etl.git
   2ab260d..dcac62e  master -> master
(venv) ardent@ardent:~/Workspace/etl$
master  ⊗ 0 ⚠ 0   ardent (45 minutes ago)  Ln 150, Col 29  Spaces: 4  UTF-8  LF  {} Python
```

Now it is time to start the instances again. If you never stopped the instances then you do not need start again. The public ip of the instance might change when you start it again. So make sure you take note of it. The process of starting is same as stopping. Go to EC2> Instance select all instance and click on Start Instance. Let us open three different terminal as before and ssh into the 3 instances.



Use the public ip's to ssh using the previous pem key. At this point you should be able to do this on your own.

```
ardent@ardent:~/Desktop$ ssh -i pyspark.pem ubuntu@54.236.77.201
The authenticity of host '54.236.77.201 (54.236.77.201)' can't be established.
ED25519 key fingerprint is SHA256:bwa9r6liAHDGN9bjfyIF5pVskhIFJrPU0hfcuHyLzhU.
This host key is known by the following other names/addresses:
    ~/.ssh/known_hosts:4: [hashed name]
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
```

```
Connection to 54.174.126.38 closed by remote host.
Connection to 54.174.126.38 closed.
ardent@ardent:~/Desktop$ ssh -i pyspark.pem ubuntu@54.236.141.139
The authenticity of host '54.236.141.139 (54.236.141.139)' can't be established.
ED25519 key fingerprint is SHA256:NPyFKjMk/at2BV7wYJnpHdDictDdMj2QR+O9gFoTvp4.
This host key is known by the following other names/addresses:
```

```
Connection to 44.202.83.45 closed by remote host.
Connection to 44.202.83.45 closed.
ardent@ardent:~/Desktop$ ssh -i pyspark.pem ubuntu@44.206.230.43
The authenticity of host '44.206.230.43 (44.206.230.43)' can't be established.
ED25519 key fingerprint is SHA256:LduE6rtn48PSmLjtdtT2BGmUBDGabemleWxfbUK94ME.
This host key is known by the following other names/addresses:
    ~/.ssh/known_hosts:10: [hashed name]
```

Now that we have connected to the master as well as 2 worker node. Let us perform below task in all nodes.

In the .bashrc file of all 3 nodes. Add/export the aws key id, secret key, and access_token we copied earlier. This is required for our instance to access s3 buckets and other aws services.

```
fi

export AWS_ACCESS_KEY_ID=ASIAS2GQZL5ENFFH3A7R
export AWS_SECRET_ACCESS_KEY=oz27QpYN8cjOk84L4JGXaXbTAaVmjkwoPuPi7A+K
export AWS_SESSION_TOKEN=IQoJb3JpZ2luX2VjENv//////////wEaCXVzLXdlc3QtMiJIMEYCIQDnG8WIvN53sIcrUkHyoXZklzWtXV40uCkDjLGh
9TQIhAJgtfCxHSdX/0lU0RcpLQFhywa1vMuylJo7CRjNRquoYKr8CCBQQABoMMTkzNzExMzk4NzI4Igw12U6n7kxUx8OA9SgqnAJdhvho8X32J9OLSKco
efHm9b6EH4BcRIhqzPDetyI/AJisT2WdjXCUGGBsR3k2JQw5auN4OXjN8whI1ZMElD58RIp9oE8YkYWnx1H6LuMg6i8TsKolGx1KLHhniIaDCXzqgMPPp
UcVVCHda1GaHY3cX90JPImreKL1PFGuPcOm8ocHHMIDpXRtGahsBgAeDvwKxsp/m3Xn5WFv46E1qChl/RS84mJAwp/kmKJrdBhBfZHkyGpvFwJaJa6lBx
nJyeeRUUjeO9UUfVcLC9i5oclQi1d/AcnT2j8Z7CfCjdEAn0WXebngr01Ql1jLiY2MHjh8OsQeI/L3x5s+ORvLQcBpMgPA5IKa3GEE+d5mAqsyTi7BCFE
927fEBjqcAQiqVPof3M46n4w31rGcXXwQXgx5IkiExErt5ntmMDmnSeERXQLayJ9TrUY+mYCbcjRhkpHrf45TwDgzTi4gy2Jy/FzF+vogVp8pnab35Ppo
/CO2+5Ah+0lYwiEeh+DTADLi3AQeYS05ySaOfEMIp/VKf3iiwOpwM1HeyLl1EiMEHuNTm1rKLVkI8pJ61GyXO169qS0g7FuCMSA==
export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64
export SPARK_HOME=/opt/spark
export PATH=$PATH:$SPARK_HOME/bin:$SPARK_HOME/sbin
export PYSPARK_PYTHON=/usr/bin/python3
```

Everytime the .bashrc file is changed it needs to be sourced to update

```
ubuntu@ip-172-31-40-25:~$ source .bashrc
```

Do this for all 3 instances.

Now, we need to download the required jars for our pyspark to be able to connect to aws. This will be done for all 3 nodes as well

Change directory to jars of /opt/spark → remember this is where we kept our spark files

```
ubuntu@ip-172-31-40-25:~$ vim .bashrc
ubuntu@ip-172-31-40-25:~$ cd /opt/spark/jars
ubuntu@ip-172-31-40-25:/opt/spark/jars$
```

Download two jars file

```
ubuntu@ip-172-31-40-25:~$ cd /opt/spark/jars
ubuntu@ip-172-31-40-25:/opt/spark/jars$ wget https://repo1.maven.org/maven2/org/apache/hadoop/hadoop-aws/3.3.1/hadoop-a
s-3.3.1.jar
--2025-08-02 11:48:42--  https://repo1.maven.org/maven2/org/apache/hadoop/hadoop-aws/3.3.1/hadoop-aws-3.3.1.jar
Resolving repo1.maven.org (repo1.maven.org)... 146.75.32.209, 2a04:4e42:78::209
Connecting to repo1.maven.org (repo1.maven.org)|146.75.32.209|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 870644 (850K) [application/java-archive]
Saving to: 'hadoop-aws-3.3.1.jar'

hadoop-aws-3.3.1.jar          100%[===============================================>] 850.24K  --.-KB/s    in 0.009s

2025-08-02 11:48:42 (90.1 MB/s) - 'hadoop-aws-3.3.1.jar' saved [870644/870644]

ubuntu@ip-172-31-40-25:/opt/spark/jars$ wget https://repo1.maven.org/maven2/com/amazonaws/aws-java-sdk-bundle/1.12.99/a
s-java-sdk-bundle-1.12.99.jar
--2025-08-02 11:49:23--  https://repo1.maven.org/maven2/com/amazonaws/aws-java-sdk-bundle/1.12.99/aws-java-sdk-bundle-1
12.99.jar
Resolving repo1.maven.org (repo1.maven.org)... 146.75.36.209, 2a04:4e42:79::209
Connecting to repo1.maven.org (repo1.maven.org)|146.75.36.209|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 246882976 (235M) [application/java-archive]
Saving to: 'aws-java-sdk-bundle-1.12.99.jar'

aws-java-sdk-bundle-1.12.99.j 100%[===============================================>] 235.45M   122MB/s    in 1.9s

2025-08-02 11:49:25 (122 MB/s) - 'aws-java-sdk-bundle-1.12.99.jar' saved [246882976/246882976]

ubuntu@ip-172-31-40-25:/opt/spark/jars$
```

wget https://repo1.maven.org/maven2/org/apache/hadoop/hadoop-aws/3.3.1/hadoop-aws-3.3.1.jar
wget https://repo1.maven.org/maven2/com/amazonaws/aws-java-sdk-bundle/1.12.99/aws-java-sdk-bundle-1.12.99.jar

Do the above jar download task for all 3 instances.

Now let us start the spark in master and worker and worker nodes. For master use start-master.sh. For worker use start-worker master_ip:7077

```
ubuntu@ip-172-31-40-25:~$ start-master.sh
starting org.apache.spark.deploy.master.Master, logging to /opt/spark/logs/spark-ubuntu-org.apache.spark.deploy.master.M
aster-1-ip-172-31-40-25.out
ubuntu@ip-172-31-40-25:~$
```

```
ubuntu@ip-172-31-84-118:~$ start-worker.sh spark://54.236.77.201:7077
starting org.apache.spark.deploy.worker.Worker, logging to /opt/spark/logs/spark-ubuntu-org.apache.spark.deploy.worker.W
orker-1-ip-172-31-84-118.out
ubuntu@ip-172-31-84-118:~$
```

```
ubuntu@ip-172-31-84-134:~$ start-worker.sh spark://54.236.77.201:7077
starting org.apache.spark.deploy.worker.Worker, logging to /opt/spark/logs/spark-ubuntu-org.apache.spark.deploy.worker.W
orker-1-ip-172-31-84-134.out
ubuntu@ip-172-31-84-134:~$
```

Verify the web ui using ip address of master and 8080 port i.e., master_ip:8080

You should get above ui with Workers listing 2 workers

Do the below tasks only in master node

In the master node. Let us install pyspark globally so that our python script can recognize the code we have written.
Initially install python3-pip using command

```
Command 'pip' not found, but can be installed with:
sudo apt install python3-pip
ubuntu@ip-172-31-40-25:~$ sudo apt install python3-pip
Reading package lists... Done
Building dependency tree... Done
```

sudo apt install python3-pip

Then install pyspark globally using below command

```
No VM guests are running outdated hypervisor (qemu) binaries on this host.
ubuntu@ip-172-31-40-25:~$ pip install --break-system-packages pyspark==3.3.1
Defaulting to user installation because normal site-packages is not writeable
Collecting pyspark==3.3.1
```

pip install --break-system-packages pyspark==3.3.1

Now let us clone our etl from github in our master node. This is where we will be executing all our codes. Make sure you change directory to your home if you are not already there using
cd ~
before running the command to clone

```
ubuntu@ip-172-31-40-25:/opt/spark/jars$ cd ~
ubuntu@ip-172-31-40-25:~$ git clone https://github.com/neotheobserver/spark-etl.git etl
Cloning into 'etl'...
remote: Enumerating objects: 18, done.
remote: Counting objects: 100% (18/18), done.
remote: Compressing objects: 100% (9/9), done.
remote: Total 18 (delta 2), reused 18 (delta 2), pack-reused 0 (from 0)
Receiving objects: 100% (18/18), 4.86 KiB | 1.21 MiB/s, done.
Resolving deltas: 100% (2/2), done.
ubuntu@ip-172-31-40-25:~$
```

For those who have been following properly use your own repo and work on it. For those who have not been able to follow can use my repo from below but make sure you do your own work as well. Notice etl after “git clone github_url “ It is the name of the directory where the repository will be cloned.

git clone https://github.com/neotheobserver/spark-etl.git etl

Change directory to etl (where we cloned our code)

```
Resolving deltas: 100% (4/4), done.
ubuntu@ip-172-31-40-25:~$ cd etl
```

Try to execute the transform code

```
ubuntu@ip-172-31-40-25:~/etl$ python3 transform/execute.py
2025-08-02 12:09:23,875 [MainThread  ] [CRITI]  Usage: python3 transform/execute.py <input_dir> <output_dir
_mem e_mem e_core e_inst
```

We get an error as we should because we did not pass the required input variables

```
_mem e_mem e_core e_inst
ubuntu@ip-172-31-40-25:~/etl$ python3 transform/execute.py s3://spotifydatasummerclass/extract/ s3://spotifydatasummercl
ass/transform/ 54.236.77.201 2g 4g 2 2
2025-08-02 12:10:46,278 [MainThread  ] [INFO ]  Starting spark session
```

Now let us pass the required parameters, input directory, output_directory, master_ip, driver_mem 2g means 2 gb, executor mem, executor core, no of executor

```
ubuntu@ip-172-31-40-25:~$ cd etl
ubuntu@ip-172-31-40-25:~/etl$ python3 transform/execute.py s3://spotifydatasummerclass/extract/ s3://spotifydatasummercl
ass/transform/ 54.236.77.201 2g 4g 2 2
2025-08-02 12:21:50,918 [MainThread  ] [INFO ]  Starting spark session
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
25/08/02 12:21:55 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java cl
asses where applicable
2025-08-02 12:21:58,793 [MainThread  ] [INFO ]  Loading data in dataframes
25/08/02 12:22:01 WARN FileStreamSink: Assume no metadata directory. Error while looking for metadata directory in the p
ath: s3://spotifydatasummerclass/extract/artists.csv.
org.apache.hadoop.fs.UnsupportedFileSystemException: No FileSystem for scheme "s3"
        at org.apache.hadoop.fs.FileSystem.getFileSystemClass(FileSystem.java:3443)
        at org.apache.hadoop.fs.FileSystem.createFileSystem(FileSystem.java:3466)
        at org.apache.hadoop.fs.FileSystem.access$300(FileSystem.java:174)
        at org.apache.hadoop.fs.FileSystem$Cache.getInternal(FileSystem.java:3574)
        at org.apache.hadoop.fs.FileSystem$Cache.get(FileSystem.java:3521)
```

This error was expected. It says it cannot find s3. This is because spark needs s3a uri unless you change the config. Let us run the code again after modify the input parameters value. Make sure you use your own bucket names, master ip and not copy mine as it will be different for everyone.

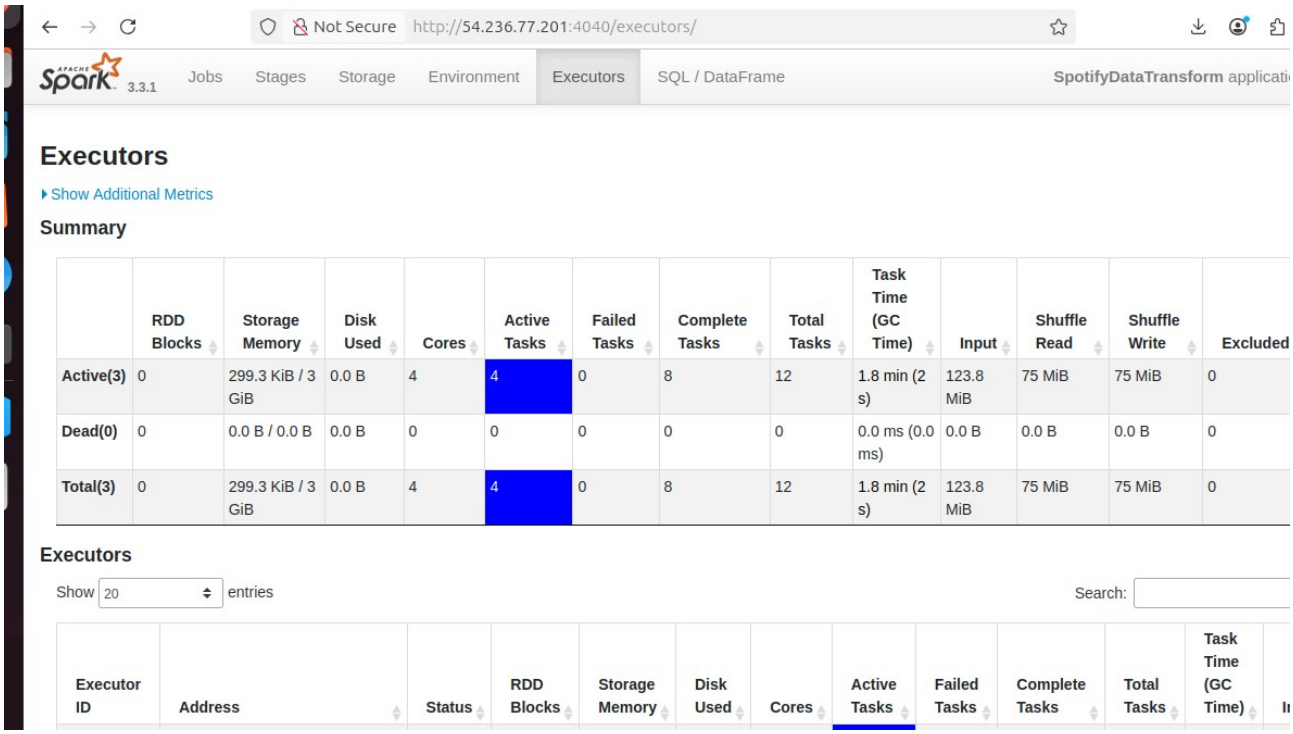
For executor memory pass a little less then what you have total as there is some overhead that needs to be accounted for

```
ubuntu@ip-172-31-40-25:~/etl$ python3 transform/execute.py s3a://spotifydatasummerclass/extract/ s3a://spotifydatasummer
class/transform/ 54.236.77.201 2g 2g 2 2
2025-08-02 12:44:01,340 [MainThread  ] [INFO ]  Starting spark session
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
25/08/02 12:44:06 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java cl
asses where applicable
2025-08-02 12:44:09,245 [MainThread  ] [INFO ]  Loading data in dataframes
25/08/02 12:44:11 WARN MetricsConfig: Cannot locate configuration: tried hadoop-metrics2-s3a-file-system.properties,hado
op-metrics2.properties
25/08/02 12:45:01 WARN package: Truncated the string representation of a plan since it was too large. This behavior can
be adjusted by setting 'spark.sql.debug.maxToStringFields'.
2025-08-02 12:45:14,809 [MainThread  ] [INFO ]  Stage 1: Cleaned data saved
2025-08-02 12:45:14,809 [MainThread  ] [INFO ]  Creation of master table started
2025-08-02 12:45:40,590 [MainThread  ] [INFO ]  Stage 2: Master table saved
2025-08-02 12:45:40,590 [MainThread  ] [INFO ]  Creation of individual tables started
2025-08-02 12:46:19,767 [MainThread  ] [INFO ]  Stage 3: Query-optimized tables saved
2025-08-02 12:46:19,768 [MainThread  ] [INFO ]  Transformation pipeline completed
2025-08-02 12:46:19,768 [MainThread  ] [INFO ]  Total time taken 0 hours, 2 minutes, 18 seconds
2025-08-02 12:46:19,768 [MainThread  ] [INFO ]  Closing down clientserver connection
ubuntu@ip-172-31-40-25:~/etl$
```

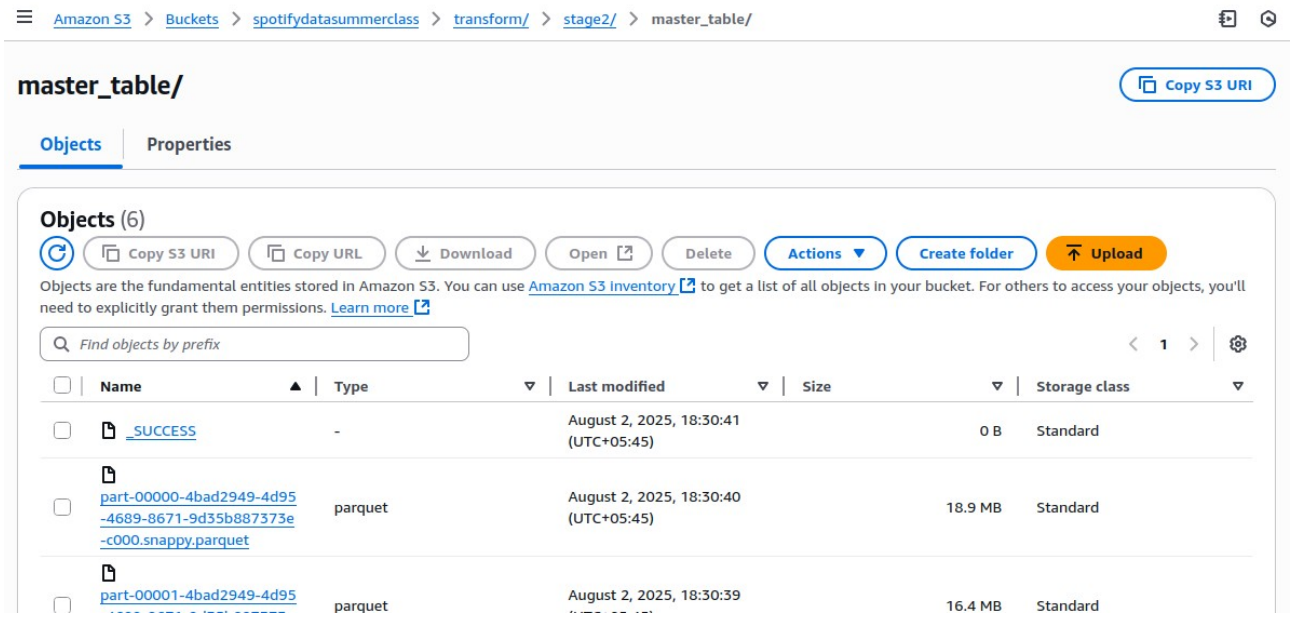
python3 transform/execute.py s3a://spotifydatasummerclass/extract/ s3a://spotifydatasummerclass/transform/ 54.236.77.201 2g 2g 2 2

Notice that the memory supplied for executor is only 2g even though our total memory is 4gb this is because there are several overheads for setup, for storage and so on. So only a portion of the memory can be used. If you created instance using different instance having different memory and core modify the input parameter accordingly

You should be able to access the spark-ui as you did for local using master_ip:4040



After the job has completed let us go to our s3 bucket and verify whether new files have been created in respective folder or not



Congratulation! You have ran your first ETL job using AWS based cloud infrastructure