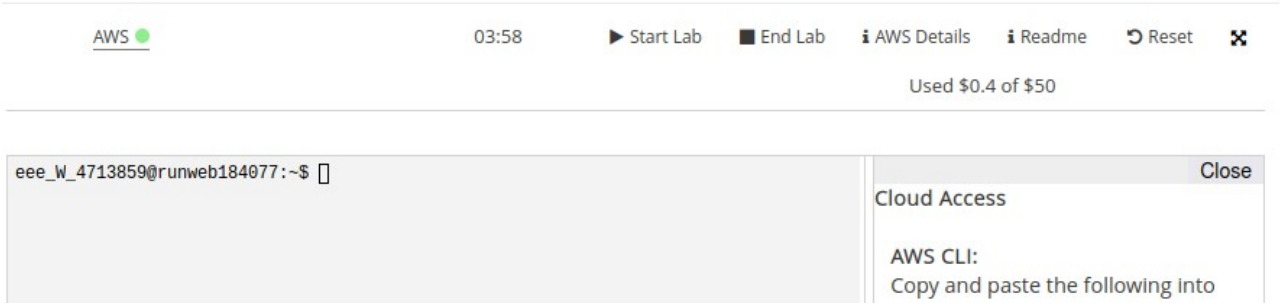
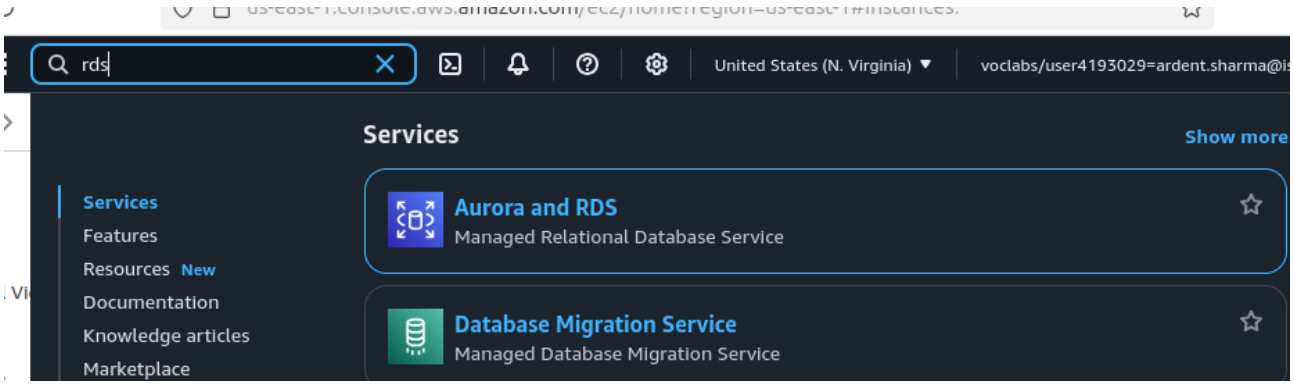
Open the module AWS learner lab and click start lab:



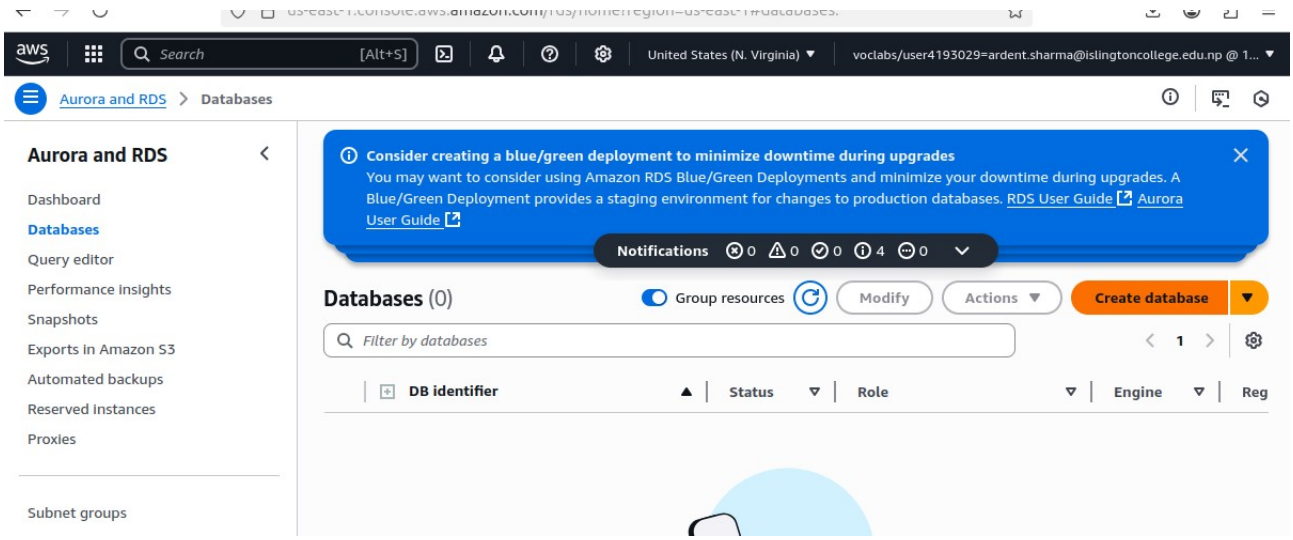
When the AWS has a green light right to it, that means our lab is ready.

If you go to ec2 and view the instances you will see that all 3 of your instances have already been started for you. If you are low on credits you should turn these off if you have other things to do. As they incur charges. These resources are not free of cost.

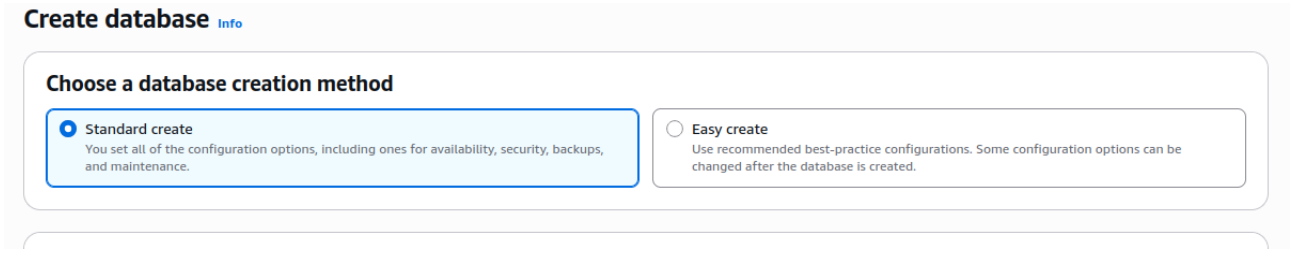
Search for rds in the search bar and click on Aurora and RDS service



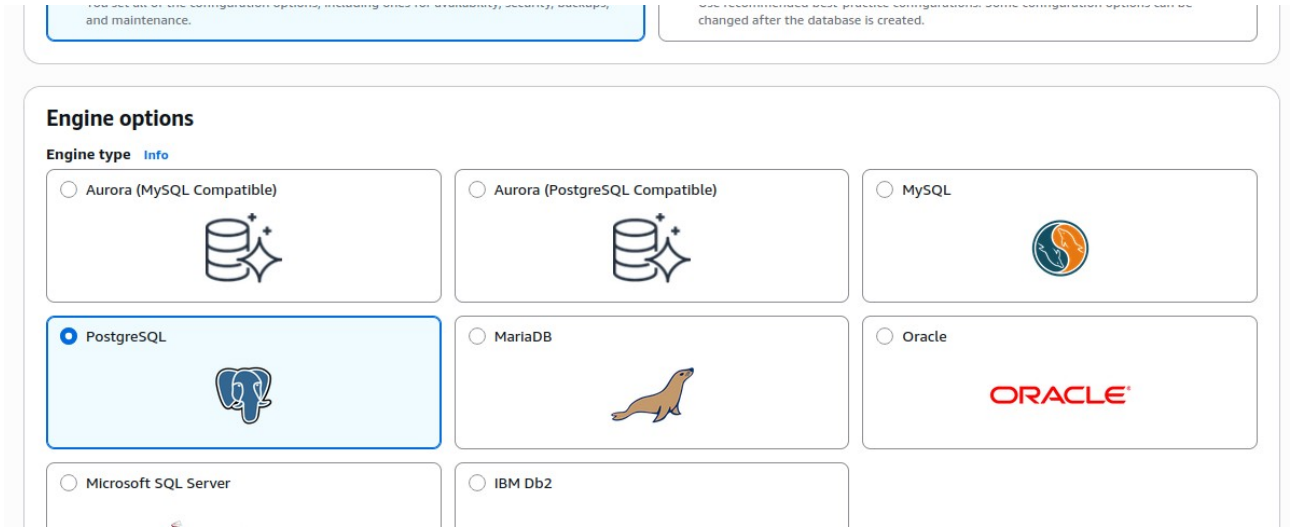
Go to databases section from left hand side and click on Create database button



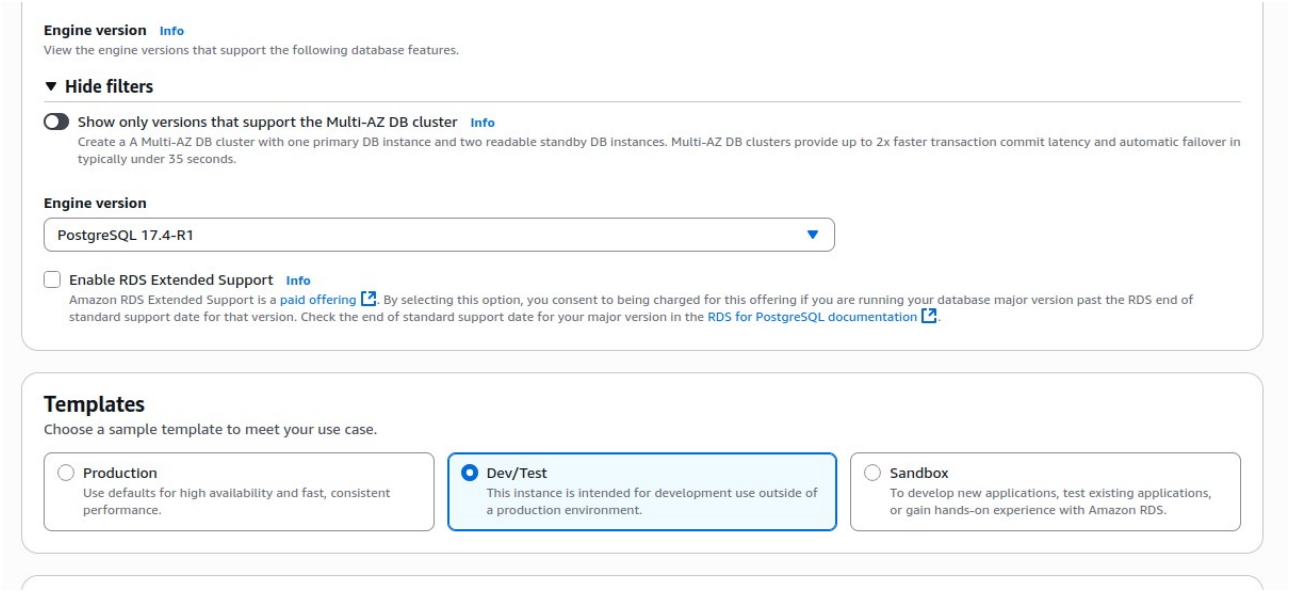
Choose standard create



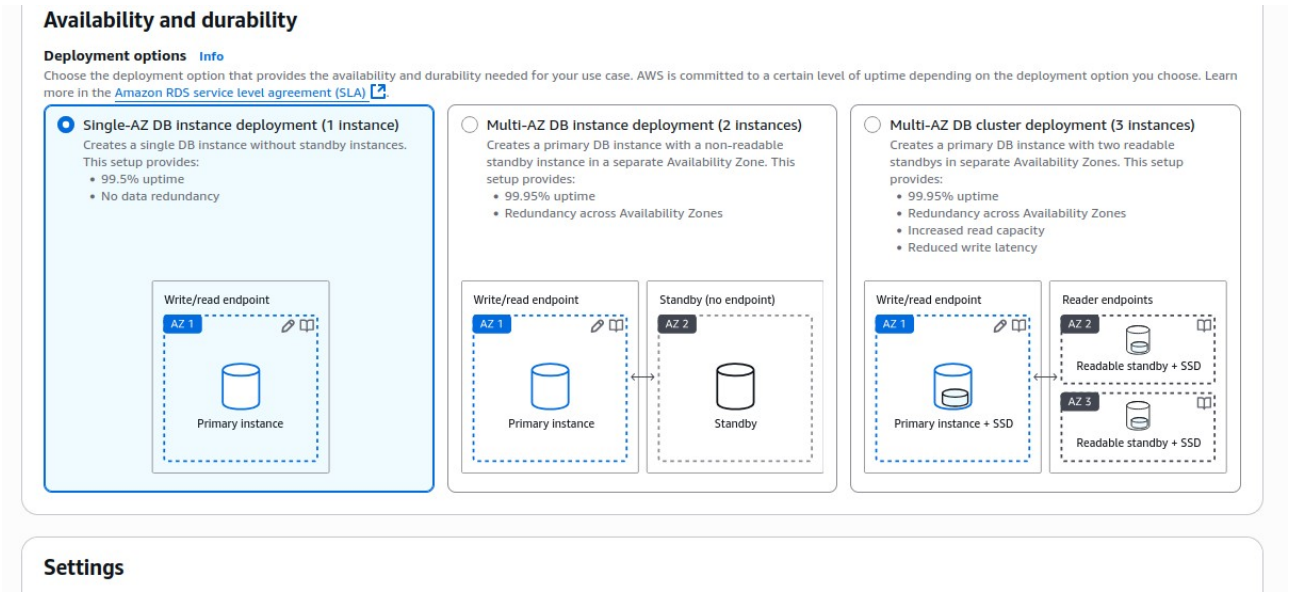
Then choose postgresql for engine options



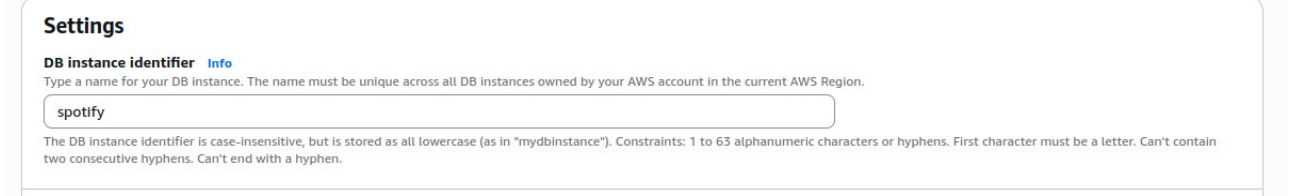
Leave the engine option to default and select Dev/Test for the template. We are not using Production as it might incur some extra charges. The setup should be same for all 3 options.



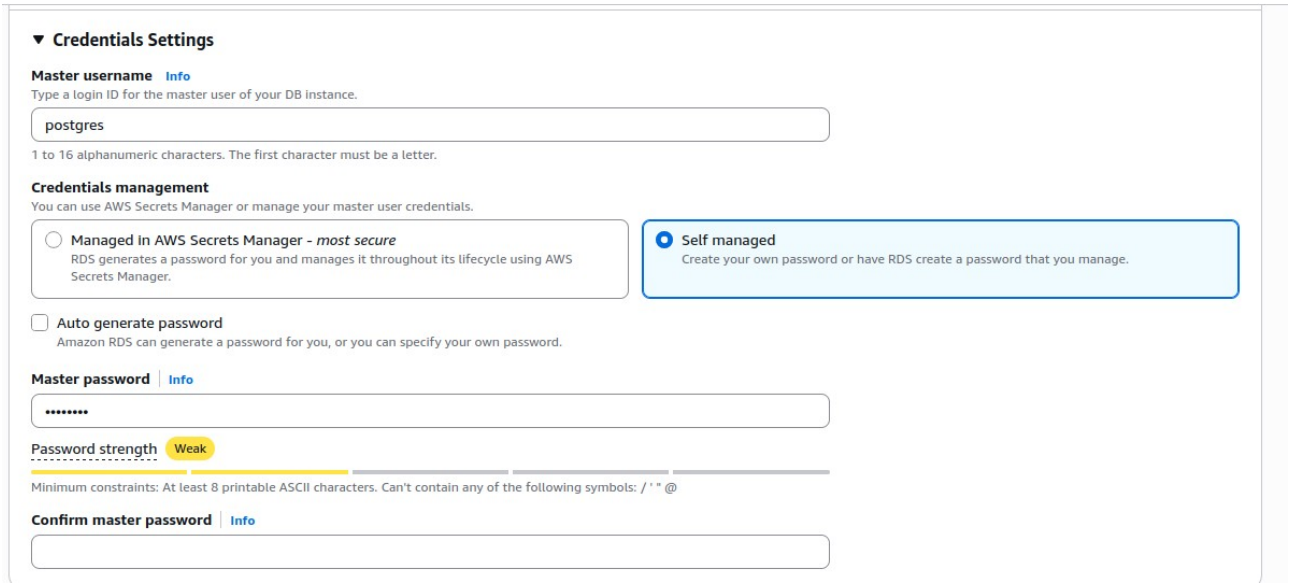
We do not care about availability and durability at this moment so we will select the first single Db instance deployment option. For production, other options will be a better choice based on the application type and expected load



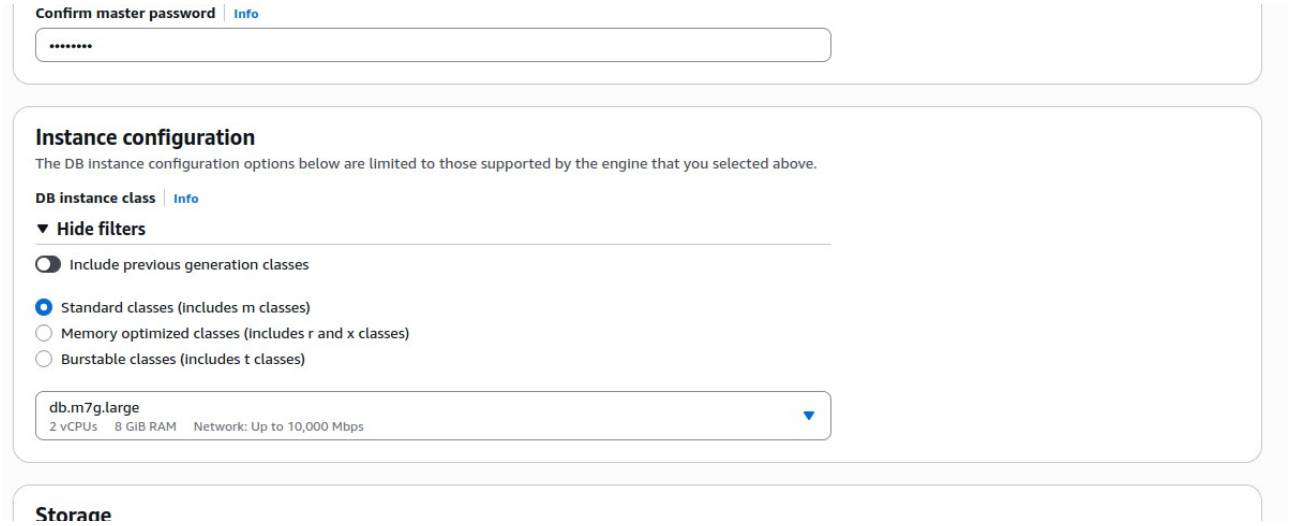
Give a name for your db instance. It can be whatever you want just has to be unique



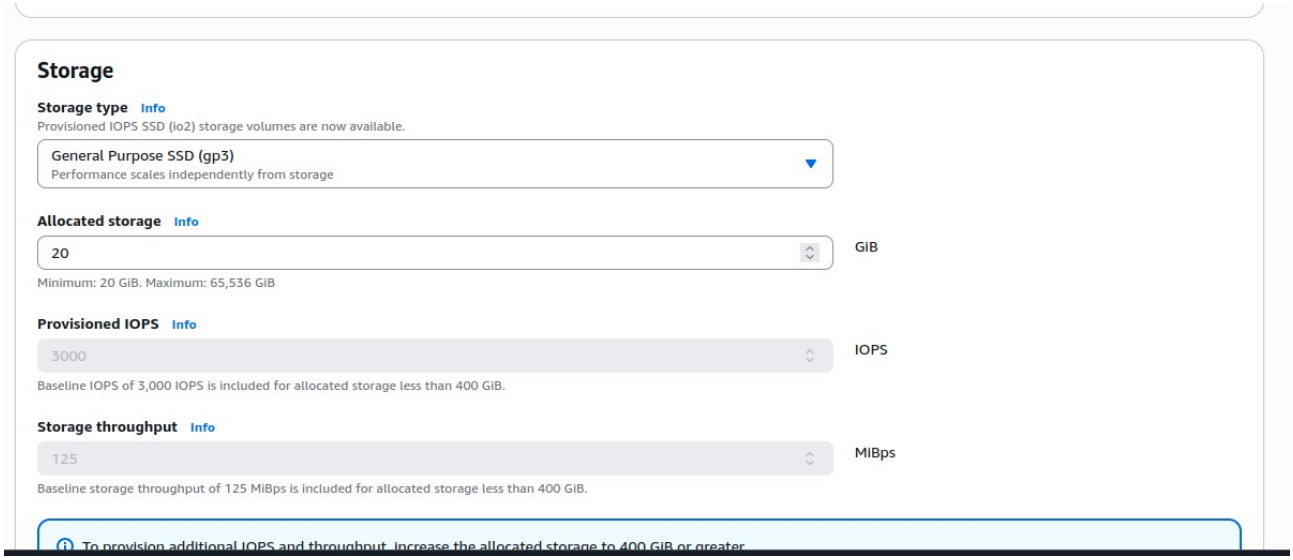
Give an username, make sure you select self-managed and give a password. Make sure you remember both of these as this is what we will need to access the database server



For instance classes select the default option i.e., Standard classes and db.m7g.large 2 cpu and 8 gb ram. This should suffice for our use case. Note that this is the ec2 instance we are choosing to setup our postgres in. We could very well do this ourselves but instead we are using managed service provided by AWS

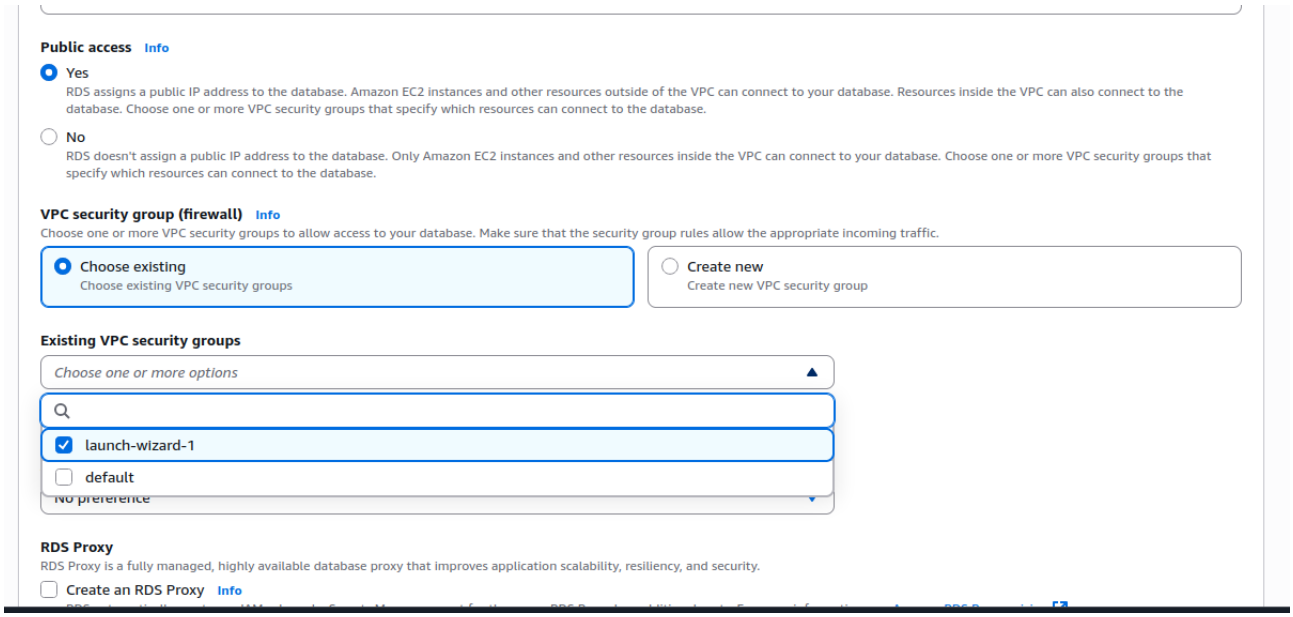


Decrease the allocated Storage to 20gb as we will not need anything more than that. Leave everything else as default

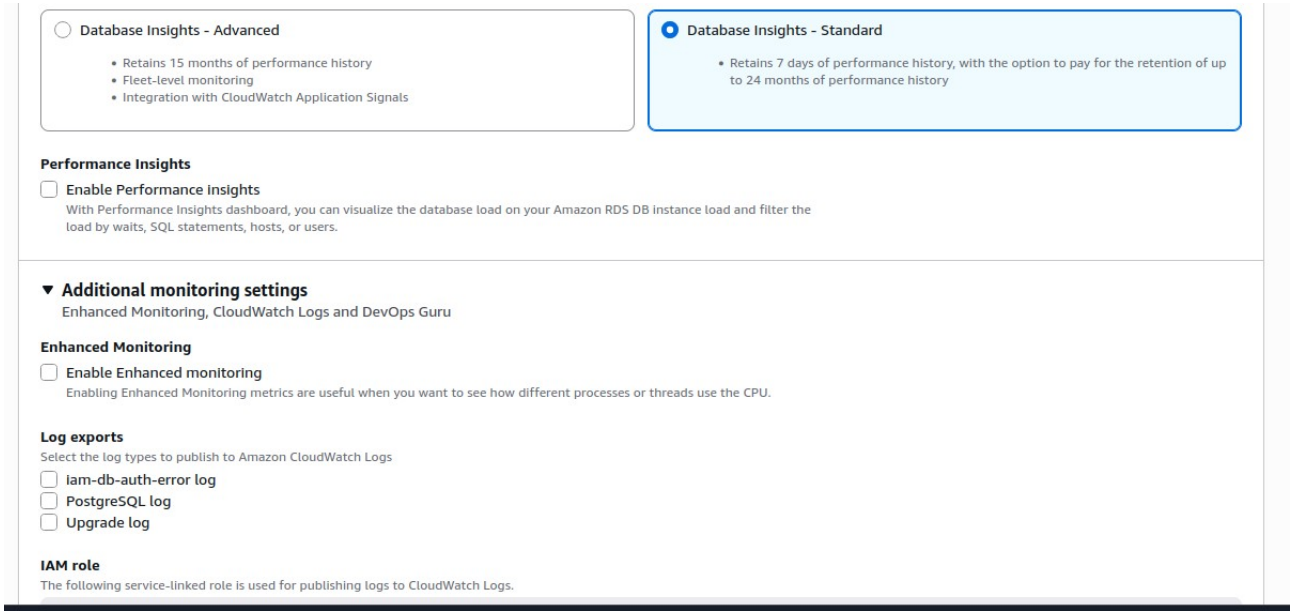


Leave everything else to default.

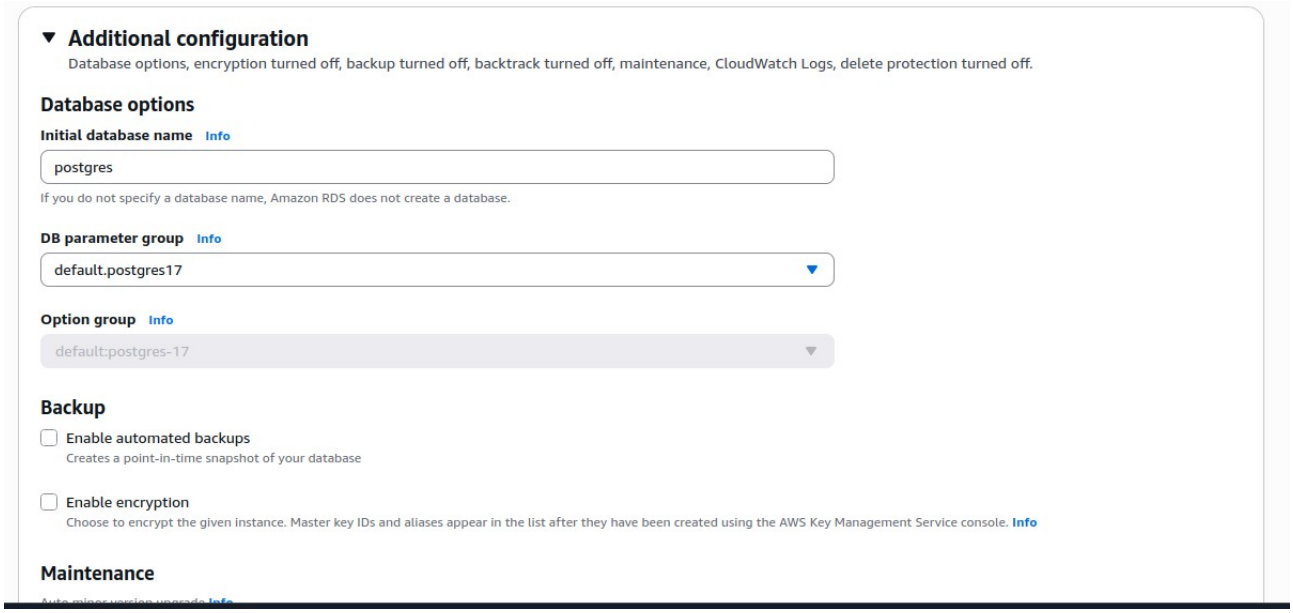
Modify the public access to yes so that we can access the database from anywhere. Again this is not a good idea for production environment. At this point you should already have this instinct. Chosse existing VPC security groups and select the one we created for ec2 instances.



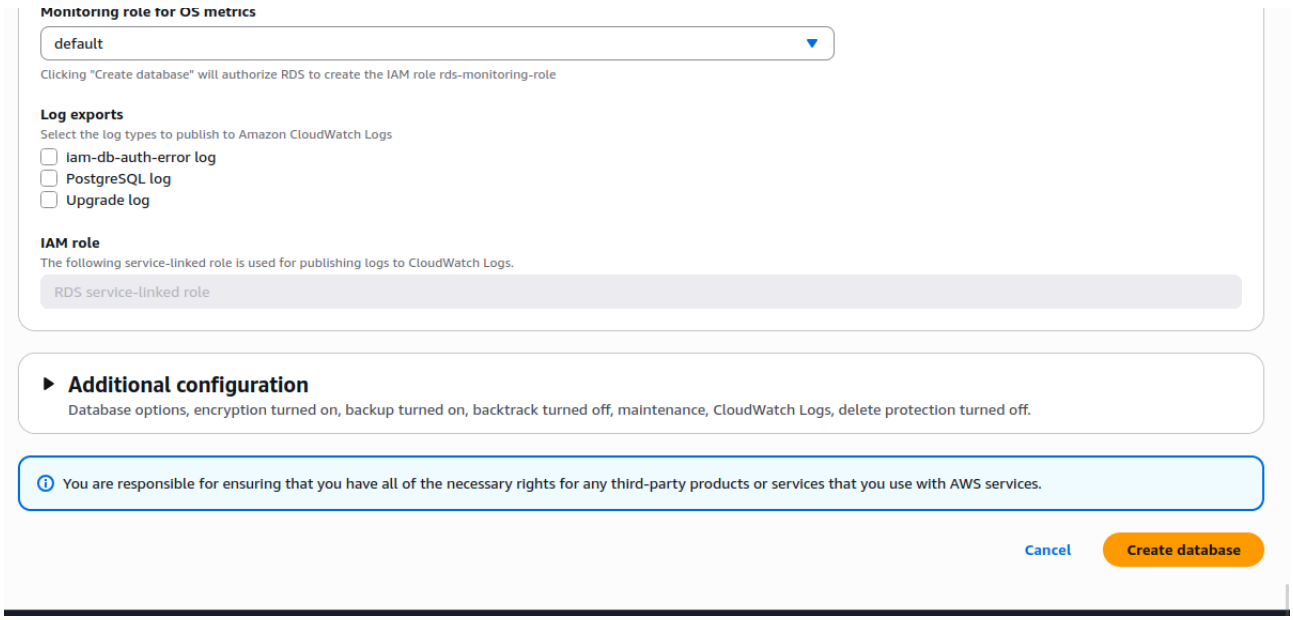
Leave most of the things to default. Untick the insights, and monitoring related stuffs.



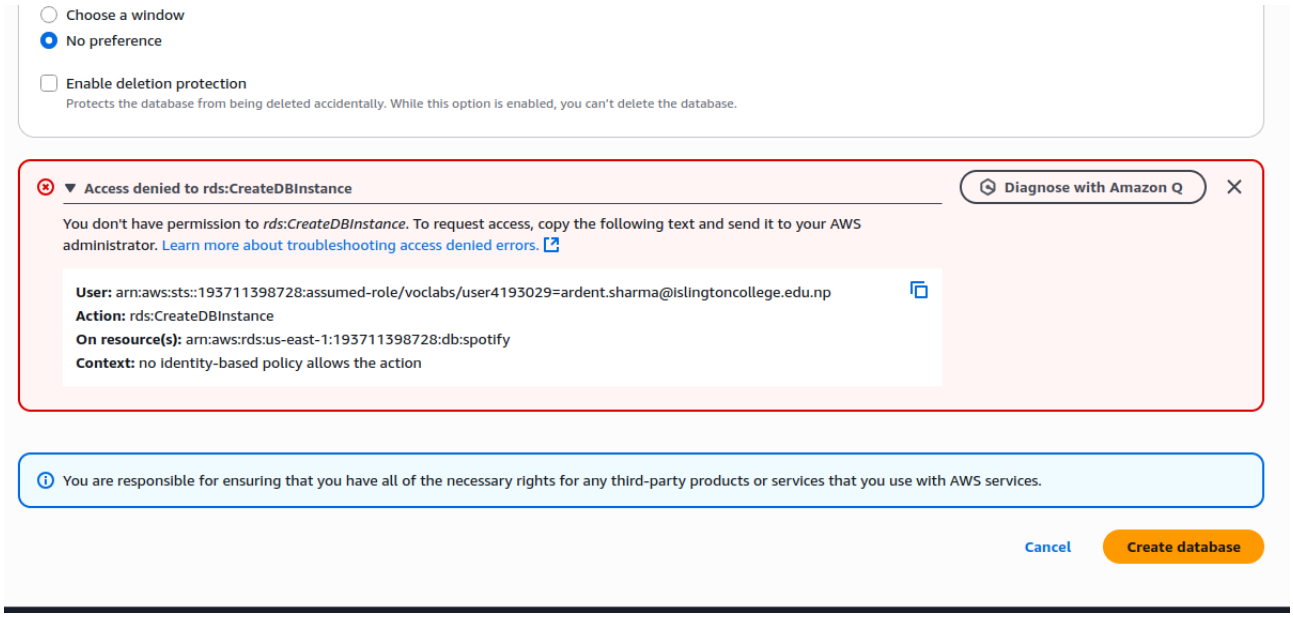
In additional configuration give name to database and disable automated backups (this will change depending on use case but enabling backup will incur extra charges).



Scroll down and select create database button



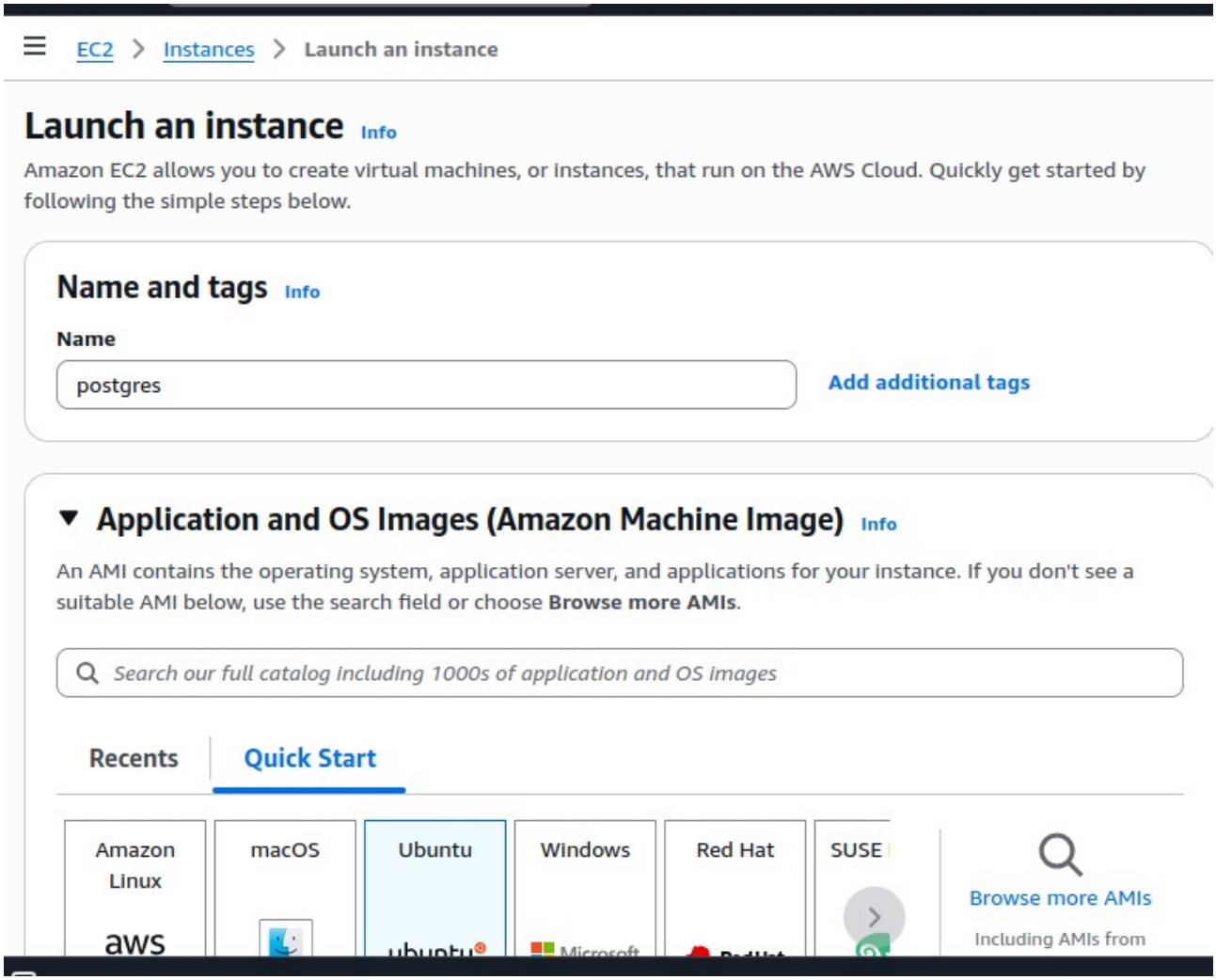
You should get the below error. Apparently this lab does not allow creation of RDS databases. We need to configure our database manually. If it worked for you, do not follow below steps as you would already have a database setup.



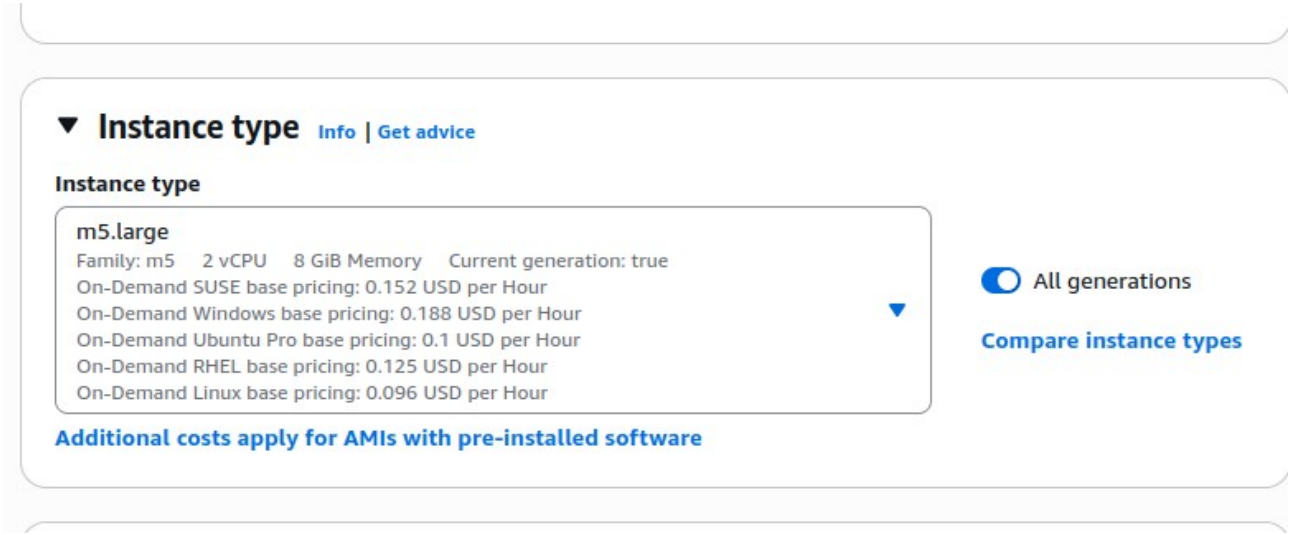
Let us create new ec2 instance and setup postgresql there.

Go to Ec2>instances>launch an instances

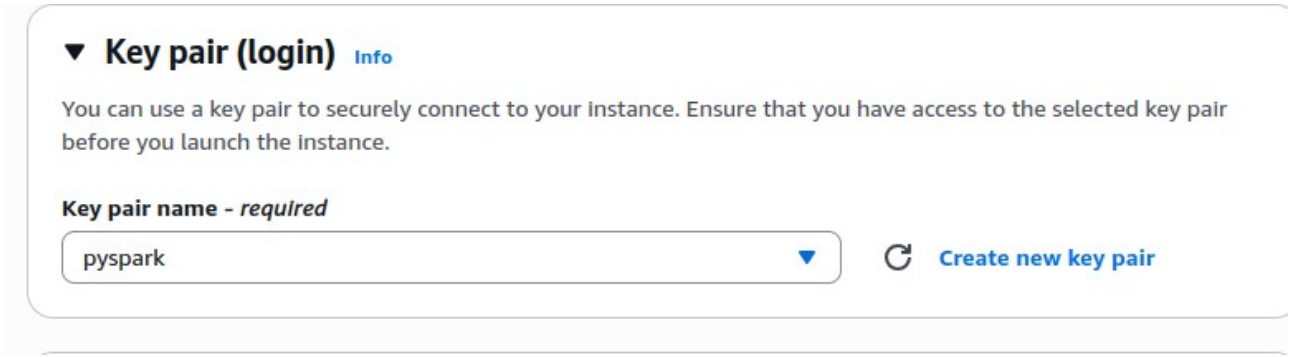
Give the instance a name. I have given it postgres. And choose Ubuntu image as we did for our spark instances. Almost everything will be the same except for the instance type.



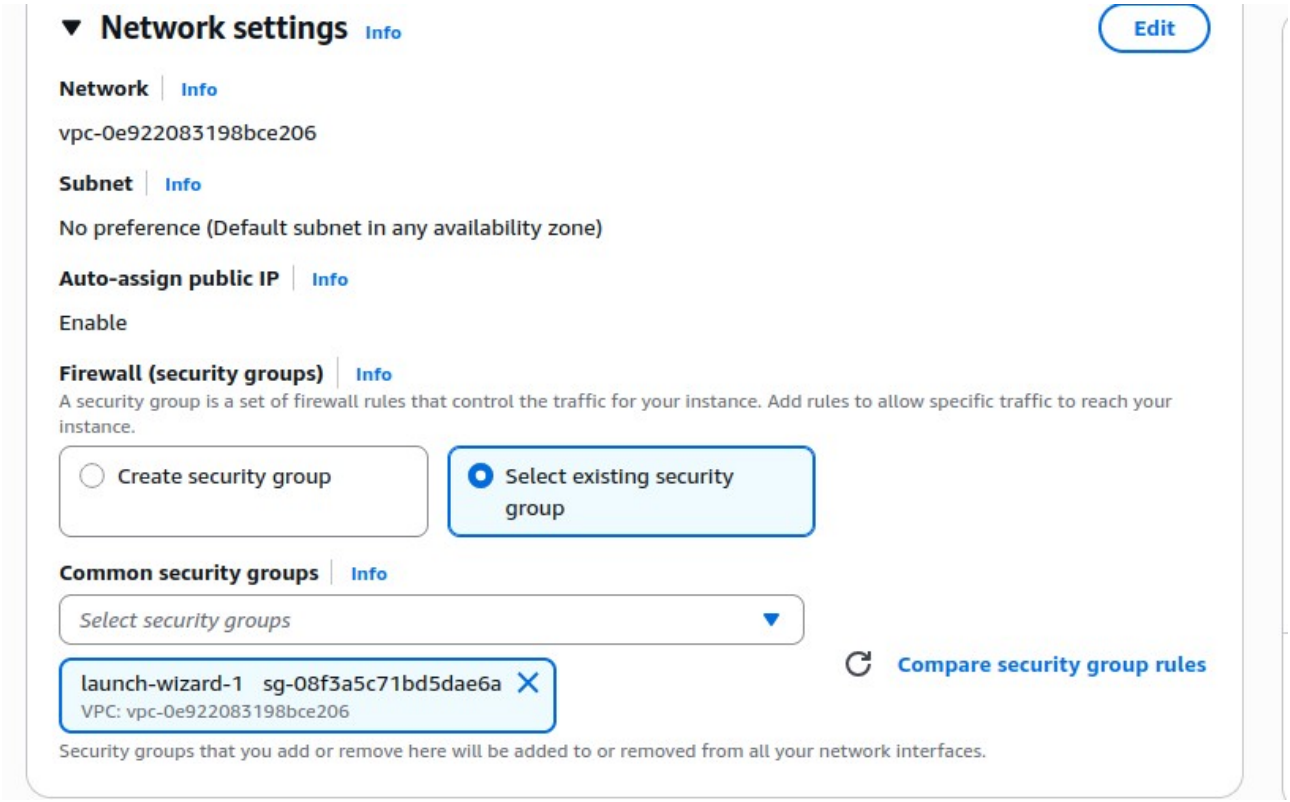
For instance type select m5.large



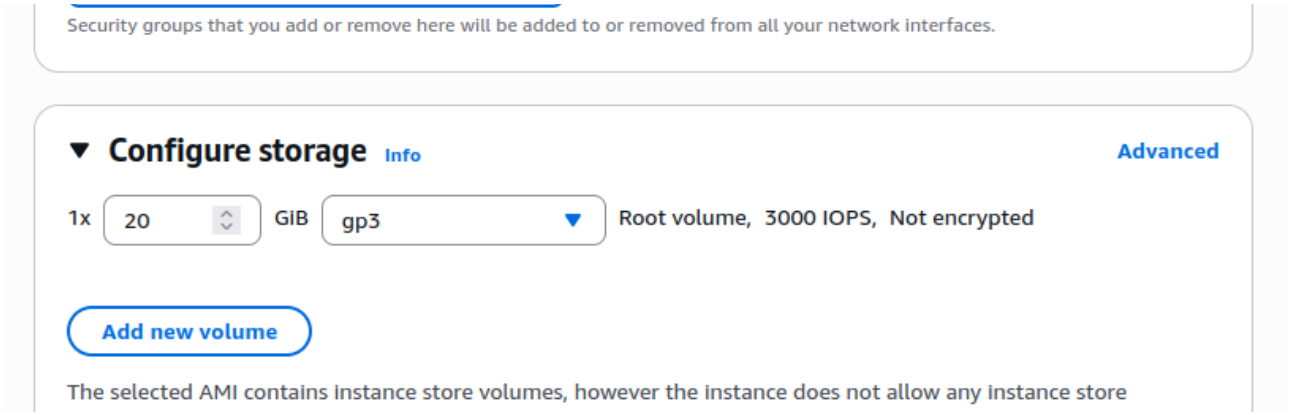
For key pair select our previous key pair. This is for ease so that we do not have maintain multiple keys. As always not recommended for production



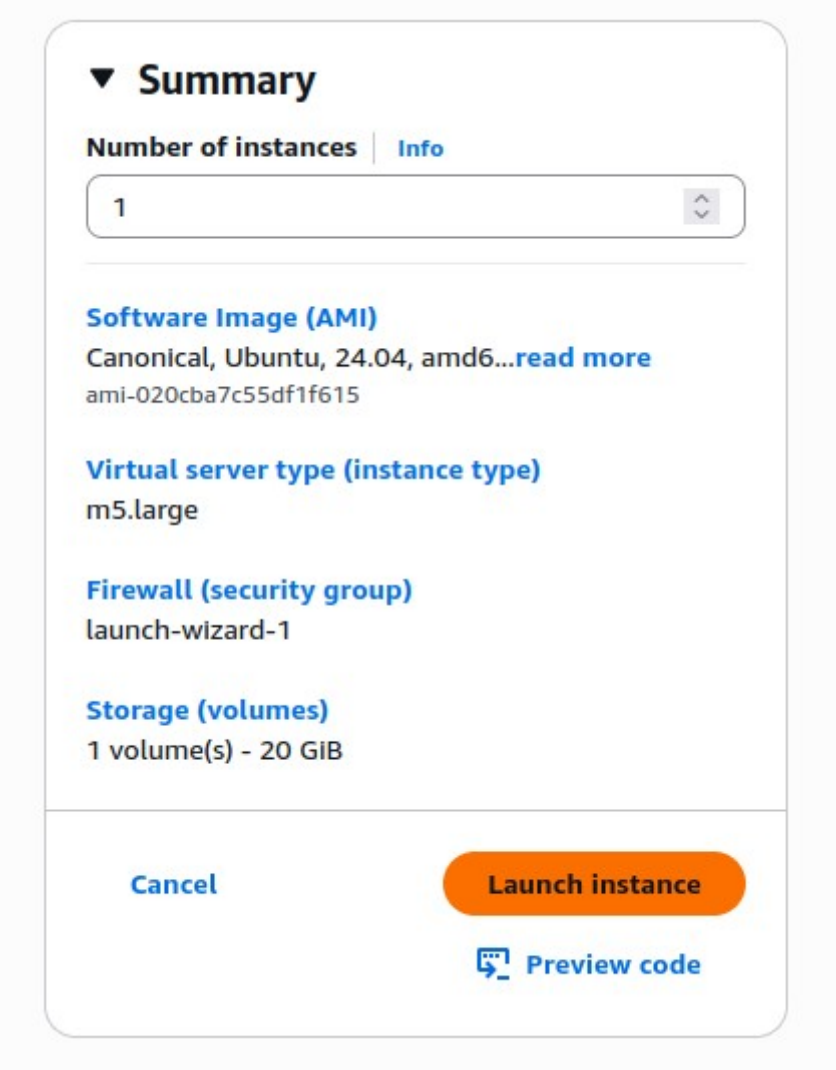
For networking and security selecting existing group as well



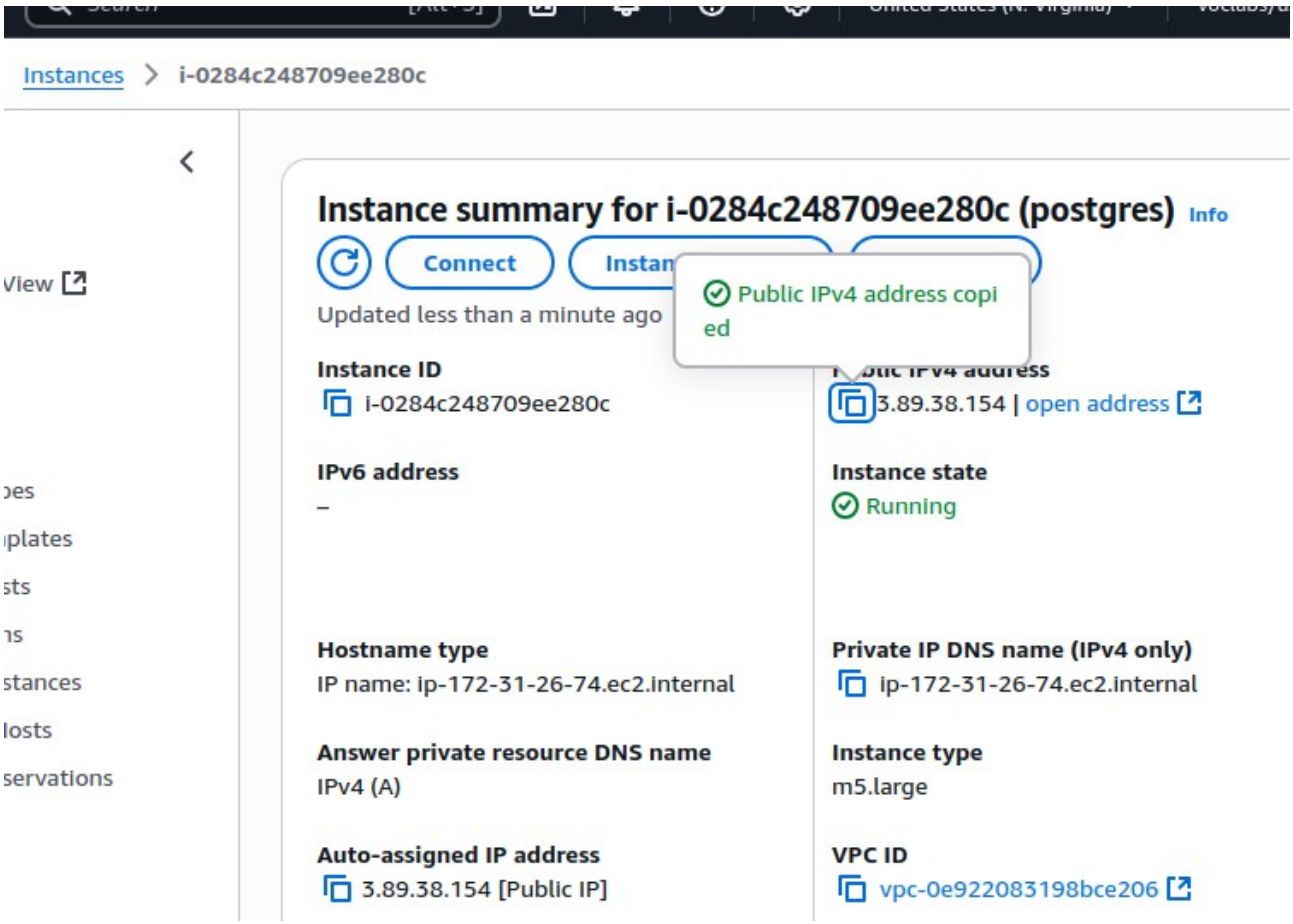
For storage select 20gb



Then in the summary section select Launch Instance



After the instance has been created copy its public ip and ssh into it



Use the same key and command as before.

```
ardent@ardent:~/Desktop$ ssh -i pyspark.pem ubuntu@3.89.38.154
The authenticity of host '3.89.38.154 (3.89.38.154)' can't be established.
ED25519 key fingerprint is SHA256:eSpBB7a+yagW+DktbPTWZHcpQZNEapXnWc58V0coC6M.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '3.89.38.154' (ED25519) to the list of known hosts.
Welcome to Ubuntu 24.04.2 LTS (GNU/Linux 6.8.0-1029-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

 System information as of Sun Aug  3 06:45:27 UTC 2025

  System load:  0.02              Temperature:           -273.1 C
  Usage of /:   9.3% of 18.33GB   Processes:             118
  Memory usage: 3%                Users logged in:       0
```

Do the initial update and upgrade as before

```
See "man sudo_root" for details.

ubuntu@ip-172-31-26-74:~$ sudo apt update
Hit:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble InRelease
```

```
98 packages can be upgraded. Run 'apt list --upgradable' to see them.
ubuntu@ip-172-31-26-74:~$ sudo apt upgrade
Reading package lists... Done
Building dependency tree... Done
```

Now let us install postgresql in this instance/server/node. The process will be exactly same as we did for our local machine.

Run the command to install postgresql

```
ubuntu@ip-172-31-26-74:~$ sudo apt install postgresql
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  libcommon-sense-perl libjson-perl libjson-xs-perl libllvm17t64 libpq5 libtypes-serialiser-
  postgresql-client-16 postgresql-client-common postgresql-common ssl-cert
Suggested packages:
  postgresql-doc postgresql-doc-16
The following NEW packages will be installed:
  libcommon-sense-perl libjson-perl libjson-xs-perl libllvm17t64 libpq5 libtypes-serialiser-
  postgresql-16 postgresql-client-16 postgresql-client-common postgresql-common ssl-cert
0 upgraded, 12 newly installed, 0 to remove and 0 not upgraded.
Need to get 43.6 MB of archives.
After this operation, 175 MB of additional disk space will be used.
Do you want to continue? [Y/n]
```

Press Y when prompted

Change the password of postgres user as we did before

```
ubuntu@ip-172-31-26-74:~$ sudo -i -u postgres
postgres@ip-172-31-26-74:~$ psql
psql (16.9 (Ubuntu 16.9-0ubuntu0.24.04.1))
Type "help" for help.

postgres=# alter user postgres with password 'postgres';
ALTER ROLE
postgres=# \q
postgres@ip-172-31-26-74:~$ exit
logout
ubuntu@ip-172-31-26-74:~$
```

We will need to make certain changes to our config file which will be different from our local setup.

Open the config file in text editor of your choice

```
logout
ubuntu@ip-172-31-26-74:~$ sudo vim /etc/postgresql/16/main/postgresql.conf
```

sudo vim /etc/postgresql/16/main/postgresql.conf
Replace vim with nano (or any other editor) if you are comfortable with it

```
# - Connection Settings -

#listen_addresses = 'localhost'         # what IP address(es) to listen on;
                                        # comma-separated list of addresses;
                                        # defaults to 'localhost'; use '*' for all
```

Uncomment the listen_address parameter and replace ‘localhost’ with ‘*’ as we want all ip’s to be able to access this database

```
# - Connection Settings -

listen_addresses = '*'          # what IP address(es) to listen on;
                                        # comma-separated list of addresses;
                                        # defaults to 'localhost'; use '*' for all
```

Save and exit. Now it is time to update the pg_hba.conf file like we did earlier

```
ubuntu@ip-172-31-26-74:~$ sudo vim /etc/postgresql/16/main/pg_hba.conf
```

sudo vim /etc/postgresql/16/main/pg_hba.conf

Move down to the connection settings. You will find the below config

```
# Database administrative login by Unix domain socket
local   all             postgres                                peer

# TYPE  DATABASE        USER            ADDRESS                 METHOD

# "local" is for Unix domain socket connections only
local   all             all                                     peer
# IPv4 local connections:
host    all             all             127.0.0.1/32            scram-sha-256
# IPv6 local connections:
host    all             all             ::1/128                 scram-sha-256
# Allow replication connections from localhost, by a user with the
```

Change the peer and scram to md5.Also for host change the ip to 0.0.0.0/0 to support all ip’s

```
# Database administrative login by Unix domain socket
local   all             postgres                                md5

# TYPE  DATABASE        USER            ADDRESS                 METHOD

# "local" is for Unix domain socket connections only
local   all             all                                     md5
# IPv4 local connections:
host    all             all             0.0.0.0/0               md5
# IPv6 local connections:
host    all             all             ::1/128                 md5
# Allow replication connections from localhost, by a user with the
```

Save and exit the file

Restart the postgresql service using below command.

```
buntu@ip-172-31-26-74:~$ sudo vim /etc/postgresql/16/main/pg_hba.conf
buntu@ip-172-31-26-74:~$ sudo systemctl restart postgresql
```

sudo systemctl restart postgresql

Let us try to connect from the system itself.

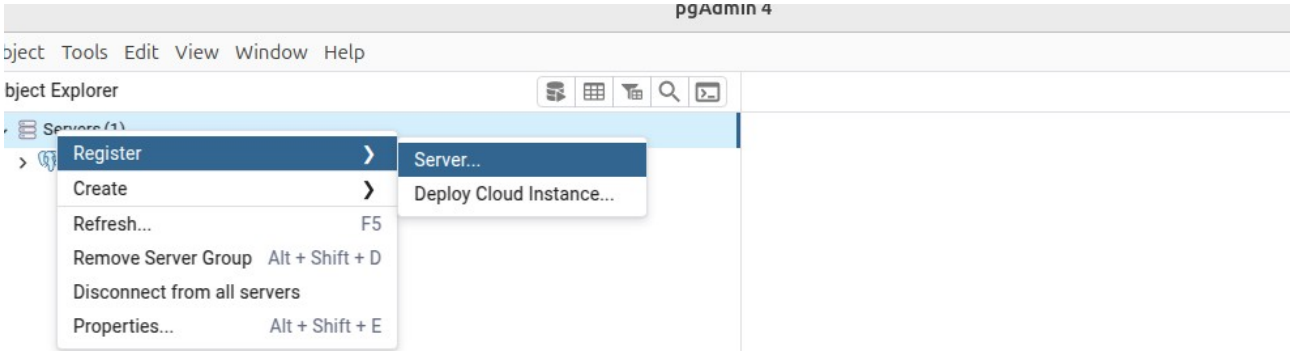
```
ubuntu@ip-172-31-26-74:~$ sudo systemctl restart postgresql
ubuntu@ip-172-31-26-74:~$ psql -U postgres
Password for user postgres:
psql (16.9 (Ubuntu 16.9-0ubuntu0.24.04.1))
Type "help" for help.

postgres=#
```

psql -U postgres

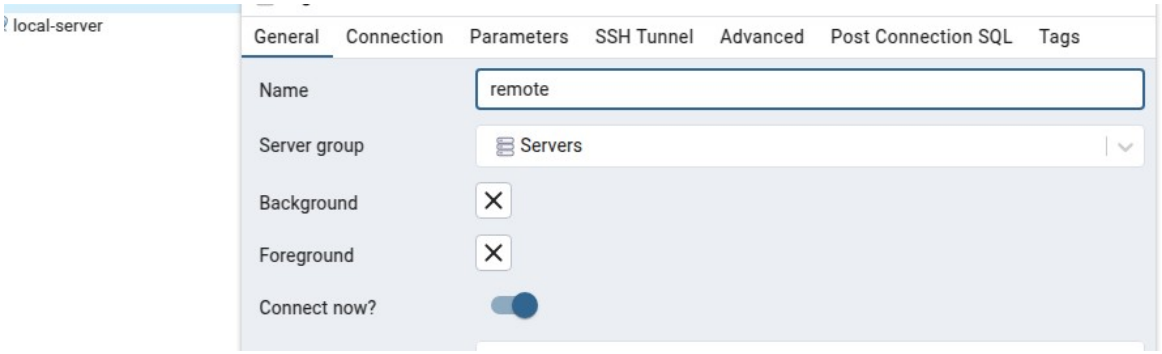
If it worked succesfully let us try from our own/local system pgadmin. If it did not succeed debug on your own.

Open pgadmin on your local machine

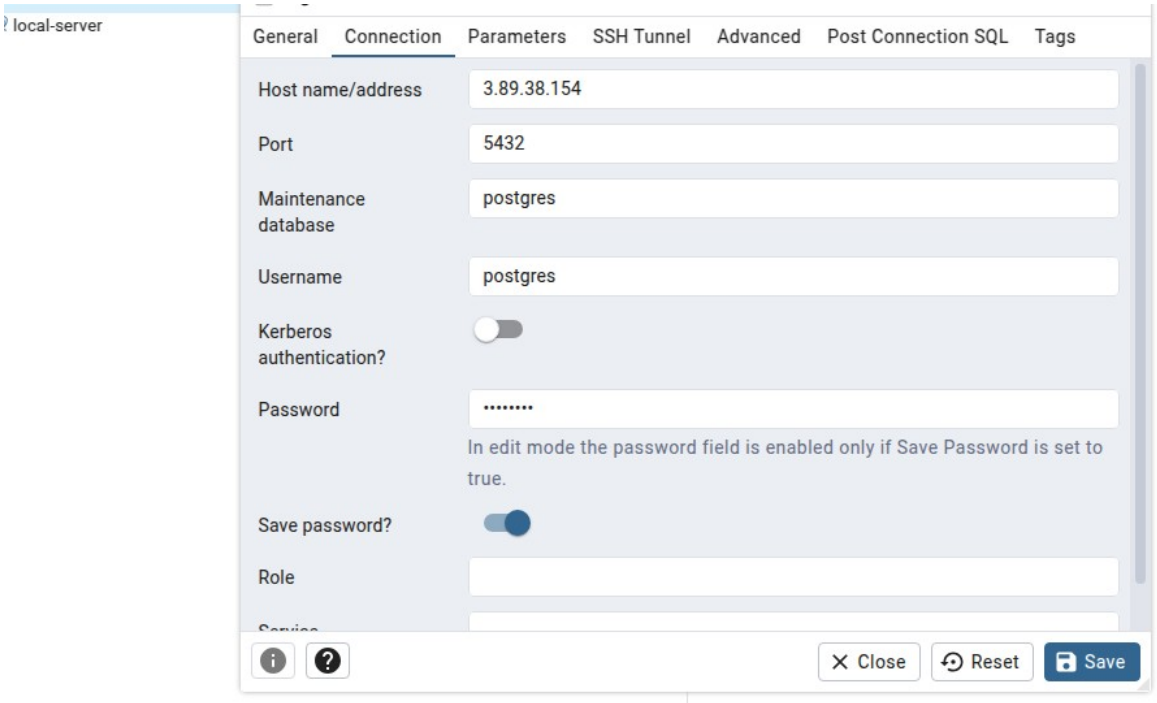
In the server right click and select register and then server



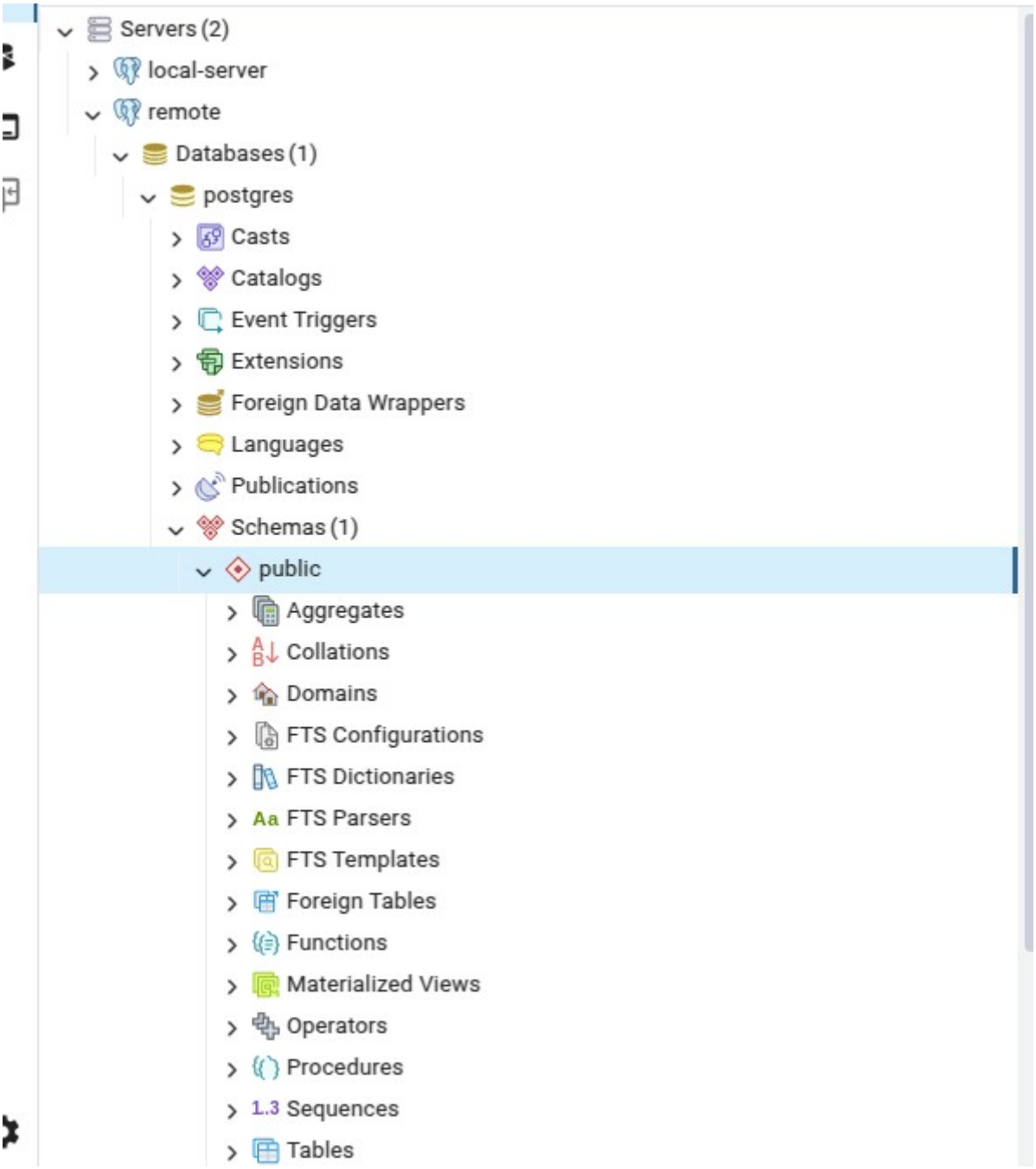
Give it a name. I have given remote to distinguish our local and remote ec2 instance based postgres



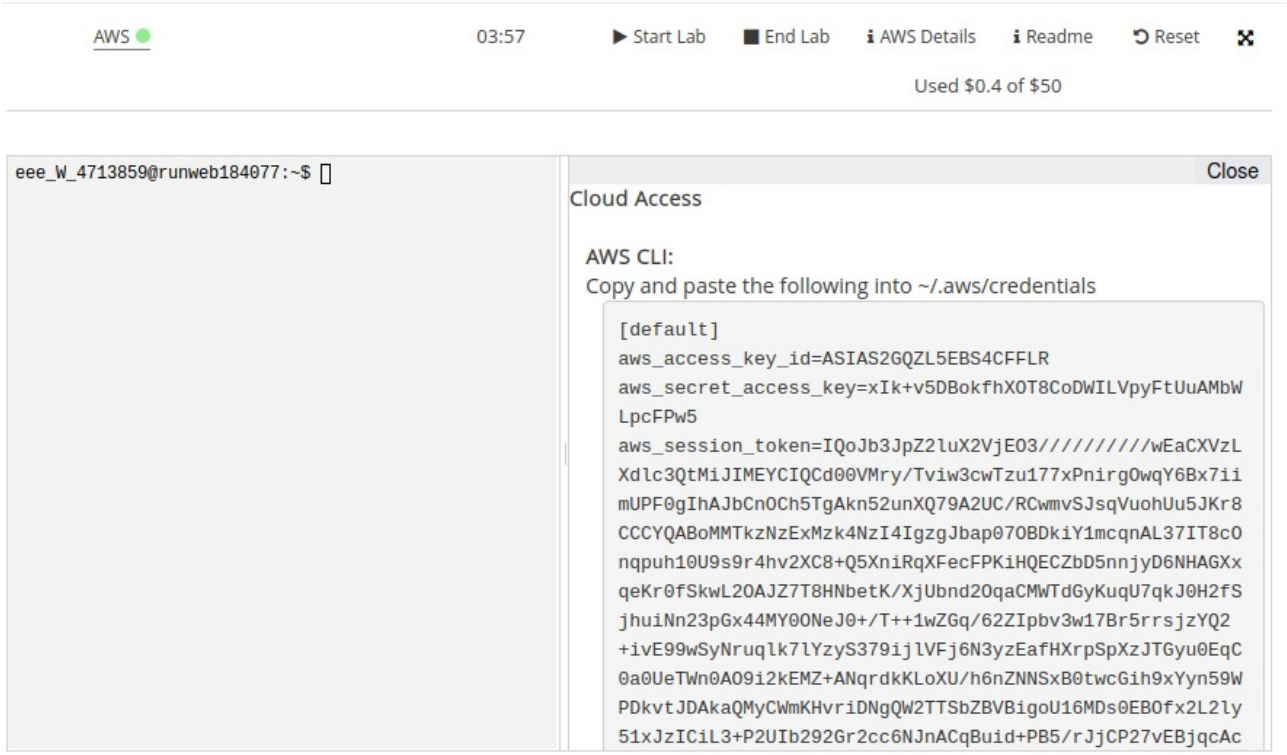
In the connection tab. For host provide the public ip of the instance. The port, database and username should be default if you did not make any changes yourself. For password entered the password you created and click save.



Remote should be added and now you should be able to view the database instance. Note that currently we do not have any table but we have verified that our database instance has been created and it can be connected from anywhere.

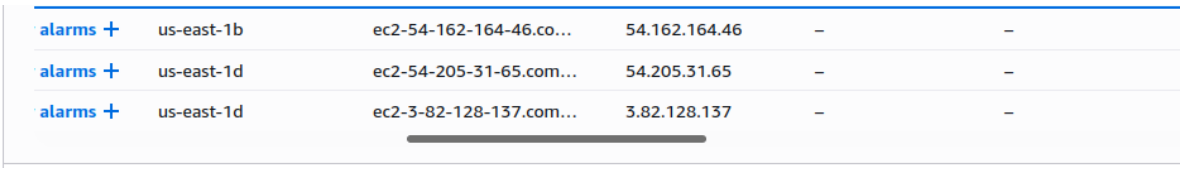


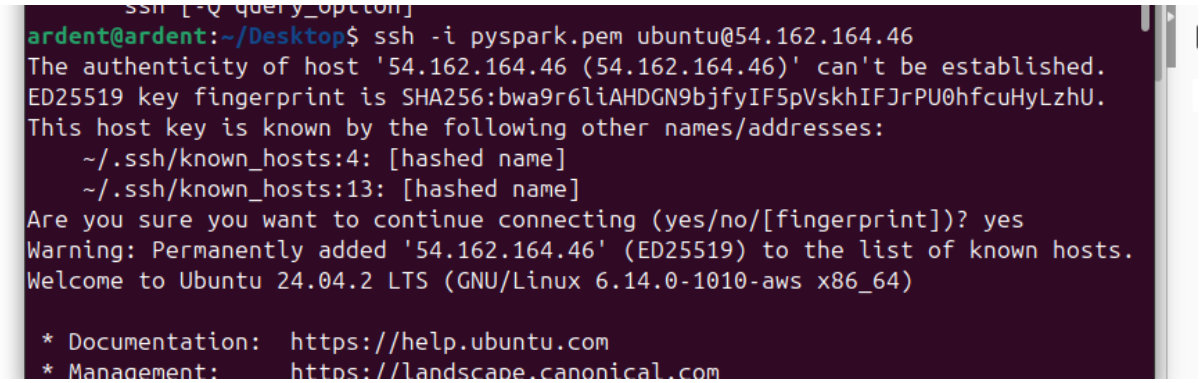
Go to AWS Details and copy the parameters available there.



This is not the case in production code. The access key id and secret key is usually same across different sessions. But since we are using learner lab we will have to get these key and setup in our .bashrc file every time we start our lab anew.

If you had stopped the spark based ec2 instances earlier then start them again. We will work on it now. Get the public ip of your spark based instances and ssh into your instance. You will have to do this for all 3 instances.





Then update the 3 values of aws_keys in your .bashrc file

```
fi

export AWS_ACCESS_KEY_ID=ASIAS2GQZL5EBS4CFFLR
export AWS_SECRET_ACCESS_KEY=xIk+v5DBokfhXOT8CoDWILVpyFtUuAMbWLpcFPw5
export AWS_SESSION_TOKEN=IQoJb3JpZ2luX2VjEO3//////////wEaCXVzLXdlc3QtMiJIMEYCIQCd00VMry/Tviw3cwTzu177xPnirgOwqY6Bx7iimUP
F0gIhAJbCnOCh5TgAkn52unXQ79A2UC/RCwmvSJsqVuohUu5JKr8CCCYQABoMMTkzNzExMzk4NzI4IgzgJbap07OBDkiY1mcqnAL37IT8cOnqpuh10U9s9r4
hv2XC8+Q5XniRqXFecFPKiHQECZbD5nnjyD6NHAGXxqeKr0fSkwL2OAJZ7T8HNbetK/XjUbnd2OqaCMWTdGyKuqU7qkJ0H2fSjhuiNn23pGx44MY0ONeJ0+/
T++1wZGq/62ZIpbv3w17Br5rrsjzYQ2+ivE99wSyNruqlk7lYzyS379ijlVFj6N3yzEafHXrpSpXzJTGyu0EqC0a0UeTWn0AO9i2kEMZ+ANqrdkKLoXU/h6n
ZNNSxB0twcGih9xYyn59WPDkvtJDAkaQMyCWmKHvriDNgQW2TTSbZBVBigoU16MDs0EBOfx2L2ly51xJzICiL3+P2UIb292Gr2cc6NJnACqBuid+PB5/rJjC
P27vEBjqcAcBIrcpnUt/RJDT3R7T/yfbDpMgSMcUlqtCB4owDB+GCXZHLtRGsQsOybsNoxOdqcCatJKbXCNBIagSiMsWR2KF91BzhctEwlgdZVgPx/RfSod/
2KBUJIY396/ooy8NEE6tU2mU7qfmamPE9FvgB0sl6frkJiMLG8otnjhkix0BK/1qXmbbKOpfocZhu4tFrYWl1ANg6BD06FJx7ag==
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

-- INSERT --                                                                    121,8         Bot
```

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

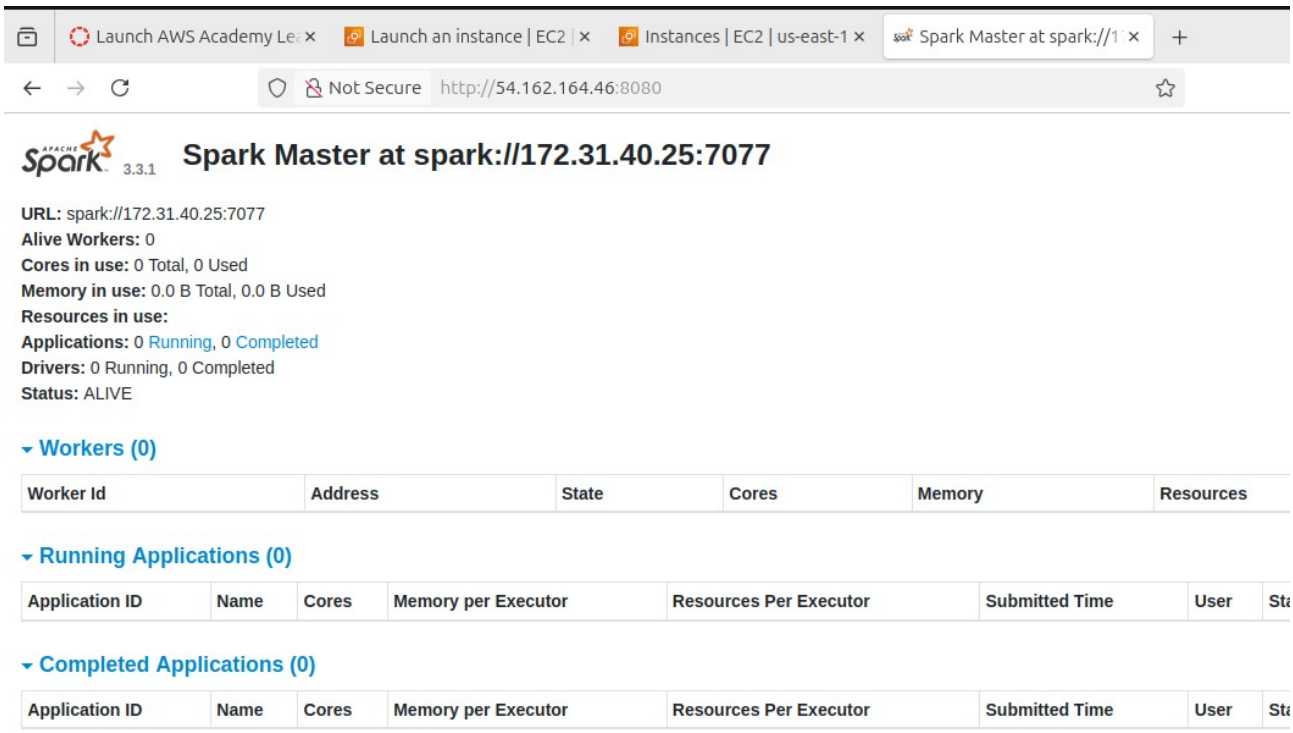
Make sure you run source .bashrc as always to update the change

Do the above two steps for all 3 instances.

On the master node run the script to run master

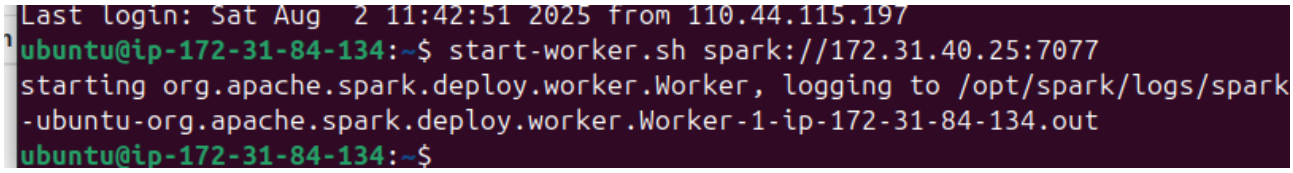
```
Last login: Sun Aug  3 05:04:37 2025 from 110.44.115.197
ubuntu@ip-172-31-40-25:~$ start-master.sh
starting org.apache.spark.deploy.master.Master, logging to /opt/spark/logs/spark
-ubuntu-org.apache.spark.deploy.master.Master-1-ip-172-31-40-25.out
ubuntu@ip-172-31-40-25:~$
```

Verify that it is running by visiting the webui master_ip:8080



Now let us add workers by going to ssh instance of slave and running start-worker script start-worker.sh spark://master_private_ip:7077

Private ip of master is always the same so this could be automatized later



Note: we use public ip to ssh because our local system is not in the same network as our instances. But these three isntaces are in the same network so they can communicate using the private ip as well as public ip

Run the above script it next worker/slave instance as well.

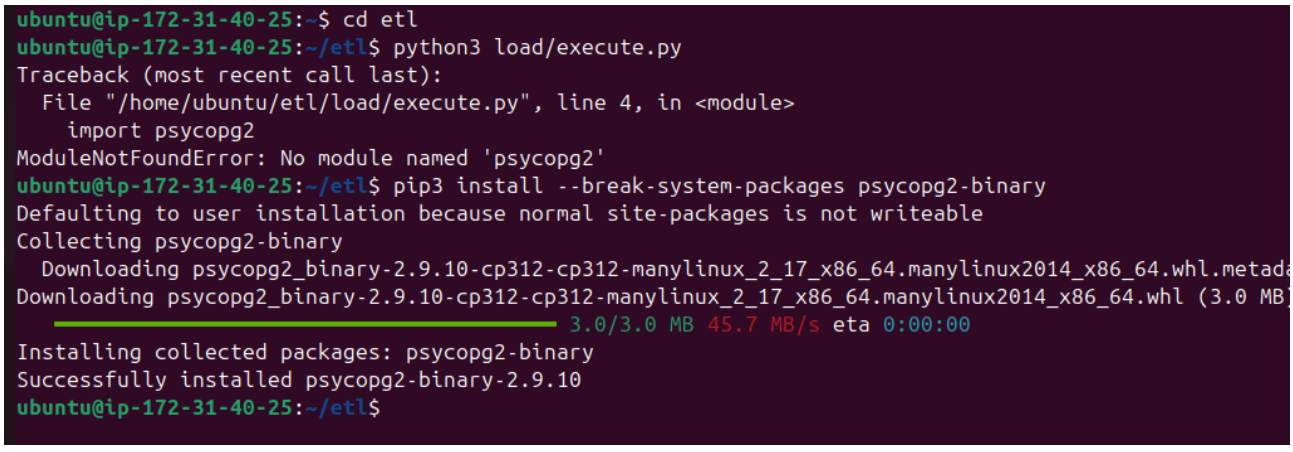
If you visit the webui after activating both the workers. It should be showing up

Note in the above image we can also see the available cores and memory for that particular worker.

Now, that everything is setup let us run our load script. That reads from s3 and loads into our newly created postgresql database.

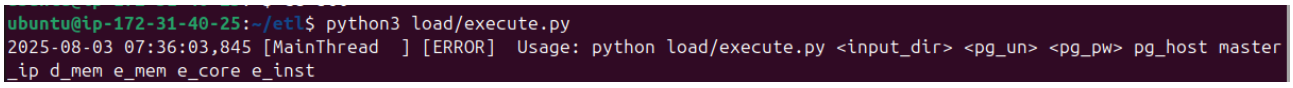
For this go to the ssh session of master instance.

If we cd into etl and try to run the load code we notice that we do not have pyscopg2 installed in the master server. Lets install that first



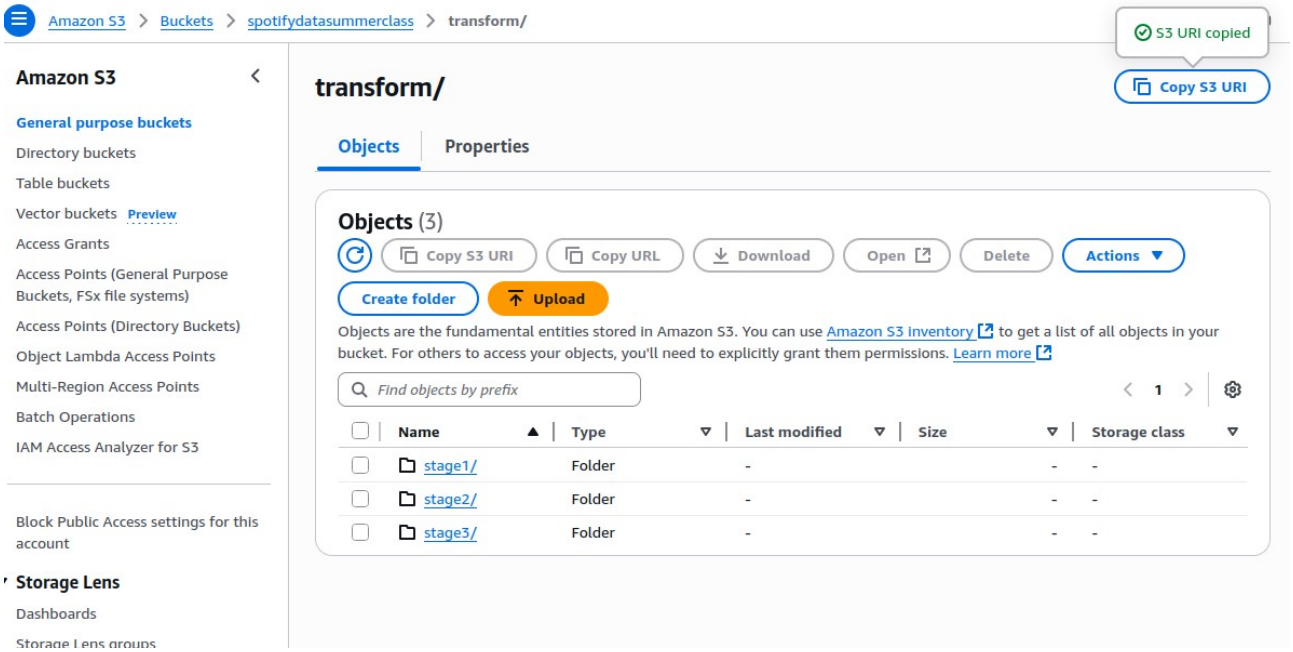
pip3 install –break-system-packages psycopg2-binary

Trying to run the code without any parameter throws an error as it should



We will be needing several inputs. input_dir in our case the the s3 uri for our transform. The pg_un and pg_pw is the username and password we created earlier. pg_host is the public ip of the ec2 instance we created earlier and installed postgresql in. master_ip is the public ip of the master. Other parameters are driver memory, executor memory, executor core, and executor instances

Let us get our s3 uri



In my case it is: s3://spotifydatasummerclass/transform/ I will add a to the uri as pyspark needs in that format

s3a://spotifydatasummerclass/transform/

The required public ips can be retrieved from instances page. After gathering required information run the code again by passing them as arguments.

```
ubuntu@ip-172-31-40-25:~/etl$ python3 load/execute.py s3a://spotifydatasummerclass/transform/ postgres postgres 3.89.38
154 54.162.164.46 2g 2700m 2 2
2025-08-03 07:44:29,691 [MainThread  ] [INFO ]  Load stage started
```

Now, the tables will be created but the data won't be transferred to postgres server using spark. If you have closely followed the manual from the beginning of the course till now you should have the idea why that is happening.

```
ubuntu@ip-172-31-40-25:~/etl$ python3 load/execute.py s3a://spotifydatasummerclass/transform/ postgres postgres 3.89.38.
154 54.162.164.46 2g 2700m 2 2
2025-08-03 07:53:24,470 [MainThread  ] [INFO ]  Load stage started
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
25/08/03 07:53:29 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java cl
asses where applicable
2025-08-03 07:53:32,866 [MainThread  ] [INFO ]  PostgreSQL tables created successfully
25/08/03 07:53:34 WARN MetricsConfig: Cannot locate configuration: tried hadoop-metrics2-s3a-file-system.properties,hado
op-metrics2.properties
2025-08-03 07:53:47,128 [MainThread  ] [WARNI]  Error loading master_table: An error occurred while calling o39.jdbc.
: java.lang.ClassNotFoundException: org.postgresql.Driver
        at java.base/java.net.URLClassLoader.findClass(URLClassLoader.java:476)
        at java.base/java.lang.ClassLoader.loadClass(ClassLoader.java:594)
        at java.base/java.lang.ClassLoader.loadClass(ClassLoader.java:527)
        at org.apache.spark.sql.execution.datasources.jdbc.DriverRegistry$.register(DriverRegistry.scala:46)
        at org.apache.spark.sql.execution.datasources.jdbc.JDBCOptions.$anonfun$driverClass$1(JDBCOptions.scala:101)
        at org.apache.spark.sql.execution.datasources.jdbc.JDBCOptions.$anonfun$driverClass$1$adapted(JDBCOptions.scala:
101)
        at scala.Option.foreach(Option.scala:407)
        at org.apache.spark.sql.execution.datasources.jdbc.JDBCOptions.<init>(JDBCOptions.scala:101)
```

If you go and check in your pgadmin remote postgres server. You will find that tables have been created but there are no data in the tables. This is because we have not added the jar file for postgres. Lets do that before we run the load code again.

Change directory to where sparks jars is located. Then use wget to download the jar file and change the directory back to our etl

```
2025-08-03 07:53:53,280 [MainThread  ] [INFO ]  Closing down clientserver connection
ubuntu@ip-172-31-40-25:~/etl$ cd /opt/spark/jars
ubuntu@ip-172-31-40-25:/opt/spark/jars$ wget https://jdbc.postgresql.org/download/postgresql-42.7.7.jar
--2025-08-03 07:57:04--  https://jdbc.postgresql.org/download/postgresql-42.7.7.jar
Resolving jdbc.postgresql.org (jdbc.postgresql.org)... 72.32.157.228, 2001:4800:3e1:1::228
Connecting to jdbc.postgresql.org (jdbc.postgresql.org)|72.32.157.228|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1098916 (1.0M) [application/java-archive]
Saving to: ‘postgresql-42.7.7.jar’

postgresql-42.7.7.jar         100%[=================================================>]   1.05M  5.17MB/s    in 0.2s

2025-08-03 07:57:05 (5.17 MB/s) - ‘postgresql-42.7.7.jar’ saved [1098916/1098916]

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

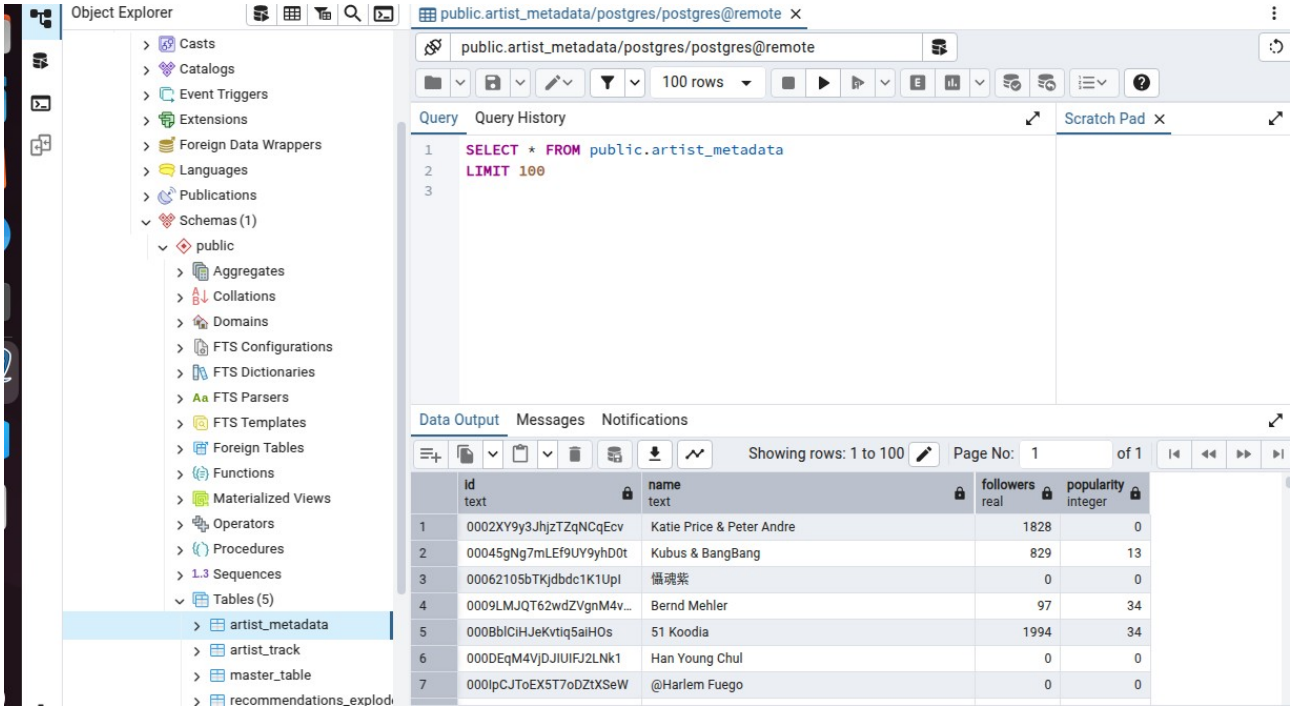
wget https://jdbc.postgresql.org/download/postgresql-42.7.7.jar

This needs to only be done in all 3 nodes

Now, let us try to run our spark code again from our master node

```
ubuntu@ip-172-31-40-25:~/etl$ python3 load/execute.py s3a://spotifydatasummerclass/transform/ postgres postgres 3.89.38.
154 54.162.164.46 2g 2700m 2 2
2025-08-03 08:09:34,023 [MainThread  ] [INFO ]  Load stage started
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
25/08/03 08:09:38 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java cl
asses where applicable
2025-08-03 08:09:42,095 [MainThread  ] [INFO ]  PostgreSQL tables created successfully
25/08/03 08:09:43 WARN MetricsConfig: Cannot locate configuration: tried hadoop-metrics2-s3a-file-system.properties,hado
op-metrics2.properties
2025-08-03 08:10:15,530 [MainThread  ] [INFO ]  Loaded master_table to PostgreSQL
2025-08-03 08:11:09,776 [MainThread  ] [INFO ]  Loaded recommendations_exploded to PostgreSQL
2025-08-03 08:11:15,015 [MainThread  ] [INFO ]  Loaded artist_track to PostgreSQL
2025-08-03 08:11:20,897 [MainThread  ] [INFO ]  Loaded track_metadata to PostgreSQL
2025-08-03 08:11:29,468 [MainThread  ] [INFO ]  Loaded artist_metadata to PostgreSQL
2025-08-03 08:11:29,469 [MainThread  ] [INFO ]  Load stage completed
2025-08-03 08:11:29,470 [MainThread  ] [INFO ]  Total time taken 0 hours, 1 minutes, 55 seconds
2025-08-03 08:11:29,470 [MainThread  ] [INFO ]  Closing down clientserver connection
ubuntu@ip-172-31-40-25:~/etl$
```

let us go to our pgadmin to verify that tables are populated with data



Congratulation! We have now loaded your data into remote postgres database using AWS cluster setup.

If the issue of driver persists we might need to stop and start the worker and slave nodes

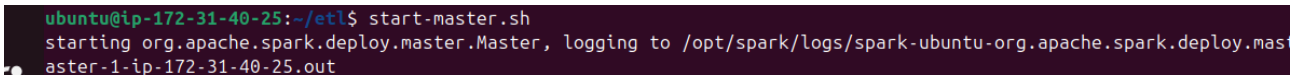
In the master node run stop-master.sh

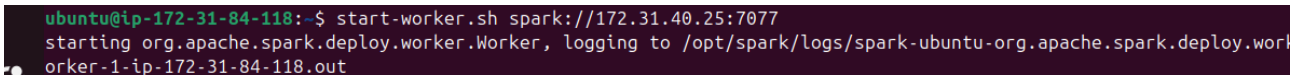
```
ubuntu@ip-172-31-40-25:~/etl$
ubuntu@ip-172-31-40-25:~/etl$
ubuntu@ip-172-31-40-25:~/etl$ stop-master.sh
stopping org.apache.spark.deploy.master.Master
ubuntu@ip-172-31-40-25:~/etl$
```

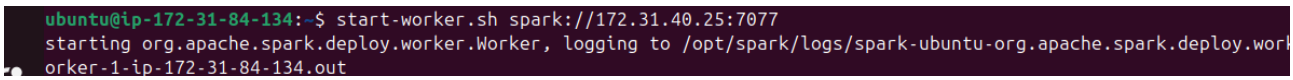
In the worker nodes run stop-worker.sh It might not be strictly necessary to stop the workers but for safetly let’s do it

Do this on both worker’s nodes

Now let us start all 3 nodes again







Verify in the web-ui that everything is working fine and master is connected to worker and run the above code