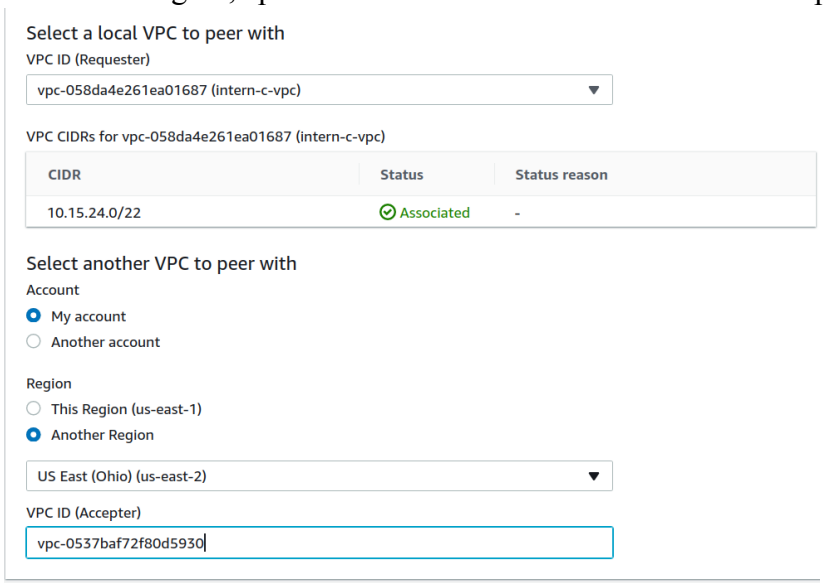
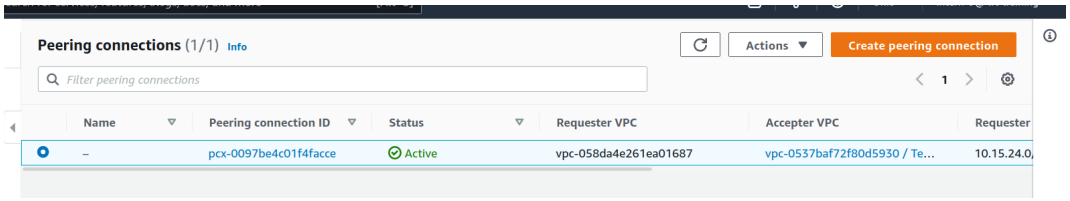
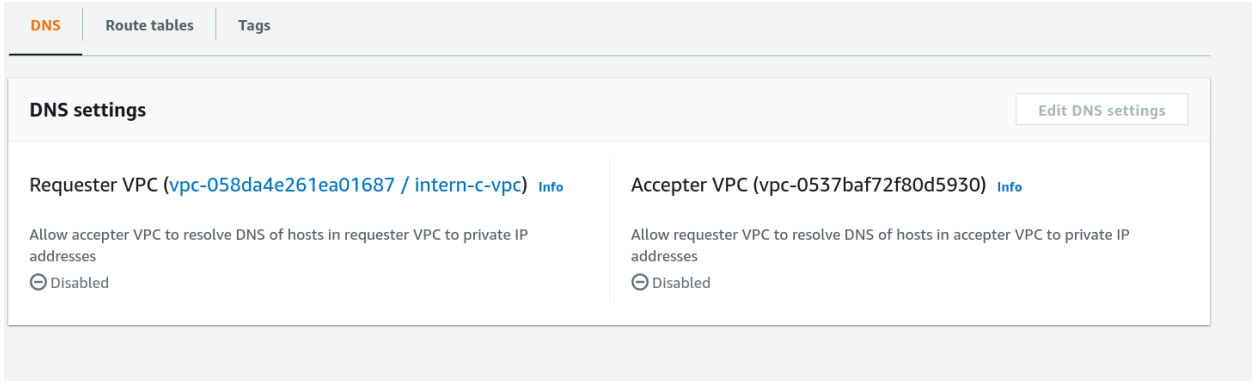
1. VPC peer from your team's VPC to the next team i.e. A->B, B->C, …, E->A.
    - Allow both directions VPC Traffic.
    - Add route to peered VPC in private Route Table.

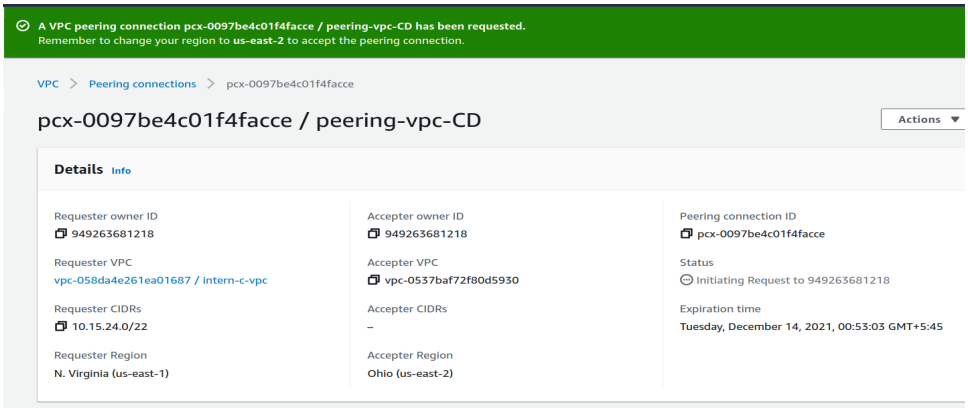
In intern C region, "peer connection" tab was selected and new peer connection was created:

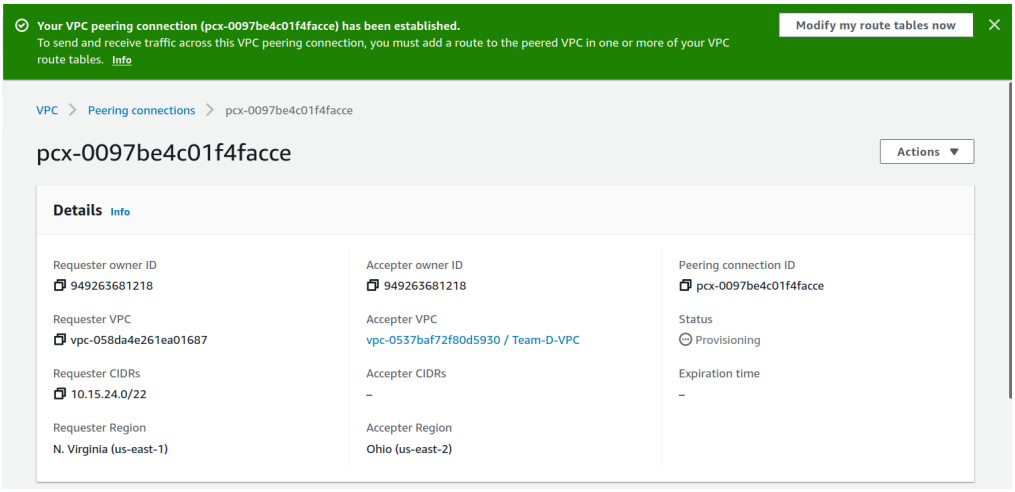




In intern-d region: goto to peer connection > action > Accept request

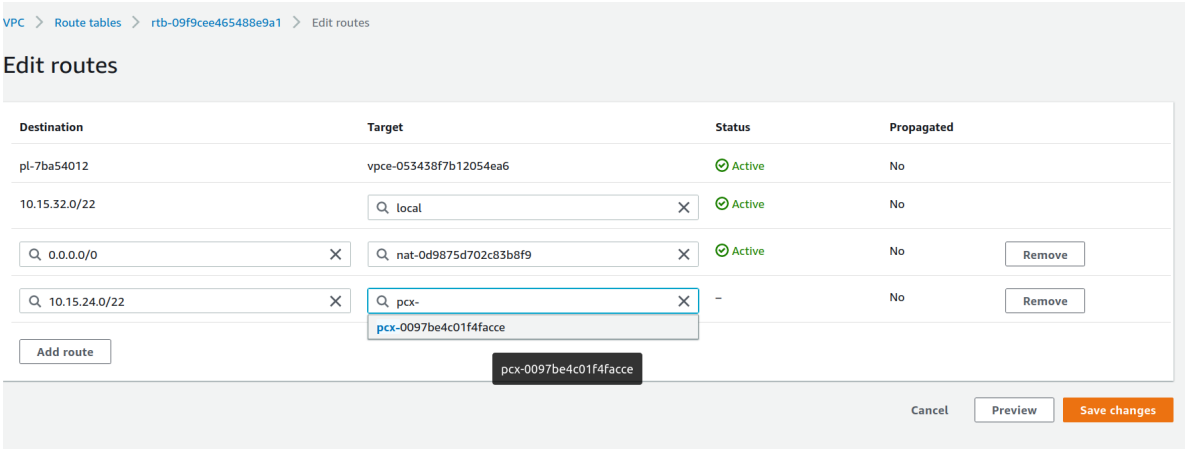


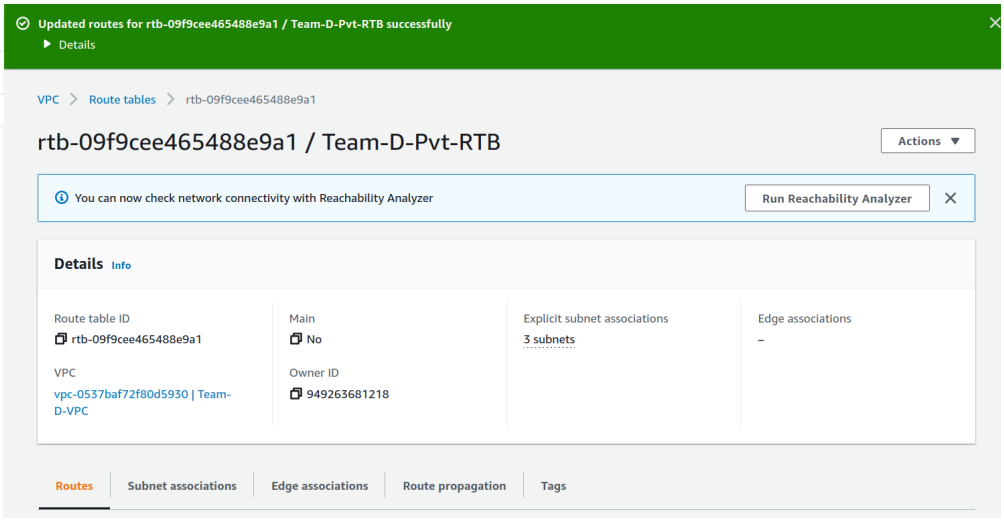




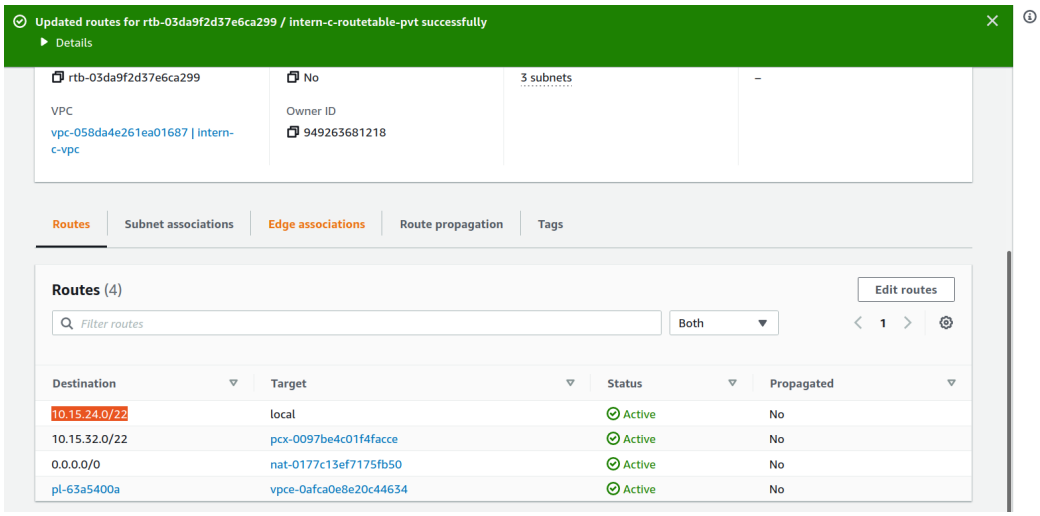
Add route to peered VPC in private Route Table.

Intern c > route table > route table private > route>edit
put the ip of intern-d privateroutetable and peerconnection-id





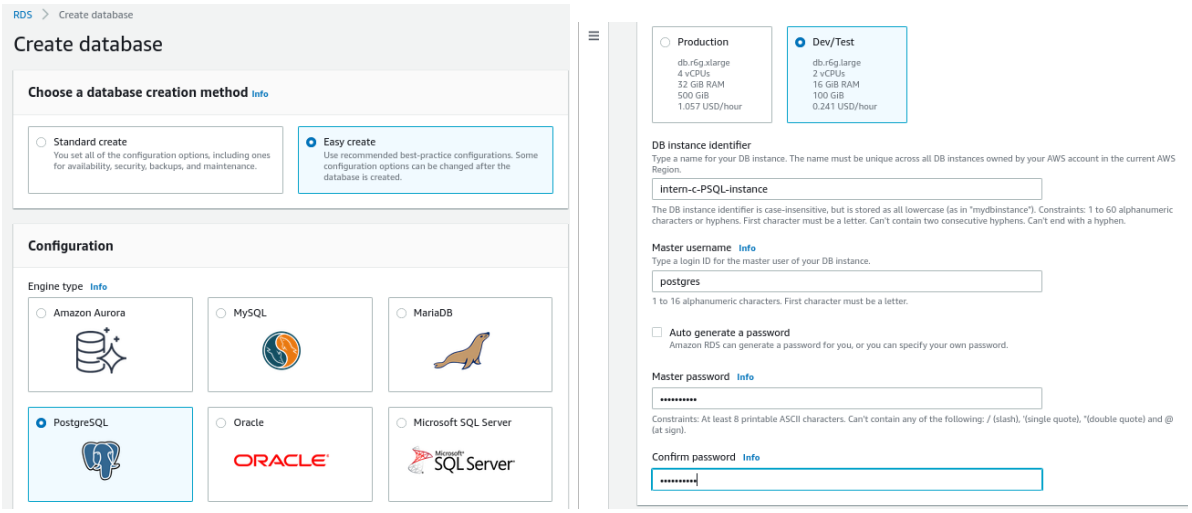
Do same as vice versa from internD



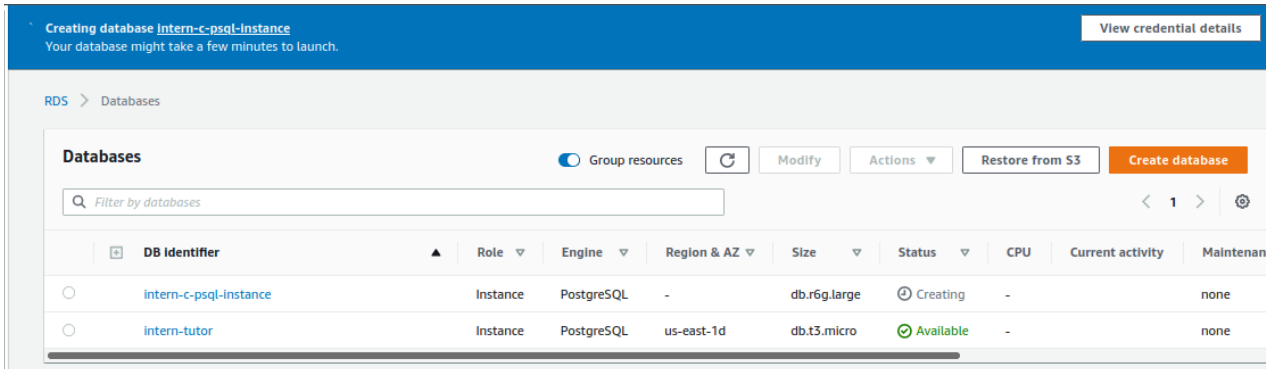
2. Create RDS cluster and instance of PostgreSQL.
    - Deploy in a private subnet.
    - In your private EC2. Connect to the RDS Postgresql.
    - (Optional) Validate you can connect to another team's (A->B and B->C) RDS (Peer must be completed)

RDS cluster was created by selecting Services>Database>RDS

And selected DB instances and created new instances of Postgresql.



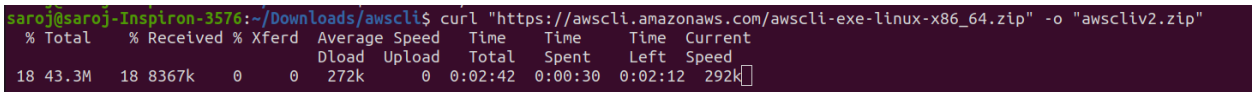
After providing an instance identifier name, username and password for our database, the database instance is finally created after a few minutes.



3. Install AWS CLIv2 and configure the lft-training profile.

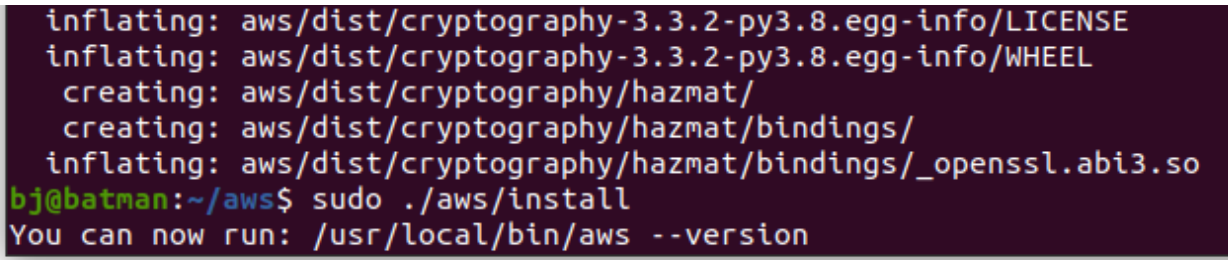
AWS CLI was installed in our host machine simply by using some commands:

curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"



unzip awscliv2.zip

sudo ./aws/install



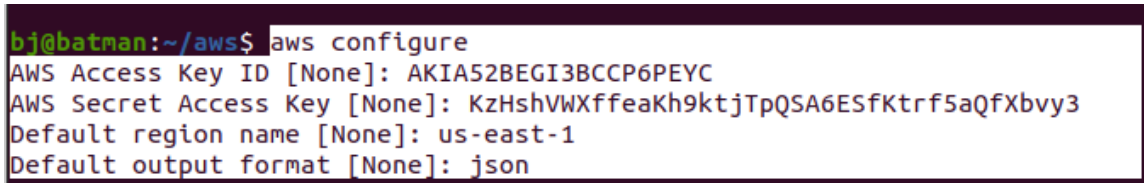
After aws is installed in our system, we can verify using command:

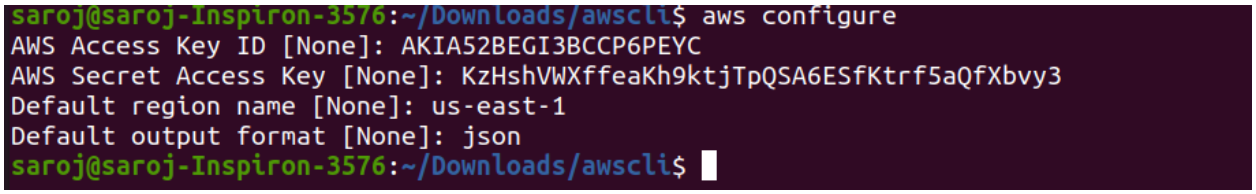
Aws --version

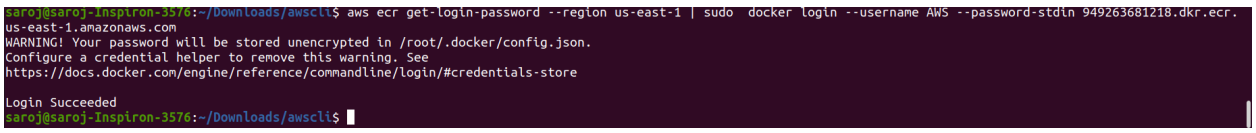
To configure the profile, we can start using these commands:

Aws configure

After entering our aws access key id and secret access key with region name and output format as json, we configured our profile for aws.







4. Create ECR and upload your Docker image created during Docker assignment Q3.
    - Each member must upload an image (in the team's ECR repo) with their name as tag.

Now, to login to our aws system:**aws ecr get-login-password --region us-east-1 | sudo docker login --username AWS --password-stdin 949263681218.dkr.ecr.us-east-1.amazonaws.com**

```
bj@batman:~/aws$ aws ecr get-login-password --region us-east-1 | sudo dock
er login --username AWS --password-stdin 949263681218.dkr.ecr.us-east-1.am
azonaws.com
WARNING! Your password will be stored unencrypted in /root/.docker/config.
json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-st
ore

Login Succeeded
```

And we can see the login has succeeded.

Now we pull docker images from our individual docker hub accounts which we have uploaded in a previous assignment of docker. All our team members did the same with our individual docker images from docker hub.

sudo docker pull bijaykandel37/nodedocker:nodedocker

sudo docker tag bijaykandel37/nodedocker:nodedocker 949263681218.dkr.ecr.us-east-1.amazonaws.com/intern-c:bijaykandel37

sudo docker push 949263681218.dkr.ecr.us-east-1.amazonaws.com/intern-c:bijaykandel37

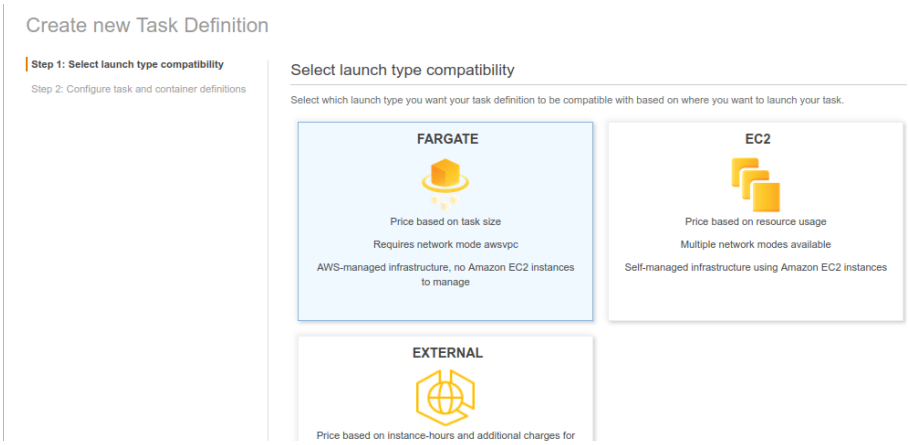
```
bj@batman:~/aws$ sudo docker pull bijaykandel37/nodedocker:nodedocker
nodedocker: Pulling from bijaykandel37/nodedocker
97518928ae5f: Pull complete
7001f79e6409: Pull complete
```

```
bj@batman:~/aws$ sudo docker tag bijaykandel37/nodedocker:nodedocker 94926
3681218.dkr.ecr.us-east-1.amazonaws.com/intern-c:bijaykandel37
bj@batman:~/aws$ docker push 949263681218.dkr.ecr.us-east-1.amazonaws.com/
intern-c:bijaykandel37
```

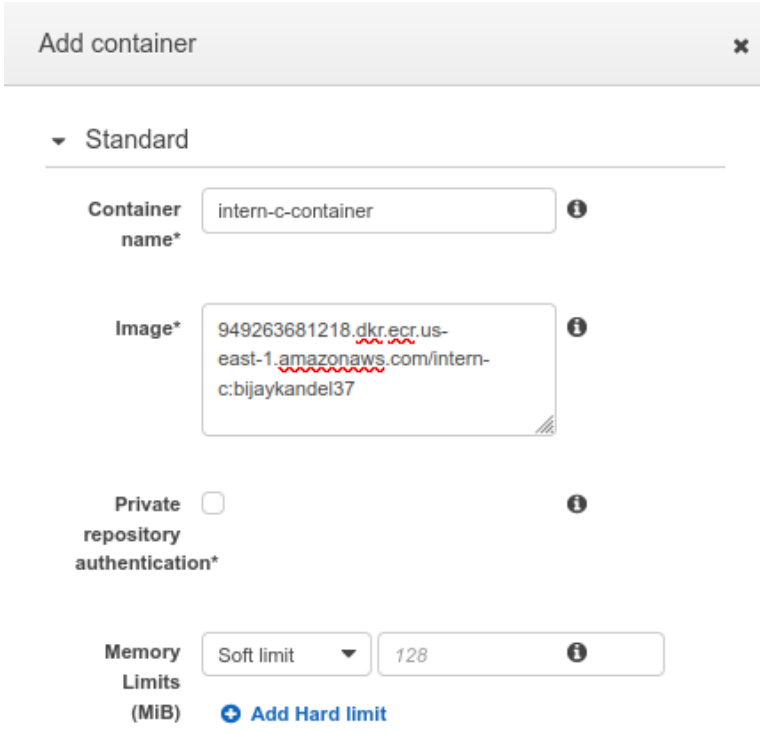
Image tag	Pushed at	Size (MB)	Image URI	Digest	Scan status	Vulnerabilities
bijaykandel37	December 07, 2021, 18:50:58 (UTC+05.75)	81.72	Copy URI	sha256:5407566781915c...	-	-
saroj	December 07, 2021, 11:57:09 (UTC+05.75)	56.73	Copy URI	sha256:b41876afcb9bfae...	-	-
prajesh	December 07, 2021, 11:55:56 (UTC+05.75)	50.69	Copy URI	sha256:6241e88cc42c62...	-	-

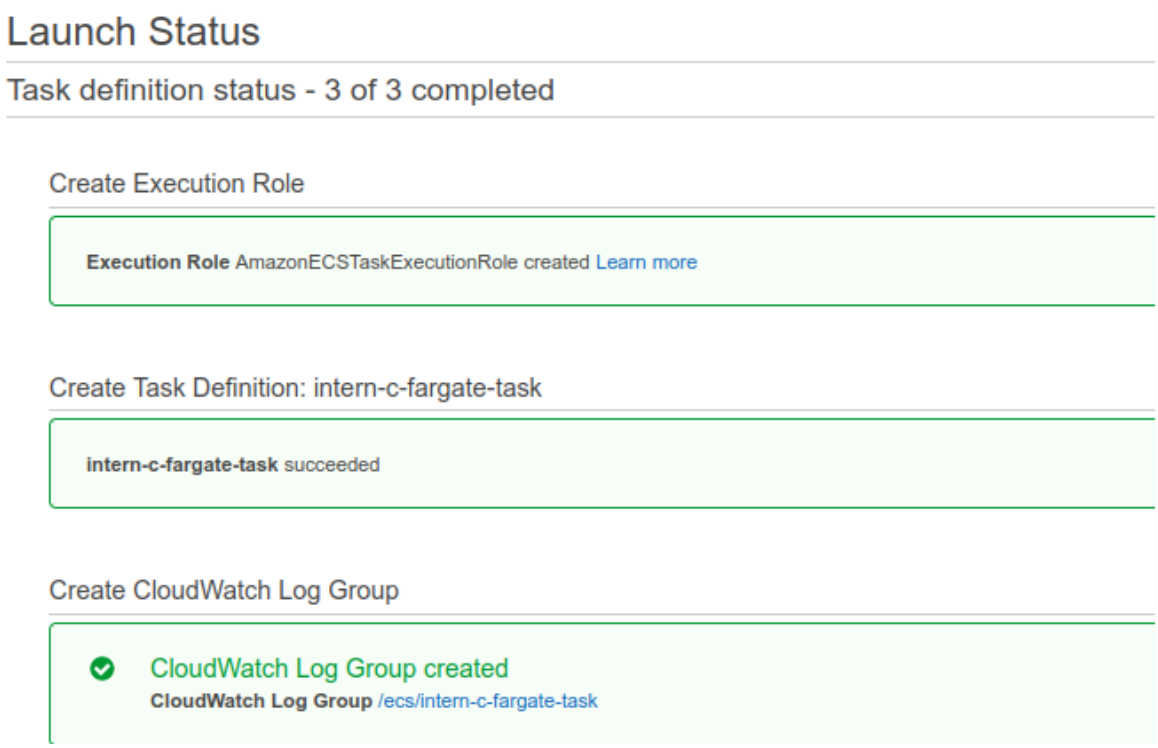
5. Deploy ***one*** of uploaded image in ECS Fargate.
    - Create task definition.
    - Deploy in Public Subnet.
    - Access via assigned Public IP and your port.

New task definition was created by selecting ECS and then Task Definitions:



And we added our container with a proper link to our deployed container.

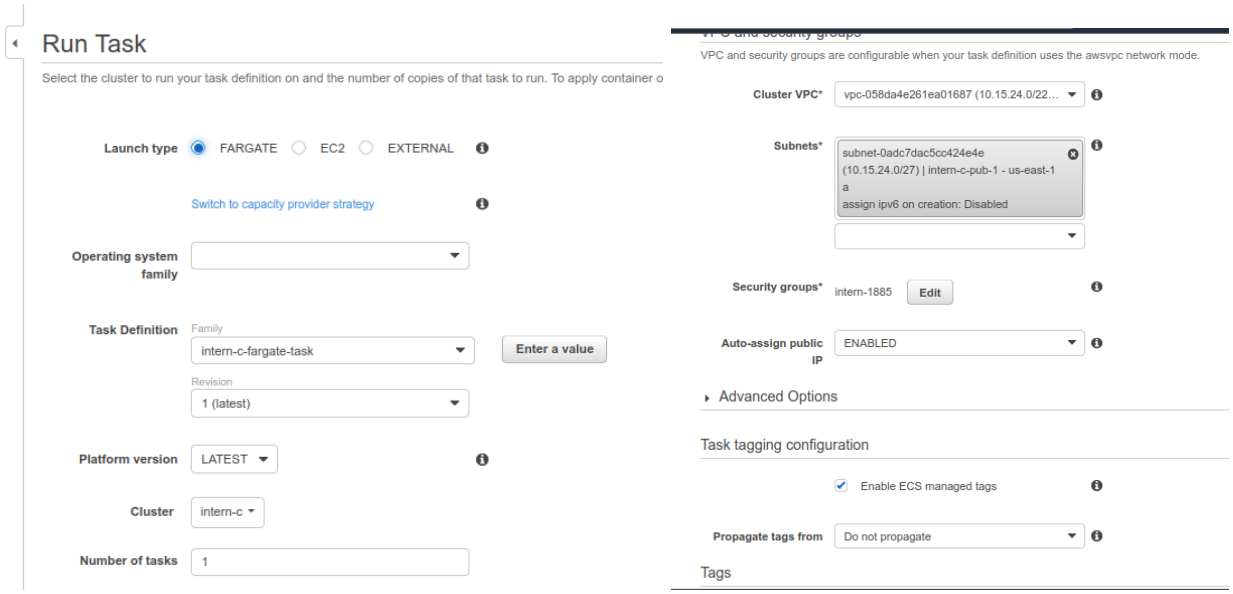




Now the task definition is created which can be seen by going to Tasks tab:

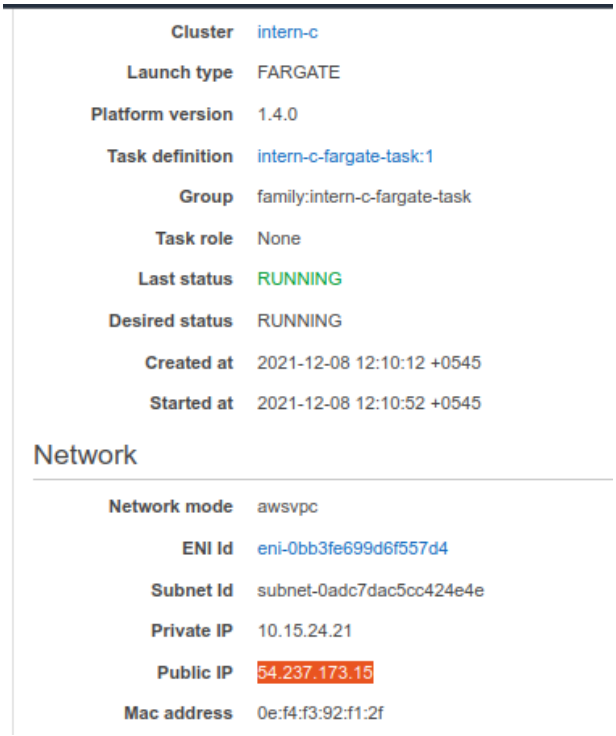
Task Definition	Latest revision status
intern-a	ACTIVE
intern-c-fargate-task	ACTIVE
intern-tutor	ACTIVE

To run the task, we selected the task and Actions>RunTask.

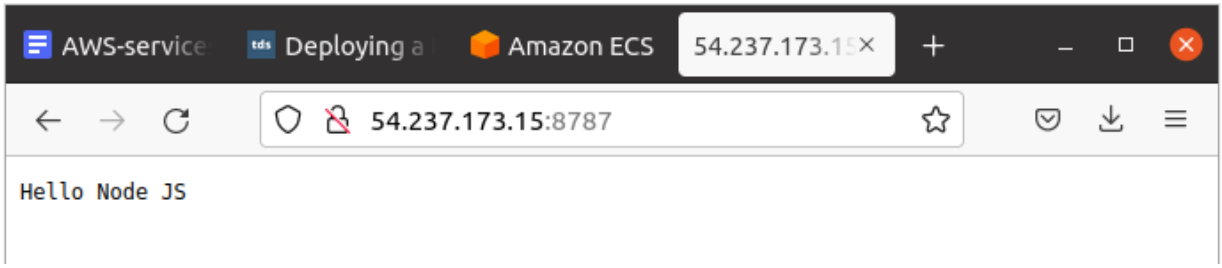


After providing all necessary details like Task definition name, cluster and cluster VPC. Also a Security Group was created for this specific app and we opened our exposed port which was 8787 in our case of the node container.

Now the task is running and we can get the public IP by going to Tasks tab and Under Network section, we can find public ip:



And in our browser, when we use the IP with our port address, the Node instance is LIVE and fully accessible on the internet.



6. Create a S3 bucket
    - Upload Dockerfile of Q4 in the bucket using AWS CLI.

Bucket was created by selecting S3 tab and then create bucket option on top right with appropriate information and Selecting Create bucket option at bottom.