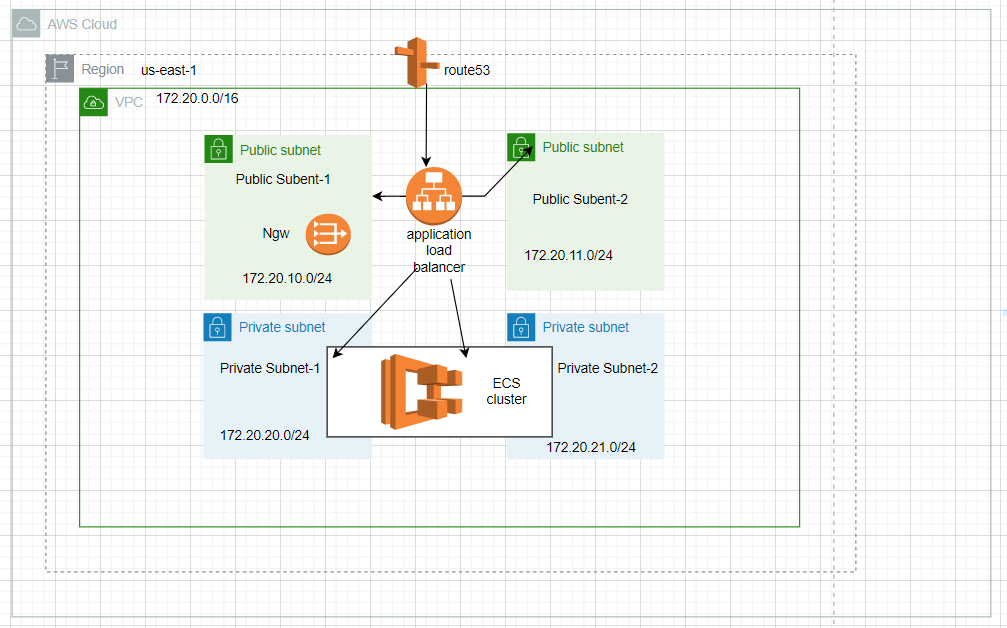
Architecture:

> To Create This Infrastructure we are using Terraform (Infrastructure as code) Tool.

**Prerequisite steps**:

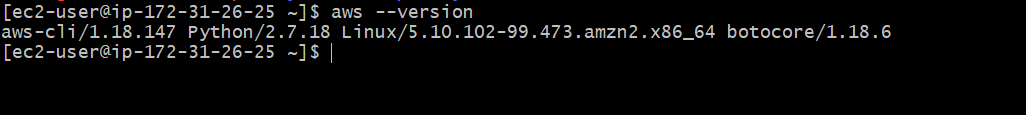
1. Make sure you have an AWS account.
2. Create aws linux instance(t2.micro),You also need an ssh key to connect to the EC2 instance that you are creating.
3. Connect to the instance using ssh key, Then install the AWS CLI

**AWS CLI**: The AWS Command Line Interface (CLI) is a unified tool to manage your AWS services. With just one tool to download and configure, you can control multiple AWS services from the command line and automate them through scripts.

For installation guidance for aws cli go through below link

[Installing or updating the latest version of the AWS CLI - AWS Command Line Interface (amazon.com)](https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html)

Confirm the installation by executing the following command.



**Configure the AWS CLI :**

> After Successful installation of aws cli, you have to configure the settings that AWS CLI uses to interact with the AWS.

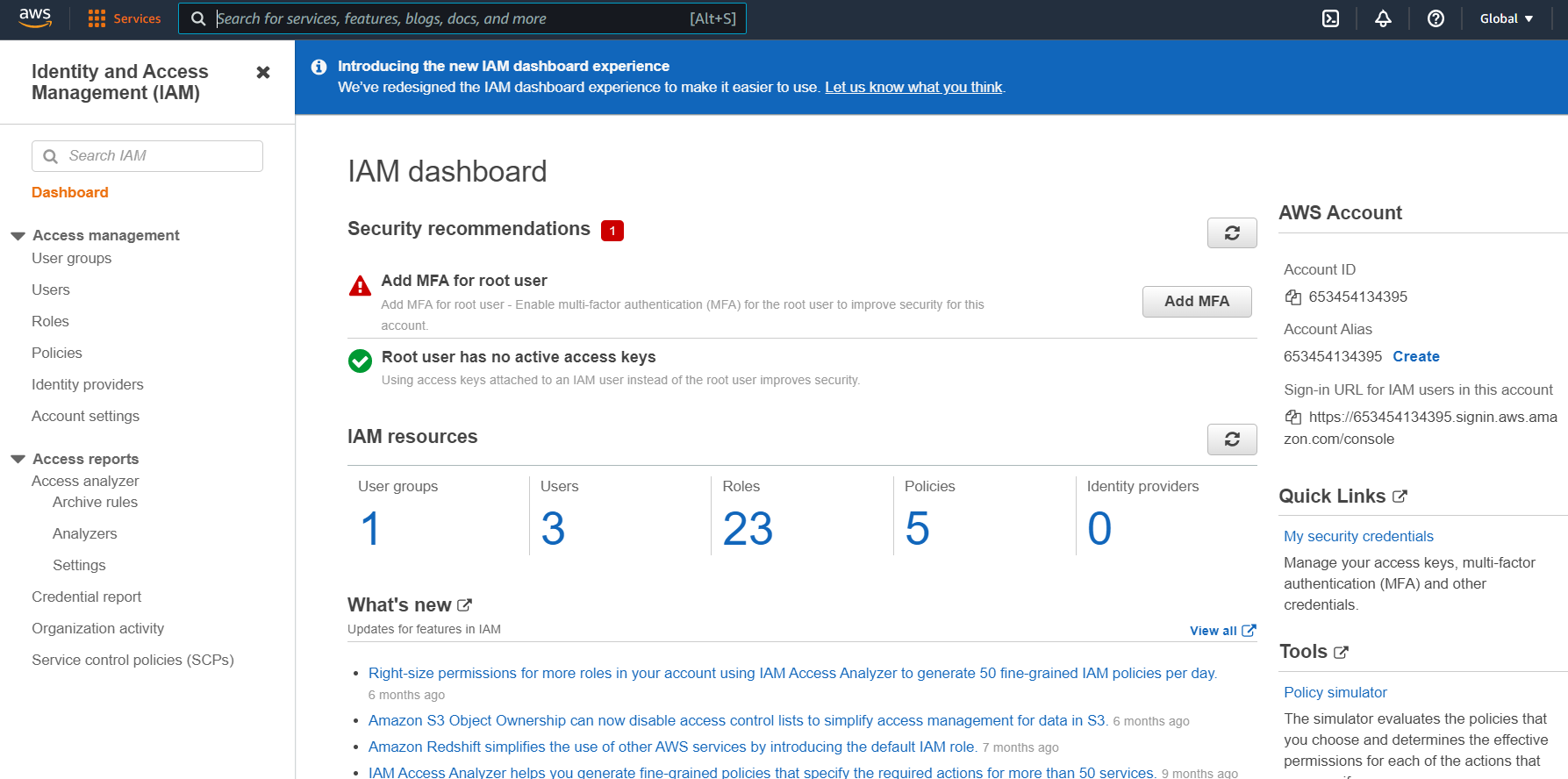
For general purpose, The aws configure command is the fastest way to set up your credentials. When you enter this command AWS cli prompts you for four pieces of information.

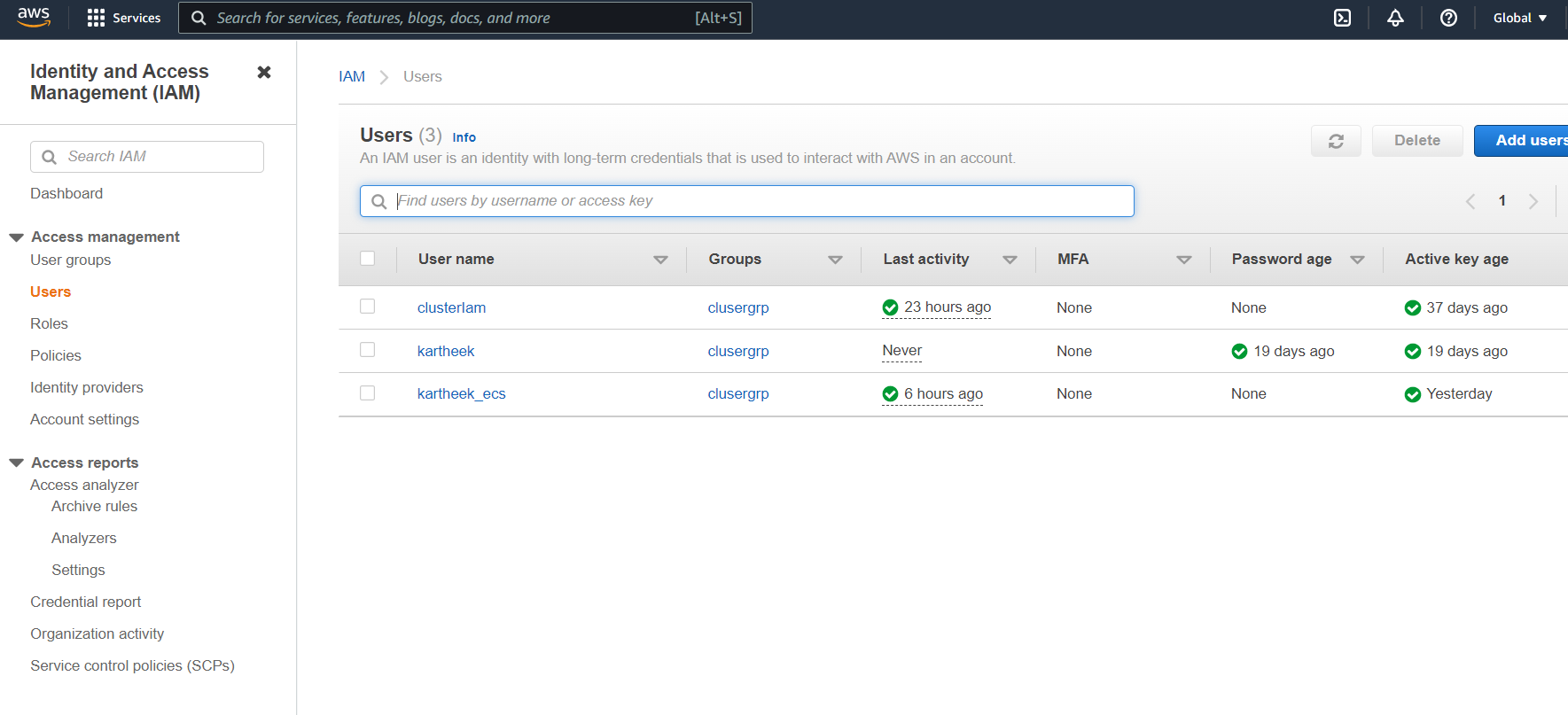
1. Access Key ID
2. Secret Access Key
3. AWS Region
4. Output Format

**Create IAM user:**

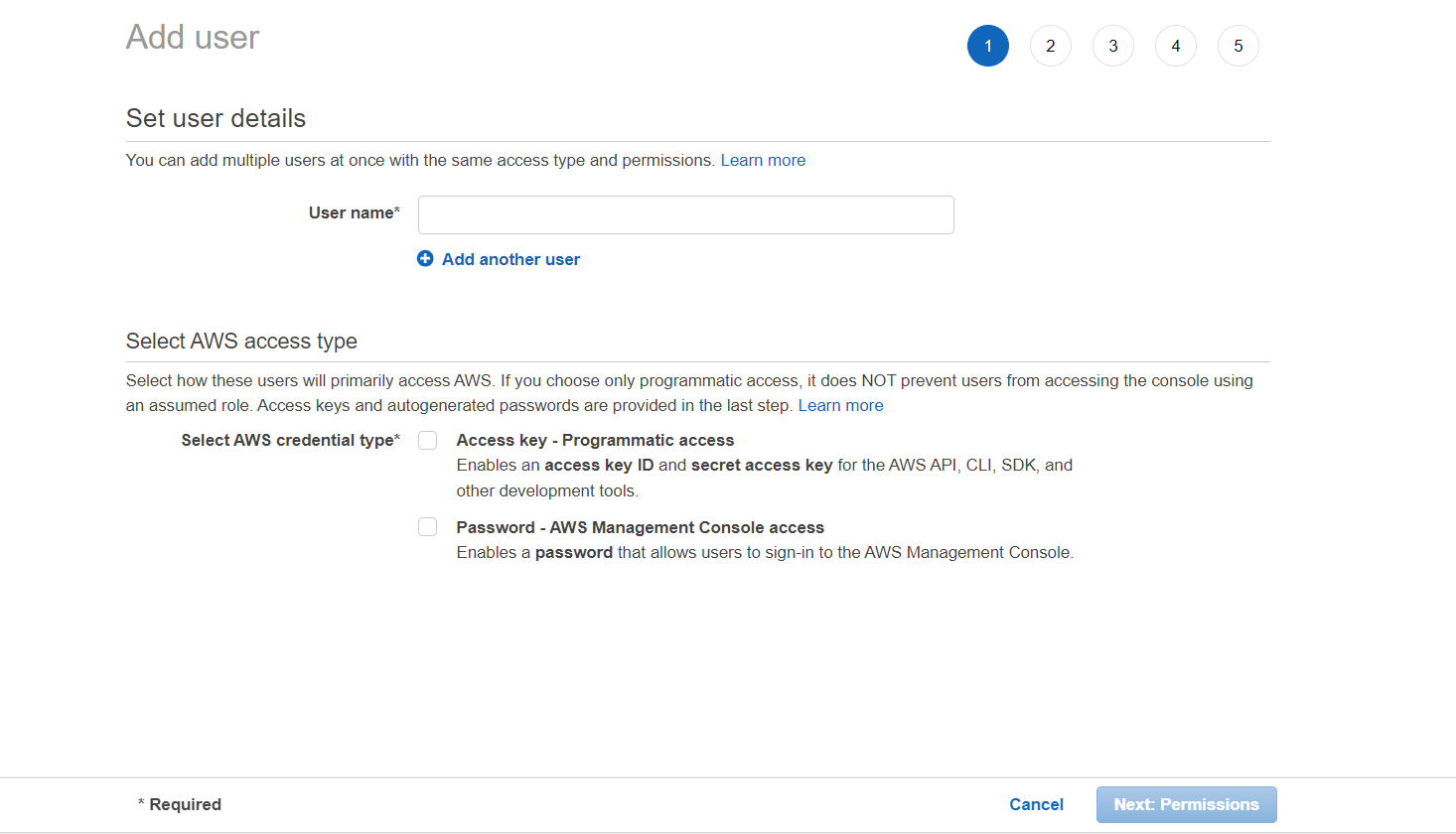
>An IAM User is an entity created in AWS that provides a way to interact with AWS resources.

> To create an IAM user, Sign into the AWS management console.

> Open the IAM console, The screen appears which is shown below.

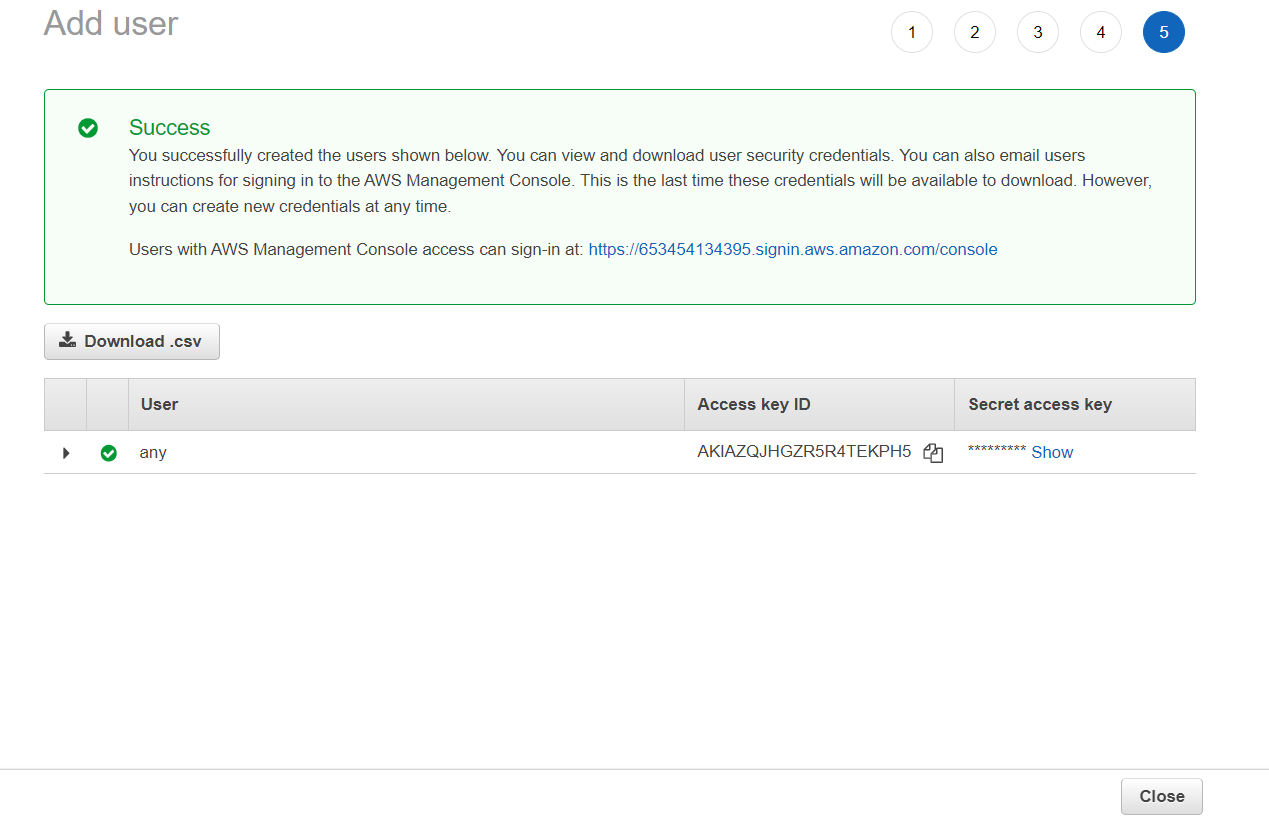
> On the navigation pane, click on the Users. After clicking on the Users, the screen appears which is shown below

> Click on the Add User to add new users to your account. After clicking on the Add User, the screen appears which is shown below:



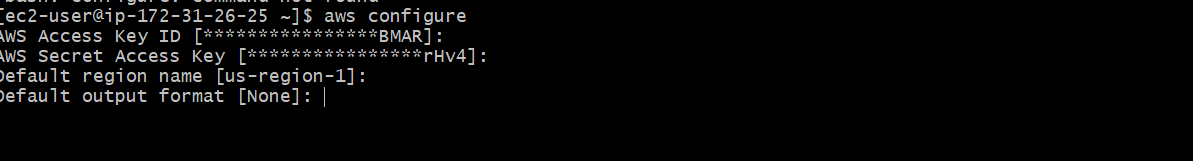
> Enter the user name as you required, and select the programmatic access then click next.

> You have to give the permission to the user, After creating success it will prompt the screen as below.



> Download the csv file and save it in a safe folder.

> Now you have to run the **aws configure** command, it will ask you for the secret key and access key id.



**Creating s3 Bucket:**

> We are using an s3 bucket to store the state files.

> **Statefile:** Terraform stores information about your infrastructure in a state file. This state file keeps track of resources created by your configuration and maps them to real-world resources

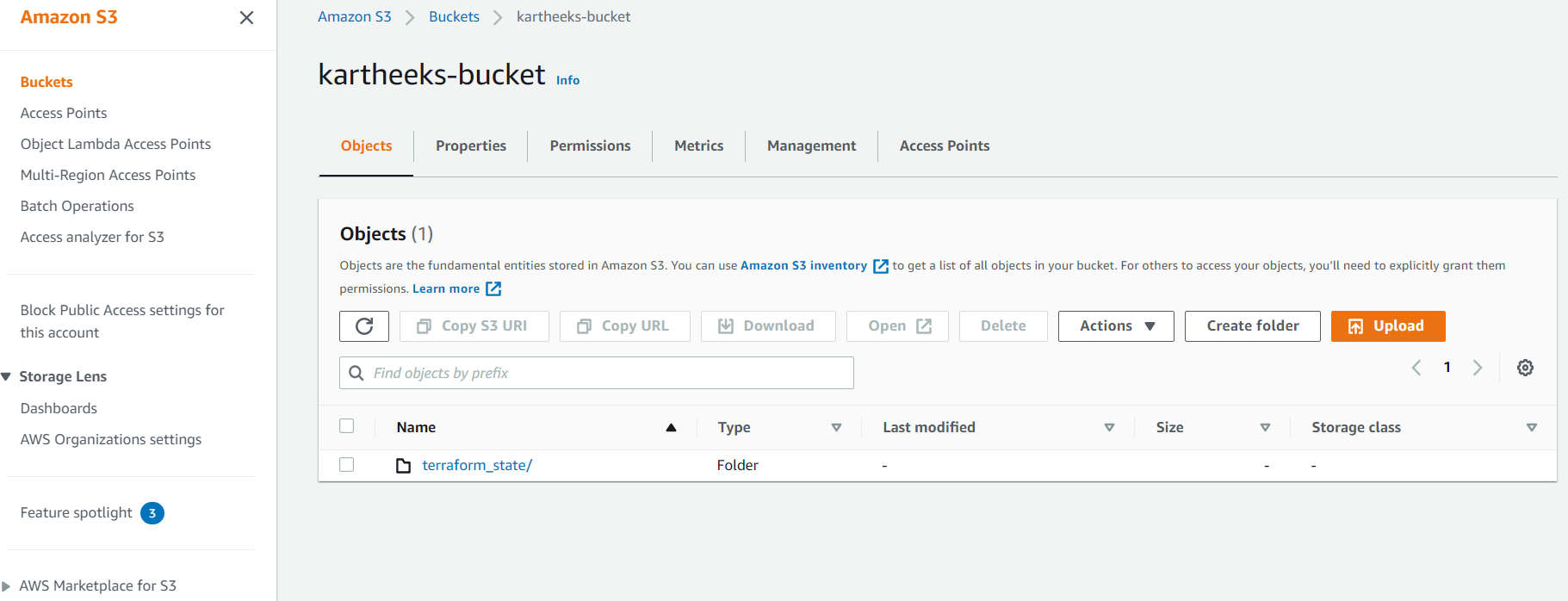
> To create the s3 bucket open the s3 service in aws console.

1. open the s3 service in aws console.
2. Choose Create bucket.The Create bucket wizard opens.
3. In Bucket name, enter a DNS-compliant name for your bucket.

The bucket name must:

* Be unique across all of Amazon S3.
* Be between 3 and 63 characters long.
* Please do not contain uppercase characters.
* Start with a lowercase letter or number.

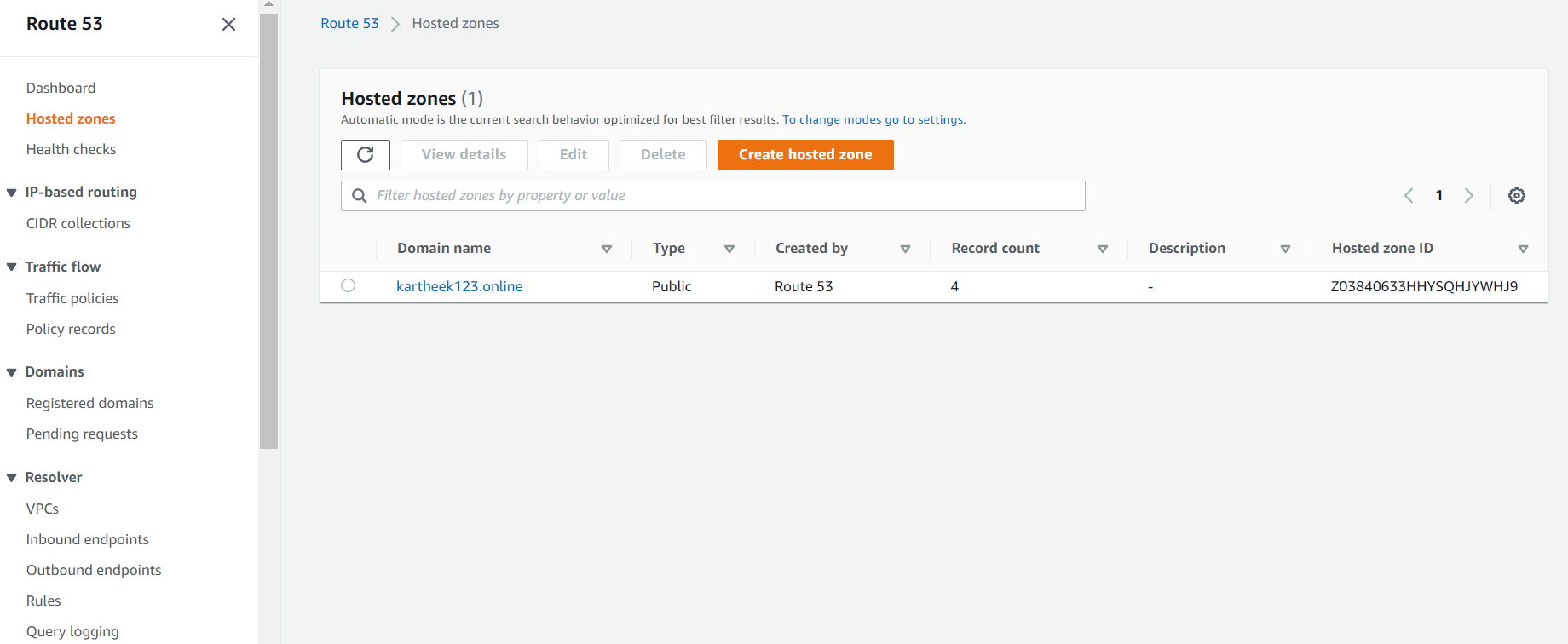
NOTE: After you create the bucket, you cannot change its name

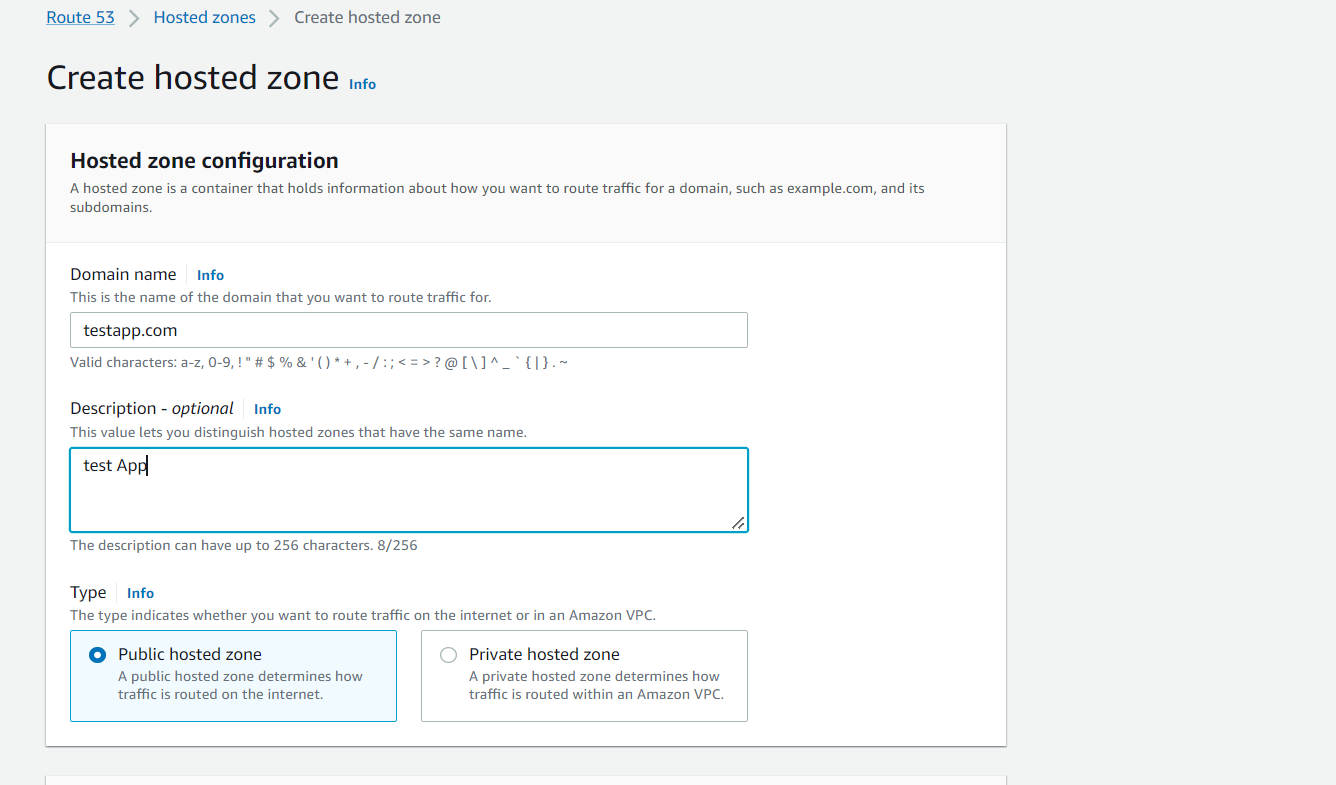


**Create route53 zone:**

A public hosted zone is a container that holds information about how you want to route traffic on the internet for a specific domain, such as example.com, and its subdomains (acme.example.com, zenith.example.com). After you create a hosted zone, you create records that specify how you want to route traffic for the domain and subdomains.

**Create hosted zone:**

**>** Goto the route53 service in aws management console. And click on hosted zones in the left side panel. It leads you to a dashboard like below.

> Click on Create Hosted Zones button to create a new hosted zone. Write the name of the domain in the Domain name field and add some description in the description field. description field is an optional field.Type leave as default. Before pressing the create button check the name of the domain. Finally press the create button. For example I am adding ‘testapp.com’ as a domain name and adding ‘test App ‘ comment in the comment field.

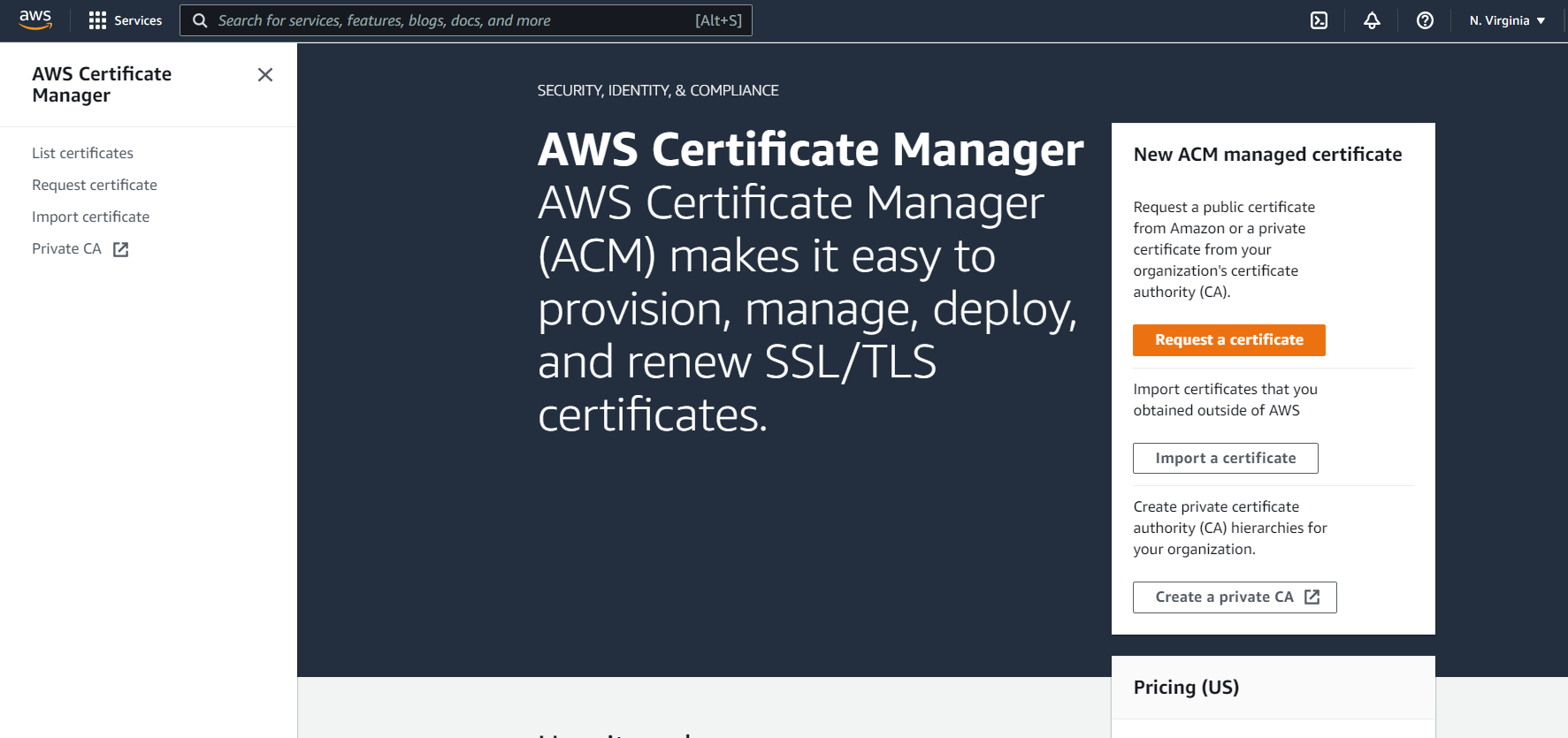
> You can see that the dashboard shows a new item in the Hosted Zone. Select the particular domain and click on it. You can see 2 record sets have been already created. One is the NS type and the other is SOA type. From here you can create the hosts of your domain. For example – www.testapp.com, docs.testapp.com, www.docs.testapp.com, blog.testapp.com, www.blog.testapp.com etc.

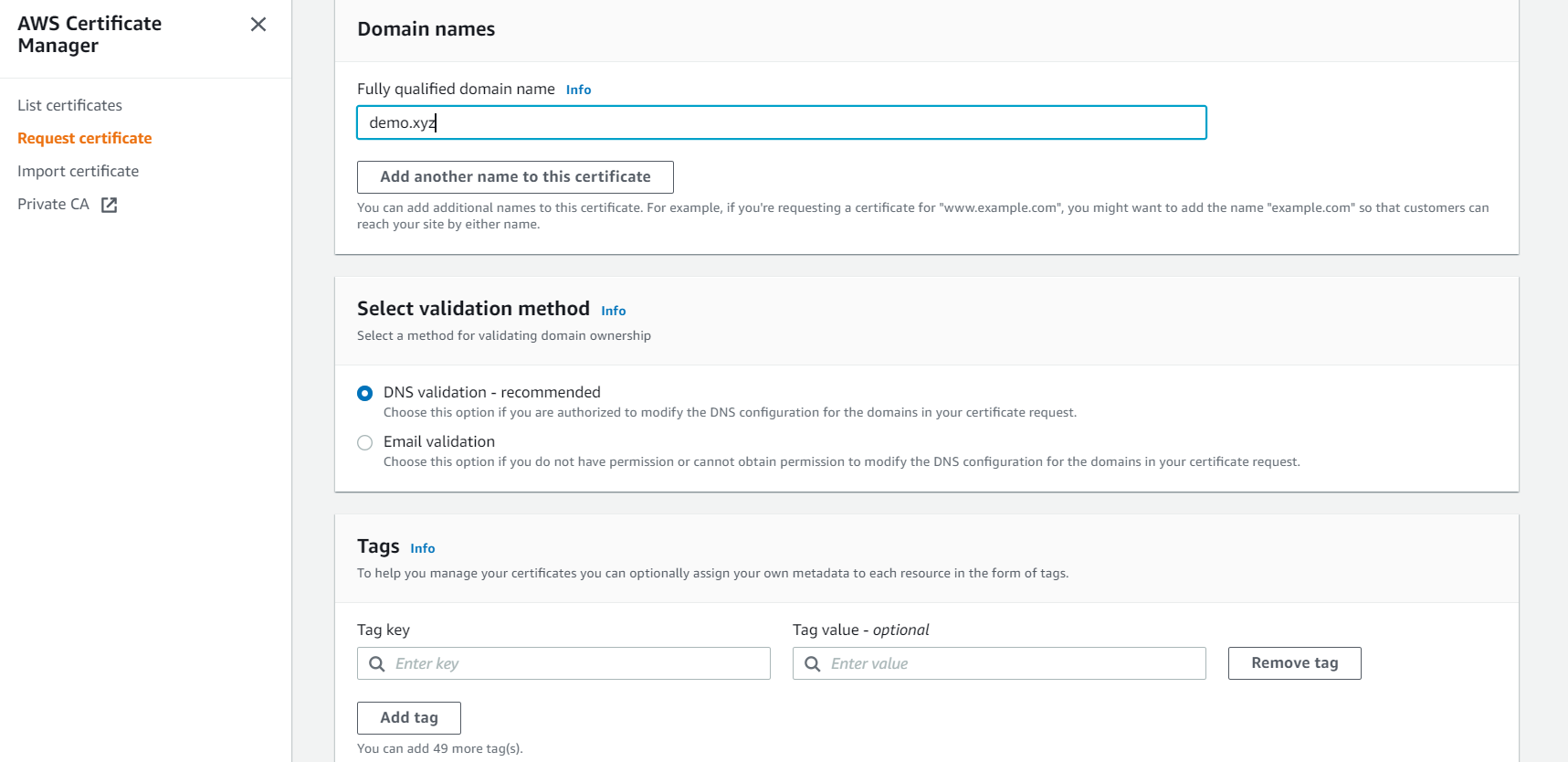
**Creating Certificate:**

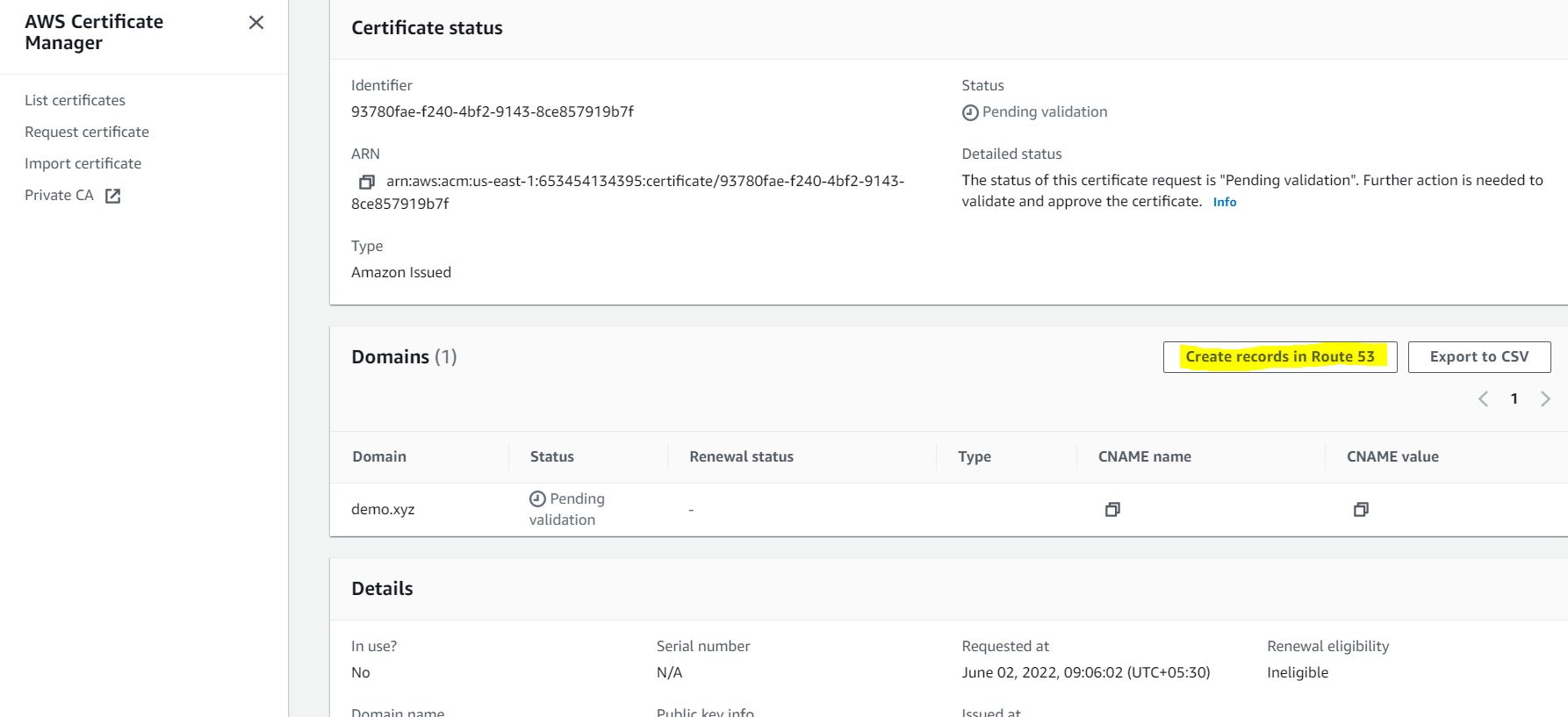
**Aws certificate manager:** AWS Certificate Manager is a service that lets you easily provision, manage, and deploy public and private Secure Sockets Layer/Transport Layer Security (SSL/TLS) certificates for use with AWS services and your internal connected resources. SSL/TLS certificates are used to secure network communications and establish the identity of websites over the Internet as well as resources on private networks. AWS Certificate Manager removes the time-consuming manual process of purchasing, uploading, and renewing SSL/TLS certificates.

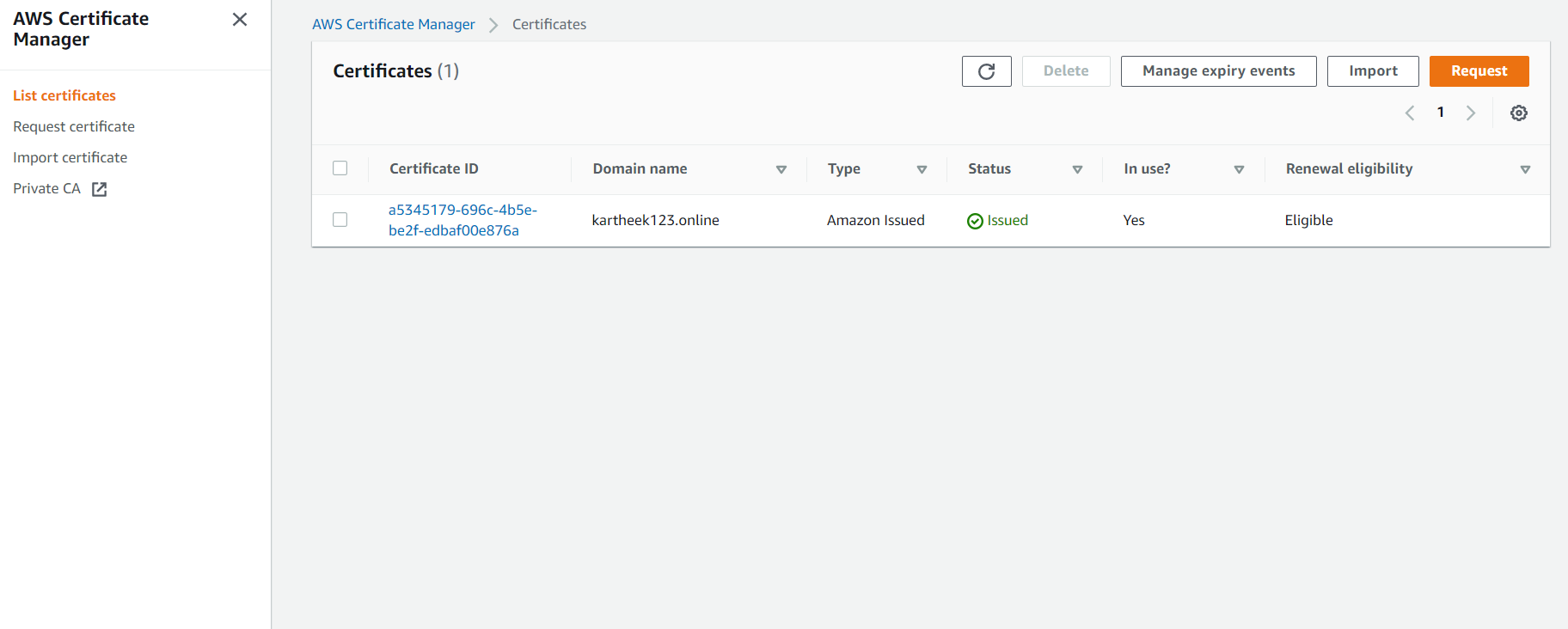
**Create aws certificates:**

**>** To create an aws certificate goto the aws management console and goto **certificate management** service.

> Choose Request a certificate to request a new certificate.

> Provide your **domain name,** and choose the dns validation method. Then click on next

> Once the certificate is created, open it and click on **Create records in route53**. then it automatically creates a **CNAME** record in the Route53 **hosted zone.**

**>** Refresh, and once the validation is completed the status of the certificate will become issued.

**Install Terraform:**

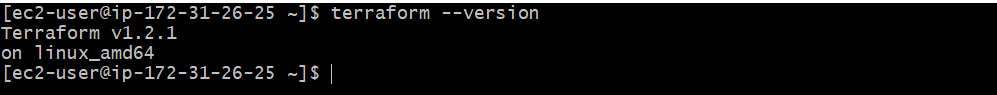
**Terraform:** HashiCorp Terraform is an infrastructure as code tool that lets you define both cloud and on-prem resources in human-readable configuration files that you can version, reuse, and share. You can then use a consistent workflow to provision and manage all of your infrastructure throughout its lifecycle. Terraform can manage low-level components like compute, storage, and networking resources, as well as high-level components like DNS entries and SaaS features.

> To create all these resources we are Using terraform as an infrastructure as a code tool.

> To run the code first we need to install the terraform in our instance which we created earlier.

> To install terraform follow this link.

<https://learn.hashicorp.com/tutorials/terraform/install-cli>

> Check installation by executing **terraform –version** command. As shown below.

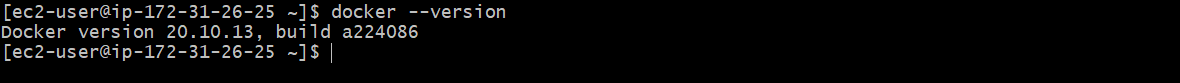
**Install Docker:**

**docker:**Docker is an open platform for developing, shipping, and running applications. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly. With Docker, you can manage your infrastructure in the same ways you manage your applications. By taking advantage of Docker’s methodologies for shipping, testing, and deploying code quickly, you can significantly reduce the delay between writing code and running it in production.

> Install the docker on the same machine which we created earlier

> To install docker follow the below link

<https://docs.docker.com/desktop/linux/install/>

> Check installation by executing **docker –version** command, as shown below

**Clone repos from Github:**

> we have already developed code in the Github repository.

> Do SSH into the instance and create the directory called **ecs,** using mkdir command.

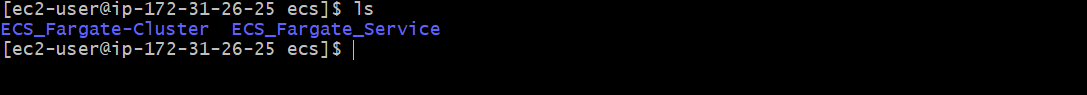
> Go inside that directory and clone the repos from the Github, link as provided below

> clone the repos from github.

Github link: <https://github.com/Kartheek39>

git clone <https://github.com/Kartheek39/ECS_Fargate-Cluster.git>

git clone <https://github.com/Kartheek39/ECS_Fargate_Service.git>

> Once cloning successfully the folders will look like this

**Creating ECS Fargate Cluster:**

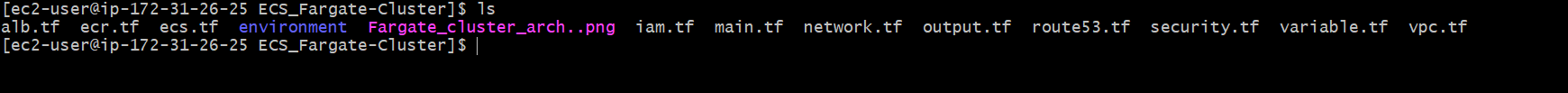
**ECS Fargate:** AWS Fargate is a technology that you can use with Amazon ECS to run [containers](https://aws.amazon.com/what-are-containers) without having to manage servers or clusters of Amazon EC2 instances. With AWS Fargate, you no longer have to provision, configure, or scale clusters of virtual machines to run containers. This removes the need to choose server types, decide when to scale your clusters, or optimize cluster packing.

When you run your tasks and services with the Fargate launch type, you package your application in containers, specify the CPU and memory requirements, define networking and IAM policies, and launch the application. Each Fargate task has its own isolation boundary and does not share the underlying kernel, CPU resources, memory resources, or elastic network interface with another task.

**Creating ecs fargate:**

> Go inside the ecs directory.

**/home/ec2-user/ecs**

> Then go inside the ECS\_Fargate-Cluster folder. And do **ls** you can see below files

**Resources:**

> As per our architecture we need to create the resources like..

Vpc

Ecs

Ecr

Alb

Route53

IAM roles

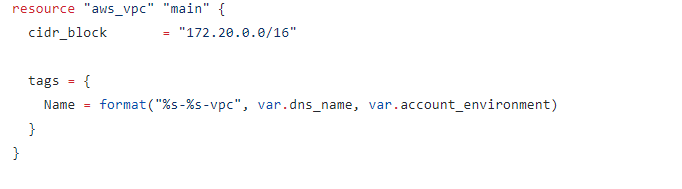
Security groups

**Vpc.tf**

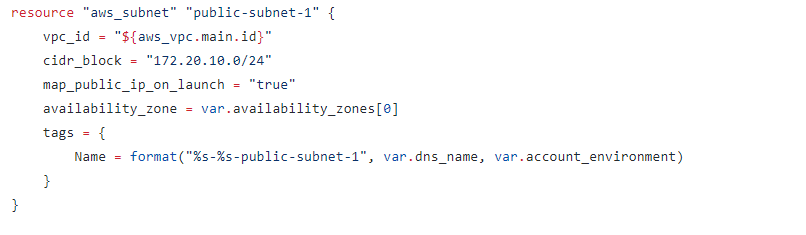
**vpc:**

> To Create vpc we created a file called vpc.tf,

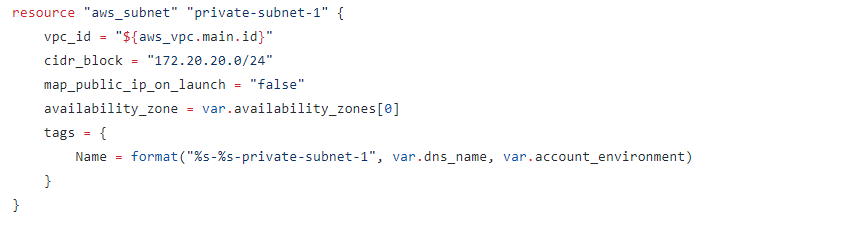
> This vpc.tf file consist of code for creating vpc and subnets(two public subnets and two private subnets)



cidr\_block: 172.20.0.0/16 allows you to use the IP address that starts with “172.20.X.X”. There are 65,536 IP addresses ready to use.

**Public subnet:**

**Private subnet:**

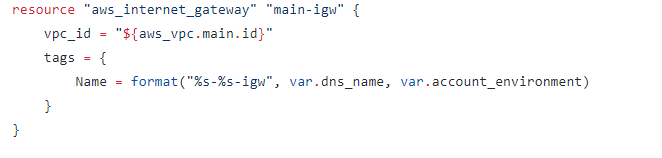
****

vpc\_id: this subnet will be the vpc just created before. We give the created VPC id to the subnet.

cidr\_block: 10.0.1.0/24. We have 254 IP addresses in this subnet

map\_public\_ip\_on\_launch: This is so important. The only difference between private and public subnet is this line. If it is true, it will be a public subnet, otherwise private.

**Network.tf:**

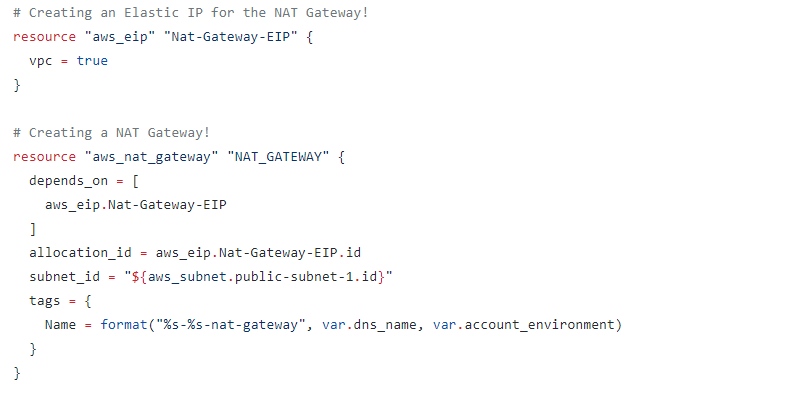
**Internet gateway:**

It enables your vpc to connect to the internet.

**Route table:**

Here we are creating a route table with cidr “0.0.0.0/0" on any ip,with Internet gateway as target.

And then associating the route table to a public subnet.

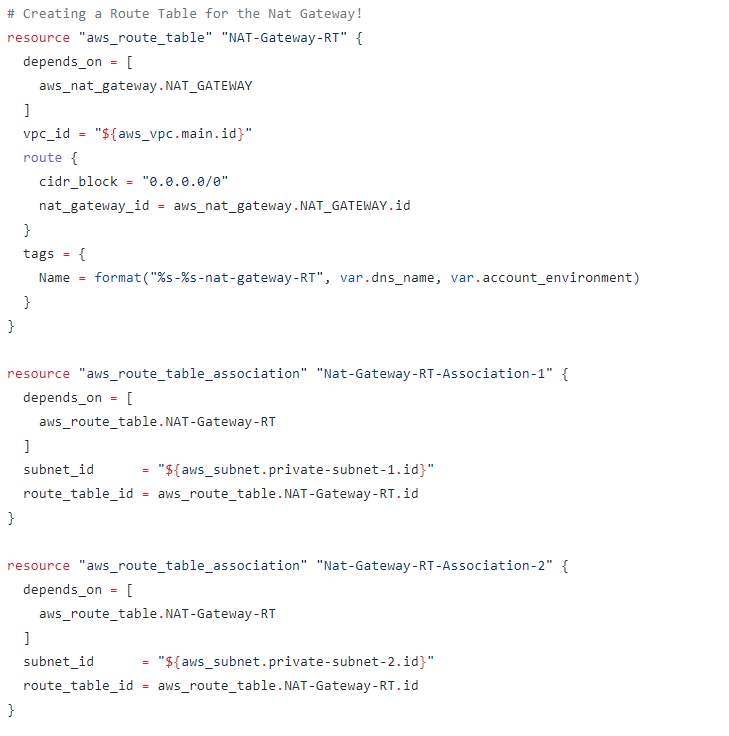
**Nat gateway:**

NAT Gateway is a highly available AWS managed service that makes it easy to connect to the Internet from instances within a private subnet in an Amazon Virtual Private Cloud (Amazon VPC).

NAT Gateway Only Provides SNAT so private subnet instances can connect to the Internet,and Vice versa connectivity can’t be established.

NAT Gateway also required an EIP (Elastic IP) ,So here we are creating an EIP and then we are creating a NAT gateway.

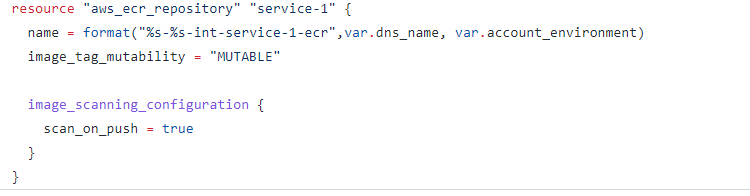
Note we are creating this NAT gateway in a public subnet becauseNAT gateway requires Internet connectivity,so Instances launched in private subnet have internet connectivity,And NAT gateway will use Internet Gateway for Internet connectivity,Remember we Associated a Route to Internet Gateway in public subnet.

**Nat route table and association:**

We have to create a route table with a target as NAT gateway and Associate it to a private subnet So instances in the private subnet can connect to the internet.

Here we are creating a route with cidr “0.0.0.0/0" meaning any ip and target as NAT gateway then we associate this table to a private subnet.

So the request from the instance inside the private subnet goes to NAT gateway in the public subnet and from NAT gateway it goes to Internet Gateway.

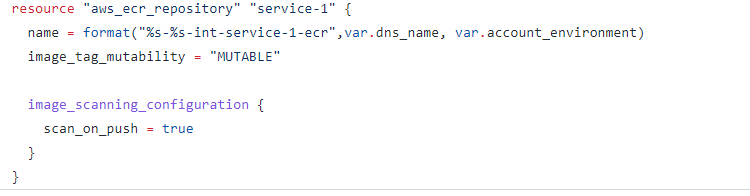
**ecr.tf:**The ECR is a repository where we're gonna store the Docker Images of the application we want to deploy. It works like the Docker Hub, if you're familiar with Docker. You can build the Docker Image locally and push it to the ECR.

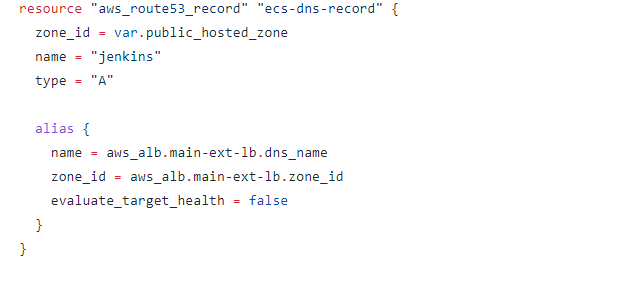
**Ecr.tf:**

Before we create the ECS Cluster, we need to create an IAM policy to enable the service to pull the image from ECR.

**iam.tf:**

****

Now let's create what we need for ECS. First we create the ECS Cluster:

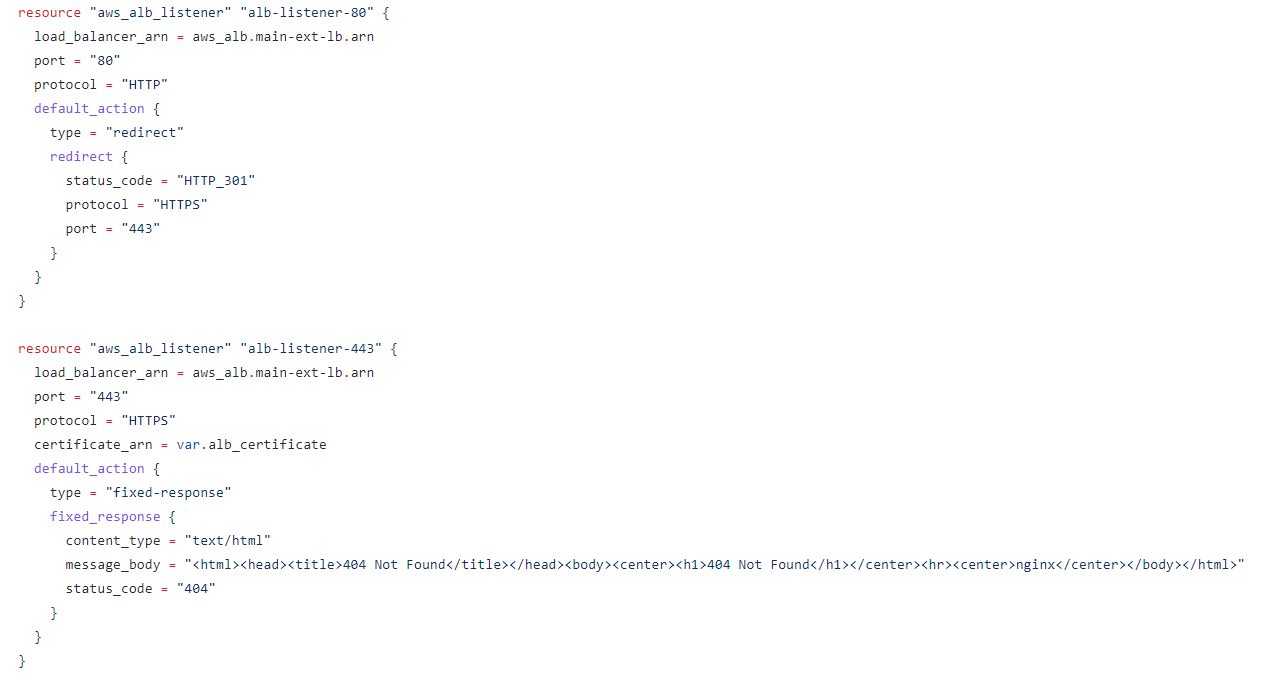
**Route53.tf: **

**>** Here we are creating the **route53 dns record,** in the hosted zone we created earlier.

**Alb.tf:**

****

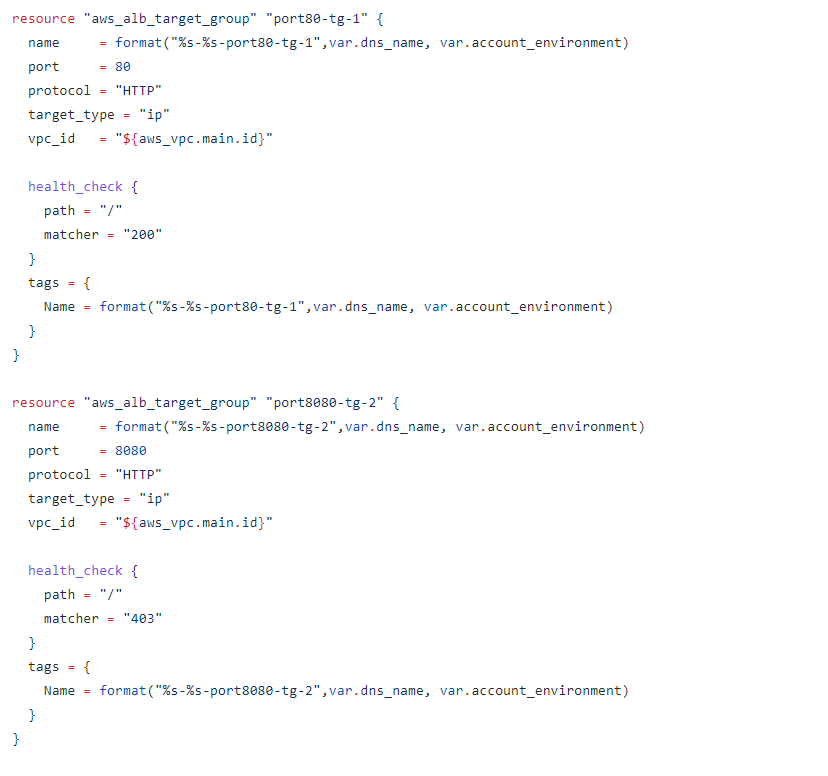
**>** Here we are creating an alb(application load balancer)**,**This is the top level component in the architecture. The ALB handles the incoming traffic, offloads SSL and balances the load.

**Alb listeners: **

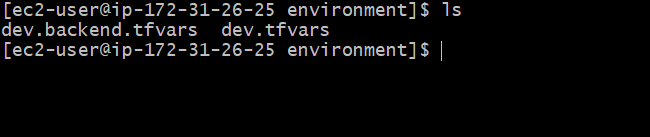
**>** Listeners are assigned a specific port to keep an ear out for incoming traffic. They can have a maximum of 50 listeners assigned to each load balancer.

**Alb listener rules:**

**>** this is where things get pretty nifty! Each listener can have many rules which means we can route traffic to different places based on two conditions; the path and/or the host.

**Target group:**

**>** Target groups are essentially the end point of the ALB architecture — When the listener rule matches a pattern for a request it gets forwarded to the correlating target group. The cool thing about target groups is they have a health check that can directly check the health of a path

> Go inside the **environment** directory, you can find below files

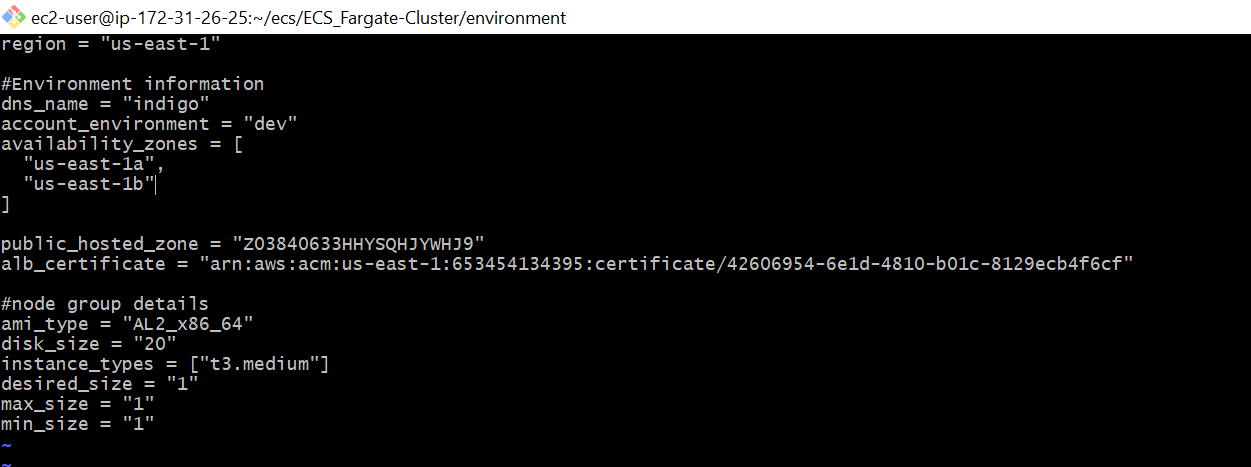
> Edit the dev.backend.tfvars file using vi/vim editor. Give the bucket name which was created earlier and mention the region as required. Leave the key as default.

**vim dev.backend.tfvars**



> In the same folder edit the dev.tfvars, This is the variables file which will be used inside the code.

**vim dev.tfvars**



Region = < as you required >

Dns\_name = < leave as default >

Account\_environment = < leave as default >

Availability\_zones = < give two availability zones based on selected region >

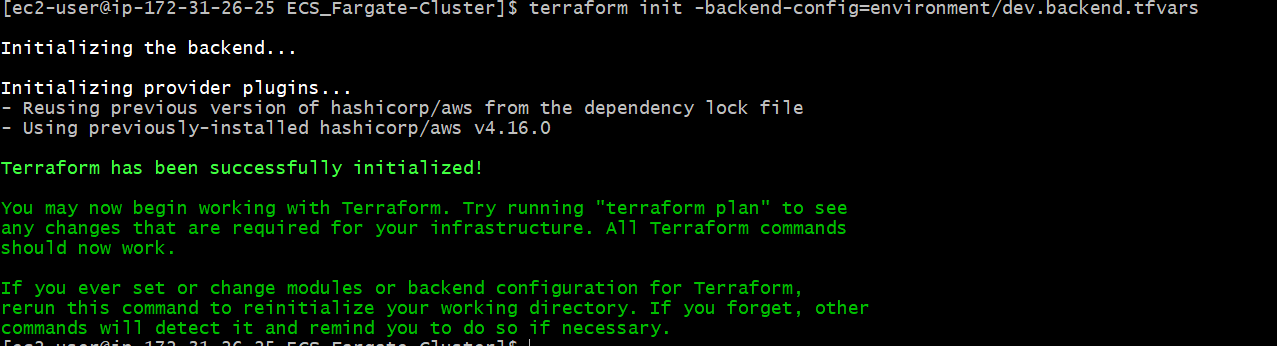
Public\_hosted\_zone = <give the hosted zone id >

Alb\_certificate = <copy the arn id of the certificate we created and paste is here>

#node group details- leave as default

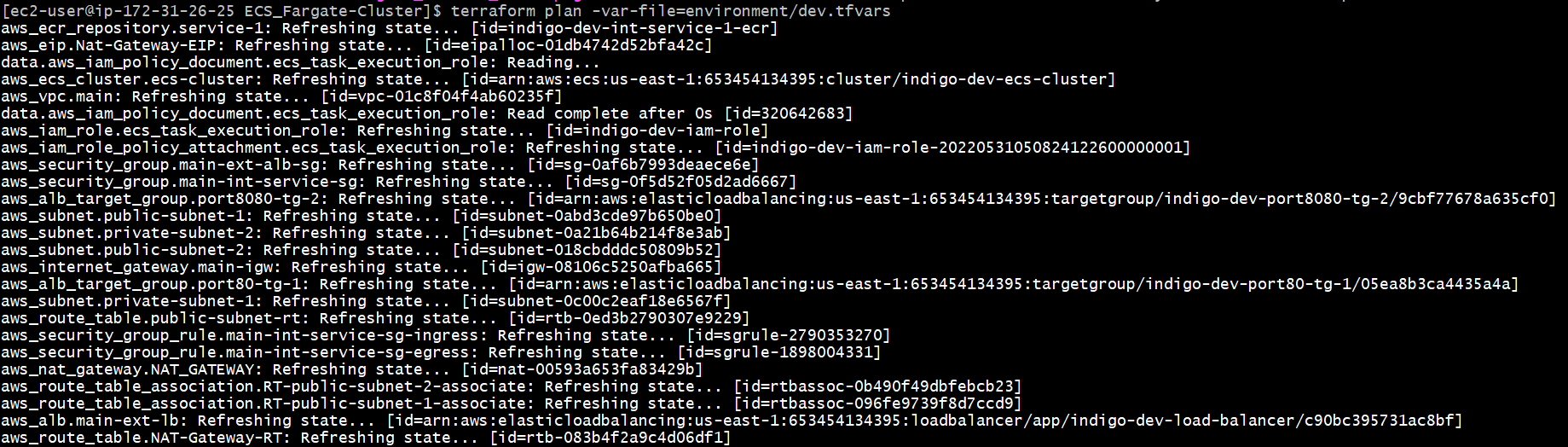
**Terraform init:** The terraform init command is used to initialize a working directory containing Terraform configuration files. This is the first command that should be run after writing a new Terraform configuration or cloning an existing one from version control. It is safe to run this command multiple times.

> Then run the terraform init command, Using the below command and the output will look like this.

**terraform init -backend-config=environment/dev.backend.tfvars** 

**Terraform plan:** The terraform plan command creates an execution plan, which lets you preview the changes that Terraform plans to make to your infrastructure. By default, when Terraform creates a plan it: Reads the current state of any already-existing remote objects to make sure that the Terraform state is up-to-date.

> then run the terraform plan command to check what resources are going to be created.

**terraform plan -var-file=environment/dev.tfvars**

**Terraform apply:** The terraform apply command performs a plan just like terraform plan does, but then actually carries out the planned changes to each resource using the relevant infrastructure provider's API. It asks for confirmation from the user before making any changes, unless it was explicitly told to skip approval.

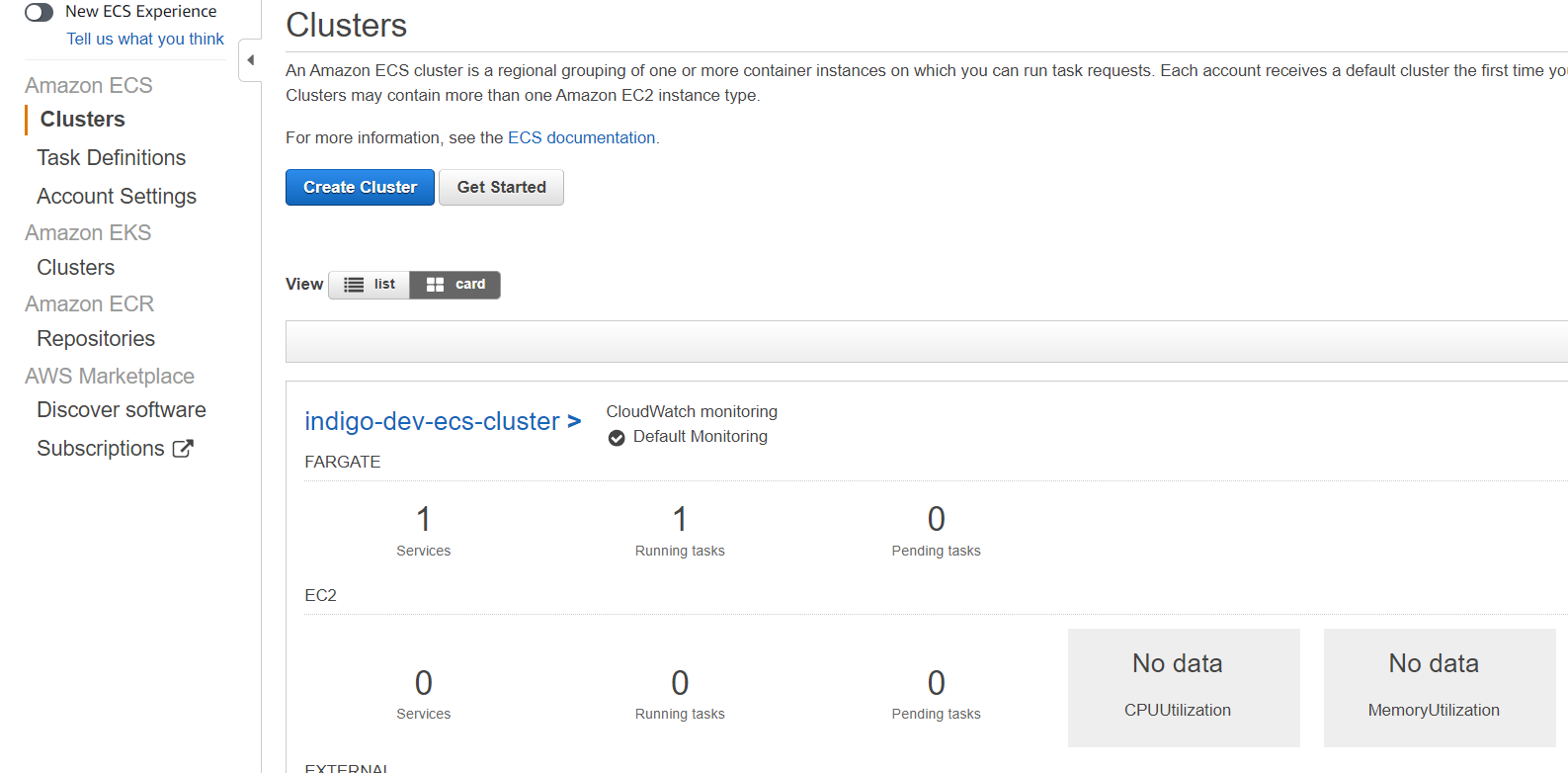
**>** After executing the terraform plan, we have to execute the **terraform apply** to create those resources in the console.

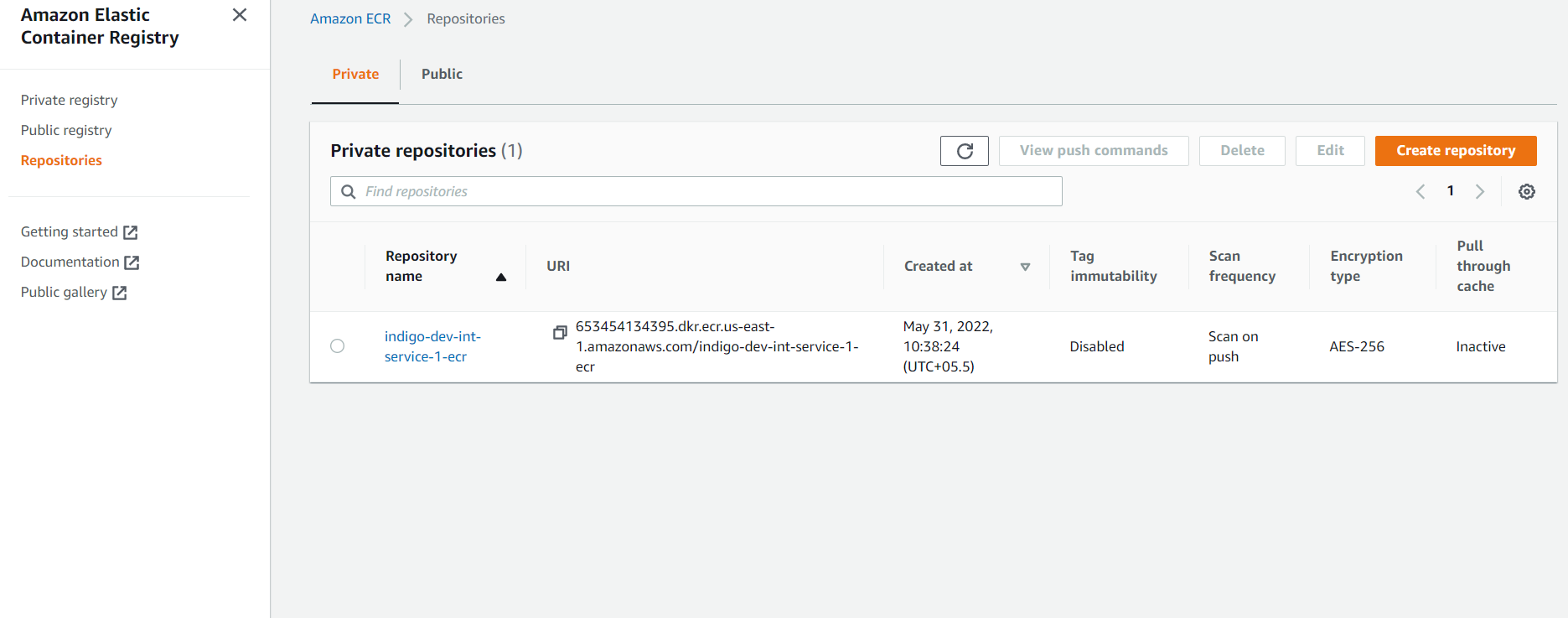
**terraform apply -var-file=environment/dev.tfvars**

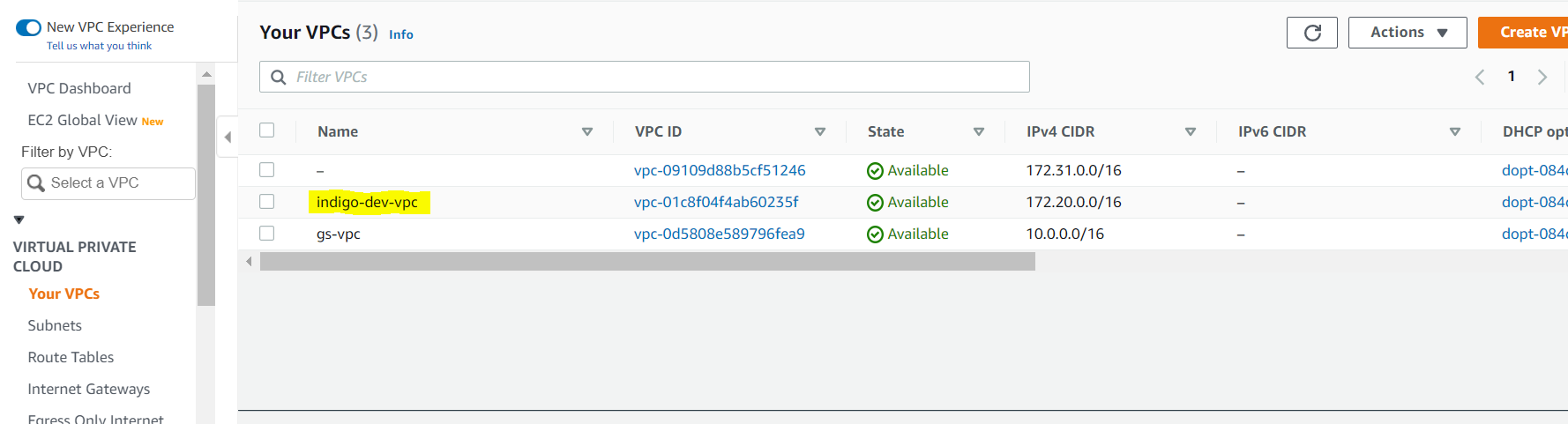
**>** completion of apply it shows the **apply completed** as shown below

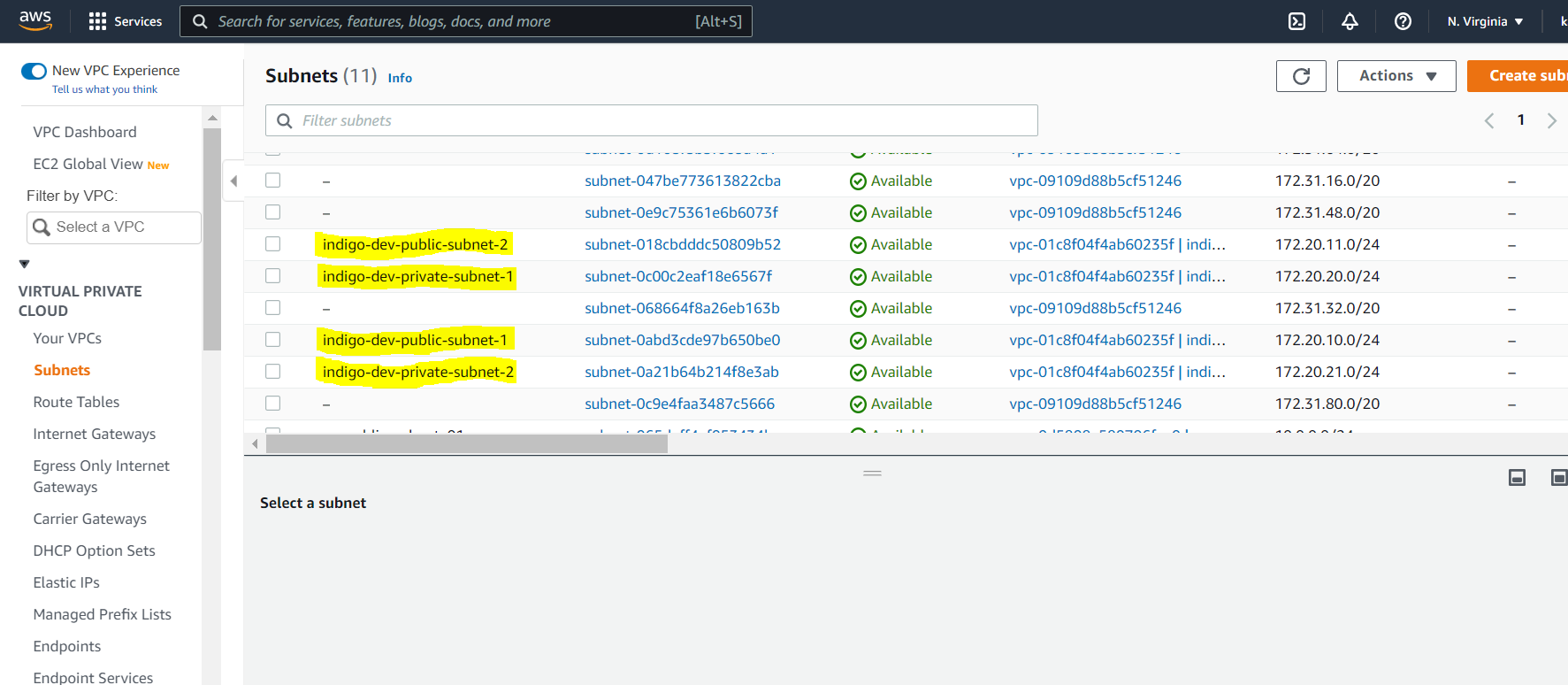
So it will create resources like ECS,ECR,ALB,IAM roles,VPC,Subnets,etc…

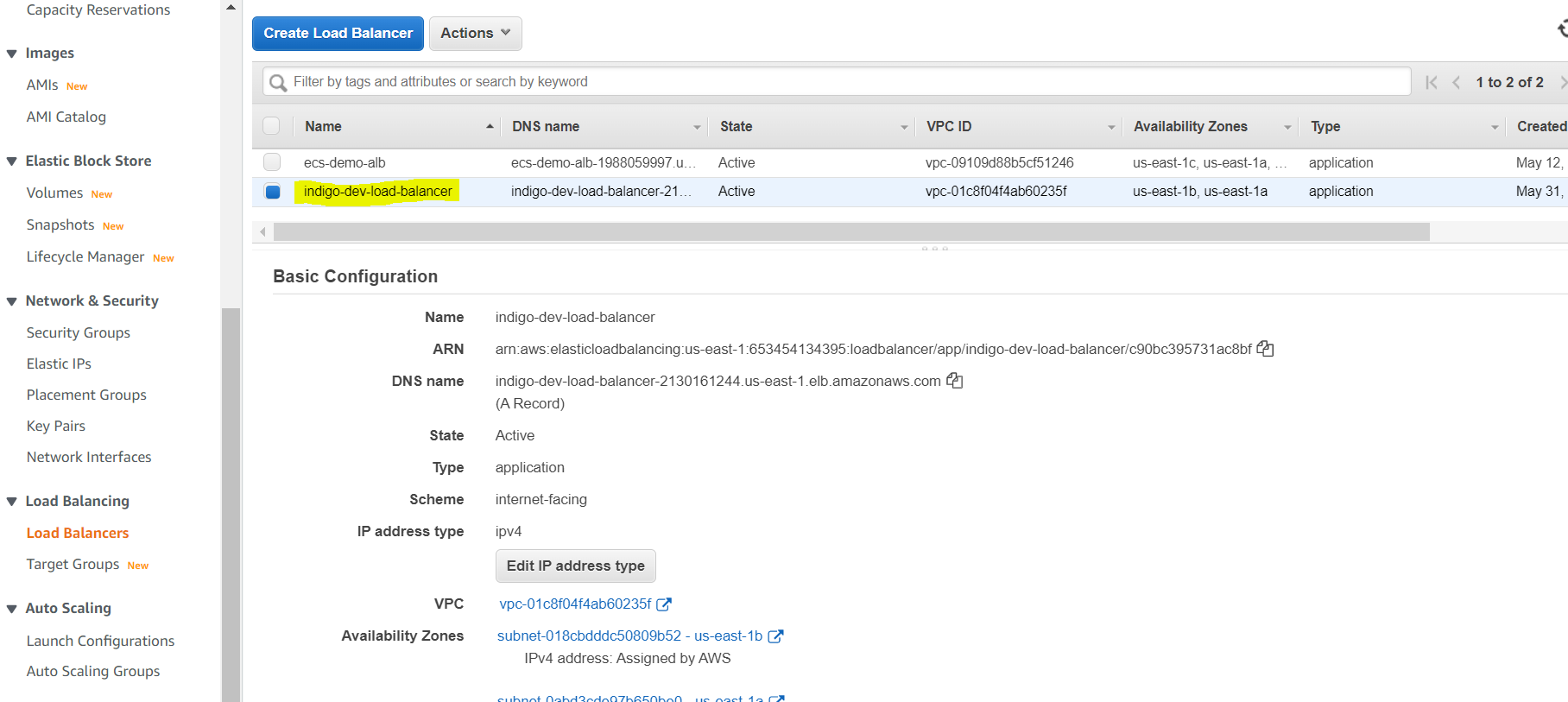
Once check in the console as they created as we mentioned in the code.

ECS:

ECR:

Vpc:

Subnets:

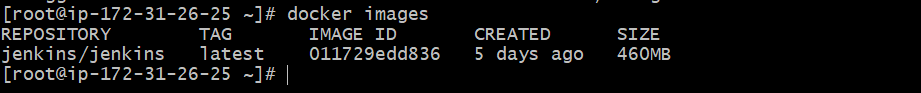
ALB:

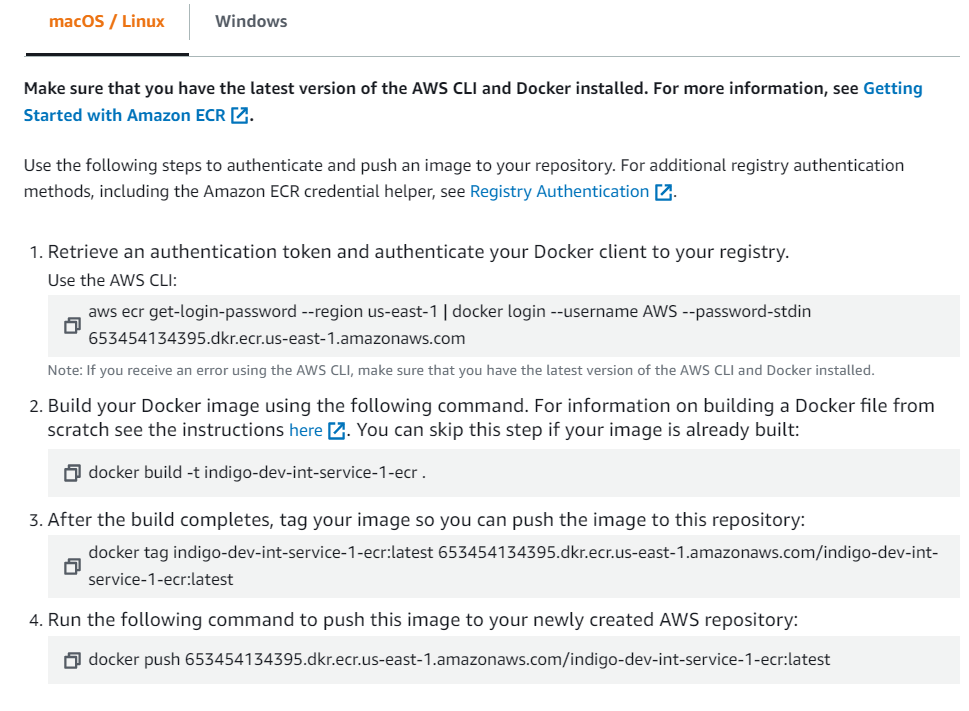
**Push Jenkins Image to ECR:**

> After the successful creation of resources, we have to push the image to ecr for that to follow the below commands.

1. Pull the jenkins image to your local instance by executing this command.

**Docker pull jenkins/jenkins**

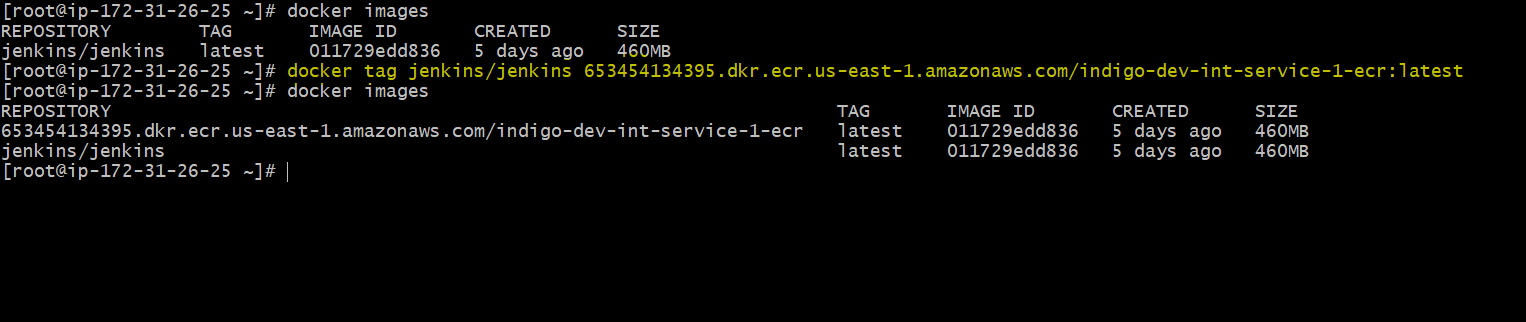
Then you can check by running **docker images** command

1. Goto container registry and click on push commands button the pop-up will appear like below.

Run the first command to authenticate client to your registry.

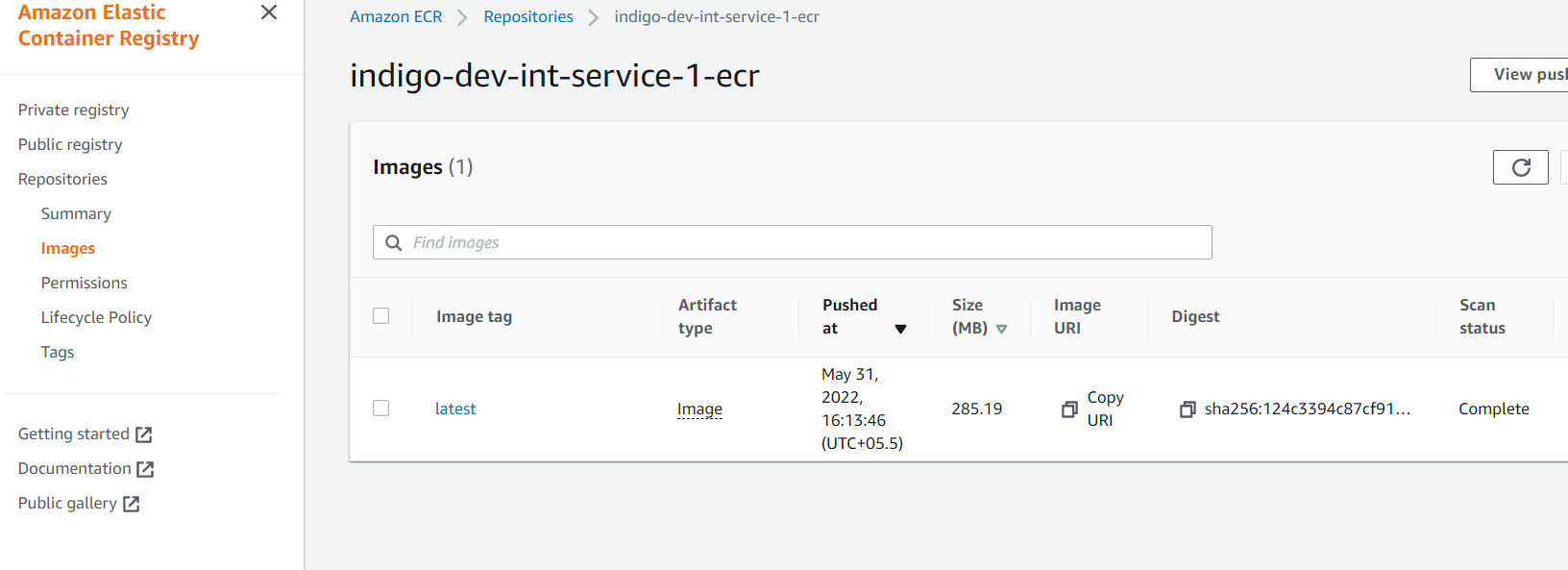
1. Then tag the jenkins image which pulled earlier, as shown in the above image.

**docker tag jenkins/jenkins 653454134395.dkr.ecr.us-east-1.amazonaws.com/indigo-dev-int-service-1-ecr:latest**

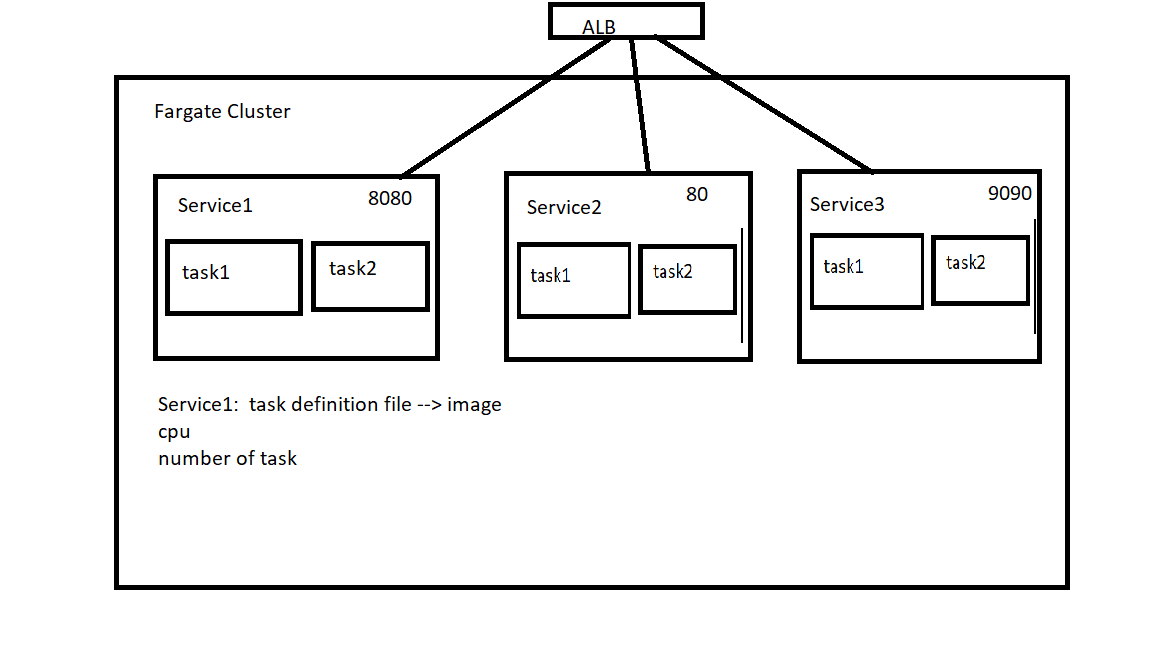


1. Push that tagged image to registry by executing the following command.

**docker push 653454134395.dkr.ecr.us-east-1.amazonaws.com/indigo-dev-int-service-1-ecr:latest**

1. Once push is successful check in the console that image is pushed to our registry.

**Creating Service:**

Architecture:

Amazon ECS services hosted on Amazon EC2 instances support the Application Load Balancer, Network Load Balancer, and Classic Load Balancer load balancer types. Amazon ECS services hosted on AWS Fargate support Application Load Balancer and Network Load Balancer only. Application Load Balancers are used to route HTTP/HTTPS (or layer 7) traffic. Network Load Balancers are used to route TCP or UDP (or layer 4) traffic. Classic Load Balancers are used to route TCP traffic.

> As of now we have created the ecs cluster and pushed image to the ecr and checked up and running. Now we have to create a service to access from outside. For this we have created an ALB.

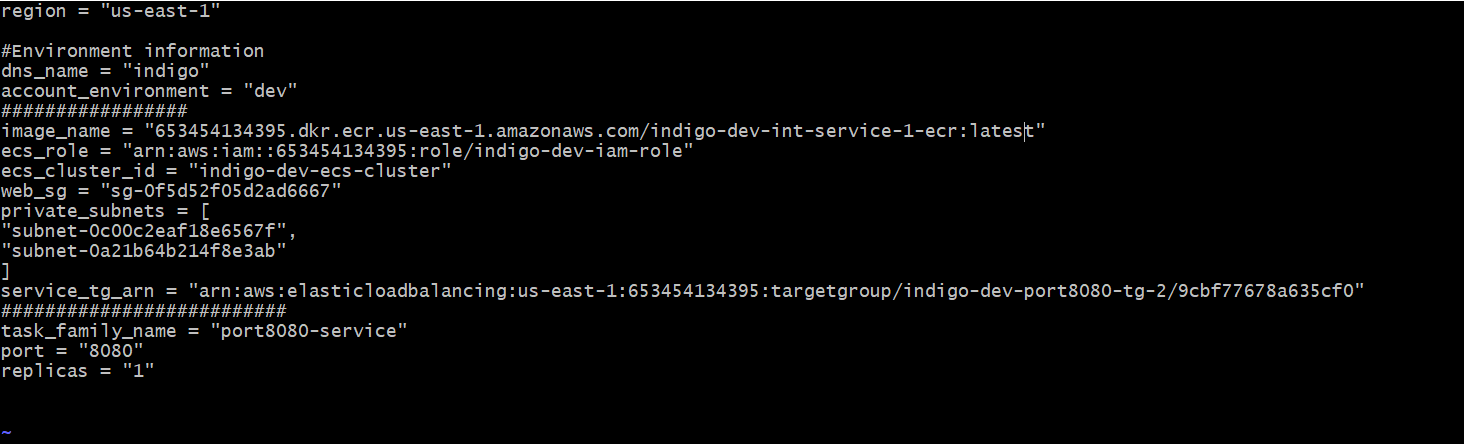
> Go inside the service directory directory.

**/home/ec2-user/ecs/ECS\_Fargate\_Service/service1**

> Edit the dev.backend.tfvars file using vi/vim editor. Give the bucket name which was created earlier and mention the region as required. Leave the key as default.

**cd environment**



> In the same folder edit the dev.tfvars using vi/vim editor , this is the variables file 

Region = <give as required>

Dns\_name = <leave as default>

Account\_environment = <leave as default>

Image\_name = <give the tagged image name we did on earlier steps>

Ecs\_role = <you will find this in output of the **ecs\_cluster** creation >

Ecs\_cluster = <leave as default>

Web\_sg = <you will find this in output of the **ecs\_cluster** creation>

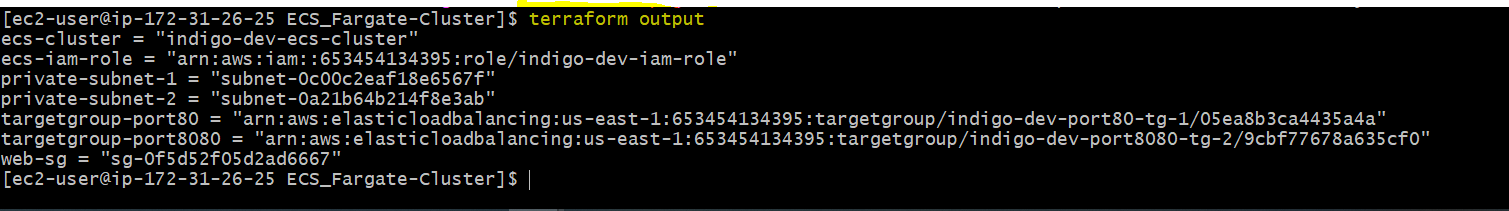
Private\_subnets = <you will find this in output of the **ecs\_cluster** creation>

Service\_tg\_arn = <you will find this in output of the **ecs\_cluster** creation>

Task\_family\_name = <leave as default>

Port = <leave as default>

Replicas = <leave as default>

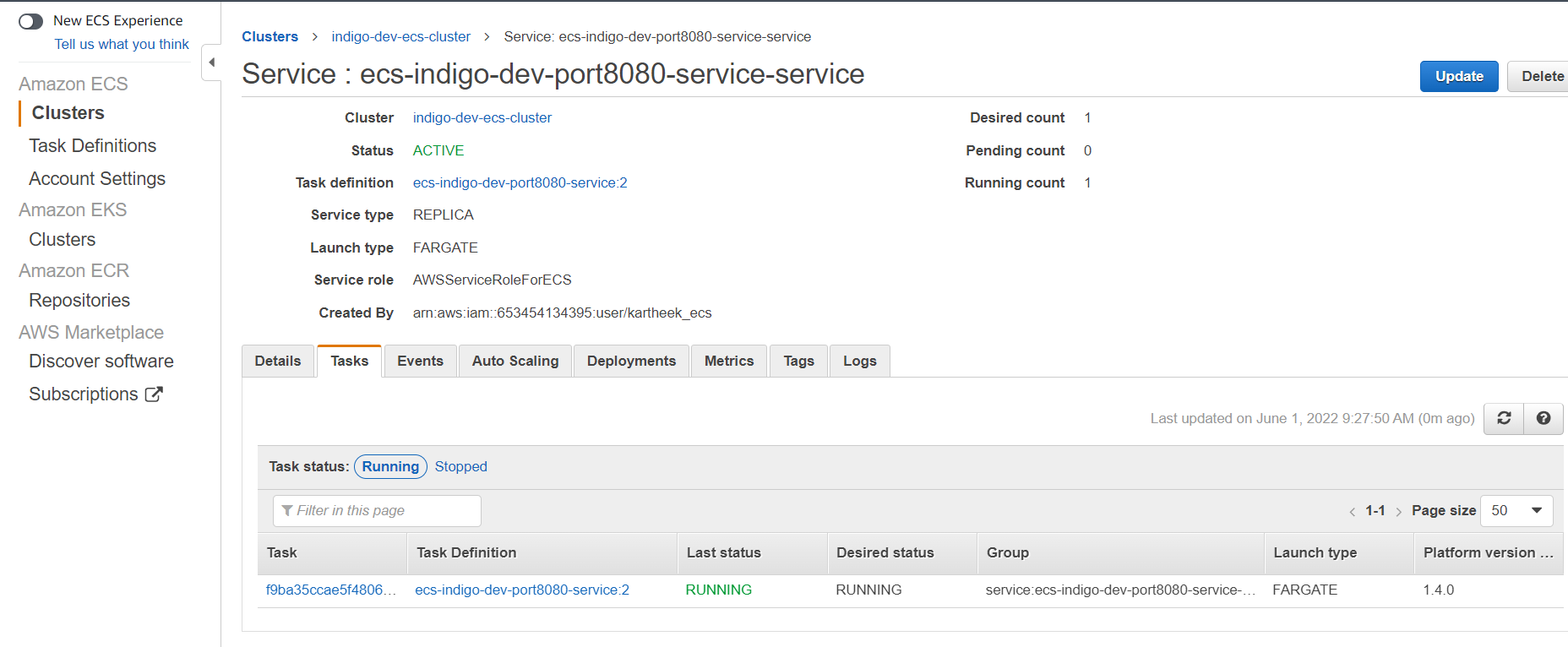
These values will be found in the console if it is difficult to find those, go to the ECS\_Fargate-Cluster and run **terraform output** as shown below, So you can find these values.

> after successfully changing the values you have to go to the ECS\_Fargate\_Service folder and run go inside the service1 folder then run the below commands

**terraform init -backend-config=environment/dev.backend.tfvars**

**terraform plan -var-file=environment/dev.tfvars**

**terraform apply -var-file=environment/dev.tfvars**

**>** So the terraform creates the service for the jenkins , goto console and checks if ecs services status is running.

> after successfully creating the service, we have to copy the dns and hit on the browser to check if jenkins dashboard is appearing or not.