Capstone project – Hotel-Side Hospital

Name: Kotian Rakshith Padmanabha

Email: rakshithpk21@gmail.com

Github link created for the project: https://github.com/kotianrakshith/CapstoneProject3

Objective: To create an automated provisioned infrastructure using Terraform, EKS cluster, EC2 instances, and Jenkins server.

Tools to use:

1. Jenkins
2. Terraform
3. AWS EC2
4. AWS EKS

Description

Hotel-Side Hospital, a globally renowned hospital chain headquartered in Australia, is aiming to streamline its operation by setting up an infrastructure within the hotel premises. However, in order to maintain seamless functioning and scalability, they require fully managed virtual machines (VMs) on the Amazon Web Services (AWS) platform.

The organization seeks an automated provisioned infrastructure solution that can enable them to effortlessly create new Amazon Elastic Kubernetes Service (EKS) clusters, whenever required, and promptly delete them when they are no longer needed. This will optimize resource allocation and enhance operational efficiency

Task (Activities)

1. Validate if Terraform is installed in the virtual machine
2. Install AWS CLI
3. Navigate to AWS IAM service, and get AWS Access key and Secret Key to connect AWS with the AWS CLI
4. Export the AWS Access Key, Secret Key, and Security Token to configure AWS CLI connectivity with AWS Cloud
5. Create terraform scripts to create a new VM using autoscaling which includes the following files: autoscaling.tf, VPC.tf, internetgateway.tf, subnets.tf (public subnet), routetable.tf, Route_table_association_with_public_subnets.tf
6. Execute terraform scripts
7. Connect to an instance and install the stress utility (The stress files are provided along with the problem statement document.)
8. Validate if autoscaling is working by putting load on autoscaling group

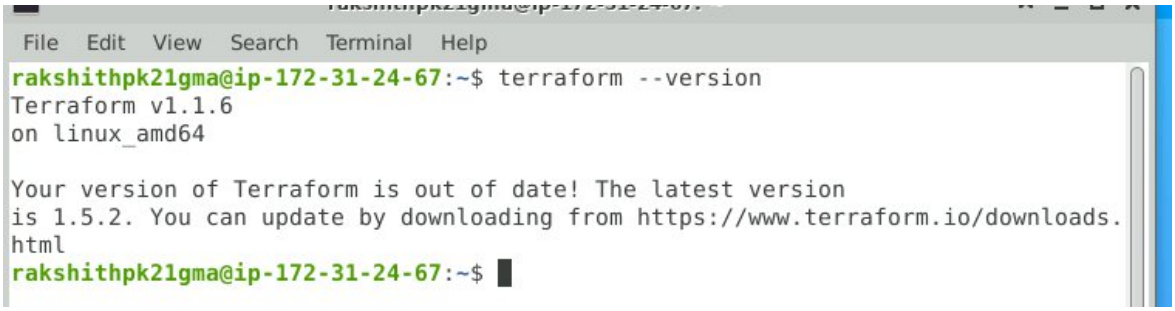
Steps performed:

1. **Validate if Terraform is installed in the virtual machine :**

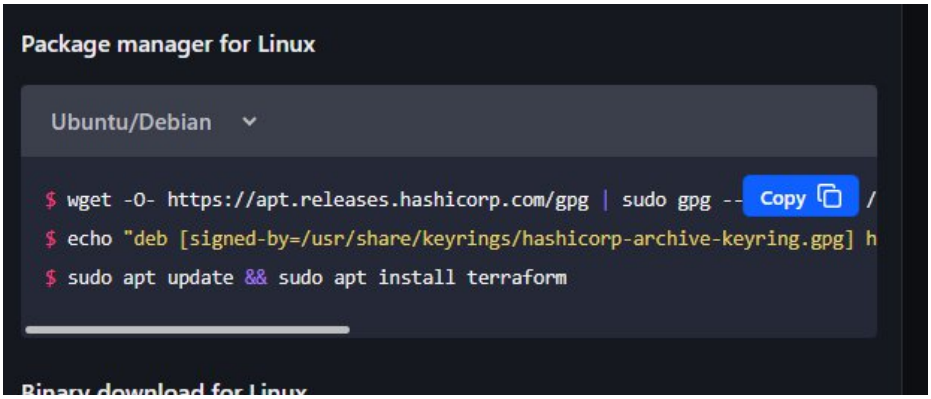
To check if the terrform is installed we can use the command

terraform --version

As we can see it is already installed:



Since the terraform is old version we need to install the new version. From the terraform website we get below commands to download and install terraform in our machine:

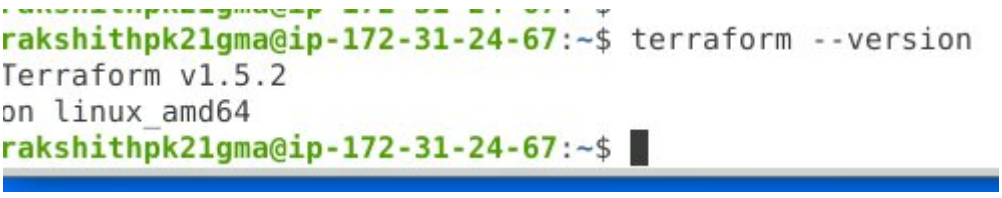


wget -O- https://apt.releases.hashicorp.com/gpg | sudo gpg --dearmor -o /usr/share/keyrings/hashicorp-archive-keyring.gpg

echo "deb [signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg] https://apt.releases.hashicorp.com $(lsb_release -cs) main" | sudo tee /etc/apt/sources.list.d/hashicorp.list

sudo apt update && sudo apt install terraform

Now if we check we can see that terraform is at the newest version



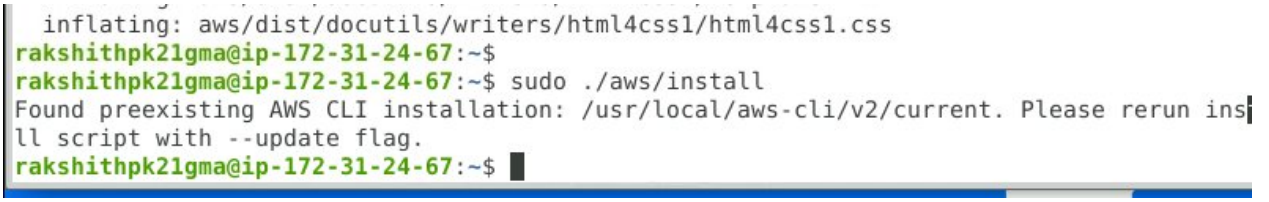
2. **Install AWS CLI**

To install the AWS CLI run the below commands:

(all the commands are taken from amazon official documentation)

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

unzip awscliv2.zip

sudo ./aws/install
```

But as we already have cli in the system pre installed it gave the below error



We can check the version using command:

```
aws --version
```

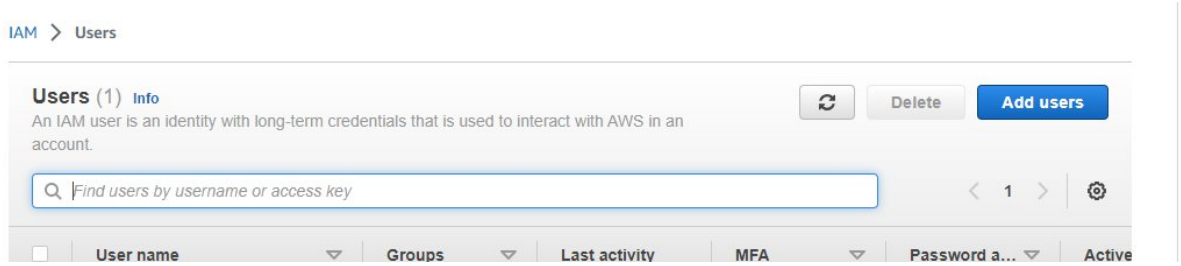
We will upgrade this also using upgrade command:

```
sudo ./aws/install --bin-dir /usr/local/bin --install-dir /usr/local/aws-cli --update
```

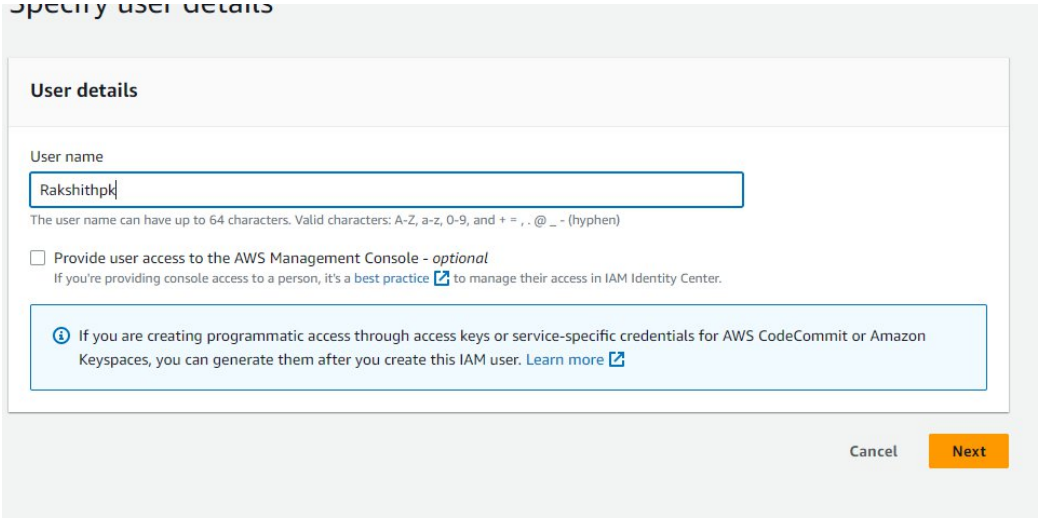
As you can see it is upgraded.

3. Navigate to AWS IAM service, and get AWS Access key and Secret Key to connect AWS with the AWS CLI

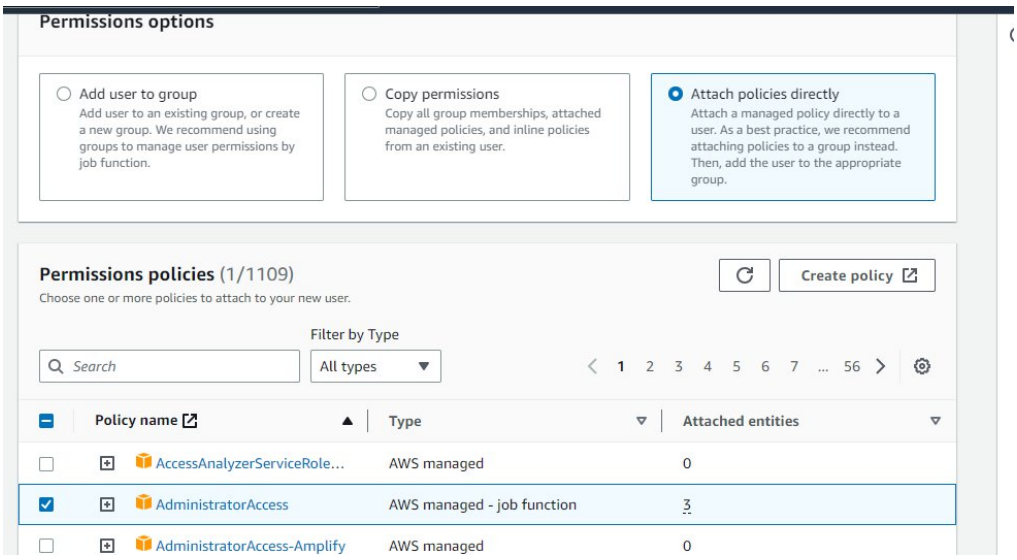
We will go to AWS IAM and then users, here we will create a user by clicking Add user button:



We give a name:

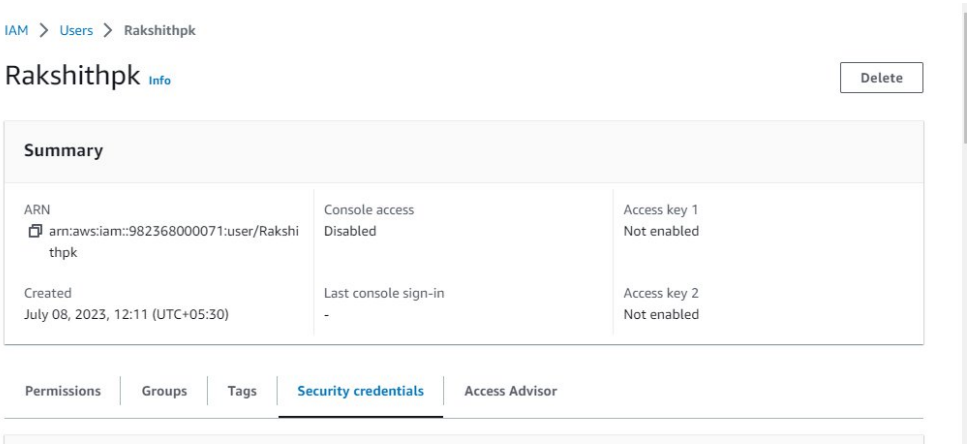


In permissions we will give administrator access:

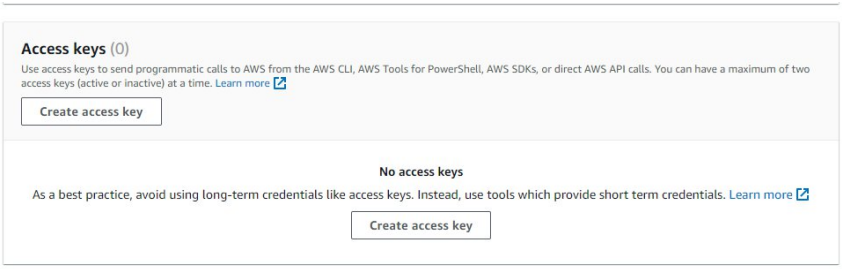


Then you create the user.

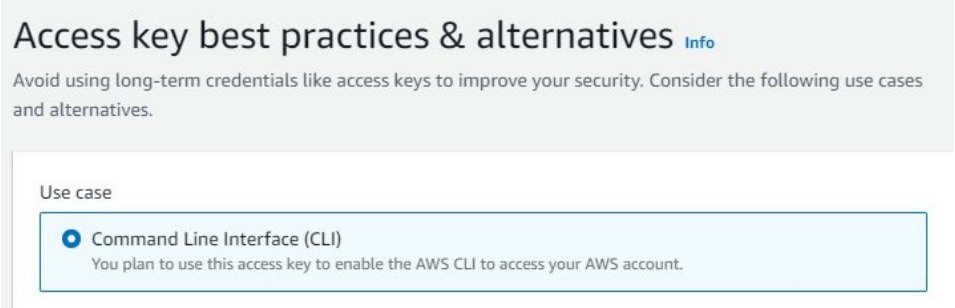
Once you create go to the users security credentials:



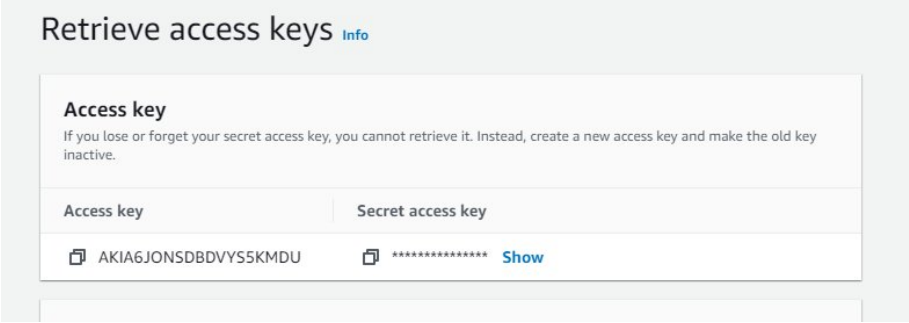
Here you should see access keys option, here click Create access key:



Chose your use case:



Once you create you will get the access key and the secret access key



Copy both the access key and the secret access key and save it securely.

4. Export the AWS Access Key, Secret Key to configure AWS CLI connectivity with AWS Cloud

Now we can configure aws on our system by command:

aws confirgure

```
File  Edit  View  Search  Terminal  Help
rakshithpk21gma@ip-172-31-24-67:~$ aws configure
AWS Access Key ID [None]: AKIA6JONSDBDVYS5KMDU
AWS Secret Access Key [None]: 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
Default region name [None]:
Default output format [None]:
rakshithpk21gma@ip-172-31-24-67:~$
```

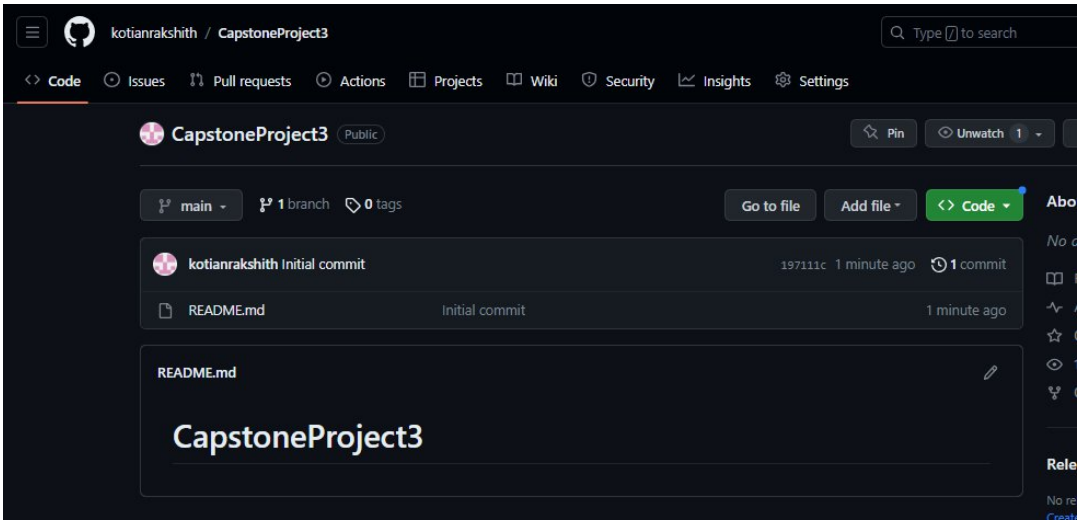
We can see that it is configured

Please note that you have to configure it also in the jenkins user as jenkins uses this user:

```
rakshithpk21gma@ip-172-31-28-215:~$ sudo su -s /bin/bash jenkins
jenkins@ip-172-31-28-215:/home/rakshithpk21gma$ aws configure
AWS Access Key ID [None]: AKIA6JONSDBDYWPLHJFP
AWS Secret Access Key [None]: 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
Default region name [None]:
Default output format [None]:
jenkins@ip-172-31-28-215:/home/rakshithpk21gma$ which terraform
/usr/bin/terraform
```

5. Setup Git and Github for storing the files.

First we are creating a new githut repository to store the link:



Now we will clone this account and start using this account and upload all the script files here:

Create a directory:

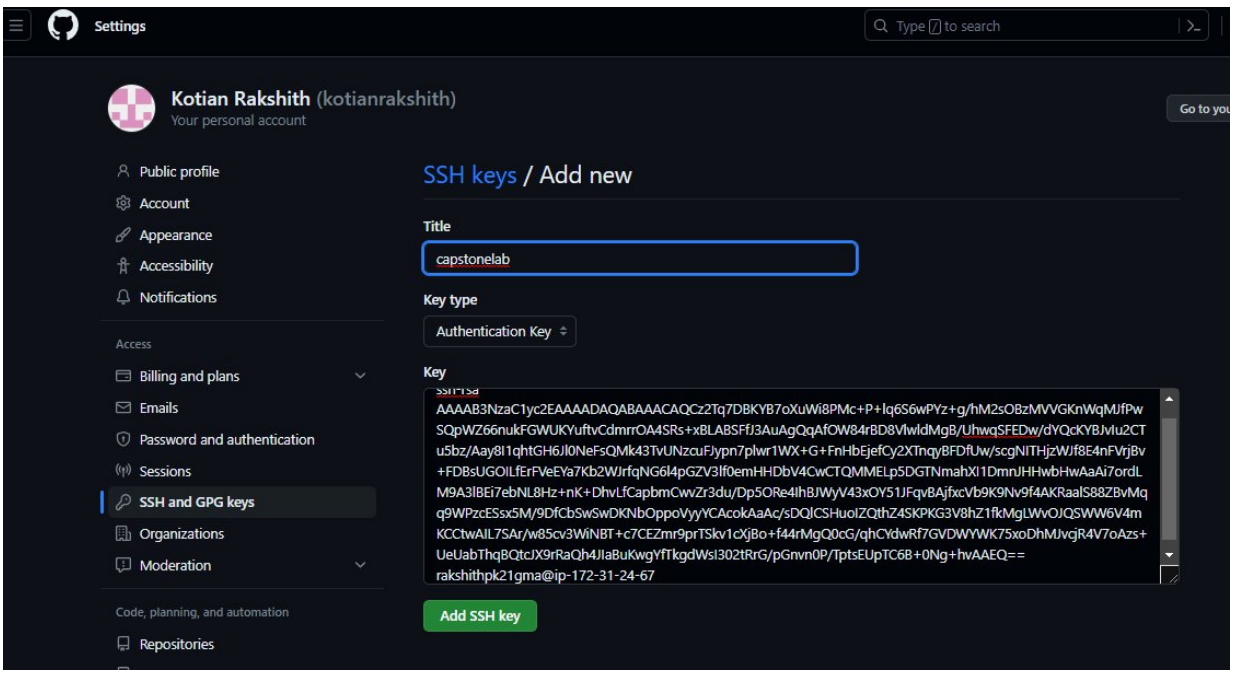
```
rakshithpk21gma@ip-172-31-24-67:~$ mkdir capstone
rakshithpk21gma@ip-172-31-24-67:~$ cd capstone
rakshithpk21gma@ip-172-31-24-67:~/capstone$ git clone git@git
```

We recieved an access error while cloning, so to solve the error we will genaerate a public key and add it to git hub.

```
rakshithpk21gma@ip-172-31-24-67:~/capstone$ ssh-keygen -t rsa -b 4096
Generating public/private rsa key pair.
Enter file in which to save the key (/home/rakshithpk21gma/.ssh/id_rsa):
/home/rakshithpk21gma/.ssh/id_rsa already exists.
Overwrite (y/n)? y
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/rakshithpk21gma/.ssh/id_rsa
Your public key has been saved in /home/rakshithpk21gma/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:wP1SZJqIU1uuyeyYYha+jD4TSJXh8WyV2+QH4y5AFrw rakshithpk21gma@ip-172-31-24-67
The key's randomart image is:
+---[RSA 4096]----+
|    o+.o.o o     |
|   .o+B.B O      |
|   ..=+* @ +     |
|  .   .E = * .   |
| o .    * S o    |
|.o . + . o       |
|  * o . .        |
| B o             |
|o.=              |
+----[SHA256]-----+
```

```
rakshithpk21gma@ip-172-31-24-67:~/capstone$ cat /home/rakshithpk21gma/.ssh/id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAACAQCz2Tq7DBKYB7oXuWi8PMc+P+lq6S6wPYz+g/hM2sOBzMVVGKnWq
MJfPwSQpWZ66nukFGWUKYuftvCdmrrOA4SRs+xBLABSFfJ3AuAgQqAfOW84rBD8VlwldMgB/UhwqSFEDw/dYQcKYB
JvIu2CTu5bz/Aay8I1qhtGH6Jl0NeFsQMk43TvUNzcuFJypn7plwr1WX+G+FnHbEjefCy2XTnqyBFDfUw/scgNITH
jzWJf8E4nFVrjBv+FDBsUGOILfErFVeEYa7Kb2WJrfqNG6l4pGZV3lf0emHHDbV4CwCTQMMELp5DGTNmahXI1DmnJ
HHwbHwAaAi7ordLM9A3lBEi7ebNL8Hz+nK+DhvLfCapbmCwvZr3du/Dp5ORe4IhBJWyV43xOY51JFqvBAjfxcVb9K
9Nv9f4AKRaalS88ZBvMqq9WPzcESsx5M/9DfCbSwSwDKNbOppoVyyYCAcokAaAc/sDQlCSHuoIZQthZ4SKPKG3V8h
Z1fkMgLWvOJQSWW6V4mKCCtwAIL7SAr/w85cv3WiNBT+c7CEZmr9prTSkv1cXjBo+f44rMgQ0cG/qhCYdwRf7GVDW
YWK75xoDhMJvcjR4V7oAzs+UeUabThqBQtcJX9rRaQh4JIaBuKwgYfTkgdWsI302tRrG/pGnvn0P/TptsEUpTC6B+
0Ng+hvAAEQ== rakshithpk21gma@ip-172-31-24-67
```

Add the key in the ssh section of the settings of github:



Once added we can clone:

```
rakshithpk21gma@ip-172-31-24-67:~/capstone$ git clone git@github.com:kotianrakshith/Capst
oneProject3.git
Cloning into 'CapstoneProject3'...
Warning: Permanently added the ECDSA host key for IP address '140.82.112.3' to the list o
f known hosts.
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (3/3), done.
rakshithpk21gma@ip-172-31-24-67:~/capstone$ ls
CapstoneProject3
```

Now you can see that we have local working repo for the remote repo

```
rakshithpk21gma@ip-172-31-24-67:~/capstone$ cd CapstoneProject3/
rakshithpk21gma@ip-172-31-24-67:~/capstone/CapstoneProject3$ ls
README.md
rakshithpk21gma@ip-172-31-24-67:~/capstone/CapstoneProject3$ git status
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
rakshithpk21gma@ip-172-31-24-67:~/capstone/CapstoneProject3$ git remote -v
origin  git@github.com:kotianrakshith/CapstoneProject3.git (fetch)
origin  git@github.com:kotianrakshith/CapstoneProject3.git (push)
rakshithpk21gma@ip-172-31-24-67:~/capstone/CapstoneProject3$
```

6. Create terraform scripts to create a new VM using auto scaling:

First we will create providers.tf file and we will add below code:

```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "4.49.0"
    }
  }
}

provider "aws" {
  region = "us-east-1"
}
```

```
File  Edit  View  Search  Terminal  Help
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "4.49.0"
    }
  }
}

provider "aws" {
  region = "us-east-1"
}
~
~
```

Now we create the vpc.tf for the VPC

```
resource "aws_vpc" "main" {
  cidr_block = "10.0.0.0/16"

  tags = {
    Name = "capstone-vpc"
  }
}
```

```
resource "aws_vpc" "main" {
  cidr_block = "10.0.0.0/16"

  tags = {
    Name = "capstone-vpc"
  }
}
```

Now let us create three subnets with the file name subnets.tf:

```
resource "aws_subnet" "subnet_a" {
  vpc_id     = aws_vpc.main.id
  cidr_block = "10.0.1.0/24"
  availability_zone = "us-east-1a"
  tags = {
    Name = "subnet_a"
  }
}
```

```
resource "aws_subnet" "subnet_b" {
  vpc_id     = aws_vpc.main.id
  cidr_block = "10.0.2.0/24"
  availability_zone = "us-east-1b"
  tags = {
    Name = "subnet_b"
  }
}
resource "aws_subnet" "subnet_c" {
  vpc_id     = aws_vpc.main.id
  cidr_block = "10.0.3.0/24"
  availability_zone = "us-east-1c"

  tags = {
    Name = "subnet_c"
  }
}
```

As of now we have created vpc, subnets terraform file:

```
File  Edit  View  Search  Terminal  Help
rakshithpk21gma@ip-172-31-24-67:~/capstone/CapstoneProject3$ vim providers.tf
rakshithpk21gma@ip-172-31-24-67:~/capstone/CapstoneProject3$ vim vpc.tf
rakshithpk21gma@ip-172-31-24-67:~/capstone/CapstoneProject3$ vim subnets.tf
rakshithpk21gma@ip-172-31-24-67:~/capstone/CapstoneProject3$
```

Now let us create internte gate terraform file: internetgateway.tf

```
resource "aws_internet_gateway" "main-igw" {
  vpc_id = aws_vpc.main.id

  tags = {
    Name = "capstone-igw"
  }
}
```

Now let us create the route table:routetable.tf

In this below code we will allow the route of all the internet(0.0.0.0/0)

```
resource "aws_route_table" "public_rt" {
  vpc_id = aws_vpc.main.id

  route {
    cidr_block = "0.0.0.0/0"
    gateway_id = aws_internet_gateway.main-igw.id
  }

  tags = {
    Name = "capstone_public_rt"
  }
}
```

```
resource "aws_route_table" "public_rt" {
  vpc_id = aws_vpc.main.id
  route {
    cidr_block = "0.0.0.0/0"
    gateway_id = aws_internet_gateway.main-igw.id
  }
  tags = {
    Name = "capstone_public_rt"
  }
}
~
```

Now we will have to do the route table association as the route table needs to be connected with all the public subnets:

We will use the file :

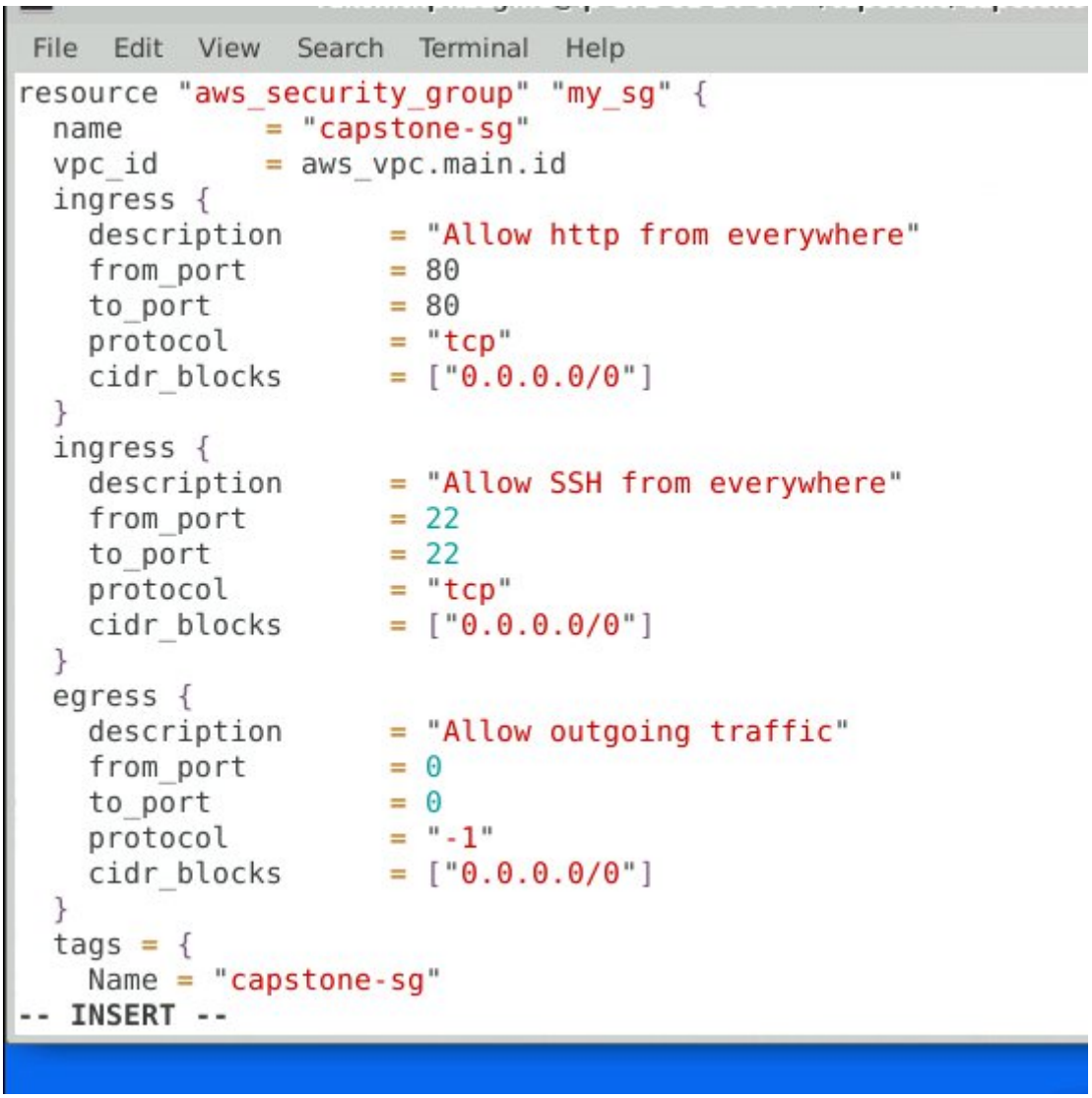
```
resource "aws_route_table_association" "a" {
 subnet_id      = aws_subnet.subnet_a.id
 route_table_id = aws_route_table.public_rt.id
}
resource "aws_route_table_association" "b" {
 subnet_id      = aws_subnet.subnet_b.id
 route_table_id = aws_route_table.public_rt.id
}
resource "aws_route_table_association" "c" {
 subnet_id      = aws_subnet.subnet_c.id
 route_table_id = aws_route_table.public_rt.id
}
```

```
File  Edit  View  Search  Terminal  Help
resource "aws_route_table_association" "a" {
  subnet_id      = aws_subnet.subnet_a.id
  route_table_id = aws_route_table.public_rt.id
}
resource "aws_route_table_association" "b" {
  subnet_id      = aws_subnet.subnet_b.id
  route_table_id = aws_route_table.public_rt.id
}
resource "aws_route_table_association" "c" {
  subnet_id      = aws_subnet.subnet_c.id
  route_table_id = aws_route_table.public_rt.id
}
~
```

Now we have to do secrity groups before we got to autoscaling

securitygroup.tf

```
resource "aws_security_group" "my_sg" {
  name        = "capstone-sg"
  vpc_id      = aws_vpc.main.id

  ingress {
    description      = "Allow http from everywhere"
    from_port        = 80
    to_port          = 80
    protocol         = "tcp"
    cidr_blocks      = ["0.0.0.0/0"]
  }

  ingress {
    description      = "Allow SSH from everywhere"
    from_port        = 22
    to_port          = 22
    protocol         = "tcp"
    cidr_blocks      = ["0.0.0.0/0"]
  }

  egress {
    description      = "Allow outgoing traffic"
    from_port        = 0
    to_port          = 0
    protocol         = "-1"
    cidr_blocks      = ["0.0.0.0/0"]
  }

  tags = {
    Name = "capstone-sg"
  }
}
```



Now we will create autoscaling.tf

This will have launch template and autoscaling group required for creation and scaling of the VMs:

```
resource "aws_launch_template" "my_template" {
  name = "capstone_template"
  image_id = "ami-06ca3ca175f37dd66"
  instance_type = "t2.micro"
  network_interfaces {
    associate_public_ip_address = true
    security_groups = [aws_security_group.my_sg.id]
  }
}

resource "aws_autoscaling_group" "my_asg" {
```

```
  name                 = "capstone_asg"

  max_size             = 3

  min_size             = 1

  health_check_type    = "EC2"

  desired_capacity     = 1

  vpc_zone_identifier  = [aws_subnet.subnet_a.id, aws_subnet.subnet_b.id,
aws_subnet.subnet_c.id]

  launch_template {

    id      = aws_launch_template.my_template.id

  }

}

resource "aws_autoscaling_policy" "my_as_policy" {

name = "capstone_as_policy"

policy_type = "TargetTrackingScaling"

autoscaling_group_name = "\${aws_autoscaling_group.my_asg.name}"

estimated_instance_warmup = 200

target_tracking_configuration {

predefined_metric_specification {

predefined_metric_type = "ASGAverageCPUUtilization"

}

   target_value = "20"

}

}
```

```
resource "aws_launch_template" "my_template" {
  name = "capstone_template"
  image_id = "ami-06ca3ca175f37dd66"
  instance_type = "t2.micro"
  network_interfaces {
    associate_public_ip_address = true
    security_groups = [aws_security_group.my_sg.id]
  }
}

resource "aws_autoscaling_group" "my_asg" {
  name                      = "capstone_asg"
  max_size                  = 3
  min_size                  = 1
  health_check_type         = "EC2"
  desired_capacity          = 1
  vpc_zone_identifier       = [aws_subnet.subnet_a.id, aws_subnet.subnet_b.id, aws_subnet.subnet_c.id]
  launch_template {
    id      = aws_launch_template.my_template.id
  }
}

resource "aws_autoscaling_policy" "my_as_policy" {
name = "capstone_as_policy"
policy_type = "TargetTrackingScaling"
autoscaling_group_name = aws_autoscaling_group.my_asg.name
estimated_instance_warmup = 200
target_tracking_configuration {
predefined_metric_specification {
predefined_metric_type = "ASGAverageCPUUtilization"
```

Now that we have created all the files:

```
File  Edit  View  Search  Terminal  Help
rakshithpk21gma@ip-172-31-24-67:~/capstone/CapstoneProject3$ vim providers.tf
rakshithpk21gma@ip-172-31-24-67:~/capstone/CapstoneProject3$ vim vpc.tf
rakshithpk21gma@ip-172-31-24-67:~/capstone/CapstoneProject3$ vim subnets.tf
rakshithpk21gma@ip-172-31-24-67:~/capstone/CapstoneProject3$ vim internetgateway.tf
rakshithpk21gma@ip-172-31-24-67:~/capstone/CapstoneProject3$ vim routetable.tf
rakshithpk21gma@ip-172-31-24-67:~/capstone/CapstoneProject3$ vim Route_table_association_
with_public_subnets.tf
rakshithpk21gma@ip-172-31-24-67:~/capstone/CapstoneProject3$ vim securitygroup.tf
rakshithpk21gma@ip-172-31-24-67:~/capstone/CapstoneProject3$ vim autoscaling.tf
rakshithpk21gma@ip-172-31-24-67:~/capstone/CapstoneProject3$
```

Lets push it to github:

```
rakshithpk21gma@ip-172-31-24-67:~/capstone/CapstoneProject3$ git status
On branch main
Your branch is up to date with 'origin/main'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        Route_table_association_with_public_subnets.tf
        autoscaling.tf
        internetgateway.tf
        providers.tf
        routetable.tf
        securitygroup.tf
        subnets.tf
        vpc.tf

nothing added to commit but untracked files present (use "git add" to track)
rakshithpk21gma@ip-172-31-24-67:~/capstone/CapstoneProject3$
```

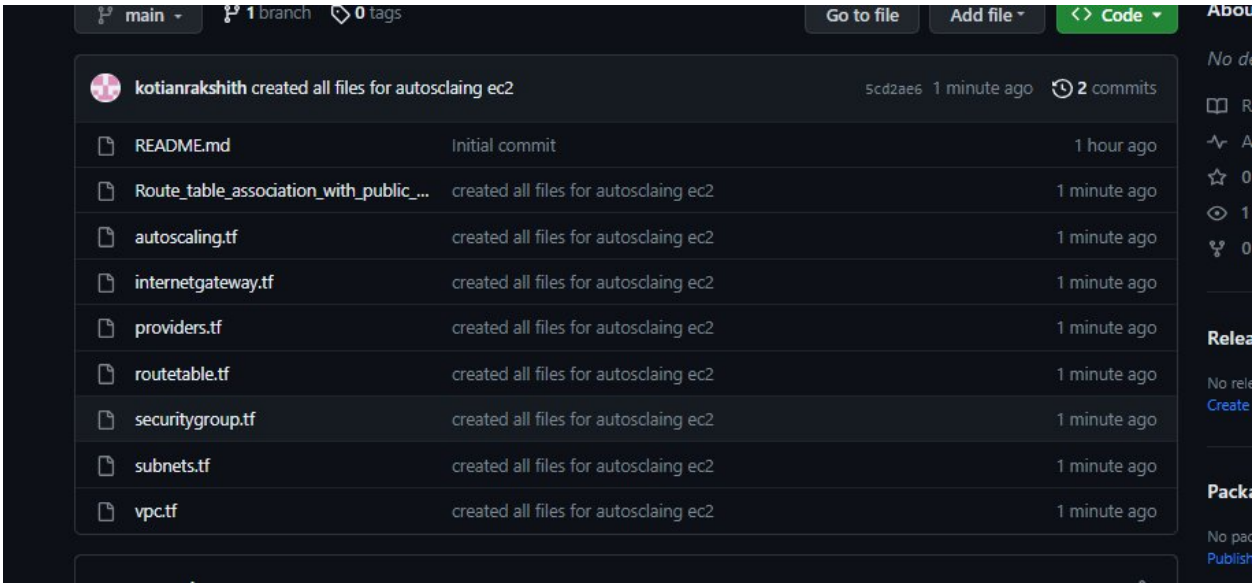
```
rakshithpk21gma@ip-172-31-24-67:~/capstone/CapstoneProject3$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   Route_table_association_with_public_subnets.tf
        new file:   autoscaling.tf
        new file:   internetgateway.tf
        new file:   providers.tf
        new file:   routetable.tf
        new file:   securitygroup.tf
        new file:   subnets.tf
        new file:   vpc.tf
```

```
rakshithpk21gma@ip-172-31-24-67:~/capstone/CapstoneProject3$ git commit -m "created all files for autosclaing ec2"
[main 5cd2ae6] created all files for autosclaing ec2
 8 files changed, 120 insertions(+)
 create mode 100644 Route_table_association_with_public_subnets.tf
 create mode 100644 autoscaling.tf
 create mode 100644 internetgateway.tf
 create mode 100644 providers.tf
 create mode 100644 routetable.tf
 create mode 100644 securitygroup.tf
 create mode 100644 subnets.tf
 create mode 100644 vpc.tf
```

Push:

```
rakshithpk21gma@ip-172-31-24-67:~/capstone/CapstoneProject3$ git push origin main
Enumerating objects: 11, done.
Counting objects: 100% (11/11), done.
Delta compression using up to 4 threads
Compressing objects: 100% (10/10), done.
Writing objects: 100% (10/10), 1.84 KiB | 1.84 MiB/s, done.
Total 10 (delta 0), reused 0 (delta 0), pack-reused 0
To github.com:kotianrakshith/CapstoneProject3.git
   197111c..5cd2ae6  main -> main
rakshithpk21gma@ip-172-31-24-67:~/capstone/CapstoneProject3$
```

Now we can see all the files in the github repository:



This is only for highly availble ec2 instance.

Now we need to write one more for EKS cluster

For EKS cluster to work first we need to create a role for both eks cluster and nodes and then add proper policies for the same

So first we will create terraform file for this: rolepolicy.tf

```
resource "aws_iam_role" "eks_cluster" {
  name = "capstone-ekscluster"
  assume_role_policy = <<POLICY
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "eks.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
POLICY
}
```

```
resource "aws_iam_role_policy_attachment" "AmazonEKSClusterPolicy" {
  policy_arn = "arn:aws:iam::aws:policy/AmazonEKSClusterPolicy"
  role       = aws_iam_role.eks_cluster.name
}

resource "aws_iam_role_policy_attachment" "AmazonEKSServicePolicy" {
  policy_arn = "arn:aws:iam::aws:policy/AmazonEKSServicePolicy"
  role       = aws_iam_role.eks_cluster.name
}

resource "aws_iam_role" "eks_nodes" {
  name = "capstone-eksnodes"
  assume_role_policy = <<POLICY
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "ec2.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
POLICY
}

resource "aws_iam_role_policy_attachment" "AmazonEKSWorkerNodePolicy" {
  policy_arn = "arn:aws:iam::aws:policy/AmazonEKSWorkerNodePolicy"
  role       = aws_iam_role.eks_nodes.name
}
```

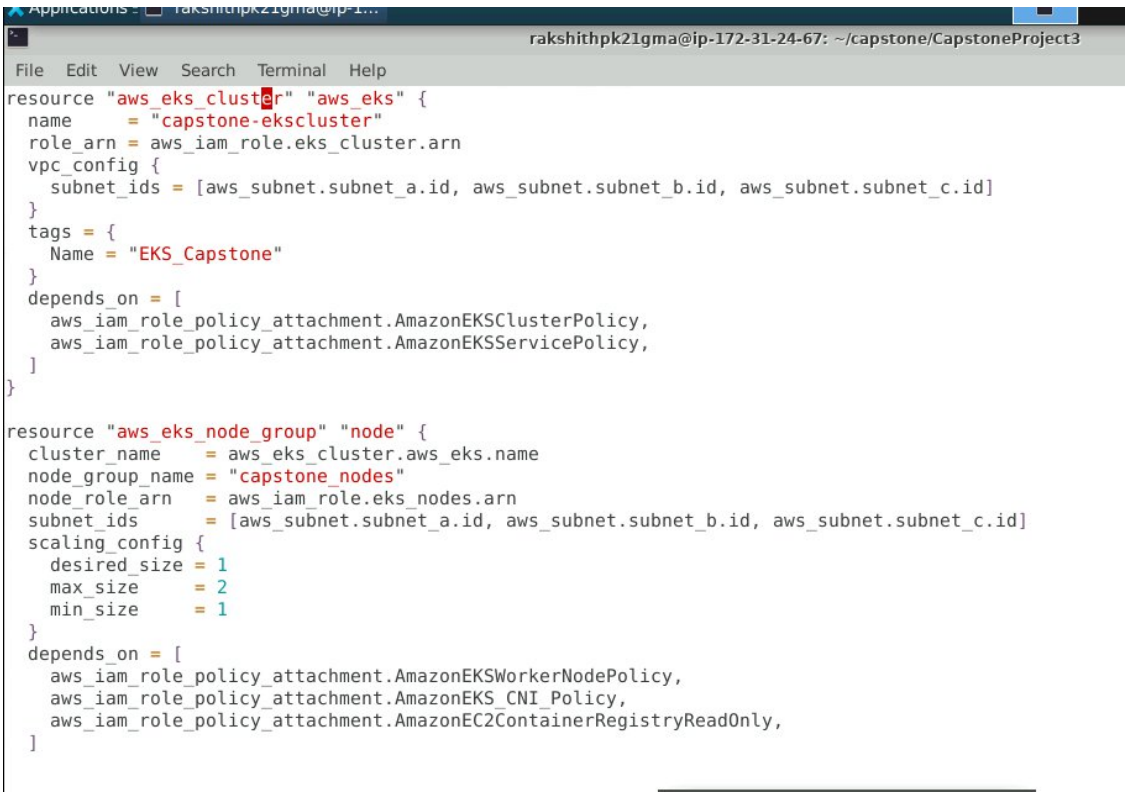
```
resource "aws_iam_role_policy_attachment" "AmazonEKS_CNI_Policy" {
  policy_arn = "arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy"
  role       = aws_iam_role.eks_nodes.name
}

resource "aws_iam_role_policy_attachment" "AmazonEC2ContainerRegistryReadOnly" {
  policy_arn = "arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryReadOnly"
  role       = aws_iam_role.eks_nodes.name
}
```

```
resource "aws_iam_role" "eks_cluster" {
  name = "capstone-ekscluster"
  assume_role_policy = <<POLICY
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "eks.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
POLICY
}

resource "aws_iam_role_policy_attachment" "AmazonEKSClusterPolicy" {
  policy_arn = "arn:aws:iam::aws:policy/AmazonEKSClusterPolicy"
  role       = aws_iam_role.eks_cluster.name
}

resource "aws_iam_role_policy_attachment" "AmazonEKSServicePolicy" {
  policy_arn = "arn:aws:iam::aws:policy/AmazonEKSServicePolicy"
  role       = aws_iam_role.eks_cluster.name
}

resource "aws_iam_role" "eks_nodes" {
  name = "capstone-eksnodes"
```

Then we will create the eks cluser and node in the file : eks.tf

```
resource "aws_eks_cluster" "aws_eks" {
  name     = "capstone-ekscluster"
  role_arn = aws_iam_role.eks_cluster.arn
  vpc_config {
    subnet_ids = [aws_subnet.subnet_a.id, aws_subnet.subnet_b.id, aws_subnet.subnet_c.id]
  }
  tags = {
    Name = "EKS_Capstone"
  }
```

```
  depends_on = [
    aws_iam_role_policy_attachment.AmazonEKSClusterPolicy,
    aws_iam_role_policy_attachment.AmazonEKSServicePolicy,
  ]
 }
 resource "aws_eks_node_group" "node" {
  cluster_name    = aws_eks_cluster.aws_eks.name
  node_group_name = "capstone_nodes"
  node_role_arn   = aws_iam_role.eks_nodes.arn
  subnet_ids      = [aws_subnet.subnet_a.id, aws_subnet.subnet_b.id, aws_subnet.subnet_c.id]
  scaling_config {
    desired_size = 1
    max_size     = 2
    min_size     = 1
  }
  depends_on = [
    aws_iam_role_policy_attachment.AmazonEKSWorkerNodePolicy,
    aws_iam_role_policy_attachment.AmazonEKS_CNI_Policy,
    aws_iam_role_policy_attachment.AmazonEC2ContainerRegistryReadOnly,
  ]
 }
```



```
rakshithpk21gma@ip-172-31-24-67:~/capstone/CapstoneProject3$ vim rolepolicy.tf
rakshithpk21gma@ip-172-31-24-67:~/capstone/CapstoneProject3$ vim eks.tf
```

Now we will add , commit and push this also to github

```
rakshithpk21gma@ip-172-31-24-67:~/capstone/CapstoneProject3$ git add rolepolicy.tf
rakshithpk21gma@ip-172-31-24-67:~/capstone/CapstoneProject3$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   eks.tf
        new file:   rolepolicy.tf
```

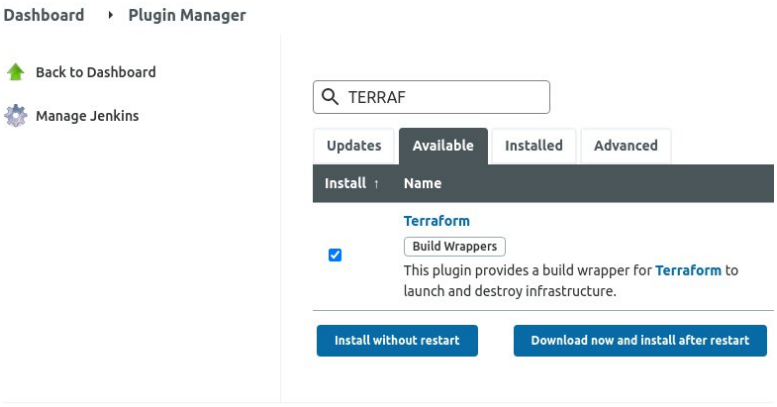
```
akshithpk21gma@ip-172-31-24-67:~/capstone/CapstoneProject3$ git commit -m "added and modified required files"
main 805f61a] added and modified required files
3 files changed, 94 insertions(+)
create mode 100644 eks.tf
create mode 100644 rolepolicy.tf
akshithpk21gma@ip-172-31-24-67:~/capstone/CapstoneProject3$ git push origin main
numerating objects: 7, done.
ounting objects: 100% (7/7), done.
```

Now we have all the files in the github:

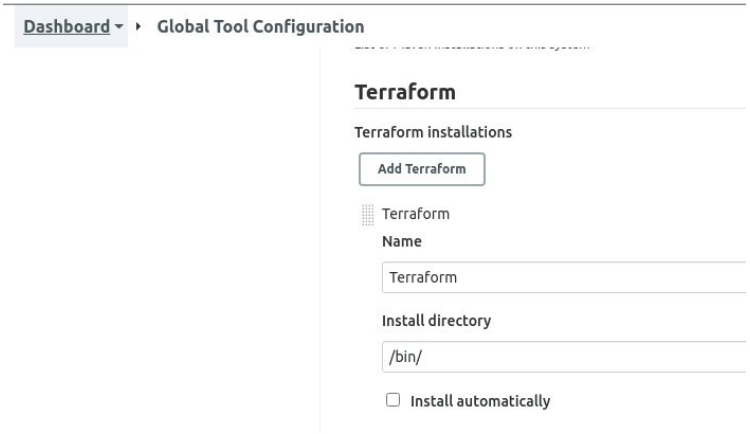
7. Execute terraform scripts:

We will use jenkins to checkout the github repository and execute the terrraform commands.

First in the jenkins we will install terraform plugin:

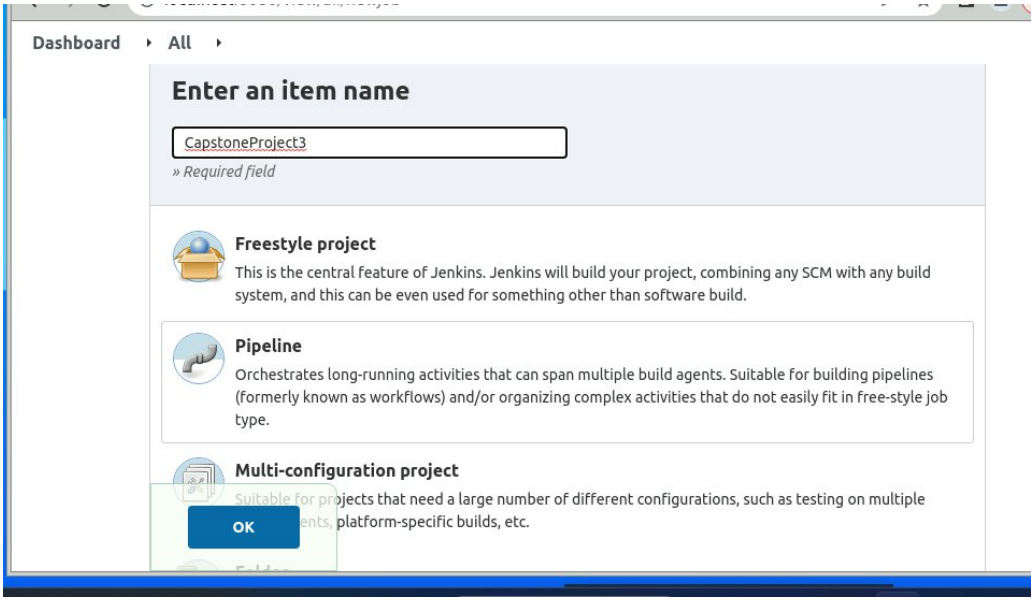


Also in the global tool configuration add terraform details:

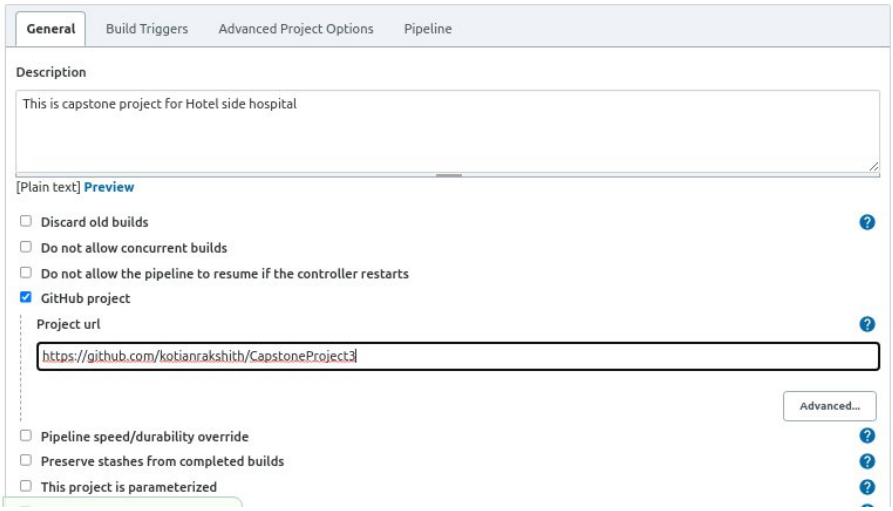


Now we will can write the checkout and apply as steps in the pipline

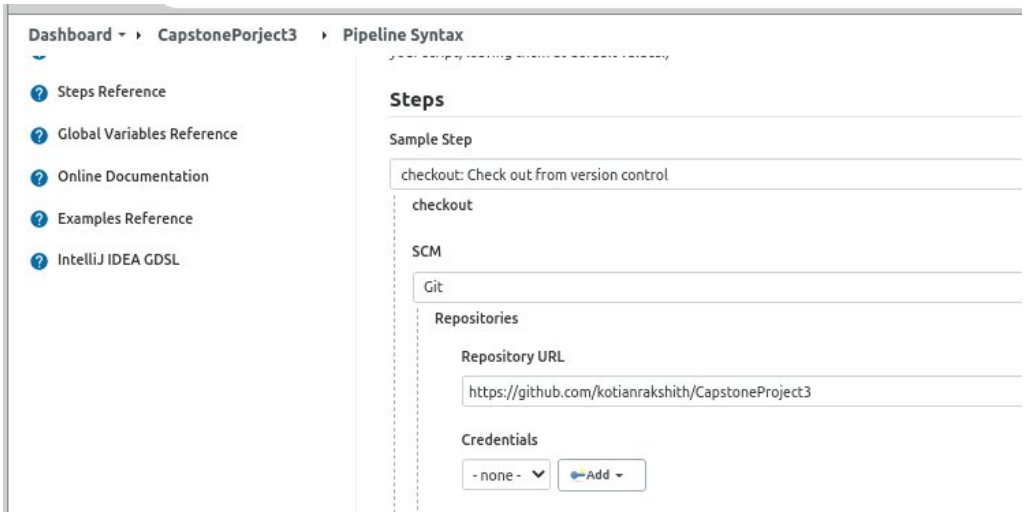
Create new pipeline project in jenkins

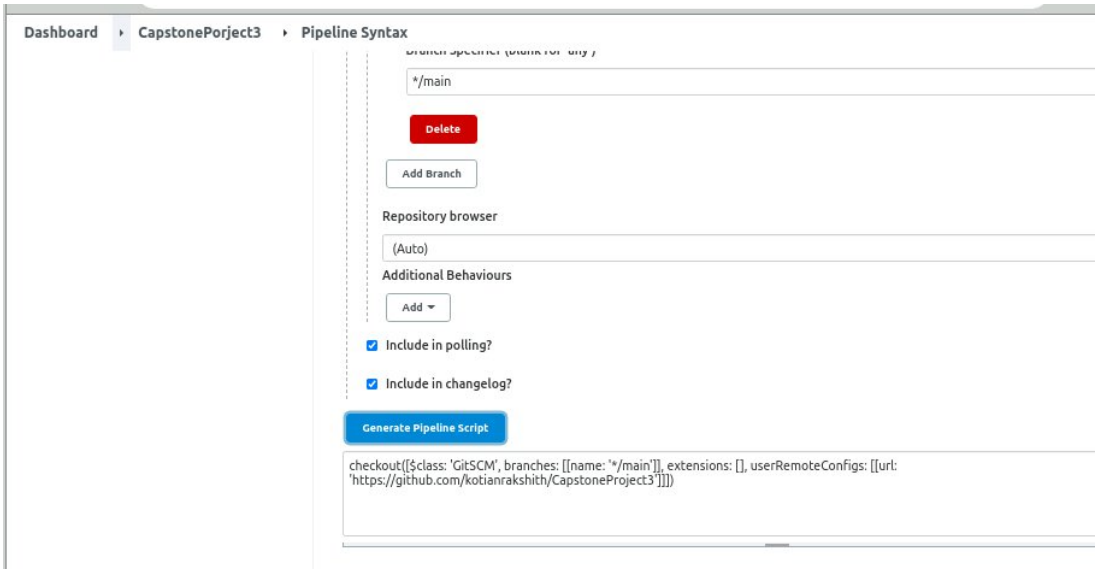


Give proper description and provide git hub project url:

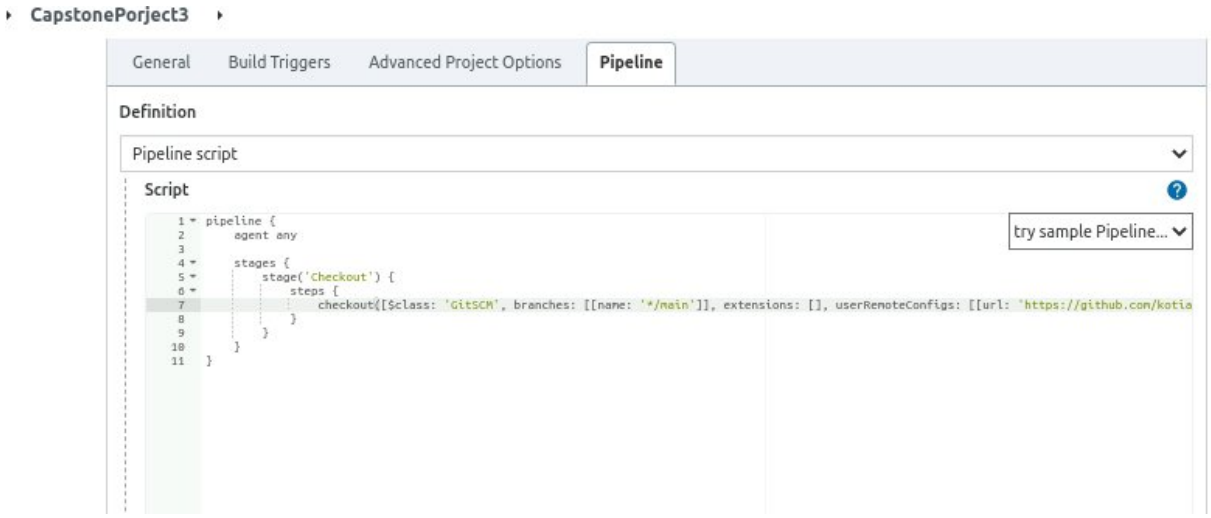


To get the checkout script we will use pipeline syntax





So we add the script we generate in the checkout stage:



Now we add init and apply stage to the pipeline as well

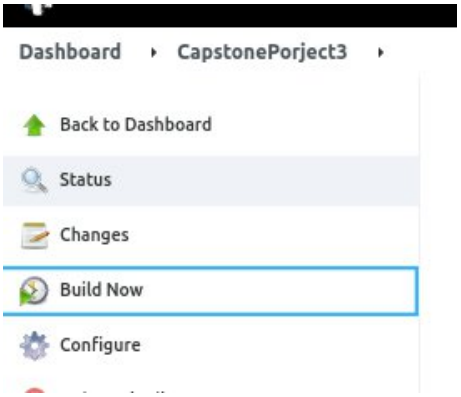
Now we have the final script:

```
pipeline {
   agent any
   tools {
      terraform 'Terraform'
   }
   stages {
      stage('Checkout') {
         steps {
            checkout([$class: 'GitSCM', branches: [[name: '*/main']], extensions: [],
userRemoteConfigs: [[url: 'https://github.com/kotianrakshith/CapstoneProject3']]])
         }
      }
      stage('Terraform init') {
         steps {
            sh 'terraform init'
```

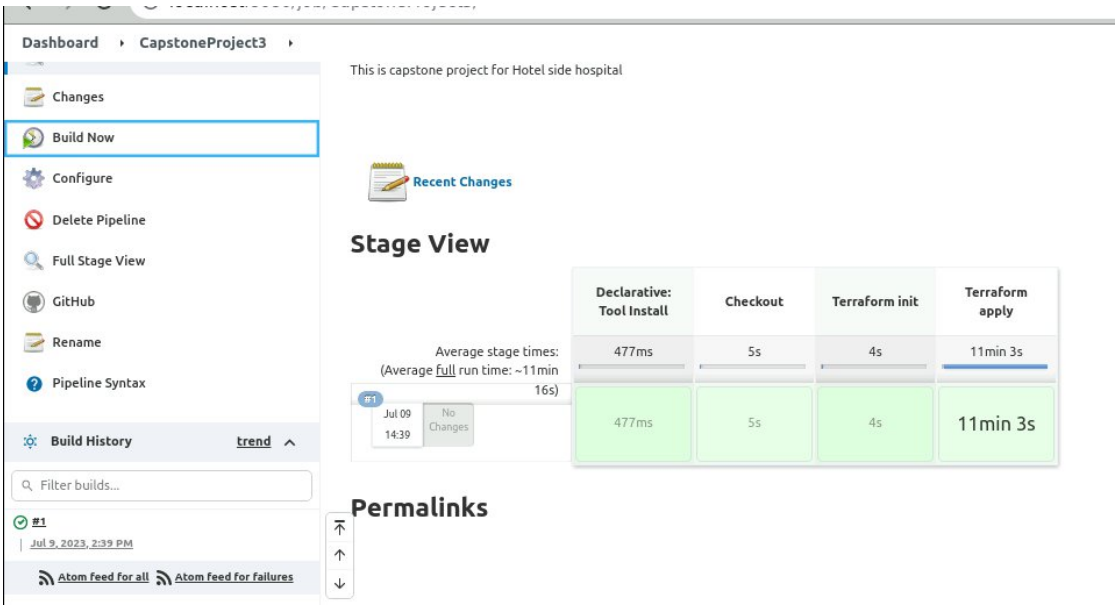
```
            }
        }
        stage('Terraform apply') {
            steps {
                sh 'terraform apply --auto-approve'
            }
        }
    }

}
```

We can save this as Jenkinsfile in the git so it can be used easily for the future.

Once saved we click on build now to start the pipeline:

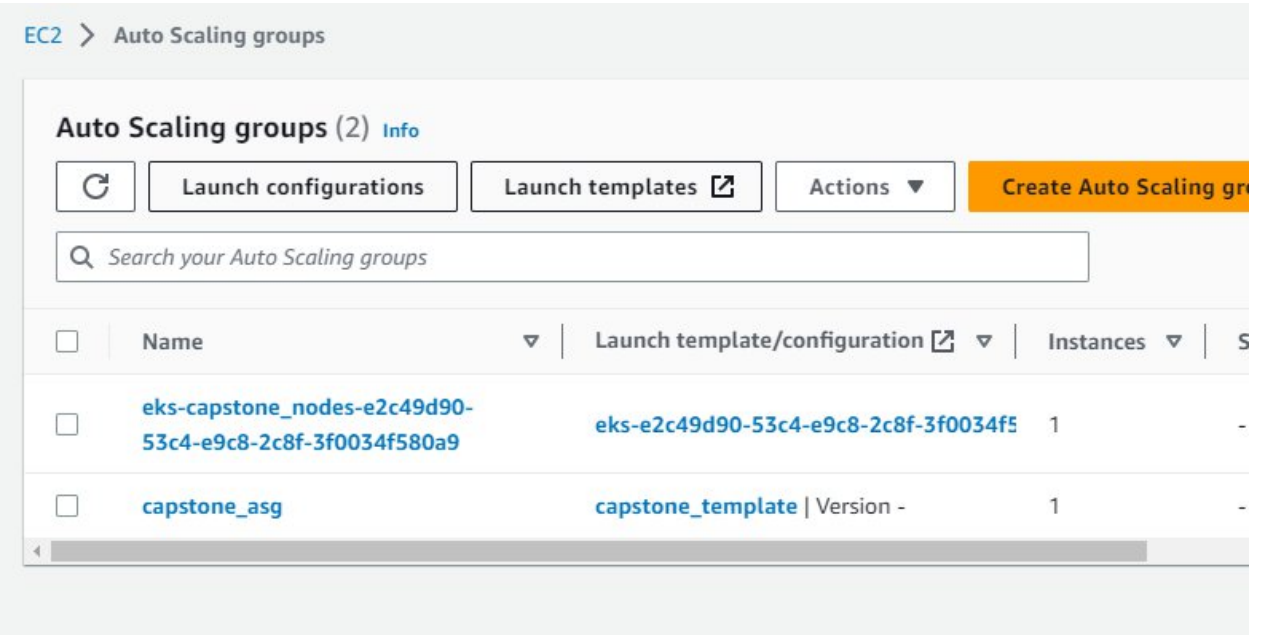


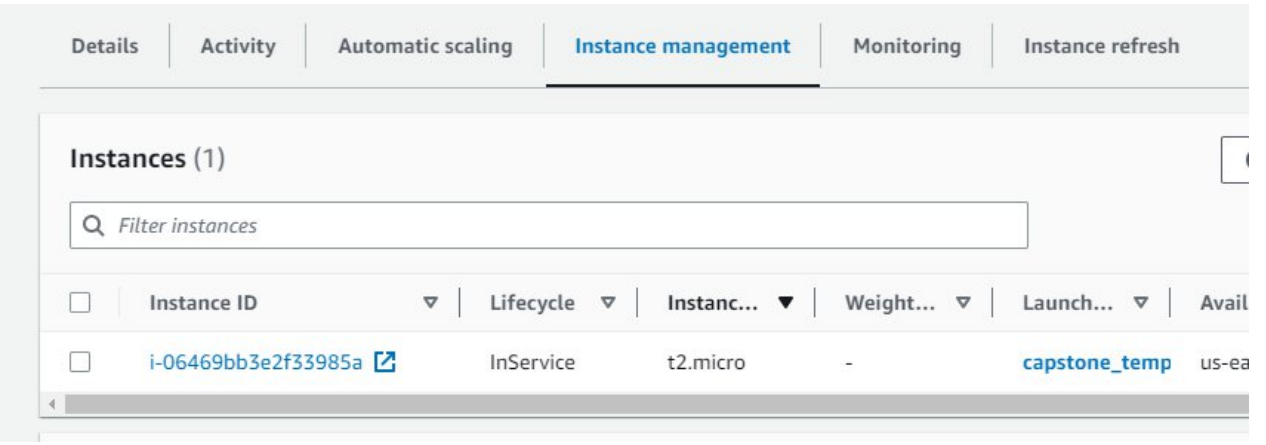
We can see that it has run successfully:

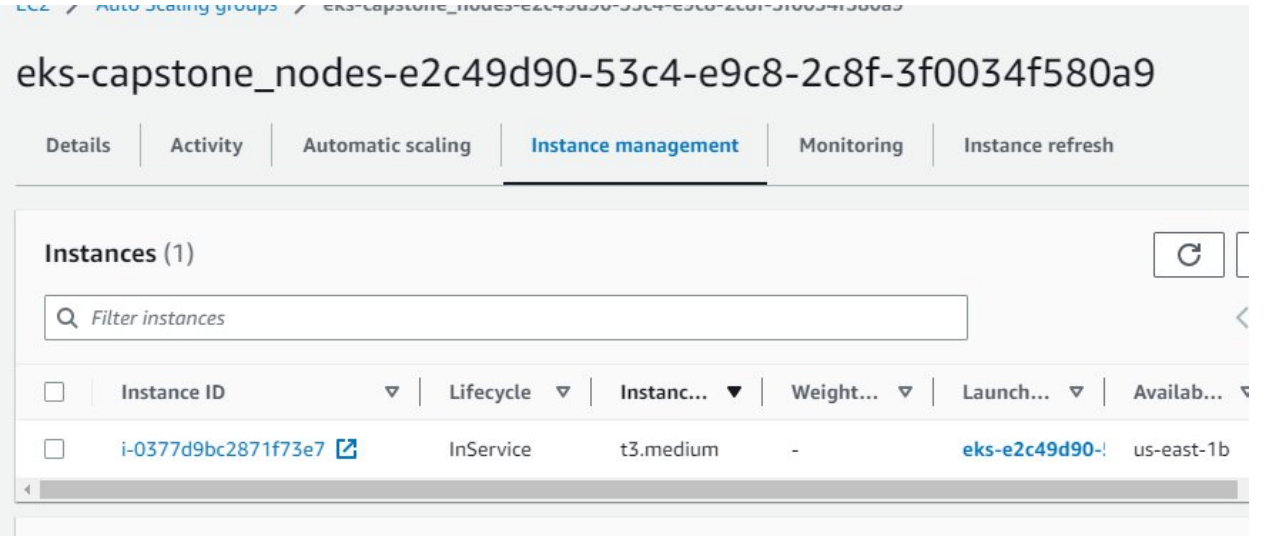


8. **Checking the deployment in AWS:**

We see in the autoscaling groups that there are two autoscaling groups, one for EKS and one for EC2 as we correctly deployed:



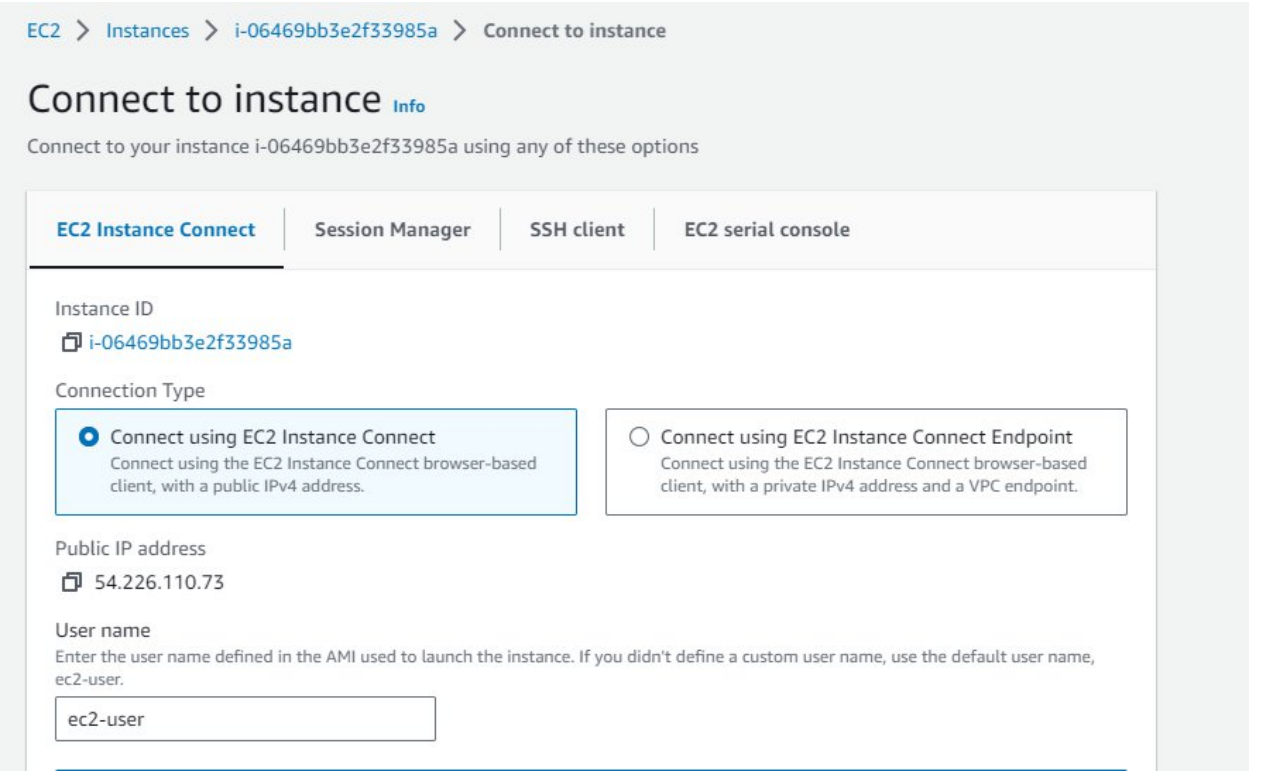




Each have one instance.

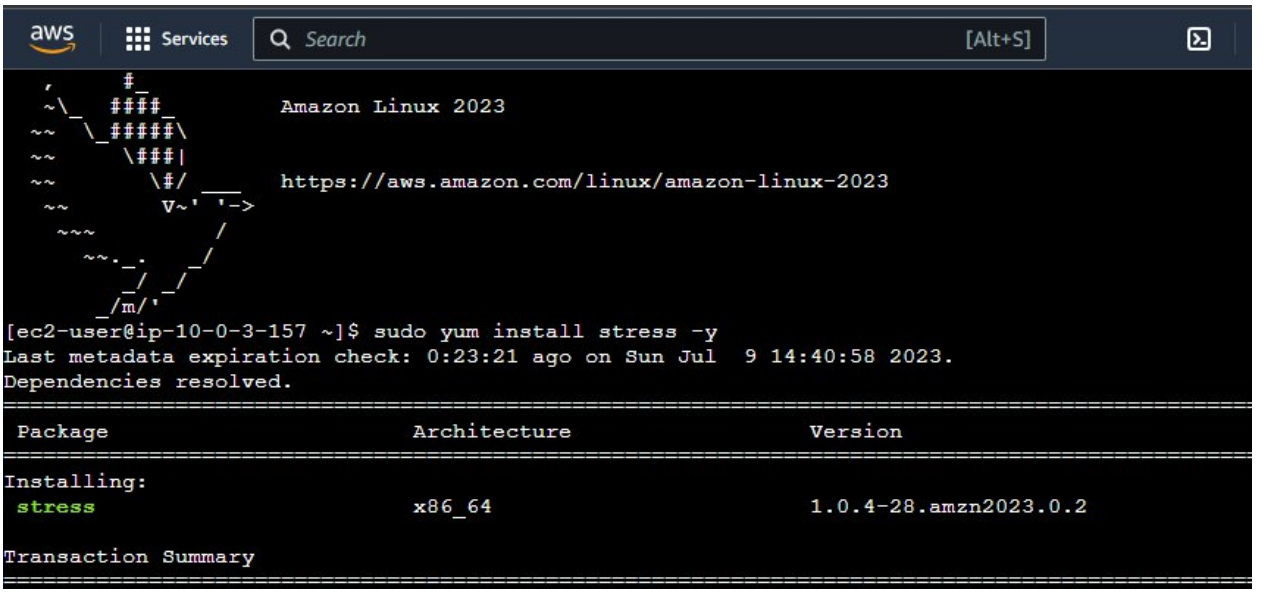
9. Connect to an instance and install the stress utility:

We will connect to one of the instance:



Here we will install the stress tool:

```
sudo yum install stress -y
```



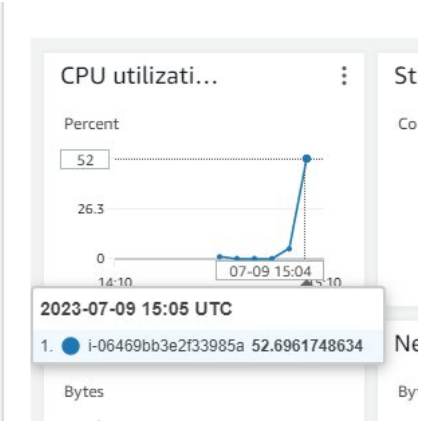
10. <u>Validate if autoscaling is working by putting load on autoscaling group:</u>

Now we will run the stress command to put load on the system:

```
sudo stress --cpu 8 -v --timeout 3000s
```

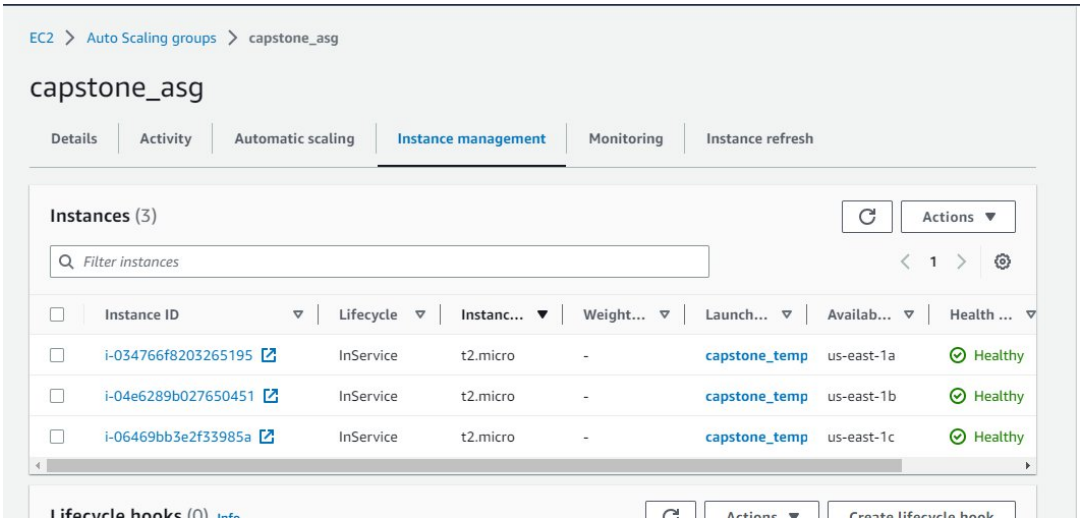
```
[ec2-user@ip-10-0-3-157 ~]$ sudo stress --cpu 8 -v --timeout 3000s
stress: info: [26309] dispatching hogs: 8 cpu, 0 io, 0 vm, 0 hdd
stress: dbug: [26309] using backoff sleep of 24000us
stress: dbug: [26309] setting timeout to 3000s
stress: dbug: [26309] --> hogcpu worker 8 [26310] forked
stress: dbug: [26309] using backoff sleep of 21000us
stress: dbug: [26309] setting timeout to 3000s
stress: dbug: [26309] --> hogcpu worker 7 [26311] forked
stress: dbug: [26309] using backoff sleep of 18000us
stress: dbug: [26309] setting timeout to 3000s
stress: dbug: [26309] --> hogcpu worker 6 [26312] forked
stress: dbug: [26309] using backoff sleep of 15000us
stress: dbug: [26309] setting timeout to 3000s
stress: dbug: [26309] --> hogcpu worker 5 [26313] forked
stress: dbug: [26309] using backoff sleep of 12000us
stress: dbug: [26309] setting timeout to 3000s
stress: dbug: [26309] --> hogcpu worker 4 [26314] forked
stress: dbug: [26309] using backoff sleep of 9000us
stress: dbug: [26309] setting timeout to 3000s
stress: dbug: [26309] --> hogcpu worker 3 [26315] forked
stress: dbug: [26309] using backoff sleep of 6000us
```

After we run for some time let us check the CPU utlization:



Cpu utlization is more than our limit.

Now if we check the autoscaling group:

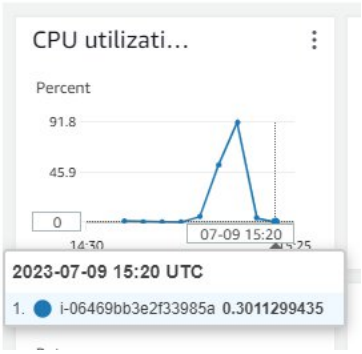


We can see that 3 instance has been deployed as it is our max limit.

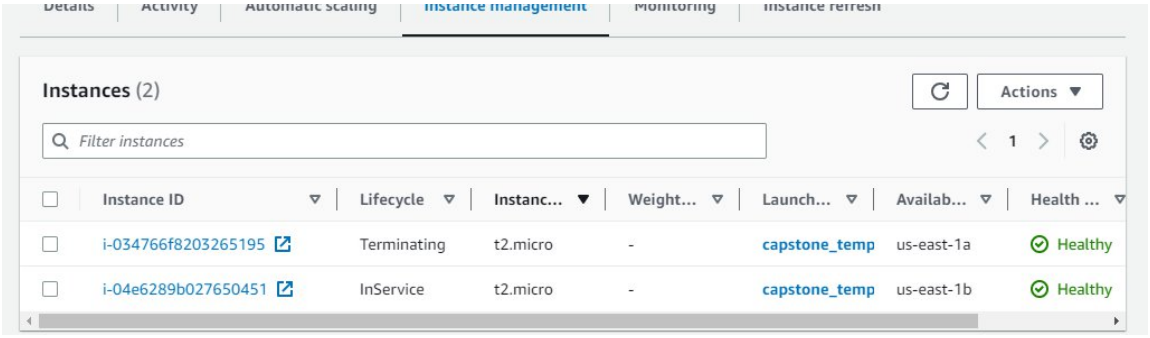
Now let us stop the stress test and wait:

```
stress: dbug: [26309] using backoff sleep of 3000us
stress: dbug: [26309] setting timeout to 3000s
stress: dbug: [26309] --> hogcpu worker 1 [26317] forked
^C
[ec2-user@ip-10-0-3-157 ~]$
```

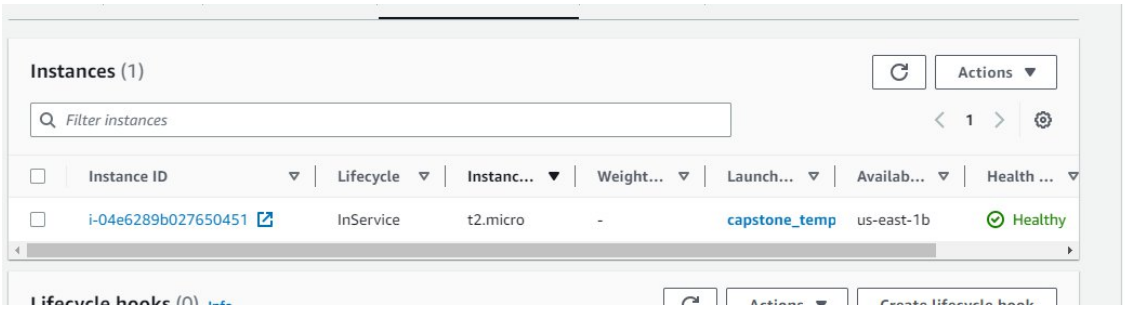
We can see that cpu utlization falls eventually to zero



Now the instance has decreased to two:



Eventually we will have only one:



So we have confimed that autoscaling works.

That concludes our project. As per the project we deployed EC2 instances and EKS with autoscaling and we checked that autoscaling works after we performed the stress test.