

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

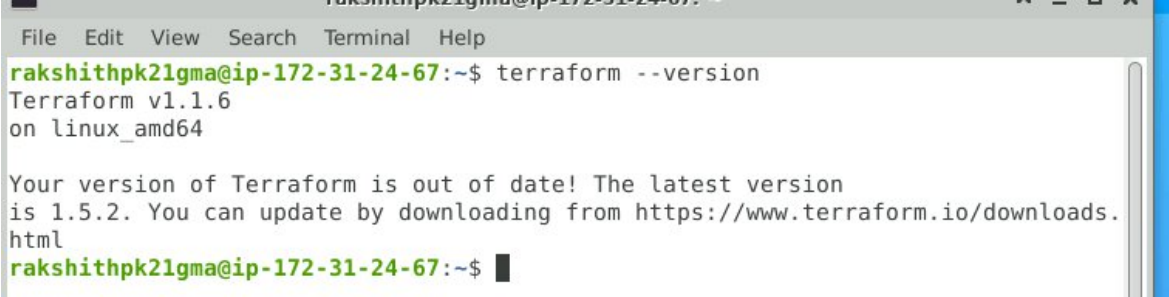
Steps performed:

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

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

```
terraform --version
```

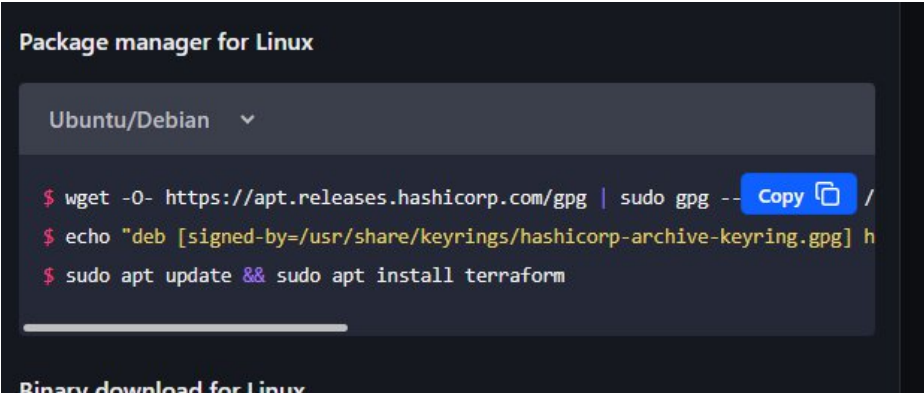
As we can see it is already installed:



```
rakshithpk21gma@ip-172-31-24-67:~$ terraform --version
Terraform v1.1.6
on linux_amd64

Your version of Terraform is out of date! The latest version
is 1.5.2. You can update by downloading from https://www.terraform.io/downloads.
html
rakshithpk21gma@ip-172-31-24-67:~$
```

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:



```
Package manager for Linux

Ubuntu/Debian ▾

$ wget -O- https://apt.releases.hashicorp.com/gpg | sudo gpg -- Copy /
$ echo "deb [signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg] h
$ sudo apt update && sudo apt install terraform

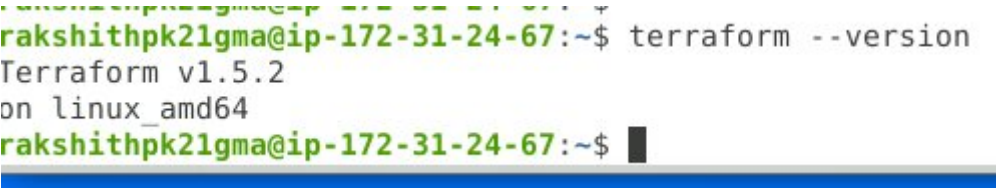
Binary download for Linux
```

```
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



```
rakshithpk21gma@ip-172-31-24-67:~$ terraform --version
Terraform v1.5.2
on linux_amd64
rakshithpk21gma@ip-172-31-24-67:~$
```

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

```
inflating: aws/dist/docutils/writers/html4css1/html4css1.css  
rakshithpk21gma@ip-172-31-24-67:~$  
rakshithpk21gma@ip-172-31-24-67:~$ sudo ./aws/install  
Found preexisting AWS CLI installation: /usr/local/aws-cli/v2/current. Please rerun ins  
ll script with --update flag.  
rakshithpk21gma@ip-172-31-24-67:~$
```

We can check the version using command:

```
aws --version
```

```
rakshithpk21gma@ip-172-31-24-67:~$ aws --version  
aws-cli/2.4.6 Python/3.8.8 Linux/5.11.0-1027-aws exe/x86_64.ubuntu.20 prompt/off  
rakshithpk21gma@ip-172-31-24-67:~$
```

We will upgrade this also using upgrade command:

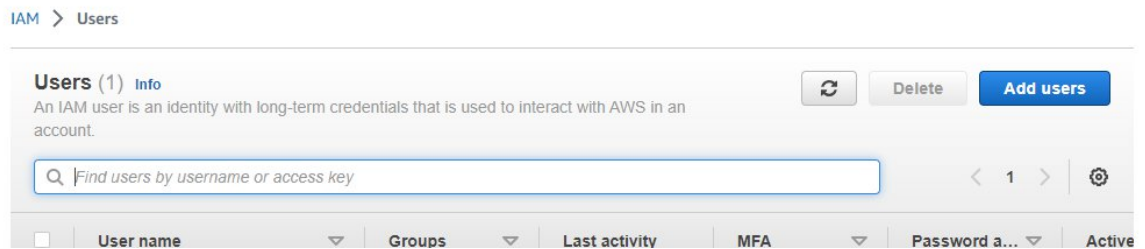
```
sudo ./aws/install --bin-dir /usr/local/bin --install-dir /usr/local/aws-cli --update
```

```
on Linux_ubuntu  
rakshithpk21gma@ip-172-31-24-67:~$ sudo ./aws/install --bin-dir /usr/local/bin --install-  
dir /usr/local/aws-cli --update  
You can now run: /usr/local/bin/aws --version  
rakshithpk21gma@ip-172-31-24-67:~$ aws --version  
aws-cli/2.13.0 Python/3.11.4 Linux/5.11.0-1027-aws exe/x86_64.ubuntu.20 prompt/off  
rakshithpk21gma@ip-172-31-24-67:~$
```

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:

Specify user details

User details

User name

The user name can have up to 64 characters. Valid characters: A-Z, a-z, 0-9, and + = , . @ _ - (hyphen)

☐ Provide user access to the AWS Management Console - *optional*
If you're providing console access to a person, it's a [best practice](#) to manage their access in IAM Identity Center.

ⓘ If you are creating programmatic access through access keys or service-specific credentials for AWS CodeCommit or Amazon Keyspaces, you can generate them after you create this IAM user. [Learn more](#)

Cancel Next

In permissions we will give administrator access:

Permissions options

☐ Add user to group
Add user to an existing group, or create a new group. We recommend using groups to manage user permissions by job function.

☐ Copy permissions
Copy all group memberships, attached managed policies, and inline policies from an existing user.

☒ Attach policies directly
Attach a managed policy directly to a user. As a best practice, we recommend attaching policies to a group instead. Then, add the user to the appropriate group.

Permissions policies (1/1109)

Choose one or more policies to attach to your new user.

Filter by Type: All types

	Policy name	Type	Attached entities
<input type="checkbox"/>	AccessAnalyzerServiceRole...	AWS managed	0
<input checked="" type="checkbox"/>	AdministratorAccess	AWS managed - job function	3
<input type="checkbox"/>	AdministratorAccess-Amplify	AWS managed	0

Then you create the user.

Once you create go to the users security credentials:

Rakshithpk [Info](#)[Delete](#)

Summary

ARN arn:aws:iam::982368000071:user/Rakshithpk	Console access Disabled	Access key 1 Not enabled
Created July 08, 2023, 12:11 (UTC+05:30)	Last console sign-in -	Access key 2 Not enabled

[Permissions](#) | [Groups](#) | [Tags](#) | [Security credentials](#) | [Access Advisor](#)

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

Access keys (0)

Use access keys to send programmatic calls to AWS from the AWS CLI, AWS Tools for PowerShell, AWS SDKs, or direct AWS API calls. You can have a maximum of two access keys (active or inactive) at a time. [Learn more](#)

[Create access key](#)

No access keys

As a best practice, avoid using long-term credentials like access keys. Instead, use tools which provide short term credentials. [Learn more](#)

[Create access key](#)

Chose your use case:

Access key best practices & alternatives [Info](#)

Avoid using long-term credentials like access keys to improve your security. Consider the following use cases and alternatives.

Use case

☒ Command Line Interface (CLI)

You plan to use this access key to enable the AWS CLI to access your AWS account.

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

Retrieve access keys [Info](#)

Access key

If you lose or forget your secret access key, you cannot retrieve it. Instead, create a new access key and make the old key inactive.

Access key

Secret access key

AKIA6JONSDBDVYS5KMDU

 ***** [Show](#)

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 configure`

```
File Edit View Search Terminal Help
rakshithpk21gma@ip-172-31-24-67:~$ aws configure
AWS Access Key ID [None]: AKIA6J0NSDBDVYS5KMDU
AWS Secret Access Key [None]: pX3Jl2sYnmV9YfIMT43fhSpoXMR7aYY1afbIMEP0
Default region name [None]:
Default output format [None]:
rakshithpk21gma@ip-172-31-24-67:~$
```

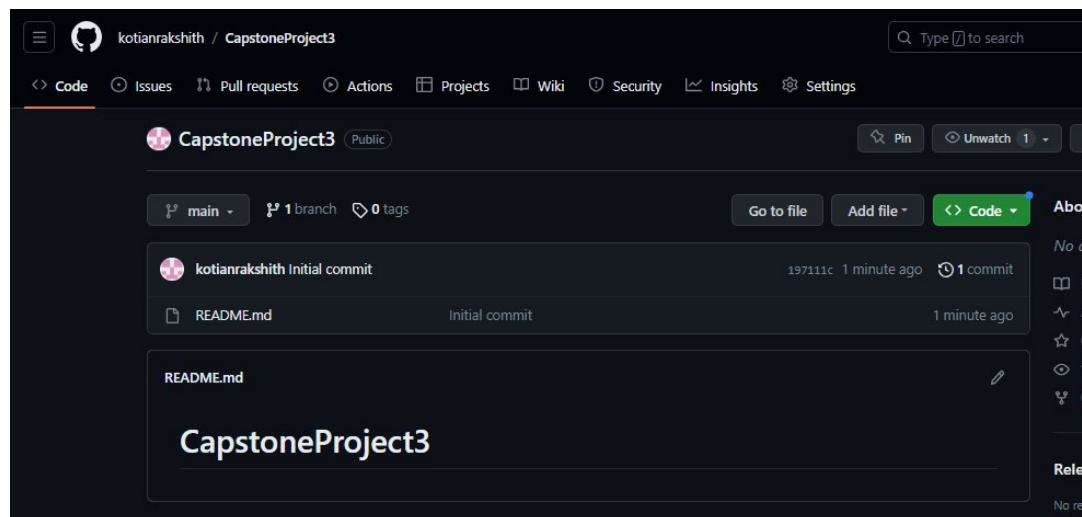
We can see that it is configured

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]: AKIA6J0NSDBDYWPLHJFP
AWS Secret Access Key [None]: vh5H5mYQ4f5bk3TpU2PPL6kPNWeBPxWatZLz5hVS
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 github 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:

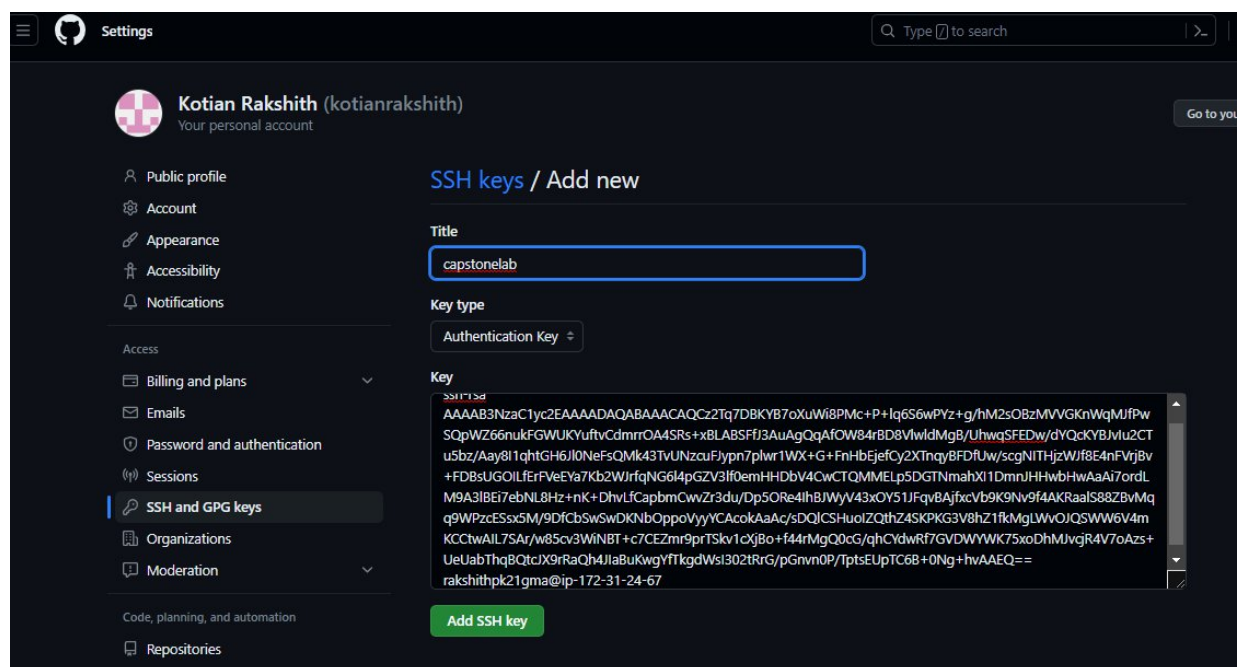
```
rakshithpk21gma@ip-172-31-24-67:~$ mkdir capstone
rakshithpk21gma@ip-172-31-24-67:~$ cd capstone
rakshithpk21gma@ip-172-31-24-67:~/capstone$ git clone git@github.com:kotianrakshith/
```


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 0
| ..=+* @ +
| . .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 AAAAB3NzaC1yc2EAAAADAQABAAQCAQCz2Tq7DBKYB7oXuWi8PMc+P+lq6S6wPYz+g/hM2s0BzMVVGKnWqMJfPwSQpWZ66nukFGWUKYuftvCdmrrOA4SRs+xBLABSFfJ3AuAgQqAfOW84rBD8VlwlMgB/UhwqSFEDw/dYQcKYB
JvIu2CTu5bz/Aay8I1qhtGH6Jl0NeFsQMk43TvUNZcuFJypn7plwr1WX+G+FnhBjefCy2XTnqyBFDfUw/scgNITH
jzWJf8E4nFVrjBv+FDBsUGOILfErFVeEYa7Kb2WJrfqNG6l4pGZV3lf0emHHDV4CwCTQMMElp5DGTnmahXI1DmnJ
HHwbHwAaAi7ordLM9A3lBEi7ebNL8Hz+nK+DhvLfCapbmCwvZr3du/Dp5ORe4IhBJWYV43xOY51JFqvBAjfxcvb9K
9Nv9f4AKRaalS88ZBvMqg9WPzcESsx5M/9DfCbSwSwDKNbOppoVyyYCAcokAaAc/sDQlCSHuoIZQthZ4SKPKG3V8h
Z1fkMgLLwV0JQSWW6V4mKCCtWAIL7SAr/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:



The screenshot shows the GitHub Settings page for the user 'Kotian Rakshith (kotianrakshith)'. The left sidebar contains navigation links: Public profile, Account, Appearance, Accessibility, Notifications, Access, Billing and plans, Emails, Password and authentication, Sessions, SSH and GPG keys (selected), Organizations, Moderation, Code, planning, and automation, and Repositories. The main content area is titled 'SSH keys / Add new'. It includes a 'Title' field with the value 'capstonelab', a 'Key type' dropdown set to 'Authentication Key', and a 'Key' text area containing the public key: 'ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQCAQCz2Tq7DBKYB7oXuWi8PMc+P+lq6S6wPYz+g/hM2s0BzMVVGKnWqMJfPwSQpWZ66nukFGWUKYuftvCdmrrOA4SRs+xBLABSFfJ3AuAgQqAfOW84rBD8VlwlMgB/UhwqSFEDw/dYQcKYBjvIu2CTu5bz/Aay8I1qhtGH6Jl0NeFsQMk43TvUNZcuFJypn7plwr1WX+G+FnhBjefCy2XTnqyBFDfUw/scgNITHjzWJf8E4nFVrjBv+FDBsUGOILfErFVeEYa7Kb2WJrfqNG6l4pGZV3lf0emHHDV4CwCTQMMElp5DGTnmahXI1DmnJHHwbHwAaAi7ordLM9A3lBEi7ebNL8Hz+nK+DhvLfCapbmCwvZr3du/Dp5ORe4IhBJWYV43xOY51JFqvBAjfxcvb9K9Nv9f4AKRaalS88ZBvMqg9WPzcESsx5M/9DfCbSwSwDKNbOppoVyyYCAcokAaAc/sDQlCSHuoIZQthZ4SKPKG3V8hZ1fkMgLLwV0JQSWW6V4mKCCtWAIL7SAr/w85cv3WiNBT+c7CEZmr9prTSkv1cXjBo+f44rMgQ0cG/qhCYdwRf7GVDWYWK75xoDhMJvcjR4V7oAzs+UeUabThqBQtCjX9rRaQh4JIaBuKwgYfTkgdWsI302tRrG/pGnvn0P/TptsEUPTC6B+0Ng+hvAAEQ== rakshithpk21gma@ip-172-31-24-67'. At the bottom of the key area is an 'Add SSH key' button.

Once added we can clone:

```
rakshithpk21gma@ip-172-31-24-67:~/capstone$ git clone git@github.com:kotianrakshith/CapstoneProject3.git
Cloning into 'CapstoneProject3'...
Warning: Permanently added the ECDSA host key for IP address '140.82.112.3' to the list of 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

```
CapstoneProject3
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"

  }

}
```

```

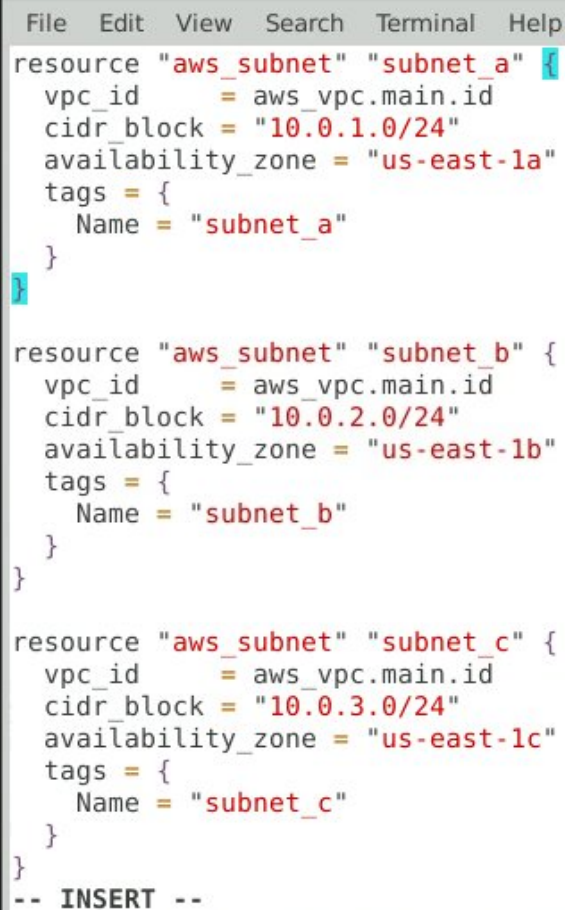
resource "aws_subnet" "subnet_b" {
  vpc_id    = aws_vpc.main.id
  cidr_block = "10.0.2.0/24"
  availability_zone = "us-east-1b"

  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"
  }
}

```



```

File Edit View Search Terminal Help
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"
  }
}
-- INSERT --

```

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 internet gateway terraform file: internetgateway.tf

```
resource "aws_internet_gateway" "main-igw" {

  vpc_id = aws_vpc.main.id


  tags = {

    Name = "capstone-igw"

  }

}
```

```
File Edit View Search Terminal Help
resource "aws_internet_gateway" "main-igw" {
  vpc_id = aws_vpc.main.id
  tags = {
    Name = "capstone-igw"
  }
}
```

Now let us create the route table: route table.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 security groups before we get 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"  
  }  
}
```



```

File Edit View Search Terminal Help
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"
  }
}
-- INSERT --

```

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
```

```

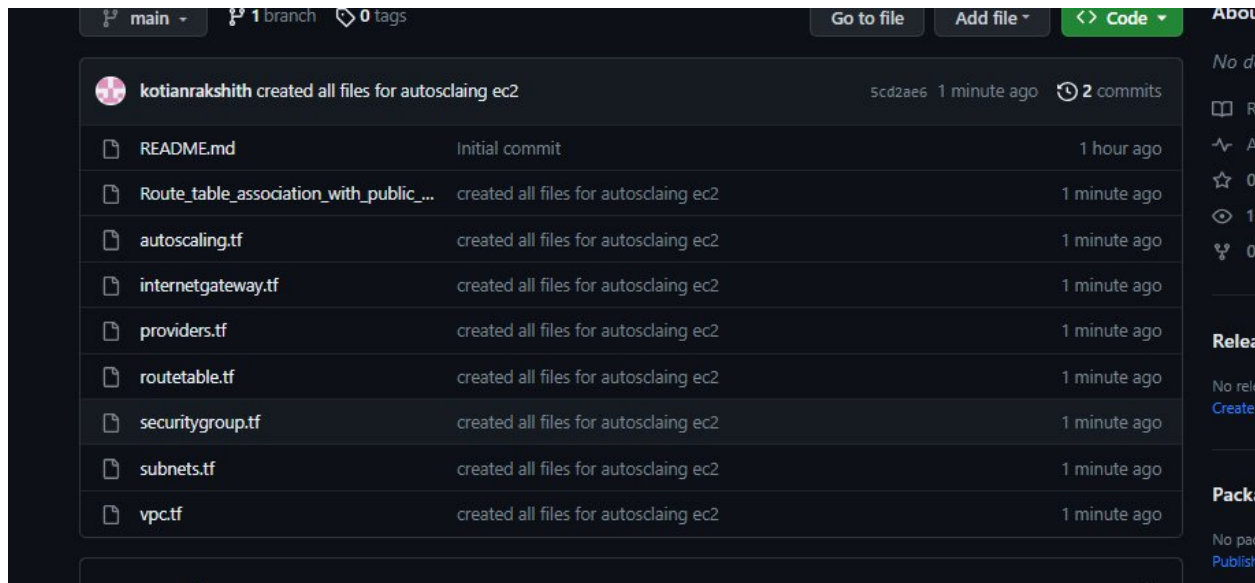
rakshithpk21gma@ip-172-31-24-67:~/capstone/CapstoneProject3$ git commit -m "created all files for autosclaing ec2"
[main 5cd2ae6] created all files for autosclaing ec2
 8 files changed, 120 insertions(+)
 create mode 100644 Route_table_association_with_public_subnets.tf
 create mode 100644 autoscaling.tf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

    ]

}

```

The screenshot shows a terminal window with a dark theme. The title bar indicates the user is 'rakshithpk21gma@ip-172-31-24-67' in the directory '~/capstone/CapstoneProject3'. The terminal displays Terraform configuration code for an AWS EKS cluster and its associated node group. The code is color-coded: resource names are in red, variable names in blue, and string literals in green. The configuration includes dependencies on IAM policies for the EKS cluster and node group, and specifies subnet IDs for the VPC configuration.

```

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

rakshithpk21gma@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
rakshithpk21gma@ip-172-31-24-67:~/capstone/CapstoneProject3$ git push origin main
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.

```

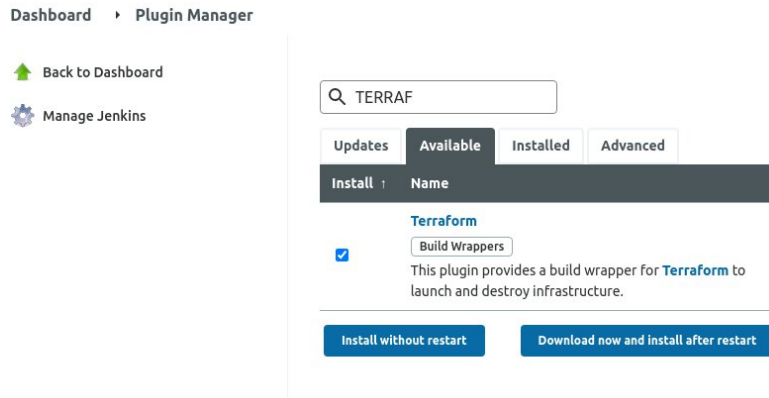
Now we have all the files in the github:

	kotianrakshith added and modified required files	805f61a 7 minutes ago	🕒 6 commits
	README.md	Initial commit	yesterday
	Route_table_association_with_public...	created all files for autosclaing ec2	20 hours ago
	autoscaling.tf	added autoscaling policy	13 hours ago
	eks.tf	added and modified required files	7 minutes ago
	internetgateway.tf	created all files for autosclaing ec2	20 hours ago
	providers.tf	created all files for autosclaing ec2	20 hours ago
	rolepolicy.tf	added and modified required files	7 minutes ago
	routetable.tf	created all files for autosclaing ec2	20 hours ago
	securitygroup.tf	created all files for autosclaing ec2	20 hours ago
	subnets.tf	added and modified required files	7 minutes ago
	vpc.tf	created all files for autosclaing ec2	20 hours ago

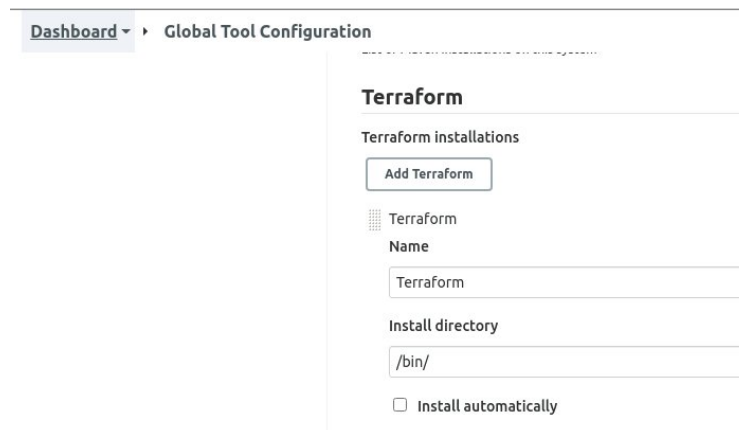
7. Execute terraform scripts:

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

First in the jenkins we will install terraform plugin:

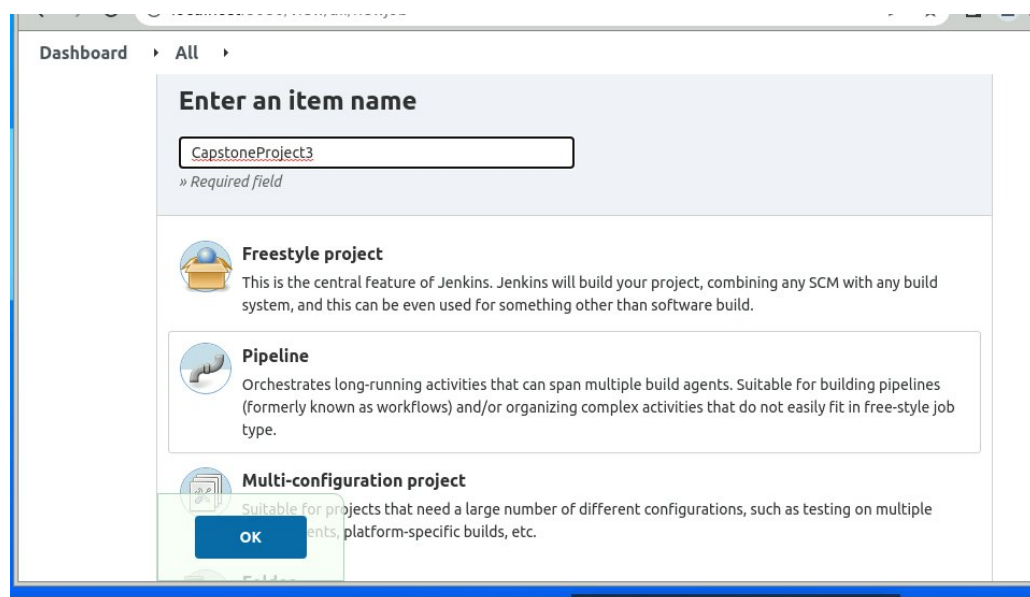


Also in the global tool configuration add terraform details:



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

Create new pipeline project in jenkins



Give proper description and provide git hub project url:

The screenshot shows the 'General' tab of a Jenkins pipeline configuration. The 'Description' field contains the text 'This is capstone project for Hotel side hospital'. Below this, there are several checkboxes: 'Discard old builds', 'Do not allow concurrent builds', 'Do not allow the pipeline to resume if the controller restarts', 'GitHub project' (which is checked), 'Pipeline speed/durability override', 'Preserve stashes from completed builds', and 'This project is parameterized'. The 'Project url' field is filled with 'https://github.com/kotianrakshith/CapstoneProject3'. There are also links for 'Plain text' and 'Preview'.

To get the checkout script we will use pipeline syntax

The screenshot shows the 'Pipeline Syntax' tab of a Jenkins pipeline configuration. The 'Steps' section is visible, showing a 'Sample Step' with the name 'checkout: Check out from version control'. The 'checkout' step is configured with the SCM set to 'Git'. The 'Repositories' section shows the 'Repository URL' as 'https://github.com/kotianrakshith/CapstoneProject3'. The 'Credentials' section shows a dropdown menu with 'none' selected and an 'Add' button.

The screenshot shows the 'Pipeline Syntax' tab of a Jenkins pipeline configuration. The 'Branches' section shows a list of branches with '*/main' selected. There are 'Delete' and 'Add Branch' buttons. The 'Repository browser' is set to '(Auto)'. The 'Additional Behaviours' section has 'Include in polling?' and 'Include in changelog?' checked. A 'Generate Pipeline Script' button is visible. Below the button, the generated script is shown:

```
checkout([$class: 'GitSCM', branches: [[name: '*/main']], extensions: [], userRemoteConfigs: [[url: 'https://github.com/kotianrakshith/CapstoneProject3']]])
```

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

General
Build Triggers
Advanced Project Options
Pipeline

Definition

Pipeline script

Script

```

1 pipeline {
2   agent any
3
4   stages {
5     stage('Checkout') {
6       steps {
7         checkout([$class: 'GitSCM', branches: [[name: '*/main']], extensions: [], userRemoteConfigs: [[url: 'https://github.com/kotia
8       }
9     }
10  }
11 }

```

try sample Pipeline...

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

Pipeline script

Script

```

1 pipeline {
2   agent any
3   tools {
4     terraform 'Terraform'
5   }
6   stages {
7     stage('Checkout') {
8       steps {
9         checkout([$class: 'GitSCM', branches: [[name: '*/main']], extensions: [], userRemoteConfigs: [[url: 'https://github.com/kotia
10      }
11    }
12    stage('Terraform init') {
13      steps {
14        sh 'terraform init'
15      }
16    }
17    stage('Terraform apply') {
18      steps {
19        sh 'terraform apply --auto-approve'
20      }
21    }
22  }
23 }

```

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'
            }
        }
    }
}

```

```

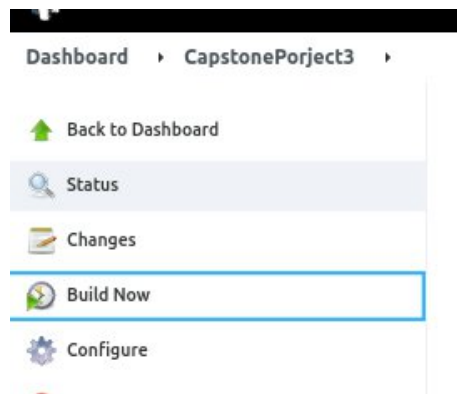
    }
}

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

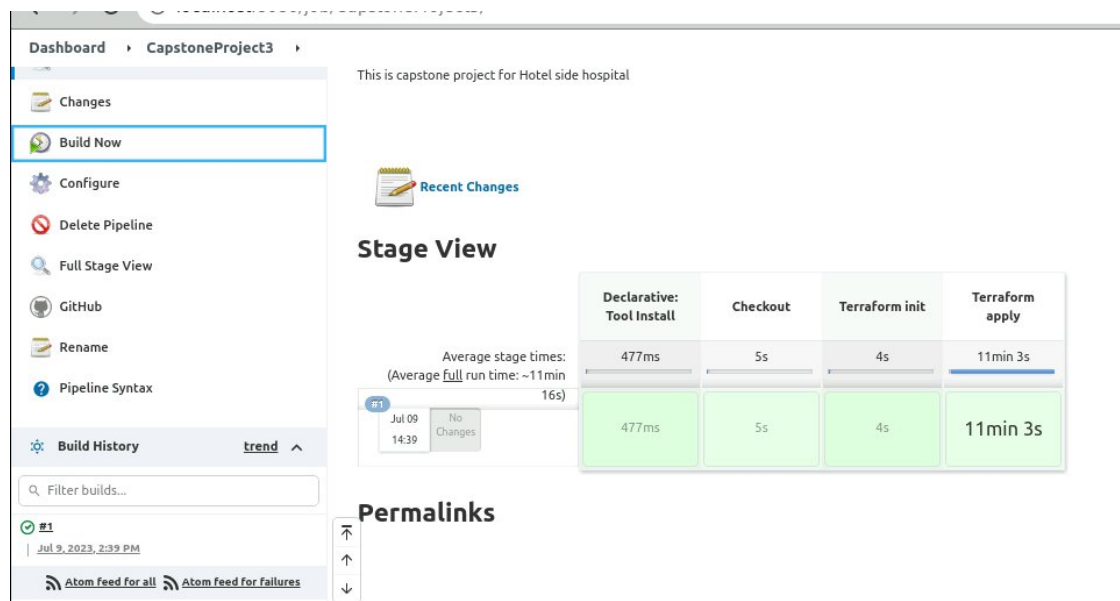
```

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:

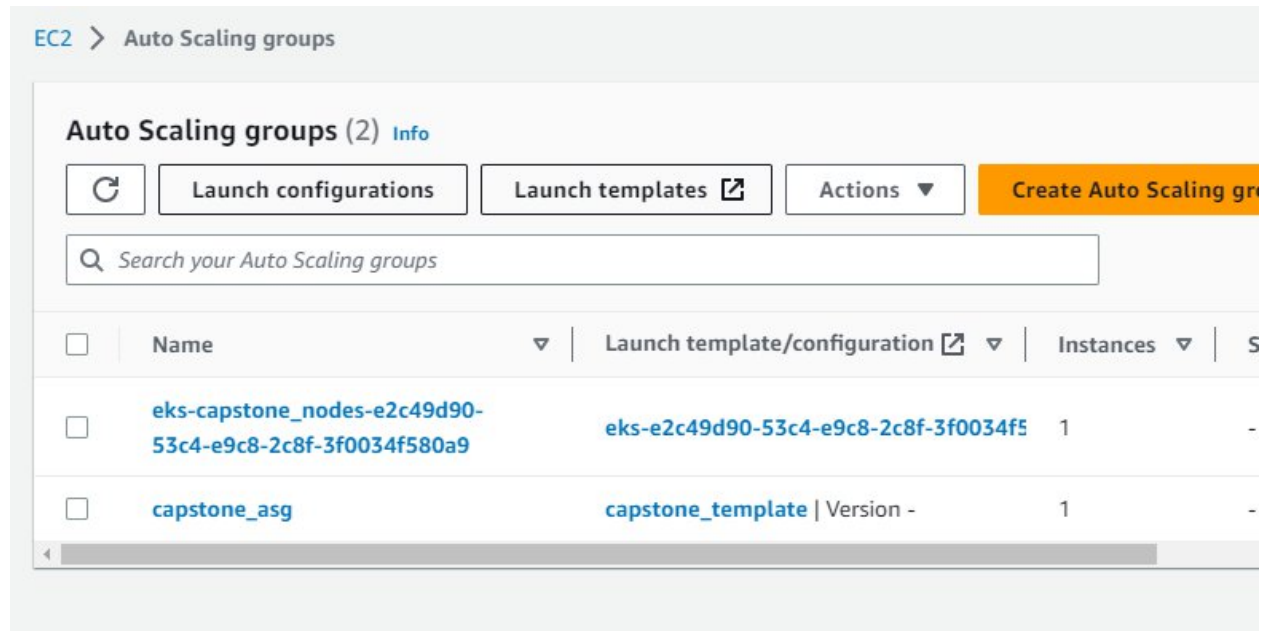


We can see that it has run successfully:

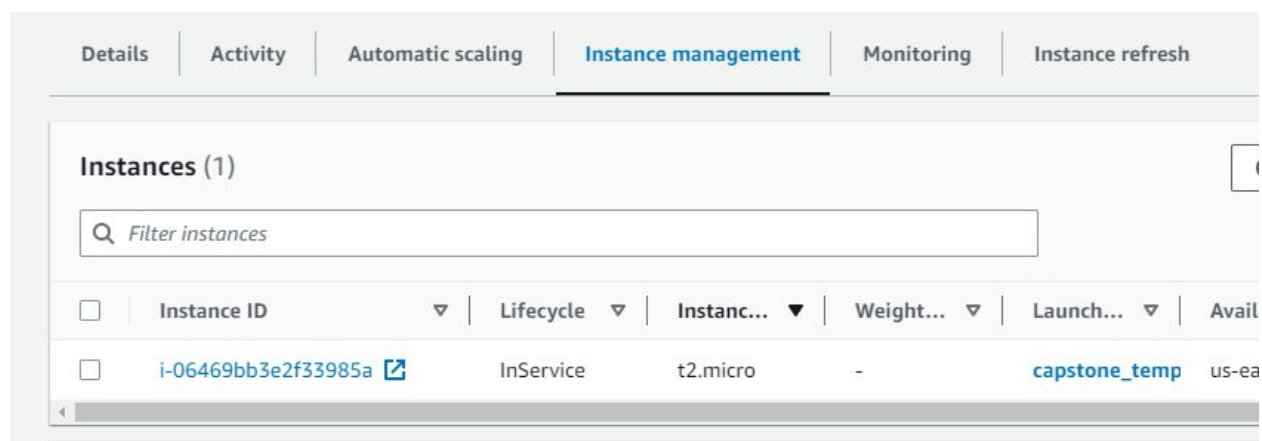


8. Checking the deployment in AWS:

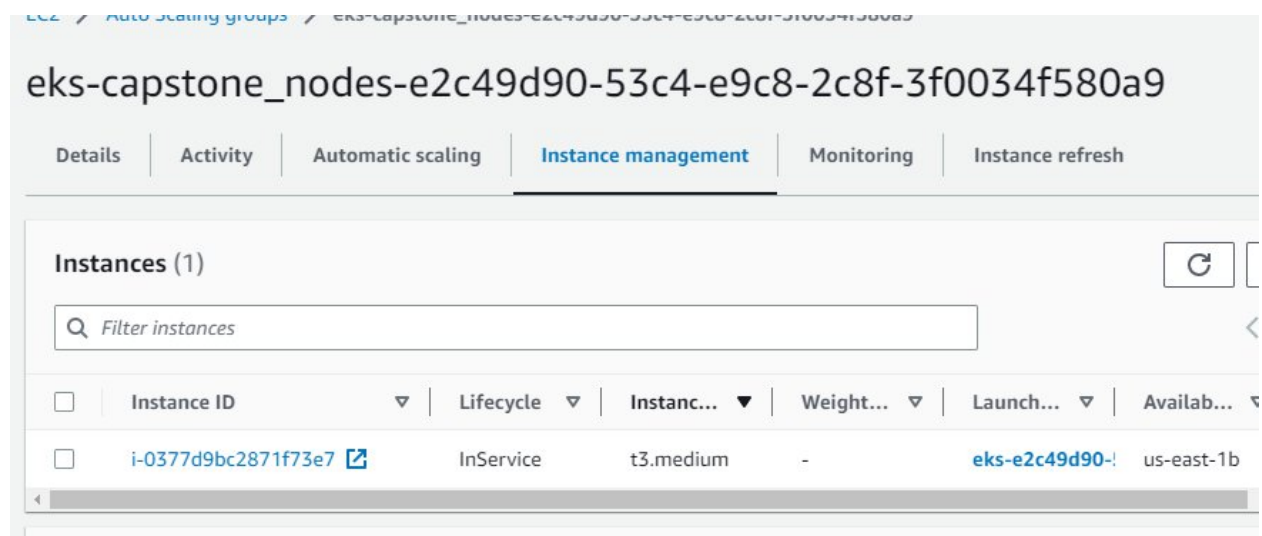
We see in the autoscaling groups that there are two autoscaling groups, one for EKS and one for EC2 as we correctly deployed:



Name	Launch template/configuration	Instances
eks-capstone_nodes-e2c49d90-53c4-e9c8-2c8f-3f0034f580a9	eks-e2c49d90-53c4-e9c8-2c8f-3f0034f5	1
capstone_asg	capstone_template Version -	1



Instance ID	Lifecycle	Instanc...	Weight...	Launch...	Avail
i-06469bb3e2f33985a	InService	t2.micro	-	capstone_temp	us-ea



Instance ID	Lifecycle	Instanc...	Weight...	Launch...	Availab...
i-0377d9bc2871f73e7	InService	t3.medium	-	eks-e2c49d90-!	us-east-1b

Each have one instance.

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

We will connect to one of the instance:

[EC2](#) > [Instances](#) > [i-06469bb3e2f33985a](#) > [Connect to instance](#)

Connect to instance [Info](#)

Connect to your instance `i-06469bb3e2f33985a` using any of these options


EC2 Instance Connect

Session Manager

SSH client

EC2 serial console

Instance ID


 `i-06469bb3e2f33985a`

Connection Type

☒ **Connect using EC2 Instance Connect**
Connect using the EC2 Instance Connect browser-based client, with a public IPv4 address.

☐ **Connect using EC2 Instance Connect Endpoint**
Connect using the EC2 Instance Connect browser-based client, with a private IPv4 address and a VPC endpoint.

Public IP address

 `54.226.110.73`

User name

Enter the user name defined in the AMI used to launch the instance. If you didn't define a custom user name, use the default user name, `ec2-user`.

Here we will install the stress tool:

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

[illegible]

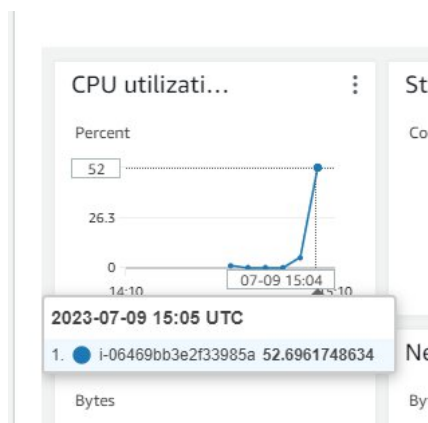
10. Validate if autoscaling is working by putting load on autoscaling group:

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: debug: [26309] using backoff sleep of 24000us
stress: debug: [26309] setting timeout to 3000s
stress: debug: [26309] --> hogcpu worker 8 [26310] forked
stress: debug: [26309] using backoff sleep of 21000us
stress: debug: [26309] setting timeout to 3000s
stress: debug: [26309] --> hogcpu worker 7 [26311] forked
stress: debug: [26309] using backoff sleep of 18000us
stress: debug: [26309] setting timeout to 3000s
stress: debug: [26309] --> hogcpu worker 6 [26312] forked
stress: debug: [26309] using backoff sleep of 15000us
stress: debug: [26309] setting timeout to 3000s
stress: debug: [26309] --> hogcpu worker 5 [26313] forked
stress: debug: [26309] using backoff sleep of 12000us
stress: debug: [26309] setting timeout to 3000s
stress: debug: [26309] --> hogcpu worker 4 [26314] forked
stress: debug: [26309] using backoff sleep of 9000us
stress: debug: [26309] setting timeout to 3000s
stress: debug: [26309] --> hogcpu worker 3 [26315] forked
stress: debug: [26309] using backoff sleep of 6000us
```

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



Cpu utilization is more than our limit.

Now if we check the autoscaling group:

EC2 > Auto Scaling groups > capstone_asg

capstone_asg

Details | Activity | Automatic scaling | **Instance management** | Monitoring | Instance refresh

Instances (3)

Filter instances

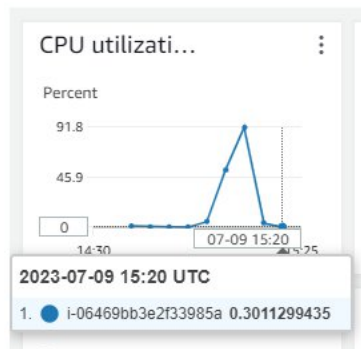
	Instance ID	Lifecycle	Instanc...	Weight...	Launch...	Availab...	Health ...
<input type="checkbox"/>	i-034766f8203265195	InService	t2.micro	-	capstone_temp	us-east-1a	Healthy
<input type="checkbox"/>	i-04e6289b027650451	InService	t2.micro	-	capstone_temp	us-east-1b	Healthy
<input type="checkbox"/>	i-06469bb3e2f33985a	InService	t2.micro	-	capstone_temp	us-east-1c	Healthy

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

We can see that cpu utilization falls eventually to zero



Now the instance has decreased to two:

The screenshot shows the AWS Management Console 'Instance management' tab. It displays a table with two instances. The first instance, i-034766f8203265195, is in the 'Terminating' lifecycle state. The second instance, i-04e6289b027650451, is in the 'InService' lifecycle state. Both instances are t2.micro type, launched by 'capstone_temp' in the 'us-east-1a' availability zone, and are 'Healthy'.

Instance ID	Lifecycle	Instanc...	Weight...	Launch...	Availab...	Health ...
i-034766f8203265195	Terminating	t2.micro	-	capstone_temp	us-east-1a	Healthy
i-04e6289b027650451	InService	t2.micro	-	capstone_temp	us-east-1b	Healthy

Eventually we will have only one:

The screenshot shows the AWS Management Console 'Instance management' tab. It displays a table with one instance. The instance, i-04e6289b027650451, is in the 'InService' lifecycle state. It is a t2.micro type, launched by 'capstone_temp' in the 'us-east-1b' availability zone, and is 'Healthy'.

Instance ID	Lifecycle	Instanc...	Weight...	Launch...	Availab...	Health ...
i-04e6289b027650451	InService	t2.micro	-	capstone_temp	us-east-1b	Healthy

So we have confirmed 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.