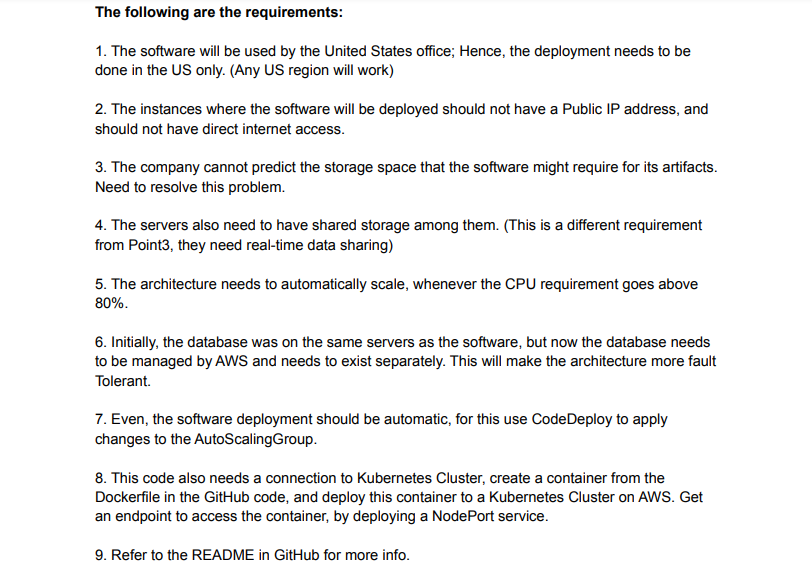
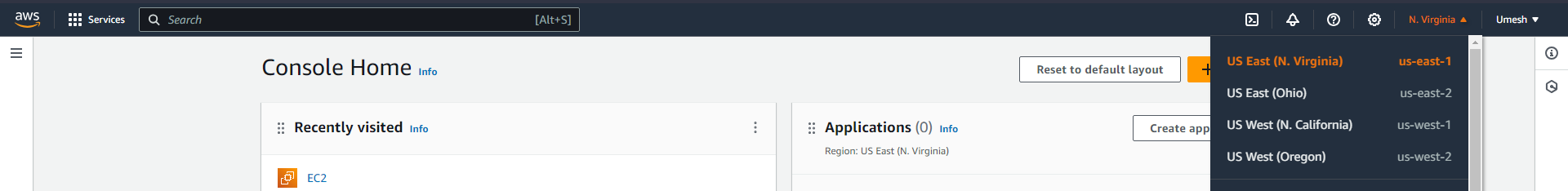
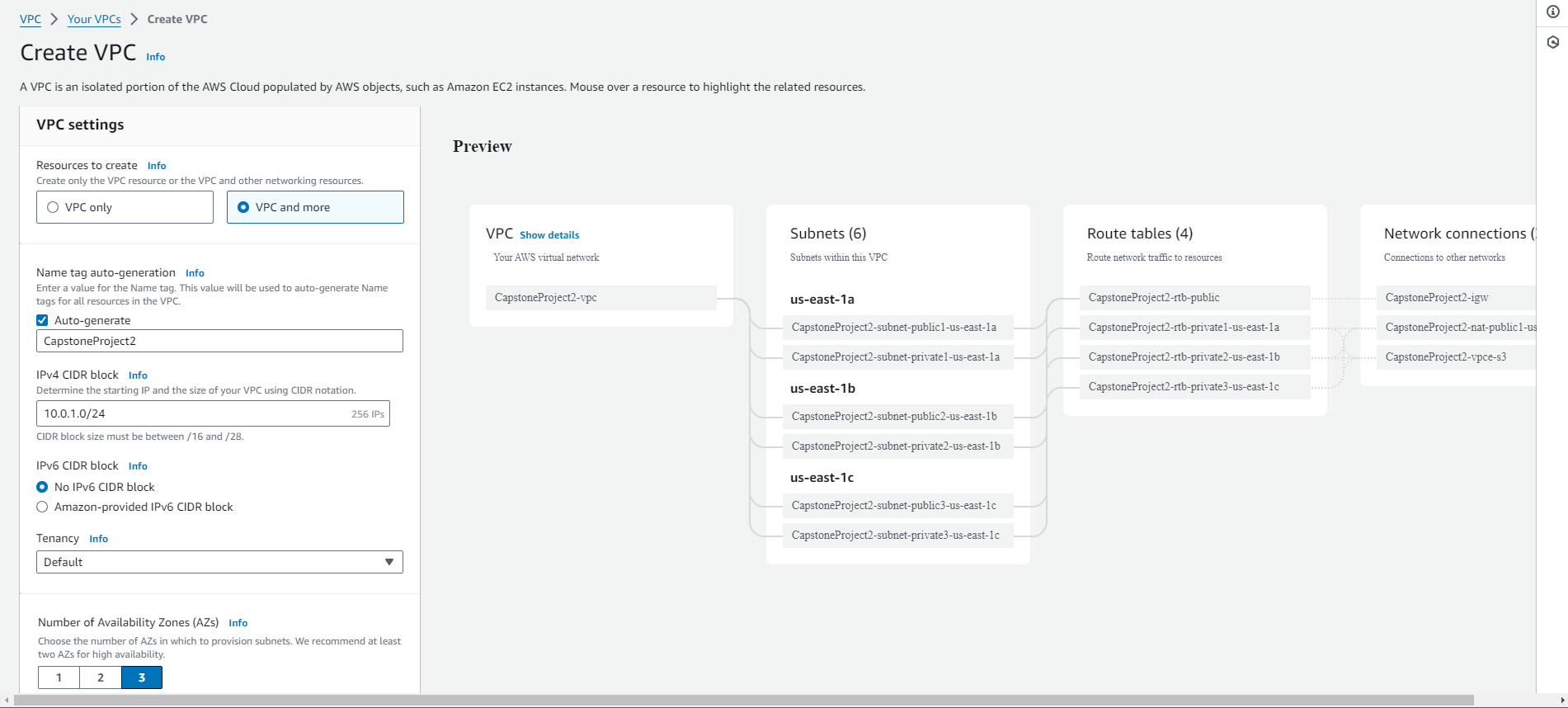
**Capstone Project – 2**

****

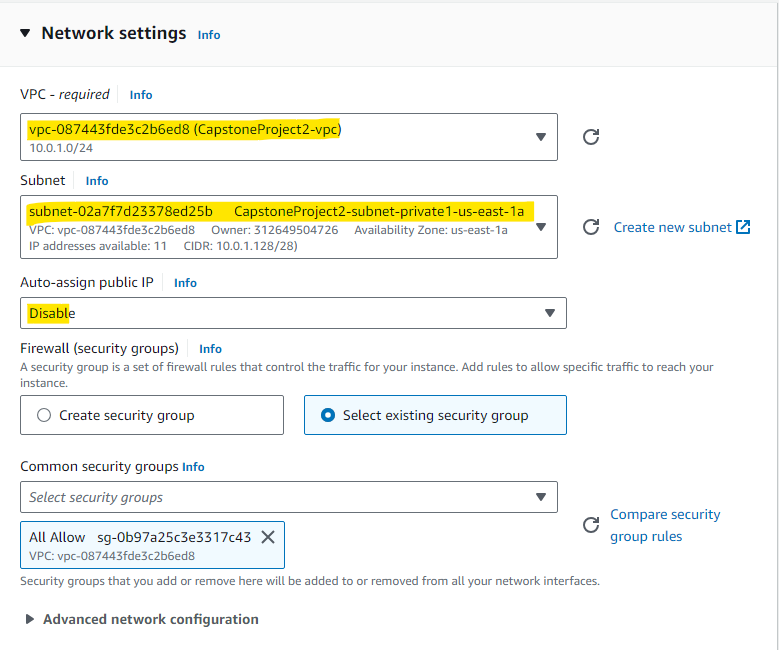
1. Deployment in the US Region : Choose the region within the US Region to proceed with this ask. I’ll be choosing North Virginia (us-east-1)



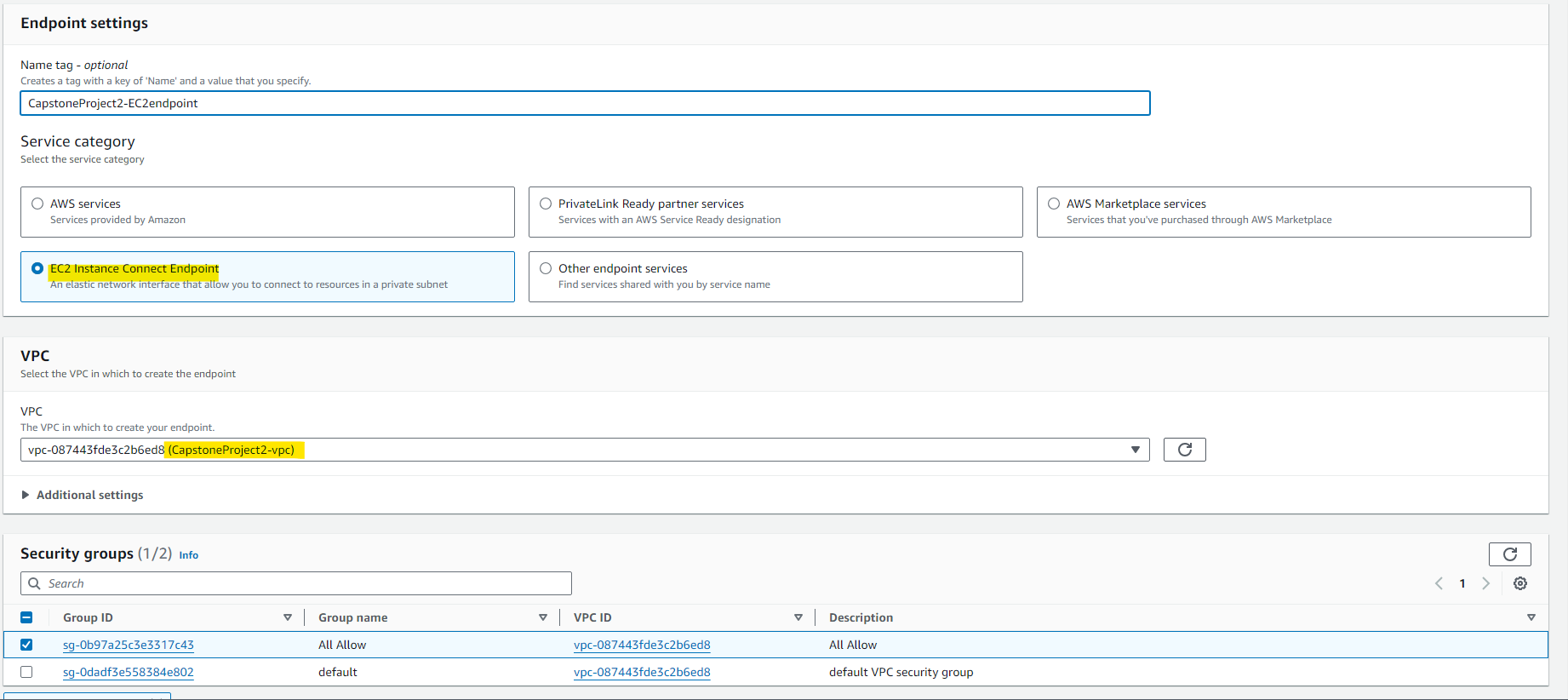
1. For the instances to not have a public IP, create a VPC and place the instance within the Private Subnet and route it via a NAT Gateway. Create it via VPC and more (auto generate setting)

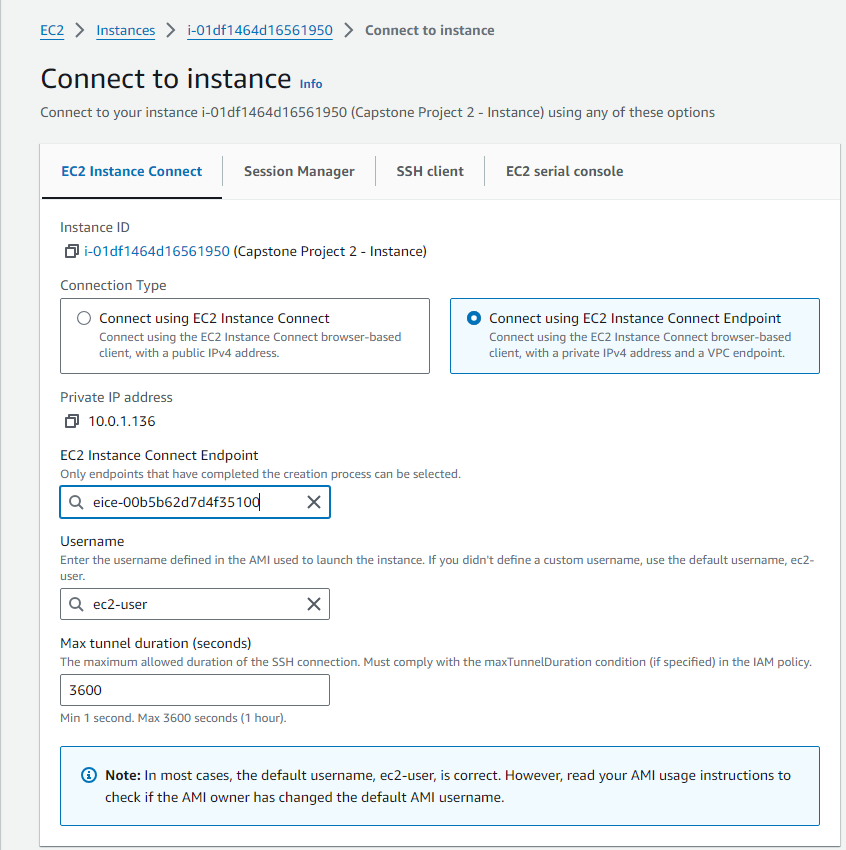


1. Once the VPC is created, go ahead and launch the instance in a private subnet and check for internet connectivity. Select the created VPC while launching the instance and also select the private subnet.

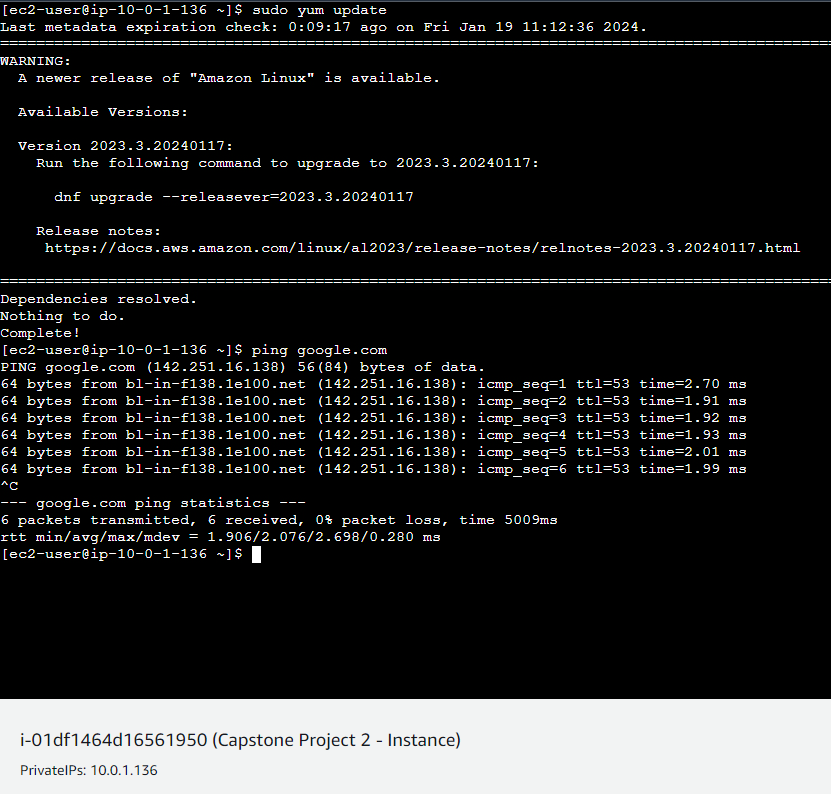


1. Connect to the Instance by creating a EC2 Instance Endpoint.

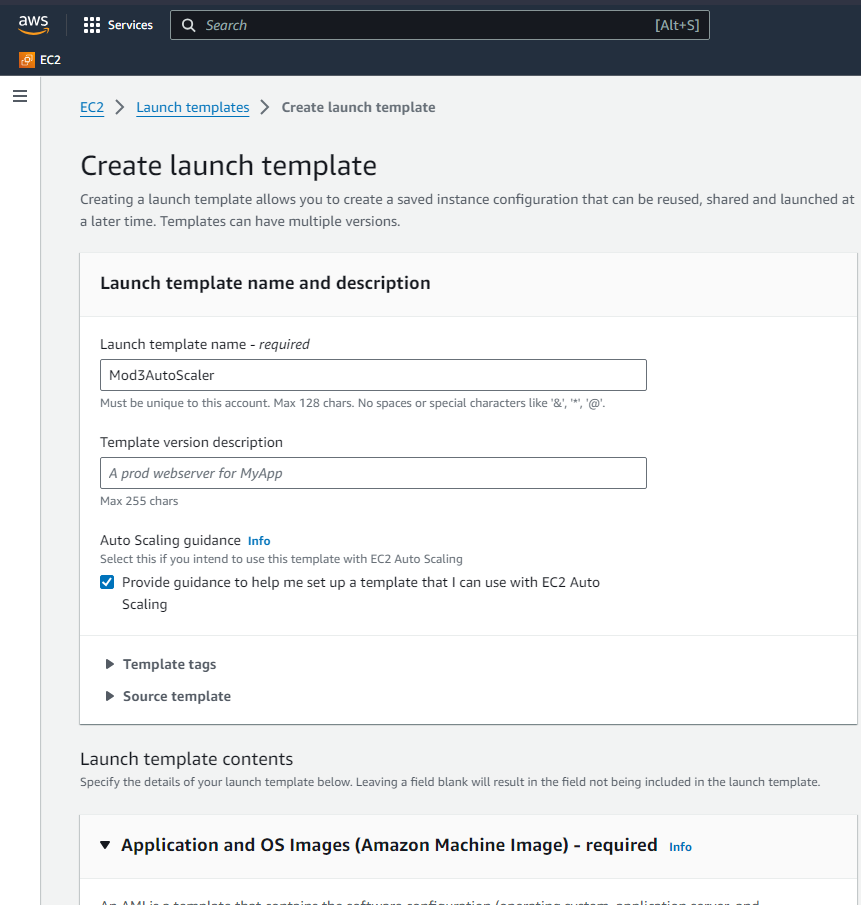


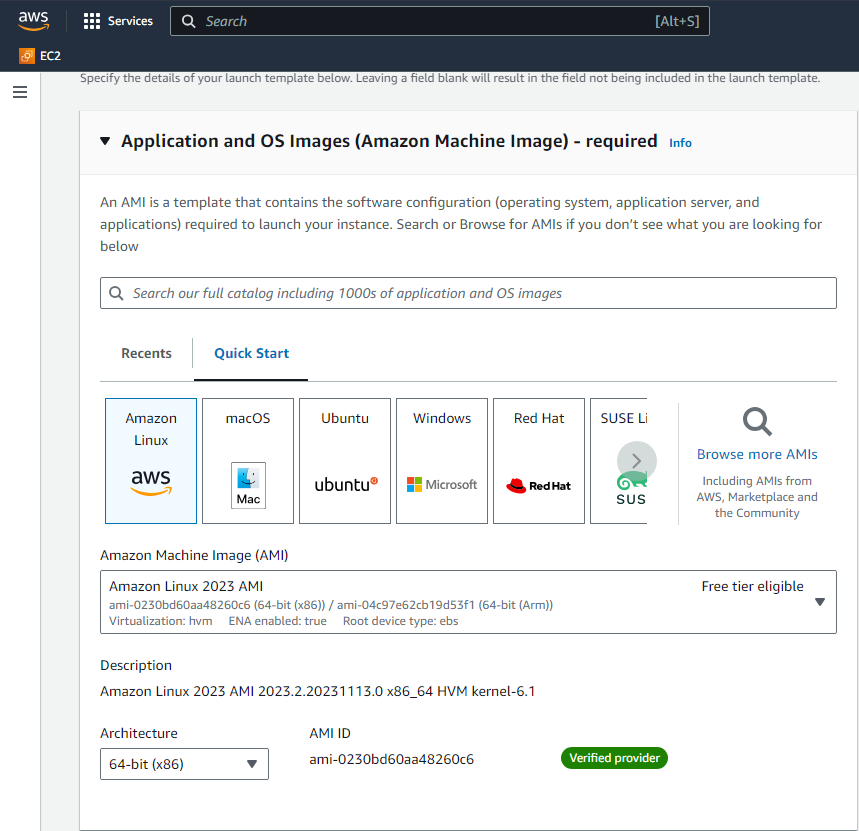


1. To check the internet connectivity, try to ping google. If it resolves and pings, that means it’s connected to internet and it’s not via Public IP.

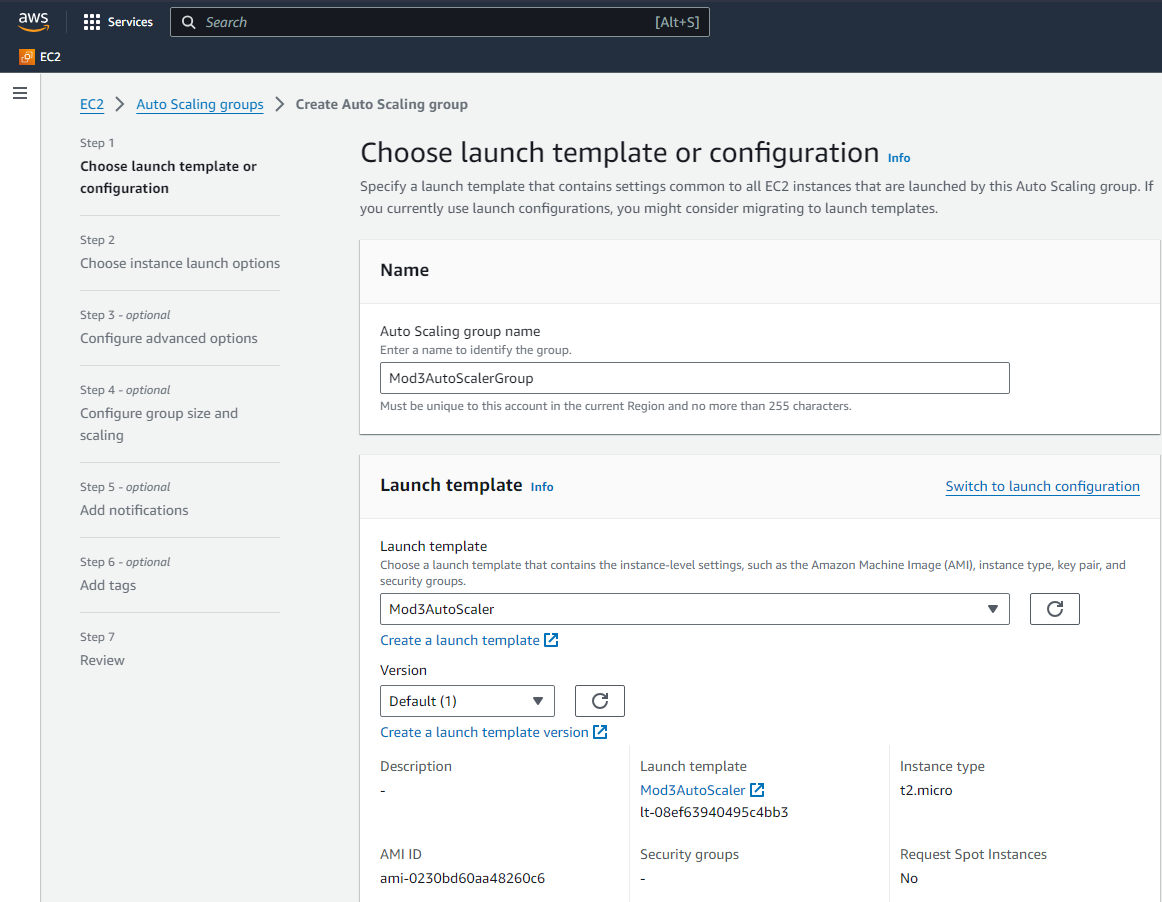


1. To address the issue of dynamic storage space, we can create EFS. It will also resolve the issue of point 4 i.e real time data sharing. I won’t be creating since it would cost $ to create EFS Storage.
2. We can create a auto scaling group which will create an instance when the cpu usage goes above 80%.
   1. Create a Launch Template.

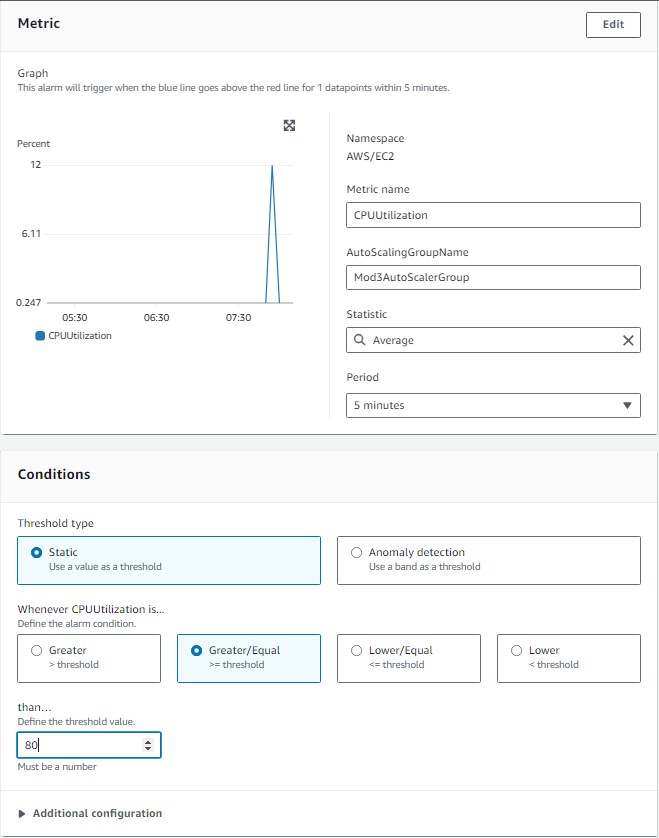


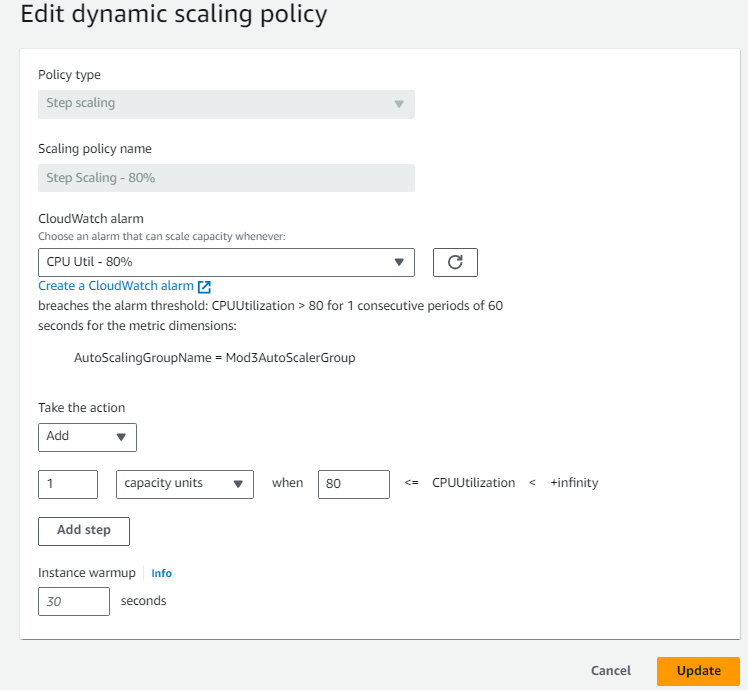


* 1. Create a Auto Scaling Group. Choose Default Options.

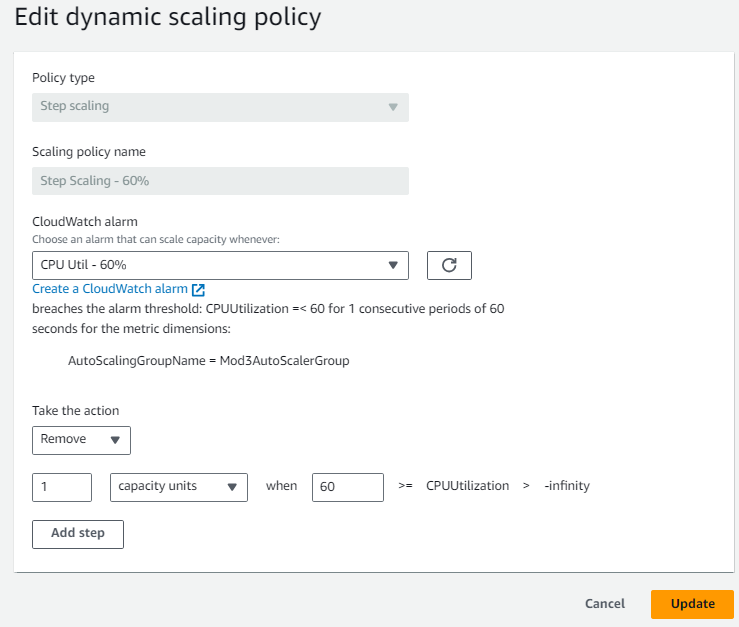


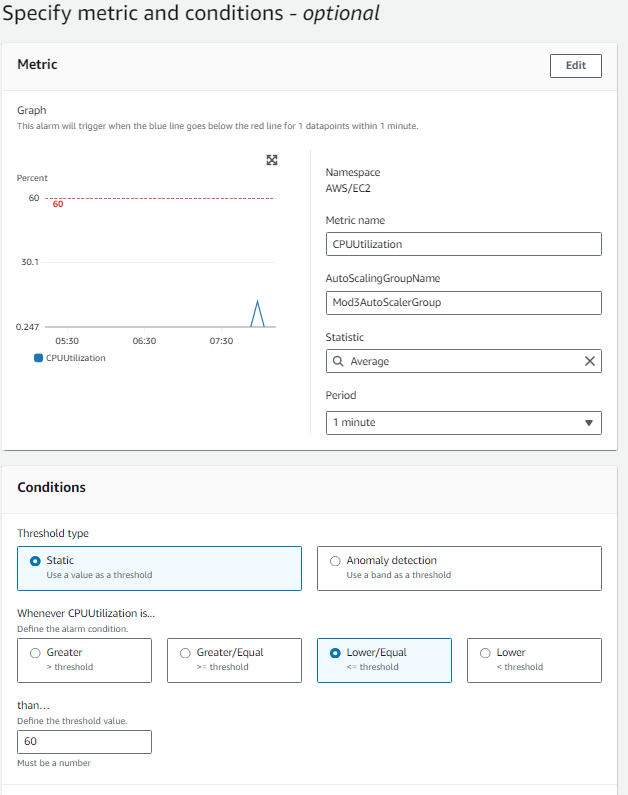
* 1. Configure the Auto Scaling Group to Scale Out when CPU Util hits 80%. Create an Alarm first, and then configure the Auto Scaling Group to Scale Out.

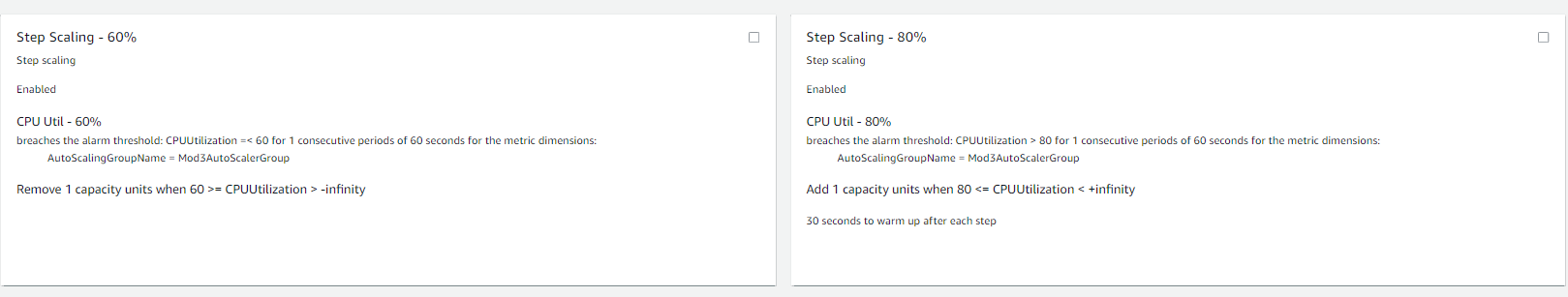




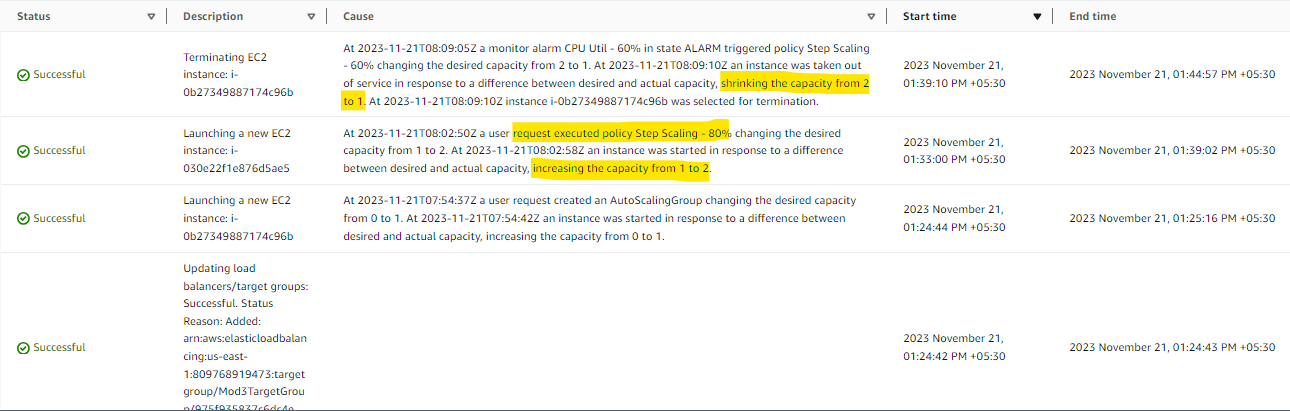
* 1. Create a Rule for Scale In when CPU Util is below 60%







* 1. Executed Scale Out Rule for Demo, Scale In follows as per Alarm since the CPU Util was under 60%



1. For the database to be managed by AWS, we can migrate the existing Database by using AWS Database Migration Service to Amazon RDS for a managed database service. This will make the architecture more fault-tolerant and scalable.
2. For the software deployment to be automation, we can use CodeDeploy.
   1. CodeDeploy Application:
      1. Create AWS CodeDeploy application and deployment group.
      2. Define deployment group that targets the Auto Scaling Group created in the earlier step.
   2. IAM Roles:
      1. Ensure that the EC2 instances in your Auto Scaling Group have an IAM role attached with the necessary permissions for CodeDeploy. The role should have permissions like codedeploy:List\*, codedeploy:Register\*, codedeploy:Get\*, and s3:Get\*.
3. We can further integrate the CodeDeploy via Jenkins by the below pipeline script.

pipeline {

agent any

stages {

stage('Checkout') {

steps {

git 'https://github.com/hshar94/aws-devops-pgp.git'

}

}

stage('Build') {

steps {

dir(capstoneproject2') {

sh './mvnw clean package' }

}

stage('Deploy with CodeDeploy') {

steps {

script {

withAWS(region: 'us-east-1', credentials: ‘aws-credentials-id') {

def applicationName = ‘capstoneproject2-codedeploymentapp’

def deploymentGroupName = capstoneproject2-codedeployment-deploymentgroup’

def deploymentId = codedeployCreateDeployment(applicationName: applicationName, deploymentGroupName: deploymentGroupName, revisionLocation: [

s3Location: [

bucket: 'capstoneproject2s3bucket’,

bundleType: 'zip',

eTag: ' eTags3’,

key: ‘app-artifact.zip',

version: ‘s3-object-version'

]

])

waitForDeploymentStatus id: deploymentId, desiredStatus: 'Succeeded', timeout: '20 min'

}

}

}

}

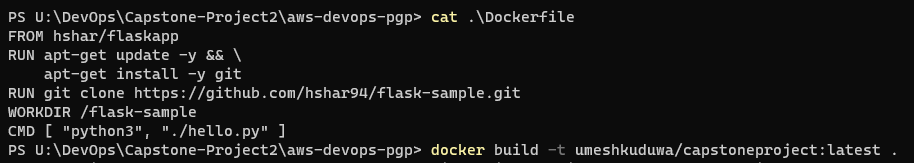
}

}

1. To create the container we first need to containerize it.
   1. Clone the Project and move inside the directory in the AWS EKS Instance.



* 1. Build and Push it in DockerHub.



* 1. Proceed to build the Kubernetes deployment by referencing this container.
     1. deployment.yaml

apiVersion: apps/v1

kind: Deployment

metadata:

name: capstoneproject-deployment

spec:

replicas: 3

selector:

matchLabels:

app: capstoneproject

template:

metadata:

labels:

app: capstoneproject

spec:

containers:

- name: capstoneproject-container

image: umeshkuduwa/capstoneproject:latest

ports:

- containerPort: 80

1. Apply the deployment to the Cluster : kubectl apply -f deployment.yaml
2. Expose it via service.yaml

apiVersion: v1

kind: Service

metadata:

name: capstoneproject-service

spec:

type: NodePort

selector:

app: capstoneproject

ports:

- protocol: TCP

port: 80

targetPort: 80

1. Apply the Service : kubectl apply -f service.yaml
2. Access the deployed application : kubectl get svc capstoneproject-service
3. Get the endpoint to access the container and specify all the details in the config.py file

# config.py

# AWS Configurations

aws\_region = 'us-east-1'

efs\_mount\_point = '/mnt/efs'

# Kubernetes Configurations

kubernetes\_endpoint = 'https:// 203.0.113.1’

kubernetes\_nodeport\_ip = ' 203.0.113.1’

kubernetes\_nodeport\_port = ‘80’