**Course Project**

Agenda for this project is to create an architecture in AWS so as to dockerize and deploy node application in a EC2 instance.

Tools used are as below:

CI/CD – Jenkins

Container – Docker

IaC – Terraform

Provisioning – Ansible

Version control – GitHub

Cloud platform – AWS

**Task 1**

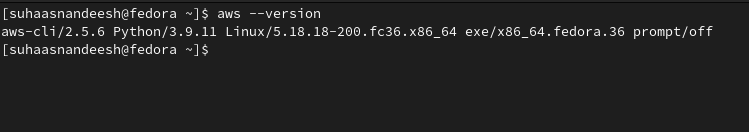
**Subtask 1**

Ensure AWS CLI is installed and configured with full access in your Linux machine.

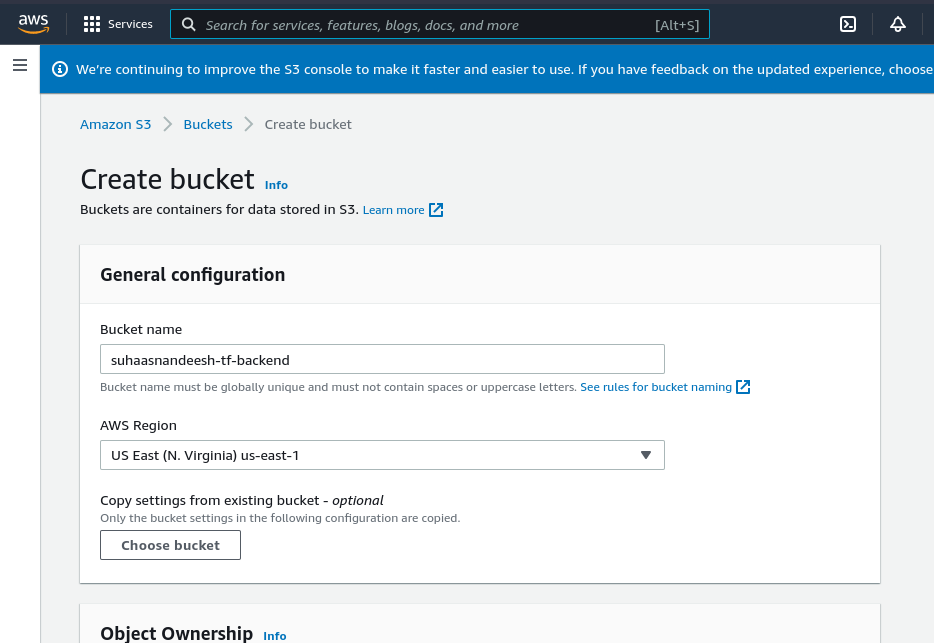
Please refer below step in order to install aws cli -

curl "https://awscli.amazonaws.com/awscli-exe-linux-x86\_64.zip" -o "awscliv2.zip"

unzip awscliv2.zip

sudo ./aws/install

S3 backend is initialised for the backend state store using Terraform. This bucket is used later in the project to use the state files of Terraform.



**Subtask 2**

In this subtask few AWS components are created using Terraform namely,

AWS VPC in us-east-1 region,

1 Internet gateway

1 NAT-Gateway in availability zone-a,

2 private and public subnets in availability zone a and b

Public route table used for IGW and public subnets and another route table for routing traffic from private subnet to public route table.

CIDR block used for VPC is /16 and the subnets created within this VPC is with CIDR block /24

Please refer to the Terraform scripts under the Task-1 folder in the code base.

**Subtask 3**

In this subtask we are creating 3 EC2 instances. Application and Jenkins instances will be created in private subnets so as to have secure traffic (no public access) and another instance (bastion) instance is created to communicate among application and Jenkins instances.

Bastion host is created in a public subnet with an ingress at port 22 to connect via SSH.

Application and Jenkins instances are created in a private subnet.

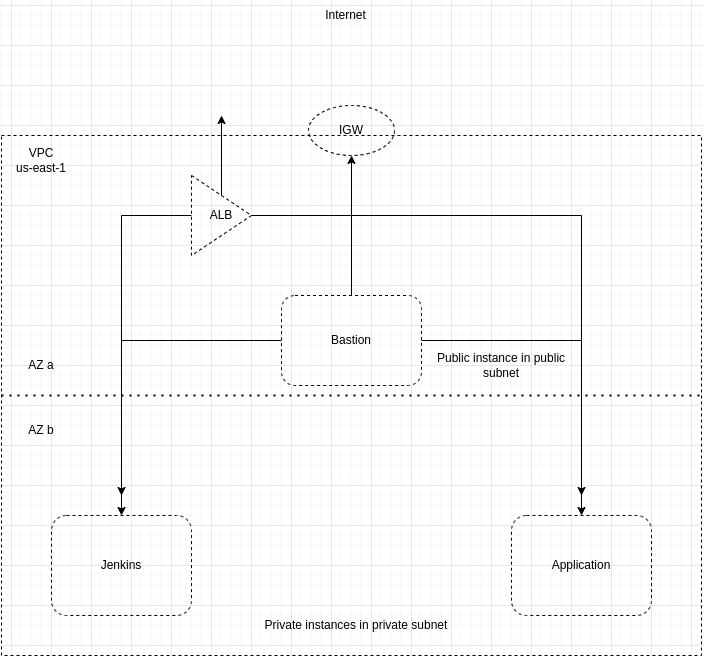
Bastion host SG - Allow self ip to ssh to ‘bastion’ instance and allow all egress.

Private Instances SG - Allow all incoming traffic from within VPC and all egress.

Public Web SG - Allow incoming to port 80 from self IP and all egress.

**Subtask 4**

Bastion, Jenkins and App instances are created using Ubuntu 20 AMI, t2.micro.



**Task 2**

**Subtask 1**

In the context of this project, ansible is installed on the local machine so that we can bootstrap private EC2 instances (Jenkins and application).

As basic checks using ansible script, both the private instances are checked for pending updates and then docker is installed on both the instances.

**Subtask 2**

Application load balancer is used to access Jenkins and Application instances from the internet. ALB is created in the same availability zones as the private instances as a requirement the ALB needs to be part of public subnets within the same availability zones as the private subnets. ALB listens on port 80.

ALB forwards /jenkins, /jenkins/\* to a target group having Jenkins host (port 8080) as the backend and it forwards /app, /app/\* to a target group having an app host (port 8080) as the backend.

**Subtask 3**

Once the ALB is set up to take /jenkins as a path to connect with Jenkins instance, we need to make sure that the Jenkins instance has Jenkins up and running to take requests.

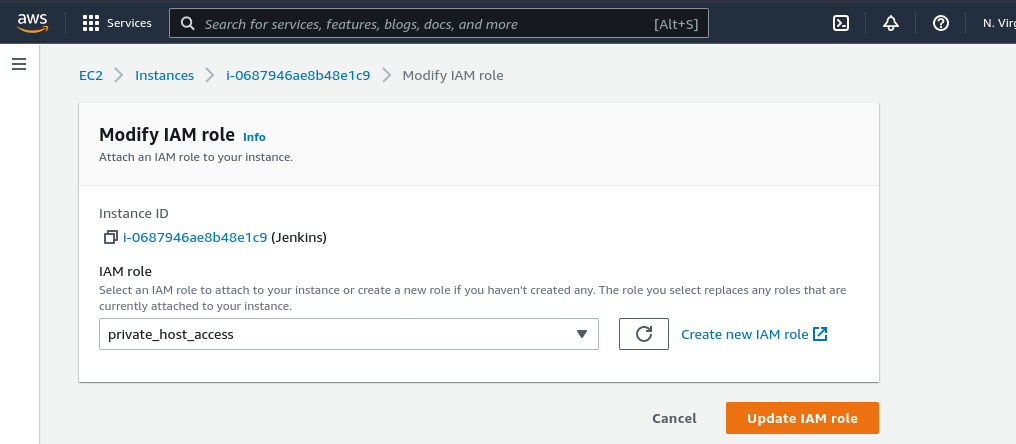
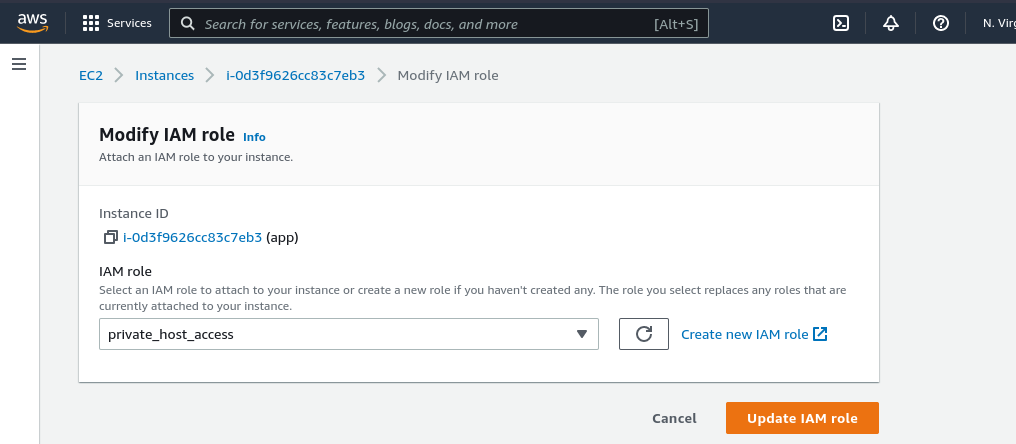
We need to manually install Jenkins on the Ubuntu instance.

Manually install Jenkins on the ‘jenkins’ instance and proceed with installing recommended plugins in jenkins. Access it via ALB endpoint to configure further (Install recommended plugins when prompted).

**Subtask 4**

A private ECR repository is created for storing docker images for the application code (sample code). An IAM role would be required to permit Jenkins and application host to connect with ECR. Ensure that ‘jenkins’ and ‘app’ hosts are authenticated to use the ECR repository you created.

Till now in order to connect to jenkins or application host we were using bastion host to ssh into these but now since jenkins has to deploy into app instance we need to create a new ssh key which will be used by jenkins to connect with app host to pull docker images and spin up container and deploy the application without the help of bastion host.



**Task 3**

**Subtask 1**

We need to create a code version repository in this case github and the sample code is pushed into that repository. The repository will be used to push other files and scripts created till now so that we can maintain versioning of changes.

Dockerfile is used to dockerise the application code to (node). The node application is exposed to port 8080.

**Subtask 2**

In this subtask jenkinsfile is created for the node application. This jenkins file has 2 stages.

In the first stage code is pulled from github which contains the docker file and application code. And in the second stage docker image is created using the pulled code from github and then this image is pushed to the ECR repository created earlier.

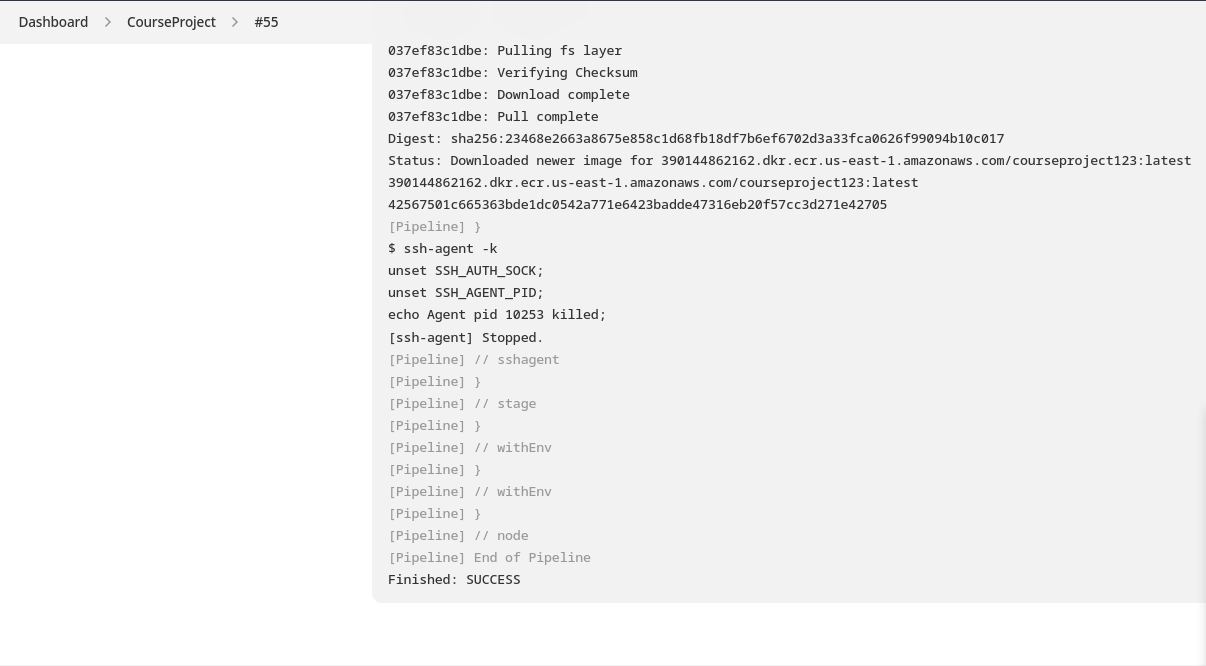
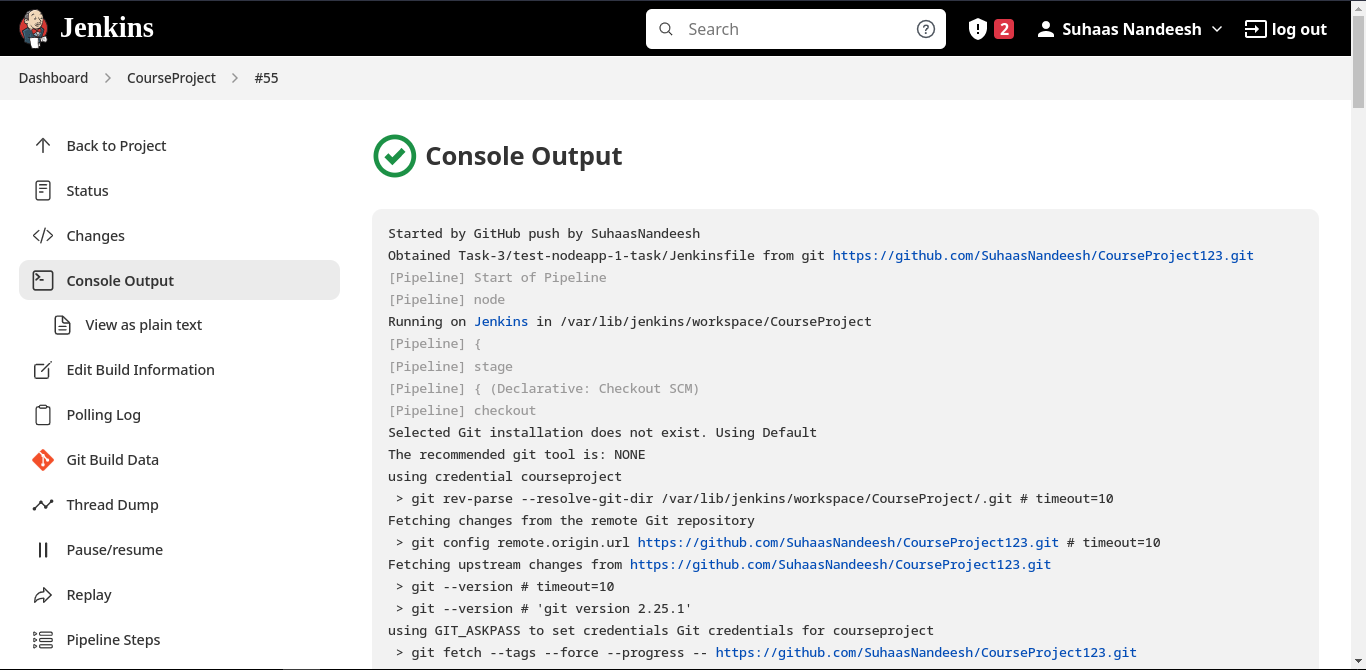
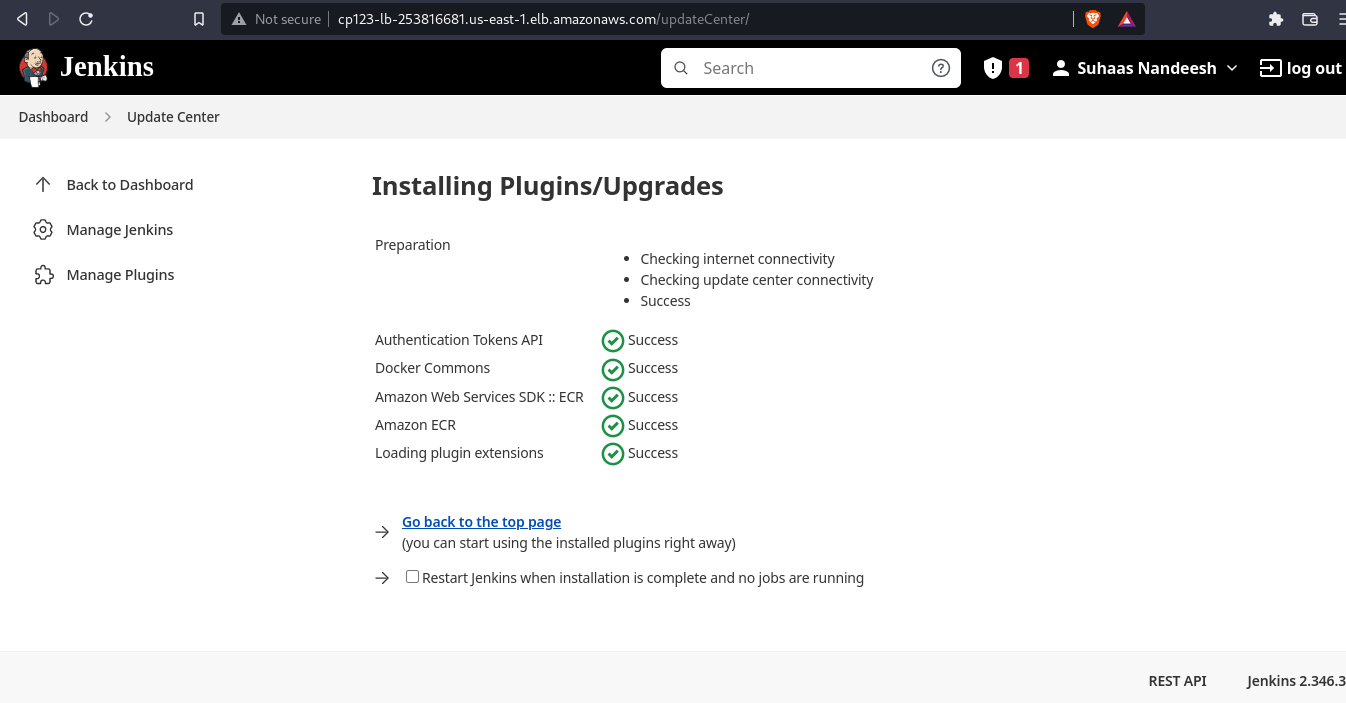
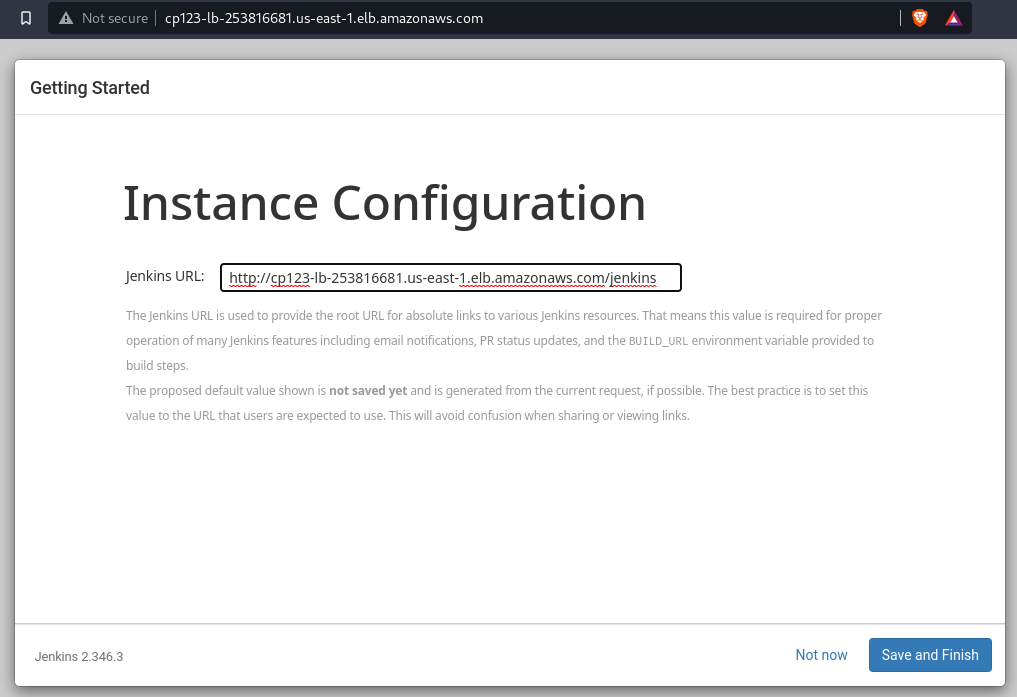
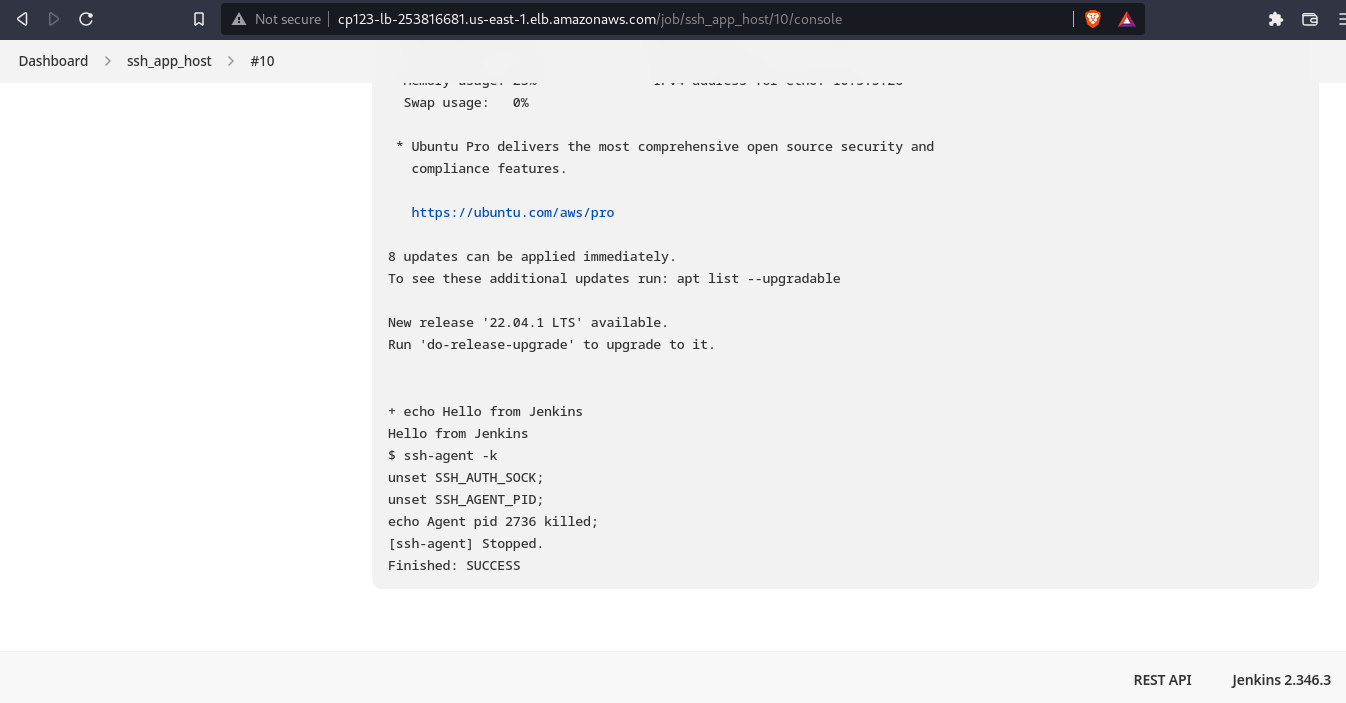
These stages are created using pipeline jobs in jenkins.

**Subtask 3**

In this subtask the docker images which were uploaded to the ECR repository are pulled into the application instance and are deployed. In order for the ECR repository to be pulled from the application instance it is better to have awscli installed in this instance as well.

Since the deploy step is part of jenkinsfile, the jenkins instance should be able to ssh into the application instance (this is possible using the ssh key generated in the previous task).

The deployment can be verified using the load balancer arn and the forwarding rule set in the Task 2



Estimated cost for this setup in AWS is as below

Considering 100% CPU utilisation and 5GB of data per month traffic for load balancer.

