**Infrastructure Optimization**

**Project Description :**

Create a DevOps infrastructure for an e-commerce application to run on high-availability mode.

**Background of the problem statement:**

A popular payment application, **EasyPay**where users add money to their wallet accounts, faces an issue in its payment success rate. The timeout that occurs with the connectivity of the database has been the reason for the issue.

While troubleshooting, it is found that the database server has several downtime instances at irregular intervals. This situation compels the company to create their own infrastructure that runs in high-availability mode.

Given that online shopping experiences continue to evolve as per customer expectations, the developers are driven to make their app more reliable, fast, and secure for improving the performance of the current system.

**Tools used in this project :**

1. EC2
2. Kubernetes
3. Docker
4. Ansible
5. <https://github.com/lfdcs/Infra_Optimization>

**Proposed Implementation Plan :**

Create a DevOps infrastructure for an e-commerce application to run on high-availability mode.

I will be creating a new EC2 instance associated with elastic IP for public IP with load balancer in AWS for kubernetes clusters in which 1 master node and 2 slaves are running for forming a kubernetes cluster to maintain an application in a high availability mode.

Ansible script is used to provision the EC2 instance of master and slave nodes, once the instance has been created with an ansible script. I will create a new kubernetes cluster and enable pods using the kubectl. I will be taking a snapshot of the ETCD database and also setting up the criteria if average utilization goes beyond 50% then autoscale up the services to maintain the load so that this will mitigate the timeout error and irregular down time.

**Tester Details:**

User named “**Pod-reader**” has been created for testing these test case and all the snapshot has been attached along with the results

**Concepts Used to Build :**

To Avoid irregular intervals downtime and scale according to the cpu load of 50% to meet the customer expectations for online shopping.

* Used IaC (Infrastructure as a Code) to provision the worker node (Server) within few seconds
* Running the application in container technology using kubernetes to automatically scale the containers based on the cpu load (of 50%) for the customer not to face any performance issue while shopping.

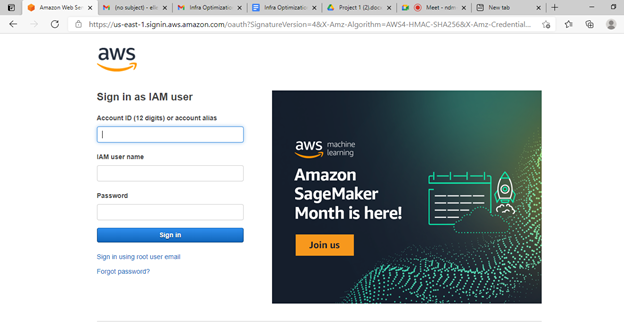
**This Documents covers the below Implementation Requirements:**

1. Create the cluster (EC2 instances with load balancer and elastic IP in case of AWS)  
2. Automate the provisioning of an EC2 instance using Ansible  
3. Install Docker and Kubernetes on the cluster  
4. Implement the network policies at the database pod to allow ingress traffic from the front-end application pod  
5. Create a new user with permissions to create, list, get, update, and delete pods  
6. Configure application on the pod  
7. Take snapshot of ETCD database  
8. Set criteria such that if the memory of CPU goes beyond 50%, environments automatically get scaled up and configured

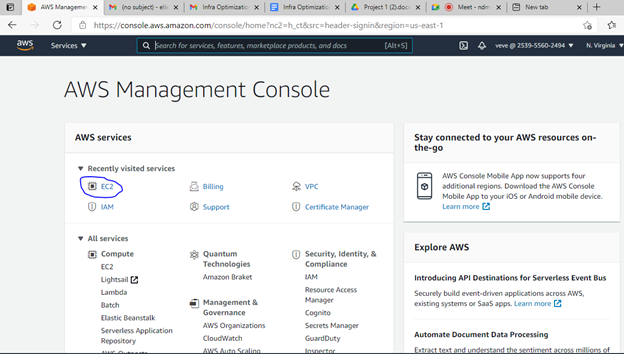
**1. Create the cluster (EC2 instances with load balancer and elastic IP in case of AWS)**

**Section 1 - Creation of the EC2 Instance :**

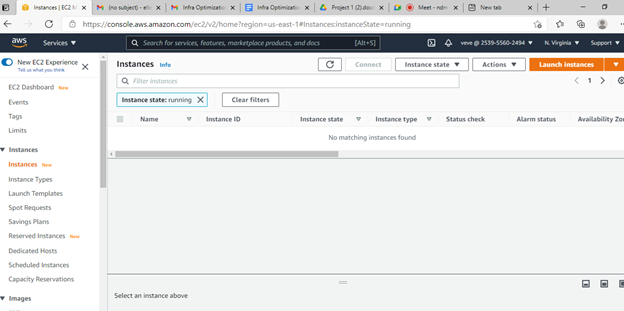
Login to the AWS Console using the URL - https://aws.amazon.com



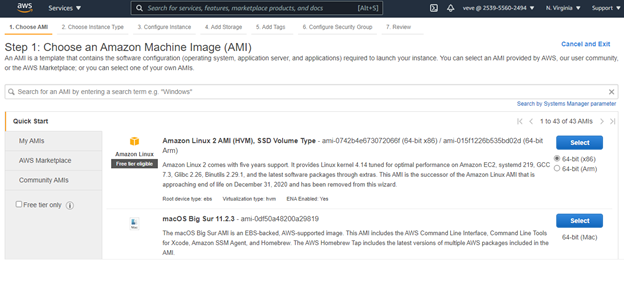
Search the Service EC2 in the AWS console.



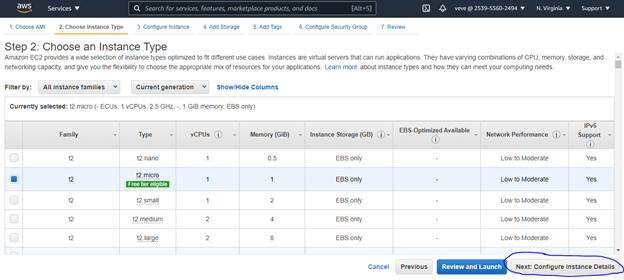
Choose the Region on the right top corner as N.Virginia.



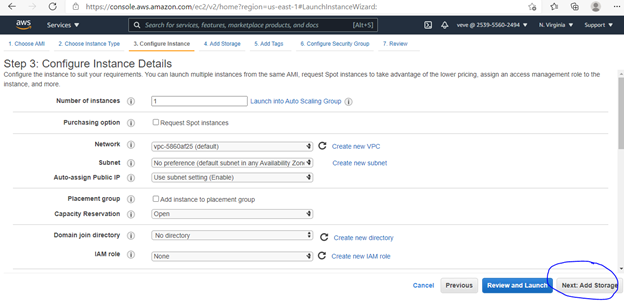
Click Launch Instance Choose the AMI in the marketplace (Amazon Linux)



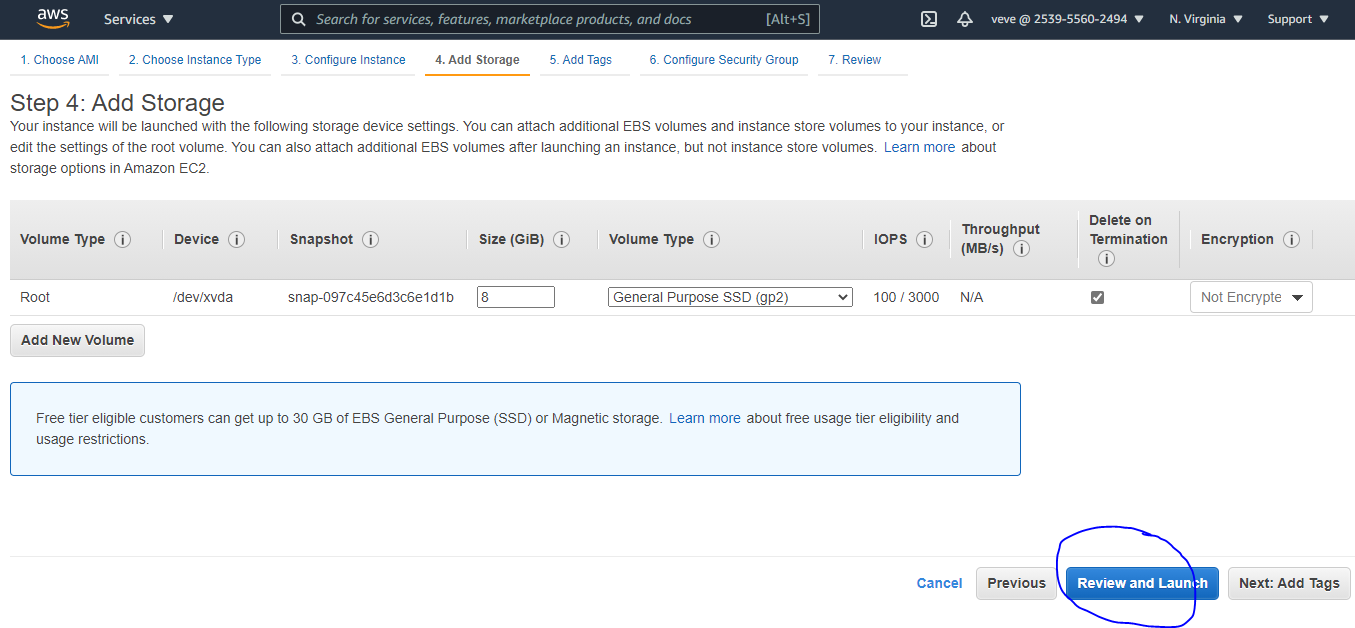
Select the Instance Type

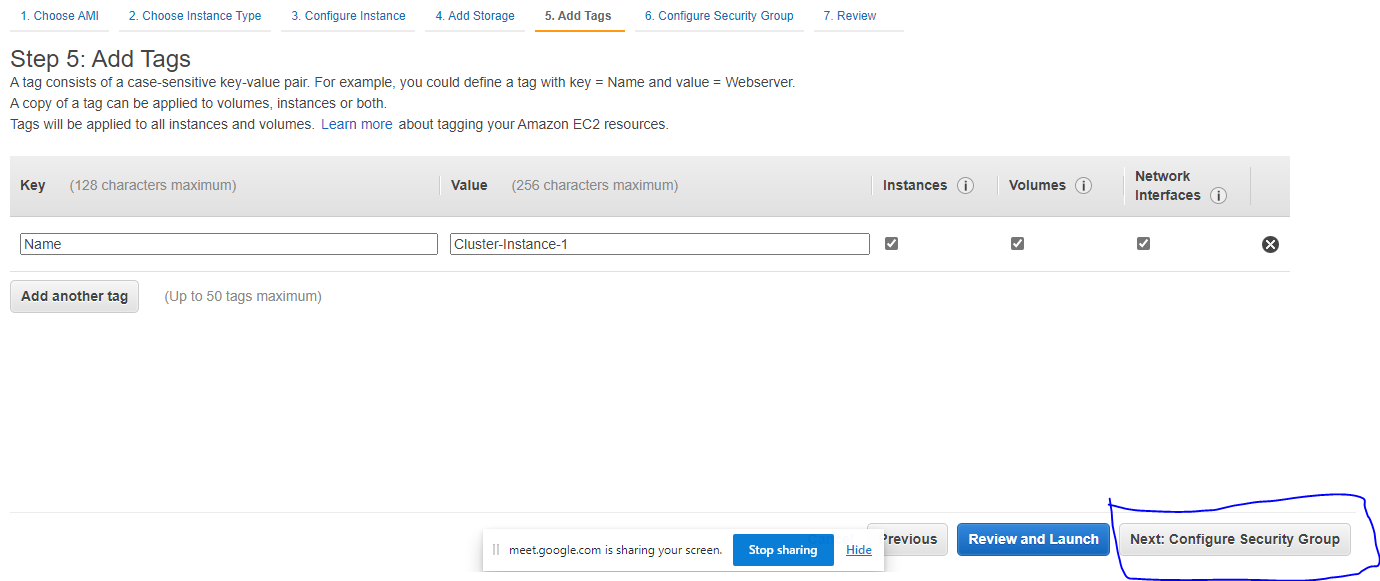


Configure the Instance details  
Select the Default VPC and Subnet and select Auto-Assign-Public IP

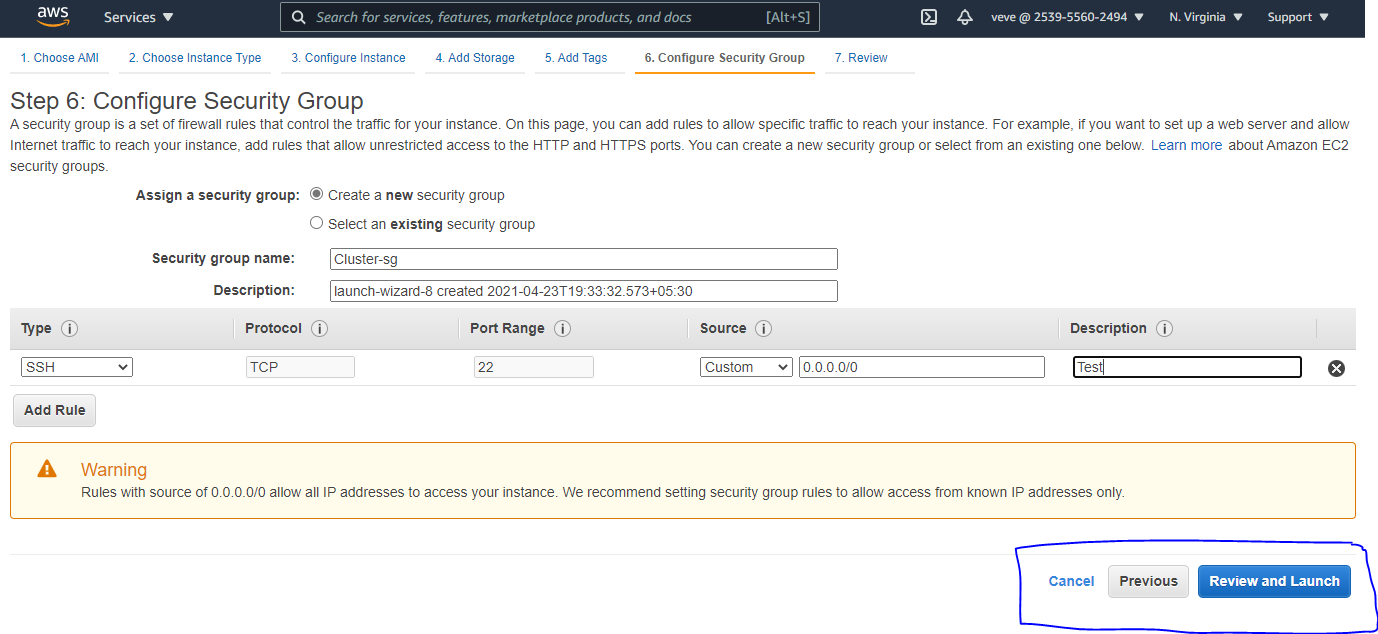


Select the default storage

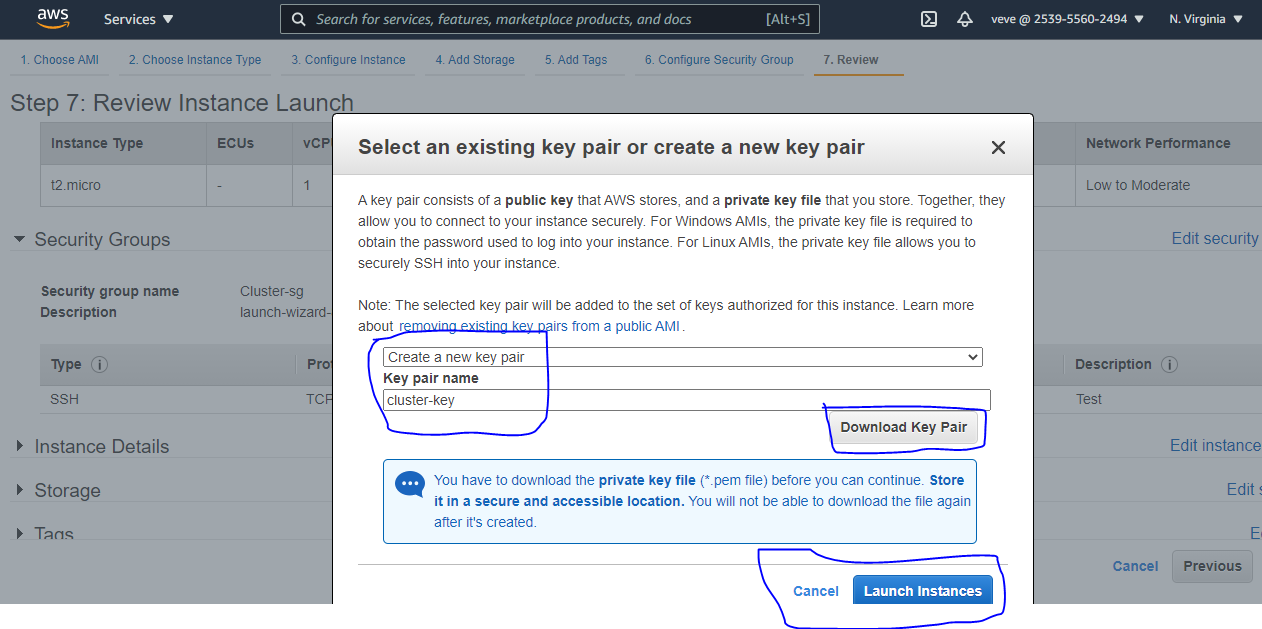


Add the required tag to the instance for the identification.

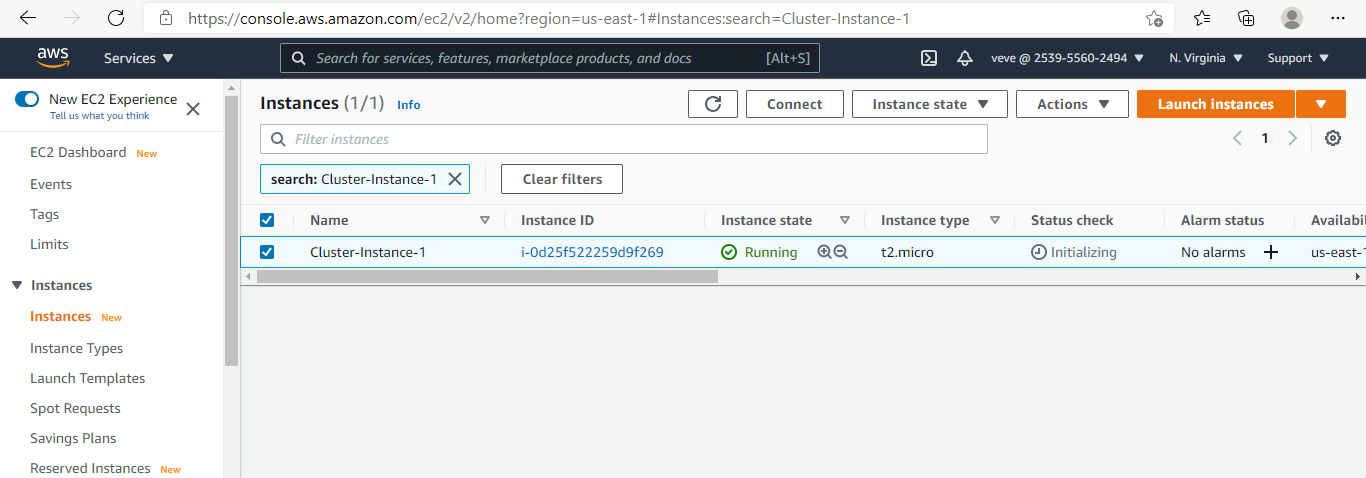
Select the create New security Group and add the Port 22 with all open for the source.



Create the new pair for the login purpose and download the PEM and store in the secure place

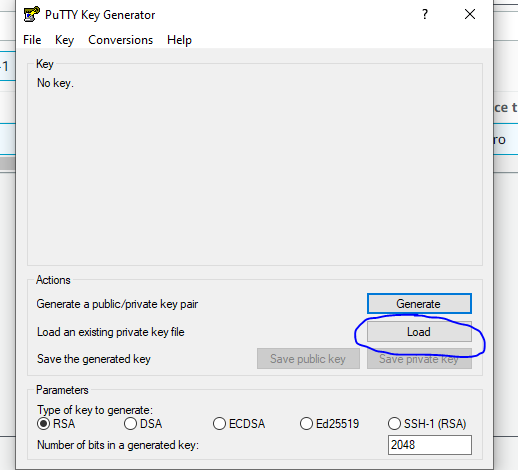


EC2 instance is created successfully.

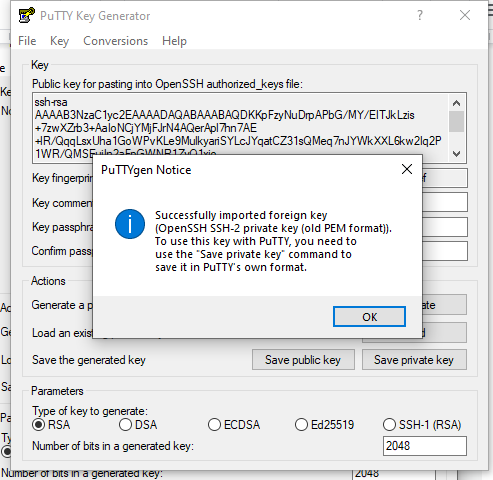


**Section 2 - Steps to Login to the EC2 Instance.**

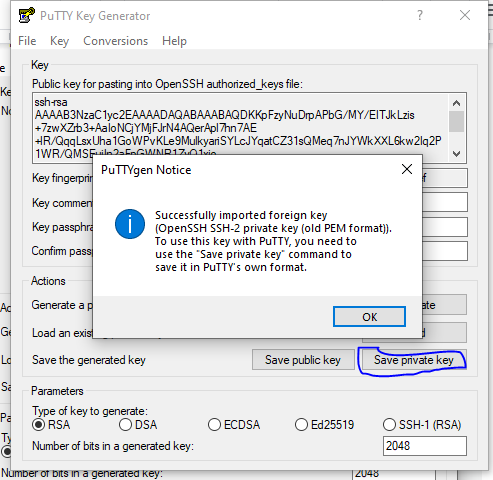
Convert the Pem to PPk file using the following steps:

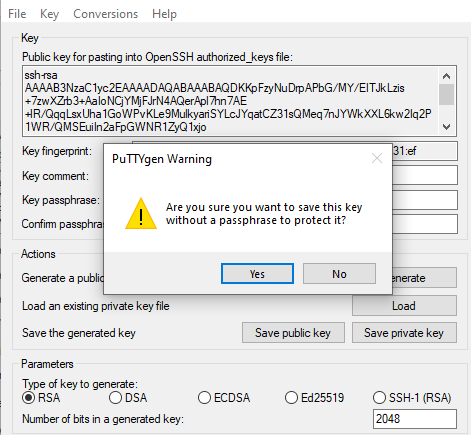


Browse the Pem Key and load the file.



Click OK and save the Private key

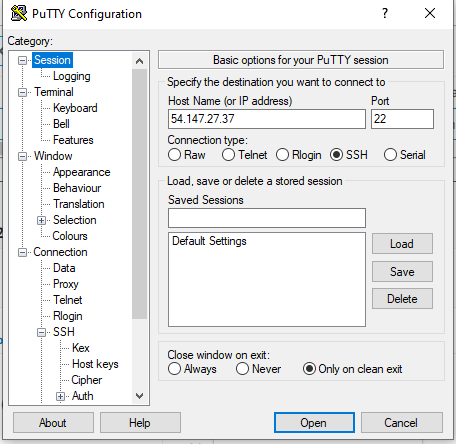




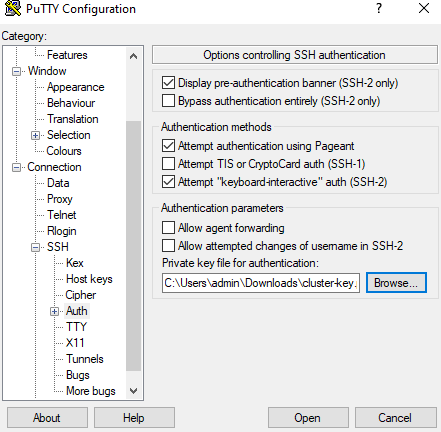
Click yes

Save the PPK file

Try Logging in to the Instance Using the Putty.Paste the IP in the Host Name.



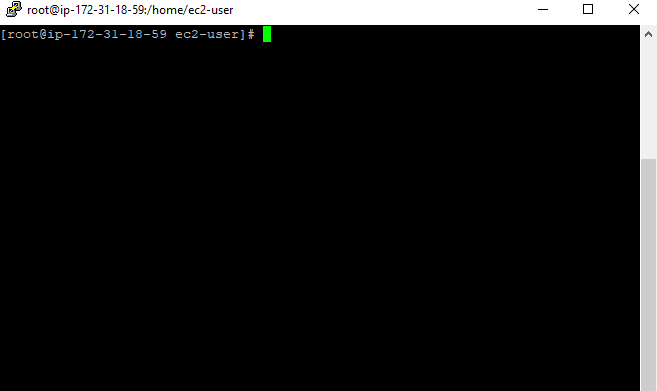
Select the Auth and Browse the PPK file and select it.



Click open



Switch to root user using the command sudo su



**Section 3 : Installation and Configuration of the Ansible**

Run the following command in the putty session:  
  
Note: type Y when were the pop up comes

1.sudo amazon-linux-extras install ansible2

2.yum makecache

3.sudo amazon-linux-extras install epel

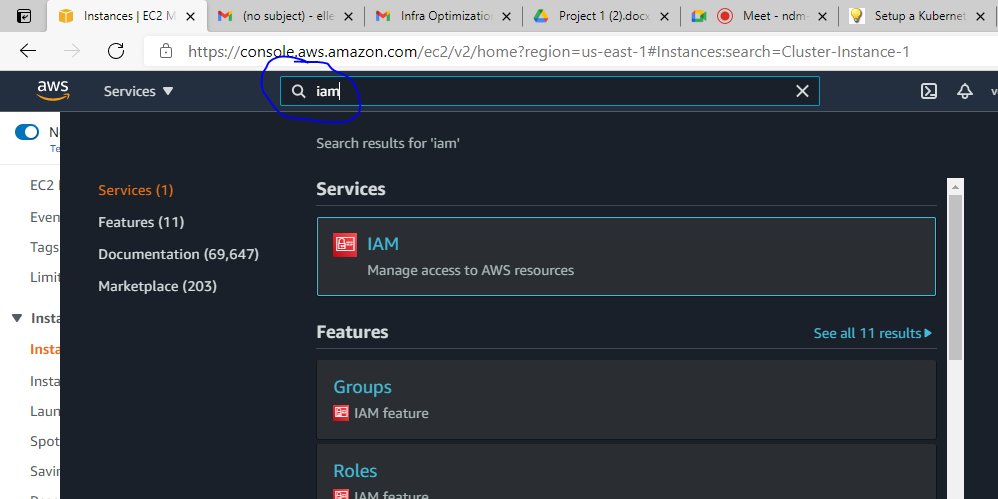
4.yum makecache

5.yum install python-pip

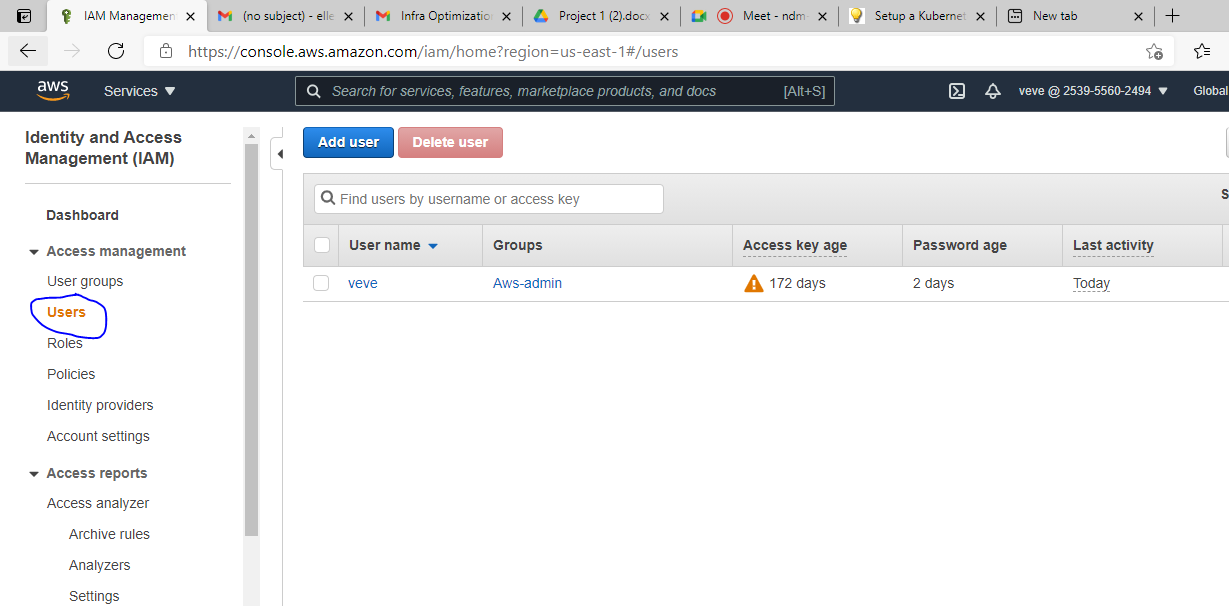
6.pip install boto

Package Installation is completed.

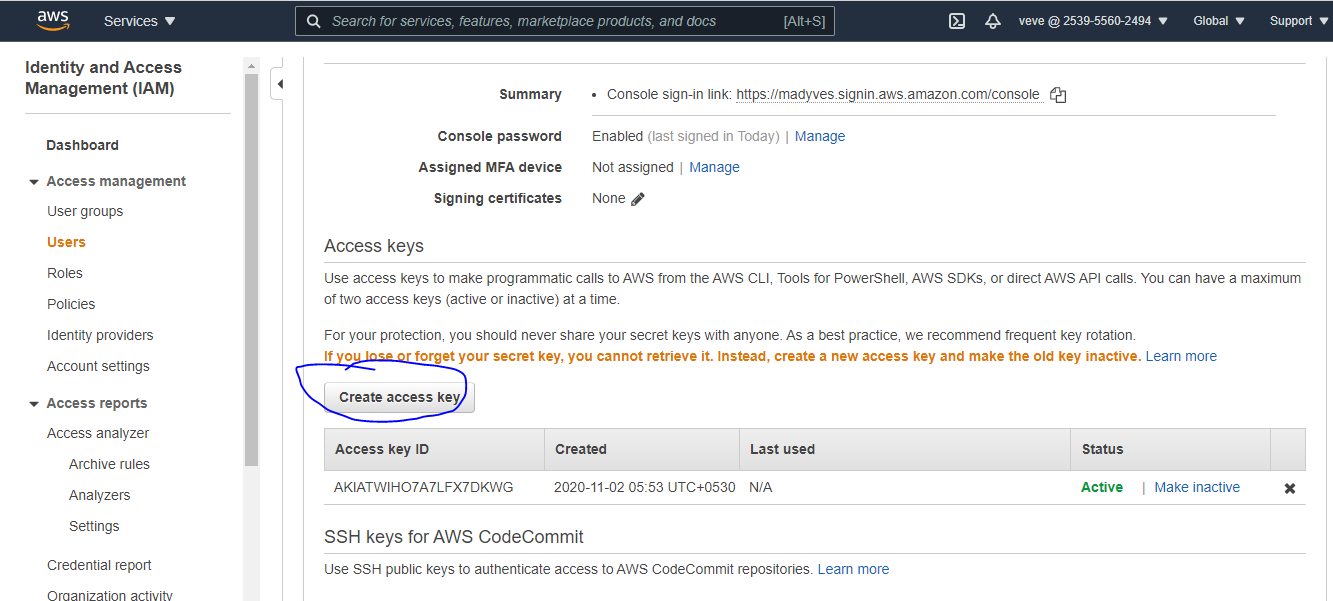
Create the Access key and the secret key for the root user or IAM user for the AWS CLI configuration.

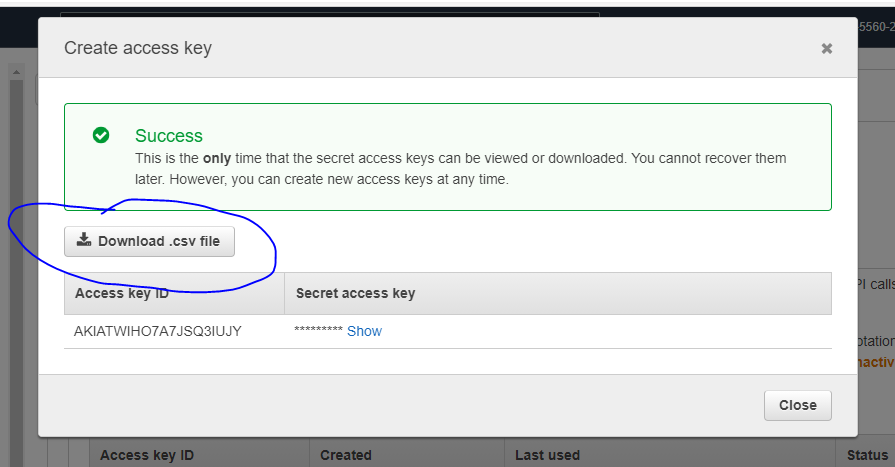


Select the users and open the specific user details



Create the Access key and download the file.



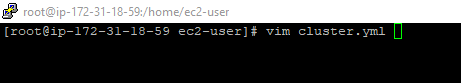


**2. Automate the provisioning of an EC2 instance using Ansible**

Steps to create the EC2 Using the Ansible script:

Create the .yml file for the anisble script:

1.vim cluster.yml



Update the file with the below content

Copy Paste the Below Script in the yml file.

- hosts: localhost

tasks:

- ec2:

aws\_access\_key: AKIATWIHO7A7JSQ3IUJY

aws\_secret\_key: 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

key\_name: cluster-key

group: Cluster-sg

instance\_type: t2.large

image: ami-0742b4e673072066f

wait: yes

wait\_timeout: 500

count: 1

instance\_tags:

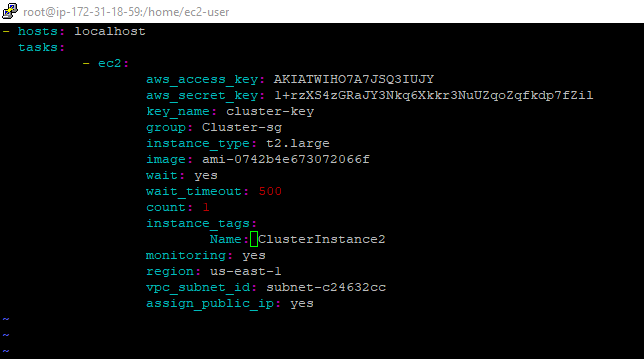
Name: ClusterInstance

monitoring: yes

region: us-east-1

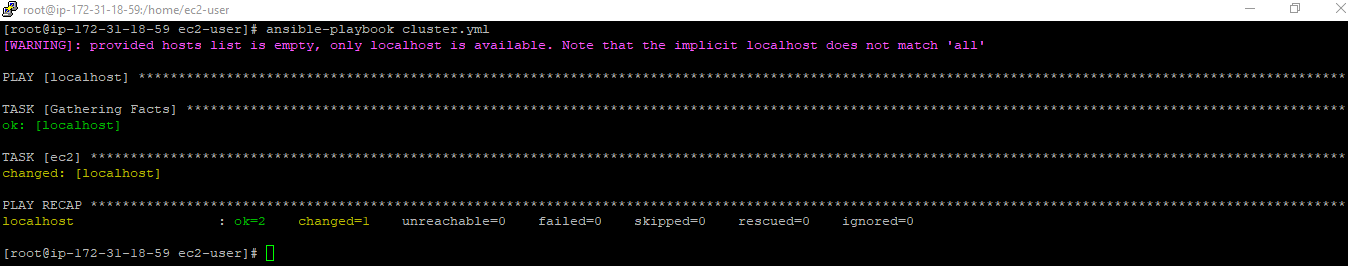
vpc\_subnet\_id: subnet-c24632cc

assign\_public\_ip: yes



Using the command :wq save the file and exit

Run the script using the command - ansible-playbook cluster.yml



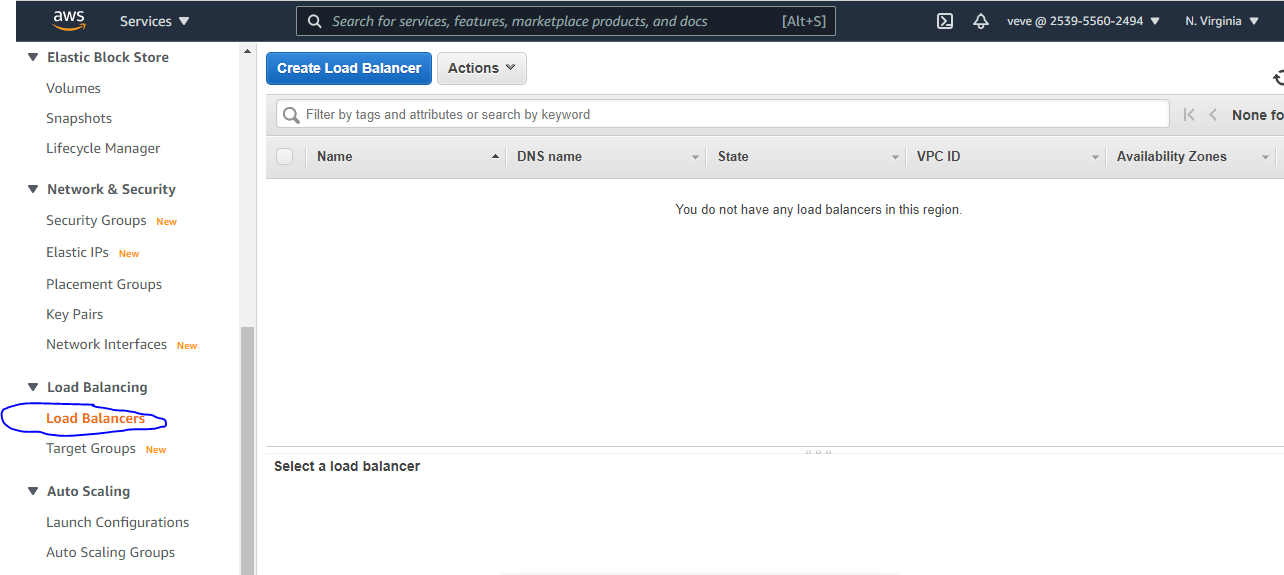
Run the command n number of times to create the as many instances required, just change the instance name to avoid any error for duplication.

To Login to the created cluster instance we can follow the same procedure that was mentioned in the above.

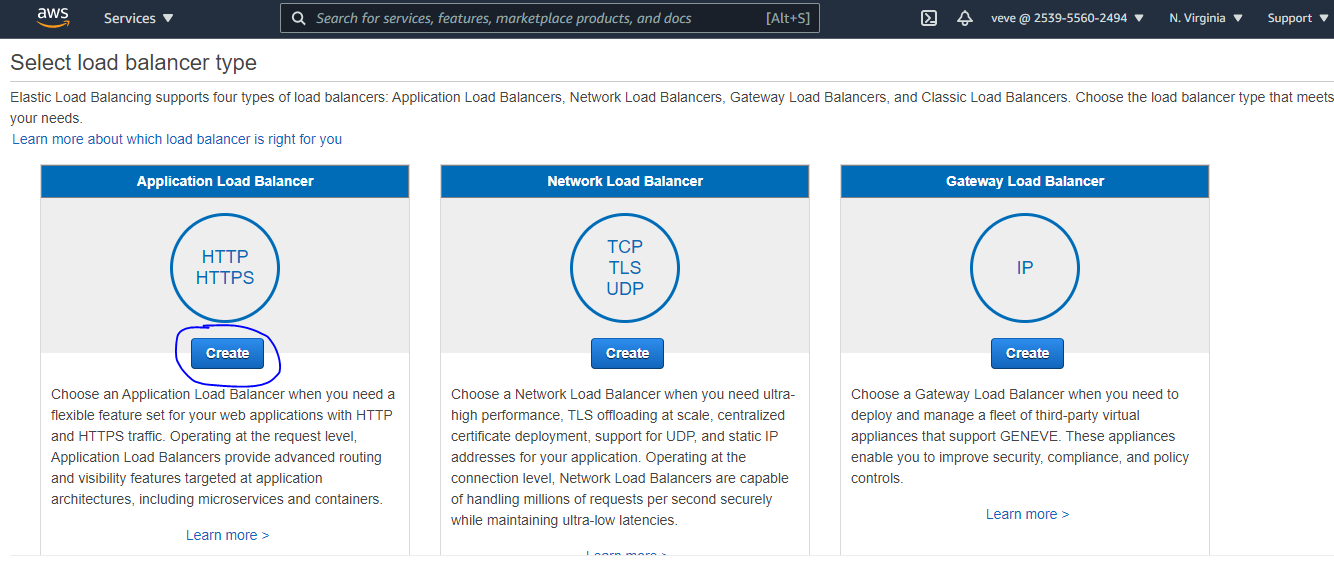
Steps to create the Load Balancer and Taget Group and Attach the instance in the Target Group:

**Section 4 : Creation of the Load Balancer and Target Group**

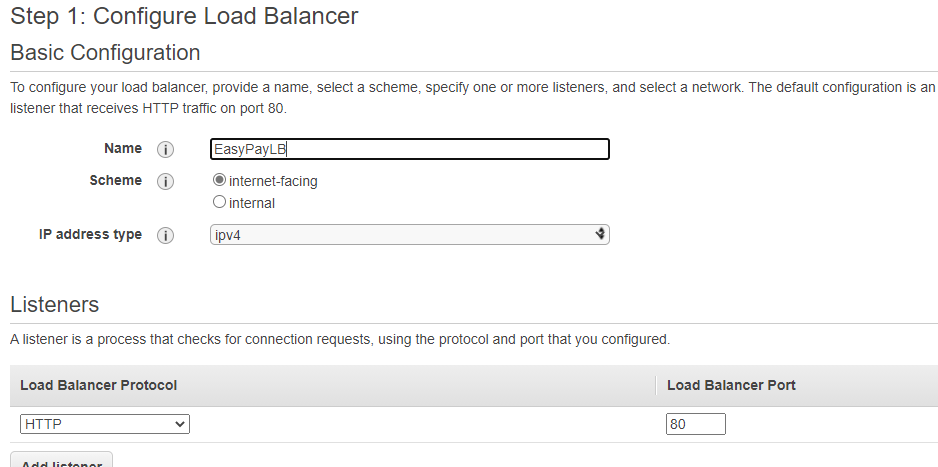
1.EC2 Console scroll down and click the load balancer service.



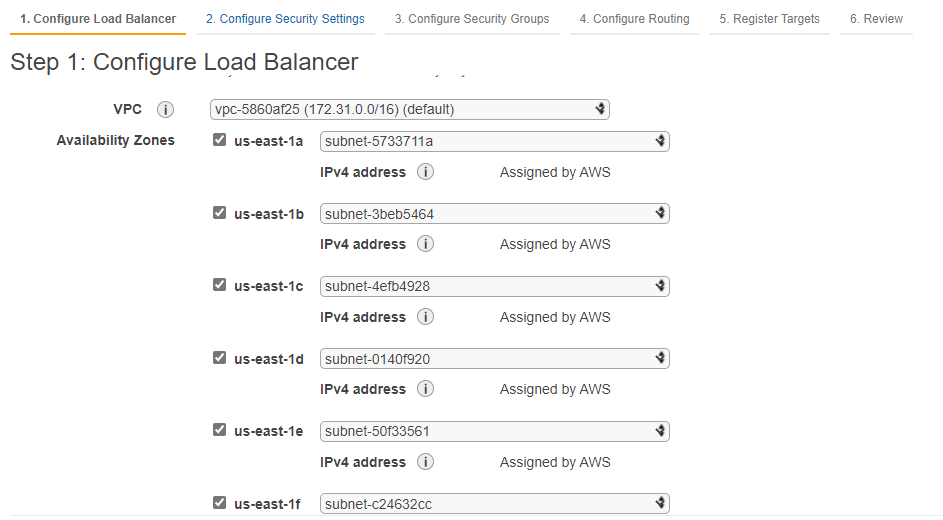
Select the Application Load Balancer:



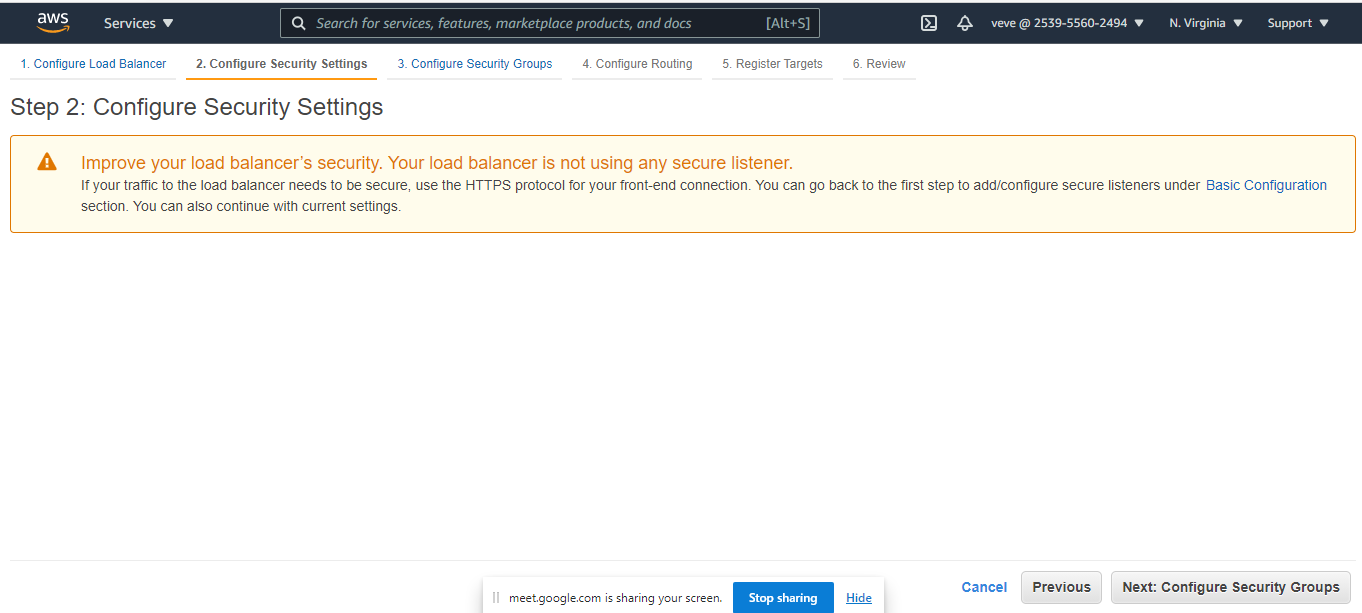
Name the Load balancer and Select the port for the incoming traffic:



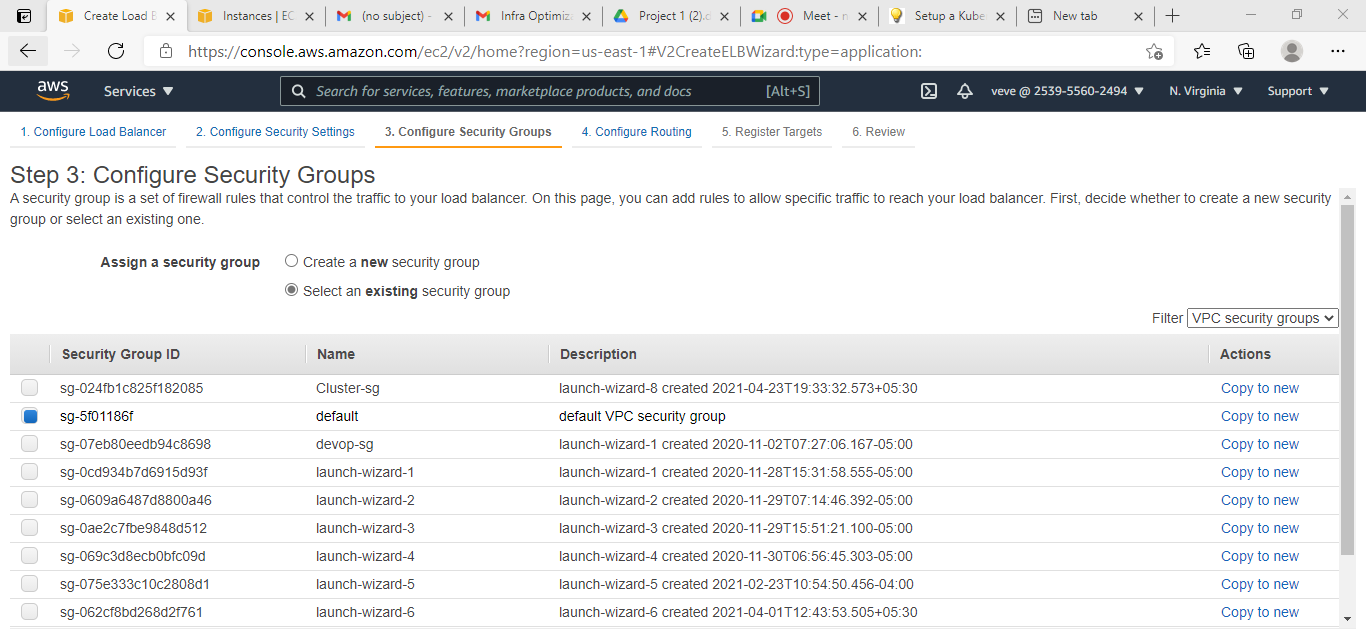
Select the VPC and Subnet:



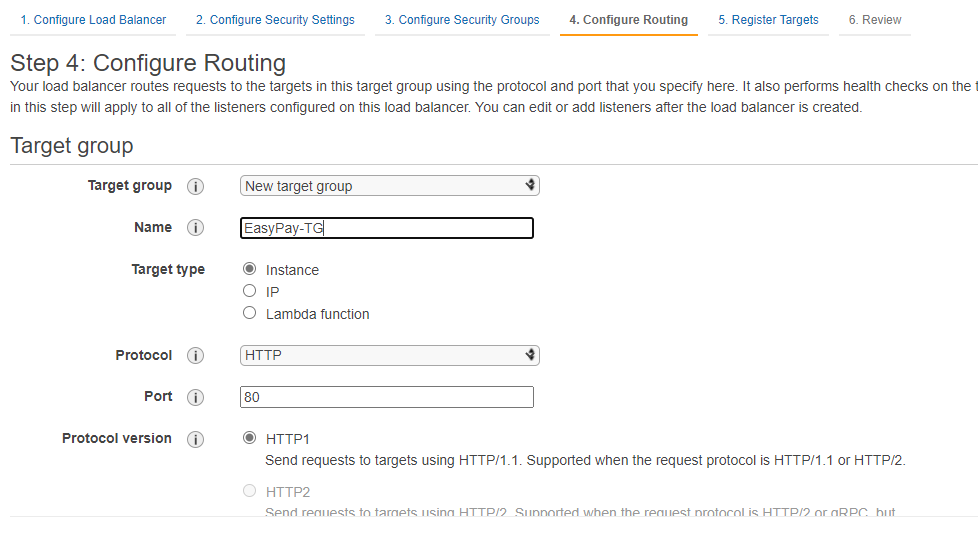
Leave the Step 2 as we are using only the HTTP port :



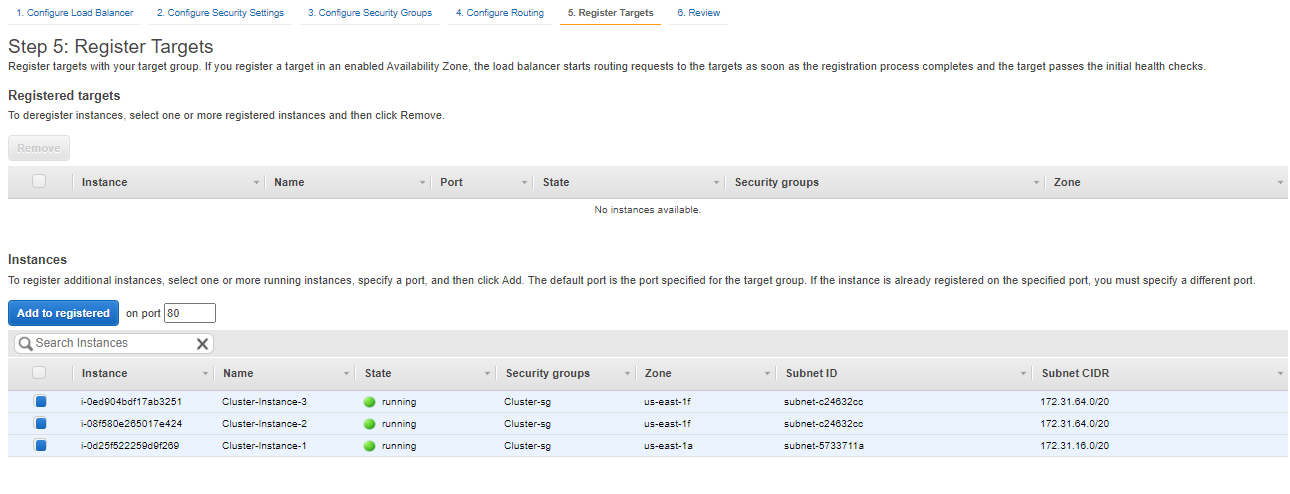
Select the default SG in the Load balancer :

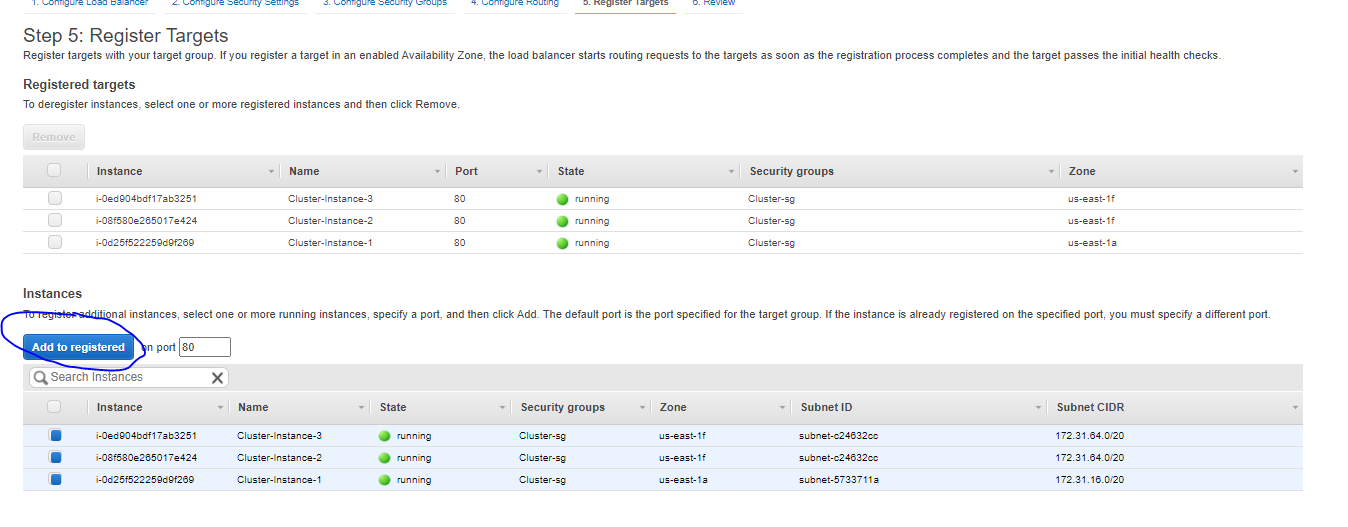


Create the target group and configure the TG.

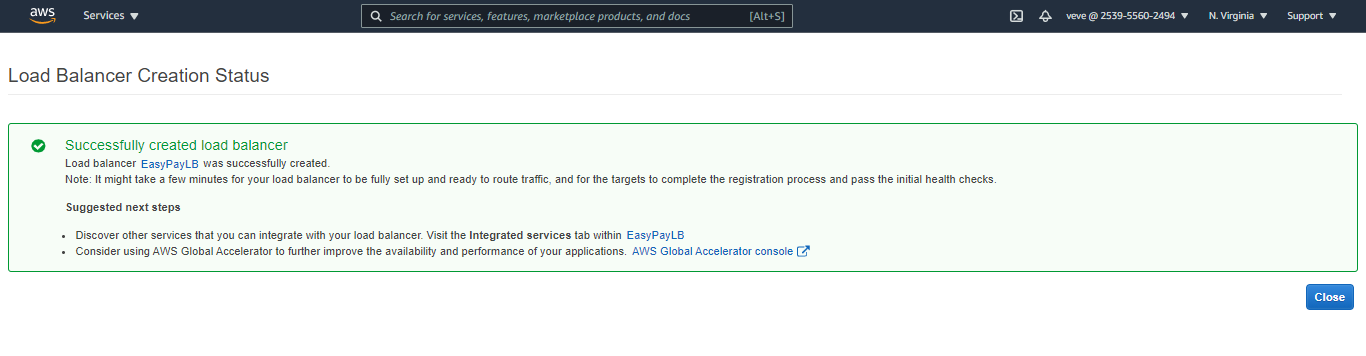


Add the Three cluster machine in the Target Group created in the Port 80





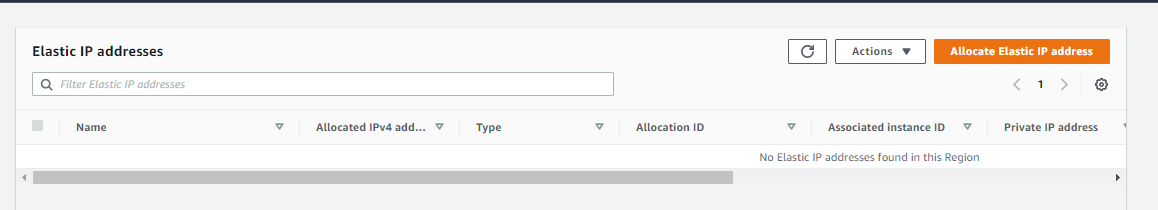
Click review and create the Load Balancer

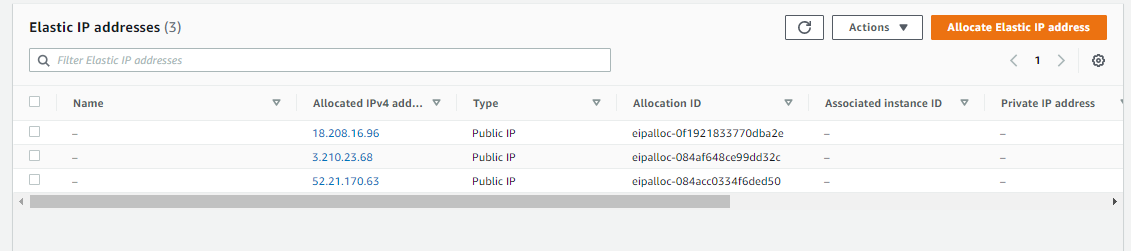


**Step 5: Creation of Elastic IP address to the Instance:**

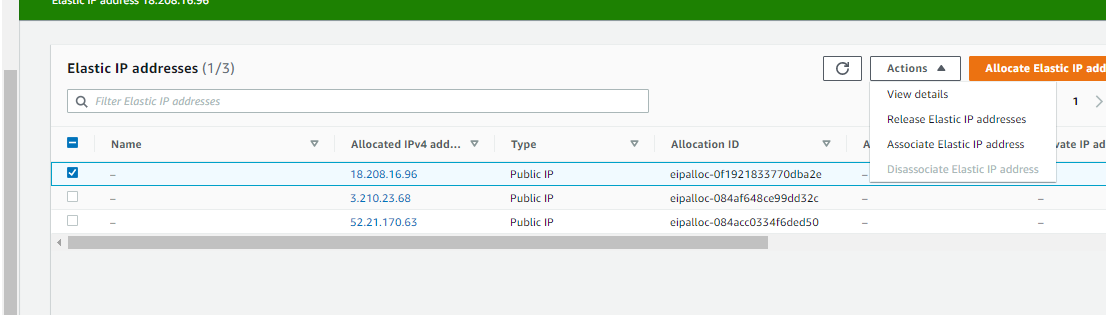
In the EC2 console select the Elastic IP

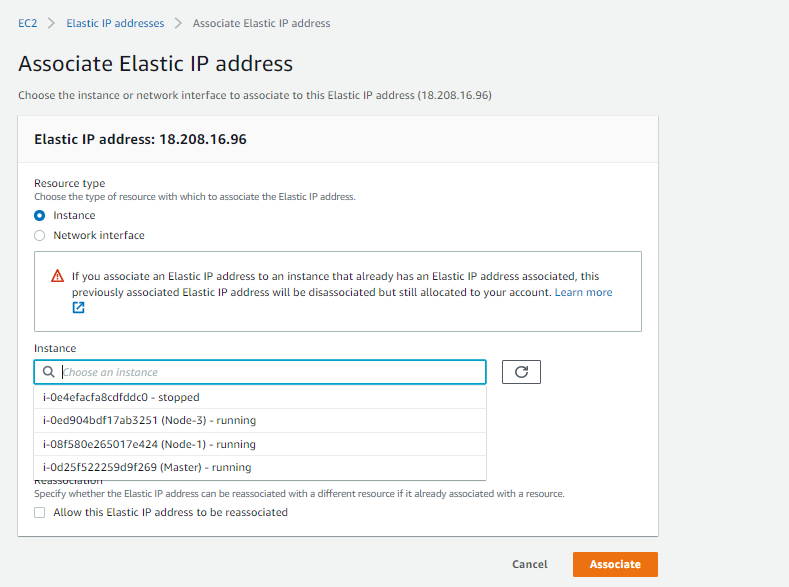
Click Allocate Elastic IP Address and Create 3 ranges.

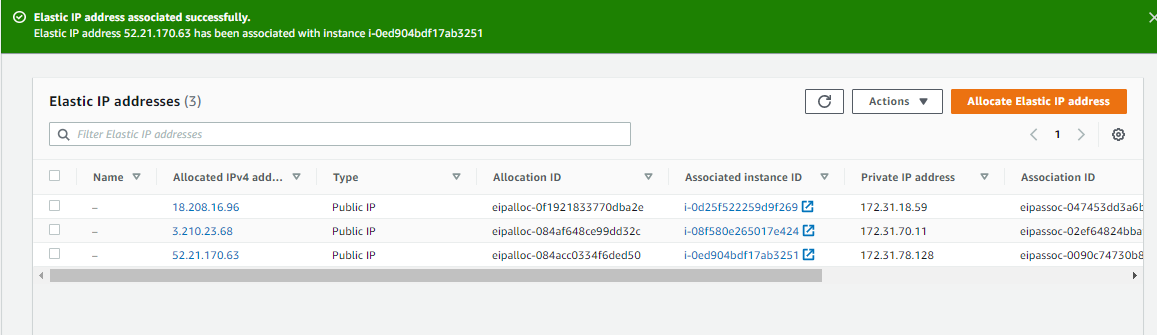




Mark one of the IP and click action and Click Associate IP to the Instance, repeat the same to the other Two instances.



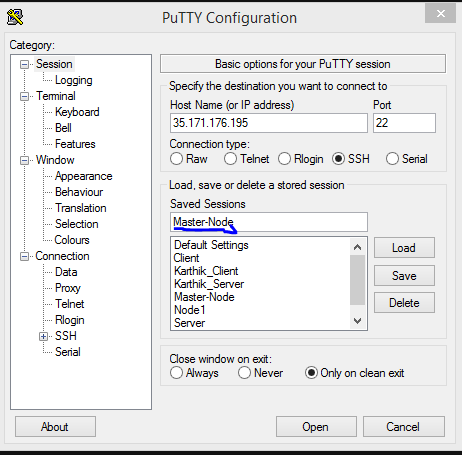


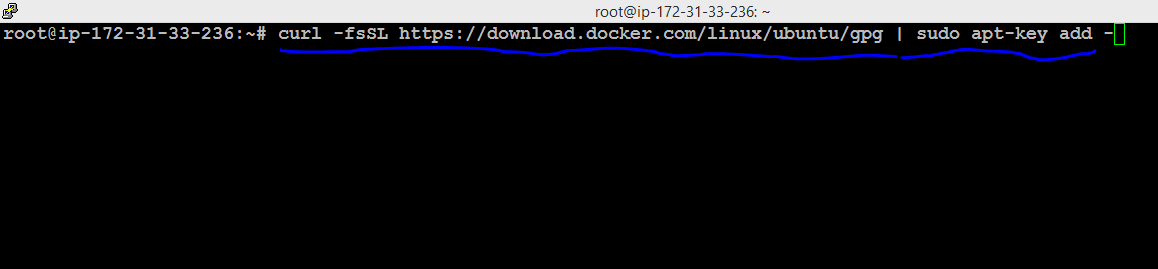


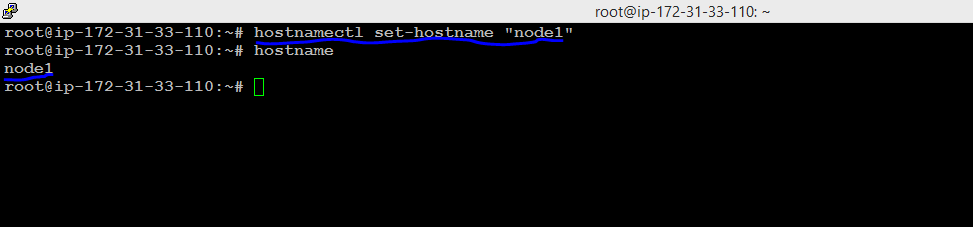
Now three EC2 instances are attached with the Elastic IP.

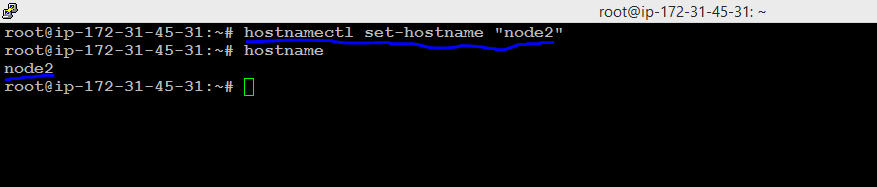
**3. Install Docker and Kubernetes on the cluster**

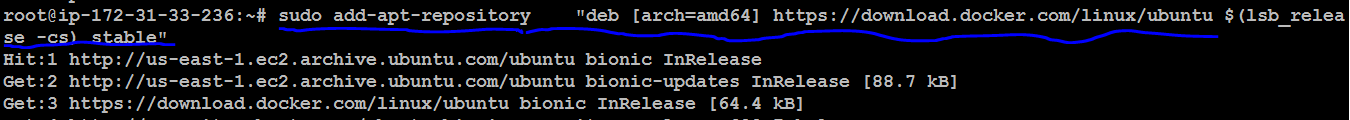
Connecting to Master-Node through Putty

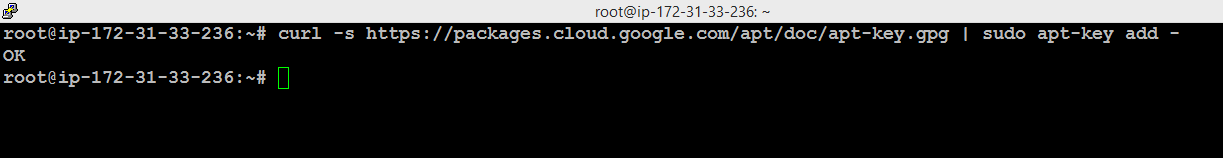


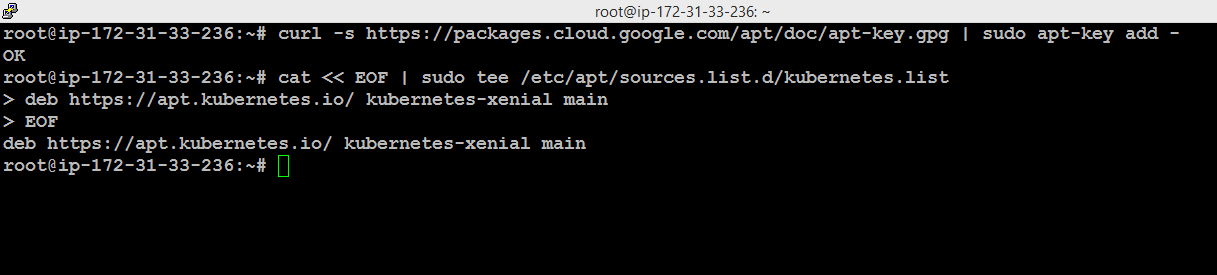


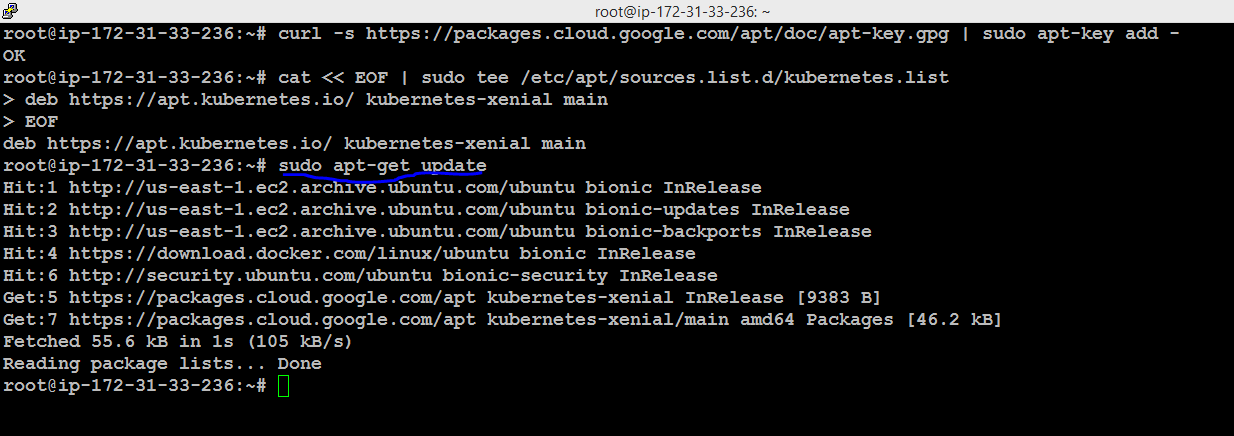


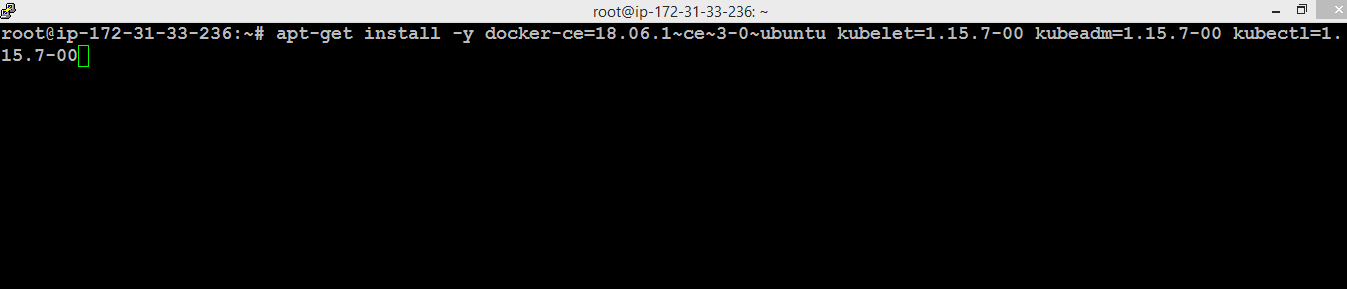


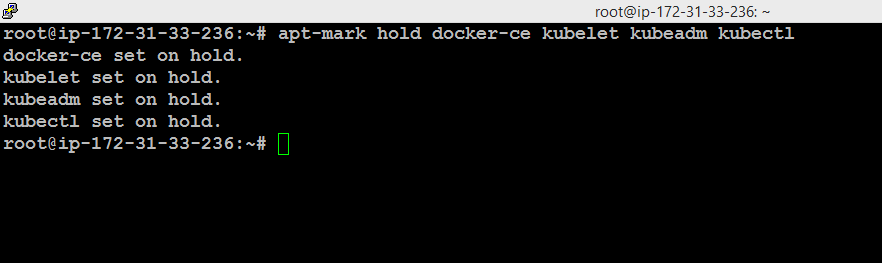


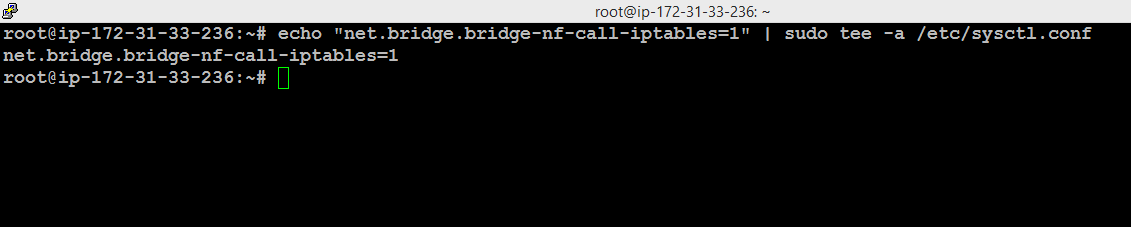


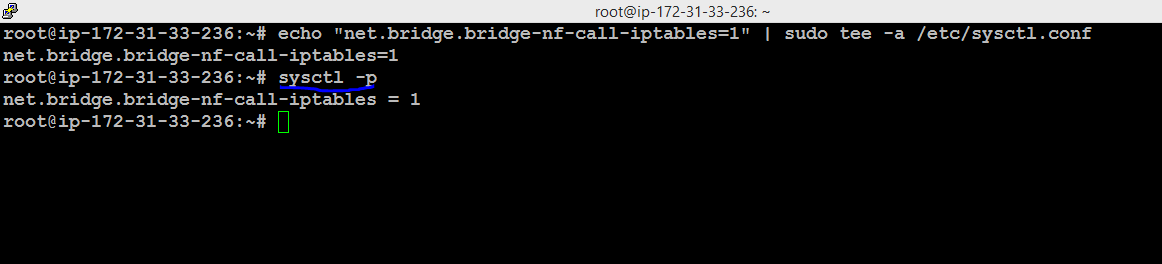


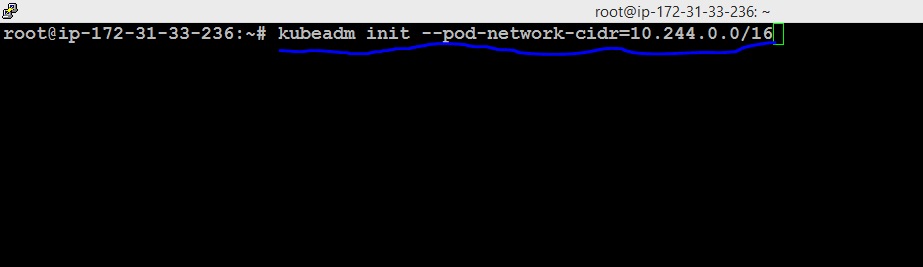


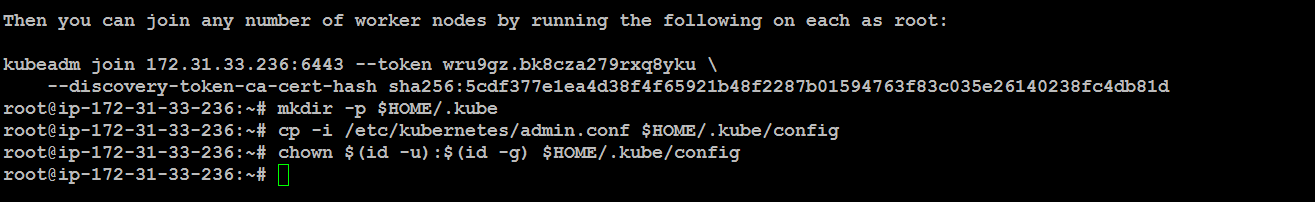


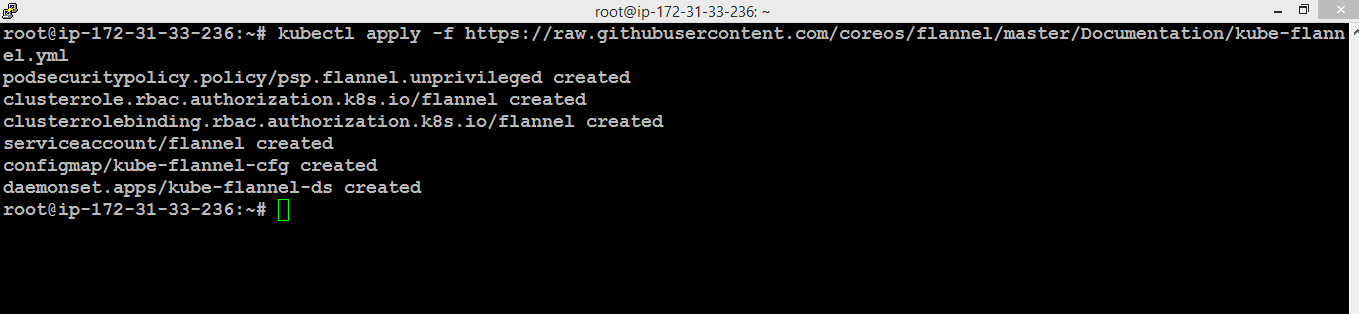






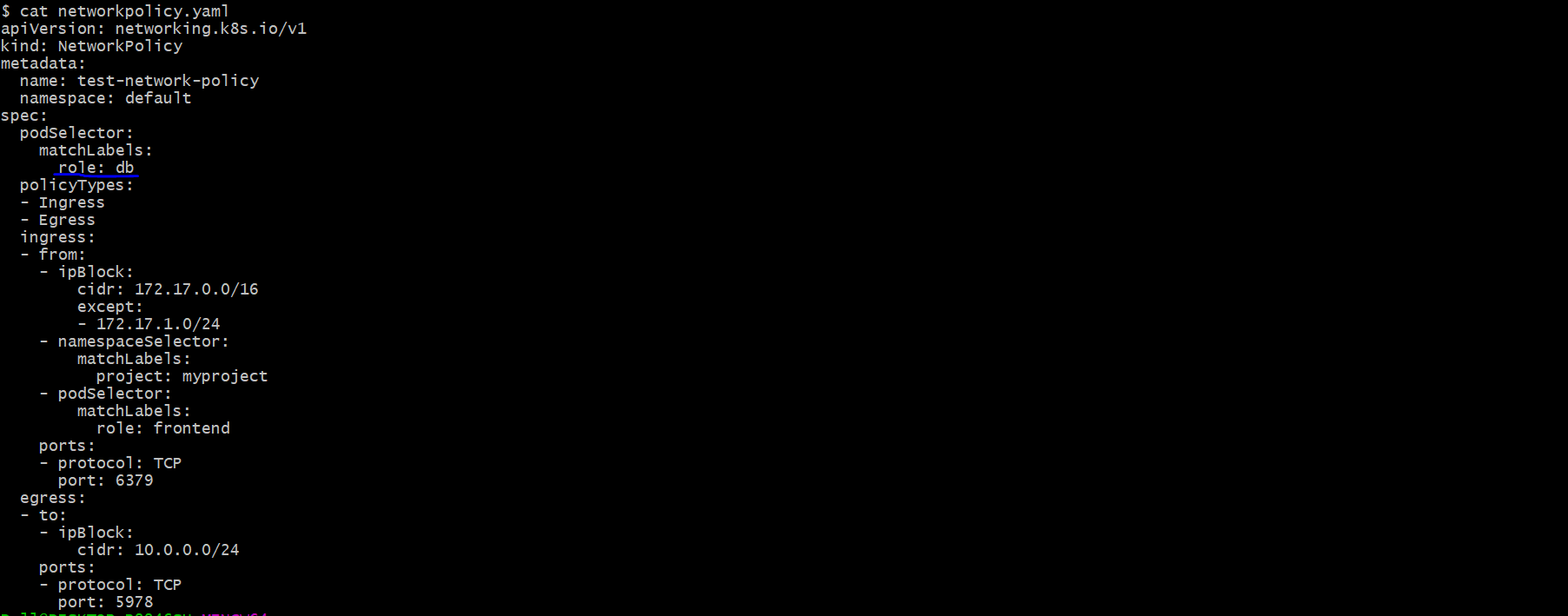




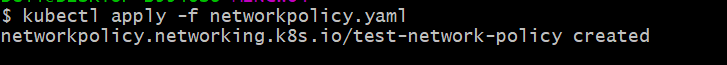


**4. Implement the network policies at the database pod to allow ingress traffic from the front-end application pod**

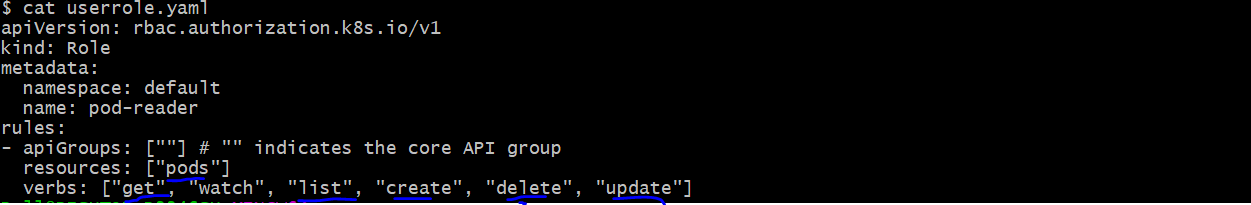
File named networkpolicy.yaml has been created for implementing network policies at the database pod to allow ingress traffic from the front-end application pod.

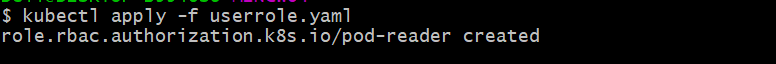


After creating the networkpolicy.yaml file, applied the policy using the kubectl apply command by mentioned the file name

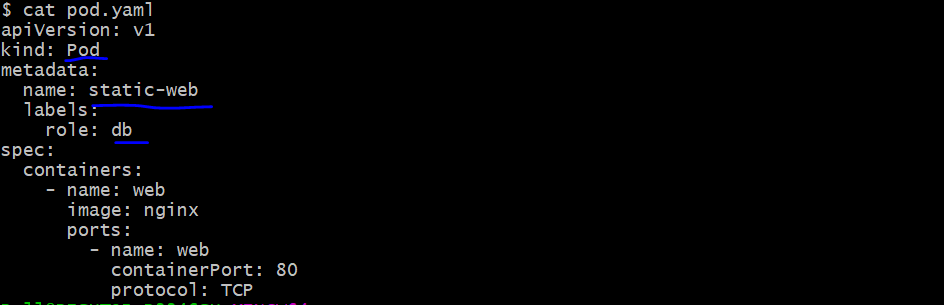


**5. Create a new user with permissions to create, list, get, update, and delete pods**

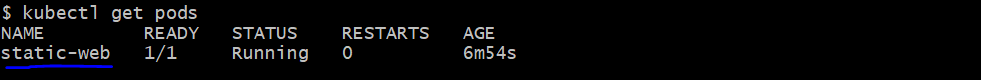
****

****

**6. Configure application on the pod**

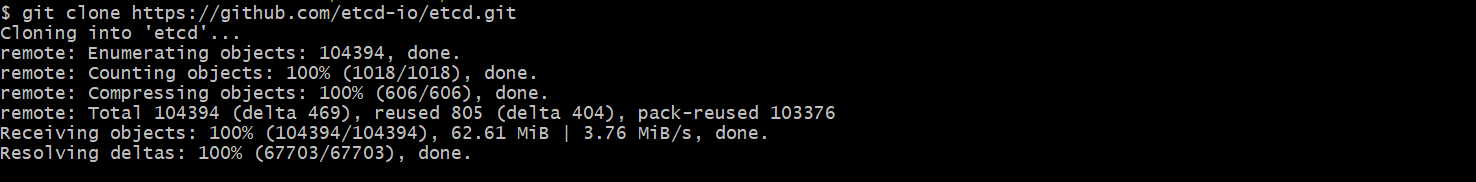




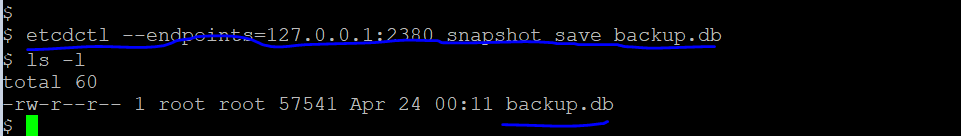


**7. Take snapshot of ETCD database**

**Download the etcd from github and Install the etcdctl**

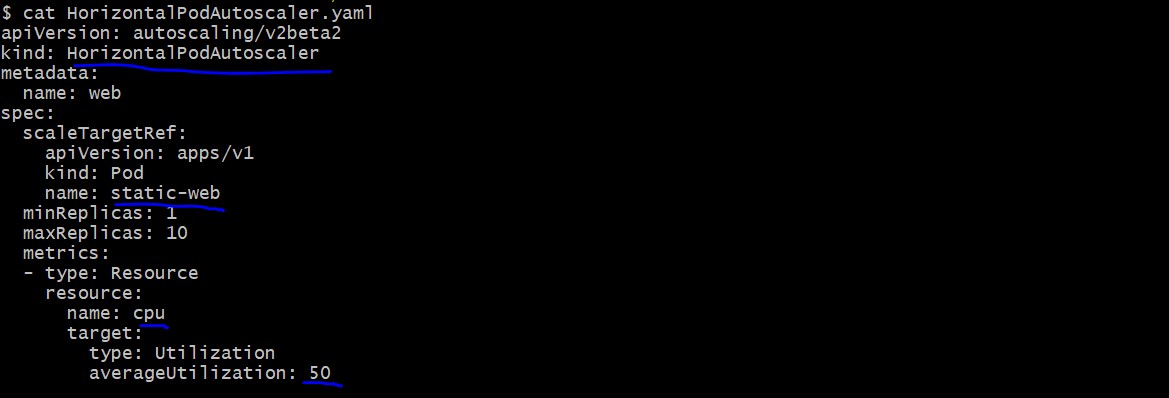
****

**snapshot using the etcdctl**

****

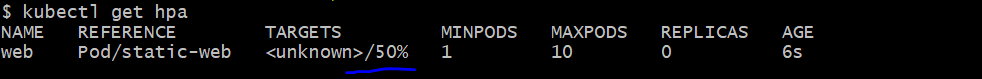
**8. Set criteria such that if the memory of CPU goes beyond 50%, environments automatically get scaled up and configured**

**Please find the Autoscaler yaml file**

****

**Apply the HorizontalpodAutoscalar.yaml file using kubectl**

****

****

**Conclusion:**

Developed a solution for more reliable, fast, and secure for improving the performance of the current system for online shopping experience that continues to evolve as per customer expectation.