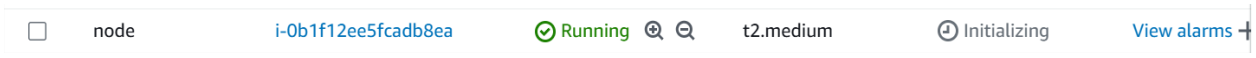
Name: Anish Kulkarni Roll No.: 29 Class: D15C AY: 2024-25

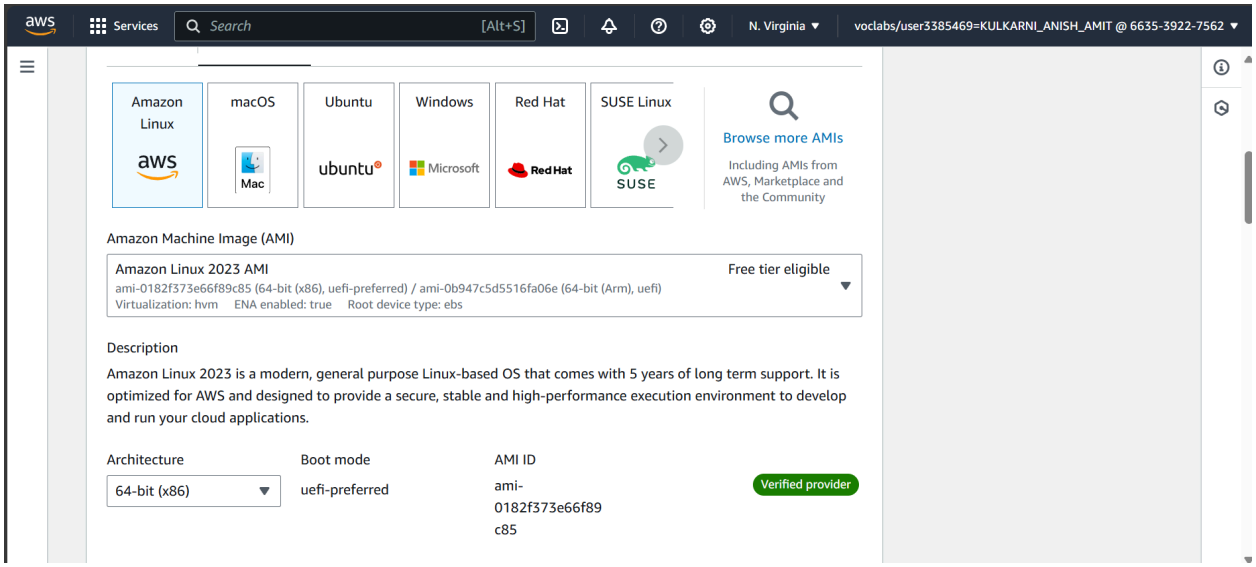
Experiment 4

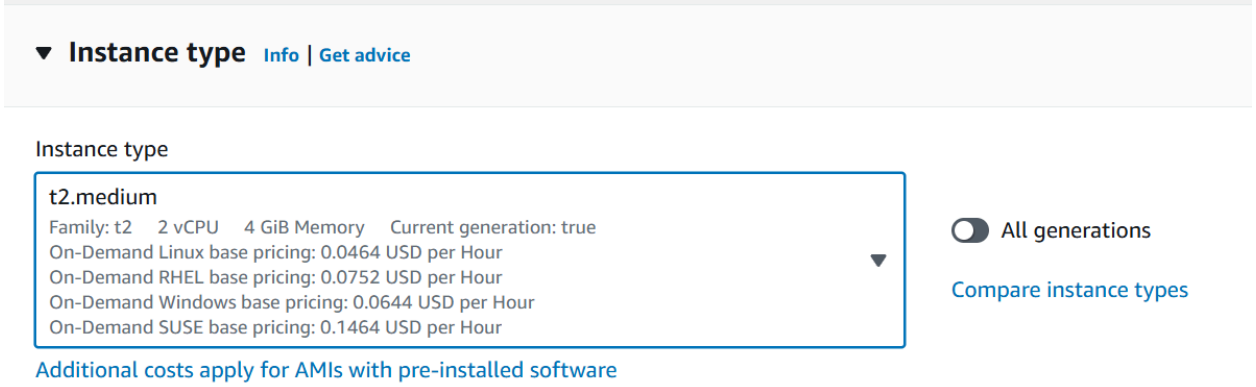
Aim: To install Kubectl and execute Kubectl commands to manage the Kubernetes cluster and deploy Your First Kubernetes Application.

Steps:

Step 1: Create an EC2 Amazon Linux instance on AWS. While doing so, make sure that t2.medium is selected as 'instance type' instead of the default t2.micro. This is because t2.medium provides more CPU, memory, and consistent performance, which are crucial for effectively running Kubernetes components and managing cluster workloads.

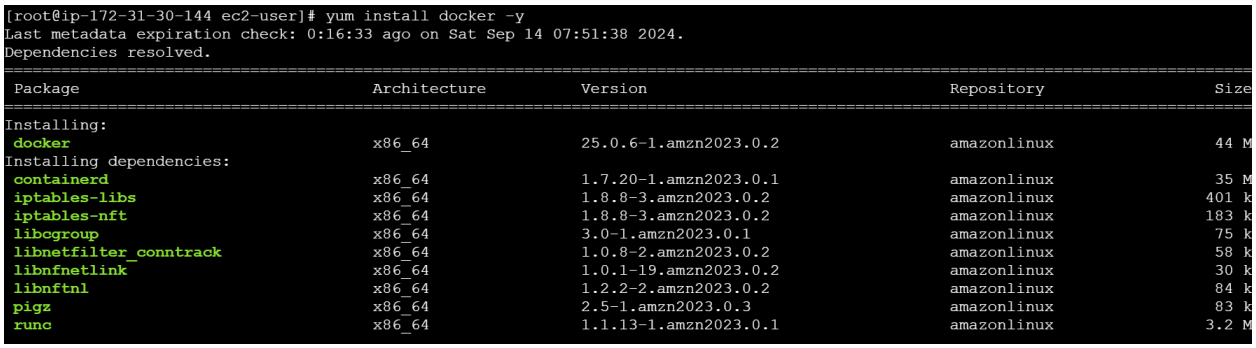






Step 2: To establish a connection with a remote server through SSH using the terminal, use the following command:-
ssh -i <keyname>.pem ubuntu@<public_ip_address>
where ‘keyname’ is the name of the key pair created by the user. “<keyname>.pem” is the name of the pem file of the key pair which is present in the ‘Downloads’ folder. The command “chmod 400 <keyname>.pem” is used to set the file permissions for the private key file (<keyname>.pem) so that only the file's owner can read it, and no one else can access or modify it.

Step 3: Install docker using the “yum install docker -y” command.

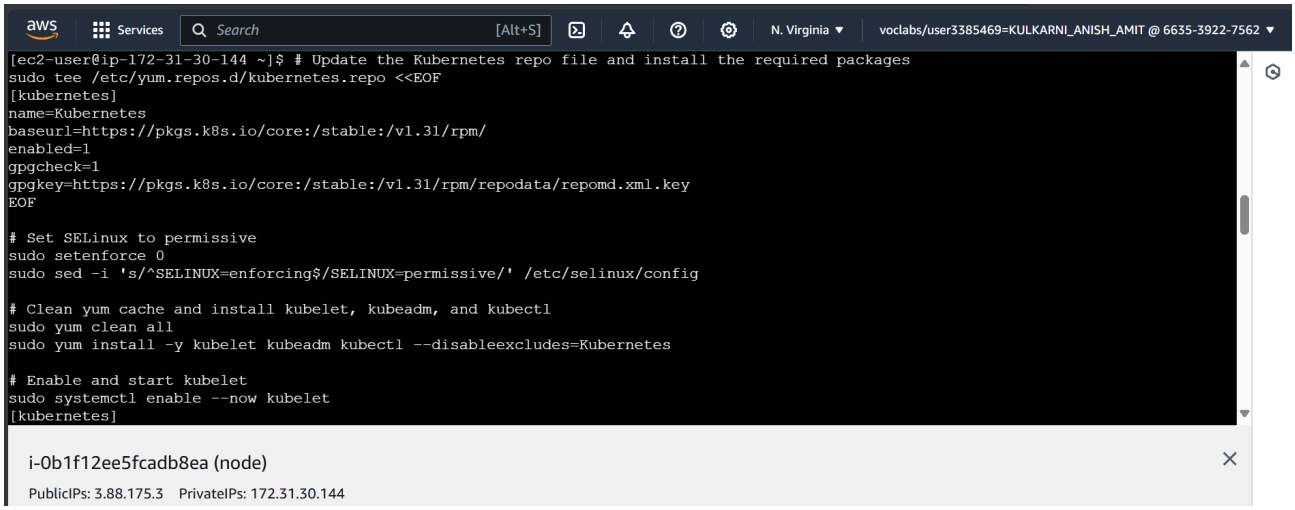


Step 4: Configure Docker to use systemd for managing cgroups by updating its configuration file, ensuring Docker starts automatically on boot, reloading the systemd configuration, and restarting Docker to apply the changes. Use the following commands to do so:

```
cd /etc/docker
cat <<EOF | sudo tee /etc/docker/daemon.json
{
"exec-opts": ["native.cgroupdriver=systemd"]
}
EOF
sudo systemctl enable docker
sudo systemctl daemon-reload
sudo systemctl restart docker
```

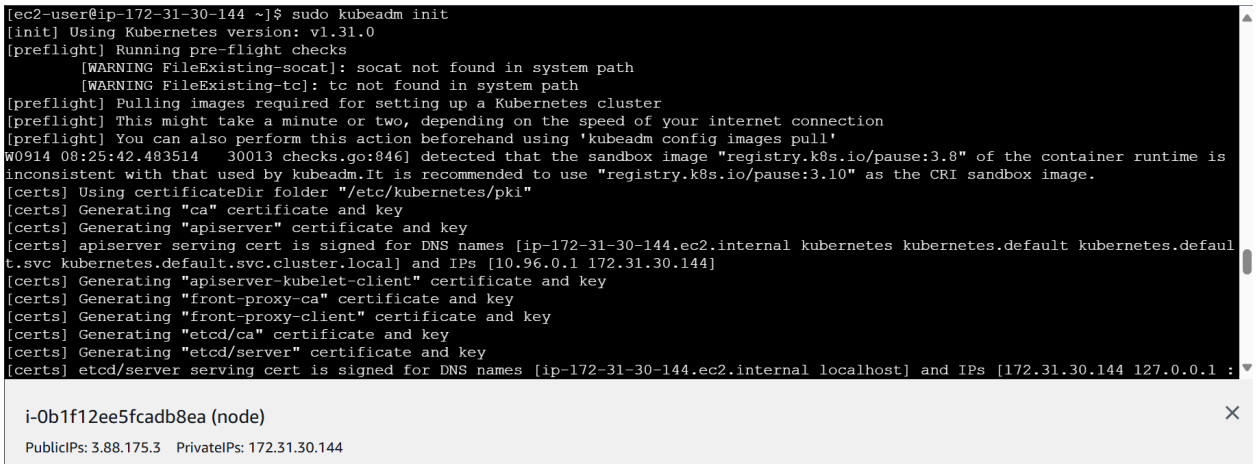
Step 5: Install kubernetes using the following commands:

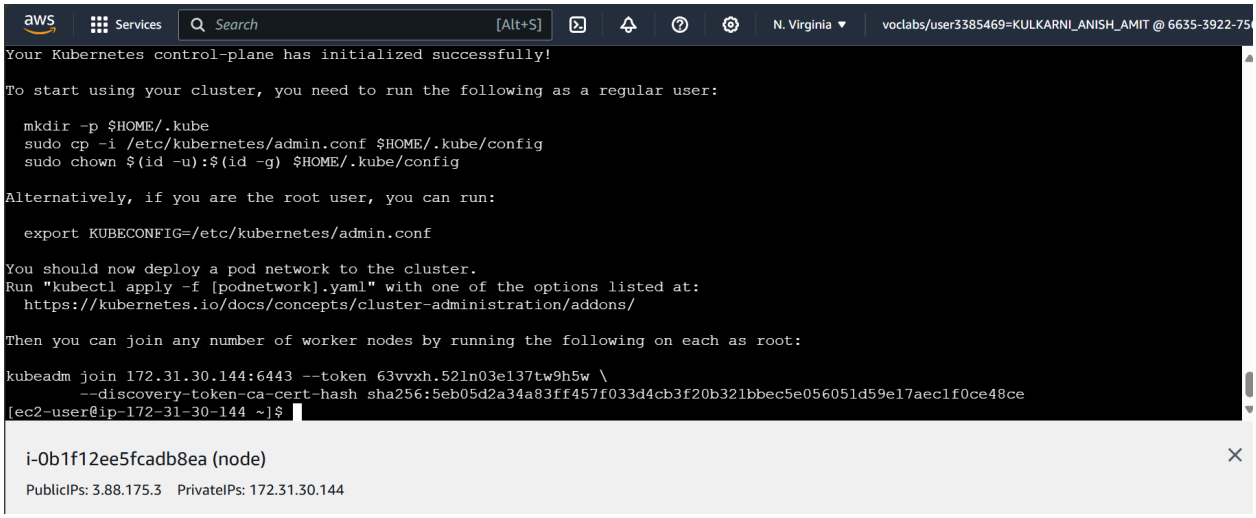
```
sudo tee /etc/yum.repos.d/kubernetes.repo <<EOF
[kubernetes]
name=Kubernetes
baseurl=https://pkgs.k8s.io/core:/stable:/v1.31/rpm/
enabled=1
gpgcheck=1
gpgkey=https://pkgs.k8s.io/core:/stable:/v1.31/rpm/repodata/repomd.xml.key
EOF

sudo setenforce 0
sudo sed -i 's/^SELINUX=enforcing$/SELINUX=permissive/' /etc/selinux/config

sudo yum clean all
sudo yum install -y kubelet kubeadm kubectl --disableexcludes=Kubernetes

sudo systemctl enable --now kubelet
```



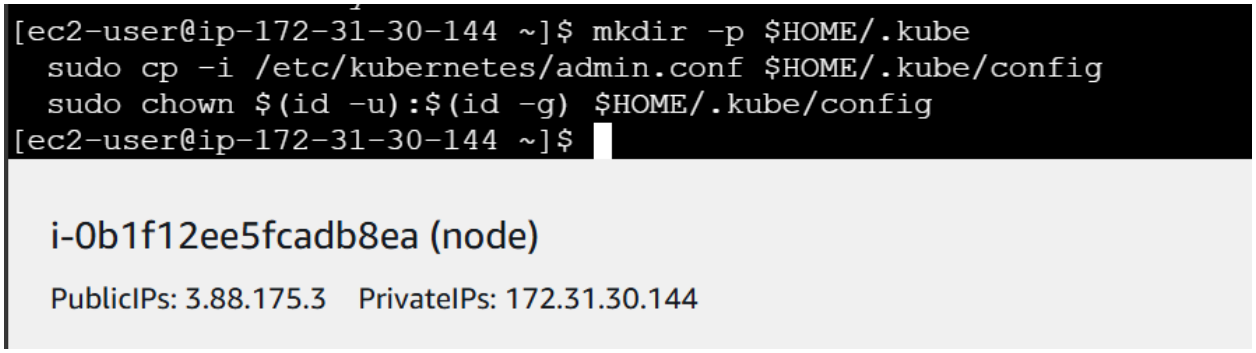

```
  Running scriptlet: kubectl-1.31.1-150500.1.1.x86_64                                                   9/9
  Verifying        : conntrack-tools-1.4.6-2.amzn2023.0.2.x86_64                                        1/9
  Verifying        : libnetfilter_cthelper-1.0.0-21.amzn2023.0.2.x86_64                                 2/9
  Verifying        : libnetfilter_cttimeout-1.0.0-19.amzn2023.0.2.x86_64                                3/9
  Verifying        : libnetfilter_queue-1.0.5-2.amzn2023.0.2.x86_64                                     4/9
  Verifying        : cri-tools-1.31.1-150500.1.1.x86_64                                                 5/9
  Verifying        : kubeadm-1.31.1-150500.1.1.x86_64                                                   6/9
  Verifying        : kubectl-1.31.1-150500.1.1.x86_64                                                   7/9
  Verifying        : kubelet-1.31.1-150500.1.1.x86_64                                                   8/9
  Verifying        : kubernetes-cni-1.5.1-150500.1.1.x86_64                                             9/9

Installed:
  conntrack-tools-1.4.6-2.amzn2023.0.2.x86_64                  cri-tools-1.31.1-150500.1.1.x86_64
  kubeadm-1.31.1-150500.1.1.x86_64                             kubectl-1.31.1-150500.1.1.x86_64
  kubelet-1.31.1-150500.1.1.x86_64                             kubernetes-cni-1.5.1-150500.1.1.x86_64
  libnetfilter_cthelper-1.0.0-21.amzn2023.0.2.x86_64           libnetfilter_cttimeout-1.0.0-19.amzn2023.0.2.x86_64
  libnetfilter_queue-1.0.5-2.amzn2023.0.2.x86_64

Complete!
Created symlink /etc/systemd/system/multi-user.target.wants/kubelet.service → /usr/lib/systemd/system/kubelet.service.
```

Step 6: Initialise the kubernetes cluster using the “sudo kubeadm init” command.

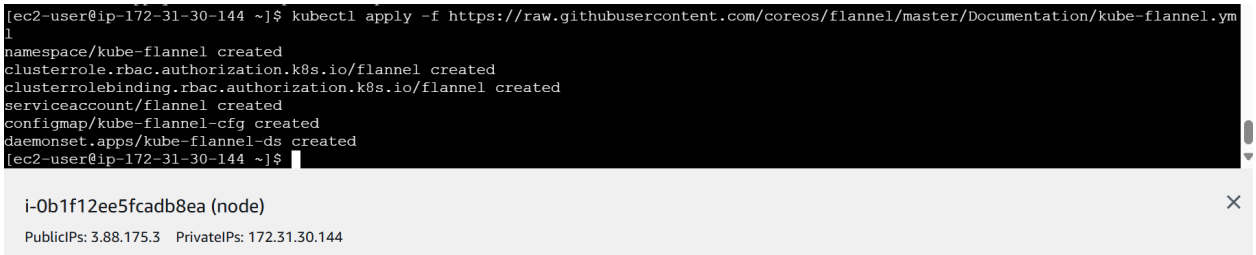




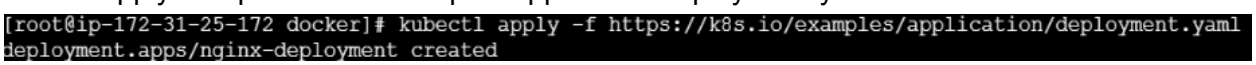
Step 7: Copy the mkdir and chown commands from the top and execute them.



Step 8: Deploy the Flannel networking plugin to the Kubernetes cluster using the following command:
kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/ k ube-flannel.yml



Step 9: Deploy the nginx server on the kubernetes cluster using the following command:
kubectl apply -f https://k8s.io/examples/application/deployment.yaml



Step 10: Execute the “kubectl get pods” command to verify if the deployment was properly created and the pod is working correctly.

```
[root@ip-172-31-25-172 docker]# kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
nginx-deployment-d556bf558-bqw22    0/1     Pending   0          66s
nginx-deployment-d556bf558-jxjxq    0/1     Pending   0          66s
```

Step 11: Here, it is observed that the status of the pods is “Pending”. To convert the status of the pods from “Pending” to “Running”, we must first execute the “kubectl describe pod nginx” command to get detailed information about the nginx pod such as its status, labels, annotations, containers, events, and resource usage.

```
[root@ip-172-31-23-234 docker]# kubectl describe pod nginx
Name:             nginx-deployment-77d8468669-7hwfr
Namespace:        default
Priority:         0
Service Account:  default
Node:             <none>
Labels:           app=nginx
                  pod-template-hash=77d8468669
Annotations:      <none>
Status:           Pending
IP:
  Warning  FailedScheduling  61s   default-scheduler  0/1 nodes are available: 1 node(s) had untolerated taint {node-role.kubernetes.io/control-plane: }. preemption
: 0/1 nodes are available: 1 Preemption is not helpful for scheduling.
```

Step 12: It is observed that the node has untolerated taints which prevents pods from being scheduled on nodes that have certain conditions or restrictions, which are specified by the taints on those nodes. To fix this, run the following command:
kubectl taint nodes --all node-role.kubernetes.io/control-plane:NoSchedule-

Step 13: Execute “kubectl get pods” command again to check if the status of the pods has been converted to “Running”.

Step 14: Forward port 8080 on your local machine to port 80 on the specified pod ($POD_NAME) using the following command:
kubectl port-forward $POD_NAME 8080:80

```
[root@ip-172-31-23-234 docker]# kubectl port-forward nginx-deployment-77d8468669-s77nc 8081:80
Forwarding from 127.0.0.1:8081 -> 80
Forwarding from [::1]:8081 -> 80
error: lost connection to pod
```

Here, it is observed that an error occurs. This is because there's a disruption or failure in the communication between the local machine and the Kubernetes pod during the kubectl port-forward operation.

Conclusion:

1. In this experiment, we learned how to install Kubectl and execute Kubectl commands to manage the Kubernetes cluster and deploy Your First Kubernetes Application.
2. First, we created an EC2 Amazon Linux instances on AWS and established its connections with a remote server through SSH.

3. Next, we installed and configured docker on all 3 machines.
4. Then, we installed and initialised kubernetes on the machine. We also added the machine to a kubernetes cluster.
5. Then, we deployed the Flannel networking plugin to the Kubernetes cluster using the “kubectl apply -f” command.
6. Then, we deployed the nginx server on the kubernetes cluster. This required the status of the pods to be changed from “Pending” to “Running” which in turn, required the untolerated taints from the pods to be removed.
7. Then, we tried to forward port 8080 on our local machine to port 80 on the specified pod where an error was encountered due to a disruption or failure in the communication between the local machine and the Kubernetes pod during the kubectl port-forward operation.