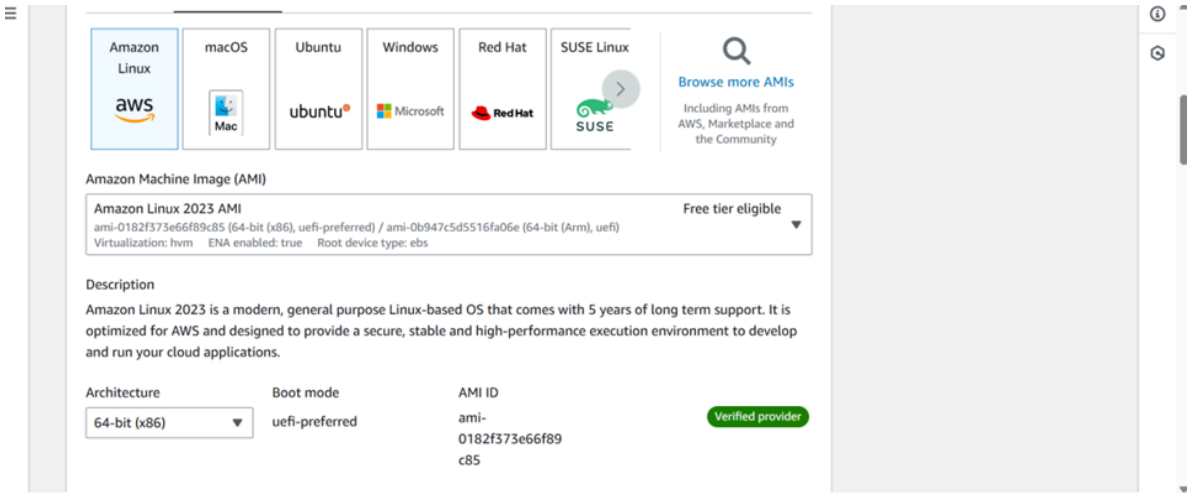
**Name: Shivam.R.Prajapati** **Div: D15C** **Roll No: 41**

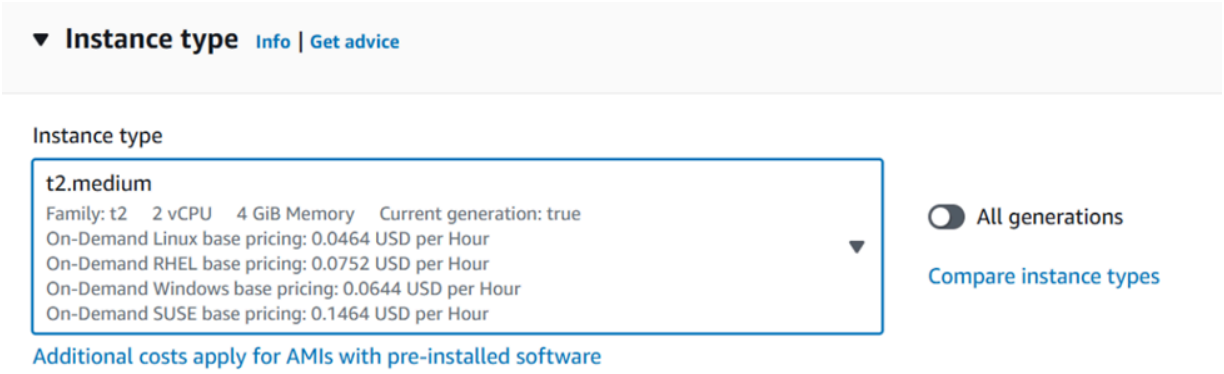
# Experiment No: 4

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

**STEPS:**

**Step1:** Launch an EC2 Amazon Linux instance on AWS, and choose the t2.medium instance type instead of the default t2.micro. The t2.medium type has more CPU and memory, which is important for running Kubernetes and managing your cluster effectively.



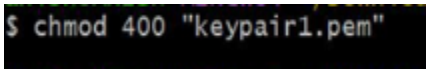

**Step 2:** To connect to a remote server via SSH in the terminal, use this command:

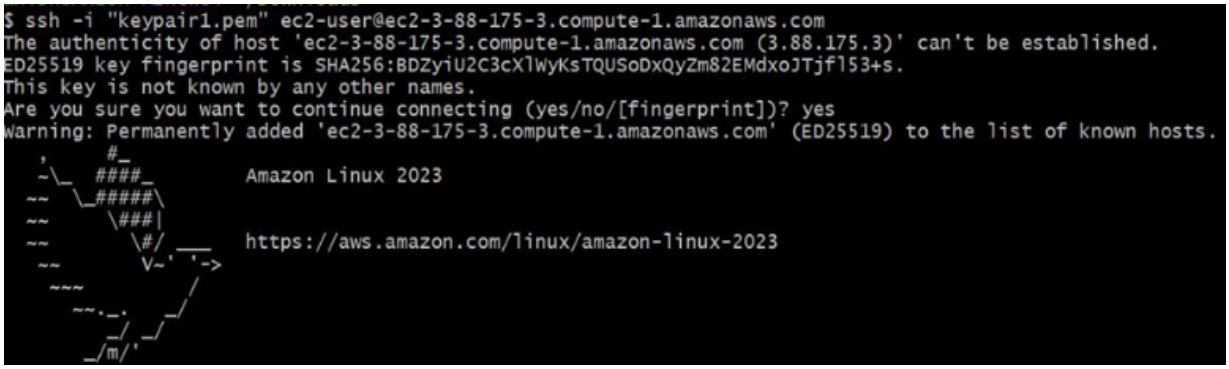
**ssh -i <keyname>.pem ubuntu@<public_ip_address>**

Here, '**keyname**' is the name of the key pair created by the user and **<keyname>.pem** is the file in your 'Downloads' folder. Before running this command, set the file permissions with:

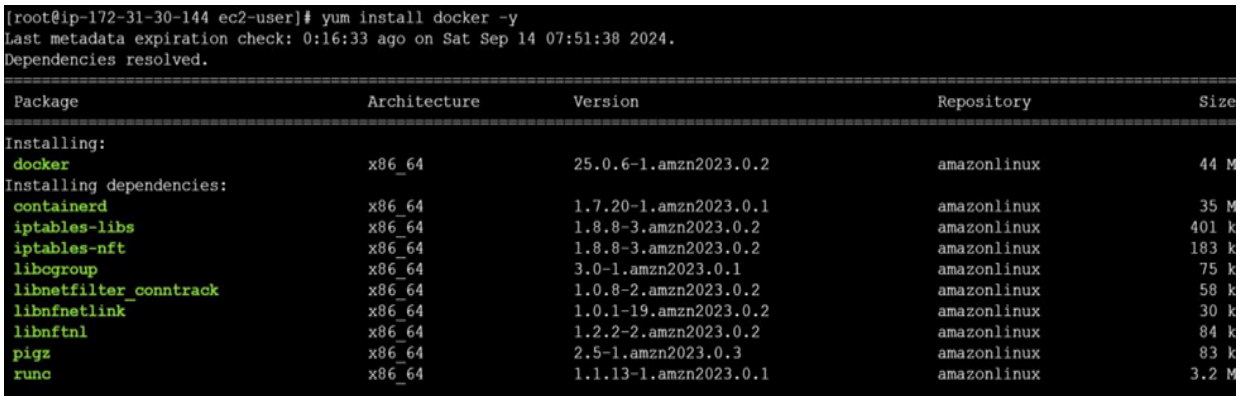
**chmod 400 <keyname>.pem**

This ensures that only you can read the key file and no one else can access or modify it. Proper file permissions are important for security, as they protect your private key from unauthorized access.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:** To configure Docker to use systemd for managing cgroups, first navigate to Docker's configuration directory with cd /etc/docker. Next, update the Docker configuration file (usually daemon.json) to include "exec-opts": ["native.cgroupdriver=systemd"]. This change ensures Docker uses systemd for managing cgroups, which improves resource management and integration with the system.

To ensure Docker starts automatically on boot, use the command **sudo systemctl enable docker.** After that, reload the systemd configuration with **sudo systemctl daemon-reload** to apply the changes. Finally, restart Docker using **sudo systemctl restart docker** to ensure the new settings take effect.

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

```
[ec2-user@ip-172-31-30-144 ~]$ # Update the Kubernetes repo file and install the required packages
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

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

# Clean yum cache and install kubelet, kubeadm, and kubectl
sudo yum clean all
sudo yum install -y kubelet kubeadm kubectl --disableexcludes=Kubernetes

# Enable and start kubelet
sudo systemctl enable --now kubelet
[kubernetes]
```

*Output is*

```
  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:** Initialize the kubernetes cluster using the **"sudo kubeadm init"** command.

```
[ec2-user@ip-172-31-30-144 ~]$ sudo kubeadm init
[init] Using Kubernetes version: v1.31.0
[preflight] Running pre-flight checks
        [WARNING FileExisting-socat]: socat not found in system path
        [WARNING FileExisting-tc]: tc not found in system path
[preflight] Pulling images required for setting up a Kubernetes cluster
[preflight] This might take a minute or two, depending on the speed of your internet connection
[preflight] You can also perform this action beforehand using 'kubeadm config images pull'
W0914 08:25:42.483514   30013 checks.go:846] detected that the sandbox image "registry.k8s.io/pause:3.8" of the container runtime is
inconsistent with that used by kubeadm.It is recommended to use "registry.k8s.io/pause:3.10" as the CRI sandbox image.
[certs] Using certificateDir folder "/etc/kubernetes/pki"
[certs] Generating "ca" certificate and key
[certs] Generating "apiserver" certificate and key
[certs] apiserver serving cert is signed for DNS names [ip-172-31-30-144.ec2.internal kubernetes kubernetes.default kubernetes.defaul
t.svc kubernetes.default.svc.cluster.local] and IPs [10.96.0.1 172.31.30.144]
[certs] Generating "apiserver-kubelet-client" certificate and key
[certs] Generating "front-proxy-ca" certificate and key
[certs] Generating "front-proxy-client" certificate and key
[certs] Generating "etcd/ca" certificate and key
[certs] Generating "etcd/server" certificate and key
[certs] etcd/server serving cert is signed for DNS names [ip-172-31-30-144.ec2.internal localhost] and IPs [172.31.30.144 127.0.0.1 :
```

```
Your Kubernetes control-plane has initialized successfully!

To start using your cluster, you need to run the following as a regular user:

  mkdir -p $HOME/.kube
  sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
  sudo chown $(id -u):$(id -g) $HOME/.kube/config

Alternatively, if you are the root user, you can run:

  export KUBECONFIG=/etc/kubernetes/admin.conf

You should now deploy a pod network to the cluster.
Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:
  https://kubernetes.io/docs/concepts/cluster-administration/addons/

Then you can join any number of worker nodes by running the following on each as root:

kubeadm join 172.31.30.144:6443 --token 63vvxh.52ln03e137tw9h5w \
        --discovery-token-ca-cert-hash sha256:5eb05d2a34a83ff457f033d4cb3f20b321bbec5e056051d59e17aec1f0ce48ce
[ec2-user@ip-172-31-30-144 ~]$
```

**Step 7:** To execute the mkdir and chown commands, first, use the mkdir command to create a new directory. For instance, you would type mkdir /path/to/directory in the terminal to set up the directory at the specified path. After creating the directory, use the

chown command to change its ownership. For example, you would execute sudo chown user:group /path/to/directory to set the ownership to the specified user and group. This ensures that the directory has the correct permissions and ownership for proper access and management.

*Copy the mkdir and chown commands from the top and execute them.*

```
[ec2-user@ip-172-31-30-144 ~]$ mkdir -p $HOME/.kube
  sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
  sudo chown $(id -u):$(id -g) $HOME/.kube/config
[ec2-user@ip-172-31-30-144 ~]$
```

**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/kube-flannel.yml**

This command downloads and applies the Flannel configuration from the provided URL, setting up Flannel as the networking solution for your cluster. This step is crucial for ensuring that the Kubernetes cluster can manage network communication between pods efficiently.


**Step 9:** Deploy the nginx server on the kubernetes cluster using the following command:

**kubectl apply -f https://k8s.io/examples/application/deployment.yaml**

```
[root@ip-172-31-25-172 docker]# kubectl apply -f https://k8s.io/examples/application/deployment.yaml
deployment.apps/nginx-deployment created
```

This command creates a new deployment named nginx and uses the official NGINX image from Docker Hub. This deployment will ensure that the NGINX server is running in your cluster and can be managed by Kubernetes.

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

This command lists all the pods in your Kubernetes cluster. Look for a pod with a name that includes nginx (or the name you specified during deployment). The output will show the status of the pods, including whether they are running, pending, or have encountered any issues.

```
[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 (mechanism) which prevents pods from being scheduled on nodes(instances) 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-**

This command removes the node-role.kubernetes.io/control-plane:NoSchedule taint from all nodes in your cluster. By removing this taint, you allow pods to be scheduled on nodes that previously had this restriction, potentially resolving the scheduling issue and enabling your pods to transition from "Pending" to "Running."

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

**Step 14:** To forward port 8080 on your local machine to port 80 on a specified pod, use the following command: **kubectl port-forward $POD_NAME 8080:80.**

This command sets up a tunnel between your local machine and the Kubernetes pod. By running this command, any traffic sent to port 8080 on your local machine is forwarded to port 80 on the pod.

```
[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 occurred . An error occurred here because the **kubectl port-forward** command is **failing to establish a communication between your local machine and the Kubernetes pod.**

## CONCLUSION:

In this experiment, we learned how to install `kubectl` and use it to manage a Kubernetes cluster and deploy our first Kubernetes application. We began by creating EC2 Amazon Linux instances on AWS and established connections to these instances via SSH. Next, we installed and configured Docker on all three machines. After that, we installed and initialized Kubernetes on one of the machines and added it to a Kubernetes cluster. We then deployed the Flannel networking plugin to the cluster using the `kubectl apply -f` command. Following this, we deployed the NGINX server to the Kubernetes cluster. To ensure the NGINX pods transitioned from "Pending" to "Running," we had to remove untolerated taints from the nodes. Finally, we attempted to forward port 8080 on our local machine to port 80 on the specified pod. However, we encountered an error due to a disruption or failure in the communication between the local machine and the Kubernetes pod during the `kubectl port-forward` operation.