

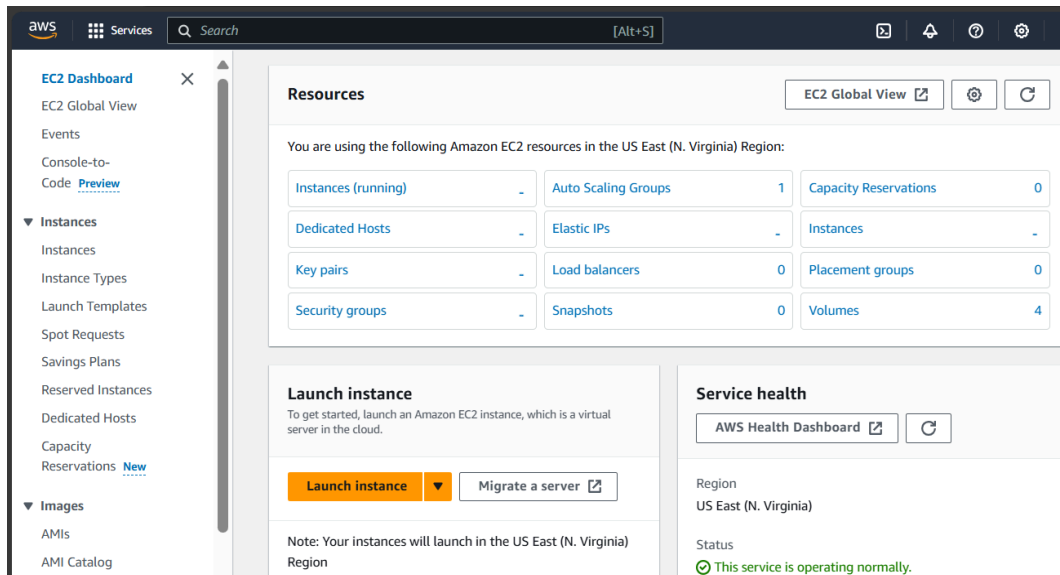
Experiment 4

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

Procedure:

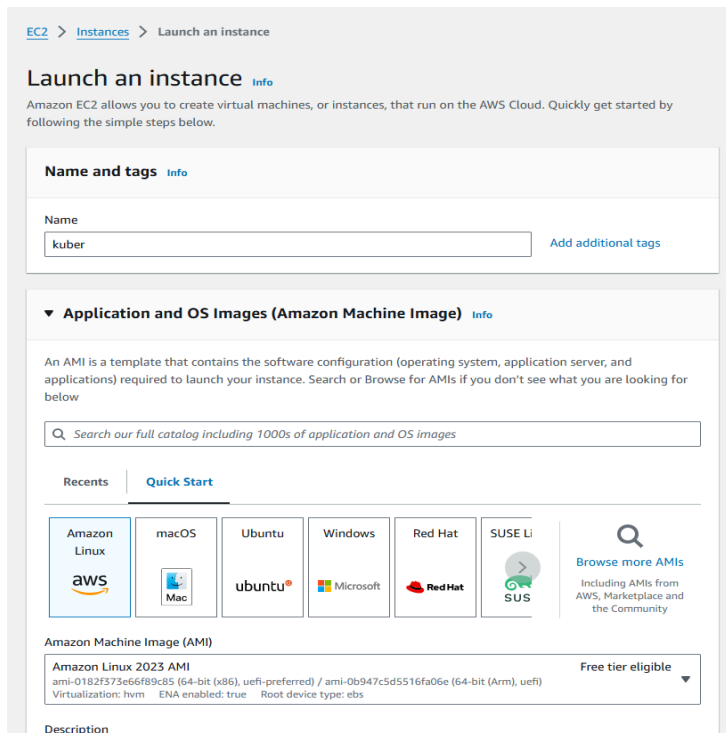
1. Creation Of EC-2 instance

- Create an EC2 AWS Linux instance on AWS .also edit the Security Group Inbound Rules to allow SSH. then select the t2.micro instance type



The screenshot shows the AWS Management Console's EC2 Dashboard. The left sidebar contains navigation links for EC2 Dashboard, EC2 Global View, Events, Console-to-Code, and a list of resources including Instances, Instance Types, Launch Templates, Spot Requests, Savings Plans, Reserved Instances, Dedicated Hosts, Capacity Reservations, and Images. The main content area is titled 'Resources' and shows a summary of EC2 resources in the US East (N. Virginia) Region. Below this, there's a 'Launch instance' section with a 'Launch instance' button and a 'Migrate a server' button. To the right, the 'Service health' section shows the AWS Health Dashboard and a status message: 'This service is operating normally.'

Resources		
You are using the following Amazon EC2 resources in the US East (N. Virginia) Region:		
Instances (running)	-	
Dedicated Hosts	-	
Key pairs	-	
Security groups	-	
Auto Scaling Groups	1	
Elastic IPs	-	
Load balancers	0	
Snapshots	0	
Capacity Reservations	0	
Instances	-	
Placement groups	0	
Volumes	4	



The screenshot shows the 'Launch an instance' wizard in the AWS Management Console. The page is titled 'Launch an instance' and provides instructions on how to create virtual machines. The 'Name and tags' section has a text input field for 'Name' with the value 'kuber' and a link to 'Add additional tags'. The 'Application and OS Images (Amazon Machine Image)' section shows a search bar and a 'Quick Start' tab. Below the search bar, there are several AMI cards for different operating systems: Amazon Linux, macOS, Ubuntu, Windows, Red Hat, and SUSE Linux. The 'Amazon Linux 2023 AMI' card is selected, showing details like 'ami-0182f373e66f89c85 (64-bit (x86), uefi-preferred)' and 'Free tier eligible'. The 'Description' section is partially visible at the bottom.

Description

Amazon Linux 2023 is a modern, general purpose Linux-based OS that comes with 5 years of long term support. It is optimized for AWS and designed to provide a secure, stable and high-performance execution environment to develop and run your cloud applications.

Architecture: 64-bit (x86) | Boot mode: uefi-preferred | AMI ID: ami-0182f373e66f89c85

Verified provider

▼ Instance type [Info](#) | [Get advice](#)

Instance type: **t2.medium**

Family: t2 | 2 vCPU | 4 GiB Memory | Current generation: true

On-Demand Linux base pricing: 0.0464 USD per Hour

On-Demand RHEL base pricing: 0.0752 USD per Hour

On-Demand Windows base pricing: 0.0644 USD per Hour

On-Demand SUSE base pricing: 0.1464 USD per Hour

☐ All generations

[Compare instance types](#)

Additional costs apply for AMIs with pre-installed software

▼ Key pair (login) [Info](#)

You can use a key pair to securely connect to your instance. Ensure that you have access to the selected key pair before you launch the instance.

Key pair name - *required*

server

[Create new key pair](#)

▼ Network settings [Info](#) [Edit](#)

Network: [Info](#)

vpc-051bba342b3626898

Subnet: [Info](#)

No preference (Default subnet in any availability zone)

Auto-assign public IP: [Info](#)

Enable

Additional charges apply when outside of [free tier allowance](#)

Firewall (security groups): [Info](#)

A security group is a set of firewall rules that control the traffic for your instance. Add rules to allow specific traffic to reach your instance.

☒ Create security group ☐ Select existing security group

We'll create a new security group called 'launch-wizard-35' with the following rules:

☒ Allow SSH traffic from **Anywhere** (0.0.0.0/0)

☐ Allow HTTPS traffic from the internet

☐ Allow HTTP traffic from the internet

Rules with source of 0.0.0.0/0 allow all IP addresses to access your instance. We recommend setting security group rules to allow access from known IP addresses only.

▼ Summary

Number of instances: [Info](#)

1

Software Image (AMI)

Amazon Linux 2023 AMI 2023.5.2...[read more](#)

ami-0182f373e66f89c85

Virtual server type (instance type)

t2.medium

Firewall (security group)

New security group

Storage (volumes)

1 volume(s) - 8 GiB

Free tier: In your first year includes 750 hours of t2.micro (or t3.micro in the Regions in which t2.micro is unavailable) instance usage on free tier AMIs per month, 750 hours of public IPv4 address usage per month, 30 GiB of EBS storage, 2 million I/Os, 1 GB of snapshots, and 100 GB of bandwidth to the internet.

[Cancel](#) [Launch instance](#) [Review commands](#)

Instances (1) [Info](#)

Find Instance by attribute or tag (case-sensitive) | All states

Instance ID: i-09dbca91fa3edcaea | Clear filters

	Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4 ...	Elastic IP
<input type="checkbox"/>	kuber	i-09dbca91fa3edcaea	Running	t2.micro	2/2 checks passed	View alarms	us-east-1a	ec2-54-211-131-109.co...	54.211.131.109	-


- Thus Kuber named -instance gets created. Then click on Id of that instance then click on connect button you will see this:

EC2 > Instances > i-09dbca91fa3edcaea > Connect to instance

Connect to instance [info](#)

Connect to your instance i-09dbca91fa3edcaea (kuber) using any of these options

[EC2 Instance Connect](#)
[Session Manager](#)
[SSH client](#)
[EC2 serial console](#)


Port 22 (SSH) is open to all IPv4 addresses
 Port 22 (SSH) is currently open to all IPv4 addresses, indicated by 0.0.0.0/0 in the inbound rule in [your security group](#). For increased security, consider restricting access to only the EC2 Instance Connect service IP addresses for your Region: 18.206.107.24/29. [Learn more](#).

Instance ID
i-09dbca91fa3edcaea (kuber)

Connection Type


☒ **Connect using EC2 Instance Connect**
 Connect using the EC2 Instance Connect browser-based client, with a public IPv4 address.

☐ **Connect using EC2 Instance Connect Endpoint**
 Connect using the EC2 Instance Connect browser-based client, with a private IPv4 address and a VPC endpoint.

Public IPv4 address
54.211.131.109

Username
Enter the username defined in the AMI used to launch the instance. If you didn't define a custom username, use the default username, ec2-user.

ec2-user


Note: In most cases, the default username, ec2-user, is correct. However, read your AMI usage instructions to check if the AMI owner has changed the default AMI username.

Cancel **Connect**

- Then go into SSH client where you will get this command
Chmod 400 "keyname.pem"
ssh -i <keyname>.pem ubuntu@<public_ip_address> copy it and then connect it and run the following command for establishing connection.(I have entered this command on git bash where i entered in downloads where server.pem is stored then as the key is not accessible hence we need to change its mode using chmod 400 "key name.pem". Then use the given command for making connections).

```
Sadneya@DESKTOP-IEPNL3D MINGW64 ~ (master)
$ cd downloads

Sadneya@DESKTOP-IEPNL3D MINGW64 ~/downloads (master)
$ chmod 400 "server.pem"

Sadneya@DESKTOP-IEPNL3D MINGW64 ~/downloads (master)
$ ssh -i "server.pem" ec2-user@ec2-54-196-176-21.compute-1.amazonaws.com
The authenticity of host 'ec2-54-196-176-21.compute-1.amazonaws.com (54.196.176.21)' can't be established.
ED25519 key fingerprint is SHA256:1E4CbWC3A0Kdn1J+99jTmUzur4joKQmThQyRcwIJzUU.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'ec2-54-196-176-21.compute-1.amazonaws.com' (ED25519)
to the list of known hosts.

#_
~\##### Amazon Linux 2023
~~\#####
~~\###|
~~\#/
~~V~'-'>
~~~
~~~.
~~~/_/
~~~/_/m/'
```

<https://aws.amazon.com/linux/amazon-linux-2023>

2. Installation of Docker

1. . For installation of Docker into the machines run the following command:
`sudo yum install docker -y`

```
[ec2-user@ip-172-31-26-174 ~]$ sudo yum install docker -y
Last metadata expiration check: 0:05:13 ago on Fri Sep 13 13:17:25 2024.
Dependencies resolved.
=====
Package                                Architecture      Version            Repository
=====
Installing:
docker                                x86_64            25.0.6-1.amzn2023.0.2  amazonlinux
Installing dependencies:
containerd                            x86_64            1.7.20-1.amzn2023.0.1  amazonlinux
iptables-libs                         x86_64            1.8.8-3.amzn2023.0.2  amazonlinux
iptables-nft                         x86_64            1.8.8-3.amzn2023.0.2  amazonlinux
libcgroup                             x86_64            3.0-1.amzn2023.0.1    amazonlinux
libnetfilter_conntrack               x86_64            1.0.8-2.amzn2023.0.2  amazonlinux
libnftnl                             x86_64            1.0.1-19.amzn2023.0.2  amazonlinux
libnftnl                             x86_64            1.2.2-2.amzn2023.0.2  amazonlinux
pigz                                  x86_64            2.5-1.amzn2023.0.3    amazonlinux
runc                                  x86_64            1.1.13-1.amzn2023.0.1  amazonlinux
=====
Transaction Summary
=====
Total                                                                    78 MB/s | 84 MB    00:01
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
  Preparing :
  Installing : runc-1.1.13-1.amzn2023.0.1.x86_64
  Installing : containerd-1.7.20-1.amzn2023.0.1.x86_64
  Running scriptlet: containerd-1.7.20-1.amzn2023.0.1.x86_64
  Installing : pigz-2.5-1.amzn2023.0.3.x86_64
  Installing : libnftnl-1.2.2-2.amzn2023.0.2.x86_64
  Installing : libnftnl-1.2.2-2.amzn2023.0.2.x86_64
  Installing : libnftnl-1.0.1-19.amzn2023.0.2.x86_64
  Installing : libnetfilter_conntrack-1.0.8-2.amzn2023.0.2.x86_64
  Installing : iptables-libs-1.8.8-3.amzn2023.0.2.x86_64
  Installing : iptables-nft-1.8.8-3.amzn2023.0.2.x86_64
  Running scriptlet: iptables-nft-1.8.8-3.amzn2023.0.2.x86_64
  Installing : libcgroup-3.0-1.amzn2023.0.1.x86_64
  Running scriptlet: docker-25.0.6-1.amzn2023.0.2.x86_64
  Installing : docker-25.0.6-1.amzn2023.0.2.x86_64
  Running scriptlet: docker-25.0.6-1.amzn2023.0.2.x86_64
Created symlink /etc/systemd/system/sockets.target.wants/docker.socket → /usr/lib/systemd/system/docker.socket.

  Verifying : containerd-1.7.20-1.amzn2023.0.1.x86_64
  Verifying : docker-25.0.6-1.amzn2023.0.2.x86_64
  Verifying : iptables-libs-1.8.8-3.amzn2023.0.2.x86_64
  Verifying : iptables-nft-1.8.8-3.amzn2023.0.2.x86_64
  Verifying : libcgroup-3.0-1.amzn2023.0.1.x86_64
  Verifying : libnetfilter_conntrack-1.0.8-2.amzn2023.0.2.x86_64
  Verifying : libnftnl-1.0.1-19.amzn2023.0.2.x86_64
  Verifying : libnftnl-1.2.2-2.amzn2023.0.2.x86_64
  Verifying : pigz-2.5-1.amzn2023.0.3.x86_64
  Verifying : runc-1.1.13-1.amzn2023.0.1.x86_64

Installed:
containerd-1.7.20-1.amzn2023.0.1.x86_64    docker-25.0.6-1.amzn2023.0.2.x86_64    iptables-libs-1.8.8-3.amzn2023.0.2.x86_64
iptables-nft-1.8.8-3.amzn2023.0.2.x86_64    libcgroup-3.0-1.amzn2023.0.1.x86_64    libnetfilter_conntrack-1.0.8-2.amzn2023.0.2.x86_64
libnftnl-1.0.1-19.amzn2023.0.2.x86_64    libnftnl-1.2.2-2.amzn2023.0.2.x86_64    pigz-2.5-1.amzn2023.0.3.x86_64
runc-1.1.13-1.amzn2023.0.1.x86_64
Complete!
```

- Then, configure cgroup in a daemon.json file by using following commands

```
cd /etc/docker
cat <<EOF | sudo tee /etc/docker/daemon.json
{
  "exec-opts": ["native.cgroupdriver=systemd"],
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "100m"
  },
  "storage-driver": "overlay2"
}
EOF
```

```
[ec2-user@ip-172-31-26-174 ~]$ cd /etc/docker
[ec2-user@ip-172-31-26-174 docker]$ cat <<EOF | sudo tee /etc/docker/daemon.json
{
  "exec-opts": ["native.cgroupdriver=systemd"],
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "100m"
  },
  "storage-driver": "overlay2"
}
EOF
{
  "exec-opts": ["native.cgroupdriver=systemd"],
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "100m"
  },
  "storage-driver": "overlay2"
}
```

- Then after this run the following command to enable and start docker and also to load the daemon.json file.

```
sudo systemctl enable docker
sudo systemctl daemon-reload
sudo systemctl restart docker
```

```
[ec2-user@ip-172-31-26-174 docker]$ sudo systemctl enable docker
Created symlink /etc/systemd/system/multi-user.target.wants/docker.service → /usr/lib/systemd/system/docker.service.
[ec2-user@ip-172-31-26-174 docker]$ sudo systemctl daemon-reload
[ec2-user@ip-172-31-26-174 docker]$ sudo systemctl restart docker
[ec2-user@ip-172-31-26-174 docker]$ docker -v
Docker version 25.0.5, build 5dc9bcc
```

- docker -v

```
[ec2-user@ip-172-31-80-126 docker]$ docker -v
Docker version 25.0.5, build 5dc9bcc
```

3. Then Install Kubernetes with the following command.

- SELinux needs to be disabled before configuring kubelet thus run the following command
sudo setenforce 0

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

```
[ec2-user@ip-172-31-26-174 docker]$ sudo setenforce 0
[ec2-user@ip-172-31-26-174 docker]$ sudo sed -i 's/^SELINUX=enforcing$/SELINUX=permissive/' /etc/selinux/config
```

- Here We are adding kubernetes using the repository whose command is given below.

```
cat <<EOF | sudo tee /etc/yum.repos.d/kubernetes.repo
[kubernetes]
name=Kubernetes
baseurl=https://pkgs.k8s.io/core:/stable:/v1.30/rpm/
enabled=1
gpgcheck=1
gpgkey=https://pkgs.k8s.io/core:/stable:/v1.30/rpm/repodata/repomd.xml.key
exclude=kubelet kubeadm kubectl cri-tools kubernetes-cni
EOF
```

```
[ec2-user@ip-172-31-26-174 docker]$ sudo setenforce 0
sudo sed -i 's/^SELINUX=enforcing$/SELINUX=permissive/' /etc/selinux/config
[ec2-user@ip-172-31-26-174 docker]$ cat <<EOF | sudo tee /etc/yum.repos.d/kubernetes.repo
[kubernetes]
name=Kubernetes
baseurl=https://pkgs.k8s.io/core:/stable:/v1.30/rpm/
enabled=1
gpgcheck=1
gpgkey=https://pkgs.k8s.io/core:/stable:/v1.30/rpm/repodata/repomd.xml.key
exclude=kubelet kubeadm kubectl cri-tools kubernetes-cni
EOF
[kubernetes]
name=Kubernetes
baseurl=https://pkgs.k8s.io/core:/stable:/v1.30/rpm/
enabled=1
gpgcheck=1
gpgkey=https://pkgs.k8s.io/core:/stable:/v1.30/rpm/repodata/repomd.xml.key
exclude=kubelet kubeadm kubectl cri-tools kubernetes-cni
```

- After that Run following command to make the updation and also to install kubelet ,kubeadm, kubectl:

```
sudo yum update
```

```
[ec2-user@ip-172-31-80-126 docker]$ sudo yum update
Kubernetes
Dependencies resolved.
Nothing to do.
Complete!
```

```
sudo yum install -y kubelet kubeadm kubectl --disableexcludes=kubernetes
```

```
[ec2-user@ip-172-31-80-126 docker]$ sudo yum install -y kubelet kubeadm kubectl --disableexcludes=kubernetes
Last metadata expiration check: 0:00:10 ago on Fri Sep 13 10:31:17 2024.
Dependencies resolved.
```

Package	Architecture	Version	Repository	Size
Installing:				
kubeadm	x86_64	1.30.5-150500.1.1	kubernetes	10 M
kubectl	x86_64	1.30.5-150500.1.1	kubernetes	17 M
kubelet	x86_64	1.30.5-150500.1.1	kubernetes	17 M
Installing dependencies:				
conntrack-tools	x86_64	1.4.6-2.amzn2023.0.2	amazonlinux	208 k
cri-tools	x86_64	1.30.1-150500.1.1	kubernetes	8.6 M
kubernetes-cni	x86_64	1.4.0-150500.1.1	kubernetes	6.7 M
libnetfilter_cthelper	x86_64	1.0.0-21.amzn2023.0.2	amazonlinux	24 k
libnetfilter_cttimeout	x86_64	1.0.0-19.amzn2023.0.2	amazonlinux	24 k
libnetfilter_queue	x86_64	1.0.5-2.amzn2023.0.2	amazonlinux	30 k
Transaction Summary				
Install	9 Packages			

```
Total
Kubernetes
Importing GPG key 0x9A296436:
  Userid : "isv:kubernetes O&S Project <isv:kubernetes@build.opensuse.org>"
  Fingerprint: DE15 B144 86CD 377B 9E87 6E1A 2346 54DA 9A29 6436
  From : https://pkgs.k8s.io/core:/stable:/v1.30/rpm/repodata/repomd.xml.key
Key imported successfully
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
  Preparing :
  Installing : kubernetes-cni-1.4.0-150500.1.1.x86_64 1/1
  Installing : cri-tools-1.30.1-150500.1.1.x86_64 1/9
  Installing : libnetfilter_queue-1.0.5-2.amzn2023.0.2.x86_64 2/9
  Installing : libnetfilter_cttimeout-1.0.0-19.amzn2023.0.2.x86_64 3/9
  Installing : libnetfilter_cthelper-1.0.0-21.amzn2023.0.2.x86_64 4/9
  Installing : libnetfilter_queue-1.0.5-2.amzn2023.0.2.x86_64 5/9
  Installing : conntrack-tools-1.4.6-2.amzn2023.0.2.x86_64 6/9
  Running scriptlet: conntrack-tools-1.4.6-2.amzn2023.0.2.x86_64 6/9
  Installing : kubelet-1.30.5-150500.1.1.x86_64 7/9
  Running scriptlet: kubelet-1.30.5-150500.1.1.x86_64 7/9
  Installing : kubeadm-1.30.5-150500.1.1.x86_64 8/9
  Installing : kubectl-1.30.5-150500.1.1.x86_64 9/9
  Running scriptlet: kubectl-1.30.5-150500.1.1.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-1.30.1-150500.1.1.x86_64 5/9
  Verifying : kubeadm-1.30.5-150500.1.1.x86_64 6/9
  Verifying : kubectl-1.30.5-150500.1.1.x86_64 7/9
  Verifying : kubelet-1.30.5-150500.1.1.x86_64 8/9
  Verifying : kubernetes-cni-1.4.0-150500.1.1.x86_64 9/9
Installed:
  conntrack-tools-1.4.6-2.amzn2023.0.2.x86_64
  kubelet-1.30.5-150500.1.1.x86_64
  libnetfilter_cthelper-1.0.0-21.amzn2023.0.2.x86_64
  cri-tools-1.30.1-150500.1.1.x86_64
  kubelet-1.30.5-150500.1.1.x86_64
  kubeadm-1.30.5-150500.1.1.x86_64
  kubernetes-cni-1.4.0-150500.1.1.x86_64
  libnetfilter_queue-1.0.5-2.amzn2023.0.2.x86_64
Complete!
```

- After installing Kubernetes, we need to configure internet options to allow bridging.

1. `sudo swapoff -a`
2. `echo "net.bridge.bridge-nf-call-iptables=1" | sudo tee -a /etc/sysctl.conf`
3. `sudo sysctl -p`

```
[ec2-user@ip-172-31-26-174 docker]$ sudo swapoff -a
echo "net.bridge.bridge-nf-call-iptables=1" | sudo tee -a /etc/sysctl.conf
sudo sysctl -p
net.bridge.bridge-nf-call-iptables=1
net.bridge.bridge-nf-call-iptables = 1
```

4. Initialize the Kubecluster

```
sudo kubeadm init --pod-network-cidr=10.244.0.0/16
```

```
[ec2-user@ip-172-31-80-126 docker]$ sudo kubeadm init --pod-network-cidr=10.244.0.0/16
I0913 10:32:44.629146 26680 version.go:256] remote version is much newer: v1.31.0; falling back to: stable-1.30
[init] Using Kubernetes version: v1.30.4
[preflight] Running pre-flight checks
```

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 172.31.26.174:6443 --token pv0yyi.xhllqhclfjr50pt8 \
--discovery-token-ca-cert-hash sha256:8293b2f6d29de466bd859007f5adbcd3a
ecb0c446ba09033d32a5846b3d434f
```

- copy the token and save for future use .
`kubeadm join 172.31.26.174:6443 --token pv0yyi.xhllqhclfjr50pt8 \`
`--discovery-token-ca-cert-hash`
`sha256:8293b2f6d29de466bd859007f5adbcd3a`
`ecb0c446ba09033d32a5846b3d434f`

- Copy the mkdir and chown commands from the top and execute them

```
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-172-31-80-126 docker]$ mkdir -p $HOME/.kube
[ec2-user@ip-172-31-80-126 docker]$ sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
[ec2-user@ip-172-31-80-126 docker]$ sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

- Then, add a common networking plugin called flannel as mentioned in the code.

kubectl apply -f

<https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml>

```
[ec2-user@ip-172-31-26-174 docker]$ kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml
namespace/kube-flannel created
clusterrole.rbac.authorization.k8s.io/flannel created
clusterrolebinding.rbac.authorization.k8s.io/flannel created
serviceaccount/flannel created
configmap/kube-flannel-cfg created
daemonset.apps/kube-flannel-ds created
```

5. Now that the cluster is up and running, we can deploy our nginx server on this cluster. Apply deployment using this following command:

kubectl apply -f <https://k8s.io/examples/pods/simple-pod.yaml>

```
[ec2-user@ip-172-31-26-174 docker]$ kubectl apply -f https://k8s.io/examples/pods/simple-pod.yaml
pod/nginx created
```

Then use **kubectl get pods** to check whether the pod gets created or not.

```
[ec2-user@ip-172-31-26-174 docker]$ kubectl get pods
NAME      READY   STATUS    RESTARTS   AGE
nginx     0/1     Pending   0           12s
```

To convert state from pending to running use following command:

kubectl describe pod nginx This command will help to describe the pods it gives reason for failure as it shows the untolerated taints which need to be untainted.

- kubectl describe pod nginx

```
[ec2-user@ip-172-31-26-174 docker]$ kubectl describe pod nginx
Name:      nginx
Namespace: default
Priority:   0
Service Account: default
Node:      <none>
Labels:    <none>
Annotations: <none>
Status:    Pending
IP:        <none>
IPs:       <none>
Containers:
  nginx:
    Image:      nginx:1.14.2
    Port:       80/TCP
    Host Port:  0/TCP
    Environment: <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-k4lj6 (ro)
```



```

Conditions:
  Type          Status
PodScheduled    False
Volumes:
  kube-api-access-k4lj6:
    Type:          Projected (a volume that contains injected data from m
multiple sources)
    TokenExpirationSeconds: 3607
    ConfigMapName:      kube-root-ca.crt
    ConfigMapOptional:  <nil>
    DownwardAPI:        true
QoS Class:       BestEffort
Node-Selectors:  <none>
Tolerations:     node.kubernetes.io/not-ready:NoExecute op=Exists for 3
00s
                 node.kubernetes.io/unreachable:NoExecute op=Exists for
300s
Events:
  Type          Reason              Age   From          Message
  ----          -
Warning        FailedScheduling    7s    default-scheduler  0/1 nodes are available: 1 no
de(s) had untolerated taint {node-role.kubernetes.io/control-plane: }. preemption:
0/1 nodes are available: 1 Preemption is not helpful for scheduling.

```

- `kubectl taint nodes --all node-role.kubernetes.io/control-plane-`

```

[ec2-user@ip-172-31-26-174 ~]$ kubectl taint nodes --all node-role.kubernetes.io
/control-plane-
node/ip-172-31-26-174.ec2.internal untainted

```

6. Now check pod status is is running perform **kubectl get pods** this command.

```

[ec2-user@ip-172-31-28-70 docker]$ kubectl get pods
NAME      READY   STATUS             RESTARTS   AGE
nginx     0/1     ContainerCreating   0           39s
[ec2-user@ip-172-31-28-70 docker]$ kubectl get pods
NAME      READY   STATUS    RESTARTS   AGE
nginx     1/1     Running   1 (45s ago)  70s

```

7. Lastly, mention the port you want to host. Here i have used localhost 8081 then check it.

`kubectl port-forward nginx 8081:80`

```

[ec2-user@ip-172-31-26-174 ~]$ kubectl port-forward nginx 8081:80
Forwarding from 127.0.0.1:8081 -> 80
Forwarding from [::1]:8081 -> 80

```

8. Verify your deployment

Open up a new terminal and ssh to your EC2 instance.

Then, use this curl command to check if the Nginx server is running.

`curl --head http://127.0.0.1:8081`

If the response is 200 OK and you can see the Nginx server name, your deployment was successful. We have successfully deployed our Nginx server on our EC2 instance.

Conclusion: Firstly I created an EC2 AWS Linux instance successfully. then installed docker and

kubernetes successfully, then initialized kubernetes which gave me token and chown and mkdir command. Then I executed mkdir and chown the command successfully. Then I installed a networking plugin called flannel successfully. Then I tried to deploy nginx which initially gave an error. Then I deployed (simple-pod.yml) nginx successfully and also checked by using the get pods command, then hosted it on localhost 8081 i.e. <http://localhost:8081> successfully.