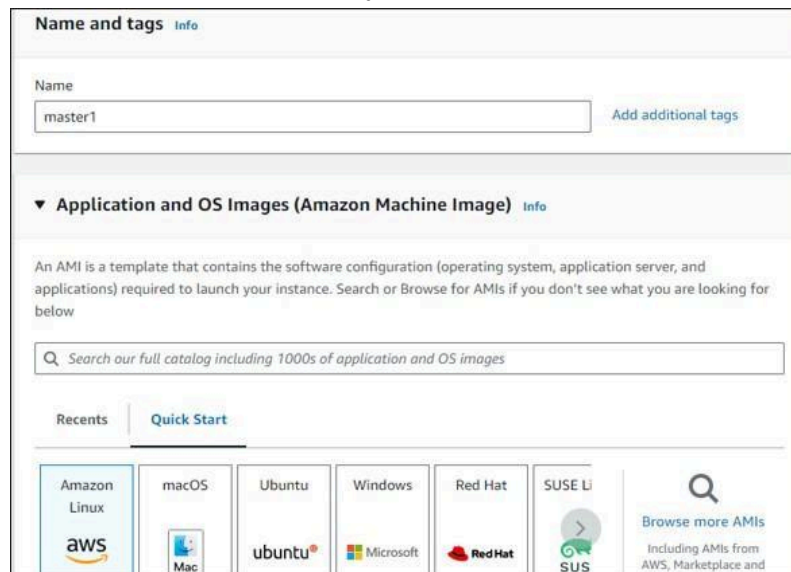


AIM: To install Kubectl and execute Kubectl commands to manage the Kubernetes cluster and deploy Your First Kubernetes Application.

Create 3 EC2 Ubuntu Instances on AWS.

Login to your AWS console. Go to services and in that search for EC2 and create 3 EC2 Ubuntu Instances as master 1, node1 and node 2. While making an instance make sure to select Amazon Linux and in linux type instead of default t2.micro select t2.medium.



Name and tags [Info](#)

Name

master1 [Add additional tags](#)

▼ **Application and OS Images (Amazon Machine Image)** [Info](#)

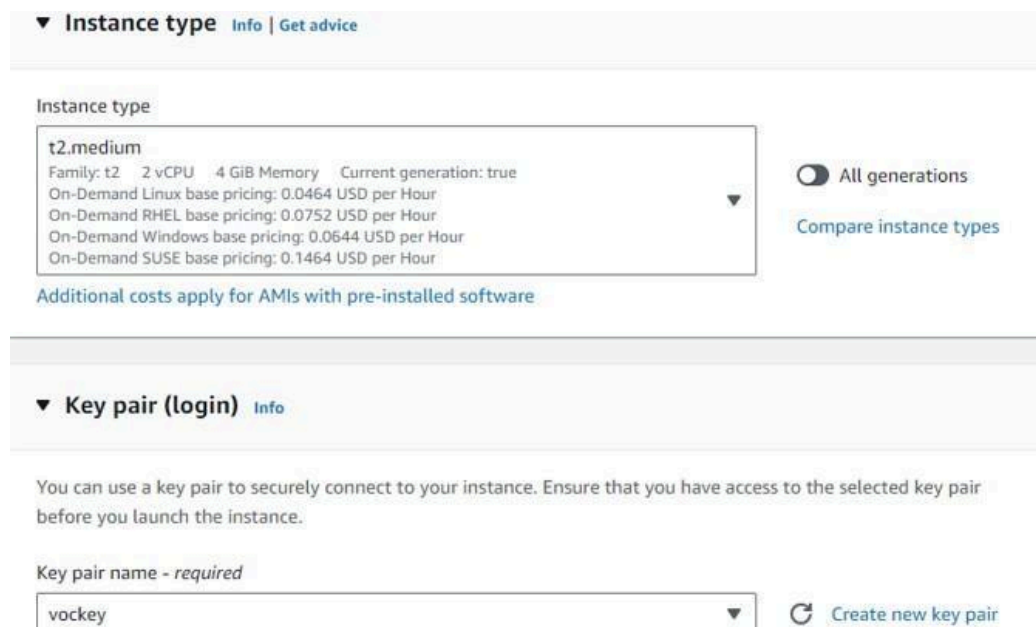
An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. Search or Browse for AMIs if you don't see what you are looking for below

Q Search our full catalog including 1000s of application and OS images

Recents **Quick Start**

Amazon Linux macOS Ubuntu Windows Red Hat SUSE Linux

[Browse more AMIs](#)
Including AMIs from AWS, Marketplace and



▼ **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

[Additional costs apply for AMIs with pre-installed software](#)

☒ All generations [Compare instance types](#)

▼ **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

vockey [Create new key pair](#)


```

RUNNING SCRIPTSET: 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      1/10
Verifying      : docker-25.0.6-1.amzn2023.0.2.x86_64      2/10
Verifying      : iptables-libs-1.8.8-3.amzn2023.0.2.x86_64  3/10
Verifying      : iptables-nft-1.8.8-3.amzn2023.0.2.x86_64  4/10
Verifying      : libcgrou-3.0-1.amzn2023.0.1.x86_64        5/10
Verifying      : libnetfilter_conntrack-1.0.8-2.amzn2023.0.2.x86_64  6/10
Verifying      : libnftnl-1.0.1-19.amzn2023.0.2.x86_64    7/10
Verifying      : libnftnl-1.2.2-2.amzn2023.0.2.x86_64     8/10
Verifying      : pigz-2.5-1.amzn2023.0.3.x86_64           9/10
Verifying      : runc-1.1.13-1.amzn2023.0.1.x86_64        10/10

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  libcgrou-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!
[root@ip-172-31-21-176 ec2-user]# systemctl start docker
[root@ip-172-31-21-176 ec2-user]#

```

INSTALLATION OF KUBERNETES

After installing and starting the docker in all the three instances ,now lets install kubernetes for the installation we use the following steps:

STEP 1:Set SELinux to permissive mode:

Syntax:`sudo setenforce 0`

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

```
[root@ip-172-31-25-172 docker]# sudo setenforce 0
sudo sed -i 's/^SELINUX=enforcing$/SELINUX=permissive/' /etc/selinux/config
```

STEP 2:Add the Kubernetes yum repository. The exclude parameter in the repository definition ensures that the packages related to Kubernetes are not upgraded upon running yum update as there's a special procedure that must be followed for upgrading Kubernetes

```
[root@ip-172-31-21-176 ec2-user]# sudo su
[root@ip-172-31-21-176 ec2-user]# yum repolist
repo id                                repo name
amazonlinux                            Amazon Linux 2023 repository
kernel-livepatch                       Amazon Linux 2023 Kernel Livepatch repository
[root@ip-172-31-21-176 ec2-user]# cat <<EOF | sudo tee /etc/yum.repos.d/kubernetes.repo
[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
exclude=kubelet kubeadm kubectl cri-tools kubernetes-cni
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
exclude=kubelet kubeadm kubectl cri-tools kubernetes-cni
[root@ip-172-31-21-176 ec2-user]#
```

i-0defb5859fc2b0488 (node1)

PublicIPs: 54.157.60.252 PrivateIPs: 172.31.21.176

STEP 3: Install kubelet, kubeadm and kubectl:

Syntax: `sudo yum install -y kubelet kubeadm kubectl`

`--disableexcludes=kubernetes`

```
Last login: Fri Sep 13 17:58:28 2024 from 18.206.107.27
[ec2-user@ip-172-31-21-176 ~]$ sudo su
[root@ip-172-31-21-176 ec2-user]# sudo yum install -y kubelet kubeadm kubectl --disableexcludes=kubernetes
Kubernetes
Dependencies resolved.
60 kB/s | 9.4 kB    00:00
=====
Package                                Architecture      Version            Repository          Size
-----
Installing:
kubeadm                                x86_64            1.31.1-150500.1.1  kubernetes          11 M
kubectl                                x86_64            1.31.1-150500.1.1  kubernetes          11 M
kubelet                                x86_64            1.31.1-150500.1.1  kubernetes          15 M
Installing dependencies:
containerd-tools                       x86_64            1.4.6-2.amzn2023.0.2  amazonlinux        208 k
cri-tools                              x86_64            1.31.1-150500.1.1  kubernetes          6.9 M
kubernetes-cni                         x86_64            1.5.1-150500.1.1  kubernetes          7.1 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
```

STEP 4: Enable the kubelet service before running kubeadm:

Syntax: `sudo systemctl enable --now kubelet`

```
[root@ip-172-31-21-176 ec2-user]# sudo systemctl enable --now kubelet
Created symlink /etc/systemd/system/multi-user.target.wants/kubelet.service → /usr/lib/systemd/system/kubelet.service.
[root@ip-172-31-21-176 ec2-user]#
```

i-0defb5859fc2b0488 (node1)

PublicIPs: 54.157.60.252 PrivateIPs: 172.31.21.176

STEP 5: It can be seen from the repolist command which lists all the repository we can see that kubernetes in installed repeat all these steps on master1 and node2.

```
[root@ip-172-31-21-176 ec2-user]# yum repolist
repo id                                repo name
amazonlinux                           Amazon Linux 2023 repository
kernel-livepatch                      Amazon Linux 2023 Kernel Livepatch repository
kubernetes                            Kubernetes
[root@ip-172-31-21-176 ec2-user]#
```

i-0defb5859fc2b0488 (node1)

PublicIPs: 54.157.60.252 PrivateIPs: 172.31.21.176

STEP 6 : This command disables swap space and configures the system to use iptables for bridged network traffic, then apply these settings.

Syntax: `sudo swapoff -a`

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

```
[root@ip-172-31-16-56 ec2-user]# 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
```

STEP 7: Initialize Kubernetes in master instance .

Syntax: `kubeadm init`

```
[root@ip-172-31-16-56 ec2-user]# 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'
W0913 18:58:26.902514 34809 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-172-31-16-56.ec2.internal kubernetes kubernetes.default kubernetes.default.svc kubernetes.default.svc.cluster.local] and IPs [10.96.0.1 172.31.16.56]
[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-172-31-16-56.ec2.internal localhost] and IPs [172.31.16.56 127.0.0.1 ::1]
[certs] Generating "etcd/peer" certificate and key
[certs] etcd/peer serving cert is signed for DNS names [ip-172-31-16-56.ec2.internal localhost] and IPs [172.31.16.56 127.0.0.1 ::1]
[certs] Generating "etcd/healthcheck-client" certificate and key
```

i-0ddf50a232db19957 (master1)

PublicIPs: 3.88.204.138 PrivateIPs: 172.31.16.56

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 172.31.16.56:6443 --token oghyi3.fnspro8pevgr0d5 \
--discovery-token-ca-cert-hash sha256:ec71ffc0d9fd79263fb8909d938da8d29e5f15a21ab5e0a17ec93514e8c4ecb8
```

Use the mkdir and chown commands shown above

```
[root@ip-172-31-16-56 ec2-user]# mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

Add a common networking plugin called flannel

Syntax: kubectl apply -f

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

```
[root@ip-172-31-16-56 ~]# 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
```

STEP 8: Apply deployment of nginx server using the following command.

Syntax:

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

```
[root@ip-172-31-16-56 ~]# kubectl apply -f https://k8s.io/examples/application/deployment.yaml
deployment.apps/nginx-deployment created
```

Check whether the pods is created or not by the following command

Syntax: kubectl get pods

```
[root@ip-172-31-16-56 ~]# kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
nginx-deployment-d556bf558-gw8v8	0/1	Pending	0	82s
nginx-deployment-d556bf558-rfk9n	0/1	Pending	0	82s

Kubectl describe pod nginx command describe the pods.

```
[root@ip-172-31-16-56 ~]# kubectl describe pod nginx
Name:          nginx-deployment-d556bf558-gw8v8
Namespace:     default
Priority:       0
Service Account: default
Node:          <none>
Labels:        app=nginx
               pod-template-hash=d556bf558
Annotations:   <none>
Status:        Pending
IP:            <none>
IPs:           <none>
Controlled By: ReplicaSet/nginx-deployment-d556bf558
Containers:
  nginx:
    Image:      nginx:1.14.2
    Port:       80/TCP
    Host Port:  0/TCP
    Environment: <none>
    Mounts:     <none>
```

```
Conditions:
  Type          Status
  PodScheduled  False
Volumes:
  kube-api-access-f9k9s:
    Type:              Projected (a volume that contains injected data from 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 300s
                     node.kubernetes.io/unreachable:NoExecute op=Exists for 300s
Events:
  Type     Reason              Age   From                      Message
  ----     ------              -
Warning   FailedScheduling    114s  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.
Warning   FailedScheduling    3m18s  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.
```

```
[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
```

STEP 9: Check whether the pods are running or not.

```
[ec2-user@ip-172-31-26-174 ~]$ kubectl get pods
NAME      READY   STATUS    RESTARTS   AGE
nginx     1/1     Running   1 (6s ago)  90s
```

STEP 10: Mention the port that you want to host
on Syntax: 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
```

STEP 11: Then verify your deployment

Conclusion : In this experiment we have created 3 EC2 instances. Setting SSH for establishing connections in that we have installed and started docker and kubernetes ,initialising kubernetes we use the mkdir and chown commands that we get by initializing the kubernetes then we add a common networking plugin called flannel then we apply deployment to nginx server we describe the pods and we check the status of the pods we mention the port that we want to host on and at the end we verify the deployment of the kubernetes application by performing the following steps we learned to deploy the our Kubernetes Application.