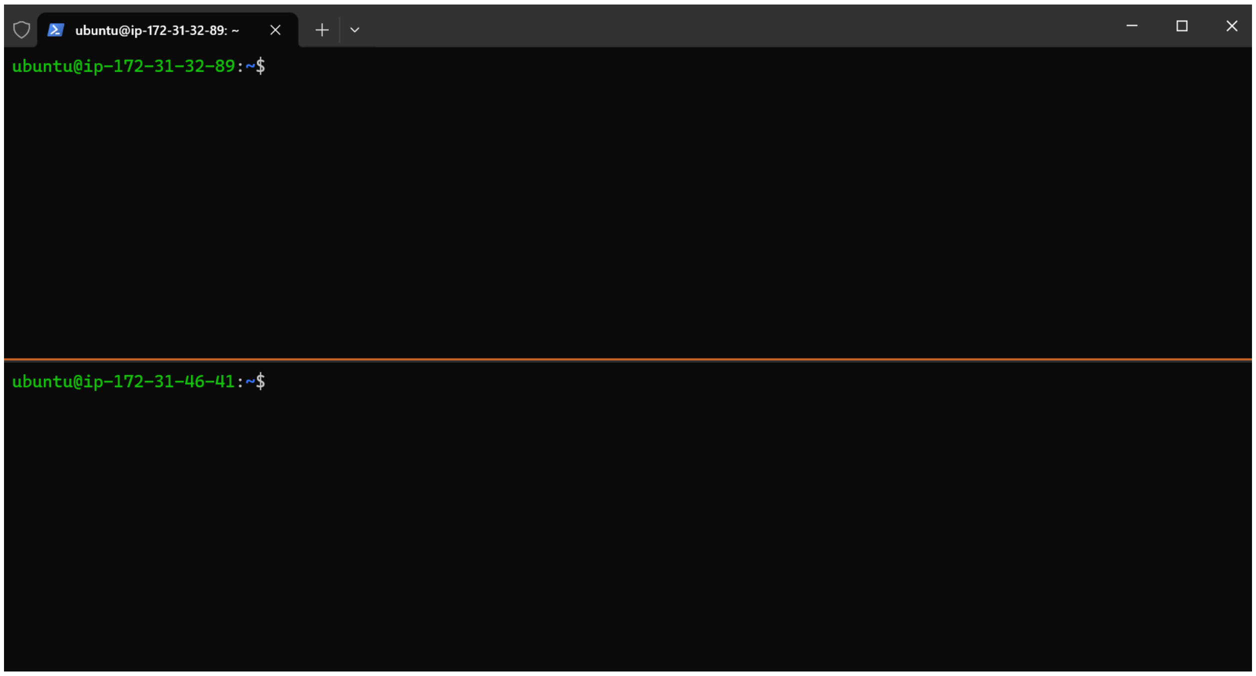
Kubernetes Cluster Setup on AWS

Step 1:Prerequisites

- Launch 2 EC2 instances (t2.medium) with Ubuntu.
- Allow all traffic in the security group.
- Connect to both nodes via SSH using multi-execution mode.



```
ubuntu@ip-172-31-32-89:~$
```

```
ubuntu@ip-172-31-46-41:~$
```

Step 2: Initial Setup (Run on All Instances)

1. Disable SWAP. We need to disable Swap for kubeadm to work properly.

```
swapoff -a
sudo sed -i '/ swap / s/^\(.*\)$/#\1/g' /etc/fstab
```

2. Enable iptables Bridged Traffic on all the Nodes. Execute the following commands on all the nodes for IPtables to see bridged traffic.

```
cat <<EOF | sudo tee /etc/modules-load.d/k8s.conf
overlay
```

```
sudo modprobe overlay
sudo modprobe br_netfilter

# sysctl params required by setup, params persist across reboots
cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf
net.bridge.bridge-nf-call-iptables  = 1
net.bridge.bridge-nf-call-ip6tables = 1
net.ipv4.ip_forward                 = 1
EOF

# Apply sysctl params without reboot
sudo sysctl --system
```

3. Install CRI-O Runtime. The basic requirement for a Kubernetes cluster is a container runtime. We can use any container runtime we want, here we will be using CRI-O.

```
OS="xUbuntu_22.04"

VERSION="1.28"

cat <<EOF | sudo tee /etc/apt/sources.list.d/devel:kubic:libcontainers:stable.list
deb
https://download.opensuse.org/repositories/devel:/kubic:/libcontainers:/stable/$OS/
/
EOF
cat <<EOF | sudo tee
/etc/apt/sources.list.d/devel:kubic:libcontainers:stable:cri-o:$VERSION.list
deb
http://download.opensuse.org/repositories/devel:/kubic:/libcontainers:/stable:/cri-o:/
$VERSION/$OS/ /
EOF
```

4. Add the GPG keys for CRI-O to the system's list of trusted keys.

```
curl -L
https://download.opensuse.org/repositories/devel:kubic:libcontainers:stable:cri-o:$V
ERSION/$OS/Release.key | sudo apt-key --keyring
/etc/apt/trusted.gpg.d/libcontainers.gpg add -
```

```
curl -L
https://download.opensuse.org/repositories/devel:/kubic:/libcontainers:/stable/$OS/
Release.key | sudo apt-key --keyring /etc/apt/trusted.gpg.d/libcontainers.gpg add -
```

5. Update and install cri-o and cri-o tools.

```
sudo apt-get update
sudo apt-get install cri-o cri-o-runc cri-tools -y
```

6. Reload the systemd configurations and enable cri-o.

```
sudo systemctl daemon-reload
sudo systemctl enable crio --now
```

The cri-tools include crictl, a CLI utility for interacting with containers created by the container runtime. When utilizing container runtimes other than Docker, you can employ the crictl utility for debugging containers on the nodes.

Step 3: Install Kubeadm & Kubelet & Kubectl

1. Install the required dependencies

```
sudo apt-get update

# apt-transport-https may be a dummy package; if so, you can skip that package

sudo apt-get install -y apt-transport-https ca-certificates curl gpg
```

2. Download the public signing key for the Kubernetes package repositories. The same signing key is used for all repositories so you can disregard the version in the URL

```
# If the directory `/etc/apt/keyrings` does not exist, it should be created before the curl command, read the note below.

# sudo mkdir -p -m 755 /etc/apt/keyrings

curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.33/deb/Release.key | sudo gpg --dearmor -o
/etc/apt/keyrings/kubernetes-apt-keyring.gpg
```

3. Add the Kubernetes APT repository to your system.

```
# This overwrites any existing configuration in /etc/apt/sources.list.d/kubernetes.list

echo 'deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg]
https://pkgs.k8s.io/core:/stable:/v1.33/deb/ /' | sudo tee /etc/apt/sources.list.d/kubernetes.list
```

4. Update the `apt` package index, install kubelet, kubeadm and kubectl, and pin their version and add hold to the packages to prevent upgrades.

```
sudo apt-get update

sudo apt-get install -y kubelet kubeadm kubectl

sudo apt-mark hold kubelet kubeadm kubectl
```

5. Add the node IP to KUBELET_EXTRA_ARGS.

```
sudo apt-get install -y jq

# Get the primary private IP of the instance
local_ip=$(hostname -I | awk '{print $1}')

# Write to /etc/default/kubelet with proper permissions
echo "KUBELET_EXTRA_ARGS=--node-ip=$local_ip" | sudo tee /etc/default/kubelet
```

Step 4: Initialize Kubeadm On Master Node

1. Set the following environment variables:

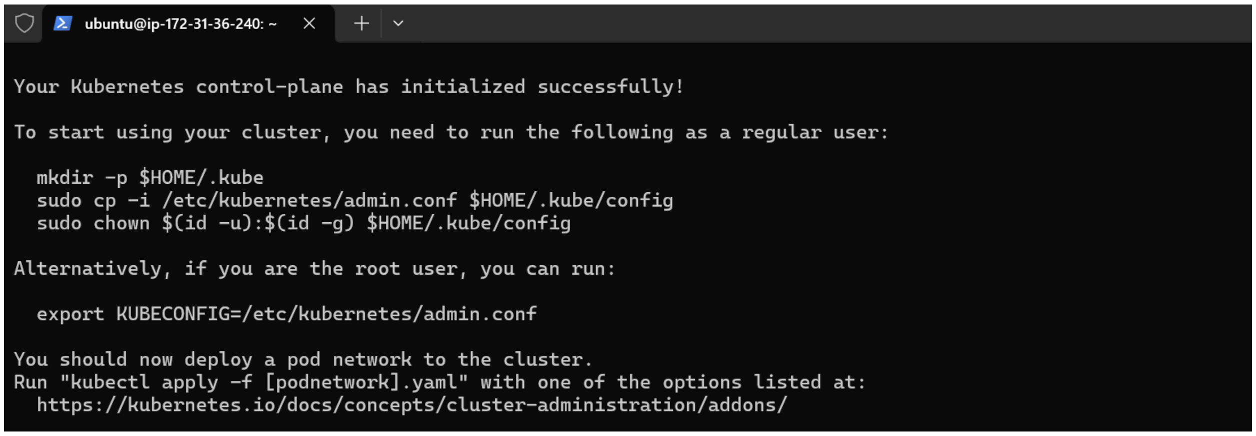
```
IPADDR=$(curl ifconfig.me && echo "")
NODENAME=$(hostname -s)
POD_CIDR="192.168.0.0/16"
```

2. Initialize the master node control plane configurations using the kubeadm command.

```
sudo kubeadm init --control-plane-endpoint=$IPADDR
```

```
--apiserver-cert-extra-sans=$IPADDR  --pod-network-cidr=$POD_CIDR
--node-name $NODENAME --ignore-preflight-errors Swap
```

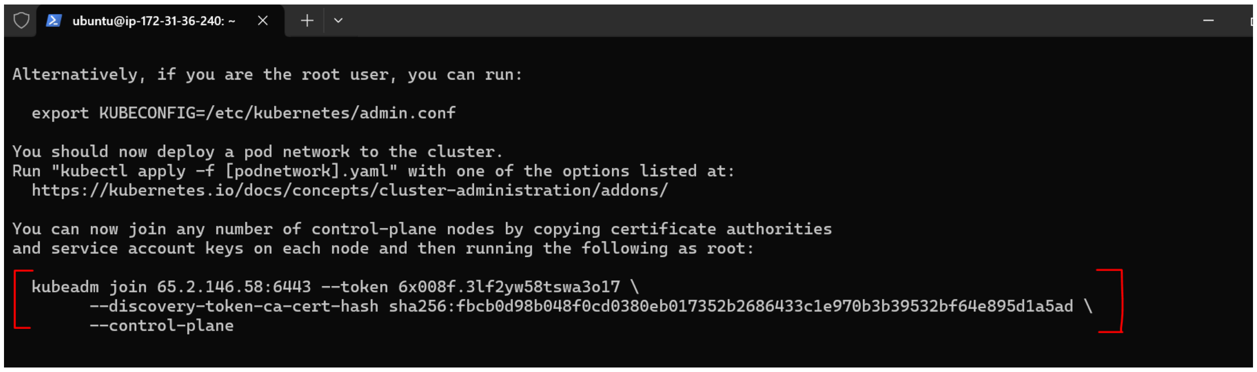
Output:



ubuntu@ip-172-31-36-240: ~

```
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/
```

Copy the join command and save it somewhere, we will need it for joining the worker node to the master.



ubuntu@ip-172-31-36-240: ~

```
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/

You can now join any number of control-plane nodes by copying certificate authorities
and service account keys on each node and then running the following as root:

  kubeadm join 65.2.146.58:6443 --token 6x008f.3lf2yw58tswa3o17 \
        --discovery-token-ca-cert-hash sha256:fbcb0d98b048f0cd0380eb017352b2686433c1e970b3b39532bf64e895d1a5ad \
        --control-plane
```

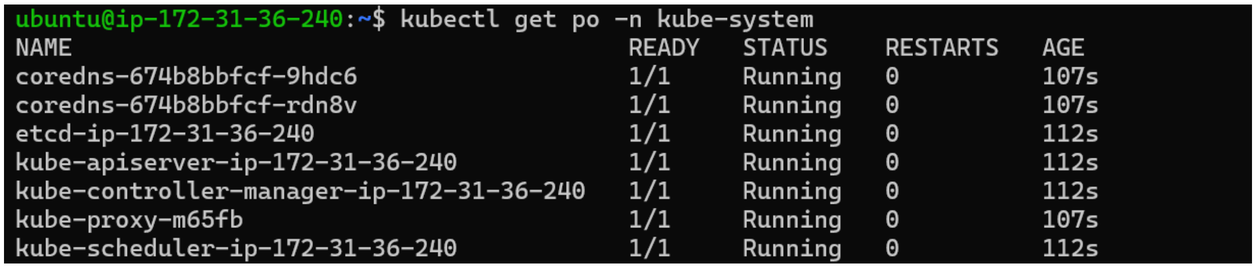
3. Use the following commands from the output to create the kubeconfig in master so that you can use kubectl to interact with cluster API.

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

4. Now, verify the kubeconfig by executing the following kubectl command to list all the pods in the kube-system namespace.

```
kubectl get po -n kube-system
```

Output:

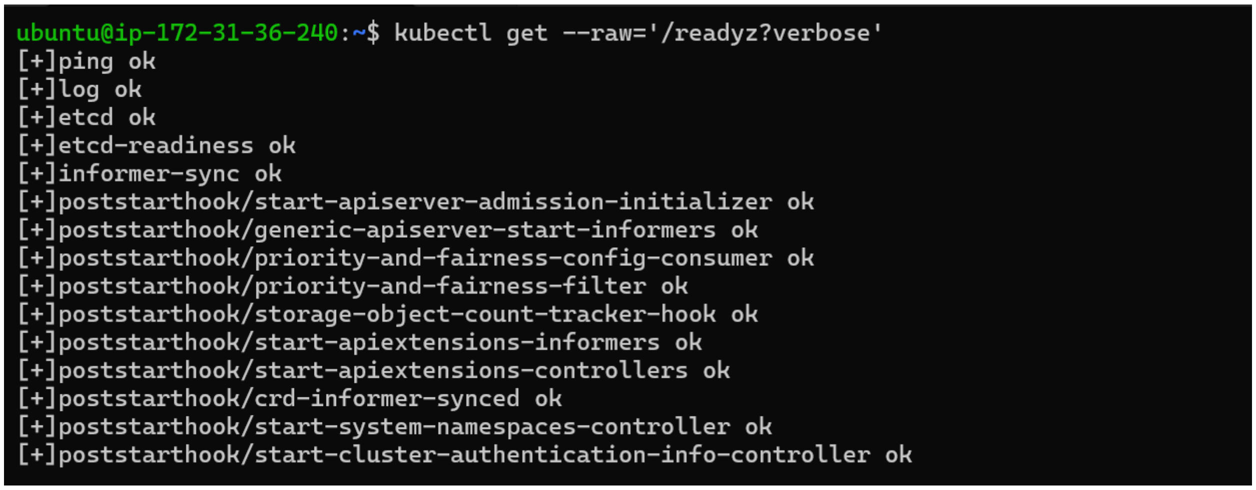


```
ubuntu@ip-172-31-36-240:~$ kubectl get po -n kube-system
NAME                                       READY   STATUS    RESTARTS   AGE
coredns-674b8bbfcf-9hdc6                   1/1     Running   0          107s
coredns-674b8bbfcf-rdn8v                   1/1     Running   0          107s
etcd-ip-172-31-36-240                      1/1     Running   0          112s
kube-apiserver-ip-172-31-36-240            1/1     Running   0          112s
kube-controller-manager-ip-172-31-36-240   1/1     Running   0          112s
kube-proxy-m65fb                           1/1     Running   0          107s
kube-scheduler-ip-172-31-36-240            1/1     Running   0          112s
```

5. You verify all the cluster component health statuses using the following command.

```
kubectl get --raw='/readyz?verbose'
```

Output:



```
ubuntu@ip-172-31-36-240:~$ kubectl get --raw='/readyz?verbose'
[+]ping ok
[+]log ok
[+]etcd ok
[+]etcd-readiness ok
[+]informer-sync ok
[+]poststarthook/start-apiserver-admission-initializer ok
[+]poststarthook/generic-apiserver-start-informers ok
[+]poststarthook/priority-and-fairness-config-consumer ok
[+]poststarthook/priority-and-fairness-filter ok
[+]poststarthook/storage-object-count-tracker-hook ok
[+]poststarthook/start-apiextensions-informers ok
[+]poststarthook/start-apiextensions-controllers ok
[+]poststarthook/crd-informer-synced ok
[+]poststarthook/start-system-namespaces-controller ok
[+]poststarthook/start-cluster-authentication-info-controller ok
```

6. By default, apps won't get scheduled on the master node. If you want to use the master node for scheduling apps, taint the master node.

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

Step 5: Install Calico Network Plugin

Kubeadm does not configure any network plugin. You need to install a network plugin of your choice for kubernetes pod networking and enable network policy.

We will use the Calico network plugin for this setup.

- Run the following commands to install the Calcio network plugin

```
kubectl create -f
https://raw.githubusercontent.com/projectcalico/calico/v3.26.1/manifests/tigera-ope
rator.yaml

curl
https://raw.githubusercontent.com/projectcalico/calico/v3.26.1/manifests/custom-re
sources.yaml -O

kubectl create -f custom-resources.yaml
```

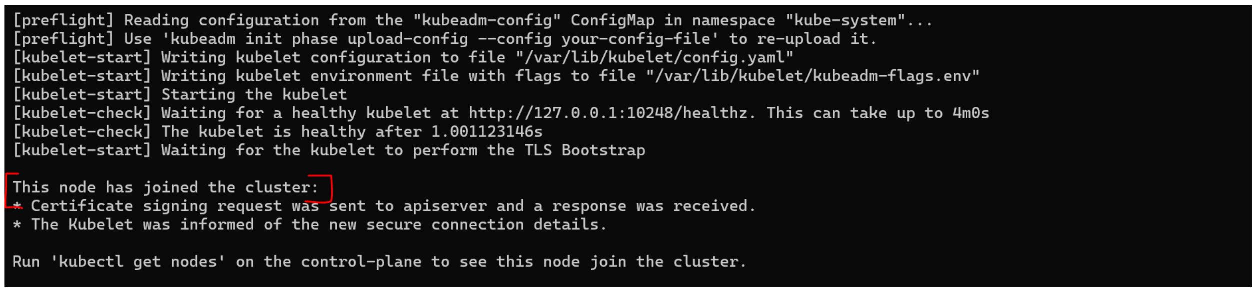
Step 6: Joining of Worker nodes to master node

1. Now, let's join the worker node to the master node using the Kubeadm join command you have got in the output while setting up the master node.

Note: Execute the commands in this section only on the worker nodes.

```
sudo kubeadm join 3.86.197.238:6443 --token 3nf11u.w40mnymbrlkr8f88 \
        --discovery-token-ca-cert-hash
sha256:d904af14e6bee5af2020e2b1a9572345403b1a1c7a095bb5e8584e04f1db366
7
```

Output:

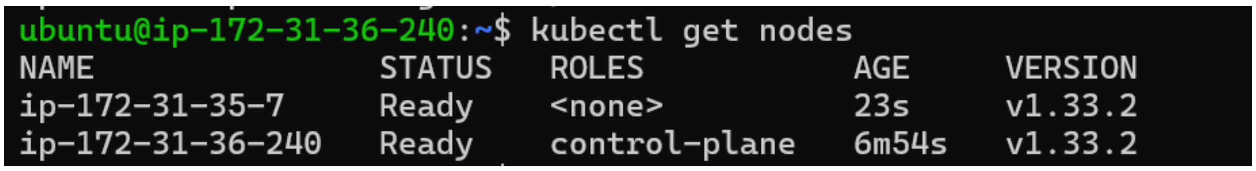


```
[preflight] Reading configuration from the "kubeadm-config" ConfigMap in namespace "kube-system"...
[preflight] Use 'kubeadm init phase upload-config --config your-config-file' to re-upload it.
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flags.env"
[kubelet-start] Starting the kubelet
[kubelet-check] Waiting for a healthy kubelet at http://127.0.0.1:10248/healthz. This can take up to 4m0s
[kubelet-check] The kubelet is healthy after 1.001123146s
[kubelet-start] Waiting for the kubelet to perform the TLS Bootstrap

This node has joined the cluster:
* Certificate signing request was sent to apiserver and a response was received.
* The Kubelet was informed of the new secure connection details.

Run 'kubectl get nodes' on the control-plane to see this node join the cluster.
```

2. Execute the kubectl command from the master node to check if the node is added to the master.

```
kubectl get nodes
```

Output:



```
ubuntu@ip-172-31-36-240:~$ kubectl get nodes
NAME                STATUS   ROLES           AGE     VERSION
ip-172-31-35-7      Ready    <none>          23s     v1.33.2
ip-172-31-36-240    Ready    control-plane   6m54s   v1.33.2
```

Step 7: Deploy a sample Nginx App

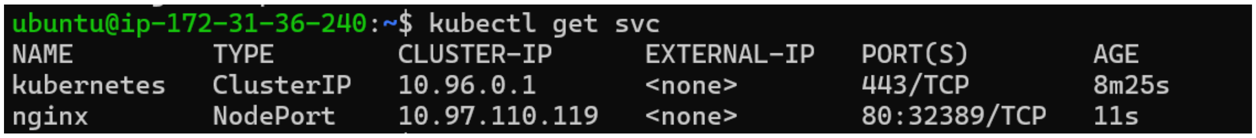
1. Create an Nginx deployment. Execute the following directly on the command line.

```
kubectl run nginx --image=nginx --port=80
kubectl expose pod nginx --port=80 --type=NodePort
```

2. A random port between 30000-32767 and assigned to nginx app.To access that port we run the following command:

```
kubectl get svc
```

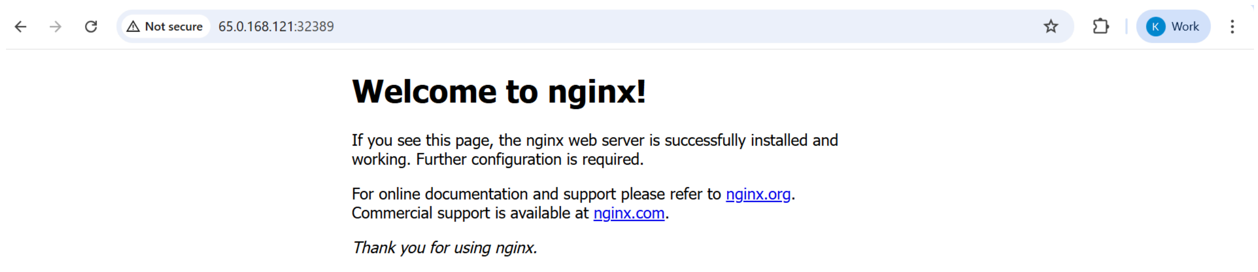
Output:



```
ubuntu@ip-172-31-36-240:~$ kubectl get svc
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
kubernetes   ClusterIP   10.96.0.1       <none>        443/TCP        8m25s
nginx        NodePort    10.97.110.119   <none>        80:32389/TCP   11s
```

Here, we can see that the nginx app has been deployed on port 32389.
We can now access that port with our master node ip or worker nodes ip and get the following result:

This shows that we have set up our cluster successfully!



Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

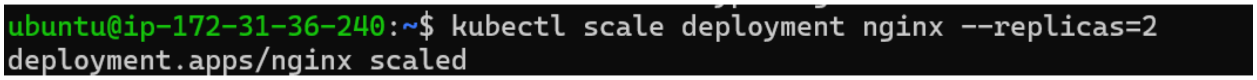
For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

If we want to scale use the following command:This tells Kubernetes to make 2 pods of the nginx deployment.

```
kubectl scale deployment nginx --replicas=2
```

Output:

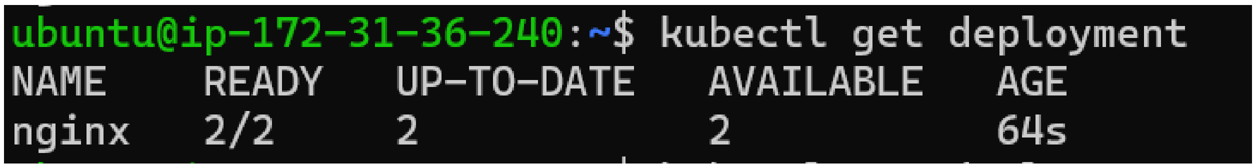


```
ubuntu@ip-172-31-36-240:~$ kubectl scale deployment nginx --replicas=2
deployment.apps/nginx scaled
```

To check deployment status:

```
kubectl get deployment nginx
```

Output:



```
ubuntu@ip-172-31-36-240:~$ kubectl get deployment
NAME     READY   UP-TO-DATE   AVAILABLE   AGE
nginx    2/2     2            2           64s
```