Install Kubernetes on Ubuntu 24.04

1) Set Host Name and Update hosts file

SSH to each Ubuntu 24.04 instance and set their respective hostname using hostnamectl command

```
$ sudo hostnamectl set-hostname "k8s-master-noble"      // Master Node

$ sudo hostnamectl set-hostname "k8s-worker01-noble"    // Worker Node 1

$ sudo hostnamectl set-hostname "k8s-worker02-noble"    // Worker Node 2
```

Add the following lines to **/etc/hosts** file on each instance.

```
192.168.1.120  k8s-master-noble

192.168.1.121  k8s-worker01-noble

192.168.1.122  k8s-worker02-noble
```

2) Disable Swap and Load Kernel Modules

It is highly recommended to disable swap space on your Ubuntu instances so that Kubernetes cluster works smoothly. Run beneath command on each instance to disable swap space.

```
$ sudo swapoff -a

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

Now, load the following kernel modules using modprobe command.

```
$ sudo modprobe overlay

$ sudo modprobe br_netfilter
```

For the permanent loading of these modules, create the file with following content.

```
$ sudo tee /etc/modules-load.d/k8s.conf <<EOF

overlay

br_netfilter

EOF
```

Next, add the kernel parameters like IP forwarding. Create a file and load the parameters using sysctl command,

```
$ sudo tee /etc/sysctl.d/kubernetes.conf <<EOT

net.bridge.bridge-nf-call-ip6tables = 1

net.bridge.bridge-nf-call-iptables = 1

net.ipv4.ip_forward = 1

EOT
```

To load the above kernel parameters, run

```
$ sudo sysctl –system
```

3) Install and Configure Containerd

Containerd provides the container run time for Kubernetes. So, Install containerd on all three instances.

First install containerd dependencies,

```
$ sudo apt install -y curl gnupg2 software-properties-common apt-transport-https ca-certificates
```

Next, add containerd repository using following set of commands.

```
$ sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmour -o
/etc/apt/trusted.gpg.d/containerd.gpg
```

```
$ sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
```

Now, install containerd using following apt command.

```
$ sudo apt update && sudo apt install containerd.io -y
```

Next, configure containerd so that it starts using **SystemdCgroup**. Run beneath commands.

```
$ containerd config default | sudo tee /etc/containerd/config.toml >/dev/null 2>&1
```

```
$ sudo sed -i 's/SystemdCgroup \= false/SystemdCgroup \= true/g' /etc/containerd/config.toml
```

Restart containerd service so that above changes come into the affect.

```
$ sudo systemctl restart containerd
```

4) Add Kubernetes Package Repository

Kubernetes packages are not available in the default package repositories of Ubuntu 24.04, so for its installation first add it’s repository. Run these steps on each instance.

Note: At the time of writing this post, latest version of Kubernetes was **1.30**. So you can this version according your requirement.

Download the public signing key for the Kubernetes package repository using curl command.

```
$ curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.30/deb/Release.key | sudo gpg --dearmor -o /etc/apt/keyrings/k8s.gpg
```

Next, add the Kubernetes repository by running following command.

```
$ echo 'deb [signed-by=/etc/apt/keyrings/k8s.gpg] https://pkgs.k8s.io/core:/stable:/v1.30/deb/ /' | sudo tee
/etc/apt/sources.list.d/k8s.list
```

5) Insall Kubernetes Components (Kubeadm, kubelet & kubectl)

Install Kubernetes components like **Kubeadm**, **kubelet** and **kubectl**, run following apt commands on all the instances.

```
$ sudo apt update
```

```
$ sudo apt install kubelet kubeadm kubectl -y
```

6) Initialize Kubernetes Cluster

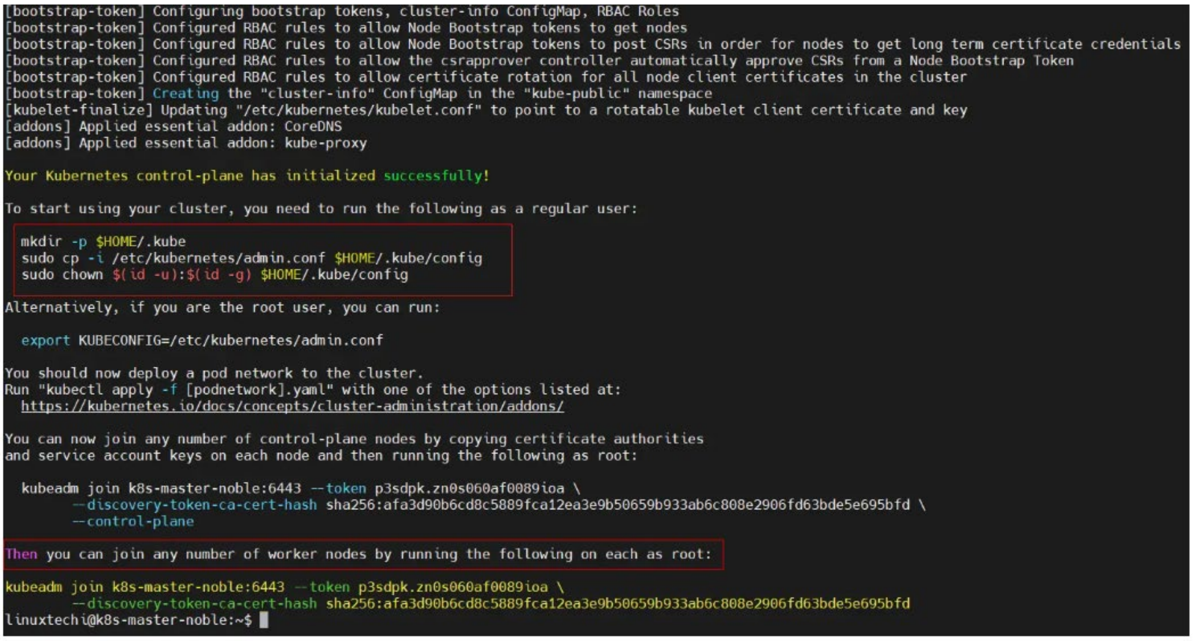
As all the prerequisites are met, now we are good to start the installation of Kubernetes on Ubuntu 24.04.

Run following **Kubeadm** command from the master node only to initialize the Kubernetes cluster.

```
$ sudo kubeadm init --control-plane-endpoint=k8s-master-noble
kubectl config set-cluster kubernetes --server=https://<CIDR>:6443
```

This command will pull the required images for your Kubernetes cluster. Once this command is executed successfully, we

will get the output something like below:



```
[bootstrap-token] Configuring bootstrap tokens, cluster-info ConfigMap, RBAC Roles
[bootstrap-token] Configured RBAC rules to allow Node Bootstrap tokens to get nodes
[bootstrap-token] Configured RBAC rules to allow Node Bootstrap tokens to post CSRs in order for nodes to get long term certificate credentials
[bootstrap-token] Configured RBAC rules to allow the csrapprover controller automatically approve CSRs from a Node Bootstrap Token
[bootstrap-token] Configured RBAC rules to allow certificate rotation for all node client certificates in the cluster
[bootstrap-token] Creating the "cluster-info" ConfigMap in the "kube-public" namespace
[kubelet-finalize] Updating "/etc/kubernetes/kubelet.conf" to point to a rotatable kubelet client certificate and key
[addons] Applied essential addon: CoreDNS
[addons] Applied essential addon: kube-proxy

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/

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 k8s-master-noble:6443 --token p3sdpk.zn0s060af0089ioa \
        --discovery-token-ca-cert-hash sha256:afa3d90b6cd8c5889fca12ea3e9b50659b933ab6c808e2906fd63bde5e695bfd \
        --control-plane

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

kubeadm join k8s-master-noble:6443 --token p3sdpk.zn0s060af0089ioa \
        --discovery-token-ca-cert-hash sha256:afa3d90b6cd8c5889fca12ea3e9b50659b933ab6c808e2906fd63bde5e695bfd
linuxtechi@k8s-master-noble:~$ █
```

In the output above, we will get a series of commands like how to start interacting with your Kubernetes cluster and command to join any worker node to join this cluster.

On the master node, run following set of commands.

$ mkdir -p $HOME/.kube

$ sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

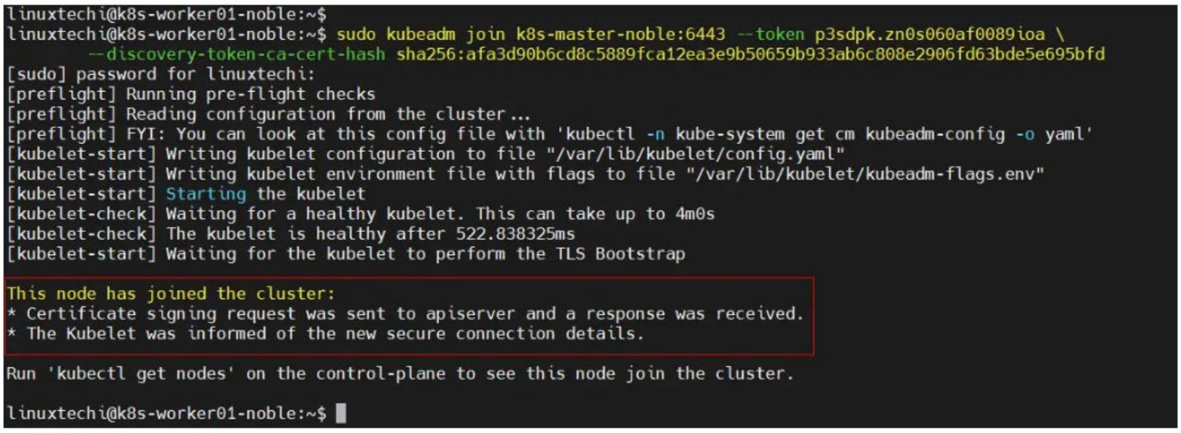
$ sudo chown $(id -u):$(id -g) $HOME/.kube/config

Next copy the command to join any worker node from the above output, run it on both the worker nodes. In my case, command would be:

$ sudo kubeadm join k8s-master-noble:6443 --token p3sdpk.zn0s060af0089ioa \

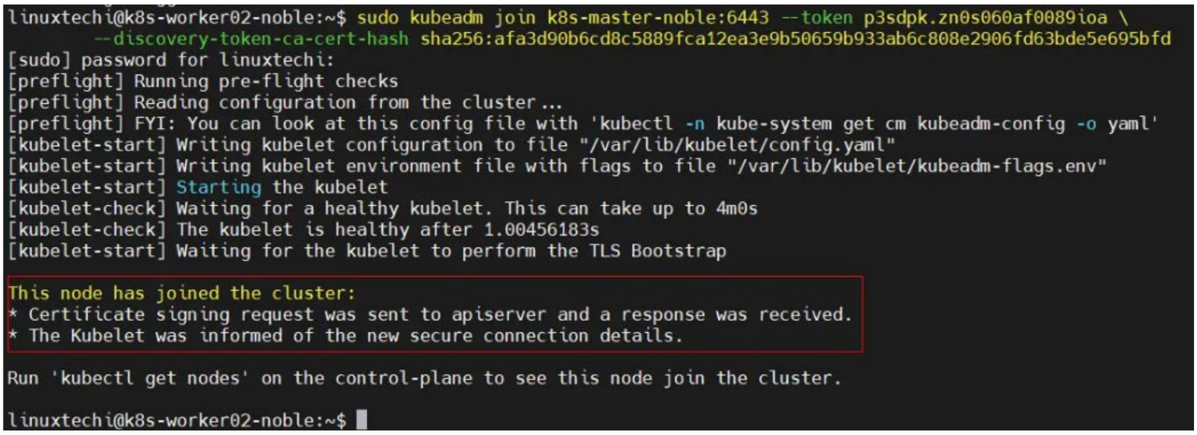
--discovery-token-ca-cert-hash sha256:afa3d90b6cd8c5889fca12ea3e9b50659b933ab6c808e2906fd63bde5e695bfd

Output from first worker node



```
linuxtechi@k8s-worker01-noble:~$
linuxtechi@k8s-worker01-noble:~$ sudo kubeadm join k8s-master-noble:6443 --token p3sdpk.zn0s060af0089ioa \
        --discovery-token-ca-cert-hash sha256:afa3d90b6cd8c5889fca12ea3e9b50659b933ab6c808e2906fd63bde5e695bfd
[sudo] password for linuxtechi:
[preflight] Running pre-flight checks
[preflight] Reading configuration from the cluster...
[preflight] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-config -o yaml'
[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. This can take up to 4m0s
[kubelet-check] The kubelet is healthy after 522.838325ms
[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.

linuxtechi@k8s-worker01-noble:~$ █
```

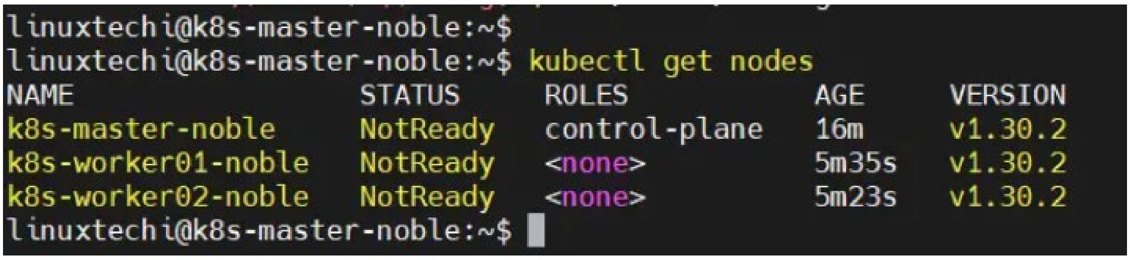
Similarly output from the second worker node



```
linuxtechi@k8s-worker02-noble:~$ sudo kubeadm join k8s-master-noble:6443 --token p3sdpk.zn0s060af0089ioa \
        --discovery-token-ca-cert-hash sha256:afa3d90b6cd8c5889fca12ea3e9b50659b933ab6c808e2906fd63bde5e695bfd
[sudo] password for linuxtechi:
[preflight] Running pre-flight checks
[preflight] Reading configuration from the cluster...
[preflight] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-config -o yaml'
[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. This can take up to 4m0s
[kubelet-check] The kubelet is healthy after 1.00456183s
[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.

linuxtechi@k8s-worker02-noble:~$
```

Now head back to the master node and run **kubectl get nodes** command to verify the status of worker nodes.

$ kubectl get nodes

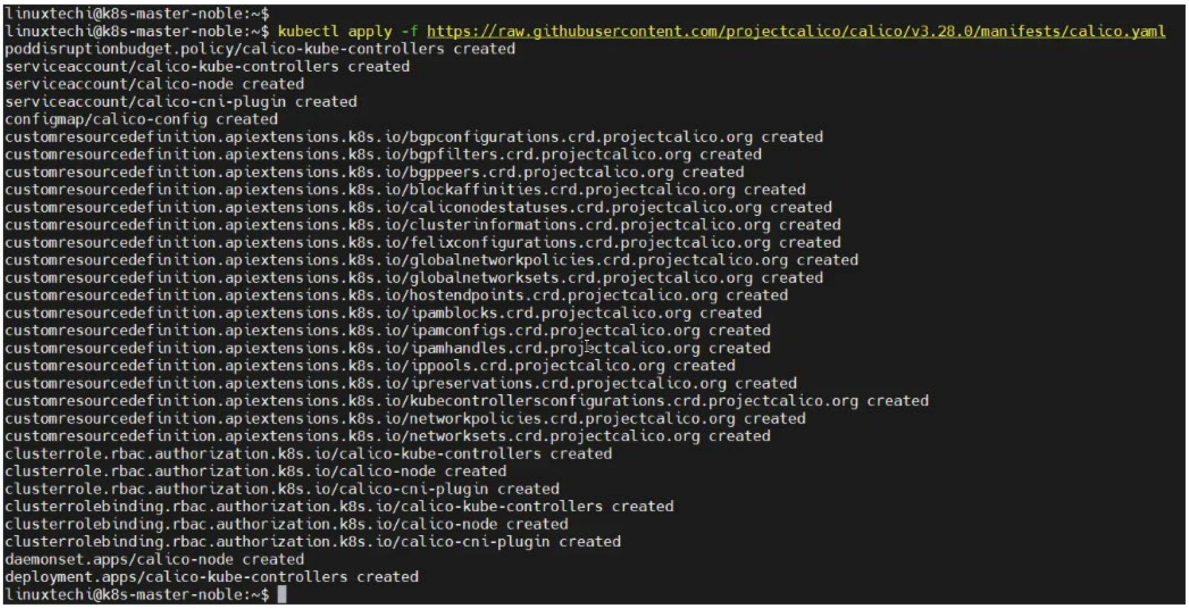


```
linuxtechi@k8s-master-noble:~$
linuxtechi@k8s-master-noble:~$ kubectl get nodes
NAME                STATUS     ROLES           AGE     VERSION
k8s-master-noble    NotReady   control-plane   16m     v1.30.2
k8s-worker01-noble  NotReady   <none>          5m35s   v1.30.2
k8s-worker02-noble  NotReady   <none>          5m23s   v1.30.2
linuxtechi@k8s-master-noble:~$
```

7) Install Calico Network Add-on Plugin

To install calico network plugin, run beneath command from the master node only.

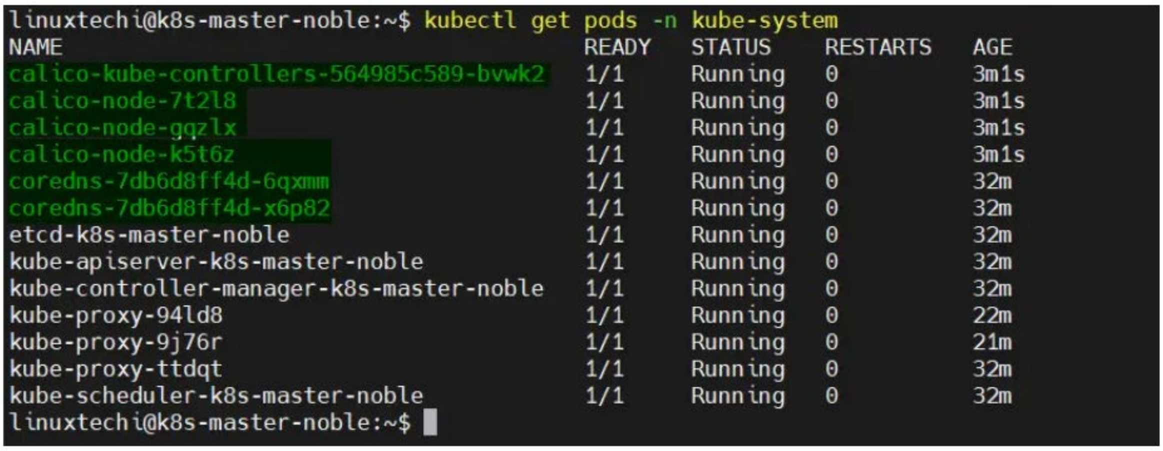
$ kubectl apply -f https://raw.githubusercontent.com/projectcalico/calico/v3.29.1/manifests/calico.yaml



```
linuxtechi@k8s-master-noble:~$
linuxtechi@k8s-master-noble:~$ kubectl apply -f https://raw.githubusercontent.com/projectcalico/calico/v3.28.0/manifests/calico.yaml
poddisruptionbudget.policy/calico-kube-controllers created
serviceaccount/calico-kube-controllers created
serviceaccount/calico-node created
serviceaccount/calico-cni-plugin created
configmap/calico-config created
customresourcedefinition.apiextensions.k8s.io/bgpconfigurations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/bgpfilters.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/bgppeers.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/blockaffinities.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/caliconodestatuses.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/clusterinformations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/felixconfigurations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/globalnetworkpolicies.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/globalnetworksets.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/hostendpoints.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ipamblocks.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ipamconfigs.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ipamhandles.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ippools.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ipreservations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/kubecontrollersconfigurations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/networkpolicies.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/networksets.crd.projectcalico.org created
clusterrole.rbac.authorization.k8s.io/calico-kube-controllers created
clusterrole.rbac.authorization.k8s.io/calico-node created
clusterrole.rbac.authorization.k8s.io/calico-cni-plugin created
clusterrolebinding.rbac.authorization.k8s.io/calico-kube-controllers created
clusterrolebinding.rbac.authorization.k8s.io/calico-node created
clusterrolebinding.rbac.authorization.k8s.io/calico-cni-plugin created
daemonset.apps/calico-node created
deployment.apps/calico-kube-controllers created
linuxtechi@k8s-master-noble:~$
```

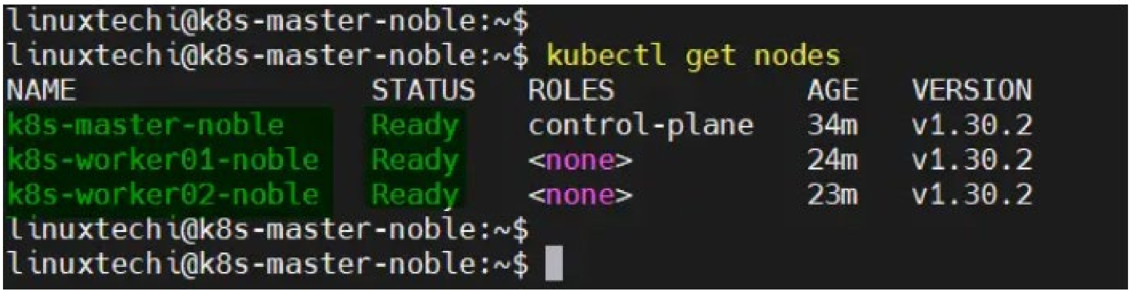
After the successful installation of calico, nodes status will change to Ready in a minute or two.

$ kubectl get pods -n kube-system



```
linuxtechi@k8s-master-noble:~$ kubectl get pods -n kube-system
NAME                                      READY   STATUS    RESTARTS   AGE
calico-kube-controllers-564985c589-bvwk2   1/1     Running   0          3m1s
calico-node-7t2l8                          1/1     Running   0          3m1s
calico-node-gqzlx                          1/1     Running   0          3m1s
calico-node-k5t6z                          1/1     Running   0          3m1s
coredns-7db6d8ff4d-6qxmm                   1/1     Running   0          32m
coredns-7db6d8ff4d-x6p82                   1/1     Running   0          32m
etcd-k8s-master-noble                      1/1     Running   0          32m
kube-apiserver-k8s-master-noble            1/1     Running   0          32m
kube-controller-manager-k8s-master-noble   1/1     Running   0          32m
kube-proxy-94ld8                           1/1     Running   0          22m
kube-proxy-9j76r                           1/1     Running   0          21m
kube-proxy-ttdqt                           1/1     Running   0          32m
kube-scheduler-k8s-master-noble            1/1     Running   0          32m
linuxtechi@k8s-master-noble:~$
```

$ kubectl get nodes



```
linuxtechi@k8s-master-noble:~$
linuxtechi@k8s-master-noble:~$ kubectl get nodes
NAME                STATUS   ROLES           AGE   VERSION
k8s-master-noble    Ready    control-plane   34m   v1.30.2
k8s-worker01-noble  Ready    <none>          24m   v1.30.2
k8s-worker02-noble  Ready    <none>          23m   v1.30.2
linuxtechi@k8s-master-noble:~$
linuxtechi@k8s-master-noble:~$
```

Output above confirms that nodes are in Ready state.

8) Test Kubernetes Installation

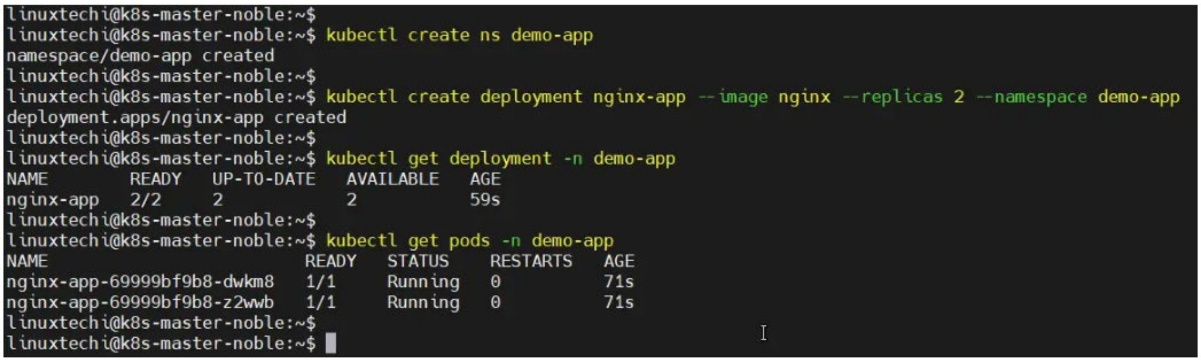
To test the Kubernetes installation, let's create nginx based deployment with replica count 2. Execute the following **kubectl** command from the master node.

$ kubectl create ns demo-app

$ kubectl create deployment nginx-app --image nginx --replicas 2 --namespace demo-app

$ kubectl get deployment -n demo-app

$ kubectl get pods -n demo-app

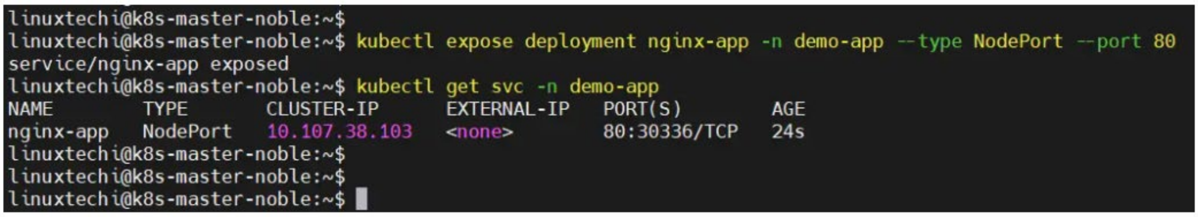


```
linuxtechi@k8s-master-noble:~$
linuxtechi@k8s-master-noble:~$ kubectl create ns demo-app
namespace/demo-app created
linuxtechi@k8s-master-noble:~$
linuxtechi@k8s-master-noble:~$ kubectl create deployment nginx-app --image nginx --replicas 2 --namespace demo-app
deployment.apps/nginx-app created
linuxtechi@k8s-master-noble:~$
linuxtechi@k8s-master-noble:~$ kubectl get deployment -n demo-app
NAME        READY   UP-TO-DATE   AVAILABLE   AGE
nginx-app   2/2     2            2           59s
linuxtechi@k8s-master-noble:~$
linuxtechi@k8s-master-noble:~$ kubectl get pods -n demo-app
NAME                         READY   STATUS    RESTARTS   AGE
nginx-app-69999bf9b8-dwkm8   1/1     Running   0          71s
nginx-app-69999bf9b8-z2wwb   1/1     Running   0          71s
linuxtechi@k8s-master-noble:~$
linuxtechi@k8s-master-noble:~$
```

Next expose this deployment using NodePort type, run

$ kubectl expose deployment nginx-app -n demo-app --type NodePort --port 80

$ kubectl get svc -n demo-app

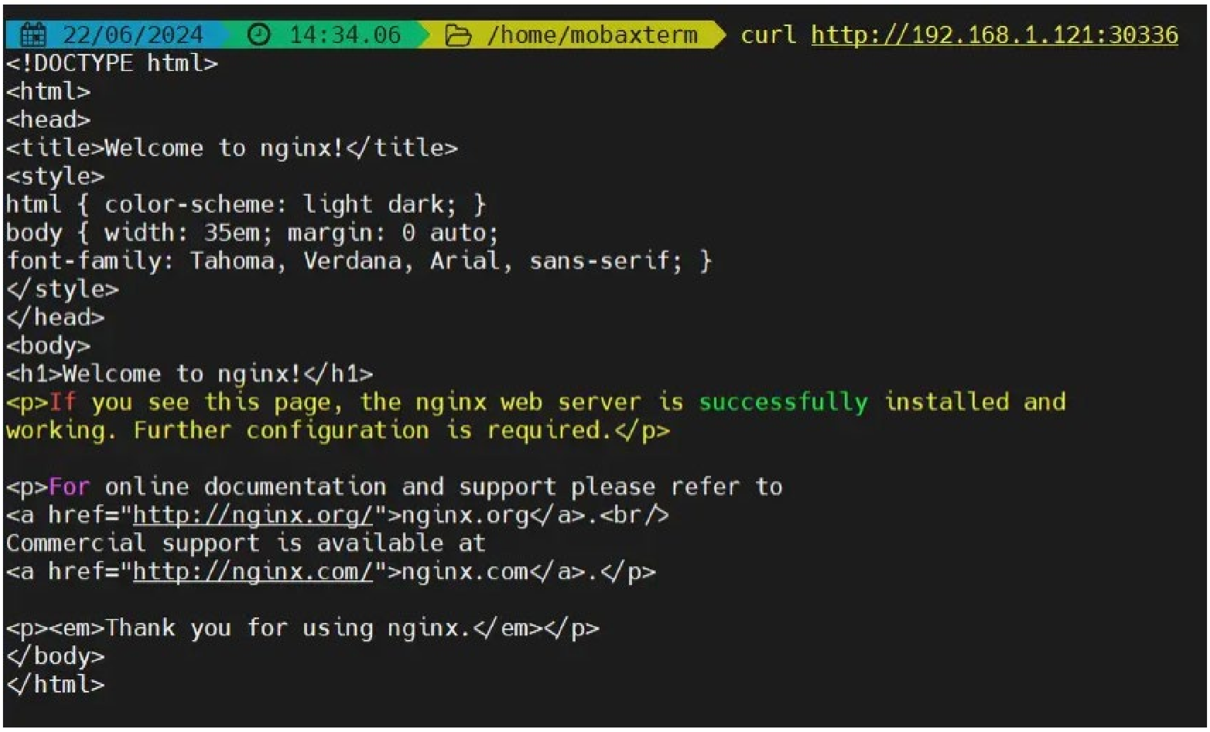


```
linuxtechi@k8s-master-noble:~$
linuxtechi@k8s-master-noble:~$ kubectl expose deployment nginx-app -n demo-app --type NodePort --port 80
service/nginx-app exposed
linuxtechi@k8s-master-noble:~$ kubectl get svc -n demo-app
NAME        TYPE       CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
nginx-app   NodePort   10.107.38.103   <none>        80:30336/TCP   24s
linuxtechi@k8s-master-noble:~$
linuxtechi@k8s-master-noble:~$
linuxtechi@k8s-master-noble:~$
```

Now try to access your application using **nodeport** as shown below

$ curl http://<Any-worker-IP>:30336

If want to check through browser
http://<Any-worker-IP>:30336



```
22/06/2024  14:34.06  /home/mobaxterm  curl http://192.168.1.121:30336
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

Great, output above confirms that we can access nginx based application outside of our Kubernetes cluster using the nodeport. This confirms that Kubernetes installation is successful.