Lesson 02 Demo 01

Creating and Configuring a Kubernetes Cluster

Objective: To set up a Kubernetes cluster by configuring hostnames, initializing the master node, joining worker nodes, and verifying the cluster's status

Tools required: kubeadm, kubectl, kubelet, and containerd

Prerequisites: Ensure you have executed **sudo kubeadm reset** on all machines. This action will clear any past configurations and prepare the machines for a new cluster setup.

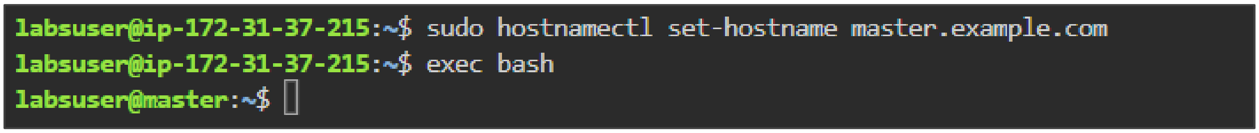
Steps to be followed:

1. Change the hostnames of all machines
2. Set up the master node
3. Join the worker nodes in the cluster

Step 1: Change the hostnames of all machines

1.1 On the master node, execute the following commands:

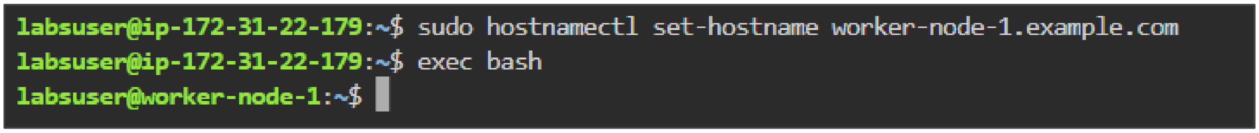
sudo hostnamectl set-hostname master.example.com
exec bash



```
labsuser@ip-172-31-37-215:~$ sudo hostnamectl set-hostname master.example.com
labsuser@ip-172-31-37-215:~$ exec bash
labsuser@master:~$ 
```

1.2 On worker1, execute the following commands:

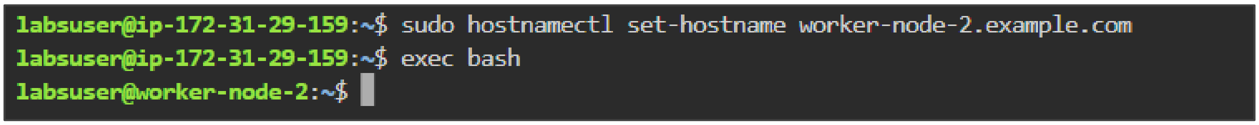
sudo hostnamectl set-hostname worker-node-1.example.com
exec bash



```
labsuser@ip-172-31-22-179:~$ sudo hostnamectl set-hostname worker-node-1.example.com
labsuser@ip-172-31-22-179:~$ exec bash
labsuser@worker-node-1:~$ 
```

1.3 On worker2, execute the following commands:

sudo hostnamectl set-hostname worker-node-2.example.com
exec bash

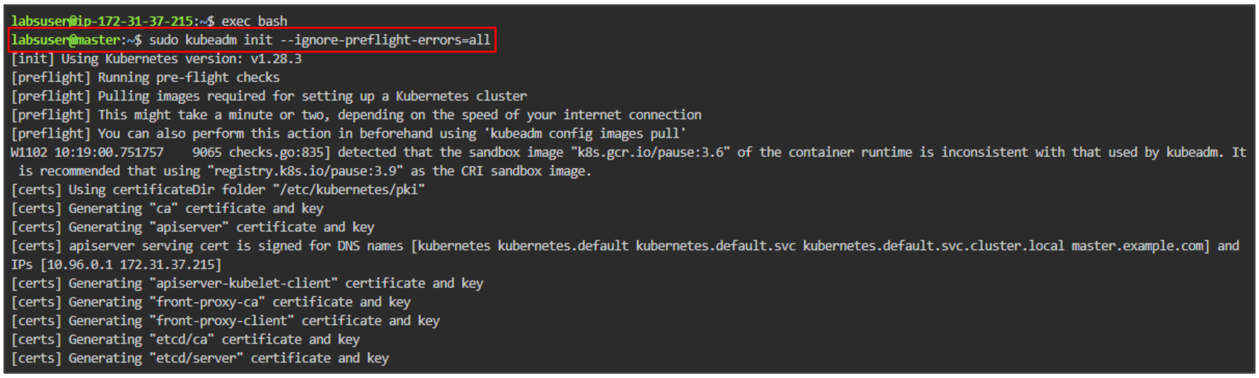


```
labsuser@ip-172-31-29-159:~$ sudo hostnamectl set-hostname worker-node-2.example.com
labsuser@ip-172-31-29-159:~$ exec bash
labsuser@worker-node-2:~$ 
```

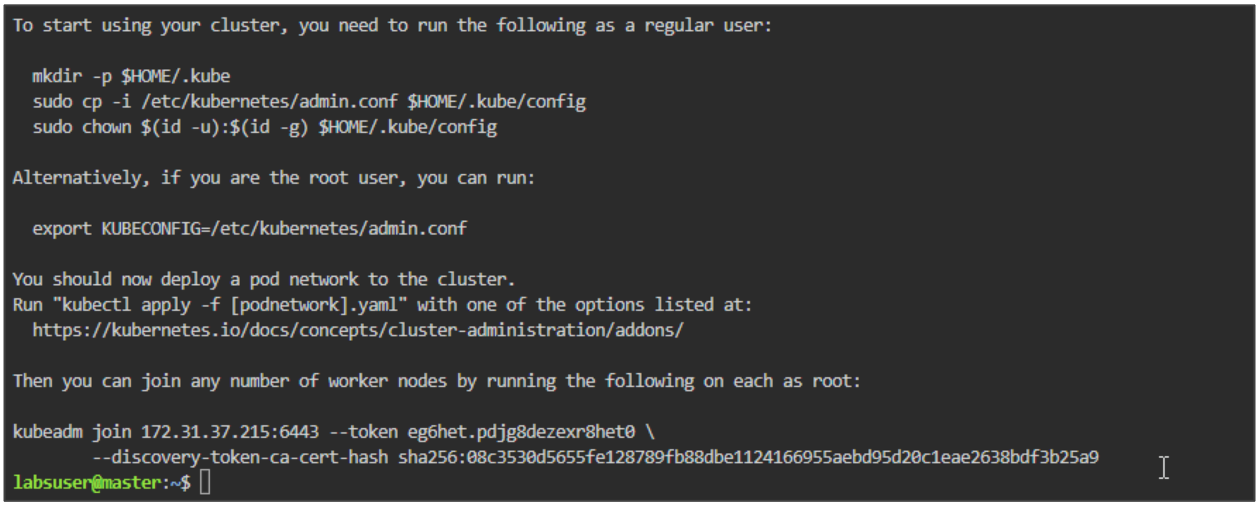
Step 2: Set up the master node

2.1 Initiate kubeadm by executing the following command:

sudo kubeadm init --ignore-preflight-errors=all



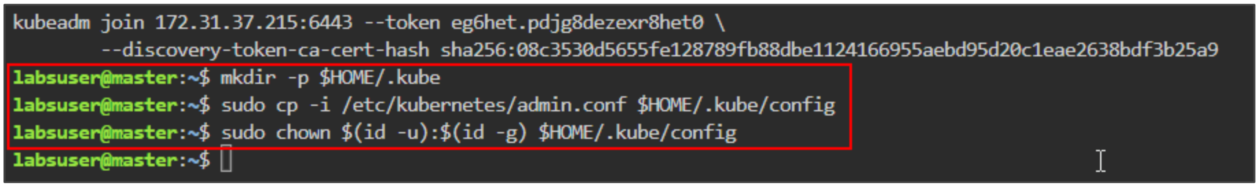
```
labsuser@ip-172-31-37-215:~$ exec bash
labsuser@master:~$ sudo kubeadm init --ignore-preflight-errors=all
[init] Using Kubernetes version: v1.28.3
[preflight] Running pre-flight checks
[preflight] Pulling images required for setting up a Kubernetes cluster
[preflight] This might take a minute or two, depending on the speed of your internet connection
[preflight] You can also perform this action in beforehand using 'kubeadm config images pull'
W1102 10:19:00.751757    9065 checks.go:835] detected that the sandbox image "k8s.gcr.io/pause:3.6" of the container runtime is inconsistent with that used by kubeadm. It
 is recommended that using "registry.k8s.io/pause:3.9" 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 [kubernetes kubernetes.default kubernetes.default.svc kubernetes.default.svc.cluster.local master.example.com] and
IPs [10.96.0.1 172.31.37.215]
[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
```



```
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 172.31.37.215:6443 --token eg6het.pdjg8dezexr8het0 \
        --discovery-token-ca-cert-hash sha256:08c3530d5655fe128789fb88dbe1124166955aebd95d20c1eae2638bdf3b25a9
labsuser@master:~$ 
```

2.2 Run the following commands to allow non-root users to access kubeadm:

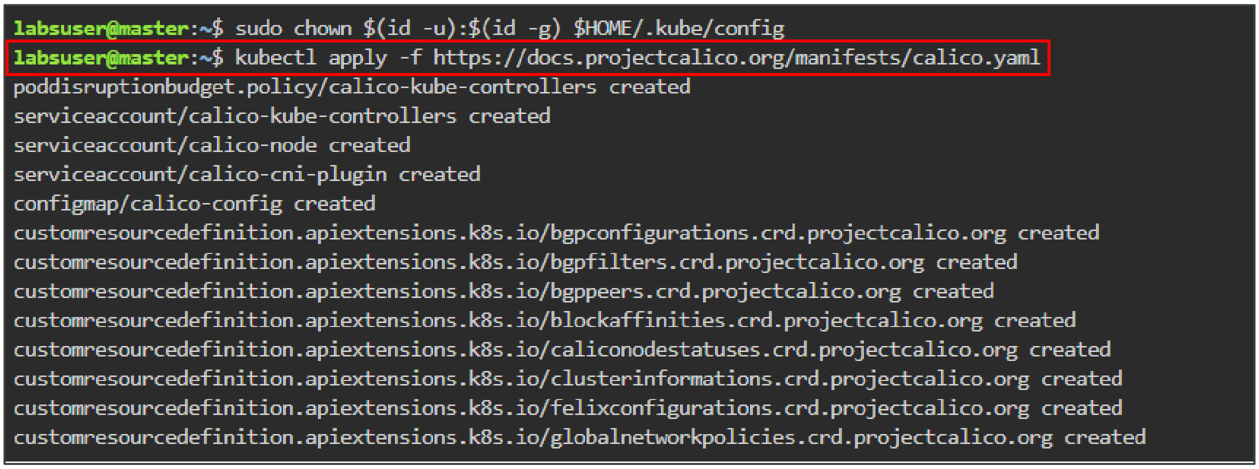
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config



```
kubeadm join 172.31.37.215:6443 --token eg6het.pdjg8dezexr8het0 \
        --discovery-token-ca-cert-hash sha256:08c3530d5655fe128789fb88dbe1124166955aebd95d20c1eae2638bdf3b25a9
labsuser@master:~$ mkdir -p $HOME/.kube
labsuser@master:~$ sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
labsuser@master:~$ sudo chown $(id -u):$(id -g) $HOME/.kube/config
labsuser@master:~$ 
```

2.3 Run the following command to deploy the weave network:

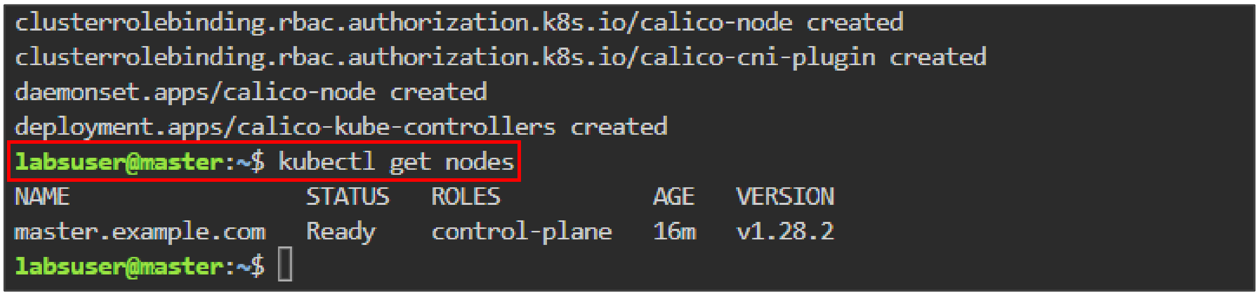
kubectl apply -f https://docs.projectcalico.org/manifests/calico.yaml



```
labsuser@master:~$ sudo chown $(id -u):$(id -g) $HOME/.kube/config
labsuser@master:~$ kubectl apply -f https://docs.projectcalico.org/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
```

2.4 To verify the master node's status, execute the following command:

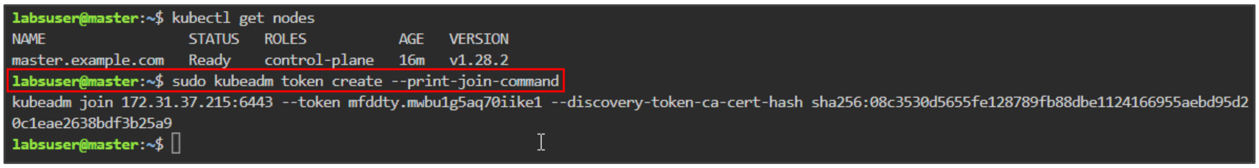
kubectl get nodes



```
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
labsuser@master:~$ kubectl get nodes
NAME                 STATUS   ROLES           AGE   VERSION
master.example.com   Ready    control-plane   16m   v1.28.2
labsuser@master:~$ 
```

You can see the master node is now ready and operational.

2.5 Run the following command to generate a command with a token for joining the worker nodes:
sudo kubeadm token create --print-join-command

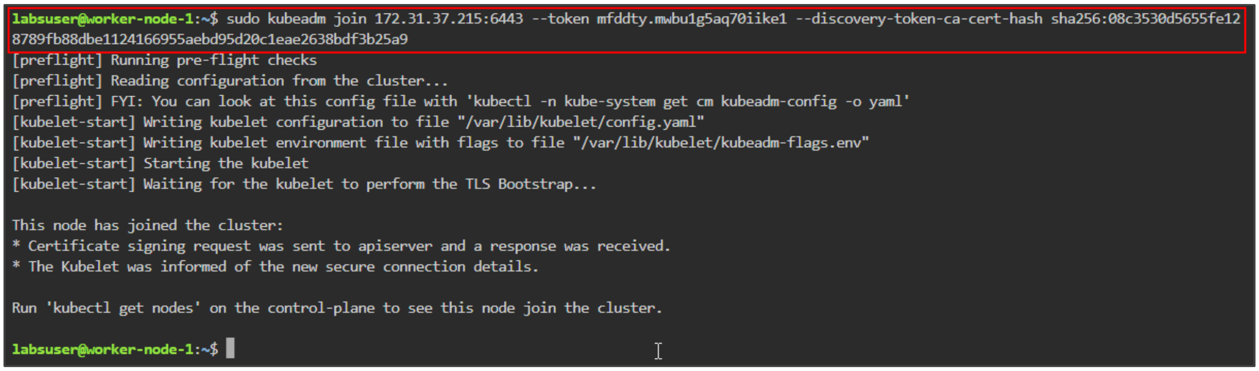


```
labsuser@master:~$ kubectl get nodes
NAME                 STATUS   ROLES           AGE   VERSION
master.example.com   Ready    control-plane   16m   v1.28.2
labsuser@master:~$ sudo kubeadm token create --print-join-command
kubeadm join 172.31.37.215:6443 --token mfddty.mwbu1g5aq70iike1 --discovery-token-ca-cert-hash sha256:08c3530d5655fe128789fb88dbe1124166955aebd95d2
0c1eae2638bdf3b25a9
labsuser@master:~$ 
```

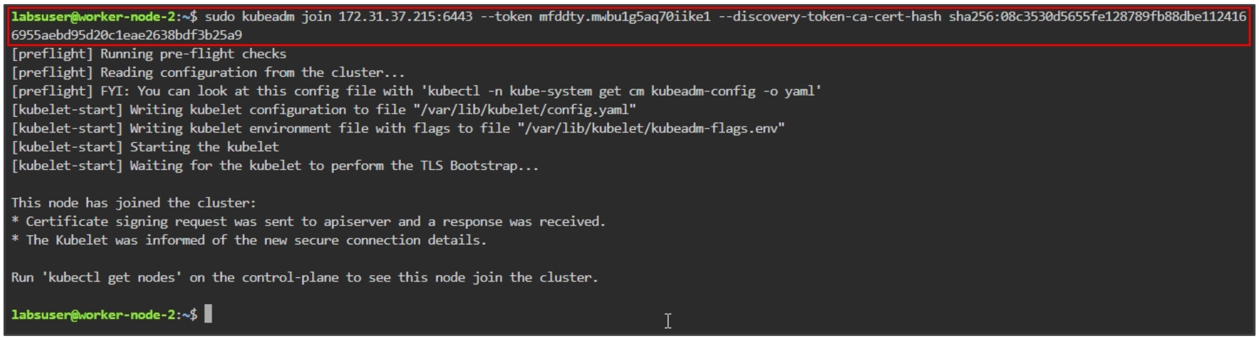
Note: Save the displayed **kubeadm join** command and token for later; you will need them to connect the worker nodes.

Step 3: Join the worker nodes in the cluster

3.1 Use the **kubeadm join** command (from step 2.5) on both worker nodes



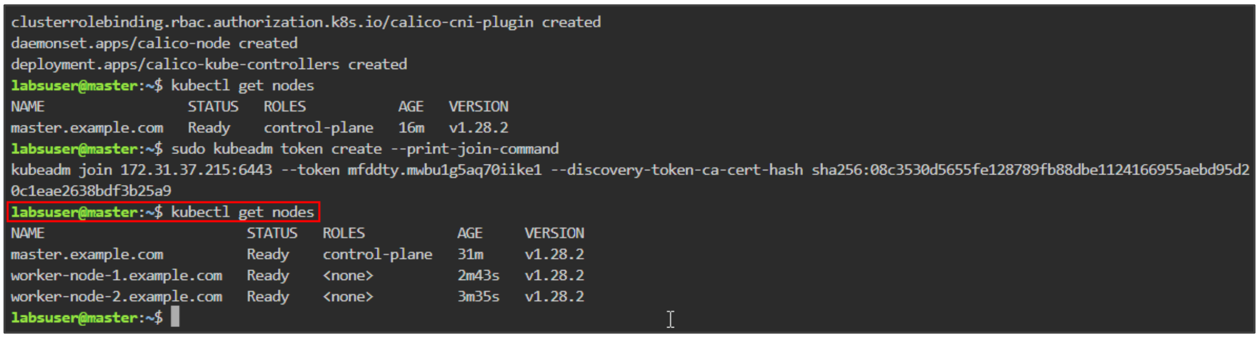
```
labsuser@worker-node-1:~$ sudo kubeadm join 172.31.37.215:6443 --token mfddty.mwbu1g5aq70iike1 --discovery-token-ca-cert-hash sha256:08c3530d5655fe12
8789fb88dbe1124166955aebd95d20c1eae2638bdf3b25a9
[preflight] Running pre-flight checks
[preflight] Reading configuration from the cluster...
[preflight] 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-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.

labsuser@worker-node-1:~$ 
```



```
labsuser@worker-node-2:~$ sudo kubeadm join 172.31.37.215:6443 --token mfddty.mwbu1g5aq70iike1 --discovery-token-ca-cert-hash sha256:08c3530d5655fe128789fb88dbe112416
6955aebd95d20c1eae2638bdf3b25a9
[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-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.

labsuser@worker-node-2:~$ 
```

Note: Ensure you use **sudo** before executing the command

3.2 Return to the master node and check if the worker nodes have joined by executing the following command:
kubectl get nodes



```
clusterrolebinding.rbac.authorization.k8s.io/calico-cni-plugin created
daemonset.apps/calico-node created
deployment.apps/calico-kube-controllers created
labsuser@master:~$ kubectl get nodes
NAME                 STATUS   ROLES           AGE   VERSION
master.example.com   Ready    control-plane   16m   v1.28.2
labsuser@master:~$ sudo kubeadm token create --print-join-command
kubeadm join 172.31.37.215:6443 --token mfddty.mwbu1g5aq70iike1 --discovery-token-ca-cert-hash sha256:08c3530d5655fe128789fb88dbe1124166955aebd95d2
0c1eae2638bdf3b25a9
labsuser@master:~$ kubectl get nodes
NAME                         STATUS   ROLES           AGE     VERSION
master.example.com           Ready    control-plane   31m     v1.28.2
worker-node-1.example.com    Ready    <none>          2m43s   v1.28.2
worker-node-2.example.com    Ready    <none>          3m35s   v1.28.2
labsuser@master:~$
```

Both worker nodes have been integrated into the cluster.

By following these steps, you have successfully set up and configured a Kubernetes cluster.