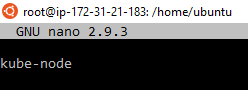
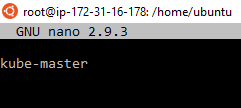
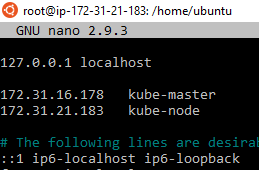
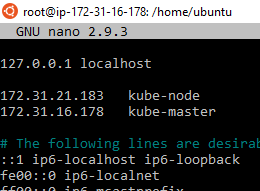
**naoKubernetes Cluster**

**Basic Setup for Both Master and Nodes:**

1. #sudo apt update
2. Turn Off swap space: #
3. # sudonano /etc/hostname – (enter hostname for master and node)



1. #sudonano /etc/hosts – (enter ip-address & hostname of both master and slave in both master/node)



### ****Setting Static IP Addresses****

Next, we will make the IP addresses used above, static for the VMs. We can do that by modifying the network interfaces file. Run the following command to open the file:

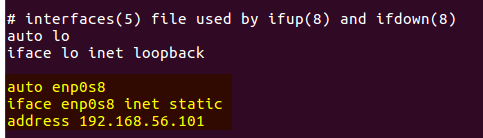
# sudonano /etc/network/interfaces

Now enter the following lines in the file.

auto enp0s8

iface enp0s8 inet static

address<*IP-Address-Of-VM*>



### ****Install OpenSSH-Server****

# sudo apt-get install openssh-server

### ****Install Docker****

Now we have to install Docker because Docker images will be used for managing the containers in the cluster

# sudosu

# apt-get update

# apt-get install -y docker.io

1. Next we have to install these 3 essential components for setting up Kubernetes environment: kubeadm, kubectl, and kubelet.

Run the following commands before installing the Kubernetes environment.

# apt-get update && apt-get install -y apt-transport-https curl

(Curl being an utility for downloading something from a link is helping the transport for data or download via the HTTP secure protocol (HTTPS) by wrapping unencrypted http in an encrypted layer known as Transport Security Layer (TSL)).

# curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | apt-key add -

(Download the Google Cloud public signing key so that authentic packages are downloaded)

# cat<<EOF >/etc/apt/sources.list.d/kubernetes.list

deb http://apt.kubernetes.io/ kubernetes-xenial main

EOF

(Add the Kubernetes apt repository)

# apt-get update

## ****Install kubeadm, KubeletAndKubectl****

Install the 3 essential components. **Kubelet** is the lowest level component in Kubernetes. It’s responsible for what’s running on an individual machine. ***Kuebadm*** is used for administrating the Kubernetes cluster. ***Kubectl*** is used for controlling the configurations on various nodes inside the cluster.

# apt-get install -y kubelet kubeadm kubectl

### ****Updating Kubernetes Configuration****

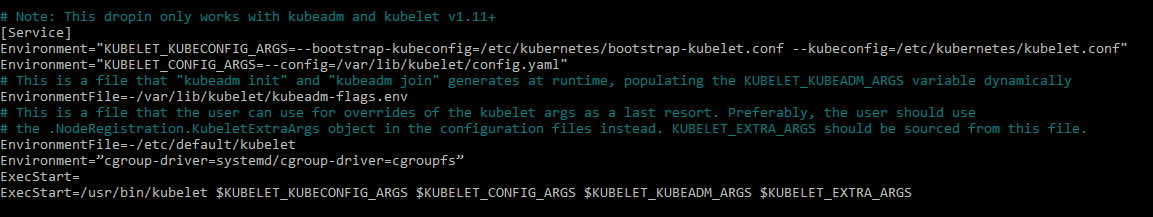
Next, we will change the configuration file of Kubernetes. Run the following command:

# nano /etc/systemd/system/kubelet.service.d/10-kubeadm.conf

This will open a text editor, enter the following line after the last “Environment Variable”:

Environment=”cgroup-driver=systemd/cgroup-driver=cgroupfs”

Environment="native.cgroupdriver=systemd”



## Cgroup drivers:

Control groups are used to constrain resources that are allocated to processes.

When [systemd](https://www.freedesktop.org/wiki/Software/systemd/) is chosen as the init system for a Linux distribution, the init process generates and consumes a root control group (cgroup) and acts as a cgroup manager. Systemd has a tight integration with cgroups and allocates a cgroup per systemd unit. It's possible to configure your container runtime and the kubelet to use cgroupfs. Using cgroupfs alongside systemd means that there will be two different cgroup managers.

A single cgroup manager simplifies the view of what resources are being allocated and will by default have a more consistent view of the available and in-use resources. When there are two cgroup managers on a system, you end up with two views of those resources. In the field, people have reported cases where nodes that are configured to use cgroupfs for the kubelet and Docker, but systemd for the rest of the processes, become unstable under resource pressure.

Changing the settings such that your container runtime and kubelet use systemd as the cgroup driver stabilized the system.

To configure this for Docker, set native.cgroupdriver=systemd.

**Caution:**

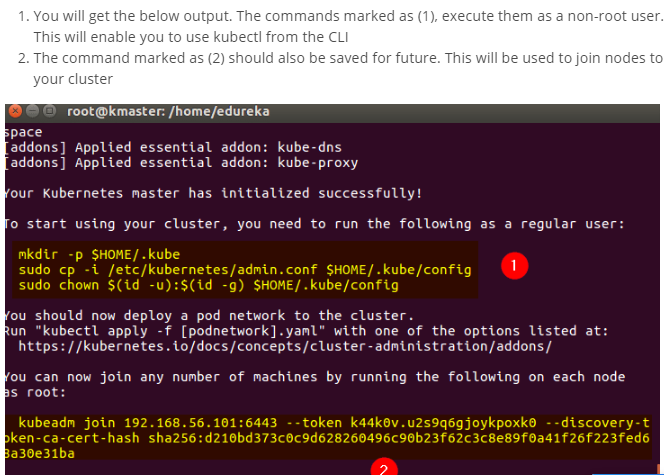
Changing the cgroup driver of a Node that has joined a cluster is strongly not recommended.  
If the kubelet has created Pods using the semantics of one cgroup driver, changing the container runtime to another cgroup driver can cause errors when trying to re-create the Pod sandbox for such existing Pods. Restarting the kubelet may not solve such errors.

If you have automation that makes it feasible, replace the node with another using the updated configuration, or reinstall it using automation.

## ****Set-up in the Kubernetes Master VM (kube-master):****

1. Install the Kubernetes Cluster by running the command below.

# kubeadm init --apiserver-advertise-address=<ip-address-of-kmaster-vm> --pod-network-cidr=192.168.0.0/16



Note: (192.168.0.0 is a reserved router/network range so we are using it as a test case, Make a record of the kubeadm join command that kubeadminit outputs. You need this command to [join nodes to your cluster](https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/create-cluster-kubeadm/#join-nodes).

The token is used for mutual authentication between the control-plane node and the joining nodes. The token included here is secret. Keep it safe, because anyone with this token can add authenticated nodes to your cluster. These tokens can be listed, created, and deleted with the kubeadm token command. See the [kubeadm reference guid](https://kubernetes.io/docs/reference/setup-tools/kubeadm/kubeadm-token/)

Token is secret and can be viewed from CP using :# kubeadm token list)

1. As mentioned before, run the commands from the above output as a non-root user

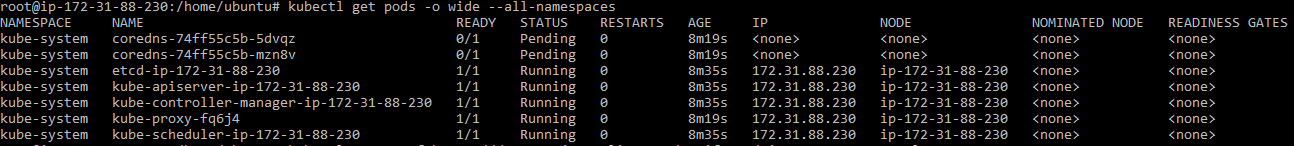
$ mkdir -p $HOME/.kube

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

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

1. To verify, if kubectl is working or not, run the following command:

$ kubectl get pods -o wide --all-namespaces



You will notice from the previous command, that all the pods are running except one: ‘kube-dns’. For resolving this we will install a pod network. To install the CALICO pod network, run the following command:

1. **Install Calico to create a pod network**
2. Install the Tigera Calico operator and custom resource definitions.

# kubectl create -f <https://docs.projectcalico.org/manifests/tigera-operator.yaml>

1. Install Calico by creating the necessary custom resource. For more information on configuration options available in this manifest, see [the installation reference](https://docs.projectcalico.org/reference/installation/api).

# kubectl create -f https://docs.projectcalico.org/manifests/custom-resources.yaml

1. Confirm that all of the pods are running with the following command.

# watchkubectl get pods -n calico-system

Wait until each pod has the STATUS of Running.

**Note**: The Tigera operator installs resources in the calico-system namespace. Other install methods may use the kube-system namespace instead.

1. Remove the taints on the master so that you can schedule pods on it.

# kubectl taint nodes --all node-role.kubernetes.io/master-

It should return the following.

node/<your-hostname> untainted

1. Confirm that you now have a node in your cluster with the following command.

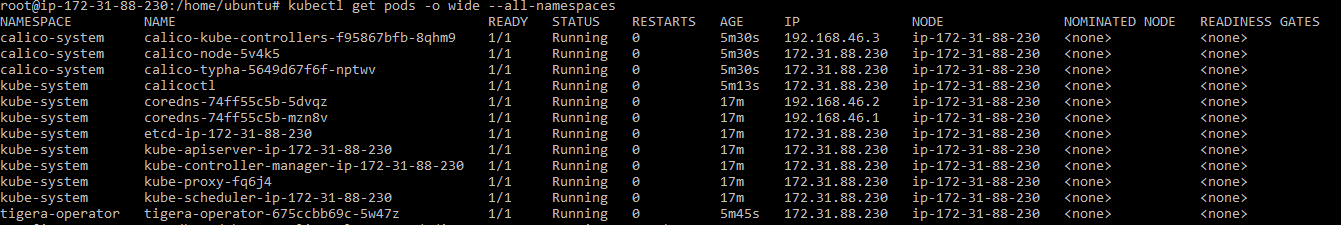
# kubectl get nodes -o wide

It should return something like the following.

NAME STATUS ROLES AGE VERSION INTERNAL-IP EXTERNAL-IP OS-IMAGE KERNEL-VERSION CONTAINER-RUNTIME

<your-hostname> Ready master 52m v1.12.2 10.128.0.28 <none> Ubuntu 18.04.1 LTS 4.15.0-1023-gcp docker://18.6.1

Congratulations! You now have a single-host Kubernetes cluster with Calico.



**Deploy Dashboard:**

1. The first thing to know about the web UI is that it can only be accessed using localhost address on the machine it runs on. This means we need to have an SSH tunnel to the server. For most OS, you can create an SSH tunnel using this command. Replace the <user> and <master\_public\_IP> with the relevant details to your Kubernetes cluster.



In case of the Cloud Cluster: I have modifies it as below.

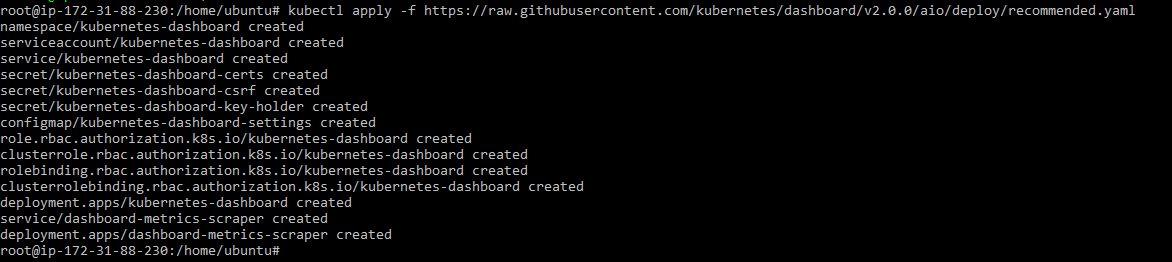
ssh -i \*.pem -L localhost:8001:127.0.0.1:8001 user@publicip

($ssh–I MYGREATLEARNING.pem -L localhost:8001:127.0.0.1:8001 ubuntu@Master Node Public-IP)

1. Deploy the Dashboard Service

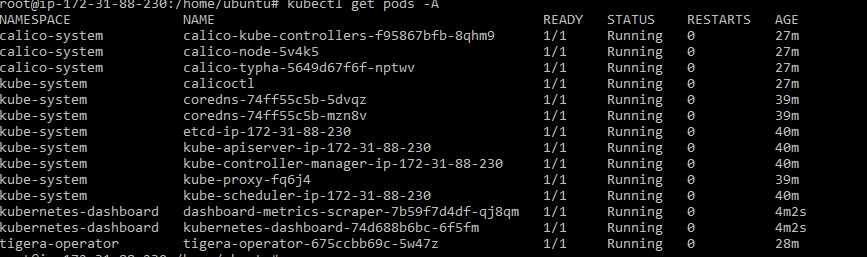
# kubectl apply -f https://raw.githubusercontent.com/kubernetes/dashboard/v2.0.0/aio/deploy/recommended.yaml

If your cluster is working correctly, you should see an output confirming the creation of a bunch of Kubernetes components like in the example below.



1. you should have two new pods running on your cluster.

# kubectl get pods–A



1. Creating User:

Admin User:

The Kubernetes dashboard supports a few ways to manage access control. In this example, we’ll be creating an admin user account with full privileges to modify the cluster and using tokens.

Start by making a new directory for the dashboard configuration files.

# mkdir ~/dashboard && cd ~/dashboard

1. Create the following configuration and save it as dashboard-admin.yaml file. Note that indentation matters in the YAML files which should use two spaces in a regular text editor.

# nano dashboard-admin.yaml

apiVersion: v1

kind: ServiceAccount

metadata:

name: admin-user

namespace: kubernetes-dashboard

---

apiVersion: rbac.authorization.k8s.io/v1

kind: ClusterRoleBinding

metadata:

name: admin-user

roleRef:

apiGroup: rbac.authorization.k8s.io

kind: ClusterRole

name: cluster-admin

subjects:

- kind: ServiceAccount

name: admin-user

namespace: kubernetes-dashboard

1. Then deploy the admin user role with the next command.

# kubectl apply -f dashboard-admin.yaml



1. Based on the troubleshooting for the Blank Dashboard below commands are helping:

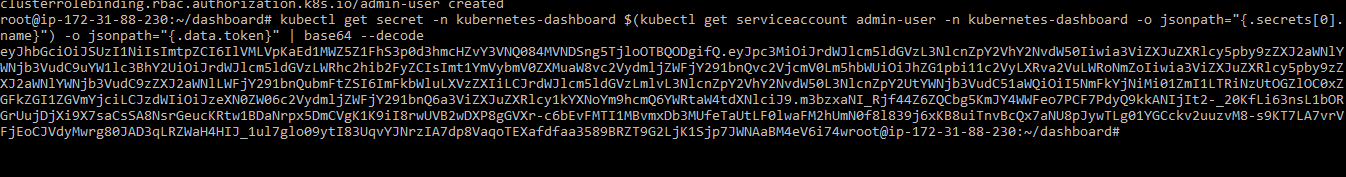
# kubectl delete clusterrolebindingkubernetes-dashboard

# kubectl create clusterrolebindingkubernetes-dashboard --clusterrole=cluster-admin --serviceaccount=kube-system:kubernetes-dashboard --user=clusterUser

1. Using this method doesn’t require setting up or memorising passwords, instead, accessing the dashboard will require a token.

Get the admin token using the command below.

root@ip-172-31-88-230:~/dashboard# kubectl get secret -n kubernetes-dashboard $(kubectl get serviceaccount admin-user -n kubernetes-dashboard -o jsonpath="{.secrets[0].name}") -o jsonpath="{.data.token}" | base64 --decode



## 9. Accessing the dashboard

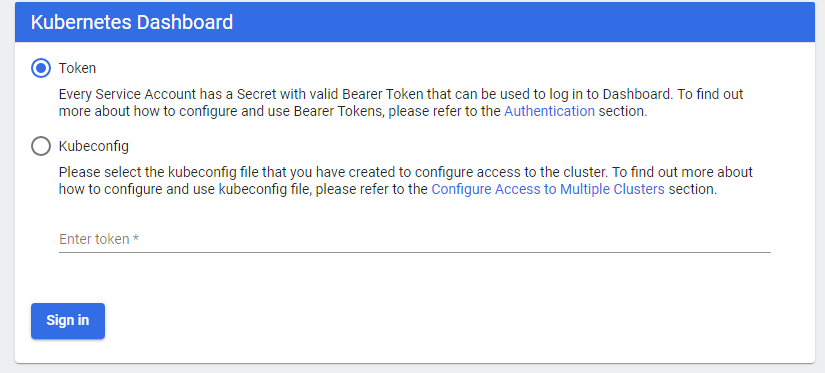
We’ve now deployed the dashboard and created user accounts for it. Next, we can get started managing the Kubernetes cluster itself.

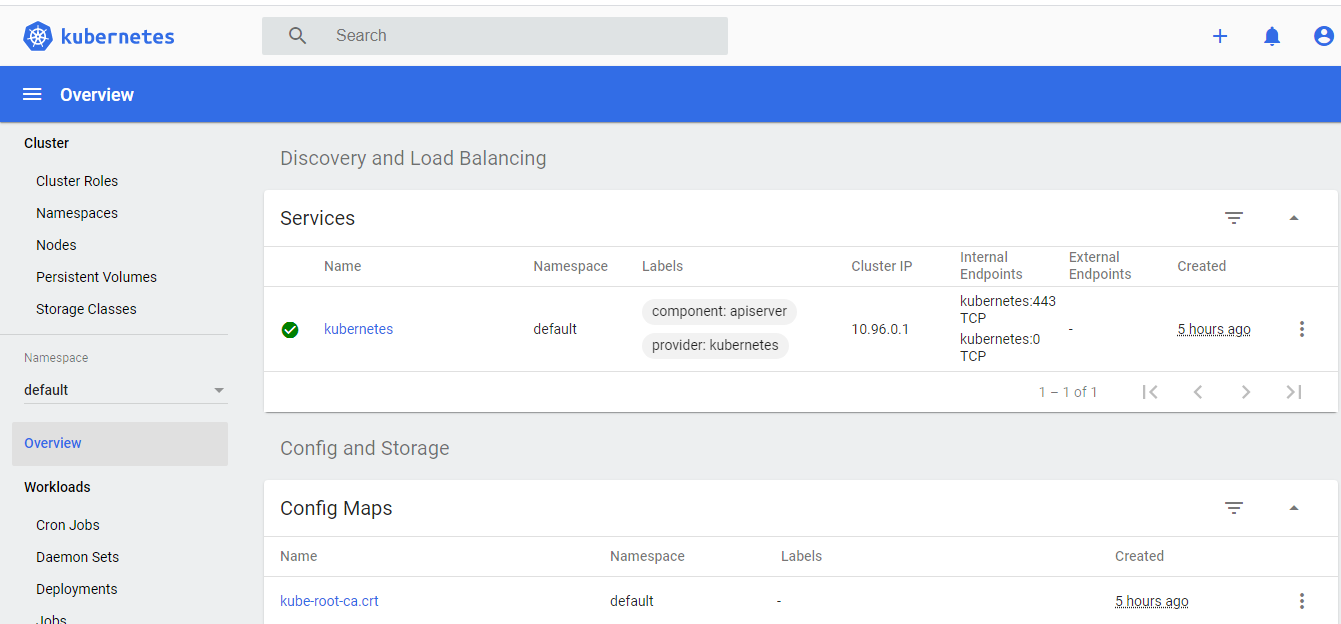
root@ip-172-31-88-230:~/dashboard# kubectl proxy

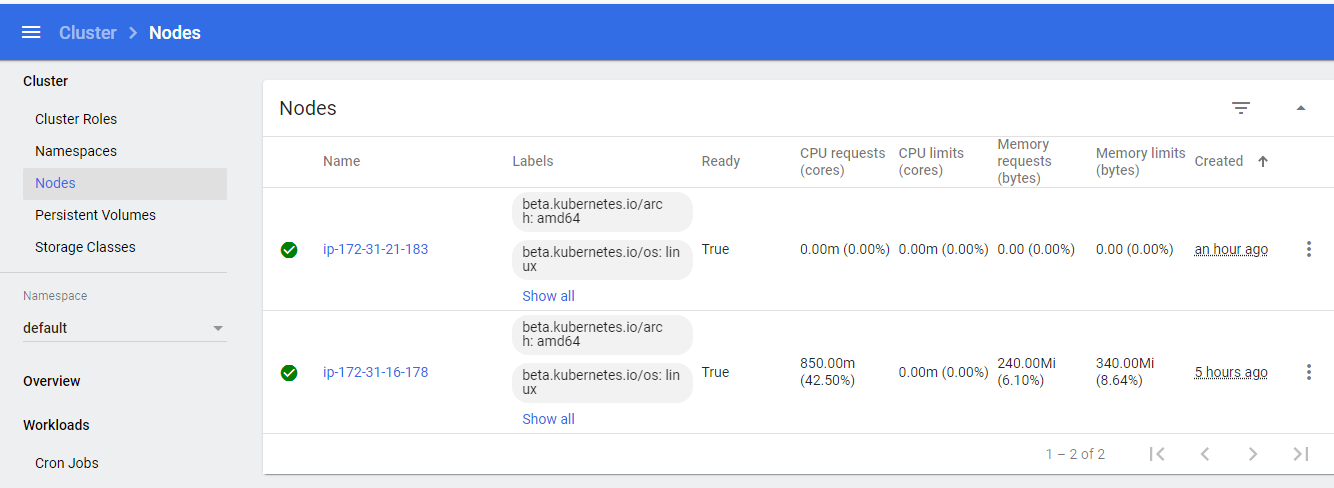


10.Access Dashboard: Now, assuming that we have already established an SSH tunnel binding to the localhost port 8001 at both ends, open a browser to the link below.

http://localhost:8001/api/v1/namespaces/kubernetes-dashboard/services/https:kubernetes-dashboard:/proxy/



eyJhbGciOiJSUzI1NiIsImtpZCI6IkhiQ1pFV0pPVWhiNDNoRFZ3T2ZrYTE1VmMtLWVUWkplM09iVDZyQTluTlkifQ..cdVJUrYT854FlC60Wdvlb2afbxrysYTVvzzozJL51xsnWwdPFNflO-r9yUAo3HYX55Xwz9RUOEotZWXM0zBrpLD8MbdX3v0PAlZaMNo\_pBn6hekM\_pUHSSfJRrT4ISZNFEOMxNC9qwlDAG9m9lBQYvCmBOUxU-p6E0117kbhoCciM3A6Rf6gc6wWUwaUkW4jNXcOFS0Q4rLnj1vi\_4EQWX-AIa0LyjUYO5Tp-tHqHztEl08HWws\_LQJI-EjvnVWl7F3dlZkQCgnig0MKEcXe1fewW0sGZJPitwCU6lCgZrdWOJamgHVE5CMXgxd6HtqUxLJAYi\_MUxA7ygjQdT2bHA

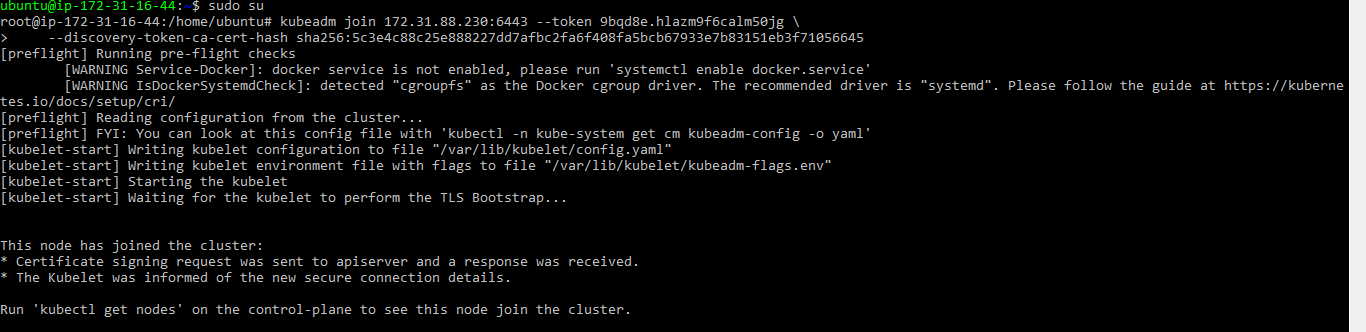


**Node Join: Login on the Node with root and node join command**

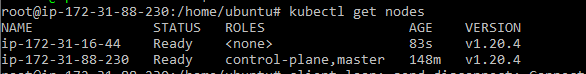
# kubeadm join MASTERIP:6443 --token 9bqd8e.hlazm9f6calm50jg \ --discovery-token-ca-cert-hash sha256:5c3e4c88c25e888227dd7afbc2fa6f408fa5bcb67933e7b83151eb3f71056645

e.g.

# kubeadm join 172.31.88.230:6443 --token 9bqd8e.hlazm9f6calm50jg \ --discovery-token-ca-cert-hash sha256:5c3e4c88c25e888227dd7afbc2fa6f408fa5bcb67933e7b83151eb3f71056645



Verify Node Status on the Master:



Problems post EC2 Stop:

Control Plane Master Node and Node bot are showing Not Ready.

How to resolve:

Master: Login and follow below.

1. # kubeadm reset
2. Delete the config manually from $HOME/.kube/config file
3. Repeat the steps from - #kubeadminit --apiserver-advertise-address=Master Node Ip --pod-network-cidr=192.168.0.0/16

Node: Login and follow below

1. # kubeadm reset
2. Delete the config manually from $HOME/.kube/config file
3. Rejoin

In case of API Version error while running deployment kubectl apply -f app-v1.yaml

then check the API version and update the YAML file

e.g. **extensions**/**v1beta1 was updated as apps/v1 in the** app-v1.yaml

kubectlapi-resources | grep deployment



kubectl convert -f ./my-deployment.yaml --output-version apps/v1

Check RBAC status:

$kubectl cluster-info dump | grep authorization-mode



$kubectlapi-versions

$ps -aef | grep -i apiserver

It will show authorization mode as RBAC

$kubectl cluster-info

$kubectl describe node

Deployment:

Service info:

kubectl get svc “service-name”

kubectl describe svc “service-name”

(Service IP is completely virtual)

kubectl get ep“service-name”

curl<CLUSTER-IP>:<PORT>

kubectl apply –f “.yaml”

kubectl delete –f “.yaml”

Pod Info:

kubectl get pods

kubectl describe pods

To see the ReplicaSet (rs) created by the Deployment, run

 kubectl get rs

kubectl get pods --show-labels

### DNS

Kubernetes offers a DNS cluster addon Service that automatically assigns dns names to other Services. You can check if it's running on your cluster:

Here we use the CoreDNS cluster addon (application name kube-dns), so you can talk to the Service from any pod in your cluster using standard methods (e.g. gethostbyname()).

kubectl get services kube-dns --namespace=kube-system

nslookup my-nginx