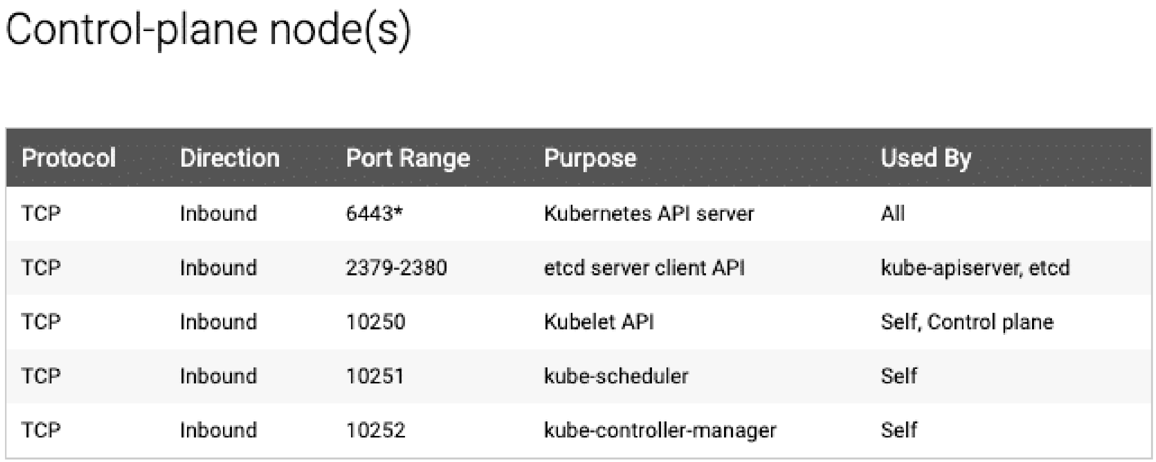
Kubeadm Setup

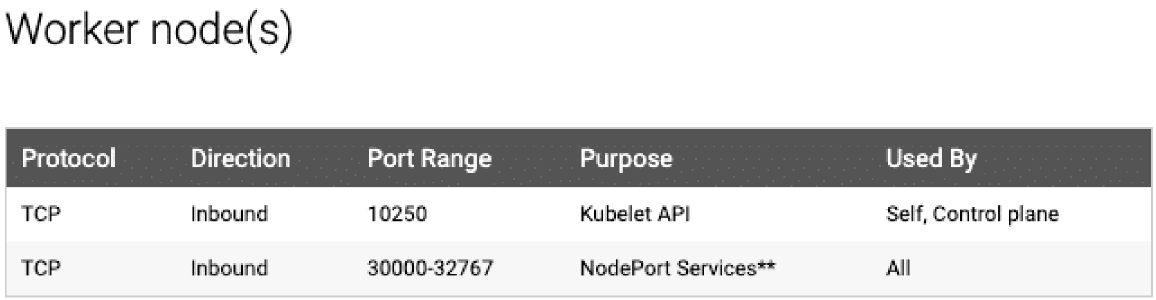
Following are the prerequisites for Kubeadm Kubernetes cluster setup

- Minimum two **Ubuntu** nodes [One master and one worker node]. You can have more worker nodes as per your requirement.
- The master node should have a minimum of **2 vCPU and 2GB RAM**.
- For the worker nodes, a minimum **of 1vCPU and 2 GB RAM** is recommended.
- 10.X.X.X/X network range with static IPs for master and worker nodes. We will be using the 192 series as the pod network range that will be used by the Calico network plugin. Make sure the Node IP range and pod IP range don't overlap.



Control-plane node(s)

Protocol	Direction	Port Range	Purpose	Used By
TCP	Inbound	6443*	Kubernetes API server	All
TCP	Inbound	2379-2380	etcd server client API	kube-apiserver, etcd
TCP	Inbound	10250	Kubelet API	Self, Control plane
TCP	Inbound	10251	kube-scheduler	Self
TCP	Inbound	10252	kube-controller-manager	Self



Worker node(s)

Protocol	Direction	Port Range	Purpose	Used By
TCP	Inbound	10250	Kubelet API	Self, Control plane
TCP	Inbound	30000-32767	NodePort Services**	All

1. Update the apt package index and install packages needed to use the Kubernetes apt repository:

```
$ sudo apt-get update -y

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

2. Execute the following commands for IPtables to see bridged traffic.

```
ubuntu@ip-172-31-5-3:~$ cat <<EOF | sudo tee /etc/modules-load.d/k8s.conf

br_netfilter

EOF

$ cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf

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

EOF
```

```
$ sudo sysctl --system
```

3. Disable swap on all the Nodes

   For kubeadm to work properly, you need to disable swap on all the nodes using the following command.

```
$ sudo swapoff -a

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

4. Install Docker Container Runtime On All The Nodes The basic requirement for a Kubernetes cluster is a container runtime. You can have any one of the following container runtimes.

   - containerd
   - CRI-O
   - Docker

   We will be using Docker for this setup.

   As a first step, we need to install Docker on all the nodes. Execute the following commands on all the nodes.

   Install the required packages for Docker.

```
$ sudo apt-get update -y

$ sudo apt-get install -y \

  apt-transport-https \

  ca-certificates \

  curl \

  gnupg \

  lsb-release
```

5. Add the Docker GPG key and apt repository.

```
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o
/usr/share/keyrings/docker-archive-keyring.gpg
```

```
$ echo \
  "deb [arch=amd64 signed-by=/usr/share/keyrings/docker-archive-keyring.gpg]
https://download.docker.com/linux/ubuntu \
  $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

6. Install the Docker community edition.

```
$ sudo apt-get update -y
$ sudo apt-get install docker-ce docker-ce-cli containerd.io -y
```

7. Add the docker daemon configurations to use systemd as the cgroup driver.

```
cat <<EOF | sudo tee /etc/docker/daemon.json
{
  "exec-opts": ["native.cgroupdriver=systemd"],
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "100m"
  },
  "storage-driver": "overlay2"
}
EOF
```

Start and enable the docker service.

```
$ sudo systemctl enable docker

$ sudo systemctl daemon-reload

$ sudo systemctl restart docker
```

8. Install Kubeadm & Kubelet & Kubectl on all Nodes

   Install the required dependencies.

```
$ sudo apt-get update
$ sudo curl -fsSLo /usr/share/keyrings/kubernetes-archive-keyring.gpg
https://packages.cloud.google.com/apt/doc/apt-key.gpg
```

9. Add the GPG key and apt repository.

```
$ echo "deb [signed-by=/usr/share/keyrings/kubernetes-archive-keyring.gpg]
https://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee
/etc/apt/sources.list.d/kubernetes.list
```

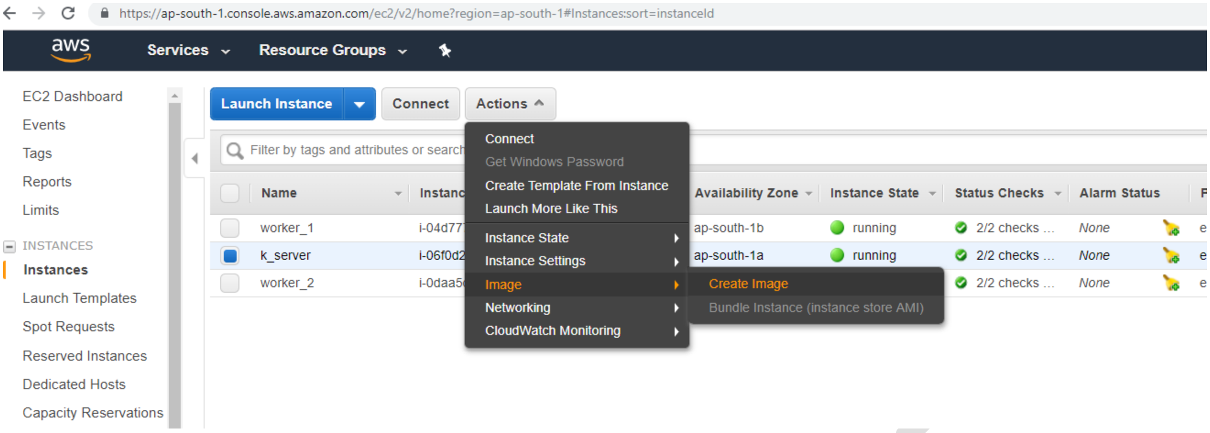
10. Update apt and install kubelet, kubeadm, and kubectl.

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

$ sudo  rm /etc/containerd/config.toml
$ sudo systemctl restart containerd
```

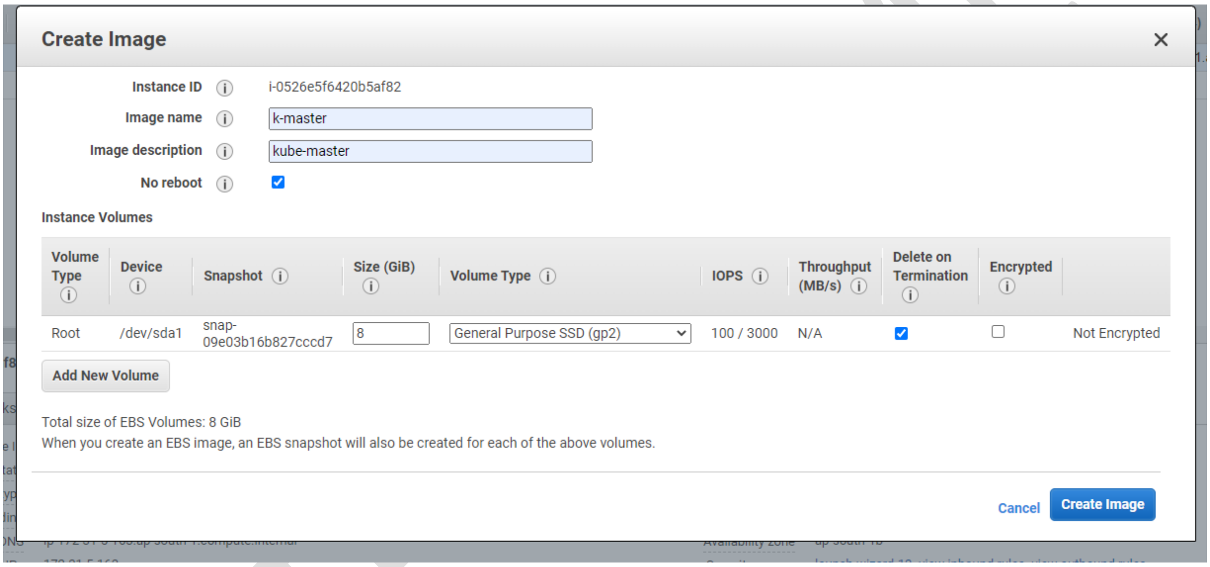
Now we have all the required utilities and tools for configuring Kubernetes components using kubeadm.

11. create a image After successfully install we need make it as a AMI image.

    Goto aws console select which server you install all the above steps goto ACTION -> IMAGE -> CREATE IMAGE



	Name	Instanc	Availability Zone	Instance State	Status Checks	Alarm Status	F
□	worker_1	i-04d77	ap-south-1b	running	2/2 checks ...	None	e
■	k_server	i-06f0d2	ap-south-1a	running	2/2 checks ...	None	e
□	worker_2	i-0daa5			2/2 checks ...	None	e



Instance ID i-0526e5f6420b5af82

Image name k-master

Image description kube-master

No reboot ☑

Instance Volumes

Volume Type	Device	Snapshot	Size (GiB)	Volume Type	IOPS	Throughput (MB/s)	Delete on Termination	Encrypted
Root	/dev/sda1	snap-09e03b16b827cccd7	8	General Purpose SSD (gp2)	100 / 3000	N/A	☑	□ Not Encrypted

Total size of EBS Volumes: 8 GiB
When you create an EBS image, an EBS snapshot will also be created for each of the above volumes.

12. Add hold to the packages to prevent upgrades.

```
$ sudo apt-mark hold kubelet kubeadm kubectl
```

13. Initialize Kubeadm On Master Node To Setup Control Plane

Execute the commands in this section only on the master node.

First, set two environment variables. Replace 172.31.8.227 with the IP of your master node.

```
$ export IPADDR="172.31.12.191"

$ export NODENAME=$(hostname -s)
```

14. Configuring a cgroup driver

https://kubernetes.io/docs/tasks/administer-cluster/kubeadm/configure-cgroup-driver/

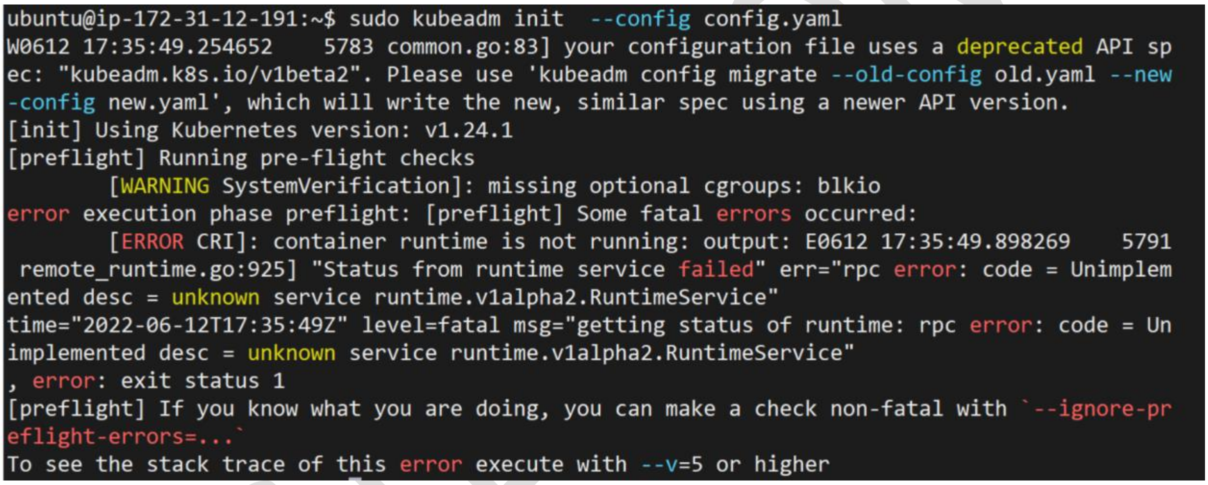
```
$ vi config.yaml

apiVersion: kubelet.config.k8s.io/v1beta1
```

```
kind: KubeletConfiguration

cgroupDriver: containerd

---

apiVersion: kubeadm.k8s.io/v1beta3

kind: ClusterConfiguration

networking:

  podSubnet: "10.244.0.0/16"
```

15. Now, initialize the master node control plane configurations using the following kubeadm command.

```
$ sudo kubeadm init  --config config.yaml
```

If you get below error



```
ubuntu@ip-172-31-12-191:~$ sudo kubeadm init  --config config.yaml
W0612 17:35:49.254652    5783 common.go:83] your configuration file uses a deprecated API sp
ec: "kubeadm.k8s.io/v1beta2". Please use 'kubeadm config migrate --old-config old.yaml --new
-config new.yaml', which will write the new, similar spec using a newer API version.
[init] Using Kubernetes version: v1.24.1
[preflight] Running pre-flight checks
        [WARNING SystemVerification]: missing optional cgroups: blkio
error execution phase preflight: [preflight] Some fatal errors occurred:
        [ERROR CRI]: container runtime is not running: output: E0612 17:35:49.898269    5791
 remote_runtime.go:925] "Status from runtime service failed" err="rpc error: code = Unimplem
ented desc = unknown service runtime.v1alpha2.RuntimeService"
time="2022-06-12T17:35:49Z" level=fatal msg="getting status of runtime: rpc error: code = Un
implemented desc = unknown service runtime.v1alpha2.RuntimeService"
, error: exit status 1
[preflight] If you know what you are doing, you can make a check non-fatal with `--ignore-pr
eflight-errors=...`
To see the stack trace of this error execute with --v=5 or higher
```

Execute below commands ,

```
$ sudo  rm /etc/containerd/config.toml

$ sudo systemctl restart containerd
```

And then initialize the kubeadm

```
$ sudo kubeadm init  --config config.yaml
```

You will get below output

```
$ sudo kubeadm init  --config config.yaml
```

W0612 17:40:08.635822   6116 common.go:83] your configuration file uses a deprecated API spec: "kubeadm.k8s.io/v1beta2". Please use 'kubeadm config migrate --old-config old.yaml --new-config new.yaml', which will write the new, similar spec using a newer API version.

[init] Using Kubernetes version: v1.24.1

[

[bootstrap-token] Creating the "cluster-info" ConfigMap in the "kube-public" namespace

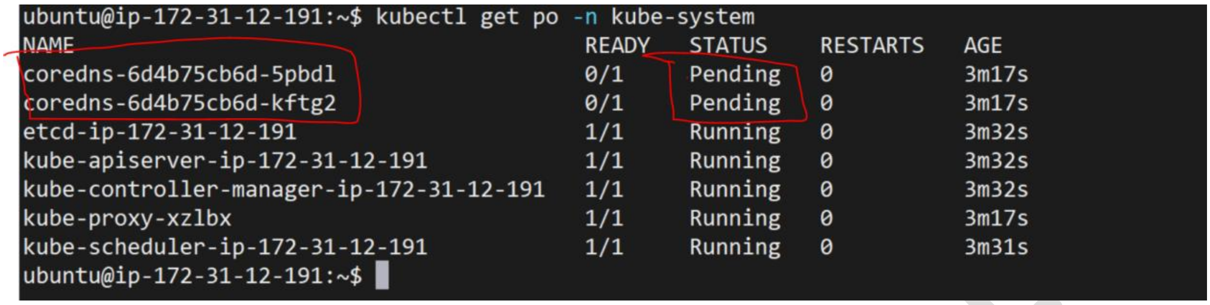
```
[kubelet-finalize] Updating "/etc/kubernetes/kubelet.conf" to point to a rotatable kubelet client certificate and key

[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/


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


kubeadm join 172.31.12.191:6443 --token lpm1gt.dny774qp3f49hffr \

        --discovery-token-ca-cert-hash sha256:aa95ab7da30d88bd611c1d4de2c9e14a8edbac26f846d0ec801e64d61fbda25f
```

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

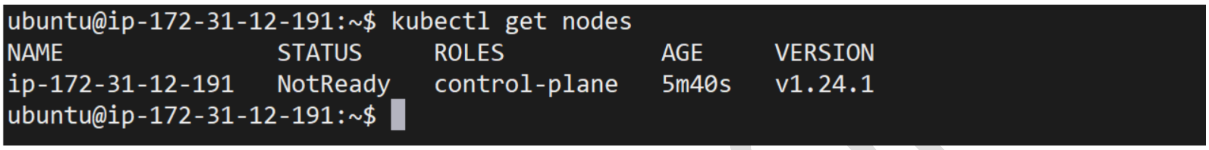
```
$ mkdir -p $HOME/.kube

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

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

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



```
ubuntu@ip-172-31-12-191:~$ kubectl get po -n kube-system
NAME                                      READY   STATUS    RESTARTS   AGE
coredns-6d4b75cb6d-5pbdl                  0/1     Pending   0          3m17s
coredns-6d4b75cb6d-kftg2                  0/1     Pending   0          3m17s
etcd-ip-172-31-12-191                     1/1     Running   0          3m32s
kube-apiserver-ip-172-31-12-191           1/1     Running   0          3m32s
kube-controller-manager-ip-172-31-12-191  1/1     Running   0          3m32s
kube-proxy-xzlbx                          1/1     Running   0          3m17s
kube-scheduler-ip-172-31-12-191           1/1     Running   0          3m31s
ubuntu@ip-172-31-12-191:~$
```



```
ubuntu@ip-172-31-12-191:~$ kubectl get nodes
NAME               STATUS     ROLES           AGE     VERSION
ip-172-31-12-191   NotReady   control-plane   5m40s   v1.24.1
ubuntu@ip-172-31-12-191:~$
```

You should see the following output. You will see the two Coredns pods in a pending state. It is the expected behavior. Once we install the network plugin, it will be in a running.

13. Installing a CNI Network.

https://kubernetes.io/docs/concepts/cluster-administration/addons/

A network is needed to enable the pods to communicate with each other. WEAVE CNI plugin is the network plugin used here.

switch the root user and run. (sudo su -)

ubuntu@ip-172-31-39-7:~$ sudo su -

root@ip-172-31-39-7:~# sysctl net.bridge.bridge-nf-call-iptables=1

switch user to Ubuntu

root@ip-172-31-39-7:~# su ubuntu

ubuntu@ip-172-31-39-7:~$ export kubever=$(kubectl version | base64 | tr -d '\n')

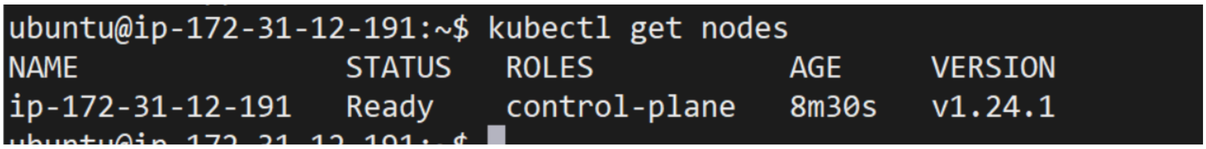
ubuntu@ip-172-31-39-7:~$ kubectl apply -f "https://cloud.weave.works/k8s/net?k8s-version=$kubever" (not working)

or

kubectl apply -f https://github.com/weaveworks/weave/releases/download/v2.8.1/weave-daemonset-k8s.yaml

Now that the CNI network has been created, give it a minute or 2 and test again. The result is as shown.

ubuntu@ip-172-31-39-7:~$ kubectl get nodes

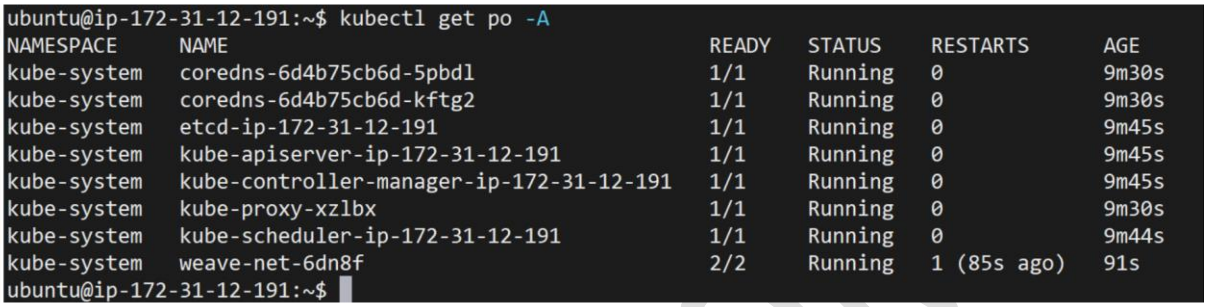


```
ubuntu@ip-172-31-12-191:~$ kubectl get nodes
NAME                STATUS   ROLES           AGE     VERSION
ip-172-31-12-191    Ready    control-plane   8m30s   v1.24.1
ubuntu@ip-172-31-12-191:~$
```

Now master node is ready and get the all pods and see the coredns pods are running or not

$ kubectl get po -n kube-system

$ kubectl get po -A



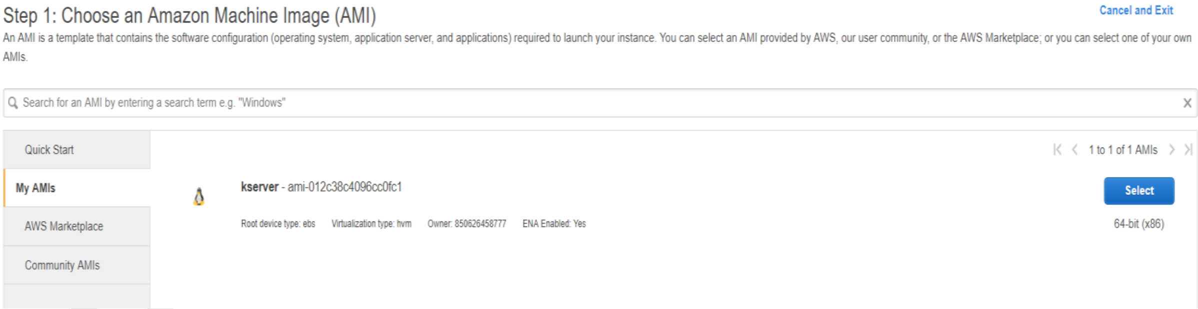
```
ubuntu@ip-172-31-12-191:~$ kubectl get po -A
NAMESPACE     NAME                                      READY   STATUS    RESTARTS      AGE
kube-system   coredns-6d4b75cb6d-5pbdl                  1/1     Running   0             9m30s
kube-system   coredns-6d4b75cb6d-kftg2                  1/1     Running   0             9m30s
kube-system   etcd-ip-172-31-12-191                     1/1     Running   0             9m45s
kube-system   kube-apiserver-ip-172-31-12-191           1/1     Running   0             9m45s
kube-system   kube-controller-manager-ip-172-31-12-191  1/1     Running   0             9m45s
kube-system   kube-proxy-xzlbx                          1/1     Running   0             9m30s
kube-system   kube-scheduler-ip-172-31-12-191           1/1     Running   0             9m44s
kube-system   weave-net-6dn8f                           2/2     Running   1 (85s ago)   91s
ubuntu@ip-172-31-12-191:~$
```

Here the CNI weave network has been installed and as a result the master node is now showing ‘READY’ and kube-dns is now showing ‘Running’ instead of pending. You will also notice the creation of a weave container weave-net-6dn8f listed above.

Now lets Creating the Kubernetes Slave Nodes

Create an instance using the AMI that was created above. For test purposes, its OK to choose t2. micro image.

Go to aws console launch instance and click left plane MY AMIs and select.



Step 1: Choose an Amazon Machine Image (AMI)

An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. You can select an AMI provided by AWS, our user community, or the AWS Marketplace; or you can select one of your own AMIs.

Search for an AMI by entering a search term e.g. "Windows"

kserver - ami-012c38c4096cc0fc1

Root device type: ebs   Virtualization type: hvm   Owner: 850626458777   ENA Enabled: Yes

64-bit (x86)

Got further and launch a instance.

Join the first Node to the Cluster. ssh into the slave node

run the join command from kubeadm init screen output above.

Don’t forgot add the security group mention above first line document.

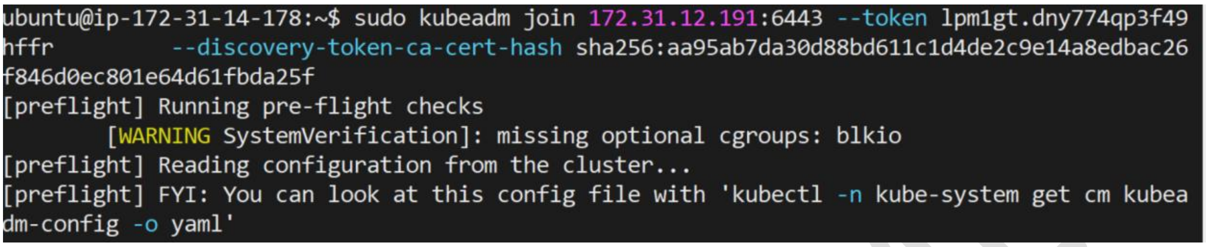
Edit instance security **group in master to allow TCP 6783 and UDP 6783/6784 ports**

Ig you forgot copy join command while you kubeadm init time use below command to get kubeadm join command.

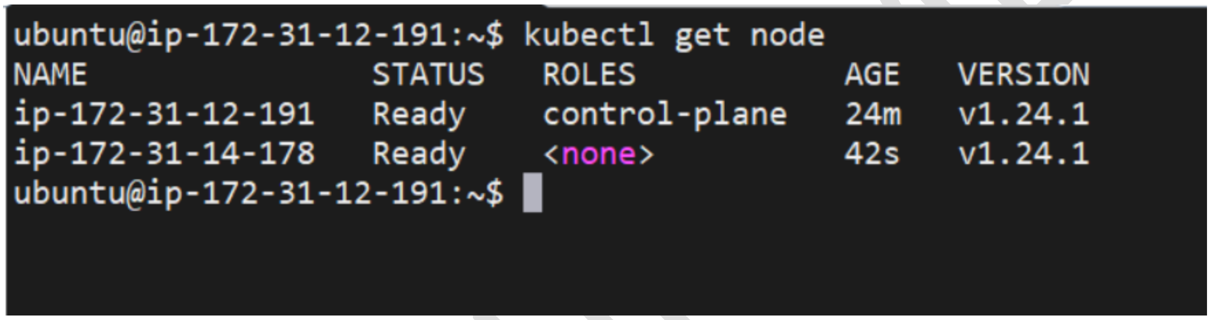
```
$ kubeadm token create --print-join-command
```

```
$ sudo kubeadm join 172.31.12.191:6443 --token lpm1gt.dny774qp3f49hffr \
        --discovery-token-ca-cert-hash
sha256:aa95ab7da30d88bd611c1d4de2c9e14a8edbac26f846d0ec801e64d61fbda25f
```

output look like below.



```
ubuntu@ip-172-31-14-178:~$ sudo kubeadm join 172.31.12.191:6443 --token lpm1gt.dny774qp3f49
hffr         --discovery-token-ca-cert-hash sha256:aa95ab7da30d88bd611c1d4de2c9e14a8edbac26
f846d0ec801e64d61fbda25f
[preflight] Running pre-flight checks
        [WARNING SystemVerification]: missing optional cgroups: blkio
[preflight] Reading configuration from the cluster...
[preflight] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubea
dm-config -o yaml'
```



```
ubuntu@ip-172-31-12-191:~$ kubectl get node
NAME                STATUS   ROLES           AGE   VERSION
ip-172-31-12-191    Ready    control-plane   24m   v1.24.1
ip-172-31-14-178    Ready    <none>          42s   v1.24.1
ubuntu@ip-172-31-12-191:~$
```

Kuberentes dash board setup

$ kubectl apply -f https://raw.githubusercontent.com/kubernetes/dashboard/v2.6.1/aio/deploy/recommended.yaml

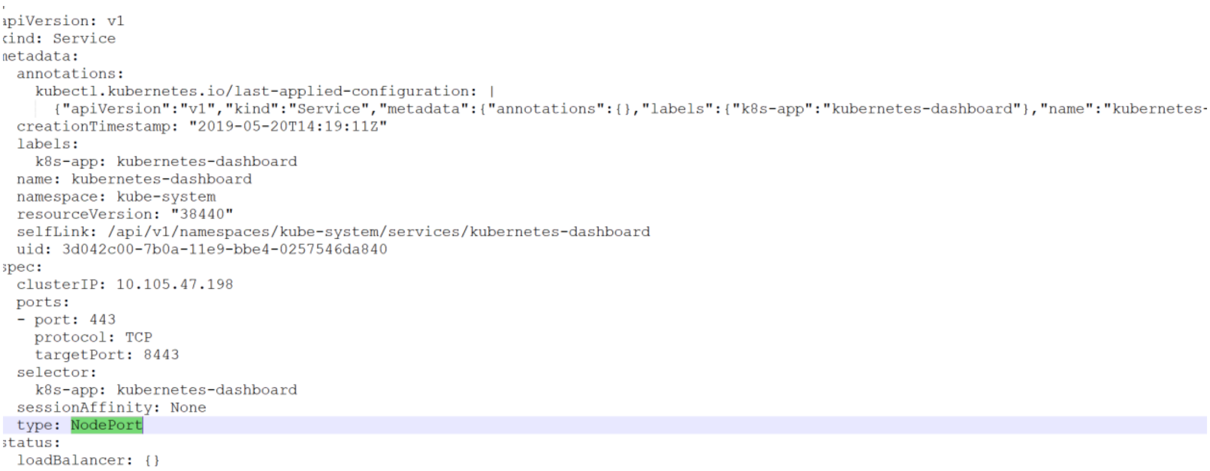
kubectl get svc --all-namespaces

kubectl -n kubernetes-dashboard get service kubernetes-dashboard

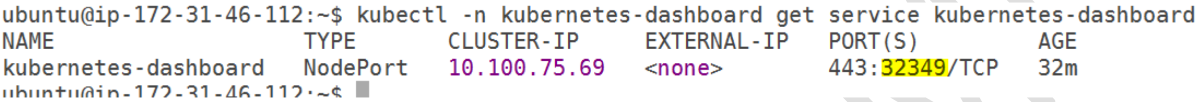
kubectl -n kubernetes-dashboard edit service kubernetes-dashboard

change ClusterIp to NodePort

kubectl -n kubernetes-dashboard get service kubernetes-dashboard



```
piVersion: v1
ind: Service
etadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |
      {"apiVersion":"v1","kind":"Service","metadata":{"annotations":{},"labels":{"k8s-app":"kubernetes-dashboard"},"name":"kubernetes-
  creationTimestamp: "2019-05-20T14:19:11Z"
  labels:
    k8s-app: kubernetes-dashboard
  name: kubernetes-dashboard
  namespace: kube-system
  resourceVersion: "38440"
  selfLink: /api/v1/namespaces/kube-system/services/kubernetes-dashboard
  uid: 3d042c00-7b0a-11e9-bbe4-0257546da840
pec:
  clusterIP: 10.105.47.198
  ports:
  - port: 443
    protocol: TCP
    targetPort: 8443
  selector:
    k8s-app: kubernetes-dashboard
  sessionAffinity: None
  type: NodePort
tatus:
  loadBalancer: {}
```



```
ubuntu@ip-172-31-46-112:~$ kubectl -n kubernetes-dashboard get service kubernetes-dashboard
NAME                   TYPE       CLUSTER-IP     EXTERNAL-IP   PORT(S)         AGE
kubernetes-dashboard   NodePort   10.100.75.69   <none>        443:32349/TCP   32m
ubuntu@ip-172-31-46-112:~$
```

take port number allow to security group.

access the dash board using node public ip with nodeport

https://<nodeip>:<port>/

it will two options config or else token.

Command line proxy

You can enable access to the Dashboard using the kubectl command-line tool, by running the following command:

$ kubectl proxy &

Creating a Service Account

We are creating Service Account with name admin-user in namespace kubernetes-dashboard first.

ubuntu@ip-172-31-5-3:~$ vi dashboard-adminuser.yaml

apiVersion: v1

kind: ServiceAccount

metadata:

  name: admin-user

  namespace: kubernetes-dashboard

$ kubectl apply -f dashboard-adminuser.yaml

Creating a ClusterRoleBinding

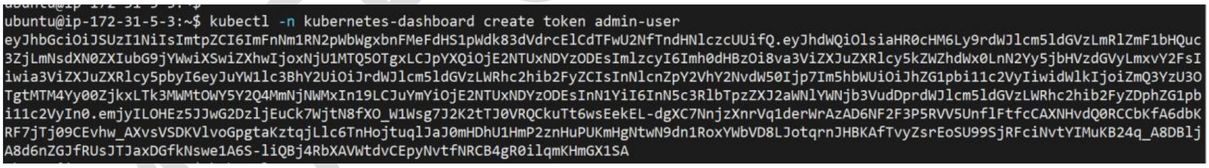
In most cases after provisioning cluster using kops, kubeadm or any other popular tool, the ClusterRole cluster-admin already exists in the cluster. We can use it and create only

ClusterRoleBinding for our **ServiceAccount**. If it does not exist then you need to create this role first and grant required privileges manually.

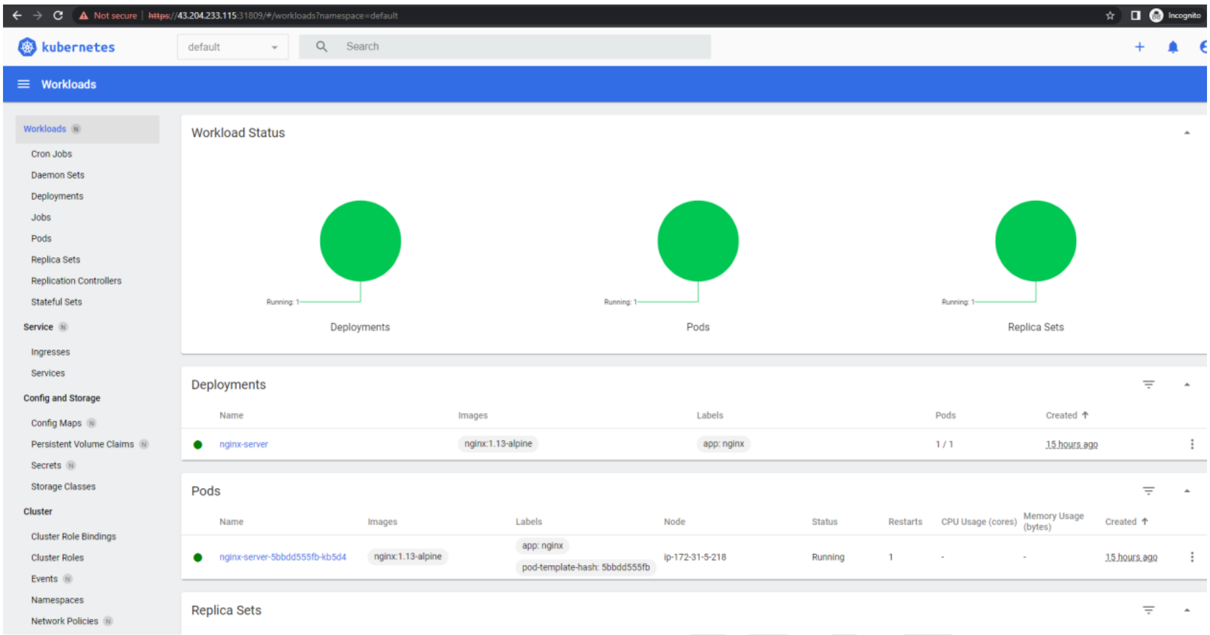
```
$ vi cluster_role_binding.yaml

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

$ kubectl apply -f cluster_role_binding.yaml
```

Getting a Bearer Token

```
$ kubectl -n kubernetes-dashboard create token admin-user
```



```
ubuntu@ip-172-31-5-3:~$
ubuntu@ip-172-31-5-3:~$ kubectl -n kubernetes-dashboard create token admin-user
eyJhbGciOiJSUzI1NiIsImtpZCI6ImFnNm1RN2pWbWgxbnFMeFdHS1pWdk83dVdrcElCdTFwU2NfTndHNlczcUUifQ.eyJhdWQiOlsiaHR0cHM6Ly9rdWJlcm5ldGVzLmRlZmF1bHQuc
3ZjLmNsdXN0ZXIubG9jYWwiXSwiZXhwIjoxNjU1MTQ5OTgxLCJpYXQiOjE2NTUxNDYzODEsImlzcyI6Imh0dHBzOi8va3ViZXJuZXRlcy5kZWZhdWx0LnN2Yy5jbHVzdGVyLmxvY2FsI
iwia3ViZXJuZXRlcy5pbyI6eyJuYW1lc3BhY2UiOiJrdWJlcm5ldGVzLWRhc2hib2FyZCIsInNlcnZpY2VhY2NvdW50Ijp7Im5hbWUiOiJhZG1pbi11c2VyIiwidWlkIjoiZmQ3YzU3O
TgtMTM4Yy00ZjkxLTk3MWMtOWY5Y2Q4MmNjNWMxIn19LCJuYmYiOjE2NTUxNDYzODEsInN1YiI6InN5c3RlbTpzZXJ2aWNlYWNjb3VudDprdWJlcm5ldGVzLWRhc2hib2FyZDphZG1pb
i11c2VyIn0.emjyILOHEz5JJwG2DzljEuCk7WjtN8fXO_W1Wsg7J2K2tTJ0VRQCkuTt6wsEekEL-dgXC7NnjzXnrVq1derWrAzAD6NF2F3P5RVV5UnflFtfcCAXNHvdQ0RCCbKfA6dbK
RF7jTj09CEvhw_AXvsVSDKVlvoGpgtaKztqjLlc6TnHojtuqlJaJ0mHDhU1HmP2znHuPUKmHgNtwN9dn1RoxYWbVD8LJotqrnJHBKAfTvyZsrEoSU99SjRFciNvtYIMuKB24q_A8DBlj
A8d6nZGJfRUsJTJaxDGfkNswe1A6S-liQBj4RbXAVWtdvCEpyNvtfNRCB4gR0ilqmKHmGX1SA
```



Deployments

	Name	Images	Labels	Pods	Created ↑
●	nginx-server	nginx:1.13-alpine	app: nginx	1 / 1	15 hours ago

Pods

	Name	Images	Labels	Node	Status	Restarts	CPU Usage (cores)	Memory Usage (bytes)	Created ↑
●	nginx-server-5bbdd555fb-kb5d4	nginx:1.13-alpine	app: nginx pod-template-hash: 5bbdd555fb	ip-172-31-5-218	Running	1	-	-	15 hours ago

Replica Sets

Remove the admin ServiceAccount and ClusterRoleBinding.

```
kubectl -n kubernetes-dashboard delete serviceaccount admin-user

kubectl -n kubernetes-dashboard delete clusterrolebinding admin-user
```