I will be using three Ubuntu 18.04 LTS system, where one system will act as **Kubernetes Master Node** and other two nodes will act as **Slave nod**e and will join the Kubernetes cluster. I am assuming minimal 18.04 LTS is installed on these three systems.

- Kubernetes Master Node – (Hostname: k8s-master , OS : Minimal Ubuntu 18.04 LTS)
- Kubernetes Slave Node 1 – (Hostname: k8s-worker-node1, OS : Minimal Ubuntu 18.04 LTS)
- Kubernetes Slave Node 2 – (Hostname: k8s-worker-node2, OS : Minimal Ubuntu

Step:1) Set Hostname and update hosts file

Login to the master node and configure its hostname using the hostnamectl command

```
ubuntu@localhost:~$ sudo hostnamectl set-hostname "k8s-master"

ubuntu@localhost:~$ exec bash

ubuntu@k8s-master:~$
```

Login to Slave / Worker Nodes and configure their hostname respectively using the hostnamectl command,

```
ubuntu@192.168.1.80    :~$ sudo hostnamectl set-hostname k8s-worker-node1

ubuntu@192.168.1.80    :~$ exec bash

ubuntu@k8s-worker-node1:~$



ubuntu@192.168.1.90    :~$ sudo hostnamectl set-hostname k8s-worker-node2

ubuntu@192.168.1.90    :~$ exec bash

ubuntu@k8s-worker-node2:~$
```

Step:2) Install and Start Docker Service on Master and Slave Nodes

Run the below apt-get command to install Docker on Master node,

```
ubuntu@k8s-master:~$ sudo apt update
```

```
ubuntu@k8s-master:~$ sudo apt-get install docker.io -y
```

Run the below apt-get command to install docker on slave nodes,

```
ubuntu@k8s-worker-node1:~$ sudo apt update

ubuntu@k8s-worker-node1:~$ sudo apt-get install docker.io -y

ubuntu@k8s-worker-node2:~$ sudo apt update

ubuntu@k8s-worker-node2:~$ sudo apt-get install docker.io -y
```

Once the Docker packages are installed on all the three systems , start and enable the docker service using below systemctl commands, these commands needs to be executed on master and slave nodes.

```
~$ sudo systemctl start docker

~$ sudo systemctl enable docker

Synchronizing state of docker.service with SysV service script with
/lib/systemd/systemd-sysv-install.

Executing: /lib/systemd/systemd-sysv-install enable docker

~$
```

Use below docker command to verify which Docker version has been installed on these systems,

```
~$ docker --version

Docker version 18.06.1-ce, build e68fc7a

~$
```

Step:3) Configure Kubernetes Package Repository on Master & Slave Nodes

Note: All the commands in this step are mandate to run on master and slave nodes
Let’s first install some required packages, run the following commands on all the nodes including master node

```
~$ sudo apt-get install apt-transport-https curl -y
```

Now add Kubernetes package repository key using the following command,

```
:~$ curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key
add

OK

:~$
```

Now configure Kubernetes repository using below apt commands, at this point of time Ubuntu 18.04 (bionic weaver) Kubernetes package repository is not available, so we will be using Xenial Kubernetes package repository.

```
:~$ sudo apt-add-repository "deb http://apt.kubernetes.io/ kubernetes-xenial main"
```

Step:4) Disable Swap and Install Kubeadm on all the nodes

Note: All the commands in this step are mandate to run on master and slave nodes Kubeadm is one of the most common method used to deploy kubernetes cluster or in other words we can say it used to deploy multiple nodes on a kubernetes cluster.

As per the Kubernetes Official web site, it is recommended to disable swap on all the nodes including master node.

Run the following command to disable swap temporary,

```
:~$ sudo swapoff -a
```

For permanent swap disable, comment out swapfile or swap partition entry in the /etc/fstab file.

Now Install Kubeadm package on all the nodes including master.

```
:~$ sudo apt-get install kubeadm -y
```

Once kubeadm packages are installed successfully, verify the kubeadm version using beneath command.

```
:~$ kubeadm version
```

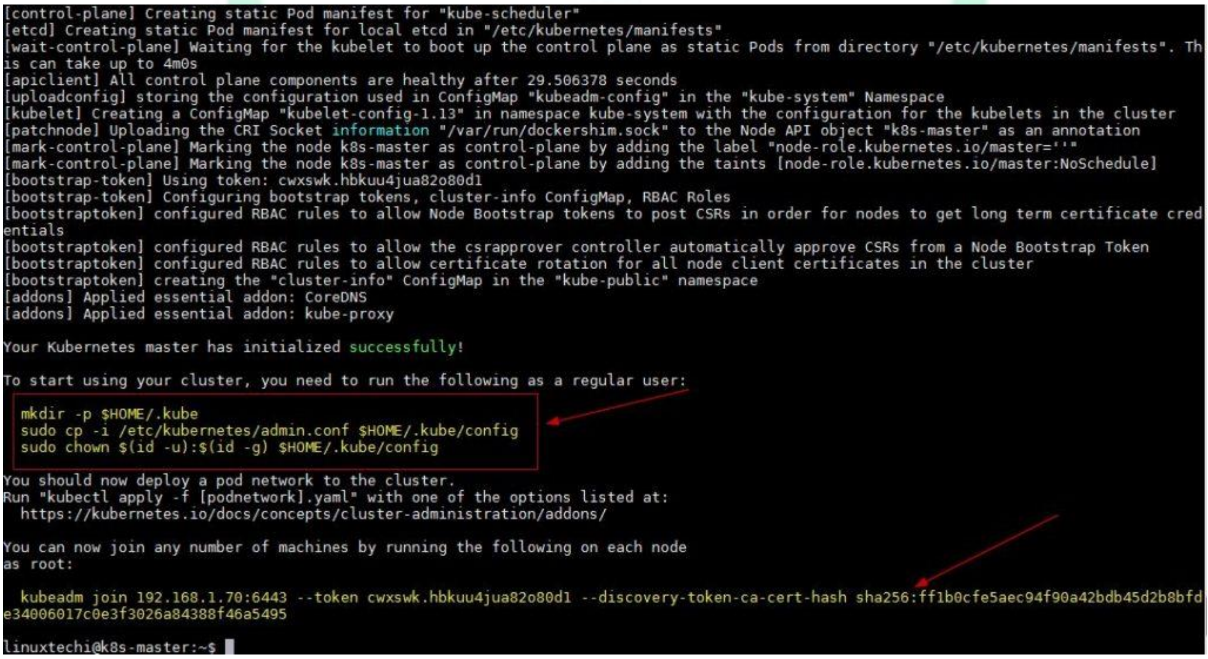
```
kubeadm version: &version.Info{Major:"1", Minor:"13", GitVersion:"v1.13.2",
GitCommit:"cff46ab41ff0bb44d8584413b598ad8360ec1def", GitTreeState:"clean",
BuildDate:"2019-01-10T23:33:30Z", GoVersion:"go1.11.4", Compiler:"gc",
Platform:"linux/amd64"}

:~$
```

Step:5) Initialize and Start Kubernetes Cluster on Master Node using Kubeadm

Use the below kubeadm command on Master Node only to initialize Kubernetes

```
ubuntu@k8s-master:~$ sudo kubeadm init --pod-network-cidr=172.168.10.0/24
```

In the above command you can use the same pod network or choose your own pod network that suits to your environment. Once the command is executed successfully, we will get the output something like below,



```
[control-plane] Creating static Pod manifest for "kube-scheduler"
[etcd] Creating static Pod manifest for local etcd in "/etc/kubernetes/manifests"
[wait-control-plane] Waiting for the kubelet to boot up the control plane as static Pods from directory "/etc/kubernetes/manifests". Th
is can take up to 4m0s
[apiclient] All control plane components are healthy after 29.506378 seconds
[uploadconfig] storing the configuration used in ConfigMap "kubeadm-config" in the "kube-system" Namespace
[kubelet] Creating a ConfigMap "kubelet-config-1.13" in namespace kube-system with the configuration for the kubelets in the cluster
[patchnode] Uploading the CRI Socket information "/var/run/dockershim.sock" to the Node API object "k8s-master" as an annotation
[mark-control-plane] Marking the node k8s-master as control-plane by adding the label "node-role.kubernetes.io/master=''"
[mark-control-plane] Marking the node k8s-master as control-plane by adding the taints [node-role.kubernetes.io/master:NoSchedule]
[bootstrap-token] Using token: cwxswk.hbkuu4jua82o80d1
[bootstrap-token] Configuring bootstrap tokens, cluster-info ConfigMap, RBAC Roles
[bootstraptoken] configured RBAC rules to allow Node Bootstrap tokens to post CSRs in order for nodes to get long term certificate cred
entials
[bootstraptoken] configured RBAC rules to allow the csrapprover controller automatically approve CSRs from a Node Bootstrap Token
[bootstraptoken] configured RBAC rules to allow certificate rotation for all node client certificates in the cluster
[bootstraptoken] creating the "cluster-info" ConfigMap in the "kube-public" namespace
[addons] Applied essential addon: CoreDNS
[addons] Applied essential addon: kube-proxy

Your Kubernetes master 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

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 machines by running the following on each node
as root:

  kubeadm join 192.168.1.70:6443 --token cwxswk.hbkuu4jua82o80d1 --discovery-token-ca-cert-hash sha256:ff1b0cfe5aec94f90a42bdb45d2b8bfd
e34006017c0e3f3026a84388f46a5495

linuxtechi@k8s-master:~$
```

Above output confirms that Master node has been initialized successfully, so to start the cluster run the beneath commands one after the another,

```
ubuntu@k8s-master:~$  mkdir -p $HOME/.kube

ubuntu@k8s-master:~$ sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

ubuntu@k8s-master:~$ sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

```
ubuntu@k8s-master:~$
```

Verify the status of master node using the following command,

```
ubuntu@k8s-master:~$ kubectl get nodes

NAME         STATUS     ROLES    AGE   VERSION

k8s-master   NotReady   master   18m   v1.13.2

ubuntu@k8s-master:~$
```

As we can see in the above command output that our master node is not ready because as of now we have not deployed any pod.

Let’s deploy the pod network, Pod network is the network through which our cluster nodes will communicate with each other. We will deploy **Flannel** as our pod network, Flannel will provide the **overlay network** between cluster nodes.

Step:6) Deploy Flannel as Pod Network from Master node and verify pod namespaces

Execute the following **kubectl** command to deploy pod network from master node

```
ubuntu@k8s-master:~$ sudo kubectl apply -f
https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-
flannel.yml
```

Output of above command should be something like below

```
clusterrole.rbac.authorization.k8s.io/flannel created

clusterrolebinding.rbac.authorization.k8s.io/flannel created

serviceaccount/flannel created

configmap/kube-flannel-cfg created

daemonset.extensions/kube-flannel-ds-amd64 created

daemonset.extensions/kube-flannel-ds-arm64 created

daemonset.extensions/kube-flannel-ds-arm created

daemonset.extensions/kube-flannel-ds-ppc64le created
daemonset.extensions/kube-flannel-ds-s390x created

ubuntu@k8s-master:~$
```

Now verify the master node status and pod namespaces using kubectl command,

```
ubuntu@k8s-master:~$ sudo  kubectl get nodes

NAME         STATUS   ROLES    AGE   VERSION

k8s-master   Ready    master   78m   v1.13.2

ubuntu@k8s-master:~$



ubuntu@k8s-master:~$ sudo  kubectl get pods --all-namespaces

NAMESPACE     NAME                                  READY   STATUS    RESTARTS   AGE

kube-system   coredns-86c58d9df4-px4sj              1/1     Running   0          79m

kube-system   coredns-86c58d9df4-wzdzk              1/1     Running   0          79m

kube-system   etcd-k8s-master                       1/1     Running   1          79m

kube-system   kube-apiserver-k8s-master             1/1     Running   1          79m

kube-system   kube-controller-manager-k8s-master    1/1     Running   1          79m

kube-system   kube-flannel-ds-amd64-9tn8z           1/1     Running   0          14m

kube-system   kube-proxy-cjzz2                      1/1     Running   1          79m

kube-system   kube-scheduler-k8s-master             1/1     Running   1          79m

ubuntu@k8s-master:~$
```

As we can see in the above output our master node status has changed to "**Ready**" and all the namespaces of pod are in running state, so this confirms that our master node is in healthy state and ready to form a cluster.

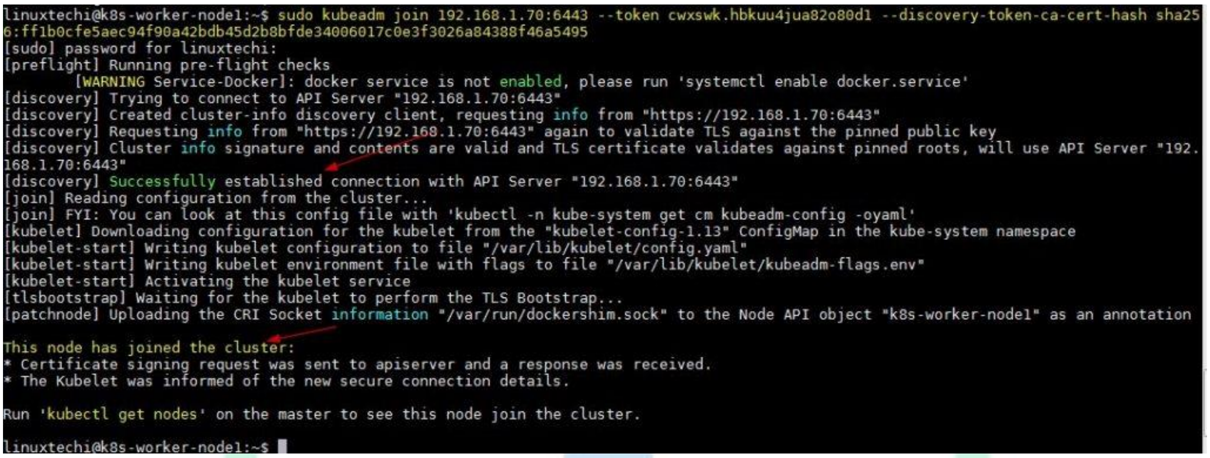
Step:7) Add Slave or Worker Nodes to the Cluster

Note: In Step 5, kubeadm command output we got complete command which we will have to use on slave or worker node to join a cluster

Login to first slave node (k8s-worker-node1) and run the following command to join the cluster,

```
ubuntu@k8s-worker-node1:~$ sudo kubeadm join 192.168.1.70:6443 --token
cwxswk.hbkuu4jua82o80d1 --discovery-token-ca-cert-hash
sha256:ff1b0cfe5aec94f90a42bdb45d2b8bfde34006017c0e3f3026a84388f46a5495
```

Output of above command should be something like this,

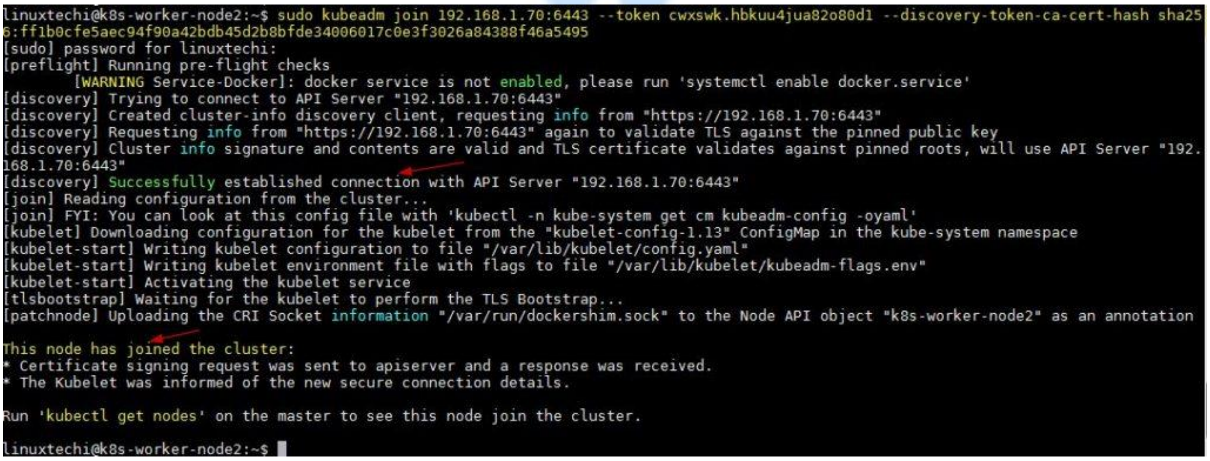


```
linuxtechi@k8s-worker-node1:~$ sudo kubeadm join 192.168.1.70:6443 --token cwxswk.hbkuu4jua82o80d1 --discovery-token-ca-cert-hash sha25
6:ff1b0cfe5aec94f90a42bdb45d2b8bfde34006017c0e3f3026a84388f46a5495
[sudo] password for linuxtechi:
[preflight] Running pre-flight checks
        [WARNING Service-Docker]: docker service is not enabled, please run 'systemctl enable docker.service'
[discovery] Trying to connect to API Server "192.168.1.70:6443"
[discovery] Created cluster-info discovery client, requesting info from "https://192.168.1.70:6443"
[discovery] Requesting info from "https://192.168.1.70:6443" again to validate TLS against the pinned public key
[discovery] Cluster info signature and contents are valid and TLS certificate validates against pinned roots, will use API Server "192.
168.1.70:6443"
[discovery] Successfully established connection with API Server "192.168.1.70:6443"
[join] Reading configuration from the cluster...
[join] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-config -oyaml'
[kubelet] Downloading configuration for the kubelet from the "kubelet-config-1.13" ConfigMap in the kube-system namespace
[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] Activating the kubelet service
[tlsbootstrap] Waiting for the kubelet to perform the TLS Bootstrap...
[patchnode] Uploading the CRI Socket information "/var/run/dockershim.sock" to the Node API object "k8s-worker-node1" as an annotation

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 master to see this node join the cluster.

linuxtechi@k8s-worker-node1:~$
```

Similarly run the same kubeadm join command on the second worker node,

```
ubuntu@k8s-worker-node2:~$ sudo kubeadm join 192.168.1.70:6443 --token
cwxswk.hbkuu4jua82o80d1 --discovery-token-ca-cert-hash
sha256:ff1b0cfe5aec94f90a42bdb45d2b8bfde34006017c0e3f3026a84388f46a5495
```

Output of above should be something like below,



```
linuxtechi@k8s-worker-node2:~$ sudo kubeadm join 192.168.1.70:6443 --token cwxswk.hbkuu4jua82o80d1 --discovery-token-ca-cert-hash sha25
6:ff1b0cfe5aec94f90a42bdb45d2b8bfde34006017c0e3f3026a84388f46a5495
[sudo] password for linuxtechi:
[preflight] Running pre-flight checks
        [WARNING Service-Docker]: docker service is not enabled, please run 'systemctl enable docker.service'
[discovery] Trying to connect to API Server "192.168.1.70:6443"
[discovery] Created cluster-info discovery client, requesting info from "https://192.168.1.70:6443"
[discovery] Requesting info from "https://192.168.1.70:6443" again to validate TLS against the pinned public key
[discovery] Cluster info signature and contents are valid and TLS certificate validates against pinned roots, will use API Server "192.
168.1.70:6443"
[discovery] Successfully established connection with API Server "192.168.1.70:6443"
[join] Reading configuration from the cluster...
[join] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-config -oyaml'
[kubelet] Downloading configuration for the kubelet from the "kubelet-config-1.13" ConfigMap in the kube-system namespace
[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] Activating the kubelet service
[tlsbootstrap] Waiting for the kubelet to perform the TLS Bootstrap...
[patchnode] Uploading the CRI Socket information "/var/run/dockershim.sock" to the Node API object "k8s-worker-node2" as an annotation

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 master to see this node join the cluster.

linuxtechi@k8s-worker-node2:~$
```

Now go to master node and run below command to check master and slave node status

```
ubuntu@k8s-master:~$ kubectl get nodes

NAME                   STATUS    ROLES     AGE     VERSION
```

```
k8s-master         Ready    master   100m   v1.13.2

k8s-worker-node1   Ready    <none>   10m    v1.13.2

k8s-worker-node2   Ready    <none>   4m6s   v1.13.2

ubuntu@k8s-master:~$
```

Above command confirm that we have successfully added our two worker nodes in the cluster and their state is Ready.This concludes that we have successfully installed and configured two node Kubernetes cluster on Ubuntu 18.04 systems.