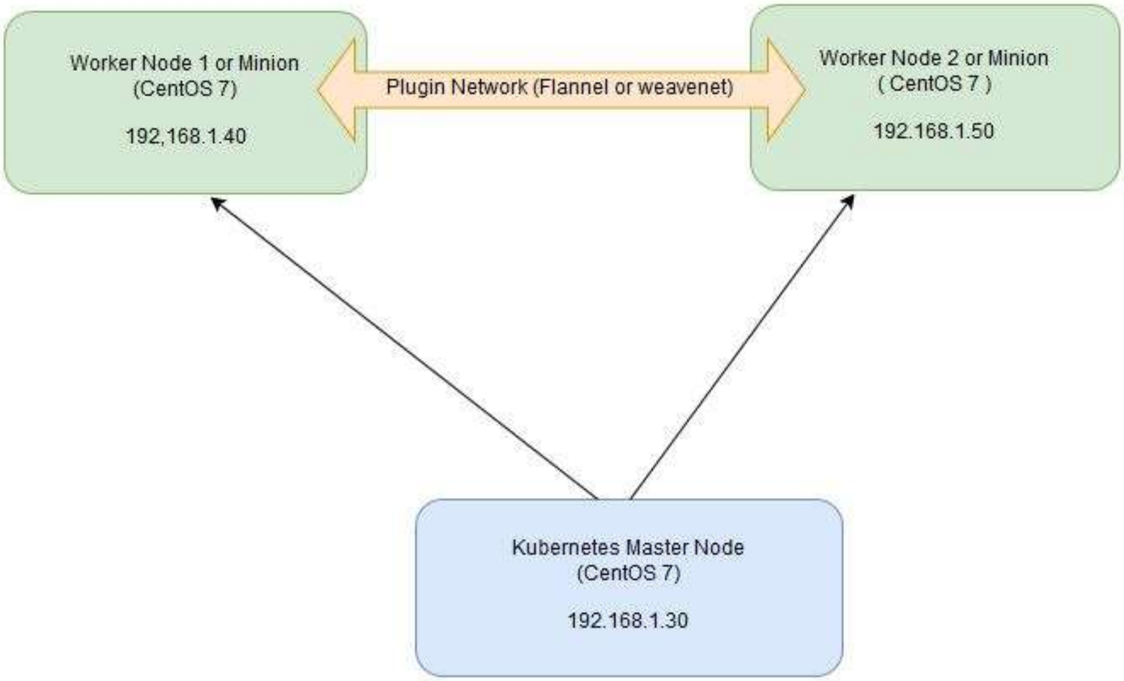
How to Install Kubernetes (k8s) 1.7 on CentOS 7 / RHEL 7



Min Virtual machine requirement

CPU → 2

RAM → 1.5 GB

Disk → 10 GB

No of Node – 3 , 1 for Kubernetes Master and 2 for the Nodes.

On the Master Node following components will be installed

- **API Server** – It provides kubernetes API using Jason / Yaml over http, states of API objects are stored in etcd
- **Scheduler** – It is a program on master node which performs the scheduling tasks like launching containers in worker nodes based on resource availability
- **Controller Manager** – Main Job of Controller manager is to monitor replication controllers and create pods to maintain desired state.
- **etcd** – It is a Key value pair data base. It stores configuration data of cluster and cluster state.
- **Kubectl utility** – It is a command line utility which connects to API Server on port 6443. It is used by administrators to create pods, services etc.

On Worker Nodes following components will be installed

- **Kubelet** – It is an agent which runs on every worker node, it connects to docker and takes care of creating, starting, deleting containers.
- **Kube-Proxy** – It routes the traffic to appropriate containers based on ip address and port number of the incoming request. In other words we can say it is used for port translation.
- **Pod** – Pod can be defined as a multi-tier or group of containers that are deployed on a single worker node or docker host.

Step 1: Disable SELinux & setup firewall rules

BELOW COMMANDS TO BE EXECUTED ON ALL THE 3 NODES (1 MASTER AND 2 CLIENT NODES)

Login to your kubernetes master node and set the hostname and disable selinux using following commands

```
exec bash

setenforce 0

sed -i --follow-symlinks
's/SELINUX=enforcing/SELINUX=disabled/g' /etc/sysconfig/selinux
```

Set the following firewall rules.

```
firewall-cmd --permanent --add-port=6443/tcp

firewall-cmd --permanent --add-port=2379-2380/tcp

firewall-cmd --permanent --add-port=10250/tcp

firewall-cmd --permanent --add-port=10251/tcp

firewall-cmd --permanent --add-port=10252/tcp

firewall-cmd --permanent --add-port=10255/tcp

firewall-cmd --reload

modprobe br_netfilter
```

```
[root@k8s-master ~]# echo '1' > /proc/sys/net/bridge/bridge-nf-call-
iptables
```

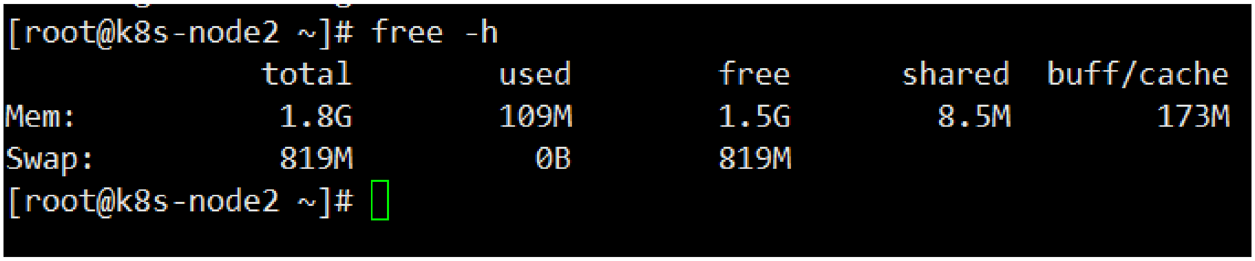
Note: In case you don’t have your own dns server then update **/etc/hosts** file on master and worker nodes

```
192.168.1.101 k8s-master

192.168.1.102 k8s-node1

192.168.1.103 k8s-node2
```

Note: -- In this scenario, we have the private network as “192.168.1.0/24” for this lab setup.

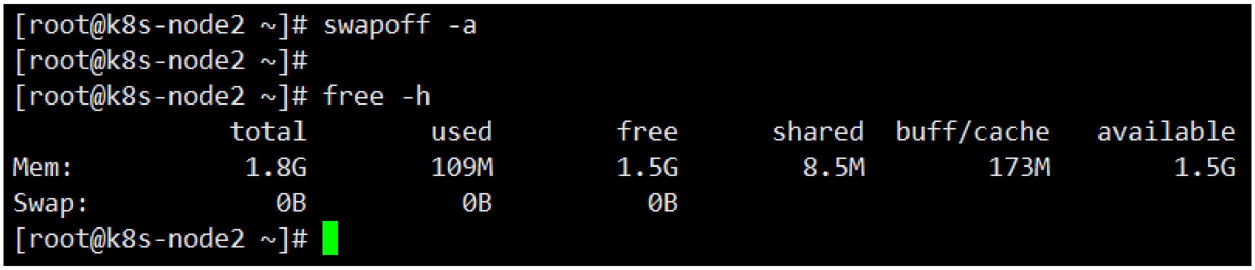
Please check the network in which you are working and accordingly, change the “4th Octect”.

Also, switch off the swap on all the 3 virtual machines as a requirement for Kubernetes Cluster



```
[root@k8s-node2 ~]# free -h
              total        used        free      shared  buff/cache
Mem:           1.8G        109M        1.5G        8.5M        173M
Swap:          819M          0B        819M
[root@k8s-node2 ~]#
```

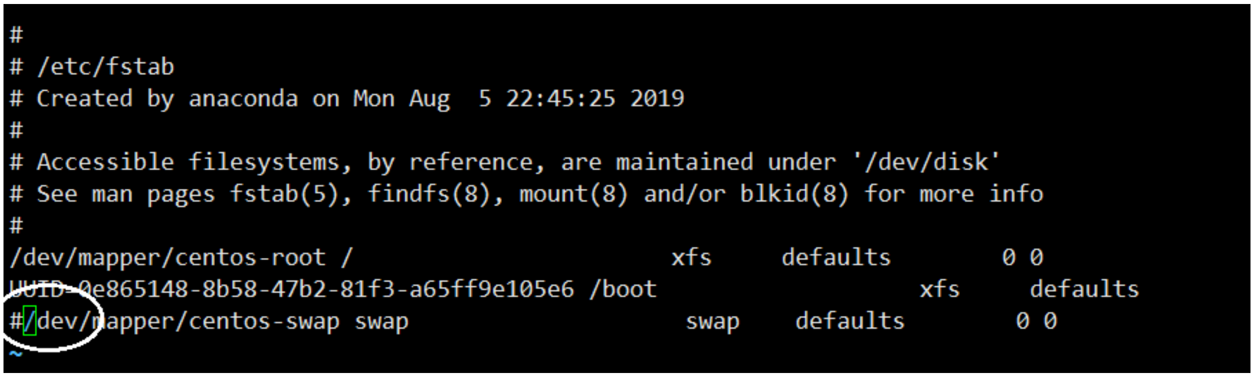
```
swapoff -a
```



```
[root@k8s-node2 ~]# swapoff -a
[root@k8s-node2 ~]#
[root@k8s-node2 ~]# free -h
              total        used        free      shared  buff/cache   available
Mem:           1.8G        109M        1.5G        8.5M        173M        1.5G
Swap:            0B          0B          0B
[root@k8s-node2 ~]#
```

Let’s make this config permanent, so that for the next reboot of the VM, the swap would be still off.

For that we would need to edit the **“/etc/fstab”** file



```
#
# /etc/fstab
# Created by anaconda on Mon Aug  5 22:45:25 2019
#
# Accessible filesystems, by reference, are maintained under '/dev/disk'
# See man pages fstab(5), findfs(8), mount(8) and/or blkid(8) for more info
#
/dev/mapper/centos-root /                       xfs     defaults        0 0
UUID=0e865148-8b58-47b2-81f3-a65ff9e105e6 /boot                   xfs     defaults        0 0
#/dev/mapper/centos-swap swap                    swap    defaults        0 0
~
```

And either “delete” the line that say swap or comment the line with “#” in the beginning of the line.

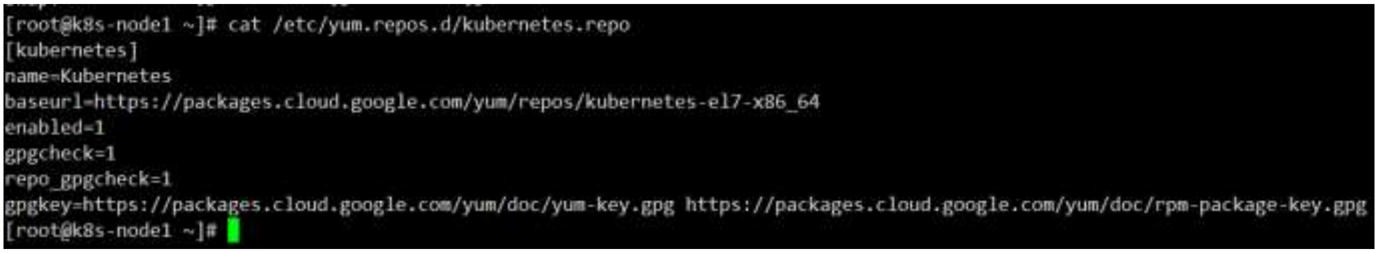
And Save and quit → “:wq”

Step 2: Configure Kubernetes Repository

BELOW COMMANDS TO BE EXECUTED ON ALL THE 3 NODES (1 MASTER AND 2 CLIENT NODES)

Kubernetes packages are not available in the default CentOS 7 & RHEL 7 repositories, Use below command to configure its package repositories.

```
[root@k8s-master ~]# vi /etc/yum.repos.d/kubernetes.repo

[kubernetes]

name=Kubernetes

baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-
x86_64

enabled=1

gpgcheck=1

repo_gpgcheck=1

gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg
https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg

[root@k8s-master ~]#
```



```
[root@k8s-node1 ~]# cat /etc/yum.repos.d/kubernetes.repo
[kubernetes]
name=Kubernetes
baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-x86_64
enabled=1
gpgcheck=1
repo_gpgcheck=1
gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg
[root@k8s-node1 ~]#
```

Make sure the alignment and space are exactly the same as the above screen shot, if not the **kubeadm** packages would not get installed, as the repo may not be recognized.

Step 3: Install Kubeadm and Docker

BELOW COMMANDS TO BE EXECUTED ON ALL THE 3 NODES (1 MASTER AND 2 CLIENT NODES)

Once the package repositories are configured, run the beneath command to install kubeadm and docker packages.

Install Docker CE

Install the latest version of Docker-ce from the docker repository.
Install the package dependencies for docker-ce.

```
[root@k8s-master ~]# yum install -y yum-utils device-mapper-persistent-data lvm2
```

Add the docker repository to the system and install docker-ce using the yum command.

```
[root@k8s-master ~]# yum-config-manager --add-repo https://download.docker.com/l
inux/centos/docker-ce.repo
```

```
[root@k8s-master ~]# yum install kubeadm docker-ce -y
```

Start and enable docker service

```
[root@k8s-master ~]# systemctl restart docker && systemctl enable docker
```

Step 4: Initialize Kubernetes Master with ‘kubeadm init’

BELOW COMMANDS TO BE EXECUTED ONLY ON MASTER NODE

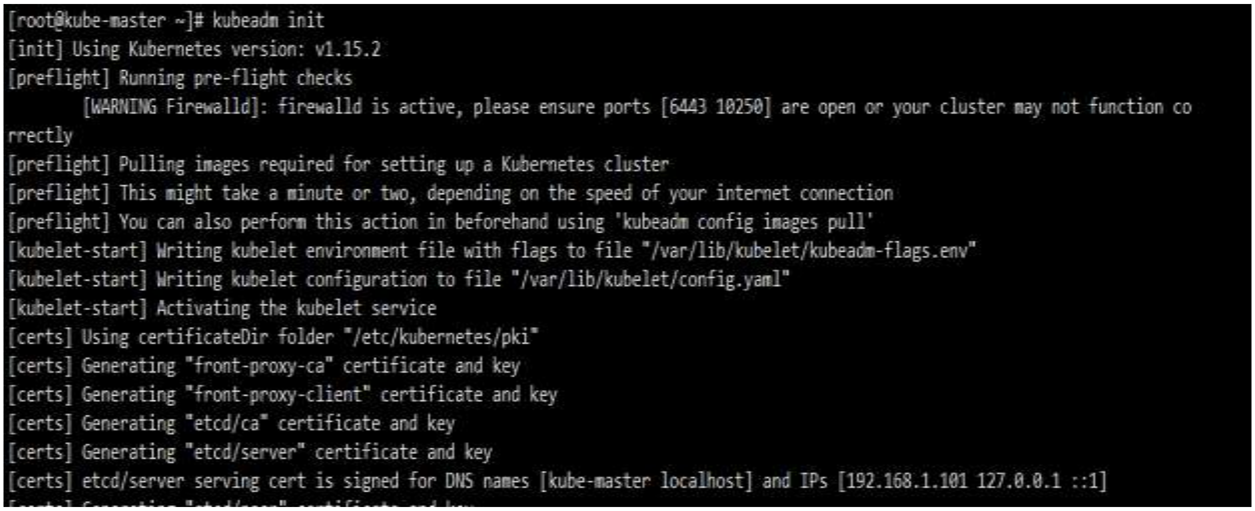
Start and enable kubectl service

```
[root@k8s-master ~]# systemctl  restart kubelet && systemctl enable kubelet
```

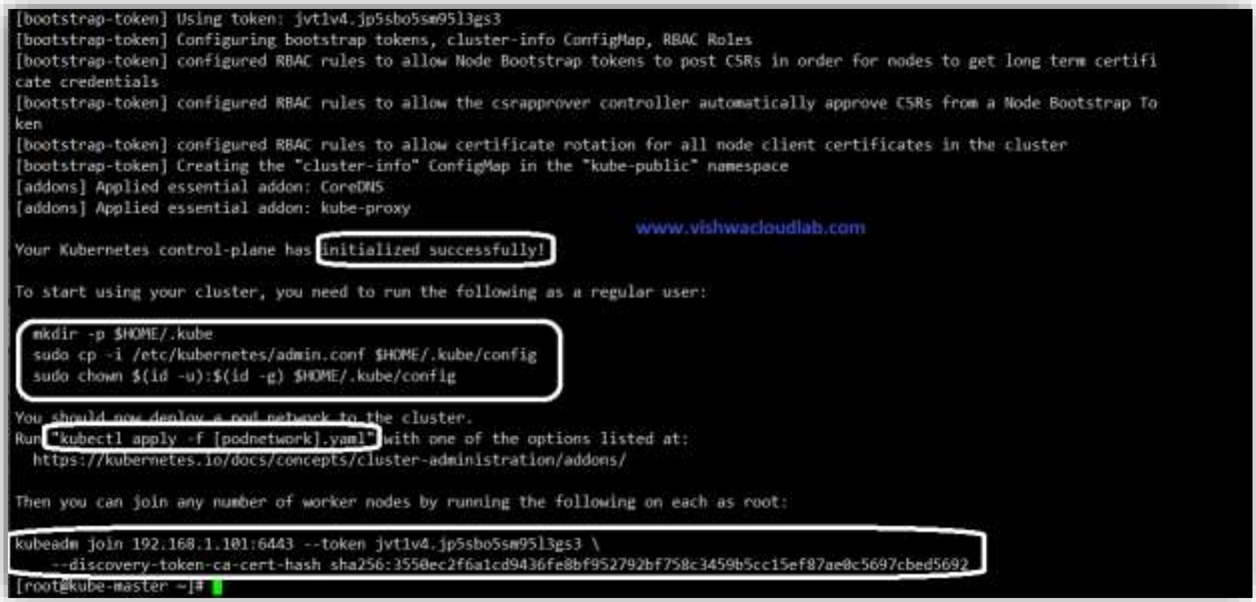
Run the beneath command to initialize and setup kubernetes master.

```
[root@kube-master ~]# kubeadm init
```

Output of above command would be something like below



```
[root@kube-master ~]# kubeadm init
[init] Using Kubernetes version: v1.15.2
[preflight] Running pre-flight checks
        [WARNING Firewalld]: firewalld is active, please ensure ports [6443 10250] are open or your cluster may not function co
rrectly
[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'
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flags.env"
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[kubelet-start] Activating the kubelet service
[certs] Using certificateDir folder "/etc/kubernetes/pki"
[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
[certs] etcd/server serving cert is signed for DNS names [kube-master localhost] and IPs [192.168.1.101 127.0.0.1 ::1]
[certs] Generating "etcd/peer" certificate and key
```



```
[bootstrap-token] Using token: jvt1v4.jp5sbo5sm95l3gs3
[bootstrap-token] Configuring bootstrap tokens, cluster-info ConfigMap, RBAC Roles
[bootstrap-token] configured RBAC rules to allow Node Bootstrap tokens to post CSRs in order for nodes to get long term certifi
cate credentials
[bootstrap-token] configured RBAC rules to allow the csrapprover controller automatically approve CSRs from a Node Bootstrap To
ken
[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
[addons] Applied essential addon: CoreDNS
[addons] Applied essential addon: kube-proxy
                                                   www.vishwacloudlab.com
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

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 192.168.1.101:6443 --token jvt1v4.jp5sbo5sm95l3gs3 \
    --discovery-token-ca-cert-hash sha256:3550ec2f6a1cd9436fe8bf952792bf758c3459b5cc15ef87ae0c5697cbed5692
[root@kube-master ~]#
```

As we can see in the output that kubernetes master has been initialized successfully. Execute the beneath commands to use the cluster as root user.

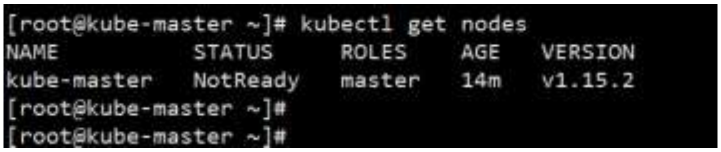
```
[root@kube-master ~]# mkdir -p $HOME/.kube

[root@kube-master ~]# cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

[root@kube-master ~]# chown $(id -u):$(id -g) $HOME/.kube/config
```

Step 5: Deploy pod network to the cluster

BELOW COMMANDS TO BE EXECUTED ONLY ON MASTER NODE

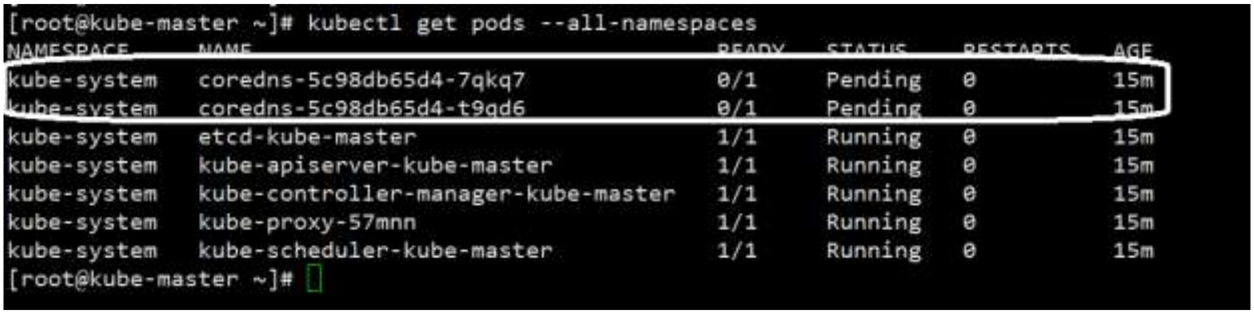
Lets try to run below commands to get status of cluster and pods.

```
[root@kube-master ~]# kubectl get nodes
```



```
[root@kube-master ~]# kubectl get nodes
NAME          STATUS     ROLES    AGE   VERSION
kube-master   NotReady   master   14m   v1.15.2
[root@kube-master ~]#
[root@kube-master ~]#
```

```
[root@kube-master ~]# kubectl get pods --all-namespaces
```



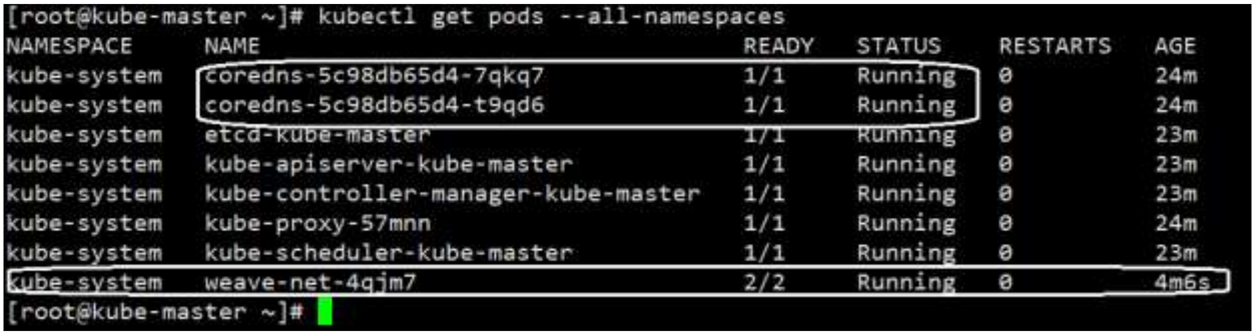
```
[root@kube-master ~]# kubectl get pods --all-namespaces
NAMESPACE     NAME                                  READY   STATUS    RESTARTS   AGE
kube-system   coredns-5c98db65d4-7qkq7              0/1     Pending   0          15m
kube-system   coredns-5c98db65d4-t9qd6              0/1     Pending   0          15m
kube-system   etcd-kube-master                      1/1     Running   0          15m
kube-system   kube-apiserver-kube-master            1/1     Running   0          15m
kube-system   kube-controller-manager-kube-master   1/1     Running   0          15m
kube-system   kube-proxy-57mnn                      1/1     Running   0          15m
kube-system   kube-scheduler-kube-master            1/1     Running   0          15m
[root@kube-master ~]#
```

To make the cluster status ready and kube-dns status running, deploy the pod network so that containers of different host communicated each other.

POD network is the overlay network between the worker nodes.

```
[root@kube-master ~]# export kubever=$(kubectl version | base64 | tr -d '\n')


[root@kube-master ~]# kubectl apply -f
"https://cloud.weave.works/k8s/net?k8s-version=$kubever"

serviceaccount "weave-net" created

clusterrole "weave-net" created

clusterrolebinding "weave-net" created

daemonset "weave-net" created

[root@kube-master ~]#
```

Now, lets chk the DNS pods again,



```
[root@kube-master ~]# kubectl get pods --all-namespaces
NAMESPACE     NAME                                 READY   STATUS    RESTARTS   AGE
kube-system   coredns-5c98db65d4-7qkq7             1/1     Running   0          24m
kube-system   coredns-5c98db65d4-t9qd6             1/1     Running   0          24m
kube-system   etcd-kube-master                     1/1     Running   0          23m
kube-system   kube-apiserver-kube-master           1/1     Running   0          23m
kube-system   kube-controller-manager-kube-master  1/1     Running   0          23m
kube-system   kube-proxy-57mnn                     1/1     Running   0          24m
kube-system   kube-scheduler-kube-master           1/1     Running   0          23m
kube-system   weave-net-4qjm7                      2/2     Running   0          4m6s
[root@kube-master ~]#
```

Note: -- This take approx. 2 -5 min sometimes. Please be patient…

Now, that the Kube Master is ready, lets configure the 2 worker nodes for this cluster.

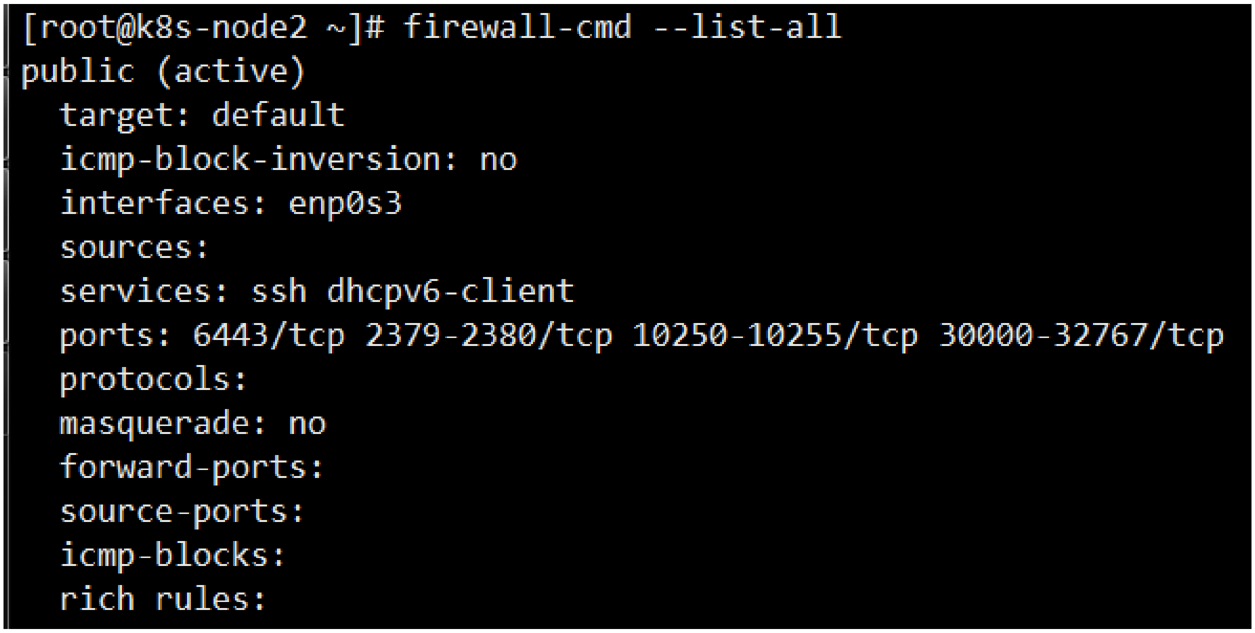
Step 6: *Perform the following steps on each worker/Client node*

Configure firewall rules on both the Client nodes

Add these firewall ports which is for the worker nodes only on the both nodes as 'k8s-node1' and 'k8s-node2' respectively

```
firewall-cmd --permanent --add-port=30000-32767/tcp


firewall-cmd  --reload
```

Output: To chk the firewall config

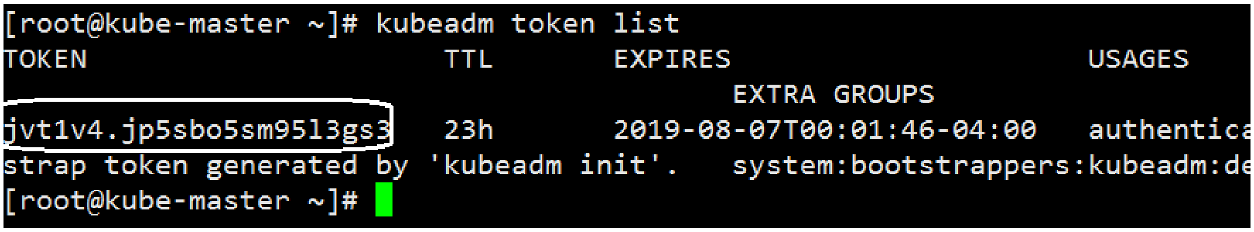


```
[root@k8s-node2 ~]# firewall-cmd --list-all
public (active)
  target: default
  icmp-block-inversion: no
  interfaces: enp0s3
  sources:
  services: ssh dhcpv6-client
  ports: 6443/tcp 2379-2380/tcp 10250-10255/tcp 30000-32767/tcp
  protocols:
  masquerade: no
  forward-ports:
  source-ports:
  icmp-blocks:
  rich rules:
```

Step 7: Now Join worker/Client nodes to master node

To join worker nodes to **Master node**, a token is required. Whenever kubernetes master initialized , then in the **output we get command and token**. Copy that command and run on both nodes.

To get the token on the master



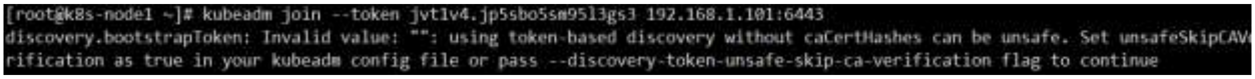
```
[root@kube-master ~]# kubeadm token list
TOKEN                      TTL       EXPIRES                      USAGES
                                             EXTRA GROUPS
jvt1v4.jp5sbo5sm9513gs3    23h       2019-08-07T00:01:46-04:00    authentica
strap token generated by 'kubeadm init'.    system:bootstrappers:kubeadm:de
[root@kube-master ~]#
```

Now, with the join command

```
[root@k8s-node1 ~]# kubeadm join --token jvt1v4.jp5sbo5sm95l3gs3 192.168.1.101:6443
```

Here, the Ip address of the master Kube.

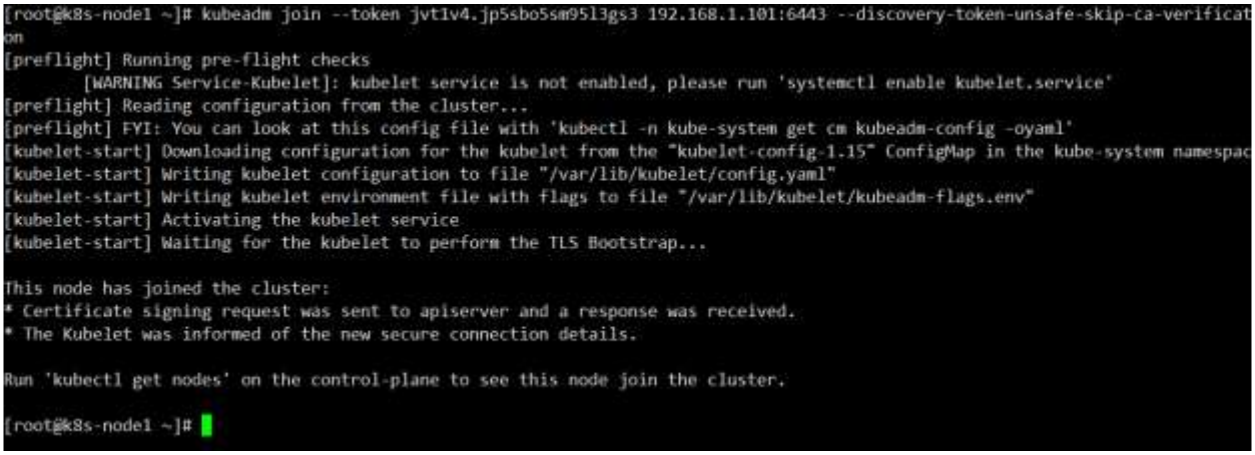
Note—you make get an warning



```
[root@k8s-node1 ~]# kubeadm join --token jvt1v4.jp5sbo5sm95l3gs3 192.168.1.101:6443
discovery.bootstrapToken: Invalid value: "": using token-based discovery without caCertHashes can be unsafe. Set unsafeSkipCAVe
rification as true in your kubeadm config file or pass --discovery-token-unsafe-skip-ca-verification flag to continue
```

Saying that token is without the certificates and it might be unsafe.

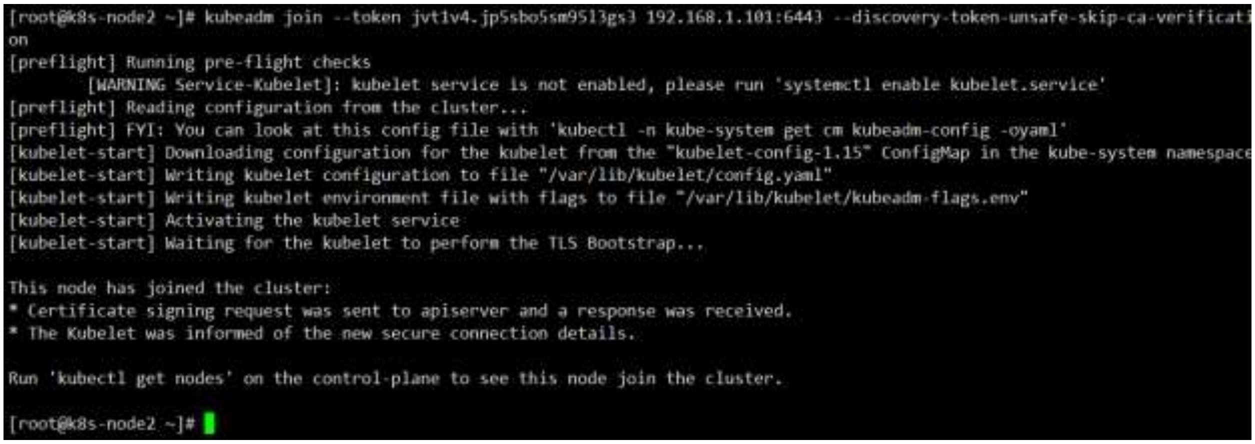
```
[root@k8s-node1 ~]# kubeadm join --token jvt1v4.jp5sbo5sm95l3gs3 192.168.1.101:6443 --
discovery-token-unsafe-skip-ca-verification
```



```
[root@k8s-node1 ~]# kubeadm join --token jvt1v4.jp5sbo5sm95l3gs3 192.168.1.101:6443 --discovery-token-unsafe-skip-ca-verificat
on
[preflight] Running pre-flight checks
        [WARNING Service-Kubelet]: kubelet service is not enabled, please run 'systemctl enable kubelet.service'
[preflight] Reading configuration from the cluster...
[preflight] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-config -oyaml'
[kubelet-start] Downloading configuration for the kubelet from the "kubelet-config-1.15" ConfigMap in the kube-system namespac
[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
[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.

[root@k8s-node1 ~]#
```

This shows that the node is successfully joined the cluster.

Similarly for Node2.

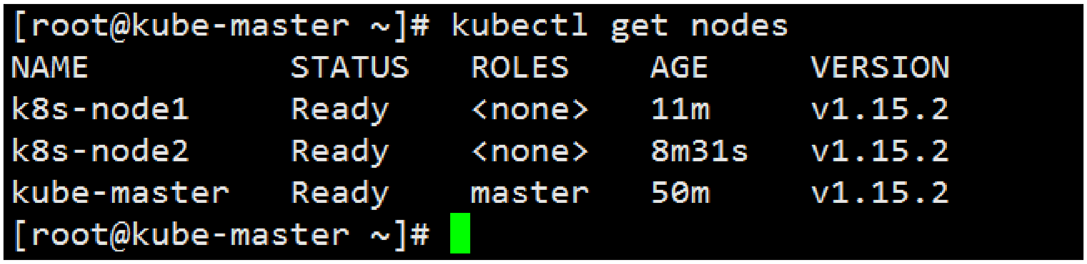


```
[root@k8s-node2 ~]# kubeadm join --token jvt1v4.jp5sbo5sm95l3gs3 192.168.1.101:6443 --discovery-token-unsafe-skip-ca-verificati
on
[preflight] Running pre-flight checks
        [WARNING Service-Kubelet]: kubelet service is not enabled, please run 'systemctl enable kubelet.service'
[preflight] Reading configuration from the cluster...
[preflight] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-config -oyaml'
[kubelet-start] Downloading configuration for the kubelet from the "kubelet-config-1.15" 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
[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.

[root@k8s-node2 ~]#
```

Now verify Nodes status from master node using kubectl command

```
[root@kube-master ~]# kubectl get nodes
```

Note: -- It takes around 5 - 9 min for all the nodes to show as “**Ready**” .



```
[root@kube-master ~]# kubectl get nodes
NAME          STATUS   ROLES    AGE     VERSION
k8s-node1     Ready    <none>   11m     v1.15.2
k8s-node2     Ready    <none>   8m31s   v1.15.2
kube-master   Ready    master   50m     v1.15.2
[root@kube-master ~]#
```

As we can see master and worker nodes are in ready status.

This concludes that kubernetes 1.7 has been installed successfully and also we have successfully joined two worker nodes.

Now we can create pods and services on this Kube Cluster.