Installing Kubernetes with kubeadm

Requirements:

You need:

* Three or more machines that meet [kubeadm's minimum requirements](https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/install-kubeadm/#before-you-begin) for the control-plane nodes. Having an odd number of control plane nodes can help with leader selection in the case of machine or zone failure.
  + including a [container runtime](https://kubernetes.io/docs/setup/production-environment/container-runtimes), already set up and working
* Three or more machines that meet [kubeadm's minimum requirements](https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/install-kubeadm/#before-you-begin) for the workers
  + including a container runtime, already set up and working
* Full network connectivity between all machines in the cluster (public or private network)
* Superuser privileges on all machines using sudo
  + You can use a different tool; this guide uses sudo in the examples.
* SSH access from one device to all nodes in the system
* kubeadm and kubelet already installed on all machines.
* A compatible Linux host. The Kubernetes project provides generic instructions for Linux distributions based on Debian and Red Hat, and those distributions without a package manager.
* 2 GB or more of RAM per machine (any less will leave little room for your apps).
* 2 CPUs or more.
* Full network connectivity between all machines in the cluster (public or private network is fine).
* Unique hostname, MAC address, and product\_uuid for every node. See [here](https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/install-kubeadm/#verify-mac-address) for more details.
* Certain ports are open on your machines. See [here](https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/install-kubeadm/#check-required-ports) for more details.
* Swap configuration. The default behaviour of a kubelet was to fail to start if swap memory was detected on a node. Swap has been supported since v1.22. And since v1.28, Swap is supported for cgroup v2 only; the NodeSwap feature gate of the kubelet is beta but disabled by default.
  + You **MUST** disable swap if the kubelet is not properly configured to use swap. For example, sudo swapoff -a will disable swapping temporarily. To make this change persistent across reboots, make sure swap is disabled in config files like /etc/fstab, systemd.swap, depending how it was configured on your system.

## Verify the MAC address and product\_uuid are unique for every node

* You can get the MAC address of the network interfaces using the command ip link or ifconfig -a
* The product\_uuid can be checked by using the command sudo cat /sys/class/dmi/id/product\_uuid

It is very likely that hardware devices will have unique addresses, although some virtual machines may have identical values. Kubernetes uses these values to uniquely identify the nodes in the cluster. If these values are not unique to each node, the installation process may [fail](https://github.com/kubernetes/kubeadm/issues/31).

## Check network adapters

## If you have more than one network adapter, and your Kubernetes components are not reachable on the default route, we recommend you add IP route(s) so Kubernetes cluster addresses go via the appropriate adapter

**Check required ports**

These [required ports](https://kubernetes.io/docs/reference/networking/ports-and-protocols/) need to be open in order for Kubernetes components to communicate with each other. You can use tools like netcat to check if a port is open. For example:

nc 127.0.0.1 6443

The pod network plugin you use may also require certain ports to be open. Since this differs with each pod network plugin, please see the documentation for the plugins about what port(s) those need.

Creation of Load Balancer with virtual IP:

Kindly get one virtual IP from infra team that will be used as Load balancer for all master nodes:

In our case it is 172.21.90.188

**Install and Configure Keepalive and HAProxy on all master / control plane nodes**

Reference document : [How to Setup Kubernetes(k8s) Cluster in HA with Kubeadm (linuxtechi.com)](https://www.linuxtechi.com/setup-highly-available-kubernetes-cluster-kubeadm/)

* + 1. sudo yum install haproxy keepalived –y
    2. sudo vi /etc/keepalived/check\_apiserver.sh

#!/bin/sh

APISERVER\_VIP=192.168.1.45

APISERVER\_DEST\_PORT=6443

errorExit() {

echo "\*\*\* $\*" 1>&2

exit 1

}

curl --silent --max-time 2 --insecure https://localhost:${APISERVER\_DEST\_PORT}/ -o /dev/null || errorExit "Error GET https://localhost:${APISERVER\_DEST\_PORT}/"

if ip addr | grep -q ${APISERVER\_VIP}; then

curl --silent --max-time 2 --insecure https://${APISERVER\_VIP}:${APISERVER\_DEST\_PORT}/ -o /dev/null || errorExit "Error GET https://${APISERVER\_VIP}:${APISERVER\_DEST\_PORT}/"

fi

save and exit the file.

* + 1. sudo chmod +x /etc/keepalived/check\_apiserver.sh
    2. Take the backup of keepalived.conf file and then truncate the file.
       - sudo cp /etc/keepalived/keepalived.conf /etc/keepalived/keepalived.conf-org
       - sudo sh -c '> /etc/keepalived/keepalived.conf'
    3. Now paste the following contents to /etc/keepalived/keepalived.conf file
       - sudo vi /etc/keepalived/keepalived.conf

! /etc/keepalived/keepalived.conf

! Configuration File for keepalived

global\_defs {

router\_id LVS\_DEVEL

}

vrrp\_script check\_apiserver {

script "/etc/keepalived/check\_apiserver.sh"

interval 3

weight -2

fall 10

rise 2

}

vrrp\_instance VI\_1 {

state MASTER

interface enp0s3

virtual\_router\_id 151

priority 255

authentication {

auth\_type PASS

auth\_pass P@##D321!

}

virtual\_ipaddress {

192.168.1.45/24

}

track\_script {

check\_apiserver

}

}

* Save and close the file.

**Note: Only two parameters of this file need to be changed for master-2 & 3 nodes. State will become SLAVE for master 2 and 3, priority will be 254 and 253 respectively.**

* + 1. **Configure HAProxy on k8s-master-1 node, edit its configuration file and add the following contents:**
       - sudo cp /etc/haproxy/haproxy.cfg /etc/haproxy/haproxy.cfg-org
       - **Remove all lines after default section and add following lines**
       - sudo vi /etc/haproxy/haproxy.cfg

**#---------------------------------------------------------------------**

**# Example configuration for a possible web application. See the**

**# full configuration options online.**

**#**

**# https://www.haproxy.org/download/1.8/doc/configuration.txt**

**#**

**#---------------------------------------------------------------------**

**#---------------------------------------------------------------------**

**# Global settings**

**#---------------------------------------------------------------------**

**global**

**# to have these messages end up in /var/log/haproxy.log you will**

**# need to:**

**#**

**# 1) configure syslog to accept network log events. This is done**

**# by adding the '-r' option to the SYSLOGD\_OPTIONS in**

**# /etc/sysconfig/syslog**

**#**

**# 2) configure local2 events to go to the /var/log/haproxy.log**

**# file. A line like the following can be added to**

**# /etc/sysconfig/syslog**

**#**

**# local2.\* /var/log/haproxy.log**

**#**

**log 127.0.0.1 local2**

**chroot /var/lib/haproxy**

**pidfile /var/run/haproxy.pid**

**maxconn 4000**

**user haproxy**

**group haproxy**

**daemon**

**# turn on stats unix socket**

**stats socket /var/lib/haproxy/stats**

**# utilize system-wide crypto-policies**

**ssl-default-bind-ciphers PROFILE=SYSTEM**

**ssl-default-server-ciphers PROFILE=SYSTEM**

**#---------------------------------------------------------------------**

**# common defaults that all the 'listen' and 'backend' sections will**

**# use if not designated in their block**

**#---------------------------------------------------------------------**

**defaults**

**mode http**

**log global**

**option httplog**

**option dontlognull**

**option http-server-close**

**option forwardfor except 127.0.0.0/8**

**option redispatch**

**retries 3**

**timeout http-request 10s**

**timeout queue 1m**

**timeout connect 10s**

**timeout client 1m**

**timeout server 1m**

**timeout http-keep-alive 10s**

**timeout check 10s**

**maxconn 3000**

**frontend apiserver**

**bind \*:8443**

**mode tcp**

**option tcplog**

**default\_backend apiserver**

**backend apiserver**

**option httpchk GET /healthz**

**http-check expect status 200**

**mode tcp**

**option ssl-hello-chk**

**balance roundrobin**

**server kube-apiserver-1 172.21.90.69:6443 check # Replace the IP address with your own.**

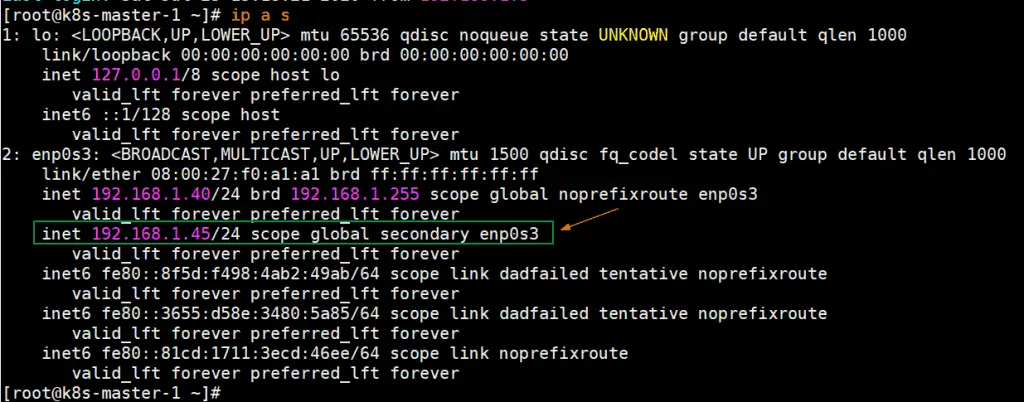
**server kube-apiserver-2 172.21.90.159:6443 check # Replace the IP address with your own.**

**server kube-apiserver-3 172.21.90.67:6443 check # Replace the IP address with your own.**

**Save and exit the file**

**For our convenience I have pasted the whole file above, please make changes as per your requirement.**

* + 1. **Now copy theses three files (check\_apiserver.sh , keepalived.conf and haproxy.cfg) from k8s-master-1 to k8s-master-2 & 3**
    2. **Run the following for loop to scp these files to master 2 and 3**
       - for f in k8s-master-2 k8s-master-3; do scp /etc/keepalived/check\_apiserver.sh /etc/keepalived/keepalived.conf root@$f:/etc/keepalived; scp /etc/haproxy/haproxy.cfg root@$f:/etc/haproxy; done
* **Note:** Don’t forget to change two parameters in keepalived.conf file that we discuss above for k8s-master-2 & 3
  + 1. Now Finally start and enable keepalived and haproxy service on all three master nodes using the following commands
       - sudo systemctl enable keepalived --now
       - sudo systemctl enable haproxy –now
    2. Once these services are started successfully, verify whether VIP (virtual IP) is enabled on k8s-master-1 node because we have marked k8s-master-1 as MASTER node in keepalived configuration file.

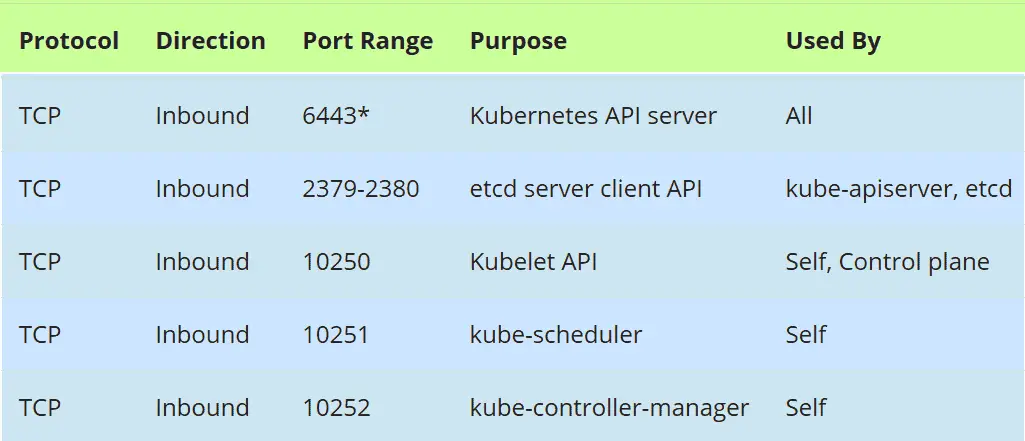


# ****Disable Swap, set SELinux as permissive and firewall rules for Master and worker nodes****

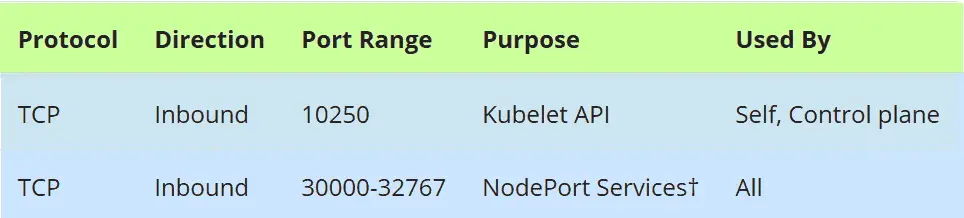
* + 1. Disable Swap Space on all the nodes including worker nodes, Run the following commands
       - sudo swapoff -a
       - sudo sed -i '/ swap / s/^\(.\*\)$/#\1/g' /etc/fstab
    2. Set SELinux as Permissive on all master and worker nodes, run the following commands,
       - sudo setenforce 0
       - sudo sed -i 's/^SELINUX=enforcing$/SELINUX=permissive/' /etc/selinux/config

Firewall Rules for Master Nodes:

In case firewall is running on master nodes, then allow the following ports in the firewall,



Firewall Rules for Worker nodes:



Install Container Run Time (CRI) Docker on Master & Worker Nodes

1. sudo yum install -y yum-utils
2. Load two required modules and add configuration to make them loadable at boot time.
   * sudo modprobe overlay
   * sudo modprobe br\_netfilter
3. cat <<EOF | sudo tee /etc/modules-load.d/containerd.conf

overlay

br\_netfilter

EOF

1. Set up other prerequisites.

* cat <<EOF | sudo tee /etc/sysctl.d/99-kubernetes-cri.conf

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

net.ipv4.ip\_forward = 1

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

EOF

1. Make the above settings applicable without restarting.
   * sudo sysctl --system

# Install containerd

Containerd - [Deploy Kubernetes Cluster on CentOS Stream with Containerd - HOSTAFRICA](https://www.hostafrica.com/blog/kubernetes/kubernetes-cluster-centos-stream-containerd/)

1. Add the official Docker repository.
   * sudo dnf config-manager --add-repo=https://download.docker.com/linux/centos/docker-ce.repo
2. Update your system and install the containerd package.
   * sudo dnf update
   * sudo dnf install -y containerd
3. Create a configuration file for containerd and set it to default.
   * sudo mkdir -p /etc/containerd
   * sudo containerd config default | sudo tee /etc/containerd/config.toml
4. Set cgroupdriver to system
   * sudo vi /etc/containerd/config.toml

Find the following section:

[plugins."io.containerd.grpc.v1.cri".containerd.runtimes.runc.options]

And change the value of SystemdCgroup to true

Once you are done, match the section in your file to the following:

[plugins."io.containerd.grpc.v1.cri".containerd.runtimes.runc.options]

BinaryName = ""

CriuImagePath = ""

CriuPath = ""

CriuWorkPath = ""

IoGid = 0

IoUid = 0

NoNewKeyring = false

NoPivotRoot = false

Root = ""

ShimCgroup = ""

SystemdCgroup = true

## Restart containerd : sudo systemctl restart containerd

Installing kubeadm, kubelet and kubectl

## References:

**Starting with Installing K8s –**

[How to Setup Kubernetes(k8s) Cluster in HA with Kubeadm (linuxtechi.com)](https://www.linuxtechi.com/setup-highly-available-kubernetes-cluster-kubeadm/)

[Kubernetes Multi-Master Node Cluster | by Lubomir Tobek | Medium](https://medium.com/@lubomir-tobek/kubernetes-multi-master-node-cluster-f2081e504983)

Containerd - [Deploy Kubernetes Cluster on CentOS Stream with Containerd - HOSTAFRICA](https://www.hostafrica.com/blog/kubernetes/kubernetes-cluster-centos-stream-containerd/)

## Set hostnames for all master and worker nodes as required.

## Also add hostnames for all in all master and worker nodes like below in /etc/hosts:

172.21.90.66 B3VM-UCTNS-WK00

172.21.90.69 B3VM-UCTNS-MS01

172.21.90.159 B3VM-UCTNS-MS02

172.21.90.67 B3VM-UCTNS-MS03

172.21.90.227 B3VM-UCTNS-WK01

172.21.90.188

Then do “bash exec”

Add the below repo

* vim /etc/yum.repos.d/kubernetes.repo

[kubernetes]

name=Kubernetes

baseurl=https://pkgs.k8s.io/core:/stable:/v1.29/rpm/

enabled=1

gpgcheck=1

gpgkey=https://pkgs.k8s.io/core:/stable:/v1.29/rpm/repodata/repomd.xml.key

exclude=kubelet kubeadm kubectl cri-tools kubernetes-cni

# *Now do install all these*

* yum install -y kubelet kubeadm kubectl --disableexcludes=Kubernetes
* sudo yum install -y yum-plugin-versionlock
* sudo yum versionlock add kubelet kubeadm kubectl

# *Now enable the kubelet service*

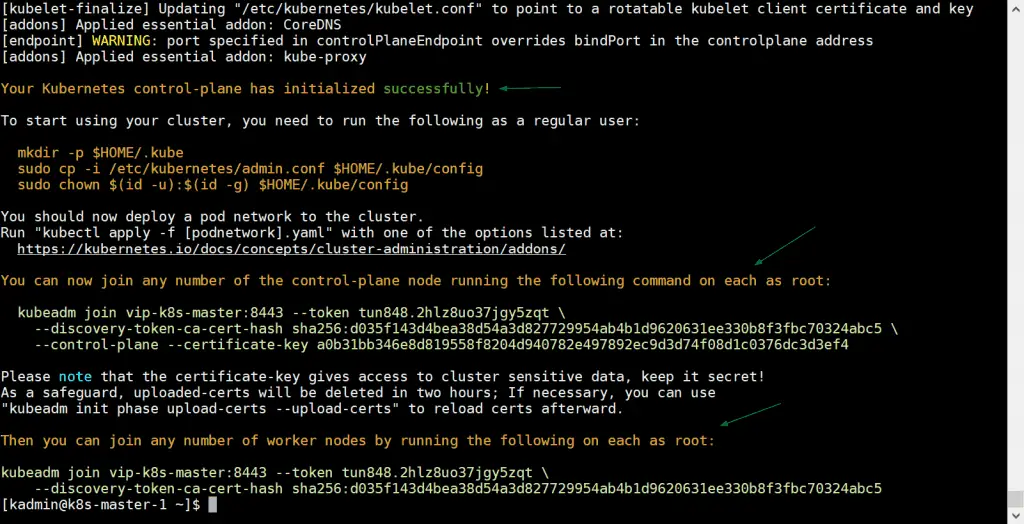
* sudo systemctl enable kubelet --now

Initialize the Kubernetes Cluster from first master node

We will be initializing the cluster from master1:

* sudo kubeadm init --control-plane-endpoint "172.21.90.188:8443" --upload-certs --pod-network-cidr=192.168.56.0/24 --service-cidr=192.168.57.0/24 --service-dns-domain=cluster.local --apiserver-advertise-address=172.21.90.69

Output of kubeadm command would be something like below:



**Great, above output confirms that Kubernetes cluster has been initialized successfully. In output we also got the commands for other master and worker nodes to join the cluster.**

**Note:** It is recommended to copy this output to a text file for future reference.

Run following commands to allow local user to use kubectl command to interact with cluster,

[kadmin@k8s-master-1 ~]$ mkdir -p $HOME/.kube

[kadmin@k8s-master-1 ~]$ sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

[kadmin@k8s-master-1 ~]$ sudo chown $(id -u):$(id -g) $HOME/.kube/config

Now, Let’s deploy pod network (CNI – Container Network Interface), in my case I going to deploy calico addon as pod network, run following kubectl command

We will be applying the Calico CNI:

Goto <https://docs.tigera.io/calico/latest/getting-started/kubernetes/self-managed-onprem/onpremises>

1. Download the Calico networking manifest for the Kubernetes API datastore.
   * curl https://raw.githubusercontent.com/projectcalico/calico/v3.27.2/manifests/calico.yaml -O
2. If you are using pod CIDR 192.168.0.0/16, skip to the next step. If you are using a different pod CIDR with kubeadm, no changes are required — Calico will automatically detect the CIDR based on the running configuration. For other platforms, make sure you uncomment the CALICO\_IPV4POOL\_CIDR variable in the manifest and set it to the same value as your chosen pod CIDR.
3. updated this file with below changes:

* calico\_bakend: "vxlan"
* - name: CALICO\_IPV4POOL\_IPIP

value: "Never"

* - name: CALICO\_IPV4POOL\_VXLAN

value: "Always"

* name: CALICO\_IPV6POOL\_VXLAN

value: "Always"

* #- -bird-live
* #- -bird-ready
* IP\_AUTODETECTION\_METHOD

env:

- name: IP\_AUTODETECTION\_METHOD

value: "interface=ens192\*"

# Use Kubernetes API as the backing datastore.

- name: DATASTORE\_TYPE

value: "kubernetes"

# Wait for the datastore.

- name: WAIT\_FOR\_DATASTORE

value: "true"

# Set based on the k8s node name.

- name: NODENAME

valueFrom:

fieldRef:

fieldPath: spec.nodeName

# Choose the backend to use.

- name: CALICO\_NETWORKING\_BACKEND

valueFrom:

configMapKeyRef:

name: calico-config

key: calico\_backend

We have set interface as ens192 but you need to set as per your requirements:

1. Customize the manifest as necessary.
2. Apply the manifest using the following command.
   * kubectl apply -f calico.yaml

Once the pod network is deployed successfully, add remaining two master nodes to cluster. Just copy the command for master node to join the cluster from the output and paste it on k8s-master-2 and k8s-master-3, example is shown below

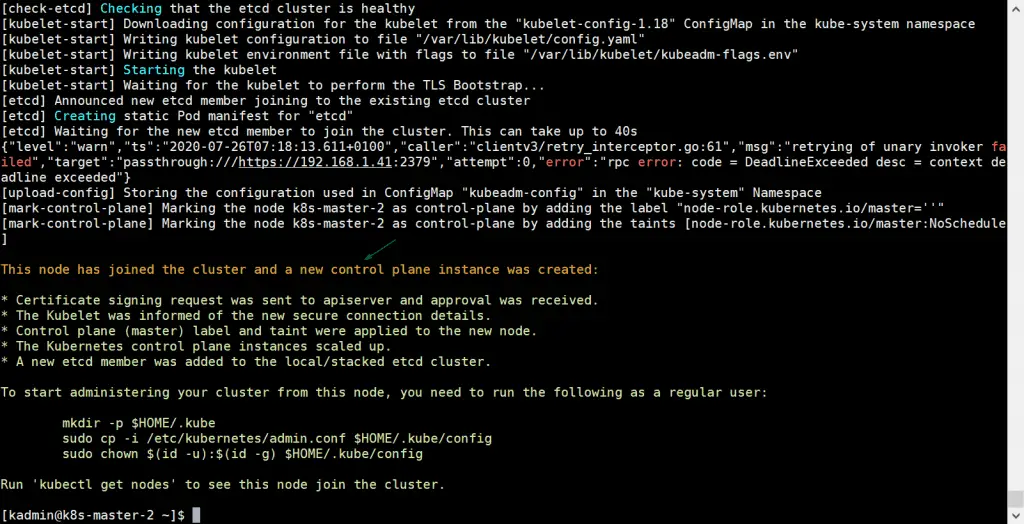
Note the kubeadm join command does not contain this tag: “--apiserver-advertise-address”. Please add it before running the join command and this tag will contain the ip of master node on which you are running the join command.

kubeadm join 172.21.90.188:8443 --token ps5814.ww340ycu72c1lrc2 \

--discovery-token-ca-cert-hash sha256:bf054eff298a6a827b32f0525766e0e69767ed3ebd6254fdd7bdb375c4fa101e \

--control-plane --certificate-key 2974ae913903d9eca3ad384fdfa38c71443e2956d6ad01d97e7c16f307014e02 --apiserver-advertise-address=172.21.90.67

**Output would be:**

****

Do the same on master 3 node also, remember to replace the IP of tag “--apiserver-advertise-address”

Above output confirms that k8s-master-3 has also joined the cluster successfully. Let’s verify the nodes status from kubectl command, go to master-1 node and execute below command

[kadmin@k8s-master-1 ~]$ kubectl get nodes

NAME STATUS ROLES AGE VERSION

k8s-master-1 Ready master 31m v1.18.6

k8s-master-2 Ready master 10m v1.18.6

k8s-master-3 Ready master 3m47s v1.18.6

Join Worker nodes to Kubernetes cluster

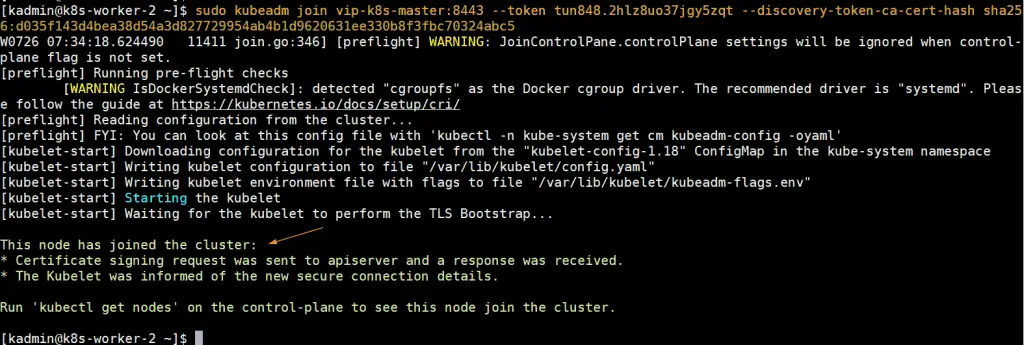
To join worker nodes to cluster, copy the command for worker node from output and past it on both worker nodes, example is shown below:

kubeadm join 172.21.90.188:8443 --token ps5814.ww340ycu72c1lrc2 \

--discovery-token-ca-cert-hash sha256:bf054eff298a6a827b32f0525766e0e69767ed3ebd6254fdd7bdb375c4fa101e

[root@B3VM-UCTNS-MS01 cni]# sudo systemctl enable kubelet –now

Output would be something like below:



Now head to k8s-master-1 node and run below kubectl command to get status worker nodes,

[kadmin@k8s-master-1 ~]$ kubectl get nodes

NAME STATUS ROLES AGE VERSION

k8s-master-1 Ready master 43m v1.18.6

k8s-master-2 Ready master 21m v1.18.6

k8s-master-3 Ready master 15m v1.18.6

k8s-worker-1 Ready <none> 6m11s v1.18.6

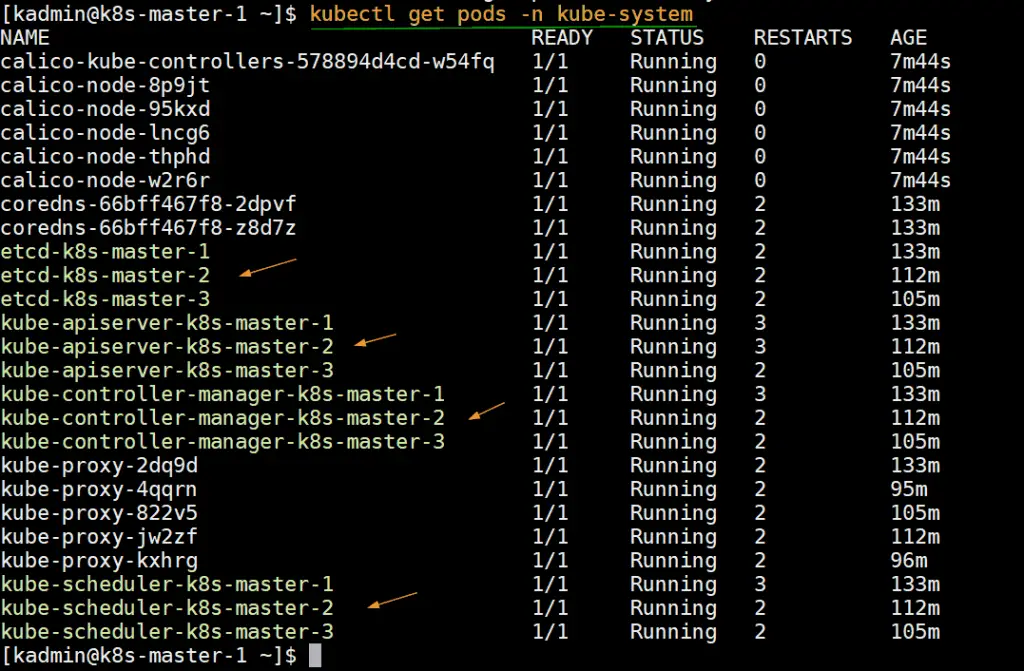
k8s-worker-2 Ready <none> 5m22s v1.18.6

[kadmin@k8s-master-1 ~]$

Above output confirms that both workers have also joined the cluster and are in ready state.

Run below command to verify the status infra pods which are deployed in kube-system namespace.

[kadmin@k8s-master-1 ~]$ kubectl get pods -n kube-system



**We need to install Docker private registry for this to store all of our images privately. Please consider opening “Docker Private Registry” document for it.**

**Installing Docker on all nodes to build and create docker images: (You can use containerd also):**

sudo yum install -y yum-utils

sudo yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo

sudo yum install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin

sudo systemctl start docker