**How to Install a Kubernetes Cluster on CentOS 8**

The process of installing a **Kubernetes Cluster** on **CentOS 8** is almost similar to that of **CentOS 7** (which you can go through [here](https://www.tecmint.com/install-kubernetes-cluster-on-centos-7/)), but the process here has a few changes. These changes, mostly revolve around the installation of **Docker**.

Starting from **CentOS 8** (and by extension **RHEL 8**), **docker** has now natively been replaced by [podman](https://www.tecmint.com/manage-containers-using-podman-in-rhel/) and **buildah** which are tools from **Redhat**. As a matter of fact, the docker package has now been removed from the default package repository.

With this move, the **Redhat** team aims to simplify the process of creating and using containers, without needing special permissions, while at the same time, maintaining compatibility with docker images and operating them without needing a daemon. **Podman** promises to offer more flexibility when running on **Kubernetes** environments, but the jury remains out there.

For this article, we will run through the process of installing **Kubernetes** on a **CentOS 8** platform, running on **Docker-CE** (**Community Edition**). In a later article, we will also run through a similar installation, using **podman** for our containers.

**Prerequisites**

Three [servers running CentOS 8](https://www.tecmint.com/centos-8-installation/) – **1 Master Node** and **2 Worker Nodes**.

It is recommended that your nodes should have at least 2 CPUs with 2GB RAM or more per machine. This is not a strict requirement but is largely driven by the needs of the application you intend to run.

Internet connectivity on all your nodes. We will be fetching **Kubernetes** and **docker** packages from the repository. Equally, you will need to make sure that the [DNF package manager](https://www.tecmint.com/dnf-commands-for-fedora-rpm-package-management/) is installed by default and can fetch packages remotely.

All your nodes should also be able to connect to one another, either on a private or public network, whichever is available.

You will also need access to an account with sudo or root privileges. In this tutorial, I will be using my root account.

**Precaution**

Most nodes generally come with unique **MAC** addresses, however, in some unique cases, some **Virtual Machines** may have identical MAC addresses. It is therefore recommended that you confirm that the **Product\_UUID** and the **MAC** address are not identical in any of the nodes.

**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.

To check the MAC address of the network interface and compare it.

# ip link

To check the **product\_uuid** and compare, run the following command.

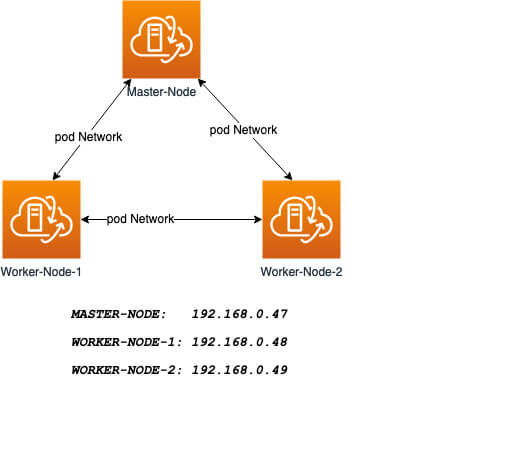
# cat /sys/class/dmi/id/product\_uuid

Logical Architecture

Our installation is designed to have the **Master-Node** controlling the **Worker Nodes**. At the end of this installation, our logical architecture will look something like this.

**Master Node** – This machine generally acts as the control plane and runs the cluster database and the API server (which the kubectl CLI communicates with).

Our 3-node **Kubernetes Cluster** will look something like this:

[](https://www.tecmint.com/wp-content/uploads/2020/01/Kubernetes-Cluster-Setup-Diagram.jpg)Kubernetes Cluster Diagram

Installation of Kubernetes Cluster on Master-Node

For **Kubernetes** to work, you will need a containerization engine. As mentioned, we will be using **Docker-CE**.

The following institutions will be performed on **CentOS 8 Master-Node**.

Step 1: Prepare Hostname, Firewall, and SELinux

On your **CentOS 8 Master-Node**, set the system hostname and update DNS in your **/etc/hosts** file.

# hostnamectl set-hostname master-node

# cat <<EOF>> /etc/hosts

192.168.0.47 master-node

192.168.0.48 node-1 worker-node-1

192.168.0.49 node-2 worker-node-2

EOF

Next, ping your **worker-node-1** and **worker-node-2** to check if your updated host file is working correctly using the [ping command](https://www.tecmint.com/linux-ping-command-examples/).

# ping 192.168.0.48

# ping 192.168.0.49

Next, disable **Selinux**, as this is required to allow containers to access the host filesystem, which is needed by pod networks and other services.

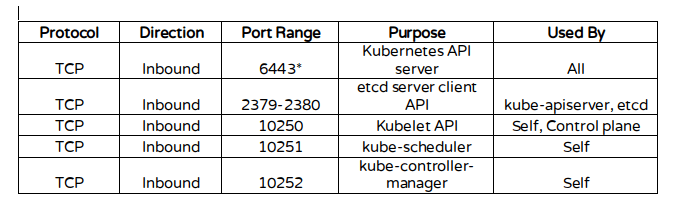
# setenforce 0

Setting **setenforce** to 0 effectively sets SELinux to permissive, which effectively disables SELinux until the next reboot. To completely disable it, use the below command and reboot.

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

# reboot

Kubernetes makes use of various ports for communication and access and these ports need to be accessible to Kubernetes and not limited by the firewall.

[](https://www.tecmint.com/wp-content/uploads/2020/01/Kubernetes-Ports.png)

Kubernetes Ports

Configure the firewall rules on the ports.

# 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

# echo '1' > /proc/sys/net/bridge/bridge-nf-call-iptables

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

$ sudo firewall-cmd --permanent --add-port=30000-32767/tcp

$ sudo firewall-cmd --reload

$ sudo modprobe br\_netfilter

$ sudo sh -c "echo '1' > /proc/sys/net/bridge/bridge-nf-call-iptables"

$ sudo sh -c "echo '1' > /proc/sys/net/ipv4/ip\_forward"

Step 2: Install Docker-CE on CentOS 8

You will need to add the **Docker** repository first as it is no longer in the default package list using the following **dnf config-manager** command.

# dnf config-manager --add-repo=https://download.docker.com/linux/centos/docker-ce.repo

Also install **containerd.io** package which is available as a daemon that manages the complete container lifecycle of its host system, from image transfer and storage to container execution and supervision to low-level storage to network attachments and beyond.

# dnf install https://download.docker.com/linux/centos/7/x86\_64/stable/Packages/containerd.io-1.2.6-3.3.el7.x86\_64.rpm

Now install the latest version of a **docker-ce** package.

# dnf install docker-ce

You can now enable and start the docker service.

# systemctl enable docker

# systemctl start docker

Step 3: Install Kubernetes (Kubeadm) on CentOS 8

Next, you will need to add **Kubernetes** repositories manually as they do not come installed by default on **CentOS 8**.

# cat <<EOF > /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

EOF

**Kubeadm** helps you bootstrap a minimum viable Kubernetes cluster that conforms to best practices. With **kubeadm**, your cluster should pass the Kubernetes Conformance tests.

**Kubeadm** also supports other cluster lifecycle functions, such as upgrades, downgrade, and managing bootstrap tokens. Kubeadm is also integration-friendly with other orchestration tools like **Ansible** and **Terraform**.

With the package repo now ready, you can go ahead and install **kubeadm** package.

# dnf install kubeadm -y

When the installation completes successfully, enable and start the service.

# systemctl enable kubelet

# systemctl start kubelet

Step 4: Create a control-plane Master with kubeadm

The Kubernetes master which acts as the **control plane** for the cluster runs a few critical services necessary for the cluster. As such, the initialization process will do a series of prechecks to ensure that the machine is ready to run Kubernetes. These prechecks expose warnings and exit on errors. **kubeadm init** then downloads and installs the cluster control plane components.

Now it’s time to initialize Kubernetes master, but before that, you must disable swap in order to run “**kubeadm init**“ command.

# swapoff -a

Initializing **Kubernetes** master is a completely automated process that is controlled by the “**kubeadm init**“ command as shown.

**C**reate cluster configuration

kubeadm config print init-defaults | tee ClusterConfiguration.yaml

**M**odify ClusterConfiguration.yaml, **replace 192.168.254.20 with your Control Plane's IP address**

sudo sed -i '/name/d' ClusterConfiguration.yaml  
sudo sed -i 's/ advertiseAddress: 1.2.3.4/ advertiseAddress: 192.168.254.20/' ClusterConfiguration.yaml  
sudo sed -i 's/ criSocket: \/var\/run\/dockershim\.sock/ criSocket: \/run\/containerd\/containerd\.sock/' ClusterConfiguration.yaml

sudo cat << EOF | cat >> ClusterConfiguration.yaml

---

apiVersion: kubelet.config.k8s.io/v1beta1

kind: KubeletConfiguration

cgroupDriver: systemd

EOF

[root@master01 ~]# cat ClusterConfiguration.yaml

apiVersion: kubeadm.k8s.io/v1beta3

bootstrapTokens:

- groups:

- system:bootstrappers:kubeadm:default-node-token

token: abcdef.0123456789abcdef

ttl: 24h0m0s

usages:

- signing

- authentication

kind: InitConfiguration

localAPIEndpoint:

advertiseAddress: 172.16.102.21

bindPort: 6443

nodeRegistration:

criSocket: unix:///var/run/containerd/containerd.sock

imagePullPolicy: IfNotPresent

taints: null

---

apiServer:

timeoutForControlPlane: 4m0s

apiVersion: kubeadm.k8s.io/v1beta3

certificatesDir: /etc/kubernetes/pki

clusterName: kubernetes

controllerManager: {}

dns: {}

etcd:

local:

dataDir: /var/lib/etcd

imageRepository: k8s.gcr.io

kind: ClusterConfiguration

kubernetesVersion: 1.24.0

networking:

dnsDomain: cluster.local

serviceSubnet: 10.96.0.0/12

scheduler: {}

---

apiVersion: kubelet.config.k8s.io/v1beta1

kind: KubeletConfiguration

cgroupDriver: systemd

**C**reate kubernetes cluster

sudo kubeadm init --config=ClusterConfiguration.yaml --cri-socket /run/containerd/containerd.sock 🡪 Optional

kubeadm init --config=ClusterConfiguration.yaml --v=5

**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

kubectl apply -f <https://docs.projectcalico.org/manifests/calico.yaml>

kubectl get nodes

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

kubeadm join 172.16.102.21:6443 --token abcdef.0123456789abcdef \

--discovery-token-ca-cert-hash sha256:3d0dcb01fec70d0de70e585b8897f717aadf2f5f5dcb3b44ce4dde53885d0565

kubectl get pods -A

Issues which i faced

[root@master01 ~]# kubeadm config images pull --v=5

I0701 16:54:23.861766 2992 initconfiguration.go:117] detected and using CRI socket: unix:///var/run/containerd/containerd.sock

I0701 16:54:23.861952 2992 interface.go:432] Looking for default routes with IPv4 addresses

I0701 16:54:23.861961 2992 interface.go:437] Default route transits interface "ens32"

I0701 16:54:23.862060 2992 interface.go:209] Interface ens32 is up

I0701 16:54:23.862133 2992 interface.go:257] Interface "ens32" has 2 addresses :[172.16.102.21/27 fe80::250:56ff:fe8f:b704/64].

I0701 16:54:23.862160 2992 interface.go:224] Checking addr 172.16.102.21/27.

I0701 16:54:23.862169 2992 interface.go:231] IP found 172.16.102.21

I0701 16:54:23.862177 2992 interface.go:263] Found valid IPv4 address 172.16.102.21 for interface "ens32".

I0701 16:54:23.862184 2992 interface.go:443] Found active IP 172.16.102.21

I0701 16:54:23.862209 2992 kubelet.go:214] the value of KubeletConfiguration.cgroupDriver is empty; setting it to "systemd"

I0701 16:54:23.872648 2992 version.go:186] fetching Kubernetes version from URL: https://dl.k8s.io/release/stable-1.txt

exit status 1

output: E0701 16:54:24.599939 3016 remote\_image.go:218] "PullImage from image service failed" err="rpc error: code = Unimplemented desc = unknown service runtime.v1alpha2.ImageService" image="k8s.gcr.io/kube-apiserver:v1.24.2"

time="2022-07-01T16:54:24+05:30" level=fatal msg="pulling image: rpc error: code = Unimplemented desc = unknown service runtime.v1alpha2.ImageService"

, error

[root@master01 ~]# crictl ps

WARN[0000] runtime connect using default endpoints: [unix:///var/run/dockershim.sock unix:///run/containerd/containerd.sock unix:///run/crio/crio.sock unix:///var/run/cri-dockerd.sock]. As the default settings are now deprecated, you should set the endpoint instead.

ERRO[0000] unable to determine runtime API version: rpc error: code = Unavailable desc = connection error: desc = "transport: Error while dialing dial unix /var/run/dockershim.sock: connect: no such file or directory"

WARN[0000] image connect using default endpoints: [unix:///var/run/dockershim.sock unix:///run/containerd/containerd.sock unix:///run/crio/crio.sock unix:///var/run/cri-dockerd.sock]. As the default settings are now deprecated, you should set the endpoint instead.

ERRO[0000] unable to determine image API version: rpc error: code = Unavailable desc = connection error: desc = "transport: Error while dialing dial unix /var/run/dockershim.sock: connect: no such file or directory"

E0701 16:56:44.995355 3120 remote\_runtime.go:536] "ListContainers with filter from runtime service failed" err="rpc error: code = Unimplemented desc = unknown service runtime.v1alpha2.RuntimeService" filter="&ContainerFilter{Id:,State:&ContainerStateValue{State:CONTAINER\_RUNNING,},PodSandboxId:,LabelSelector:map[string]string{},}"

FATA[0000] listing containers: rpc error: code = Unimplemented desc = unknown service runtime.v1alpha2.RuntimeService

Solutions:

[root@worker01 ~]# vim /usr/lib/systemd/system/kubelet.service.d/10-kubeadm.conf

Environment="KUBELET\_EXTRA\_ARGS=--runtime-request-timeout=15m --image-service-endpoint=unix:///run/containerd/containerd.sock --cgroup-driver=systemd"

[root@worker01 ~]# rm /etc/containerd/config.toml

[root@worker01 ~]# systemctl restart containerd