

# Kubernetes Setup Using Kubeadm

## To set up a cluster on AWS using Kubeadm, you need the following:

- A compatible Linux host.
- 2 GB or more of RAM per machine and at least 2 CPUs.
- Full network connectivity between all machines in the cluster.
- Unique hostname for each host. Change hostname of the machines using hostnamectl.
- Ensure that certain ports are open on your machines.
- Disable swap. Disabling swap is essential for the kubelet to work properly.
- Install Containerd on all machines.

## Kubernetes Setup Using Kubeadm In AWS EC2 instance

Prerequisites:-	ubuntu:latest
1 - Control plane	t2.medium and above
2 - Worker nodes	t2.micro and above

Note: Open below required Ports In AWS Security Groups

Reference link : <https://kubernetes.io/docs/reference/ports-and-protocols/>

### Kubernetes Master Server required Ports

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	10259	kube-scheduler	Self
TCP	Inbound	10257	kube-controller-manager	Self

### Slave/worker nodes required Ports

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

## Below steps are common for both master server and worker nodes.

- Switch to root user

```
sudo -i
```

- Use 'hostnamectl' command to change the hostname of both the server and worker nodes

```
hostnamectl set-hostname kubserver = controlplane  
  
hostnamectl set-hostname worker1    = worker1/node01  
  
hostnamectl set-hostname worker2    = worker2/node02
```

- To apply the changes, please log out of and then log back into the instances.

```
sudo -i
```

- we must disable swap in order for the kubelet to work properly

```
swapoff -a
```

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

## Installing a container runtime

we can install containerd in multiple ways

### 1. from apt-get

```
sudo apt update  
sudo apt install containerd
```

### 2. From the official binaries

#### Step 1: Installing containerd

1. Download the containerd-\*.tar.gz archive from  
<https://github.com/containerd/containerd/releases>

```
wget https://github.com/containerd/containerd/releases/download/v1.7.5/containerd-1.7.5-linux-amd64.tar.gz
```

2. extract it under /usr/local

```
tar Cxzf /usr/local containerd-1.7.5-linux-amd64.tar.gz
```

### 3. setting up containerd as systemd service

```
mkdir -p /usr/local/lib/systemd/system
wget -P /usr/local/lib/systemd/system/
https://raw.githubusercontent.com/containerd/containerd/main/containerd.service

systemctl daemon-reload
systemctl enable --now containerd
```

## Step 2: Installing runc

1. Download the runc binary from <https://github.com/opencontainers/runc/releases> , and install it as /usr/local/sbin/runc

```
wget https://github.com/opencontainers/runc/releases/download/v1.1.9/runc.amd64
install -m 755 runc.amd64 /usr/local/sbin/runc
```

## Step 3: Installing CNI plugins

1. Download the cni-plugins-\*.tgz archive from <https://github.com/containernetworking/plugins/releases> , and extract it under /opt/cni/bin

```
wget https://github.com/containernetworking/plugins/releases/download/v1.3.0/cni-plugins-linux-amd64-v1.3.0.tgz

mkdir -p /opt/cni/bin
tar Cxzf /opt/cni/bin cni-plugins-linux-amd64-v1.3.0.tgz
```

## Step 4: Installing crictl [Cli tool]

```
wget https://github.com/kubernetes-sigs/cri-tools/releases/download/v1.28.0/crictl-v1.28.0-linux-amd64.tar.gz

sudo tar zxvf crictl-v1.28.0-linux-amd64.tar.gz -C /usr/local/bin
rm -f crictl-v1.28.0-linux-amd64.tar.gz

cat <<EOF | sudo tee /etc/crictl.yaml
runtime-endpoint: unix:///run/containerd/containerd.sock
image-endpoint: unix:///run/containerd/containerd.sock
timeout: 2
debug: false
pull-image-on-create: false
EOF
```

## Forwarding IPv4 and letting iptables see bridged traffic

- Execute the below mentioned instructions

```
cat <<EOF | sudo tee /etc/modules-load.d/k8s.conf
overlay
br_netfilter
EOF

sudo modprobe overlay
sudo modprobe br_netfilter

# sysctl params required by setup, params persist across reboots
cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf
net.bridge.bridge-nf-call-iptables = 1
net.bridge.bridge-nf-call-ip6tables = 1
net.ipv4.ip_forward = 1
EOF

# Apply sysctl params without reboot
sudo sysctl --system

sysctl net.bridge.bridge-nf-call-iptables net.bridge.bridge-nf-call-ip6tables
net.ipv4.ip_forward
modprobe br_netfilter
sysctl -p /etc/sysctl.conf
```

- Verify that the br\_netfilter, overlay modules are loaded by running the following commands:

```
lsmod | grep br_netfilter
lsmod | grep overlay
```

## Installing kubeadm:

**Below steps are common for both master server and worker nodes.**

1. Update the APT package index and install the necessary packages to enable the use of the Kubernetes APT repository.
-

```
sudo apt-get update  
apt-get update && sudo apt-get install -y apt-transport-https curl
```

2. Download the public signing key for the Kubernetes package repositories

```
mkdir -p /etc/apt/keyrings/  
curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.28/deb/Release.key | sudo gpg --  
dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg
```

3. Add the appropriate Kubernetes apt repository:

```
echo 'deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg]  
https://pkgs.k8s.io/core:/stable:/v1.28/deb/ /' | sudo tee  
/etc/apt/sources.list.d/kubernetes.list
```

4. Update the apt package index, install kubelet, kubeadm and kubectl

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

5. apt-mark hold will prevent the package from being automatically upgraded or removed.

```
apt-mark hold kubelet kubeadm kubectl containerd
```

**From here below steps are only for master server. make sure you execute them only on master.**

**1. downloading component images on master.**

```
kubeadm config images pull
```

**2. Initializing your control-plane node**

- The control-plane node is the machine where the control plane components run, including etcd (the cluster database) and the API Server (which the kubectl command line tool communicates with).
- If you have plans to upgrade this single control-plane kubeadm cluster to high availability you should specify the `--control-plane-endpoint` to set the shared endpoint for all control-plane nodes. Such an endpoint can be either a DNS name or an IP address of a load-balancer.

```
kubeadm init
```

```

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 172.31.24.155:6443 --token yolpa1.zky7ws22p1kk1e22 \
--discovery-token-ca-cert-hash sha256:86ac30b64f12f6f24b10ac36bb9a881ee5c813321d894871507d90501c037871
root@controlplane:~#

```

### Note: Copy the join token and save it

```

kubeadm join 172.31.24.155:6443 --token yolpa1.zky7ws22p1kk1e22 \
--discovery-token-ca-cert-hash
sha256:86ac30b64f12f6f24b10ac36bb9a881ee5c813321d894871507d90501c037871

```

### 3. 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
```

### 4. To verify, if kubectl is working or not, run the following command.

```
kubectl get pods
```

```

root@controlplane:~# kubectl get pods -n kube-system
NAME                                READY   STATUS    RESTARTS   AGE
coredns-5dd5756b68-29br9           0/1     Pending   0           5m32s
coredns-5dd5756b68-fr8sw           0/1     Pending   0           5m32s
etcd-controlplane                  1/1     Running   0           5m36s
kube-apiserver-controlplane         1/1     Running   0           5m36s
kube-controller-manager-controlplane 1/1     Running   0           5m36s
kube-proxy-sfkkm                   1/1     Running   0           5m33s
kube-scheduler-controlplane         1/1     Running   0           5m36s
root@controlplane:~#

```

- You will notice from the previous command, that all the pods are running except one:

‘coredns’.

- For resolving this we will install a # pod network.

## 5. Installing pod network

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

**weave net pod network, execute below command**

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

```
root@controlplane:~# kubectl get pods -n kube-system
NAME                                READY   STATUS    RESTARTS   AGE
coredns-5dd5756b68-29br9           1/1     Running   0           9m20s
coredns-5dd5756b68-fr8sw           1/1     Running   0           9m20s
etcd-controlplane                  1/1     Running   0           9m24s
kube-apiserver-controlplane         1/1     Running   0           9m24s
kube-controller-manager-controlplane 1/1     Running   0           9m24s
kube-proxy-sfkkm                   1/1     Running   0           9m21s
kube-scheduler-controlplane         1/1     Running   0           9m24s
weave-net-j82zn                     2/2     Running   1 (3m2s ago) 3m12s
root@controlplane:~#
```

5. To check the status of kube server, run this command

```
kubectl get nodes
```

```
root@controlplane:~# kubectl get nodes
NAME             STATUS    ROLES    AGE   VERSION
controlplane     Ready    control-plane 10m   v1.28.1
root@controlplane:~#
```

## 7. Add Worker Machines to Kubernetes Master

- Copy kubeadm join token from server-command-line and execute in Worker Nodes to join nodes to cluster

```
kubeadm join 172.31.24.155:6443 --token yo1pa1.zky7ws22p1kk1e22 \
--discovery-token-ca-cert-hash
sha256:86ac30b64f12f6f24b10ac36bb9a881ee5c813321d894871507d90501c037871
```



```

root@worker1:~# kubeadm join 172.31.24.155:6443 --token yolpa1.zky7ws22p1kk1e22 \
--discovery-token-ca-cert-hash sha256:86ac30b64f12f6f24b10ac36bb9a881ee5c813321d89487
1507d90501c037871
[preflight] Running pre-flight checks
[preflight] Reading configuration from the cluster...
[preflight] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm
-config -o yaml'
[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] Starting the kubelet
[kubelet-start] Waiting for the kubelet to perform the TLS Bootstrap...

This node has joined the cluster:
* Certificate signing request was sent to apiservert 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@worker1:~# █

```

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.

8. To verify the worker node status, run 'kubectl get nodes' on the control-plane

```

kubectl get nodes

```

NAME	STATUS	ROLES	AGE	VERSION
controlplane	Ready	control-plane	19m	v1.28.1
worker1	Ready	<none>	44s	v1.28.1

```

root@controlplane:~# █

```