**Kubernetes Cluster Setup Using Kubeadm**

Following are the high-level steps involved in setting up a kubeadm-based Kubernetes cluster.

1. Install container runtime on all nodes- We will be using [cri-o](https://cri-o.io/).
2. Install Kubeadm, Kubelet, and kubectl on all the nodes.
3. Initiate Kubeadm control plane configuration on the master node.
4. Save the node join command with the token.
5. Install the Calico network plugin.
6. Join the worker node to the master node (control plane) using the join command.
7. Validate all cluster components and nodes.
8. Install Kubernetes Metrics Server
9. Deploy a sample app and validate the app

All the steps given in this guide are referred from the official Kubernetes documentation and related GitHub project pages. If you want to understand every cluster component in detail, refer to the comprehensive [Kubernetes Architecture](https://devopscube.com/kubernetes-architecture-explained/). Now let’s get started with the setup.

**Enable iptables Bridged Traffic on all the Nodes**

Execute the following commands on **all the nodes** for IPtables to see bridged traffic.

* 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 = 1net.ipv4.ip\_forward = 1 EOF
* # Apply sysctl params without reboot
* sudo sysctl --system

**Disable swap on all the Nodes**

For kubeadm to work properly, you need to disable swap on all the nodes using the following command.

* sudo swapoff –a (crontab -l 2>/dev/null; echo "@reboot /sbin/swapoff -a") | crontab - || true

The fstab entry will make sure the swap is off on system reboots. You can also, control swap errors using the kubeadm parameter --ignore-preflight-errors Swap we will look at it in the latter part.

**Install CRI-O Runtime On All The Nodes**

The basic requirement for a Kubernetes cluster is a [container runtime](https://devopscube.com/what-is-docker/). You can have any one of the following container runtimes.

1. CRI-O
2. contained
3. Docker Engine (using cri-dockerd)

We will be using CRI-O instead of [Docker](https://devopscube.com/what-is-docker/) for this setup as [Kubernetes deprecated Docker engine](https://kubernetes.io/blog/2022/02/17/dockershim-faq/). As a first step, we need to install cri-o on all the nodes. Execute the following commands on all the nodes.

Create the .conf file to load the modules at boot up

* cat <<EOF | sudo tee / etc/modules-load.d/crio.conf overlay br\_netfilter EOF
* # Set up required sysctl params, these persist across reboots.
* 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 = 1EOF

Execute the following commands to enable overlayFS & VxLan pod communication.

* sudo modprobe overlay
* sudo modprobe br\_netfilter

Set up required sysctl params, these persist across reboots.

* 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

Reload the parameters.

* sudo sysctl –system nable **cri-o** repositories for version 1.23 OS="xUbuntu\_20.04" VERSION="1.23"
* cat <<EOF | sudo tee /etc/apt/sources.list.d/devel:kubic:libcontainers:stable.list deb https://download.opensuse.org/repositories/devel:/kubic:/libcontainers:/stable/$OS/ /EOF
* cat <<EOF | sudo tee /etc/apt/sources.list.d/devel:kubic:libcontainers:stable:cri-o:$VERSION.list deb http://download.opensuse.org/repositories/devel:/kubic:/libcontainers:/stable:/cri-o:/$VERSION/$OS/ /EOF

Add the gpg keys.

* curl -L https://download.opensuse.org/repositories/devel:kubic:libcontainers:stable:cri-o:$VERSION/$OS/Release.key | sudo apt-key --keyring /etc/apt/trusted.gpg.d/libcontainers.gpg add –
* curl -L https://download.opensuse.org/repositories/devel:/kubic:/libcontainers:/stable/$OS/Release.key | sudo apt-key --keyring /etc/apt/trusted.gpg.d/libcontainers.gpg add -

Update and install crio and crio-tools.

* sudo apt-get update
* sudo apt-get install cri-o cri-o-runc cri-tools -y

Reload the systemd configurations and enable cri-o.

sudo systemctl daemon-reload

sudo systemctl enable crio --now

The cri-tools contain **crictl**, a CLI utility to interact with the containers created by the contianer runtime. When you use container runtimes other than Docker, you can use the crictl utility to debug containers on the nodes. Also, it is useful in CKS certification where you need to debug containers.

**Install Kubeadm & Kubelet & Kubectl on all Nodes**

Install the required dependencies :

* sudo apt-get update
* sudo apt-get install -y apt-transport-https ca-certificates curl
* sudo curl -fsSLo /usr/share/keyrings/kubernetes-archive-keyring.gpg https://packages.cloud.google.com/apt/doc/apt-key.gpg

Add the GPG key and apt repository.

* echo "deb [signed-by=/usr/share/keyrings/kubernetes-archive-keyring.gpg] https://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee /etc/apt/sources.list.d/kubernetes.list

Update apt and install the latest version of kubelet, kubeadm, and kubectl.

* sudo apt-get update –y
* sudo apt-get install -y kubelet kubeadm kubectl

**Note**: If you are preparing for Kubernetes certification, install the specific version of kubernetes. For example, the current Kubernetes version for CKA, CKAD and CKS exams is kubernetes version 1.26

You can use the following commands to find the latest versions.

* sudo apt update
* apt-cache madison kubeadm | tac

Specify the version as shown below.

* sudo apt-get install -y kubelet=1.26.1-00 kubectl=1.26.1-00 kubeadm=1.26.1-00

Add hold to the packages to prevent upgrades.

* sudo apt-mark hold kubelet kubeadm kubectl

Now we have all the required utilities and tools for configuring Kubernetes components using kubeadm.

Add the node IP to KUBELET\_EXTRA\_ARGS.

* sudo apt-get install -y jq local\_ip="$(ip --json a s | jq -r '.[] | if .ifname == "eth1" then .addr\_info[] | if .family == "inet" then .local else empty end else empty end')" cat > /etc/default/kubelet << EOF
* KUBELET\_EXTRA\_ARGS=--node-ip=$local\_ip EOF

**Initialize Kubeadm On Master Node To Setup Control Plane**

Here you need to consider two options.

1. **Master Node with Private IP:** If you have nodes with only private IP addresses and the API server would be accessed over the private IP of the master node.
2. **Master Node With Public IP:**If you are setting up aKubeadm cluster on Cloud platforms and you need master Api server access over the Public IP of the master node server.

Only the Kubeadm initialization command differs for Public and Private IPs.

Execute the commands in this section only on the master node.

If you are using **Private IP for master Node**,

Set the following environment variables. Replace 10.0.0.10 with the IP of your master node.

IPADDR="10.0.0.10"

NODENAME=$(hostname -s)

POD\_CIDR="192.168.0.0/16"

If you want to use the**Public IP of the master**node,

Set the following environment variables. The **IPADDR** **variable** will be automatically set to the server’s public IP using ifconfig.me curl call. You can also replace it with a public IP address

IPADDR=$(curl ifconfig.me && echo "")

NODENAME=$(hostname -s)

POD\_CIDR="192.168.0.0/16"

Now, initialize the master node control plane configurations using the kubeadm command.

For a Private IP address-based setup use the following init command.

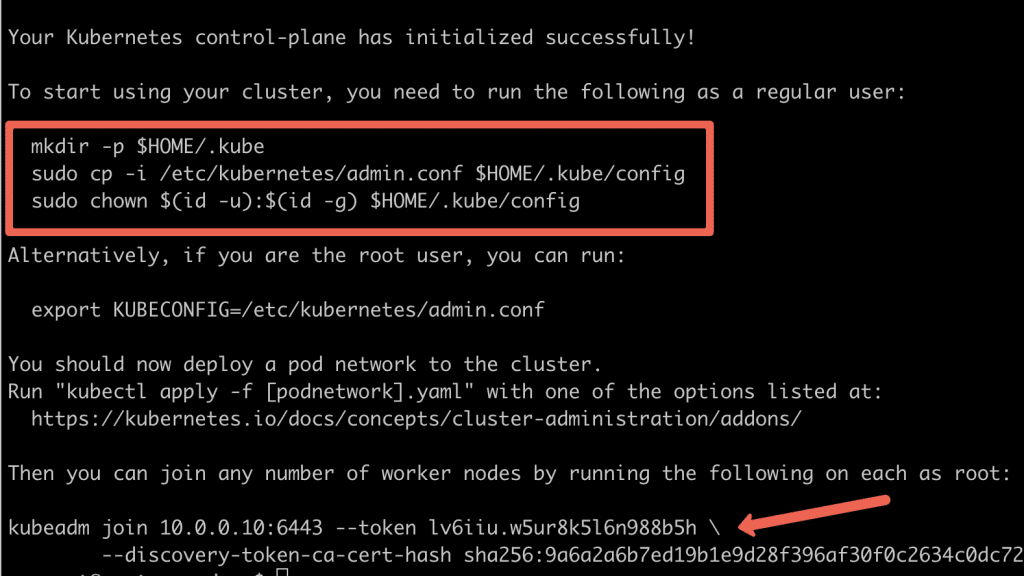
* sudo kubeadm init **--apiserver-advertise-address**=$IPADDR --apiserver-cert-extra-sans=$IPADDR --pod-network-cidr=$POD\_CIDR --node-name $NODENAME --ignore-preflight-errors Swap

--ignore-preflight-errors Swap is actually not required as we disabled the swap initially. For public IP address-based setup use the following init command. Here instead of --apiserver-advertise-address we use --control-plane-endpoint parameter for the API server endpoint.

* sudo kubeadm init **--control-plane-endpoint**=$IPADDR --apiserver-cert-extra-sans=$IPADDR --pod-network-cidr=$POD\_CIDR --node-name $NODENAME --ignore-preflight-errors Swap

All the other steps are the same as configuring the master node with private IP.

**Note:** You can also pass the kubeadm configs as a file when initializing the cluster. See [Kubeadm Init with config file](https://kubernetes.io/docs/reference/setup-tools/kubeadm/kubeadm-init/" \l "config-file" \t "_blank). On a successful kubeadm initialization, you should get an output with [kubeconfig file](https://devopscube.com/kubernetes-kubeconfig-file/) location and the **join command with the token** as shown below. Copy that and save it to the file. we will need it for **joining the worker node to the master**.

[](https://devopscube.com/wp-content/uploads/2021/05/kubeadm.png)

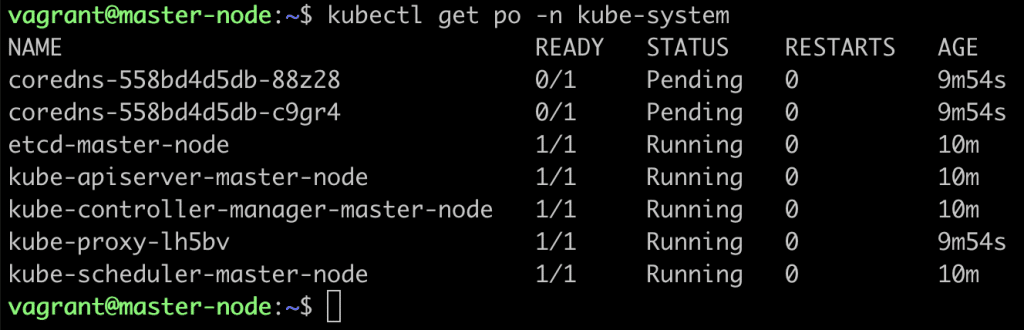
Use the following commands from the output to create the kubeconfig in master so that you can use kubectl to interact with cluster API.

* mkdir -p $HOME/.kube
* sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
* sudo chown $(id -u):$(id -g) $HOME/.kube/config

Now, verify the kubeconfig by executing the following kubectl command to list all the pods in the kube-system namespace.

* kubectl get po -n kube-system

You should see the following output. You will see the two Coredns pods in a pending state. It is the expected behavior. Once we install the network plugin, it will be in a running state

[](https://devopscube.com/wp-content/uploads/2021/05/pods.png)

**Note**: You can copy the admin.conf file from the master to your workstation in $HOME/.kube/config location if you want to execute kubectl commands from the workstation. You verify all the cluster component health statuses using the following command.

* kubectl get --raw='/readyz?verbose'

You can get the cluster info using the following command.

* kubectl cluster-info

By default, apps won’t get scheduled on the master node. If you want to use the master node for scheduling apps, taint the master node.

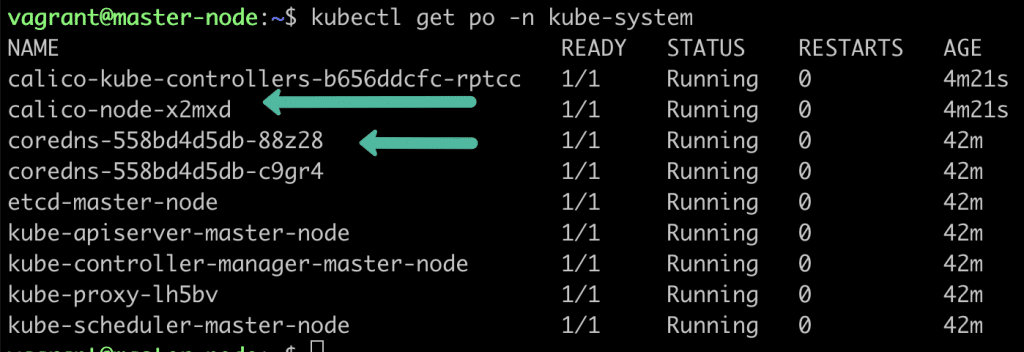
* kubectl taint nodes --all node-role.kubernetes.io/master-

**Install Calico Network Plugin for Pod Networking** Kubeadm does not configure any network plugin. You need to install a network plugin of your choice. I am using the Calico network plugin for this setup.

**Note**: Make sure you execute the kubectl command from where you have configured the kubeconfig file. Either from the master of your workstation with the connectivity to the kubernetes API. Execute the following command to install the calico network plugin on the cluster.

* kubectl apply -f https://raw.githubusercontent.com/projectcalico/calico/v3.25.0/manifests/calico.yaml

After a couple of minutes, if you check the pods in kube-system namespace, you will see calico pods and running CoreDNS pods.

[](https://devopscube.com/wp-content/uploads/2021/05/calico-pods.png)

**Join Worker Nodes To Kubernetes Master Node**

We have set up cri-o, kubelet, and kubeadm utilities on the worker nodes as well.

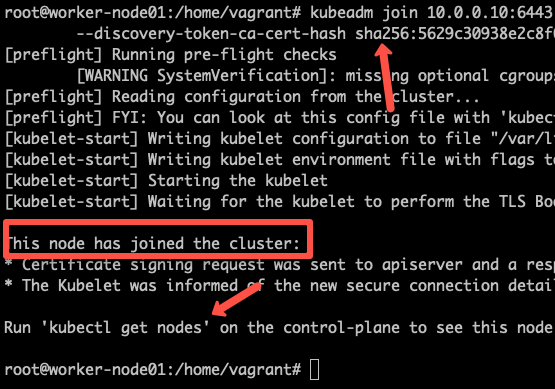
Now, let’s join the worker node to the master node using the Kubeadm join command you have got in the output while setting up the master node. If you missed copying the join command, execute the following command in the master node to recreate the token with the join command.

* kubeadm token create --print-join-command

Here is what the command looks like. Use sudo if you running as a normal user. This command performs the [TLS bootstrapping](https://kubernetes.io/docs/reference/access-authn-authz/kubelet-tls-bootstrapping/) for the nodes.

* sudo kubeadm join 10.128.0.37:6443 --token j4eice.33vgvgyf5cxw4u8i \ --discovery-token-ca-cert-hash sha256:37f94469b58bcc8f26a4aa44441fb17196a585b37288f85e22475b00c36f1c61

On successful execution, you will see the output saying, “This node has joined the cluster”.

[](https://devopscube.com/wp-content/uploads/2022/09/image-25.png)

Now execute the **kubectl command from the master node** to check if the node is added to the master.

* kubectl get nodes

Example output,

root@master-node:/home/vagrant# kubectl get nodes

NAME STATUS ROLES AGE VERSION

master-node Ready control-plane 14m v1.24.6

worker-node01 Ready <none> 2m13s v1.24.6

worker-node02 Ready <none> 2m5s v1.24.6

In the above command, the ROLE is <none> for the worker nodes. You can add a label to the worker node using the following command. Replace worker-node01 with the hostname of the worker node you want to label.

* kubectl label node worker-node01 node-role.kubernetes.io/worker=worker

You can further add more nodes with the same join command.

**Setup Kubernetes Metrics Server**

Kubeadm doesn’t install [metrics server](https://devopscube.com/setup-prometheus-monitoring-on-kubernetes/) component during its initialization. We have to install it separately.To verify this, if you run the top command, you will see the Metrics API not available error.

* root@master-node:~# kubectl top nodes

**error: Metrics API not available**

To install the metrics server, execute the following metric server manifest file. It deploys metrics server version v0.6.2

* kubectl apply -f https://raw.githubusercontent.com/techiescamp/kubeadm-scripts/main/manifests/metrics-server.yaml

This manifest is taken from the official [metrics serve](https://github.com/kubernetes-sigs/metrics-server)r repo. I have added the

* --kubelet-insecure-tls

flag to the container to make it work in the local setup and hosted it separately. Or else, you will get the following error.

because it doesn't contain any IP SANs" node=""

Once the metrics server objects are deployed, it takes a minute for you to see the node and pod metrics using the top command.

kubectl top nodes

You should be able to view the node metrics as shown below.

root@master-node:/home/vagrant# kubectl top nodes

NAME CPU(cores) CPU% MEMORY(bytes) MEMORY%

master-node 111m 5% 1695Mi 44%

worker-node01 28m 2% 1078Mi 57%

worker-node02 219m 21% 980Mi 52%

You can also view the pod CPU and memory metrics using the following command.

* kubectl top pod -n kube-system

**Deploy A Sample Nginx Application**

Now that we have all the components to make the cluster and applications work, let’s deploy a sample Nginx application and see if we can access it over a NodePort

Create an Nginx [deployment](https://devopscube.com/kubernetes-deployment-tutorial/). Execute the following directly on the command line. It deploys the pod in the default namespace.

cat <<EOF | kubectl apply -f -

apiVersion: apps/v1

kind: Deployment

metadata:

name: nginx-deployment

spec:

selector:

matchLabels:

app: nginx

replicas: 2

template:

metadata:

labels:

app: nginx

spec:

containers:

- name: nginx

image: nginx:latest

ports:

- containerPort: 80 EOF

Expose the Nginx deployment on a **NodePort 32000**

cat <<EOF | kubectl apply -f -

apiVersion: v1

kind: Service

metadata:

name: nginx-service

spec:

selector:

app: nginx

type: NodePort

ports:

- port: 80

targetPort: 80

nodePort: 32000

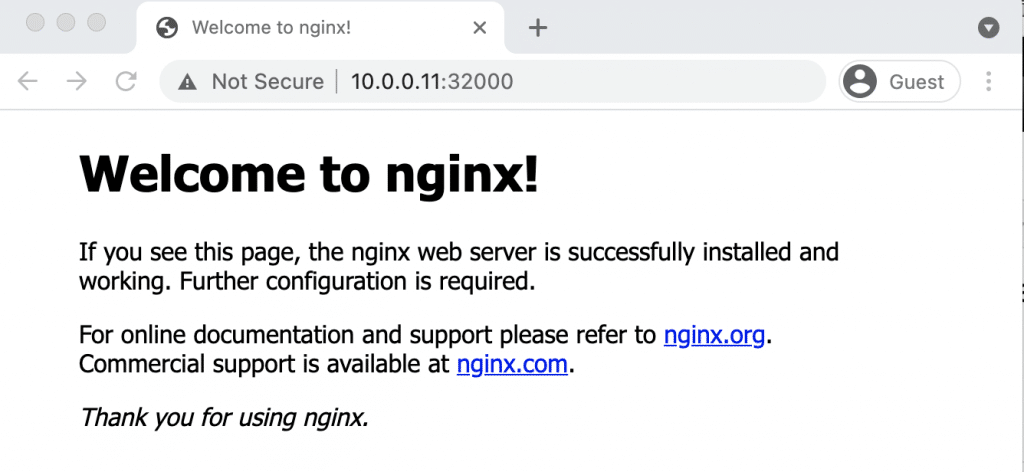
EOF

Check the pod status using the following command.

* kubectl get pods

Once the deployment is up, you should be able to access the Nginx home page on the allocated NodePort.

For example,

[](https://devopscube.com/wp-content/uploads/2021/05/nginx.png)

**Possible Kubeadm Issues**

Following are the possible issues you might encounter in the kubeadm setup.

1. **Pod Out of memory and CPU:** The master node should have a minimum of 2vCPU and 2 GB memory.
2. **Nodes cannot connect to Master:** Check the firewall between nodes and make sure all the nodes can talk to each other on the required kubernetes ports.
3. **Calico Pod Restarts:** Sometimes, if you use the same IP range for the node and pod network, Calico pods may not work as expected. So make sure the node and pod IP ranges don’t overlap. Overlapping [IP addresses](https://devopscube.com/ip-address-tutorial/) could result in issues for other applications running on the cluster as well.

For other pod errors, check out the [kubernetes pod troubleshooting](https://devopscube.com/troubleshoot-kubernetes-pods/) guide.

If your server doesn’t have a minimum of 2 vCPU, you will get the following error.

[ERROR NumCPU]: the number of available CPUs 1 is less than the required 2

If you use public IP with --apiserver-advertise-address parameter, you will have failed master node components with the following error. To rectify this error, use --control-plane-endpoint parameter with the public IP address.

kubelet-check] Initial timeout of 40s passed.

Unfortunately, an error has occurred:

timed out waiting for the condition

This error is likely caused by:

- The kubelet is not running

- The kubelet is unhealthy due to a misconfiguration of the node in some way (required cgroups disabled)

If you are on a systemd-powered system, you can try to troubleshoot the error with the following commands:

* systemctl status kubelet
* journalctl -xeu kubelet

**Kubernetes Cluster Important Configurations**

Following are the important cluster configurations you should know.

| **Configuration** | **Location** |
| --- | --- |
| Static Pods Location (etcd, api-server, controller manager and scheduler) | /etc/kubernetes/manifests |
| TLS Certificates location (kubernetes-ca, etcd-ca and kubernetes-front-proxy-ca) | /etc/kubernetes/pki |
| Admin Kubeconfig File | /etc/kubernetes/admin.conf |
| Kubelet configuration | /var/lib/kubelet/config.yaml |

**Upgrading Kubeadm Cluster**

Using kubeadm you can upgrade the kubernetes cluster for the same version patch or a new version.

Kubeadm upgrade doesn’t introduce any downtime if you upgrade one node at a time.

To do hands-on, please refer to my step-by-step guide on [Kubeadm cluster upgrade](https://devopscube.com/upgrade-kubernetes-cluster-kubeadm/)