A Kubernetes cluster consists of master node (control plane) and worker nodes. We generally run application workload on worker nodes and master nodes are used as control plane, as it manages the worker nodes and pods in the cluster.

Kubernetes is a popular container orchestration tool.

**Features:**

* Load balancing
* Self-healing
* High availability/ensure no downtime/maintain fault tolerance
* Performance enhancement
* Auto-scaling.

**Kubernetes Terminologies**

**Pod**: It represents one or more containers running in a cluster.

**Service**: An abstract way to access pod/application.

**Namespace**: It is used to remove name collision within a cluster. It supports multiple virtual clusters on the same physical cluster.

**Node**: Kubernetes worker machine.

**Cluster**: Consisting of a group of nodes running containerized applications on Kubernetes.

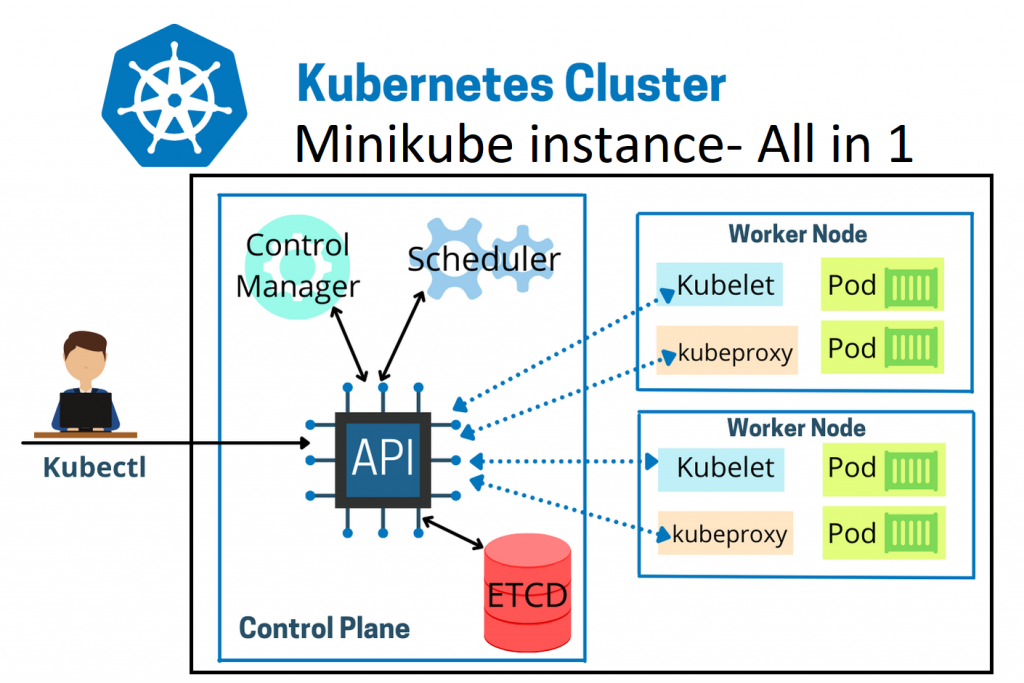
**ReplicaSets**: Several replicas of running pods. It helps in achieving high availability and scalability.

**Label**: Giving a name to Kubernetes objects so that it can be identified across the system.

**Kubelet**: It is an agent that runs on each node and checks if the containers are running in the pods.

**Kubectl**: Command-line utility to interact with the Kubernetes API server.

**Kube-proxy**: Network proxy which contains all the network rules on each node in the cluster.



Kubernetes follows master-slave architecture. Kubernetes architecture has a master node and worker nodes.

There are four components of a **master node components**:

* Kube API server
* Controller
* Scheduler
* etcd

And the **worker node has three components:**

* Kubelet
* kube-proxy
* container runtime

**There are several ways to install Kubernetes, including:**

* **Single-node installation:** Good for development, testing, and practices, or for previewing Kubernetes.
* **Unmanaged Kubernetes installation:** You manage everything yourself, as it's not managed by a cloud vendor.
* **Kubespray:** A combination of Ansible and Kubernetes that provides deployment flexibility.
* **Kubeadm:** A popular method that's good for multi-node clusters and real-time setups.
* **Kops:** An open-source tool that's good for managing and creating small-scale, production-grade clusters.
* **Minikube:** A free, open-source tool that runs a single-node cluster in a virtual machine on your laptop.

## Prerequisites (recommended but resource specification may change depends on your workloads)

* Minimal install Ubuntu 22.04

**Master:**

* RAM: 2Gi or more
* Cores/Vcpu: 2 or more
* Storage: 20 Gig

**Worker:**

* RAM: 1Gi or more
* Cores/Vcpu: 2 or more
* Storage: 20 Gig
* Sudo user with admin rights
* Internet connectivity on each node

Note: provision EC2 nodes and let’s proceed further for installation.

**Overall, installing Kubernetes on Ubuntu involves steps such as:**

1. Disabling swap;
2. Setting up hostnames;
3. Setting up the IPV4 bridge on all nodes;
4. Installing Kubernetes components on all nodes;
5. Installing Docker or a suitable containerization tool;
6. Initializing the Kubernetes cluster using kubeadm;
7. Install the [Calico network plugin](https://docs.tigera.io/calico/latest/about/) (operator).
8. Join the worker node to the master node (control plane) using the join command.
9. Test your kubernetes cluster installation by deploying nginx application.

# Kubeadm Setup Prerequisites

Following are the prerequisites for **Kubeadm Kubernetes cluster setup**.

1. Minimum two **Ubuntu nodes** [One master and one worker node]. You can have more worker nodes as per your requirement.
2. The master node should have a minimum of **2 vCPU and 2GB RAM**.
3. For the worker nodes, a minimum of 1vCPU and 2 GB RAM is recommended.
4. **10.X.X.X/X**network range with static IPs for master and worker nodes. We will be using the **192.x.x.x** series as the pod network range that will be used by the Calico network plugin. Make sure the Node IP range and pod IP range don’t overlap.

***Note****: If you are setting up the cluster in the corporate network behind a proxy, ensure set the proxy variables and have access to the container registry and docker hub. Or talk to your network administrator to whitelist****registry.k8s.io****to pull the required images.*

# Kubeadm Port Requirements

# Please refer to the following image and make sure all the ports are allowed for the control plane (master) and the worker nodes. If you are setting up the kubeadm cluster cloud servers, ensure you allow the ports in the firewall configuration.

## **Control plane**

| **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 |

# Although etcd ports are included in control plane section, you can also host your own etcd cluster externally or on custom ports.

## **Worker node(s)**

| **Protocol** | **Direction** | **Port Range** | **Purpose** | **Used By** |
| --- | --- | --- | --- | --- |
| TCP | Inbound | 10250 | Kubelet API | Self, Control plane |
| TCP | Inbound | 10256 | kube-proxy | Self, Load balancers |
| TCP | Inbound | 30000-32767 | NodePort Services† | All |

Default port range for [NodePort Services](https://kubernetes.io/docs/concepts/services-networking/service/) from 30000 to 32767

All default port numbers can be overridden. When custom ports are used those ports need to be open instead of defaults mentioned here.

One common example is API server port that is sometimes switched to 443. Alternatively, the default port is kept as is and API server is put behind a load balancer that listens on 443 and routes the requests to API server on the default port.

## 1) Set hostname for all the nodes

Login to master node and set hostname via hostnamectl command,

$ sudo hostnamectl set-hostname "k8master.example.com"

// on master node

$ exec bash

On the worker nodes, run

$ sudo hostnamectl set-hostname "k8worker1.example.com"

// 1st worker node

$ sudo hostnamectl set-hostname "k8worker2.example.com"

// 2nd worker node

$ exec bash

Add the IP and hostname lines in /etc/hosts file on each node

Note: below entries are as per my servers ip and domain addresses.

192.168.1.173   k8master.example.com k8master

192.168.1.174   k8worker1.example.com k8worker1

192.168.1.175   k8worker2.example.com k8worker2

## 2) Disable Swap space on all the nodes

You might know about swap space on hard drives, which OS systems try to use as if it were RAM. Operating systems try to move less frequently accessed data to the swap space to free up RAM for more immediate tasks.

However, accessing data in swap is much slower than accessing data in RAM because hard drives are slower than RAM.

Kubernetes schedules work based on the understanding of available resources.

If workloads start using swap, it can become difficult for Kubernetes to make accurate scheduling decisions. Therefore, it’s recommended to disable swap before installing Kubernetes.

You can do it with the following command. The sudo swapoff - command temporarily disables swap on your system.

Then, the sudo sed -i '/ swap / s/^/#/' /etc/fstab command modifies a configuration file to keep the swap remains off even after a system reboot.

Make sure to run the following commands on all the nodes.

$ sudo swapoff -a

$ sudo sed -i '/ swap / s/^/#/' /etc/fstab

Follow my below YouTube link *“How to Create a Linux SWAP FILE”.*

<https://youtu.be/FhZFcVJMfh4?si=wuh3djjU__asbym1>

### 3) Set up the IPV4 bridge traffic on all the nodes

To configure the IPV4 bridge on all nodes, execute the following commands on each node.

Here we are tweaking some kernel parameters and setting them using **sysctl**.

$ sudo tee /etc/modules-load.d/containerd.conf <<EOF

overlay

br\_netfilter

EOF

$ sudo modprobe overlay

$ sudo modprobe br\_netfilter

$ sudo tee /etc/sysctl.d/kubernetes.conf <<EOT

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

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

net.ipv4.ip\_forward = 1

EOT

Reload the above changes, by running below command on all the nodes

$ sudo sysctl --system

## 4) Install kubelet, kubeadm, and kubectl on all the nodes

Let’s install kubelet, kubeadm, and kubectl on each node to create a Kubernetes cluster. They play an important role in managing a Kubernetes cluster.

Kubelet is the node agent that runs on every node and is responsible for ensuring containers are running in a Pod as specified by the Pod's specifications. (Pods are the smallest deployable units in a Kubernetes cluster).

kubeadm, which is used to bootstrap a Kubernetes cluster, including setting up the master node and helping worker nodes join the cluster.

Kubectl is a CLI tool for Kubernetes to run commands to perform various actions such as deploying applications, inspecting resources, and managing cluster operations directly from the terminal.

Before installing them, you must update the package index with the [sudo apt-get update](https://www.cherryservers.com/blog/sudo-apt-update) command.

$ sudo apt-get update

Next, we have to ensure that we can download and install packages from the internet securely.

sudo apt-get install -y apt-transport-https ca-certificates curl

Next, we have to create a directory where we'll store a special key that verifies the authenticity of Kubernetes packages. It's like checking an ID card before allowing someone into a building.

sudo mkdir /etc/apt/keyrings // sometimes this file may exist

Let’s fetch the public key from Google and store it in the folder we created in the previous step. This key is important to verify that the Kubernetes packages we download are genuine and haven't been tampered with.

Download the GPG key for the Kubernetes APT repository **on all the nodes.**

At the time of this document, I’m using the kubernetes version v1.29. if you can update KUBERNETES\_VERSION with latest LTS version and run the below commands so that you will get latest packages installed.

KUBERNETES\_VERSION=1.29

curl -fsSL https://pkgs.k8s.io/core:/stable:/v$KUBERNETES\_VERSION/deb/Release.key | sudo gpg --dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg echo "deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] https://pkgs.k8s.io/core:/stable:/v$KUBERNETES\_VERSION/deb/ /" | sudo tee /etc/apt/sources.list.d/kubernetes.list

Let’s refresh the apt package index to see new items by running the sudo apt-get update command again.

sudo apt-get update -y

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

apt-cache madison kubeadm | tac

To**install the latest version** from the repo, use the following command without specifying any version.

sudo apt install -y kubelet kubeadm kubectl

## or to Specify the version as shown below. Here I am using **1.29.0-1.1**

sudo apt-get install -y kubelet=1.29.0-1.1 kubectl=1.29.0-1.1 kubeadm=1.29.0-1.1

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.

## 5) Install Container 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. containerd
3. Docker Engine (using cri-dockerd)

This document explained with two types of CRI options, such as containerd and cri-o. Any one of the container runtime you can use as container runtime interface.

## Docker

The Docker platform allows you to create, distribute, and run applications within containers. These containers offer a lightweight and portable environment, ensuring consistent performance across various setups. Docker serves as the container runtime, playing a vital role in Kubernetes by facilitating the efficient management and deployment of containerized applications.

Install Docker with the command

sudo apt install docker.io

Next, configure containerd on all nodes to ensure its compatibility with Kubernetes. First, create a folder for the configuration file with the command.

sudo mkdir /etc/containerd

Then, create a default configuration file for containerd and save it as config.toml using the command.

sudo sh -c "containerd config default > /etc/containerd/config.toml"

After running these commands, you need to modify the config.toml file to locate the entry that sets "SystemdCgroup" to false and changes its value to true. This is important because Kubernetes requires all its components, and the container runtime uses systemd for cgroups.

sudo sed -i 's/ SystemdCgroup = false/ SystemdCgroup = true/' /etc/containerd/config.toml

Next, restart containerd and kubelet services to apply the changes you made on all nodes.

sudo systemctl restart containerd.service

sudo systemctl restart kubelet.service

You will want to start kubelet service whenever the machine boots up, which you can do by running the command.

sudo systemctl enable kubelet.service

Add your local user to docker group so that your local user run docker commands without sudo.

Add your user to the 'docker' group:

$ sudo usermod -aG docker $USER && newgrp docker

Note: To make above changes into the affect logout and login.

## CRI-O

**Note:**We are using cri-o instead if [containerd](https://containerd.io/) because, in [Kubernetes certification](https://devopscube.com/best-kubernetes-certifications/) exams, cri-o is used as the container runtime in the exam clusters.

Execute the following commands **on all the nodes** to install required dependencies and the latest version of CRIO.

sudo apt-get update -y

sudo apt-get install -y software-properties-common curl apt-transport-https ca-certificates

curl -fsSL https://pkgs.k8s.io/addons:/cri-o:/prerelease:/main/deb/Release.key |

gpg --dearmor -o /etc/apt/keyrings/cri-o-apt-keyring.gpg

echo "deb [signed-by=/etc/apt/keyrings/cri-o-apt-keyring.gpg] https://pkgs.k8s.io/addons:/cri-o:/prerelease:/main/deb/ /" |

tee /etc/apt/sources.list.d/cri-o.list

sudo apt-get update -y

sudo apt-get install -y cri-o

sudo systemctl daemon-reload

sudo systemctl enable crio --now

sudo systemctl start crio.service

Install crictl.

VERSION="v1.28.0" wget https://github.com/kubernetes-sigs/cri-tools/releases/download/$VERSION/crictl-$VERSION-linux-amd64.tar.gz sudo tar zxvf crictl-$VERSION-linux-amd64.tar.gz -C /usr/local/bin rm -f crictl-$VERSION-linux-amd64.tar.gz

**crictl**, a CLI utility to interact with the containers created by the container runtime.

When you use container runtimes other than Docker, you can use the **crictl utility** to debug containers on the nodes.

## Initialize the Kubernetes cluster on the master node

Here you need to consider two options.

1. **Master Node with Private IP:**

If you have nodes with only private IP addresses the API server would be accessed over the private IP of the 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"

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.

1. **Master Node with Public IP:**

If you are setting up a Kubeadm cluster on Cloud platforms and you need master Api server access over the Public IP of the master node server.

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"

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

after initialize completed kubeadm is deployed several components to manage and orchestrate the cluster.

Some examples of these components are kube-apiserver, kube-controller-manager, kube-scheduler, etcd, kube-proxy.

To manage the cluster, you should configure kubectl on the master node. Create the .kube directory in your home folder and copy the cluster's admin configuration to your personal .kube directory.

Next, change the ownership of the copied configuration file to give the user the permission to use the configuration file to interact with the cluster.

Here are the commands you need to do this.

mkdir -p $HOME/.kube

sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

sudo chown $(id -u):$(id -g) $HOME/.kube/config

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/control-plane-

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 behaviour. Once we **install the network plugin**, it will be in a running state.

## 7) Add worker nodes to the cluster

Once you have configured the master node, you can add worker nodes to the cluster. When initializing Kubeadm on the master node, you will receive a token that you can use to add worker nodes.

To add the worker nodes to the Kubernetes cluster, use the kubeadm join command. (It looks like this below command. You can get the variable values from the above steps)

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 &lt;MASTER\_NODE\_IP>:&lt;API\_SERVER\_PORT> --token &lt;TOKEN> --discovery-token-ca-cert-hash &lt;CERTIFICATE\_HASH>

before running the kubeadm join command on worker node, take a image backup and create as many worker nodes you wish and run the kubeadm join on each node to register with master node.

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

## 8) Install Calico Network Plugin for Pod Networking

Run the following commands on the master node to deploy the Calico operator.

A network plugin is required to enable communication between pods in the cluster.

Run following kubectl command to install Calico network plugin from the master node

kubectl apply -f <https://raw.githubusercontent.com/projectcalico/calico/v3.26.0/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.

kubectl get po -n kube-system

## 9) Test Your Kubernetes Cluster Installation

To test Kubernetes installation, let’s try to deploy nginx based application and try to access it.

$ kubectl create deployment nginx-app --image=nginx --replicas=2

Check the status of nginx-app deployment

$ kubectl get deployment nginx-app

Expose the deployment as NodePort,

$ kubectl expose deployment nginx-app --type=NodePort --port=80

service/nginx-app exposed

Run following commands to view service status

$ kubectl get svc nginx-app

$ kubectl describe svc nginx-app

Use following curl command to access nginx based application,

$ curl http://<woker-node-ip-addres>:31246

$ curl http://192.168.1.174:31246

Publicly access, use http://<worker-node-publicIP>:31246

Great, above output confirms that nginx based application is accessible.