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.

A diagram of a computer system

Description automatically generated

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

## Prerequisites

* Minimal install Ubuntu 22.04
* Minimum 2GB RAM or more
* Minimum 2 CPU cores / or 2 vCPU
* 20 GB free disk space on /var or more
* 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;
7. Configuring kubectl and Calico;
8. Adding worker nodes;

## 1) Set hostname on Each Node

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 following 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 & Add kernel Parameters

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

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

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

$ 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, run

$ sudo sysctl --system

## 4) Install kubelet, kubeadm, and kubectl on each node

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 file 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.

At the time of this document, I’m using the kubernetes version v1.28. if you can update it with latest LTS version and run the commands so that you will get latest packages installed.

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

Next, we need to tell the apt package manager where to find Kubernetes packages for downloading.

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

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

sudo apt-get update

Now we are ready to install kubelet, kubeadm, and kubectl by running the sudo apt install -y kubelet kubeadm kubectl command.

sudo apt install -y kubelet kubeadm kubectl

## 5) Install Containerd Runtime – 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

## 6) Initialize the Kubernetes cluster on the master node

When you initialize a Kubernetes control plane using kubeadm, several components are deployed to manage and orchestrate the cluster.

Some examples of these components are kube-apiserver, kube-controller-manager, kube-scheduler, etcd, kube-proxy. ‘We need to download the images of these components by running the following command.

sudo kubeadm config images pull

Next, initialize your master node. The --pod-network-cidr flag is setting the IP address range for the pod network.

sudo kubeadm init --control-plane-endpoint=master.example.com

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

## 7) Configure kubectl and Calico

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 create -f~~ [~~https://raw.githubusercontent.com/projectcalico/calico/v3.26.1/manifests/tigera-operator.yaml~~](https://raw.githubusercontent.com/projectcalico/calico/v3.26.1/manifests/tigera-operator.yaml)

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

~~Next, download the custom resources file for Calico, which contains definitions of the various resources that Calico will use.~~

~~curl https://raw.githubusercontent.com/projectcalico/calico/v3.26.1/manifests/custom-resources.yaml -O~~

~~Using the following command, modify the CIDR in the downloaded custom resources to match your pod network. Here, you're using the sed command to change the default CIDR value in the Calico custom resources to match the CIDR you used in the kubeadm init command.~~

~~sed -i 's/cidr: 192\.168\.0\.0\/16/cidr: 10.10.0.0\/16/g' custom-resources.yaml~~

~~Finally, tell kubectl to create the resources defined in the custom-resources.yaml file.~~

~~kubectl create -f custom-resources.yaml~~

## 8) 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)

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.

## 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

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