How to setup a Kubernetes Cluster:

#K8s setup:

Installing Kubernetes with deployment tools

There are many methods and tools for setting up your own production Kubernetes cluster. For example:

- kubeadm
- Cluster API: A Kubernetes sub-project focused on providing declarative APIs and tooling to simplify provisioning, upgrading, and operating multiple Kubernetes clusters.
- kops: An automated cluster provisioning tool. For tutorials, best practices, configuration options and information on reaching out to the community, please check the k0ps website for details.
- kubespray: A composition of Ansible playbooks, inventory, provisioning tools, and domain knowledge for generic OS/Kubernetes clusters configuration management tasks. You can reach out to the community on Slack channel #kubespray.

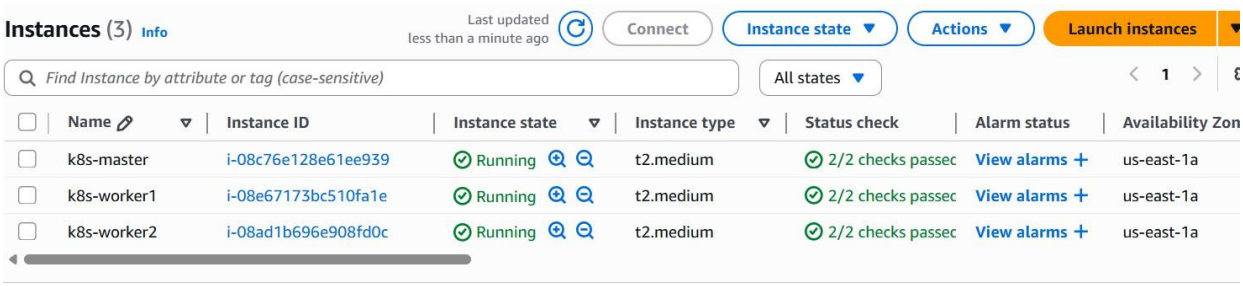
Lets go with : Kubeadm

Before you begin

- A compatible Linux host. The Kubernetes project provides generic instructions for Linux distributions based on Debian and Red Hat, and those distributions without a package manager.
- 2 GB or more of RAM per machine (any less will leave little room for your apps).
- 2 CPUs or more for control plane machines.
- Full network connectivity between all machines in the cluster (public or private network is fine).
- Unique hostname, MAC address, and product_uuid for every node..

Lets create 3 Ec2 instances on AWS, in which 1 is for Master-node , 2 are for worker-node : Create with these names for better clarity :

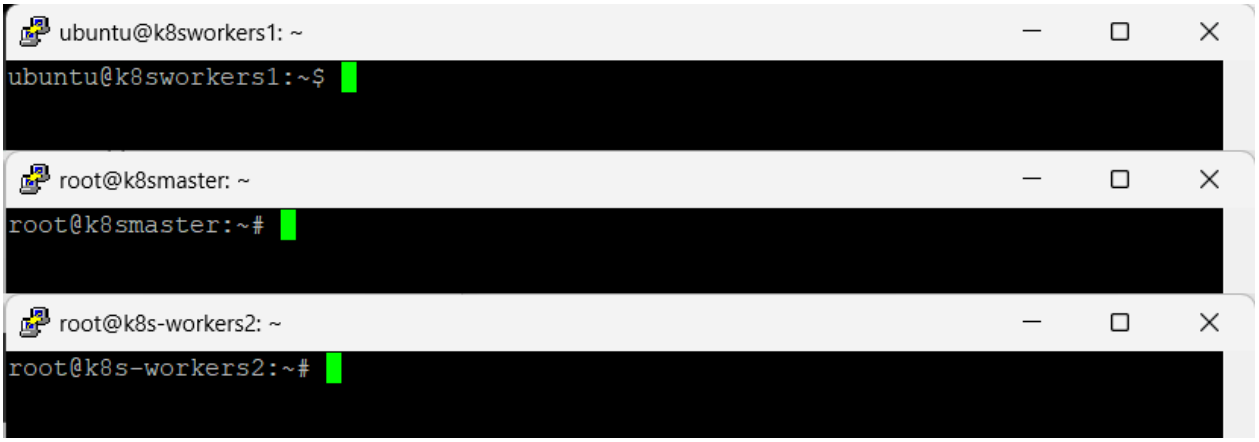
- Kube-MasterNode
- Kube-WorkerNode-1
- Kube-WorkerNode-2



Also create a Elastic IP for the MasterNode.

Access all the 3 Nodes through ssh and set them ready for cluster package installations.

Set the hostname like below to avoid confusion , use : " hostnamectl set-hostname k8master"



#From now on we have to do the same implementation in all the Nodes to some extinct.

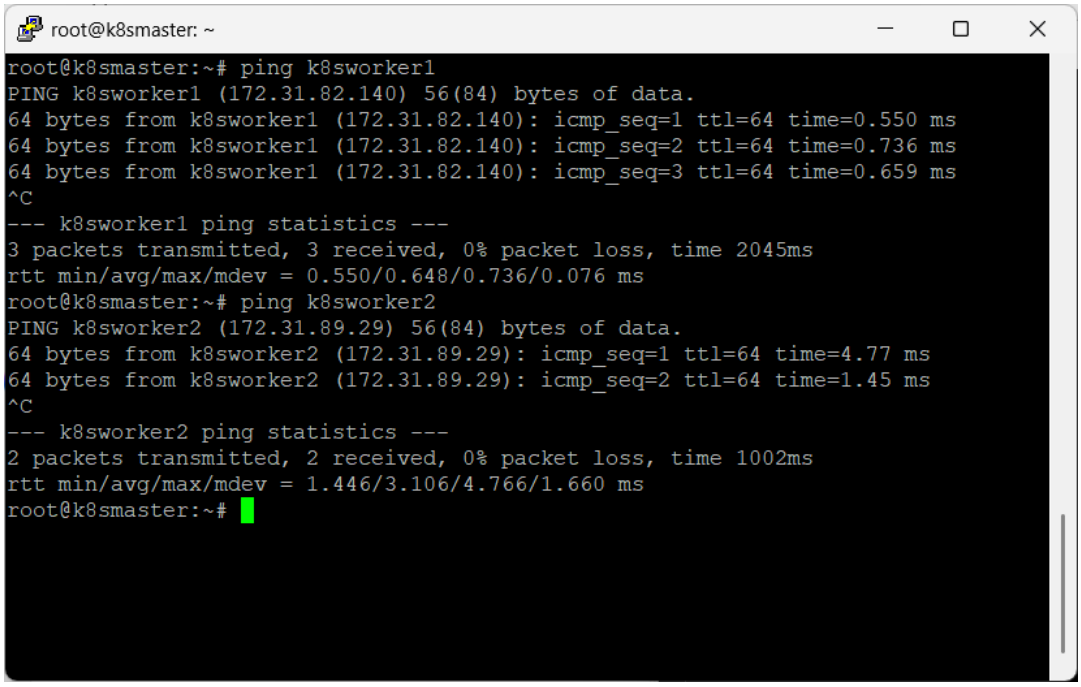
- Edit the host file : vim /etc/hosts

172.31.93.25 k8smaster

172.31.88.44 k8sworker1

172.31.83.157 k8sworker2

- After editing the files , try ping test : like " ping k8worker-1 " from master node – it should work and show the result like below . It should be same with worker nodes also when tried ping with any other node in cluster.



```
root@k8sworkers1: ~
root@k8sworkers1:~# ping k8sworkers2
ping: k8sworkers2: Temporary failure in name resolution
root@k8sworkers1:~# ping k8sworker2
PING k8sworker2 (172.31.89.29) 56(84) bytes of data.
64 bytes from k8sworker2 (172.31.89.29): icmp_seq=1 ttl=64 time=1.95 ms
64 bytes from k8sworker2 (172.31.89.29): icmp_seq=2 ttl=64 time=0.298 ms
64 bytes from k8sworker2 (172.31.89.29): icmp_seq=3 ttl=64 time=0.535 ms
^C
--- k8sworker2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2036ms
rtt min/avg/max/mdev = 0.298/0.927/1.950/0.729 ms
root@k8sworkers1:~# ping k8smaster
PING k8smaster (172.31.95.168) 56(84) bytes of data.
64 bytes from k8smaster (172.31.95.168): icmp_seq=1 ttl=64 time=0.593 ms
64 bytes from k8smaster (172.31.95.168): icmp_seq=2 ttl=64 time=0.480 ms
^C
--- k8smaster ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1052ms
rtt min/avg/max/mdev = 0.480/0.536/0.593/0.056 ms
root@k8sworkers1:~#
```

```
root@k8s-workers2: ~
root@k8s-workers2:~# ping k8smaster
PING k8smaster (172.31.95.168) 56(84) bytes of data.
64 bytes from k8smaster (172.31.95.168): icmp_seq=1 ttl=64 time=0.330 ms
64 bytes from k8smaster (172.31.95.168): icmp_seq=2 ttl=64 time=0.968 ms
64 bytes from k8smaster (172.31.95.168): icmp_seq=3 ttl=64 time=0.344 ms
^C
--- k8smaster ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2050ms
rtt min/avg/max/mdev = 0.330/0.547/0.968/0.297 ms
root@k8s-workers2:~# ping k8sworker2
PING k8sworker2 (172.31.89.29) 56(84) bytes of data.
64 bytes from k8sworker2 (172.31.89.29): icmp_seq=1 ttl=64 time=0.012 ms
64 bytes from k8sworker2 (172.31.89.29): icmp_seq=2 ttl=64 time=0.026 ms
64 bytes from k8sworker2 (172.31.89.29): icmp_seq=3 ttl=64 time=0.035 ms
^C
--- k8sworker2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2041ms
rtt min/avg/max/mdev = 0.012/0.024/0.035/0.009 ms
root@k8s-workers2:~#
```

➔ Notes : I found 100% packet loss while executing this !

"Ping is working with the nodes, so why is ICMP important, and what is it?"

When you run: ping k8worker-1

This happens:

1. Your system sends **ICMP Echo Request** packets to the IP.

2. The remote system (if online and allowed) sends back **ICMP Echo Reply** packets.
3. If replies are received, ping works. If not, you get 100% packet loss.

This is what happened with me , so let's troubleshoot this :

What is ICMP?

ICMP stands for **Internet Control Message Protocol**. It's a network protocol used for **sending error messages and operational information**, such as:

- Whether a host is reachable
- Whether a network path exists

The ping command uses **ICMP Echo Request** and waits for an **ICMP Echo Reply** from the target.

That means:

- Your **worker nodes** allow ICMP traffic.
- But the machine `k8worker-1` (possibly an EC2 instance) **does not** allow ICMP traffic.

This is usually due to:

- **AWS Security Groups** not allowing ICMP
- **Local firewall** (e.g., `iptables`, `firewalld`) on the target system blocking ICMP

Conclusion

- **Ping needs ICMP** to work.
- If ICMP is blocked, `ping` will fail — even if the server is running and reachable by other protocols like SSH.
- You should **allow ICMP in Security Group** and/or **local firewall** if you want `ping` to work.

➔ Verify the MAC address and product_uuid are unique for every node

- You can get the MAC address of the network interfaces using the command ip link or ifconfig -a
- The product_uuid can be checked by using the command : sudo cat /sys/class/dmi/id/product_uuid

➔ **Checked for MAC address of all the nodes found Unique : used " ip a "**

root@k8smaster: ~

```
root@k8smaster:~# sudo cat /sys/class/dmi/id/product_uuid
ec21b177-9737-80f1-7d9a-c7dcf592bc5f
root@k8smaster:~# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group defaul
t qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
       valid_lft forever preferred_lft forever
2: enX0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9001 qdisc mq state UP group defa
ult qlen 1000
    link/ether 12:ce:32:d4:d9:db brd ff:ff:ff:ff:ff:ff
    inet 172.31.95.168/20 metric 100 brd 172.31.95.255 scope global dynamic enX0
       valid_lft 2244sec preferred_lft 2244sec
    inet6 fe80::10ce:32ff:fed4:d9db/64 scope link
       valid_lft forever preferred_lft forever
root@k8smaster:~#
```

root@k8sworkers1: ~

```
root@k8sworkers1:~# sudo cat /sys/class/dmi/id/product_uuid
ec21f74b-5b8b-d077-a297-e1906e9c2e21
root@k8sworkers1:~# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group defaul
t qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
       valid_lft forever preferred_lft forever
2: enX0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9001 qdisc mq state UP group defa
ult qlen 1000
    link/ether 12:ee:8e:f8:c3:65 brd ff:ff:ff:ff:ff:ff
    inet 172.31.82.140/20 metric 100 brd 172.31.95.255 scope global dynamic enX0
       valid_lft 2325sec preferred_lft 2325sec
    inet6 fe80::10ee:8eff:fef8:c365/64 scope link
       valid_lft forever preferred_lft forever
root@k8sworkers1:~#
```

```
root@k8s-workers2: ~
root@k8s-workers2:~# sudo cat /sys/class/dmi/id/product_uuid
ec2a0dd0-1db9-a056-9036-8cc9a351ec6c
root@k8s-workers2:~# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group defaul
t qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
       valid_lft forever preferred_lft forever
2: enX0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9001 qdisc mq state UP group defa
ult qlen 1000
    link/ether 12:a0:57:82:f4:03 brd ff:ff:ff:ff:ff:ff
    inet 172.31.89.29/20 metric 100 brd 172.31.95.255 scope global dynamic enX0
       valid_lft 2409sec preferred_lft 2409sec
    inet6 fe80::10a0:57ff:fe82:f403/64 scope link
       valid_lft forever preferred_lft forever
root@k8s-workers2:~#
```

➔ **Check for required Ports :**

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
UDP	Inbound	30000-32767	NodePort Services†	All

See the documentation and set the ports accordingly :

➔ Swap configuration:

The default behavior of a kubelet is to fail to start if swap memory is detected on a node. This means that swap should either be disabled or tolerated by kubelet.

Installing a container runtime

To run containers in Pods, Kubernetes uses a container runtime.

This page provides an outline of how to use several common container runtimes with Kubernetes.

- containerd
- CRI-O
- Docker Engine
- Mirantis Container Runtime

➔ Install and configure prerequisites

Network configuration

By default, the Linux kernel does not allow IPv4 packets to be routed between interfaces. Most Kubernetes cluster networking implementations will change this setting (if needed), but some might expect the administrator to do it for them. (Some might also expect other sysctl parameters to be set, kernel modules to be loaded, etc; consult the documentation for your specific network implementation.)

Enable IPv4 packet forwarding

To manually enable IPv4 packet forwarding:

```
# sysctl params required by setup, params persist across reboots
cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf
net.ipv4.ip_forward = 1
EOF
```

```
# Apply sysctl params without reboot
sudo sysctl --system
```

Getting started with containerd

Installing containerd

Option 1: From the official binaries

The official binary releases of containerd are available for the `amd64` (also known as `x86_64`) and `arm64` (also known as `aarch64`) architectures.
Typically, you will have to install runc and CNI plugins from their official sites too.

Step 1: Installing containerd

Download the `containerd-<VERSION>-<OS>-<ARCH>.tar.gz` archive from https://github.com/containerd/containerd/releases , verify its sha256sum, and extract it under `/usr/local`:

Click on that link it will navigate to :

▾ Assets 25

containerd-2.1.1-attestation.intoto.jsonl	12 KB	last week
containerd-2.1.1-linux-amd64.tar.gz	31.7 MB	last week
containerd-2.1.1-linux-amd64.tar.gz.sha256sum	102 Bytes	last week
containerd-2.1.1-linux-arm64.tar.gz	29.1 MB	last week
containerd-2.1.1-linux-arm64.tar.gz.sha256sum	102 Bytes	last week
containerd-2.1.1-linux-ppc64le.tar.gz	29.2 MB	last week
containerd-2.1.1-linux-ppc64le.tar.gz.sha256sum	104 Bytes	last week
containerd-2.1.1-linux-riscv64.tar.gz	29.3 MB	last week
containerd-2.1.1-linux-riscv64.tar.gz.sha256sum	104 Bytes	last week
containerd-2.1.1-linux-s390x.tar.gz	31.4 MB	last week
Source code (zip)		last week
Source code (tar.gz)		last week

Show all 25 assets

Copy that link : and download it to all the nodes in cluster :

wget https://github.com/containerd/containerd/releases/download/v2.1.1/containerd-2.1.1-linux-amd64.tar.gz

```
root@k8master:~# ls
containerd-2.1.1-linux-amd64.tar.gz  snap
```

After downloading that file : do replace the tar file in place of document syntax :

Extract the file into /usr/local :

tar Cxzvf /usr/local containerd-2.1.1-linux-amd64.tar.gzbin/

After running this it should show like below :

bin/

bin/containerd-shim-runc-v2

bin/containerd

bin/containerd-stress

bin/ctr

- ## To manage containerd with systemd

systemd

If you intend to start containerd via systemd, you should also download the `containerd.service` unit file from https://raw.githubusercontent.com/containerd/containerd/main/containerd.service into `/usr/local/lib/systemd/system/containerd.service`, and run the following commands:

wget https://raw.githubusercontent.com/containerd/containerd/main/containerd.service

```
root@k8worker-2:~# ls
containerd-2.1.1-linux-amd64.tar.gz  containerd.service  snap
root@k8worker-2:~#
```

Move containerd.service to /usr/lib/system/system/ : use below command

mv containerd.service /usr/lib/systemd/system/

Run the below commands to reload and enable containerd :

systemctl daemon-reload

systemctl enable --now containerd

Installing runc

Download the `runc.<ARCH>` binary from https://github.com/opencontainers/runc/releases , verify its sha256sum, and install it as `/usr/local/sbin/runc`

Use that link , you will navigate to :

libseccomp-2.5.6.tar.gz	617 KB	Apr 29
libseccomp-2.5.6.tar.gz.asc	833 Bytes	Apr 29
runc.386	9.75 MB	Apr 29
runc.386.asc	858 Bytes	Apr 29
runc.amd64	11.3 MB	Apr 29
runc.amd64.asc	858 Bytes	Apr 29
runc.arm64	10.8 MB	Apr 29

Copy that link to download : do it on all nodes

wget https://github.com/opencontainers/runc/releases/download/v1.3.0/runc.amd64

```
root@k8master:~# ls
containerd-2.1.1-linux-amd64.tar.gz  runc.amd64  snap
root@k8master:~#
```

Now install the runc using below command:

install -m 755 runc.amd64 /usr/local/sbin/runc

```
root@k8master:~# ls /usr/local/sbin/
runc
```

Installing CNI plugins :

Download the `cni-plugins-<OS>-<ARCH>-<VERSION>.tgz` archive from https://github.com/containernetworking/plugins/releases , verify its sha256sum, and extract it under `/opt/cni/bin`:

Use that link , it will navigate to the repo with CNI plugin:

wget https://github.com/containernetworking/plugins/releases/download/v1.7.1/cni-plugins-linux-amd64-v1.7.1.tgz

```
root@k8worker-1:~# ls
cni-plugins-linux-amd64-v1.7.1.tgz   runc.amd64
containerd-2.1.1-linux-amd64.tar.gz  snap
```

Create a directory :

mkdir -p /opt/cni/bin

Now extract the downloaded file :

tar Cxzvf /opt/cni/bin cni-plugins-linux-amd64-v1.7.1.tgz

```
root@k8master:~# mkdir -p /opt/cni/bin
root@k8master:~# tar Cxzvf /opt/cni/bin cni-plugins-linux-amd64-v1.7.1.tgz
./
./dummy
./tap
./sbr
./bandwidth
./LICENSE
./host-device
./dhcp
./firewall
./loopback
./host-local
./static
./ipvlan
./vlan
./vrf
./bridge
./ptp
./portmap
./macvlan
./README.md
./tuning
```

Till now all the dependencies have been installed :

Now create a Directory for containerd:

mkdir –p /etc/containerd

To generate the default configuration file for `containerd`, run:

```
containerd config default | sudo tee /etc/containerd/config.toml
```

Explanation:

- `containerd config default`: Generates the default config.
- `| sudo tee /etc/containerd/config.toml`: Saves the output to `/etc/containerd/config.toml` with root permissions.

Configuring a cgroup driver

Both the container runtime and the kubelet have a property called "cgroup driver", which is important for the management of cgroups on Linux machines.

Edit the configuration file :

vim /etc/containerd/config.toml

SystemdCgroup = true

```
        cni_max_conf_num = 0
        snapshotter = ''
        sandboxer = 'podsandbox'
        io_type = ''

        [plugins.'io.containerd.cri.v1.runtime'.containerd.runtimes.runc.options]
          BinaryName = ''
          CriuImagePath = ''
          CriuWorkPath = ''
          IoGid = 0
          IoUid = 0
          NoNewKeyring = false
          Root = ''
          ShimCgroup = ''
          SystemdCgroup = true

    [plugins.'io.containerd.cri.v1.runtime'.cni]
      bin_dir = ''
      bin_dirs = ['/opt/cni/bin']
```

Restart containerd for changes to take effect:

systemctl restart containerd

systemctl status containerd

```
containerd.service - containerd container runtime
   Loaded: loaded (/usr/lib/systemd/system/containerd.service; enabled; preset: enabled)
   Active: active (running) since Fri 2025-05-30 10:20:19 UTC; 12s ago
     Docs: https://containerd.io
  Process: 15442 ExecStartPre=/sbin/modprobe overlay (code=exited, status=0/SUCCESS)
 Main PID: 15445 (containerd)
    Tasks: 7
   Memory: 12.5M (peak: 16.0M)
      CPU: 87ms
   CGroup: /system.slice/containerd.service
           └─15445 /usr/local/bin/containerd

ay 30 10:20:19 k8master containerd[15445]: time="2025-05-30T10:20:19.065680916Z" level=inf
ay 30 10:20:19 k8master containerd[15445]: time="2025-05-30T10:20:19.065702243Z" level=inf
ay 30 10:20:19 k8master containerd[15445]: time="2025-05-30T10:20:19.065713316Z" level=inf
ay 30 10:20:19 k8master containerd[15445]: time="2025-05-30T10:20:19.065736442Z" level=inf
```

Now its time to Install: kubeadm , kubelet, kubectl on all the nodes

Update the `apt` package index, install kubelet, kubeadm and kubectl, and pin their version:

```
apt update -y

# apt-transport-https may be a dummy package; if so, you can skip that package

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

If the directory `/etc/apt/keyrings` does not exist, it should be created before the curl command, read the note below.
# sudo mkdir -p -m 755 /etc/apt/keyrings

curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.33/deb/Release.key | sudo gpg --dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg

# This overwrites any existing configuration in /etc/apt/sources.list.d/kubernetes.list
echo 'deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] https://pkgs.k8s.io/core:/stable:/v1.33/deb/ /' | sudo tee /etc/apt/sources.list.d/kubernetes.list
```

```
apt update -y
apt install -y kubelet kubeadm kubectl
apt-mark hold kubelet kubeadm kubectl
```

Initialize the cluster only on control plane : i.,e **on masternode only**

kubeadm init --pod-network-cidr=192.168.0.0/16

Install the Network-driver – calico :

Download the Calico networking manifest for the Kubernetes API datastore.

curl https://raw.githubusercontent.com/projectcalico/calico/v3.30.0/manifests/calico-typha.yaml -o calico.yaml

1. **Apply the manifest using the following command.**

```
kubectl apply -f calico.yaml
```

after runnin the apply command we have add worker nodes to the control nodes by this command:

kubeadm token create --print-join-command

output;

**kubeadm join 192.168.1.100:6443 --token abcdef.0123456789abcdef **

--discovery-token-ca-cert-hash sha256:1234567890abcdef1234567890abcdef1234567890abcdef1234567890abcdef

Got an error while running:

Kubectl get nodes

Result :

```
root@k8master:~# kubectl get nodes
NAME        STATUS     ROLES           AGE     VERSION
k8master    NotReady   control-plane   9m13s   v1.33.1
```

Forgot to open the port in Secutity group inbound rules :

Port no.-> 6443

Protocol -> custom tcp

Ipv4 -.> Anywhere

Then try again : **kubectl get nodes**

```
root@k8master:~# kubectl get nodes
NAME          STATUS     ROLES           AGE   VERSION
k8master      Ready      control-plane   35m   v1.33.1
k8worker-1    NotReady   <none>          27s   v1.33.1
k8worker-2    NotReady   <none>          27s   v1.33.1
```

But here worker nodes showing the status – NotReady : - Check Calico pods:

kubeadm init --pod-network-cidr=192.168.0.0/16

```
root@k8master:~# kubectl get nodes
NAME          STATUS   ROLES           AGE     VERSION
k8master      Ready    control-plane   39m     v1.33.1
k8worker-1    Ready    <none>          4m32s   v1.33.1
k8worker-2    Ready    <none>          4m32s   v1.33.1
root@k8master:~#
```

kubectl get nodes -o wide

```
root@k8master:~#  kubectl get nodes -o wide
NAME          STATUS   ROLES           AGE   VERSION   INTERNAL-IP    EXTERNAL-IP   OS-IMAGE             KERNEL-VERSION   CONTAINER-RUNTIME
k8master      Ready    control-plane   56m   v1.33.1   172.31.10.4    <none>        Ubuntu 24.04.2 LTS   6.8.0-1024-aws   containerd://2.1.1
k8worker-1    Ready    <none>          20m   v1.33.1   172.31.39.47   <none>        Ubuntu 24.04.2 LTS   6.8.0-1024-aws   containerd://2.1.1
k8worker-2    Ready    <none>          20m   v1.33.1   172.31.80.40   <none>        Ubuntu 24.04.2 LTS   6.8.0-1024-aws   containerd://2.1.1
```

Now try to create a pod :

Kubectl run <pod name> --image=<image name>

kubectl get pods

kubectl delete pod mypod

```
root@k8master:~# kubectl run mypod --image=nginx
pod/mypod created
root@k8master:~# kubectl get pods
NAME    READY   STATUS    RESTARTS   AGE
mypod   1/1     Running   0          12s
root@k8master:~# kubectl delete pod mypod
pod "mypod" deleted
root@k8master:~# kubectl get pods
No resources found in default namespace.
root@k8master:~#
```

Some other tests :

kubectl create deployment mynginx --image=nginx # Create a deployment

kubectl scale deployment mynginx --replicas=3 **# Scale to 3 replicas**

kubectl get deployments **# List deployments**

kubectl get svc

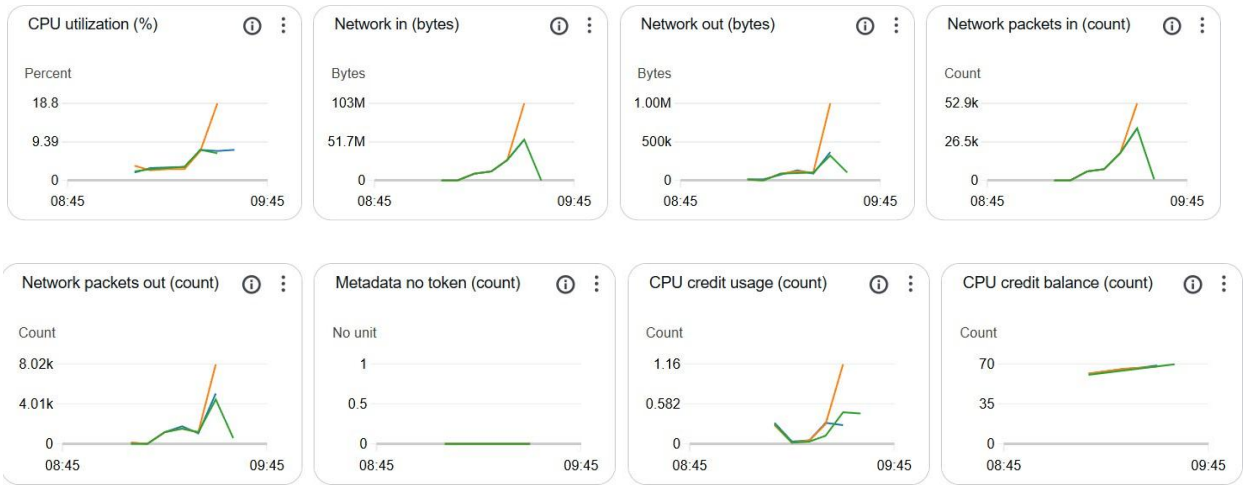
```
root@k8master:~# kubectl get svc
NAME          TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
kubernetes    ClusterIP   10.96.0.1    <none>        443/TCP   61m
root@k8master:~# kubectl get deployments
NAME      READY   UP-TO-DATE   AVAILABLE   AGE
mynginx   3/3     3            3           3m43s
root@k8master:~#
```

```
root@k8master:~# kubectl get events
LAST SEEN   TYPE      REASON                   OBJECT                      MESSAGE
59m         Normal    Starting                 node/k8master               Starting kubelet.
59m         Warning   InvalidDiskCapacity      node/k8master               invalid capacity 0 on image filesystem
59m         Normal    NodeAllocatableEnforced  node/k8master               Updated Node Allocatable limit across pods
59m         Normal    NodeHasSufficientMemory  node/k8master               Node k8master status is now: NodeHasSufficientMemory
59m         Normal    NodeHasNoDiskPressure    node/k8master               Node k8master status is now: NodeHasNoDiskPressure
59m         Normal    NodeHasSufficientPID     node/k8master               Node k8master status is now: NodeHasSufficientPID
59m         Normal    RegisteredNode           node/k8master               Node k8master event: Registered Node k8master in Controller
59m         Normal    Starting                 node/k8master
44m         Normal    NodeReady                node/k8master               Node k8master status is now: NodeReady
32m         Normal    Starting                 node/k8master               Starting kubelet.
32m         Warning   InvalidDiskCapacity      node/k8master               invalid capacity 0 on image filesystem
32m         Normal    NodeAllocatableEnforced  node/k8master               Updated Node Allocatable limit across pods
32m         Normal    NodeHasSufficientMemory  node/k8master               Node k8master status is now: NodeHasSufficientMemory
32m         Normal    NodeHasNoDiskPressure    node/k8master               Node k8master status is now: NodeHasNoDiskPressure
```

```
root@k8master:~# kubectl create deployment mynginx --image=nginx
deployment.apps/mynginx created
root@k8master:~# kubectl get pods
NAME                        READY   STATUS    RESTARTS   AGE
mynginx-5cb564657c-f6ctn    1/1     Running   0          10s
root@k8master:~# kubectl scale deployment mynginx --replicas=3
deployment.apps/mynginx scaled
root@k8master:~# kubectl get pods
NAME                        READY   STATUS              RESTARTS   AGE
mynginx-5cb564657c-b4w9n    0/1     ContainerCreating   0          5s
mynginx-5cb564657c-f6ctn    1/1     Running             0          29s
mynginx-5cb564657c-zhj4b    0/1     ContainerCreating   0          5s
root@k8master:~# kubectl get pods
NAME                        READY   STATUS    RESTARTS   AGE
mynginx-5cb564657c-b4w9n    1/1     Running   0          16s
mynginx-5cb564657c-f6ctn    1/1     Running   0          40s
mynginx-5cb564657c-zhj4b    1/1     Running   0          16s
root@k8master:~#
```

kubectl get pods -o wide

```
root@k8master:~# kubectl get pods -o wide
NAME                        READY   STATUS    RESTARTS   AGE     IP               NODE         NOMINATED NODE   READINESS GATES
mynginx-5cb564657c-b4w9n    1/1     Running   0          5m21s   192.168.162.194  k8worker-2   <none>           <none>
mynginx-5cb564657c-f6ctn    1/1     Running   0          5m45s   192.168.0.2      k8worker-1   <none>           <none>
mynginx-5cb564657c-zhj4b    1/1     Running   0          5m21s   192.168.162.193  k8worker-2   <none>           <none>
root@k8master:~#
```


CPU utilization (%)
Percent
18.8
9.39
0
08:45
09:45
Network in (bytes)
Bytes
103M
51.7M
0
08:45
09:45
Network out (bytes)
Bytes
1.00M
500k
0
08:45
09:45
Network packets in (count)
Count
52.9k
26.5k
0
08:45
09:45
Network packets out (count)
Count
8.02k
4.01k
0
08:45
09:45
Metadata no token (count)
No unit
1
0.5
0
08:45
09:45
CPU credit usage (count)
Count
1.16
0.582
0
08:45
09:45
CPU credit balance (count)
Count
70
35
0
08:45
09:45

Thank you