

Aim: To understand the Kubernetes Cluster Architecture, install and Spin Up a Kubernetes Cluster on Linux Machines/Cloud

1. Create 3 EC-2 instances with all running on Ubuntu as OS with inbound SSH allowed and the proper key pair

Launch an instance [Info](#)

Amazon EC2 allows you to create virtual machines, or instances, that run on the AWS Cloud. Quickly get started by following the simple steps below.

Name and tags [Info](#)

Name

[Add additional tags](#)

▼ Application and OS Images (Amazon Machine Image) [Info](#)

An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. Search or Browse for AMIs if you don't see what you are looking for below


Recents

Quick Start

Amazon Linux

aws


macOS




Ubuntu

ubuntu


Windows




Red Hat



SUSE Linux





[Browse more AMIs](#)

Including AMIs from AWS, Marketplace and the Community

To efficiently run a Kubernetes cluster, select an instance type of at least t2.medium as Kubernetes recommends at least 2 vCPU to run smoothly

▼ Instance type
[Info](#) | [Get advice](#)

Instance type

t2.medium
Family: t2 2 vCPU 4 GiB Memory Current generation: true
On-Demand Linux base pricing: 0.0464 USD per Hour
On-Demand RHEL base pricing: 0.0752 USD per Hour
On-Demand Windows base pricing: 0.0644 USD per Hour
On-Demand SUSE base pricing: 0.1464 USD per Hour

☒ All generations
[Compare instance types](#)

Additional costs apply for AMIs with pre-installed software

▼ Key pair (login)
[Info](#)

You can use a key pair to securely connect to your instance. Ensure that you have access to the selected key pair before you launch the instance.

Key pair name - *required*

kub1

[Create new key pair](#)

Also edit the inbound rules accordingly for the master as well as node/worker machines

Master :

Inbound rules										
Inbound rules (7)										
<input type="text" value="Search"/> Manage tags Edit inbound rules										
<input type="checkbox"/>	Name	Security group rule...	IP version	Type	Protocol	Port range	Source	Description		
<input type="checkbox"/>	-	sgr-055612586f49177...	IPv4	HTTP	TCP	80	0.0.0.0/0	-		
<input type="checkbox"/>	-	sgr-07d74e0579139b...	IPv4	All TCP	TCP	0 - 65535	0.0.0.0/0	-		
<input type="checkbox"/>	-	sgr-0e5f813a074e97305	IPv4	Custom TCP	TCP	10251	0.0.0.0/0	-		
<input type="checkbox"/>	-	sgr-0fd779d78650549...	IPv4	Custom TCP	TCP	10252	0.0.0.0/8	-		
<input type="checkbox"/>	-	sgr-0ab976736e5975...	IPv4	Custom TCP	TCP	6443	0.0.0.0/24	-		
<input type="checkbox"/>	-	sgr-020e888451459e...	IPv4	SSH	TCP	22	0.0.0.0/0	-		
<input type="checkbox"/>	-	sgr-0b72d997ec8e4c20a	IPv4	Custom TCP	TCP	10250	0.0.0.0/32	-		

Worker 1 and Worker 2:

Inbound rules (6)									
<input type="text" value="Search"/> < 1 > ⌂									
<input type="checkbox"/>	Name	Security group rule...	IP version	Type	Protocol	Port range	Source	Description	
<input type="checkbox"/>	-	sgr-0be6fa8163e229558	IPv4	Custom TCP	TCP	30000 - 32767	0.0.0.0/0	-	
<input type="checkbox"/>	-	sgr-06d2c234c61af508e	IPv4	Custom TCP	TCP	10250	0.0.0.0/0	-	
<input type="checkbox"/>	-	sgr-0e5853c91cde89d87	IPv4	HTTP	TCP	80	0.0.0.0/0	-	
<input type="checkbox"/>	-	sgr-031248dbc56ff4687	IPv4	SSH	TCP	22	0.0.0.0/0	-	
<input type="checkbox"/>	-	sgr-0bc0b5370fd55a4f	IPv4	All traffic	All	All	0.0.0.0/0	-	
<input type="checkbox"/>	-	sgr-0ddccad44425bb2...	IPv4	All TCP	TCP	0 - 65535	0.0.0.0/0	-	

In this way create 3 instances namely master, worker-1 and worker-2

Instances (3) <small>info</small>											
<input type="text" value="Find instance by attribute or tag (case-sensitive)"/> All states Last updated less than a minute ago Refresh Connect Instance state Actions Launch instances < 1 > ⌂											
<input type="checkbox"/>	Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4 ...	Elastic IP	IPv6
<input type="checkbox"/>	master	i-0604dbbc26a0a4f01	Running	t2.medium	Initializing	View alarms	us-east-1a	ec2-54-85-79-186.com...	54.85.79.186	-	-
<input type="checkbox"/>	worker-1	i-05548bd7fe0a7292f	Running	t2.medium	Initializing	View alarms	us-east-1a	ec2-54-196-211-209.co...	54.196.211.209	-	-
<input type="checkbox"/>	worker-2	i-007a15dade39c85b0	Running	t2.medium	Initializing	View alarms	us-east-1a	ec2-18-209-62-85.com...	18.209.62.85	-	-

2. SSH into all 3 machines each in separate terminal

a. You can do it through the aws console directly

```

Welcome to Ubuntu 24.04.1 LTS (GNU/Linux 6.8.0-1016-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/pro

System information as of Wed Oct  2 08:10:57 UTC 2024

System load:  0.09          Processes:      138
Usage of /:   65.1% of 6.71GB Users logged in:  1
Memory usage: 19%          IPv4 address for enX0: 172.31.93.13
Swap usage:   0%

 * Ubuntu Pro delivers the most comprehensive open source security and
   compliance features.

   https://ubuntu.com/aws/pro

Expanded Security Maintenance for Applications is not enabled.

6 updates can be applied immediately.
5 of these updates are standard security updates.
To see these additional updates run: apt list --upgradable

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status

Last login: Wed Oct  2 07:24:25 2024 from 103.250.98.143
ubuntu@ip-172-31-93-13:~$

```

Or

b. Locate your key from the Downloads folder and open it in cmd and paste this command

ssh -i <-your-key->.pem ec2-user<ip-address of instance>

```
PS C:\Users\Avan> cd Downloads
PS C:\Users\Avan\Downloads> ssh -i "kub1.pem" ubuntu@ec2-44-201-103-71.compute-1.amazonaws.com
Welcome to Ubuntu 24.04.1 LTS (GNU/Linux 6.8.0-1016-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/pro

System information as of Wed Oct  2 07:24:24 UTC 2024

System load:  0.08               Processes:           117
Usage of /:   23.1% of 6.71GB    Users logged in:    0
Memory usage: 5%                IPv4 address for enX0: 172.31.93.13
Swap usage:   0%

Expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status

Last login: Wed Oct  2 07:22:47 2024 from 103.250.98.143
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.
```

With this you can continue your commands through local terminal

3. From now on, until mentioned, perform these steps on all 3 machines.

Install Docker

sudo yum install docker -y

```
[ec2-user@ip-172-31-31-212 ~]$ sudo yum install docker -y
Last metadata expiration check: 0:33:43 ago on Thu Sep 12 13:11:13 2024.
Dependencies resolved.

Package Architecture Version
Installing:
docker x86_64 25.0.6-1.amzn2023.0.2
Installing dependencies:
containerd x86_64 1.7.20-1.amzn2023.0.1
iptables-libs x86_64 1.8.8-3.amzn2023.0.2
iptables-nft x86_64 1.8.8-3.amzn2023.0.2
libbogroup x86_64 3.0-1.amzn2023.0.1
libnetfilter_comtrack x86_64 1.0.8-2.amzn2023.0.2
libnftnl x86_64 1.0.1-19.amzn2023.0.2
libnftnl x86_64 1.2.2-2.amzn2023.0.2
pigz x86_64 2.5-1.amzn2023.0.3
runC x86_64 1.1.13-1.amzn2023.0.1

Transaction Summary
```

Then, configure cgroup in a daemon.json file by using following commands

- `cd /etc/docker`
- `cat <<EOF | sudo tee /etc/docker/daemon.json`
`{`
 `"exec-opts": ["native.cgroupdriver=systemd"],`
 `"log-driver": "json-file",`
 `"log-opts": {`
 `"max-size": "100m"`
 `},`
 `"storage-driver": "overlay2"`
`}`
`EOF`

```
[ec2-user@ip-172-31-20-75 ~]$ cd /etc/docker
[ec2-user@ip-172-31-20-75 docker]$ cat <<EOF | sudo tee /etc/docker/daemon.json
{
  "exec-opts": ["native.cgroupdriver=systemd"],
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "100m"
  },
  "storage-driver": "overlay2"
}
EOF
{
  "exec-opts": ["native.cgroupdriver=systemd"],
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "100m"
  },
  "storage-driver": "overlay2"
}
[ec2-user@ip-172-31-20-75 docker]$ ls
daemon.json  kubectl
```

- `sudo systemctl enable docker`
- `sudo systemctl daemon-reload`
- `sudo systemctl restart docker`
- `docker -v`

```
[ec2-user@ip-172-31-31-212 docker]$ sudo systemctl enable docker
Created symlink /etc/systemd/system/multi-user.target.wants/docker.service → /usr/lib/systemd/system/docker.service.
[ec2-user@ip-172-31-31-212 docker]$ sudo systemctl daemon-reload
[ec2-user@ip-172-31-31-212 docker]$ sudo systemctl restart docker
[ec2-user@ip-172-31-31-212 docker]$ docker -v
Docker version 25.0.5, build 5dc9bcc
[ec2-user@ip-172-31-31-212 docker]$
```

4. Install Kubernetes on all 3 machines

Run the below command to install Kubernetes.

```
curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.31/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:/v1.31/deb/ ' | sudo tee /etc/apt/sources.list.d/kubernetes.list
```

```
ubuntu@ip-172-31-93-13:~$ curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.31/deb/Release.key
| sudo gpg --dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg
ubuntu@ip-172-31-93-13:~$ echo 'deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg]
https://pkgs.k8s.io/core:/stable:/v1.31/deb/ ' | sudo tee /etc/apt/sources.list.d/kubernetes.list
deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] https://pkgs.k8s.io/core:/stable:/v1.31/deb/
ubuntu@ip-172-31-93-13:~$ sudo apt-get update
sudo apt-get install -y kubelet kubeadm kubectl
sudo apt-mark hold kubelet kubeadm kubectl
Hit:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble InRelease
Hit:2 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates InRelease
Hit:3 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-backports InRelease
Hit:4 http://security.ubuntu.com/ubuntu noble-security InRelease
Hit:5 https://download.docker.com/linux/ubuntu noble InRelease
Get:6 https://prod-cdn.packages.k8s.io/repositories/isv:/kubernetes:/core:/stable:/v1.31/deb InRelease [1186 B]
Get:7 https://prod-cdn.packages.k8s.io/repositories/isv:/kubernetes:/core:/stable:/v1.31/deb Packages [4865 B]
Fetched 6051 B in 1s (10.6 kB/s)
Reading package lists... Done
W: https://download.docker.com/linux/ubuntu/dists/noble/InRelease: Key is stored in legacy trusted.gpg keyring (/etc/apt/trusted.gpg), see the DEPRECATION section in apt-key(8) for details.
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  conntrack cri-tools kubernetes-cni
The following NEW packages will be installed:
  conntrack cri-tools kubeadm kubectl kubelet kubernetes-cni
```

Running kernel seems to be up-to-date.

No services need to be restarted.

No containers need to be restarted.

No user sessions are running outdated binaries.

No VM guests are running outdated hypervisor (qemu) binaries on this host.

kubelet set on hold.

kubeadm set on hold.

kubectrl set on hold.

Run this command to install the kubelet server

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

```
ubuntu@ip-172-31-93-13:~$ sudo apt-get update
sudo apt-get install -y kubelet kubeadm kubectl
sudo apt-mark hold kubelet kubeadm kubectl
Hit:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble InRelease
Hit:2 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates InRelease
Hit:3 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-backports InRelease
Hit:4 http://security.ubuntu.com/ubuntu noble-security InRelease
Hit:5 https://download.docker.com/linux/ubuntu noble InRelease
Get:6 https://prod-cdn.packages.k8s.io/repositories/isv:/kubernetes:/core:/stable:/v1.31/deb InRelease [1186 B]
Get:7 https://prod-cdn.packages.k8s.io/repositories/isv:/kubernetes:/core:/stable:/v1.31/deb Packages [4865 B]
Fetched 6051 B in 1s (10.6 kB/s)
Reading package lists... Done
W: https://download.docker.com/linux/ubuntu/dists/noble/InRelease: Key is stored in legacy trusted.gpg keyring (/etc/apt/trusted.gpg), see the DEPRECATION section in apt-key(8) for details.
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  conntrack cri-tools kubernetes-cni
```

Run the below commands to containerize the server

- `sudo systemctl enable --now kubelet`
- `sudo apt-get install -y containerd`

```
ubuntu@ip-172-31-93-13:~$ sudo systemctl enable --now kubelet
sudo apt-get install -y containerd
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following packages were automatically installed and are no longer required:
  docker-buildx-plugin docker-ce-cli docker-ce-rootless-extras
  docker-compose-plugin libltdl7 libslirp0 pigz slirp4netns
Use 'sudo apt autoremove' to remove them.
The following additional packages will be installed:
  runc
The following packages will be REMOVED:
  containerd.io docker-ce
The following NEW packages will be installed:
  containerd runc
0 upgraded, 2 newly installed, 2 to remove and 6 not upgraded.
```

Now the next command initializes the containerd configuration file with default settings, creating the config.toml file, which will be used by containerd to manage containers

```
ubuntu@ip-172-31-93-13:~$ sudo mkdir -p /etc/containerd
sudo containerd config default | sudo tee /etc/containerd/config.toml
disabled_plugins = []
imports = []
oom_score = 0
plugin_dir = ""
required_plugins = []
root = "/var/lib/containerd"
state = "/run/containerd"
temp = ""
version = 2

[cgroup]
  path = ""

[debug]
  address = ""
  format = ""
  gid = 0
  level = ""
  uid = 0

[grpc]
  address = "/run/containerd/containerd.sock"
  gid = 0
  max_recv_message_size = 16777216
  max_send_message_size = 16777216
  tcp_address = ""
  tcp_tls_ca = ""
  tcp_tls_cert = ""
  tcp_tls_key = ""
  uid = 0

[metrics]
  address = ""
  grpc_histogram = false
```

```
[proxy_plugins]

[stream_processors]

[stream_processors."io.containerd.ocicrypt.decoder.v1.tar"]
  accepts = ["application/vnd.oci.image.layer.v1.tar+encrypted"]
  args = ["--decryption-keys-path", "/etc/containerd/ocicrypt/keys"]
  env = ["OCICRYPT_KEYPROVIDER_CONFIG=/etc/containerd/ocicrypt/ocicrypt_keyprovider.conf"]
]
  path = "ctd-decoder"
  returns = "application/vnd.oci.image.layer.v1.tar"

[stream_processors."io.containerd.ocicrypt.decoder.v1.tar.gz"]
  accepts = ["application/vnd.oci.image.layer.v1.tar+gzip+encrypted"]
  args = ["--decryption-keys-path", "/etc/containerd/ocicrypt/keys"]
  env = ["OCICRYPT_KEYPROVIDER_CONFIG=/etc/containerd/ocicrypt/ocicrypt_keyprovider.conf"]
]
  path = "ctd-decoder"
  returns = "application/vnd.oci.image.layer.v1.tar+gzip"

[timeouts]
  "io.containerd.timeout.bolt.open" = "0s"
  "io.containerd.timeout.metrics.shimstats" = "2s"
  "io.containerd.timeout.shim.cleanup" = "5s"
  "io.containerd.timeout.shim.load" = "5s"
  "io.containerd.timeout.shim.shutdown" = "3s"
  "io.containerd.timeout.task.state" = "2s"

[ttrpc]
  address = ""
  gid = 0
  uid = 0
```

Restarting and running the container with these commands

- `sudo systemctl restart containerd`

- sudo systemctl enable containerd
- sudo systemctl status containerd

```
ubuntu@ip-172-31-93-13:~$ sudo systemctl restart containerd
sudo systemctl enable containerd
sudo systemctl status containerd
● containerd.service - containerd container runtime
   Loaded: loaded (/usr/lib/systemd/system/containerd.service; enabled; p>
   Active: active (running) since Wed 2024-10-02 07:41:44 UTC; 207ms ago
     Docs: https://containerd.io
   Main PID: 4835 (containerd)
      Tasks: 7
   Memory: 13.7M (peak: 14.1M)
      CPU: 64ms
   CGroup: /system.slice/containerd.service
           └─4835 /usr/bin/containerd

Oct 02 07:41:44 ip-172-31-93-13 containerd[4835]: time="2024-10-02T07:41:44">
Oct 02 07:41:44 ip-172-31-93-13 containerd[4835]: time="2024-10-02T07:41:44">
Oct 02 07:41:44 ip-172-31-93-13 containerd[4835]: time="2024-10-02T07:41:44">
Oct 02 07:41:44 ip-172-31-93-13 containerd[4835]: time="2024-10-02T07:41:44">
Oct 02 07:41:44 ip-172-31-93-13 containerd[4835]: time="2024-10-02T07:41:44">
Oct 02 07:41:44 ip-172-31-93-13 containerd[4835]: time="2024-10-02T07:41:44">
Oct 02 07:41:44 ip-172-31-93-13 containerd[4835]: time="2024-10-02T07:41:44">
Oct 02 07:41:44 ip-172-31-93-13 containerd[4835]: time="2024-10-02T07:41:44">
Oct 02 07:41:44 ip-172-31-93-13 systemd[1]: Started containerd.service - co>
Oct 02 07:41:44 ip-172-31-93-13 containerd[4835]: time="2024-10-02T07:41:44">
```

Below command :

- sudo apt-get install -y socat

installs the socat utility, a versatile networking tool used for bidirectional data transfer between two endpoints, commonly utilized in Kubernetes for port forwarding and networking tasks.

```
ubuntu@ip-172-31-93-13:~$ sudo apt-get install -y socat
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following packages were automatically installed and are no longer required:
  docker-buildx-plugin docker-ce-cli docker-ce-rootless-extras
  docker-compose-plugin libltdl7 libslirp0 pigz slirp4netns
Use 'sudo apt autoremove' to remove them.
The following NEW packages will be installed:
  socat
0 upgraded, 1 newly installed, 0 to remove and 6 not upgraded.
Need to get 374 kB of archives.
After this operation, 1649 kB of additional disk space will be used.
Get:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu/noble/main amd64 socat amd64 1.8.0.0-4
build3 [374 kB]
Fetched 374 kB in 0s (18.0 MB/s)
Selecting previously unselected package socat.
(Reading database ... 68203 files and directories currently installed.)
Preparing to unpack .../socat_1.8.0.0-4build3_amd64.deb ...
Unpacking socat (1.8.0.0-4build3) ...
Setting up socat (1.8.0.0-4build3) ...
Processing triggers for man-db (2.12.0-4build2) ...
Scanning processes...
Scanning linux images...

Running kernel seems to be up-to-date.

No services need to be restarted.

No containers need to be restarted.

No user sessions are running outdated binaries.
```

5. Perform this ONLY on the Master machine

Initialize kubernetes by typing below command

- `sudo kubeadm init --pod-network-cidr=10.244.0.0/16`

```
ubuntu@ip-172-31-93-13:~$ sudo kubeadm init --pod-network-cidr=10.244.0.0/16
[init] Using Kubernetes version: v1.31.0
[preflight] Running pre-flight checks
[preflight] Pulling images required for setting up a Kubernetes cluster
[preflight] This might take a minute or two, depending on the speed of your internet connection
[preflight] You can also perform this action beforehand using 'kubeadm config images pull'
W1002 07:42:59.444277 5045 checks.go:846] detected that the sandbox image "registry.k8s.io/pause:3.8" of the container runtime is inconsistent with that used by kubeadm. It is recommended to use "registry.k8s.io/pause:3.10" as the CRI sandbox image.
[certs] Using certificateDir folder "/etc/kubernetes/pki"
[certs] Generating "ca" certificate and key
[certs] Generating "apiserver" certificate and key
[certs] apiserver serving cert is signed for DNS names [ip-172-31-93-13 kubernetes kubernetes.default kubernetes.default.svc kubernetes.default.svc.cluster.local] and IPs [10.96.0.1 172.31.93.13]
[certs] Generating "apiserver-kubelet-client" certificate and key
[certs] Generating "front-proxy-ca" certificate and key
[certs] Generating "front-proxy-client" certificate and key
[certs] Generating "etcd/ca" certificate and key
[certs] Generating "etcd/server" certificate and key
[certs] etcd/server serving cert is signed for DNS names [ip-172-31-93-13 localhost] and IPs [172.31.93.13 127.0.0.1 ::1]
[certs] Generating "etcd/peer" certificate and key
[certs] etcd/peer serving cert is signed for DNS names [ip-172-31-93-13 localhost] and IPs [172.31.93.13 127.0.0.1 ::1]
```

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 172.31.93.13:6443 --token v6qmnw.lhucizuo02h2fkwo \
--discovery-token-ca-cert-hash sha256:78b5edd6f35bd7763eb06d22cdfed85728c12119f083a11e434b96fba530da0c
```

Copy the mkdir and chown commands from the top and execute them

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

Copy this join link and save it in clipboard (copy from your output as it different for each master instance)

Example :

```
sudo kubeadm join 172.31.93.13:6443 --token v6qmnw.lhucizuo02h2fkwo \
--discovery-token-ca-cert-hash sha256:78b5edd6f35bd7763eb06d22cdfed85728c12119f083a11e434b96fba530da0c
```

```
ubuntu@ip-172-31-84-116:~$ sudo kubeadm join 172.31.93.13:6443 --token v6qmnw.lhucizuo02h2fkwo \
--discovery-token-ca-cert-hash sha256:78b5edd6f35bd7763eb06d22cdfed85728c12119f083a11e434b96fba530da0c
[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-check] Waiting for a healthy kubelet at http://127.0.0.1:10248/healthz. This can take up to 4m0s
[kubelet-check] The kubelet is healthy after 501.194701ms
[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.
```

Running the kubectl get nodes to check the cluster connection

```
ubuntu@ip-172-31-93-13:~$ kubectl get nodes
```

NAME	STATUS	ROLES	AGE	VERSION
ip-172-31-84-116	NotReady	<none>	39s	v1.31.1
ip-172-31-91-194	NotReady	<none>	35s	v1.31.1
ip-172-31-93-13	NotReady	control-plane	3m59s	v1.31.1

Since Status is **NotReady** we have to add a network plugin. And also we have to give the name to the nodes.

kubectl apply -f <https://docs.projectcalico.org/manifests/calico.yaml>

```
ubuntu@ip-172-31-93-13:~$ kubectl apply -f https://docs.projectcalico.org/manifests/calico.yaml
poddisruptionbudget.policy/calico-kube-controllers created
serviceaccount/calico-kube-controllers created
serviceaccount/calico-node created
configmap/calico-config created
customresourcedefinition.apiextensions.k8s.io/bgpconfigurations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/bgppeers.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/blockaffinities.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/caliconodestatuses.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/clusterinformations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/felixconfigurations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/globalnetworkpolicies.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/globalnetworksets.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/hostendpoints.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ipamblocks.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ipamconfigs.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ipamhandles.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ippools.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ippreservations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/kubecontrollersconfigurations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/networkpolicies.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/networksets.crd.projectcalico.org created
clusterrole.rbac.authorization.k8s.io/calico-kube-controllers created
clusterrole.rbac.authorization.k8s.io/calico-node created
clusterrolebinding.rbac.authorization.k8s.io/calico-kube-controllers created
clusterrolebinding.rbac.authorization.k8s.io/calico-node created
daemonset.apps/calico-node created
deployment.apps/calico-kube-controllers created
```

6. Now check the status for kubelet server

sudo systemctl status kubelet

7. Now Run command **kubectl get nodes -o wide** we can see Status is ready

```
ubuntu@ip-172-31-93-13:~$ kubectl get nodes -o wide
```

NAME	STATUS	ROLES	AGE	VERSION	INTERNAL-IP
EXTERNAL-IP	OS-IMAGE		KERNEL-VERSION	CONTAINER-RUNTIME	
ip-172-31-84-116	Ready	Node1	4m52s	v1.31.1	172.31.84.116
<none>	Ubuntu 24.04.1 LTS		6.8.0-1016-aws	containerd://1.7.12	
ip-172-31-91-194	Ready	Node2	4m48s	v1.31.1	172.31.91.194
<none>	Ubuntu 24.04.1 LTS		6.8.0-1016-aws	containerd://1.7.12	
ip-172-31-93-13	Ready	control-plane	8m12s	v1.31.1	172.31.93.13
<none>	Ubuntu 24.04.1 LTS		6.8.0-1016-aws	containerd://1.7.12	

8. Renaming the master and worker nodes

kubectl label node ip-172-31-84-116 kubernetes.io/role=Node1

(Similarly for your own ip you can rename the nodes and master)

```
ubuntu@ip-172-31-93-13:~$ kubectl label node ip-172-31-84-116 kubernetes.io/role=Node1
node/ip-172-31-84-116 labeled
```

9. Run command **kubectl get nodes -o wide** . And Hence we can see we have Successfully connected Node 1 and Node 2 to the Master. or

kubectl get nodes

```
ubuntu@ip-172-31-93-13:~$ kubectl get nodes -o wide
```

NAME	STATUS	ROLES	AGE	VERSION	INTERNAL-IP	EXTERNAL-IP	OS-IMAGE	KERNEL-VERSION	C
ip-172-31-84-116	Ready	Node1	4m52s	v1.31.1	172.31.84.116	<none>	Ubuntu 24.04.1 LTS	6.8.0-1016-aws	c
ip-172-31-91-194	Ready	Node2	4m48s	v1.31.1	172.31.91.194	<none>	Ubuntu 24.04.1 LTS	6.8.0-1016-aws	c
ip-172-31-93-13	Ready	control-plane	8m12s	v1.31.1	172.31.93.13	<none>	Ubuntu 24.04.1 LTS	6.8.0-1016-aws	c


```
ubuntu@ip-172-31-93-13:~$ kubectl get nodes
```

NAME	STATUS	ROLES	AGE	VERSION
ip-172-31-84-116	Ready	Node1	5m7s	v1.31.1
ip-172-31-91-194	Ready	Node2	5m3s	v1.31.1
ip-172-31-93-13	Ready	control-plane	8m27s	v1.31.1

X

Conclusion :

In this experiment, we aimed to deploy Kubernetes in Docker, connecting a master node with two worker nodes. We encountered several challenges, starting with SSH inbound rule misconfigurations, which were resolved by enabling the correct rules for secure access. It became clear that using **t2.medium** or **t3** instances was essential for sufficient resources to run Kubernetes smoothly. Despite these adjustments, the worker nodes failed to join the cluster. Although the master node was ready, the issue seemed to stem from improper configuration of the worker nodes' kubelet or a networking problem, such as the worker nodes being unable to reach the master node's API server. This could be due to incorrect firewall settings, missing API server certificates, or failure in configuring the kubeadm join process on the worker nodes.