**Kubeadm prerequisites:**

We need to create the EC2 instances with the following configurations:

* Launch one Ubuntu instance (22.04) with minimum of 4 vCPU and 8GB RAM for the master node.
* Launch two Ubuntu instances with minimum of 2vCPU and 4 GB RAM for the worker nodes.

**Kubeadm port requirements:**

Enable the following ports in the security groups of the instances:

* Control plane node:

`6443/tcp` for Kubernetes API Server

`2379-2380` for etcd server client API

`10248-10260` for Kubelet API, Kube-scheduler, Kube-controller-manager, Read-Only Kubelet API, Kubelet health

`80,8080,443` Generic Ports

`30000-32767` for NodePort Services

* Worker plane node:

`10248-10260` for Kubelet API etc

`30000-32767` for NodePort Services

**Cluster setup steps:**

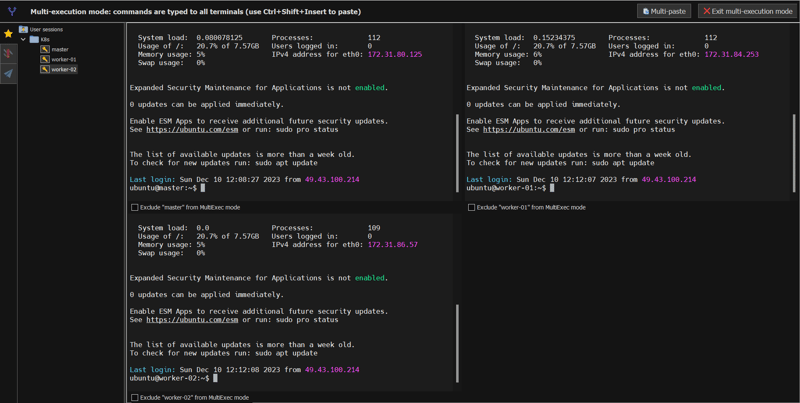
Here are the major steps involved in this setup:

1. Launch 2/3 EC2 t2.xlarge instances with the specified ports open.
2. Install container runtime on all nodes. We will use [cri-o](https://cri-o.io/).
3. Install Kubeadm, Kubelet, and kubectl on all the nodes.
4. Initiate Kubeadm control plane configuration on the master node.
5. Install the Calico network plugin.
6. Join the worker node to the master node using the join command.
7. Validate all cluster components and nodes.
8. Deploy a sample app and validate the app to test our cluster.

Let's start with the setup now!

**Launch EC2 instances:**

* Launch 3 t2.medium instances and SSH into them through MobaXterm.
* Open the 3 nodes in the multi-execution mode.

[](https://media.dev.to/cdn-cgi/image/width=800%2Cheight=%2Cfit=scale-down%2Cgravity=auto%2Cformat=auto/https%3A%2F%2Fdev-to-uploads.s3.amazonaws.com%2Fuploads%2Farticles%2F4dy9cib1k3tzspfnjcqh.png)

Refer to this [video](https://youtu.be/hUoYVZlioRc?si=qMJUQiUJrwMWHQqd) if you're new to MobaXterm.

**Run on all the nodes of the cluster as root user-**

**1. Disable SWAP**

We need to disable Swap for kubeadm to work properly.

swapoff -a

sed -i '/ swap / s/^\(.\*\)$/#\1/g' /etc/fstab

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

Execute the following commands on all the nodes for IPtables to see bridged traffic. Here we are tweaking some kernel parameters and setting them using sysctl.

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 = 1

net.ipv4.ip\_forward = 1

EOF

# Apply sysctl params without reboot

sudo sysctl --system

**3. Install CRI-O Runtime**

The basic requirement for a Kubernetes cluster is a container runtime. We can use any container runtime we want, here we will be using CRI-O.

* Enable cri-o repositories for version 1.28

OS="xUbuntu\_22.04"

VERSION="1.28"

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 for CRI-O to the system’s list of trusted 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 include crictl, a CLI utility for interacting with containers created by the container runtime. When utilizing container runtimes other than Docker, you can employ the crictl utility for debugging containers on the nodes.

**4. Install Kubeadm & Kubelet & Kubectl**

* Install the required dependencies
* sudo apt-get update
* *# apt-transport-https may be a dummy package; if so, you can skip that package*
* sudo apt-get install -y apt-transport-https ca-certificates curl gpg
* Download the GPG key
* *# 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:/v1.29/deb/Release.key | sudo gpg --dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg
* Add the Kubernetes APT repository to your system.
* echo 'deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] https://pkgs.k8s.io/core:/stable:/v1.29/deb/ /' | sudo tee /etc/apt/sources.list.d/kubernetes.list
* Update apt repo

sudo apt-get update -y

* Install the latest version

sudo apt-get install -y kubelet kubeadm kubectl

sudo systemctl enable --now kubelet

* Add hold to the packages to prevent upgrades.

sudo apt-mark hold kubelet kubeadm kubectl

* 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

We now, have all the required utilities and tools for configuring Kubernetes components using kubeadm.

Now, let's initialize Kubeadm On Master Node To Setup Control Plane.

**5. Initialize Kubeadm On Master Node**

* Execute the commands in this section only on the master node.
* Set the following environment variables.

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

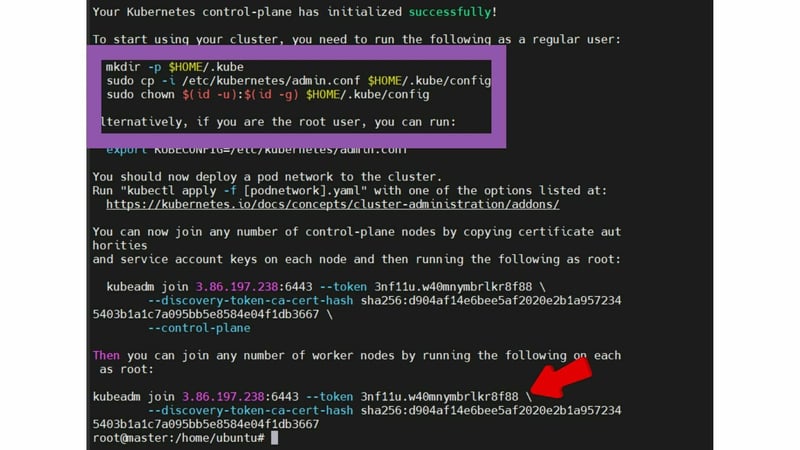
NODENAME=$(hostname -s)

POD\_CIDR="192.168.0.0/16"

* Initialize the master node control plane configurations using the kubeadm command.

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

You should get the following output:

[](https://media.dev.to/cdn-cgi/image/width=800%2Cheight=%2Cfit=scale-down%2Cgravity=auto%2Cformat=auto/https%3A%2F%2Fdev-to-uploads.s3.amazonaws.com%2Fuploads%2Farticles%2Fo0ivtprkghh8r36gf6no.jpg)  
Copy the join command save it somewhere, we will need it for joining the worker node to the master.

* 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

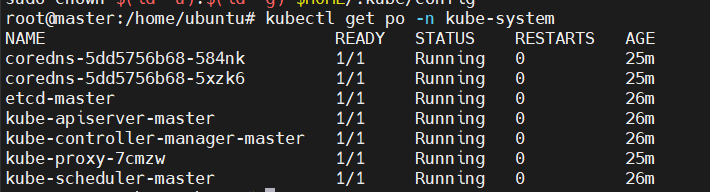
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

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

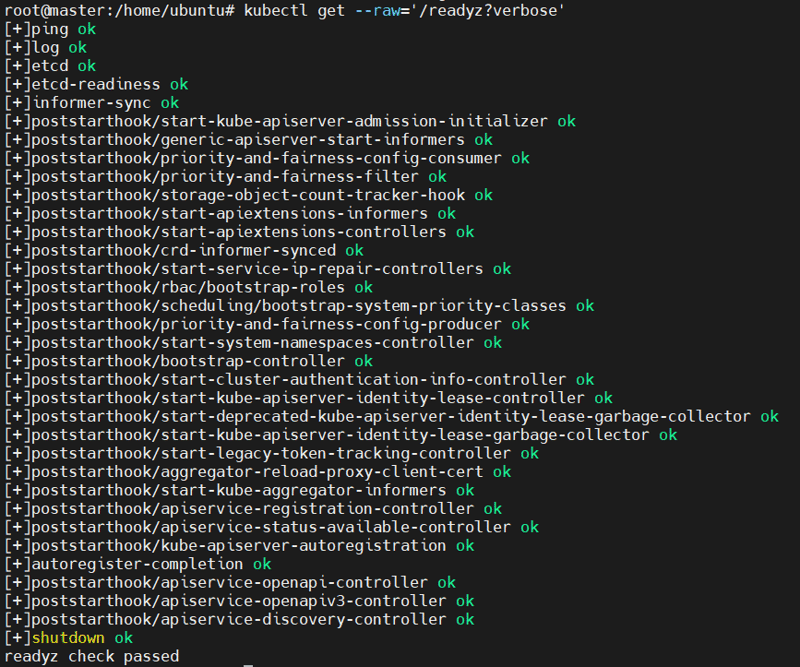
You should get the following output:

[](https://media.dev.to/cdn-cgi/image/width=800%2Cheight=%2Cfit=scale-down%2Cgravity=auto%2Cformat=auto/https%3A%2F%2Fdev-to-uploads.s3.amazonaws.com%2Fuploads%2Farticles%2Fag3w6wd7a50j3iqd9pbb.png)

* You verify all the cluster component health statuses using the following command.

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

You should get the following output:

[](https://media.dev.to/cdn-cgi/image/width=800%2Cheight=%2Cfit=scale-down%2Cgravity=auto%2Cformat=auto/https%3A%2F%2Fdev-to-uploads.s3.amazonaws.com%2Fuploads%2Farticles%2F2npktsyhk6bswy026osh.png)

* 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-

Next, we will be installing CNI tool for pod networking.

**6. Install Calico Network Plugin**

Kubeadm does not configure any network plugin. You need to install a network plugin of your choice for kubernetes pod networking and enable network policy.

We will use the Calico network plugin for this setup.

* Run the following commands to install the Calcio network plugin

kubectl create -f https://raw.githubusercontent.com/projectcalico/calico/v3.26.1/manifests/tigera-operator.yaml

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

kubectl create -f custom-resources.yaml

**7. Joining of Worker nodes to master node**

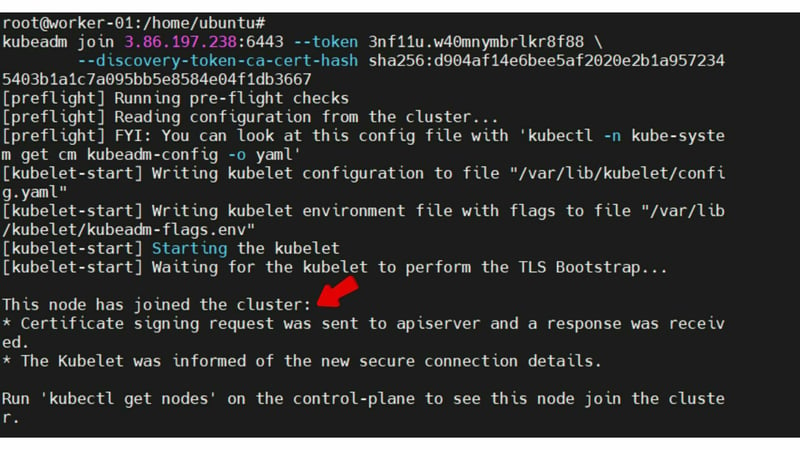
Now, let’s join the worker node to the master node using the Kubeadm join command you have got in the output while setting up the master node.

* Execute the commands in this section only on the worker nodes.
* Here is what the command looks like in my case

kubeadm join 3.86.197.238:6443 --token 3nf11u.w40mnymbrlkr8f88 \

--discovery-token-ca-cert-hash sha256:d904af14e6bee5af2020e2b1a9572345403b1a1c7a095bb5e8584e04f1db3667

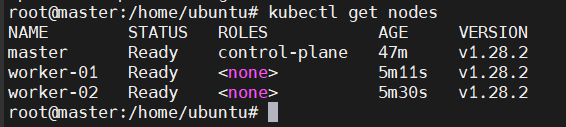
You should get the following output:

[](https://media.dev.to/cdn-cgi/image/width=800%2Cheight=%2Cfit=scale-down%2Cgravity=auto%2Cformat=auto/https%3A%2F%2Fdev-to-uploads.s3.amazonaws.com%2Fuploads%2Farticles%2Feev6c2ixgvia0aakytxd.jpg)

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

kubectl get nodes

You will get the following output:

[](https://media.dev.to/cdn-cgi/image/width=800%2Cheight=%2Cfit=scale-down%2Cgravity=auto%2Cformat=auto/https%3A%2F%2Fdev-to-uploads.s3.amazonaws.com%2Fuploads%2Farticles%2Fomdtlgw5a2ay8o9ugpgf.png)

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

**8. Deploy a sample Nginx App**

Create an Nginx deployment. Execute the following directly on the command line.

kubectl run nginx --image=nginx --port=80

kubectl expose pod nginx --port=80 --type=NodePort

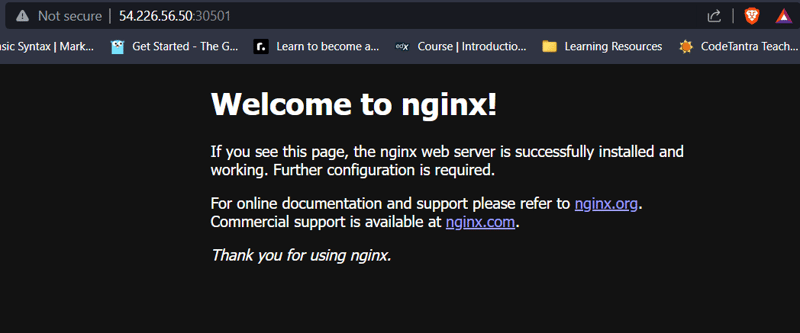
A random port between 30000-32767 and assigned to nginx app.  
To access that port we run the following command:

kubectl get svc

You get an output like this:

[](https://media.dev.to/cdn-cgi/image/width=800%2Cheight=%2Cfit=scale-down%2Cgravity=auto%2Cformat=auto/https%3A%2F%2Fdev-to-uploads.s3.amazonaws.com%2Fuploads%2Farticles%2Fq6l9olzxhqaowdl0ejw3.jpg)  
Here, we can see that the nginx app has been deployed on the port 30501.

We can now access that port with our master node ip or worker nodes ip and get the following result:

[](https://media.dev.to/cdn-cgi/image/width=800%2Cheight=%2Cfit=scale-down%2Cgravity=auto%2Cformat=auto/https%3A%2F%2Fdev-to-uploads.s3.amazonaws.com%2Fuploads%2Farticles%2F7g8q228nhq7211qiz9fp.png)

This shows that we have setup our cluster succesfully!