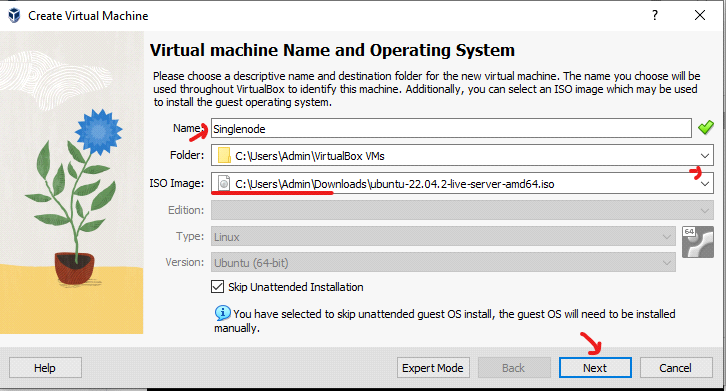
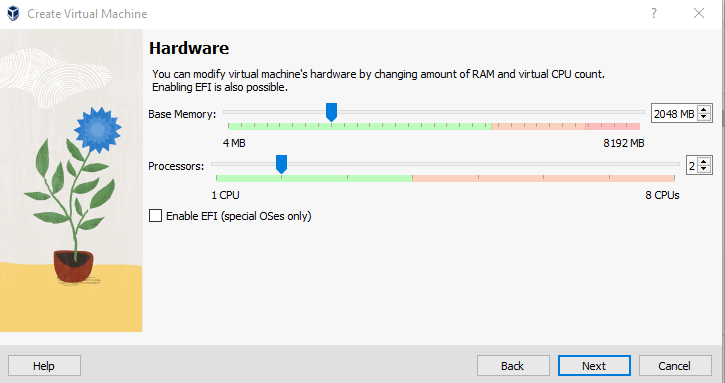
Kubernetes single node setup using kubeadm tool (CRI containerd)

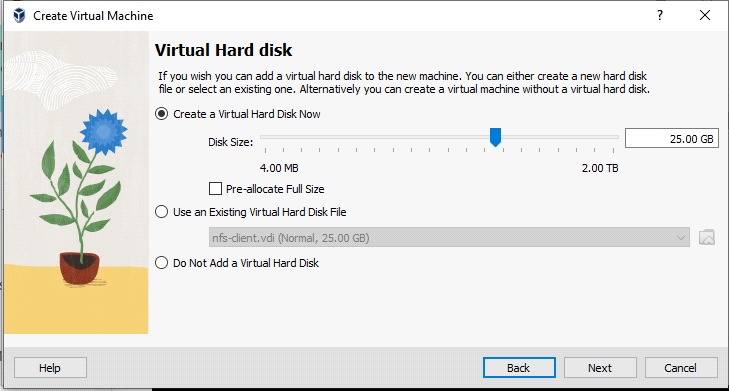
System specifications:

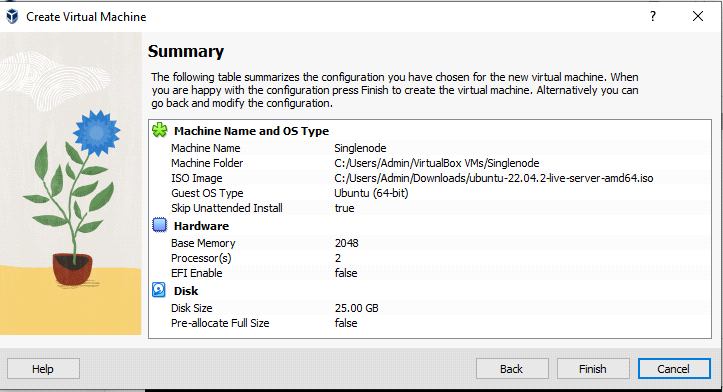
* OS: Ubuntu 22.04 LTS server
* RAM: 4GB or more
* Disk: 30GB+
* CPU: 2 core or more
* Swap Memory: Diable
* Single/Two nodes: VirtualBox was used to launch both nodes (Master Hostname: controlplane and Worker Hostname: computeplaneone)
* Network: Bridge was used to have internal and external access (if not we can use 2 interface, one for NAT and one as host-only adaptor)

Create virtual machine. Give the name and select ISO image and enable skip unattended Installation option. Skip unattended – before start installation to do some particular changes we need to select this option. Click on next.

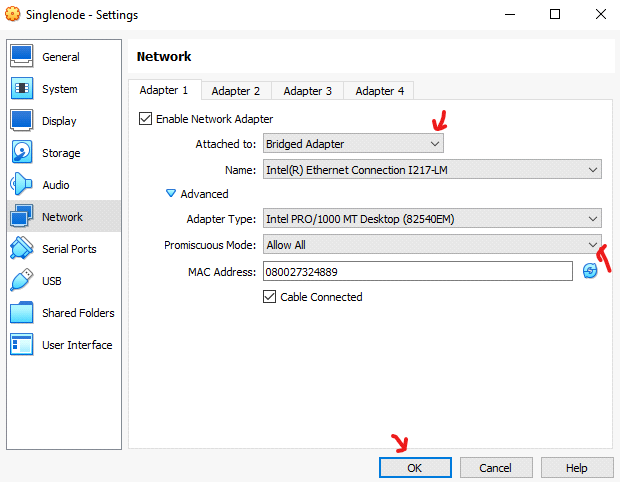








Go to setting – networking – select Bridge network – allow all (to connect with base machine and within the VM’s as well)



Do the remaining installation process as usual.

In the system If reboot is required we will see the file like reboot-required in this path /var/run/reboot-required means if the file is not their then reboot is not required.

<<https://github.com/swapnachatla/kubernetes_latest_manifest/blob/main/Kubernetes/01-kubernetes-architecture-Installation/03-k8s-setup-kubeadm-containerd.md>>

Before installing kubernetes we need to install, enable some modules and networking fileds.

Make the IP add VM must be static for that go iside of the particular path

# vi /etc/netplan/00-installer-config.yaml

network:

ethernets:

enp0s3:

dhcp4: false

addresses: ["192.168.1.62/24"]

routes:

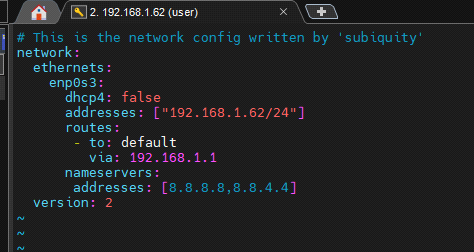
- to: default

via: 192.168.1.1

nameservers:

addresses: [8.8.8.8,8.8.4.4]

version: 2



Save & quit from that file (esc:wq!).

To apply the changes, execute the below command.

# netplan apply



First we need to enable overlay and Bridge-network filters. For that we can use the below commands.

modprobe overlay

modprobe br\_netfilter

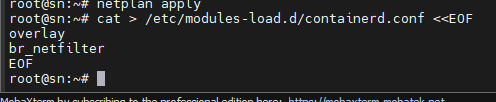
But the drawback of using these commands is after rebooting the system we need to execute the commands again. To enable them permanently we use the below commands.

cat > /etc/modules-load.d/containerd.conf <<EOF

overlay

br\_netfilter

EOF



Now setup the sysctl parameters.

# Setup required sysctl params, these persist across reboots.

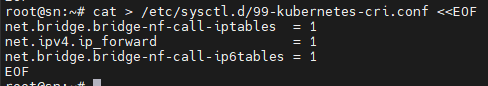
cat > /etc/sysctl.d/99-kubernetes-cri.conf <<EOF

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

net.ipv4.ip\_forward = 1

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

EOF

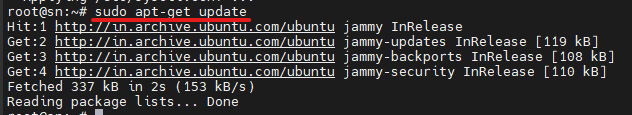


Now to continue without rebooting use the below command.

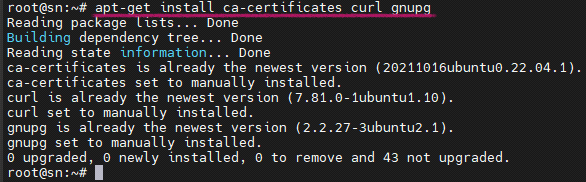
sysctl --system

Now install containerd package. For that we can use the official Docker install on Ubuntu document but don’t install docker just install containerd pkg only. <<https://docs.docker.com/engine/install/ubuntu/>>

# sudo apt-get update

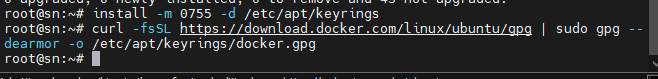


# apt-get install ca-certificates curl gnupg



# install -m 0755 -d /etc/apt/keyrings

# curl -fsSL <https://download.docker.com/linux/ubuntu/gpg> | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg



# chmod a+r /etc/apt/keyrings/docker.gpg

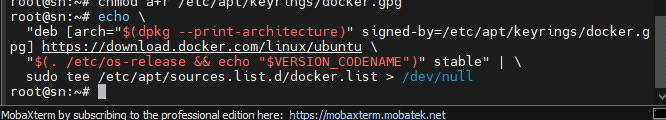


# echo \

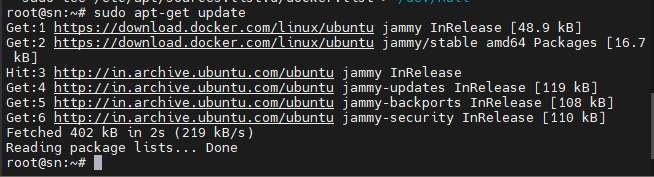
"deb [arch="$(dpkg --print-architecture)" signed-by=/etc/apt/keyrings/docker.gpg] <https://download.docker.com/linux/ubuntu> \

"$(. /etc/os-release && echo "$VERSION\_CODENAME")" stable" | \

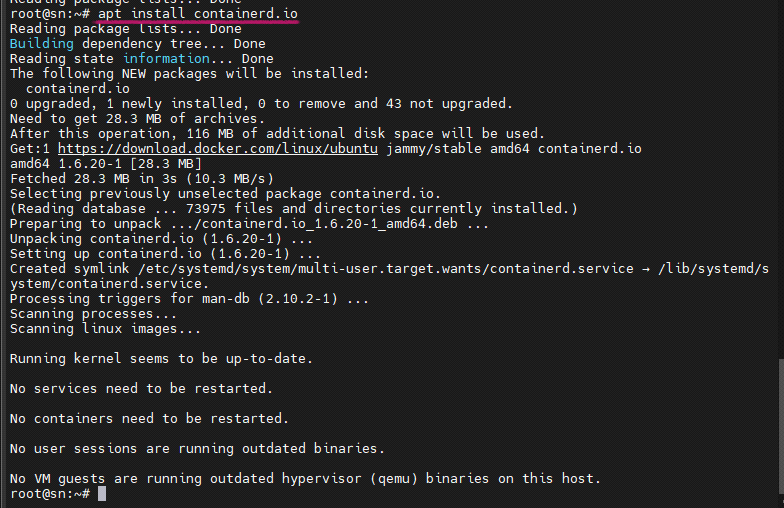
sudo tee /etc/apt/sources.list.d/docker.list > /dev/null



# apt-get update



# apt install containerd.io



After installing containerd.io package we need to configure containerd.

# mkdir -p /etc/containerd

# containerd config default > /etc/containerd/config.toml



Now we need to change systemCgroup false to true.

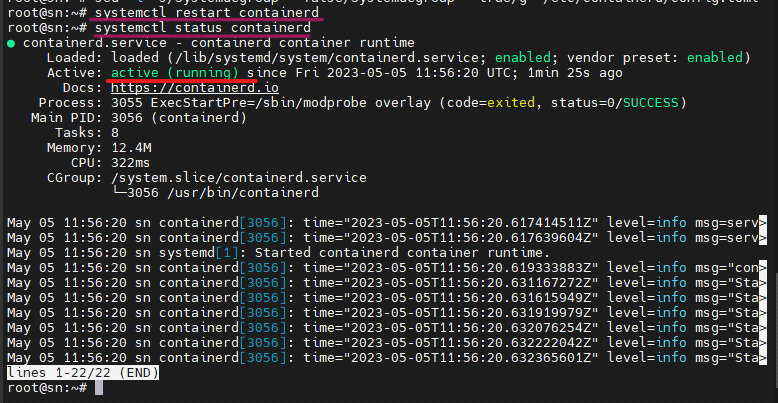
sed -i 's/SystemdCgroup = false/SystemdCgroup = true/g' /etc/containerd/config.toml



restart containerd

# systemctl restart containerd

# systemctl status containerd



To interact with containerd we use docker cli commands before. But here, we are not using Docker. So, now to interact with containerd we are using crictl CLI commands it is available by default.

To execute crictl CLI commands, ensure we create a configuration file as mentioned below

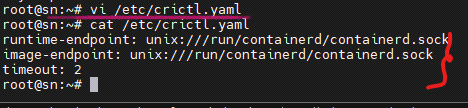
# vi /etc/crictl.yaml

runtime-endpoint: unix:///run/containerd/containerd.sock

image-endpoint: unix:///run/containerd/containerd.sock

timeout: 2

Crictl command will follow the yaml file to communicate with containerd.

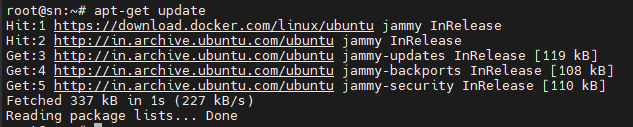


Install kubernetes package:

For install kubernetes follow the official document. In Browser just search like <Kubeadm install> then select the official document it’s URL looks like this

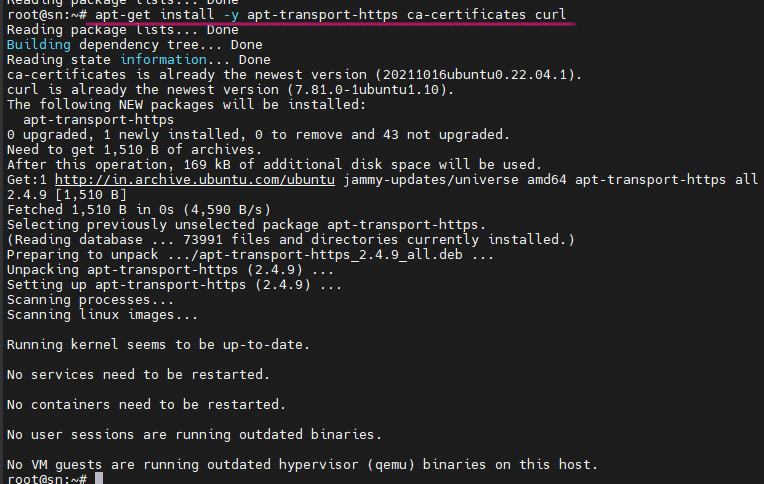
< <https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/install-kubeadm/>>

# apt-get update



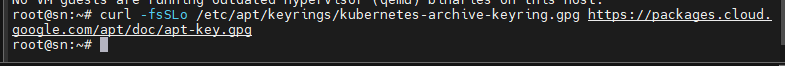
Now install the pre-required packages which are needed for kubernetes

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



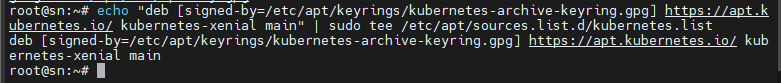
Download the Gpg key for kubernetes repository.

# curl -fsSLo /etc/apt/keyrings/kubernetes-archive-keyring.gpg <https://packages.cloud.google.com/apt/doc/apt-key.gpg>



Create a repository for kubernetes packages to be installed and download.

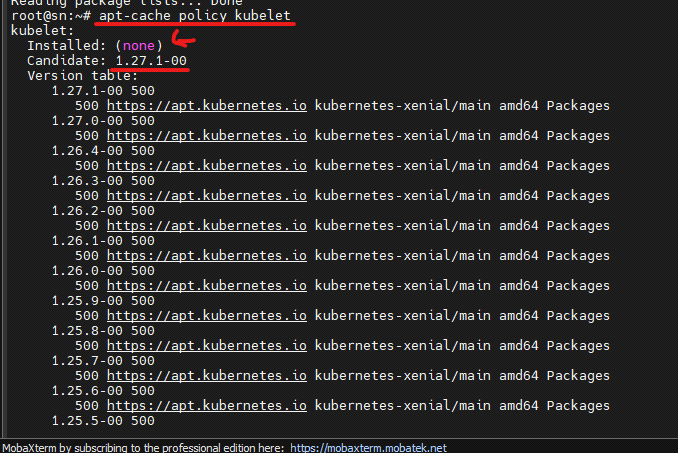
# echo "deb [signed-by=/etc/apt/keyrings/kubernetes-archive-keyring.gpg] <https://apt.kubernetes.io/> kubernetes-xenial main" | sudo tee /etc/apt/sources.list.d/kubernetes.list



# apt update

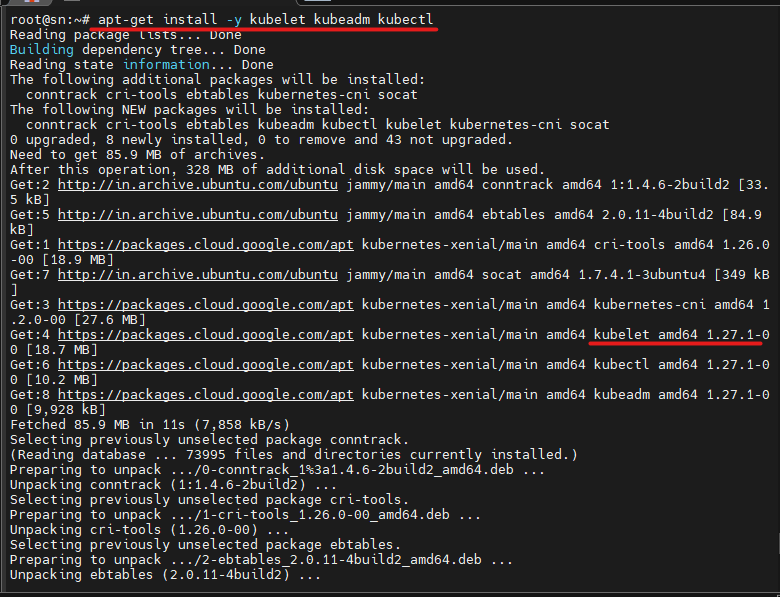
The below command will show the patch version. Candidate: showing the version, Installed: none means now package is not installed

# apt-cache policy kubelet



Now install the kubelet, kubeadm, kubectl packages. And the version of the package installed is 1.27

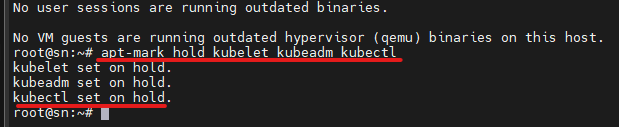
# apt-get install -y kubelet kubeadm kubectl



By using the below command the packages will not upgrade automatically.

# apt-mark hold kubelet kubeadm kubectl

See in the output it is showing that the packages set on hold.

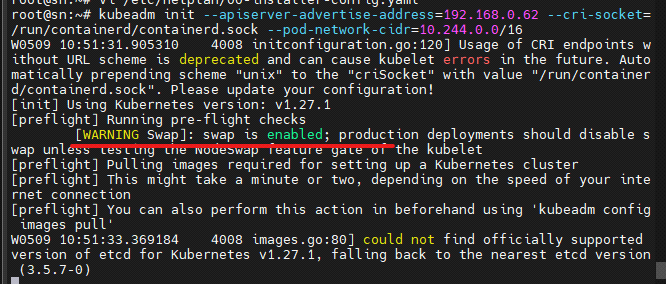


**On Control Plane Node:**

Now run the command to create kubernetes cluster that particular task called as bootstrapping the control plane node.

In the below command change the apiserver-advertise-address with the VM IP address. In CIDR the IP which we mentioned is allotted to the every pod created in this kubernetes cluster.

# kubeadm init --apiserver-advertise-address=192.168.0.62 --cri-socket=/run/containerd/containerd.sock --pod-network-cidr=10.244.0.0/16



In the above page we are getting the warning: swap is enabled. Now we need to disable the swap memory.

**Note:**

**Disabling swap memory:**

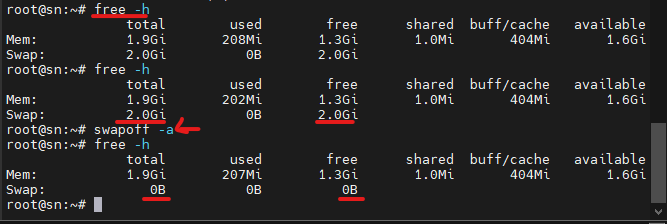
The below command is for checking the memory.

# free -h

To disable swap memory temporarily use the below command

# swapoff –a

See in the below image after executing swapoff –a command the swap memory is showing 0B.



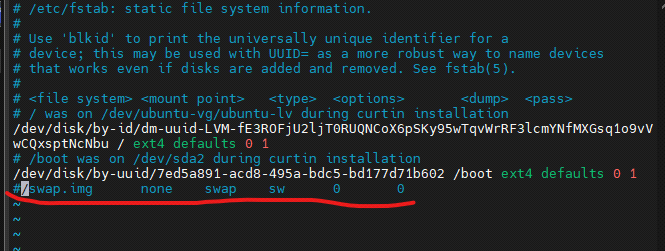
To disable the swap memory permanently go to the fstab file and comment out the swap memory entries line. It should be look like this

/swap.img none swap sw 0 0

Open the file using below command

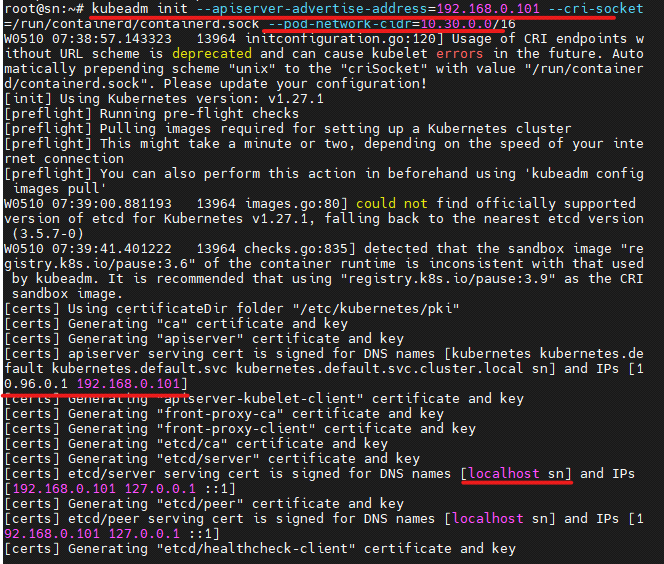
# vi /etc/fstab

After commenting out the entries it should be look like the below image.

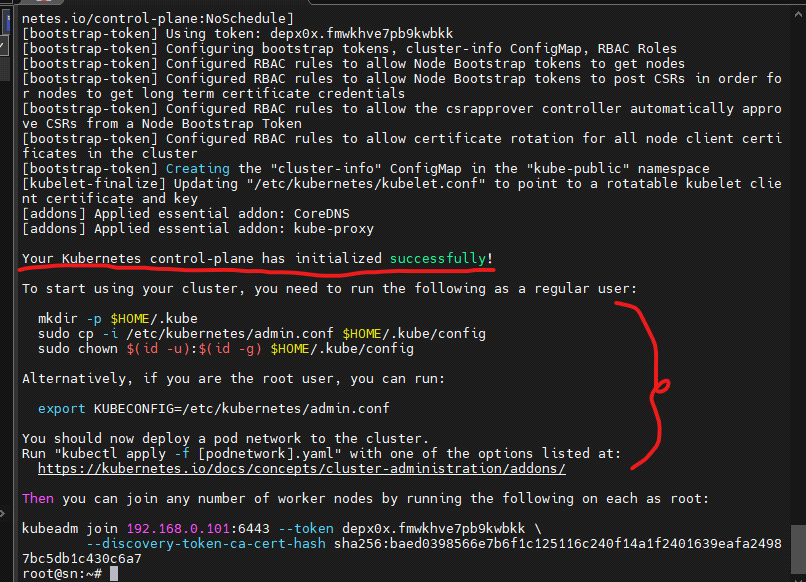


Now rerun the command for creating control plane node

# kubeadm init --apiserver-advertise-address=192.168.0.120 --cri-socket=/run/containerd/containerd.sock --pod-network-cidr=10.30.0.0/16



Now all the folders and files are getting created. After that we will get the message like kubernetes control-plane has initialized successfully. But till now we are not done with kubernetes setup just bootstrapping setup is completed

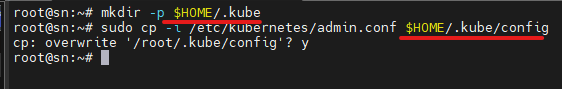


Now create .kube hidden directory. Without having a kube config file we can’t communicate with the kubernetes cluster.

# mkdir -p $HOME/.kube

Now store the admin.conf file under the hidden directory. By default the kube config file store under this path /etc/kubernetes/admin.conf. now copy this file to Home directory $HOME/.kube/config

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



For root users the below command is not needed. Ignore the below command and install calico network.

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

Now install networking interface in the kubernetes cluster. We are using calico CNI.

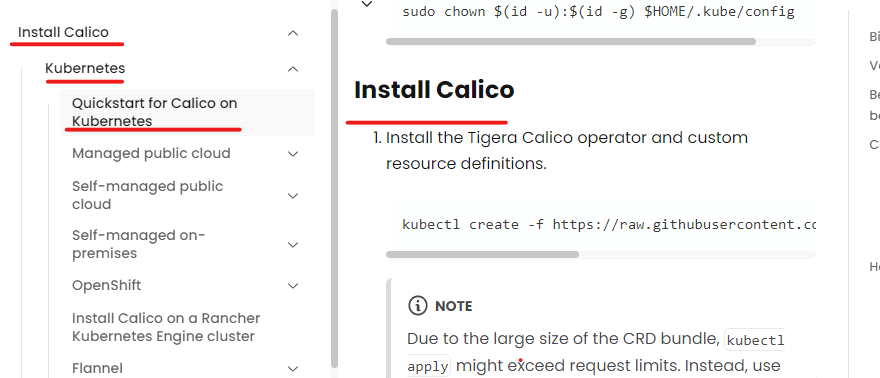
**Steps for Installing calico CNI in kubernetes cluster**

In browser just search like calico kubernetes Then u will get official document of calico

<https://docs.tigera.io/calico/latest/getting-started/kubernetes/>

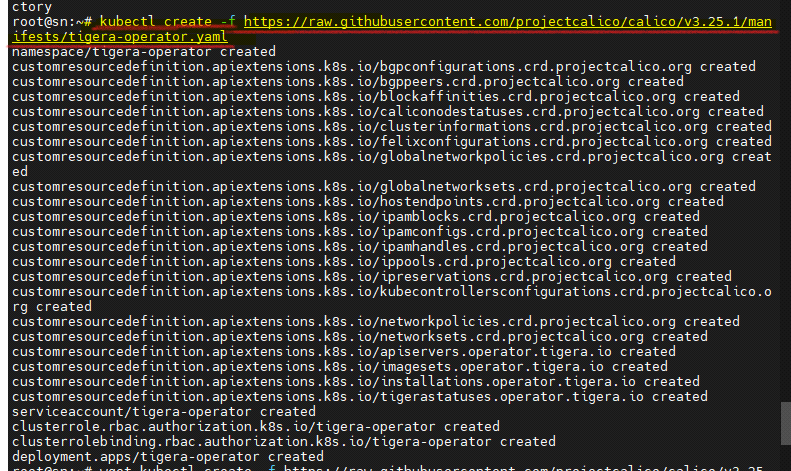
go to Install calico in sidebar – click on kubernetes – Quickstart for calico on Kubernetes

In this page scroll down upto install calico steps.



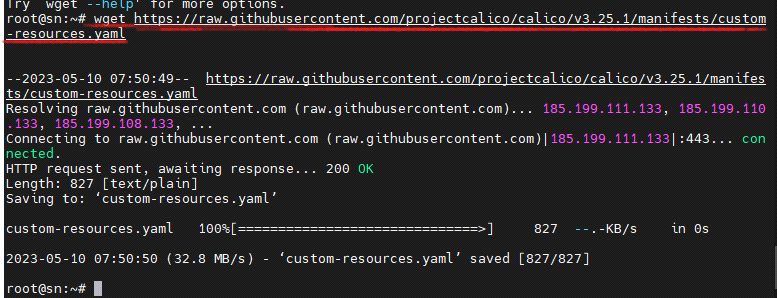
* Now install Tiger calico operator and custom resource definition.

# kubectl create -f <https://raw.githubusercontent.com/projectcalico/calico/v3.25.1/manifests/tigera-operator.yaml>



* Install Calico by creating the necessary custom resource. For more information on configuration options available in this manifest, see [the installation reference](https://docs.tigera.io/calico/latest/reference/installation/api).

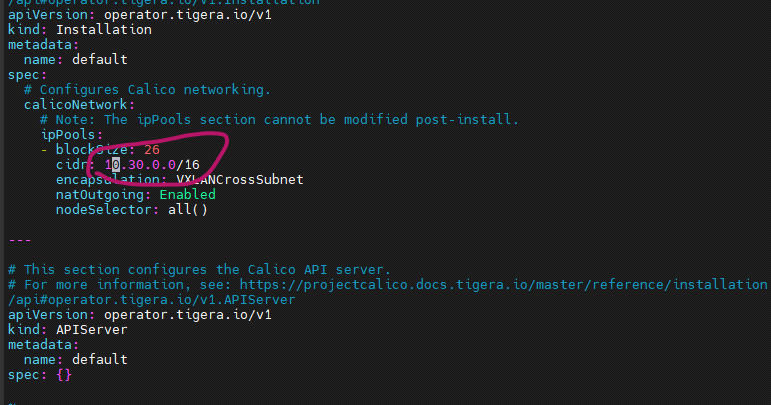
# wget <https://raw.githubusercontent.com/projectcalico/calico/v3.25.1/manifests/custom-resources.yaml>



Now open the custom-resource.yaml file and modify the CIDR. because the CIDR which we used is 10.30.0.0/16

# ls

# vi custom-resources.yaml

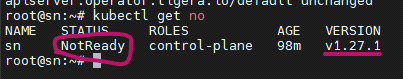


# After update, apply the YAML file

kubectl apply -f custom-resources.yaml

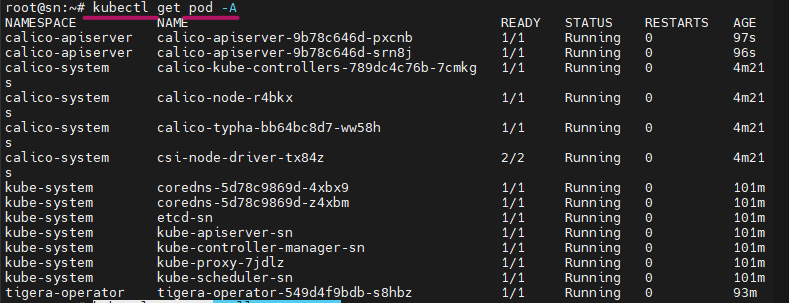


Now check the nodes. It is not ready.



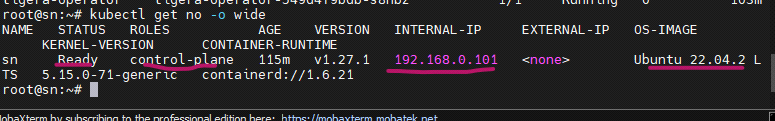
# kubectl get pod –A (or)

# kubectl get po --all-namespaces



To know more information about the node use the below command. Now the node is ready.

# kubectl get no -o wide



Now kubernetes setup in this node is completed.

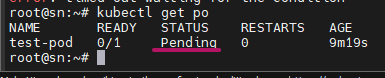
Let’s create a pod in this

# kubectl run test-pod --image=nginx -t –i



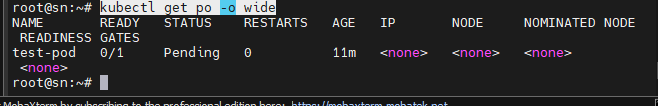
In the above picture we got an error. Let’s see what is the reason behind the error.

# kubectl get po



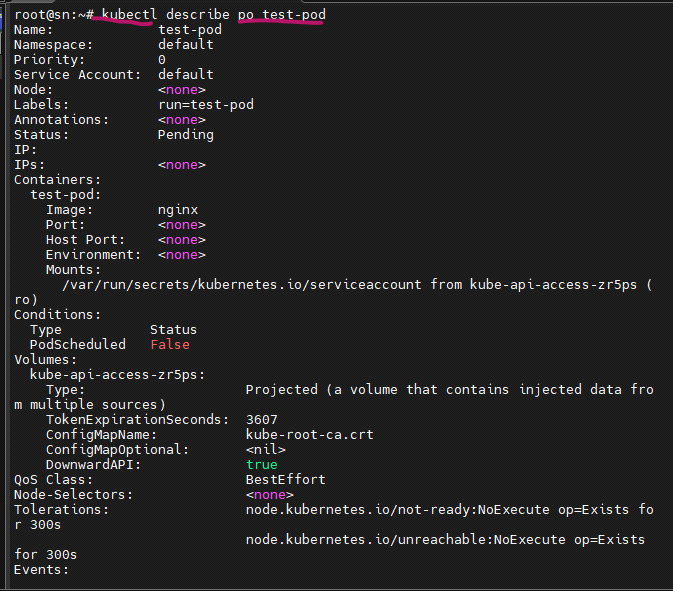
In the above picture the pod is under pending status.

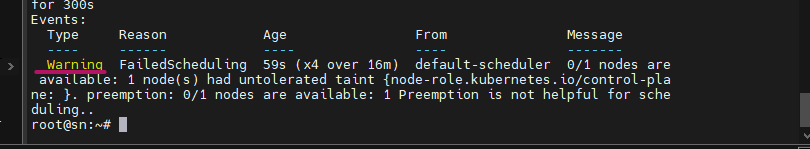
# kubectl get po -o wide



To know the reason behind the pod is pending execute the below command.

# kubectl describe po test-pod



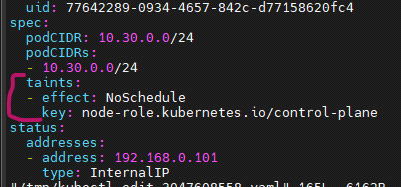


In the above picture we got a warning. Now we have only one node. By default, control plane node allows to launch the pod or application. To overcome this challenge we need to make the control plane node to compute plane node as well for that we need to remove some taint

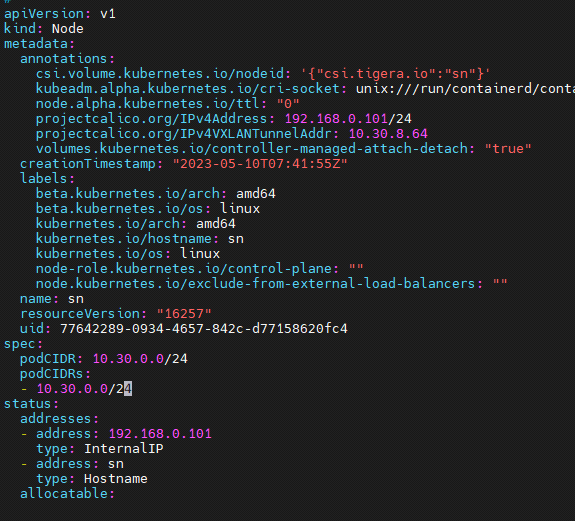
# kubectl get node

# kubectl edit node sn

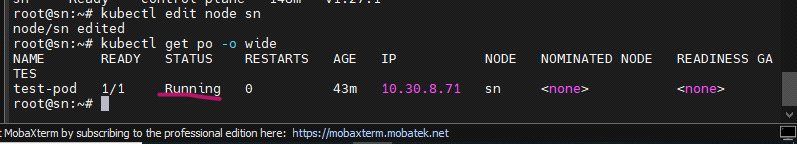
After executing the above command it will open the file in that file the content looks like below. In that content remove taints, effect, key these three lines. The use of these three entries is it will note allow to create a pod in the control plane node.



After removing the entries, the page looks like below.



Now check the status of the pod. The pod status is running and it is allotted with CIDR range which we provide.



Means now we are allowing this node to create pods.

* If the router gets changed of system in which kubernetes is installed at that time we need to do the following steps
* Open the below configuration file and change the IP using the series of the base system IP after another router is attached.

# vi /etc/netplan/00-installer-config.yaml

# This is the network config written by 'subiquity'

network:

ethernets:

enp0s3:

dhcp4: false

addresses: ["192.168.0.62/24"]

routes:

- to: default

via: 192.168.0.1

nameservers:

addresses: [8.8.8.8,8.8.4.4]

version: 2

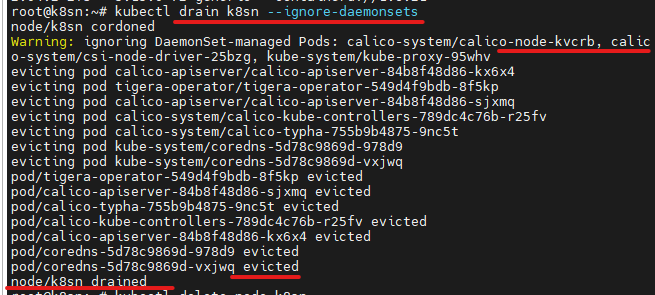
# netplan apply

* Now follow the as it is installation steps for installing kubeadm and its dependencies.

**--> If we delete the node in kubernetes singlenode set up follow the below steps.**

1. To remove the node first we need to drain the control plane node use the below step to drain the node.Before removing the control plane node, you should drain it to ensure that the workload running on the node is rescheduled to other nodes in the cluster.

# kubectl drain <control-plane-node-name> --ignore-daemonsets



Replace <control-plane-node-name> with the name of the control plane node you want to remove.

The --ignore-daemonsets flag allows the drain operation to skip DaemonSet pods, ensuring that essential system services continue to run on other nodes.

2. Delete the control plane node: After the control plane node is drained, you can delete it from the cluster using the following command:

# kubectl delete node <node-name>

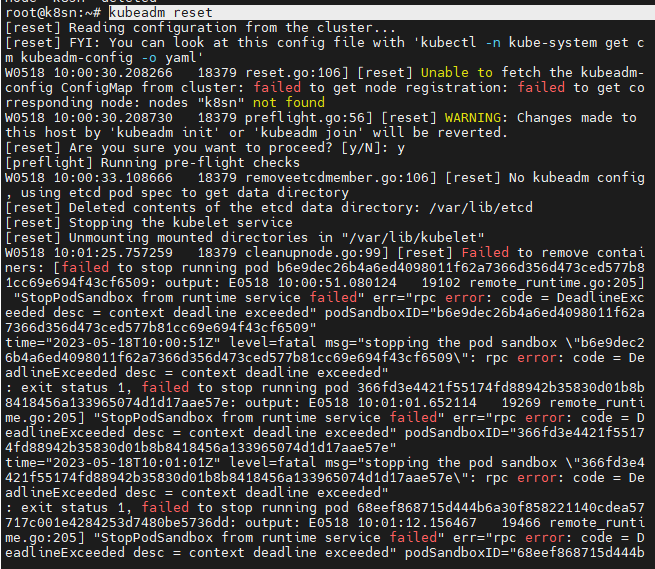


Replace <control-plane-node-name> with the name of the control plane node.

Deleting the node removes it from the cluster's control plane and stops it from participating in the cluster's management

If you installed Kubernetes using a package manager like kubeadm, you may need to run kubeadm reset on the control plane node to remove the Kubernetes components.

# kubeadm reset



The reset process does not reset or clean up iptables rules or IPVS tables.

If you wish to reset iptables, you must do so manually by using the "iptables" c ommand.

If your cluster was setup to utilize IPVS, run ipvsadm --clear (or similar)

to reset your system's IPVS tables.

1. Ensure that you have root or administrative privileges, as resetting iptables requires administrative access.

2. Save a backup of your current iptables configuration (optional but recommended). You can use the following command to save a copy of the current configuration to a file:

# sudo iptables-save > iptables\_backup.txt



3. Flush all the existing iptables rules. This will remove all the rules from the tables and chains:

# sudo iptables -F

4. Delete all the custom user-defined chains:

# sudo iptables -X



5. Set the default policies for the INPUT, OUTPUT, and FORWARD chains to ACCEPT:

# sudo iptables -P INPUT ACCEPT

# sudo iptables -P OUTPUT ACCEPT

# sudo iptables -P FORWARD ACCEPT



6. If you have IPv6 rules, you can also reset them using the ip6tables command. Use the following commands to flush IPv6 rules and set default policies:

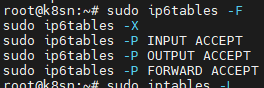
# sudo ip6tables -F

# sudo ip6tables -X

# sudo ip6tables -P INPUT ACCEPT

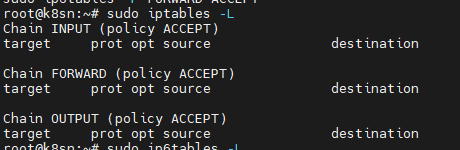
# sudo ip6tables -P OUTPUT ACCEPT

# sudo ip6tables -P FORWARD ACCEPT



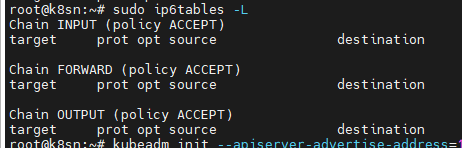
7. Verify that iptables is reset by listing the rules. Running the following command should show no rules or chains:

# sudo iptables -L



Similarly, you can verify the IPv6 rules using the ip6tables command:

# sudo ip6tables -L



By following these steps, you should have successfully reset iptables to the default configuration. It's important to exercise caution when modifying or resetting firewall rules to avoid inadvertently blocking or allowing unintended network traffic.

The reset process does not clean your kubeconfig files and you must remove them manually.

# rm -rf $HOME/.kube/config



The command `kubeadm init` is used to initialize a Kubernetes control plane on a master node. Let's break down the options used in the command:

- `--apiserver-advertise-address=192.168.0.120`: This option specifies the IP address that the Kubernetes API server will advertise. It determines which IP address clients should use to connect to the Kubernetes API server. In this case, it is set to `192.168.0.120`.

- `--cri-socket=/run/containerd/containerd.sock`: This option specifies the path to the container runtime interface (CRI) socket file. It defines the communication endpoint between Kubernetes and the container runtime. Here, it is set to `/run/containerd/containerd.sock`, indicating that the container runtime used is Containerd.

- `--pod-network-cidr=10.30.0.0/16`: This option specifies the network range to be used for pods in the cluster. It defines the IP address range from which pod IP addresses will be assigned. Here, it is set to `10.30.0.0/16`, indicating that pods will have IP addresses within the `10.30.0.0` subnet.

By running this `kubeadm init` command with the provided options, you will initialize a Kubernetes control plane on the master node, configure the API server to advertise the specified IP address, use Containerd as the container runtime, and assign pod IP addresses from the `10.30.0.0/16` subnet. Note that this command assumes you have installed Kubernetes and its dependencies properly before running `kubeadm init`.

# kubeadm init --apiserver-advertise-address=192.168.0.120 --cri-socket=/run/containerd/containerd.sock --pod-network-cidr=10.30.0.0/16

from here to continue the ---(**On Control Plane Node:)**

root@k8sn:~# mkdir -p $HOME/.kube

root@k8sn:~# cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

root@k8sn:~# kubectl create -f [**https://raw.githubusercontent.com/projectcalico/calico/v3.25.1/manifests/tigera-operator.yaml**](https://raw.githubusercontent.com/projectcalico/calico/v3.25.1/manifests/tigera-operator.yaml)

root@k8sn:~# **wget** [**https://raw.githubusercontent.com/projectcalico/calico/v3.25.1/manifests/custom-resources.yaml**](https://raw.githubusercontent.com/projectcalico/calico/v3.25.1/manifests/custom-resources.yaml)

root@k8sn:~# vi custom-resources.yaml

root@k8sn:~# kubectl apply -f custom-resources.yaml

root@k8sn:~# kubectl get po --all-namespaces

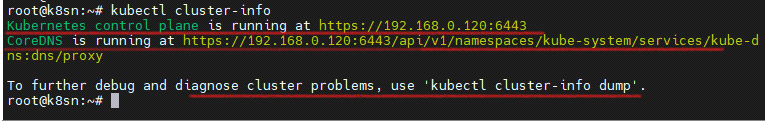
# kubectl get no -o wide

# kubectl edit node sn

remove taints, effect, key lines from that file

By running kubectl cluster-info, you can quickly obtain essential information about the Kubernetes cluster and its core components. This command is useful for verifying the connectivity and availability of the control plane and other critical services in the cluster.

# kubectl cluster-info



The output will provide the URLs of the Kubernetes control plane and the KubeDNS service.

<master-url>: This is the URL of the Kubernetes control plane, which includes the API server. It is the primary endpoint for interacting with the cluster.

<kube-dns-url>: This is the URL of the KubeDNS service, which provides DNS resolution for the cluster. It allows you to resolve the names of Kubernetes services and pods within the cluster.

Additionally, the output may include information about other core services running in the cluster, such as the Kubernetes dashboard or other custom services you may have deployed.

**References:**

* GitHub URL: <https://github.com/swapnachatla/kubernetes_latest_manifest/blob/main/Kubernetes/01-kubernetes-architecture-Installation/03-k8s-setup-kubeadm-containerd.md>
* Docker: <https://docs.docker.com/engine/install/ubuntu/>
* kubernetes: <https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/install-kubeadm/>
* calico: <https://docs.tigera.io/calico/latest/getting-started/kubernetes/>
* <https://www.ibm.com/docs/da/control-desk/7.6.1.x?topic=kubernetes-installing-kubeadm-kubelet-kubectl>