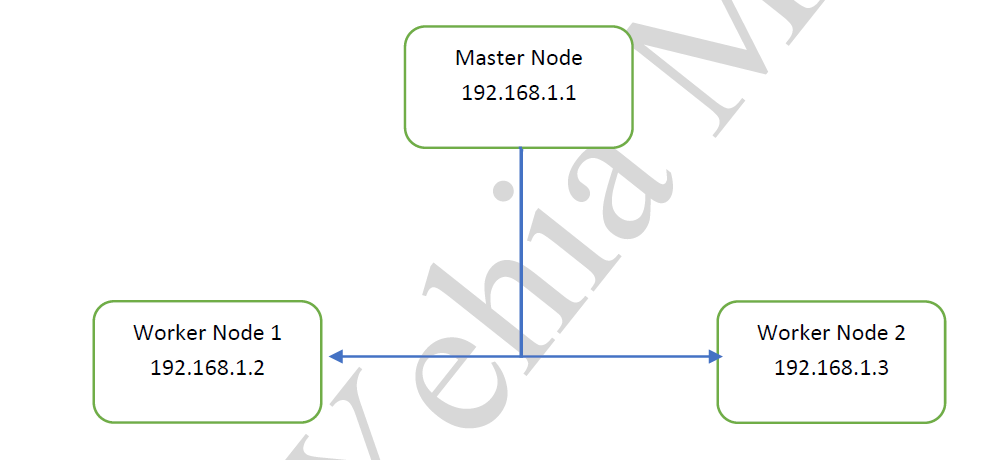
* **Install Kubernates(k8s) from scratch: -**
* ➢ Requirements: -
* 2 ubuntu machines – OS: Ubuntu 20.04.
* 2- 2 GB or more of RAM per machine.
* 3- 2 CPUs or more.
* 4- Full network connectivity between all machines in the cluster, (you can disable firewall just testing environment).
* 5- Unique hostname, MAC address, and product\_uuid for every node.
* 6- Root user.

Architecture: -



**Installation Steps: -**

**a) Command run through ( Master node – Worker Node 1 – Worker Node 2 ) :-**

**1- You MUST disable swap $$ sudo swapoff -a**

**Then disable swap as a below $$ nano /etc/fstab**

**//** Swap is a Linux kernel parameter that controls the relative weight given to swapping out runtime memory, as opposed to dropping pages from the system page cache. Swap can be set to values between 0 and 100 inclusive. A low value causes the kernel to avoid swapping, a higher value causes the kernel to try to use swap space. The default value is 60, and for most desktop systems, setting it to 100 may affect the overall performance, whereas setting it lower (even 0) may decrease response latency.

1. **- Set up the IPV4 bridge on all nodes ( run all below at one time).**

**Setting bridge network based in IPv4 address which would be responsible to run on everynode.**

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

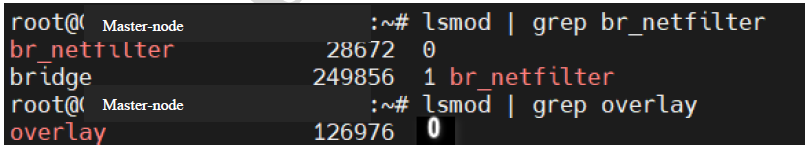
# Apply sysctl params without reboot

sudo sysctl --system

**3- Verify that the br\_netfilter, overlay modules are loaded by running the following commands:**

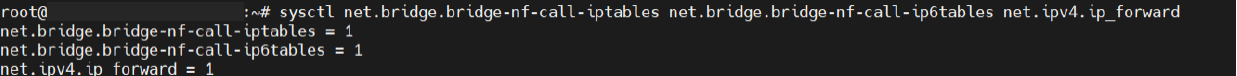
**$$lsmod | grep br\_netfilter**

**$$lsmod | grep overlay**

****

**Verify that the net.bridge.bridge-nf-call-iptables, net.bridge.bridge-nf-call-ip6tables, and net.ipv4.ip\_forward system variables are set to 1 in your sysctl config by running the following command:**

**$$sysctl net.bridge.bridge-nf-call-iptables net.bridge.bridge-nf-call-ip6tables net.ipv4.ip\_forward**



**Disable firewall (to make sure 3 machines can connect together): -**

**$$ufw status**



**Install Docker:**

**Run the following command to uninstall all conflicting packages:**

**for pkg in docker.io docker-doc docker-compose docker-compose-v2 podman-docker containerd runc; do sudo apt-get remove $pkg; done**

# Add Docker's official GPG key:

sudo apt-get update

sudo apt-get install ca-certificates curl

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

sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/docker.asc

sudo chmod a+r /etc/apt/keyrings/docker.asc

# Add the repository to Apt sources:

echo \

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

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

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

sudo apt-get update

Install Docker packages:

sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin

Verify that the Docker Engine installation is successful by running the hello-world image.

sudo docker run hello-world

Install Docker CRI:

Download the package

$ wget https://github.com/Mirantis/cri-dockerd/releases/download/v0.2.0/cri-dockerd-v0.2.0-linux-amd64.tar.gz

Unzip the package

$ tar xvf cri-dockerd-v0.2.0-linux-amd64.tar.gz

If you’re on Linux, move the cri-dockerd binary to your usr/local/bin directory:

$ sudo mv ./cri-dockerd /usr/local/bin/

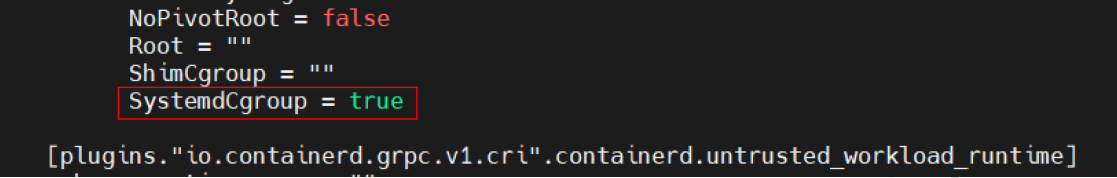
Check to see if it is successfully installed:

$ cri-dockerd --help

**Set up Cgroup as a true (systemd)**

**Cgroup stands for “control group”**

**$$ nano /etc/containerd/config.toml**



**Then restart service to read new configuration.**

$$ systemctl restart containerd.service

**- Let’s install kubelet, kubeadm, and kubectl to create a Kubernetes cluster. They play an important role in managing a Kubernetes cluster.**

**Update the apt package index and install packages needed to use the Kubernetes apt repository:**

**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 public signing key for the Kubernetes package repositories. The same signing key is used for all repositories so you can disregard the version in the URL:

*# 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:/v1.30/deb/Release.key | sudo gpg --dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg

Add the appropriate Kubernetes apt repository. Please note that this repository have packages only for Kubernetes 1.30; for other Kubernetes minor versions, you need to change the Kubernetes minor version in the URL to match your desired minor version (you should also check that you are reading the documentation for the version of Kubernetes that you plan to install).

*# 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:/v1.30/deb/ /' | sudo tee /etc/apt/sources.list.d/kubernetes.list

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

sudo apt-get update

sudo apt-get install -y kubelet kubeadm kubectl

sudo apt-mark hold kubelet kubeadm kubectl

Enable the kubelet service before running kubeadm:

**sudo systemctl enable --now kubelet**

1. **Command run through ( Master node only – 192.168.1.1 ) :-**
2. **- initialize your master node. The --pod-network-cidr flag is setting the IP address range for the pod network (Must use range 10.244.0.0 for flannel network package – Next step).**
3. When running as Regular user/Normal user.
4. **192.168.1.1 is the IP of master node.**

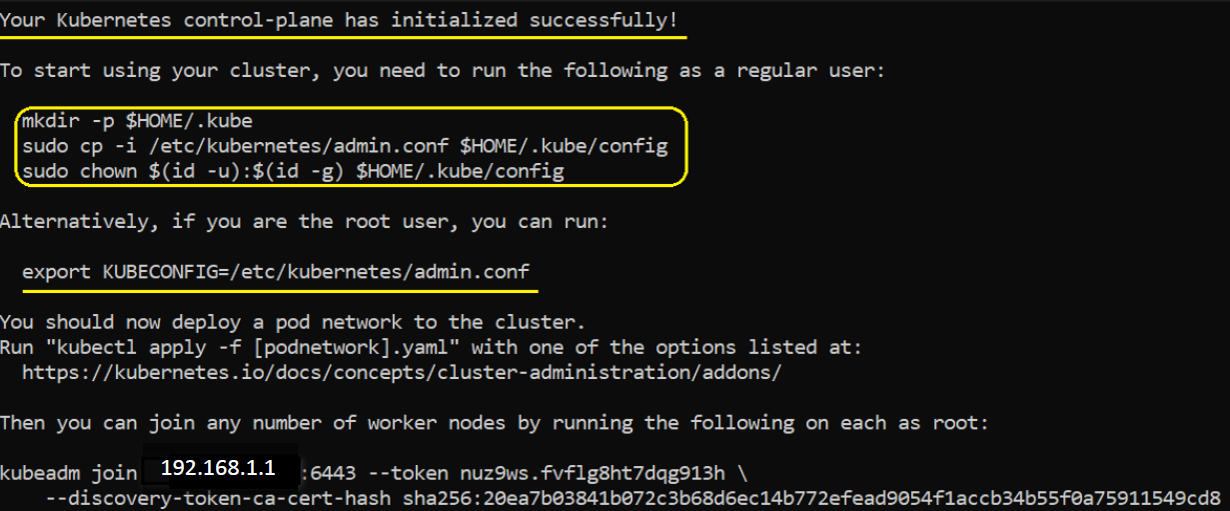
**$$kubeadm init --pod-network-cidr=10.244.0.0/16 --apiserver-advertise-address=192.168.1.1**

mkdir -p $HOME/.kube

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

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

The result of initialize is below:



When working as root user:

export KUBECONFIG=/etc/kubernetes/admin.conf

echo 'export KUBECONFIG=/etc/kubernetes/admin.conf' >> $HOME/.bashrc

Delete the config.toml file and restart containerd then try the init command like below:

$ rm /etc/containerd/config.toml

$ systemctl restart containerd

$ kubeadm init

1. **c) Join workers node to the cluster through (WorkerNode1 192.168.1.2 - WorkerNode2 192.168.1.3) in upper photo.**
2. **- From machine WorkerNode1 run**

**$$ kubeadm join 192.168.1.1:6443 --token d2tgh8.f9q3vjf1i5t1uneu \**

**--discovery-token-ca-cert-hash sha256:59b6ac2294eb69ccf84743fc2b9ea5113b64bbe5ea0d5372938b1e81468c47da**

**Copy certificate from MasterNode path /root/.kube/config**

**Then paste it in WorkerNode1 in same path /root/.kube/config**

**- From machine WorkerNode2 run**

**$$ kubeadm join 192.168.1.1:6443 --token d2tgh8.f9q3vjf1i5t1uneu \**

**--discovery-token-ca-cert-hash sha256:59b6ac2294eb69ccf84743fc2b9ea5113b64bbe5ea0d5372938b1e81468c47da**

**Copy certificate from MasterNode path /root/.kube/config**

**Then paste it in WorkerNode2 in same path /root/.kube/config**

1. **d) Command run through ( Master node only 192.168.1.1 ) :-**
2. **- Install flannel package for network solution.**

**$$ sudo kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml**

1. Finally,

**$$ kubectl get nodes**

