**Kubernetes Installation**

**Prerequisites**

* Master node- 2GB Ram and 2 CPUs
* Worker node - 2GB Ram and 2 CPUs
* Sudo privilege

#### Step 1: Install Kubernetes

You must run the commands in both nodes to install Kubernetes.

1. You will start by installing the **apt-transport-https** package which enables working with **http** and **https** in Ubuntu’s repositories.

**sudo apt install apt-transport-https curl**

1. add the Kubernetes signing key to both nodes by executing the command  
   **curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add**
2. we add the Kubernetes repository as a package source on both nodes using the following command

**echo "deb https://apt.kubernetes.io/ kubernetes-xenial main" >> ~/kubernetes.list**

**sudo mv ~/kubernetes.list /etc/apt/sources.list.d**

1. After that, update the nodes

**sudo apt update**

1. Installing the various tools that make up Kubernetes: kubeadm, kubelet, kubectl, and kubernetes-cni.

**sudo apt-get install -y kubelet kubeadm kubectl kubernetes-cni**

#### Step 2: Disabling Swap Memory

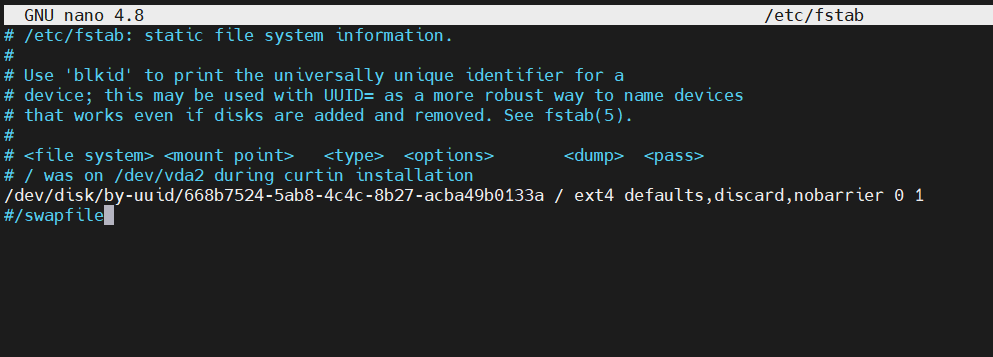
Kubernetes fails to function in a system that is using swap memory. Hence, it must be disabled in the master node and all worker nodes.

**sudo swapoff -a**

This command disables swap memory until the system is rebooted. We have to ensure that it remains off even after reboots. This has to be done on the master and all worker nodes. We can do this by editing the **fstab** file and commenting out the **/swapfile** line with a **#**. Open the file with the nano text editor by entering the following command.

**sudo nano /etc/fstab**

Inside the file, comment out the **swapfil**e line as shown in the screenshot below.



If you do not see the **swapfile line**, just ignore it. Save and close the file when you are done editing. Follow the same process for both nodes. Now, swap memory settings will remain off, even after your server reboots.

#### Step 3: Setting Unique Hostnames

Your nodes must have unique hostnames for easier identification.

For Master - **sudo hostnamectl set-hostname kubernetes-master**

For Worker node - **sudo hostnamectl set-hostname kubernetes-worker**

#### Step 4: Letting Iptables See Bridged Traffic

For the master and worker nodes to correctly see bridged traffic, you should ensure **net.bridge.bridge-nf-call-iptables** is set to **1** in your **confi**g.

**lsmod | grep br\_netfilter**

**sudo modprobe br\_netfilter** - Optional

**sudo sysctl net.bridge.bridge-nf-call-iptables=1**

#### Step 5: Changing Docker Cgroup Driver

#### By default, Docker installs with cgroupfs as the cgroup driver. Kubernetes recommends that Docker should run with systemd as the driver.

Install docker:

**sudo apt-get install docker.io**

**sudo mkdir /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**

**sudo systemctl enable docker**

**sudo systemctl daemon-reload**

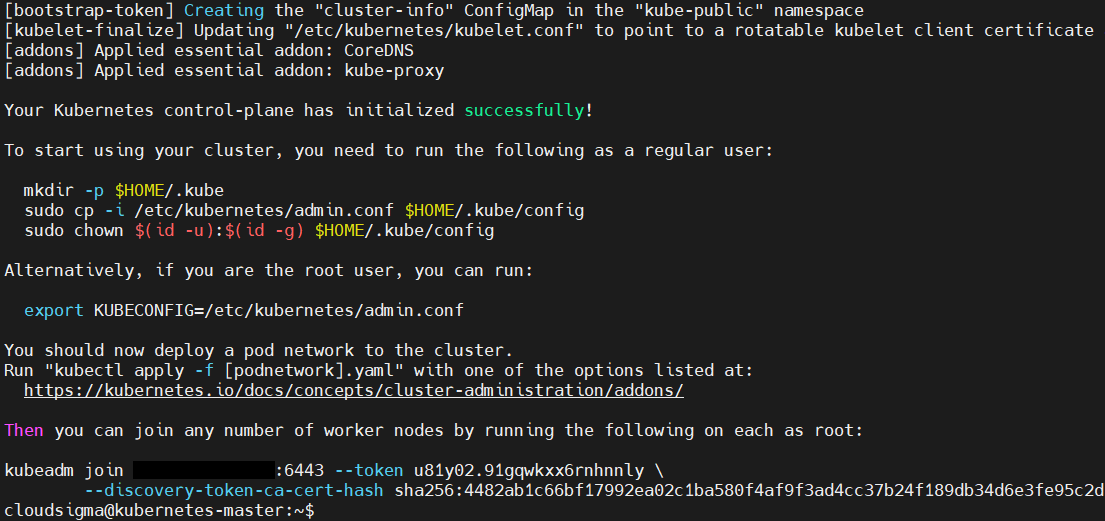
**sudo systemctl restart docker**

#### Step 6: Initializing the Kubernetes Master Node

Execute the following command to initialize the kubernetes-master

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

The screenshot below shows that the initialization was successful. We have also added a flag to specify the pod network with the IP **10.244.0.0**, It’s the default IP that the **[kube-flannel](https://blog.laputa.io/kubernetes-flannel-networking-6a1cb1f8ec7c" \t "_blank)** [uses](https://blog.laputa.io/kubernetes-flannel-networking-6a1cb1f8ec7c" \t "_blank).



In the output, you can see the **kubeadm** join command and a unique token that you will run on the worker node and all other worker nodes that you want to join onto this cluster. Next, **copy-paste** this command as you will use it later in the worker node.

In the output, Kubernetes also displays some additional commands that you should run as a regular user on the master node before you start to use the cluster. Let’s run these commands:

**mkdir -p $HOME/.kube**

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

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

The token generated is **valid only for 24hours**. In case the 24 hours exceed we need to generate the new token using the command “**sudo kubeadm token create**” .The current token can be view from master using the below command.

**sudo kubeadm token list**

The CA cert hash is used for the node to join in secure manner. This also can be view from the below command. Combining this token and hash we can join the nodes.

**openssl x509 -pubkey -in /etc/kubernetes/pki/ca.crt | openssl rsa -pubin -outform der 2>/dev/null | openssl dgst -sha256 -hex | sed 's/^.\* //'**

#### Step 7: Deploying a Pod Network

A pod network facilitates communication between servers and it’s necessary for the proper functioning of the Kubernetes cluster. You can read more about [Kubernetes Cluster Networking](https://kubernetes.io/docs/concepts/cluster-administration/networking/) from the official docs. We will be using the [**Flannel**](https://kubernetes.io/docs/concepts/cluster-administration/networking/#flannel)**pod network** for this tutorial. Flannel is a simple overlay network that satisfies the Kubernetes requirements.

Before we deploy the pod network, we need to check on the firewall status. If you have enabled the firewall after following step 5 of the [tutorial on setting up your Ubuntu server](https://www.cloudsigma.com/how-to-set-up-your-ubuntu-18-04-server/), you must first add a firewall rule to create exceptions for port 6443 (the default port for Kubernetes). Run the following [ufw](https://wiki.ubuntu.com/UncomplicatedFirewall" \t "_blank) commands on both **master and worker** nodes:

**sudo ufw allow 6443**

**sudo ufw allow 6443/tcp**

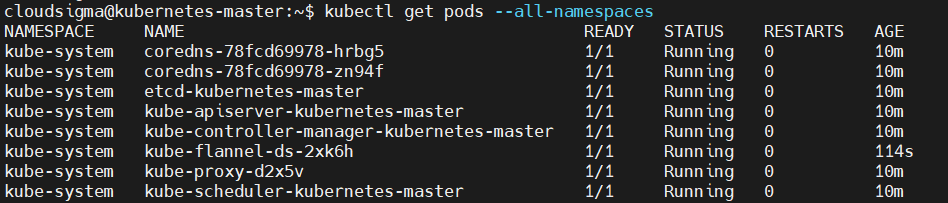
After that, you can run the following two commands to deploy the pod network on the **master node**:

**kubectl apply -f** [**https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml**](https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml)

**kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/k8s-manifests/kube-flannel-rbac.yml**

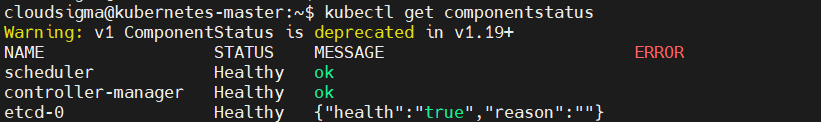
Run the following command to confirm that everything is fired up:

**kubectl get pods --all-namespaces**



You can also view the health of the components using the get component status command:

**kubectl get componentstatus**



If you see the **unhealthy status**, modify the following files and delete the line at (spec->containers->command) containing this phrase - --port=0 :

**sudo nano /etc/kubernetes/manifests/kube-scheduler.yaml**

Do the same for this file:

**sudo nano /etc/kubernetes/manifests/kube-controller-manager.yaml**

Finally, restart the Kubernetes service:

**sudo systemctl restart kubelet.service**

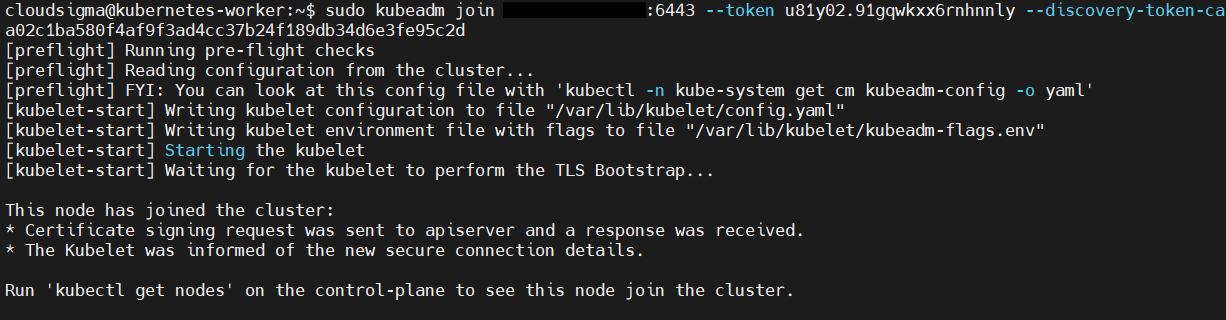
#### Step 8: Joining Worker Nodes to the Kubernetes Cluster

With the kubernetes-master node up and the pod network ready, we can join our **worker nodes** to the cluster. In this tutorial, we only have one worker node, so we will be working with that. If you have more worker nodes, you can always follow the same steps as we will explain below to join the cluster.

First, log into your worker node on a separate terminal session. You will use your kubeadm join command that was shown in your terminal when we initialized the master node in Step 6. Execute the command:

EX:-

**sudo kubeadm join 127.0.0.188:6443 --token u81y02.91gqwkxx6rnhnnly --discovery-token-ca-cert-hash sha256:4482ab1c66bf17992ea02c1ba580f4af9f3ad4cc37b24f189db34d6e3fe95c2d**

You should see similar output like the screenshot below when it completes joining the cluster:

Once the joining process completes, switch the master node terminal and execute the following command to confirm that your worker node has joined the cluster:

**kubectl get nodes**

