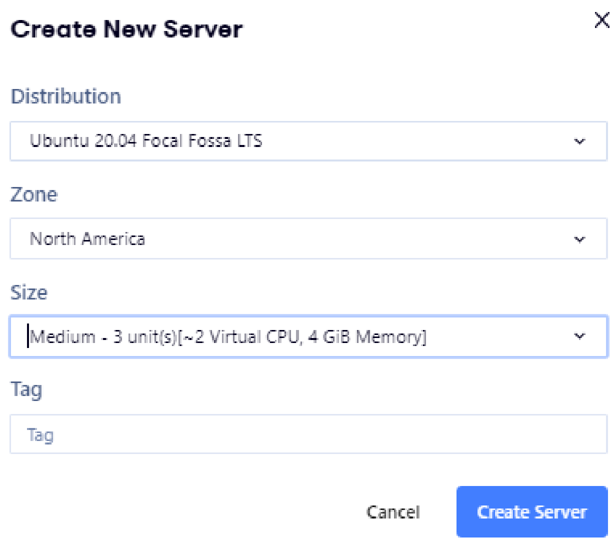
# Building a Kubernetes Cluster

In the cloud playground we will be going to the "Cloud Servers" tab. Here you will create 3 servers, 1 as which will act as the control plane server. And the remaining 2 will server as worker nodes. You will use Ubuntu 20.04 Focal Fossa LTS with a medium size for all 3.

***Figure 1-1***



Create New Server

Distribution: Ubuntu 20.04 Focal Fossa LTS

Zone: North America

Size: Medium - 3 unit(s)[~2 Virtual CPU, 4 GiB Memory]

Tag: Tag

Once all 3 servers are created, it would be best to go ahead and open all 3 up and change the password. Once that is complete, we will want to change the hostname of the sever so we can identify what is what when configuring.

## Setting Hostnames

1. We will start with the first server. This will be our control plane sever. The command you will use is

**sudo hostnamectl set-hostname k8s-control**

2. Enter the password and move on to the other two servers. These will be the worker nodes. Enter the same command but the end will be different.

**sudo hostnamectl set-hostname k8s-worker1**

    a. On the last sever just change the one to the number two

***When that is completed on all 3 servers. You can type exit and sign back in that way you can tell what server you're in when in the prompt.**

## Host File with mappings

Now that the hostnames are set, go back to the control plane server. Here, we will setup a host file with mappings for all of the servers so they can talk to each other using the hostnames.

3. To go into the host file, enter the following command below

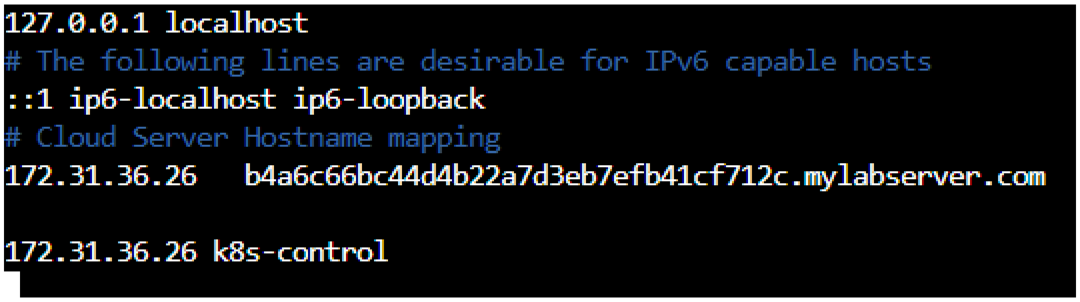
**Sudo vi /etc/hosts**

Once in there, you will see things already in there. We will add new entries at the end.

4. We will enter the private IP address of the control plane server, so the one we are currently on, so it is already listed. So, we can copy the IP address and paste it.

***To make edits to the host file you must press the "insert" key to enter things**

***Figure 1-2***

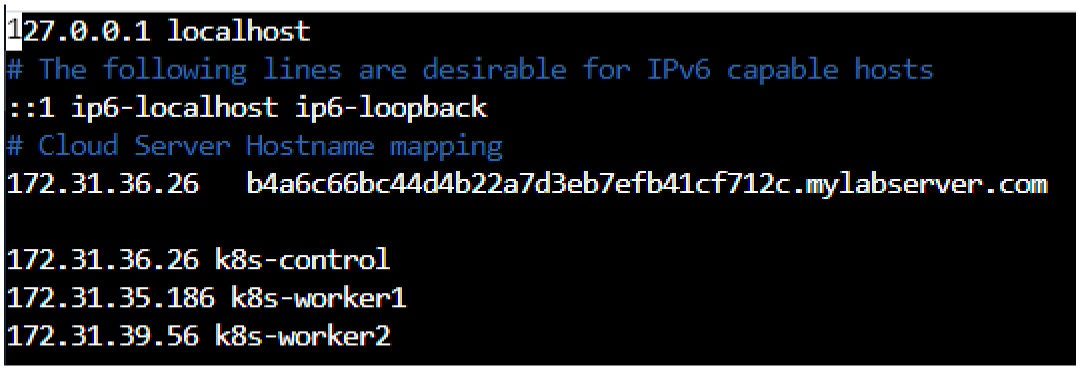


```
127.0.0.1 localhost
# The following lines are desirable for IPv6 capable hosts
::1 ip6-localhost ip6-loopback
# Cloud Server Hostname mapping
172.31.36.26   b4a6c66bc44d4b22a7d3eb7efb41cf712c.mylabserver.com

172.31.36.26 k8s-control
```

So, if anything tries to reach out to the hostname (Which in this case "k8s-control") will get the appropriate IP. It is important to use the private IP, because the public IP can change if the server restarts when using cloud playground.

5. Next step will be to add in the workers1-2 servers in with the private IP and hostname.
    a. To save type :wq (which means write and quit) :qa means quit without saving

***Figure 1-3***



```
127.0.0.1 localhost
# The following lines are desirable for IPv6 capable hosts
::1 ip6-localhost ip6-loopback
# Cloud Server Hostname mapping
172.31.36.26   b4a6c66bc44d4b22a7d3eb7efb41cf712c.mylabserver.com

172.31.36.26 k8s-control
172.31.35.186 k8s-worker1
172.31.39.56 k8s-worker2
```

6. You would do the same for the worker1/2 server. Just a simple copy and paste in the host file

So once all is complete. The next step is to install container D. Which again, this next step would need to be completed on all 3 servers.

7. Down below is how you enable some kernel modules.

```
cat << EOF | sudo tee /etc/modules-load.d/container.conf
overlay
br_netfilter
EOF
```

***REMEMBER case sensitive**

Above the ***/etc/modules-load.d/container.conf***… This will load the kernel module on startup. We will also add ***overlay*** and ***br_netfilter*** to that file. So everytime the server starts those 2 modules will be enabled.

8. Once you input that and type in the password. We now have to input the follow commands.

```
sudo modprobe overlay
sudo modprobe br_netfilter
```

These commands above will immediately enable these modules. So, adding them to the file, will be like a “make sure” it is enabled without having to restart the server.

## System level configurations

Down below are just networking related configurations that Kubernetes is going to need for its networking to work as expected. So we are adding this to a config file in “sysctl.d”. Of course, these configurations will be loaded on startup.

```
cat <<EOF | sudo tee /etc/sysctl.d/99-kubernetes-cri.conf
net.bridge.bridge-nf-call-iptables =1
net.ipv4.ip_forward = 1
net.bridge.bridge-nf-call-ip6tables =1
EOF
```

Once that is complete, we need to apply them immediately

9. The command to use to apply is

```
sudo sysctl --system
```

**^Double - -**

## Installing Container D

10. One all of that is installed. We are ready to install container D.

**sudo apt-get update && sudo apt-get install -y containerd**

Now we are going to setup a container D config file.

11. So, to create a container D config file we have to create a directory for it.

**sudo mkdir -p /etc/containerd**

***This is just for where your config file to live in**

We now want to generate this file using the containerd config default command. This will generate its contents of the config file.

12. Generate contents of the config file

**sudo containerd config default | sudo tee /etc/containerd/config.toml**

With the **containerd config default** command, that will generate the contents of the configuration file. And we want to pipe it to add an additional command to save the contents in the **containerd/config.toml** file.

13. Now we want to verify if the containerd is actually using that configuration we will restart it

**sudo systemctl restart containerd**

## Installing Kubernetes Packages

To install packages, Kubernetes does require to have swap to be disabled

14. To turn off/disable swap.

**sudo swapoff -a**

15. So now we can install packages

**sudo apt-get update && sudo apt-get install -y apt-transport-https curl**

These are the packages that are listed in the Kubernetes documentation part of the install process. They may be installed but we want to make sure.

16. Now that is installed, we want to set up a packages repository for the Kubernetes packages

**curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -**

So here, you will start by downloading the gpg key for the repository and add it using the apt-key add

17. Now we configure the repository itself

```
cat << EOF | sudo tee /etc/apt/sources.list.d/Kubernetes.list
deb https://apt.kubernetes.io/ Kubernetes-xenial main
EOF
```

18. We then now want to do an update

```
sudo apt-get update
```

We are doing this to update to update our local package listings from that new repository

19. So now we install the Kubernetes packages

```
sudo apt-get install -y kubelet=1.24.0-00 kubeadm=1.24.0-00 kubectl=1.24.0-00
```

**^ I marked this red because this did not work.**

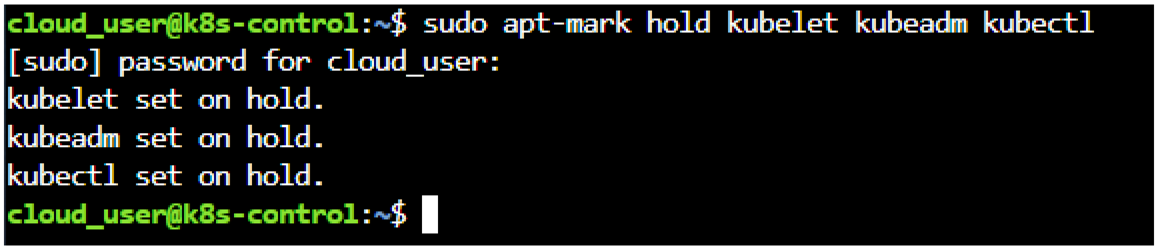
https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/install-kubeadm/

Going to this link helped. I did steps 2-4 and it worked.

20. Now we want to hold these packages so we can processed to installing other things.

```
sudo apt-mark hold kubelet kubeadm kubectl
```

***Figure 1-4***



```
cloud_user@k8s-control:~$ sudo apt-mark hold kubelet kubeadm kubectl
[sudo] password for cloud_user:
kubelet set on hold.
kubeadm set on hold.
kubectl set on hold.
cloud_user@k8s-control:~$
```

The reasoning behind the **apt-mark hold** command, is to put these packages on hold as of now so they won't automatically install, upgrade or remove. If we don't do this command then we can't do anything else with this on "unhold". This is a sense of manual control over when we update Kubernetes

- **kubeadm**: the command to bootstrap the cluster.
- **kubelet**: the component that runs on all of the machines in your cluster and does things like starting PODs and containers.
- **kubectl**: the command line until to talk to your cluster.

So far, we have installed and configured containerd, as well as Kubernetes packages installed on the control plane server.

## Setting up the Kube config

21. So now we want to successfully initialize the cluster. ONLY do this on the control-plane server. We don't need to do this on the worker nodes.

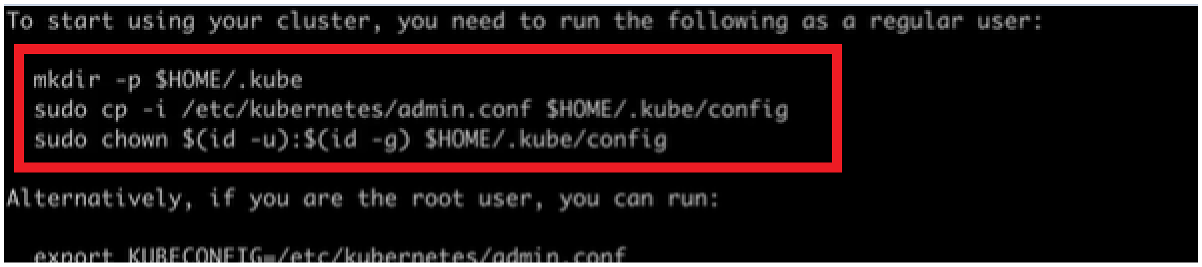
**sudo kubeadm init --pod-network-cidr 192.168.0.0/16 --kubernetes-version 1.25.0**

The kubeadm init command is for initiating it. Passing the pod network cidr. The ip address is just a range we will use for our internal pod network. And our networking plugin will require that same value.

Now that the cluster has been initialized, we need to setup the cube config. Which is a file that is going to allow us to authenticate and interact with the cluster using kubectl commands

Luckily enough, the commands for the cube is slightly above the command we just ran.

***Figure 1-5***



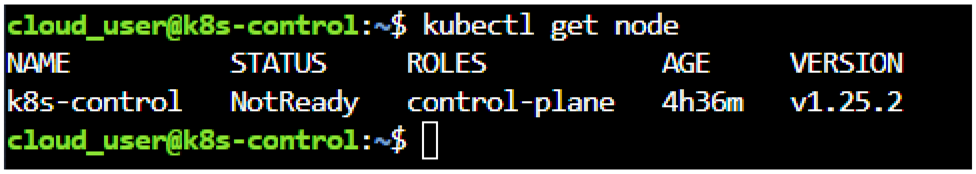
```
To start using your cluster, you need to run the following as a regular user:

  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
```

Once that is completed, this will setup our kube config so I can interact with the cluster.

22. To test that out, type in this command

**kubectl get node**

***Figure 1-6***



```
cloud_user@k8s-control:~$ kubectl get node
NAME          STATUS     ROLES           AGE     VERSION
k8s-control   NotReady   control-plane   4h36m   v1.25.2
cloud_user@k8s-control:~$
```

Above you can see we got a response. The age was from the last time I typed in the commands listed in ***Figure 1-5***

We are now able to interact with the cluster which shows 1 node (Control plane server) Which is in the not ready status because we haven't configured our networking plugins.

23. Before we add our worker nodes to the cluster. Let's go ahead and configure the networking plugin. This will be on the same control plane server.

**kubectl apply -f https://raw.githubusercontent.com/projectcalico/calico/v3.24.1/manifests/tigera-operator.yaml**

This will install the calico networking plugin. Which of course there are many more networking plugins available.

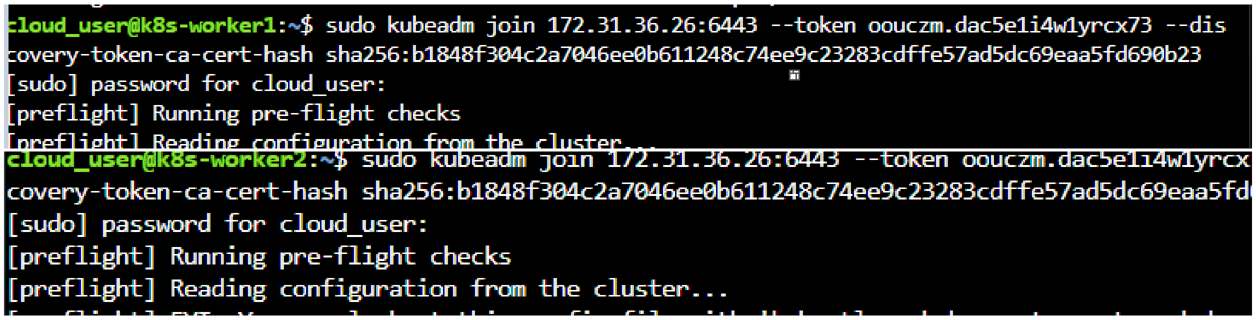
Now that our networking plugin is installed. Lets now join our worker1/2 nodes to the cluster

24. In order to join the cluster, we need a join command with a token

**kubeadm token create –print-join-command**

Once that command is finished it will populate a kubeadm join string. Copy that string and paste it with sudo into the worker1/2 nodes.

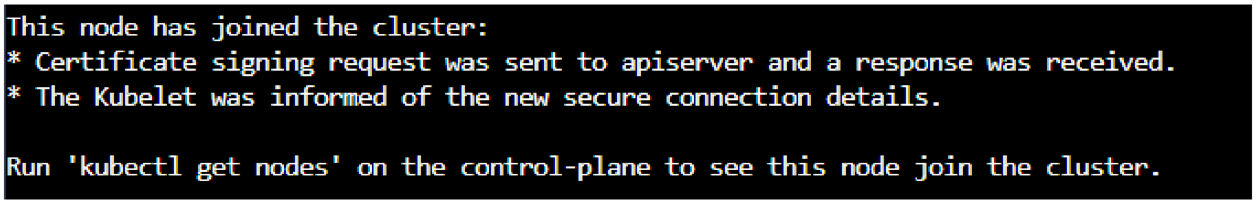
Figure 1-7



```
cloud_user@k8s-worker1:~$ sudo kubeadm join 172.31.36.26:6443 --token oouczm.dac5e1i4w1yrcx73 --dis
covery-token-ca-cert-hash sha256:b1848f304c2a7046ee0b611248c74ee9c23283cdffe57ad5dc69eaa5fd690b23
[sudo] password for cloud_user:
[preflight] Running pre-flight checks
[preflight] Reading configuration from the cluster...
cloud_user@k8s-worker2:~$ sudo kubeadm join 172.31.36.26:6443 --token oouczm.dac5e1i4w1yrcx
covery-token-ca-cert-hash sha256:b1848f304c2a7046ee0b611248c74ee9c23283cdffe57ad5dc69eaa5fd
[sudo] password for cloud_user:
[preflight] Running pre-flight checks
[preflight] Reading configuration from the cluster...
```

***Worker (Node) 1 and 2***

Figure 1-8



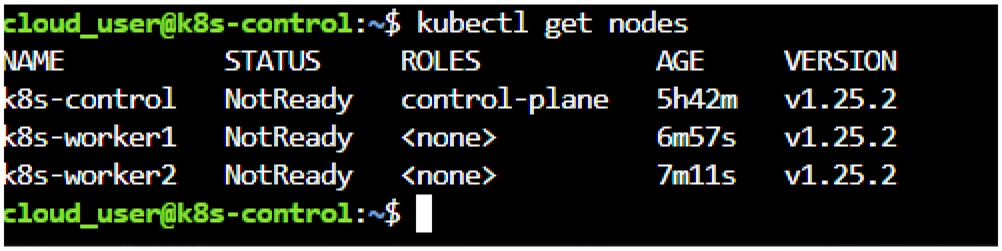
```
This node has joined the cluster:
* Certificate signing request was sent to apiserver and a response was received.
* The Kubelet was informed of the new secure connection details.

Run 'kubectl get nodes' on the control-plane to see this node join the cluster.
```

Before leaving both nodes, verify that it says this node has joined the cluster.

25. So now that both have joined the cluster. Go back to the control plane and verify if they indeed have joined

Remember type in **kubectl get nodes** command

Figure 1-10



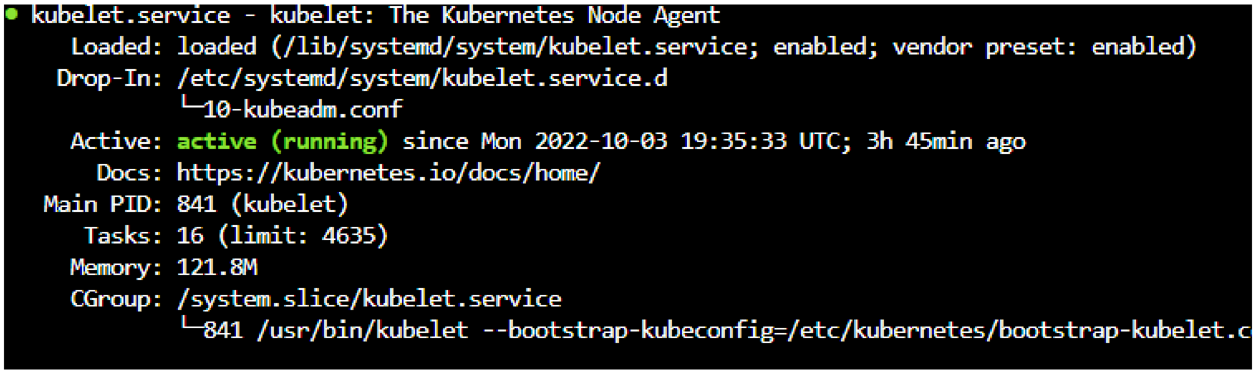
```
cloud_user@k8s-control:~$ kubectl get nodes
NAME          STATUS     ROLES           AGE     VERSION
k8s-control   NotReady   control-plane   5h42m   v1.25.2
k8s-worker1   NotReady   <none>          6m57s   v1.25.2
k8s-worker2   NotReady   <none>          7m11s   v1.25.2
cloud_user@k8s-control:~$
```

## TROUBLESHOOTING

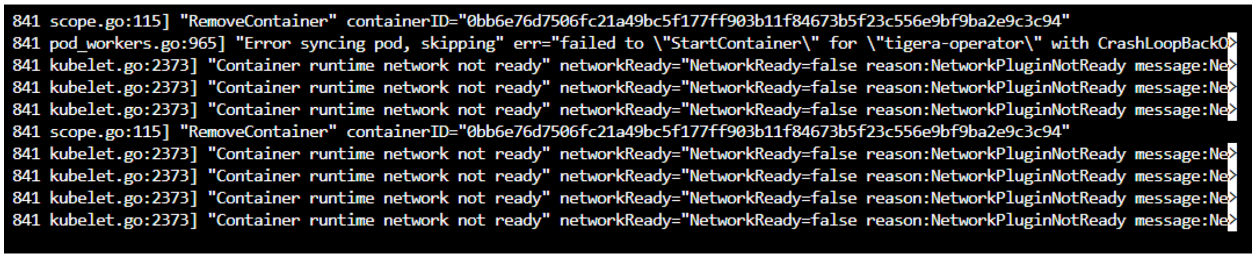
Here I am noticing my status has not changed after 5hrs.

Typing in the command **sudo systemctl status kubelet** lets me see the status of the kublet on the worker node

Figure 1-11



```
● kubelet.service - kubelet: The Kubernetes Node Agent
     Loaded: loaded (/lib/systemd/system/kubelet.service; enabled; vendor preset: enabled)
    Drop-In: /etc/systemd/system/kubelet.service.d
             └─10-kubeadm.conf
     Active: active (running) since Mon 2022-10-03 19:35:33 UTC; 3h 45min ago
       Docs: https://kubernetes.io/docs/home/
   Main PID: 841 (kubelet)
      Tasks: 16 (limit: 4635)
     Memory: 121.8M
     CGroup: /system.slice/kubelet.service
             └─841 /usr/bin/kubelet --bootstrap-kubeconfig=/etc/kubernetes/bootstrap-kubelet.c
```



```
841 scope.go:115] "RemoveContainer" containerID="0bb6e76d7506fc21a49bc5f177ff903b11f84673b5f23c556e9bf9ba2e9c3c94"
841 pod_workers.go:965] "Error syncing pod, skipping" err="failed to \"StartContainer\" for \"tigera-operator\" with CrashLoopBackO>
841 kubelet.go:2373] "Container runtime network not ready" networkReady="NetworkReady=false reason:NetworkPluginNotReady message:Ne>
841 kubelet.go:2373] "Container runtime network not ready" networkReady="NetworkReady=false reason:NetworkPluginNotReady message:Ne>
841 kubelet.go:2373] "Container runtime network not ready" networkReady="NetworkReady=false reason:NetworkPluginNotReady message:Ne>
841 scope.go:115] "RemoveContainer" containerID="0bb6e76d7506fc21a49bc5f177ff903b11f84673b5f23c556e9bf9ba2e9c3c94"
841 kubelet.go:2373] "Container runtime network not ready" networkReady="NetworkReady=false reason:NetworkPluginNotReady message:Ne>
841 kubelet.go:2373] "Container runtime network not ready" networkReady="NetworkReady=false reason:NetworkPluginNotReady message:Ne>
841 kubelet.go:2373] "Container runtime network not ready" networkReady="NetworkReady=false reason:NetworkPluginNotReady message:Ne>
841 kubelet.go:2373] "Container runtime network not ready" networkReady="NetworkReady=false reason:NetworkPluginNotReady message:Ne>
```

26. After seeing this I decided to restart it by typing

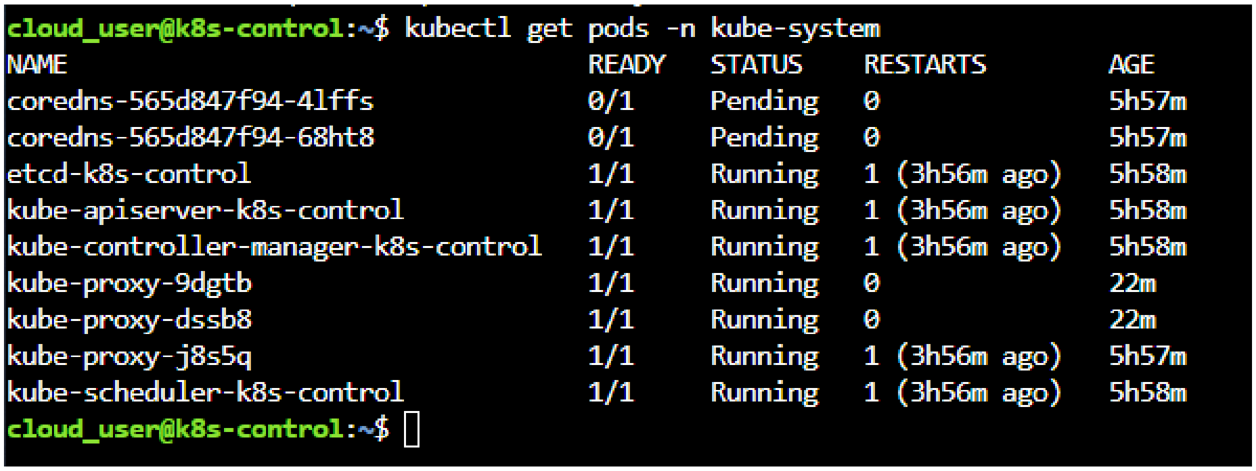
**sudo systemctl restart kubelet**

If a Pod is Running but not ready it means that the Readiness probe is failing. When the Readiness probe is failing, the Pod isn't attached to the Service, and no traffic is forwarded to that instance.

27. Another command I used to diagnose the pods

**kubectl get pods -n kube-system**

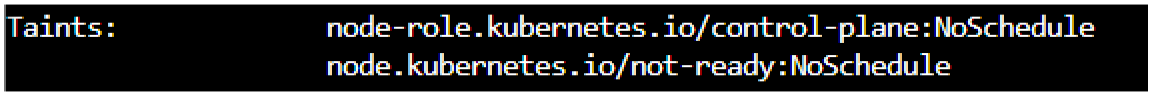
***Figure 1-12***



```
cloud_user@k8s-control:~$ kubectl get pods -n kube-system
NAME                                  READY   STATUS    RESTARTS        AGE
coredns-565d847f94-4lffs              0/1     Pending   0               5h57m
coredns-565d847f94-68ht8              0/1     Pending   0               5h57m
etcd-k8s-control                      1/1     Running   1 (3h56m ago)   5h58m
kube-apiserver-k8s-control            1/1     Running   1 (3h56m ago)   5h58m
kube-controller-manager-k8s-control   1/1     Running   1 (3h56m ago)   5h58m
kube-proxy-9dgtb                      1/1     Running   0               22m
kube-proxy-dssb8                      1/1     Running   0               22m
kube-proxy-j8s5q                      1/1     Running   1 (3h56m ago)   5h57m
kube-scheduler-k8s-control            1/1     Running   1 (3h56m ago)   5h58m
cloud_user@k8s-control:~$
```

Typing in the command **kubectl describe node k8s-control** shows a lot. And scrolling up Iunder the "taints section. We see it says not ready.

***Figure 1-13***



```
Taints:             node-role.kubernetes.io/control-plane:NoSchedule
                    node.kubernetes.io/not-ready:NoSchedule
```

And now thinking about it… The top 2 of that say pending may have something to do with plane and the worker nodes not running. So we shall see.

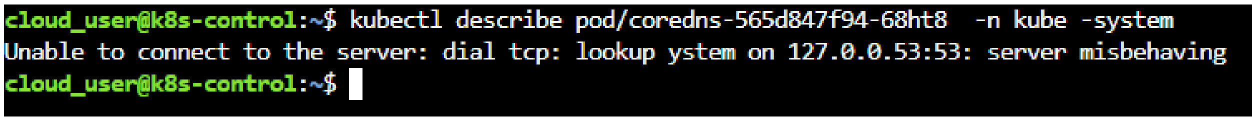
28. Ping both worker nodes to the control-plane

Both worker nodes can ping the k8s-control

29. So lets get a description on one of those pods

**kubectl describe pod/coredns-565d847f94-68ht8  -n kube -system**

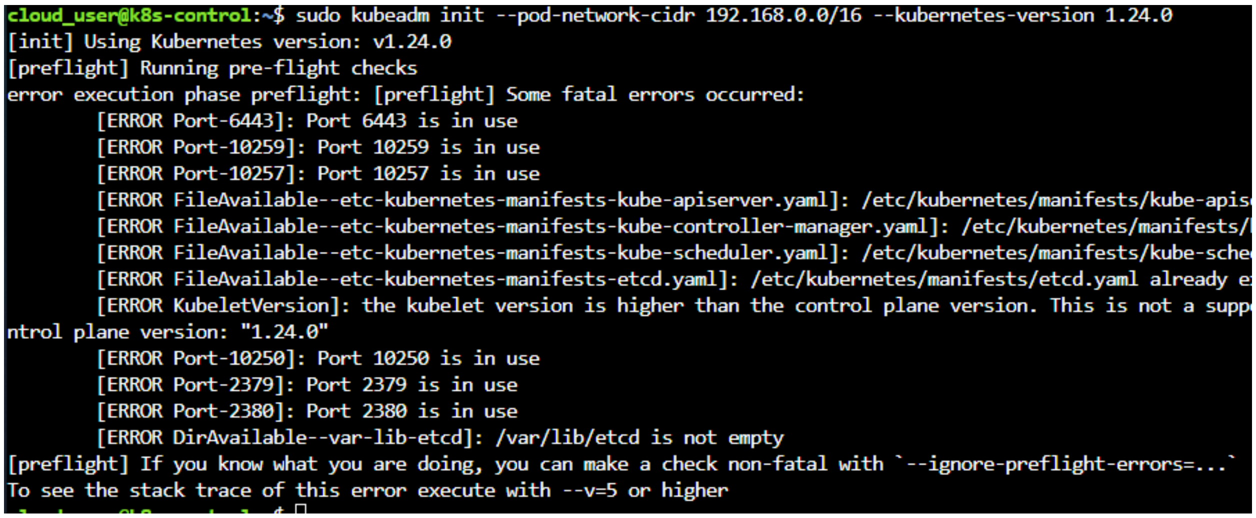
***Figure 1-14***



```
cloud_user@k8s-control:~$ kubectl describe pod/coredns-565d847f94-68ht8  -n kube -system
Unable to connect to the server: dial tcp: lookup ystem on 127.0.0.53:53: server misbehaving
cloud_user@k8s-control:~$
```

30. So after I did another init command to initialize the cluster again. But this time I changed the version back to 1.24 from 1.25 and it still didn't work. I say that because I am still seeing errors. Still keeping the same IP range. But it worked last time without any errors.

***Figure 1-15***

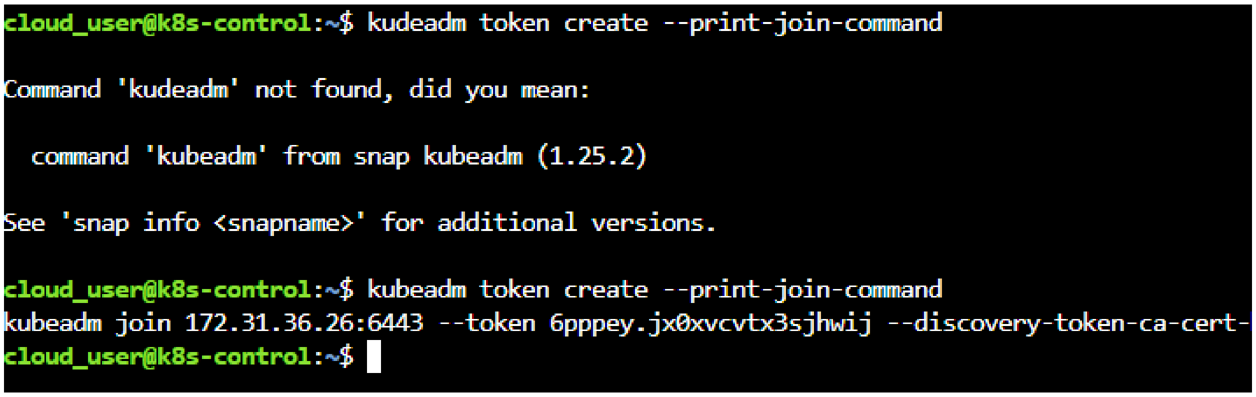


```
cloud_user@k8s-control:~$ sudo kubeadm init --pod-network-cidr 192.168.0.0/16 --kubernetes-version 1.24.0
[init] Using Kubernetes version: v1.24.0
[preflight] Running pre-flight checks
error execution phase preflight: [preflight] Some fatal errors occurred:
        [ERROR Port-6443]: Port 6443 is in use
        [ERROR Port-10259]: Port 10259 is in use
        [ERROR Port-10257]: Port 10257 is in use
        [ERROR FileAvailable--etc-kubernetes-manifests-kube-apiserver.yaml]: /etc/kubernetes/manifests/kube-apis
        [ERROR FileAvailable--etc-kubernetes-manifests-kube-controller-manager.yaml]: /etc/kubernetes/manifests/
        [ERROR FileAvailable--etc-kubernetes-manifests-kube-scheduler.yaml]: /etc/kubernetes/manifests/kube-sche
        [ERROR FileAvailable--etc-kubernetes-manifests-etcd.yaml]: /etc/kubernetes/manifests/etcd.yaml already e
        [ERROR KubeletVersion]: the kubelet version is higher than the control plane version. This is not a supp
ntrol plane version: "1.24.0"
        [ERROR Port-10250]: Port 10250 is in use
        [ERROR Port-2379]: Port 2379 is in use
        [ERROR Port-2380]: Port 2380 is in use
        [ERROR DirAvailable--var-lib-etcd]: /var/lib/etcd is not empty
[preflight] If you know what you are doing, you can make a check non-fatal with `--ignore-preflight-errors=...`
To see the stack trace of this error execute with --v=5 or higher
```

31. Looking back at the previous steps (Step 24) it looks like I may have forgot one command.

**kubeadm token create –print-join-command**

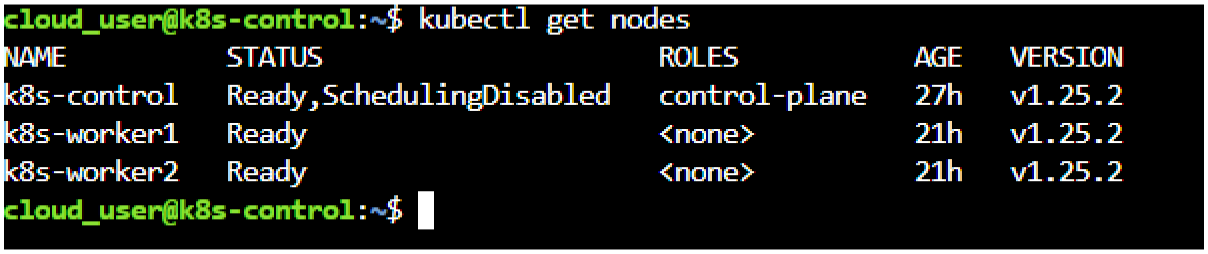
***Figure 1-16***



```
cloud_user@k8s-control:~$ kudeadm token create --print-join-command

Command 'kudeadm' not found, did you mean:

  command 'kubeadm' from snap kubeadm (1.25.2)

See 'snap info <snapname>' for additional versions.

cloud_user@k8s-control:~$ kubeadm token create --print-join-command
kubeadm join 172.31.36.26:6443 --token 6pppey.jx0xvcvtx3sjhwij --discovery-token-ca-cert-
cloud_user@k8s-control:~$
```

Maybe

After doing this command again in the control plane server it was now ready. It does say scheduling disabled. But as long as it says “ready” we are good.



```
cloud_user@k8s-control:~$ kubectl get nodes
NAME          STATUS                      ROLES           AGE   VERSION
k8s-control   Ready,SchedulingDisabled    control-plane   27h   v1.25.2
k8s-worker1   Ready                       <none>          21h   v1.25.2
k8s-worker2   Ready                       <none>          21h   v1.25.2
cloud_user@k8s-control:~$
```

*One thing I did learn was to better my troubleshooting methods. I feel as if maybe adding in those commands that doesn’t need to be added can hurt in the long run. So be careful on what you do. But luckily this is a playground/sandbox environment.