Setup k8s Cluster on QEMU/KVM VMs

1 Pre-knowledge

You will need to know a little bit about QEMU and KVM for a better understanding to set up the VMs.

1.1 The View of QEMU/KVM Stack [1]

QEMU could emulate the VM's processor through dynamic binary translation technology and provide different hardware and device models to the VMs. Moreover, QEMU is compatible with various VM OSs. KVM is a successful case of full virtualization for Linux on x86 hardware with CPUs supporting Intel Virtualization Technology (VT) or AMD Virtualization (AMD-V) extensions.

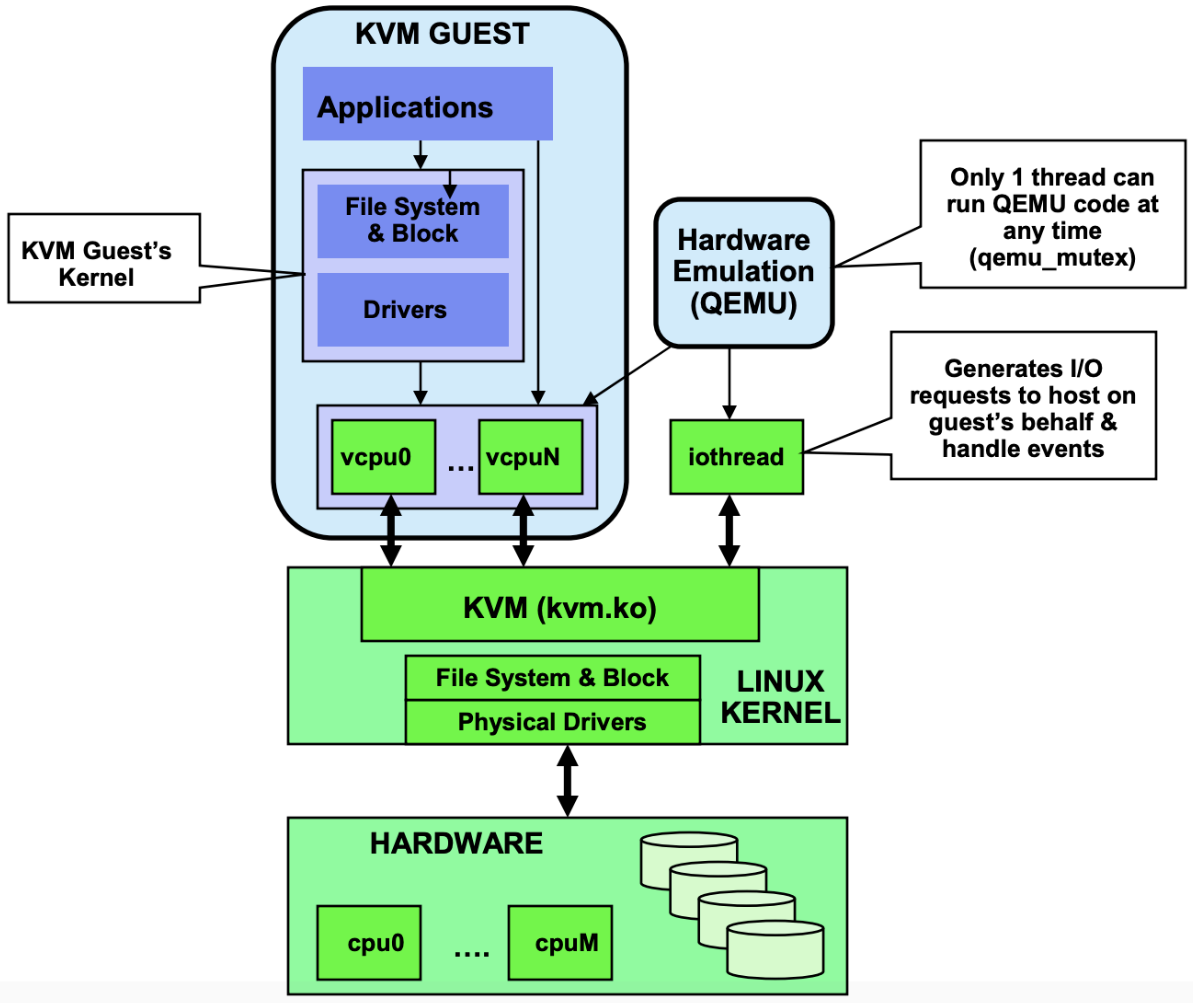


Figure 1: The View of QEMU/KVM Stack

1.2 System Requirements for QEMU/KVM Virtualization

First, KVM requires a CPU with virtualization extensions, found on most consumer CPUs. These extensions are called Intel VT or AMD-V. To check whether you have CPU support, run the following command:

```
# egrep '^flags.*(vmx|svm)' /proc/cpuinfo
```

If this command results in nothing printed, your system does not support the relevant virtualization extensions. You can still use QEMU/KVM, but the emulator will fall back to software virtualization, which is much slower.

Take Fedora as an example [2], A minimal command-line Fedora system requires 600MB of storage, and at least 756MB is recommended for each guest of a modern operating system.

2 Architecture Overview

I have three Fedora 40 QEMU/KVM VMs running on my Fedora 40 Host, the host will pass through CPU configuration to the VMs, and VMs are using the “NAT” network mode. In the later setup for the k8s cluster, one VM will be the control plane and the other two VMs will be the worker nodes.

2.1 Host’s and VMs’ Configurations

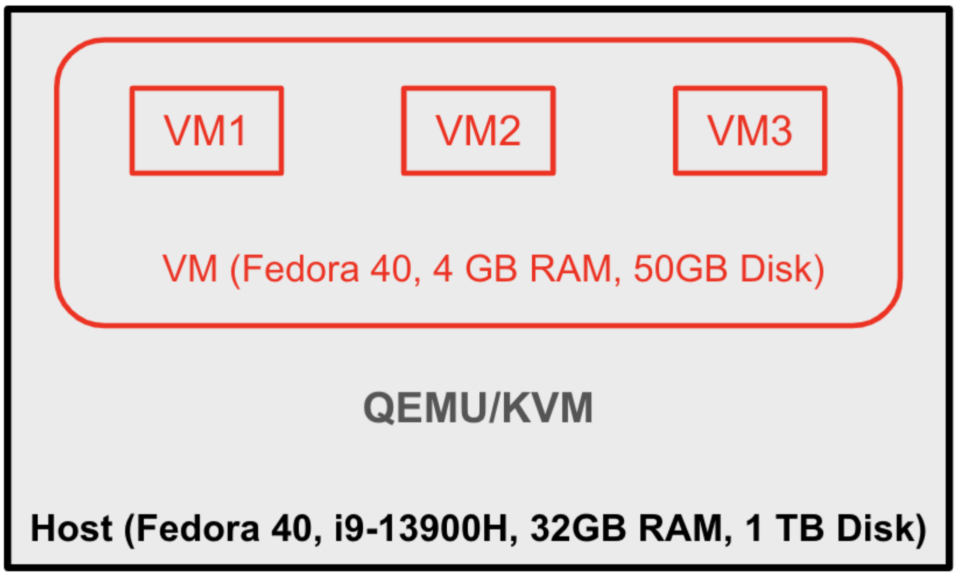

VM1
VM2
VM3
VM (Fedora 40, 4 GB RAM, 50GB Disk)
QEMU/KVM
Host (Fedora 40, i9-13900H, 32GB RAM, 1 TB Disk)

Figure 2: Host’s and VMs’ configurations

2.2 k8s (1.30 in the context of this document) Cluster Architecture

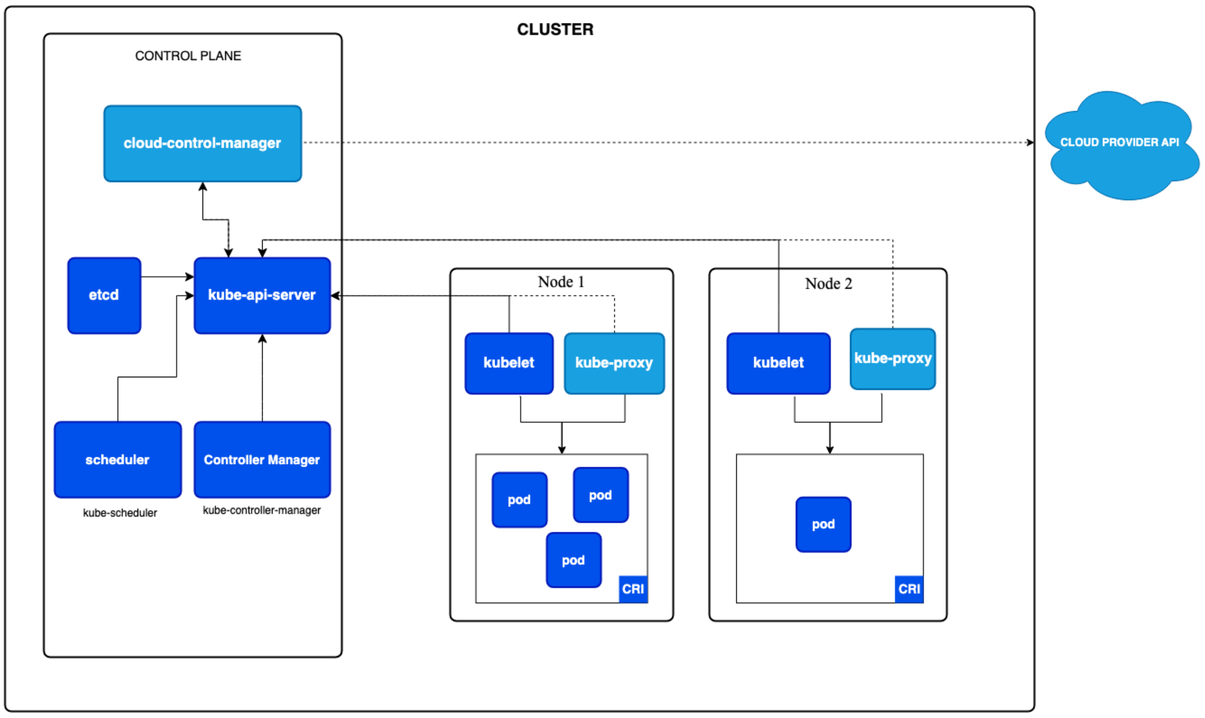


Figure 3: k8s cluster architecture [3]

3 Setup QEMU/KVM VMs

3.1 Configure Mirror List for Fedora 40 Host

```
# backup the old configuration
# mv /etc/yum.repos.d/fedora.repo /etc/yum.repos.d/fedora.repo.backup
# mv /etc/yum.repos.d/fedora-updates.repo /etc/yum.repos.d/fedora-updates.repo.backup
# obtain fedora.repo
# wget -O /etc/yum.repos.d/fedora.repo http://mirrors.aliyun.com/repo/fedora.repo
# fedora-updates.repo
# wget -O /etc/yum.repos.d/fedora-updates.repo http://mirrors.aliyun.com/repo/fedora-updates.repo
```

3.2 Install Packages for Virtualization Group

Run the following command to install the mandatory and default packages in the virtualization group. After the packages are installed, start the libvirtd service. Also, make sure KVM kernel modules are properly loaded.

```
# sudo dnf install @virtualization -y
# sudo systemctl start libvirtd
# sudo systemctl enable libvirtd

# lsmod | grep kvm
```

```
kvm_intel            446464  17
kvm                 1445888  12 kvm_intel
```

3.3 Install QEMU/KVM VM via Virt-Manager

Virt-Manager is recommended for the users to install VMs [4]. However, to simplify the steps, you could download a cloud image directly. Then, you could just choose “Import existing disk image” when you “New VM” via Virt-Manager GUI, and make other settings by default.

```
# wget -O /var/lib/libvirt/images/Fedora-Cloud-Base-Generic.x86_64-40-1.14.qcow2
https://download.fedoraproject.org/pub/fedora/linux/releases/40/Cloud/x86_64/images/Fedora-Cloud-Base-Generic.x86_64-40-1.14.qcow2
```

Reset the password for the qcow2 image downloaded above

```
# dnf install guestfs-tools -y
# virt-customize -a /var/lib/libvirt/images/Fedora-Cloud-Base-Generic.x86_64-40-1.14.qcow2
--root-password password:YOURPASSWORD
```

3.4 Resize the Disk for VM If Needed

Downloading the cloud image from the official website directly might be convenient, but you might need to extend the size for the qcow2 file as 5 GB might not be enough for long-term usage.

Turn off the VM first, as qcow2 files cannot be written by more than one process at one time.

```
# qemu-img info /var/lib/libvirt/images/Fedora-Cloud-Base-Generic.x86_64-40-1.14.qcow2 | grep
"virtual size"
virtual size: 5 GiB (5368709120 bytes)

# qemu-img resize /var/lib/libvirt/images/Fedora-Cloud-Base-Generic.x86_64-40-1.14.qcow2 +45G
Image resized.

# qemu-img info /var/lib/libvirt/images/Fedora-Cloud-Base-Generic.x86_64-40-1.14.qcow2 | grep
"virtual size"
virtual size: 50 GiB (53687091200 bytes)
```

Now boot the VM, and extend the disk as follows.

```
# virsh domblklist rezise-qcow2-image
 Target   Source
---------------------------------------------------------------------------------
 vda      /var/lib/libvirt/images/Fedora-Cloud-Base-Generic.x86_64-40-1.14.qcow2

# virsh console rezise-qcow2-image
Connected to domain 'rezise-qcow2-image'
Escape character is ^] (Ctrl + ])
localhost login: root
Password:
```

```
[root@localhost ~]# fdisk -l
GPT PMBR size mismatch (10485759 != 104857599) will be corrected by write.
The backup GPT table is not on the end of the device.
Disk /dev/vda: 50 GiB, 53687091200 bytes, 104857600 sectors
…
Device       Start      End Sectors  Size Type
/dev/vda1     2048     6143    4096    2M BIOS boot
/dev/vda2     6144   210943  204800  100M EFI System
/dev/vda3   210944  2258943 2048000 1000M Linux extended boot
/dev/vda4  2258944 10485726 8226783  3.9G Linux root (x86-64)
…

[root@localhost ~]# fdisk /dev/vda
Welcome to fdisk (util-linux 2.40-rc1).
…
partitions on this disk.
Command (m for help): d
Partition number (1-4, default 4):
Partition 4 has been deleted.
Command (m for help): n
Partition number (4-128, default 4):
First sector (2258944-104857566, default 2258944):
Last sector, +/-sectors or +/-size{K,M,G,T,P} (2258944-104857566, default 104855551):
Created a new partition 4 of type 'Linux filesystem' and of size 48.9 GiB.
Partition #4 contains a btrfs signature.
Do you want to remove the signature? [Y]es/[N]o: N
Command (m for help): t
Partition number (1-4, default 4):
Partition type or alias (type L to list all): 44
Changed type of partition 'Linux filesystem' to 'Linux LVM'.
Command (m for help): w
The partition table has been altered.
Syncing disks.

[root@localhost ~]# reboot
[root@localhost ~]# fdisk /dev/vda
…
Device       Start       End   Sectors  Size Type
/dev/vda1     2048      6143      4096    2M BIOS boot
/dev/vda2     6144    210943    204800  100M EFI System
/dev/vda3   210944   2258943   2048000 1000M Linux extended boot
/dev/vda4  2258944 104855551 102596608 48.9G Linux LVM
…
```

3.5 List all VMs

At the end, you will have three VMs. You can see them from both Virt-Manager or “virsh” command as follows, feel free to configure the hostname for all the VMs as well based on your settings.

From the host, all the VMs’ states and IPs could be seen as follows.

```
# virsh list --all
 Id   Name           State
-----------------------------
 37   ControlPlane   running
 38   Node1          running
 39   Node2          running

# virsh domifaddr ControlPlane
 Name       MAC address          Protocol     Address
-------------------------------------------------------------------------------
 vnet33     52:54:00:6e:49:68    ipv4         192.168.124.249/24

# virsh domifaddr Node1
 Name       MAC address          Protocol     Address
-------------------------------------------------------------------------------
 vnet34     52:54:00:60:eb:37    ipv4         192.168.124.130/24

# virsh domifaddr Node2
 Name       MAC address          Protocol     Address
-------------------------------------------------------------------------------
 vnet35     52:54:00:54:88:c7    ipv4         192.168.124.220/24
```

Add the following content to the host and all three VMs.

```
# cat /etc/hosts
…
192.168.124.249 controlPlane.example.org controlPlane
192.168.124.130 node1.example.org node1
192.168.124.220 node2.example.org node2
```

4 Setup k8s Cluster on QEMU/KVM VMs

Overall, the best way to set up k8s cluster is to follow the documents from kubernetes.io, here you could have the most simplest steps to set everything up.

In the context of this document, you will need to make sure all the steps from 4.1 to 4.4 are executed in all three VMs created above.

4.1 Network Configuration - Enable IPv4 packet forwarding

```
# cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf
net.ipv4.ip_forward = 1
EOF
net.ipv4.ip_forward = 1
# sudo sysctl --system
…
# sysctl net.ipv4.ip_forward
net.ipv4.ip_forward = 1
```

4.2 Install Container Runtimes (Containerd in the context of this document)

There are a few methods to install containerd, more details could be seen getting started with containerd,I will just provide one method here.

```
#wget
https://github.com/containerd/containerd/releases/download/v1.7.18/containerd-1.7.18-linux-amd64.ta
r.gz
# tar Cxzvf /usr/local containerd-1.7.18-linux-amd64.tar.gz

# wget https://github.com/opencontainers/runc/releases/download/v1.1.13/runc.amd64
# install -m 755 runc.amd64 /usr/local/sbin/runc

#wget
https://github.com/containernetworking/plugins/releases/download/v1.5.1/cni-plugins-linux-amd64-v1.
5.1.tgz
# mkdir -p /opt/cni/bin
# tar Cxzvf /opt/cni/bin cni-plugins-linux-amd64-v1.5.1.tgz

# mkdir -p /usr/local/lib/systemd/system/
# touch /usr/local/lib/systemd/system/containerd.service
```

COPY the content from https://raw.githubusercontent.com/containerd/containerd/main/containerd.service to file "/usr/local/lib/systemd/system/containerd.service", and then execute the following two commands, then you could start "containerd" via "systemd".

```
# systemctl daemon-reload
# systemctl enable --now containerd
```

4.3 Configure Containerd

4.3.1 Use the systemd cgroup driver in /etc/containerd/config.toml with runc

```
# mkdir -p /etc/containerd
# containerd config default > /etc/containerd/config.toml
# cat /etc/containerd/config.toml | grep "SystemdCgroup = false"
            SystemdCgroup = false
# sed -i 's/SystemdCgroup = false/SystemdCgroup = true/g' /etc/containerd/config.toml
# cat /etc/containerd/config.toml | grep "SystemdCgroup = true"
             SystemdCgroup = true
# systemctl restart containerd
```

4.3.2 Override the sandbox (pause) image if needed

```
# cat /etc/containerd/config.toml | grep sandbox_image
    sandbox_image = "registry.k8s.io/pause:3.8"
```

The default value of “sandbox_image” in /etc/containerd/config.toml will make pulling images from “registry.k8s.io”, while it is not all the time available in Chinese mainland. Thus, it is recommended to override the sandbox image with aliyun as follows.

```
# cat /etc/containerd/config.toml | grep sandbox_image
   # sandbox_image = "registry.k8s.io/pause:3.8"
   sandbox_image = "registry.aliyuncs.com/google_containers/pause:3.9"
# systemctl restart containerd
```

4.3.3 Configure image/runtime-endpoint for crictl if possible

In case you have multiple container runtimes in your environment, it would be recommended to configure “containerd” as image-endpoint and runtime-endpoint for crictl.

```
# cat /etc/crictl.yaml
runtime-endpoint: unix:///run/containerd/containerd.sock
image-endpoint: unix:///run/containerd/containerd.sock

# systemctl restart containerd
```

4.4 Install Kubernetes with deployment tools (kubeadm in the context of this document)

There are some checkpoints before you begin, which could be seen link, if you start from a clean environment, then there should not be too much to worry about. However, it would always be better to make sure everything is as expected.

If you have failed in “kubeadm init”, remember to do “kubeadm reset” to clean all the resources before your next trial.

4.4.1 Turn off Swap as Recommended

By default, for Fedora 33 and later versions, “zram-generator-defaults” is present means swap-on-zram is set-up during startup. You will need to remove the package to turn it off [5].

```
# dnf remove zram-generator-defaults -y
# swapoff -a
# cat /proc/swaps
Filename                                Type            Size            Used            Priority
#
```

4.4.2 Set SELinux to Permissive Mode (for k8s 1.30 and Red Hat-based distributions)

```
#  sudo setenforce 0
#  sudo sed -i 's/^SELINUX=enforcing$/SELINUX=permissive/' /etc/selinux/config
```

4.4.3 Turn off the Firewalld If Possible

```
# systemctl stop firewalld
# systemctl disable firewalld
# systemctl restart containerd
```

4.4.4 Add the Kubernetes Mirror List and Install Packages

Silimar to overriding the sand_image, it is recommended to use the mirror list from aliyun to install k8s related packages.

```
# cat <<EOF | sudo tee /etc/yum.repos.d/kubernetes.repo
[kubernetes]
name=Kubernetes
baseurl=https://mirrors.aliyun.com/kubernetes-new/core/stable/v1.30/rpm/
enabled=1
gpgcheck=1
gpgkey=https://mirrors.aliyun.com/kubernetes-new/core/stable/v1.30/rpm/repodata/repomd.xml.key
EOF
```

Install the packages and enable “kubelet” service.

```
#  dnf install -y kubelet kubeadm kubectl
#  systemctl enable --now kubelet
```

4.5 Create a cluster with kubeadm

4.5.1 Initialize control-plane node

In the “kubeadm-config.yaml”, “imageRepository” is configured with aliyun’s registry in case of the default value of “imageRepository” is not accessible in Chinese mainland.

```
[root@controlPlane ~]#  cat kubeadm-config.yaml
apiVersion: kubeadm.k8s.io/v1beta3
kind: ClusterConfiguration
kubernetesVersion: stable
imageRepository: registry.aliyuncs.com/google_containers
networking:
   podSubnet: 10.244.0.0/16

[root@controlPlane ~]# kubeadm init --config=kubeadm-config.yaml --v=5
…
Your Kubernetes control-plane has initialized successfully!
…
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
You should now deploy a pod network to the cluster.
```

```
Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:
  https://kubernetes.io/docs/concepts/cluster-administration/addons/
Then you can join any number of worker nodes by running the following on each as root:
kubeadm join 192.168.124.249:6443 --token 6gbpg9.1kee56jowc32f61d \
        --discovery-token-ca-cert-hash
sha256:02afd1295119365c79a1dfe522dac8c617df3ef5031e1edb7d95b586ed7164f7
```

4.5.2 Join worker nodes to cluster

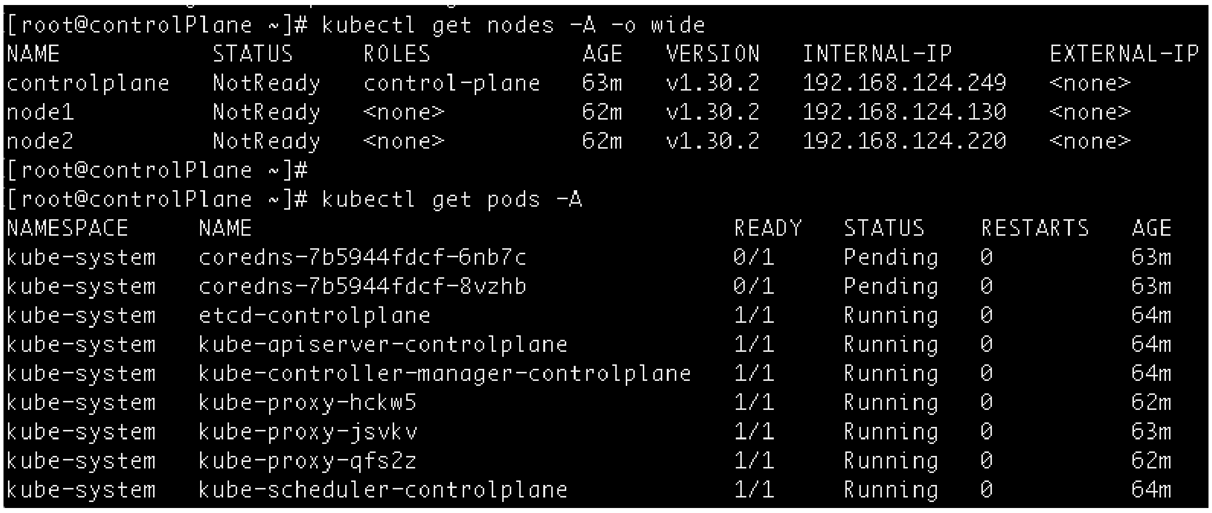
Run the “kubeadm join” command which could be seen once your “kubeadm init” succeeds.

```
[root@node1 ~]# kubeadm join 192.168.124.249:6443 --token 6gbpg9.1kee56jowc32f61d \
                                                        --discovery-token-ca-cert-hash
sha256:02afd1295119365c79a1dfe522dac8c617df3ef5031e1edb7d95b586ed7164f7
[preflight] Running pre-flight checks
…
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.
```

```
[root@node2 ~]# kubeadm join 192.168.124.249:6443 --token 6gbpg9.1kee56jowc32f61d \
                                                        --discovery-token-ca-cert-hash
sha256:02afd1295119365c79a1dfe522dac8c617df3ef5031e1edb7d95b586ed7164f7
[preflight] Running pre-flight checks
…
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.
```

4.5.3 Check Cluster Status

now, you should have a k8s cluster with 1 control-plane and two nodes as follows. Note: core-dns-* pods are still in “Pending” status cause there is no pod network add-on installed in the k8s cluster yet.



```
[root@controlPlane ~]# kubectl get nodes -A -o wide
NAME           STATUS     ROLES           AGE   VERSION   INTERNAL-IP       EXTERNAL-IP
controlplane   NotReady   control-plane   63m   v1.30.2   192.168.124.249   <none>
node1          NotReady   <none>          62m   v1.30.2   192.168.124.130   <none>
node2          NotReady   <none>          62m   v1.30.2   192.168.124.220   <none>
[root@controlPlane ~]#
[root@controlPlane ~]# kubectl get pods -A
NAMESPACE     NAME                                   READY   STATUS    RESTARTS   AGE
kube-system   coredns-7b5944fdcf-6nb7c               0/1     Pending   0          63m
kube-system   coredns-7b5944fdcf-8vzhb               0/1     Pending   0          63m
kube-system   etcd-controlplane                      1/1     Running   0          64m
kube-system   kube-apiserver-controlplane            1/1     Running   0          64m
kube-system   kube-controller-manager-controlplane   1/1     Running   0          64m
kube-system   kube-proxy-hckw5                       1/1     Running   0          62m
kube-system   kube-proxy-jsvkv                       1/1     Running   0          63m
kube-system   kube-proxy-qfs2z                       1/1     Running   0          62m
kube-system   kube-scheduler-controlplane            1/1     Running   0          64m
```

Further Steps

Now you will need to install a pod network add-on for the k8s cluster.

References

[1] https://docs.huihoo.com/virtualization/2010/02-kvm-storage-stack-performance.pdf
[2]
https://docs.fedoraproject.org/en-US/quick-docs/virtualization-getting-started/#_system_requirements
[3] https://kubernetes.io/docs/concepts/architecture/
[4]
https://docs.fedoraproject.org/en-US/quick-docs/virtualization-getting-started/#_creating_a_guest_with_virt_manager
[5] https://www.fedoraproject.org/wiki/Changes/SwapOnZRAM
[6] https://github.com/containerd/containerd/blob/main/docs/getting-started.md