Node

- Hardware
- Operating system
- MicroK8s
- Networking

Hardware

There are a couple of considerations when choosing your hardware or virtual "hardware" for use as Kubernetes nodes.

- MicroK8s requires at least 3 nodes to work in HA mode, so we'll start with 3 VMs
- While MicroK8s is quite lightweight, by the time you start adding the storage capability you will need a reasonable amount of memory. Recommended minimum spec for this guide is 2 CPUs and 4GB RAM. More is obviously better.
- If using Ceph as a storage backend, each VM will need two block devices (disks). One should be formatted and used as a normal OS device, and the other should be left untouched so it can be claimed by Ceph. The OS disk will also contain cached container images so could get quite large. I've allowed 16GB for the OS disk and Ceph requires a minimum of 10GB for its disk.
- If running in VirtualBox, place all VMs either in the same NAT network, or bridged to the host network. Ideally have static IPs.
- If you are running on bare metal, make sure the machines are on the same network, or at least on networks that can talk to each other.

Operating system

This guide focuses on Fedora Server but should be applicable to many distributions with minor tweaks. CentOS or Ubuntu Server or would also work just as well.

- Don't create a swap partition on these machines
- Make sure ntp is enabled for accurate time
- Make sure the VMs have static IPs or DHCP reservations, so their IPs won't change

Snap

Snap is a package manager that contains MicroK8s. It comes preinstalled on Ubuntu, but if you're on CentOS, Fedora or others, you'll need to install it on all your nodes:

https://snapcraft.io/docs/installing-snap-on-centos

```
sudo dnf -y install epel-release
sudo dnf -y install snapd
sudo systemctl enable --now snapd
sudo ln -s /var/lib/snapd/snap /snap
```

MicroK8s

MicroK8s is a lightweight, pre-packaged Kubernetes distribution which is easy to use and works well for small deployments.

Install

Install MicroK8s 1.19.1 or greater from Snap on all your nodes. https://microk8s.io/

```
# To install
sudo snap install microk8s --classic --channel=1.22/stable

# Or to upgrade
sudo snap refresh microk8s --classic --channel=1.22/stable

sudo usermod -a -G microk8s jonathan
sudo chown -f -R jonathan ~/.kube
microk8s status --wait-ready
```

Cluster

Enable microk8s HA mode on all nodes, which allows any of the worker nodes to also behave as a master, instead of just being a worker node. This must be enabled before nodes are joined to the master. https://microk8s.io/docs/high-availability

```
microk8s enable ha-cluster
```

Enable microk8s clustering, which allows you to add multiple worker nodes to your existing master node https://microk8s.io/docs/clustering

On the first node:

```
[jonathan@kube01 ~]$ microk8s add-node
From the node you wish to join to this cluster, run the following:
microk8s join 192.168.0.41:25000/xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

Then execute the join command on the other nodes, to join them to the master. Verify that they are correctly joined:

```
[jonathan@kube01 ~]$ microk8s kubectl get nodes
NAME                        STATUS   ROLES    AGE     VERSION
kube02.jonathangazeley.com   Ready    <none>   27m     v1.18.8-33+c9c9e3f85f05f9
kube01.jonathangazeley.com   Ready    <none>   6d16h   v1.18.8-33+c9c9e3f85f05f9
kube03.jonathangazeley.com   Ready    <none>   86s     v1.18.8-33+c9c9e3f85f05f9
```

Finally make sure that full HA mode is enabled:

```
[jonathan@kube01 ~]$ microk8s status
microk8s is running
high-availability: yes
  datastore master nodes: 192.168.0.41:19001 192.168.0.42:19001 192.168.0.43:19001
  datastore standby nodes: none
addons:
  enabled:
...
```

We've already checked that all 3 nodes are up. Now let's make sure pods are being scheduled on all nodes:

```
[jonathan@kube01 ~]$ kubectl get pods --all-namespaces -o wide
NAMESPACE     NAME                                       READY   STATUS          RESTARTS   AGE    IP
NODE
kube-system   calico-node-bqqqd                          1/1     Running         0          112m   192.168.0.41
kube01.jonathangazeley.com
kube-system   calico-node-z4sxd                          1/1     Running         0          110m   192.168.0.43
kube03.jonathangazeley.com
kube-system   calico-kube-controllers-847c8c99d-4qblz    1/1     Running         0          115m
10.1.58.1     kube01.jonathangazeley.com
kube-system   coredns-86f78bb79c-t2sgt                   1/1     Running         0          109m   10.1.111.65
kube02.jonathangazeley.com
kube-system   calico-node-t5skc                          1/1     Running         0          111m   192.168.0.42
kube02.jonathangazeley.com
```

Networking

Firewall

Create firewall rules for your nodes, so they can communicate with each other. Refer to the ports guide https://microk8s.io/docs/ports

```
sudo firewall-cmd --permanent --add-port=6443/tcp□□□# Kube API server
sudo firewall-cmd --permanent --add-port=2379-2380/tcp□□# etcd
sudo firewall-cmd --permanent --add-port=10250/tcp□□□# kubelet
sudo firewall-cmd --permanent --add-port=10251/tcp□□□# kube-scheduler
sudo firewall-cmd --permanent --add-port=10252/tcp□□□# kube-controller-manager
sudo firewall-cmd --permanent --add-port=10255/tcp
sudo firewall-cmd --permanent --add-port=25000/tcp□□□# microk8s cluster
sudo firewall-cmd --permanent --add-port=19001/tcp
sudo firewall-cmd --permanent --add-port=4789/udp□□□# Calico with VXLAN
sudo firewall-cmd --permanent --add-port=5473/tcp□□□# Calico with Typha
sudo firewall-cmd --permanent --add-port={8285,8472}/udp□# Flannel
sudo firewall-cmd --add-masquerade --permanent
sudo firewall-cmd --zone=trusted --add-interface=vxlan.calico --permanent
sudo firewall-cmd --permanent --add-port=30000-32767/tcp□# NodePorts on control plane IP
sudo firewall-cmd --reload
```

Or the cheat way:

```
# Fedora
sudo systemctl disable --now firewalld


# Ubuntu
sudo ufw disable
```