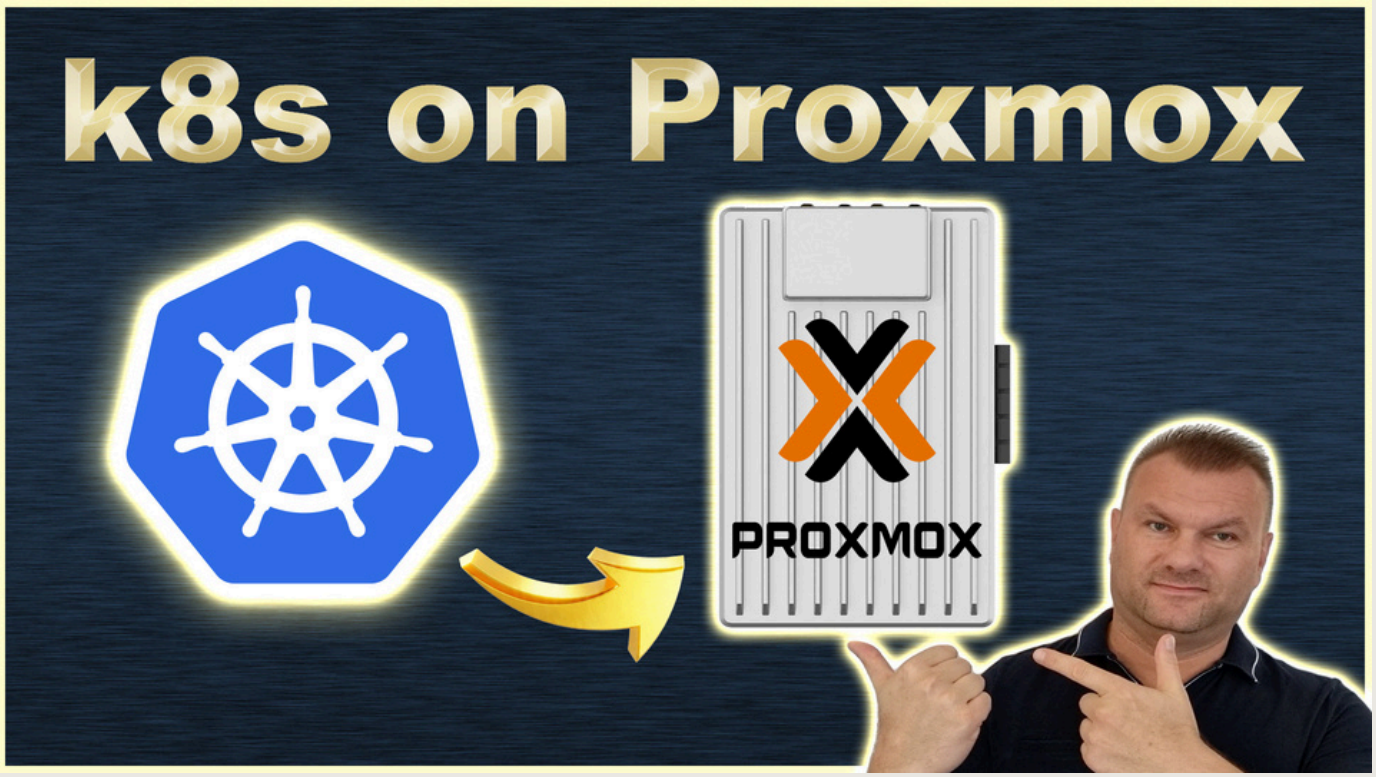
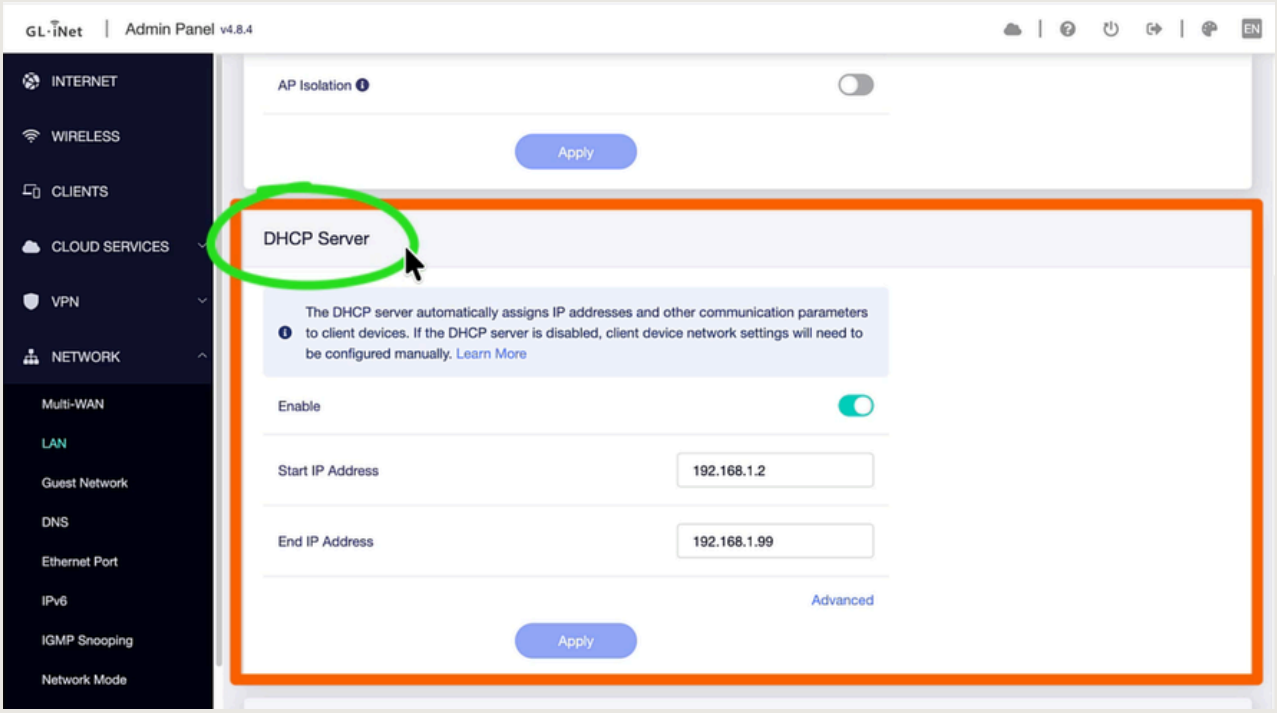
# Run Full kubernetes cluster on Proxmox!




Link to Automation Avenue platform

This PDF will help you deploy full kubernetes cluster on Proxmox VE hypervisor. Please use this PDF together with GitHub repo you can find **HERE**

_As per that document - we first need to make sure we have available static ip addresses that we can assign manually to our virtual machines and then also to MetalLB load balander.
I do it by going to my router and limit its DHCP scope:




My DHCP can only assign ip addresses from 192.168.1.2 to 192.168.1.99, which means the 192.168.1.100 to 192.168.1.254 is my pool of static ip addresses that I can assign manually.

We can now create at least 3 virtual machines - 1 will be acting as our kubernetes master node,and remaning 2 will be our worker nodes running containers.

I do that by creating 1 virtual machine manually, then I create a templates based on this manually created machine I can find in /etc/pve/qemu-server/<vm_id>.conf configuration file.

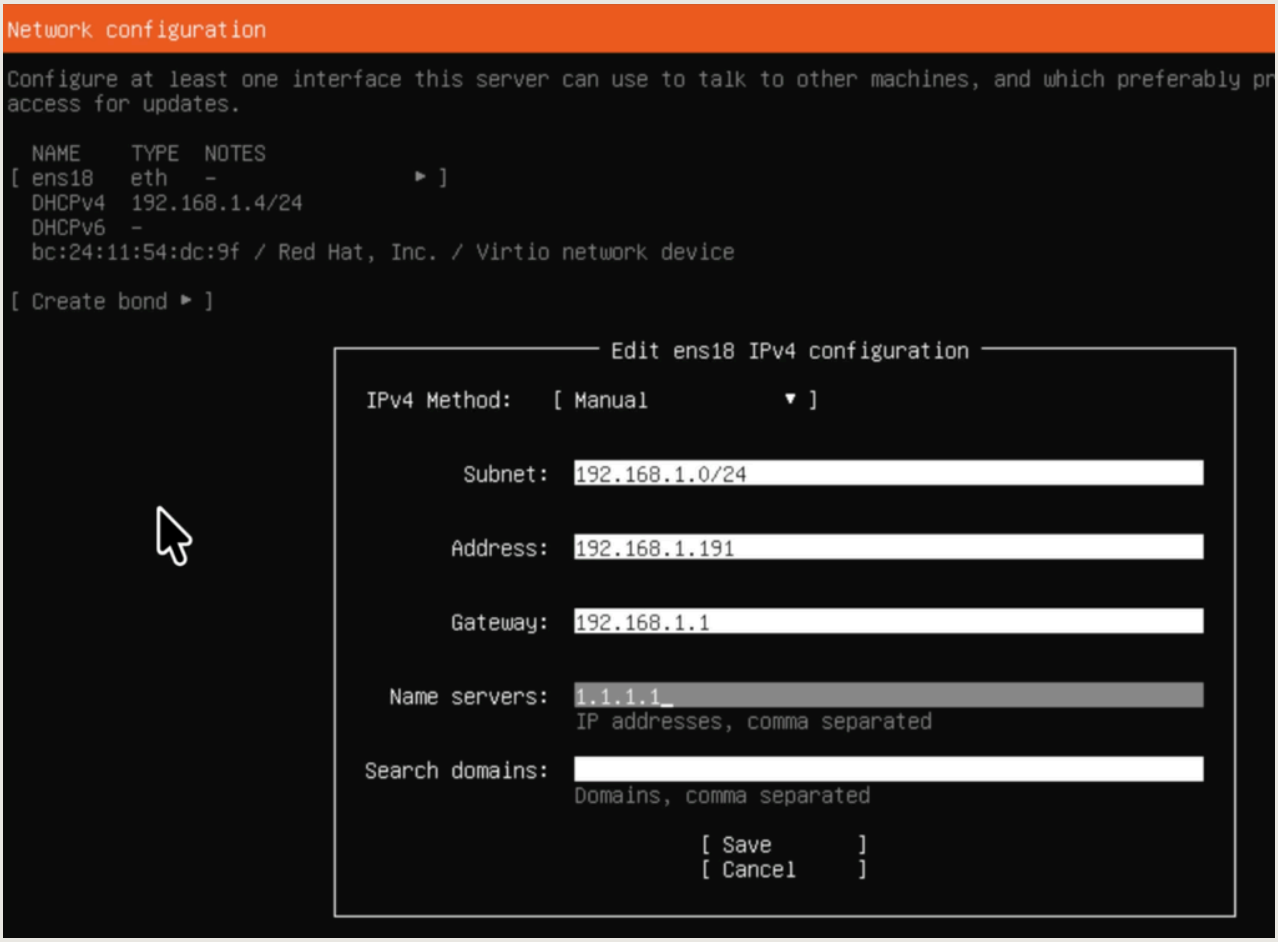
As you can see - i use Ubuntu 26.04 LTS image, but you can use different debian based image or even completely different linux distribution (the commands will be different then for you though)


This is template for master node:

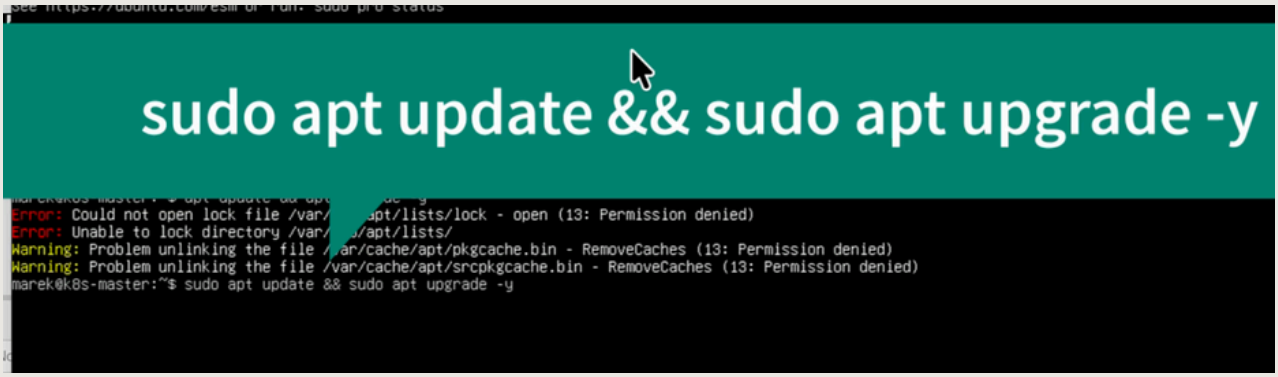
*qm create 191 *
*--name k8s-master *
*--agent 1 *
*--balloon 0 *
*--cores 4 *
*--sockets 1 *
*--cpu host *
*--memory 4096 *
*--numa 0 *
*--scsihw virtio-scsi-single *
*--net0 virtio,bridge=vmbr0,firewall=1 *
*--ide2 local:iso/ubuntu-26.04-snapshot3-live-server-amd64.iso,media=cdrom *
*--boot "order=scsi0;ide2;net0" *
*--scsi0 transcend:50,discard=on,iothread=1,ssd=1*

but I suggest you copy them from the GitHub repo instead.

We can then go through the VM installation process setting the VM machine ip addresses to static ip addresses from the pool outside the scope of router DHCP:



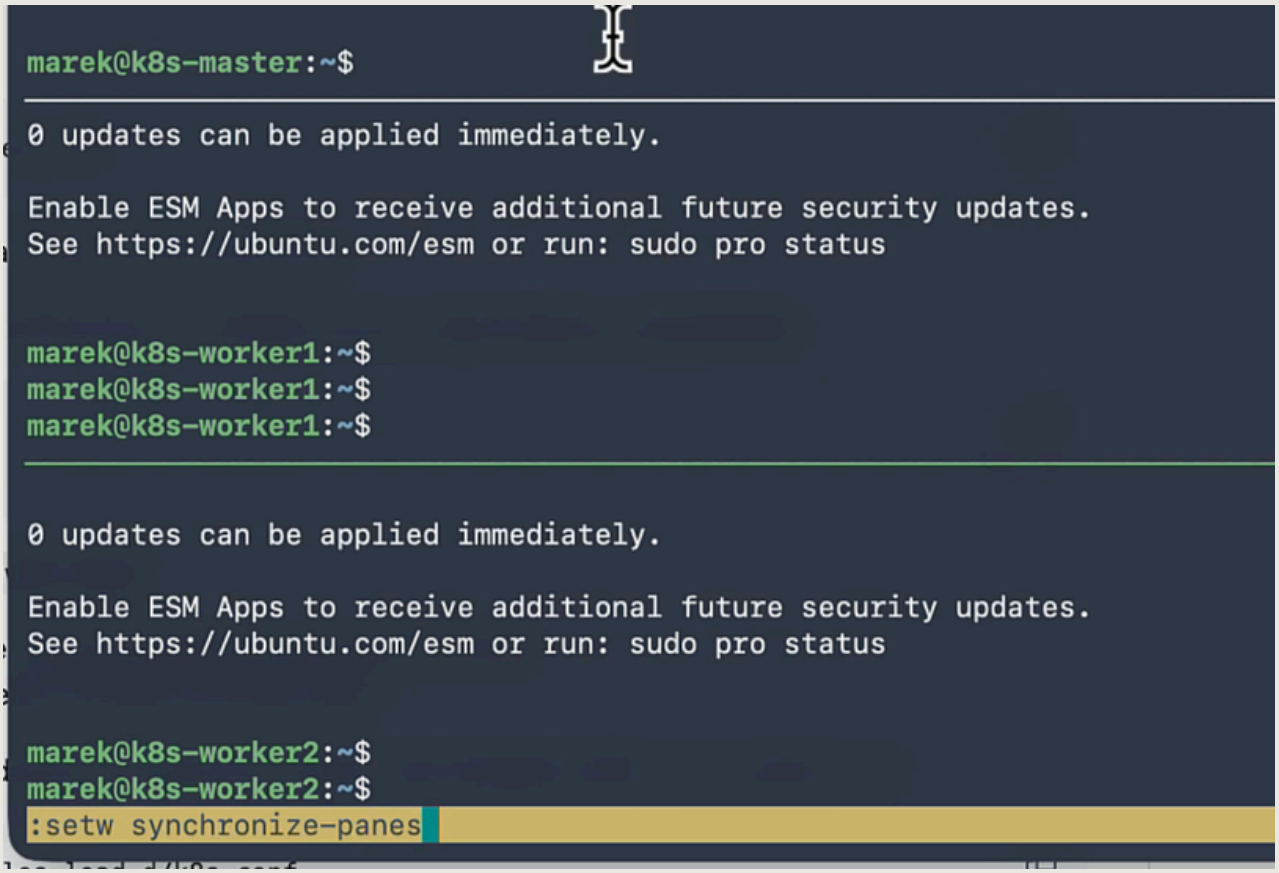

My master node will get ip address 192.168.1.191, and worker nodes 192.168.1.192 and .193.
Then we run *sudo apt update && sudo apt update -y* and reboot each VM:



Then for next step we can use TMUX to run commands on all VMs at the same time.
On Mac you install TMUX with *brew install tmux*.
Some tmux commands:

^*b* + *%* - will split screen vertically ( so press ctrl + b, release it, then shift + 5 )
^*b* + *"* - split screen horizontally
^*b* + *arrows* - switch between panes on current window
^*b* + *x* - close current session ( warning will pop up asking if you are sure )
^*d* - close current tmux pane without warning .

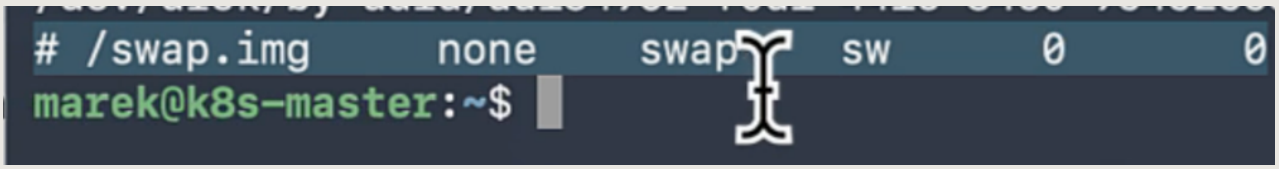
I will use ^b + " twice to split screen horizontally into 3 sections:



We can then run following commands:

*sudo apt install qemu-guest-agent -y*
*sudo swapoff -a*
*sudo nano /etc/fstab*

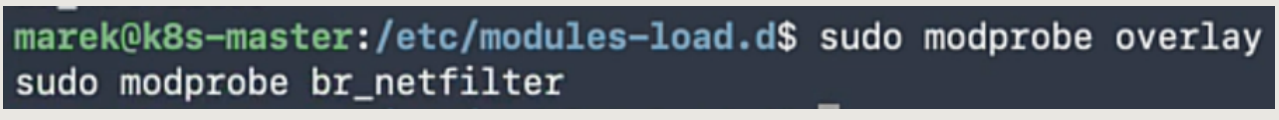
and in that /etc/fstab we comment out the swap.img line.
This is to install qemu agent and disable swap




Next we run this to load kernel modules :

cat <<EOF | sudo tee /etc/modules-load.d/k8s.conf
overlay
br_netfilter
EOF


and we run those commands to load those modules into memory:



Now we run these (still on all 3 virtual machines):

*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*

```
marek@k8s-master:/etc/modules-load.d$ 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
net.bridge.bridge-nf-call-iptables  = 1
net.bridge.bridge-nf-call-ip6tables = 1
net.ipv4.ip_forward                 = 1
```

and run that to apply:
*sudo sysctl --system*
We should see the 3 lines with net.bridge as below:

```
net.ipv4.conf.default.rp_filter = 2
net.ipv4.conf.all.rp_filter = 2
kernel.yama.ptrace_scope = 1
vm.mmap_min_addr = 65536
net.bridge.bridge-nf-call-iptables = 1
net.bridge.bridge-nf-call-ip6tables = 1
net.ipv4.ip_forward = 1
marek@k8s-master:/etc/sysctl.d$
```

Next we install containerd :
*sudo apt install containerd -y*

and create a config file for it:
*sudo mkdir -p /etc/containerd*
*containerd config default | sudo tee /etc/containerd/config.toml > /dev/null*

Then we need to amend one entry in that config file by running:

*sudo sed -i 's/SystemdCgroup = false/SystemdCgroup = true/g' /etc/containerd/config.toml*
After that - you should get output like that:

```
marek@k8s-master:/etc/containerd$ cat config.toml | grep Systemd
            SystemdCgroup = false
marek@k8s-master:/etc/containerd$ sudo sed -i 's/SystemdCgroup = false/SystemdCgroup = true/g' /etc/containerd/config.toml
marek@k8s-master:/etc/containerd$ cat config.toml | grep Systemd
            SystemdCgroup = true
marek@k8s-master:/etc/containerd$
```

Then you enable and restart containerd daemon:
*sudo systemctl enable containerd*
*sudo systemctl restart containerd*

Next we need to update repository so it contains kubernetes repo and installs necessary components:

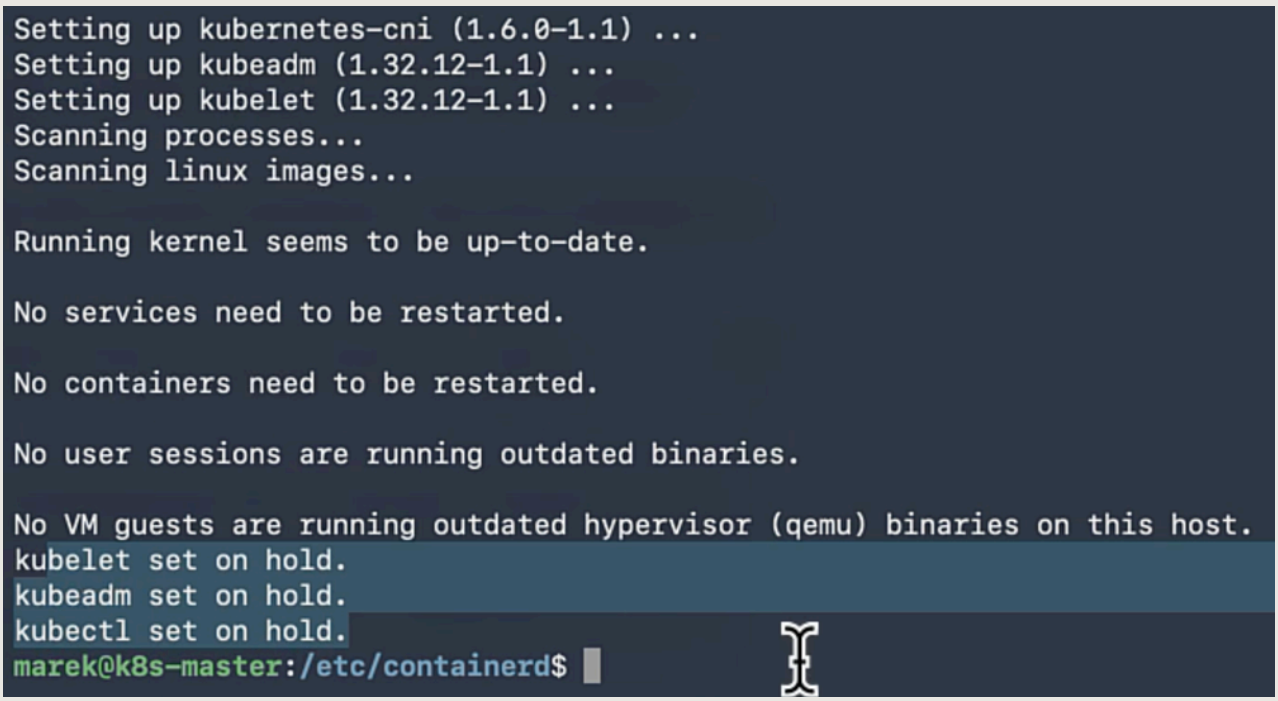
*sudo apt update && sudo apt install -y apt-transport-https ca-certificates curl gpg*

*# Download the public signing key*
*sudo mkdir -p -m 755 /etc/apt/keyrings*
*curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.32/deb/Release.key | sudo gpg --dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg*

*# Add the repository*
*echo 'deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] https://pkgs.k8s.io/core:/stable:/v1.32/deb/ /' | sudo tee /etc/apt/sources.list.d/kubernetes.list*

*sudo apt update*
*sudo apt install -y kubelet kubeadm kubectl*
*sudo apt-mark hold kubelet kubeadm kubectl*

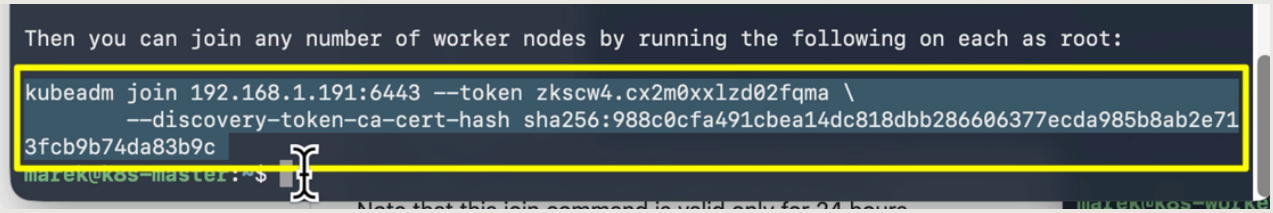
Again - remember that commands are to be pasted from GitHub repo rather than from here.
This process might take a while, but at the end you should see something like that:




Then reboot all 3 virtual machines, and next commands are to be run on master node only:

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

That command will generate another command that can be run on worker nodes to join the cluster:



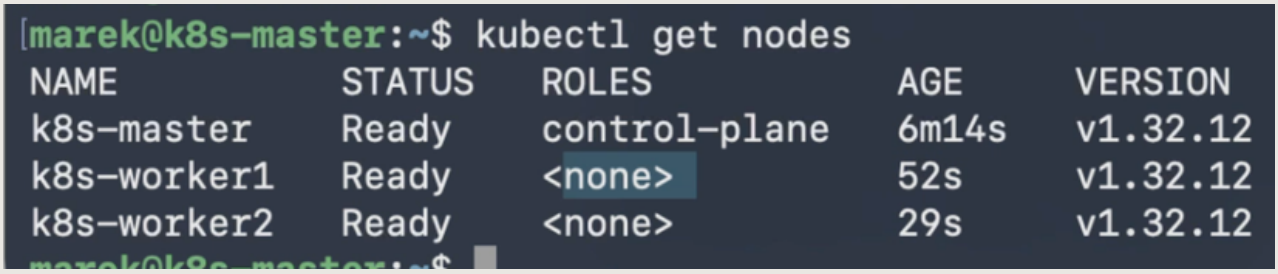
Before we run it on our worker nodes though, there are 2 other things we need to run on our master node:

*mkdir -p $HOME/.kube*
*sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config*
*sudo chown $(id -u):$(id -g) $HOME/.kube/config*

*kubectl apply -f https://github.com/flannel-io/flannel/releases/latest/download/kube-flannel.yml*

That last ‘flannel’ one creates so called CNI - Container Network Interface.
We can now apply that previous kubeadm join command on our worker nodes:

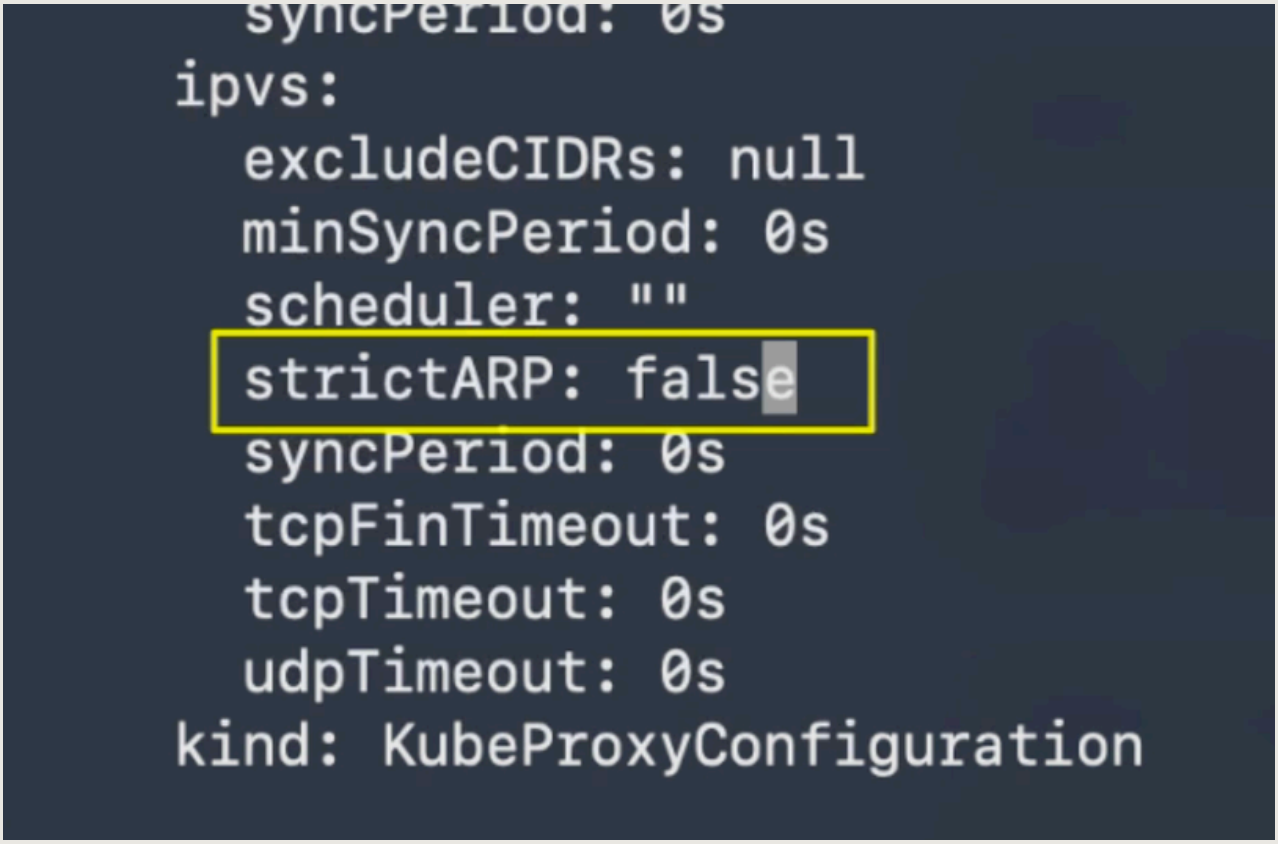

If you now run on master ‘kubectl get nodes’ you should see all master and worker nodes



That’s basically it, however you will not be able to run service type ‘LoadBalancer’.
We need to configure that service separately.
I will use MetalLB as its very popular solution for this purpose.
We first run:

*kubectl edit configmap -n kube-system kube-proxy*

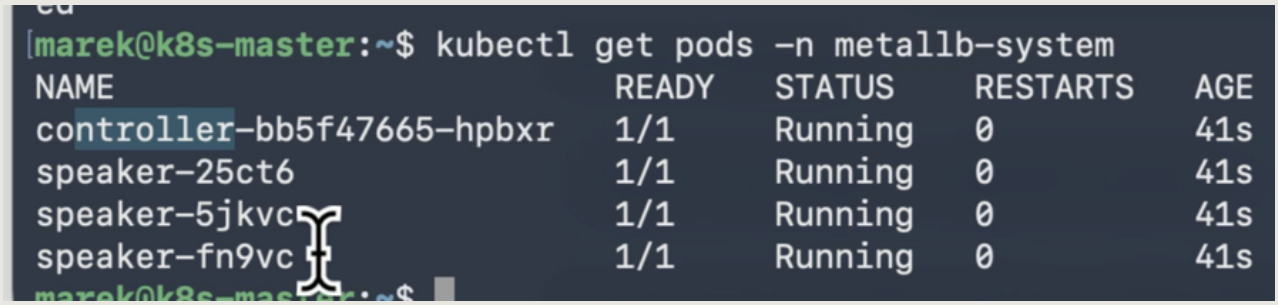
and we need to edit strictARP from false to true in ‘ipvs’ section:



You now apply that MetalLB with:

*kubectl apply -f https://raw.githubusercontent.com/metallb/metallb/v0.14.9/config/manifests/metallb-native.yaml*

and if you run
*kubectl get pods -n metallb-system*
you should see 4 pods running:




This MetalLB needs a pool of static ip addresses though, so we create new file - I called it proxmox-ip-pool.yaml and I paste that:

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  name: marek-pool
  namespace: metallb-system
spec:
  addresses:
  - 192.168.1.240-192.168.1.245 # CHANGE THIS to your desired range
---
apiVersion: metallb.io/v1beta1
kind: L2Advertisement
metadata:
  name: layer2-advert
  namespace: metallb-system
spec:
  ipAddressPools:
  - marek-pool
```

Then we apply that with command:
*kubectl apply -f proxmox-ip-pool.yaml*
and our MetalLB and everything else should be up and running as expected.

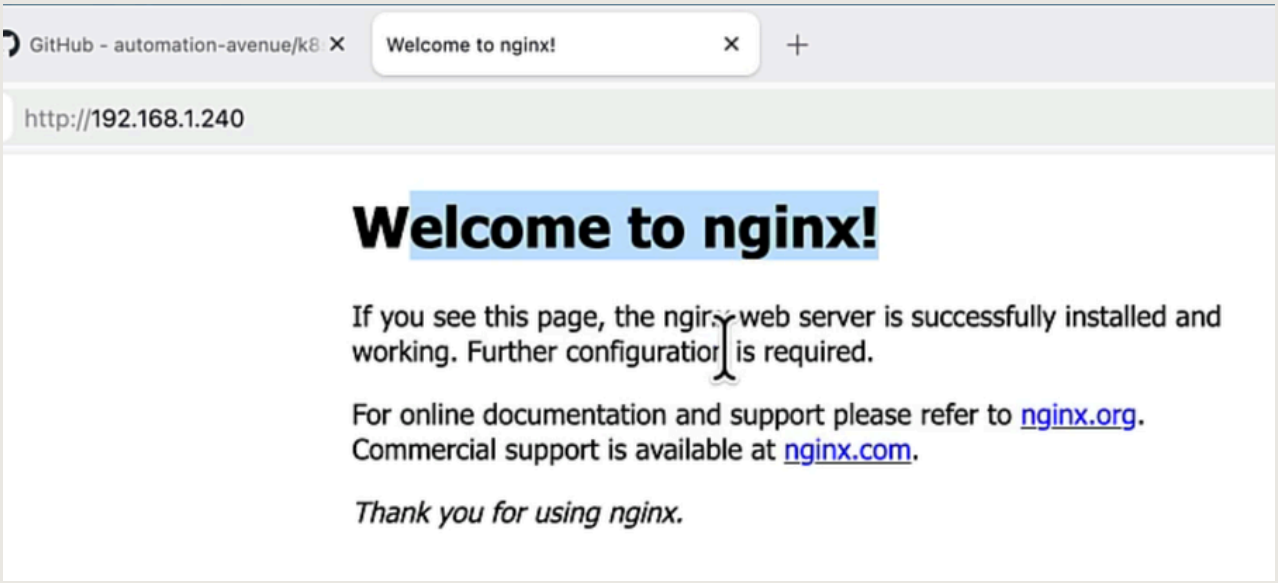
We can start deploying services to our kubernetes cluster.
As a test - lets deploy 2 nginx pods that will be under nginx LoadBalancer service called nginx-service.
We create a yaml file called nginx-test.yaml and we paste that:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-test
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:latest
        ports:
        - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
spec:
  selector:
    app: nginx
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
  type: LoadBalancer
```

Now we deploy it with:
*kubectl apply -f nginx-test.yaml*

and if we run:
*kubectl get svc nginx-service*
we should see output like that:


You can test it by trying to access it from your web browser by going to external ip address (in my case 192.168.1.240) and to external port (in my case 80) to test




That output proves everything works as expected.

Congratulations! You have fully working Kubernetes Cluster on your Proxmox VE :)