Creating the Cluster

Configurations

I created my nodes by making clones from an Cloud-init template of an Ubuntu server that I set up. To check out how I created a template with Ubuntu Cloud Images and cloud-init on Proxmox, check out my ***Proxmox VM Book***.

After getting getting my Ubuntu Servers up and running, installing qemu-guest-agent, and upgrading all packages, I took a snapshot of the VM to have something to rollback to incase I mess up at any point. Taking snapshots is a great practice especially when learning new technologies, as it saves you time from recreating from scratch and allows you to roll back to certain states you save. The following are steps I took on my Master-Node to configure and prepare my nodes for my Kubernetes Cluster.

After establishing my SSH connection to my server, I followed the steps outlined below and ran the corresponding commands to configure my VMs

VM Set Up and Initial Configurations

Disable Swap

Kubernetes does not get along with swap enabled, so I need to disable it. To do this, I ran the command in my terminal window to temporarily disable swap:

```
sudo swapoff -a
```

Next, I ran the command to edit the fstab file and comment out and keep swap turned off even after reboots:

```
sudo nano /etc/fstab
```

To confirm I've done this correctly, run the command to view swap usage:

```
free -m
```

```
austin@k8-n3:~$ free -m
               total        used        free      shared  buff/cache   available
Mem:            3911         165        3558           0         187        3530
Swap:              0           0           0
austin@k8-n3:~$
```

Hostnames Static IP or Static Leases

Your nodes will each require a static IP or Static DHCP lease. I've set up Static Leases for all of my nodes via OPNsense. To assign static IPs directly on the server, you'll need to edit and make changes to the .yaml file in the netplan directory.

You'll also need to make sure each VM node has been assigned a hostname. Run the command to view your hostname:

```
cat /etc/hostname
```

Install Container Runtime (containerd)

Kubernetes requires a container runtime. I'll be using Containerd. To get started with this, update all your packages and then run the following command:

```
sudo apt install containerd
```

Check that the service is running by following command:

```
systemctl status containerd
```

```
austin@k8-n3:~$ systemctl status containerd
● containerd.service - containerd container runtime
     Loaded: loaded (/lib/systemd/system/containerd.service; enabled; vendor preset: enabled)
     Active: active (running) since Thu 2024-04-04 22:09:35 UTC; 34s ago
       Docs: https://containerd.io
    Process: 1096 ExecStartPre=/sbin/modprobe overlay (code=exited, status=0/SUCCESS)
   Main PID: 1098 (containerd)
      Tasks: 8
     Memory: 13.8M
        CPU: 309ms
     CGroup: /system.slice/containerd.service
             └─1098 /usr/bin/containerd
```

After confirming it's running, create a new directory for containerd within /etc by running the following command:

```
sudo mkdir /etc/containerd
```

Next, write the default configuration to containerd by running the following command:

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

Now, use whatever editor you like, I use nano, and edit the config file. Search for runc.options within the file, and change the systemdCgroup value to true:

```
sudo nano /etc/containerd/config.toml
```

Enter Ctrl+W to search within the file for runc.options and change the SystemdCgroup value

```
[plugins."io.containerd.grpc.v1.cri".containerd.runtimes.runc.options]
  BinaryName = ""
  CriuImagePath = ""
  CriuPath = ""
  CriuWorkPath = ""
  IoGid = 0
  IoUid = 0
  NoNewKeyring = false
  NoPivotRoot = false
  Root = ""
  ShimCgroup = ""
  SystemdCgroup = true
```

Systemctl.conf Configuration

Run the following command to edit the sysctl.conf file:

```
sudo nano /etc/sysctl.conf
```

Find the line that enables packet forwarding for IPv4 and uncomment it so it can be read by the system:

```
# Uncomment the next line to enable packet forwarding for IPv4
#net.ipv4.ip_forward=1
```

```
# Uncomment the next line to enable packet forwarding for IPv4
net.ipv4.ip_forward=1
```

This allows your nodes to communicate with each other and is crucial to get your cluster to work

k8s.conf Configuration

The next file to edit is the k8s.conf file. Run the following command:

```
sudo nano /etc/modules-load.d/k8s.conf
```

This is a blank file. Add "br_netfilter" and save the file:

The bridge netfilter essentially ensures that network bridging is supported throughout the cluster. After this is done. reboot your VMs.

Kubernetes Installation

With our VMs running, we can now install Kubernetes packages. To do this, we'll need to add the Kubernetes repository gpg key and then install the repository itself. Then we can run the commands to install kubeadm, kubelet, and kubectl.

Install Kubernetes using Native Package Management

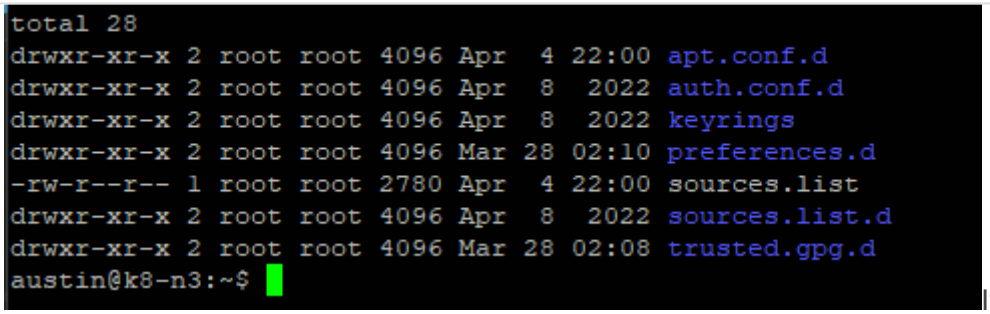
Run the command to install necessary packages in order to use the Kubernetes apt packages:

```
sudo apt-get install -y apt-transport-https ca-certificates curl
```

- You might not have to install these commands, but I ran it as a precautionary measure

Run the command to make sure you have the "keyrings" directory prior to downloading the public signing key:

```
ls -l /etc/apt
```

- 

If you don't have this

  directory create one with the mkdir comamnd

Download the public signing key for Kubernetes package repositories. I'm downloading the most current stable release v1.29:

```
curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.29/deb/Release.key | sudo gpg --dearmor -o
/etc/apt/keyrings/kubernetes-apt-keyring.gpg
```

Run the command to add the Kubernetes apt repositories:

```
echo 'deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] https://pkgs.k8s.io/core:/stable:/v1.29/deb/
/' | sudo tee /etc/apt/sources.list.d/kubernetes.list
```

Run the commands to update and upgrade apt repository and packages, and then install Kubernetes:

```
sudo apt-get update && sudo apt-get upgrade
sudo apt-get install -y kubectl kubeadm kubelet
```

Create Worker Template Node (optional)

At this stage. we now have a node that is properly configured and has the Kubernetes packages installed. I created a template of this VM so that in the future, I can add nodes to any cluster much faster and skip all of this configuration and initial set up. It's relatively simple to do this in Proxmox, but first, we'll want to clean up our VM so that configurations like static assignments and machine-id won't get cloned. Run the following commands to do so:

First, clean cloud-init with the following command:

```
sudo cloud-init clean
```

Remove the instances in the cloud repository by running the following command:

```
sudo rm -rf /var/lib/cloud/instances
```

Next, reset the machine-id (this avoids having your clones use the same static IP) by running the following command:

```
sudo truncate -s 0 /etc/machine-id
```

Remove the machine-id in the dbus directory by running the following command, and then create a symbolic link by running the following commands:

```
sudo rm /var/lib/dbus/machine-id
sudo ln -s /etc/machine-id /var/lib/dbus/machine-id
```

You can confirm this is done with ls -l and then power off this VM and and convert it to a template.

Initialize Pod Network on your Controller Node

From my template, I've created 4 clones:

- k-ctrlr
- k8-n1
- k8-n2
- k8-n3

My cluster will have 1 Controller node, and 3 worker nodes. The specs for my nodes are as follows:

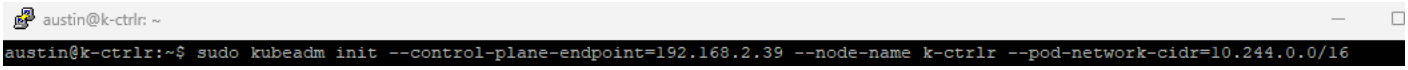
- Controller
  - 4 Gb RAM
  - 4 CPU Cores
- Worker
  - 2 GB RAM
  - 2 CPU Cores

After powering up the node vms, ensure they have static IPs, or assign Static leases in your router (OPNsense in my case).

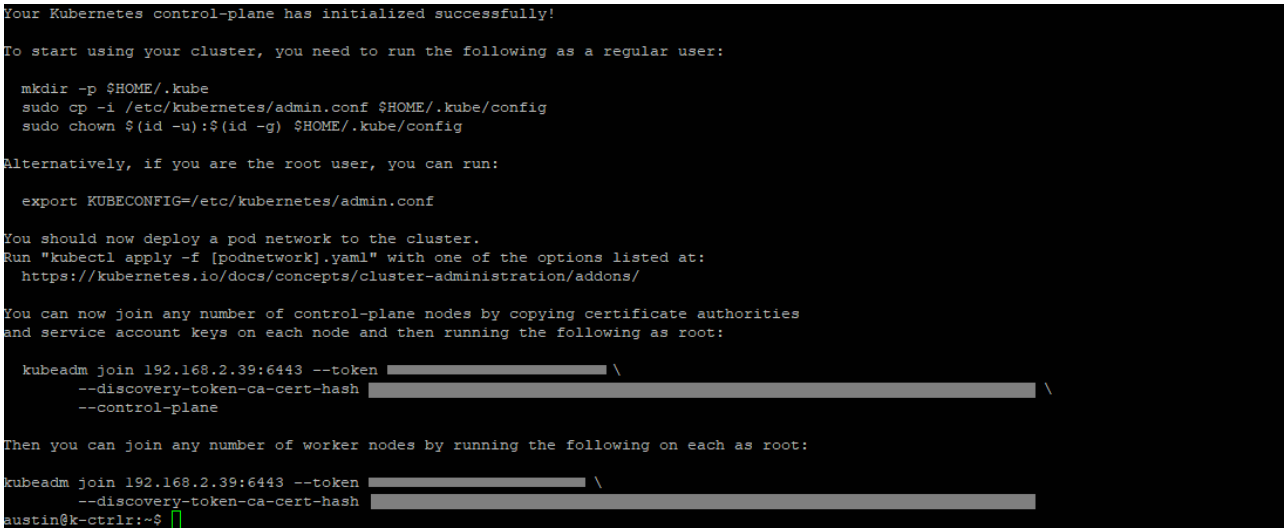
To initialize the following command, run the following command after editing certain parameters:

```
sudo kubeadm init --control-plane-endpoint=192.168.2.39 --node-name k-ctrlr --pod-network-cidr=10.244.0.0/16
```

- For --control-plan-endpoint, edit this to make it your Controller Nodes IP. In my case, I've assigned k-ctrlr 192.168.2.39
- For --node-name, enter your Controller Node's hostname, in my case it is k-ctrlr
- Leave pod network the same. Changing this is possible, but will require additional configurations.



After initialization, you'll see the following output, which contains commands including keys and tokens to add nodes to the cluster:



I've greyed out my tokens and hash value. Save these in a .txt file or somewhere accessible as we'll be using this in the near future to join our other 3 nodes to the cluster.

To complete this initialization and to allow yourself to control your cluster as a regular user run the following commands:

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

After doing this, your cluster has been initialized and is almost ready to add nodes.

Overlay Network

After you initialize your Controller node, run the command to view your pods:

```
kubectl get pods --all-namespaces
```

You'll notice that the coreDNS is pending. Kubernetes clusters require a Container Network Interface (CNI) based Network Add On. I used flannel, but there are plenty of options to choose from and can be found in the Kubernetes Install Docs.

To install and apply the add on run the following command:

```
# kubectl apply -f <add-on.yaml> is the command
# I ran the command below to install Flannel
kubectl apply -f https://github.com/flannel-io/flannel/releases/latest/download/kube-flannel.yml
```

If using flannel, check out their github repo and the ReadME.txt for more installation instructions.

Adding Nodes to Cluster

When adding nodes to your cluster, you'll need the tokens saved during initialization to run commands in your worker nodes windows. If you don't have these, you can generate new tokens with the following command:

```
kubeadm token create --print-join-command
```

- If joining nodes after 24 hours since token creation, you'll need to generate new tokens

Run the following command in each of your Node terminal windows:

```
sudo kubeadm join 192.168.2.39:6443 --token entertokenhere --discovery-token-ca-cert-hash hashvaluehere
```

- Replace 192.168.2.39 with your controller nodes Static IP
- Replace entertokenhere with your token you generated
- Replace hashvaluehere with the sha256 value you generated

The node will run some checks and will then join the cluster:

```
[preflight] Running pre-flight checks
[preflight] Reading configuration from the cluster...
[preflight] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubead
m-config -o yaml'
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubead
m-flags.env"
[kubelet-start] Starting the kubelet
[kubelet-start] Waiting for the kubelet to perform the TLS Bootstrap...

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.

austin@k8-n3:~$
```

To confirm it's joined your cluster and check your node statuses, run the following command in your Controller Node:

```
kubectl get nodes
```

```
austin@k-ctrlr:~$ kubectl get nodes
NAME      STATUS   ROLES           AGE     VERSION
k-ctrlr   Ready    control-plane   16h     v1.29.3
k8-n1     Ready    <none>          16h     v1.29.3
k8-n2     Ready    <none>          4m54s   v1.29.3
k8-n3     Ready    <none>          4m43s   v1.29.3
austin@k-ctrlr:~$
```

- You can now see the Name, Status, Role, Age, and Kubernetes version being run

Your cluster is now created and ready for you to launch and work with whatever services you'd like! Check out the next page to see how I connected this cluster to my Portainer docker container!

Revision #11
Created 4 April 2024 02:15:38 by Austin
Updated 5 April 2024 02:43:50 by Austin