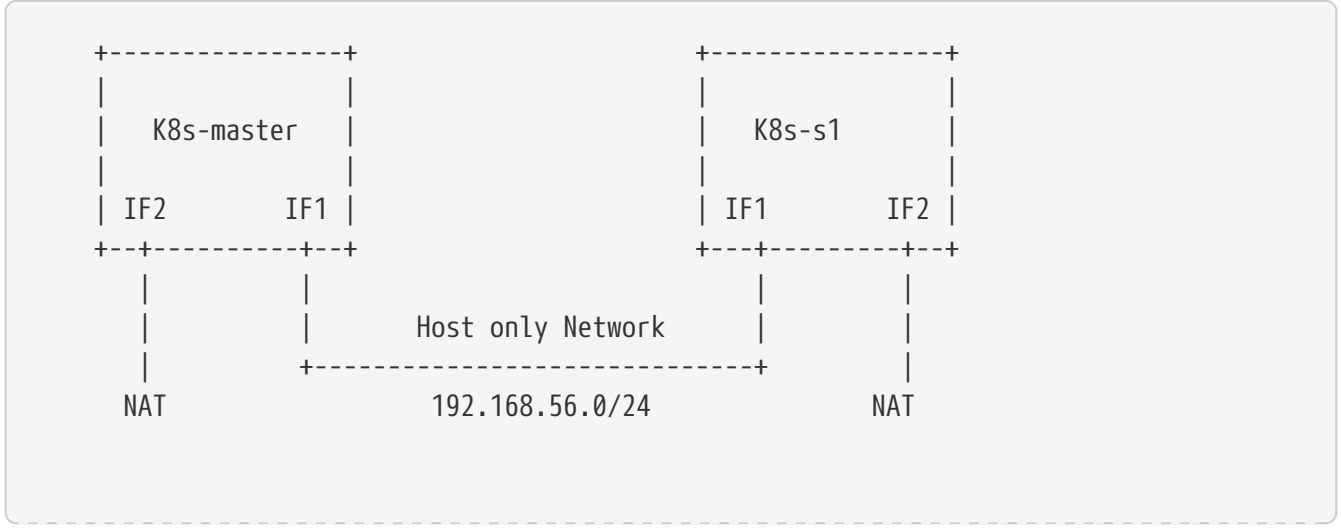
Kubernetes MultiNode cluster on VirtualBox

Table of Contents

A step by step description of bringing up kubernetes cluster using VirtualBox 6.0 with ubuntu 18.04 server.

VirtualBox setup

Create two VMs k8s-master and k8s-worker as shown in below diagram.



VM hardware configuration.

- Kubernetes master/controller node hardware requirement

Component	Value (min)
Processor	2 core
RAM	4GB
Network Adapter-1	Host only network
Network Adapter-2	NAT

- kubernetes worker/slave node hardware requirement

Component	Value (min)
Processor	1 core
RAM	2GB
Network Adapter-1	Host only network
Network Adapter-2	NAT

Prepare Servers

Execute the below step in both the VMs.

Setup network.

- Shutdown the VMs.
- Modify VM network, with Adapter 1 for Host Only Network and Adapter 2 for NAT.
- Enable Nested Virtualization

```
$ VBoxManage modifyvm <vm-name> --nested-hw-virt on
```

Example

```
$ VBoxManage modifyvm k8s-master --nested-hw-virt on
$ VBoxManage modifyvm k8s-qoekwe --nested-hw-virt on
```

Nested Virtualization is supported for Intel i7 7Gen+ processors

- Start VM.

- Execute below command and note down the interfaces (generally for Virtual box VMs network interface will start with enp0s).

```
$ sudo ifconfig -a
```

- Setup static network.

  Open file `/etc/netplan/00-installer-config.yaml` and add below configuration.

  k8s-master

```
network:
 ethernets:
  enp0s3:
   addresses: [192.168.56.10/24]
   dhcp4: false
  enp0s8:
   addresses: []
   dhcp4: true
 version: 2
```

  k8s-worker

```
network:
 ethernets:
  enp0s3:
   addresses: [192.168.56.12/24]
   dhcp4: false
  enp0s8:
   addresses: []
   dhcp4: true
 version: 2
```

- Update `/etc/hosts` such that command `hostname -i` outputs ip which is routable from other VM.

  example(master)

```
192.168.56.10 k8s-master
192.168.56.12 k8s-worker
```

  example(worker)

```
192.168.56.10 k8s-master
192.168.56.12 k8s-worker
```

This step is required since we've chosen host-only adapter and our default route is provided by NAT network, if we choose Bridge network this step can

> be skipped.

- Reboot VMs.
- Update Packages.
  - Update apt sources.

```
$ sudo apt update
```

  - Install packages

```
$ sudo apt upgrade
```

Install docker

- Install utility packages

```
$ apt-get install -y \
    apt-transport-https \
    ca-certificates \
    curl \
    gnupg-agent \
    software-properties-common
```

- Add GPG key

```
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
```

- Add docker repository to apt sources.

```
$ sudo add-apt-repository \
   "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
   $(lsb_release -cs) \
   stable"
```

- Update apt sources and install docker

```
$ sudo apt update ; sudo apt install -y docker-ce docker-ce-cli containerd.io
```

- Verify docker service is started.

```
$ sudo systemctl status docker
```

Install and configure kubernetes.

Prepare server

Execute the below steps on both the VMs.

1. Letting iptables see bridged traffic

```
cat <<EOF | sudo tee /etc/sysctl.d/98-kubernetes-cni.conf
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
EOF
```

Make sure that the br_netfilter module is loaded before this step. This can be done by running lsmod | grep br_netfilter. To load it explicitly call sudo modprobe br_netfilter.

2. Reload configuration

```
sudo sysctl --system
```

3. Add Kubernetes source gpg.

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

4. Next add kubernetes repository

```
$ sudo apt-add-repository "deb http://apt.kubernetes.io/ kubernetes-xenial main"
```

Browse http://apt.kubernetes.io/ and search for the corresponding ubuntu version, since there was no kubernetes-bionic, installing kubernetes-xenial in my case.

5. Install `kuberenetes` packages

```
$ sudo apt update; sudo apt install -y kubeadm kubelet kubectl
```

6. Prevent automatic update of pages

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

7. Turn off swap

```
$ sudo swapoff -a
```

8. Turn off swap permanently in `/etc/fstab` by executing belo command.

```
$ sudo sed -i '/swap/s/^\(.*\)$/#\1/g' /etc/fstab
```

9. Add private ip to kubelet. Create file /etc/default/kubelet and following content in the file.

```
KUBELET_EXTRA_ARGS=--node-ip=192.168.56.10
```

Change ip address as per master and worker node ips

10. Configure docker `daemon.json`.

```
cat <<EOF | sudo tee /etc/docker/daemon.json
{
  "exec-opts": ["native.cgroupdriver=systemd"],
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "100m"
  },
  "storage-driver": "overlay2"
}
EOF
```

11. Install ipvs, we shall be using ipvs for LB.

```
sudo apt install -y ipvsadm
```

12. configure ipvsadm

```
cat <<EOF | sudo tee /etc/sysctl.d/98-ipvsadm.conf
net.ipv4.ip_forward=1

net.ipv4.conf.all.arp_ignore=1
net.ipv4.conf.all.arp_announce=2
EOF
```

13. update ipvsadm configuration

```
cat <<EOF | sudo tee  /etc/default/ipvsadm
# ipvsadm

# if you want to start ipvsadm on boot set this to true
AUTO="true"

# daemon method (none|master|backup)
DAEMON="master"

# use interface (eth0,eth1...)
IFACE="enp0s3"

# syncid to use
# (0 means no filtering of syncids happen, that is the default)
# SYNCID="0"
EOF
```

Change the network interface accordingly

14. Reload configuration

```
sudo sysctl --system
```

15. Load modules

```
sudo modprobe -- ip_vs
sudo modprobe -- ip_vs_rr
sudo modprobe -- ip_vs_wrr
sudo modprobe -- ip_vs_sh
sudo modprobe -- ip_vs_sed
sudo modprobe -- ip_vs_lblc

sudo lsmod | grep -e ip_vs -e  nf_conntrack
```

16. Reload configuration and restart `docker` and `kubelet` service.

```
sudo systemctl daemon-reload
sudo systemctl restart docker
sudo systemctl restart kubelet
sudo systemctl restart ipvsadm
```

17. **(Optional)**Reboot VMs.

Configure Master VM.

1. Create `kubeadm-config.yaml` file that consists of cluster configuration.

```
$ cat <<EOF  | tee kubeadm-config.yaml
apiVersion: kubeadm.k8s.io/v1beta2
bootstrapTokens:
- groups:
  - system:bootstrappers:kubeadm:default-node-token
  ttl: 24h0m0s
  usages:
  - signing
  - authentication
kind: InitConfiguration
localAPIEndpoint:
  advertiseAddress: 192.168.56.10
  bindPort: 6443
---
apiServer:
  timeoutForControlPlane: 4m0s
apiVersion: kubeadm.k8s.io/v1beta2
certificatesDir: /etc/kubernetes/pki
clusterName: kubernetes
#controlPlaneEndpoint: 192.168.1.6:6443
controllerManager: {}
dns:
  type: CoreDNS
etcd:
  local:
    dataDir: /var/lib/etcd
imageRepository: k8s.gcr.io
kind: ClusterConfiguration
#kubernetesVersion: v1.18.0
networking:
  dnsDomain: cluster.local
  serviceSubnet: 10.96.0.0/12
  podSubnet: 10.244.0.0/16
scheduler: {}
FeatureGates:
  ServiceTopology: true
  TopologyManager: true
---
apiVersion: kubeproxy.config.k8s.io/v1alpha1
bindAddress: 0.0.0.0
ipvs:
  scheduler: "lblc"
  strictARP: true
kind: KubeProxyConfiguration
mode: "ipvs"
```

```
EOF
```

If decide to change pod network, remember to change the pod network in `flannel.yaml` while setting up pod network in below step.

You can geneate the above configuration file using below command

```
$  sudo kubeadm config print init-defaults  --component-configs
KubeProxyConfiguration
```

2. Bring up cluster now.

```
$ sudo kubeadm init --config kubeadm-config.yaml
```

3. As output of above command shows to execute below command, run below command to update auth details for `kubectl` command.

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

4. Note down (copy to notepad) the `kubeadm` join command which needs to be executed on worker nodes to join to the cluster.
5. Install pod network.
   a. If you are using bridge network which has default route then you can directly install flannel using below command. However if you had given host-only adapter IP address for API server. then skip this step and follow next steps.

```
$ kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml
```

   b. Download flannel.yaml

```
$ wget https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml
```

   c. Open `kube-flannel.yaml` file search for daemonset kind and add additional arugment `--iface=<host-only-adaper-interface>`, note there are multiple daemonset definitions, either update all of them or execute `kubectl get nodes --show-labels` and get the arch and update corresponding arch daemonset only.

```
      containers:
      - name: kube-flannel
        image: quay.io/coreos/flannel:v0.11.0-amd64
```

```
        command:
        - /opt/bin/flanneld
        args:
        - --ip-masq
        - --kube-subnet-mgr
        - --iface=enp0s3 ①
        resources:
          requests:
            cpu: "100m"
            memory: "50Mi"
          limits:
```

① Added my host-only adapter here

d. if you had changed pod network while executing `kubeadm init`, replace `10.244.0.0/16` which chosen pod network above in `net-conf.json` section.

e. Now Create pod netwrok by applying updated yaml file

```
$ kubectl apply -f kube-flannel.yml
```

6. Verify all necessary pods are started

```
$ kubectl get pods -A
```

output:

```
NAMESPACE     NAME                                READY   STATUS    RESTARTS   AGE
kube-system   coredns-86c58d9df4-c68gd            1/1     Running   0          6m41s
kube-system   coredns-86c58d9df4-q5bht            1/1     Running   0          6m41s
kube-system   etcd-k8s-master                     1/1     Running   0          6m6s
kube-system   kube-apiserver-k8s-master           1/1     Running   0          5m59s
kube-system   kube-controller-manager-k8s-master  1/1     Running   0          5m56s
kube-system   kube-flannel-ds-amd64-stb29         1/1     Running   0          49s
kube-system   kube-proxy-882ms                    1/1     Running   0          6m41s
kube-system   kube-scheduler-k8s-master           1/1     Running   0          5m54s
```

7. We can check if master node is ready or not by executing below command.

```
$ kubectl get nodes
```

Joining worker node to cluster.

1. Now go to worker node and execute the join command previously saved when you were executing kubeadm on master.

```
$  sudo kubeadm join 192.168.56.10:6443 --token t0j1zi.v5lojsnpjh9r0rbn \
      --discovery-token-ca-cert-hash sha256:40b1142d9002003ab5b085776b8b8cba4a41ceaafab06429c49eaedc2b2939fa
```

The above command is sample, the values are dynamically generated.

2. Now go back to master and execute the below command, you should be able to see slave node added.

```
$ kubectl get nodes
```

output:

```
NAME            STATUS   ROLES    AGE     VERSION
k8s-master      Ready    master   56m     v1.18.4
k8s-worker      Ready    <none>   2m49s   v1.18.4
```

Install Helm v3

Helm package manager consists of helm client [Helm v3 deprecated need for server side component(tiller) which was running as pod on k8s]. We will install helm client on our master server itself.

Install helm client.

1. Install helm client using script on master server using below set of commands.

```
$ curl -fsSL -o get_helm.sh
https://raw.githubusercontent.com/helm/helm/master/scripts/get-helm-3
$ chmod 700 get_helm.sh
$ ./get_helm.sh
```

2. Verify Helm is running

```
$ helm list
```

output

```
NAME    NAMESPACE       REVISION        UPDATED STATUS  CHART   APP VERSION
```

Configure Kubernetes Dashboad (v2).

Install Dashboard

1. Add dashboard repository.

```
$ helm repo add kubernetes-dashboard https://kubernetes.github.io/dashboard/
```

2. Update dashboard repository.

```
$ helm repo update
```

3. Install dashboard.

```
helm install kubernetes-dashboard \
   kubernetes-dashboard/kubernetes-dashboard \
   --namespace kube-system \
   --set fullnameOverride=kubernetes-dashboard \
   --set serviceAccount.name=admin-user \
   --set metricsScraper.enabled=true \
   --set service.type=NodePort
```

By default dashboard is not recommanded to be accessed from outside the VM, if you are using ubuntu desktop you can run below command and access the dashboard using proxy at url `http://localhost:8001/api/v1/namespaces/kube-system/services/https:kubernetes-dashboard:/proxy/` however we will be choosing `NodePort` to access dashboard using host vm's IP as we set `service.type=NodePort`.

+

```
$ kubectl proxy
```

4. Wait till dashboard pod is running.

```
$ kubectl get pods -n kube-system -w
```

output:

```
NAMESPACE     NAME                                    READY   STATUS    RESTARTS   AGE
kube-system   coredns-86c58d9df4-c68gd                1/1     Running   0          11m
kube-system   coredns-86c58d9df4-q5bht                1/1     Running   0          11m
kube-system   etcd-k8s-master                         1/1     Running   0          10m
kube-system   kube-apiserver-k8s-master               1/1     Running   0          10m
kube-system   kube-controller-manager-k8s-master      1/1     Running   0          10m
kube-system   kube-flannel-ds-amd64-stb29             1/1     Running   0          5m18s
kube-system   kube-proxy-882ms                        1/1     Running   0          11m
```

```
kube-system   kube-scheduler-k8s-master              1/1     Running   0          10m
kube-system   kubernetes-dashboard-57df4db6b-5phx2   1/1     Running   0          35s
```

5. Execute the below command and note down the port on which dashboard can be accessed

```
$ kubectl get service --all-namespaces
```

output:

```
NAMESPACE     NAME                   TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
default       kubernetes             ClusterIP   10.96.0.1        <none>        443/TCP          21m
kube-system   kube-dns               ClusterIP   10.96.0.10       <none>        53/UDP,53/TCP    21m
kube-system   kubernetes-dashboard   NodePort    10.110.253.116   <none>        443:32608/TCP    10m
```

The dashboard port is **32608** in my case.

6. Now we can access dashboard at URL. https://192.168.56.10:32608

Port will be dynamically generated and port should be replaced from step 5.

7. Add admin-user to cluster role by editing `kubectl edit clusterrolebinding cluster-admin -n kube-system` and add code snippet to subjects section.

```
- kind: ServiceAccount
  name: admin-user
  namespace: kubernetes-dashboard
```

8. Generate the Bearer Token to access Dashboard

```
$ kubectl -n kube-system describe secret $(kubectl -n kube-system get secret | \
      grep admin-user | \
      awk '{print $1}')
```

output:

```
Name:         admin-user-token-4nwz2
Namespace:    kube-system
Labels:       <none>
Annotations:  kubernetes.io/service-account.name: admin-user
              kubernetes.io/service-account.uid: a1e3ca50-1dab-11e9-9d52-080027aba7cb

Type:  kubernetes.io/service-account-token

Data
====
ca.crt:     1025 bytes
namespace:  11 bytes
token:
eyJhbGciOiJSUzI1NiIsImtpZCI6IiJ9.eyJpc3MiOiJrdWJlcm5ldGVzL3NlcnZpY2VhY2NvdW50Iiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3V
```

```
udC9uYW1lc3BhY2UiOiJrdWJlLXN5c3RlbSIsImt1YmVybmV0ZXMuaW8vc2VydmljZWFjY291bnQvc2VjcmV0Lm5hbWUiOiJhZG1pbi1c2VyLXRva2V
uLTRud3oyIiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9zZXJ2aWNlLWFjY291bnQubmFtZSI6ImFkbWluLXVzZXIiLCJrdWJlcm5ldGVzLml
vL3NlcnZpY2VhY2NvdW50L3NlcnZpY2UtYWNjb3VudC51aWQiOiJhMWUzY2E1MC0xZGFiLTExZTktOWQ1Mi0wODAwMjdhYmE3Y2IiLCJzdWIiOiJzeXN
0ZW06c2VydmljZWFjY291bnQ6a3ViZS1zeXN0ZW06YWRtaW4tdXNlciJ9.YHRkrY1dPsrf1N4LU6qGqCPPl617faeBbHelJAdWXD3TvvZMYnQdMvZuWt
FZjVMxXPdgXDud17eCffDXBg5bRAs1sxd7B37IbXVULrYFoMR-
B0MjOa3eLx1edO_gvE6ZqpyPpdWxC0hWYI0P9cQ78oyZEZ0RDNctTus0qRpVrHpP5ZIMhfRPknV8zxxF-zGf8Xg8ni1NxUOHHB-
DYO1T6gd4v65JgD2ohLS4N9rLpq_MrA7nc13R4sE6zDIgYi5V7kZYz0Zx72qAaV4oOGMDTr0FPP7q3m9SrH8uO3UOUe9tkp_ce8-
7V9hJW8AbPHu3rLNBw2dOGnOk59yNe3jv5w
```

Copy the token and paste it into token field in the URL to Dashboard and login to dashboard.

Due to security prevention UI can be accessed only via `FireFox`.

Installing Metrics server.

1. Add stable repository

```
$ helm repo add stable https://kubernetes-charts.storage.googleapis.com
```

2. Update helm repository

```
$ helm repo update
```

3. Execute below helm command to install metrics server.

```
$ helm install metrics-server stable/metrics-server \
     --namespace kube-system \
     --set args="{--logtostderr,--kubelet-insecure-tls,--kubelet-preferred-address-
types=InternalIP,Hostname,InternalDNS,ExternalDNS,ExternalIP}"
```

4. Wait for pod to come up after that you can execute top command to get resource usage.

```
$ kubectl top nodes
```

5. Alternatively instead of using helm, download latest released component.yaml from metrics-server github release page and then add following two args to metrics-server container[in args section after below secure-port=4443]. and finally install using `kubectl apply -f component.yaml'

```
- --kubelet-insecure-tls
- --kubelet-preferred-address
-types=InternalIP,Hostname,InternalDNS,ExternalDNS,ExternalIP
```

Installing Traefik Ingress Controller.

1. Add traefik repository

```
$ helm repo add traefik https://containous.github.io/traefik-helm-chart
```

2. Update helm repository

```
$ helm repo update
```

3. Execute below helm command to install traefik.

```
$ helm install traefik traefik/traefik --namespace kube-system
```

Installing MetalLB.

1. Execute following commands, This will deploy MetalLB to your cluster, under the metallb-system namespace

```
$ kubectl apply -f
https://raw.githubusercontent.com/metallb/metallb/v0.9.3/manifests/namespace.yaml
$ kubectl apply -f
https://raw.githubusercontent.com/metallb/metallb/v0.9.3/manifests/metallb.yaml
$ kubectl create secret generic -n metallb-system memberlist --from
-literal=secretkey="$(openssl rand -base64 128)"
```

2. Create L2 configuration into config-map file.

```
cat <<EOF | tee metallb-l2-config.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: metallb-system
  name: config
data:
  config: |
    address-pools:
    - name: default
      protocol: layer2
      addresses:
      - 192.168.56.240-192.168.56.250
EOF
```

3. Apply the configuration

```
kubectl apply -f metallb-l2-config.yaml
```

4. Now if you can list service and notice that `external ip` field has ip address set from above pool. This can be noticed for traefik service.

```
kubectl get svc -n kube-system
```

Validation.

You can deploy a sample angular application uploaded here

Troubleshoot.

https://kubernetes.io/docs/tasks/administer-cluster/dns-debugging-resolution/ - DNS not resolving

https://kubernetes.io/docs/setup/independent/troubleshooting-kubeadm/ - kubeadm

https://github.com/kubernetes/kubernetes/tree/master/pkg/proxy/ipvs - ipvs

References

https://github.com/helm/charts/tree/master/stable/traefik - traefik

https://github.com/helm/charts/tree/master/stable/nginx-ingress - nginx

https://kubernetes.io/docs/concepts/services-networking/ingress/ - ingress controller

https://metallb.universe.tf - metallb