Deploying a self-managed cluster in Kubernetes using Kubeadm- by Atush

Login to Azure portal

Create a resource Group via Azure CLI prompt or cloud shell

$ az group create --name k8s-lab-rg3 --location eastus

$ az network vnet create --name k8s-lab-vnet --resource-group k8s-lab-rg3 --location eastus --address-prefixes 172.10.0.0/16 --subnet-name k8s-lab-net1 --subnet-prefixes 172.10.1.0/24

Create the instances, one master and three worker nodes.

Set the variables in Azure CLI prompt or cloud shell

RG=k8s-lab-rg3

LOCATION=eastus

SUBNET=$(az network vnet show --name k8s-lab-vnet -g $RG --query subnets[0].id -o tsv)

Create a master VM (note secret has been already generated for provisioning Linux VMs earlier)

Please follow link to generate ssh keys https://docs.microsoft.com/en-us/azure/virtual-machines/linux/mac-create-ssh-keys

```
$ az vm create --name kube-master \
  --resource-group $RG \
  --location $LOCATION \
  --image UbuntuLTS \
  --admin-user azureuser \
  --ssh-key-values ~/.ssh/id_rsa.pub \
  --size Standard_DS2_v2 \
  --data-disk-sizes-gb 10 \
  --subnet $SUBNET \
  --public-ip-address-dns-name kube-master-lab
```

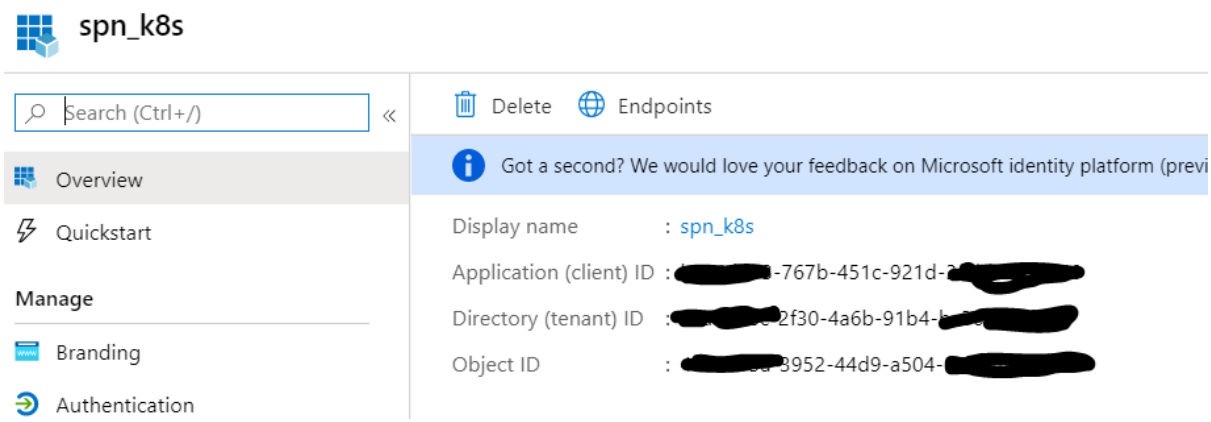
Create Availability set in Azure for worker nodes

```
$ az vm availability-set create --name kubeadm-nodes-as --resource-group $RG
```

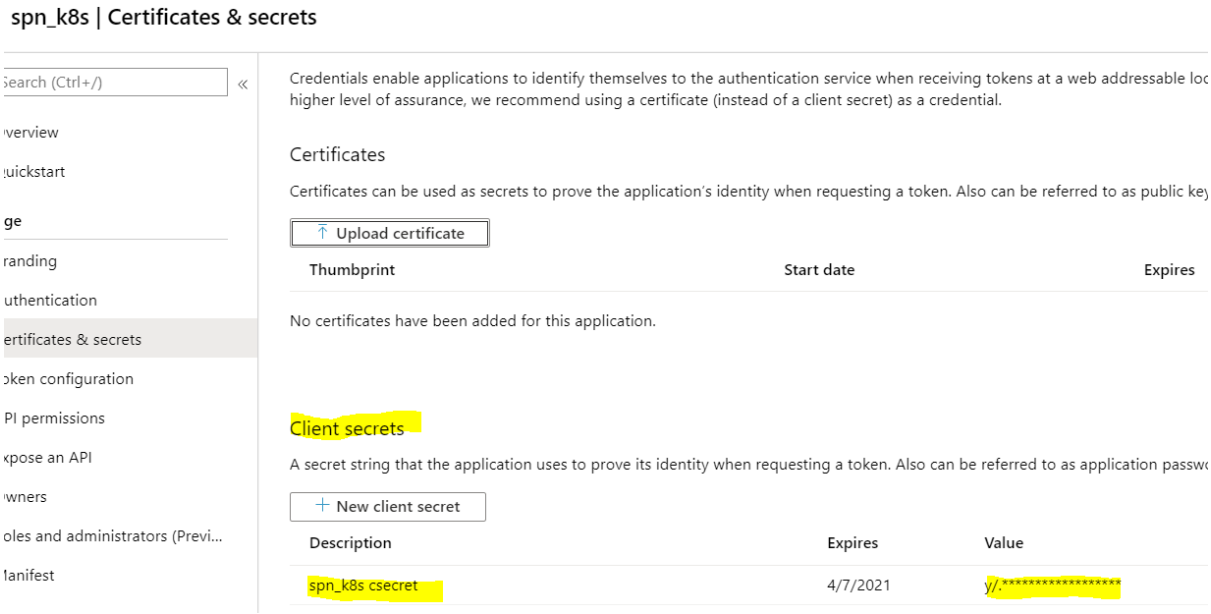
Create three worker nodes

```
$ for i in 0 1 2 ; do
  echo "Creating Kubernetes Node ${i}"
  az vm create --name kube-node-${i} \
    --resource-group $RG \
    --location $LOCATION \
    --availability-set kubeadm-nodes-as \
    --image UbuntuLTS \
    --admin-user azureuser \
    --ssh-key-values ~/.ssh/id_rsa.pub \
    --size Standard_DS2_v2 \
    --data-disk-sizes-gb 10 \
    --subnet $SUBNET \
    --public-ip-address-dns-name kube-node-lab-${i}
Done
```

Create a service principal in your Azure subscription and note down below things to be used in /etc/kubernetes/cloud.conf

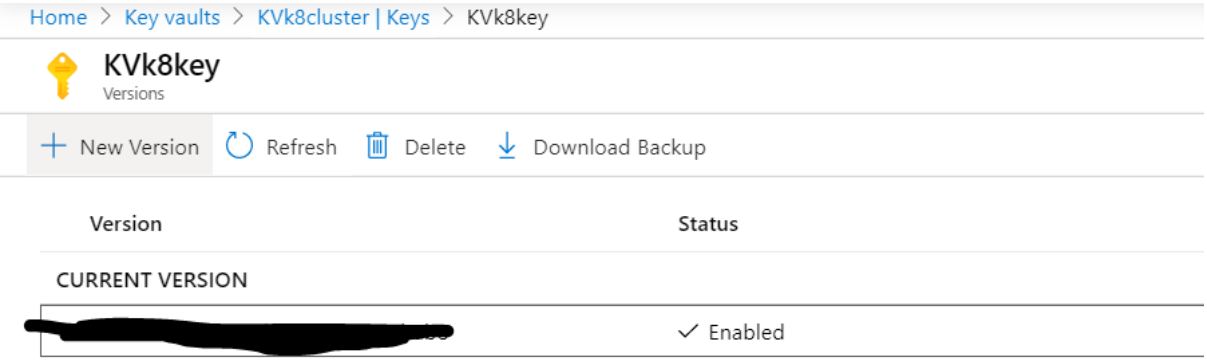


Create a client secret



Note down: - Client ID, Tenant id, Subscription id, Client secret. Use the same in cloud.conf file in later steps.

Create a Key vault to be used in cloud.conf file in later steps



Note down, KV name, KV key name, and the current version being used.

Prepare the Kubernetes cluster master and node instances

- Install docker

$ sudo apt-get update

Add Docker's official GPG key

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

Verify that you now have the key with the fingerprint 9DC8 5822 9FC7 DD38 854A E2D8 8D81 803C 0EBF CD88, by searching for the last 8 characters of the fingerprint.

$ sudo apt-key fingerprint 0EBFCD88

Use the following command to set up the stable repository.

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

List the available versions in the repo

$ sudo apt-cache madison docker-ce

5:19.03.8~3-0

Install a specific version using the version string from the second column, for example, 5:19.03.8~3-0~ubuntu-bionic

$ sudo apt-get install -y docker-ce=5:19.03.8~3-0~ubuntu-bionic docker-ce-cli containerd.io

- Configure Docker daemon for Kubernetes.

$ sudo vi /etc/docker/daemon.json

Add below lines in the file and save it

```
{
  "exec-opts": ["native.cgroupdriver=systemd"],
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "100m"
  },
  "storage-driver": "overlay2"
}
```

The Docker daemon uses the HTTP_PROXY, HTTPS_PROXY, and NO_PROXY environmental variables in its start-up environment to configure HTTP or HTTPS proxy behavior. You cannot configure these environment variables using the daemon.json file. It overrides the default docker.service file.

sudo mkdir -p /etc/systemd/system/docker.service.d

Flush changes

sudo systemctl daemon-reload

Restart docker

sudo systemctl restart docker

- Configure Kubernetes apt repo and install kubeadm

sudo apt-get install -y apt-transport-https

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

cat <<EOF | sudo tee /etc/apt/sources.list.d/kubernetes.list deb https://apt.kubernetes.io/ kubernetes-xenial main EOF

Letting iptables see bridged traffic

As a requirement for your Linux Node's iptables to correctly see bridged traffic, you should ensure net.bridge.bridge-nf-call-iptables is set to 1 in your sysctl config, e.g.

cat <<EOF > /etc/sysctl.d/k8s.conf

net.bridge.bridge-nf-call-ip6tables = 1

net.bridge.bridge-nf-call-iptables = 1

EOF

sysctl --system

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 modprobe br_netfilter

sudo apt-get update

sudo apt-get install -y kubelet kubeadm Kubectl

sudo apt-mark hold kubelet kubeadm Kubectl

- Repeat steps to install docker, configure docker daemon and install Kubectl, Kubeadm and kubelet on each of the worker nodes

- Create Kubeadm file. Make sure the Kubernetes version is the one which you have installed. In this case we have used 1.18. By default, pod subnet as 192.168.x.x, you can change the same as per your range

```
apiVersion: kubeadm.k8s.io/v1beta2
kind: InitConfiguration
nodeRegistration:
  kubeletExtraArgs:
    cloud-provider: "azure"
    cloud-config: "/etc/kubernetes/cloud.conf"
---
apiVersion: kubeadm.k8s.io/v1beta2
kind: ClusterConfiguration
kubernetesVersion: v1.18.0
apiServer:
  extraArgs:
    cloud-provider: "azure"
    cloud-config: "/etc/kubernetes/cloud.conf"
  extraVolumes:
  - name: cloud
    hostPath: "/etc/kubernetes/cloud.conf"
    mountPath: "/etc/kubernetes/cloud.conf"
controllerManager:
  extraArgs:
```

```
    cloud-provider: "azure"
    cloud-config: "/etc/kubernetes/cloud.conf"
  extraVolumes:
  - name: cloud
    hostPath: "/etc/kubernetes/cloud.conf"
    mountPath: "/etc/kubernetes/cloud.conf"
networking:
  serviceSubnet: "192.168.0.0/16"
  podSubnet: "192.168.0.0/16"
```

Create /etc/kubernetes/cloud.conf file, it will contain the configuration for the Azure Cloud Provider.

```
azureuser@kube-master:~$ cat /etc/kubernetes/cloud.conf
{
    "cloud":"AzurePublicCloud",
    "tenantId": "XXX",
    "subscriptionId": "XXX",
    "aadClientId": "XXX",
    "aadClientSecret": "XXXX",
    "resourceGroup": "k8s-lab-rg4",
    "location": "eastus",
    "vmType": "standard",
    "subnetName": "k8s-lab-net1",
    "securityGroupName": "kube-masterNSG",
    "vnetName": "k8s-lab-vnet",
    "vnetResourceGroup": "",
    "routeTableName": "",
    "primaryAvailabilitySetName": "kubeadm-nodes-as",
    "primaryScaleSetName": "",
    "cloudProviderBackoffMode": "v2",
    "cloudProviderBackoff": true,
    "cloudProviderBackoffRetries": 6,
    "cloudProviderBackoffDuration": 5,
    "cloudProviderRatelimit": true,
    "cloudProviderRateLimitQPS": 10,
```

```
  "cloudProviderRateLimitBucket": 100,
  "cloudProviderRatelimitQPSWrite": 10,
  "cloudProviderRatelimitBucketWrite": 100,
  "useManagedIdentityExtension": false,
  "userAssignedIdentityID": "",
  "useInstanceMetadata": true,
  "loadBalancerSku": "Basic",
  "disableOutboundSNAT": false,
  "excludeMasterFromStandardLB": false,
  "providerVaultName": "xxxx",
  "maximumLoadBalancerRuleCount": 250,
  "providerKeyName": "XXX",
  "providerKeyVersion": "xxx"
}
```

Bootstrap master k8s node

Initialize the master, or control plane node, by passing kuebadm.yaml as configuration parameter. Make sure that the instance name in Azure is the same as the hostname or kubeadm will fail fail to initialize the kubelet

sudo kubeadm init --config kubeadm.yml

It will setup the master node and install all the components. Post install it will show below output

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

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 172.10.1.4:6443 --token t3f6du.wiy41o9mm42f08fi ***

--discovery-token-ca-cert-hash sha256:1fe233cbd04fc1d250605f1be68fca6e5a345d7218d540847bc7693046b44db7

Create Kubeconfig so that k8s cluster is usable

mkdir -p $HOME/.kube

sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

sudo chown $(id -u):$(id -g) $HOME/.kube/config

Install network addon "Calico"

Now install calico or any other networking addon. To use Calico, follow below instructions. You can also use weave or flannel.

https://docs.projectcalico.org/v3.9/getting-started/kubernetes/installation/calico

1. Download the calico.yaml file.

   curl https://docs.projectcalico.org/v3.9/manifests/calico.yaml -O

2. The above command will download the calico.yaml file in your pwd of your master node. Below are the contents of the file

   If you are using a different pod CIDR, use the following commands to set an environment variable called POD_CIDR containing your pod CIDR and replace 192.168.0.0/16 in the manifest with your pod CIDR.

   sudo POD_CIDR="<your-pod-cidr>" sed -i -e "s?10.11.0.0/16?$POD_CIDR?g" calico.yaml

3. Apply the yaml manifest file

   kubectl apply -f calico.yaml

4. Check the status of nodes and pods : DNS and calico pods should be in running state now

```
azureuser@kube-master:~$ kubectl get pods -n kube-system -o wide
NAME                                       READY   STATUS    RESTARTS   AGE     IP              NODE          NOMINATED NODE   READINESS GATES
calico-kube-controllers-5fc5dbfc47-15g8v   1/1     Running   0          9m14s   192.168.221.193 kube-master   <none>           <none>
calico-node-dh6p7                          1/1     Running   1          76s     172.10.1.7      kube-node-2   <none>           <none>
calico-node-f8p67                          1/1     Running   1          3m52s   172.10.1.5      kube-node-0   <none>           <none>
calico-node-hw7s9                          1/1     Running   1          9m14s   172.10.1.4      kube-master   <none>           <none>
calico-node-kpsp5                          0/1     Running   1          42s     172.10.1.6      kube-node-1   <none>           <none>
coredns-66bff467f8-gwlc5                   1/1     Running   0          12m     192.168.169.129 kube-node-2   <none>           <none>
coredns-66bff467f8-z8mjs                   1/1     Running   0          12m     192.168.221.194 kube-master   <none>           <none>
etcd-kube-master                           1/1     Running   0          13m     172.10.1.4      kube-master   <none>           <none>
kube-apiserver-kube-master                 1/1     Running   0          13m     172.10.1.4      kube-master   <none>           <none>
kube-controller-manager-kube-master        1/1     Running   0          13m     172.10.1.4      kube-master   <none>           <none>
kube-proxy-9kb5k                           1/1     Running   0          42s     172.10.1.6      kube-node-1   <none>           <none>
kube-proxy-mhdfj                           1/1     Running   0          3m52s   172.10.1.5      kube-node-0   <none>           <none>
kube-proxy-qbd7z                           1/1     Running   0          12m     172.10.1.4      kube-master   <none>           <none>
kube-proxy-xcfqt                           1/1     Running   0          76s     172.10.1.7      kube-node-2   <none>           <none>
kube-scheduler-kube-master                 1/1     Running   0          13m     172.10.1.4      kube-master   <none>           <none>
```

5. Run command **Ifconfig** to check if tun has been created by calico

```
tunl0: flags=193<UP,RUNNING,NOARP>  mtu 1440
        inet 192.168.221.192  netmask 255.255.255.255
        tunnel   txqueuelen 1000  (IPIP Tunnel)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

azureuser@kube-master:~$
```

```
docker0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 172.17.0.1  netmask 255.255.0.0  broadcast 172.17.255.255
        ether 02:42:cc:43:39:82  txqueuelen 0  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.10.1.4  netmask 255.255.255.0  broadcast 172.10.1.255
        inet6 fe80::20d:3aff:fe99:8f05  prefixlen 64  scopeid 0x20<link>
        ether 00:0d:3a:99:8f:05  txqueuelen 1000  (Ethernet)
        RX packets 183368  bytes 201942201 (201.9 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 84549  bytes 17140329 (17.1 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Join any number of worker nodes by running the following on each as root:

```
kubeadm join 172.10.1.4:6443 --token t3f6du.wiy41o9mm42f08fi \
  --discovery-token-ca-cert-hash
sha256:1fe233cbd04fc1d250605f1be68fca6e5a345d7218d540847bc7693046b44db7
```

Please download all the required files present at github:

https://github.com/Atul7696/kubeadm-deploy-azure