Deploying a Kubernetes Cluster in Azure using Kubeadm

The first thing that we need to do is create 3 VMs. The first one will act as the master node and the other two will be the minions. We will use Ubuntu 18.04 as the operating system and place all our VMs in a single VNet

# Create resource group
```
az group create -n CustomKubeCluster -l eastus
```

# Create VNet
```
az network vnet create -g CustomKubeCluster -n KubeVNet
--address-prefix 172.0.0.0/16 --subnet-name MySubnet --subnet-prefix
172.0.0.0/24
```

# Create Master Node - This will have a public IP
```
az vm create -n kube-master -g CustomKubeCluster --image
Canonical:UbuntuServer:18.04-LTS:18.04.201804262  --size
Standard_DS2_v2  --data-disk-sizes-gb 10 --generate-ssh-keys
--public-ip-address-dns-name ahmedkubeadm-master
```

# Create the two worker nodes
```
az vm create -n kube-worker-1  -g CustomKubeCluster --image
Canonical:UbuntuServer:18.04-LTS:18.04.201804262  --size
Standard_DS2_v2 --data-disk-sizes-gb 10  --generate-ssh-keys
--public-ip-address-dns-name ahmedkubeadm-worker-1
```

```
az vm create -n kube-worker-2 -g CustomKubeCluster --image
Canonical:UbuntuServer:18.04-LTS:18.04.201804262 --size Standard_DS2_v2
--data-disk-sizes-gb 10  --generate-ssh-keys  --public-ip-address-dns-name
ahmedkubeadm-worker-2
```

Now we are ready to get started. The first step is to install docker and kubeadm on all of the nodes. We are going to ssh into all nodes all run the same set of commands.

# These steps need to be performed on all nodes
```
sudo apt update
```

```
# Install Docker
sudo apt install docker.io -y
sudo systemctl enable docker

# Get the gpg keys for Kubeadm
curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo
apt-key add
sudo apt-add-repository "deb http://apt.kubernetes.io/
kubernetes-xenial main"

# Install Kubeadm
sudo apt-get install -y kubeadm=1.19.1-00 kubelet=1.19.1-00
kubectl=1.19.1-00
```

Now that kubeadm has been setup the next thing we're going to do is initialize kubeadm. We will need to run the init script on the master.

```
sudo kubeadm init
```

Once the command successfully runs it will output the join command which we will then need to run on the minions. We'll copy the join command for now — we'll need it in a future step.
Now, we will have to copy the admin.conf file so that we can run the kubectl command as a regular user (as opposed to root) from the master. The admin.conf file contains information such as the user, cert, etc needed to for kubectl to connect to your cluster.

```
mkdir $HOME/.kube

# Copy conf file to .kube directory for current user
sudo cp /etc/kubernetes/admin.conf $HOME/.kube/config

# Change ownership of file to current user and group
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

Now we will run the join command on the worker nodes

```
sudo kubeadm join 172.X.X.X:6443  --token tvbvra.fpp8fXXXXXqn9w
--discovery-token-ca-cert-hash
sha256:e61c2bb32f435f7be6fec9f7XXXXc5520e0e2a6385eX9298251238c
```

You should get an output saying This node has joined the cluster.

Now, the last thing to do before testing the setup is to install a pod network in the cluster. The pod network allows pods to communicate with each other across nodes. We will be using weave for this example, however you could also use any other solution such as flannel or calico.

```
	# Install Weave
	kubectl apply -f
"https://cloud.weave.works/k8s/net?k8s-version=$(kubectl version |
base64 | tr -d '\n')"
	# Check if the pods are running
	kubectl get pods -n kube-system
```

Once the pods are running we can now go ahead and deploy a test application:

```
# Create a test deployment
kubectl run --image=nginx webserver

# Create a service
kubectl expose --type NodePort --port 80 deployment webserver

# Get the Port for the service
kubectl get svc
```

To test out the service we will need to open up the NSG for that port. In my case the NodePort for the service was 30767

```
az vm open-port -g CustomKubeCluster --name kube-master --port 30767
```

Now if you hit the URL *http://<master>.eastus.cloudapp.azure.com:30767* you should see the familiar “Welcome to Nginx” page.

With this you should have a working, self-managed Kubernetes cluster running on Azure