Exercise 2.2: Deploy a New Cluster

Deploy a Master Node using Kubeadm

1. Review the script to install and begin the configuration of the master kubernetes server. You may need to change the **find** command search directory which uses tilde for your home directory depending on how and where you downloaded the tarball.

> A **find** command is shown if you want to locate and copy to the current directory instead of creating the file. Mark the command for reference as it may not be shown for future commands.
> `student@ckad-1:~$ find ~ -name <YAML File>`
> `student@ckad-1:~$ cp LFD259/<Some Path>/<YAML File> .`

```
student@ckad-1:~$ find ~ -name k8sMaster.sh
```

```
student@ckad-1:~$ less LFD259/SOLUTIONS/s_02/EXAMPLES/k8sMaster.sh
```

SH k8sMaster.sh

```
#!/bin/bash -x
## TxS 5-2019
## v1.14.1 CKAD
echo "This script is written to work with Ubuntu 16.04"
sleep 3
echo
echo "Disable swap until next reboot"
echo
sudo swapoff -a

echo "Update the local node"
sudo apt-get update && sudo apt-get upgrade -y
echo
echo "Install Docker"
sleep 3

sudo apt-get install -y docker.io
echo
echo "Install kubeadm and kubectl"
sleep 3

sudo sh -c
↪  "echo 'deb http://apt.kubernetes.io/ kubernetes-xenial main' >> /etc/apt/sources.list.d/kubernetes.list"

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

sudo apt-get update

sudo apt-get install -y kubeadm=1.14.1-00 kubelet=1.14.1-00 kubectl=1.14.1-00
```


SH

```
echo
echo "Installed - now to get Calico Project network plugin"

## If you are going to use a different plugin you'll want
## to use a different IP address, found in that plugins
## readme file.

sleep 3

sudo kubeadm init --kubernetes-version 1.14.1 --pod-network-cidr 192.168.0.0/16

sleep 5

echo "Running the steps explained at the end of the init output for you"

mkdir -p $HOME/.kube

sleep 2

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

sleep 2

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

echo "Download Calico plugin and RBAC YAML files and apply"

wget https://tinyurl.com/yb4xturm -O rbac-kdd.yaml

wget https://tinyurl.com/y8lvqc9g  -O calico.yaml

kubectl apply -f rbac-kdd.yaml

kubectl apply -f calico.yaml

echo
echo
sleep 3
echo "You should see this node in the output below"
echo "It can take up to a mintue for node to show Ready status"
echo
kubectl get node
echo
echo
echo "Script finished. Move to the next step"
```

2. Run the script as an argument to the **bash** shell. You will need the `kubeadm join` command shown near the end of the output when you add the worker/minion node in a future step. Use the **tee** command to save the output of the script, in case you cannot scroll back to find the `kubeadm join` in the script output. Please note the following is one command and then its output. It may be easier to copy files to your home directory first.

```
student@ckad-1:~$ cp LFD259/SOLUTIONS/s_02/EXAMPLES/k8sMaster.sh .

student@ckad-1:~$ bash k8sMaster.sh | tee ~/master.out

<output_omitted>

Your Kubernetes master 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 \verb?kubectl apply -f [podnetwork].yaml? with one
    of the options listed at:
    https://kubernetes.io/docs/concepts/cluster-administration/addons/

    You can now join any number of machines by running the
    following on each node as root:

  kubeadm join 10.128.0.3:6443 --token 69rdjq.2x2012j9ncexy37b
  --discovery-token-ca-cert-hash

sha256:72143e996ef78301191b9a42184124416aebcf0c7f363adf9208f9fa599079bd

<output_omitted>

+ kubectl get node
NAME              STATUS     ROLES     AGE       VERSION
ckad-1            NotReady   master    18s       v1.14.1
+ echo
+ echo 'Script finished. Move to the next step'
Script finished. Move to the next step
```

Deploy a Minion Node

3. Open a separate terminal into your **second node**. Having both terminal sessions allows you to monitor the status of the cluster while adding the second node. Find and copy the `k8sSecond.sh` file to the second node then view it. You should see the same early steps as on the master system.

```
student@ckad-2:~$ less k8sSecond.sh
```

SH k8sSecond.sh

```
#!/bin/bash -x
## TxS 5-2019
## CKAD for 1.14.1
##
echo "  This script is written to work with Ubuntu 16.04"
echo
sleep 3
echo "  Disable swap until next reboot"
echo
sudo swapoff -a

echo "  Update the local node"
sleep 2
sudo apt-get update && sudo apt-get upgrade -y
echo
sleep 2

echo "  Install Docker"
sleep 3
sudo apt-get install -y docker.io
```


SH

```
echo
echo "  Install kubeadm and kubectl"
sleep 2
sudo sh -c
↪  "echo 'deb http://apt.kubernetes.io/ kubernetes-xenial main' >> /etc/apt/sources.list.d/kubernetes.list"

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

sudo apt-get update

sudo apt-get install -y kubeadm=1.14.1-00 kubelet=1.14.1-00 kubectl=1.14.1-00

echo
echo "  Script finished. You now need the kubeadm join command"
echo "  from the output on the master node"
echo
```

4. Run the script on the **second node**.

```
student@ckad-2:~$ bash k8sSecond.sh
<output_omitted>
```

5. When the script is done the minion node is ready to join the cluster. The `kubeadm join` statement can be found near the end of the `kubeadm init` output on the master node. It should also be in the file `master.out` as well. Your nodes will use a different IP address and hashes than the example below. You'll need to pre-pend `sudo` to run the script copied from the master node.

```
student@ckad-2:~$ sudo kubeadm join --token 118c3e.83b49999dc5dc034 \
 10.128.0.3:6443 --discovery-token-ca-cert-hash \
sha256:40aa946e3f53e38271bae24723866f56c86d77efb49aedeb8a70cc189bfe2e1d
<output_omitted>
```

Configure the Master Node

6. Return to the master node. We will configure command line completion and verify both nodes have been added to the cluster. The first command will configure completion in the current shell. The second command will ensure future shells have completion.

```
student@ckad-1:~$ source <(kubectl completion bash)

student@ckad-1:~$ echo "source <(kubectl completion bash)" >> ~/.bashrc
```

7. Verify that both nodes are part of the cluster. Until we remove taints the nodes may not reach `Ready` state.

```
student@ckad-1:~$ kubectl get node
NAME        STATUS     ROLES     AGE         VERSION
ckad-1      NotReady   master    4m11s       v1.14.1
ckad-2      NotReady   <none>    3m6s        v1.14.1
```

8. We will use the **kubectl** command for the majority of work with Kubernetes. Review the help output to become familiar with commands options and arguments.

```
student@ckad-1:~$ kubectl --help
```



```
kubectl controls the Kubernetes cluster manager.

Find more information at:
  https://kubernetes.io/docs/reference/kubectl/overview/

Basic Commands (Beginner):
  create         Create a resource from a file or from stdin.
  expose         Take a replication controller, service,
 deployment or pod and expose it as a new Kubernetes Service
  run            Run a particular image on the cluster
  set            Set specific features on objects
  run-container  Run a particular image on the cluster. This
 command is deprecated, use "run" instead

Basic Commands (Intermediate):
<output_omitted>
```

9. With more than 40 arguments, you can explore each also using the `--help` option. Take a closer look at a few, starting with `taint` for example.

```
student@ckad-1:~$ kubectl taint --help
Update the taints on one or more nodes.

  * A taint consists of a key, value, and effect. As an argument
   here, it is expressed as key=value:effect.
  * The key must begin with a letter or number, and may contain
   letters, numbers, hyphens, dots, and underscores, up to
   253 characters.
  * Optionally, the key can begin with a DNS subdomain prefix
   and a single '/',
like example.com/my-app
<output_omitted>
```

10. By default the master node will not allow general containers to be deployed for security reasons. This is via a `taint`. Only containers which tolerate this taint will be scheduled on this node. As we only have two nodes in our cluster we will remove the taint, allowing containers to be deployed on both nodes. This is not typically done in a production environment for security and resource contention reasons. The following command will remove the taint from all nodes, so you should see one success and one `not found` error. The worker/minion node does not have the taint to begin with. Note the **minus sign** at the end of the command, which removes the preceding value.

```
student@ckad-1:~$  kubectl describe nodes | grep -i Taint
Taints:             node-role.kubernetes.io/master:NoSchedule
Taints:             node.kubernetes.io/not-ready:NoSchedule

student@ckad-1:~$ kubectl taint nodes --all node-role.kubernetes.io/master-
node/ckad-1 untainted
taint "node-role.kubernetes.io/master:" not found
```

11. Check that both nodes are without a `Taint`. If they both are without taint the `nodes` should now show as `Ready`. It may take a minute or two for all pods to enter `Ready` state.

```
student@ckad-1:~$ kubectl describe nodes | grep -i taint
Taints:           <none>
Taints:           <none>

student@ckad-1:~$ kubectl get nodes
NAME                STATUS   ROLES    AGE     VERSION
ckad-1              Ready    master   6m1s    v1.14.1
ckad-2              Ready    <none>   5m31s   v1.14.1
```

