Exercise 2.2: Deploy a New Cluster

Deploy a Control Plane Node using Kubeadm

1. Log into your nodes using **PuTTY** or using **SSH** from a terminal window. Unless the instructor tells you otherwise the user name to use will be **student**. You may need to change the permissions on the `pem` (or `ppk` on windows) file as shown in the following commands. Your file name and node IP address will probably be different.

   If using PuTTY, search for instructions on using a `ppk` key to access an instance. Use the **student** username when asked by PuTTY.

```
localTerm:~$ chmod 400 LF-Class.pem
localTerm:~$ ssh -i LF-Class.pem student@WW.XX.YY.ZZ
```

```
student@cp:~$
```

2. Use the **wget** command shown above to download then expand the course tarball to your cp node. Install **wget** if the command is not found on your instance.

3. Decide to use **cri-o** container engine the course is written for, or the older **docker**.

Very Important

The course is written to use **cri-o**, which is newer than **Docker**. There may be some hiccups, but the community is moving away from using **Docker** as the default container engine.

If **cri-o** has issues you can build a cluster using **Docker** instead, following the instructions in the `docker-setup.txt` file included in the course tarball. Only one engine should be installed. Both the **cp** and **worker** should be installed with the same engine. If you install **Docker** skip forward to the `Configure the Control Plane Node` section, found in later in the lab. As well where you see podman and crictl commands you would use docker, with the same operands and arguments. The output will be similar, but may have slight differences.

4. Review the script to install and begin the configuration of the cp 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@cp:~$ find $HOME -name <YAML File>
student@cp:~$ cp LFD259/<Some Path>/<YAML File> .
```

```
student@cp:~$ find $HOME -name k8scp.sh
```

```
student@cp:~$ more LFD259/SOLUTIONS/s_02/k8scp.sh
```

k8scp.sh

```
# Bring node to current versions and install an editor and other software
sudo apt-get update && sudo apt-get upgrade -y
```

SH

```
sudo apt-get install -y vim nano libseccomp2

# Prepare for cri-o
sudo modprobe overlay
sudo modprobe br_netfilter

echo "net.bridge.bridge-nf-call-iptables = 1" | sudo tee -a /etc/sysctl.d/99-kubernetes-cri.conf
echo "net.ipv4.ip_forward = 1" | sudo tee -a /etc/sysctl.d/99-kubernetes-cri.conf
echo "net.bridge.bridge-nf-call-ip6tables = 1" | sudo tee -a /etc/sysctl.d/99-kubernetes-cri.conf

sudo sysctl --system

# Add an alias for the local system to /etc/hosts
sudo sh -c "echo '$(hostname -i) cp' >> /etc/hosts"

# Set the versions to use
export OS=xUbuntu_18.04

export VERSION=1.21

#Add repos and keys
echo
↪  "deb http://download.opensuse.org/repositories/devel:/kubic:/libcontainers:/stable:/cri-o:/$VERSION/$OS/ /"
↪  | sudo tee -a /etc/apt/sources.list.d/cri-0.list

curl -L
↪  http://download.opensuse.org/repositories/devel:/kubic:/libcontainers:/stable:/cri-o:/$VERSION/$OS/Release.key
↪  | sudo apt-key add -

echo "deb https://download.opensuse.org/repositories/devel:/kubic:/libcontainers:/stable/$OS/ /" | sudo tee
↪  -a /etc/apt/sources.list.d/libcontainers.list

sudo apt-get update

# Install cri-o
sudo apt-get install -y cri-o cri-o-runc podman buildah

sleep 3

# Fix a bug, may not always be needed
sudo sed -i 's/,metacopy=on//g' /etc/containers/storage.conf


sleep 3

sudo systemctl daemon-reload

sudo systemctl enable crio

sudo systemctl start crio
# In case you need to check status:     systemctl status crio

# Add Kubernetes repo and software
sudo sh -c
↪  "echo 'deb http://apt.kubernetes.io/ kubernetes-xenial main' >> /etc/apt/sources.list.d/kubernetes.list"

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

sudo apt-get update

sudo apt-get install -y kubeadm=1.21.1-00 kubelet=1.21.1-00 kubectl=1.21.1-00

# Now install the cp using the kubeadm.yaml file from tarball
sudo kubeadm init --config=$(find / -name kubeadm.yaml 2>/dev/null )

sleep 5

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

```sh
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 "Apply Calico network plugin from ProjectCalico.org"
echo "If you see an error they may have updated the yaml file"
echo "Use a browser, navigate to the site and find the updated file"

kubectl apply -f https://docs.projectcalico.org/manifests/calico.yaml

echo

# Add alias for podman to docker for root and non-root user
echo "alias sudo="sudo "" | tee -a $HOME/.bashrc
echo "alias docker=podman" | tee -a $HOME/.bashrc

# Add Helm to make our life easier
wget https://get.helm.sh/helm-v3.5.4-linux-amd64.tar.gz
tar -xf helm-v3.5.4-linux-amd64.tar.gz
sudo cp linux-amd64/helm /usr/local/bin/

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"
```

5. 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.

   Using **Ubuntu 18** you may be asked questions during the installation. Allow restarts (yes) and use the local, installed software if asked during the update, usually (option 2). Add your timezone info if asked.

   Copy files to your home directory first.

```
student@cp:~$ cp LFD259/SOLUTIONS/s_02/k8scp.sh .

student@cp:~$ bash k8scp.sh | tee $HOME/cp.out
```

```
<output_omitted>

Your Kubernetes cp 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/

      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.2x20l2j9ncexy37b
  --discovery-token-ca-cert-hash

sha256:72143e996ef78301191b9a42184124416aebcf0c7f363adf9208f9fa599079bd

<output_omitted>


NAME           STATUS     ROLES                    AGE       VERSION
cp             NotReady   control-plane,master     19s       v1.21.1

This is the line from control plane /etc/hosts to add to the
worker node /etc/hosts file:
10.128.0.18 k8scp

Script finished. Move to the next step
```

Deploy a Worker Node

6. Open a separate terminal into your **second node**, which will be your worker. Having both terminal sessions allows you to monitor the status of the cluster while adding the second node. Change the color or other characteristic of the second terminal to make it visually distinct from the first. This will keep you from running commands on the incorrect instance, which probably won't work.

   Use the previous **wget** command download the tarball to the worker node. Extract the files with **tar** as before. Find and copy the `k8sSecond.sh` file to `student`'s home directory then view it. You should see the same early steps as found in the cp setup script.

   ```
   student@worker:˜$ more k8sSecond.sh
   ```

k8sSecond.sh

```
# Script to install a worker of the cluster
# Bring node to current versions and install an editor and other software
sudo apt-get update && sudo apt-get upgrade -y

sudo apt-get install -y vim nano libseccomp2

# Prepare for cri-o
sudo modprobe overlay
sudo modprobe br_netfilter

echo "net.bridge.bridge-nf-call-iptables = 1" | sudo tee -a /etc/sysctl.d/99-kubernetes-cri.conf
echo "net.ipv4.ip_forward = 1" | sudo tee -a /etc/sysctl.d/99-kubernetes-cri.conf
echo "net.bridge.bridge-nf-call-ip6tables = 1" | sudo tee -a /etc/sysctl.d/99-kubernetes-cri.conf

sudo sysctl --system


# Set the versions to use
export OS=xUbuntu_18.04

export VERSION=1.21

#Add repos and keys
echo
↪  "deb http://download.opensuse.org/repositories/devel:/kubic:/libcontainers:/stable:/cri-o:/$VERSION/$OS/ /"
↪  | sudo tee -a /etc/apt/sources.list.d/cri-0.list
```

SH

```
curl -L
↪  http://download.opensuse.org/repositories/devel:/kubic:/libcontainers:/stable:/cri-o:/$VERSION/$OS/Release.key
↪  | sudo apt-key add -

echo "deb https://download.opensuse.org/repositories/devel:/kubic:/libcontainers:/stable/$OS/ /" | sudo tee
↪  -a /etc/apt/sources.list.d/libcontainers.list

sudo apt-get update

# Install cri-o
sudo apt-get install -y cri-o cri-o-runc podman buildah

# A bug fix to get past a cri-o update
sudo sed -i 's/,metacopy=on//g' /etc/containers/storage.conf


sleep 3

sudo systemctl daemon-reload

sudo systemctl enable crio

sudo systemctl start crio
# In case you need to check status:     systemctl status crio

# Add Kubernetes repo and software
sudo sh -c
↪  "echo 'deb http://apt.kubernetes.io/ kubernetes-xenial main' >> /etc/apt/sources.list.d/kubernetes.list"

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

sudo apt-get update

sudo apt-get install -y kubeadm=1.21.1-00 kubelet=1.21.1-00 kubectl=1.21.1-00

echo
echo "Script finished. Move to the next step"
echo
echo "You will need to type sudo then copy the entire *kubeadm join*"
echo "command from the cp to this worker."
echo
```

7. Run the script on the **second node**. Again please note you may have questions during the update. Allow daemons to restart, type `yes`, and use the local installed version, usually option 2.

```
student@worker:~$ bash k8sSecond.sh
```

```
<output_omitted>
```

8. 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 cp node. It should also be in the file `cp.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 cp node. Also note that some non-Linux operating systems and tools insert extra characters when multi-line samples are copied and pasted. Copying one line at a time solves this issue.

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

```
<output_omitted>
```

Configure the Control Plane Node

9. Return to the cp node. Install a text editor. While the lab uses **vim**, any text editor such as **emacs** or **nano** will work. Be aware that Windows editors may have issues with special characters. Also install the **bash-completion** package, if not already installed. Use the locally installed version of a package if asked.

```
student@cp:˜$ sudo apt-get install bash-completion vim -y
```

```
<output_omitted>
```

10. 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. You may need to exit the shell and log back in for command completion to work without error.

```
student@cp:˜$ source <(kubectl completion bash)

student@cp:˜$ echo "source <(kubectl completion bash)" >> $HOME/.bashrc
```

11. Verify that both nodes are part of the cluster. And show a `Ready` state.

```
student@cp:˜$ kubectl get node
```

```
NAME        STATUS   ROLES                    AGE        VERSION
cp          Ready    control-plane,master     4m11s      v1.21.1
worker      Ready    <none>                   61s        v1.21.1
```

12. 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@cp:˜$ 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

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

13. 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@cp:˜$ 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>
```

14. By default the cp 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@cp:~$  kubectl describe nodes | grep -i taint
```

```
Taints:             node-role.kubernetes.io/master:NoSchedule
Taints:             <node>
```

```
student@cp:~$ kubectl taint nodes --all node-role.kubernetes.io/master-
```

```
node/cp untainted
error: taint "node-role.kubernetes.io/master:" not found
```

15. 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 infrastructure pods to enter `Ready` state, such that the `nodes` will show a `Ready` state.

```
student@cp:~$ kubectl describe nodes | grep -i taint
```

```
Taints:           <none>
Taints:           <none>
```

```
student@cp:~$ kubectl get nodes
```

```
NAME      STATUS   ROLES                  AGE     VERSION
master    Ready    control-plane,master   6m1s    v1.21.1
worker    Ready    <none>                 5m31s   v1.21.1
```