Exercise 3.1: Install Kubernetes

Overview

There are several Kubernetes installation tools provided by various vendors. In this lab we will learn to use **kubeadm**. As a community-supported independent tool, it is planned to become the primary manner to build a Kubernetes cluster.

Platforms: GCP, AWS, VirtualBox, etc

The labs were written using **Ubuntu** instances running on **G**oogle **C**loud **P**latform (**GCP**). They have been written to be vendor-agnostic so could run on AWS, local hardware, or inside of virtualization to give you the most flexibility and options. Each platform will have different access methods and considerations. As of v1.19.0 the minimum (as in barely works) size for **VirtualBox** is 3vCPU/4G memory/5G minimal OS for `cp` and 1vCPU/2G memory/5G minimal OS for worker node. Most other providers work with 2CPU/7.5G.

If using your own equipment you will have to disable swap on every node. There may be other requirements which will be shown as warnings or errors when using the **kubeadm** command. While most commands are run as a regular user, there are some which require root privilege. Please configure **sudo** access as shown in a previous lab. You If you are accessing the nodes remotely, such as with **GCP** or **AWS**, you will need to use an SSH client such as a local terminal or **PuTTY** if not using **Linux** or a Mac. You can download **PuTTY** from www.putty.org. You would also require a `.pem` or `.ppk` file to access the nodes. Each cloud provider will have a process to download or create this file. If attending in-person instructor led training the file will be made available during class.

Very Important

Please disable any firewalls while learning Kubernetes. While there is a list of required ports for communication between components, the list may not be as complete as necessary. If using **GCP** you can add a rule to the project which `allows all traffic to all ports`. Should you be using **VirtualBox** be aware that inter-VM networking will need to be set to `promiscuous mode`.

In the following exercise we will install Kubernetes on a single node then grow the cluster, adding more compute resources. Both nodes used are the same size, providing 2 vCPUs and 7.5G of memory. Smaller nodes could be used, but would run slower, and may have strange errors.

YAML files and White Space

Various exercises will use YAML files, which are included in the text. You are encouraged to write the files when possible, as the syntax of YAML has white space indentation requirements that are important to learn. An important note, **do not** use tabs in your YAML files, **white space only. Indentation matters.**

If using a PDF the use of copy and paste often does not paste the single quote correctly. It pastes as a back-quote instead. You will need to modify it by hand. The files have also been made available as a compressed **tar** file. You can view the resources by navigating to this URL:

https://training.linuxfoundation.org/cm/LFS258

To login use user: `LFtraining` and a password of: `Penguin2014`

Once you find the name and link of the current file, which will change as the course updates, use **wget** to download the file into your node from the command line then expand it like this:

```
$ wget https://training.linuxfoundation.org/cm/LFS258/LFS258_V2021-06-18_SOLUTIONS.tar.xz \
              --user=LFtraining --password=Penguin2014

$ tar -xvf LFS258_V2021-06-18_SOLUTIONS.tar.xz
```

(**Note**: depending on your PDF viewer, if you are cutting and pasting the above instructions, the underscores may disappear and be replaced by spaces, so you may have to edit the command line by hand!)

Bionic

While **Ubuntu 18** `bionic` has become the typical version to deploy, the Kubernetes repository does not yet have matching binaries at the time of this writing. The `xenial` binaries can be used until an update is provided.

Install Kubernetes

Log into your control plane (cp) and worker nodes. If attending in-person instructor led training the node IP addresses will be provided by the instructor. You will need to use a `.pem` or `.ppk` key for access, depending on if you are using **ssh** from a terminal or **PuTTY**. The instructor will provide this to you.

1. Open a terminal session on your first node. For example, connect via **PuTTY** or **SSH** session to the first **GCP** node. The user name may be different than the one shown, `student`. The IP used in the example will be different than the one you will use.

```
[student@laptop ~]$  ssh -i LFS258.pem student@35.226.100.87
```

```
The authenticity of host '54.214.214.156 (35.226.100.87)' can't be established.
ECDSA key fingerprint is SHA256:IPvznbkx93/Wc+ACwXrCcDDgvBwmvEXC9vmYhk2Wo1E.
ECDSA key fingerprint is MD5:d8:c9:4b:b0:b0:82:d3:95:08:08:4a:74:1b:f6:e1:9f.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '35.226.100.87' (ECDSA) to the list of known hosts.
<output_omitted>
```

2. Use the **wget** command to download and extract the course tarball to your node.

3. Become `root` and update and upgrade the system. You may be asked a few questions. Allow restarts and keep the local version currently installed. Which would be a `yes` then a 2.

```
student@cp:~$ sudo -i
```

```
root@cp:~# apt-get update && apt-get upgrade -y
```

```
<output_omitted>

You can choose this option to avoid being prompted; instead,
all necessary restarts will be done for you automatically
so you can avoid being asked questions on each library upgrade.
```

```
Restart services during package upgrades without asking? [yes/no] yes
```

```
<output_omitted>

A new version (/tmp/fileEbke6q) of configuration file /etc/ssh/sshd_config is
available, but the version installed currently has been locally modified.

```

```
  1. install the package maintainer's version
  2. keep the local version currently installed
  3. show the differences between the versions
  4. show a side-by-side difference between the versions
  5. show a 3-way difference between available versions
  6. do a 3-way merge between available versions
  7. start a new shell to examine the situation
```

```
What do you want to do about modified configuration file sshd_config? 2
```

```
<output_omitted>
```

4. Install a text editor like **nano**, **vim**, or **emacs**. Any will do, the labs use a popular option, **vim**.

```
root@cp:~# apt-get install -y vim
```

```
<output-omitted>
```

5. The main choices for a container environment are **Docker** and **cri-o**. We suggest **Docker** for class, as **cri-o** is not yet the default when building the cluster with **kubeadm** on Ubuntu.

   The **cri-o** engine is the default in Red Hat products and is being implemented by others. Installing **Docker** is a single command. At the moment it takes several steps to install and configure **crio**.

Very Important

If you want extra challenge use cri-o. Otherwise install Docker

   Please note, install Docker **OR** cri-o. If both are installed the **kubeadm** init process search pattern will use Docker. Also be aware that if you choose to use cri-o you may find encounter different output than shown in the book.

   (a) **If** using Docker:

```
root@cp:~# apt-get install -y docker.io
```

```
<output-omitted>
```

   (b) **If** using CRI-O:

   i. Use the **modprobe** command to load the `overlay` and the `br_netfilter` modules.

```
root@cp:~# modprobe overlay

root@cp:~# modprobe br_netfilter
```

   ii. Create a **sysctl** config file to enable IP forwarding and netfilter settings persistently across reboots.

```
root@cp:~# vim /etc/sysctl.d/99-kubernetes-cri.conf
```

```
net.bridge.bridge-nf-call-iptables  = 1
net.ipv4.ip_forward                 = 1
net.bridge.bridge-nf-call-ip6tables = 1
```

   iii. Use the **sysctl** command to apply the config file.

```
root@cp:~# sysctl --system
```

```
.....
* Applying /etc/sysctl.d/99-kubernetes-cri.conf ...
net.bridge.bridge-nf-call-iptables = 1
net.ipv4.ip_forward = 1
net.bridge.bridge-nf-call-ip6tables = 1
* Applying /etc/sysctl.d/99-sysctl.conf ...
* Applying /etc/sysctl.conf ...
```

iv. Add the CRI-O software repository. First set two parameters, one for the version of the OS, note the `x` in front of `Ubuntu_18.04`, the other the version of **cri-o**, which will be `1.21`.

```
root@cp:~# export OS=xUbuntu_18.04

root@cp:~# export VER=1.20
```

v. Add a new repository for the **cri-o** software. Note the use of both the variables we just set. The command can be on one line, we only use command line continuation to avoid a confusing wrap on the page.

```
root@cp:~#  echo \
"deb http://download.opensuse.org/repositories/devel:/kubic:/libcontainers:/stable:/cri-o:/$VER/$OS/ /" \
 | tee -a /etc/apt/sources.list.d/cri-0.list
```

vi. Load the keys for the packages. You should be able to re-use the same URL and add `Release.key` to the end before sending the output to **apt-key add**.

```
root@cp:~# curl -L \
http://download.opensuse.org/repositories/devel:/kubic:/libcontainers:/stable:/cri-o:/$VER/$OS/Release.key \
| apt-key add -
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  1093  100  1093    0     0   4814      0 --:--:-- --:--:-- --:--:--  4814
OK
```

vii. Add the repository for `libcontainer` information, then add the package key. Note only one variable is being used.

```
root@cp:~# echo \
"deb https://download.opensuse.org/repositories/devel:/kubic:/libcontainers:/stable/$OS/ /" \
| tee -a /etc/apt/sources.list.d/libcontainers.list

root@cp:~# curl \
-L https://download.opensuse.org/repositories/devel:/kubic:/libcontainers:/stable/$OS/Release.key \
| apt-key add -
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  1093  100  1093    0     0   2962      0 --:--:-- --:--:-- --:--:--  2954
OK
```

```
root@cp:~# apt-get update
```

viii. We can now install **cri-o** and the **runc** engine. Be aware the version may lag behind updates to Kubernetes software.

```
root@cp:~# apt-get install -y cri-o cri-o-runc
```

```
<output_omitted>
```

ix. Enable cri-o and ensure it is running.

```
root@cp:~# systemctl daemon-reload
root@cp:~# systemctl enable crio

root@cp:~# systemctl start crio
root@cp:~# systemctl status crio
```

```
crio.service - Container Runtime Interface for OCI (CRI-O)
  Loaded: loaded (/usr/lib/systemd/system/crio.service; disabled; vendor preset: enabled)
  Active: active (running) since Mon 2020-02-03 17:00:34 UTC; 7s ago
    Docs: https://github.com/cri-o/cri-o
....
```

6. Add a new repo for kubernetes. You could also download a tar file or use code from GitHub. Create the file and add an entry for the main repo for your distribution. We are using the `Ubuntu 18.04` but the `kubernetes-xenial` repo of the software, also include the key word `main`. Note there are four sections to the entry.

```
root@cp:~# vim /etc/apt/sources.list.d/kubernetes.list
```

```
deb  http://apt.kubernetes.io/  kubernetes-xenial  main
```

7. Add a GPG key for the packages. The command spans three lines. You can omit the backslash when you type. The `OK` is the expected output, not part of the command.

```
root@cp:~# curl -s \
   https://packages.cloud.google.com/apt/doc/apt-key.gpg \
   | apt-key add -
```

```
OK
```

8. Update with the new repo declared, which will download updated repo information.

```
root@cp:~# apt-get update
```

```
<output-omitted>
```

9. Install the software. There are regular releases, the newest of which can be used by omitting the equal sign and version information on the command line. Historically new versions have lots of changes and a good chance of a bug or five. As a result we will hold the software at the recent but stable version we install. In a later lab we will update the cluster to a newer version.

```
root@cp:~# apt-get install -y \
        kubeadm=1.20.1-00 kubelet=1.20.1-00 kubectl=1.20.1-00
```

```
<output-omitted>
```

```
root@cp:~# apt-mark hold kubelet kubeadm kubectl
```

```
kubelet set on hold.
kubeadm set on hold.
kubectl set on hold.
```

10. Deciding which pod network to use for Container Networking Interface (**CNI**) should take into account the expected demands on the cluster. There can be only one pod network per cluster, although the **CNI-Genie** project is trying to change this.

    The network must allow container-to-container, pod-to-pod, pod-to-service, and external-to-service communications. As **Docker** uses host-private networking, using the `docker0` virtual bridge and `veth` interfaces would require being on that host to communicate.

    We will use **Calico** as a network plugin which will allow us to use `Network Policies` later in the course. Currently **Calico** does not deploy using CNI by default. Newer versions of **Calico** have included RBAC in the main file. Once downloaded look for the expected IPV4 range for containers to use in the configuration file.

```
root@cp:~# wget https://docs.projectcalico.org/manifests/calico.yaml
```

11. Use **less** to page through the file. Look for the IPV4 pool assigned to the containers. There are many different configuration settings in this file. Take a moment to view the entire file. The `CALICO_IPV4POOL_CIDR` must match the value given to **kubeadm init** in the following step, whatever the value may be. Avoid conflicts with existing IP ranges of the instance.

```
root@cp:~# less calico.yaml
```

calico.yaml

```
....
    # The default IPv4 pool to create on startup if none exists. Pod IPs will be
```

```
    # chosen from this range. Changing this value after installation will have
    # no effect. This should fall within `--cluster-cidr`.
            - name: CALICO_IPV4POOL_CIDR
              value: "192.168.0.0/16"
....
```

12. Find the IP address of the primary interface of the cp server. The example below would be the `ens4` interface and an IP of `10.128.0.3`, yours may be different. There are two ways of looking at your IP addresses.

```
root@cp:~# hostname -i
10.128.0.3
```

```
root@cp:~# ip addr show
....
2: ens4: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1460 qdisc mq state UP group default qlen 1000
    link/ether 42:01:0a:80:00:18 brd ff:ff:ff:ff:ff:ff
    inet 10.128.0.3/32 brd 10.128.0.3 scope global ens4
       valid_lft forever preferred_lft forever
    inet6 fe80::4001:aff:fe80:18/64 scope link
       valid_lft forever preferred_lft forever
....
```

13. Add an local DNS alias for our cp server. Edit the `/etc/hosts` file and add the above IP address and assign a name `k8scp`.

```
root@cp:~# vim /etc/hosts

10.128.0.3 k8scp    #<-- Add this line
127.0.0.1 localhost
....
```

14. Create a configuration file for the cluster. There are many options we could include, and they differ for **Docker** and **cri-o**. For **Docker** we will only set the control plane endpoint, software version to deploy and podSubnet values. There are a lot more variables to set when using **cri-o**. Use the file included in the course tarball. After our cluster is initialized we will view other default values used. Be sure to use the node alias, not the IP so the network certificates will continue to work when we deploy a load balancer in a future lab.

IF USING DOCKER

```
root@cp:~# vim kubeadm-config.yaml    #<-- Only for Docker
```

kubeadm-config.yaml

```
apiVersion: kubeadm.k8s.io/v1beta2
kind: ClusterConfiguration
kubernetesVersion: 1.20.1              #<-- Use the word stable for newest version
controlPlaneEndpoint: "k8scp:6443"  #<-- Use the node alias not the IP
networking:
  podSubnet: 192.168.0.0/16            #<-- Match the IP range from the Calico config file
```

IF USING CRI-O

```
root@cp:~# find /home -name kubeadm-crio.yaml    #<-- Assuming tarball was expanded by non-root user

root@cp:~# cp <path_from_above> .
```

15. Initialize the cp. Read through the output line by line. Expect the output to change as the software matures. At the end are configuration directions to run as a non-root user. The token is mentioned as well. This information can be found later with the **kubeadm token list** command. The output also directs you to create a pod network to the cluster, which will be our next step. Pass the network settings **Calico** has in its configuration file, found in the previous step. **Please note:** the output lists several commands which following exercise steps will complete.

    Note: Change the config file if you are using cri-o.

```
root@cp:~# kubeadm init --config=kubeadm-config.yaml --upload-certs \
      | tee kubeadm-init.out      # Save output for future review
```

Please Note

What follows is output of **kubeadm init** from **Docker**. Read the next step prior to further typing.

```
[init] Using Kubernetes version: v1.20.1
[preflight] Running pre-flight checks
        [WARNING IsDockerSystemdCheck]: detected "cgroupfs" as the
  Docker cgroup driver. The recommended driver is "systemd".

<output_omitted>

You can now join any number of the control-plane node
running the following command on each as root:

kubeadm join k8scp:6443 --token vapzqi.et2p9zbkzk29wwth \
 --discovery-token-ca-cert-hash sha256:f62bf97d4fba6876e4c3ff645df3fca969c06169dee3865aab9d0bca8ec9f8cd \
 --control-plane --certificate-key 911d41fcada89a18210489afaa036cd8e192b1f122ebb1b79cce1818f642fab8

Please note that the certificate-key gives access to cluster sensitive
data, keep it secret!
As a safeguard, uploaded-certs will be deleted in two hours; If
necessary, you can use
"kubeadm init phase upload-certs --upload-certs" to reload certs afterward.

Then you can join any number of worker nodes by running the following
on each as root:

kubeadm join k8scp:6443 --token vapzqi.et2p9zbkzk29wwth \
 --discovery-token-ca-cert-hash sha256:f62bf97d4fba6876e4c3ff645df3fca969c06169dee3865aab9d0bca8ec9f8cd
```

16. As suggested in the directions at the end of the previous output we will allow a `non-root` user admin level access to the cluster. Take a quick look at the configuration file once it has been copied and the permissions fixed.

```
root@cp:~# exit
```

```
logout
```

```
student@cp:~$ mkdir -p $HOME/.kube

student@cp:~$ sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

student@cp:~$ sudo chown $(id -u):$(id -g) $HOME/.kube/config

student@cp:~$ less .kube/config
```

```
apiVersion: v1
clusters:
- cluster:
<output_omitted>
```

17. Apply the network plugin configuration to your cluster. Remember to copy the file to the current, non-root user directory first.

```
student@cp:~$ sudo cp /root/calico.yaml .

student@cp:~$ kubectl apply -f calico.yaml
```

```
configmap/calico-config created
customresourcedefinition.apiextensions.k8s.io/felixconfigurations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ipamblocks.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/blockaffinities.crd.projectcalico.org created
<output_omitted>
```

18. While many objects have short names, a **kubectl** command can be a lot to type. We will enable **bash** auto-completion. Begin by adding the settings to the current shell. Then update the `$HOME/.bashrc` file to make it persistent. Ensure the `bash-completion` package is installed. If it was not installed, log out then back in for the shell completion to work.

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

<exit and log back in>

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

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

19. Test by describing the node again. Type the first three letters of the sub-command then type the **Tab** key. Auto-completion assumes the `default` namespace. Pass the namespace first to use auto-completion with a different namespace. By pressing **Tab** multiple times you will see a list of possible values. Continue typing until a unique name is used. First look at the current node (your node name may not start with `cp`), then look at pods in the `kube-system` namespace. If you see an error instead such as `-bash: _get_comp_words_by_ref: command not found` revisit the previous step, install the software, log out and back in.

```
student@cp:~$ kubectl des<Tab> n<Tab><Tab> cp<Tab>

student@cp:~$ kubectl -n kube-s<Tab> g<Tab> po<Tab>
```

20. View other values we could have included in the `kubeadm-config.yaml` file when creating the cluster.

```
student@cp:~$ sudo kubeadm config print init-defaults
```

```
apiVersion: kubeadm.k8s.io/v1beta2
bootstrapTokens:
- groups:
  - system:bootstrappers:kubeadm:default-node-token
  token: abcdef.0123456789abcdef
  ttl: 24h0m0s
  usages:
  - signing
  - authentication
kind: InitConfiguration
<output_omitted>
```