THE LINUX FOUNDATION | Training & Certification

Lab 1.1 - Install Kubernetes and Create a Cluster

Overview

In this lab, you'll be installing Kubernetes and creating three Kubernetes clusters, one for each of the service mesh technologies you'll be using in this course: Linkerd, Istio, and Consul.

To follow along with this course, there are a few ways that you can implement a lab environment:

- Creating a Kind cluster locally, like on your Mac
- Creating a Kind cluster on a Linux desktop with a GUI, like Ubuntu
- Creating a Kubernetes cloud cluster, like GKE or EKS.

All of the labs will work through this course regardless of what option you choose. However, there will be differences. For example, you'll see in some of the labs that Kind is stopped and started for each cluster. If you're using GKE, you wouldn't do that. Instead, you'd just have three separate GKE clusters that you run workloads on.

To make this simple, free, and as consistent as possible, most of the labs will use an easily replicable and easy-to-follow installation method using Kubernetes in Docker (kind). All subsequent lab exercises assume you are using a **`kind`** cluster. For the purposes of keeping the explanations simple, we'll use the phrase "VM" to talk about the Kind cluster. "VM" could mean your local computer or it could mean an Ubuntu desktop. To keep things interchangeable, we'll use "VM".

Please keep in mind that if you decide to use another type of Kubernetes cluster, for example, in the cloud, it may accrue cost depending on how many credits you have on your cloud account. Ensure to use a cloud calculator for confirmation.

Kind consists of:

- The necessary packages (Go, container image builder, cluster creation, etc.)
- The CLI (`kind`) to use Kind
- Container images (Docker) to run Systemd, Kubernetes

Throughout this course, you will only be using one cluster at a time, and the other clusters must be stopped so they don't interfere with the one that's being used. Each lab will tell you when you need to start or stop a cluster, and you'll be given the commands to use each time.

1. You will use a virtual machine (VM) to host your Kubernetes clusters. Using your preferred cloud provider (please note the cloud provider will accrue cost) or hypervisor, provision an Ubuntu VM (version 20.04 LTS) with the following characteristics:

    a. The VM must have at least 2 CPUs, 8 GB of memory, and 30 GB of boot disk space.

    b. The VM must use an IP address that is reachable from your computer.

    c. The VM must have network access for TCP ports 80 (HTTP), 443 (HTTPS), and 8080 (multiple graphical user interfaces). You may need to add firewall rules that permit this activity; always be cautious when altering firewall rules so you do not inadvertently allow unwanted network traffic.

2. Log into your Ubuntu VM through a command-line interface, like SSH.

3. The latest version of Docker Engine must be installed in the Ubuntu VM. See https://docs.docker.com/engine/install/ubuntu/#install-using-the-repository if you need instructions on installing Docker Engine in Ubuntu. Follow the instructions under the **Install using the repository** heading. Skip the instruction about installing a specific version, because you'll be installing the latest version in the item before that, and stop when you reach the "Install from a package" heading.

4. To test that your Docker installation is working, issue this command and confirm that your output looks similar to the text below:

```
yourname@ubuntu-vm:~$ sudo docker run hello-world

Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
0e03bdcc26d7: Pull complete
Digest:
sha256:49a1c8800c94df04e9658809b006fd8a686cab8028d33cfba2cc049724
254202
```

```
Status: Downloaded newer image for hello-world:latest
Hello from Docker!

This message shows that your installation appears to be working
correctly.

To generate this message, Docker took the following steps:
1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the
Docker Hub.
   (amd64)
3. The Docker daemon created a new container from that image
which runs the
   executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client,
which sent it
   to your terminal.

To try something more ambitious, you can run an Ubuntu container
with:
$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
https://hub.docker.com/

For more examples and ideas, visit:
https://docs.docker.com/get-started/
```

5. To move forward with Kind, you must belong to the **`docker`** user group. Join it by issuing the command below.

```
yourname@ubuntu-vm:~$ sudo usermod -aG docker $USER

[no output if the command succeeds]
```

6. You will need to restart your session after issuing the previous command so that the change in group membership takes effect. You can restart your session simply by logging out and back in.

```
yourname@ubuntu-vm:~$ exit
```

```
[existing session will close; start another command-line session
to log back into the Ubuntu VM]
```

7. **`kubectl`** must be installed in the VM. See the Install kubectl binary with curl on Linux documentation page for instructions on installing **`kubectl`**. Follow the instructions under the “Install kubectl binary with curl on Linux” heading to download the latest release. When you’re done, verify that **`kubectl`** is installed by issuing this command and confirming that your output looks similar to the text below:

```
yourname@ubuntu-vm:~$ kubectl version --short

Client Version: version.Info{Major:"1", Minor:"19",
GitVersion:"v1.19.3", GitCommit:"1e11e4a2108024935ecfcb2912226c
edeafd99df", GitTreeState:"clean",
BuildDate:"2020-10-14T12:50:19Z", GoVersion:"go1.15.2",
Compiler:"gc", Platform:
"linux/amd64"}
```

8. **`helm`** must be installed in the VM. For instructions on installing it, see the “From Script” page. Follow the instructions under the “From Script” heading to fetch and execute the installer script. When you’re done, you should verify that **`helm`** is installed. Enter the following command and make sure your output looks similar to the output below:

```
yourname@ubuntu-vm:~$ helm version

version.BuildInfo{Version:"v3.4.0",
GitCommit:"7090a89efc8a18f3d8178bf47d2462450349a004",
GitTreeState:"clean", GoVersion:"go1.14.10"}
```

9. The next step is to enter the following three commands in the Ubuntu VM to install **`kind`** and set the permissions correctly. See the kind documentation for full installation information. Please note that the version in this lab may be a different version that you see on the Kind documentation, as the version may have been updated. The recommendation is to check that prior.

```
yourname@ubuntu-vm:~$ curl -Lo ./kind
https://kind.sigs.k8s.io/dl/v0.14.0/kind-linux-amd64


% Total     % Received % Xferd Average Speed Time Time Time
Current
```

```
Dload Upload Total Spent Left Speed
100    97 100    97  0  0   184 0 --:--:-- --:--:-- --:--:-- 184
100   629 100   629  0  0   879 0 --:--:-- --:--:-- --:--:-- 879
100 9900k 100 9900k  0  0 10.9M 0 --:--:-- --:--:-- --:--:--
10.9M

yourname@ubuntu-vm:~$ chmod +x ./kind

[no output if the command succeeds]

yourname@ubuntu-vm:~$ sudo mv ./kind /usr/local/bin/kind

[no output if the command succeeds]
```

10. To confirm that Kind was installed, run the following and you should see output similar to the below (it might be a different version number depending on what point in time you're doing this lab):

```
yourname@ubuntu-vm:~$ helm version
kind v0.8.1 go1.14.2 linux/amd64
```

11. After **kind** is installed, you will provision the first cluster, which will be for the Consul service mesh, by running the command below in the Ubuntu VM. The command will create the Kubernetes cluster and allow it to bind to port 80 (HTTP) and 443 (HTTPS). This is necessary for making sure your Ingress controller can route traffic. Note that it may take a few minutes for the provisioning to complete.

```
yourname@ubuntu-vm:~$ cat <<EOF | kind create cluster --name
consul --config=-
kind: Cluster
apiVersion: kind.x-k8s.io/v1alpha4
nodes:
- role: control-plane
  kubeadmConfigPatches:
  - |
    kind: InitConfiguration
    nodeRegistration:
      kubeletExtraArgs:
        node-labels: "ingress-ready=true"
  extraPortMappings:
  - containerPort: 80
    hostPort: 80
    protocol: TCP
  - containerPort: 443
    hostPort: 443
    protocol: TCP
EOF

Creating cluster "consul" ...
✓ Ensuring node image (kindest/node:v1.18.2) 🖼
✓ Preparing nodes 📦
✓ Writing configuration 📜
✓ Starting control-plane 🕹️
✓ Installing CNI 🔌
✓ Installing StorageClass 💾
Set kubectl context to "kind-consul"
You can now use your cluster with:

kubectl cluster-info --context kind-consul

Thanks for using kind! 😊
```

12. Once **`kind`** finishes creating your new cluster for Consul, point **`kubectl`** at the cluster and confirm that the Kubernetes master and KubeDNS are running:

```
yourname@ubuntu-vm:~$ kubectl cluster-info --context kind-consul

Kubernetes master is running at https://127.0.0.1:43963
KubeDNS is running at
https://127.0.0.1:43963/api/v1/namespaces/kube-system/services/ku
be-dns:dns/proxy

To further debug and diagnose cluster problems, use 'kubectl
cluster-info dump'.
```

13. Stop the cluster for Consul by issuing this command (give this command a few seconds to run as the output may not happen right away):

```
yourname@ubuntu-vm:~$ docker stop $(docker container ls -a -f
name=consul-control-plane -q)

2e64e6e909e1
```

14. Now you will provision the second cluster, for the Istio service mesh, by running the command below in the Ubuntu VM. This is identical to the command you issued for Consul except it gives this cluster a different name—"**`istio`**" instead of "**`consul`**". Note that it may take a few minutes for the provisioning to complete.

```
yourname@ubuntu-vm:~$ cat <<EOF | kind create cluster --name
istio --config=-
kind: Cluster
apiVersion: kind.x-k8s.io/v1alpha4
nodes:
- role: control-plane
  kubeadmConfigPatches:
  - |
    kind: InitConfiguration
    nodeRegistration:
      kubeletExtraArgs:
        node-labels: "ingress-ready=true"
  extraPortMappings:
  - containerPort: 80
    hostPort: 80
    protocol: TCP
  - containerPort: 443
    hostPort: 443
    protocol: TCP
EOF

Creating cluster "istio" ...
✓ Ensuring node image (kindest/node:v1.18.2) 🖼
✓ Preparing nodes 📦
✓ Writing configuration 📜
✓ Starting control-plane 🕹️
✓ Installing CNI 🔌
✓ Installing StorageClass 💾
Set kubectl context to "kind-istio"
You can now use your cluster with:

kubectl cluster-info --context kind-istio

Thanks for using kind! 😊
```

15. Once **`kind`** finishes creating your new cluster for Istio, point **`kubectl`** at the cluster and confirm that the Kubernetes master and KubeDNS are running:

```
yourname@ubuntu-vm:~$ kubectl cluster-info --context kind-istio

Kubernetes master is running at https://127.0.0.1:33559
KubeDNS is running at
https://127.0.0.1:33559/api/v1/namespaces/kube-system/services/ku
be-dns:dns/proxy

To further debug and diagnose cluster problems, use 'kubectl
cluster-info dump'.
```

16. Stop the cluster for Istio by issuing this command:

```
yourname@ubuntu-vm:~$ docker stop $(docker ps -a -f
name=istio-control-plane -q)

2d1d09fadf21
```

17. Now you will provision the third cluster, for the Linkerd service mesh, by running the command below in the Ubuntu VM. This cluster will be named "**`linkerd`**". Note that it may take a few minutes for the provisioning to complete.

```
yourname@ubuntu-vm:~$ cat <<EOF | kind create cluster --name
linkerd --config=-
kind: Cluster
apiVersion: kind.x-k8s.io/v1alpha4
nodes:
- role: control-plane
  kubeadmConfigPatches:
  - |
    kind: InitConfiguration
    nodeRegistration:
      kubeletExtraArgs:
        node-labels: "ingress-ready=true"
  extraPortMappings:
  - containerPort: 80
    hostPort: 80
    protocol: TCP
  - containerPort: 443
    hostPort: 443
    protocol: TCP
EOF
```

```
Creating cluster "linkerd" ...
✓ Ensuring node image (kindest/node:v1.18.2) 🖼
✓ Preparing nodes 📦
✓ Writing configuration 📜
✓ Starting control-plane 🕹️
✓ Installing CNI 🔌
✓ Installing StorageClass 💾
Set kubectl context to "kind-linkerd"
You can now use your cluster with:

kubectl cluster-info --context kind-linkerd

Thanks for using kind! 😊
```

18. Once **`kind`** finishes creating your new cluster for Linkerd, point **`kubectl`** at the cluster and confirm that the Kubernetes master and KubeDNS are running:

```
yourname@ubuntu-vm:~$ kubectl cluster-info --context kind-linkerd

Kubernetes master is running at https://127.0.0.1:33499
KubeDNS is running at
https://127.0.0.1:33499/api/v1/namespaces/kube-system/services/ku
be-dns:dns/proxy

To further debug and diagnose cluster problems, use 'kubectl
cluster-info dump'.
```

19. Stop the Linkerd cluster by issuing this command:

```
yourname@ubuntu-vm:~$ docker stop $(docker ps -a -f
name=linkerd-control-plane -q)

8af4c08524df
```

20. Congrats! You have successfully completed the setup of your clusters.

    Please Note: Depending on how long you're planning on taking to do the next labs, the Kind clusters may have to be deleted and re-created. Engineers have noticed that if you wait around 1 week or more, the clusters don't work the way they did in the beginning, as random errors occur. If you see this, you'll most likely have to run **`kind delete cluster --name name_of_cluster`** and start it again.

Also, if you plan on shutting down a cluster, the state isn't preserved.