Install & Configure Minikube

Objective: Construct Minikube tool to make it easy to run Kubernetes locally, for this course's labs

Preparation: Make sure to verify system information needed to choose installation option

Outcome: Kubernetes cluster running locally, inside a VM on your laptop

Data Files: Ask Instructor

Minikube is a tool that makes it easy to run Kubernetes locally. Minikube runs a single-node Kubernetes cluster inside a VM on your laptop for users looking to try out Kubernetes or develop with it day-to-day.

Minikube supports Kubernetes features such as:
DNS, NodePorts, ConfigMaps and Secrets, Dashboards, Container Runtime: Docker, and rkt, Enabling CNI (Container Network Interface), Ingress

Check over the installation requirements:

OS X

- VirtualBox, or VMWare Fusion
- xhyve driver

  From `https://github.com/zchee/docker-machine-driver-`

`xhyve#install`:

```
$ brew install docker-machine-driver-xhyve
// docker-machine-driver-xhyve need root owner and uid
$ sudo chown root:wheel $(brew --prefix)/opt/docker-machi
ne-driver-xhyve/bin/docker-machine-driver-xhyve
$ sudo chmod u+s $(brew --prefix)/opt/docker-machine-driv
er-xhyve/bin/docker-machine-driver-xhyve
```

**Linux
**

- VirtualBox
- KVM

  Minikube is currently tested against docker-machine-driver-kvm 0.7.0.

  From `https://github.com/dhiltgen/docker-machine-kvm#quick-start-instructions`:

```
$ sudo curl -L https://github.com/dhiltgen/docker-machine
-kvm/releases/download/v0.7.0/docker-machine-driver-kvm -
o /usr/local/bin/docker-machine-driver-kvm
$ sudo chmod +x /usr/local/bin/docker-machine-driver-kvm

// Install libvirt and qemu-kvm on your system, e.g.
// Debian/Ubuntu
$ sudo apt install libvirt-bin qemu-kvm
// Fedora/CentOS/RHEL
$ sudo yum install libvirt-daemon-kvm kvm
```

```
// Add yourself to the libvirtd group (use libvirt group
for rpm based distros) so you don't need to sudo
// Debian/Ubuntu
$ sudo usermod -a -G libvirtd $(whoami)
// Fedora/CentOS/RHEL
$ sudo usermod -a -G libvirt $(whoami)

// Update your current session for the group change to ta
ke effect
// Debian/Ubuntu
$ newgrp libvirtd
// Fedora/CentOS/RHEL
$ newgrp libvirt
```

**VT-x/AMD-v virtualization must be enabled in BIOS
**

**Kubectl (See the kubectl installation lab for more details)
**

Step 1. Installing Minikube v0.17.1 (https://github.com/kubernetes/minikube/r

osx

```
$ curl -Lo minikube https://storage.googleapis.com/miniku
be/releases/v0.17.1/minikube-darwin-amd64 && chmod +x min
ikube && sudo mv minikube /usr/local/bin/
```

Linux

```
$ curl -Lo minikube https://storage.googleapis.com/minikube/releases/v0.17.1/minikube-linux-amd64 && chmod +x minikube && sudo mv minikube /usr/local/bin/
```

Feel free to leave off the `sudo mv minikube /usr/local/bin` if you would like to add minikube to your path manually.

Windows [Experimental]

Download the `minikube-windows-amd64.exe` file from `https://github.com/kubernetes/minikube/releases`, rename it `minikube.exe` and add it to your `$PATH`

Step 2. Quick Start

Here's a brief demo of minikube usage. If you want to change the VM driver add the appropriate `--vm-driver=xxx` flag to `minikube start`. Minikube supports the following drivers:

- virtualbox
- vmwarefusion
- kvm
- xhyve

  This course is designed to use Vitualbox and VMWare primary, and even then, only use VMWareFusion if VirtualBox isn't available to be used.

NOTE: The IP below is dynamic and can change. It can be retrieved with `minikube ip`.

1. Follow along with the instructions below to quickly launch a minikube cluster locally.

```
$ minikube start
```

Output:

```
Starting local Kubernetes cluster...
Running pre-create checks...
Creating machine...
Starting local Kubernetes cluster...
```

2.

```
$ kubectl run hello-minikube --image=gcr.io/google_containers/echoserver:1.4 --port=8080
```

Output:

```
deployment "hello-minikube" created
```

3.

```
$ kubectl expose deployment hello-minikube --type=NodePort
```

Output:

```
service "hello-minikube" exposed
```

4. We have now launched an echoserver pod but we have to wait until the pod is up before curling/accessing it via the exposed service. To check whether the pod is up and running we can use the following:

```
$ kubectl get pod
```

Output:

```
NAME                                 READY     STATUS
       RESTARTS   AGE
hello-minikube-3383150820-vctvh   1/1       ContainerCrea
ting   0          3s
```

5. We can see that the pod is still being created from the ContainerCreating status. Let’s run it again in a minute.

```
$ kubectl get pod
```

Output:

```
NAME                                 READY     STATUS    RES
TARTS   AGE
hello-minikube-3383150820-vctvh   1/1       Running   0
        13s
```

6. We can see that the pod is now Running and we will now be able to curl it:

```
$ curl $(minikube service hello-minikube --url)
```

Output:

```
CLIENT VALUES:
client_address=192.168.99.1
command=GET
real path=/
...
```

7. To stop the minikube cluster, run the folliwing command:

```
$ minikube stop
```

Output:

```
Stopping local Kubernetes cluster...
Stopping "minikube"...
```

<!-- THIS DOES NOT GO –

Step 3. [OPTIONAL] Reusing the Docker daemon

(Ask the instructor if this step will work for you.) When using a single VM of Kubernetes, it’s really handy to reuse the minikube’s built-in Docker daemon; as this means you don’t have to build a docker registry on your host machine and push the image into it - you can just build inside the same docker daemon as minikube which speeds up local experiments. Just make sure you tag your Docker image with something other than ‘latest’ and use that tag while you pull the image. Otherwise, if you do not specify version of your image, it will be assumed as :latest, with pull image policy of Always correspondingly, which may eventually result in ErrImagePull as you may not have any versions of your Docker image out there in the default docker registry (usually DockerHub) yet.

1. To be able to work with the docker daemon on your mac/linux host use the docker-env command in your shell:

```
$ eval $(minikube docker-env)
```

2. Now we are able to use docker on the command line on your host mac/linux machine talking to the docker daemon inside the minikube VM. We can affirm that by running the following:

```
$ docker ps
```

3. On Centos 7, docker may report the following error:

```
Could not read CA certificate "/etc/docker/ca.pem": open
/etc/docker/ca.pem: no such file or directory
```

4. The fix is to update `/etc/sysconfig/docker` to ensure that minikube's environment changes are respected:

```
< DOCKER_CERT_PATH=/etc/docker
> if [ -z "${DOCKER_CERT_PATH}" ]; then
>   DOCKER_CERT_PATH=/etc/docker
> fi
```

5. Remember to turn off the `imagePullPolicy:Always`, as otherwise Kubernetes won't use images you built locally.

-->

Step 3. Other Useful Tips & Tricks

1. Kubectl-The minikube start command creates a "kubectl context" called "minikube". This context contains the configuration to communicate with your minikube cluster. Minikube sets this context to default automatically, but if you need to switch back to it in the future, run:

```
kubectl config use-context minikube
```

Or, pass the context on each command like this:

```
kubectl get pods --context=minikube.
```

2. Dashboard-To access the Kubernetes Dashboard, run this

command in a shell after starting minikube to get the address:

```
minikube dashboard
```

> *NOTE: This course won't ulitize the dashboard, but feel free to look around to gain more understanding on Kubernetes.*

Step 4. Clean Up

Conclusion

In order to work with Kubernetes, a active cluster is needed. To do this we need several running host servers, or computers. Another option is we could spin up nodes from a cloud provider like, AWS or Google Cloud Service. However, we have bypassed the cost and commitment of those options by using Kuberenetes' Minikube. In this lesson, we have setup Minikube, which will serve as our cluster. Minikube is a tool, we have set it up, configured it, and now we will use it to complete future labs.