

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

Step 1. Install MiniKube

Mac

```
$ brew cask install minikube
```

Linux

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

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 Virtualbox 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...
```

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

Output:

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

```
$ 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 --watch
```

Output:

NAME		READY	STATUS
RESTARTS	AGE		
hello-minikube-3383150820-vctvh	0/1	ContainerCr	
eating 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	R
ESTARTS	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. Dashboard-To access the Kubernetes Dashboard, run this command in a shell after starting minikube to get the address:

```
minikube dashboard
```

8. To stop the minikube cluster, run the following command:

```
$ minikube stop
```

Output:

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

9. Delete the deployment, run the following command:

```
$ kubectl delete deployment hello-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.

->

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 Kubernetes' 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.