Exploring Minikube Commands

Let's learn about some other useful Minikube commands.

WE'LL COVER THE FOLLOWING

Checking the Environment Variables

Another useful Minikube command to output the environment variables is `docker-env`:

```
minikube docker-env
```

The **output** is as follows.

```
export DOCKER_TLS_VERIFY="1"
export DOCKER_HOST="tcp://192.168.99.100:2376"
export DOCKER_CERT_PATH="/Users/educative/.minikube/certs"
export DOCKER_API_VERSION="1.23"
# Run this command to configure your shell:
# eval $(minikube docker-env)
```

If you have worked with Docker Machine, you'll notice that the output is the same. Both `docker-machine env` and `minikube docker-env` serve the same purpose. They output the environment variables required for a local Docker purpose. They output the environment variables required for a local Docker

client to communicate with a remote Docker server. In this case, that Docker server is the one inside a VM created by Minikube.

> 📝 **Note :** We assume that you already have Docker installed on your machine. If that's not the case, please go to the Install Docker page and follow the instructions for your operating system.

Configuring the Shell

Once Docker is installed, we can connect the client running on your laptop with the server in the Minikube VM.

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

We evaluated (created) the environment variables provided through the `minikube docker-env` command. As a result, every command we send to our local Docker client will be executed on the Minikube VM.

> 📝 The above command will not result in any output to the console.

Verification

We can test that easily by, for example, listing all the running containers on that VM.

```
docker container ls
```

The containers listed in the output are those required by Kubernetes. We can, in a way, consider them system containers. We won't discuss each of them. As a matter of fact, we won't discuss any of them. At least, not right away. All you need to know, at this point, is that they make Kubernetes work.

Since almost everything in that VM is a container, pointing the local Docker client to the service inside, it should be all you need (besides `kubectl`). Still, in some cases, you might want to SSH into the VM.

some cases, you might want to SSH into the VM.

```
minikube ssh
docker container ls
exit
```

We entered into the Minikube VM, listed containers, and got out. There's no reason to do anything else beyond showing that SSH is possible, even though you probably won't use it.

What else is there to verify? We can, for example, confirm that `kubectl` is also pointing to the Minikube VM.

```
kubectl config current-context
```

The **output** should be a single word, `minikube`, indicating that `kubectl` is configured to talk to Kubernetes inside the newly created cluster.

As an additional verification, we can list all the nodes of the cluster.

```
kubectl get nodes
```

The **output** is as follows.

```
NAME     STATUS ROLES  AGE VERSION
minikube Ready  master 31m v1.14.0
```

It should come as no surprise that there is only one node, conveniently called `minikube`.

If you are experienced with Docker Machine or Vagrant, you probably noticed a similar pattern. Minikube commands are almost exactly the same as those from Docker Machine which, on the other hand, are similar to those from Vagrant.

Let's make a sneak peek into the components currently running in our tiny cluster.

```
kubectl get all --all-namespaces
```

Behold, the cluster in all its glory. It's made out of many building blocks we are yet to explore. Moreover, those are only the beginning. We'll be adding more as our needs and knowledge increase. For now, remember that there are many moving pieces. We won't go into details just yet. That would be too much to start with.

Common Minikube Commands

Going back to minikube, we can do all the common things we would expect from a virtual machine.

Stopping Minikube

The `minikube stop` command can be used to stop your cluster. This command shuts down the Minikube Virtual Machine, but preserves all cluster state and data. Starting the cluster again will restore it to its previous state.

```
minikube stop
```

Starting Minikube

We can *start* it again.

```
minikube start
```

Deleting Minikube

The `minikube delete` command can be used to delete the cluster. This command shuts down and deletes the Minikube Virtual Machine. No data or state is preserved.

```
minikube delete
```

Specifying the Kubernetes Version

One interesting feature is the ability to specify which Kubernetes version we'd like to use.

Since Kubernetes is still a young project, we can expect quite a lot of changes at a rapid pace. That will often mean that our production cluster might not be running the latest version. On the other hand, we should strive to have our local environment as close to production as possible (within reason).

We can create a new cluster based on, let's say, Kubernetes **v1.9.4**.

```
minikube start \
    --vm-driver=virtualbox \
    --kubernetes-version="v1.14.0"
```

We created a new cluster. Let's output versions of the client and the server.

```
    kubectl version --output=yaml
```

The **output** of the command is as follows.

```
clientVersion:
  buildDate: "2019-04-08T17:11:31Z"
  compiler: gc
  gitCommit: ...
  gitTreeState: clean
  gitVersion: v1.14.1
  goVersion: go1.12.1
  major: "1"
  minor: "11"
  platform: darwin/amd64
serverVersion:
  buildDate: "2019-03-25T15:45:25Z"
  compiler: gc
  gitCommit: ...
  gitTreeState: clean
  gitVersion: v1.14.0
  goVersion: go1.12.1
  major: "1"
  minor: "9"
  platform: linux/amd64
```

If you focus on the `serverVersion` section, you'll notice that the `major` version is `1` and the `minor` is `9`. The version may be different on your machine as the versions keep changing with time.

In the next lesson, we will test our understanding of this chapter with the help of a quick quiz.