Lab 28 - Kubernetes

* For a kubernetes cluster, we'll be using a tool called minikube. The minikube tool makes it easy to run a single-node kubernetes cluster on a VM that can be run on a laptop or development machine. Install minikube. On macOS X, you can use HomeBrew to do this:

**$ brew install minikube**

* We'll also be using the kubernetes CLI tools in this recipe, so install those. On macOS X, using HomeBrew, you can type as follows:

**$ brew install kubernetes-cli**

* Now we're ready to start our single-node kubernetes cluster. You can do this by running minikube start:

**$ minikube start**  
**Starting local Kubernetes v1.10.0 cluster...**  
**Starting VM...**  
**Getting VM IP address...**  
**Moving files into cluster...**  
**Setting up certs...**  
**Connecting to cluster...**  
**Setting up kubeconfig...**  
**Starting cluster components...**  
**Kubectl is now configured to use the cluster.**  
**Loading cached images from config file**

* Next, set the minikube cluster up as the default configuration for the kubectl CLI tool:

**$ kubectl config use-context minikube**  
**Switched to context "minikube".**

* Verify that everything is configured properly by running the cluster-info command:

**$ kubectl cluster-info**  
**Kubernetes master is running at https://192.168.99.100:8443**  
**KubeDNS is running at https://192.168.99.100:8443/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy**

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

* You should now be able to launch the kubernetes dashboard in a browser:

**$ minikube dashboard**  
**Waiting, endpoint for service is not ready yet...**  
**Opening kubernetes dashboard in default browser...**

* The minikube tool uses a number of environment variables to configure the CLI client. Evaluate the environment variables with the following command:

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

* Next, we'll build the docker image for our service using the Dockerfile file created in the previous recipe:

**$ docker build -t message-service:0.1.1**

* Finally, run the message-service command on the kubernetes cluster, telling kubectl the correct image to use and the port to expose:

**$ kubectl run message-service --image=message-service:0.1.1 --port=8082 --image-pull-policy=Never**

* We can verify that the message-service command is running in the kubernetes cluster by listing the pods on the cluster:

**$ kubectl get pods**  
**NAME READY STATUS RESTARTS AGE**  
**message-service-87d85dd58-svzmj 1/1 Running 0 3s**

* In order to access the message-service command, we'll need to expose it as a new service:

**$ kubectl expose deployment message-service --type=LoadBalancer**  
**service/message-service exposed**

* We can verify the previous command by listing services on the kubernetes services:

**$ kubectl get services**  
  
**NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE**  
**kubernetes ClusterIP 10.96.0.1 <none> 443/TCP 59d**  
**message-service LoadBalancer 10.105.73.177 <pending> 8082:30382/TCP 4s**

* The minikube tool has a convenient command for accessing a service running on the kubernetes cluster. Running the following command will list the URL that the message-service command is running on:

**$ minikube service list message-service**  
**|-------------|----------------------|-----------------------------|**  
**| NAMESPACE | NAME | URL |**  
**|-------------|----------------------|-----------------------------|**  
**| default | kubernetes | No node port |**  
**| default | message-service | http://192.168.99.100:30382 |**  
**| kube-system | kube-dns | No node port |**  
**| kube-system | kubernetes-dashboard | http://192.168.99.100:30000 |**  
**|-------------|----------------------|-----------------------------|**

* Use curl to try and make a request against the service to verify that it's working. Congratulations! You've deployed the message-service command on kubernetes.