**Build and install a node service with Docker**

This last example is a bit more complex and requires the creation of a small node server, build it into a Docker image with a Dockerfile, and run it in Kubernetes.

Let’s create a simple node project *hello-node:*

$ mkdir hello-node && cd hello-node && touch Dockerfile server.js

$ tree  
.  
├── Dockerfile  
└── server.js

Let’s create a simple http server that returns a **Hello World!** response:

$ vi server.js

var http = require('http');  
var handleRequest = function(request, response) {  
 response.writeHead(200);  
 response.end('Hello World!');  
};  
var helloServer = http.createServer(handleRequest);  
helloServer.listen(8080);

Let’s edit the Dockerfile to declare t

Let’s edit the Dockerfile to declare this image will use node 4.4, and the container will run the service by executing the server.js file:

$ vi Dockerfile

FROM node:4.4  
EXPOSE 8080  
COPY server.js .  
CMD node server.js

Before issuing any Docker commands, let’s set the Docker environment. Similarly to running eval $(docker-machine env), we generate Docker environment variables for the minikube runtime using the command minikube docker-env:

$ eval $(minikube docker-env)

Now, let’s build the image. This will take some time, as it will fetch images for dependencies like node 4.4 from the Docker hub. Once finished, you’ll have a new Docker image ready to deploy (note the trailing dot `.` in the command: this tells Docker to build the current directory):

$ docker build -t hello-node:v1 .  
Sending build context to Docker daemon 6.144 kB  
Step 1 : FROM node:4.4  
 ---> abb6383ef5fe  
Step 2 : EXPOSE 8080  
 ---> Running in a4330e4790a7  
 ---> a1f021471d1c  
Removing intermediate container a4330e4790a7  
Step 3 : COPY server.js .  
 ---> 1b394ed6fbaf  
Removing intermediate container 31c5e5e505bf  
Step 4 : CMD node server.js  
 ---> Running in 6e62fbd3174f  
 ---> ff428ba5193c  
Removing intermediate container 6e62fbd3174f  
Successfully built ff428ba5193c

$ docker images  
REPOSITORY TAG IMAGE ID CREATED SIZE  
**hello-node v1 ff428ba5193c 20 seconds ago 656.9 MB**gcr.io/google\_containers/kubernetes-dashboard-amd64 v1.1.0 d023c050c065 10 days ago 58.65 MB  
node 4.4 abb6383ef5fe 3 weeks ago 656.9 MB  
gcr.io/google\_containers/echoserver 1.4 a90209bb39e3 8 weeks ago 140.4 MB  
gcr.io/google-containers/kube-addon-manager-amd64 v2 a876fb07f9c2 9 weeks ago 231.1 MB  
gcr.io/google\_containers/pause-amd64 3.0 99e59f495ffa 11 weeks ago 746.9 kB

Now we can deploy the hello-node pod to our local Kubernetes cluster via kubectl:

$ kubectl run hello-node --image=hello-node:v1 --port=8080  
deployment "hello-node" created

$ kubectl get pods  
NAME READY STATUS RESTARTS AGE  
hello-node-2686040790-0t8q4 1/1 Running 0 1m

$ kubectl get deployments  
NAME DESIRED CURRENT UP-TO-DATE AVAILABLE AGE  
hello-node 1 1 1 1 1m

As before, we must expose the deployment to an external IP address and port in order to access it via curl:

$ kubectl expose deployment hello-node --type=NodePort  
service "hello-node" exposed

$ kubectl get services  
NAME CLUSTER-IP EXTERNAL-IP PORT(S) AGE  
hello-node 10.0.0.77 <nodes> 8080/TCP 12s  
kubernetes 10.0.0.1 <none> 443/TCP 3d

$ curl $(minikube service hello-node --url)  
Hello World!%

#### Cleanup

Let’s not forget to delete the service and deployment for hello-node, and shut down the minikube cluster once finished:

$ kubectl delete service,deployment hello-node  
service "hello-node" deleted  
deployment "hello-node" deleted

$ minikube stop  
Stopping local Kubernetes cluster...  
Machine stopped.