Kubernetes App Deployment

Objectives

- Run a hello world Node.js application.
- Deploy the application to Minikube.
- View application logs.
- Update the application image.

Create a Minikube cluster

Determine whether you can access sites like https://cloud.google.com/container-registry/ directly without a proxy, by opening a new terminal and using

```
curl --proxy "" https://cloud.google.com/container-registry/
```

Make sure that the Docker daemon is started. You can determine if docker is running by using a command such as:

```
docker images
```

If NO proxy is required, start the Minikube cluster:

```
minikube start --vm-driver=hyperkit
```

If a proxy server is required, use the following method to start Minikube cluster with proxy setting:

```
minikube start --vm-driver=hyperkit --docker-env HTTP_PROXY=http://your-http-proxy-host:your-http-proxy-port  --docker-env HTTPS_PROXY=http(s)://your-https-proxy-host:your-https-proxy-port
```

The `--vm-driver=hyperkit` flag specifies that you are using Docker for Mac. The default VM driver is VirtualBox.

Now set the Minikube context. The context is what determines which cluster `kubectl` is interacting with. You can see all your available contexts in the `~/.kube/config` file.

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

Verify that `kubectl` is configured to communicate with your cluster:

```
kubectl cluster-info
```

Open the Kubernetes dashboard in a browser:

```
minikube dashboard
```

Create your Node.js application

The next step is to write the application. Save this code in a folder named `hellonode` with the filename `server.js`:

minikube/server.js

```
var http = require('http');


var handleRequest = function(request, response) {
  console.log('Received request for URL: ' + request.url);
  response.writeHead(200);
  response.end('Hello World!');
```

minikube/server.js

```
};

var www = http.createServer(handleRequest);

www.listen(8080);
```

Run your application:

```
node server.js
```

You should be able to see your “Hello World!” message at http://localhost:8080/.

Stop the running Node.js server by pressing Ctrl-C.

The next step is to package your application in a Docker container.

Create a Docker container image

Create a file, also in the `hellonode` folder, named `Dockerfile`. A Dockerfile describes the image that you want to build. You can build a Docker container image by extending an existing image. The image in this tutorial extends an existing Node.js image.

minikube/Dockerfile

```
FROM node:6.9.2

EXPOSE 8080

COPY server.js .

CMD node server.js
```

This recipe for the Docker image starts from the official Node.js LTS image found in the Docker registry, exposes port 8080, copies your `server.js` file to the image and starts the Node.js server.

Because this tutorial uses Minikube, instead of pushing your Docker image to a registry, you can simply build the image using the same Docker host as the Minikube VM, so that the images are automatically present. To do so, make sure you are using the Minikube Docker daemon:

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

Note: Later, when you no longer wish to use the Minikube host, you can undo this change by running `eval $(minikube docker-env -u)`.

Build your Docker image, using the Minikube Docker daemon (mind the trailing dot):

```
docker build -t hello-node:v1 .
```

Now the Minikube VM can run the image you built.

Create a Deployment

A Kubernetes *Pod* is a group of one or more Containers, tied together for the purposes of administration and networking. The Pod in this tutorial has only one Container. A Kubernetes *Deployment* checks on the health of your Pod and restarts the Pod's Container if it terminates. Deployments are the recommended way to manage the creation and scaling of Pods.

Use the `kubectl run` command to create a Deployment that manages a Pod. The Pod runs a Container based on your `hello-node:v1` Docker image. Set the `--image-pull-`

`policy` flag to `Never` to always use the local image, rather than pulling it from your Docker registry (since you haven't pushed it there):

```
kubectl run hello-node --image=hello-node:v1 --port=8080 --image-pull-policy=Never
```

View the Deployment:

```
kubectl get deployments
```

Output:

```
NAME         DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
hello-node   1         1         1            1           3m
```

View the Pod:

```
kubectl get pods
```

Output:

```
NAME                          READY     STATUS    RESTARTS   AGE
hello-node-714049816-ztzrb    1/1       Running   0          6m
```

View cluster events:

```
kubectl get events
```

View the `kubectl` configuration:

```
kubectl config view
```

For more information about `kubectl`commands, see the kubectl overview.

Create a Service

By default, the Pod is only accessible by its internal IP address within the Kubernetes cluster. To make the `hello-node` Container accessible from outside the Kubernetes virtual network, you have to expose the Pod as a Kubernetes *Service*.

From your development machine, you can expose the Pod to the public internet using the `kubectl expose` command:

```
kubectl expose deployment hello-node --type=LoadBalancer
```

View the Service you just created:

```
kubectl get services
```

Output:

```
NAME         TYPE         CLUSTER-IP    EXTERNAL-IP    PORT(S)    AGE
hello-node   ClusterIP    10.0.0.71     <pending>      8080/TCP   6m
kubernetes   ClusterIP    10.0.0.1      <none>         443/TCP    14d
```

The `--type=LoadBalancer` flag indicates that you want to expose your Service outside of the cluster. On cloud providers that support load balancers, an external IP address would be provisioned to access the Service. On Minikube, the `LoadBalancer` type makes the Service accessible through the `minikube service` command.

```
minikube service hello-node
```

This automatically opens up a browser window using a local IP address that serves your app and shows the “Hello World” message.

Assuming you’ve sent requests to your new web service using the browser or curl, you should now be able to see some logs:

```
kubectl logs <POD-NAME>
```

Update your app

Edit your `server.js` file to return a new message:

```
response.end('Hello World Again!');
```

Build a new version of your image (mind the trailing dot):

```
docker build -t hello-node:v2 .
```

Update the image of your Deployment:

```
kubectl set image deployment/hello-node hello-node=hello-node:v2
```

Run your app again to view the new message:

```
minikube service hello-node
```

Enable addons

Minikube has a set of built-in addons that can be enabled, disabled and opened in the local Kubernetes environment.

First list the currently supported addons:

```
minikube addons list
```

Output:

```
- storage-provisioner: enabled
- kube-dns: enabled
- registry: disabled
- registry-creds: disabled
- addon-manager: enabled
- dashboard: disabled
- default-storageclass: enabled
- coredns: disabled
- heapster: disabled
- efk: disabled
- ingress: disabled
```

Minikube must be running for these commands to take effect. To enable `heapster` addon, for example:

```
minikube addons enable heapster
```

Output:

```
heapster was successfully enabled
```

View the Pod and Service you just created:

```
kubectl get po,svc -n kube-system
```

Output:

```
NAME                                    READY     STATUS    RESTARTS   AGE
pod/heapster-zbwzv                      1/1       Running   0          2m
pod/influxdb-grafana-gtht9              2/2       Running   0          2m

NAME                            TYPE        CLUSTER-IP   EXTERNAL-IP
PORT(S)             AGE
service/heapster                NodePort    10.0.0.52    <none>
80:31655/TCP        2m
service/monitoring-grafana      NodePort    10.0.0.33    <none>
80:30002/TCP        2m
service/monitoring-influxdb     ClusterIP   10.0.0.43    <none>
8083/TCP,8086/TCP   2m
```

Open the endpoint to interacting with heapster in a browser:

```
minikube addons open heapster
```

Output:

```
Opening kubernetes service kube-system/monitoring-grafana in default
browser...
```

Clean up

Now you can clean up the resources you created in your cluster:

```
kubectl delete service hello-node

kubectl delete deployment hello-node
```

Optionally, force removal of the Docker images created:

```
docker rmi hello-node:v1 hello-node:v2 -f
```

Optionally, stop the Minikube VM:

```
minikube stop

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

Optionally, delete the Minikube VM:

```
minikube delete
```