Now let’s push this to Docker Hub. If you haven’t logged into the Docker Hub via your command line, you must do this now, and enter your username and password:

$ docker login

Login with your Docker ID to push and pull images from Docker Hub. If you don't have a Docker ID, head over to https://hub.docker.com to create one.

Username:

Password:

Login Succeeded

$

$ docker push danielbryantuk/djshopfront:1.0

The push refers to a repository [docker.io/danielbryantuk/djshopfront]

9b19f75e8748: Pushed

...

cf4ecb492384: Pushed

1.0: digest: sha256:8a6b459b0210409e67bee29d25bb512344045bd84a262ede80777edfcff3d9a0 size: 2210

**Deploying onto Kubernetes**

Now let’s run this container within Kubernetes. First, change the “kubernetes” directory in the root of the project:

$ cd ../kubernetes

Open the shopfront-service.yaml Kubernetes deployment file and have a look at the contents:

---

apiVersion: v1

kind: Service

metadata:

name: shopfront

labels:

app: shopfront

spec:

type: NodePort

selector:

app: shopfront

ports:

- protocol: TCP

port: 8010

name: http

---

apiVersion: v1

kind: ReplicationController

metadata:

name: shopfront

spec:

replicas: 1

template:

metadata:

labels:

app: shopfront

spec:

containers:

- name: shopfront

image: danielbryantuk/djshopfront:latest

ports:

- containerPort: 8010

livenessProbe:

httpGet:

path: /health

port: 8010

initialDelaySeconds: 30

timeoutSeconds: 1

The first section of the yaml file creates a Service named “shopfront” that will route TCP traffic targeting this service on port 8010 to pods with the label “app: shopfront”. The second section of the configuration file creates a ReplicationController that specifies Kubernetes should run one replica (instance) of our shopfront container, which we have declared as part of the “spec” (specification) labelled as “app: shopfront”. We have also specified that the 8010 application traffic port we exposed in our Docker container is open, and declared a “livenessProbe” or healthcheck that Kubernetes can use to determine if our containerized application is running correctly and ready to accept traffic. Let’s start minikube and deploy this service (note that you may need to change the specified minikube CPU and Memory requirements depending on the resources available on your development machine):

$ minikube start --cpus 2 --memory 4096

Starting local Kubernetes v1.7.5 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.

$ kubectl apply -f shopfront-service.yaml

service "shopfront" created

replicationcontroller "shopfront" created

You can view all Services within Kubernetes by using the “kubectl get svc” command. You can also view all associated pods by using the “kubectl get pods” command (note that the first time you issue the get pods command, the container may not have finished creating, and is marked as not yet ready):

$ kubectl get svc

NAME CLUSTER-IP EXTERNAL-IP PORT(S) AGE

kubernetes 10.0.0.1 <none> 443/TCP 18h

shopfront 10.0.0.216 <nodes> 8010:31208/TCP 12s

$ kubectl get pods

NAME READY STATUS RESTARTS AGE

shopfront-0w1js 0/1 ContainerCreating 0 18s

$ kubectl get pods

NAME READY STATUS RESTARTS AGE

shopfront-0w1js 1/1 Running 0 2m

We have now successfully deployed our first Service into Kubernetes!

**Time for a smoke test**

Let’s use curl to see if we can get data from the shopfront application’s healthcheck endpoint:

$ curl $(minikube service shopfront --url)/health

{"status":"UP"}

You can see from the results of the curl against the application/health endpoint that the application is up and running, but we need to deploy the remaining microservice application containers before the application will function as we expect it to.

**Building the remaining applications**

Now that we have one container up and running let’s build the remaining two supporting microservice applications and containers:

$ cd ..

$ cd productcatalogue/

$ mvn clean install

…

$ docker build -t danielbryantuk/djproductcatalogue:1.0 .

...

$ docker push danielbryantuk/djproductcatalogue:1.0

...

$ cd ..

$ cd stockmanager/

$ mvn clean install

...

$ docker build -t danielbryantuk/djstockmanager:1.0 .

...

$ docker push danielbryantuk/djstockmanager:1.0

…

At this point we have built all of our microservices and the associated Docker images, and also pushed the images to Docker Hub. Let’s now deploy the productcatalogue and stockmanager services to Kubernetes.

**Deploying the entire Java application in Kubernetes**

In a similar fashion to the process we used above to deploy the shopfront service, we can now deploy the remaining two microservices within our application to Kubernetes:

$ cd ..

$ cd kubernetes/

$ kubectl apply -f productcatalogue-service.yaml

service "productcatalogue" created

replicationcontroller "productcatalogue" created

$ kubectl apply -f stockmanager-service.yaml

service "stockmanager" created

replicationcontroller "stockmanager" created

$ kubectl get svc

NAME CLUSTER-IP EXTERNAL-IP PORT(S) AGE

kubernetes 10.0.0.1 <none> 443/TCP 19h

productcatalogue 10.0.0.37 <nodes> 8020:31803/TCP 42s

shopfront 10.0.0.216 <nodes> 8010:31208/TCP 13m

stockmanager 10.0.0.149 <nodes> 8030:30723/TCP 16s

$ kubectl get pods

NAME READY STATUS RESTARTS AGE

productcatalogue-79qn4 1/1 Running 0 55s

shopfront-0w1js 1/1 Running 0 13m

stockmanager-lmgj9 1/1 Running 0 29s

Depending on how quickly you issue the “kubectl get pods” command, you may see that all of the pods are not yet running. Before moving on to the next section of this article wait until the command shows that all of the pods are running (maybe this is a good time to brew a cup of tea!)

**Viewing the complete application**

With all services deployed and all associated pods running, we now should be able to access our completed application via the shopfront service GUI. We can open the service in our default browser by issuing the following command in minikube:

$ minikube service shopfront

If everything is working correctly, you should see the following page in your browser: