Your supermarket company is in the process of moving their infrastructure to a Kubernetes platform in the cloud. This is sometimes challenging, because some of the older, legacy portions of that infrastructure have non-standard requirements. One of these legacy applications is a web service that provides a list of the various types of fruit the company sells in its stores.

This service has already been packaged into a container image, but there is one special requirement: The legacy app is hard-coded to only serve content on port 8775, but the team wants to be able to access the service using the standard port 80. Your task is to build a Kubernetes pod that runs this legacy container and uses the ambassador design pattern to expose access to the service on port 80.

This setup will need to meet the following specifications:

* The pod should have the name fruit-service.
* The fruit-service pod should have a container that runs the legacy fruit service image: linuxacademycontent/legacy-fruit-service:1.
* The fruit-service pod should have an ambassador container that runs the haproxy:1.7 image and proxies incoming traffic on port 80 to the legacy service on port 8775 (the HAProxy configuration for this is provided below).
* Port 80 should be exposed as a containerPort. Note that you do not need to expose port 8775.
* The HAProxy configuration should be stored in a ConfigMap called fruit-service-ambassador-config.
* The HAProxy config should be provided to the ambassador container using a volume mount that places the data from the ConfigMap in a file at /usr/local/etc/haproxy/haproxy.cfg.
* haproxy.cfg should contain the following configuration data:

global

daemon

maxconn 256

defaults

mode http

timeout connect 5000ms

timeout client 50000ms

timeout server 50000ms

listen http-in

bind \*:80

server server1 127.0.0.1:8775 maxconn 32

Once your pod is up and running, it's a good idea to test it to make sure you can access the service from within the cluster using port 80. In order to do this, you can create a busybox pod in the cluster, and then run a command to attempt to access the service from within the busybox pod.

Create a descriptor for the busybox pod called busybox.yml.

apiVersion: v1

kind: Pod

metadata:

name: busybox

spec:

containers:

- name: myapp-container

image: radial/busyboxplus:curl

command: ['sh', '-c', 'while true; do sleep 3600; done']

Create the busybox testing pod.

kubectl apply -f busybox.yml

Use this command to access fruit-service using port 80 from within the busybox pod.

kubectl exec busybox -- curl $(kubectl get pod fruit-service -o=custom-columns=IP:.status.podIP --no-headers):80

If the service is working, you should see some JSON listing various types of fruit.

Solution:

**Forwarding Port Traffic with an Ambassador Container**

**Introduction**

Welcome to the lab. Pull up a chair. What we're essentially going to do here is implement a multi-container pod using the Ambassador design pattern. We've got a legacy app that *only* listens on port **8775**. Instead of opening up a special port though, we're just going to allow the regular **80** in from the internet. Our setup will still aim any traffic bound for the legacy app to it, but it will be on **8775**, internally, *after* it has come in from the internet on *80*.

Let's use the credentials provided on the Linux Academy lab page, and log into the Kubernetes Master.

**Create a ConfigMap Containing the Configuration for the HAProxy Ambassador**

Create a YAML definition file called fruit-service-ambassador-config.yml using vi or whichever text editor you like. It needs to contain the following text:

apiVersion: v1

kind: ConfigMap

metadata:

name: fruit-service-ambassador-config

data:

haproxy.cfg: |-

global

daemon

maxconn 256

defaults

mode http

timeout connect 5000ms

timeout client 50000ms

timeout server 50000ms

listen http-in

bind \*:80

server server1 127.0.0.1:8775 maxconn 32

Create the ConfigMap in the cluster from the YAML definition file:

kubectl apply -f fruit-service-ambassador-config.yml

**Create a Multi-Container Pod Which Provides Access to the Legacy Service on Port 80**

Create a YAML definition file for the pod called fruit-service.yml with these contents:

apiVersion: v1

kind: Pod

metadata:

name: fruit-service

spec:

containers:

- name: legacy-fruit-service

image: linuxacademycontent/legacy-fruit-service:1

- name: haproxy-ambassador

image: haproxy:1.7

ports:

- containerPort: 80

volumeMounts:

- name: config-volume

mountPath: /usr/local/etc/haproxy

volumes:

- name: config-volume

configMap:

name: fruit-service-ambassador-config

Create the pod in the cluster:

kubectl apply -f fruit-service.yml

Let's see if it spun up with a quick kubectl get pods. It may take a few seconds.

**Testing**

Now, let's check on whether or not we can access fruit-service from another pod, on port *80*

We can create a busybox pod to use for testing by first creating a file called busybox.yml, with these contents:

apiVersion: v1

kind: Pod

metadata:

name: busybox

spec:

containers:

- name: myapp-container

image: radial/busyboxplus:curl

command: ['sh', '-c', 'while true; do sleep 3600; done']

Now let's create the busybox testing pod:

kubectl apply -f busybox.yml

We should check to see if it's running with another kubectl get pods.

We're going to use the busybox pod to test the legacy service on port 80. This command uses a subcommand to get the cluster's IP address for the pod, then executes a curl command in the busybox pod to access the legacy service on port 80:

kubectl exec busybox -- curl $(kubectl get pod fruit-service -o=custom-columns=IP:.status.podIP --no-headers):80

If everything is working, we should see some JSON listing various types of fruit.