Building Self-Healing Containers in Kubernetes

You are working for BeeBox, a company that provides regular shipments of bees to customers. The company is in the process of deploying some applications to Kubernetes that handle data processing related to shipping.

One of these application components is a pod called beebox-shipping-data located in the default namespace. Unfortunately, the application running in this pod has been crashing repeatedly. While the developers are looking into why this application is crashing, you have been asked to implement a self-healing solution in Kubernetes to quickly recover whenever the application crashes.

Luckily, the application can be fixed when it crashes simply by restarting the container. Modify the pod configuration so the application will automatically restart when it crashes. You can detect an application crash then requests to port 8080 on the container return an HTTP 500 status code.

Set a Restart Policy to Restart the Container When It Is Down

We will start off by seeing what and how many pods we are working with

1. Checking the pods information

kubectl get pods -o wide

Figure 1-1

```
cloud_user@k8s-control:~$ kubectl get pods -o wide
NAME                  READY   STATUS    RESTARTS   AGE   IP               NODE
beebox-shipping-data  1/1     Running   0          48m   192.168.194.65   k8s-worker1
busybox               1/1     Running   0          48m   192.168.194.66   k8s-worker1
```

We want to verify the beebox shipping data pod status. Yes it does saying it is in the running state but that doesn't mean everything is really working as expected.

2. Make a request from busybox pod to beebox pod

kubectl exec busybox -- curl 192.168.194.65

Figure 1-2

```
cloud_user@k8s-control:~$ kubectl exec busybox -- curl 192.168.194.65:8080
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100    22    0    22    0     0   1586      0 --:--:-- --:--:-- --:--:--  1692
Internal server error.cloud_user@k8s-control:~$
```

So it does look like something is actually a miss even though the pod is up and running. And obviously there are some internal problems that the pod needs to be restarted in order to fix that.

We could restart the pod and that could fix the pod. But in this lab we are being asked to come up with an automated solution. So we have to check the restart policy and make sure it is appropriate to do what we needs to be done.

3. We need to get a YAML descriptor for that pod

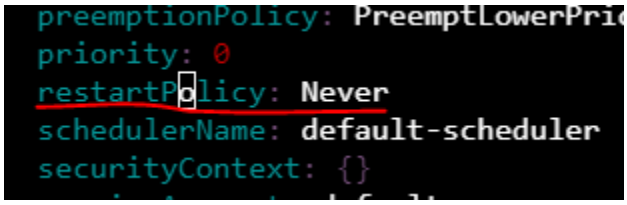
kubectl get pod beebox-shipping-data -o yaml > beebox-shipping-data.yml

4. One that command has been entered we now have to edit the beebox-shipping yml file

vi beebox-shipping-data.yml

Here we need to look for the pods restart policy which is under spec:

Figure 1-3

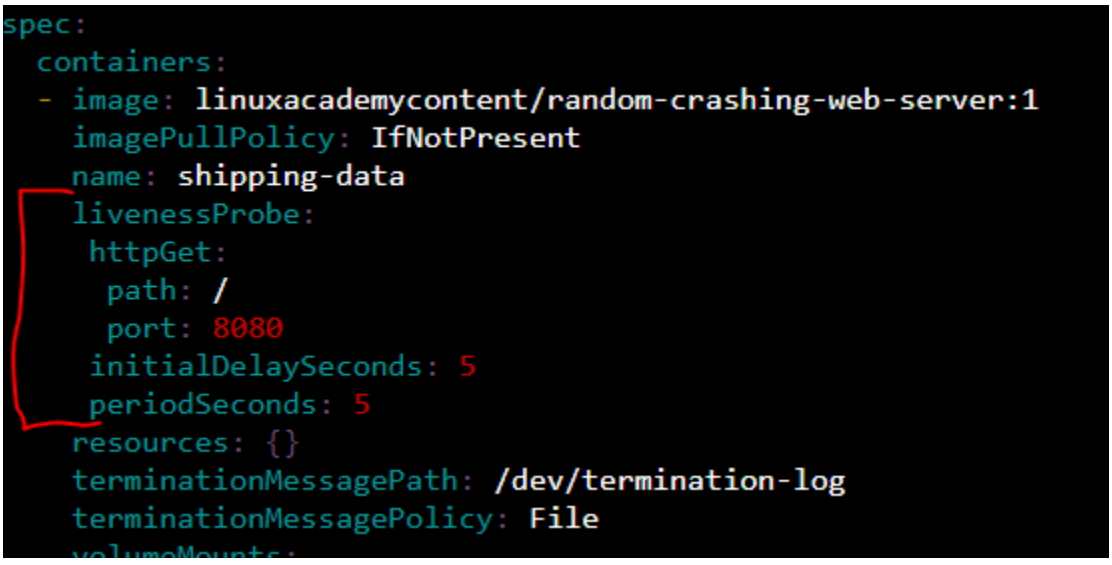


^ With a reset policy set to "Never" that means the pod is never going to restart even if the K8s cluster is able to detect a problem within the container. So we change the never to "Always"

Since we have set the restart policy, the next step would be to tell cluster what to do in the event when a container is broken. But in order for that to work. We need the cluster to actually be able to detect when the container is broken.

5. Go under the container and we have to modify the first container to add a livenessProbe to detect when there's a problem

Figure 1-4



Above on the bracket on the left, this is what I have added in to the yml file

- Since this container is a web server we will use an httpGet: livenessProbe
    - Which means in order to test the container the Kubernetes cluster is going to make a request to that container and basically determine if the container is healthy based on that response
- The initialDelaySeconds is the number of seconds that the cluster is going to wait to begin running the livenessProbe when the pod starts up
- The periodSeconds which is how often will this livenessProbe will run
    - So in this case we put 5. So every 5 seconds, a request is going to be made to this container at the root path, port number: 8080

6. In order to make a change to our existing pod, we can't update the pod, we need to delete and recreate it

kubectl delete pod beebox-shipping-data

7. Once deleted do the apply -f command

kubectl apply -f beebox-shipping-data.yml

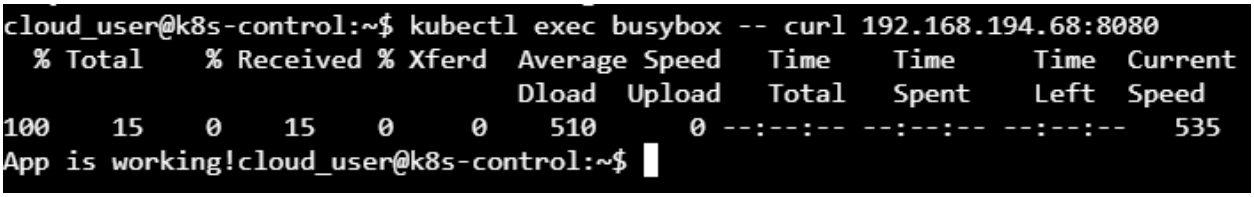
8. Once that is created see the pods and the status (-o wide command)

*The IP address should have changed since we deleted the pod and recreated it

9. Run the curl command with the IP on the beebox pod (*Note… The IP may have changed since we created a new one; similar to step 2)

kubectl exec busybox -- curl 192.168.194.68:8080

Figure 1-5



^Above you can see the app is working and we do not have an internal error like before

10. Verify the pods restart

kubectl get pods

This is a helpful command to see what pods are having issues and what not.

Or include kubectl get pods -o wide