Exercise: Pods

Pods are the smallest, most basic deployable objects in Kubernetes. A Pod represents a single instance of a running process in your cluster. Pods contain one or more containers, such as Docker containers. Although you want deploy pods directly (static pods), knowledge for defining pods manifest files will be used for defining more complex Kubernetes resources like Controllers.

Practice1: Simple pods operations

Note: Try not to do a copy/paste on commands requests unless you are instructed to do so. Copy/paste will not help you to learn Kubernetes!

1. Login to Azure and connect to your AKS cluster.
2. Check how many pods run under the default namespace. Run **kubectl get pods**.
3. You should not see any pod under the default namespace. Now check all namespaces. Run **kubectl get pods –all-namespace**.
4. How many pods do you see? Who deployed these pods? Why are they deployed?
5. Now deploy you first pod using the imperative approach. Run **kubectl run nginx --image=nginx**.
6. Validate if the pods has been created. What is the status of your pod?
7. Check the logs coming out of your pod. Run **kubectl logs nginx**.
8. Run following command to check current resource consumption of your pod: **kubectl top pod nginx**.
9. Check on which Node your pods has been scheduled. Run **kubectl get pods –o wide**.
10. Try to find the same information but this time running **kubectl describe pod nginx.**
11. Delete your pod using **kubectl delete pod nginx.**
12. Let's find the image used on one of the coredns pods under the kube-system namespace.
13. Once again list all pods under all namespaces.
14. Note one of the coredns pods. Now run **kubectl describe pod <coredns-name> -n kube-system**. Replace the <coredns-name> place holder with noted name.
15. Inspect the output and locate the image information.
16. Now let us check the logs of the metrics-server pod. Run the same command as in step 7 but don't forget to add the namespace in which this pod is created.

Practice2: Working with pod manifest files

Note: Try not to do a copy/paste on commands requests unless you are instructed to do so.

Copy/paste will not help you to learn Kubernetes!

1. Now it is time to deploy pod using manifest file (declarative approach). Copy the following code block on your local computer in a file called redis.yaml:

```
apiVersion: v11
kind: pod
metadata:
name: static-web
labels:
```

```
role: myrole
specs:
containers:
- name: redis
image: redis123
```

2. Try to deploy the pod defined in redis.yaml. Run **kubectl create –f redis.yaml**.
3. You will receive errors on your screen. Your next task will be to correct the syntax of the code you just copied. You can use the online Kubernetes documentation or you can search the internet in general.
4. When you solve all the syntax errors your pod should be deployed but is it running? What is the status of your pod?
5. Check the events associated with this pod. Run the **kubectl describe pod static-web** command. What are the events showing? Why your pod is not running?
6. Find the correct image (check the Docker hub page) and correct it in the manifest.
7. Locate the image information and put the correct image name. Redeploy the pod (fist run **kubectl delete pod static-web** to delete the pod, then run kubectl create once again).
8. Check the status of your pod. It should be running now.
9. Now you can delete the pod. Try to delete it using the **kubectl delete –f redis.yaml**.
10. Your next task is to create and test nginx pod definition. Your definition should use the nginx official image, should use label named app with value frontend and should publish port 80. Make sure you complete this task because we will use this template in our next Labs. Your nginx pod should be running without any issues.
11. Final task of this practice will be to define pod definition with following details:
    - Image=memcached
    - Port= 11211
    - Label app=web
    - CPU request=0.35 cores
    - RAM request=0.15 GB
    - CPU limit=0.5 cores
    - Ram limit=0.25 GB
    - Restart policy=Never
12. Don’t forget to try your pod definition.

Practice3: Multi-container pods

Note: Try not to do a copy/paste on commands requests unless you are instructed to do so. Copy/paste will not help you to learn Kubernetes!

1. Once finished you can try to create multi-container pod definition. Your multi-container pod should use redis and nginx containers with port 6379 and 80 published respectively. Label name should be app with value web.
2. Note that in reality there is no sense to put the redis and nginx under the same pod but it can be done for the purpose of learning.
3. Deploy your multi-container pod. It should have running status. What is written under Ready column when you kubectl get the pods? Why your pod displays different values for ready?
4. Kubectl describe you new pod, and locate the containers section. How many containers are listed?
5. Delete all the pods under the default namespace.
6. Don't delete any of the manifest files you have created so far.

Practice4: Probes

Note: Try not to do a copy/paste on commands requests unless you are instructed to do so. Copy/paste will not help you to learn Kubernetes!

1. First we will create and test liveness probe with exec test. Create a file named probes_exec.yaml with following content:

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    test: liveness
  name: liveness-exec
spec:
  containers:
  - name: liveness
    image: k8s.gcr.io/busybox
    args:
    - /bin/sh
    - -c
    - touch /tmp/healthy; sleep 30; rm -rf /tmp/healthy; sleep 600
    livenessProbe:
      exec:
        command:
        - cat
        - /tmp/healthy
      initialDelaySeconds: 5
      periodSeconds: 5
```

2. Examine the containers args commands especially the line that start with touch. This bash pipeline will help us to test the liveness probes.
3. Run **kubectl create –f probes_exec.yaml**.

4. Run **kubectl describe pod liveness-exec** immediately after you deploy the pod. The output should indicate that no liveness probes have failed yet.
5. After 35 seconds, view the Pod events again. Run **kubectl describe pod liveness-exec.**
6. At the bottom of the output, there should be a messages indicating that the liveness probes have failed, and the containers have been killed and recreated.
7. Wait another 30 seconds, and verify that the container has been restarted. Run **kubectl get pod liveness-exec.**
8. The output should show that RESTARTS has been incremented.
9. We will continue with HTTP probe. Create file named probes_http.yaml with following content:

```yaml
apiVersion: v1
kind: Pod
metadata:
  labels:
    test: liveness
  name: liveness-http
spec:
  containers:
  - name: liveness
    image: k8s.gcr.io/liveness
    args:
    - /server
    livenessProbe:
      httpGet:
        path: /healthz
        port: 8080
        httpHeaders:
        - name: Custom-Header
          value: Awesome
      initialDelaySeconds: 3
      periodSeconds: 3
```

10. Just for your info, /healtz handler has following function implemented:

```go
http.HandleFunc("/healthz", func(w http.ResponseWriter, r *http.Request) {
    duration := time.Now().Sub(started)
    if duration.Seconds() > 10 {
        w.WriteHeader(500)
        w.Write([]byte(fmt.Sprintf("error: %v", duration.Seconds())))
    } else {
        w.WriteHeader(200)
        w.Write([]byte("ok"))
    }
})
```

11. For the first 10 seconds that the container is alive, the /healthz handler returns a status of 200. After that, the handler returns a status of 500.
12. Run **kubectl create –f probes_http.yaml.**
13. Immediately run (you only have 10 secs to run this command) **kubectl describe pod liveness-http.**
14. Your pod should be live and running.
15. After 10 seconds, view Pod events to verify that liveness probes have failed and the container has been restarted. Run again **kubectl describe pod liveness-http.**
16. You should see the same output as in step 7. Kubelet will reboot he container.
17. We continue with TCP probes. Create file named probes_tcp.yaml with following content:

```
apiVersion: v1
kind: Pod
metadata:
 name: liveness-tcp
 labels:
  app: goproxy
spec:
 containers:
 - name: goproxy
  image: k8s.gcr.io/goproxy:0.1
  ports:
  - containerPort: 8080
  livenessProbe:
   tcpSocket:
    port: 9999 #8080 is valid port
   initialDelaySeconds: 15
   periodSeconds: 20
```

18. Run **kubectl create –f probes_tcp.yaml.**
19. Immediately run (you only have 10 secs to run this command) **kubectl describe pod liveness-tcp.**
20. Your pod should be live and running.
21. After 10 seconds, view Pod events to verify that liveness probes have failed and the container has been restarted. Run again **kubectl describe pod liveness-tcp.**
22. You should see the same output as in step 7 and 16. Kubelet will reboot he container.
23. Our last job will be to define one readiness probe using HTTP test.
24. Create file named readiness_http.yaml with following content:

```
apiVersion: v1
kind: Pod
metadata:
 name: readiness-http
 labels:
  app: test
spec:
 containers:
 - name: nginx
  image: nginx
```

```
ports:
- containerPort: 80
readinessProbe:
 initialDelaySeconds: 1
 periodSeconds: 2
 timeoutSeconds: 1
 successThreshold: 1
 failureThreshold: 1
 httpGet:
  host:
  scheme: HTTP
  path: /
  httpHeaders:
  - name: Host
   value: myapplication1.com
  port: 80
```

25. Run **kubectl create –f readiness_http.yaml**.
26. Run **kubectl get pods –A** to see the status of your pod.
27. Pods and their status and ready states will be displayed; our pod should be in running state.
28. Run **kubectl describe pod readiness-http**. Examine the events for this pod. Everything should be OK.
29. Now delete the pod and edit the readiness_http.yaml so that the port parameter has 81 value.
30. Run again **kubectl create –f readiness_http.yaml.**
31. Run **kubectl get pods –A** to see the status of your pod. You should see that the pod is running but it is not in ready state.
32. Describe the pod. Run **kubectl describe pod readiness-http.**
33. From the events we can see that readiness probe failed due to the connection being refused therefore pod will not receive any traffic.
34. Delete all pods under the default namespace.
35. Don’t delete any manifest files created so far.