# 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

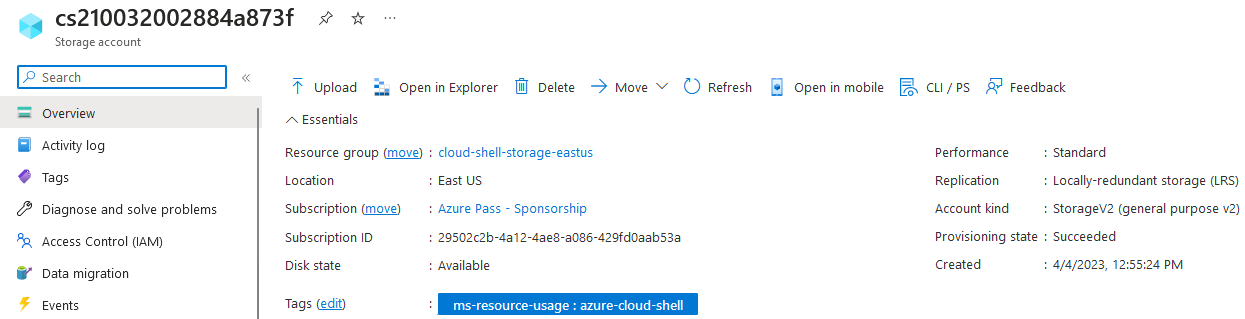
**Practice 1: Simple pods operations**

First, we create AKS cluster in the Azure portal.

Graphical user interface, text

Description automatically generated

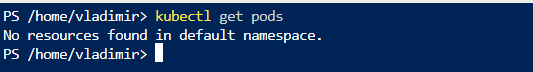
Then we create storage account so we can use Cloud Shell.



We open Cloud Shell, select PowerShell and connect it with our AKS cluster.

Import-AzAksCredential -ResourceGroupName myResourceGroup -Name myAKSCluster

First, we check to see how many pods run under the default namespace with the command **kubectl get pods**.



By default it is empty so we can check under all namespaces with the command **kubectl get pods -A (**this is short for -all-namespace). Now we can see all the deployed pods, their number may vary and in my case there were 18 pods.

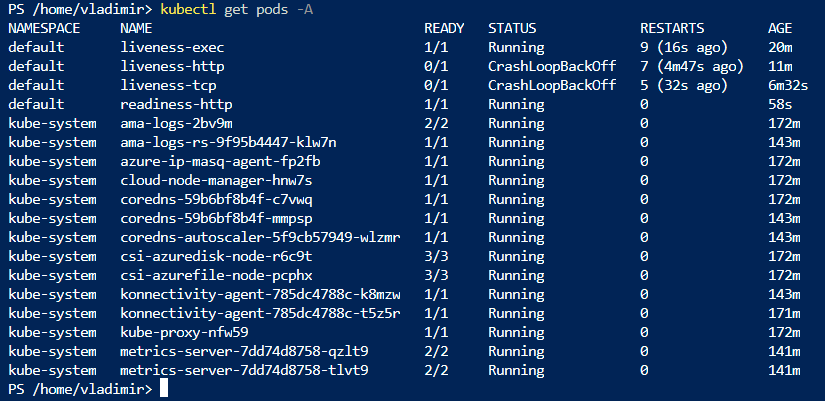
Now we deploy our first pod using the imperative approach and run the command **kubectl run nginx –image=nginx**. We can see the status of our pod and verify that it is created with running **kubectl get pods**, then we can check the logs coming out of our pod with **kubectl logs nginx**.

To check the resource consumption of our pod we run the command **kubectl top pod nginx.**

To check on which Node our pod has been scheduled we run **kubectl get pods -o wide**. We can try to find the same information running **kubectl describe pod nginx**.

We can now delete the pod using **kubectl delete pod nginx.**

Once again we list all the pods under all namespaces **kubectl get pods -A,** and we get one of the coredns pods name. We run the command **kubectl describe pod coredns-59b6bf84f-c7vwq -n kube-system**

****

We examine the output and locate the image information:

Image: mcr.microsoft.com/oss/kubernetes/coredns:v1.9.3

Image: mcr.microsoft.com/oss/kubernetes/metrics-server:v0.6.1

**Practice 2: Working with pod manifest files**

Now we need to deploy pod using manifest file. We name our first manifest file **redis.yaml**.

From the given code block we can notice that it has several errors:

* apiVersion: v11 – should be v1
* specs: - should be spec:
* The indentation is incorrect
* Image: redis123 – is not correct and will show **ImagePullBackOff**

So we make the changes and use the following specification for our manifest file:

apiVersion: v1

kind: Pod

metadata:

name: static-web

labels:

app: redis

role: myrole

spec:

containers:

- name: redis

image: asterixlegaulois/redis123

And then we execute the command **kubectl create -f redis.yaml** to create our pod. Now we check the status of our pod **kubectl get pods** and then **kubectl describe pod static-web** to see the detailed description of the pod.

I can see that my pod is running and we can now delete the pod with the command **kubectl delete -f redis.yaml**.

For our next task we need to create and test nginx pod definition.

It should:

* use the nginx official image
* use label named app with value frontend
* publish port 80

So we create a file nginx.yaml with the following details:

apiVersion: v1

kind: Pod

metadata:

name: nginx

labels:

app: frontend

spec:

containers:

- name: nginx-container

image: nginx:1.23.4

ports:

- containerPort: 80

Then we use the manifest to create the pod with the command **kubectl create -f nginx.yaml** and then check if it is runningwith **kubectl get pods.**

For the final task we create a file memcached.yaml with the following definition:

apiVersion: v1

kind: Pod

metadata:

name: memcached

spec:

containers:

- name: app

image: memcached

ports:

- containerPort: 11211

resources:

requests:

cpu: 350m

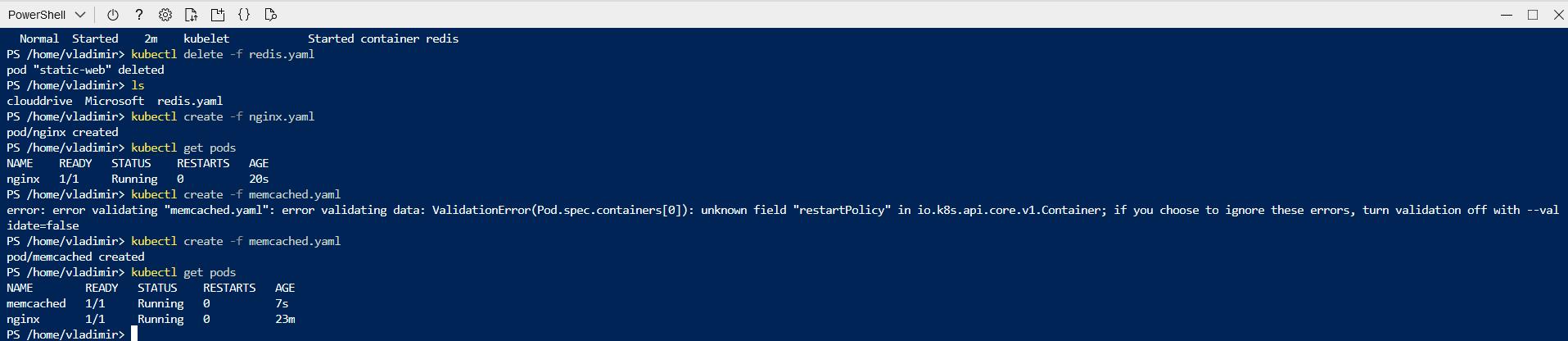
memory: 150Mi

limits:

cpu: 500m

memory: 250Mi

restartPolicy: Never



**Practice 3: Multi-container pods**

For this task I created the multi-container pod with creating a manifest with the following definition:

apiVersion: v1

kind: Pod

metadata:

name: webapp

labels:

app: web

spec:

containers:

- name: nginx

image: nginx:1.23.4

ports:

- containerPort: 80

- name: redis

image: redis:5.0.4

ports:

- containerPort: 6379

We deploy with **kubectl create -f webapp.yaml** and check the status of the pod with **kubectl get pods.** We can see that the pod displays 2 containers under the Ready column.

We can now delete all the pods within the default namespace **kubectl delete --all pods**

**Practice 4: Probes**

For this exercise we create pods from given manifests and check the output

The first manifest that we create is **probes\_exec.yaml** with the following definition:

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

We create the pod from the manifest **kubectl create -f probes\_exec.yaml** and immediately run **kubectl describe pod liveness-exec** to see that no liveness probes have failed yet.

After 35 seconds we execute the same command again to see that liveness probes have failed, and the containers have been killed and recreated.

We wait another 30 seconds and verify that the container has been restarted because the RESTARTS has been incremented. We do this with executing the command **kubectl get pod liveness-exec.**

The second manifest that we create is **probes\_http.yaml** with the following definition:

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

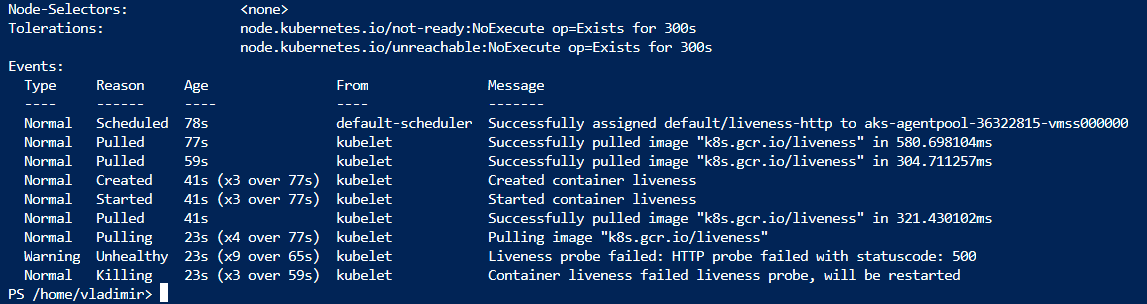
value: Awesome

initialDelaySeconds: 3

periodSeconds: 3

We create the pod with running **kubectl create -f probes\_http.yaml** and immediately run **kubectl describe pod liveness-http** in the first 10 seconds to see that the container is alive and the /healthz handler returns status of 200, and after that it returns status of 500.

We can see the same output as the previous pod. Kubelet will reboot the container.

****

We continue with TCP probes. We create the manifest probes\_tcp.yaml with the following definition:

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

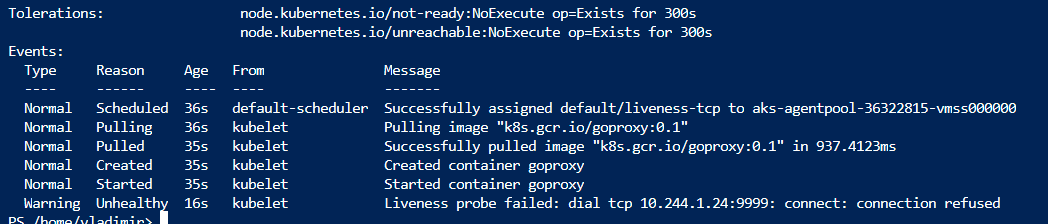
httpHeaders:

initialDelaySeconds: 15

periodSeconds: 20

Then we do the same steps as the previous pod, create the pod with **kubectl create -f probes\_tcp** and then **kubectl describe pod liveness-tcp** in the first 10 seconds to see that it is running.

After 30 seconds we can do **kubectl describe pod liveness-tcp** to verify the same as in the previous pods.



Our last manifest will be **readiness\_http.yaml** created with the following definition:

apiVersion: v1

kind: Pod

metadata:

labels:

app: test

name: readiness-http

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: /

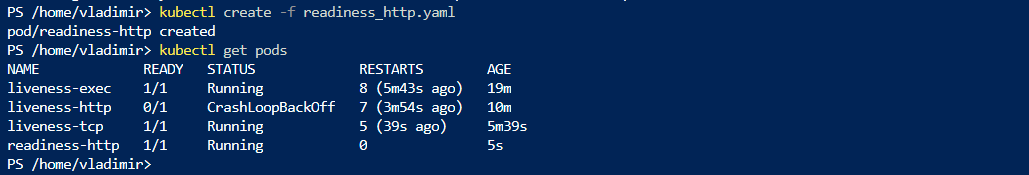
httpHeaders:

- name: Host

value: myapplication1.com

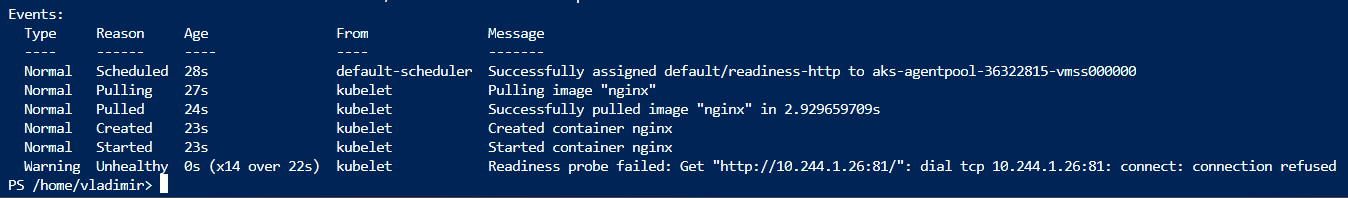
port: 80

We create the pod with the command **kubectl create -f readiness\_http.yaml.** Then we get the status of our pod with **kubectl get pods -A** and verify that it is in the running state.



We examine the events of the pod with **kubectl describe pod readiness-http** and then delete the pod and edit the **readiness-http.yaml** file. We enter the value 81 for the port parameter.

We create the pod again **kubectl create -f readiness\_http.yaml** and check the status **kubectl get pods -A.** Now the pod is running but it is not in ready state. To see why we execute the command **kubectl describe pod readiness-http** and can see in the events that it failed due to refused connection and the pod will not receive any traffic.



We delete all the pods under the default namespace **kubectl delete --all pods.**