**Practice1: Simple pods operations**

1. Login to Azure and connect to your AKS cluster.

Graphical user interface, text, application

Description automatically generated

**I created kubernetes cluster on Azure and connected whit that cluster.**

1. Check how many pods run under the default namespace. Run **kubectl get pods**.

Text

Description automatically generated

3. You should not see any pod under the default namespace. Now check all namespaces. Run **kubectl get pods**

**–all-namespace**.

Graphical user interface, text

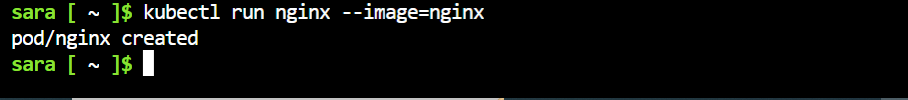
Description automatically generated

**I used the command *kubectl get pods --all-namespaces***

4. How many pods do you see? Who deployed these pods? Why are they deployed?

**They are deployed for the use of Kubernetes in Azure and they are deployed by default when creating a cluster in Azure.**

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?

Text

Description automatically generated

7. Check the logs coming out of your pod. Run **kubectl logs nginx**.

Graphical user interface, text

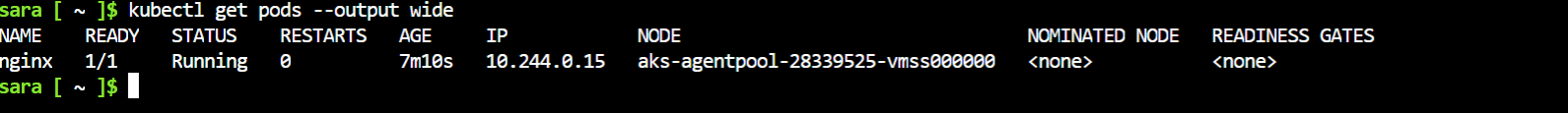
Description automatically generated

8. Run following command to check current resource consumption of your pod: **kubectl top pod nginx**.

Text

Description automatically generated

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.**

**Text

Description automatically generated**

**Text

Description automatically generated**

**The circled ones are the information from the previous step. The difference between these commands is that the first one gives us information about all pods and the second one gives us more detailed information about the selected pod.**

11. Delete your pod using **kubectl delete pod nginx.**

**A screenshot of a computer

Description automatically generated with medium confidence**

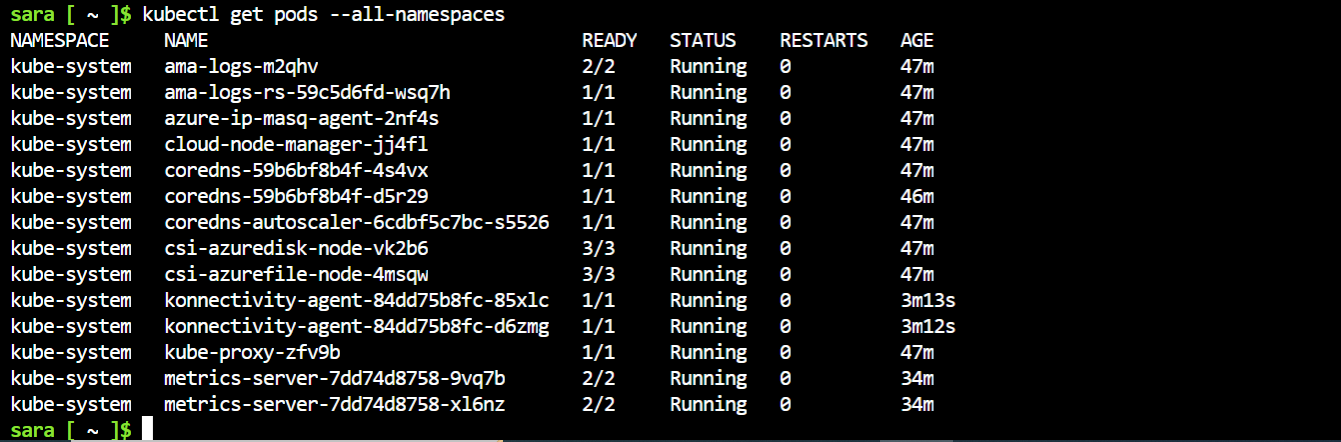
12. Let’s find the image used on one of the coredns pods under the kube-system namespace.

Text

Description automatically generated

**I used this command to show all images in use.**

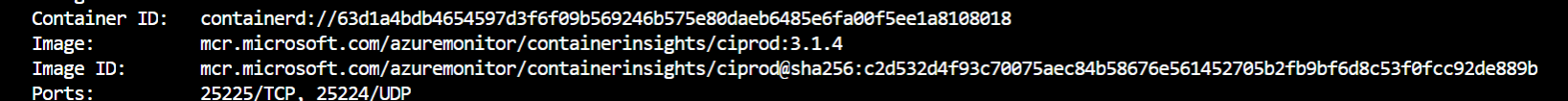
13. Once again list all pods under all namespaces.

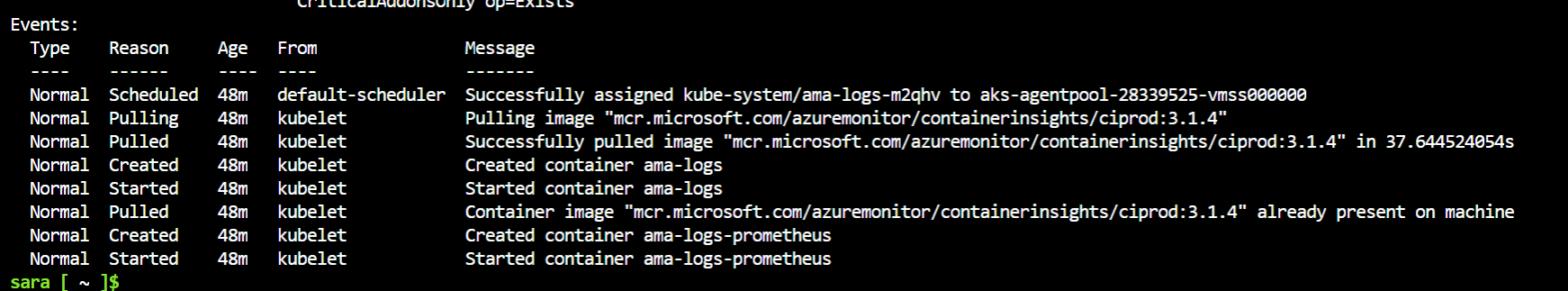


14. Note one of the coredns pods. Now run **kubectl describe pod <coredns-name> -n kube-system**. Replace the

<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.

Graphical user interface, application, Excel

Description automatically generated

**We can check the log from the cluster that we created in azure.**

**Practice2: Working with pod manifest files**

**NOTE: Everything is in adbox because I’m running in admin mode, and can’t change any permissions here because is a work laptop. Even the docker desktop is running as admin.**

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**.

**Text

Description automatically generated**

**Text

Description automatically generated**

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.

Text

Description automatically generated

**These are all mistakes in the yml file, so I googled examples of correct yml files and I changed the content of the file to:**

apiVersion: v1

kind: Pod

metadata:

  name: static-web

  labels:

spec:

  containers:

  - name: redis

    image: redis:5.0.4

4. When you solve all the syntax errors your pod should be deployed but is it running? What is the status of

your pod?

**Screenshot from before Status: RUNNING (but this is because I changed the image redis123 to redis5:0:4)**

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? **As I explained in the previous question. If I hadn’t changed the image it wouldn’t say running. If the status was not running I would have to delete the pod redeployed with the correct image.**

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**.

Graphical user interface, text

Description automatically generated

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.

Graphical user interface, application

Description automatically generated

apiVersion: v1

kind: Pod

metadata:

  name: nginx

  labels:

    app: frontend

spec:

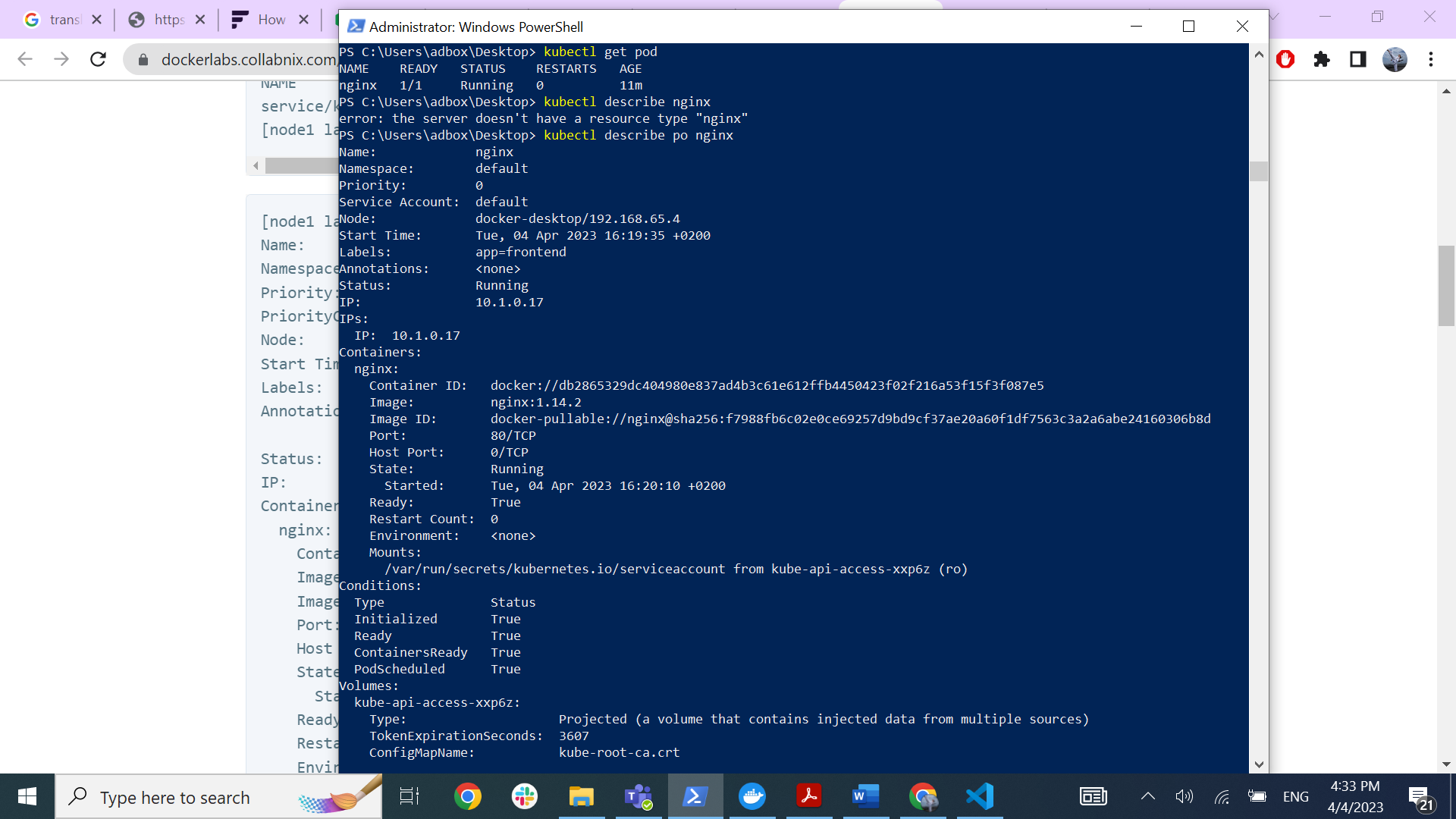
  containers:

  - name: nginx

    image: nginx:1.14.2

    ports:

    - containerPort: 80



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

apiVersion: v1

kind: Pod

metadata:

  name: memcached-web

  labels:

    app: web

spec:

  containers:

  - name: memcached

    image: memcached

    ports:

    - containerPort: 11211

    resources:

      limits:

        memory: "0.25Gi"

        cpu: "0.5"

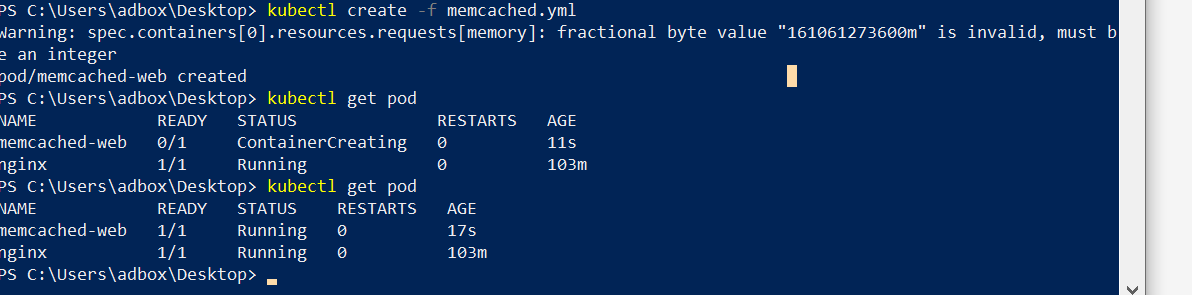
      requests:

        cpu: "0.35"

        memory: "0.15Gi"

    imagePullPolicy: IfNotPresent

  restartPolicy: Never



12. Don’t forget to try your pod definition.

**Practice3: Multi-container pods**

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.

apiVersion: v1

kind: Pod

metadata:

  name: multi-container

  labels:

    app: web

spec:

  containers:

  - name: redis

    image: redis:latest

    ports:

    - containerPort: 6379

  - name: nginx

    image: nginx:latest

    ports:

    - containerPort: 80

Graphical user interface, text

Description automatically generated

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? **2 of 2 containers running**

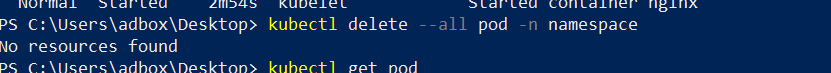
when you kubectl get the pods? Why your pod displays different values for ready?

4. Kubectl describe you new pod, and locate the containers section. How many containers are listed?

Graphical user interface, text

Description automatically generated

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**

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.

Text

Description automatically generated

5. After 35 seconds, view the Pod events again. Run **kubectl describe pod liveness-exec.**

Text

Description automatically generated

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.

Text

Description automatically generated with low confidence

7. Wait another 30 seconds, and verify that the container has been restarted. Run **kubectl get pod livenessexec.**

8. The output should show that RESTARTS has been incremented.

Shape

Description automatically generated

9. We will continue with HTTP probe. Create file named probes\_http.yaml with following content:

**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:

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"))

}

})

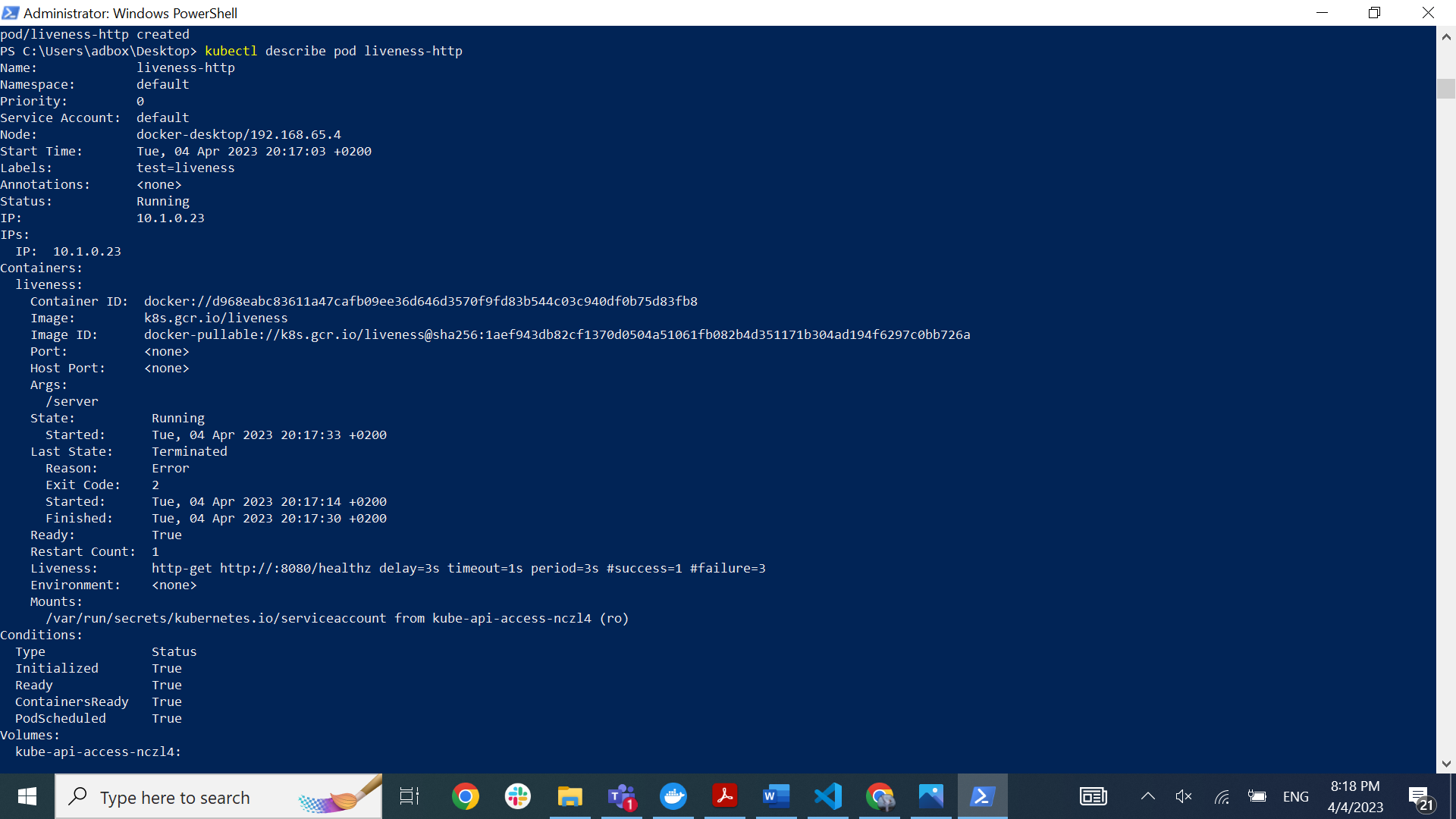
11. For the first 10 seconds that the container is alive, the /healthz handler returns a status of 200. After that,

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.**

**Text

Description automatically generated**

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.

Text

Description automatically generated

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.**

Text

Description automatically generated

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.

Text

Description automatically generated

32. Describe the pod. Run **kubectl describe pod readiness-http.**

Text

Description automatically generated

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.