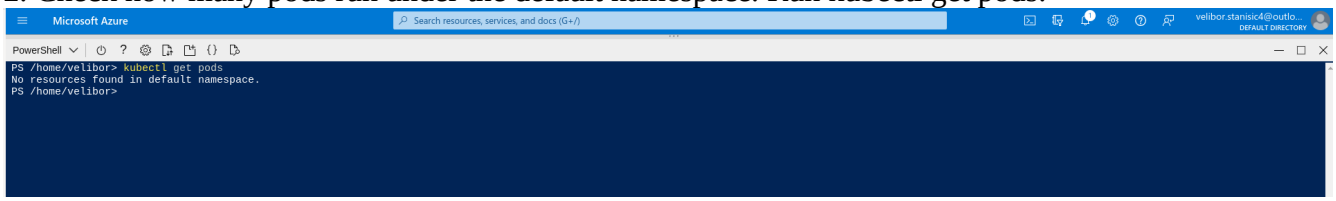
Exercise: Pods – Velibor Stanisic

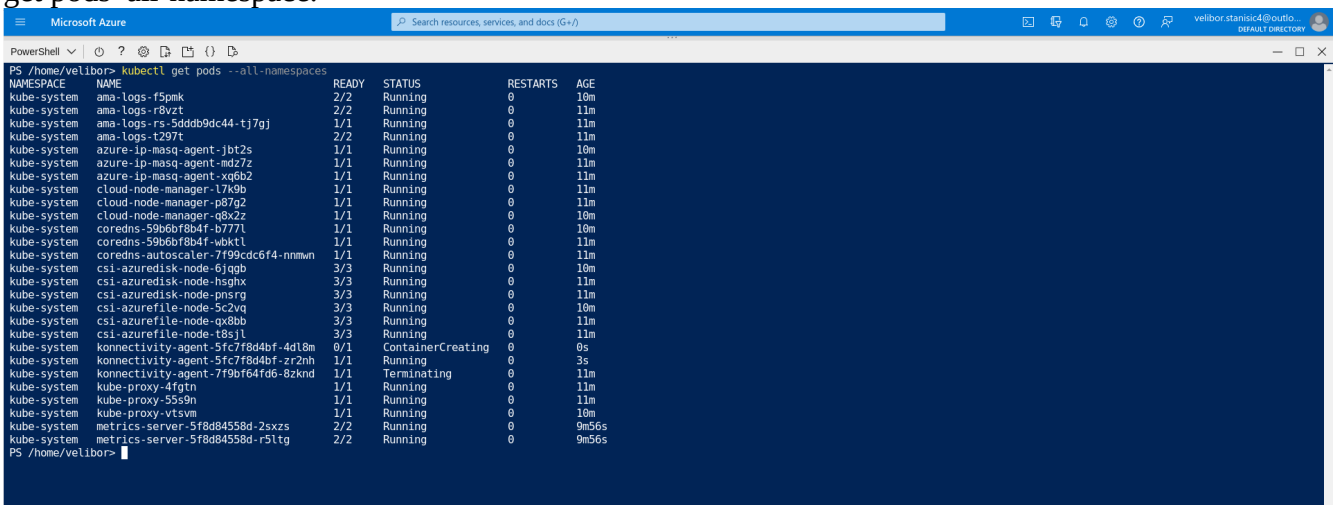
Pods are the smallest, most basic deployable objects in Kubernetes. A Pod represents a single instance of a running process in your cluster. Pods contain one or more containers, such as Docker containers. Although you want deploy pods directly (static pods), knowledge for defining pods manifest files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

Practice1: Simple pods operations

1. Login to Azure and connect to your AKS cluster.
2. Check how many pods run under the default namespace. Run kubectl get pods.

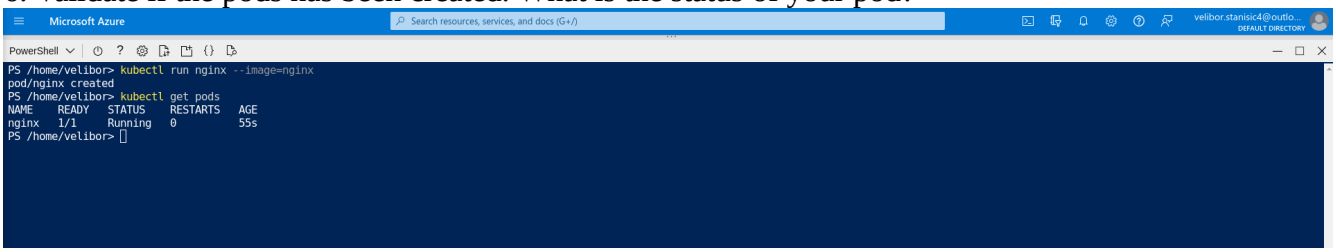


3. You should not see any pod under the default namespace. Now check all namespaces. Run kubectl get pods–all-namespace.



4. How many pods do you see? Who deployed these pods? Why are they deployed?
   The reason for deployment depends on the application or service that is running in the pods.

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.

```
PS /home/velibor> kubectl logs nginx
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
2023/04/03 20:22:04 [notice] 1#1: using the "epoll" event method
2023/04/03 20:22:04 [notice] 1#1: nginx/1.23.4
2023/04/03 20:22:04 [notice] 1#1: built by gcc 10.2.1 20210110 (Debian 10.2.1-6)
2023/04/03 20:22:04 [notice] 1#1: OS: Linux 5.4.0-1104-azure
2023/04/03 20:22:04 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2023/04/03 20:22:04 [notice] 1#1: start worker processes
2023/04/03 20:22:04 [notice] 1#1: start worker process 29
2023/04/03 20:22:04 [notice] 1#1: start worker process 30
PS /home/velibor>
```

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.

```
PS /home/velibor> kubectl top pod nginx
NAME    CPU(cores)   MEMORY(bytes)
nginx   0m           3Mi
PS /home/velibor> kubectl get pods -o wide
NAME    READY   STATUS    RESTARTS   AGE    IP          NODE                              NOMINATED NODE   READINESS GATES
nginx   1/1     Running   0          5m9s   10.244.1.6  aks-agentpool-18326435-vmss000000  <none>           <none>
PS /home/velibor>
```

10. Try to find the same information but this time running kubectl describe pod nginx.

```
PS /home/velibor> kubectl describe pod nginx
Name:             nginx
Namespace:        default
Priority:         0
Service Account:  default
Node:             aks-agentpool-18326435-vmss000000/10.224.0.5
Start Time:       Mon, 03 Apr 2023 20:21:59 +0000
Labels:           run=nginx
Annotations:      <none>
Status:           Running
IP:               10.244.1.6
IPs:
  IP:  10.244.1.6
Containers:
  nginx:
    Container ID:   containerd://371ae65f5b853ffead6a83eb3eaeb855369f8a66084f89847d7433b55d211004
    Image:          nginx
    Image ID:       docker.io/library/nginx@sha256:2ab30d6ac53580a6db8b657abf0f68d75360ff5cc1670a85acb5bd85ba1b19c0
    Port:           <none>
    Host Port:      <none>
    State:          Running
      Started:      Mon, 03 Apr 2023 20:22:04 +0000
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-dmpgc (ro)
Conditions:
  Type              Status
  Initialized       True
  Ready             True
  ContainersReady   True
  PodScheduled      True
Volumes:
  kube-api-access-dmpgc:
    Type:                    Projected (a volume that contains injected data from multiple sources)
    TokenExpirationSeconds:  3607
    ConfigMapName:           kube-root-ca.crt
    ConfigMapOptional:       <nil>
    DownwardAPI:             true
QoS Class:                   BestEffort
Node-Selectors:              <none>
Tolerations:                 node.kubernetes.io/not-ready:NoExecute op=Exists for 300s
                             node.kubernetes.io/unreachable:NoExecute op=Exists for 300s

Events:
  Type    Reason     Age    From               Message
  ----    ------     ----   ----               -------
  Normal  Scheduled  6m48s  default-scheduler  Successfully assigned default/nginx to aks-agentpool-18326435-vmss000000
```

11. Delete your pod using kubectl delete pod nginx.

```
PS /home/velibor> kubectl delete pod nginx
pod "nginx" deleted
PS /home/velibor>
```

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.

```
PS /home/velibor> kubectl get pods --all-namespaces
NAMESPACE     NAME                                  READY   STATUS    RESTARTS   AGE
kube-system   ama-logs-rs-5dddb9dc44-tcxnc          1/1     Running   0          116s
kube-system   ama-logs-t297t                        2/2     Running   0          26m
kube-system   ama-logs-w98tv                        2/2     Running   0          108s
kube-system   azure-ip-masq-agent-8gw4w             1/1     Running   0          114s
kube-system   azure-ip-masq-agent-mdz7z             1/1     Running   0          26m
kube-system   cloud-node-manager-6tcxq              1/1     Running   0          106s
kube-system   cloud-node-manager-l7k9b              1/1     Running   0          26m
kube-system   coredns-59b6bf8b4f-j4pb9              1/1     Running   0          117s
kube-system   coredns-59b6bf8b4f-lll86              1/1     Running   0          96s
kube-system   coredns-autoscaler-7f99cdc6f4-jg6zj   1/1     Running   0          117s
kube-system   csi-azuredisk-node-hsghx              3/3     Running   0          26m
kube-system   csi-azuredisk-node-l8cgv              3/3     Running   0          105s
kube-system   csi-azurefile-node-t8sjl              3/3     Running   0          26m
kube-system   csi-azurefile-node-zxd4c              3/3     Running   0          112s
kube-system   konnectivity-agent-5fc7f8d4bf-4phql   1/1     Running   0          117s
kube-system   konnectivity-agent-5fc7f8d4bf-zr2nh   1/1     Running   0          15m
kube-system   kube-proxy-55s9n                      1/1     Running   0          26m
kube-system   kube-proxy-z7jwq                      1/1     Running   0          113s
kube-system   metrics-server-8655f897d8-jqwp7       2/2     Running   0          117s
kube-system   metrics-server-8655f897d8-kf9cc       2/2     Running   0          13m
PS /home/velibor>
```

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.

```
PS /home/velibor> kubectl describe pod coredns-59b6bf8b4f-j4pb9 -n kube-system
Name:                 coredns-59b6bf8b4f-j4pb9
Namespace:            kube-system
Priority:             2000001000
Priority Class Name:  system-node-critical
Service Account:      coredns
Node:                 aks-agentpool-18326435-vmss000000/10.224.0.5
Start Time:           Mon, 03 Apr 2023 20:32:33 +0000
Labels:               k8s-app=kube-dns
                      kubernetes.io/cluster-service=true
                      pod-template-hash=59b6bf8b4f
                      version=v20
Annotations:          prometheus.io/port: 9153
Status:               Running
IP:                   10.244.1.9
IPs:
  IP:           10.244.1.9
Controlled By:  ReplicaSet/coredns-59b6bf8b4f
Containers:
  coredns:
    Container ID:  containerd://06df0d964ee3bbafb05f2746b93eaf3fd77b72154123a728191309156999b877
    Image:         mcr.microsoft.com/oss/kubernetes/coredns:v1.9.3
    Image ID:      sha256:c38f956b642366c8eeb0babfda6b0bb2aa92f27a968589804cadb445f6df72d6
    Ports:         53/UDP, 53/TCP, 9153/TCP
    Host Ports:    0/UDP, 0/TCP, 0/TCP
    Args:
      -conf
      /etc/coredns/Corefile
    State:          Running
      Started:      Mon, 03 Apr 2023 20:32:35 +0000
    Ready:          True
    Restart Count:  0
    Limits:
      cpu:     3
      memory:  500Mi
    Requests:
      cpu:     100m
      memory:  70Mi
    Liveness:   http-get http://:8080/health delay=60s timeout=5s period=10s #success=1 #failure=5
    Readiness:  http-get http://:8181/ready delay=0s timeout=1s period=10s #success=1 #failure=3
    Environment:
      KUBERNETES_PORT_443_TCP_ADDR:  mordor-dns-55msjzsy.hcp.westeurope.azmk8s.io
      KUBERNETES_PORT:               tcp://mordor-dns-55msjzsy.hcp.westeurope.azmk8s.io:443
      KUBERNETES_PORT_443_TCP:       tcp://mordor-dns-55msjzsy.hcp.westeurope.azmk8s.io:443
      KUBERNETES_SERVICE_HOST:       mordor-dns-55msjzsy.hcp.westeurope.azmk8s.io
    Mounts:
      /etc/coredns from config-volume (ro)
      /etc/coredns/custom from custom-config-volume (ro)
```

15. Inspect the output and locate the image information.

```
Containers:
  coredns:
    Container ID:  containerd://06df0d964ee3bbafb05f2746b93eaf3fd77b72154123a728191309156999b877
    Image:         mcr.microsoft.com/oss/kubernetes/coredns:v1.9.3
    Image ID:      sha256:c38f956b642366c8eeb0babfda6b0bb2aa92f27a968589804cadb445f6df72d6
    Ports:         53/UDP, 53/TCP, 9153/TCP
```

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.

```
PS /home/velibor> kubectl logs metrics-server-7dd74d8758-8h92k -n kube-system
Defaulted container "metrics-server-vpa" out of: metrics-server-vpa, metrics-server
ERROR: logging before flag.Parse: I0403 20:35:36.971033       1 pod_nanny.go:68] Invoked by [/pod_nanny --config-dir=/etc/config --cpu=44m --extra-cpu=0.5m --memory=51Mi --extra-memory=4Mi --poll-period=180000 -
-threshold=5 --deployment=metrics-server --container=metrics-server]
ERROR: logging before flag.Parse: I0403 20:35:36.971084       1 pod_nanny.go:69] Version: 1.8.14
ERROR: logging before flag.Parse: I0403 20:35:36.971576       1 pod_nanny.go:85] Watching namespace: kube-system, pod: metrics-server-7dd74d8758-8h92k, container: metrics-server.
ERROR: logging before flag.Parse: I0403 20:35:36.971592       1 pod_nanny.go:86] storage: MISSING, extra_storage: 0Gi
ERROR: logging before flag.Parse: I0403 20:35:36.973304       1 pod_nanny.go:189] Failed to read data from config file "/etc/config/NannyConfiguration": open /etc/config/NannyConfiguration: no such file or direc
tory, using default parameters
ERROR: logging before flag.Parse: I0403 20:35:36.973324       1 pod_nanny.go:116] cpu: 44m, extra_cpu: 0.5m, memory: 51Mi, extra_memory: 4Mi
ERROR: logging before flag.Parse: I0403 20:35:36.973337       1 pod_nanny.go:145] Resources: [{Base:{i:{value:44 scale:-3} d:{Dec:<nil>} s:44m Format:DecimalSI} ExtraPerNode:{i:{value:5 scale:-4} d:{Dec:<nil>} s
: Format:DecimalSI} Name:cpu} {Base:{i:{value:53477376 scale:0} d:{Dec:<nil>} s:51Mi Format:BinarySI} ExtraPerNode:{i:{value:4194304 scale:0} d:{Dec:<nil>} s:4Mi Format:BinarySI} Name:memory}]
PS /home/velibor>
```

Practice2: Working with pod manifest files

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
```

```
apiVersion: v11
kind: pod
metadata:
name: static-web
labels:
role: myrole
specs:
containers:
- name: redis
image: redis123
```

```
U:---  redis.yaml     All L10    (Fundamental +4)
Wrote /home/dttw/Practice2/redis.yaml
```

2. Try to deploy the pod defined in redis.yaml. Run kubectl create –f redis.yaml.

```
~/Practice2 $
~/Practice2 $ kubectl create -f redis.yaml
error: resource mapping not found for name: "" namespace: "" from "redis.yaml": no matches for kind "pod" in version "v
1"
ensure CRDs are installed first
~/Practice2 $ kubectl create -f redis.yaml
error: resource mapping not found for name: "" namespace: "" from "redis.yaml": no matches for kind "pod" in version "v
1"
ensure CRDs are installed first
~/Practice2 $
U:---  *eshell*      Bot L69    (Eshell +3)
```

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.

```
emacs@dttw
File  Edit  Options  Buffers  Tools  Help
apiVersion: v1
kind: Pod
metadata:
  name: static-web
  labels:
    role: myrole
spec:
  containers:
    - name: redis
      image: redis123
-:---  redis.yaml     All L10    (Fundamental +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?

```
~/Practice2 $
~/Practice2 $ kubectl get pods
NAME         READY   STATUS             RESTARTS   AGE
static-web   0/1     ImagePullBackOff   0          7m11s
~/Practice2 $
```

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?

```
emacs@dttw
File  Edit  Options  Buffers  Tools  Help
~/Practice2 $
~/Practice2 $ kubectl describe pod static-web
Name:             static-web
Namespace:        default
Priority:         0
Service Account:  default
Node:             minikube/192.168.39.150
Start Time:       Tue, 04 Apr 2023 01:09:02 +0200
Labels:           role=myrole
Annotations:      <none>
Status:           Pending
IP:               10.244.0.3
IPs:
  IP:  10.244.0.3
Containers:
  redis:
    Container ID:
    Image:          redis123
    Image ID:
    Port:           <none>
    Host Port:      <none>
    State:          Waiting
      Reason:       ImagePullBackOff
    Ready:          False
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-kbrcv (ro)
Conditions:
  Type              Status
  Initialized       True
  Ready             False
  ContainersReady   False
U:---  *eshell*      15% L129   (Eshell +3)
```

6. Find the correct image (check the Docker hub page) and correct it in the manifest.

```
emacs@dttw
File  Edit  Options  Buffers  Tools  Help
apiVersion: v1
kind: Pod
metadata:
  name: static-web
  labels:
    role: myrole
spec:
  containers:
    - name: redis
      image: redis:6.2
-:---  redis.yaml     All L10    (Fundamental +4)
Wrote /home/dttw/Practice2/redis.yaml
```

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).

```
~/Practice2 $
~/Practice2 $ kubectl delete pod static-web
pod "static-web" deleted
~/Practice2 $ kubectl create -f redis.yaml
pod/static-web created
~/Practice2 $
```

8. Check the status of your pod. It should be running now.

```
~/Practice2 $
~/Practice2 $ kubectl get pods
NAME         READY   STATUS    RESTARTS   AGE
static-web   1/1     Running   0          2m5s
~/Practice2 $
```

9. Now you can delete the pod. Try to delete it using the kubectl delete –f redis.yaml.

```
~/Practice2 $
~/Practice2 $ kubectl delete -f redis.yaml
pod "static-web" deleted
~/Practice2 $
```

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.

```
emacs@dttw
File  Edit  Options  Buffers  Tools  Help
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod
  labels:
    app: frontend
spec:
  containers:
  - name: nginx
    image: nginx
    ports:
    - containerPort: 80
```

```
~/Practice2 $ kubectl create -f nginx.yaml
pod/nginx-pod created
```

```
~/Practice2 $ kubectl get pods
NAME        READY   STATUS    RESTARTS   AGE
nginx-pod   1/1     Running   0          40s
```

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

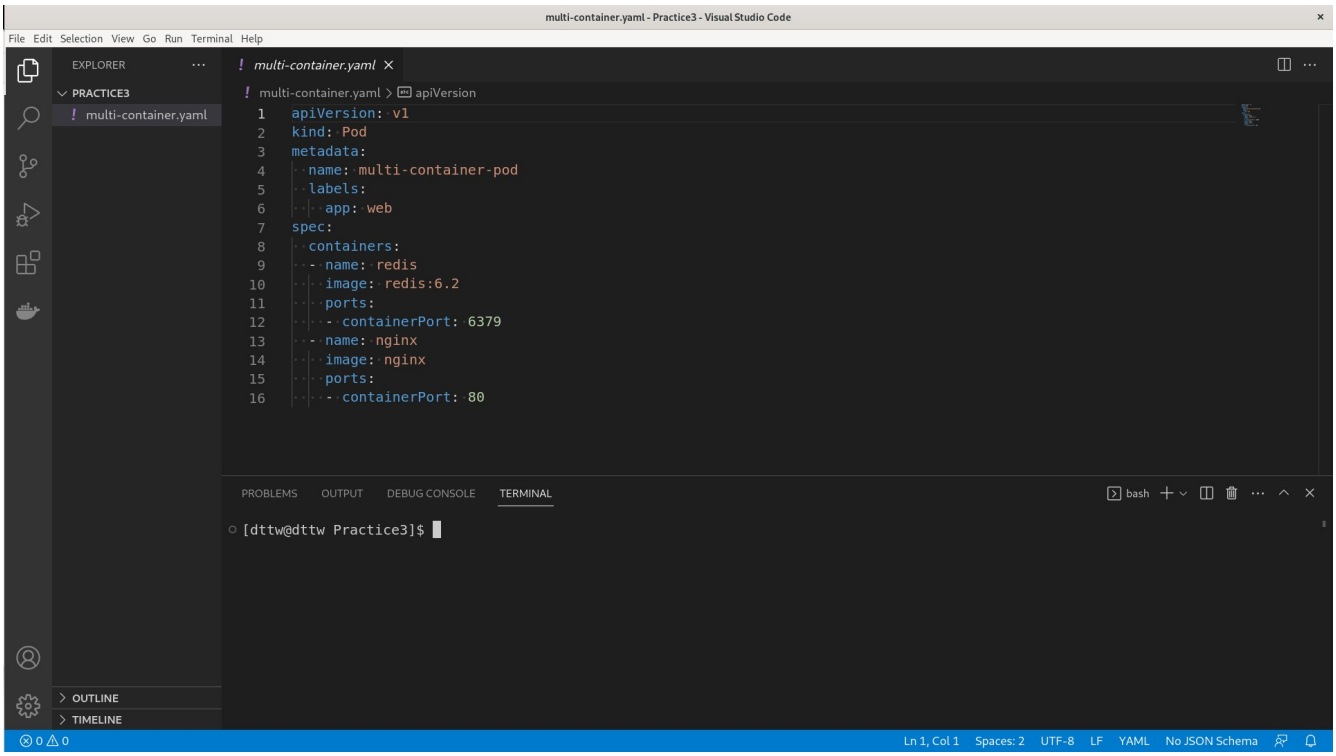
```
emacs@dttw
File  Edit  Options  Buffers  Tools  Help
apiVersion: v1
kind: Pod
metadata:
  name: memcached-pod
  labels:
    app: web
spec:
  containers:
  - name: memcached
    image: memcached
    ports:
    - containerPort: 11211
    resources:
      requests:
        cpu: 0.35
        memory: 150Mi
      limits:
        cpu: 0.5
        memory: 250Mi
    restartPolicy: Never
```

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

```
~/Practice2 $ kubectl create -f memcached.yaml
pod/memcached-pod created
~/Practice2 $ kubectl get pods
NAME            READY   STATUS    RESTARTS   AGE
memcached-pod   1/1     Running   0          8s
nginx-pod       1/1     Running   0          11m
~/Practice2 $
```

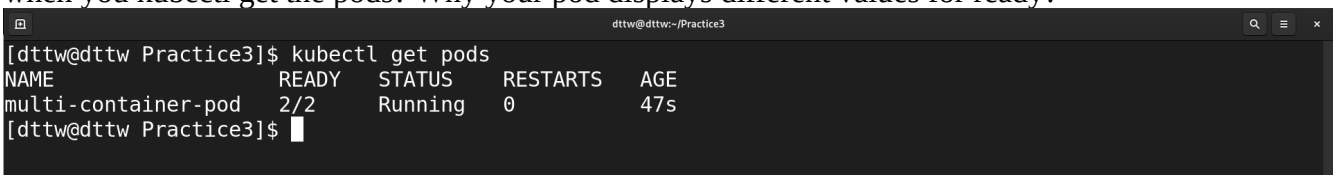
Practice3: Multi-container pods

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.



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?

```
Containers:
  redis:
    Container ID:   docker://d57f11f627d31e3805dc35b1222bc01c6333ae912569f18e803b29666b796506
    Image:          redis:6.2
    Image ID:       docker-pullable://redis@sha256:09fa1f52ad0cc05590a837eb787799bd768074e6566f53c7a83992
43cad456b4
    Port:           6379/TCP
    Host Port:      0/TCP
    State:          Running
      Started:      Tue, 04 Apr 2023 02:10:41 +0200
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-jbvb9 (ro)
  nginx:
    Container ID:   docker://e48da74a401b74257f3bb52771a0d954a1055f8f26b03021e9d55ab33184b525
    Image:          nginx
    Image ID:       docker-pullable://nginx@sha256:2ab30d6ac53580a6db8b657abf0f68d75360ff5cc1670a85acb5bd
85ba1b19c0
```

5. Delete all the pods under the default namespace.

```
[dttw@dttw Practice3]$ kubectl delete pods --all
pod "multi-container-pod" deleted
[dttw@dttw Practice3]$ kubectl get pods
No resources found in default namespace.
[dttw@dttw Practice3]$
```

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:

```yaml
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.

```
[dttw@dttw Practice4]$ kubectl create -f probes_exec.yaml
pod/liveness-exec created
[dttw@dttw Practice4]$
```

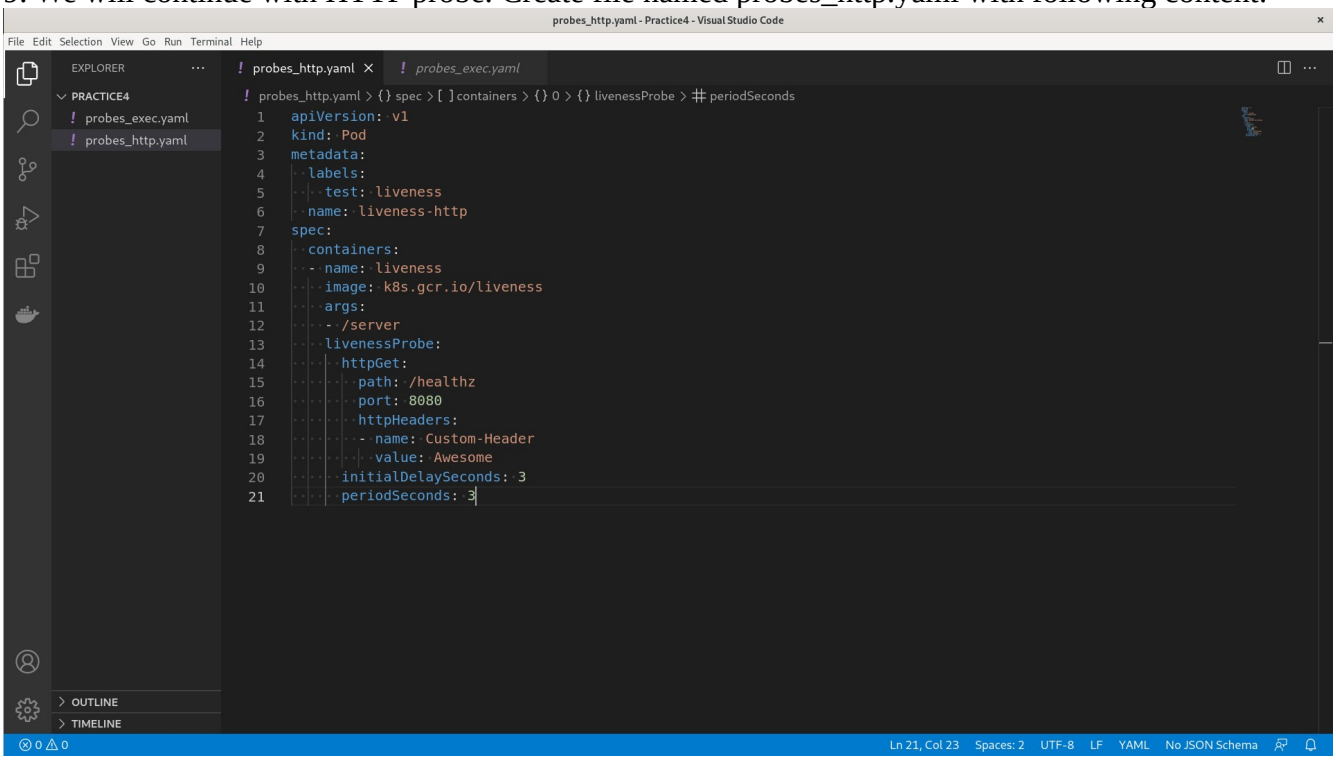
4. Run kubectl describe pod liveness-exec immediately after you deploy the pod. The output should indicate that no liveness probes have failed yet.
5. After 35 seconds, view the Pod events again. Run kubectl describe pod liveness-exec.
6. At the bottom of the output, there should be a messages indicating that the liveness probes have failed, and the containers have been killed and recreated.

```
ding waiting)
  Warning  Unhealthy  2m5s (x9 over 4m45s)   kubelet            Liveness probe failed: cat: can't open '/tmp/healthy': No such file or directory
  Normal   Killing    2m5s (x3 over 4m35s)   kubelet            Container liveness failed liveness probe, will be restarted
  Normal   Pulling    95s (x4 over 5m20s)    kubelet            Pulling image "k8s.gcr.io/busybox"
  Normal   Pulled     18s                    kubelet            Successfully pulled image "k8s.gcr.io/busybox" in 1.868340372s (1.868358012s inclu
ding waiting)
[dttw@dttw Practice4]$
```

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.

```
    Restart Count:  3
    Liveness:       exec [cat /tmp/healthy] delay=5s timeout=1s period=5s #success=1 #failure=3
```

9. We will continue with HTTP probe. Create file named probes_http.yaml with following content:



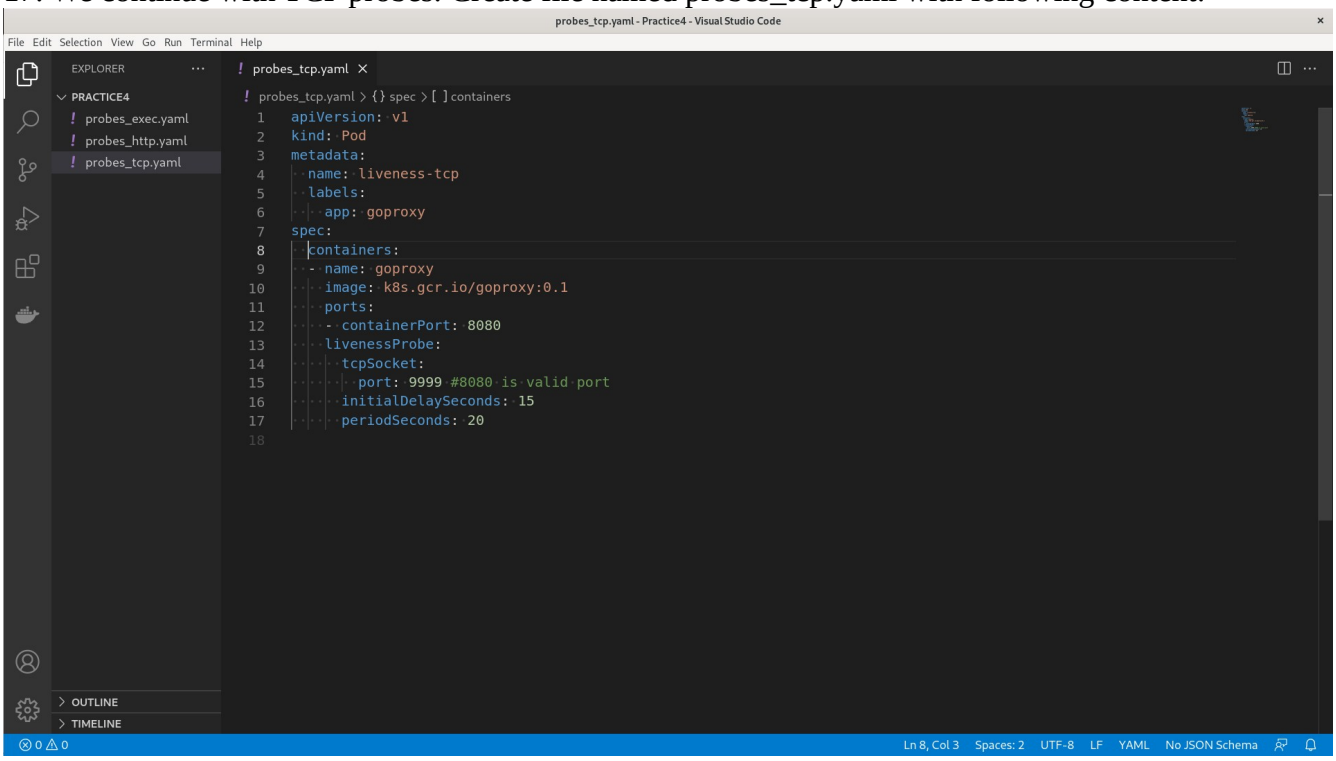
11. For the first 10 seconds that the container is alive, the /healthz handler returns a status of 200. After that, the handler returns a status of 500.

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.

```
  IP:  10.244.0.10
Containers:
  liveness:
    Container ID:  docker://90d9ad3d7fb8c4092359a39814fadd6316c71251ee2fb1fe3165a743a1752332
    Image:         k8s.gcr.io/liveness
    Image ID:      docker-pullable://k8s.gcr.io/liveness@sha256:1aef943db82cf1370d0504a51061fb082b4d351171b304ad194f6297c0bb726a
    Port:          <none>
    Host Port:     <none>
    Args:
      /server
    State:          Running
      Started:      Tue, 04 Apr 2023 02:45:06 +0200
    Ready:          True
    Restart Count:  0
    Liveness:       http-get http://:8080/healthz delay=3s timeout=1s period=3s #success=1 #failure=3
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-hzzk4 (ro)
Conditions:
  Type              Status
  Initialized       True
  Ready             True
  ContainersReady   True
```

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:



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.

```
dttw@dttw:~/Practice4
[dttw@dttw Practice4]$ kubectl create -f probes_tcp.yaml
pod/liveness-tcp created
[dttw@dttw Practice4]$ kubectl describe pod liveness-tcp
Name:             liveness-tcp
Namespace:        default
Priority:         0
Service Account:  default
Node:             minikube/192.168.39.150
Start Time:       Tue, 04 Apr 2023 02:58:44 +0200
Labels:           app=goproxy
Annotations:      <none>
Status:           Pending
IP:
IPs:              <none>
Containers:
  goproxy:
    Container ID:
    Image:          k8s.gcr.io/goproxy:0.1
    Image ID:
    Port:           8080/TCP
    Host Port:      0/TCP
    State:          Waiting
      Reason:       ContainerCreating
    Ready:          False
    Restart Count:  0
    Liveness:       tcp-socket :9999 delay=15s timeout=1s period=20s #success=1 #failure=3
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-jrrjk (ro)
Conditions:
  Type              Status
  Initialized       True
  Ready             False
  ContainersReady   False
  PodScheduled      True
Volumes:
  kube-api-access-jrrjk:
    Type:                    Projected (a volume that contains injected data from multiple sources)
    TokenExpirationSeconds:  3607
    ConfigMapName:           kube-root-ca.crt
    ConfigMapOptional:       <nil>
```

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:

readiness_http.yaml - Practice4 - Visual Studio Code

File Edit Selection View Go Run Terminal Help

EXPLORER

PRACTICE4

- probes_exec.yaml
- probes_http.yaml
- probes_tcp.yaml
- readiness_http.yaml

readiness_http.yaml > {} spec > [ ] containers > {} 0 > {} readinessProbe > {} httpGet

```yaml
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
```

OUTLINE

TIMELINE

0 0 | Ln 27, Col 1 | Spaces: 2 | UTF-8 | LF | YAML | No JSON Schema

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.

dttw@dttw:~/Practice4

```
[dttw@dttw Practice4]$ kubectl get pods -A
NAMESPACE     NAME                                 READY   STATUS             RESTARTS         AGE
default       liveness-exec                        0/1     CrashLoopBackOff   14 (2m37s ago)   43m
default       liveness-http                        0/1     CrashLoopBackOff   11 (2m35s ago)   21m
default       liveness-tcp                         1/1     Running            6 (106s ago)     7m46s
default       readiness-http                       1/1     Running            0                26s
kube-system   coredns-787d4945fb-w49x7             1/1     Running            0                124m
kube-system   etcd-minikube                        1/1     Running            0                124m
kube-system   kube-apiserver-minikube              1/1     Running            0                124m
kube-system   kube-controller-manager-minikube     1/1     Running            0                124m
kube-system   kube-proxy-pcn2p                     1/1     Running            0                124m
kube-system   kube-scheduler-minikube              1/1     Running            0                124m
kube-system   storage-provisioner                  1/1     Running            2 (123m ago)     124m
[dttw@dttw Practice4]$
```

28. Run kubectl describe pod readiness-http. Examine the events for this pod. Everything should be OK.

```
dttw@dttw:~/Practice4
kube-system   kube-scheduler-minikube            1/1     Running            0                 127m
kube-system   storage-provisioner                1/1     Running            2 (126m ago)      127m
[dttw@dttw Practice4]$ kubectl describe pod readiness-http
Name:             readiness-http
Namespace:        default
Priority:         0
Service Account:  default
Node:             minikube/192.168.39.150
Start Time:       Tue, 04 Apr 2023 03:06:04 +0200
Labels:           app=test
Annotations:      <none>
Status:           Running
IP:               10.244.0.12
IPs:
  IP:  10.244.0.12
Containers:
  nginx:
    Container ID:   docker://f9d435574ebc7c4bdd732616368c054954285dd764b040ac1749f98f2bb96af3
    Image:          nginx
    Image ID:       docker-pullable://nginx@sha256:2ab30d6ac53580a6db8b657abf0f68d75360ff5cc1670a85acb5bd85ba1b19c0
    Port:           80/TCP
    Host Port:      0/TCP
    State:          Running
      Started:      Tue, 04 Apr 2023 03:06:07 +0200
    Ready:          True
    Restart Count:  0
    Readiness:      http-get http://:80/ delay=1s timeout=1s period=2s #success=1 #failure=1
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-nrm6r (ro)
Conditions:
  Type              Status
  Initialized       True
  Ready             True
  ContainersReady   True
  PodScheduled      True
Volumes:
  kube-api-access-nrm6r:
    Type:                    Projected (a volume that contains injected data from multiple sources)
    TokenExpirationSeconds:  3607
    ConfigMapName:           kube-root-ca.crt
```

29. Now delete the pod and edit the readiness_http.yaml so that the port parameter has 81 value.

```
readiness_http.yaml - Practice4 - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
PRACTICE4
  probes_exec.yaml
  probes_http.yaml
  probes_tcp.yaml
  readiness_http.yaml
readiness_http.yaml > {} spec > [ ] containers > {} 0 > {} readinessProbe > {} httpGet > # port
1  apiVersion: v1
2  kind: Pod
3  metadata:
4    name: readiness-http
5    labels:
6      app: test
7  spec:
8    containers:
9    - name: nginx
10     image: nginx
11     ports:
12     - containerPort: 80
13     readinessProbe:
14       initialDelaySeconds: 1
15       periodSeconds: 2
16       timeoutSeconds: 1
17       successThreshold: 1
18       failureThreshold: 1
19       httpGet:
20         host:
21         scheme: HTTP
22         path: /
23         httpHeaders:
24         - name: Host
25           value: myapplication1.com
26         port: 81
27
OUTLINE
TIMELINE
0 0
Ln 26, Col 17  Spaces: 2  UTF-8  LF  YAML  No JSON Schema
```

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.

```
dttw@dttw:~/Practice4
[dttw@dttw Practice4]$ kubectl create -f readiness_http.yaml
pod/readiness-http created
[dttw@dttw Practice4]$ kubectl get pods -A
NAMESPACE     NAME                                READY   STATUS             RESTARTS         AGE
default       liveness-exec                       0/1     CrashLoopBackOff   16 (3m17s ago)   51m
default       liveness-http                       1/1     Running            14 (5m13s ago)   29m
default       liveness-tcp                        0/1     CrashLoopBackOff   7 (4m6s ago)     16m
default       readiness-http                      0/1     Running            0                9s
kube-system   coredns-787d4945fb-w49x7            1/1     Running            0                132m
kube-system   etcd-minikube                       1/1     Running            0                132m
kube-system   kube-apiserver-minikube             1/1     Running            0                133m
kube-system   kube-controller-manager-minikube    1/1     Running            0                133m
kube-system   kube-proxy-pcn2p                    1/1     Running            0                132m
kube-system   kube-scheduler-minikube             1/1     Running            0                133m
kube-system   storage-provisioner                 1/1     Running            2 (131m ago)     132m
[dttw@dttw Practice4]$
```

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

```
dttw@dttw:~/Practice4
[dttw@dttw Practice4]$ kubectl describe pod readiness-http
Name:             readiness-http
Namespace:        default
Priority:         0
Service Account:  default
Node:             minikube/192.168.39.150
Start Time:       Tue, 04 Apr 2023 03:14:41 +0200
Labels:           app=test
Annotations:      <none>
Status:           Running
IP:               10.244.0.13
IPs:
  IP:  10.244.0.13
Containers:
  nginx:
    Container ID:   docker://aba9bbe3fd40b0c86add1618f21543728b5d5307371710ecacc18136b203815f
    Image:          nginx
    Image ID:       docker-pullable://nginx@sha256:2ab30d6ac53580a6db8b657abf0f68d75360ff5cc1670a85acb5bd85ba1b19c0
    Port:           80/TCP
    Host Port:      0/TCP
    State:          Running
      Started:      Tue, 04 Apr 2023 03:14:46 +0200
    Ready:          False
    Restart Count:  0
    Readiness:      http-get http://:81/ delay=1s timeout=1s period=2s #success=1 #failure=1
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-kzszv (ro)
Conditions:
  Type              Status
  Initialized       True
  Ready             False
  ContainersReady   False
  PodScheduled      True
Volumes:
  kube-api-access-kzszv:
    Type:                    Projected (a volume that contains injected data from multiple sources)
    TokenExpirationSeconds:  3607
    ConfigMapName:           kube-root-ca.crt
    ConfigMapOptional:       <nil>
    DownwardAPI:             true
```

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.

```
dttw@dttw:~/Practice4
[dttw@dttw Practice4]$ kubectl delete pod --all --namespace=default
pod "liveness-exec" deleted
pod "liveness-http" deleted
pod "liveness-tcp" deleted
pod "readiness-http" deleted
[dttw@dttw Practice4]$
```

35. Don’t delete any manifest files created so far.