

Ques 5: Configure liveness and readiness probes for pods in AKS cluster.

SOLN :

### What are Probes in Kubernetes?

Probe Type	Purpose
<b>Liveness Probe</b>	Checks if the container is alive. If it fails, the container is restarted.
<b>Readiness Probe</b>	Checks if the container is <i>ready</i> to serve traffic. If it fails, it's removed from Service endpoints.

Used together, they help **AKS maintain high availability and reliability** of applications

### Step-by-Step: Configure Probes in Deployment

Below is an example YAML for a pod with both **liveness** and **readiness** probes configured:

we are creating a new deployment for testing the same :

Save the above YAML as `probes-deployment.yaml`

1. Run:

```
kubectl apply -f probes-deployment.yaml
```

Check status:

```
kubectl get pods
```

```
kubectl describe pod <your-pod-name>
```

```

Administrator: Windows PowerShell

PS C:\WINDOWS\system32> kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
myapp-deploy-7968ff6446-nnxcn      1/1     Running   0           80m
myapp-deploy-888884667-whm6s       1/1     Running   0           16s
myapp-deploy-888884667-wnl4c       0/1     Running   0           7s
PS C:\WINDOWS\system32> kubectl describe pod myapp-deploy-888884667-whm6s
Name:                               myapp-deploy-888884667-whm6s
Namespace:                          default
Priority:                             0
Service Account:                     default
Node:                                aks-userpool-92868701-vmss000001/10.224.0.4
Start Time:                          Tue, 01 Jul 2025 00:05:06 +0530
Labels:                              app=myapp
                                      pod-template-hash=888884667
Annotations:                          <none>
Status:                              Running
IP:                                  10.244.1.155
IPs:
  IP:                                10.244.1.155
Controlled By:                       ReplicaSet/myapp-deploy-888884667
Containers:
  myapp-container:
    Container ID:                     containerd://d7cf627b03801919112a73d5e0d996273e6c0141da5fc45a66738ec0b319a55f
    Image:                           prateek2004/my-frontent
    Image ID:                         docker.io/prateek2004/my-frontent@sha256:58bedad0762aca5ffa1d2e7f2f9d275b5677bca263ceb0dee
15888b7a
    Port:                            80/TCP
    Host Port:                        0/TCP
    State:                            Running
      Started:                        Tue, 01 Jul 2025 00:05:08 +0530
    Ready:                            True
    Restart Count:                     0
    Liveness:                         http-get http://:80/ delay=15s timeout=1s period=10s #success=1 #failure=3
    Readiness:                        http-get http://:80/ delay=5s timeout=1s period=5s #success=1 #failure=3
    Environment:                      <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-p6fkx (ro)
Conditions:
  Type                               Status
  PodReadyToStartContainers          True
  Initialized                         True
  Ready                              True
  ContainersReady                    True
  PodScheduled                       True
Volumes:
  kube-api-access-p6fkx:
    Type:                            Projected (a volume that contains injected data from multiple sources)
    TokenExpirationSeconds:           3607
    ConfigMapName:                    kube-root-ca.crt
    Optional:                         false
    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   84s   default-scheduler  Successfully assigned default/myapp-deploy-888884667-whm6s to ak
ool-92868701-vmss000001
  Normal  Pulling     83s   kubelet        Pulling image "prateek2004/my-frontent"

```

## Probe Status:

- Liveness: http-get http://:80/ → **Passing**
- Readiness: http-get http://:80/ → **Passing**
- Ready: True
- Restart Count: 0

This confirms:




- Pod is stable
- Health checks are configured correctly
- No crashes or restarts due to failed probes

I configured HTTP-based liveness and readiness probes for the frontend container in my AKS deployment. Initially, the probes were failing because they pointed to non-existent paths (`/healthz` and `/ready`). I identified that the application serves content on the root path (`/`), and updated the probe paths accordingly.

After applying the updated deployment configuration, both probes started succeeding. The pod transitioned to `Ready: True` and stayed stable with no restarts. This demonstrates successful health check configuration using Kubernetes probes to monitor application availability and startup readiness.

### Problem in Real Life:



Imagine you're running a real website (like Flipkart, Tabcura, etc.) inside Kubernetes. What if:

1.  The app crashes or gets stuck (e.g., due to a memory leak)?
2.  The app is **slow to start**, but Kubernetes thinks it's ready?
3.  A deployment rollout happens, but some pods aren't actually ready to serve traffic?


---

### ✅ Here's Where Probes Help

#### 1. Liveness Probe: Are you alive?

-  **Use Case:** If the app **gets stuck** or hangs due to bugs, **Kubernetes will kill and restart** the container automatically.
-  **Real-world example:** A microservice in Amazon app stuck in a loop — Kubernetes detects it's "unhealthy" and restarts it.

#### 2. Readiness Probe: Are you ready to serve traffic?

-  **Use Case:** Sometimes, the container is running but **not ready to take requests** (e.g., loading config, initializing DB).

- 🔄 **\*\*Kubernetes will wait before sending traffic** to it. If it fails later, traffic is routed away.
- 🕒 **Real-world example:** A payment service is still connecting to Razorpay APIs — Kubernetes won't send users' payments there until it's ready.

So we can say that **Real-World Use Case:**

In production environments, readiness and liveness probes help maintain app availability and reliability.

- If a pod is **not ready**, Kubernetes **avoids sending user traffic** to it — improving user experience.
- If the app is **stuck or unhealthy**, the liveness probe helps Kubernetes **self-heal** by restarting it.

This ensures zero downtime, smoother rollouts, and better handling of unexpected failures — which is **critical in real-time systems like e-commerce, healthcare, and financial apps**.

**Let's simulate a failed liveness probe — to understand how Kubernetes *self-heals* unhealthy pods.**

### **What We'll Do:**

We will intentionally fail the liveness probe by pointing it to a non-existing path (e.g., `/failcheck`). Kubernetes will keep checking this path, get a 404, and restart the pod after 3 failed attempts.

#### **1. Modify Your Deployment YAML**

Change the liveness probe like this:

`livenessProbe:`

`httpGet:`

`path: /failcheck # This path does NOT exist in your app`

`port: 80`

`initialDelaySeconds: 5`

```
periodSeconds: 5
```

failureThreshold: 3

```
kubectl apply -f liveness-fail.yaml
```

```
Administrator: Windows PowerShell
PS C:\WINDOWS\system32> notepad liveness-fail.yaml
PS C:\WINDOWS\system32> kubectl apply -f liveness-fail.yaml
>>
deployment.apps/myapp-deploy configured
PS C:\WINDOWS\system32>
```

```
PS C:\WINDOWS\system32> kubectl get pods
>>
NAME                                READY   STATUS    RESTARTS   AGE
myapp-deploy-75df44f776-mvdph      1/1     Running   5 (57s ago) 2m48s
myapp-deploy-75df44f776-rjwvl      0/1     Running   5 (49s ago) 2m39s
PS C:\WINDOWS\system32> kubectl get svc
```

```

C:\Windows\system32> kubectl describe pod myapp-deploy-75df44f776-mvdp
Name: myapp-deploy-75df44f776-mvdp
Namespace: default
Priority: 0
Service Account: default
Node: aks-userpool-92868701-vmss000001/10.224.0.4
Start Time: Tue, 01 Jul 2025 00:16:39 +0530
Labels: app=myapp
        pod-template-hash=75df44f776
Annotations: <none>
Status: Running
IP: 10.244.1.220
IPs:
  IP: 10.244.1.220
Controlled By: ReplicaSet/myapp-deploy-75df44f776
Containers:
  myapp-container:
    Container ID: containerd://92f0ff3b120ccd808deb7234e5c6c658612145093ae445fb677742b947957ea
    Image: prateek2004/my-frontent
    Image ID: docker.io/prateek2004/my-frontent@sha256:58bedad0762aca5ffa1d2e7f2f9d275b5677bca263ceb0deed5cca67
    Port: 80/TCP
    Host Port: 0/TCP
    State: Running
      Started: Tue, 01 Jul 2025 00:18:11 +0530
    Last State: Terminated
      Reason: Completed
      Exit Code: 0
      Started: Tue, 01 Jul 2025 00:17:46 +0530
      Finished: Tue, 01 Jul 2025 00:18:09 +0530
    Ready: True
    Restart Count: 4
    Liveness: http-get http://:80/failcheck delay=5s timeout=1s period=5s #success=1 #failure=3
    Readiness: http-get http://:80/ delay=5s timeout=1s period=5s #success=1 #failure=3
    Environment: <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-zkgff (ro)
Conditions:
  Type                               Status
  PodReadyToStartContainers         True
  Initialized                       True
  Ready                             True
  ContainersReady                   True
  PodScheduled                      True
Volumes:
  kube-api-access-zkgff:
    Type: Projected (a volume that contains injected data from multiple sources)
    TokenExpirationSeconds: 3607
    ConfigMapName: kube-root-ca.crt
    Optional: false
    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   101s   default-scheduler  Successfully assigned default/myapp-deploy-75df44f776-mvdp to aks-userpool-92868701-vmss000001
  Normal  Pulled      100s   kubelet         Successfully pulled image "prateek2004/my-frontent" in 1.482s (1.482s including waiting). Image size: 29619932 bytes.

```

### What's Happening:

- **myapp-deploy-75df44f776-mvdph**
    - **Status: Running but READY 0/1 → not passing probes.**
    - **Restart count is 1 → it already failed the liveness probe once and got restarted.**
  - **myapp-deploy-75df44f776-rjwv1**
    - **Status: Running and READY 1/1 → this pod is healthy and passing probes.**
- 

### Conclusion:

- **Your failing liveness probe simulation is working!**
- **Kubernetes detected `/failcheck` doesn't exist (likely returned HTTP 404), so it restarted the unhealthy pod — showcasing self-healing in AKS.**

**Hence As an additional effort, I configured a failing liveness probe (`/failcheck`) on one of the pods. Kubernetes detected the failure through the liveness probe and restarted the unhealthy container automatically. This demonstrates Kubernetes' built-in self-healing mechanism, which ensures the application remains highly available and resilient even if individual pods become unresponsive or unhealthy.**