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

SOLN :

What are Probes in Kubernetes?

Probe Type	Purpose
Liveness Probe	Checks if the container is alive. If it fails, the container is restarted.
Readiness Probe	Checks if the container is *ready* 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 :

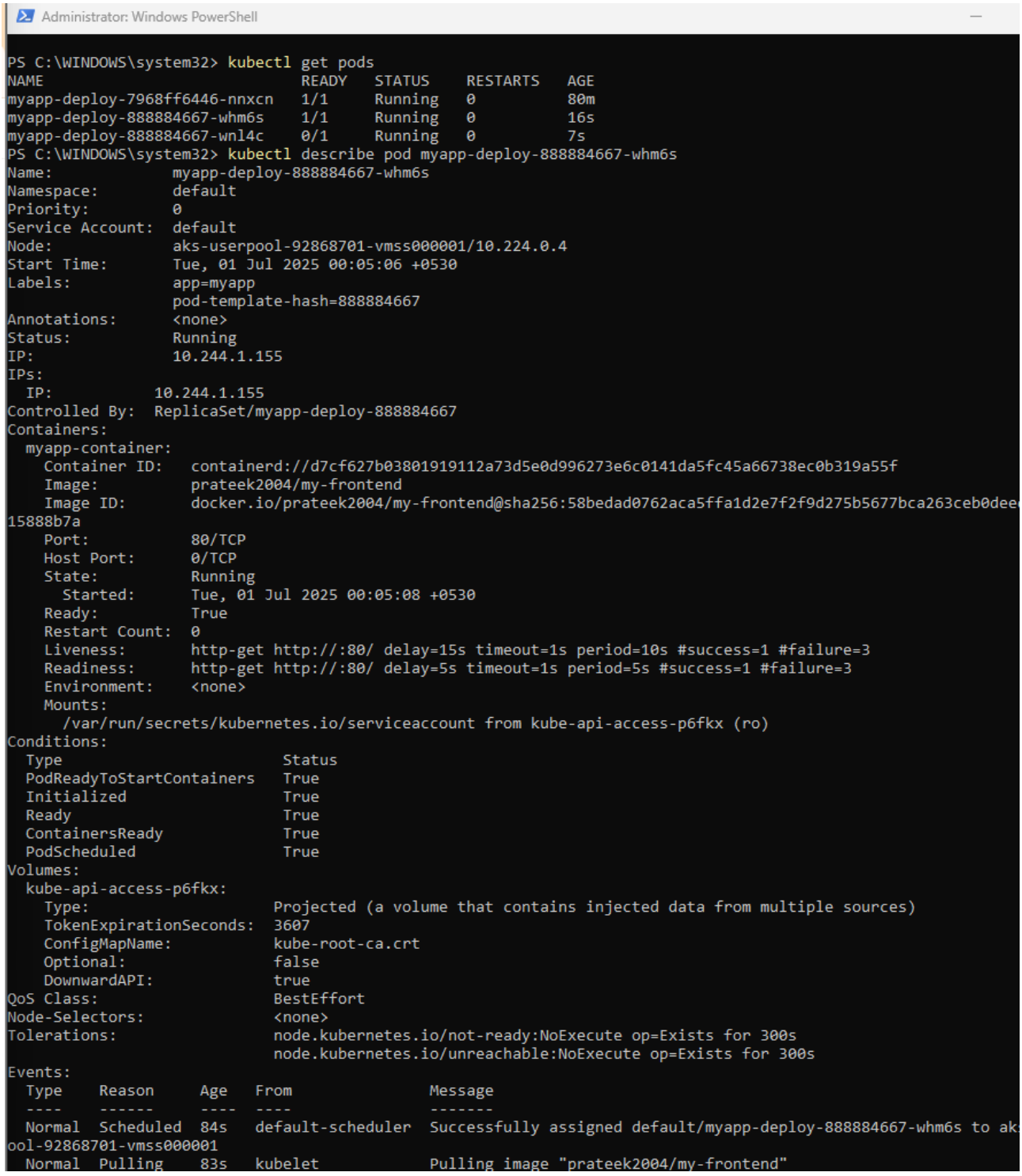
Save the above YAML as `probes-deployment.yaml`

1. Run:

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

Check status:

kubectl get pods
kubectl describe pod <your-pod-name>



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

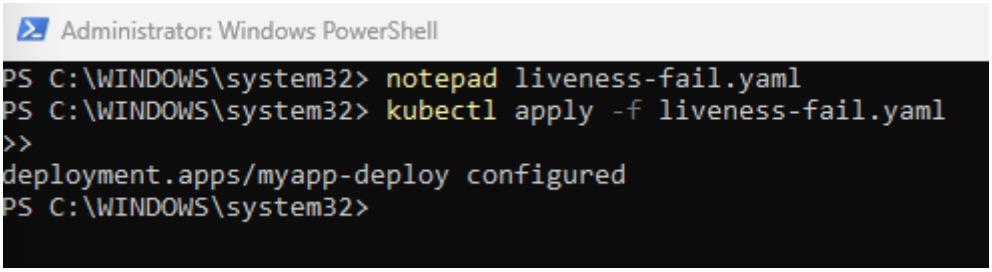
Change the liveness probe like this:

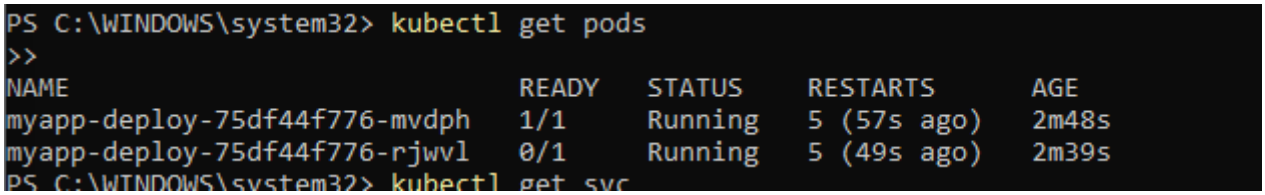
```
livenessProbe:
  httpGet:
    path: /failcheck   # This path does NOT exist in your app
    port: 80
  initialDelaySeconds: 5
```

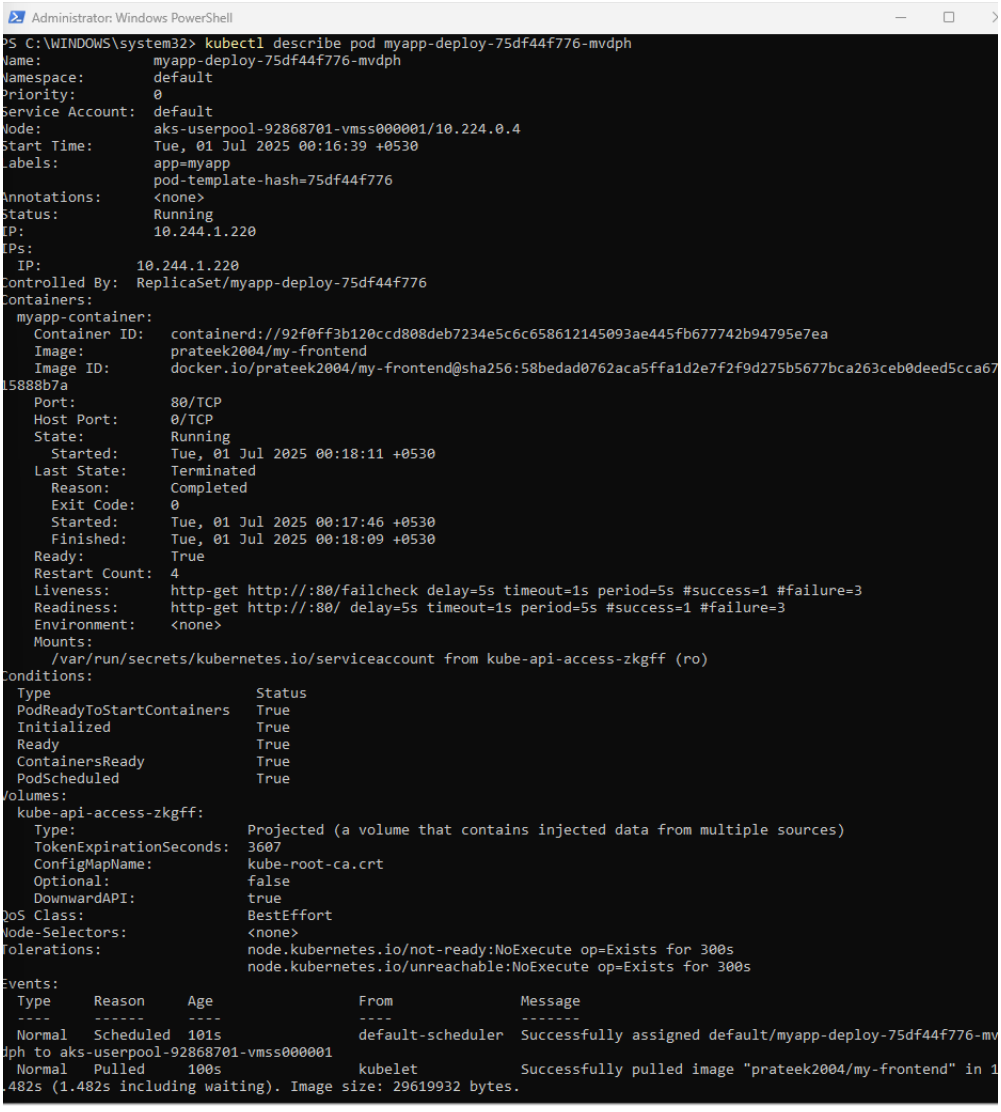
periodSeconds: 5

failureThreshold: 3

kubectl apply -f liveness-fail.yaml







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-rjwvl`**
  - **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.