Kubernetes may restart the container or take it out of operation if a Liveness Probe or Readiness Probe fails, which might affect the stability of the application. You can diagnose and fix these problems more effectively if you know the causes and fixes.

- ✓ **What exactly Liveness & Readiness Probes means?**
  - ➢ **Liveness Probe** → Checks whether the container is **alive** (responsive). If it fails, Kubernetes **restarts the pod**.
  - ➢ **Readiness Probe** → Checks if the container is **ready to accept traffic**. If it fails, Kubernetes **removes the pod from load balancer rotations** until it recovers.

```yaml
livenessProbe:
  httpGet:
    path: /health
    port: 8080
  initialDelaySeconds: 15
  periodSeconds: 10
  timeoutSeconds: 3
  failureThreshold: 3

readinessProbe:
  httpGet:
    path: /ready
    port: 8080
  initialDelaySeconds: 10
  periodSeconds: 5
  timeoutSeconds: 2
  failureThreshold: 3
```

Detailed Explination :

```yaml
livenessProbe:
  httpGet:
    path: /health
    port: 8080
  initialDelaySeconds: 5
  periodSeconds: 10
```

- ✓ **What it does:**
  - ➢ **Liveness probe** checks if the container is still running and hasn't frozen.
  - ➢ Starts probing **5 seconds** after the container starts.
  - ➢ Repeats the check every **10 seconds**.

- If /health fails (non-2xx), Kubernetes **kills and restarts** the container.

```
readinessProbe:
  httpGet:
    path: /ready
    port: 8080
  initialDelaySeconds: 5
  periodSeconds: 10
```

- ✓ **What it does exactly :**
  - **Readiness probe** checks if the container is ready to receive traffic.
  - Begins probing **5 seconds** after startup.
  - Every **10 seconds**, Kubernetes checks /ready.
  - If it fails, the container stays running but is **removed from the service load balancer** (no traffic routed).
- ✓ **Separate endpoints**: /health and /ready should ideally have different logic.
  - /health: Can be a simple "app is up" check.
  - /ready: Should check downstream dependencies like DB, cache, etc.

A Few Typical Reasons and Solutions for Probe Failures :

- ✓ **Probe Path or Port is Incorrect**

Problem: The probe is targeting an **invalid endpoint or wrong port**.

Fix:

- ➢ Check if the application actually **exposes the health endpoint** using:

```
curl -I http://localhost:8080/health
```

- ➢ Ensure the probe **targets the correct path & port** in the YAML file.

- ✓ **Application Startup Delay**

Problem: The container **takes too long to start**, causing early probe failures.

Fix: Increase initialDelaySeconds to allow extra startup time.

```
livenessProbe:
  initialDelaySeconds: 30  # Allow the app more time to initialize
```

- ✓ **Network Issues in which Pod Cannot Reach Probe Endpoint**

Problem: The probe fails because **network policies** block access to the container's endpoint.

Fix:

- ➢ Verify connectivity **inside the pod**:

kubectl exec -it <pod-name> -- curl -I http://localhost:8080/health

- ➢ Check **network policies** and firewall settings using:

kubectl get networkpolicies

- ✓ **Resource Constraints in which High CPU/Memory Usage**

Problem: The pod **struggles with performance**, causing delayed responses.

Fix:

- Check resource limits using:

  kubectl describe pod <pod-name> | grep -i Limits

- Increase CPU & memory limits:

```
resources:
  requests:
    memory: "512Mi"
    cpu: "250m"
  limits:
    memory: "1Gi"
    cpu: "500m"
```

- ✓ **Requests (requests.memory & requests.cpu)**
  - **Minimum amount of resources** the container will get.
  - Kubernetes **guarantees** this much memory/CPU to the container.
  - Helps with scheduling: Kubernetes ensures the node has enough resources **before** placing the pod.
- ✓ **Limits (limits.memory & limits.cpu)**
  - **Maximum resources** a container can use.
  - Prevents resource-hungry containers from consuming **too much** and affecting others.
  - If the container exceeds **CPU limits**, it **gets throttled**.
  - If it exceeds **memory limits**, Kubernetes **kills the pod (OOMKilled)**.

Example:

```
resources:
  limits:
    memory: "1Gi"
    cpu: "500m"
```

Memory Limit: 1Gi

- The container can use up to **1GB of RAM**, but not more.
- **CPU Limit: 500m**
- The container can use **500 millicores (0.5 vCPU)** at max.

*If the app tries to use **more than 500m CPU**, Kubernetes **throttles** it. If the app tries to use **more than 1Gi memory**, Kubernetes **kills the container** with an OOMKilled error.*

```
resources:
  requests:
    memory: "512Mi"
    cpu: "250m"
```

- **Memory Request: 512MiB** → This pod is guaranteed **512 megabytes of RAM** when scheduled.
- **CPU Request: 250m** → This pod gets **250 millicores** (0.25 vCPU) as its baseline.

*The container will always get at least **512Mi memory and 0.25 CPU** when running.*