Kubernetes Pod Probes

Introduction

Kubernetes probes help ensure your applications remain healthy and available by continuously checking their status inside Pods.

This presentation covers:

- What probes are
- How each probe works
- When and why to use each

What Are Probes?

Kubernetes Probes are health checks run by the Kubelet on containers:

- **Liveness Probe**: Is the app still running?
- **Readiness Probe**: Is the app ready to serve traffic?
- **Startup Probe**: Has the app finished starting up?

They can run using:

- HTTP GET
- TCP Socket
- Exec command

Liveness Probe

Purpose: Detects if the app is still running.

Behavior: If it fails, the Pod is restarted.

Use case: Recover from deadlocks or infinite loops.

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

Readiness Probe

Purpose: Determines if the app is ready to handle requests.

Behavior: Pod is removed from Service endpoints if it fails.

Use case: During warm-up or when dependencies aren't ready.

```
readinessProbe:
  httpGet:
    path: /ready
    port: 8080
  initialDelaySeconds: 5
  periodSeconds: 3
```

Startup Probe

Purpose: Allows slow-starting apps to complete before other probes run.

Behavior: Disables liveness until it passes.

Use case: Apps like Java Spring Boot that need time to start.

```
startupProbe:
  httpGet:
    path: /startup
    port: 8080
  failureThreshold: 30
  periodSeconds: 10
```

Probe Timing Fields

- **initialDelaySeconds**: How long to wait after container starts
- **periodSeconds**: How often to perform the probe
- **timeoutSeconds**: Max time to wait for a response
- **failureThreshold**: # of failures before marking unhealthy
- **successThreshold**: # of successes before marking healthy again

These fields let you tune probes based on your app's behavior.

Probe Types

- `httpGet`: Best for HTTP services with health endpoints
- `tcpSocket`: Good for database or non-HTTP apps
- `exec`: Runs a command inside the container

Choose based on the app type and what's meaningful for health.

Common Anti-patterns

- Using `/` as the probe path instead of a dedicated `/health` or `/ready`
- Restarting containers with liveness probe on transient errors
- Setting too low timeouts leading to false negatives
- Not using a startup probe for slow apps

Avoid these to prevent unnecessary restarts and outages.

Monitoring & Debugging Probes

- Use `kubectl describe pod` to view probe failure events
- Use `kubectl logs` to correlate with app behavior
- Setup metrics-server or Prometheus to track probe stats

Watch for patterns of failure and adjust thresholds accordingly.

Real-World Example

A Spring Boot app:

- Takes 60s to start
- Requires DB connection
- Can hang under load

Suggested Probes:

- Startup: Allow 70s for boot
- Readiness: Check DB before traffic
- Liveness: Restart if hung

Conclusion

- Probes help Kubernetes maintain app health and reliability.
- Use all three for robust production deployments.
- Fine-tune delays and thresholds per app needs.