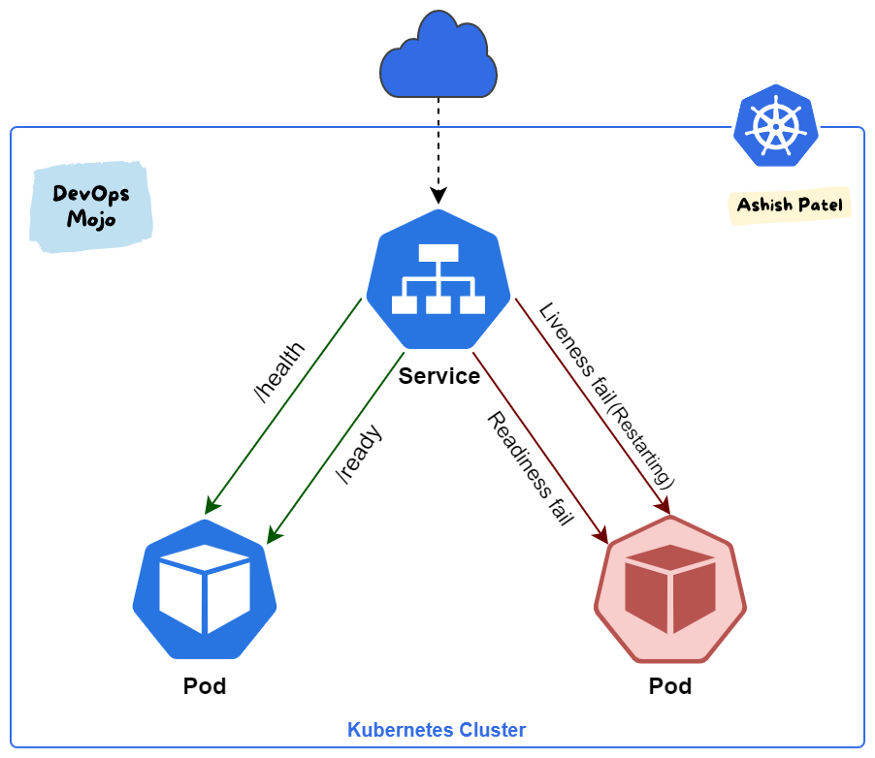
**Kubernetes — Probes (Liveness, Readiness, and Startup) Overview**

Introduction to Types of Probes and Configure Health Checks using Probes in Kubernetes.



Kubernetes provides probes (health checks) to monitor and act on the state of Pods (Containers) and to make sure only healthy Pods serve traffic. With help of Probes, we can control when a pod should be deemed started, ready for service, or live to serve traffic.

**Probes are used to detect:**

* Containers that haven’t started yet and can’t serve traffic.
* Containers that are overwhelmed and can’t serve additional traffic.
* Containers that are completely dead and not serving any traffic.

**Types of Probes**

Kubernetes gives you three types of health checks probes:

1. Liveness
2. Readiness
3. Startup

**Liveness Probe**

* Liveness probes let Kubernetes know whether your app (running in a container inside Pod) is healthy.
* Indicates whether the container is running.
* If app is healthy, Kubernetes will not interfere with pod functioning. If app is unhealthy, Pod will be marked as unhealthy. If a Pod fails health-checks continuously, the Kubernetes terminates the pod and starts a new one.
* Liveness probes could catch a deadlock, where an application is running, but unable to make progress. Restarting a container in such a state can help to make the application more available despite bugs.
* Kubelet uses liveness probes to know when to restart a container. If the liveness probe fails, the kubelet kills the container, and the container is subjected to its restart policy.
* Configure a health check that periodically checks if your application is still alive. Containers can be restarted if the check fails.
* Kubelet will periodically send a GET request to the Pod and expect a response; if the Pod responds between 200–399 then the kubelet considers that Pod is healthy; otherwise, the Pod will be marked as unhealthy.

**Use Cases**

* Use Liveness Probe to remove unhealthy pods.
* If you’d like your container to be killed and restarted if a probe fails, then specify a liveness probe, and specify a restartPolicy of Always or OnFailure.
* Use a liveness probe for a container that can fail with no possible recovery.
* If your container cannot crash by itself when there is an unexpected error occur, then use liveness probes. Using liveness probes can overcome some of the bugs the process might have.

**Recommendation**

The recommended way to implement the liveness probe is for application to expose a /health HTTP endpoint. When receiving a request on this endpoint, the application should send a “200 OK” response if it is considered healthy. Healthy means the following:

* Application is running.
* Its main dependencies are met (e.g. app can access database).

**Readiness Probe**

* Readiness probes let Kubernetes know when your app (running in a container inside Pod) is ready to serve traffic.
* Indicates whether the container is ready to respond to requests.
* Kubernetes makes sure the readiness probe passes before allowing a service to send traffic to the pod.
* Unlike a liveness probe, a readiness probe doesn’t kill the container. If the readiness probe fails, Kubernetes simply hides the container’s Pod from corresponding Services, so that no traffic is redirected to it.
* Liveness probes do not wait for readiness probes to succeed. If you want to wait before executing a liveness probe you should use initialDelaySeconds or startupProbe.
* A side-effect of using Readiness Probes is that they can increase the time it takes to update Deployments.

**Use Cases**

* Use Readiness Probe to detect partial unavailability.
* If you’d like to start sending traffic to a Pod only when a probe succeeds, specify a readiness probe.
* If you want your container to be able to take itself down for maintenance, you can specify a readiness probe that checks an endpoint specific to readiness that is different from the liveness probe.

**Recommendation**

The recommended way to implement the readiness probe is for your application to expose a /ready HTTP endpoint. When it receives a request on this endpoint, the application should send a “200 OK” response if it is ready to receive traffic. Ready to receive traffic means the following:

* Application is healthy.
* Any potential initialization steps are completed.
* Any valid request sent to the application doesn’t result in an error.

**Startup Probe**

* Startup probes let Kubernetes know whether your app (running in a container inside Pod) has properly started.
* Indicates whether the application within the container is started.
* Startup probe has higher priority over the two other probe types. Until the Startup Probe succeeds, all the other Probes are disabled.
* Startup probes are useful in situations where your app can take a long time to start, or could occasionally fail on startup.
* Rather than set a long liveness interval, you can configure a separate configuration for probing the container as it starts up, allowing a time longer than the liveness interval would allow.

**Use Cases**

* Use Startup Probe for applications that take longer to start.
* When your app needs additional time to startup, you can use the Startup Probe to delay the Liveness and Readiness Probe.
* If your container needs to work on loading large data, configuration files, or migrations during startup, you can use a startup probe.

**Probes Key Points**

* While Liveness probe detects failures in an app that are resolved by terminating the Pod (i.e. restarting the app), Readiness Probe detects conditions where the app may be temporarily unavailable.
* Liveness probe passes when the app itself is healthy, but the readiness probe additionally checks that each required back-end service is available. This helps you avoid directing traffic to Pods that can only respond with error messages.
* By combining liveness and readiness probes, you can instruct Kubernetes to automatically restart pods or remove them from backend groups. If your app has a strict dependency on back-end services, you can implement both liveness and readiness probes.
* By default, the probe will stop if the application is not ready after three attempts. In case of a liveness probe, it will restart the container. In the case of a readiness probe, it will mark pods as unhealthy.
* In many applications, the /health and /ready endpoints are merged into a single /health endpoint because there is no real difference between their healthy and ready states.

**Ways to check Pods/Containers’ health using Probes**

Kubelet can check a Pods’ health in three ways. Each probe must define exactly one of these mechanisms:

**HTTP**

* We define a Port number along with the URL. Kubernetes pings this path, and if it gets an HTTP response in the 200 or 300 range, it marks the app as healthy. Otherwise, it is marked as unhealthy.
* HTTP probes are the most common type of custom probe.
* Even if your app isn’t an HTTP server, you can create a lightweight HTTP server inside your app to respond to the liveness probe.

**TCP**

* Kubernetes tries to establish a TCP connection on the specified port. If it can establish a connection, the container is considered healthy; if it can’t it is considered unhealthy.
* TCP probes come in handy if you have a scenario where HTTP probes or Command probes don’t work well.
* For example, a gRPC or FTP service is a prime candidate for this type of probe.

**Command**

* Kubernetes runs a command inside the container. If the command returns with exit code 0, then the container is marked as healthy. Otherwise, it is marked unhealthy.
* This type of probe is useful when you can’t or don’t want to run an HTTP server, but can run a command that can check whether or not your app is healthy.

**Probe Parameters**

Probes have a number of fields that can be used to more precisely control the behavior of liveness and readiness checks.

* initialDelaySeconds: Number of seconds after the container has started before probes are initiated. (default: 0, minimum: 0)
* periodSeconds: How often to perform the probe (i.e. frequency). (default: 10, minimum: 1)
* timeoutSeconds: Number of seconds after which the probe times out. (default: 1, minimum: 1)
* successThreshold: Minimum consecutive successes for the probe to be considered successful after failure. (default: 1, minimum: 1)
* failureThreshold: How many failed results were received to transition from a healthy to a failure state. (default: 3, minimum: 1)

apiVersion: v1

kind: Pod

metadata:

labels:

app: myapp

name: myapp

spec:

containers:

- name: myapp

image: k8s.gcr.io/myapp

livenessProbe:

httpGet:

path: /healthz

port: 8080

httpHeaders: # optional

- name: Custom-Header

value: Awesome

initialDelaySeconds: 3

periodSeconds: 3

readinessProbe:

tcpSocket:

port: 8080

initialDelaySeconds: 5

periodSeconds: 10

startupProbe:

httpGet:

path: /healthz

port: 8080

# port: liveness-port, # way-2: for apply to all probes types

failureThreshold: 30

periodSeconds: 10

# Kubernetes — Liveness and Readiness Probes — Difference



Liveness and Readiness probes are used to control the health of an application running inside a Pod’s container. Both of them are very similar in functionality, and usage.

# Liveness Probe

Suppose that a Pod is running our application inside a container, but due to some reason let’s say memory leak, cpu usage, application deadlock etc the application is not responding to our requests, and stuck in error state.

Liveness probe checks the container health as we tell it do, and if for some reason the liveness probe fails, it restarts the container. We can define liveness probe in 3 ways:

## Liveness command

**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: 3  
 periodSeconds: 5**

We are creating a container with name **liveness**, and as the container initialise we use the following command:

**- touch /tmp/healthy; sleep 30; rm -rf /tmp/healthy; sleep 600**

to create a file **healthy** at path **/tmp/healthy**, and delete it after 30 seconds.

**exec:  
 command:  
 - cat  
 - /tmp/healthy**

This commands tell the liveness probe to open file at path **/tmp/healthy**, and if it can’t the liveness probe will fail and container will restart.

**initialDelaySeconds: 3**

This is the delay which tells kubelet to wait for 3 seconds before performing the first probe

**periodSeconds: 5**

This field specifies that kubelet should perform a probe every 5 seconds.

So, according to above example our container will start and work fine for first 30 seconds, and after that liveness probe will fail and restart the container.

## Liveness HTTP request

**livenessProbe:  
 httpGet:  
 path: /healthz  
 port: 8080  
 initialDelaySeconds: 3  
 periodSeconds: 3**

In this case the kubelet will send **HTTP GET** request to **/healthz**endpoint at port **8080** of the application running inside the container. If the response is an error liveness probe will. Otherwise, it will consider the application to be alive.

## TCP Liveness probe

**livenessProbe:  
 tcpSocket:  
 port: 8080  
 initialDelaySeconds: 15  
 periodSeconds: 20**

In this case the kubelet will try to open a tcp socket at port 8080 in container running the application. If it succeeds the application will be considered healthy, otherwise the probe will fail, and the container will restart.

# Readiness Probe

In some cases we would like our application to be alive, but not serve traffic unless some conditions are met e.g, populating a dataset, waiting for some other service to be alive etc. In such cases we use readiness probe. If the condition inside readiness probe passes, only then our application can serve traffic.

Readiness probe is defined in 3 ways exactly like the Liveness probe above. We just need to replace **livenessProbe** with **readinessProbe** like this:

**readinessProbe:  
 exec:  
 command:  
 - cat  
 - /tmp/healthy  
 initialDelaySeconds: 5  
 periodSeconds:** 5

# Summary

Both liveness & readiness probes are used to control the health of an application. Failing liveness probe will restart the container, whereas failing readiness probe will stop our application from serving traffic.