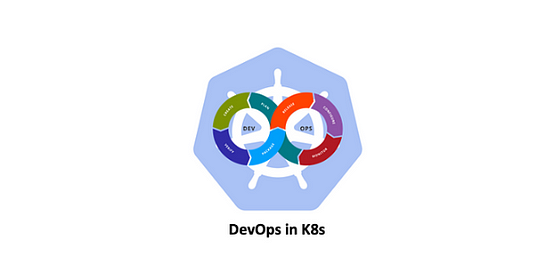
**K8s Tips — Pod Probes Introduction**



**What are K8s Pod Probes**

In K8s, Pod probes are diagnostics performed periodically by the kubelet on a container. K8s uses three types of probes, each serving a different purpose:

* **Startup probes:** These probes indicate whether the application within the container has started successfully. It aims to help slow-starting containers get up and running. If the startup probe fails, K8s will kill the container, and the container’s failure policy will dictate further action (e.g., it may be restarted). This probe is only executed at startup, and once successful, both liveness and readiness probes take over.
* **Liveness probes:** This ascertains whether the container is running. If the liveness probe fails, K8s will kill the container, and the container’s failure policy will apply. The policy might be to restart the container, in which case the kubelet will attempt to restart it. The liveness probe helps to catch situations where an application is running but unable to make progress.
* **Readiness probes:** This investigates whether the container is ready to handle requests. If the readiness probe fails, the Pod will be removed from service load balancers, and won’t receive any traffic through K8s Services. Once the readiness probe passes, the Pod will be capable of taking requests. This feature ensures that only Pods capable of serving requests receive them.

By understanding and properly configuring these probes, you can create self-healing applications that respond well to unexpected issues.

**What Happens if no Probes?**

If a Pod doesn’t have any probes configured, K8s will simply assume that the Pod’s containers are alive and ready to accept traffic as soon as they are started. K8s will not take any action to verify the actual status of the containers in the Pod. Take the following Pod configuration as an example:

apiVersion: v1  
kind: Pod  
metadata:  
 name: simple-pod  
spec:  
 containers:  
 - name: nginx-container  
 image: nginx

If you run the kubectl command to create the Pod, you will observe the following outputs:

$ kubectl apply -f simple-pod.yaml  
pod/simple-pod created  
  
$ kubectl describe pod simple-pod  
Events:  
 Type Reason Age From Message  
 ---- ------ ---- ---- -------  
 Normal Scheduled 10s default-scheduler Successfully assigned default/simple-pod to minikube

Normal Pulling 9s kubelet Pulling image "nginx"

Normal Pulled 2s kubelet Successfully pulled image "nginx" in 6.574376891s

Normal Created 2s kubelet Created container nginx-container

Normal Started 2s kubelet Started container nginx-container

This can have a few implications:

**Availability**: If a container in a Pod crashes or falls into an unhealthy state (like a deadlock), K8s won’t know about it and won’t take any action to remedy the situation. This can affect the availability of your application.

**Unnecessary Traffic**: K8s doesn’t know when the Pod is ready to serve traffic without a readiness probe. Traffic might be sent to a Pod that is still starting up or initializing, leading to errors.

**Slow Startup**: In the absence of a startup probe, if a container doesn’t start within the configured period of the liveness probe, K8s might mistakenly think that the container is unhealthy and kill it, even though it is just slow to start.

So, while it’s possible to run a Pod without probes, it might lead to unnecessary problems. It is generally a good practice to configure appropriate probes for your Pods to ensure that K8s can maintain the health of your applications effectively.

**How Does Probe Work Under the Hood**

Behind the scene, the kubelet runs each type of probe differently based on its configuration. For example:

spec:  
 containers:  
 - name: my-app  
 image: my-image  
 livenessProbe:  
 exec:  
 command:  
 - cat  
 - /tmp/healthy  
 initialDelaySeconds: 5  
 periodSeconds: 5

In the above example, the kubelet will run the command cat /tmp/healthy within the container every 5 seconds, starting 5 seconds after the container has started. If the command succeeds (i.e., the file exists), the kubelet considers the container to be alive and healthy.

Note that when the kubelet needs to execute a command in a container, it doesn’t literally “go into” the container itself. Instead, it communicates with the container runtime (like Docker or containerd) and tells it to run the command in the specified container.

**Here’s a simplified view of the process:**

* kubelet periodically checks the liveness/readiness/startup probes configuration for each container in the Pod based on the periodSeconds parameter.
* If a probe is configured with an exec action, kubelet sends a request to the container runtime to execute the specified command inside the targeted container.
* The container runtime runs this command inside the container’s environment.
* The command’s exit status is returned back up the chain — from the container runtime to the kubelet.
* kubelet uses this exit status to determine the result of the probe.

**Pod Probe Actions**

These probes can be executed with three different actions:

* **Exec**: Executes a specified command inside the container. If it returns a zero-exit code, the probe is successful.

livenessProbe:  
 exec:  
 command:  
 - cat  
 - /tmp/healthy\_file  
 initialDelaySeconds: 5  
 periodSeconds: 5

* **HTTP GET**: Executes an HTTP GET request against the container’s IP address, a port and path you specify. If the response has a status code greater than or equal to 200 but less than 400, the probe is successful.

readinessProbe:  
 httpGet:  
 path: /api/ready  
 port: 8080  
 initialDelaySeconds: 5  
 periodSeconds: 5

* **TCP Socket**: Tries to open a TCP socket in the specified container. If it can establish a connection, the probe is successful.

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

For all actions, you can optionally specify the initialDelaySeconds (how long to wait before performing the first probe) and periodSeconds (how often to run the probe). They have the following default values:

* **initialDelaySeconds**: This field defaults to 0. This means that the probe is initiated immediately as soon as the container starts.
* **periodSeconds**: This field defaults to 10. This means that the probe is performed every 10 seconds.

**Startup Probe**

If a startup probe fails, K8s assumes that the application inside the container has failed to start. It then kills the container and, depending on the restart policy of the Pod, it may also restart the container. The liveness and readiness probes will not be executed until the startup probe passes.

Here’s an example of a startup probe configuration:

apiVersion: v1  
kind: Pod  
metadata:  
 name: startup-probe-demo  
spec:  
 containers:  
 - name: startup-probe-container  
 image: my-app:1.0  
 startupProbe:  
 httpGet:  
 path: /healthz  
 port: 8080  
 failureThreshold: 30  
 periodSeconds: 10

In this example, an HTTP GET request is made to the /healthz endpoint every 10 seconds (periodSeconds: 10). If the endpoint does not return a successful response within 30 attempts (failureThreshold: 30), the kubelet kills the container, and the container’s restart policy applies.

Best practices:

* **Set an appropriate failure threshold and period:** Since startup probes are designed for slow-starting containers, the failureThreshold and periodSeconds parameters should be set to give your container enough time to start. The failureThreshold \* periodSeconds should be long enough to cover the worst-case startup time.
* **Use the same command as the liveness probe:** Typically, if an application is ready for a liveness check, it’s ready to start. So, in many cases, the startup probe will have the same command as the liveness probe.
* **Avoid overlapping with liveness/readiness probes:** Make sure that the startup time (failureThreshold \* periodSeconds) is long enough that the liveness and readiness probes don’t start running before the application is ready.

**Readiness Probe**

If a readiness probe fails, K8s assumes that the Pod is not ready to serve traffic. The Pod’s IP address will be removed from the endpoints of all services that match the Pod. This means the Pod will stop receiving any new traffic, but it does not affect existing connections. The readiness probe failure does not cause the container to be restarted.

Example of Readiness probe:

apiVersion: v1  
kind: Pod  
metadata:  
 name: readiness-probe-demo  
spec:  
 containers:  
 - name: my-app  
 image: my-app:1.0  
 readinessProbe:  
 httpGet:  
 path: /ready  
 port: 8080  
 initialDelaySeconds: 5  
 periodSeconds: 5

In this example, K8s will start to send HTTP GET requests to the /ready endpoint 5 seconds after the container starts. It will repeat this check every 5 seconds. If the endpoint returns a successful HTTP status code (between 200 and 399), the container is marked as ready and can start receiving traffic.

**Best practices for readiness probes:**

* **Set an appropriate initial delay:** Use the initialDelaySeconds parameter to give your application enough time to start before the readiness checks begin. If your application is not ready when the first check runs, it will be marked as not ready and won’t receive any traffic from Services.
* **Choose the right probe type for your application:** If your application has an HTTP interface, an httpGet probe is a good choice. If you need to check a certain file is present or certain process is running, an exec probe is more appropriate. If you need to check if a port is open, use a tcpSocket probe.
* **Understand the implications of failure:** If a readiness probe fails, the Pod will be removed from service load balancers and won’t receive traffic through Services until it passes the readiness check again. Make sure that your application can recover from this situation.
* **Avoid long checks:** A long readiness check might delay the time when the Pod starts to receive traffic. If the readiness check needs to perform a long operation (like warming up a cache), it might be better to do that operation ahead of time, or during the start of the application, instead of during the readiness check.

**Liveness Probe**

If a liveness probe fails, K8s assumes that the container has become unresponsive or deadlocked and needs to be restarted. K8s will kill the container and then restart it, according to the Pod’s restart Policy.

Example of Liveness probe:

apiVersion: v1  
kind: Pod  
metadata:  
 name: liveness-probe-demo  
spec:  
 containers:  
 - name: my-app  
 image: my-app:1.0  
 livenessProbe:  
 httpGet:  
 path: /healthz  
 port: 8080  
 initialDelaySeconds: 15  
 periodSeconds: 20

In this case, K8s will send an HTTP GET request to the /healthz endpoint on port 8080 of the application running in the container. Fifteen seconds after the container starts, this check is performed every 20 seconds. If the endpoint returns a successful HTTP status code (between 200 and 399), the liveness probe is successful. If it does not, Kubernetes restarts the container.