**Probes**

Probes are kubelet’s answer to the health checks, there are three handlers:

* [**ExecAction**](https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.19/#execaction-v1-core): Command execution check, if the command’s exit status is 0, it is considered a success.
* **TCPSocketAction**: TCP check to determine if the port is open, if open, it is considered a success.
* **HTTPGetAction**: HTTP check to determine if the status code is equal to or above 200 and below 400.

Each type of probe has common configurable fields:

* **initialDelaySeconds**: Probes start running after initialDelaySeconds after container is started (default: 0)
* **periodSeconds**: How often probe should run (default: 10)
* **timeoutSeconds**: Probe timeout (default: 1)
* **successThreshold**: Required number of successful probes to mark container healthy/ready (default: 1)
* **failureThreshold**: When a probe fails, it will try failureThreshold times before deeming unhealthy/not ready (default: 3)

These parameters need to be configured per your application’s spec.

[**#**](https://loft.sh/blog/kubernetes-readiness-probes-examples-and-common-pitfalls?utm_medium=reader&utm_source=other&utm_campaign=blog_kubernetes-probes-startup-liveness-readiness#exec-probe)**Exec Probe**

Exec action has only one field, and that is command. Exit status of the command is checked, and the status of zero (0) means it is healthy, and other value means it is unhealthy.

readinessProbe:

initialDelaySeconds: 1

periodSeconds: 5

timeoutSeconds: 1

successThreshold: 1

failureThreshold: 1

exec:

command:

- cat

- /etc/nginx/nginx.conf

### TCP Probe

We need to define host and port parameters, host parameter defaults to the cluster-internal pod IP.

readinessProbe:

initialDelaySeconds: 1

periodSeconds: 5

timeoutSeconds: 1

successThreshold: 1

failureThreshold: 1

tcpSocket:

host:

port: 80

**HTTP Probe**

HTTP Probe has additional options to configure.

* **host**: Host/IP to connect to (default: pod IP)
* **scheme**: Scheme to use when making the request (default: HTTP)
* **path**: Path
* **httpHeaders**: An array of headers defined as header/value map
* **port**: Port to connect to

Tip: If you need to set the Host header of the HTTP, please do so on httpHeaders, instead of the host parameter.

readinessProbe:

initialDelaySeconds: 1

periodSeconds: 2

timeoutSeconds: 1

successThreshold: 1

failureThreshold: 1

httpGet:

host:

scheme: HTTP

path: /

httpHeaders:

- name: Host

value: myapplication1.com

port: 80

initialDelaySeconds: 5

periodSeconds: 5

**Readiness Probes in Kubernetes**

Kubernetes runs readiness probes to understand when it can send traffic to a pod, i.e., to transition the pod to Ready state. For example, if a pod is used as a backend endpoint for a service, a readiness probe will determine if the pod will receive traffic or not. The readiness probe is executed throughout the pod’s lifetime; this means that readiness probes continue to be executed after reaching the Ready state. Also, our application can make itself unavailable for maintenance or for some background tasks as well by responding with different values to the probe.

A pod transitions the following states:

* **PodScheduled**: Pod is scheduled on an available node
* **ContainersReady**: All containers in Ready state
* **Initialized**: All init containers are started
* **Ready**: Pod is ready to receive traffic (readiness probe)

## Example: Sample Nginx Deployment

To see readiness probes in action, we can configure a sample app, a simple NGINX web server in this case. Here we have defined a simple deployment configuration:

apiVersion: apps/v1

kind: Deployment

metadata:

name: k8s-probes

labels:

app: nginx

spec:

replicas: 1

selector:

matchLabels:

app: nginx

template:

metadata:

labels:

app: nginx

spec:

containers:

- name: nginx

image: nginx

ports:

- containerPort: 80

readinessProbe:

initialDelaySeconds: 1

periodSeconds: 2

timeoutSeconds: 1

successThreshold: 1

failureThreshold: 1

httpGet:

host:

scheme: HTTP

path: /

httpHeaders:

- name: Host

value: myapplication1.com

port: 80

Save this configuration to a file called k8s-probes-deployment.yaml and apply it with kubectl apply -f k8s-probes-deployment.yaml.

apiVersion: v1

kind: Service

metadata:

labels:

app: nginx

name: nginx

namespace: default

spec:

ports:

- name: nginx-http-port

port: 80

selector:

app: nginx

sessionAffinity: None

type: NodePort

<https://loft.sh/blog/kubernetes-readiness-probes-examples-and-common-pitfalls?utm_medium=reader&utm_source=other&utm_campaign=blog_kubernetes-probes-startup-liveness-readiness>

Liveness probes check if the pod is healthy, and if the pod is deemed unhealthy, it will trigger a restart; this action is different than the action of [Readiness Probes I discussed in my previous post](https://loft.sh/blog/kubernetes-readiness-probes-examples-common-pitfalls/?utm_medium=reader&utm_source=other&utm_campaign=blog_kubernetes-liveness-probes-examples-and-common-pitfalls).

Let’s look at the components of the probes and dive into how to configure and troubleshoot Liveness Probes.

## Probes

Probes are health checks that are executed by kubelet.

All probes have five parameters that are crucial to configure.

* **initialDelaySeconds**: Time to wait after the container starts. (default: 0)
* **periodSeconds**: Probe execution frequency (default: 10)
* **timeoutSeconds**: Time to wait for the reply (default: 1)
* **successThreshold**: Number of successful probe executions to mark the container healthy (default: 1)
* **failureThreshold**: Number of failed probe executions to mark the container unhealthy (default: 3)

You need to analyze your application’s behavior to set these probe parameters.

There are three types of probes:

### Exec Probe

Exec probe executes a command inside the container without a shell. The command’s exit status determines a healthy state - zero is healthy; anything else is unhealthy.

livenessProbe:

initialDelaySeconds: 1

periodSeconds: 5

timeoutSeconds: 1

successThreshold: 1

failureThreshold: 1

exec:

command:

- cat

- /etc/nginx/nginx.conf

### TCP Probe

TCP probe checks if a TCP connection can be opened on the port specified. An open port is deemed a success, closed port or reset are deemed unsuccessful.

livenessProbe:

initialDelaySeconds: 1

periodSeconds: 5

timeoutSeconds: 1

successThreshold: 1

failureThreshold: 1

tcpSocket:

host:

port: 80

### HTTP Probe

HTTP probe makes an HTTP call, and the status code determines the healthy state, between including 200 and excluding 400 is deemed success. Any status code apart from those mentioned is deemed unhealthy.

Here are HTTP Probes additional parameters to configure.

* **host**: IP address to connect to (default: pod IP)
* **scheme**: HTTP scheme (default: HTTP)
* **path**: HTTP path to call to
* **httpHeaders**: Any custom headers you want to send.
* **port**: Connection port.

*Tip: If Host header is required, than use httpHeader.*

An example of an HTTP probe.

livenessProbe:

initialDelaySeconds: 1

periodSeconds: 2

timeoutSeconds: 1

successThreshold: 1

failureThreshold: 1

httpGet:

host:

scheme: HTTP

path: /

httpHeaders:

- name: Host

value: myapplication1.com

port: 80

initialDelaySeconds: 5

periodSeconds: 5

## Liveness Probes in Kubernetes

Kubelet executes liveness probes to see if the pod needs a restart. For example, let’s say we have a microservice written in Go, and this microservice has some bugs on some part of the code, which causes a freeze in runtime. To avoid hitting the bug, we can configure a liveness probe to determine if the microservice is in a frozen state. This way, the microservice container will be restarted and come to a pristine condition.

If your application gracefully exits when encountering such an issue, you won’t necessarily need to configure liveness probes, but there can still be bugs you don’t know about. The pod will be restarted as per the configured/default restart policy.

## Example: Sample Nginx Deployment

We will deploy Nginx as a sample app. below is the deployment and service configuration.

apiVersion: apps/v1

kind: Deployment

metadata:

name: k8s-probes

labels:

app: nginx

spec:

replicas: 1

selector:

matchLabels:

app: nginx

template:

metadata:

labels:

app: nginx

spec:

containers:

- name: nginx

image: nginx

ports:

- containerPort: 80

livenessProbe:

initialDelaySeconds: 1

periodSeconds: 2

timeoutSeconds: 1

successThreshold: 1

failureThreshold: 1

httpGet:

host:

scheme: HTTP

path: /

httpHeaders:

- name: Host

value: myapplication1.com

port: 80

Write this configuration to a file called k8s-probes-deployment.yaml, and apply it with kubectl apply -f k8s-probes-deployment.yaml command.

apiVersion: v1

kind: Service

metadata:

labels:

app: nginx

name: nginx

namespace: default

spec:

ports:

- name: nginx-http-port

port: 80

selector:

app: nginx

sessionAffinity: None

type: NodePort

<https://loft.sh/blog/kubernetes-liveness-probes-examples-and-common-pitfalls?utm_medium=reader&utm_source=other&utm_campaign=blog_kubernetes-probes-startup-liveness-readiness>

<https://loft.sh/blog/kubernetes-probes-startup-liveness-readiness/>

<https://www.containiq.com/post/kubernetes-readiness-probe>

<https://kubebyexample.com/learning-paths/application-development-kubernetes/lesson-4-customize-deployments-application-2>

<https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-startup-probes/>

<https://medium.com/@AADota/kubernetes-liveness-and-readiness-probes-difference-1b659c369e17>

<https://developers.redhat.com/blog/2020/11/10/you-probably-need-liveness-and-readiness-probes#example_4__putting_it_all_together>

<https://cloud.google.com/blog/products/containers-kubernetes/kubernetes-best-practices-setting-up-health-checks-with-readiness-and-liveness-probes>