Probes

» Liveness & Readiness Probes

Overview

Create some simple Pods using the `alpine` image to test `Liveness` and `Readiness` probes.

These checks are performed by the `kubelet` that is running on the Node hosting the Pod.

Liveness (TCP)

Add a liveness probe that attempts to connect to TCP socket `637` for verification that the container is “alive”.

```
cat > liveness-tcp-pod.yaml <<EOF
apiVersion: v1
kind: Pod
metadata:
  name: liveness-tcp-pod
  labels:
    test: liveness-tcp
spec:
  containers:
  - name: liveness-tcp-pod
    image: alpine
    command: ["/bin/sleep"]
    args: ["3600"]
    livenessProbe:
      tcpSocket:
        port: 637
      initialDelaySeconds: 30
      periodSeconds: 3
EOF
```

Create the pod based on the manifest that we created.

```
kubectl create -f liveness-tcp-pod.yaml
```

Verify the pod was successfully created.

```
kubectl get pods -l test=liveness-tcp
```

Run `nc` inside the pod and have it listen on port 637

```
kubectl exec -it liveness-tcp-pod -- /bin/sh -c "nc -v -l 0.0.0.0 -p 637"
```

You should be able to see the kubelet's liveness checks begin after the 30 second initial delay.

In another terminal window: Observe the container status in real-time.

```
kubectl get pod liveness-tcp-pod -w -o yaml
```

In the original terminal window, kill the `nc` program by typing `Ctrl+c` .

The container will automatically restart once the liveness probe can no longer connect to the pod on 637.

```
kubectl get pods -l test=liveness-tcp
```

Delete the pod using the manifest.

```
kubectl delete -f liveness-tcp-pod.yaml
```

Readiness (TCP)

Let's create another pod, this time using a readiness probe.

```
cat > readiness-tcp-pod.yaml <<EOF
apiVersion: v1
kind: Pod
metadata:
  name: readiness-tcp-pod
  labels:
    test: readiness-tcp
spec:
  containers:
  - name: readiness-tcp-pod
    image: alpine
    command: ["/bin/sleep"]
    args: ["3600"]
    readinessProbe:
      tcpSocket:
        port: 637
      initialDelaySeconds: 5
      periodSeconds: 3
EOF
```

Create the pod using the manifest.

```
kubectl create -f readiness-tcp-pod.yaml
```

Verify the pod was successfully created, but observe `READY (0/1)` .

If this pod was exposed as a deployment, its endpoint would be removed from any associated services at this time.

```
kubectl get pods -l test=readiness-tcp
```

In another terminal window: Run the previous `nc` command inside the new pod to observe the `kubelet` readiness tcp probes.

```
kubectl exec -it readiness-tcp-pod -- /bin/sh -c "nc -v -l 0.0.0.0 -p 637"
```

While kublet probes the pod, check the `READY` status of your pod in the other terminal window. It should change to `(1/1)` .

```
kubectl get pods -l test=readiness-tcp
```

Delete the pod when you are finished.

```
kubectl delete -f readiness-tcp-pod.yaml
```

Readiness (HTTP)

With an `HTTP` liveness probe, any `2xx` or `3xx` return code indicates that the service is alive and healthy. Any other code indicates failure.

It is the container maintainer's responsibility to create a handler within the container to return the proper return codes.

Let's create a pod using the `httpGet` liveness probe.

```
cat > liveness-http-pod.yaml <<EOF
apiVersion: v1
kind: Pod
metadata:
  name: liveness-http-pod
  labels:
    test: liveness-http
spec:
  containers:
  - name: liveness-http-pod
    args:
    - /server
    image: gcr.io/google_containers/liveness
    livenessProbe:
      httpGet:
        path: /healthz
        port: 8080
        httpHeaders:
          - name: X-Custom-Header
            value: Awesome
      initialDelaySeconds: 3
      periodSeconds: 3
EOF
```

This particular container runs the following `golang` program:

```
http.HandleFunc("/healthz", func(w http.ResponseWriter, r *http.Request) {
    duration := time.Now().Sub(started)
    if duration.Seconds() > 10 {
        w.WriteHeader(500)
        w.Write([]byte(fmt.Sprintf("error: %v", duration.Seconds())))
    } else {
        w.WriteHeader(200)
        w.Write([]byte("ok"))
    }
})
```

Create the pod based on the manifest that we created.

```
kubectl create -f liveness-http-pod.yaml
```

Verify the pod was successfully created.

```
kubectl get pods -l test=liveness-http
```

Observe the container status in real-time. The container will be seen as alive for the first 10 seconds and then dead after returning the 500 return code.

```
kubectl get pod liveness-http-pod -w -o yaml
```

Delete the pod when you are finished.

```
kubectl delete -f liveness-http-pod.yaml
```

Another option for liveness/readiness is defining a specific command to run *inside* a container.

Exit status of `0` is treated as live/healthy and non-zero is unhealthy.

```
cat > liveness-exec-pod.yaml <<EOF
apiVersion: v1
kind: Pod
metadata:
  name: liveness-exec-pod
  labels:
    test: liveness-exec
spec:
  containers:
  - name: liveness-exec-pod
    args:
    - /bin/sh
    - -c
    - touch /tmp/healthy; sleep 30; rm -rf /tmp/healthy; sleep 600
    image: gcr.io/google_containers/busybox
    livenessProbe:
      exec:
        command:
        - cat
        - /tmp/healthy
      initialDelaySeconds: 5
      periodSeconds: 5
EOF
```

Create the pod based on the manifest.

```
kubectl create -f liveness-exec-pod.yaml
```

Verify that the pod was successfully created.

```
kubectl get pods -l test=liveness-exec
```

Observe the container status in real-time. The container will not restart for the first 30 seconds. Once the `healthy` file is removed, the `cat` command will exit code 1 and the container is then considered unhealthy and is restarted.

```
kubectl get pod liveness-exec-pod -w -o yaml
```

Clean Up

Delete the pod based on the manifest.

```
kubectl delete -f liveness-exec-pod.yaml
```

View any remaining resources.

```
kubectl get all
```