In this scenario, you'll learn how Kubernetes checks containers health using Readiness and Liveness Probes.

Readiness Probes checks if an application is ready to start processing traffic. This probe solves the problem of the container having started, but the process still warming up and configuring itself meaning it's not ready to receive traffic.

Liveness Probes ensure that the application is healthy and capable of processing requests.

```
controlplane $ cat deploy.yaml
kind: List
apiVersion: v1
items:
- kind: ReplicationController
  apiVersion: v1
  metadata:
    name: frontend
    labels:
      name: frontend
  spec:
    replicas: 1
    selector:
      name: frontend
    template:
      metadata:
        labels:
          name: frontend
      spec:
        containers:
        - name: frontend
          image: katacoda/docker-http-server:health
          readinessProbe:
            httpGet:
              path: /
              port: 80
            initialDelaySeconds: 1
            timeoutSeconds: 1
          livenessProbe:
            httpGet:
              path: /
              port: 80
            initialDelaySeconds: 1
            timeoutSeconds: 1
- kind: ReplicationController
  apiVersion: v1
  metadata:
    name: bad-frontend
    labels:
      name: bad-frontend
  spec:
    replicas: 1
    selector:
      name: bad-frontend
    template:
```

```
    metadata:
      labels:
        name: bad-frontend
    spec:
      containers:
      - name: bad-frontend
        image: katacoda/docker-http-server:unhealthy
        readinessProbe:
          httpGet:
            path: /
            port: 80
          initialDelaySeconds: 1
          timeoutSeconds: 1
        livenessProbe:
          httpGet:
            path: /
            port: 80
          initialDelaySeconds: 1
          timeoutSeconds: 1
- kind: Service
  apiVersion: v1
  metadata:
    labels:
      app: frontend
      kubernetes.io/cluster-service: "true"
    name: frontend
  spec:
    type: NodePort
    ports:
    - port: 80
      nodePort: 30080
    selector:
      app: frontend
```

Step 2 - Readiness Probe

When deploying the cluster, two pods were also deployed to demonstrate health checking. You can view the deployment with `cat deploy.yaml`.

When deploying the Replication Controller, each Pod has a Readiness and Liveness check. Each check has the following format for performing a healthcheck over HTTP.

```
livenessProbe:
  httpGet:
    path: /
    port: 80
  initialDelaySeconds: 1
  timeoutSeconds: 1
```

The settings can be changed to call different endpoints, for example, /ping, based on your application.

Get Status

The first Pod, *bad-frontend* is an HTTP service which always returns a 500 error indicating it hasn't started correctly. You can view the status of the Pod with `kubectl get pods --selector="name=bad-frontend"`

Kubectl will return the Pods deployed with our particular label. Because the healthcheck is failing, it will say that zero containers are ready. It will also indicate the number of restart attempts of the container.

To find out more details of why it's failing, describe the Pod.

```
pod=$(kubectl get pods --selector="name=bad-frontend"
--output=jsonpath={.items..metadata.name}) kubectl describe pod $pod
```

Readiness OK

Our second Pod, *frontend*, returns an OK status on launch.

```
kubectl get pods --selector="name=frontend"
```

Step 3 - Liveness Probe

With our second Pod currently running in a health state, we can simulate a failure occurring.

At present, no crashes should have occurred. `kubectl get pods --selector="name=frontend"`

Crash Service

The HTTP server has an additional endpoint that will cause it to return 500 errors. Using *kubectl exec* it's possible to call the endpoint.

```
pod=$(kubectl get pods --selector="name=frontend"
--output=jsonpath={.items..metadata.name}) kubectl exec $pod -- /usr/bin/curl
-s localhost/unhealthy
```

Liveness

Based on the configuration, Kubernetes will execute the Liveness Probe. If the Probe fails, Kubernetes will destroy and re-create the failed container. Execute the above command to crash the service and watch Kubernetes recover it automatically.

```
kubectl get pods --selector="name=frontend"
```

The check may take a few moments to detect.