Application Health Probes

Objectives

- Describe how Kubernetes uses health probes during deployment, scaling, and failover of applications.

Kubernetes Probes

Health probes are an important part of maintaining a robust cluster. Health probes enable the cluster to determine the status of an application by repeatedly probing it for a response.

A set of health probes affect a cluster's ability to do the following tasks:

- Mitigate crashes by automatically attempting to restart failing pods.
- Manage failover and load balancing by sending requests only to healthy pods.
- Monitor pods by determining whether and when they are failing.
- Scale applications by determining when a new replica is ready to receive requests.

Authoring Probe Endpoints

Application developers code health probe endpoints during application development. These endpoints determine the health and status of the application. For example, a data-driven application might report a successful health probe only if it can connect to the database.

Health probe endpoints must perform quickly because the cluster calls them often. Endpoints must not perform complicated database queries or many network calls.

Probe Types

Kubernetes provides the following types of probes: startup, readiness, and liveness. Depending on the application, you might configure one or more of these types.

Startup Probes

A *startup probe* determines when an application's startup process is complete. Unlike a liveness probe, a startup probe is not called after the probe succeeds. If the startup probe does not succeed after a configurable timeout, then the pod restarts based on its `restartPolicy` value.

Add a startup probe to applications that have a long start time. A startup probe enables the liveness probe to remain short and responsive.

Readiness Probes

A *readiness probe* determines whether the application is ready to serve requests. If the readiness probe fails, then Kubernetes removes the pod's IP address from the service resource to prevent client traffic from reaching the application.

Readiness probes help to detect temporary issues that might affect your applications. For example, the application might be temporarily unavailable when it starts, because it must establish initial network connections, load files in a cache, or perform initial tasks that take time to complete. The application might occasionally need to run long batch jobs, which make it temporarily unavailable to clients.

Kubernetes continues to run the probe even after the application fails a check. If the probe succeeds again, then Kubernetes adds back the pod's IP address to the service resource, and requests are sent to the pod again.

In such cases, the readiness probe addresses a temporary issue and improves application availability.

Liveness Probes

Like a readiness probe, a *liveness probe* is called throughout the lifetime of the application. Liveness probes determine whether the application container is in a healthy state. If an application fails its liveness probe enough times, then the cluster restarts the pod according to its restart policy.

Unlike a startup probe, liveness probes run after the application's initial startup process is complete. Usually, this mitigation is by restarting or re-creating the pod.

Types of Tests

When defining a probe, you must specify one of the following types of test to perform:

HTTP GET
Each time that the probe runs, the cluster sends an HTTP GET request to the specified HTTP endpoint. The test is considered a success if the request responds with an HTTP response code between `200` and `399`. Other responses cause the test to fail.

Container command
Each time that the probe runs, the cluster runs the specified command in the container. If the command exits with a status code of `0`, then the test succeeds. Other status codes cause the test to fail.

TCP socket
Each time that the probe runs, the cluster attempts to open a socket to the container on the specified port. The test succeeds only if the connection is established.

Timings and Thresholds

All the types of probes include timing variables. The `period seconds` variable defines how often, in seconds, the probe runs. The `failure threshold` defines how many failed attempts are required before the probe itself fails.

For example, a probe with a failure threshold of 3 and period seconds of 5 can fail up to three times before the overall probe fails. With this configuration, an issue can persist for at least 10 seconds before mitigation begins. Running probes too often wastes cluster resources. Balance these values when you configure probes.

Adding Probes by Using a YAML File

Because probes are defined on a pod template, you can add probes to workload resources such as deployments. To add a probe to an existing deployment, update and apply its YAML file or use the `oc edit` command.

The following YAML excerpt defines a deployment pod template with a liveness probe:

```
apiVersion: apps/v1
kind: Deployment
...output omitted...
spec:
...output omitted...
  template:
    spec:
      containers:
      - name: web-server
        ...output omitted...
        livenessProbe: (1)
          failureThreshold: 6 (2)
          periodSeconds: 10 (3)
          httpGet: (4)
            path: /health (5)
            port: 3000 (6)
```

(1) Defines a liveness probe.

(2) Specifies how many times the probe must fail before mitigating.

(3) Defines how often the probe runs.

(4) Sets the probe as an HTTP request and defines the request port and path.

(5) Specifies the HTTP path to send the request to.

(6) Specifies the port to send the HTTP request over.

Adding Probes via the CLI

Use the `oc set probe` command to add or modify a probe on a resource. The following command adds a readiness probe to a deployment named `front-end`:

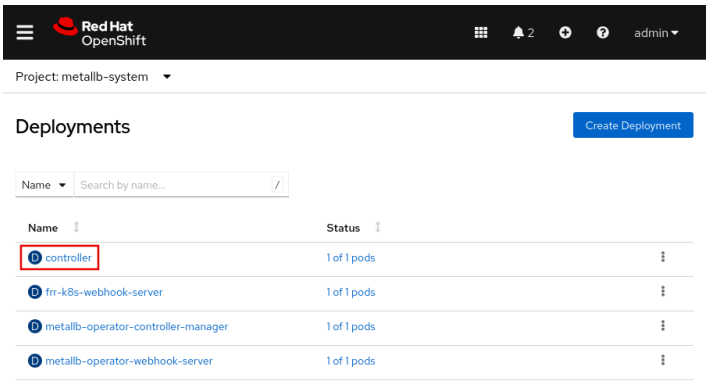
```
[user@host ~]$ oc set probe deployment/front-end \
--readiness \ 1
--failure-threshold 6 \ 2
--period-seconds 10 \ 3
--get-url http://:8080/healthz 4
```

1. Defines a readiness probe.
2. Sets how many times the probe must fail before mitigating.
3. Sets how often the probe runs.
4. Sets the probe as an HTTP request, and defines the request port and path.

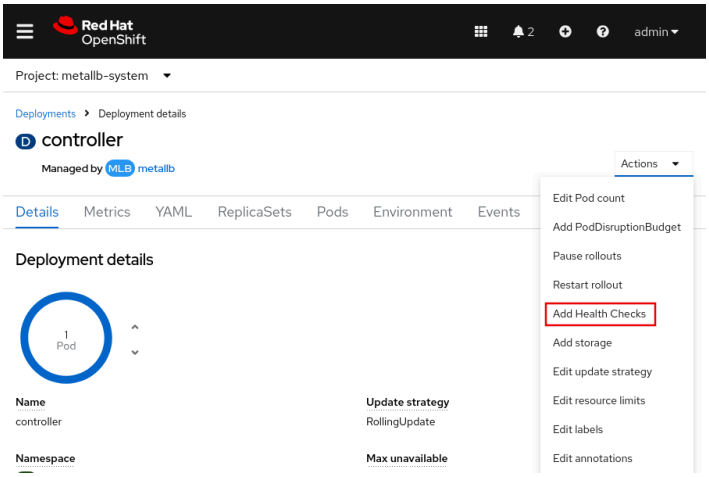
> **NOTE**
>
> The `set probe` command is exclusive to RHOCP and `oc`.

Adding Probes via the Web Console

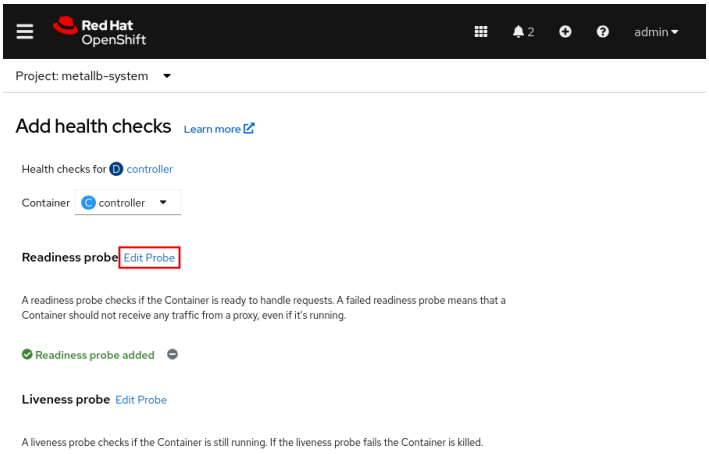
To add or modify a probe on a deployment from the web console, go to the **Workloads** → **Deployments** menu and select a deployment.



Click **Actions** and click **Add Health Checks**.



Click **Edit probe** to specify the readiness type, the HTTP headers, the path, the port, and more.



REFERENCES

[Configure Liveness, Readiness, and Startup Probes]

For more information about health probes, refer to the *Monitoring Application Health by Using Health Checks* chapter in the Red Hat OpenShift Container Platform 4.18 *Building Applications* documentation at https://docs.redhat.com/en/documentation/openshift_container_platform/4.18/html-single/building_applications/index#application-health