Chapter 6. Configure Applications for Reliability

Application High Availability with Kubernetes
Guided Exercise: Application High Availability with Kubernetes
Application Health Probes
Guided Exercise: Application Health Probes
Reserving Compute Capacity for Applications
Guided Exercise: Reserving Compute Capacity for Applications
Limit Compute Capacity for Applications
Guided Exercise: Limit Compute Capacity for Applications
Application Autoscaling
Guided Exercise: Application Autoscaling
Lab: Configure an Application for Reliability
Quiz: Configure an Application for Reliability
Summary

Abstract

Goal	Configure applications to work with Kubernetes for high availability and resilience.
Sections	• Application High Availability with Kubernetes (and Guided Exercise) • Application Health Probes (and Guided Exercise) • Reserving Compute Capacity for Applications (and Guided Exercise) • Limit Compute Capacity for Applications (and Guided Exercise) • Application Autoscaling (and Guided Exercise)
Lab	• Configure an Application for Reliability

Application High Availability with Kubernetes

Objectives

- Describe how Kubernetes works to keep applications running after failures.

Concepts of Deploying Highly Available Applications

High availability (HA) is a goal of making applications more robust and resistant to runtime failures. Implementing HA techniques decreases the likelihood that an application is completely unavailable to users.

In general, HA can protect an application from failures in the following contexts:

- From itself in the form of application bugs
- From its environment, such as networking issues
- From other applications that exhaust cluster resources

Additionally, HA practices can protect the cluster from applications, such as one with a memory leak.

Writing Reliable Applications

At its core, cluster-level HA tooling mitigates worst-case scenarios. Although HA is not a substitute for fixing application-level issues, it augments developer mitigations. Applications must work with the cluster so that Red Hat OpenShift Container Platform can best handle failure scenarios.

Red Hat OpenShift Container Platform expects the following behaviors from applications:

- Tolerates restarts
- Responds to health probes, such as the startup, readiness, and liveness probes
- Supports multiple simultaneous instances
- Has well-defined and well-behaved resource usage
- Operates with restricted privileges

Although the cluster can run applications that lack the preceding behaviors, applications with these behaviors better use the reliability and HA features that Kubernetes provides.

Most HTTP-based applications provide an endpoint to verify application health. The cluster can be configured to observe this endpoint and mitigate potential issues for the application.

The application is responsible for providing such an endpoint. Developers must decide how the application determines its state.

For example, if an application depends on a database connection, then the application might respond with a healthy status only when the database is reachable. However, not all applications that make database connections need such a check. This decision is at the discretion of the developers.

Kubernetes Application Reliability

If an application pod crashes, then it cannot respond to requests. Depending on the configuration, the cluster can automatically restart the pod. If the application fails without crashing the pod, then the pod continues to run but does not receive requests. The cluster can detect and act on a running but unhealthy pod only with the appropriate health probes.

Kubernetes uses the following HA techniques to improve application reliability:

- Restarting pods: By configuring a restart policy on a pod, the cluster restarts misbehaving instances of an application.
- Probing: By using health probes, the cluster detects when applications cannot respond to requests, and can automatically act to mitigate the issue.
- Horizontal scaling: When the application load changes, the cluster can scale the number of replicas to match the load.

These techniques are explored throughout this chapter.

Restarting Pods

A container in a pod can terminate for various reasons. The `restartPolicy` field of a pod specification controls whether the kubelet restarts the containers in that pod. The possible values are `Always`, `OnFailure`, and `Never`. The default value is `Always`.

Policy	**Description**
`Always`	The kubelet always restarts a terminated container. This policy is the default.
`OnFailure`	The kubelet restarts a terminated container only if it exited with a non-zero exit code.
`Never`	The kubelet never restarts a terminated container.

NOTE

For application workloads, you typically use higher-level controllers such as `Deployment` or `StatefulSet` to manage pods. These controllers ensure that a specified number of replicas are always running. If a pod that is managed by a `Deployment` fails, then the controller automatically creates another pod to replace it, which is a core HA mechanism. These controllers require the pod template's `restartPolicy` field to be set to `Always`.