Implementing Health Checks in Microservices

Steve Gordon

MICROSOFT DEVELOPER TECHNOLOGIES MVP

@stevejgordon   www.stevejgordon.co.uk

Overview

- Learn about ASP.NET Core health checks

- Discuss importance of health checks for distributed, microservice-based apps

- Add a liveness health check

- Add a database readiness health check

- Create a shared project for code reuse

- Customize health check responses

- Consider API dependency health checks

- Create a custom health check
  - Readiness check for Azure Service Bus

Introducing Health Checks

Health Checks

Monitor the health of web applications
- Expose an HTTP endpoint

One or more checks may be executed

Health check results indicate whether a service can be successfully consumed

Often used by container orchestrators
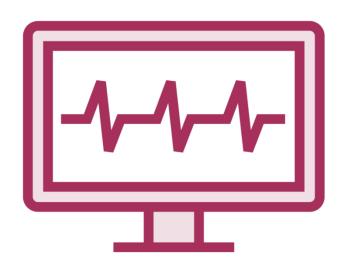
May be used for uptime monitoring

ASP.NET Core includes a built-in health check library

Liveness Health Checks

A healthy service can successfully respond to HTTP requests

Failures may indicate a critical issue or hung application

- Often results in a restart of a service

Kubernetes uses liveness endpoints to determine the health of pods

AWS ECS application load balancers expect a liveness endpoint

Liveness checks:

Liveness checks:

- Often sufficient for web applications

Liveness checks:

- Often sufficient for web applications

- Prefer basic liveness checks
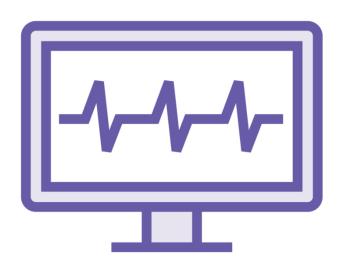
Liveness checks:

- Often sufficient for web applications

- Prefer basic liveness checks

- Web applications which can return an HTTP response are fundamentally live

Readiness Health Checks

Test more than simple HTTP liveness
- Availability of critical dependencies
- Completion of start up work
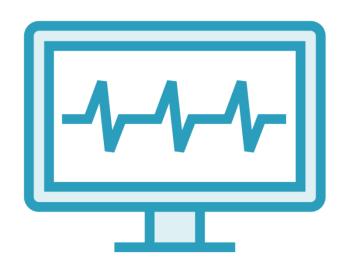
May take longer to return as healthy
- Unhealthy readiness checks do not usually trigger a restart
- Traffic will not be sent until healthy

Readiness checks are performed periodically during the life of a service

Startup Health Checks

Kubernetes includes a special probe executed only at startup

Similar to readiness health check

Once healthy, no longer checked while the service is running

Must be healthy before traffic is sent

- Long running work may need to complete at start up
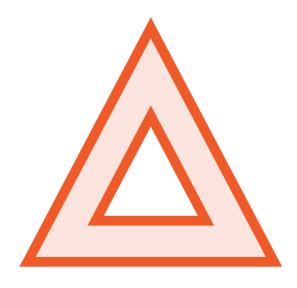
Health Check Statuses

Healthy
Indicates the application is operating correctly

Degraded
Indicates the service is live, but some functionality may be unavailable

Unhealthy
Indicates that the application may be unable to operate or is not ready

Health Checks and Microservices
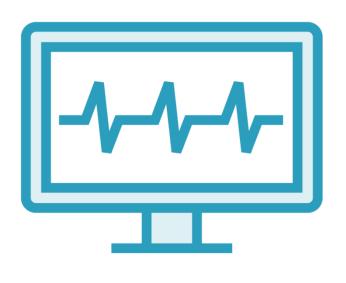
Challenge

Monitoring and maintaining overall health at scale can be quite complicated.

Health Checks and Microservices

Microservices scale the challenge of monitoring and maintaining health

Health checks support monitoring overall application status across microservices

Microservices are often deployed and run using container orchestrators

- Used to restart faulty instances
- Validate health of new deployments and help route traffic

Microservices have unique dependencies

Consider

The types of health check you need to monitor
and maintain overall application health.

Health checks are a key consideration when building microservices.

Demo

Add a liveness health check endpoint

View the health check response

Unhealthy instances may be restarted by container orchestrators.

Demo

Add a health check for the DbContext

Add a readiness health check endpoint

DbContext Health Check

Resolves an instance of DbContext from DI

Attempts to call CanConnectAsync

By default, failures are considered critical and use the unhealthy status

Health check result status can be changed to degraded when fallback options exist

Demo
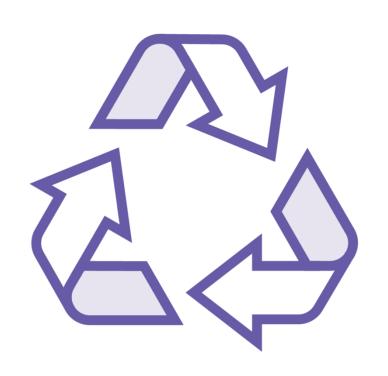
Add a shared project to the solution

Create an endpoint mapping extension method

Shared Projects

Allow sharing of code, assets and resources across multiple project types

Use is limited to specific project types, including console applications

Do not produce compiled output

- Compiled into referencing projects

Do not require NuGet feed to distribute

Project needs to be accessible during builds of dependent services

Demo

Add a custom response writer to return JSON formatted responses

Configure health check endpoints

Demo

Add health checks to the web application

Add a health check of a downstream API
- Consider pros and cons

Liveness and readiness health checks only check that the service can handle HTTP requests.

Health Checking API Dependencies

Take care to avoid cascading failures

Web Application

Healthy

Event Catalog

Healthy

Web Application

Healthy

Event Catalog

Unhealthy

Web Application
Healthy
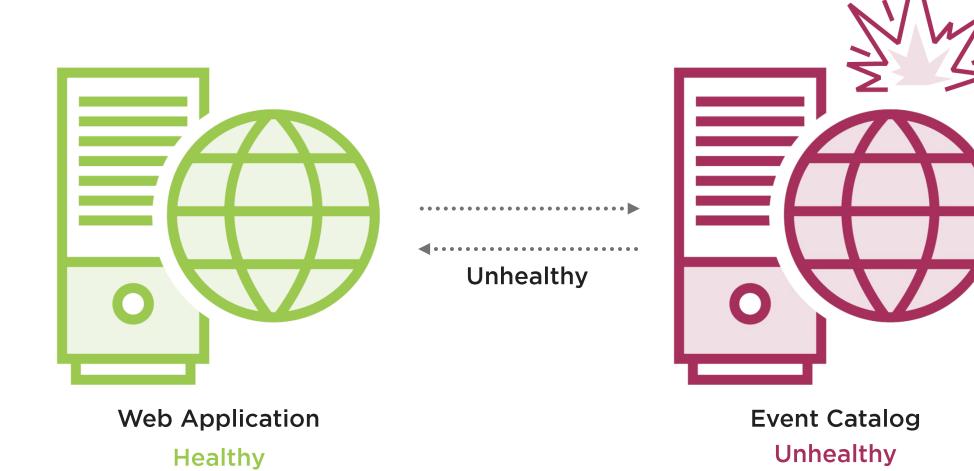
Unhealthy

Event Catalog
Unhealthy
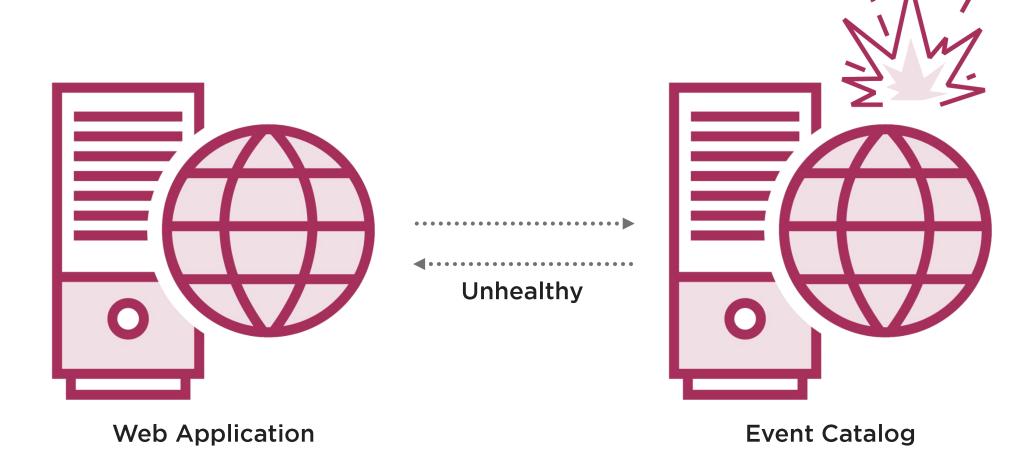
Web Application
Unhealthy

Unhealthy

Event Catalog
Unhealthy

Health Checking API Dependencies

Take care to avoid cascading failures

Use appropriate health check statuses
- Use degraded rather than unhealthy

Report via readiness health checks

Consider load on downstream APIs

Consider if the health check adds value
- Resiliency techniques should be used

Consider using metrics or logs to alert on downstream failures

Let the container orchestrator handle failing health checks.

Demo

Create a custom Azure Service Bus health check

Apply the health check in the Orders API

Summary

Learned about ASP.NET Core health checks

Applied health checks to microservices
- Added a liveness health check
- Added a database readiness health check

Created a shared project for code reuse

Customized health check responses

Added an API dependency health check
- Discussed the tradeoffs

Created a custom Azure Service Bus readiness health check

Continuation Exercise

Why not continue to add logging and health checks to the remaining microservices in the sample application?

Steve Gordon

MICROSOFT DEVELOPER TECHNOLOGIES MVP

@stevejgordon   www.stevejgordon.co.uk