Elaboro	Documento	Versión	Descripción	Fecha
Luis Sánchez Martínez	Patrón Circuit Breaker.	1.0	Creación de documento	05/04/2022

Circuit Breaker

Introducción

La mayoría de las personas que han desarrollado microservicios conocen las ventajas que la arquitectura de microservicios ofrece en comparación con una arquitectura monolito, permitiendo simplificar el desarrollo conforme las aplicaciones crecen y se hacen más complejas.

La arquitectura de microservicios ha llegado con sus propios retos, uno de estos problemas es la falla en cascada, en la cual uno de los microservicios puede tener un gran impacto sobre otros que dependan de este y causar una falla en todo el sistema.

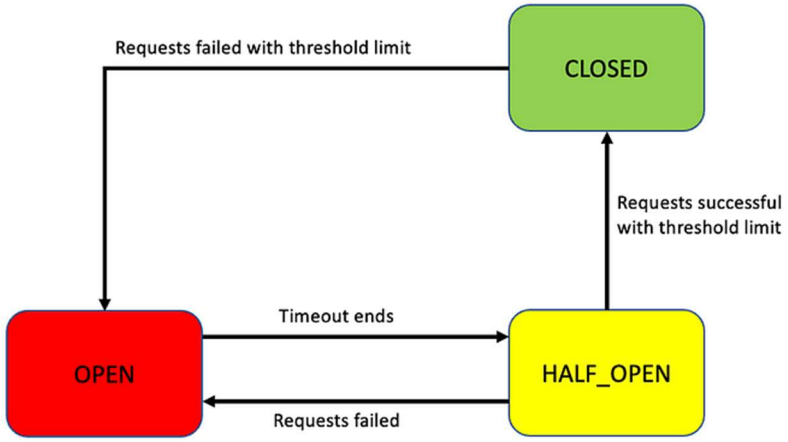
Por lo tanto, uno de los patrones para prevenir la falla en cascada es el patrón Circuit Breaker.

Circuit Breaker

La idea del patrón Circuit Breaker es prevenir la llamada a microservicios con una falla en su funcionamiento, servicios que estén tirados o que tarden en responder. Esto puede permitir a que de lado del cliente eviten el manejo de solicitudes con fallas y darle al servicio un tiempo para recuperarse.

Por ejemplo, considerar el siguiente escenario:

1. El servicio permiso-service invoca al servicio tipousuario-service, pero el servicio tipousuario-service no responde o tarda en responder.
2. Las opciones que tiene el servicio permiso-service son esperar por la respuesta del servicio tipousuario-service o manejar la excepción encontrada.
3. Las peticiones al servicio permiso-service pueden llevar a una mala experiencia por parte del cliente.
4. El uso del patrón Circuit Breaker podría ayudar evitando que las peticiones al servicio tipousuario-service lleguen por un periodo de tiempo, esperando que la lentitud del servicio desaparezca y habilitando un pequeño numero de peticiones para validar si el servicio ya se reestableció para continuar con la operación normal. De lo contrario otra vez comenzara un periodo de tiempo de espera.



El patron Circuit Breaker tiene tres estados: closed, open y half open.

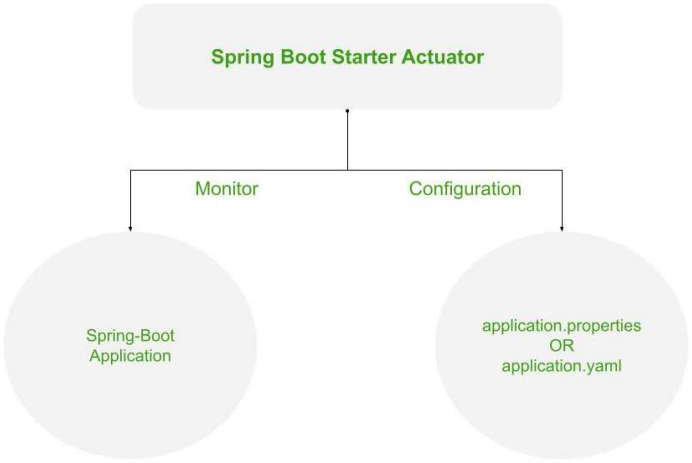
1. Closed: es el estado inicial de un circuit breaker. Cuando un microservicio corre e interactúa suave, el circuito esta cerrado. Este se mantiene continuamente monitoreando el numero de fallos que ocurren dentro del periodo de tiempo configurado. Si la tasa de fallos excede el limite especificado, su estado podría cambiar a abierto, sino este podría reiniciar el contador de fallos y el perido de tiempo de espera.
2. Open: durante el estado abierto, el circuito podrá bloquear el flujo de comunicación entre los microservicios. Las peticiones realizadas podrían fallar y las excepciones podrían ser lanzadas. El estado abierto se mantiene hasta que el tiempo de espera termina, entonces cambiará a un estado medio-abierto.
3. Half open: en el estado medio abierto, el circuito permite un numero limitado de peticiones que dejara pasar. Si la tasa de fallos es más grande que el límite permitido, cambia de nuevo al estado abierto, de o contrario cambia al estado cerrado.

Spring Boot Actuator

Los dos aspectos más importantes del ciclo de vida de una aplicación son: desarrollar y gestionar una aplicación. Es importante saber que pasa internamente en la aplicación. Cuando se lleva al ambiente productivo la gestión de la aplicación llega a ser un tema crítico. Por lo tanto, se recomienda que la aplicación sea monitoreada mientras esta siendo desarrollada y cunado se despliega en un ambiente productivo.

Spring Boot proporciona en la dependencia de actuator que puede ser usada para monitorear y gestionar la aplicación. Para tal motivo se pueden usar las rutas:

- /actuator
- /actuator/health



Ventajas de usar actuator:

- Reduce tiempos de inactividad.
- Aumenta la productividad.
- Mejora la gestión de la aplicación.
- Mejora la eficiencia de la aplicación.

Referencias

https://www.geeksforgeeks.org/spring-boot-actuator/?ref=lbp

https://medium.com/@truongbui95/circuit-breaker-pattern-in-spring-boot-d2d258b75042