Universidad de Guadalajara.

Computación Tolerante a fallas.

Diego Ivan Becerra Gonzalez.

Sección D06.

Objetivo.

Responder a las preguntas y generar un ejemplo utilizando MicroProfile, Quarkus y Docker.

Desarrollo.

Historia.

El lenguaje de programación Java fue creado por James Gosling y su equipo en Sun Microsystems en la década de 1990. Inicialmente Java era conocido como Oak o Green.

La primera versión del lenguaje Java es publicada por Sun Microsystems en 1995. Y es en la versión del lenguaje JDK 1.0.2, cuando pasa a llamarse Java, corría el año 1996.

En las primeras versiones de Java 1.1, 1.2 y 1.3 es en la que el lenguaje va tomando forma, con la inclusión de tecnologías como JavaBeans, JDBC para el acceso a base de datos, RMI para las invocaciones en remoto, Collections para la gestión de múltiples estructuras de datos o AWT para el desarrollo gráfico, entre otros.

Originalmente, fue concebido como un lenguaje de programación para dispositivos electrónicos, pero rápidamente se convirtió en una herramienta popular para el desarrollo de aplicaciones empresariales.

Java se ha dividido en dos ediciones principales: Java SE (Standard Edition) y Java EE (Enterprise Edition). Java SE es la versión principal de Java y proporciona un conjunto básico de características y funcionalidades para el desarrollo de aplicaciones. Java EE, por otro lado, es una plataforma empresarial más completa que ofrece un conjunto de características adicionales, como la integración con servidores de aplicaciones y frameworks de desarrollo web.

En 2005, Sun Microsystems abrió el código fuente de Java bajo la licencia de software libre GPL. En 2010, Oracle Corporation adquirió Sun Microsystems y asumió el control del desarrollo y mantenimiento de Java.

En 2017, Oracle transfirió el control de la plataforma Java EE a la Fundación Eclipse, que la renombró como Jakarta EE. Jakarta EE continúa siendo una plataforma empresarial completa que proporciona un conjunto de características adicionales para el desarrollo de aplicaciones empresariales.

Además de Java EE, existen otras tecnologías empresariales basadas en Java, como MicroProfile. MicroProfile es una especificación de plataforma de microservicios que proporciona un conjunto de características y funcionalidades adicionales para el desarrollo de aplicaciones basadas en microservicios.

También hay varios frameworks de desarrollo basados en Java, como Spring Boot y Quarkus. Spring Boot es un framework de desarrollo de aplicaciones web que proporciona un conjunto completo de características y funcionalidades para el desarrollo de aplicaciones web empresariales. Quarkus, por otro lado, es un framework de desarrollo de aplicaciones Java diseñado para ejecutarse en entornos de contenedores y proporciona un rendimiento excepcionalmente rápido.

Por último, Gradle es una herramienta de construcción de software de código abierto que se utiliza para automatizar el proceso de compilación y construcción de aplicaciones Java y otros lenguajes de programación. Gradle es compatible con Java y con otros frameworks y herramientas de desarrollo de Java, como Spring Boot y Quarkus.

¿Qué es Java EE?

Según la definición de Sun, Java Enterprise Edition (Java EE) es el estándar de la industria para desarrollar aplicaciones Java portables, robustas, escalables y seguras en el lado del servidor (server-side). Basado en Java SE, proporciona APIs para servicios web, modelo de componentes, gestión y comunicación.

¿Qué es Java SE?

Java SE o Java Standar Edition (antes conocida como Plataforma Java 2 o J2SE), constituye la base del lenguaje de programación Java sobre el que se desarrollan Java EE y Java ME. Por ejemplo, Java Enterprise Edition incluye todas las clases en Java SE, más otras específicas.

Java JSE está orientado a desarrollar aplicaciones bajo la arquitectura cliente / servidor, sin que tenga soporte a tecnologías para internet.

Con Java SE es posible programar y ejecutar aplicaciones de escritorio y applets.

¿Qué es Jakarta EE?

En 2017 Oracle decide que deja de controlar el desarrollo de Java EE y decide pasarlo a la comunidad para que sea guiado en un proceso más abierto y flexible por la Eclipse Foundation. Manteniendo el modelo de JCP que se seguía en su desarrollo.

Si bien en este proceso de traspaso se decide no pasar la marca Java EE y la comunidad tiene que renombrar el proyecto a Jakarta EE.

Así que Jakarta EE pasa a ser la nueva plataforma opensource de Java EE gestionada por Eclipse Foundation.}

¿Qué es MicroProfile?

MicroProfile es una especificación de plataforma de microservicios que proporciona un conjunto de características y funcionalidades adicionales para el desarrollo de aplicaciones basadas en microservicios.

¿Qué es Spring boot?

Spring Boot es un framework desarrollado para el trabajo con Java como lenguaje de programación. Se trata de un entorno de desarrollo de código abierto y gratuito.

¿Qué es Quarkus?

Quarkus fue creado para permitir a los desarrolladores de Java crear aplicaciones para un mundo moderno y nativo de la nube. Quarkus es un marco Java nativo de Kubernetes adaptado a GraalVM y HotSpot, elaborado a partir de las mejores bibliotecas y estándares Java. El objetivo es convertir a Java en la plataforma líder en Kubernetes y entornos sin servidor, al tiempo que ofrece a los desarrolladores un marco para abordar una gama más amplia de arquitecturas de aplicaciones distribuidas.

¿Qué es Maven?

Maven se utiliza en la gestión y construcción de software. Posee la capacidad de realizar ciertas tareas claramente definidas, como la compilación del código y su empaquetado. Es decir, hace posible la creación de software con dependencias incluidas dentro de la estructura del JAR. Es necesario definir todas las dependencias del proyecto (librerías externas utilizadas) en un fichero propio de todo proyecto Maven, el POM (Project Object Model). Este es un archivo en formato XML que contiene todo lo necesario para que a la hora de generar el fichero ejecutable de nuestra aplicación este contenga todo lo que necesita para su ejecución en su interior.

Sin embargo, la característica más importante de Maven es su capacidad de trabajar en red. Cuando definimos las dependencias de Maven, este sistema se encargará de ubicar las librerías que deseamos utilizar en Maven Central, el cual es un repositorio que contiene cientos de librerías constantemente actualizadas por sus creadores. Maven permite incluso buscar versiones más recientes o más antiguas de un código dado y agregarlas a nuestro proyecto. Todo se hará de forma automática sin que el usuario tenga que hacer nada más que definir las dependencias.

¿Qué es Gradle?

Gradle es una herramienta de automatización de compilación del código abierto, que obtuvo una rápida popularidad ya que fue diseñada fundamentalmente para construir multiproyectos, utilizando conceptos provenientes de Apache Maven.

Gradle mejora principalmente las siguientes funcionalidades de Maven:

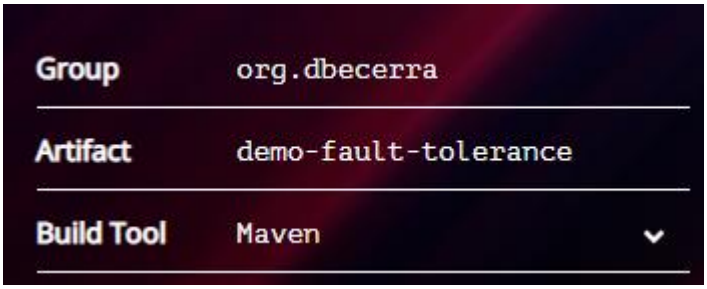
- Lenguaje: Gradle no emplea el lenguaje XML, sino que se basa en DSL ya que se focaliza en la resolución de un problema específico, colaborando en construcciones sumamente estructuradas, eficientes y mantenibles para múltiples proyectos.

- Gestión del ciclo de vida: Añade la capacidad de soportar todo el proceso de vida del software (desde la compilación, pruebas, análisis estadístico e implementación).

- Personalización: Para personalizar una tarea simplemente se añade al fichero groovy (build.gradle).

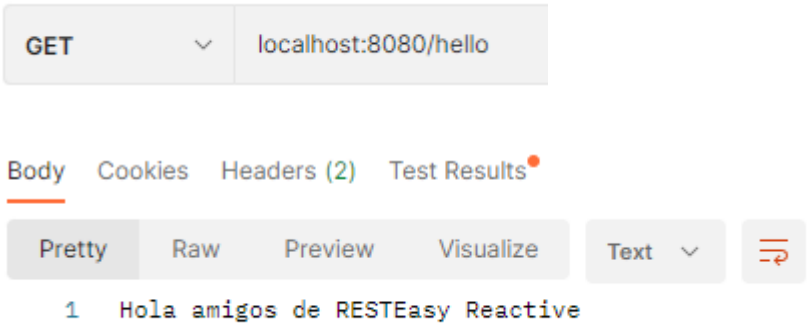
Ejemplo.

Lo primero que haces es crear nuestro proyecto desde la página de quarkus, ahí lo configuramos y elegimos las librerías con las que trabajaremos.



Después la descargamos en un archivo zip y la descomprimimos.

Abrimos la carpeta con Intelij Idea. Ahí usamos el comando ““mvn compile quarkus:dev” para iniciar la simulación de nuestra API. Para comprobar que funcione usamos postman y nos muestra lo siguiente:



Ahora proseguimos a la parte de tolerancia a fallas, para eso creamos una clase llamada Person y su controlador. La clase tendrá los atributos de nombre, id y correo, y el controlador lo que tendrá serán los métodos que manejarán el funcionamiento de la clase, incluyendo los métodos para tolerar las fallas.

Los métodos del controlador son los siguientes:

```
public List<Person> getPersonList(){
    LOGGER.info( msg: "Ejecutando person list");
    doFail();
    //doWait();
    return this.personList;
}
```

Nos retorna la información de un objeto Person.

```
public List<Person> getPersonFallbackList(){
    var person = new Person( personId: -1L,  name: "Diego",  email: "diego.bgonzalez@alumnos.udg.com");
    return List.of(person);
}
```

Es la función por default que nos retorna en caso de que haya una falla.

```
public void doWait(){
    var random = new Random();
    try {
        LOGGER.warning( msg: "Haciendo un sleep");
        Thread.sleep( millis: (random.nextInt( bound: 10) + 4) * 1000L);
    }catch (Exception ex){

    }
}
```

Hace que pase cierto tiempo para forzar a que la ejecución de la función se tarde y probar el método destinado a cuando el microservicio tarda.

```java
public void doFail(){
    var random = new Random();
    if(random.nextBoolean()){
        LOGGER.warning( msg: "Se produce una falla");
        throw  new RuntimeException("Haciendo que la implementacion falle");
    }
}
```

Simula el fallo del microservicio.

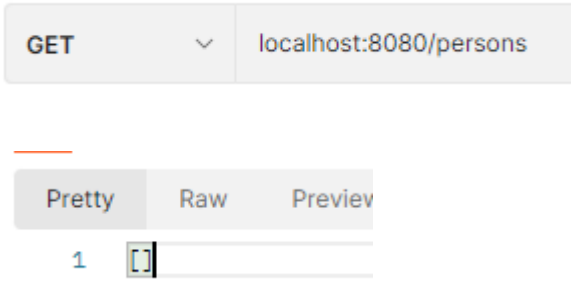
Y ahora viene la parte importante, al método principal del controlador, que es getPersonList, se le agregaron unos “atributos”, por así decirlo, que permiten controlar el comportamiento del microservicio en caso de que ocurran ciertas condiciones. Estos “atributos” o funcionalidades las provee la biblioteca de tolerancia a fallas, las cuales son:

```java
//@Timeout(value = 5000L)
//@Retry(maxRetries = 4)
//@CircuitBreaker(failureRatio = 0.1, delay = 15000L)
//@Bulkhead(value = 0)
@Fallback(fallbackMethod = "getPersonFallbackList")
public List<Person> getPersonList(){
    LOGGER.info( msg: "Ejecutando person list");
    doFail();
    //doWait();
    return this.personList;
}
```

- Timeout lo que permite es controlar la ventana de tiempo que tiene el microservicio para responder, y en caso de no hacerlo, cae en el fallback.
- Retry hace que en caso de que ocurra una falla, se vuelve a hacer la petición, esto se repite tantas veces lo indiques y si se llega a ese numero de fallas seguidas, ahora si cae en el fallback.

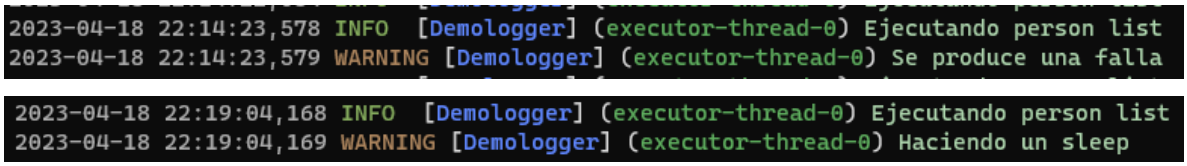
- CircuitBreaker establece que, si se cumple una condición de porcentaje de fallas por cada 10 intentos, el microservicio tarda cierto tiempo en volver a recibir peticiones.
- Bulkhead lo que establece es el número máximo de peticiones que puede manejar al mismo tiempo, si esto se supera, caen en el fallback.
- Fallback le da un camino alternativo el cual seguir en caso de que el microservicio falle.

Esta es la respuesta común del API al hacer una petición.



Sin embargo, si forzamos un error, obtenemos el siguiente mensaje definido por la función getPersonFallbackList:

Este error puede ser causado por diferentes acciones, forzar el error con una función, hacer que tarde demasiado ejecutándose o indicar que el máximo de peticiones debe ser 0. Además, podemos utilizar un logger para ver que parte del código se ejecutó.



Conclusión.

La librería de tolerancia a fallas fue muy comprensible de entender en cuanto a la manera de implementarla, creo que sus funcionalidades eran bastante intuitivas, y me pareció bien que tuviera tantas maneras de tratar los posibles errores. La verdad no probe a fondo la librería, pero supongo que tiene mas funcionalidades aparte de las mostradas en el video, por lo que creo se pueden crear reglas de manejo de errores complejas a partir de la combinación de muchas reglas simples.

Bibliografía.

Agamboa. (2022, 3 octubre). *MicroProfile overview*. Adam Gamboa G - Developer. https://blog.adamgamboa.dev/es/microprofile-overview-2/

Briceño, G. (2017, 5 octubre). *Que es Eclipse MicroProfile*. Club de Tecnología. https://www.clubdetecnologia.net/blog/2017/que-es-eclipse-microprofile/

Caules, C. Á. (2023). ¿Qué es Jakarta EE? *Arquitectura Java*. https://www.arquitecturajava.com/jakarta-ee/

Historia del lenguaje Java. (2023, 15 abril). Manual Web. https://www.manualweb.net/java/historia-java/#:~:text=El%20lenguaje%20Java%20fue%20desarrollado,Java%2C%20corr%C3%ADa%20el%20a%C3%B1o%201996.

IBM Documentation. (s. f.). https://www.ibm.com/docs/es/odm/8.5.1?topic=application-java-se-java-ee-applications

Nacho. (s. f.). *Sesión 5: Introducción a Java EE*. http://www.jtech.ua.es/ayto/ctj/restringido/apuntes/sesion05-apuntes.htm

Netec. (2019, 7 mayo). Historia de Java, un camino lleno de curiosidades *Netec*. https://www.netec.com/post/historia-y-curiosidades-de-java

¿Qué es Quarkus? (s. f.). Quarkus. https://es.quarkus.io/about/

School, T. (2022). ¿Qué es Spring Boot y para qué sirve? *Tokio School*. https://www.tokioschool.com/noticias/spring-boot/

Solis, G. D. (2019). MicroProfile, el éxito de microservicios depende de la cultura de tu Empresa. *MicroProfile*. https://microprofile.io/2019/03/28/microprofile-el-exito-de-microservicios-depende-de-la-cultura-de-tu-empresa/

Thoth, & Thoth. (2020). ¿Qué es Java JSE? *Formatalent Business School*. https://formatalent.com/que-es-java-jse/

Víctor Orozco. (2021, 8 febrero). *Tolerancia a fallas con MicroProfile, Quarkus y Docker* [Vídeo]. YouTube. https://www.youtube.com/watch?v=sTkolTRuPlE