Tolerancia a fallas con MicroProfile, Quarkus y Docker.

Investigación y ejemplo Microprofile

Gutiérrez Villalobos Dayal

Código: 216586382

Computación tolerante a fallos

Lunes y miércoles: 11:00-12:55

Sección D06

Maestro: López Franco Michel Emanuel

Docker

Docker es una tecnología de organización en contenedores siendo útil para poder crear los mismos y usarlos de una manera eficiente, haciendo posible que se manejen estos contenedores como máquinas virtuales las cuales son muy livianas y modulares. Este tipo de tecnología sirve para poder copiar entornos y trasladarlos, gracias a la flexibilidad que tienen los contenedores haciendo que se muy útil para optimizar aplicaciones para la nube. El propósito de los contenedores es ejecutar varios procesos y aplicaciones por separado para que se pueda aprovechar mejor la infraestructura y, al mismo tiempo, conservar la seguridad que se obtendría con los sistemas individuales. Como su modelo de implementación se basa en imágenes, es posible llegar a compartir toda una aplicación o servicios con todas sus dependencias en varios entornos

Microprofile

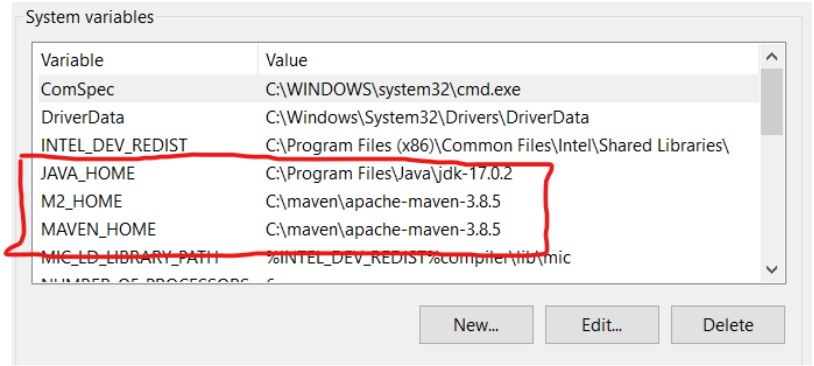
MicroProfile es una colección de APIs y tecnologías Java EE que juntas forman la línea de base para crear Microservicios, que tiene como objetivo proporcionar la portabilidad de la aplicación en múltiples entornos de ejecución

Quarkus

Quarkus es un marco de Java integral creado en Kubernetes para las compilaciones originales y las máquinas virtuales Java (JVM), el cual permite optimizar esta plataforma especialmente para los contenedores y para los entornos sin servidor, de nube y de Kubernetes. Lo interesante al menos para esta tarea es que se diseñó para que trabaje con las bibliotecas marcos y estándares de MicroProfile

Desarrollo de ejemplo

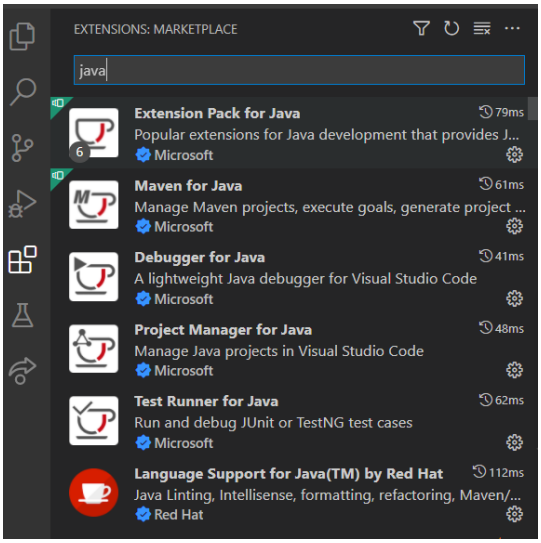
Lo primero que se tuvo que realizar es la instalación de todos los componentes necesarios para que la aplicación funcione de una manera correcta, lo primero que se tuvo que instalar fue JDK en mi caso la versión 17.0.2, para poder realizar esto fue tan sencillo como solo tener que ir a su página oficial y descargar el instalador y listo. Lo siguiente que se tuvo que poner en la máquina en donde es que se está trabajando es Maven, para poder lograr esto lo que se tuvo que hacer es ir a la página oficial de Apache Maven, descargar los binarios y tener que ponerlos en un directorio de nuestra preferencia, después agregar ciertas variables de entorno que son las que se muestran en la imagen dentro del rectángulo rojo



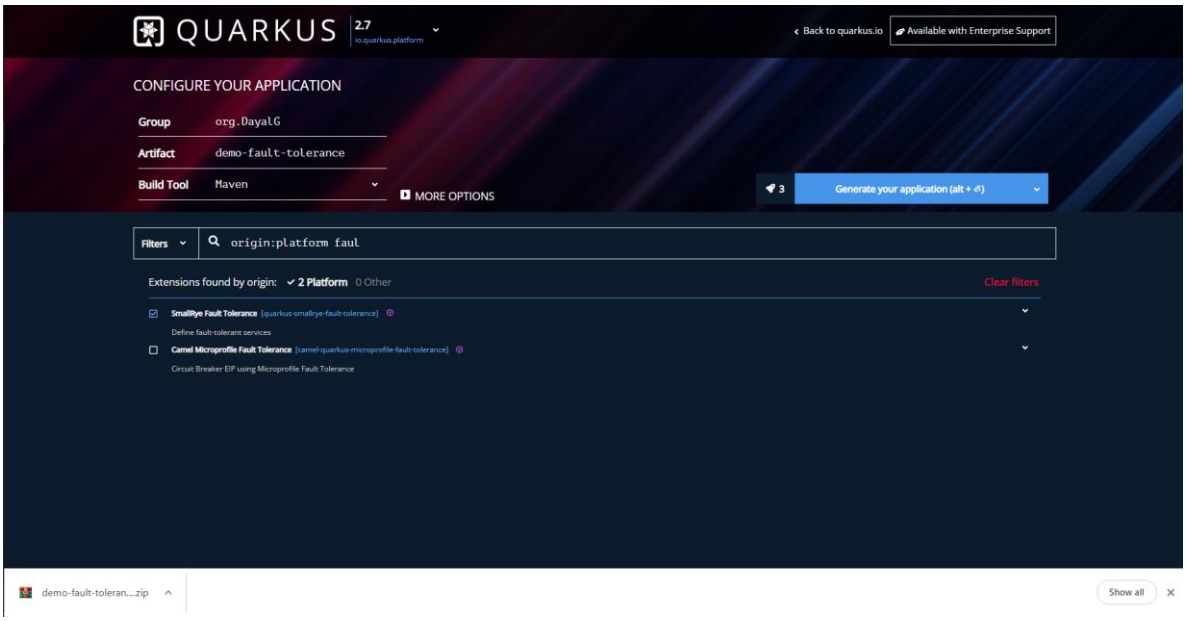
Así como después de a ver esto agregar al path de las variables de entorno lo siguiente

```
%M2_HOME%\bin
```

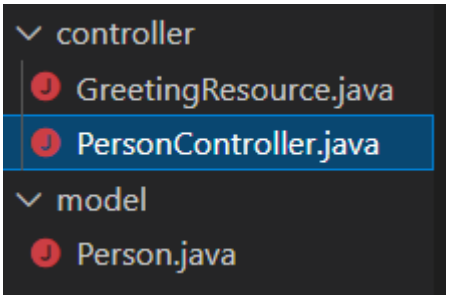
Una vez hecho esto, como decidí que mi entorno de trabajo sería VS Code simplemente descargué las extensiones correspondientes para poder trabajar en el proyecto



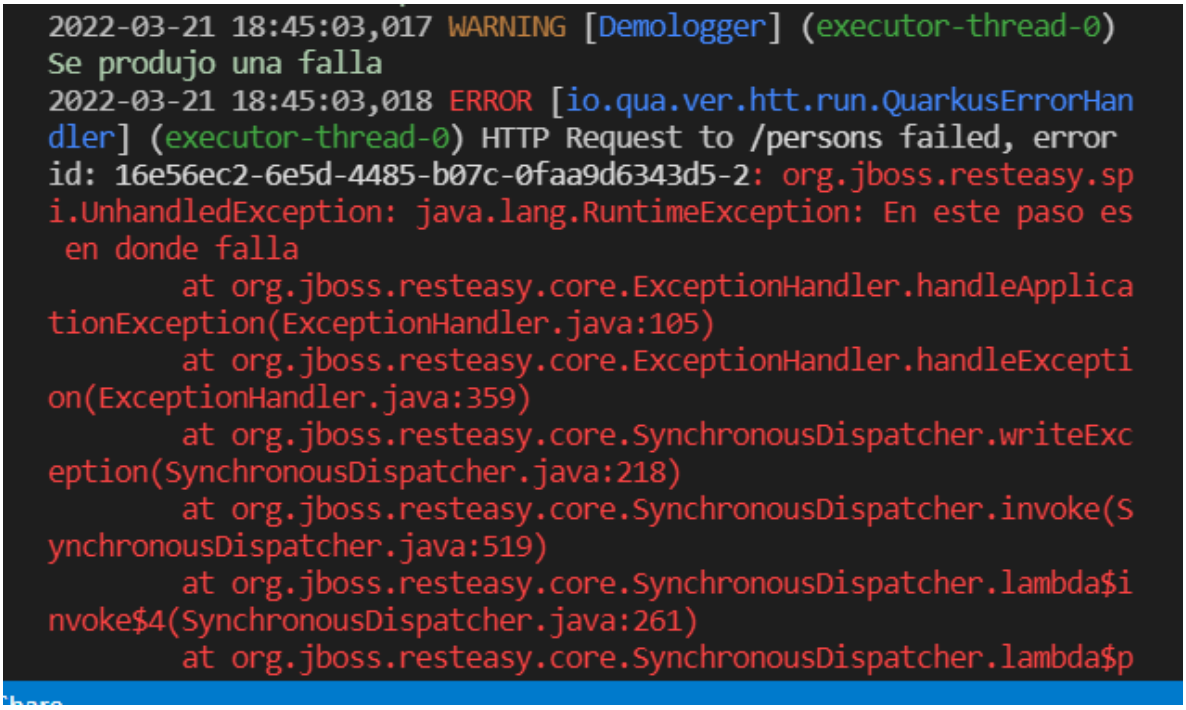
Después de tener todo lo necesario ya instalado y configurado de una manera correcta lo que se realizó fue la creación de el ejemplo en la página de Quarkus.io, para esto se estableció tanto un nombre como las extensiones adecuadas para el ejemplo y una vez que todo se tuvo seleccionado de una manera correcta se procedió a descargar nuestro proyecto como zip



Despu’es ya trabajando dentro de nuestro programa se hizo una clase persona en donde solo contiene el nombre, el Id y un correo, ya que solo es para la demostración de cómo es que fault tolarence funciona se continuo con el mismo ejemplo solo que con nuestros datos y en nuestra máquina



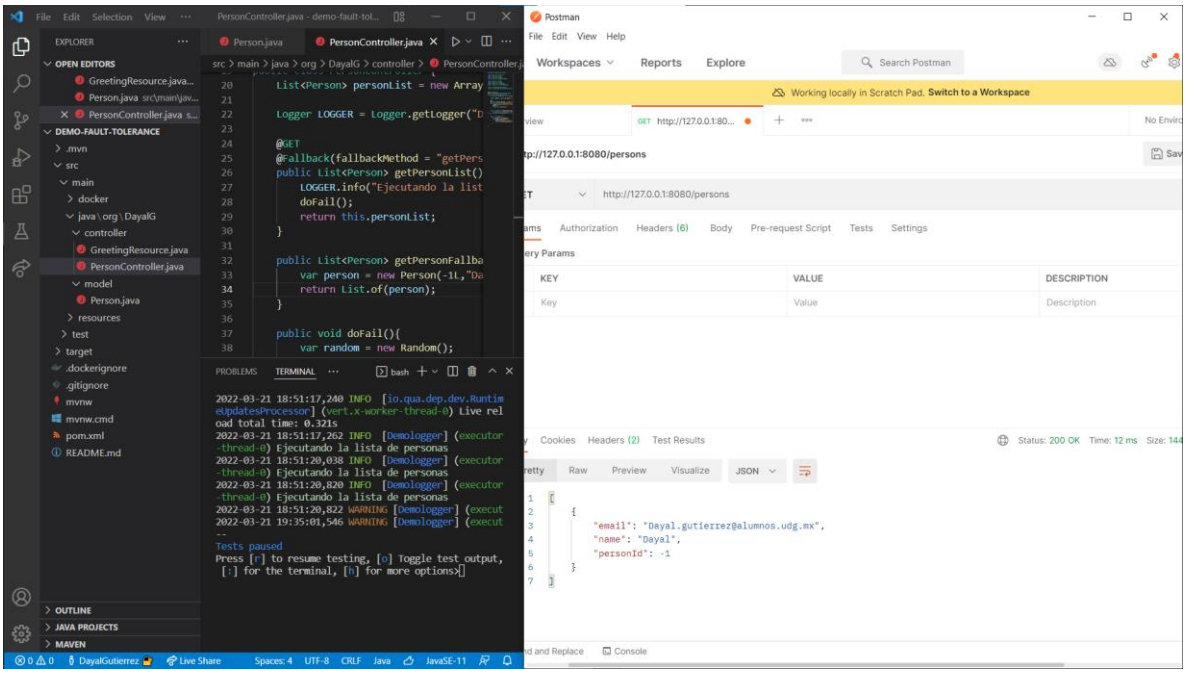
En un principio nuestra API si es que algún proceso fallaba este nos mostraba un error después del warning que nosotros habíamos puesto



Sim embargo con fault tolerance de MicroProfile podemos declarar una forma en la cual podemos hacer una segunda vía en caso de que la primera no funcione de una manera correcta. En el caso de este ejemplo la segunda vía declarada

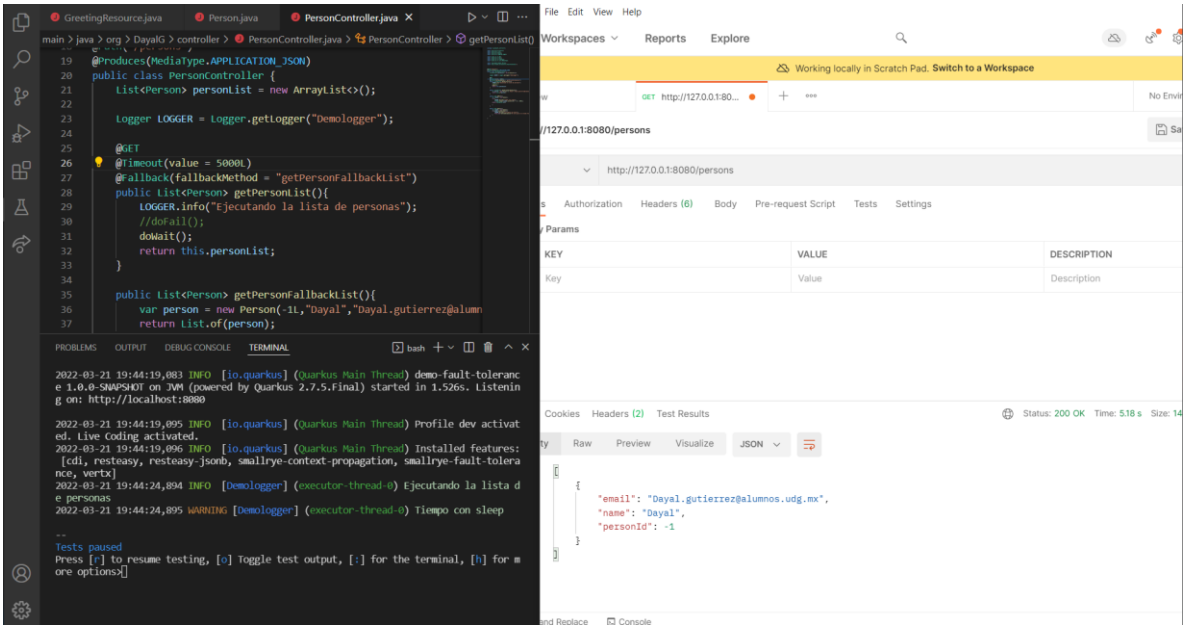
```java
@GET
@Fallback(fallbackMethod = "getPersonFallbackList")
public List<Person> getPersonList(){
    LOGGER.info("Ejecutando la lista de personas");
    doFail();
    return this.personList;
}

public List<Person> getPersonFallbackList(){
    var person = new Person(-1L,"Dayal","Dayal.gutierrez@alumnos.udg.mx");
    return List.of(person);
}
```

Es una en la cual cuando es que se la API falla por alguna razón, no tirará un error si no que seguirá las instrucciones de la vía alterna que en este caso es mostrar los datos de la persona que nosotros estamos enviando



Como se puede observar en el ejemplo cuando es que la aplicación llega a fallar nuestra aplicación muestra los datos que nosotros declaramos en el código.

De igual forma Hay más formas en las que se puede llegar a caer en situaciones en las que se tienen que llegar a tener en cuenta ciertas circunstancias como no solo es que pueda fallar si no que el servicio que se requiere tarde demasiado.



En este caso si es que se llega a taradar mucho la resuesta también se considera como un error y se puede ir por el camino alternativo

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

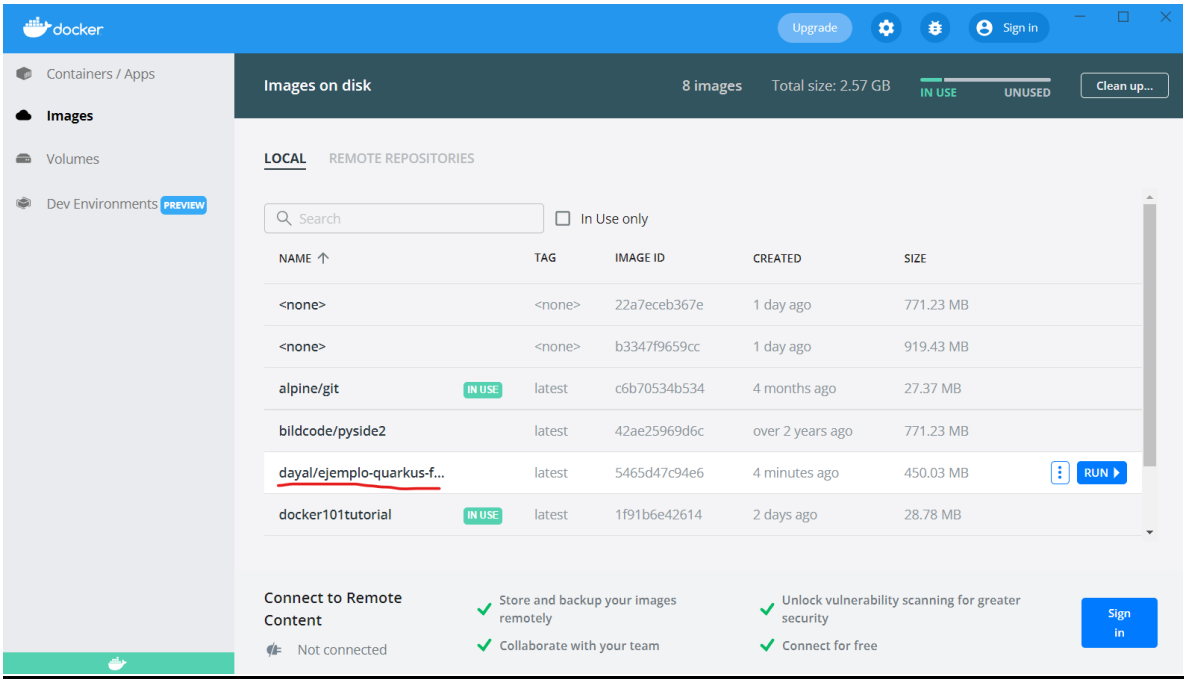
De igual manera se pueden usar diferentes técnicas con distintas naturalezas como puede ser el retry el cuál nos permite tener una cierta cantidad de fallas y sólo si es que se alcanza ése máximo de fallas se toma como falla si no simplemente lo intenta de nuevo. Por otro lado CircuitBreaker en donde se pueden establecer un circuito el cual puede permanecer abierto por tato tiempo se diga de acuerdo con el criterio que nosotros pongamos, también existe la opción de poder hacer que nuestro servicio solo pueda atender cierta cantidad de peticiones con Bulkhead, haciendo que si la cantidad de peticiones que se tienen supera a la que nosotros hemos establecido las nuevas se vayan directamente por la vía alterna

Docker

Para poder crear un Docker con este ejemplo se escribió el siguiente comando que ya lo proporciona el mismo proyecto creado por quarkus, solo que se le cambió el nombre

```
Dayal@DESKTOP-QR90KPG MINGW64 ~/Documents/Clases/Tolerante a fallas/Docker y Microprofile/demo-fault-tolerance
$ docker build -f src/main/docker/Dockerfile.jvm -t dayal/ejemplo-quarkus-fault-tolerance .
[+] Building 2.1s (4/9)
 => [internal] load build definition from Dockerfile.jvm
 => => transferring dockerfile: 5.33kB
 => [internal] load .dockerignore
 => => transferring context: 115B
 => [internal] load metadata for registry.access.redhat.com/ubi8/openjdk-11:1.11
 => [1/5] FROM registry.access.redhat.com/ubi8/openjdk-11:1.11@sha256:5e0bf98dce8ef86b77701c8cf19deee3630237b996dd99dd35bd2c1c2933c495
 => => resolve registry.access.redhat.com/ubi8/openjdk-11:1.11@sha256:5e0bf98dce8ef86b77701c8cf19deee3630237b996dd99dd35bd2c1c2933c495
 => => sha256:5e0bf98dce8ef86b77701c8cf19deee3630237b996dd99dd35bd2c1c2933c495 1.47kB / 1.47kB
 => => sha256:87571db92ec7bd68cdf3d25e2abfd844989c8b0bab49a8ec5e2fcc81df2d1d96 950B / 950B
 => => sha256:17b7fccdb815e280e7afa45fbc458b9d8bd99a7e6f6b4464f9100599e13b3d96 7.50kB / 7.50kB
 => => sha256:1d273cf0c199c6c723f6a09f3637ae3b3e124fc663e51e395f297f60b6baa05f 1.05MB / 39.62MB
 => => sha256:759b69d3a3ddf803810f5cb6b366c76f9f4991a1ef3a2b16bc7fc67c2cd78b42 1.75kB / 1.75kB
 => => sha256:6f15bb127d828f0ba29f2151f5ecb4a6fe67d915ed213f89d5580fef3000cce4 0B / 117.77MB
 => [internal] load build context
 => => transferring context: 14.92MB
```

Y como se puede observar se crea una imagen de forma correcta



Sin embargo, este tipo de cosas sobre Docker las trataré en el ejemplo de este.

Conclusiones

Es de mucho interés lo que puede realizar MicroProfile con Quarkus y el fault tolerance, me recordó un poco a prefect, sin embargo el desarrollo de esta actividad me costó un poco de entender más que nada la apliación debido a que mis conocimientos sobre java son muy pocos, pero creo que la herramienta puede ser muy útil debido a todas las características que tiene y a la amplia comunidad que tiene.

Link repositorio

https://github.com/DayalGutierrez/Ejemplo-Quarkus.git

Bibliografía

- IBM. (2021, 6 julio). What is MicroProfile? IBM Developer. Recuperado 20 de marzo de 2022, de https://developer.ibm.com/series/what-is-microprofile/

- Red hat. (2018, 9 enero). ¿Qué es Docker? Recuperado 20 de marzo de 2022, de https://www.redhat.com/es/topics/containers/what-is-docker

- Red hat. (2020, 13 enero). ¿Qué es Quarkus? Recuperado 20 de marzo de 2022, de https://www.redhat.com/es/topics/cloud-native-apps/what-is-quarkus

- Víctor Orozco. (2021, 8 febrero). Tolerancia a fallas con MicroProfile, Quarkus y Docker [Vídeo]. YouTube. https://www.youtube.com/watch?v=sTkolTRuPlE&t=2s