



Tolerancia a Fallos con MicroProfile, Quarkus y Docker

Nombre: Diana Fernanda Castillo Rebolledo

Materia: Computación Tolerante a Fallas

Horario: L-I 11:00 am-1:00 pm

Profesor: Dr. Michel Emanuel López Franco

Sección: D06

¿Qué es MicroProfile?

MicroProfile es una especificación que junta una colección de librerías de Java EE y otras tecnologías que forman una base para poder crear microservicios y resolver diferentes problemas en su arquitectura, esto con el fin de proporcionar la portabilidad de una aplicación en distintos entornos en los que se quiera ejecutar.

Como es una especificación, existen diferentes proveedores que la implementan, como Quarkus, Thorntail, TomEE, Payara Micro, Websphere Liberty, entre otros.

¿Qué es Quarkus?

Quarkus es un marco o ecosistema (estructura) de Java que utiliza las bibliotecas y los estándares principales de Java, entre ellos MicroProfile. Esto se desarrolló con el objetivo de ejecutar aplicaciones Java en contenedores. Quarkus se diseñó en torno a dar prioridad a los contenedores, por lo que se encuentra optimizado para disminuir el uso de la memoria y acelerar los tiempos de inicio, compila aplicaciones que consumen un décimo de la memoria en comparación con Java tradicional.

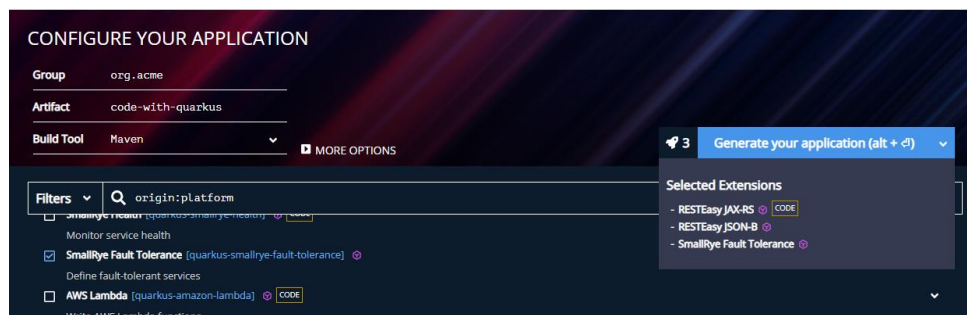
¿Qué es Docker?

Docker es una plataforma de software que permite crear y ejecutar aplicaciones de una manera rápida. Esta plataforma empaqueta software en unidades que son llamadas contenedores, estos incluyen todo lo necesario para que funcione la aplicación, como bibliotecas y otras herramientas. De esta manera se pueden implementar las aplicaciones en cualquier entorno con la certeza de que estas funcionarán al momento de su ejecución.

Dentro del contenedor se definen una serie de instrucciones (dockerfile) que permiten crear una imagen con la que poder arrancar a dicho contenedor.

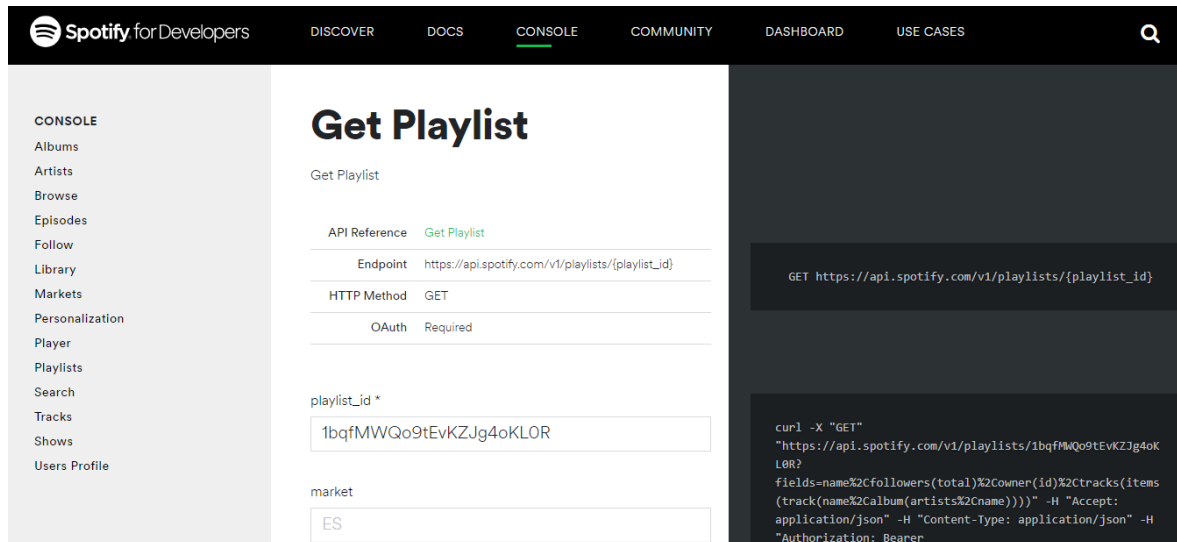
Una vez definido lo anterior, se explicará un ejemplo utilizando estas herramientas enfocándonos en la tolerancia a fallas.

Primeramente, se generó un proyecto nuevo en la página de Quarkus, en esta página se puede configurar la aplicación que se desea realizar ya que se le pueden añadir varias dependencias dependiendo del proyecto a desarrollar:



En este proyecto incluí la librería de SmallRye Fault Tolerance que es una implementación de MicroProfile que proporciona herramientas para la tolerancia a fallos en aplicaciones. Cuando se terminó de configurar el proyecto lo descargué como un zip y al descomprimirlo ya estaba listo para comenzar a trabajar en una API de REST.

Para este ejemplo, utilicé la API de Spotify para verificar los datos de una playlist:



The screenshot shows the Spotify for Developers console. On the left is a sidebar with a 'CONSOLE' menu and a list of API endpoints including Albums, Artists, Browse, Episodes, Follow, Library, Markets, Personalization, Player, Playlists, Search, Tracks, Shows, and Users Profile. The main area is titled 'Get Playlist' and contains a table with API Reference details: Endpoint (https://api.spotify.com/v1/playlists/{playlist_id}), HTTP Method (GET), and OAuth (Required). Below the table are input fields for 'playlist_id' (containing '1bqfMWQo9tEvKZJg4oKLOR') and 'market' (containing 'ES'). On the right side of the console, there are two code blocks: the first shows the REST API call 'GET https://api.spotify.com/v1/playlists/{playlist_id}' and the second shows the equivalent curl command with headers for Accept, Content-Type, and Authorization (Bearer token).

Al colocar el id de la playlist y solicitar un token de autorización en la página de Spotify, aparece al lado derecho la petición que debe realizarse para acceder a los datos de esa playlist, en este caso:

```
curl -X "GET"
"https://api.spotify.com/v1/playlists/1bqfMWQo9tEvKZJg4oKLOR?
fields=name%2Cfollowers(total)%2Cowner(id)%2Ctracks(items
(track(name%2Calbum(artists%2Cname))))" -H "Accept:
application/json" -H "Content-Type: application/json" -H
"Authorization: Bearer
BQBbfc0DPb9fzLF9hZUDUrc5eENE01gl3rmar3QH-dvIC-
cTEaEwb4Tx3jl22LRqrn7EzNT50kAxSbgXsTy79JQxt_ft9z2-
d0igQ9putpxowiHM7FEFvz6WZFdZcgzQ0k4eOgKE-ai0gKK3-
0xazrtTh1gL3cgPk6P9"
```

Para poder realizar esta solicitud en el proyecto primero definí un objeto URL y coloqué la URL que genera Spotify, pero sin encabezados, después utilicé HttpURLConnection para generar la solicitud HTTP y definir el tipo de solicitud, que en este caso fue GET ya que se estaban solicitando datos. Luego para ya agregar los tres encabezados que vienen en Spotify a la solicitud usé setRequestProperty, uno de los encabezados requiere el token de autorización, por lo que definí al principio un string con el token para mayor comodidad, pues este token expira después de unos minutos y se necesita actualizar por otro para que la solicitud funcione.

```

@GET
@Produces(MediaType.TEXT_PLAIN)
public String playlist() throws IOException {
    String token = "BQBbfC0DPb9fzLF9hZUDUrc5eENE01gl3rmar3QH-dvIC-cTEaEwb4Tx3jl22LRqrn7EzNT50kAxSbgXsT";
    String resultado;
    JSONObject jsonObject;

    URL url = new URL("https://api.spotify.com/v1/playlists/1bqfMWQo9tEvKZJg4oKL0R?fields=name%2Cfollowers");
    HttpURLConnection connection = (HttpURLConnection) url.openConnection();
    connection.setRequestMethod("GET");
    connection.setRequestProperty("Accept", "application/json");
    connection.setRequestProperty("Content-Type", "application/json");
    connection.setRequestProperty("Authorization", "Bearer "+token);

    resultado = convertir(connection);

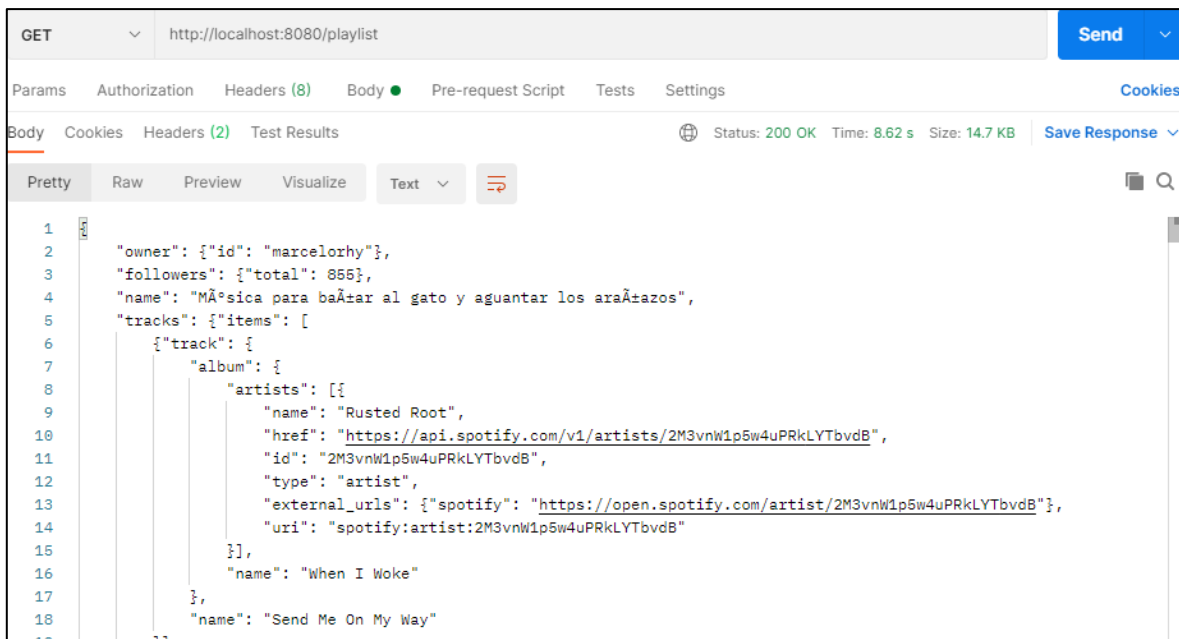
    jsonObject = new JSONObject(resultado);

    return jsonObject.toString(4);
}

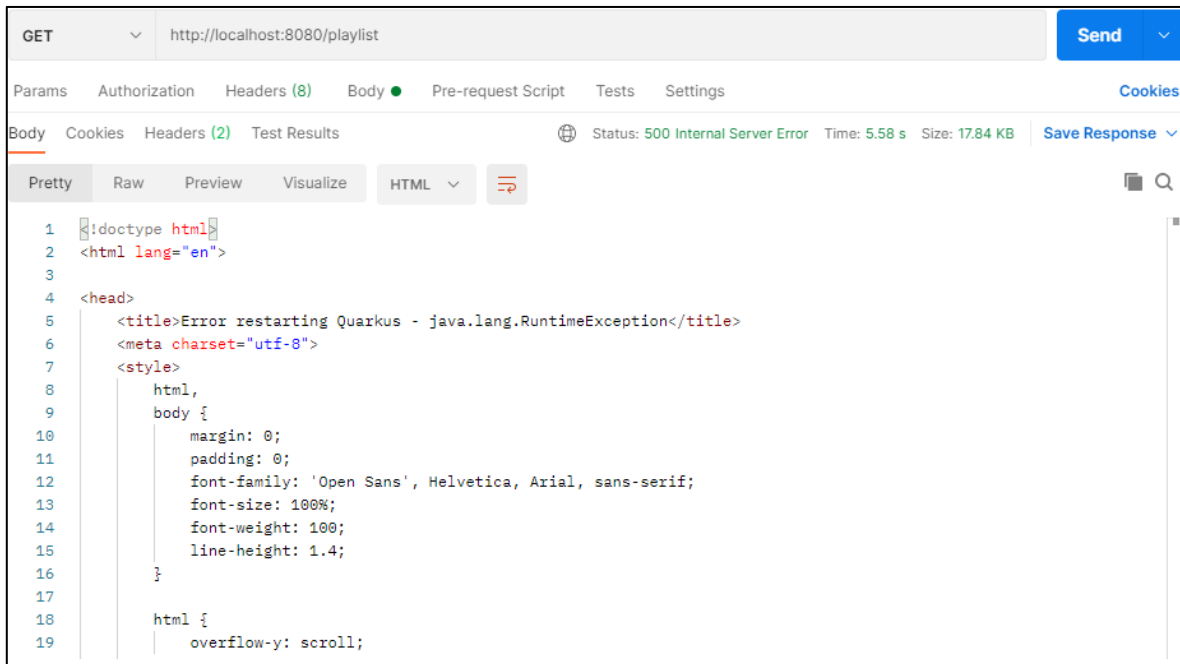
```

Después de realizar la solicitud, en la función “convertir” se obtiene el flujo de entrada de la conexión y este se va leyendo y convirtiendo a una cadena de caracteres para al final retornarla y convertirla a un objeto json para que se pueda apreciar mejor, pues la solicitud genera la respuesta con formato json.

Para comprobar que funciona, se ejecutó el proyecto y en Postman se realizó la solicitud GET a <http://localhost:8080/playlist> ya que a esa clase se le añadió el path de /playlist. Se aprecia que se recuperaron los datos de cada canción que hay en la playlist de Spotify.



Por el contrario, si hay un error en el token o este ya expir\u00f3, o por alguna otra raz\u00f3n no se puede completar la solicitud con \u00e9xito, marcar\u00e1 un error:

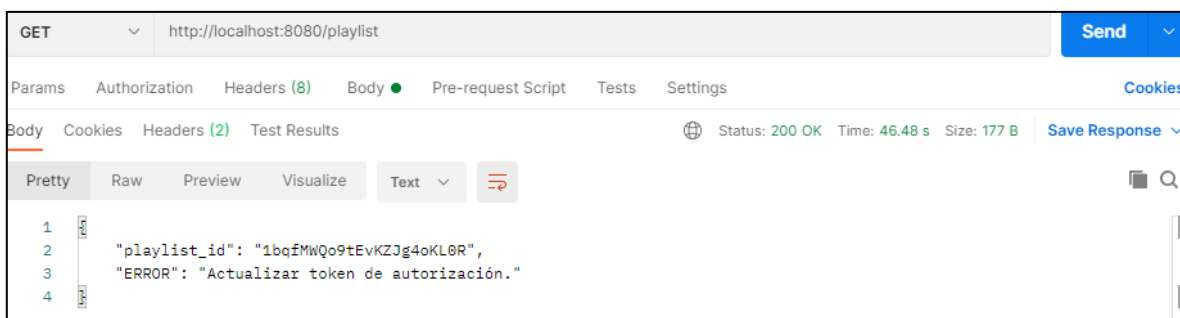


Para evitar que esto ocurra, si es que ocurre algún fallo en la solicitud, se puede dar una respuesta alternativa con `@Fallback` en el que se especifica el método alternativo que se va a ejecutar en caso de que falle la solicitud, en este caso el método alternativo retorna el id de la playlist y un mensaje de error. También con `@Timeout` se especifica el máximo de segundos que se debe intentar ejecutar la solicitud antes de ir al fallback.

```

@GET
@Timeout(value=5000L)
@Fallback(fallbackMethod = "getFallback")
@Produces(MediaType.TEXT_PLAIN)
public String playlist() throws IOException {
    String token = "BOAwkylWLGPIIsEOxR5FEkcOHNRcYg

```



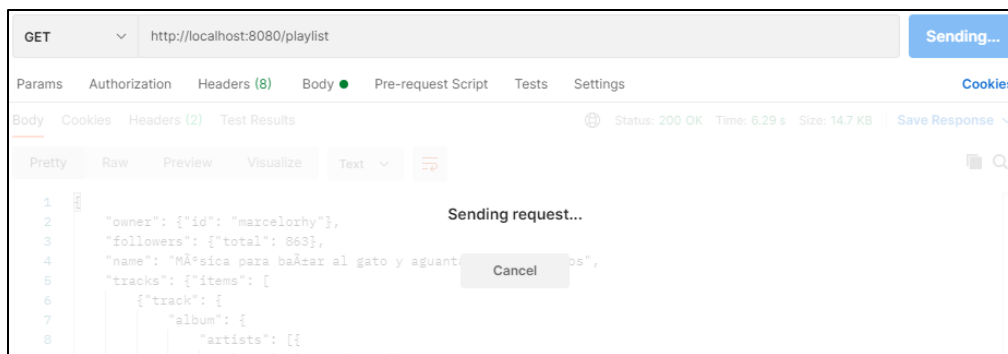
También se puede hacer que la solicitud vuelva a intentarse un determinado número de veces si es que hubo algún problema, esto se hace con `@Retry`, especificando el máximo número de reintentos que se harán, en este caso 10 intentos.

```

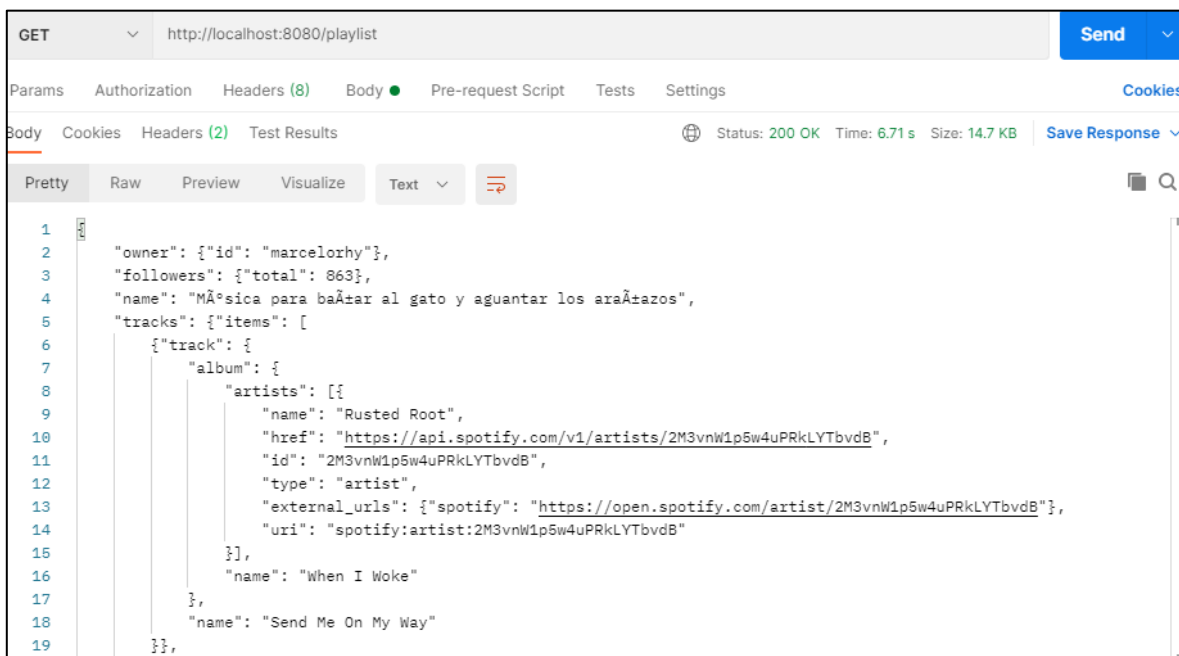
@GET
//@Timeout(value=5000L)
@Retry(maxRetries = 10)
@Fallback(fallbackMethod = "getFallback")
@Produces(MediaType.TEXT_PLAIN)
public String playlist() throws IOException {
    String token = "BQAwkvVLWGPUseQxB5FkcQHNBc

```

Para probarlo quité un carácter del token para que este fuera erróneo, y aquí se observa cómo está intentando realizar la solicitud:



En ese lapso de tiempo arreglé el token y ya se enviaron los datos de manera exitosa:

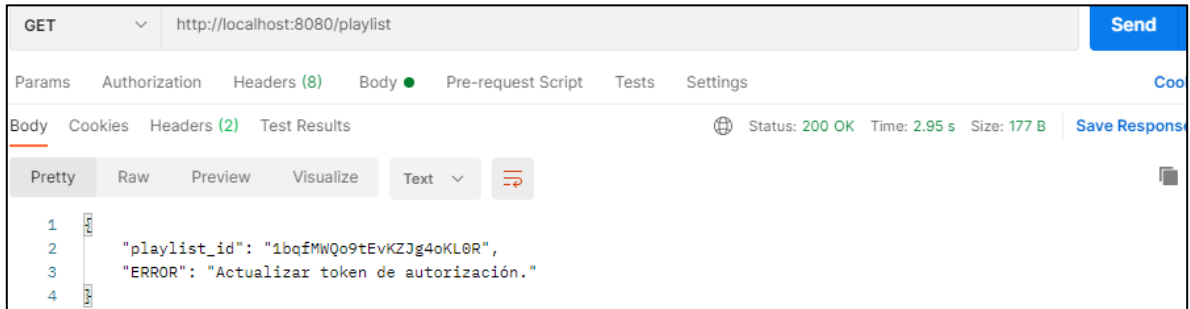


Por último, con `@Bulkhead` se puede definir cuántas peticiones simultáneas se pueden tener, aquí por ejemplo al definir 0 clientes simultáneos, al intentar enviar la solicitud, esta va a fallar:

```

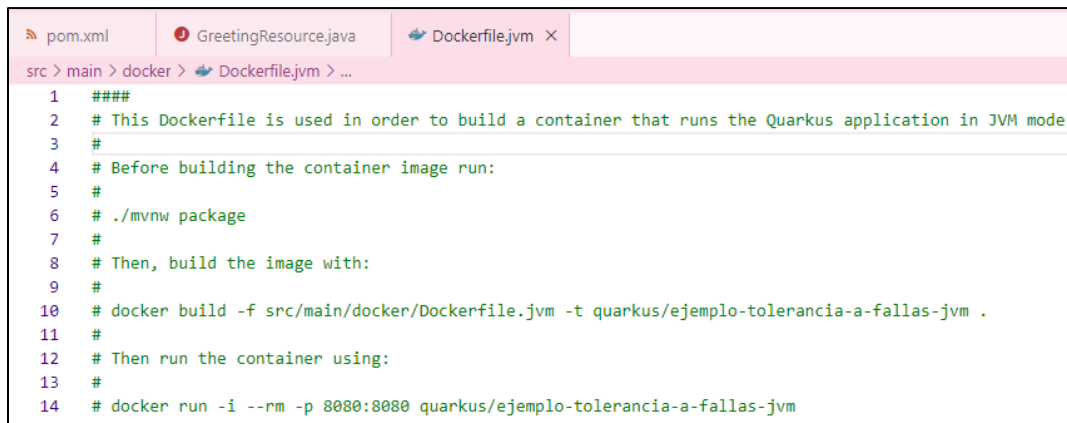
@Bulkhead(value = 0)
@Fallback(fallbackMethod = "getFallback")
@Produces(MediaType.TEXT_PLAIN)
public String playlist() throws IOException {
    String token = "BQAwkvVLWGPUseQxB5FkcQHNBcYGL4bsI

```

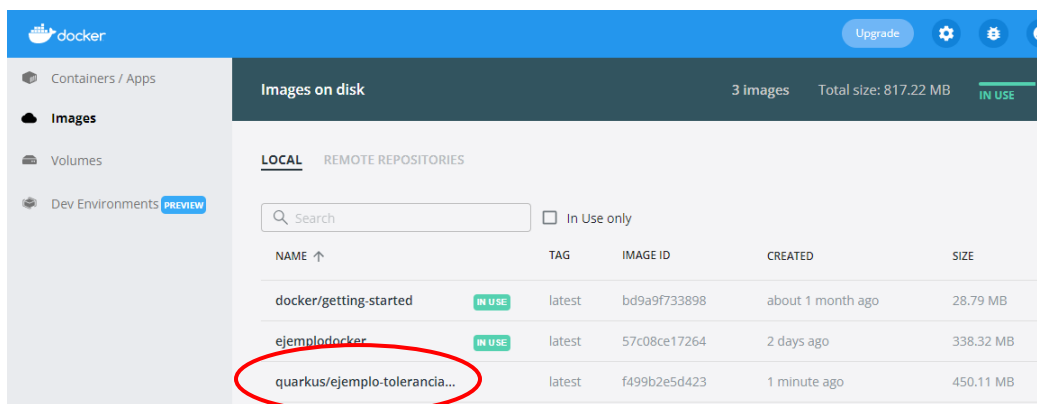


Pero si se define 1 cliente simultáneo, entonces la solicitud se envía correctamente.

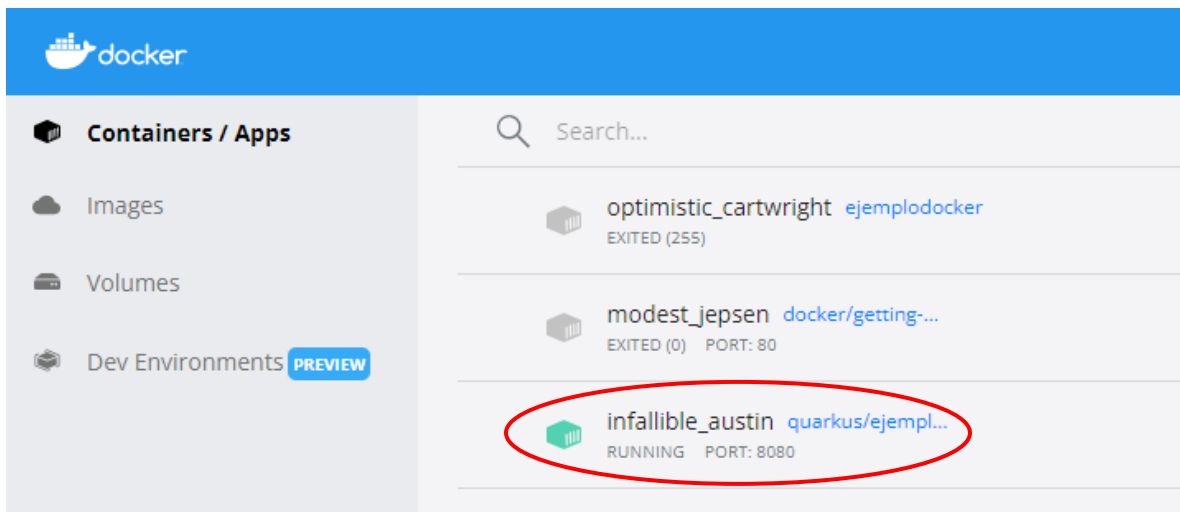
Por último, se puede empaquetar la aplicación en Docker, para ello se tienen que ejecutar los comandos que vienen en Dockerfile.jvm, mismos archivos de Docker que genera Quarkus:



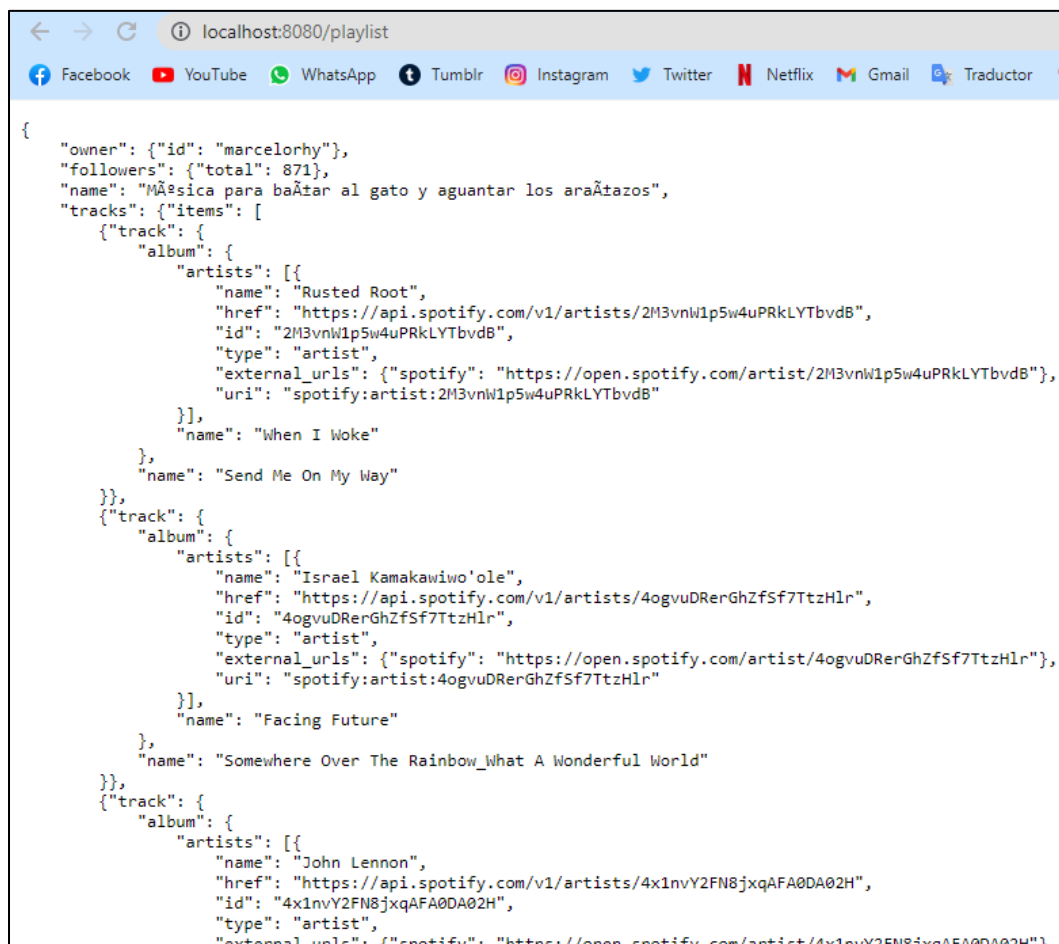
Una vez construida la imagen, se puede observar en la interfaz de Docker:



Al ejecutar el contenedor este aparece también en Docker:



Y al solicitar los datos estos se muestran correctamente:



Código en GitHub: <https://github.com/diana-castillo/MicroProfile-Quarkus-y-Docker.git>

Conclusiones:

Esta actividad me pareció muy distinta a las demás debido a que yo no programo en el lenguaje de Java, pero gracias a esto pude familiarizarme un poco con algunos conceptos de este lenguaje, y el trabajar con MicroProfile en Quarkus me ayudó a comprender el cómo funcionan estas herramientas para manejar la tolerancia a fallos en Java, pues uno de los propósitos de estas herramientas es el poder desarrollar aplicaciones portables que puedan ser ejecutadas en cualquier entorno de trabajo, además de poder tener al alcance distintas bibliotecas como las de MicroProfile para poder tratar los posibles errores que puedan surgir, todo en un mismo espacio de trabajo, lo cual me parece más sencillo de ejecutar y comprender.

Referencias:

Díaz Solís, G. (2019, 28 marzo). *MicroProfile, el éxito de microservicios depende de la cultura de tu Empresa*. MicroProfile. Recuperado 20 de marzo de 2022, de <https://microprofile.io/2019/03/28/microprofile-el-exito-de-microservicios-depende-de-la-cultura-de-tu-empresa/>

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

de la Vega, R. (s. f.). *Cómo enviar solicitudes HTTP en Java*. Pharos. Recuperado 20 de marzo de 2022, de <https://pharos.sh/como-enviar-solicitudes-http-en-java/>

Parzibyte. (2019, 14 febrero). *Petición HTTP GET en Java para consumir HTML o JSON*. Parzibyte's blog. Recuperado 20 de marzo de 2022, de <https://parzibyte.me/blog/2019/02/14/peticion-http-get-java-consumir-html-json/>

Convertir cadena en objeto JSON en Java. (2020, 28 diciembre). DelftStack. Recuperado 20 de marzo de 2022, de <https://www.delftstack.com/es/howto/java/java-convert-string-to-json-object/>

Orozco, V. [Víctor Orozco]. (2021, 8 febrero). *Tolerancia a fallas con MicroProfile, Quarkus y Docker* [Video]. YouTube. <https://www.youtube.com/watch?t=2&v=sTkoITRuPIE&feature=youtu.be>