

# MicroProfile, Quarkus, y Docker

Computación Tolerante a Fallas

**Alumno:** /

**Código:**

**Profesor(a):** Dr. Michel Emanuel López Franco

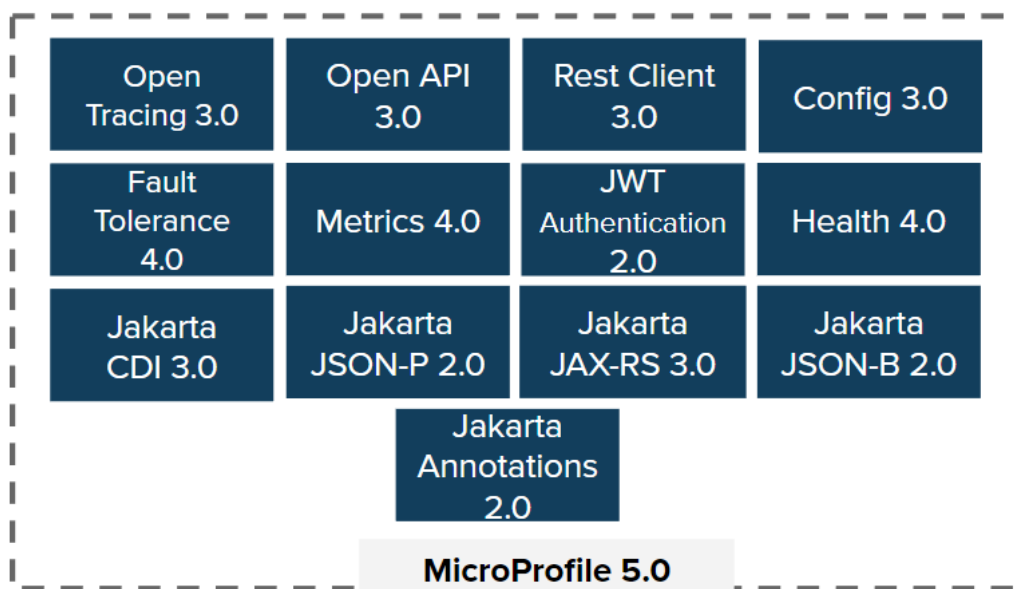
**Sección:** D06

**Fecha de Entrega:** 17 de octubre de 2022

## MicroProfile

Así se le conoce a una especificación orientada a la comunidad diseñada para proveer una definición base de plataforma que optimice Enterprise Java para las arquitecturas de microservicios y que pueda soportar la portabilidad entre distintas instancias de la especificación e incluso otros entornos donde se quiera ejecutar. Una amplia comunidad de individuos, organizaciones, y vendedores colaboran en este proyecto de código abierto.

Los componentes de la especificación se muestran en la siguiente imagen:



Existen distintos desarrolladores que la implementan, como Microsoft, Quarkus, TomEE, Payara Micro, etc.

## Quarkus

Es un framework de Java 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. Los componentes tecnológicos clave que lo rodean son OpenJDK HotSpot y GraalVM.

Entre sus funcionalidades, incluye:

- Programación en vivo, para poder verificar de inmediato los efectos de los cambios en el código.
- Programación imperativa y reactiva.
- Configuración unificada.
- Creación sencilla de archivos ejecutables de forma original.

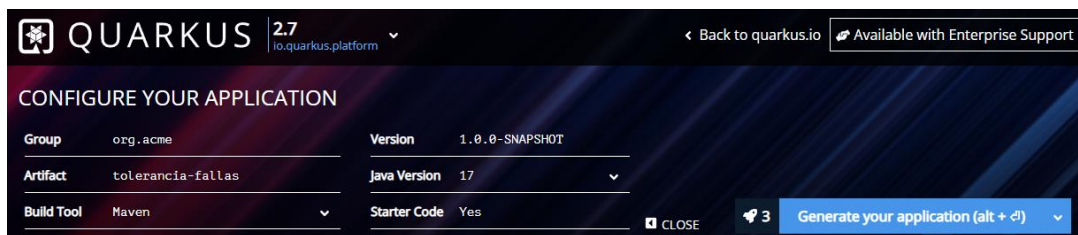
## Docker

Es una plataforma de software que empaqueta software en unidades estandarizadas llamadas contenedores que incluyen todo lo necesario para que el software se ejecute, incluidas bibliotecas, herramientas de sistema, código y tiempo de ejecución. De esta forma, es posible ejecutar aplicaciones rápidamente, además de implementar y ajustar la escala de aplicaciones rápidamente en cualquier entorno con la certeza de saber que su código se ejecutará.

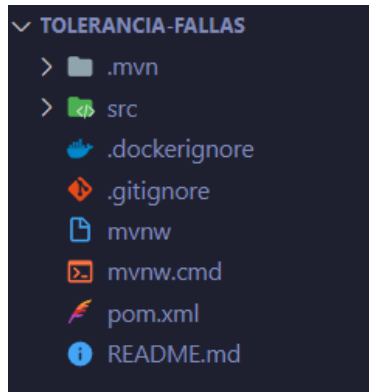
Cada contenedor contiene una serie de instrucciones que indicaran la forma en que debe arrancar.

## Ejemplo

Lo primero que hice fue generar una aplicación dentro de Quarkus. Se deben agregar las dependencias JAX-RS, JSON-B, y SmallRye Fault Tolerance, que es una implementación de MicroProfile.



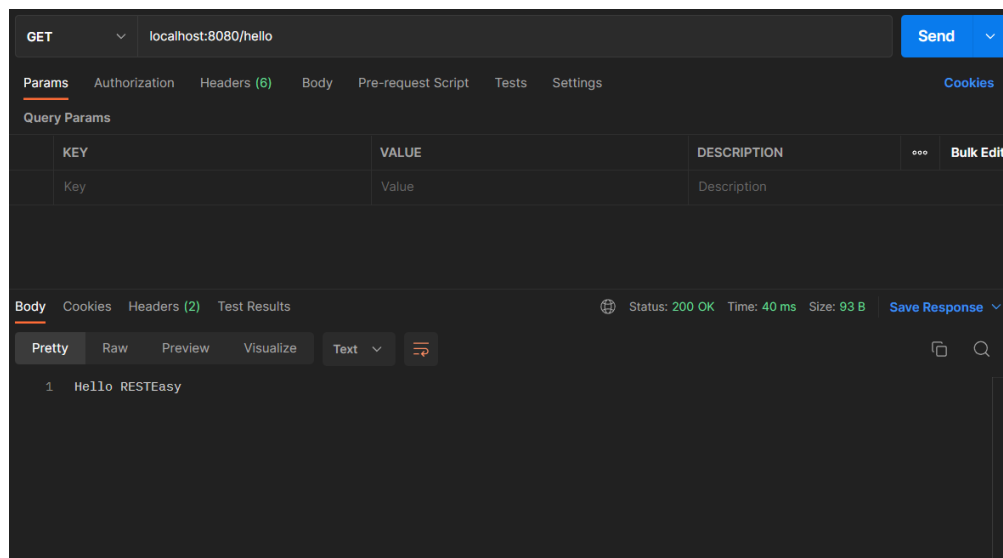
Nos genera los siguientes archivos en un ZIP que podemos descargar.



Luego, es necesario instalar Apache Maven en el entorno de trabajo local para poder utilizar la aplicación. Ejecutamos el comando `mvn compile quarkus:dev` para lanzar la aplicación en un puerto aleatorio, , que en mi caso fue el 8080, en forma de una API REST.



Utilizando el software de Postman, realizamos una consulta de prueba, que como vemos fue exitosa.



El programa que voy a implementar es similar a los anteriores que he realizado, en el que hago una petición a la API de Unsplash para obtener una imagen de un tema específico. Para realizar las peticiones en Java, se deben importar las siguientes librerías:

```
import java.io.IOException;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;
```

Algunas otras que Quarkus necesita para que el programa funcione correctamente son:

```
import javax.ws.rs.GET;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;

import org.eclipse.microprofile.faulttolerance.*;

import org.json.JSONException;
import org.json.JSONObject;
```

Todo el programa se englobará dentro de una clase denominada *imagen*, para evitar mayores complicaciones. Además, se indica que el tipo de medio que produce es una aplicación JSON.

```
@Path ("/imagen")
@Produces(MediaType.APPLICATION_JSON)
public class Imagen {
```

Lo primero que hacemos es establecer los decoradores del método, *timeout* (establecer el tiempo de espera entre intentos que en este caso son 5000 segundos), *retry* (cuantas veces reintentará el programa antes de terminarlo), *bulkhead* (cuantas peticiones se pueden realizar al mismo tiempo), *fallback* (que hacer en caso de error), y *produces*.

```

@GET
@Timeout(value=5000L)
@Retry(maxRetries = 10)
@Bulkhead(value = 1)
@Fallback(fallbackMethod = "getFallback")
@Produces(MediaType.TEXT_PLAIN)

```

La función principal debe llevar el mismo nombre que el path asignado a la clase GreetingResource. Dentro de esta, declaramos una variable cadena para guardar el URL al que haremos la petición, un objeto JSON, una variable cadena para guardar el resultado, y una variable de tipo URL que guardara el URL creado anteriormente.

```

public String imagen() throws IOException, JSONException{
    String url = "https://api.unsplash.com/photos/random?query=Art&client_id=6Sutk4ov8sdxeT6tfo9KPXxlcz41sAGf7eCpBlywyw";
    JSONObject json;
    String resultado;

    URL request = new URL (url);

```

Para realizar la petición HTTP, tenemos que utilizar un objeto de la clase HttpURLConnection y establecer ciertos parámetros.

```

HttpURLConnection connection = (HttpURLConnection) request.openConnection();
connection.setRequestMethod("GET");
connection.setRequestProperty("Accept", "application/json");
connection.setRequestProperty("Content-Type", "application/json");

```

Ahora, debemos descomponer la respuesta a la petición, ya que esta viene en binario y la necesitamos en JSON. Esto lo haremos dentro de otra función que recibe como parámetro la conexión que establecimos con la API. Con el método *getInputStream* creamos un objeto de lectura de buffer, y declaramos algunas variables extra que nos ayudaran en la conversión. Usando un ciclo while, iteramos por el flujo de bytes convirtiendo cada uno de los bytes a una cadena de tipo StringBuilder. Finalmente, convertimos el resultado a una cadena común, y lo convertimos en JSON.

```

public String convertirImagen (HttpURLConnection connection) throws IOException, JSONException{
    BufferedReader rd = new BufferedReader(new InputStreamReader(connection.getInputStream()));
    StringBuilder resultado = new StringBuilder();
    String linea;
    String res;

    while ((linea = rd.readLine()) != null) { // Mientras el BufferedReader se pueda leer, agregar contenido a resultado
        resultado.append(linea);
    }
    rd.close(); // Cerrar el BufferedReader

    res = resultado.toString(); // Regresar resultado, pero como cadena, no como StringBuilder

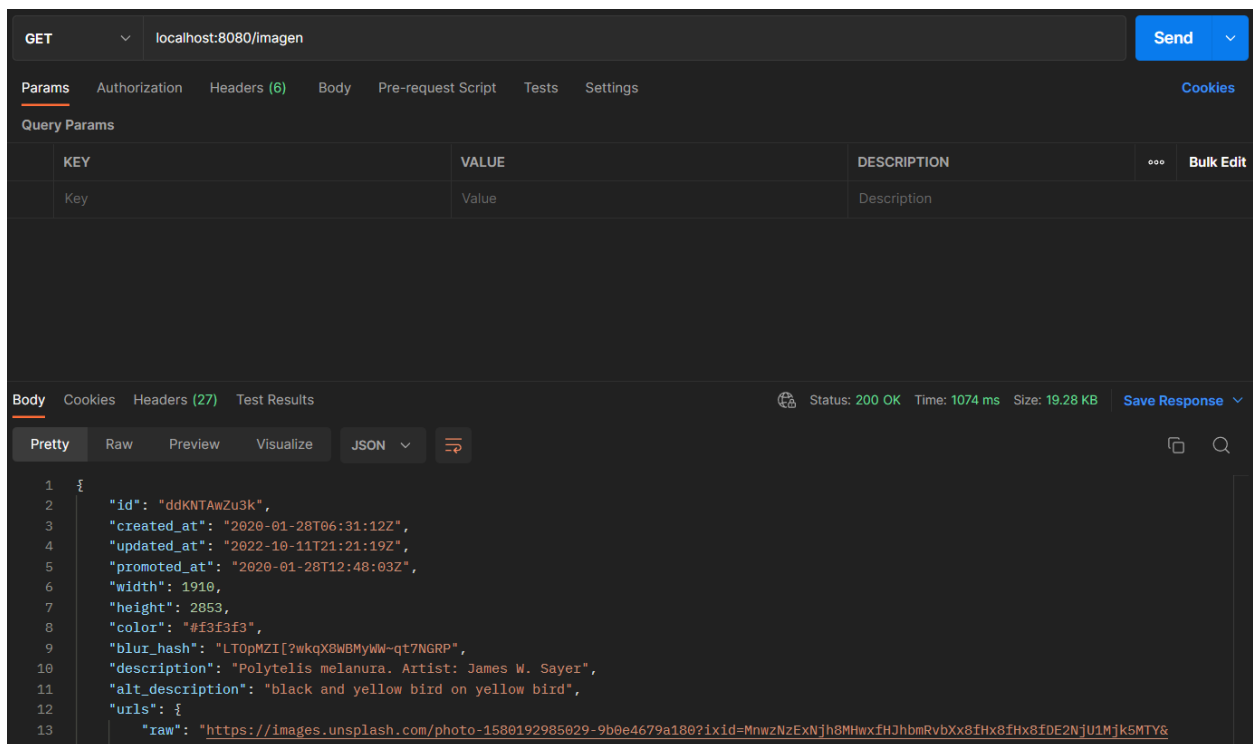
    return res;
}

```

El valor de retorno de la función es asignado a resultado. Para concluir el proceso, creamos un objeto JSON y le pasamos los datos formateados, que serán retornados por la función en forma de cadena.

```
resultado = convertirImagen(connection);  
  
json = new JSONObject(resultado);  
  
return json.toString();
```

Al hacer una petición desde Postman al puerto 8080, obtenemos como resultado el JSON de una imagen.



The screenshot shows the Postman interface for a GET request to `localhost:8080/imagen`. The response status is 200 OK, with a time of 1074 ms and a size of 19.28 KB. The response body is displayed in JSON format, showing a JSON object with the following fields:

KEY	VALUE	DESCRIPTION
Key	Value	Description

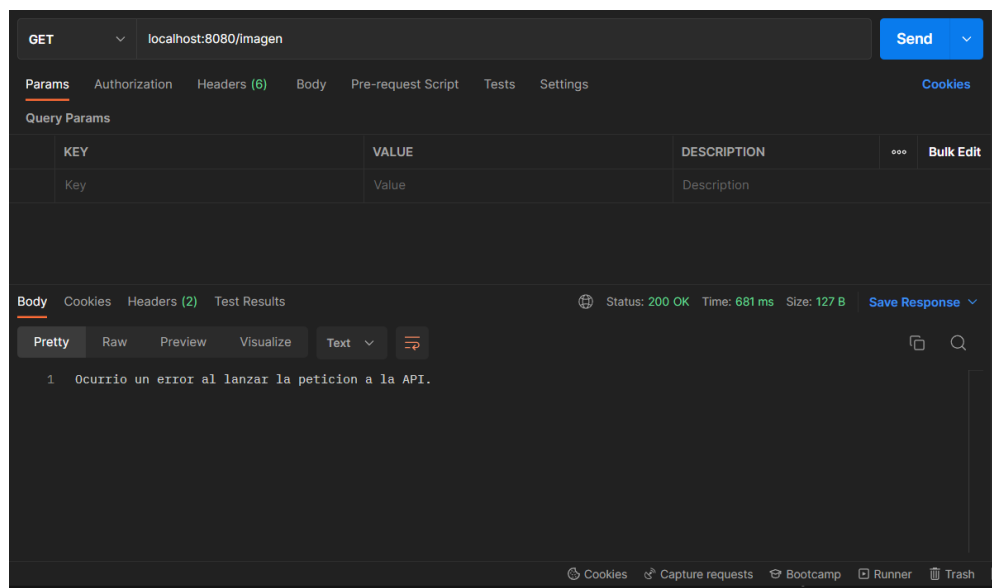
```
{  
  "id": "ddKNTAwZu3k",  
  "created_at": "2020-01-28T06:31:12Z",  
  "updated_at": "2022-10-11T21:21:19Z",  
  "promoted_at": "2020-01-28T12:48:03Z",  
  "width": 1910,  
  "height": 2853,  
  "color": "#f3f3f3",  
  "blur_hash": "LTOpMZI[?wkqX8WMyWW~qt7NGRP",  
  "description": "Polytelis melanura. Artist: James W. Sayer",  
  "alt_description": "black and yellow bird on yellow bird",  
  "urls": {  
    "raw": "https://images.unsplash.com/photo-1580192985029-9b0e4679a180?ixid=MnwZnZExNjh8MHwxfh8jbmRvbXx8fHx8fHx8fDE2NjU1Mjk5MTY&fm=jpg&crop=entropy&cs=tinysrgb&f=jpg&q=80"  }  
}
```

Si la abrimos en el navegador, se muestra la siguiente imagen, que coincide con el tema de arte indicado en los parámetros.



La función a la que llama Fallback se describe más abajo, y simplemente envía un mensaje diciendo que ocurrió un error al lanzar la petición a la API.

```
public String getFallback() throws JSONException {  
    String mensaje = "Ocurrio un error al lanzar la peticion a la API.";  
    return mensaje;  
}
```



Desde la terminal, se muestra que todo fue ejecutado correctamente.



```
--\_\_V///_|/_-V///_/_/\n-/////_||_,/,\\/_\\\n--\\\\\\//_|//|\\\_\\_\n2022-10-11 18:30:34,067 INFO [io.quarkus] (Quarkus Main Thread) tolerancia-fallas 1.  
0.0-SNAPSHOT on JVM (powered by Quarkus 2.7.6.Final) started in 0.365s. Listening on:  
http://localhost:8080\n  
2022-10-11 18:30:34,072 INFO [io.quarkus] (Quarkus Main Thread) Profile dev activate  
d. Live Coding activated.\n2022-10-11 18:30:34,075 INFO [io.quarkus] (Quarkus Main Thread) Installed features:  
[cdi, resteasy, resteasy-jsonb, smallrye-context-propagation, smallrye-fault-toleranc  
e, vertx]\n2022-10-11 18:30:34,078 INFO [io.qva.dep.dev.RuntimeUpdatesProcessor] (vert.x-worker  
-thread-0) Live reload total time: 0.419s\n  
--  
  
All 1 test is passing (0 skipped), 0 tests were run in 20ms. Tests completed at 18:30  
:33 due to changes to Imagen.class.
```

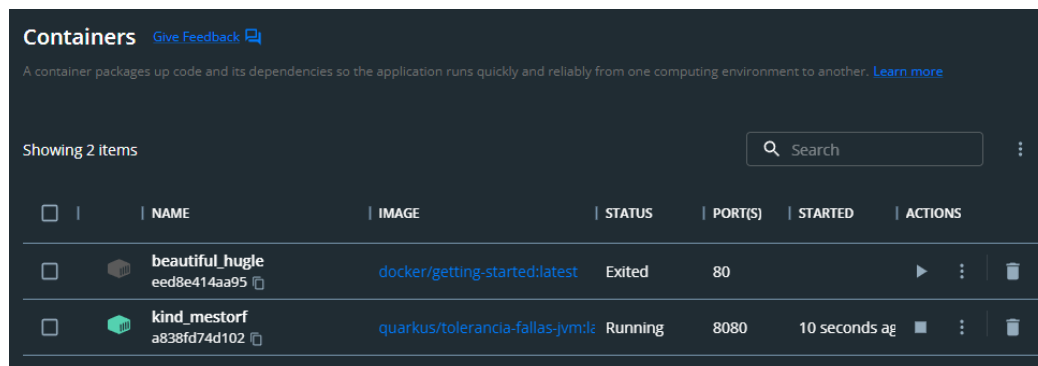
Podemos cargar la aplicación a Docker para utilizarla como un microservicio. Para ello, es necesario instalar dicho software en el equipo y usar el siguiente comando.

```
$ docker build -f src/main/docker/Dockerfile.jvm -t quarkus/tolerancia-fallas-jvm .
```

Y corremos el programa con el siguiente comando.

```
$ docker run -i --rm -p 8080:8080 quarkus/tolerancia-fallas
```

Dentro de la GUI de Docker se muestra como activa y corriendo.



Si accedemos a la ruta desde el navegador, obtendremos una consulta exitosa.

```
localhost:8080/imagen
localhost:8080/imagen
{"topic_submissions":{"arts-culture":{"approved_on":"2020-04-06T14:20:25Z","status":"approved"}}, "current_user_collections":
[], "color": "#d9d9d9", "sponsorship": null, "created_at": "2017-11-04T14:23:18Z", "description": "Where comfort meets design, design
meets art and art meets photography", "public_domain": false, "urls": {"small": "https://images.unsplash.com/photo-1509805225007-
73e8ba4b5be8?crop=entropy&cs=tinysrgb&fit=max&fm=jpg&ixid=MnwzNzExNjh8MwxfHjhbMRvbXx8fHx8fHx8fDE2NjU1MzM0MjQ&ixlib=rb-
1.2.1&q=80&w=400", "small_s3": "https://s3.us-west-2.amazonaws.com/images.unsplash.com/small/photo-1509805225007-
73e8ba4b5be8", "thumb": "https://images.unsplash.com/photo-1509805225007-73e8ba4b5be8?
crop=entropy&cs=tinysrgb&fit=max&fm=jpg&ixid=MnwzNzExNjh8MwxfHjhbMRvbXx8fHx8fHx8fDE2NjU1MzM0MjQ&ixlib=rb-
1.2.1&q=80&w=200", "raw": "https://images.unsplash.com/photo-1509805225007-73e8ba4b5be8?
ixid=MnwzNzExNjh8MwxfHjhbMRvbXx8fHx8fHx8fDE2NjU1MzM0MjQ&ixlib=rb-1.2.1", "regular": "https://images.unsplash.com/photo-
1509805225007-73e8ba4b5be8?
crop=entropy&cs=tinysrgb&fit=max&fm=jpg&ixid=MnwzNzExNjh8MwxfHjhbMRvbXx8fHx8fHx8fDE2NjU1MzM0MjQ&ixlib=rb-
1.2.1&q=80&w=1080", "full": "https://images.unsplash.com/photo-1509805225007-73e8ba4b5be8?
crop=entropy&cs=tinysrgb&fm=jpg&ixid=MnwzNzExNjh8MwxfHjhbMRvbXx8fHx8fHx8fDE2NjU1MzM0MjQ&ixlib=rb-
1.2.1&q=80"}, "alt_description": "white chair beside five paintings", "updated_at": "2022-10-
11T14:02:32Z", "downloads": 49373, "links": {"download": "https://unsplash.com/photos/BdVQU-NDtA8/download?
ixid=MnwzNzExNjh8MwxfHjhbMRvbXx8fHx8fHx8fDE2NjU1MzM0MjQ", "download_location": "https://api.unsplash.com/photos/BdVQU-
NDtA8/download?ixid=MnwzNzExNjh8MwxfHjhbMRvbXx8fHx8fHx8fDE2NjU1MzM0MjQ", "self": "https://api.unsplash.com/photos/BdVQU-
NDtA8", "html": "https://unsplash.com/photos/BdVQU-NDtA8"}, "id": "BdVQU-NDtA8", "views": 8443936, "height": 3569, "likes": 1000, "topics":
```

## Conclusiones

Esta actividad fue todo un reto por distintos motivos. El principal es que nunca había trabajado con el lenguaje de programación Java, y aunque la sintaxis es muy similar a la de otros lenguajes de medio nivel, me fue complicado encontrar las formas de realizar solicitudes y convertir a JSON los datos obtenidos. Además, Quarkus no es muy descriptivo con los errores de compilación, y gracias a eso perdí bastante tiempo intentando averiguar que partes del código me estaban dando problemas. Sin embargo, al final resolví todo mediante investigaciones en foros de la web.

El reporte fue realizado sin problema, así como la investigación, y espero las próximas actividades no sean tan desafiantes, pero que brinden un buen nivel de aprendizaje.

## Bibliografía

Contenedores de Docker | ¿Qué es Docker? | AWS. (s. f.). Amazon Web Services, Inc. Recuperado 10 de octubre de 2022, de <https://aws.amazon.com/es/docker/>

¿Qué es Quarkus? (s. f.). Recuperado 10 de octubre de 2022, de <https://www.redhat.com/es/topics/cloud-native-apps/what-is-quarkus>

IBM Developer. (s. f.). Recuperado 10 de octubre de 2022, de <https://developer.ibm.com/series/what-is-microprofile/>