

Práctica Microservicios – SPS

JOSE EDUARDO PEÑA REBOLLO

PENALALO016@GMAIL.COM

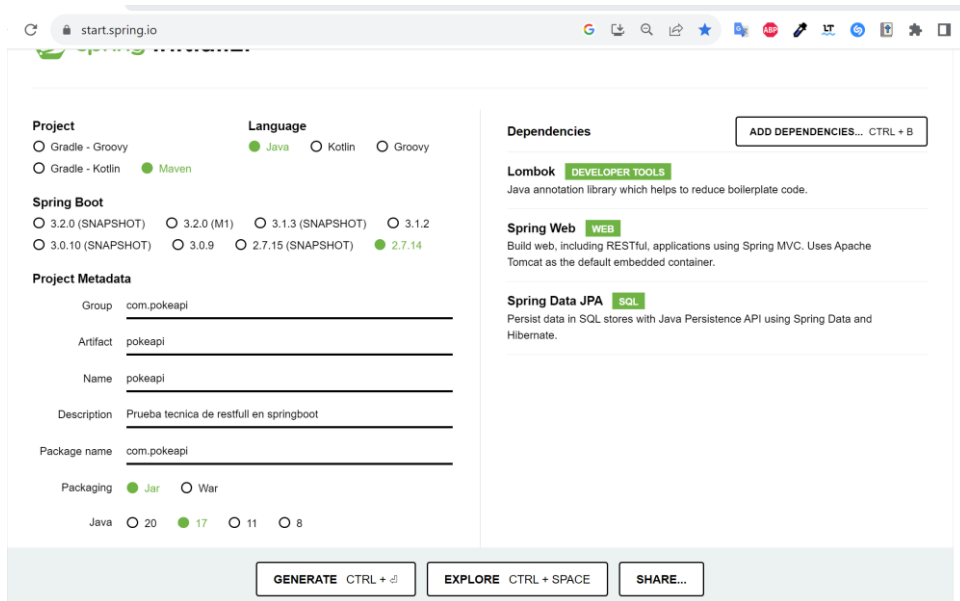
<https://github.com/Edward-Pena/>

POKEAPI

Práctica Microservicios – SPS.....	1
Inicio.....	2
Construcción del microservicio.....	4
Despliegue.....	10
DOCKER	14
Conclusión	16

Inicio.

Para comenzar el proyecto nos dirigimos a <https://start.spring.io/> para obtener el template del servicio que vamos a construir.

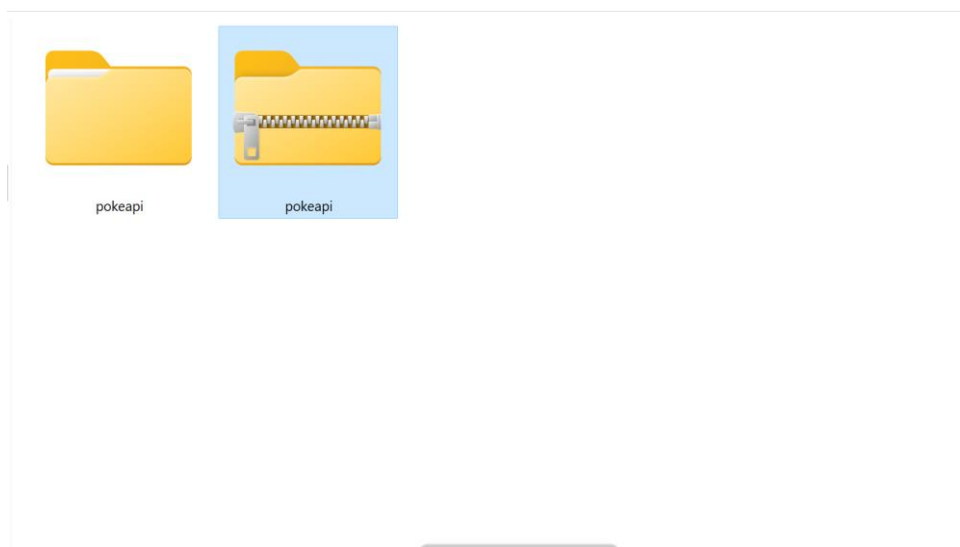


The screenshot shows the Spring Start web interface. It is divided into several sections for configuring a new project:

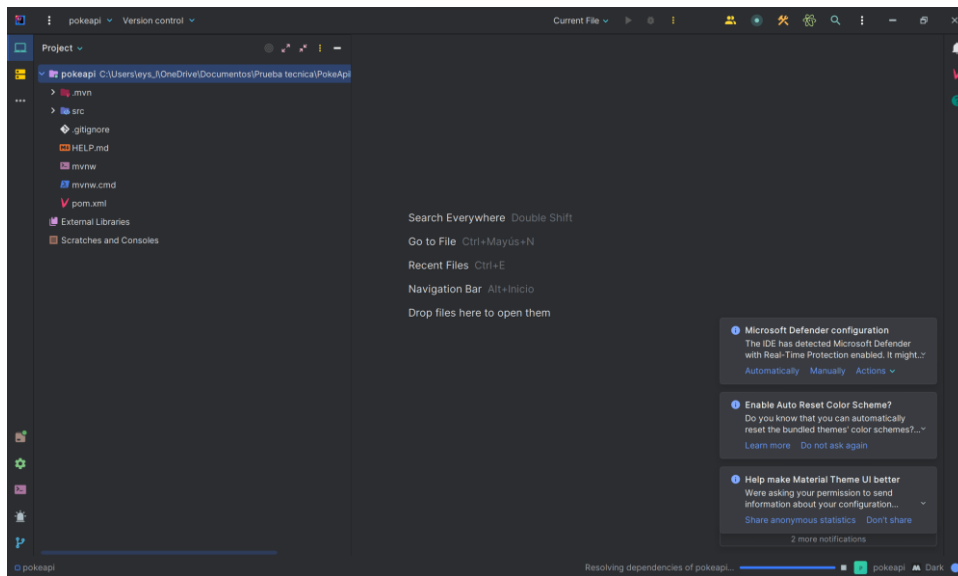
- Project:** Options for **Gradle - Groovy**, **Gradle - Kotlin**, and **Maven** (selected).
- Language:** Options for **Java** (selected), **Kotlin**, and **Groovy**.
- Spring Boot:** Version selection including 3.2.0 (SNAPSHOT), 3.2.0 (M1), 3.1.3 (SNAPSHOT), 3.1.2, 3.0.10 (SNAPSHOT), 3.0.9, 2.7.15 (SNAPSHOT), and **2.7.14** (selected).
- Project Metadata:** Fields for Group (com.pokeapi), Artifact (pokeapi), Name (pokeapi), Description (Prueba tecnica de restfull en springboot), and Package name (com.pokeapi).
- Packaging:** Options for **Jar** (selected) and **War**.
- Java:** Version selection including 20, **17** (selected), 11, and 8.
- Dependencies:** A section with a button "ADD DEPENDENCIES... CTRL + B" and a list of dependencies: **Lombok** (DEVELOPER TOOLS), **Spring Web** (WEB), and **Spring Data JPA** (SQL).

At the bottom, there are three buttons: **GENERATE CTRL + G**, **EXPLORE CTRL + SPACE**, and **SHARE...**

Seleccionamos proyecto, lenguaje, versión y dependencias. Damos en generar y nos descargara un comprimido.



Descomprimos el archivo generado y abrimos en el editor en este caso intelliject

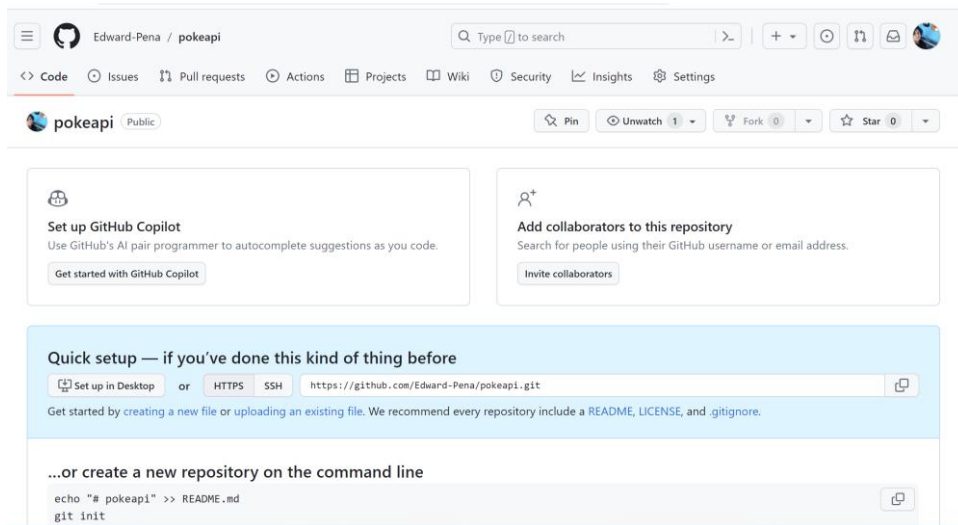


Esperamos que carguen las dependencias.

Debemos tener instalado:

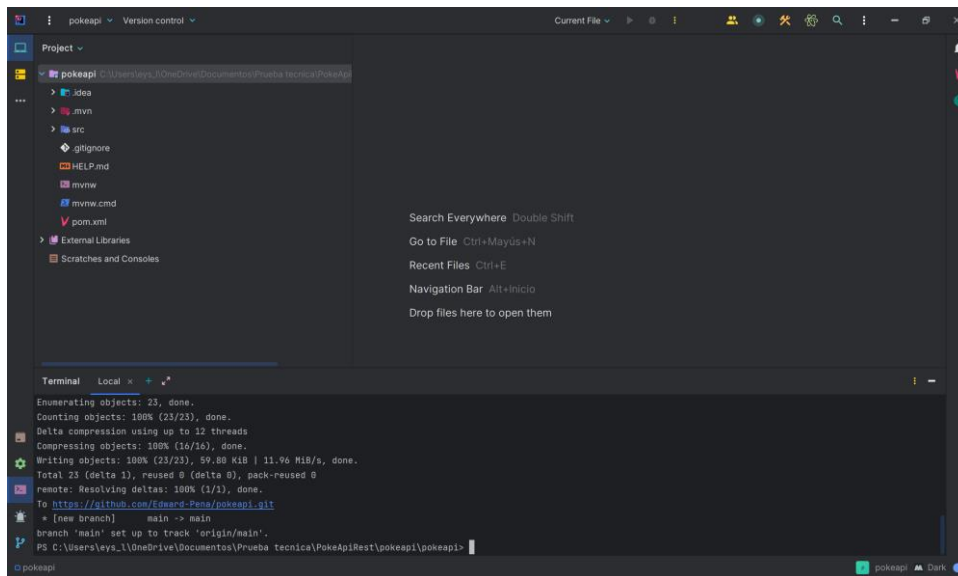
- JDK 17
- MAVEN
- DOCKER

Mientras tanto creamos un repositorio

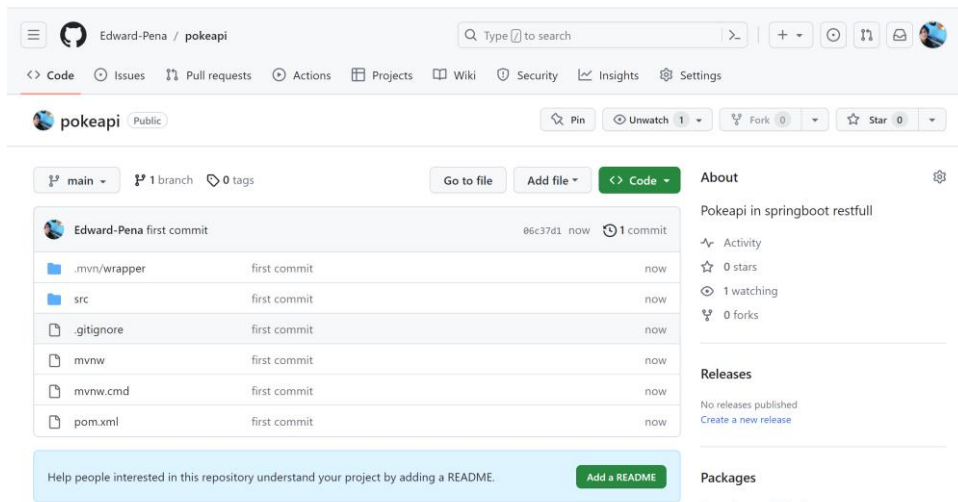


Una vez creado lo inicializamos.

Subimos el repositorio



Y verificamos que todo esté en orden para comenzar a trabajar



Ahora bien, podemos comenzar.

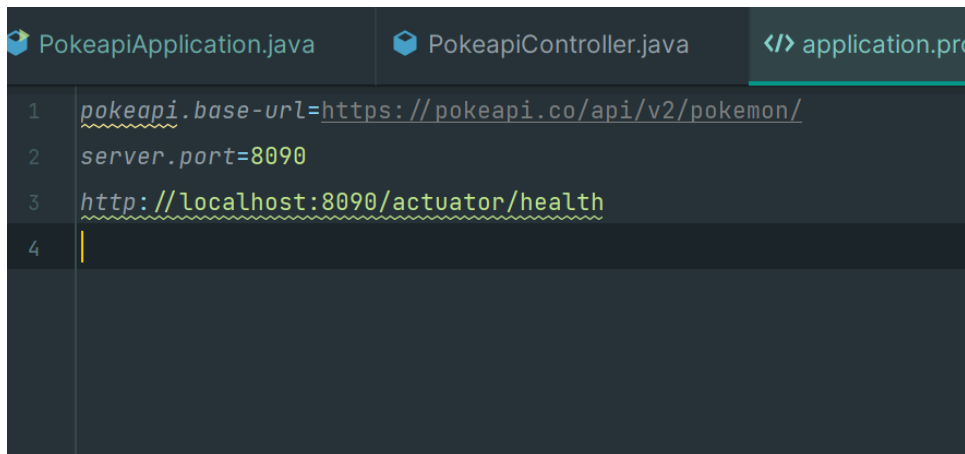
Abriremos la documentación de [PokéAPI \(pokeapi.co\)](https://pokeapi.co) y obtenemos la información que necesitamos para construir nuestro servicio.

En este caso la API solo admite GET entonces lo dirigiremos a obtener los datos de un Pokémon por nombre utilizando la api <https://pokeapi.co/api/v2/pokemon/{nombre}>

Construcción del microservicio.

Comenzamos con el controlador.

Definimos las propiedades.



```
1 pokeapi.base-url=https://pokeapi.co/api/v2/pokemon/
2 server.port=8090
3 http://localhost:8090/actuator/health
4 |
```

Estas propiedades se implementan más adelante.

Creamos nuestro RESTController



```
@GetMapping(value = "/pokemon", produces = MediaType.APPLICATION_JSON_VALUE)
public Pokemon getOnePokemon(@RequestParam("name") String name) {
    String url = ApiUrl+name;
    RestTemplate restTemplate = new RestTemplate();
    try {
        ResponseEntity<String> responseEntity = restTemplate.exchange(url, HttpMethod.GET, null, String.class);

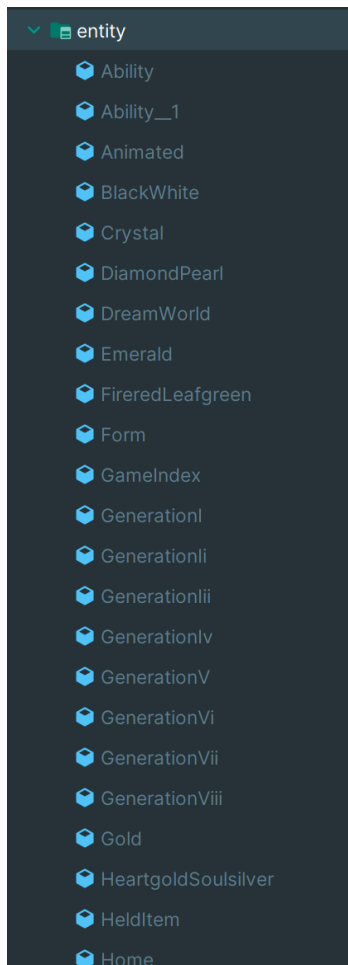
        if (responseEntity.getStatusCode().is2xxSuccessful()){
            String jsonResponse = responseEntity.getBody();
            ObjectMapper objectMapper = new ObjectMapper();
            Pokemon pokemon = objectMapper.readValue(jsonResponse, Pokemon.class);
            return pokemon;
        } else if (responseEntity.getStatusCode().is4xxClientError()){
            LOGGER.info("CLIENT ERROR");
        } else if (responseEntity.getStatusCode().is5xxServerError()){
            LOGGER.info("SERVER ERROR");
        } else {
            LOGGER.severe("UNKNOWN ERROR");
        }
    } catch (Exception e) {
        LOGGER.severe("Error a causa de: "+e);
    }
    return null;
}
```

Donde @GetMapping nos permite hacer nuestro request al api en formato GET y le solicitamos que va a recibir un parámetro name.

Creamos nuestra variable url para guardar la dirección de la API, y creamos nuestro RestTemplate que nos permite hacer nuestra llamada a la api externa.

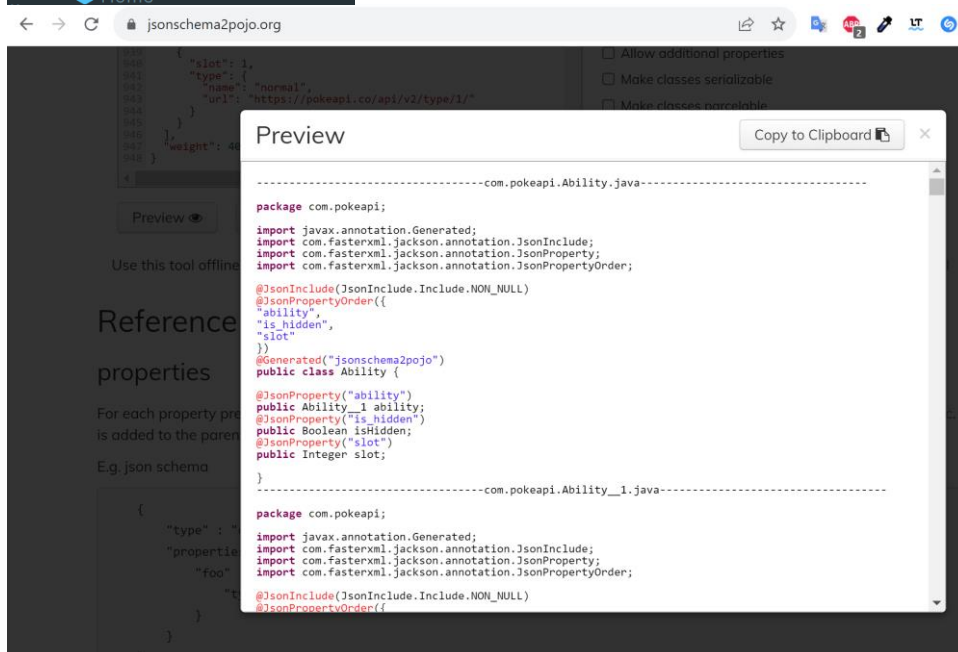
Hacemos un try para cachear errores y definimos que si la respuesta es ok (status code 200) vamos a obtener la respuesta del api y guardarla en un objeto tipo response.

Una vez obteniendo la respuesta se mapean los datos obtenidos en formato json almacenándolos en nuestras entidades ya definidas.

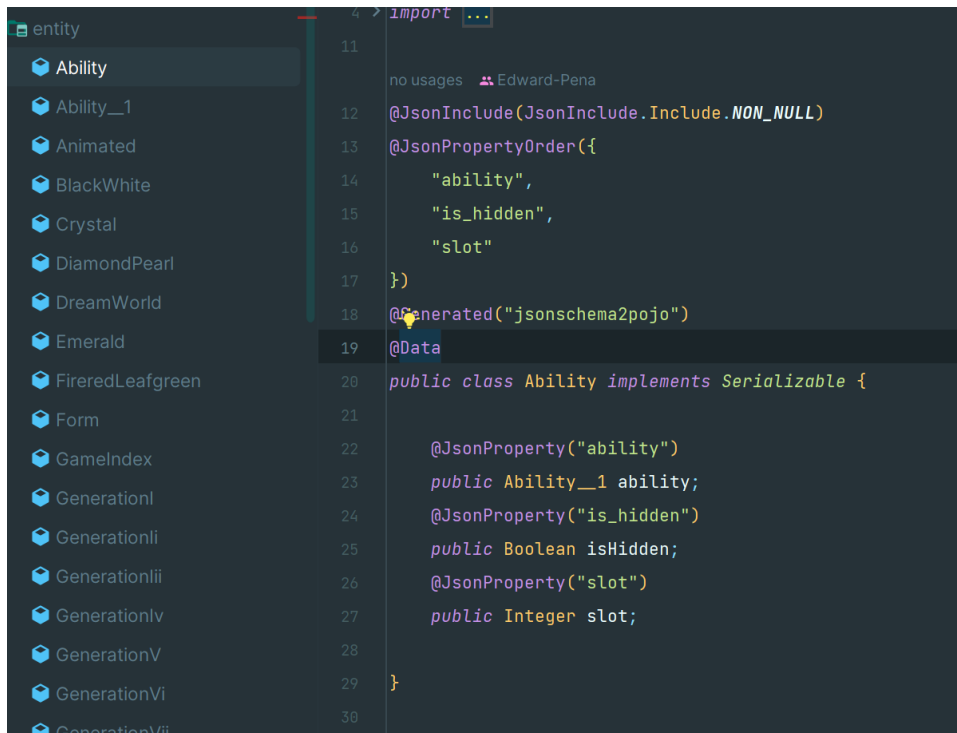


Se crea una entidad por cada valor que se obtiene del api para este proceso nos ayudamos de una herramienta llamada <https://www.jsonschema2pojo.org/> la cual nos permite generar estos entitis a partir de la respuesta JSON.

Aquí se muestra el ejemplo de cómo a partir de la respuesta json se genera el pojo.



Descargamos los archivos y los ajustamos al contenido.



```
entity
├── Ability
├── Ability__1
├── Animated
├── BlackWhite
├── Crystal
├── DiamondPearl
├── DreamWorld
├── Emerald
├── FireredLeafgreen
├── Form
├── GameIndex
├── GenerationI
├── GenerationIi
├── GenerationIii
├── GenerationIv
├── GenerationV
├── GenerationVi
└── GenerationVii

4  > import ...
11
no usages Edward-Pena
12 @JsonInclude(JsonInclude.Include.NON_NULL)
13 @JsonPropertyOrder({
14     "ability",
15     "is_hidden",
16     "slot"
17 })
18 @Generated("jsonschema2pojo")
19 @Data
20 public class Ability implements Serializable {
21
22     @JsonProperty("ability")
23     public Ability__1 ability;
24     @JsonProperty("is_hidden")
25     public Boolean isHidden;
26     @JsonProperty("slot")
27     public Integer slot;
28
29 }
30
```

Nos aseguramos que sea `@Data` y que implemente la propiedad serializable.

Es importante para la deserialización de la respuesta en los objetos.

Una vez terminado se agregan clases de status como el health check por springboot.



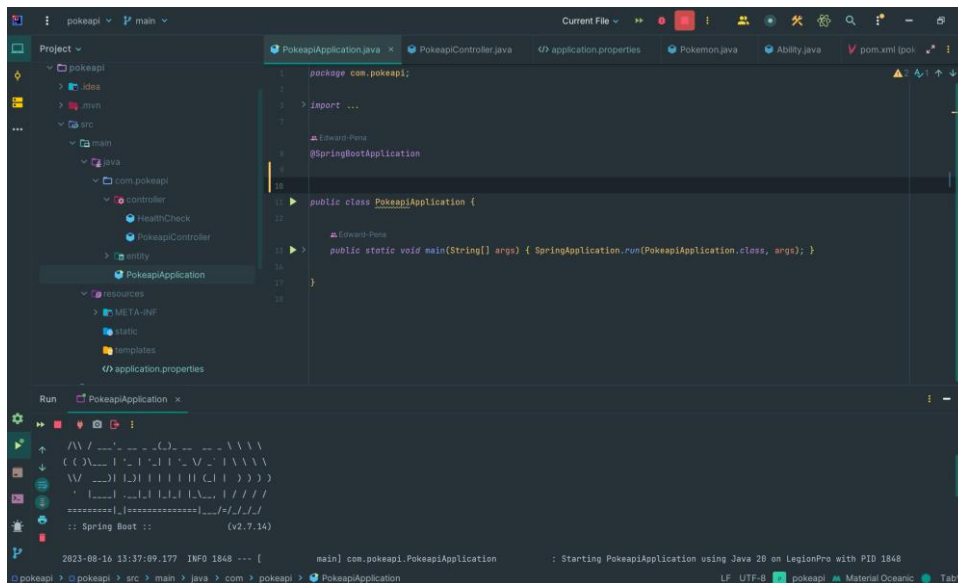
```
package com.pokeapi.controller;

import org.springframework.boot.actuate.health.Health;
import org.springframework.boot.actuate.health.HealthIndicator;
import org.springframework.stereotype.Component;

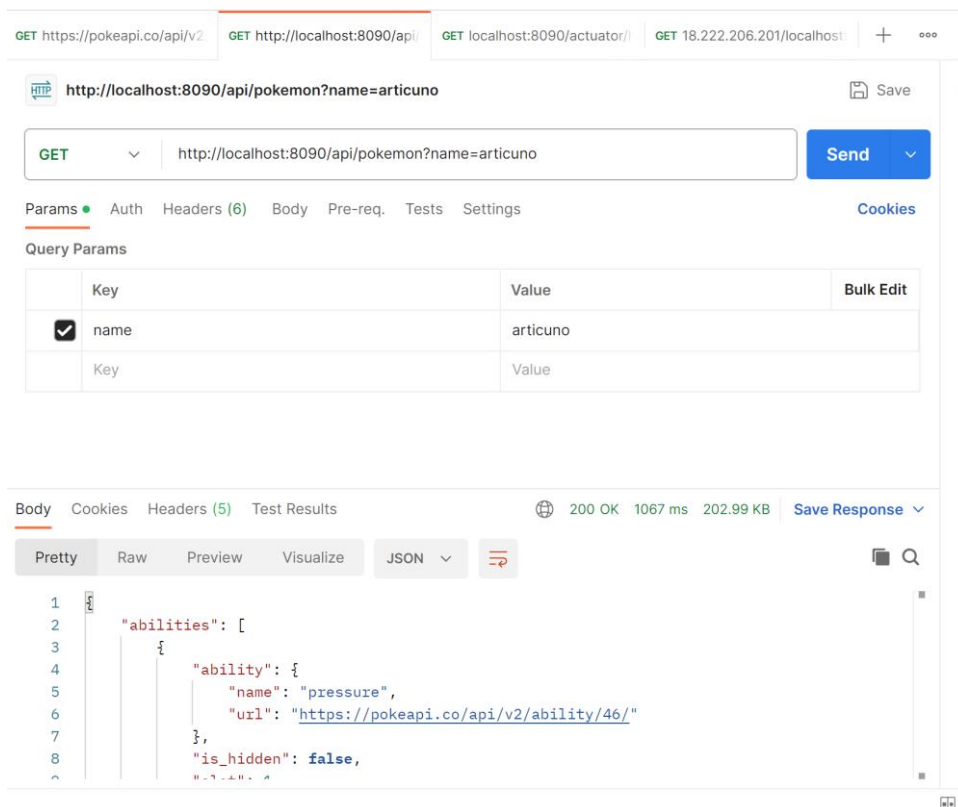
no usages Edward-Pena
@Component
public class HealthCheck implements HealthIndicator {

    Edward-Pena
    @Override
    public Health health() {
        return Health.up().build();
    }
}
```

Una vez terminado definimos los puertos y lanzamos nuestra aplicación.



Y verificamos que funciona con <https://www.postman.com/>



Aquí logramos ver que efectivamente se tiene una respuesta como se esperaba. JAR.

Vamos a lanzar el proyecto desde el jar de la consola de comandos de windows (cmd)


```
C:\Windows\System32\cmd.exe
Microsoft Windows [Versión 10.0.22621.2134]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\eyls_l\OneDrive\Documentos\Pruebatecnica\PokeApiRest\pokeapi\pokeapi\target>java -jar pokeapi-0.0.1-SNAPSHOT.jar
```

Escribimos el siguiente comando `java -jar` y el nombre del jar y le damos enter.

Si todo sale bien nuestro proyecto estará corriendo

```
C:\Users\eyls_l\OneDrive\Documentos\Pruebatecnica\PokeApiRest\pokeapi\pokeapi\target>java -jar pokeapi-0.0.1-SNAPSHOT.jar

  ____ _
 / ___| | | |
| |___| | | |
 \___|_|_|_|_|

:: Spring Boot ::
          (v2.7.14)

2023-08-16 13:40:26.771 INFO 26048 --- [main] com.pokeapi.PokeapiApplication : Starting PokeapiApp
lication v0.0.1-SNAPSHOT using Java 17.0.7 on LegionPro with PID 26048 (C:\Users\eyls_l\OneDrive\Documentos\Pruebatecnica
\PokeApiRest\pokeapi\pokeapi\target\pokeapi-0.0.1-SNAPSHOT.jar started by eys_l in C:\Users\eyls_l\OneDrive\Documentos\Pr
uebatecnica\PokeApiRest\pokeapi\pokeapi\target)
2023-08-16 13:40:26.775 INFO 26048 --- [main] com.pokeapi.PokeapiApplication : No active profile s
et, falling back to 1 default profile: "default"
2023-08-16 13:40:27.874 INFO 26048 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized
with port(s): 8090 (http)
2023-08-16 13:40:27.884 INFO 26048 --- [main] o.apache.catalina.core.StandardService : Starting service [T
omcat]
2023-08-16 13:40:27.885 INFO 26048 --- [main] org.apache.catalina.core.StandardEngine : Starting Servlet en
gine: [Apache Tomcat/9.0.78]
2023-08-16 13:40:27.984 INFO 26048 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring
embedded WebApplicationContext
2023-08-16 13:40:27.984 INFO 26048 --- [main] w.s.c.ServletWebServerApplicationContext : Root WebApplication
Context: initialization completed in 1158 ms
2023-08-16 13:40:28.489 INFO 26048 --- [main] o.s.b.a.e.web.EndpointLinksResolver : Exposing 1 endpoint
(s) beneath base path '/actuator'
2023-08-16 13:40:28.534 INFO 26048 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on p
ort(s): 8090 (http) with context path ''
2023-08-16 13:40:28.547 INFO 26048 --- [main] com.pokeapi.PokeapiApplication : Started PokeapiAppl
ication in 2.196 seconds (JVM running for 2.573)
```

Hacemos una prueba más en postman.

The screenshot shows a REST client interface with the following details:

- Request:** Method: GET, URL: `http://localhost:8090/api/pokemon?name=moltres`
- Params:** A table with one entry:

Key	Value
name	moltres
- Response:** Status: 200 OK, Time: 1000 ms, Size: 201.9 KB. The response body is in JSON format:


```

19   },
20   "base_experience": 290,
21   "forms": [
22     {
23       "name": "moltres",
24       "url": "https://pokeapi.co/api/v2/pokemon-form/146/"
25     }
26   ],

```

Y vemos que está funcionando.

Despliegue.

Para el despliegue este jar lo subiremos a una maquina virtual de aws para ello nos dirigimos a aws console.

<https://us-east-2.console.aws.amazon.com/>

Y creamos una nueva instancia en EC2

Asignamos nombre a la instancia.

The screenshot shows the 'Launch an instance' page in the AWS Management Console. The 'Name and tags' section is visible, with the 'Name' field containing the text 'POKEAPI'. The page includes instructions on how to create virtual machines on the AWS Cloud.

Creamos su KEY para poder entrar a ella desde SSH

Create key pair ✕

Key pair name

Key pairs allow you to connect to your instance securely.

The name can include upto 255 ASCII characters. It can't include leading or trailing spaces.

Key pair type


☒ **RSA**
 RSA encrypted private and public key pair

☐ **ED25519**
 ED25519 encrypted private and public key pair

Private key file format

☒ **.pem**
 For use with OpenSSH

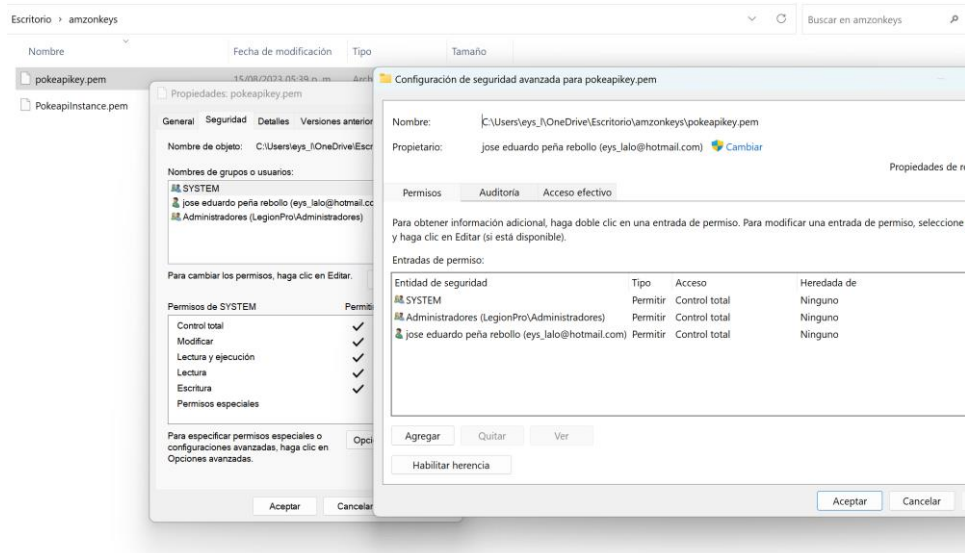
☐ **.ppk**
 For use with PuTTY

 When prompted, store the private key in a secure and accessible location on

Cancel Create key pair

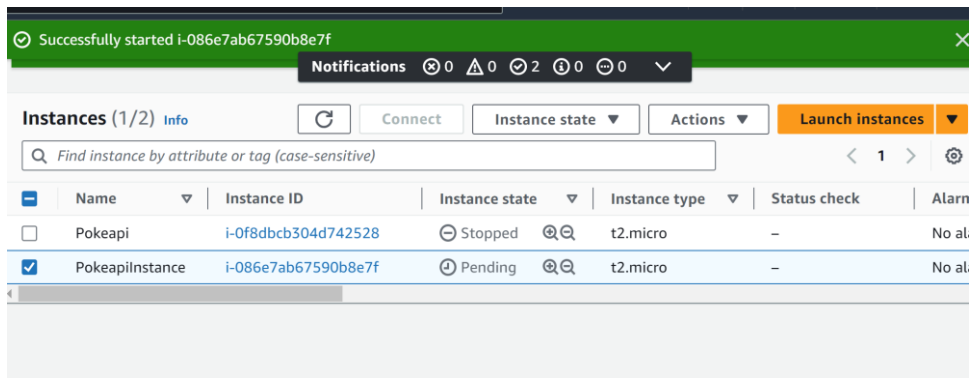
Guardamos la key y es importante hacer esta key no pública.

Para ellos vamos al archivo propiedades, seguridad avanza y quitamos herencia.

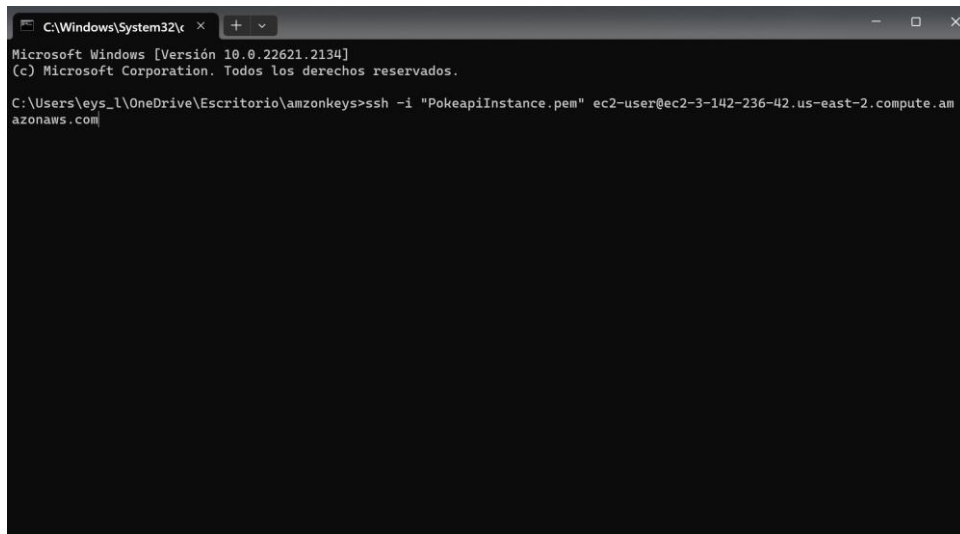


Es importante esta parte para conectarnos a nuestra maquina linux/ubuntu.

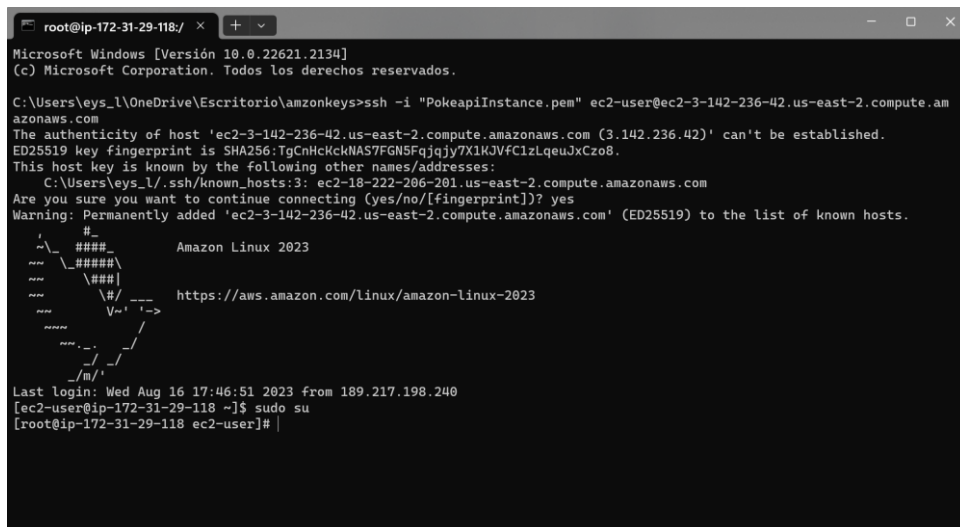
Una vez hecho eso lanzamos nuestra instancia.



Y accedemos a la misma por medio de SSH en la línea de comandos



Copiamos la dirección de SSH que te arroja el sistema y entramos.



una vez entrando comprobamos que tengamos git para poder pasar el JAR desde línea de comandos.

Es importante también instalar java open jdk 17

```
root@ip-172-31-29-118: ~  
bisect      Use binary search to find the commit that introduced a bug  
diff        Show changes between commits, commit and working tree, etc  
grep        Print lines matching a pattern  
log         Show commit logs  
show        Show various types of objects  
status      Show the working tree status  
  
grow, mark and tweak your common history  
branch      List, create, or delete branches  
commit      Record changes to the repository  
merge       Join two or more development histories together  
rebase      Reapply commits on top of another base tip  
reset       Reset current HEAD to the specified state  
switch      Switch branches  
tag         Create, list, delete or verify a tag object signed with GPG  
  
collaborate (see also: git help workflows)  
fetch       Download objects and refs from another repository  
pull        Fetch from and integrate with another repository or a local branch  
push        Update remote refs along with associated objects  
  
'git help -a' and 'git help -g' list available subcommands and some  
concept guides. See 'git help <command>' or 'git help <concept>'  
to read about a specific subcommand or concept.  
See 'git help git' for an overview of the system.  
[root@ip-172-31-29-118 ec2-user]# java -version  
java version "17.0.8" 2023-07-18 LTS  
Java(TM) SE Runtime Environment (build 17.0.8+9-LTS-211)  
Java HotSpot(TM) 64-Bit Server VM (build 17.0.8+9-LTS-211, mixed mode, sharing)  
[root@ip-172-31-29-118 ec2-user]#
```

Una vez descargado el clone de git e instalado java lo inicializamos desde consola con el comando `java -jar` y el nombre del jar.

```
root@ip-172-31-29-118:/ +  
pokeapideploy  
[root@ip-172-31-29-118 api]# cd pokeapideploy/  
[root@ip-172-31-29-118 pokeapideploy]# ls  
pokeapi-0.0.1-SNAPSHOT.jar  
[root@ip-172-31-29-118 pokeapideploy]# java -jar pokeapi-0.0.1-SNAPSHOT.jar  
  
      ____      _      _      _      _      _      _      _      _      _      _      _      _      _      _  
     / ___ \    / \    / \    / \    / \    / \    / \    / \    / \    / \    / \    / \    / \    / \    / \  
    / /___) \  /_/ \  /_/ \  /_/ \  /_/ \  /_/ \  /_/ \  /_/ \  /_/ \  /_/ \  /_/ \  /_/ \  /_/ \  /_/ \  /_\  
   /_____\__/ /___\ /___\ /___\ /___\ /___\ /___\ /___\ /___\ /___\ /___\ /___\ /___\ /___\ /___\ /___\ /___\  
===== (C) 2016 by the PokeAPI team. All rights reserved. =====  
          (v2.7.14)  
-- Spring Boot --  
  
2023-08-16 19:50:59.135 INFO 2542 --- [main] com.pokeapi.PokeapiApplication : Starting PokeapiApplication v0.0.1-SNAPSHOT using Java 1  
7.0.8 on ip-172-31-29-118.us-east-2.compute.internal with PID 2542 (/var/pokeapi/api/pokeapideploy/pokeapi-0.0.1-SNAPSHOT.jar started by root in /var/pokeap  
i/api/pokeapideploy)  
2023-08-16 19:50:59.150 INFO 2542 --- [main] com.pokeapi.PokeapiApplication : No active profile set, falling back to 1 default profile  
: "default"  
2023-08-16 19:51:02.757 INFO 2542 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8090 (http)  
2023-08-16 19:51:02.786 INFO 2542 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]  
2023-08-16 19:51:02.787 INFO 2542 --- [main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.78]  
2023-08-16 19:51:03.106 INFO 2542 --- [main] o.a.e.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext  
2023-08-16 19:51:03.107 INFO 2542 --- [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in  
3782 ms  
2023-08-16 19:51:05.065 INFO 2542 --- [main] o.s.b.a.e.web.EndpointLinksResolver : Exposing 1 endpoint(s) beneath base path '/actuator'  
2023-08-16 19:51:05.282 INFO 2542 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8090 (http) with context path  
''  
2023-08-16 19:51:05.240 INFO 2542 --- [main] com.pokeapi.PokeapiApplication : Started PokeapiApplication in 7.351 seconds (JVM running  
for 8.892)
```

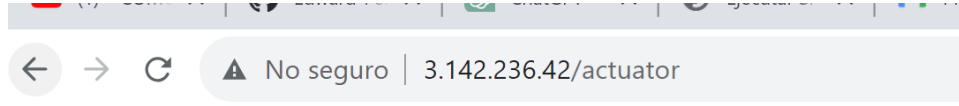
Si todo sale bien lanzamos nuestro servicio de spring y lo tenemos listo para usar.

Nos conectamos a la url que nos arroja aws 3.142.236.42 y ahora podremos usar esa dirección.

← → ↻ ⚠ No seguro | 3.142.236.42

It works!

Hasta aquí termine la practica no pude resolver el porque la direccion me arroja un



Not Found

The requested URL was not found on this server.

DOCKER

En todo caso tenemos DOCKER

```
.\Users\keys_l\OneDrive\Documentos\Pruebatecnica\PokeApiRest\pokeapi\pokeapi>docker build -t "pokeapi-docker" .
[+] Building 9.2s (4/6)
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 163B
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load metadata for docker.io/library/openjdk:8-alpine
=> [internal] load build context
=> => transferring context: 19.97MB
=> [1/2] FROM docker.io/library/openjdk:8-alpine@sha256:94792824df2df33402f201713f932b58cb9de94a0cd524164a0f2283
=> => resolve docker.io/library/openjdk:8-alpine@sha256:94792824df2df33402f201713f932b58cb9de94a0cd524164a0f2283
=> sha256:c2274a1a8e2786ee9101b08f7611f9ab8019e368dce1e325d3c284a0ca33397 10.49MB / 70.73MB
=> sha256:94792824df2df33402f201713f932b58cb9de94a0cd524164a0f2283343547b3 1.64kB / 1.64kB
=> sha256:44b3cea369c947527e266275cee85c71a81f20fc5076f6ebb5a13f19015dce71 947B / 947B
=> sha256:a3562aa0b991a88cfe8172847c8be6dbf6e46340b759c2b782f8b8be45342717 3.40kB / 3.40kB
=> sha256:e7c96db7181be991f19a9fb6975cd8bd73c65f4a2681348e63a141a2192a5f10 2.76MB / 2.76MB
=> sha256:f910a506b6c1dbec766725d70356f695ae2bf2bea6224d8e8c7c6ad4f3664a2 238B / 238B
=> extracting sha256:e7c96db7181be991f19a9fb6975cd8bd73c65f4a2681348e63a141a2192a5f10
=> extracting sha256:f910a506b6c1dbec766725d70356f695ae2bf2bea6224d8e8c7c6ad4f3664a2
```

Creamos nuestro build con docker build -t "y la etiqueta". Es importante el punto final.

(cambiar la version a java17)

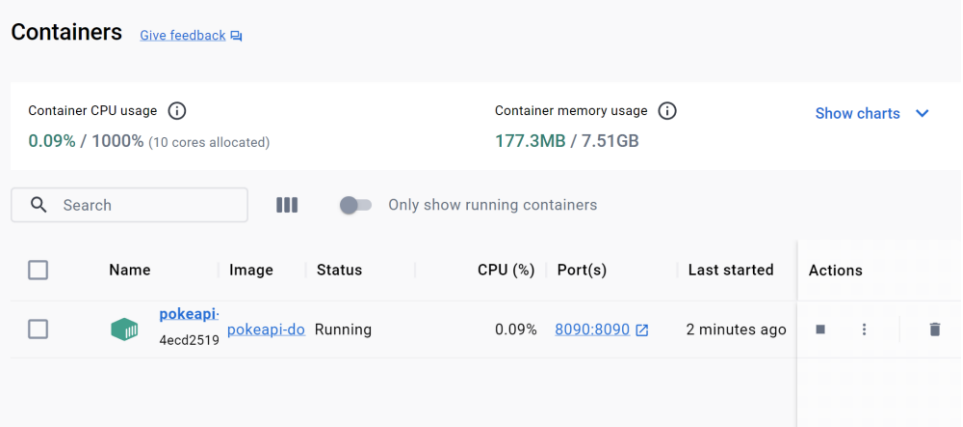
```
C:\Users\keys_l\OneDrive\Documentos\Pruebatecnica\PokeApiRest\pokeapi\pokeapi>docker run --name pokeapi-springboot-docker -p 8090:8090 pokeapi-docker:latest

:: Spring Boot ::
2023-08-16 20:17:06.157 INFO 1 --- [main] com.pokeapi.PokeapiApplication : Starting PokeapiApplication v0.0.1-SNAPSHOT using Java 17.0
2023-08-16 20:17:06.160 INFO 1 --- [main] com.pokeapi.PokeapiApplication : No active profile set, falling back to 1 default profile: "
Default"
2023-08-16 20:17:06.988 INFO 1 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8090 (http)
2023-08-16 20:17:06.997 INFO 1 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2023-08-16 20:17:06.997 INFO 1 --- [main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.78]
2023-08-16 20:17:07.054 INFO 1 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2023-08-16 20:17:07.055 INFO 1 --- [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 848
ms
2023-08-16 20:17:07.067 INFO 1 --- [main] o.s.b.a.e.web.EndpointLinksResolver : Exposing 1 endpoint(s) beneath base path '/actuador'
2023-08-16 20:17:07.096 INFO 1 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8090 (http) with context path ''
2023-08-16 20:17:07.508 INFO 1 --- [main] com.pokeapi.PokeapiApplication : Started PokeapiApplication in 1.641 seconds (JVM running fo
r 1.935s)
```

Y corremos nuestro proyecto con.



docker run --name pokeapi-springboot-docker -p 8090:8090 pokeapi-docker:latest

O en todo caso desde docker desktop



Hacemos pruebas de que funciona.

GET https://pokeapi.co/api/v2/ GET http://localhost:8090/api/ GET localhost:8090/actuador/ GET 18.222.206.201/localhost + ...

 <http://localhost:8090/api/pokemon?name=zapdos>  Save



GET Send

Params Auth Headers (6) Body Pre-req. Tests Settings Cookies

Query Params

	Key	Value	Bulk Edit
<input checked="" type="checkbox"/>	name	zapdos	
	Key	Value	

Body Cookies Headers (5) Test Results 200 OK 824 ms 207.35 KB Save Response ▾

Pretty Raw Preview Visualize JSON  

```
19 ],
20   "base_experience": 290,
21   "forms": [
22     {
23       "name": "zapdos",
24       "url": "https://pokeapi.co/api/v2/pokemon-form/145/"
25     }
26   ],
27   "flavor_text_entries": [
```

Y listo.

En todo caso los problemas que me llegaron a surgir los resolví con <https://chat.openai.com/>

La documentación en Swagger no la pude documentar debido a un error poco común que no tuve tiempo de resolver.

La url del proyecto de git se encuentra en: <https://github.com/Edward-Pena/pokeapi> o en su caso el repositorio <https://github.com/Edward-Pena/pokeapi.git>

Conclusión

Nuestro microservicio nos permite consultar un pokemon para obtener todas sus estadísticas hasta el 2021. Utilizando el nombre del pokemon.

Ejemplo Articuno entraría como:

<http://localhost:8090/api/pokemon?name=articuno> y la respuesta será en formato JSON para que pueda ser manipulada después si es requerido.

Aunque es un microservicio pequeño nos da una idea cómo funciona.