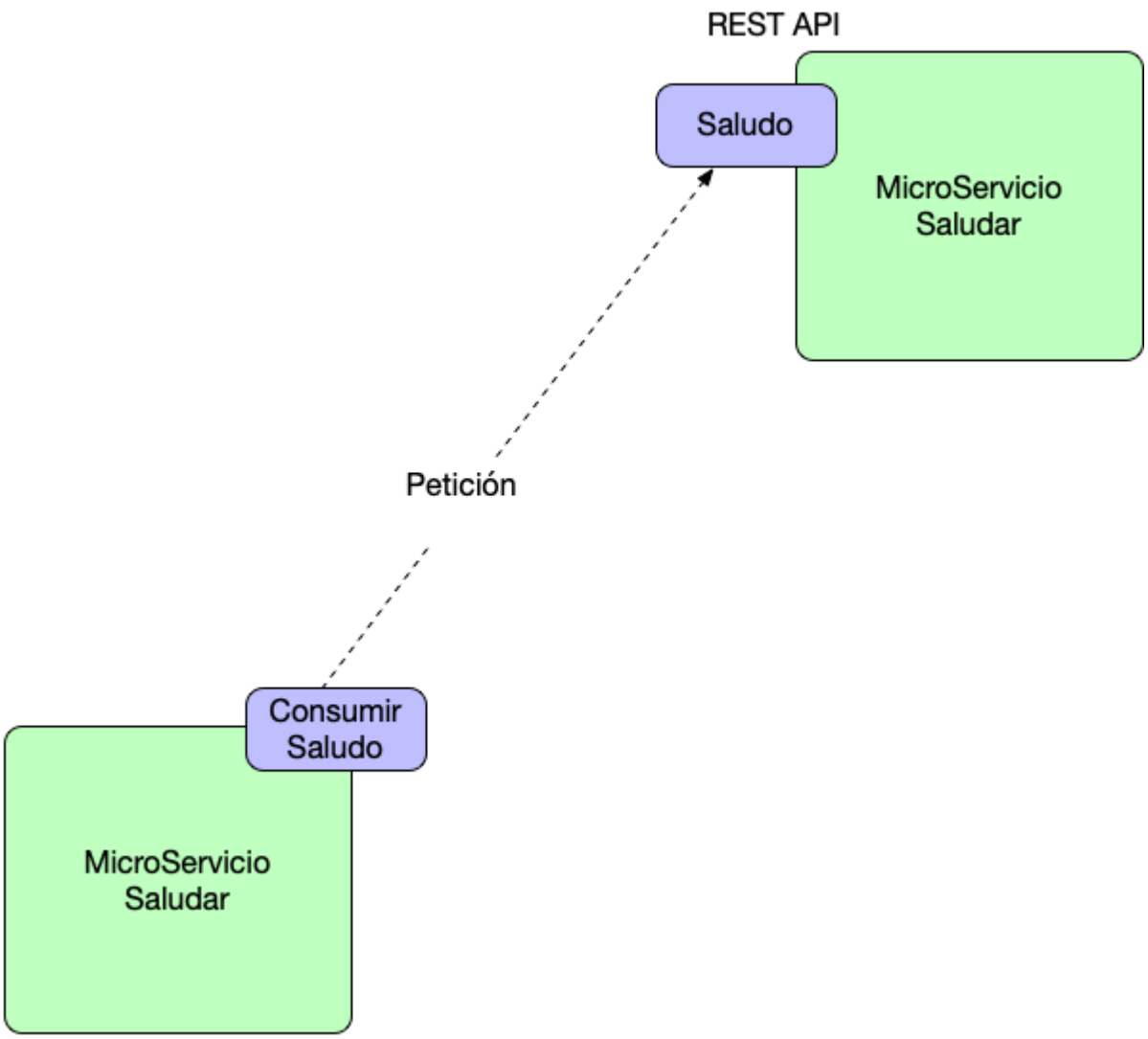
Tabla de Contenidos

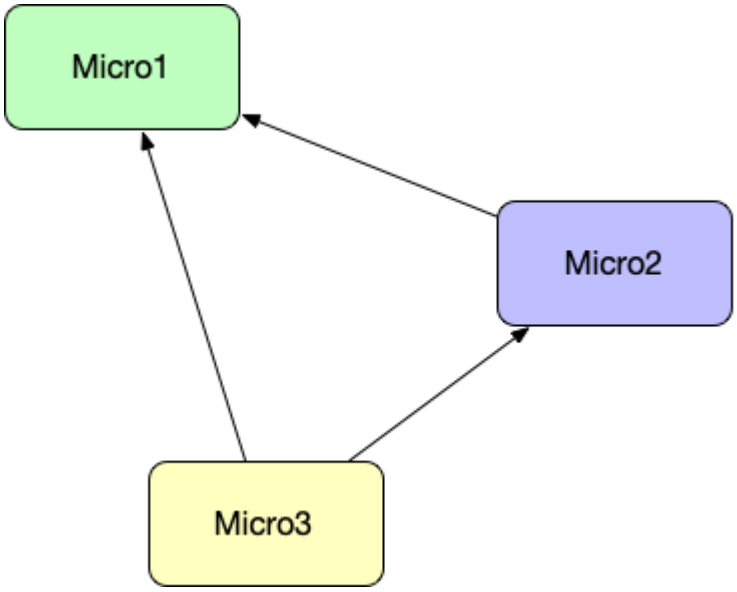

- ¿Qué es un Microservicio?
- Docker y su relación
- Ventajas de los Microservicios con Spring Boot
- Ejemplo Práctico
- Micro 1: Publicar un Saludo
- Paso 2: Creamos el Controlador
- Paso 3: Clase Principal del MicroServicio
  - Micro 2: Consumir el Saludo
    - Paso 1: Añadimos las dependencias en pom.xml
    - Paso 2: Configuramos el RestTemplate
    - Paso 3: Creamos el Controlador
    - Paso 4: Clase Principal del Segundo:
  - Ejecutando
  - Conclusión


MicroServicio ¿Qué es? : Hoy en día, los Microservicios son uno de los enfoques más populares para construir aplicaciones escalables y fáciles de mantener. Si estás en el mundo del desarrollo en Java, seguramente habrás escuchado hablar de Spring Boot como una de las mejores herramientas para crear Microservicios. En este artículo, veremos de forma sencilla qué son los Microservicios y cómo podemos construir dos de ellos usando Spring Boot. Uno publicará un saludo, y otro lo consumirá usando RestTemplate haciendo del rol de cliente.



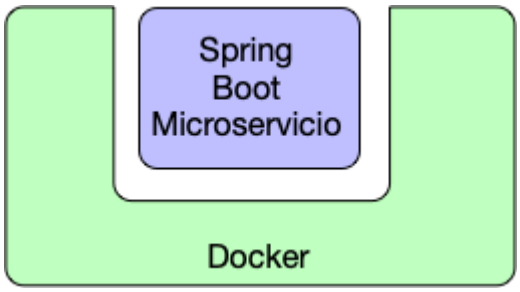

# ¿Qué es un Microservicio?

Un microservicio es simplemente una pequeña aplicación que se encarga de hacer una única tarea. A diferencia de una aplicación monolítica, donde todo está junto en un único bloque, los microservicios dividen esa gran aplicación en pequeñas partes que funcionan de manera independiente. Cada microservicio tiene su propio ciclo de vida: puedes desplegar, actualizar y escalar uno sin que afecte a los demás.




## Docker y su relación

Docker y los Microservicios se complementan porque estos dividen una aplicación en partes más pequeñas e independientes, mientras que Docker facilita ejecutar cada una de esas partes (microservicios) en contenedores aislados. Esto simplifica el despliegue, la gestión y la escalabilidad de cada uno, ya que los contenedores son ligeros, portátiles y aseguran que cada servicio tenga su propio entorno de ejecución sin conflictos con otros servicios.




## Ventajas de los Microservicios con Spring Boot

Spring Boot hace que la creación de microservicios sea rápida y sencilla gracias a su enfoque minimalista y su integración con el ecosistema de Spring. Algunas de las ventajas clave de usar Spring Boot para microservicios son:

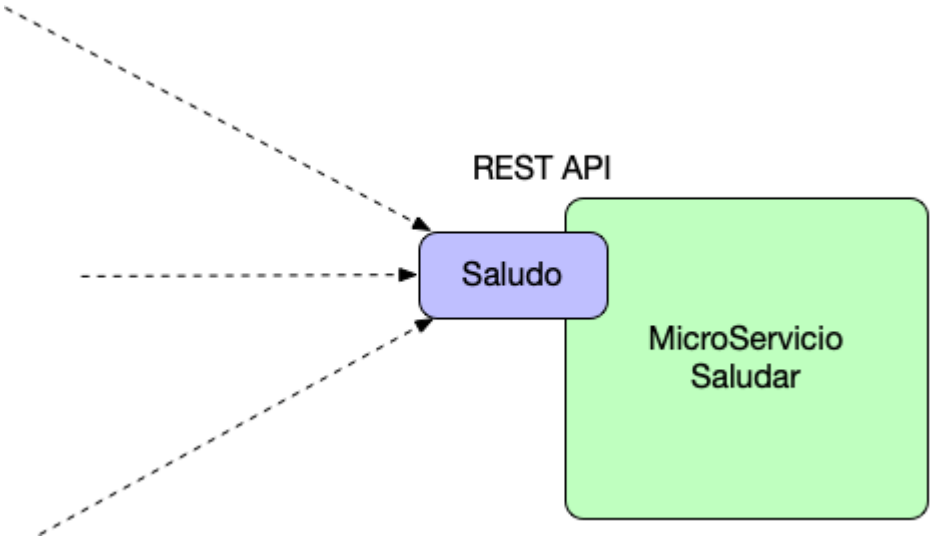
1. Desarrollo rápido: Menos configuración, más código funcional.
2. Despliegue independiente: Cada microservicio es un servicio independiente que puedes desplegar sin tocar los demás.
3. Escalabilidad: Puedes escalar solo los microservicios que lo necesiten.

## Ejemplo Práctico

Vamos a crear dos Micros muy simples. El primer Micro va a exponer un saludo, y el segundo va a consumir ese saludo y mostrárnoslo. Vamos a usar RestTemplate para que el segundo consuma el API del primero.

## Micro 1: Publicar un Saludo

Este primeo es responsable de devolver un mensaje de saludo cuando le hacemos una petición HTTP.




Paso 1: Añadimos las dependencias en pom.xml partiendo de un proyecto de Spring Boot Initializer

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
```

```
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
</dependencies>
```

# Paso 2: Creamos el Controlador

Este controlador será el encargado de manejar la petición y devolver el saludo.

```
package com.example.microservicio1;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class SaludoController {

    @GetMapping("/saludo")
    public String saludar() {
        return "Hola desde el primer microservicio";
    }
}
```

Cuando alguien haga una solicitud a http://localhost:8080/saludo, este controlador responderá con el mensaje: “Hola desde el primer microservicio”.

# Paso 3: Clase Principal del MicroServicio

Esta es la clase principal que inicia el micro

```
package com.example.microservicio1;

import org.springframework.boot.SpringApplication;
```

```
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Micro1Application {

    public static void main(String[] args) {
        SpringApplication.run(Micro1Application.class, args);
    }
}
```

## Micro 2: Consumir el Saludo

Este segundo micro va a usar RestTemplate para hacer una petición al primer micro y mostrar el saludo que recibe.

### Paso 1: Añadimos las dependencias en `pom.xml`

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
</dependencies>
```

### Paso 2: Configuramos el RestTemplate

RestTemplate es la herramienta que vamos a usar para consumir el API del primer micro.

```
package com.example.microservicio2;

import org.springframework.boot.web.client.RestTemplateBuilder;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
```

```
import org.springframework.web.client.RestTemplate;

@Configuration
public class RestTemplateConfig {

    @Bean
    public RestTemplate restTemplate(RestTemplateBuilder builder) {
        return builder.build();
    }
}
```

## Paso 3: Creamos el Controlador

Este controlador será el encargado de consumir el saludo del primero y devolverlo al usuario.




```
package com.example.microservicio2;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.client.RestTemplate;
```

```
@RestController
public class ConsumidorController {

    @Autowired
    private RestTemplate restTemplate;

    @GetMapping("/consumir-saludo")
    public String consumirSaludo() {
        String saludo =
restTemplate.getForObject("http://localhost:8080/saludo",
String.class);
        return "El saludo recibido es: " + saludo;
    }
}
```

En este caso, cuando hagamos una solicitud a  http://localhost:8081/consumir-saludo, este microservicio va a hacer una petición al primero en http://localhost:8080/saludo y nos devolverá el saludo que recibe.

## Paso 4: Clase Principal del Segundo:

```
package com.example.microservicio2;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Micro2Application {

    public static void main(String[] args) {
        SpringApplication.run(Microservicio2Application.class, args);
    }
```

```
}
```

## Ejecutando

1. Primer Micro: Inicia el primer microservicio en el puerto 8080 ejecutando la clase Microservicio1Application.
2. Segundo Micro: Inicia el segundo microservicio en un puerto diferente, como 8081, ejecutando la clase Microservicio2Application.
   Cuando visites http://localhost:8081/consumir-saludo, verás el saludo: “El saludo recibido es: Hola desde el primer microservicio”.

## Conclusión

Este ejemplo muestra lo sencillo que es construir MicroService en Java usando Spring Boot. Con herramientas como RestTemplate, puedes hacer que los MicroServices se comuniquen entre sí de forma fácil y rápida. Este enfoque te permitirá tener aplicaciones más modulares y fáciles de mantener a largo plazo.

- Spring REST Client con RestTemplates
- Curso Spring Data y buenas prácticas
- Curso Spring Boot y Micro
- Spring Cloud Eureka Server y Discovery