Icono *Training Consulting*

Formación

MOD-111 Programación Reactiva con Spring WebFlux

2025

VIEWNEXT
AN IBM SUBSIDIARY

Formador

Ana Isabel Vegas

INGENIERA INFORMÁTICA con Master Máster Universitario en Gestión y Análisis de Grandes Volúmenes de Datos: Big Data, tiene la certificación PCEP en Lenguaje de Programación Python y la certificación JSE en Javascript. Además de las certificaciones SCJP Sun Certified Programmer for the Java 2 Platform Standard Edition, SCWD Sun Certified Web Component Developer for J2EE 5, SCBCD Sun Certified Business Component Developer for J2EE 5, SCEA Sun Certified Enterprise Architect for J2EE 5.

Desarrolladora de Aplicaciones FULLSTACK, se dedica desde hace + de 20 años a la CONSULTORÍA y FORMACIÓN en tecnologías del área de DESARROLLO y PROGRAMACIÓN.

training@iconotc.com

MOD-111 Programación Reactiva con Spring WebFlux

❑ **Duración:** 15 horas

❑ **Modalidad:** Presencial / Remoto

❑ **Fechas/Horario:**

- Días 24, 26 y 27 Marzo 2025
- Horario de 15:00 a 20:00 hs

❑ **Contenidos:**

- Programación Reactiva
  - Orientación a eventos
  - Programación asíncrona
  - Patrones Observable, Iterator, ...
  - Publicador/Suscriptor
- Reactive Streams
  - Componentes Flow API
  - Implementaciones
  - Interoperabilidad
  - Streams y Reactive Streams
- Reactor Project
  - Reactivo no bloqueante con contrapresitón
  - Flux y Mono
  - Creación de observables y suscripciones
  - Hot Versus Cold
  - Multiple Subscribers
- Spring WebFlux
  - Reactive API Rest
  - Controladores anotados
  - Puntos finales funcionales
  - WebClient

Programación Reactiva

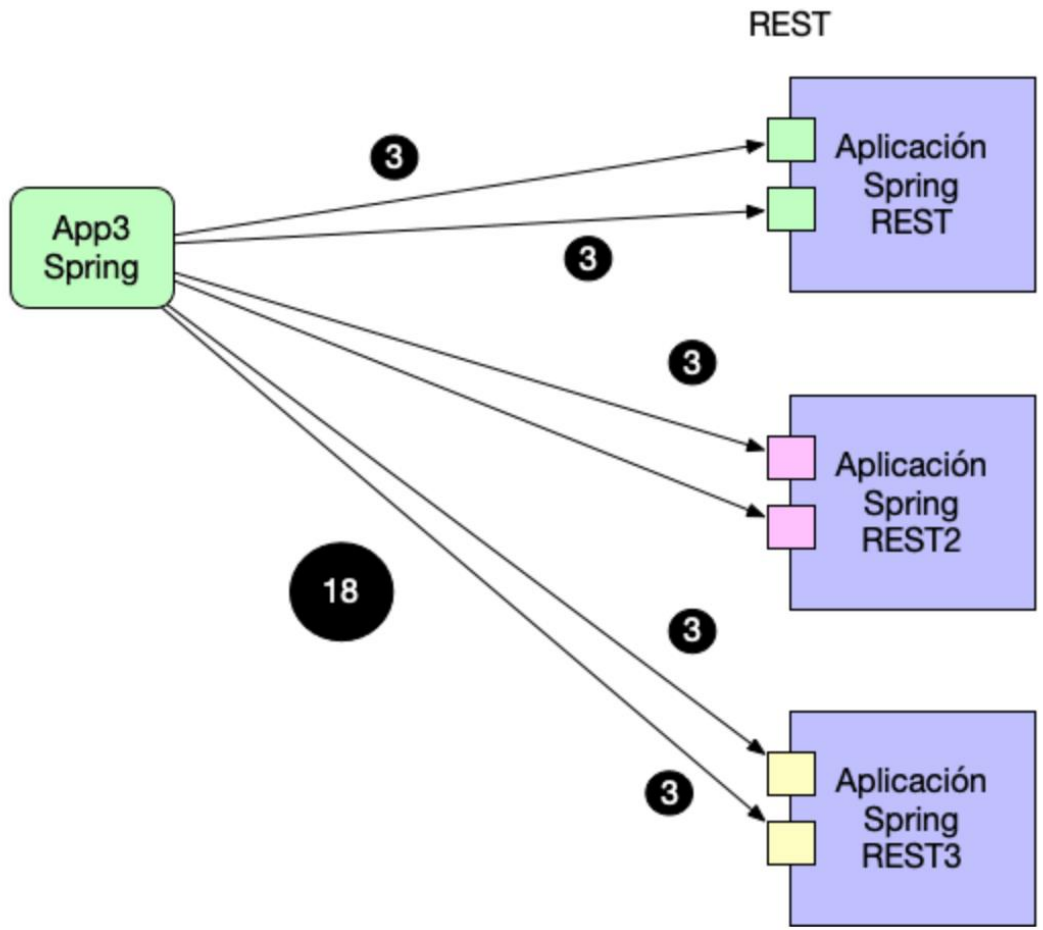
Tema 1

Programación Reactiva

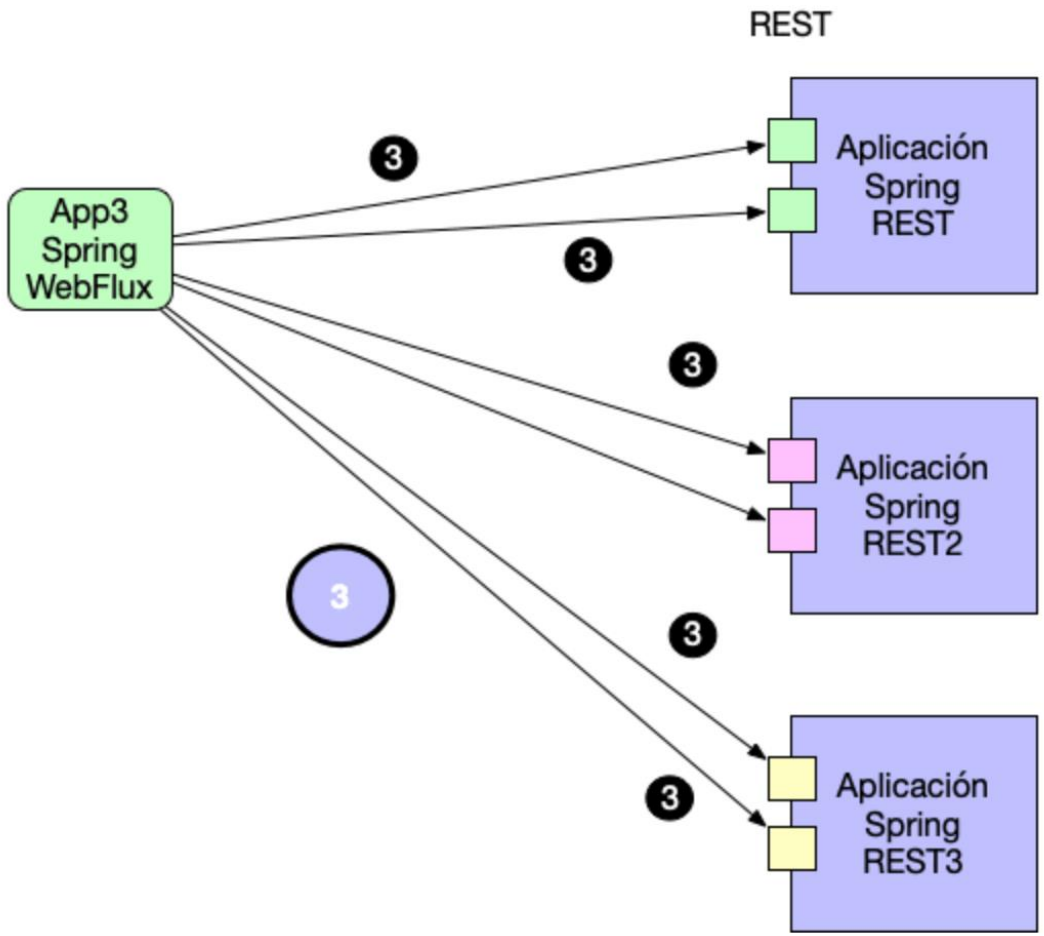
- La programación reactiva es un paradigma enfocado en el trabajo con flujos de datos finitos o infinitos de manera asíncrona.

- La motivación detrás de este nuevo paradigma procede de la necesidad de responder a las limitaciones de escalado presentes en los modelos de desarrollo actuales, que se caracterizan por su desaprovechamiento del uso de la CPU debido al I/O, el sobreuso de memoria (enormes thread pools) y la ineficiencia de las interacciones bloqueantes.

Operación bloqueante



Operación NO bloqueante



Evolución de la programación Reactiva

- Patrón Observer del Gang of Four. Tienen los inconvenientes de su simplicidad y falta de opciones de composición.

- RxJava. Evolucionó el concepto de Observable/Observer incorporando la composición de operaciones, pero presentaba limitaciones arquitectónicas.

- RxJava 2.x, Project Reactor y Akka Streams.

Reactive Streams

Tema 2

Reactive Streams

- El uso de Reactive Streams es similar al del patrón Iterator (incluyendo Java 8 Streams) con una clara diferencia, el primero es push-based mientras que el segundo es pull-based:

Evento	Iterable (push)	Reactive (pull)
Obtener dato	next()	onNext(Object data)
Error	throws Exception	onError(Exception)
Fin	!hasNext()	onComplete()

- Iterable delega en el desarrollador las llamadas para obtener los siguientes elementos. Por contra, los Publisher de Reactive Streams son los encargados de notificar al Subscriber la llegada de nuevos elementos de la secuencia.

Mejoras

- Composición y legibilidad: Mejoran la comprensión del código y la composición de operaciones.
- Operadores: permiten aplicar transformaciones sobre los flujos de datos.
- Backpressure (contrapresion): debido al flujo push-based, se pueden dar situaciones donde un Publisher genere más elementos de los que un Subscriber puede consumir. Para evitarlo se han establecido los siguientes mecanismos:
    - Los Subscriber pueden indicar el número de datos que quieren o pueden procesar mediante la operación subscriber.request(n), de manera que el Publisher nunca les enviará más de n elementos.
    - Los Publisher pueden aplicar diferentes operaciones para evitar saturar a los subscriptores lentos (buffers, descarte de datos, etc.).

Reactor Project

Tema 3

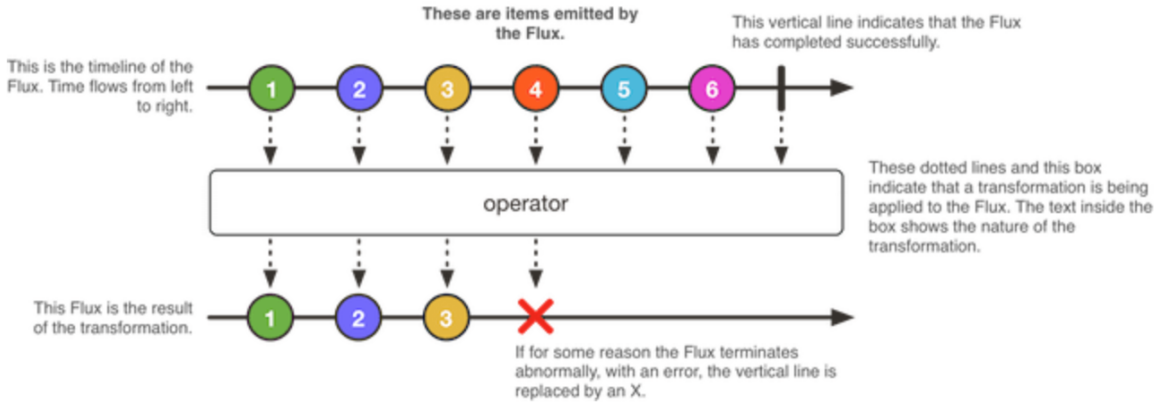
Que es Reactor?

- Fue concebido con la implicación del equipo responsable de RxJava 2 por lo que comparten gran parte de la base arquitectónica. Su principal ventaja es que al ser parte de Pivotal ha sido la elegida como fundación del futuro Spring 5 WebFlux Framework.

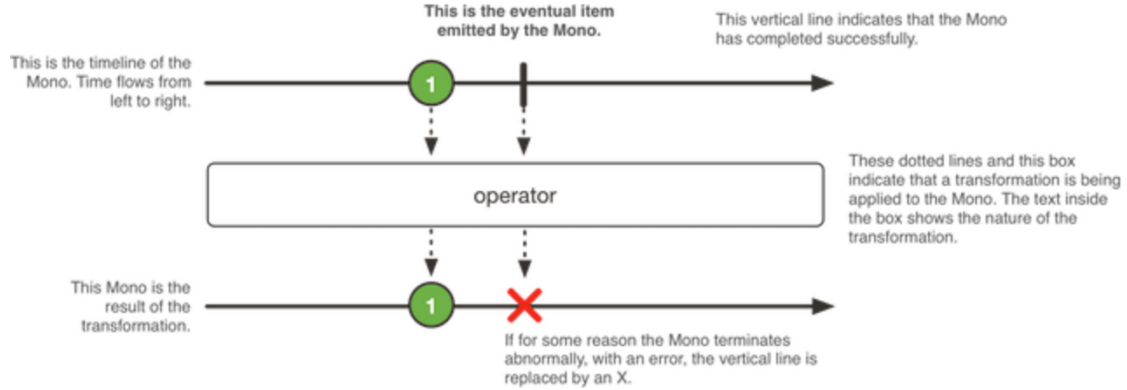
- Este API introduce los tipos Flux y Mono como implementaciones de Publisher, los cuales generan series de 0…N y 0…1 elementos respectivamente.
    - Flux: Flux hace referencia a un conjunto de elementos .
    - Mono: Mono hace referencia a un elemento de programación asincrona a uno solo

Flux vs Mono

- Flux:



- Mono:



Dependencia

```
<dependency>
    <groupId>io.projectreactor</groupId>
    <artifactId>reactor-core</artifactId>
</dependency>
```

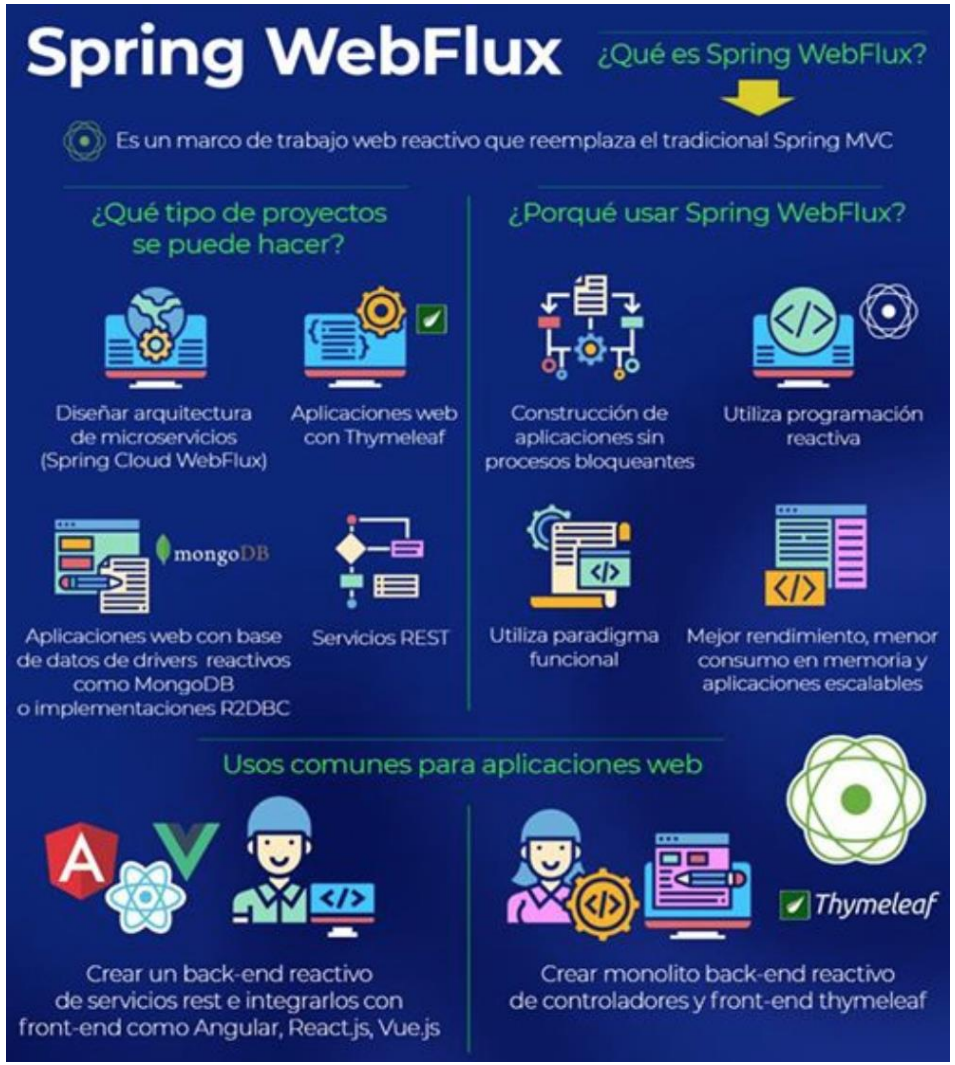
Práctica

- Crear flujos
- Manejo de excepciones
- Convertir Flux a Mono
- Combinar flujos
- Operadores map, filter, flatMap, range
- Intervalos infinitos
- Manejando contrapresión
- …..

Spring WebFlux

Tema 4

WebFlux


Spring WebFlux
¿Qué es Spring WebFlux?
Es un marco de trabajo web reactivo que reemplaza el tradicional Spring MVC
¿Qué tipo de proyectos se puede hacer?
Diseñar arquitectura de microservicios (Spring Cloud WebFlux)
Aplicaciones web con Thymeleaf
mongoDB
Aplicaciones web con base de datos de drivers reactivos como MongoDB o implementaciones R2DBC
Servicios REST
¿Porqué usar Spring WebFlux?
Construcción de aplicaciones sin procesos bloqueantes
Utiliza programación reactiva
Utiliza paradigma funcional
Mejor rendimiento, menor consumo en memoria y aplicaciones escalables
Usos comunes para aplicaciones web
Crear un back-end reactivo de servicios rest e integrarlos con front-end como Angular, React.js, Vue.js
Thymeleaf
Crear monolito back-end reactivo de controladores y front-end thymeleaf

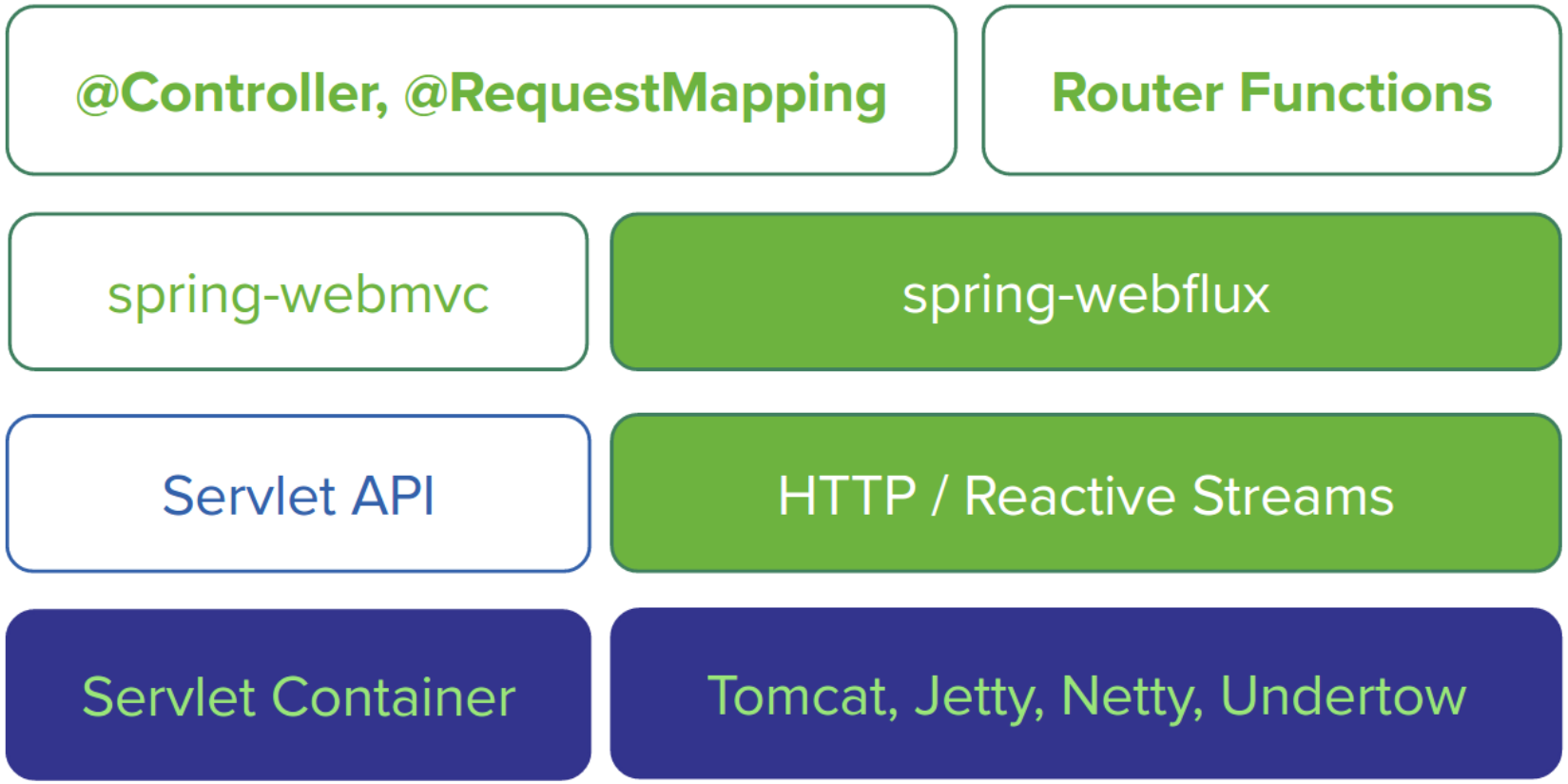
Que es Spring WebFlux?

- Desde la versión 5.x de Spring se introduce Spring Webflux como marco de trabajo (framework) de aplicaciones web reactivas.

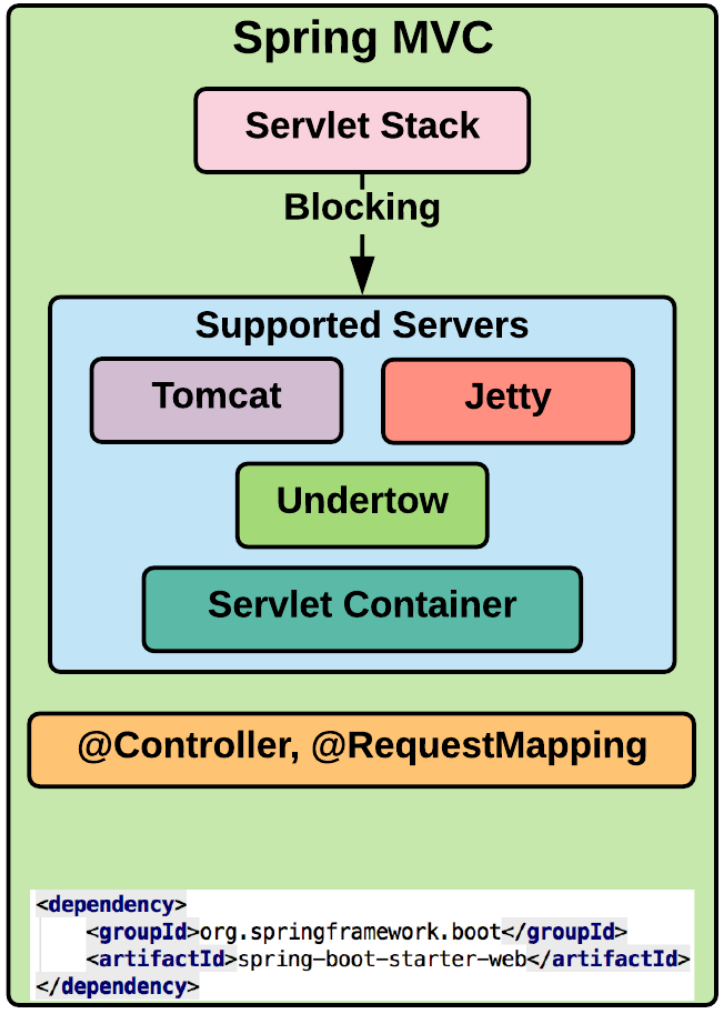
- Esta basado en reactive streams (http://www.reactive-streams.org/) y corre en servidores web como: Netty, Undertow, u contenedores que soporte la especificacion de Servlets 3.1 o superior.

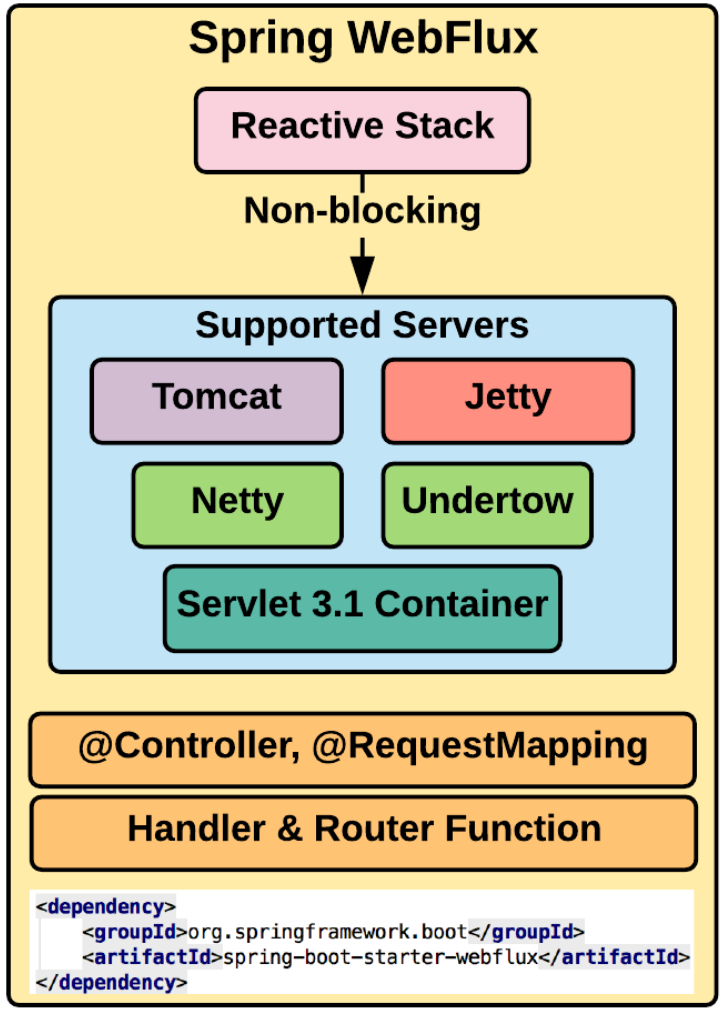
- Spring Webflux usa el proyecto Reactor como base para la implementación de reactive stream

Arquitectura del Framework

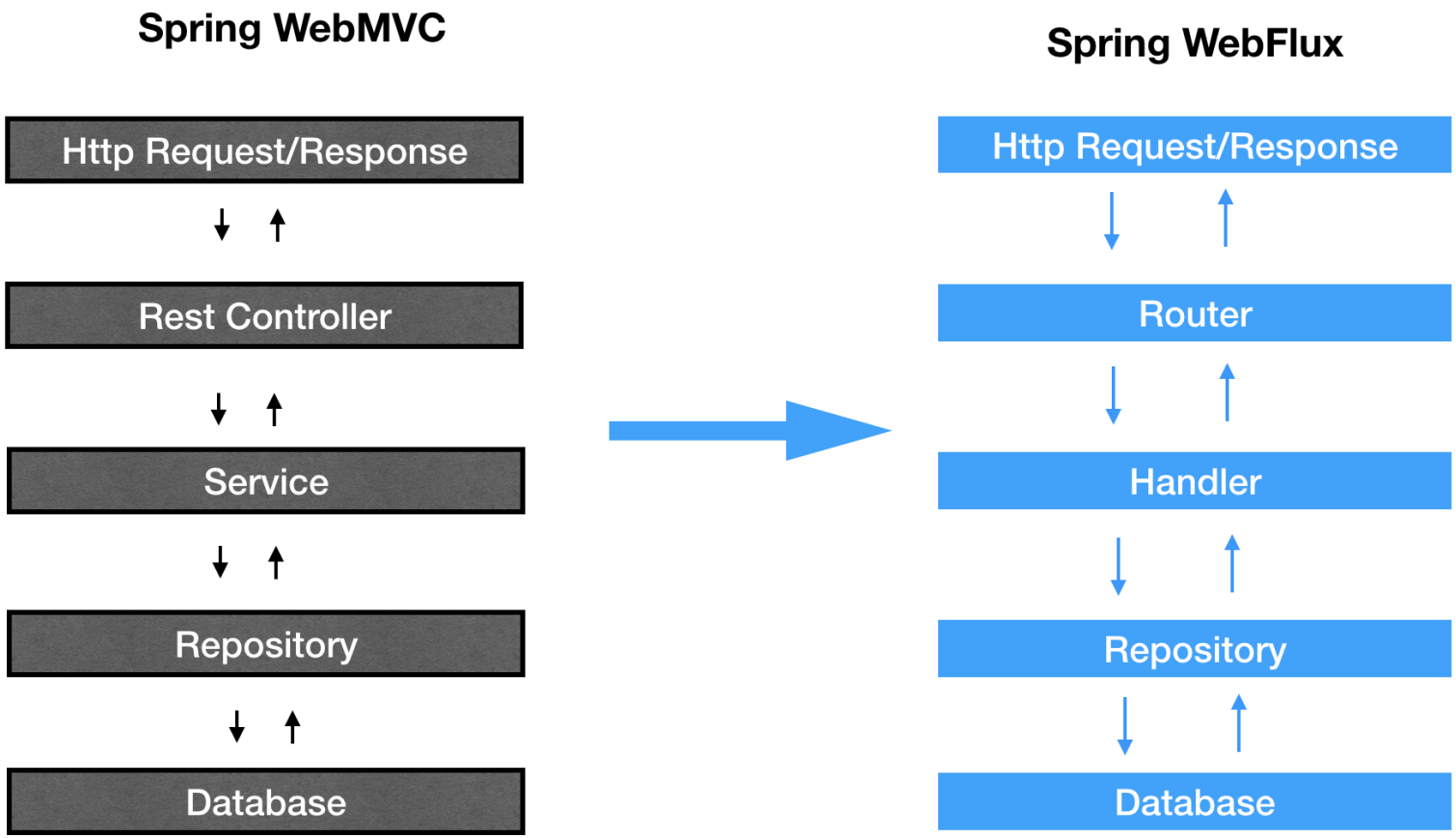


Spring MVC vs Spring WebFlux





Componentes Spring WebFlux



Dependencia

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

Práctica

- Crear API Rest utilizando WebFlux.
- Exponer un EndPoint reactivo.

Acceso a datos

- Actualmente tenemos dos tipos de Bases de Datos:
  - Relacionales, basadas en SQL
  - No relacionales, que se consideran almacenes JSON.

- Las bases de datos no relacionales tienen soporte para la programación reactiva, pero las bases de datos relacionales no tienen soporte por lo tendremos que aplicar cualquiera de estas soluciones:
  - Convertir los resultados de las queries a Flux o Mono
  - Utilizar R2DBC

Acceso a datos con MongoDB

- MongoDB, a pesar de ser una base de datos relativamente joven (su desarrollo empezó en octubre de 2007) se ha convertido en todo un referente a la hora de usar bases de datos NoSQL y está listo para entornos de producción ágiles, de alto rendimiento y con gran carga de trabajo.

- En lugar de guardar los datos en tablas como se hace en las base de datos relacionales con estructuras fijas, las bases de datos NoSQL, como MongoDB, guarda estructuras de datos en documentos con formato JSON y con un esquema dinámico (MongoDB llama ese formato BSON).

- Ejemplo de documento almacenado en MongoDB:

```
{
    "_id": ObjectId("4efa8d2b7d284dad101e4bc7"),
    "Last Name": "PELLERIN",
    "First Name": "Franck",
    "Age": 29,
    "Address": {
        "Street": "1 chemin des Loges",
        "City": "VERSAILLES"
    }
}
```

Acceso a datos con MongoDB

- Para descargar MongoDB debemos irnos a su pagina de descargas: https://www.mongodb.com/download-center/community donde encontrareis la versión adecuada a vuestra plataforma.

- Una vez descargados los binarios de MongoDB para Windows, se extrae el contenido del fichero descargado (ubicado normalmente en el directorio de descargas) en C:\.

- Renombra la carpeta a mongodb: C:\mongodb
-
- MongoDB es autónomo y no tiene ninguna dependencia del sistema por lo que se puede usar cualquier carpeta que elijas. La ubicación predeterminada del directorio de datos para Windows es “C:\data\db". Crea esta carpeta.

- Para iniciar MongoDB, ejecutar desde la Línea de comandos
-

```
C:\mongodb\bin\mongod.exe
```

- Esto iniciará el proceso principal de MongoDB. El mensaje “waiting for connections" indica que el proceso mongod.exe se está ejecutando con éxito.

Acceso a datos con MongoDB

- Dependencia necesaria:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-mongodb-reactive</artifactId>
</dependency>
```

```java
public class Producto {

    @Id
    private String id;

    private String descripcion;
    private double precio;

    public String getDescripcion() {
        return descripcion;
    }

    public void setDescripcion(String descripcion) {
        this.descripcion = descripcion;
    }

    public double getPrecio() {
        return precio;
    }

    public void setPrecio(double precio) {
        this.precio = precio;
    }
}
```

Acceso a datos con MongoDB

```
// Levantar el servidor de mongo
// comando mongod

// Desde la consola con permisos de administrador

//Consultar todos los productos
// curl http://localhost:8080
// curl http://localhost:8080/productos

//Alta de producto y consulta para su verificacion
// curl -i -X POST -H "Content-Type:application/json" -d '{"descripcion": "Enchufe", "precio":5.32 }' http://localhost:8080/productos
// curl http://localhost:8080/productos

//Busqueda de un producto por su id y luego por su descripcion
// curl http://localhost:8080/productos/5d13409d5161a02ec107c7a9
// curl http://localhost:8080/productos/search
// curl http://localhost:8080/productos/search/findByDescripcion?descripcion=Enchufe

//Modificar precio del producto y consulta para su verificacion
// curl -X PUT -H "Content-Type:application/json" -d '{"descripcion": "Enchufe", "precio":7.12 }' http://localhost:8080/productos/5d13409d5161a02ec107c7a9
// curl http://localhost:8080/productos/5d13409d5161a02ec107c7a9

//Borrar un producto y consulta para su verificacion
// curl -X DELETE http://localhost:8080/productos/5d13409d5161a02ec107c7a9
// curl http://localhost:8080/productos
```

Repositorio MongoDB

```
public interface ProductosDAO extends ReactiveMongoRepository<Producto, String>{

}
```

API Rest

```
// http://localhost:8081/api/productos
@GetMapping("/productos")
public Flux<Producto> todos(){
    return service.todos();
}
```

R2DBC

- Esta librería nos proporciona el soporte reactivo para trabajar con bases de datos relacionales

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-r2dbc</artifactId>
</dependency>
```

- Es necesario incluir el driver reactivo de la base de datos que estemos utilizando

  - H2 (`io.r2dbc:r2dbc-h2`)
  - MariaDB (`org.mariadb:r2dbc-mariadb`)
  - Microsoft SQL Server (`io.r2dbc:r2dbc-mssql`)
  - MySQL (`dev.miku:r2dbc-mysql`)
  - jasync-sql MySQL (`com.github.jasync-sql:jasync-r2dbc-mysql`)
  - Postgres (`io.r2dbc:r2dbc-postgresql`)
  - Oracle (`com.oracle.database.r2dbc:oracle-r2dbc`)

H2

- Vamos a trabajar con una base de datos en memoria H2, para ello necesitamos agregar la siguiente dependencia al pom.xml

```
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
</dependency>
```

R2DBC

- Debemos mapear la entidad a manejar en la base de datos:

```java
@Table("alumnos")
public class Alumno implements Serializable{

    @Id
    @Column("ID")
    private Integer id;

    @Column("NOMBRE")
    private String nombre;

    @Column("APELLIDO")
    private String apellido;

    @Column("NOTA")
    private double nota;
```

R2DBC

- Necesitamos crear la tabla por lo que crearemos el siguiente script en un fichero llamado data.sql

```
CREATE TABLE alumnos(
    ID INT AUTO_INCREMENT PRIMARY KEY,
    NOMBRE VARCHAR(50),
    APELLIDO VARCHAR(50),
    NOTA DECIMAL(10,2)
);
```

R2DBC

- Y por ultimo tener el repositorio donde se generaran las queries de forma automatica;

```
public interface AlumnosDAO extends R2dbcRepository<Alumno, Integer>{

    @Query("select * from alumnos")
    public Flux<Alumno> miQuery();
}
```

Funcional Endpoints

Tema 5

Router

```
@Bean
public RouterFunction<ServerResponse> rutas(HandlerProductos handler){
    return RouterFunctions
            .route(RequestPredicates.GET("/handler/productos"), handler::todos)
            .andRoute(RequestPredicates.GET("/handler/productos/{id}"), handler::buscar)
            .andRoute(RequestPredicates.POST("/handler/productos"), handler::nuevo)
            .andRoute(RequestPredicates.DELETE("/handler/productos/{id}"), handler::borrar)
            .andRoute(RequestPredicates.PUT("/handler/productos/{id}"), handler::modificar);
```

Handler

```
GET /handler/productos/{id}
public Mono<ServerResponse> buscar(ServerRequest request){
    // Comprobar si existe el producto o no
    return service.buscarProducto(request.pathVariable("id"))
            .flatMap(p -> ServerResponse
                    .ok()
                    .contentType(MediaType.APPLICATION_JSON)
                    .body(Mono.just(p), Producto.class))
                    .switchIfEmpty(ServerResponse.notFound()
            .build());
}
```

Clientes Reactivos

Tema 6

WebClient

- Es la alternativa a RestTemplate o Feign

```
@Bean
public WebClient webClient(){
    return WebClient.create("http://localhost:8080/api/productos");
}
```

Testing

Tema 7

WebTestClient

- Pruebas unitarias con WebTestClient que es lo habitual con programación Reactiva

```java
@RunWith(SpringRunner.class)
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)
public class SpringBootWebfluxApirestApplicationTests {

    @Autowired
    private WebTestClient client;

    @Test
    public void listarTest() {

        client.get()
        .uri("/api/v2/productos")
        .accept(MediaType.APPLICATION_JSON_UTF8)
        .exchange()
        .expectStatus().isOk()
        .expectHeader().contentType(MediaType.APPLICATION_JSON_UTF8)
        .expectBodyList(Producto.class)
        .hasSize(9);
    }

}
```

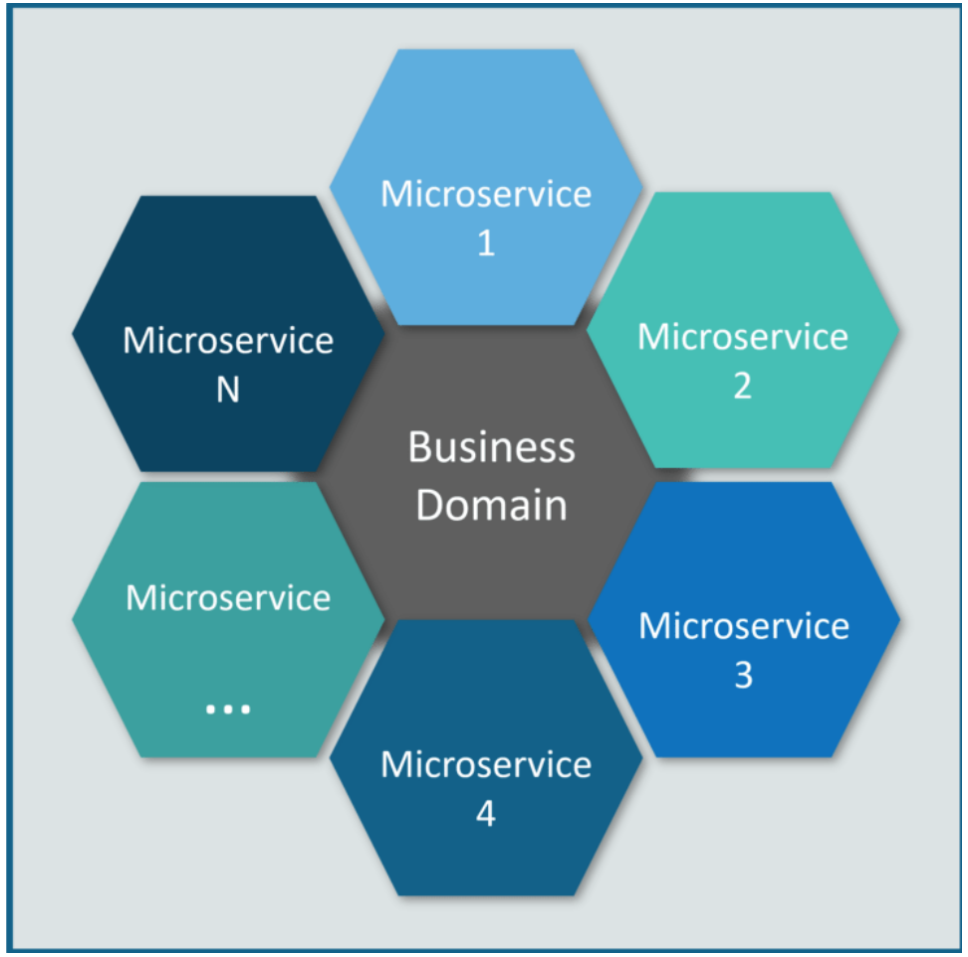
Introducción a los Microservicios

Tema 8

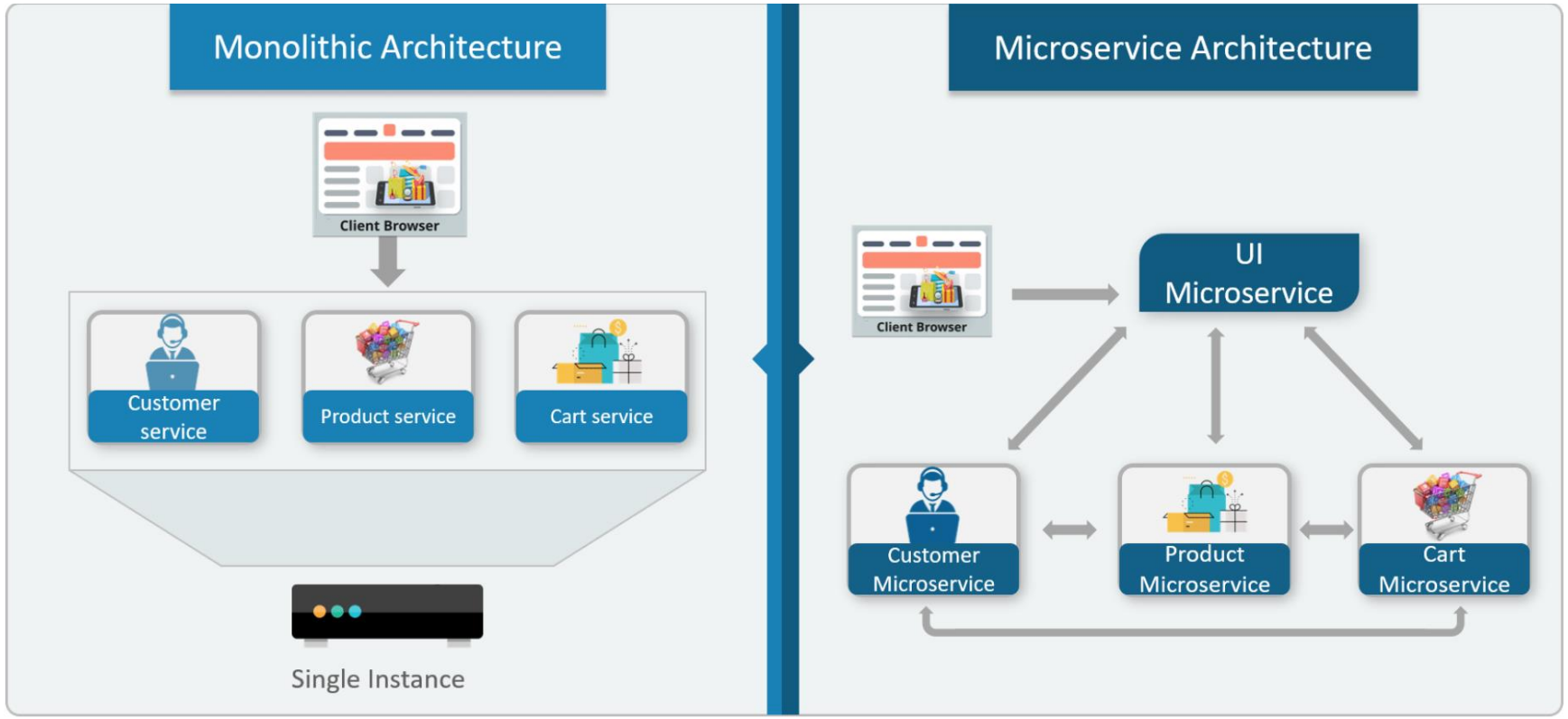
Qué es un microservicio?

- Según *Martin Fowler* y *James Lewis* explican en su artículo *Microservices*, los **microservicios** se definen como un estilo arquitectural, es decir, una forma de desarrollar una aplicación, basada en un conjunto de pequeños servicios, cada uno de ellos ejecutándose de forma autónoma y comunicándose entre si mediante mecanismos livianos, generalmente a través de peticiones **REST** sobre HTTP por medio de sus **APIs**.

Qué es un microservicio?



Arquitectura monolítica vs Arquitectura microservicios



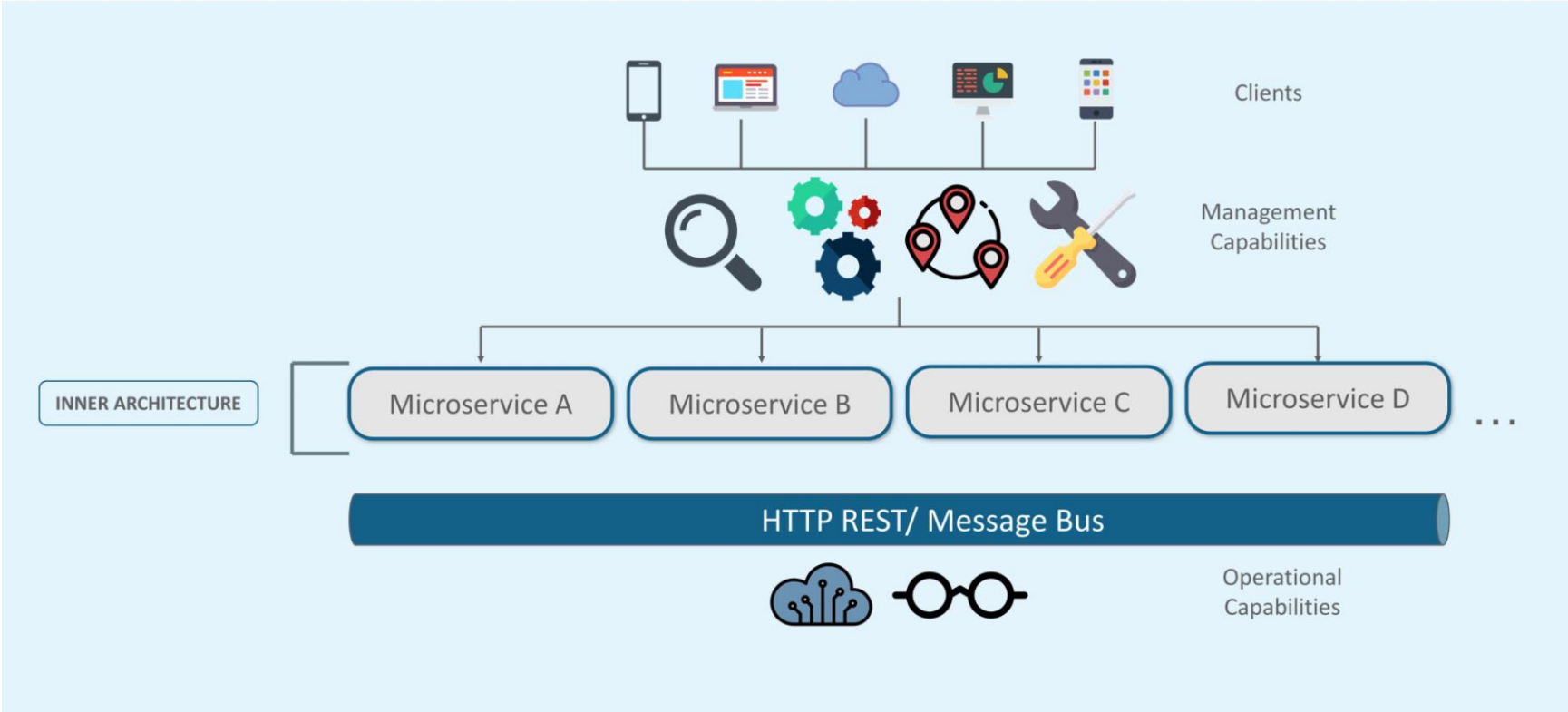
Tendencia en el desarrollo

- La tendencia es que las aplicaciones sean diseñadas con un ***enfoque orientado a microservicios***, construyendo múltiples servicios que colaboran entre si, en lugar del ***enfoque monolítico***, donde se construye y despliega una única aplicación que contenga todas las funcionalidades.

Características de los Microservicios

- Pueden ser auto-contenidos, de tal forma que incluyen todo lo necesario para prestar su servicio
- Servicios pequeños, lo que facilita el mantenimiento. Ej: Personas, Productos, Posición Global, etc
- Principio de responsabilidad única: cada microservicio hará una única cosa, pero la hará bien
- Políglotas: una arquitectura basada en microservicios facilita la integración entre diferentes tecnologías (lenguajes de programación, BBDD…etc)
- Despliegues unitarios: los microservicios pueden ser desplegados por separado, lo que garantiza que cada despliegue de un microservicio no implica un despliegue de toda la plataforma. Tienen la posibilidad de incorporar un servidor web embebido como Tomcat o Jetty
- Escalado eficiente: una arquitectura basada en microservicios permite un escalado elástico horizontal, pudiendo crear tantas instancias de un microservicio como sea necesario.

Arquitectura microservicios



Arquitectura microservicios

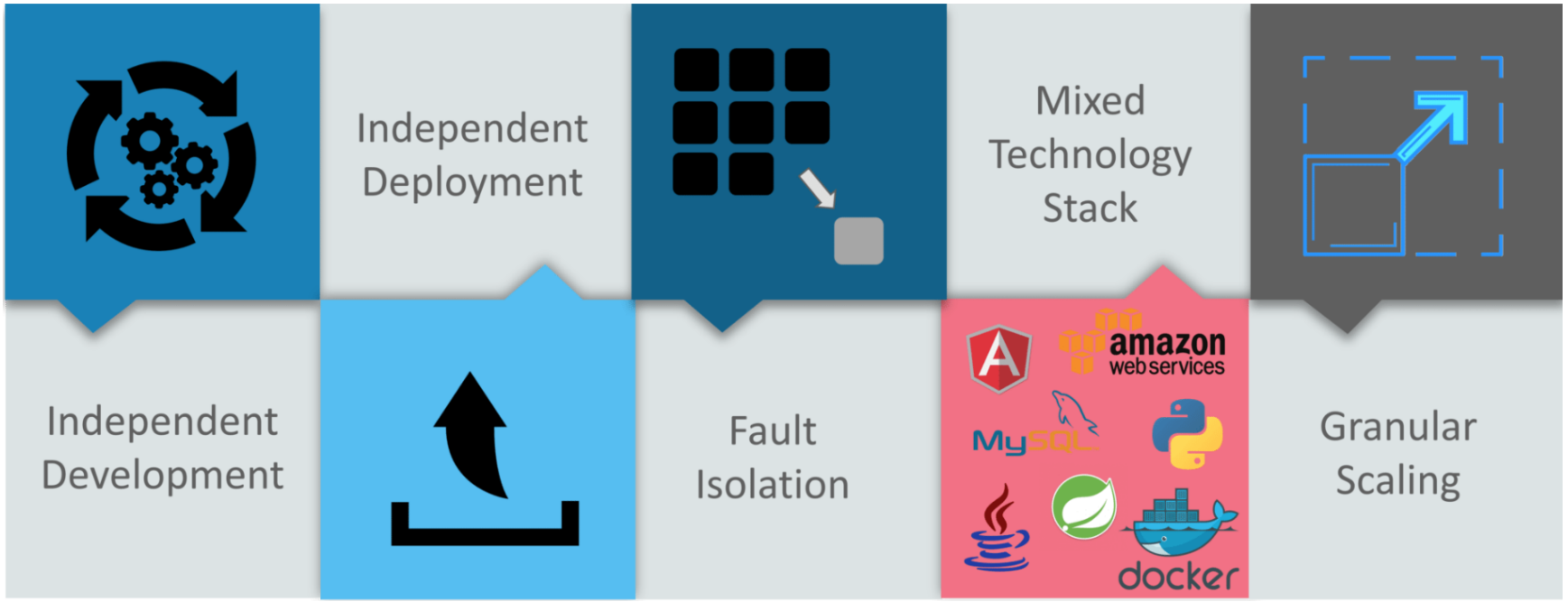
- Diferentes clientes de diferentes dispositivos intentan usar diferentes servicios como búsqueda, creación, configuración y otras capacidades de administración
- Todos los servicios se separan según sus dominios y funcionalidades y se asignan a microservicios individuales.
- Estos microservicios tienen su propio balanceador de carga y entorno de ejecución para ejecutar sus funcionalidades y al mismo tiempo captura datos en sus propias bases de datos.
- Todos los microservicios se comunican entre sí a través de un servidor sin estado que es REST o Message Bus.
- Los microservicios conocen su ruta de comunicación con la ayuda de Service Discovery y realizan capacidades operativas tales como automatización, monitoreo
- Luego, todas las funcionalidades realizadas por los microservicios se comunican a los clientes a través de la puerta de enlace API.
- Todos los puntos internos están conectados desde la puerta de enlace API. Por lo tanto, cualquiera que se conecte a la puerta de enlace API se conecta automáticamente al sistema completo

Características de los microservicios

Características de los microservicios

- Desacoplamiento: los servicios dentro de un sistema se desacoplan en gran medida. Por lo tanto, la aplicación en su conjunto se puede construir, modificar y escalar fácilmente.
- Componentes: los microservicios se tratan como componentes independientes que se pueden reemplazar y actualizar fácilmente.
- Capacidades empresariales: los microservicios son muy simples y se centran en una sola capacidad
- Autonomía: los desarrolladores y los equipos pueden trabajar de forma independiente, lo que aumenta la velocidad.
- Entrega continua: permite lanzamientos frecuentes de software, a través de la automatización sistemática de la creación, prueba y aprobación del software.
- Responsabilidad: Los microservicios no se centran en aplicaciones como proyectos. En cambio, tratan las aplicaciones como productos de los que son responsables.
- Gobernanza descentralizada: el enfoque está en usar la herramienta adecuada para el trabajo correcto. Eso significa que no hay un patrón estandarizado o ningún patrón tecnológico. Los desarrolladores tienen la libertad de elegir las mejores herramientas útiles para resolver sus problemas
- Agilidad - Los microservicios apoyan el desarrollo ágil. Cualquier nueva característica puede ser desarrollada rápidamente y descartada nuevamente

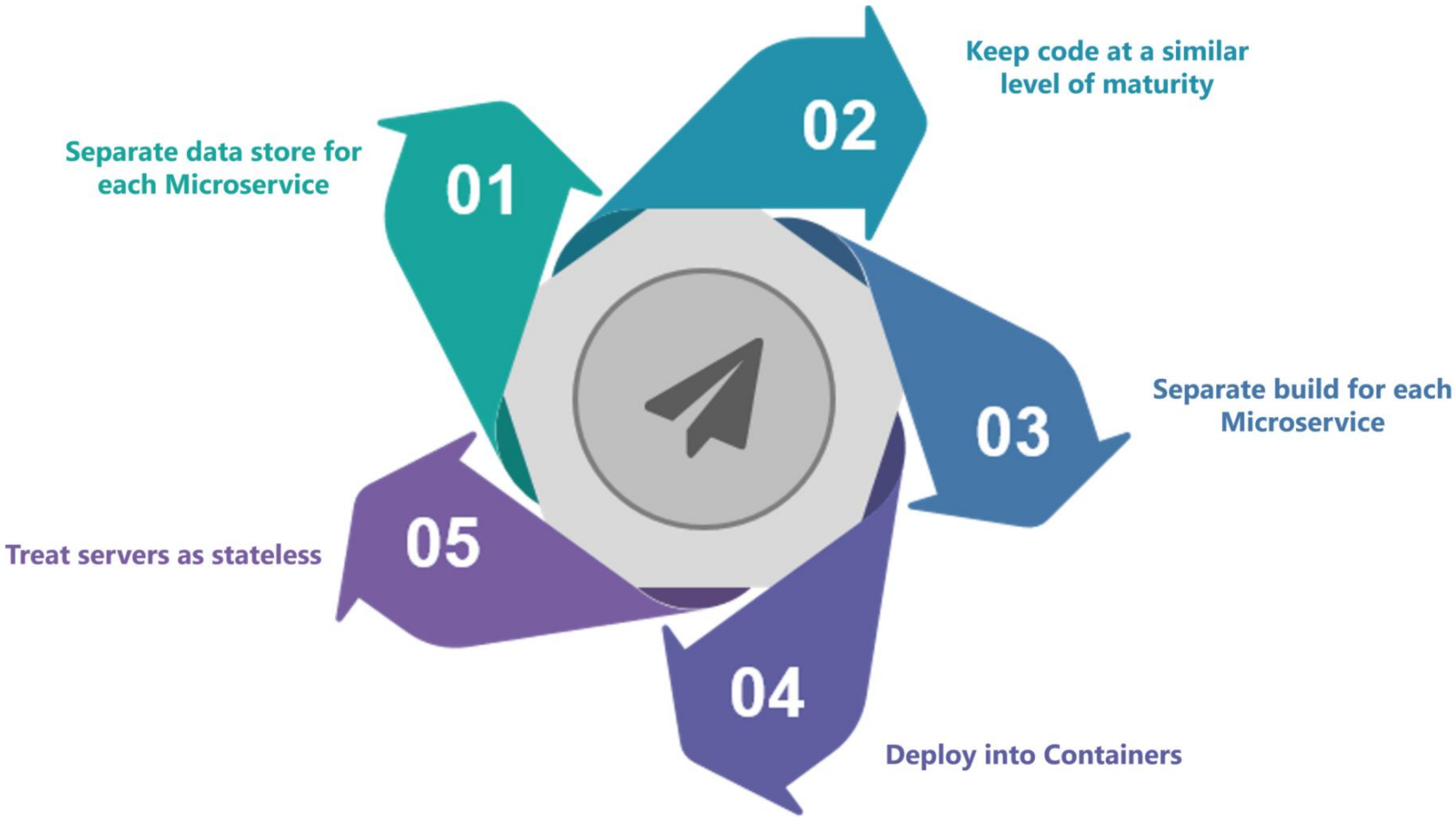
Ventajas de los microservicios



Ventajas de los microservicios

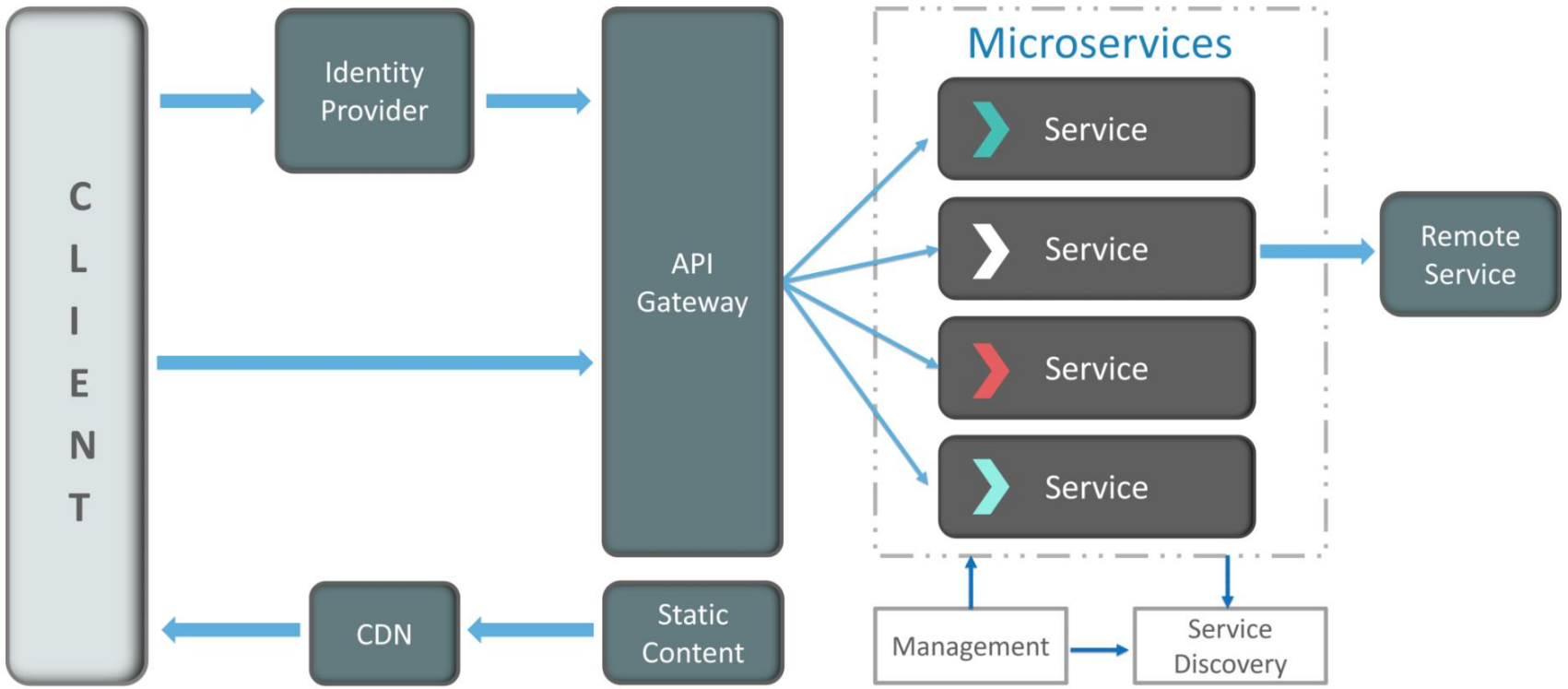
- Desarrollo independiente: todos los microservicios se pueden desarrollar fácilmente según su funcionalidad individual
- Implementación independiente: en función de sus servicios, se pueden implementar individualmente en cualquier aplicación
- Aislamiento de fallos: incluso si un servicio de la aplicación no funciona, el sistema continúa funcionando
- Pila de tecnología mixta: se pueden utilizar diferentes lenguajes y tecnologías para crear diferentes servicios de la misma aplicación
- Escalado granular: los componentes individuales pueden escalarse según la necesidad, no es necesario escalar todos los componentes

Buenas prácticas de diseño

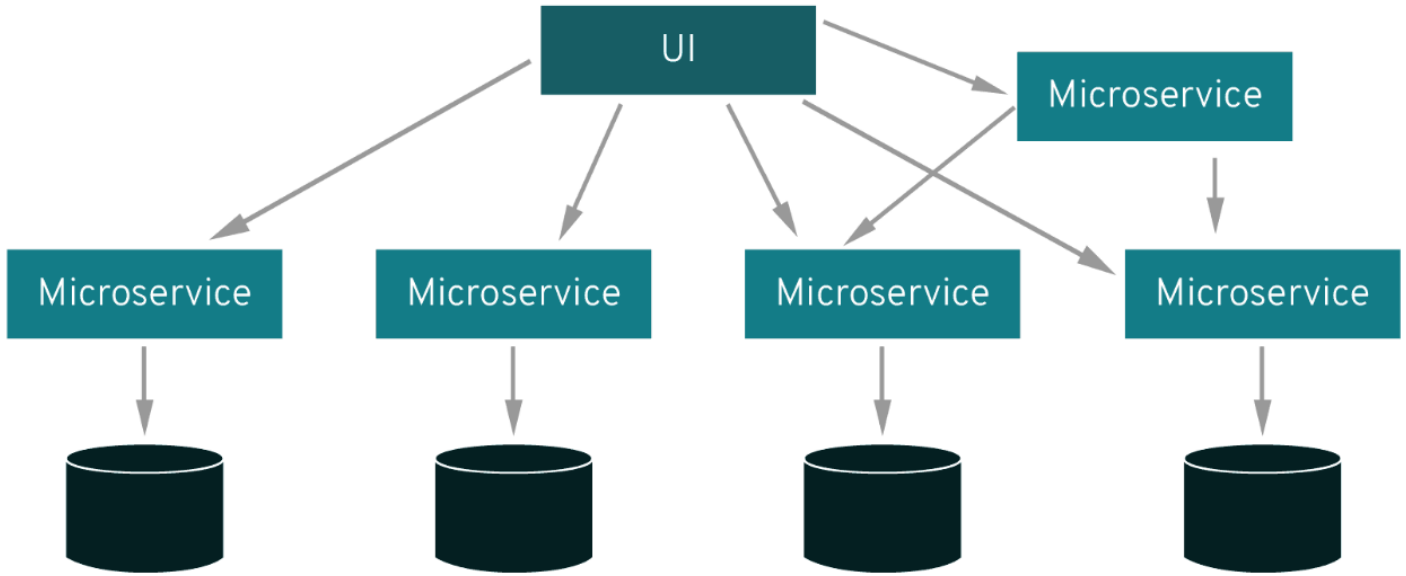


Quien los utiliza?

Componentes de la arquitectura



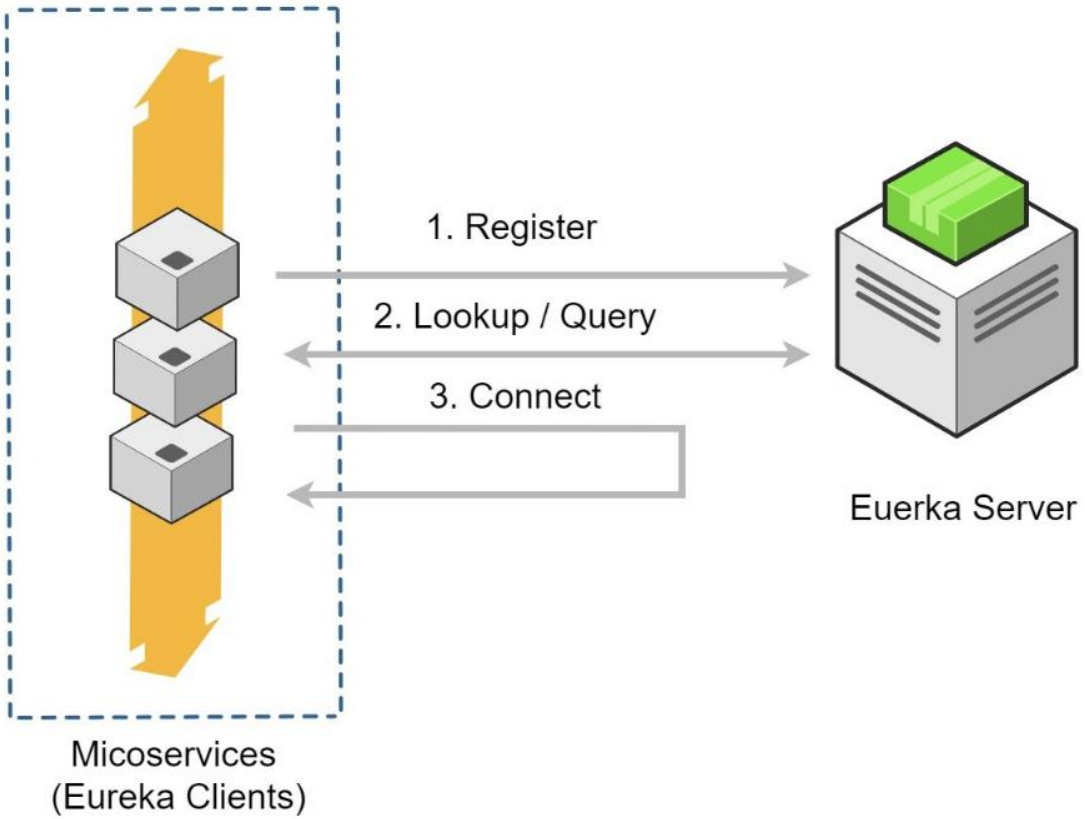
Consulta entre servicios



Consulta entre servicios

- Los microservicios se crean de forma independiente y se comunican entre sí. Además, si se produce un error individual, este no provoca una interrupción de toda la aplicación.
- La comunicación se lleva a cabo a través de peticiones Rest.
- Dos formas de implementar el cliente de la petición:
  - RestTemplate
  - Feign
- Peticiones reactivas:
  - WebClient

Eureka Netflix



Eureka Netflix

- Eureka es un servicio rest que permite al resto de microservicios registrarse en su directorio.

- Esto es muy importante, puesto que no es Eureka quien registra los microservicios, sino los microservicios los que solicitan registrarse en el Eureka.

Eureka Netflix

- Cuando un microservicio registrado en Eureka arranca, envía un mensaje a Eureka indicándole que está disponible.
- El servidor Eureka almacenará la información de todos los microservicios registrados así como su estado.
- La comunicación entre cada microservicio y el servidor Eureka se realiza mediante heartbeats cada X segundos.
- Si Eureka no recibe un heartbeat de un determinado microservicio, pasados 3 intervalos, el microservicio será eliminado del registro.
- Además de llevar el registro de los microservicios activos, Eureka también ofrece al resto de microservicios la posibilidad de "descubrir" y acceder al resto de microservicios registrados.
- Por ello Eureka es considerado un servicio de registro y descubrimiento de microservicios

Icono *Training Consulting*

MOD-111 Programación Reactiva con Spring WebFlux

Completa nuestra encuesta de satisfacción a través del QR

GRACIAS

2025

VIEWNEXT
AN IBM SUBSIDIARY