

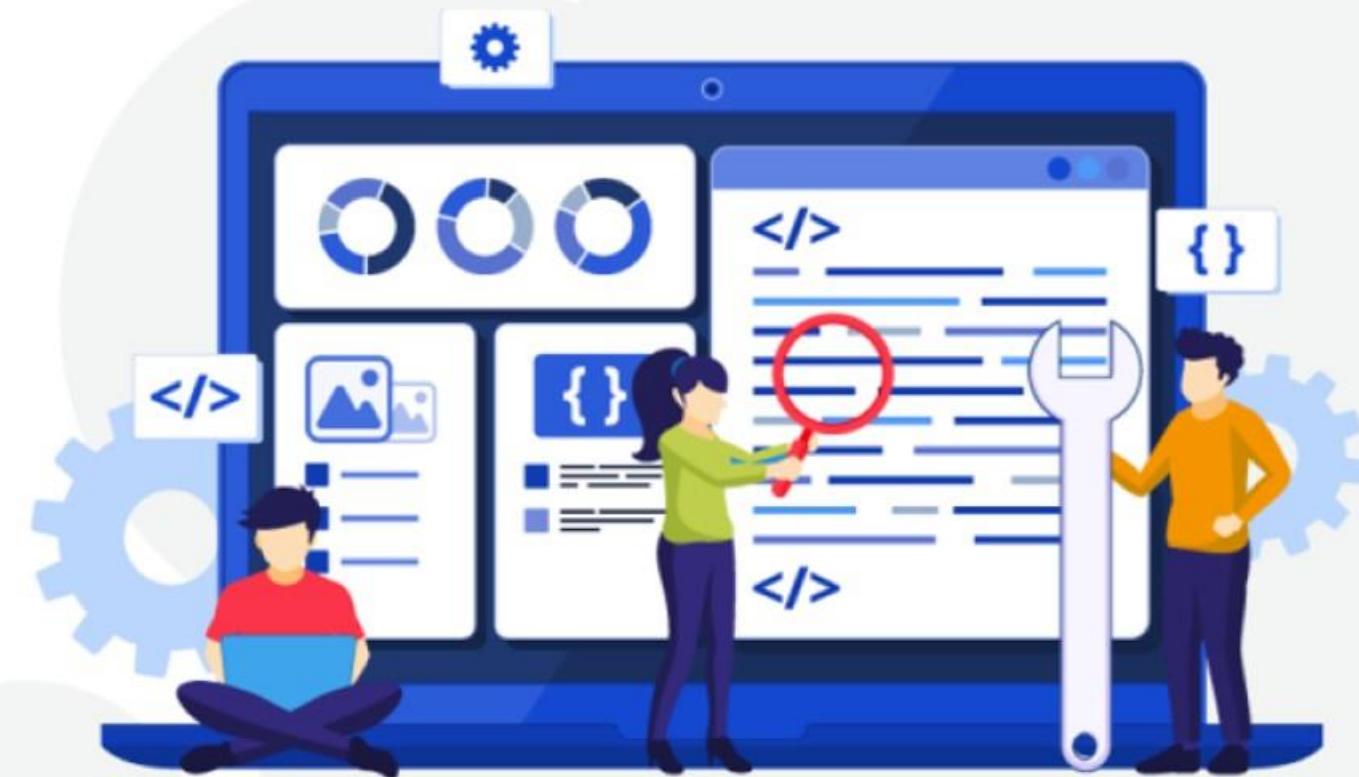


CICLO 4a

[FORMACIÓN POR CICLOS]

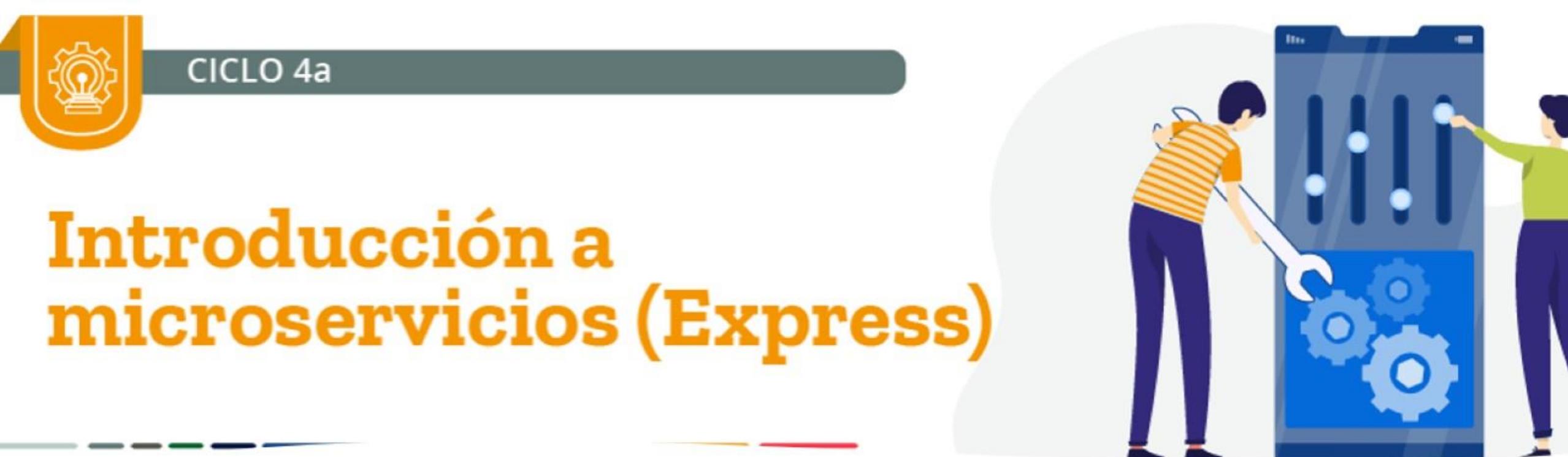
## Desarrollo de **APLICACIONES WEB**

**Semana 5:**  
**Introducción a**  
**microservicios**  
**(Express)**



**UNIVERSIDAD  
DE ANTIOQUIA**

Facultad de Ingeniería



## Objetivos de aprendizaje

- Construir arquitecturas basadas en microservicios que nos permitan atomizar las reglas del negocio dando más flexibilidad y eficiencia a la hora de implementar y desplegar artefactos funcionales dentro de la arquitectura empresarial corporativa.

# Agenda



DEFINICIÓN



ARQUITECTURAS



EJEMPLOS



EXPRESS



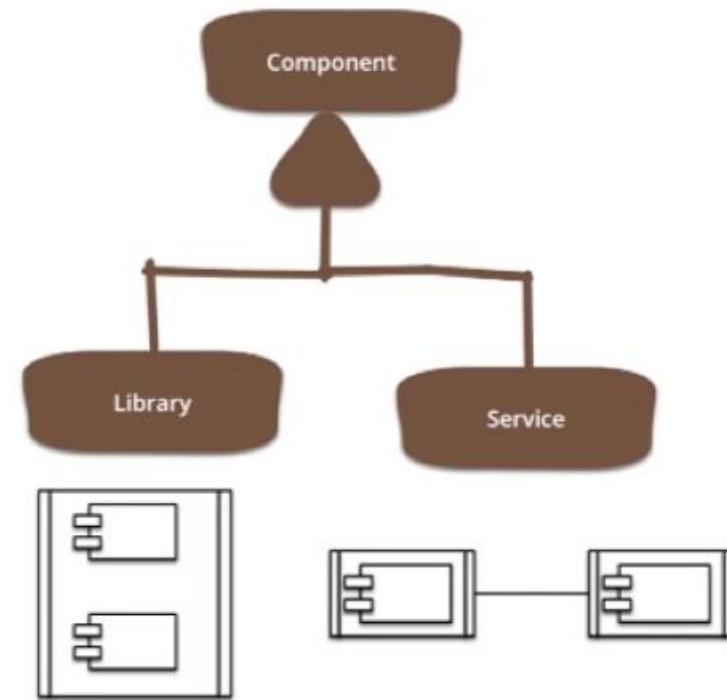
BIBLIOGRAFÍA

# Microservicios

- Un microservicio propone su propia arquitectura y su propio lenguaje de programación, consiguiendo de esta manera versatilidad para todas las formas tecnológicas que se encuentran en la industria del desarrollo de software.
- En esta presentación se aclararán los temas introductorios principales para comprender los microservicios y su uso.

# Microservicios

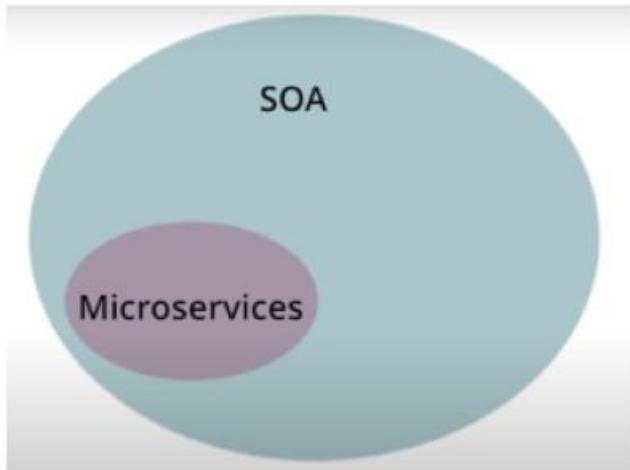
- Los microservicios son un sistema de desarrollo de software que ha venido creciendo en popularidad en los últimos años, influyendo de manera positiva en aspectos como el tiempo, el rendimiento y la estabilidad de los proyectos.
- Los microservicios proponen su propia arquitectura permitiendo funcionar como un conjunto de pequeños servicios que se ejecutan de manera independiente y autónoma, creados con diferentes lenguajes de programación.



# Historia

---

## How Big?

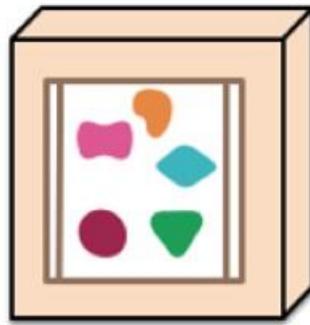


Diego Iván Oliveros Acosta @scalapp.co

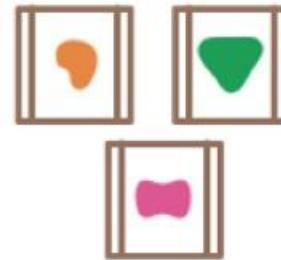
*A monolithic application puts all its functionality into a single process...*



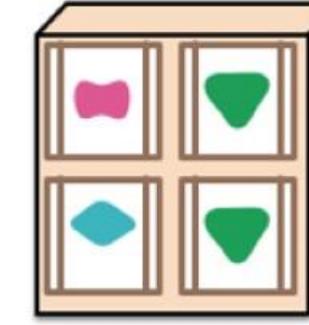
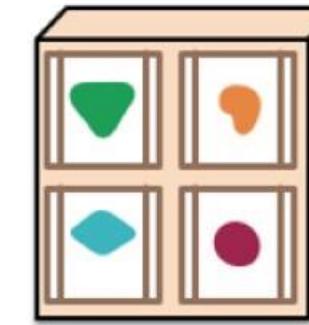
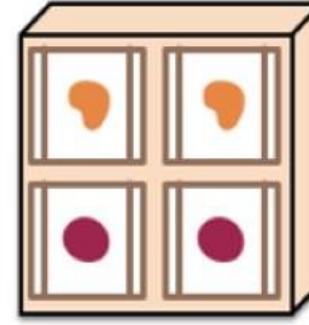
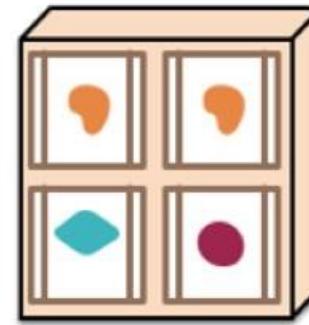
*... and scales by replicating the monolith on multiple servers*



*A microservices architecture puts each element of functionality into a separate service...*



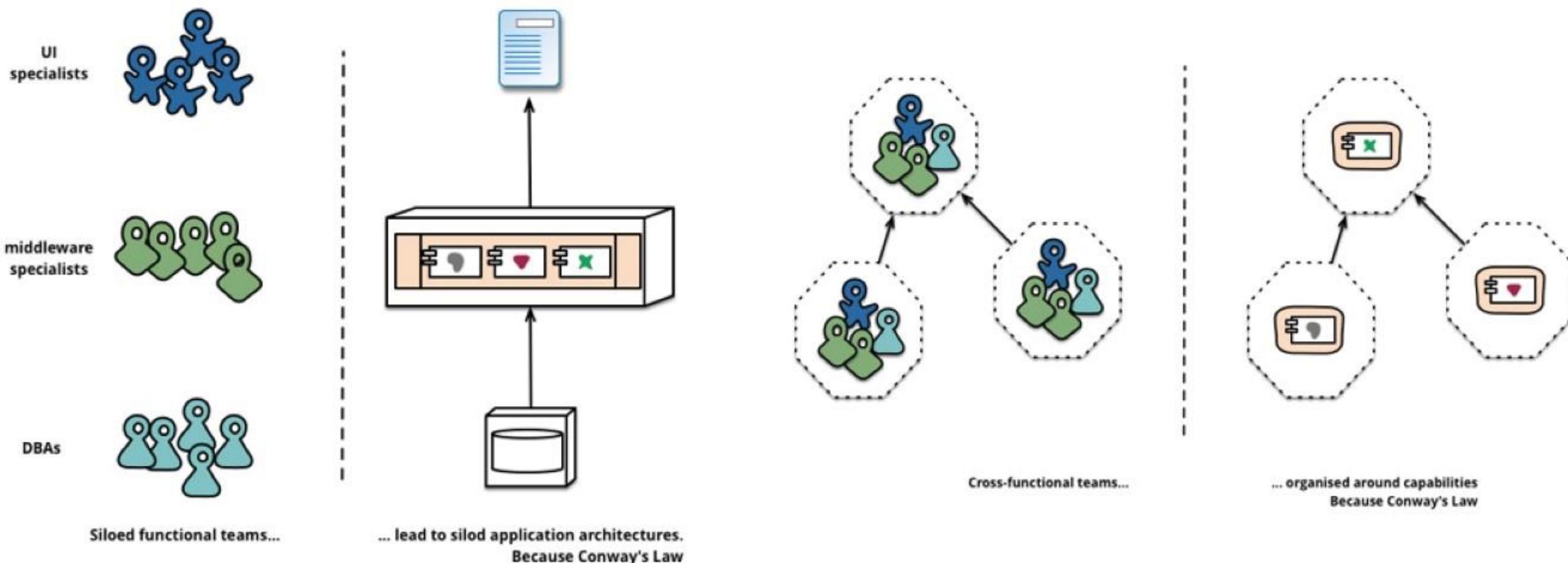
*... and scales by distributing these services across servers, replicating as needed.*

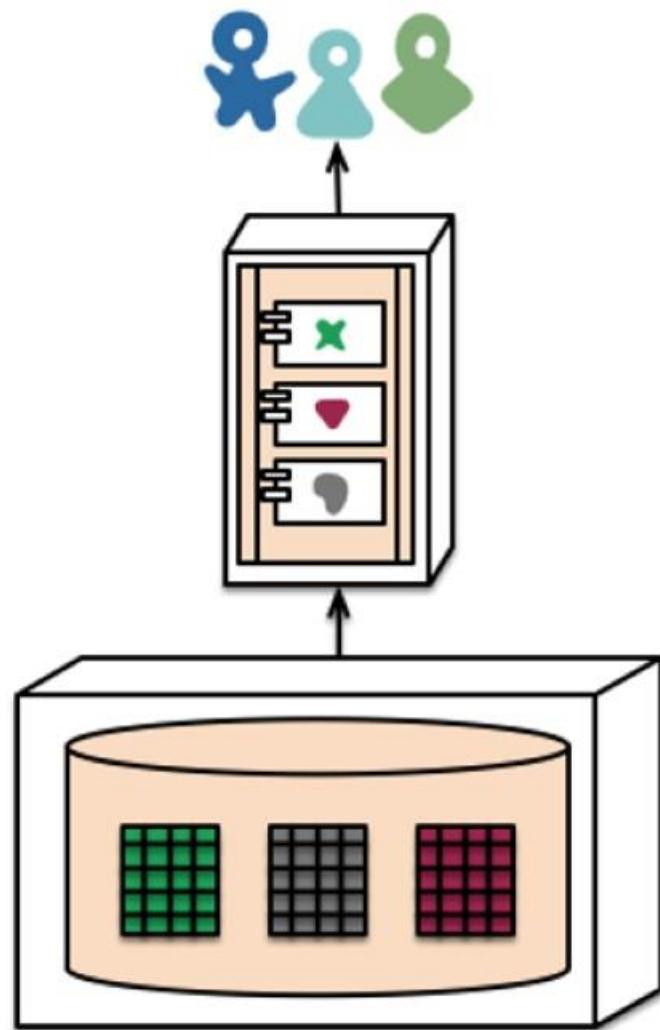


- Tradeoff:

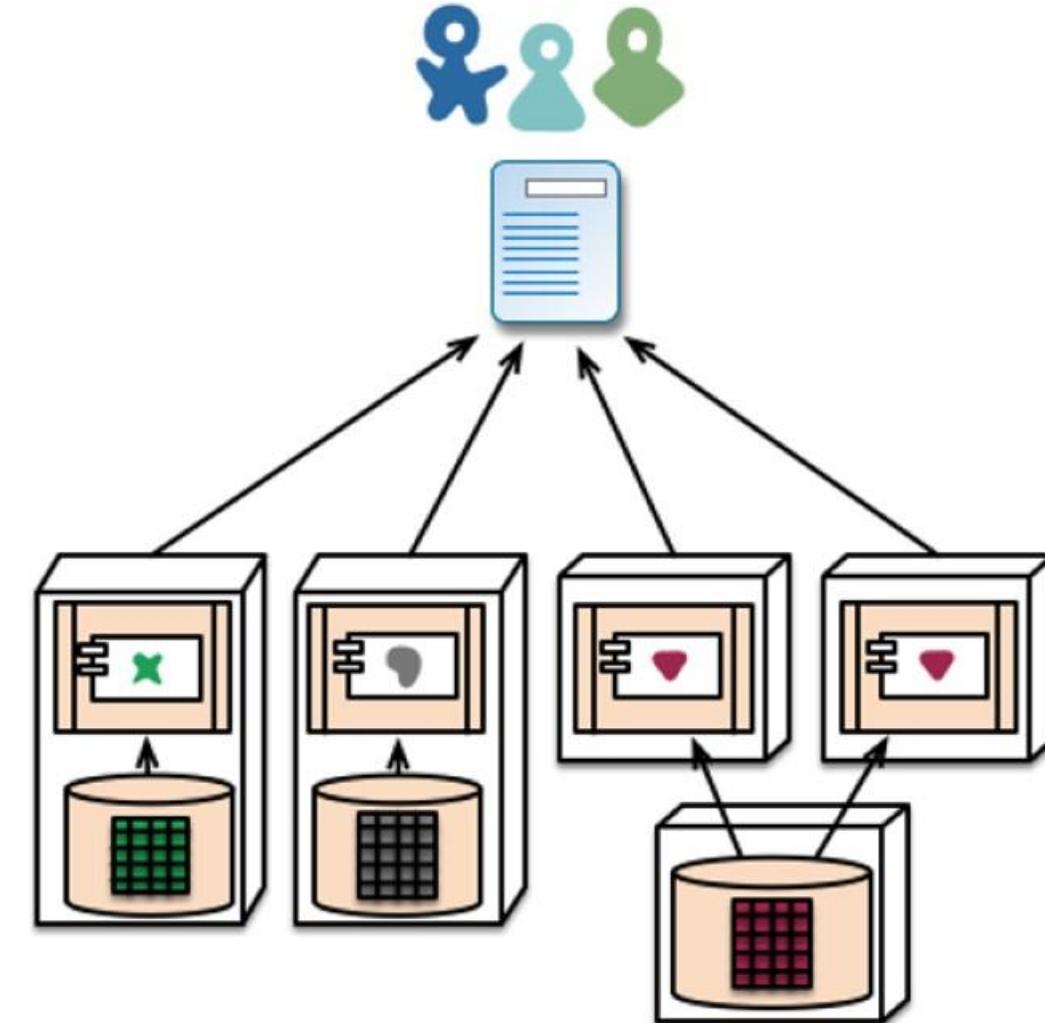
Simplicidad  
consistencia  
refactorización entre módulos  
monitoreo?

- Comunicación asíncrona
- Despliegue parcial
- Disponibilidad (shopping)
- Preservar modularidad
- Múltiple plataforma





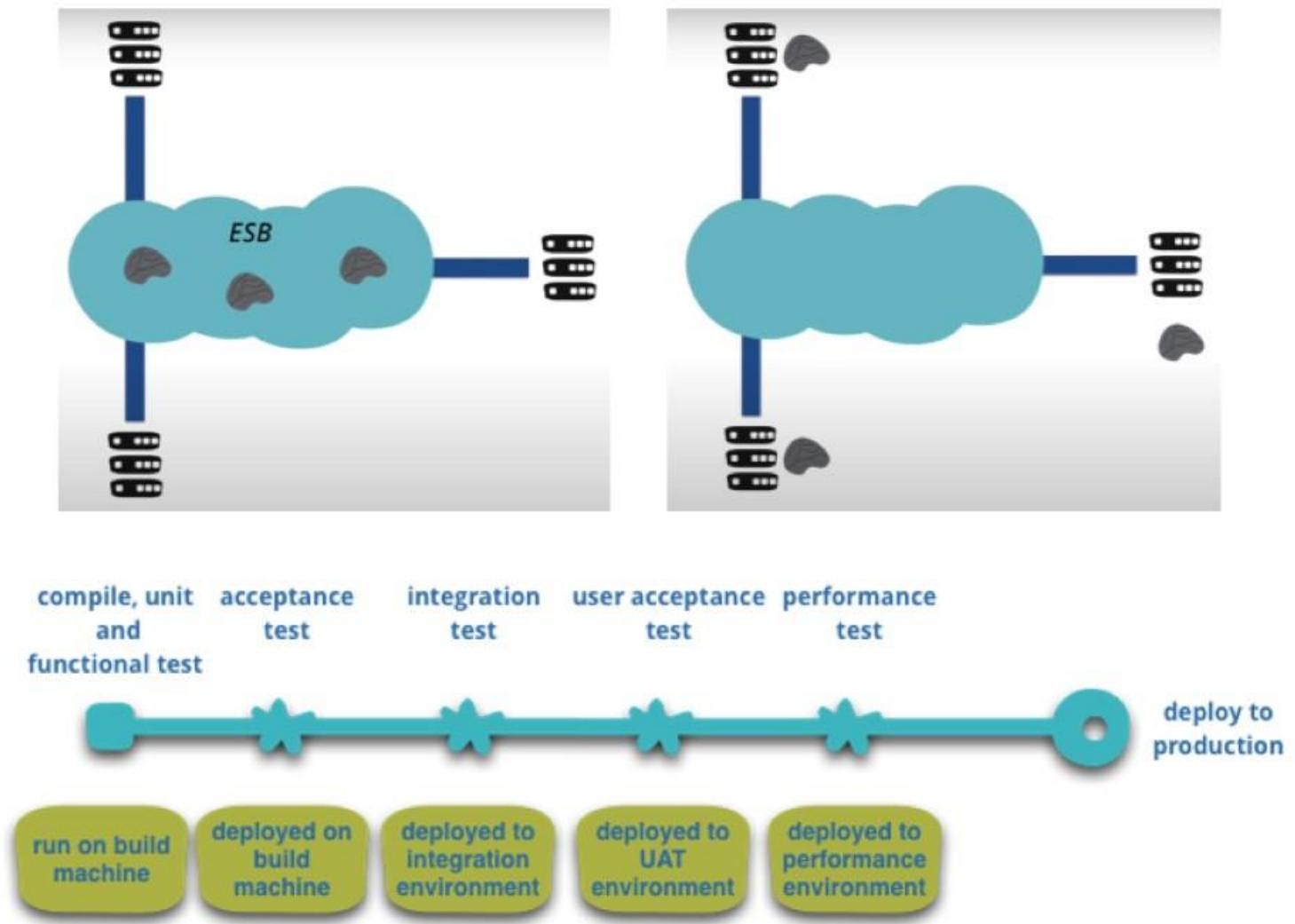
monolith - single database



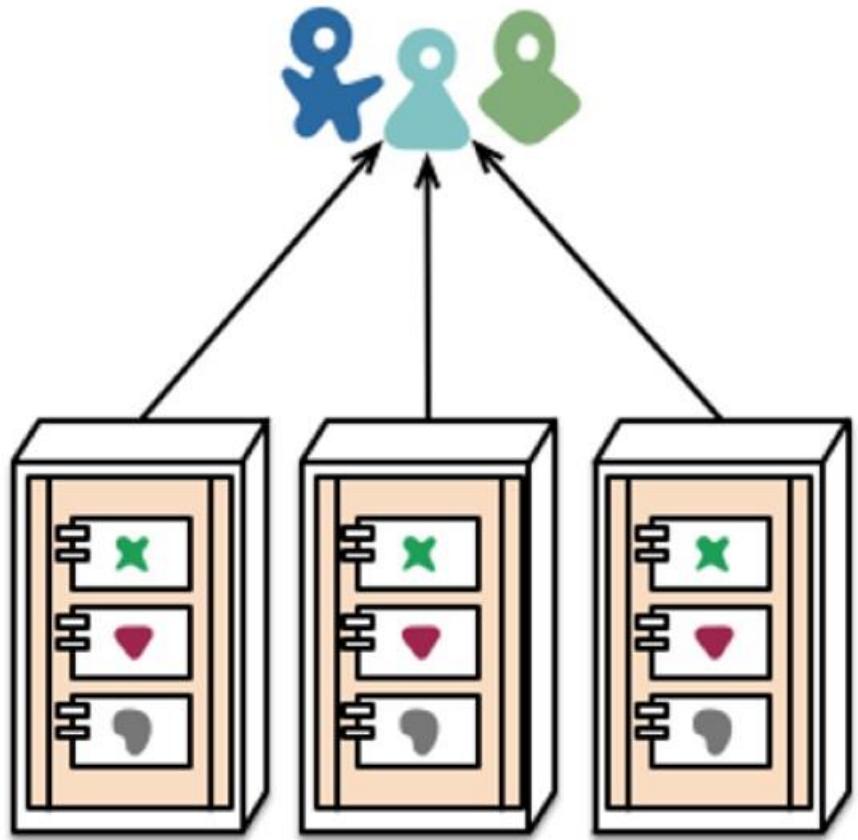
microservices - application databases

Diego Iván Oliveros Acosta @scalapp.co

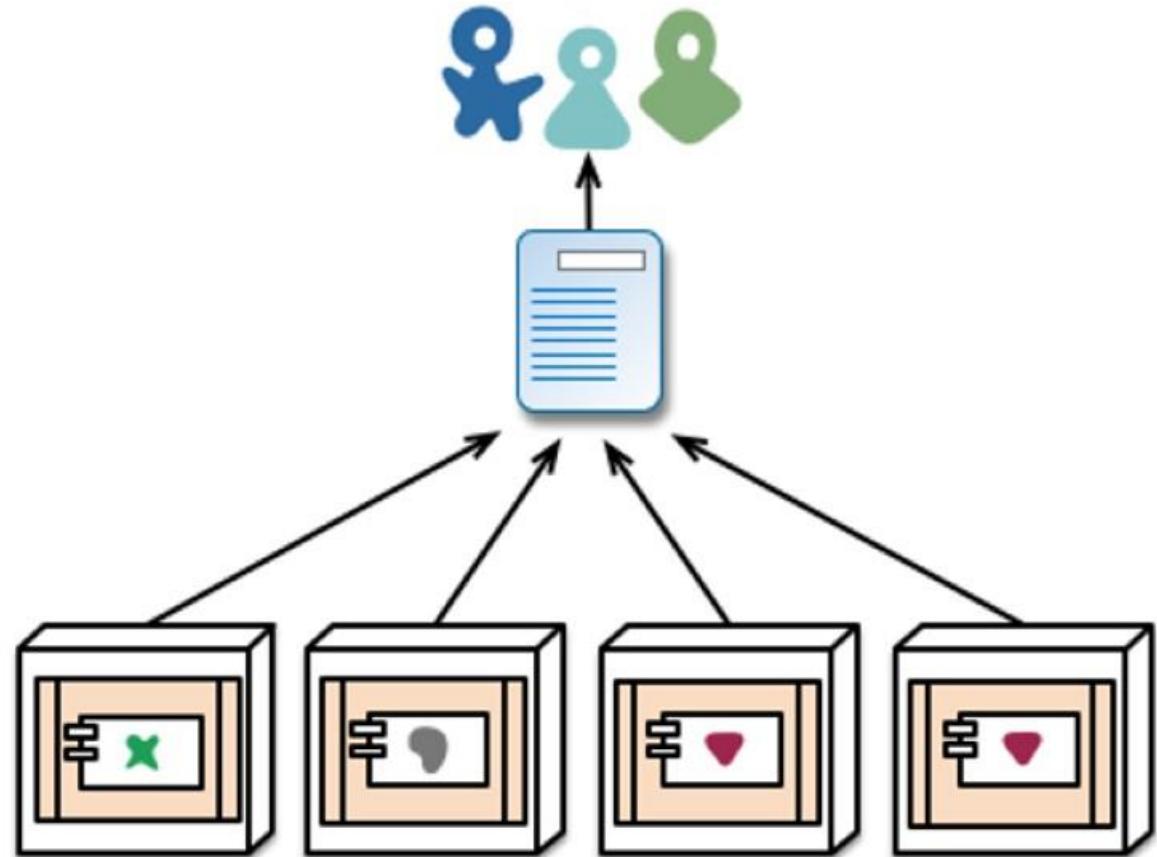
# Infrastructure Automation



Diego Iván Oliveros Acosta @scalapp.co



monolith - multiple modules in the same process



microservices - modules running in different processes

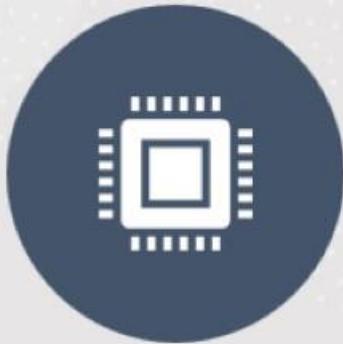
# Microservicios

Para aplicar una arquitectura orientada a microservicios sobre soluciones IT ya desarrolladas, se propone interactuar con dos capas:

- Una capa que actúa de manera interna conteniendo los componentes del microservicio puro con acciones complejas menores.
- Una capa externa alrededor que se construye del microservicio y que contendrá las capacidades requeridas de los servicios expuestos en estas arquitecturas.



# Microservicios



LOS MICROSERVICIOS SE COMUNICAN ENTRE SÍ A TRAVÉS DE PETICIONES HTTP HACIA SUS API.



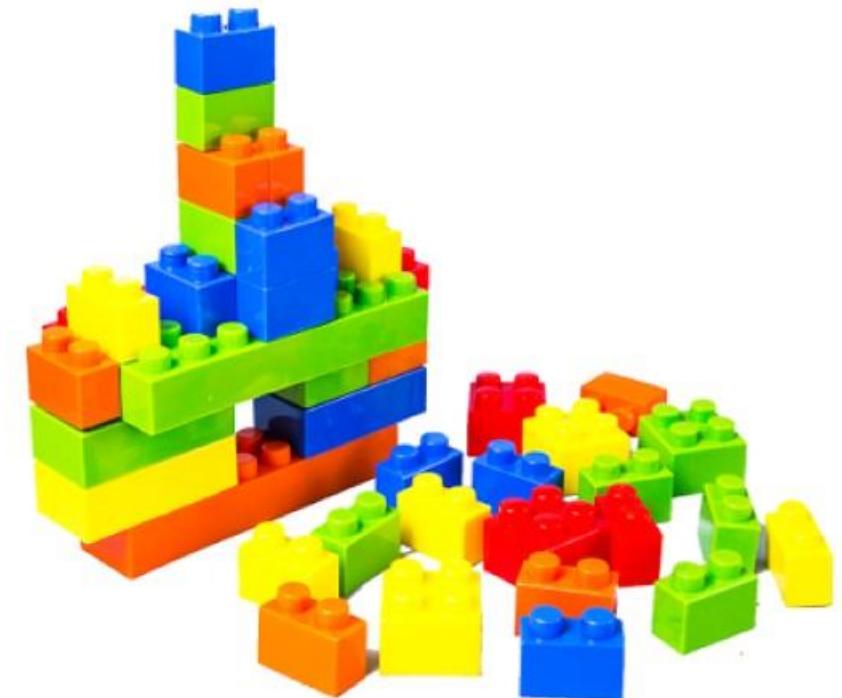
EN ARQUITECTURAS DE MICROSERVICIOS DEBE HABER UN GRUPO MÍNIMO DE MICROSERVICIOS QUE GESTIONEN ELEMENTOS COMUNES.



UNO DE LOS PUNTOS FUERTES DE ESTE TIPO DE SERVICIOS ES SU ESCALABILIDAD.

# Microservicios

- Pueden ser desplegados y modificados sin afectar otros aspectos funcionales de la aplicación en general del proyecto o producto.
- Cada microservicio es independiente porque posee su propia base de datos, evitando así la sobrecarga y el encolamiento de solicitudes (*request*) desde el cliente.



# Microservicios

- La ventajas de los microservicios son:
  - Equipo de trabajo mínimo
  - Escalabilidad
  - Funcionalidad modular, módulos independientes
  - Libertad del desarrollador de desarrollar y despliegue de servicios de manera independiente
  - Uso de contenedores que permiten rápidamente el despliegue y el desarrollo de la aplicación

Diego Iván Oliveros Acosta @scalapp.co

# Introducción a microservicios con Netflix OSS

Diego Iván Oliveros Acosta

# Agenda

Generalidades

Microservicios con Netflix OSS

Components

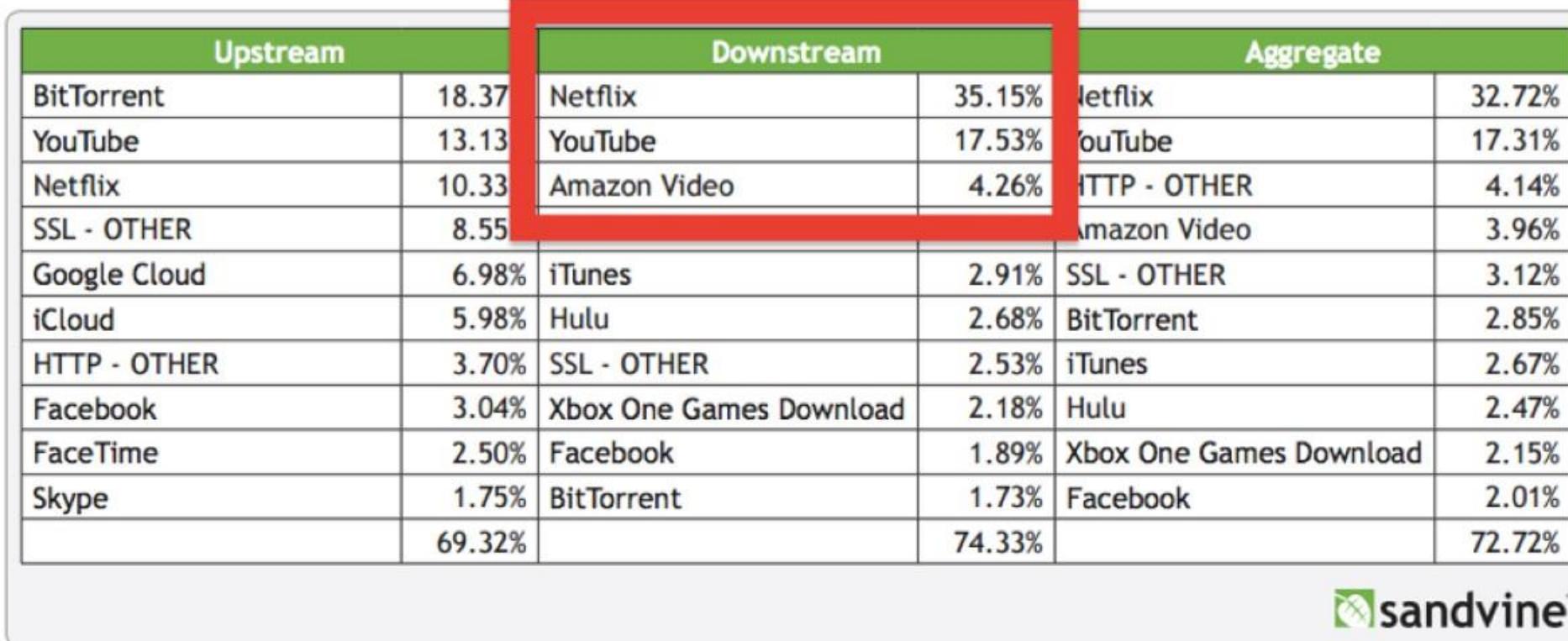
- Eureka
- Zuul
- Hystrix
- Feign
- Ribbon

Referencias

# Microservicios Netflix OSS

Netflix OSS es un conjunto de herramientas y componentes de software que permiten desarrollar servicios de manera fácil y rápida.

La arquitectura de Netflix está compuesta por más de 500 microservicios y cuenta con más de 50 millones de suscriptores.



# Algunas opiniones:

- "All teams will henceforth expose their data and functionality through service interfaces" Jeff Bezos
- "que tan grande o pequeño debe ser un microservicio?" y "cual es la diferencia entre microservicios y Arquitectura Orientada a Servicios (SOA)" Martin Fowler y James Lewis

# ¿Por qué microservicios?

- Motivos para el cambio:
  - Disponibilidad (24/7), escalado, velocidad
  - Data center vs AWS
  - Agilizar desarrollo y despliegue
- Timing:
  - 2009 comienzan a migrar de una arquitectura monolítica a una arquitectura cloud basada en MS
  - Para finales 2010 - el front se había movido a AWS
  - Para finales 2011 - todo su monolito en AWS descompuesto en cientos de microservicios
  - 2012 - empiezan a liberar código como Open Source



**NETFLIX | OSS**

# Características de un micro servicio

1. Un microservicio es un servicio que cubre una y solo una funcionalidad muy concreta. Esta funcionalidad es expuesta al resto de sistemas en un api.
2. Cada ms se ejecuta en un proceso y puede ser desplegado de forma independiente a los demás.
3. Tiene su propia db
4. Pueden estar implementados con diferentes lenguajes, bd, plataformas
5. Utiliza mecanismos de comunicación tradicionales: http, colas, bus de servicios...
6. Responsable de una única capacidad /funcionalidad
7. Se debe poder desplegar de manera individual

# Arquitectura de microservicios

- Es una arquitectura basada en una forma ligera de SOA donde el objetivo de los servicios es hacer una única cosa y hacerla bien
- Es una arquitectura SOA compuesta por componentes con un bajo nivel de acoplamiento que tienen un contexto claramente definido
- Es un sistema distribuido con un número generalmente grande de ms que colaboran entre ellos. Cada Ms se ejecuta como un proceso y proporciona solo una pequeña parte de la funcionalidad, donde el conjunto funciona en base a la colaboración de los mismos. Para colaborar se comunican por un medio que no esta acoplado a una plataforma concreta.

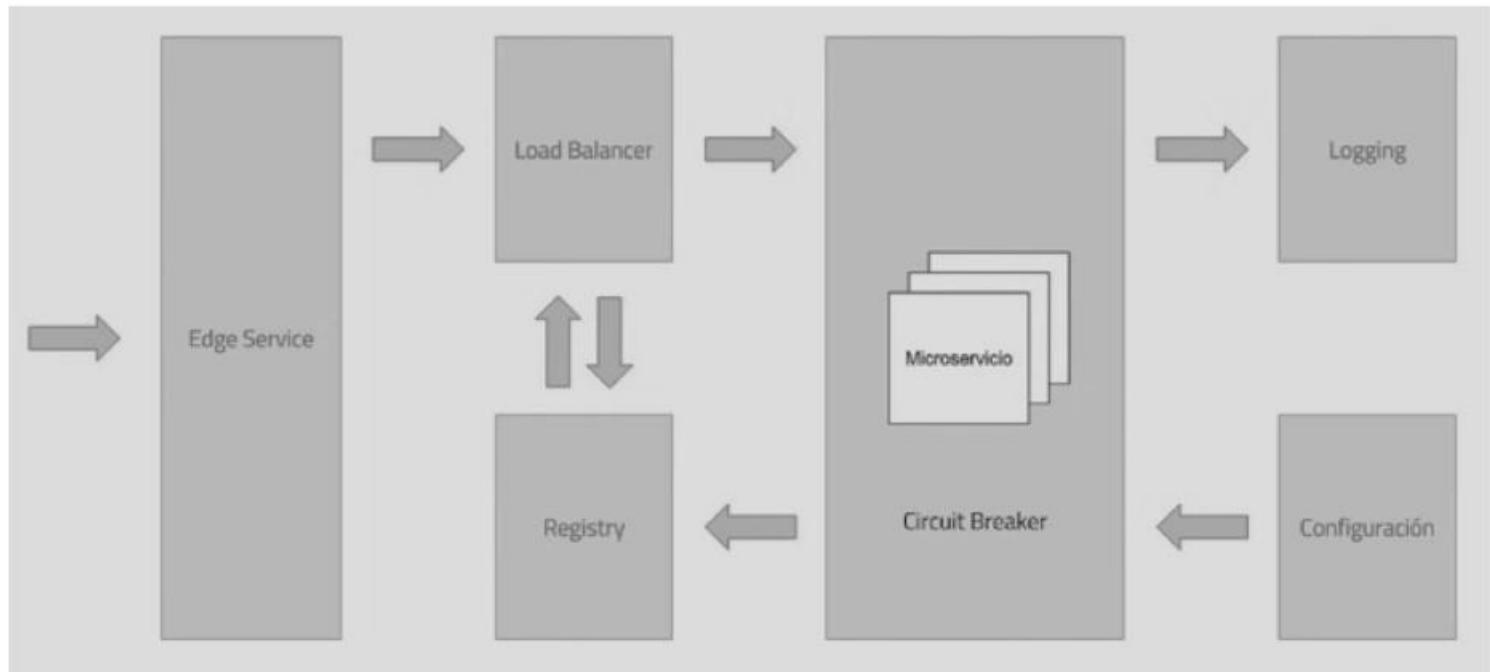
# Tipos de sub arquitectura

- Orientada al dominio
- Orientada al recurso
- Orientada a la funcionalidad
- Orientada al caso de negocio
- Mixta o por capas.
- Capa de acceso a recurso
- Capa de lógica de negocio
- Capa de integración



# Patrones de arquitectura que regresan a la arquitectura monolítica.

1. Registry
2. Config
3. Load balancer
4. Circuit breaker
5. Edge service
6. Log centralizados



# Atributos de calidad



SEGURIDAD



RENDIMIENTO O  
ALTA  
DISPONIBILIDAD



DIFFERENTE STACK  
TECNOLÓGICO



CONSISTENCIA DE  
DATOS



EQUIPOS  
DIFERENTES



FRECUENCIA DE  
CAMBIOS

# Gestión y ciclo de vida del servicio

- Gestión de la configuración ci/cd
- Montarlo
- Probar
- Monitorizar
- Mantener / reemplazar
- Refactorizar
- Escalar vs crecer
- Romper

# Consistencia eventual

- Los recursos están distribuidos en múltiples sistemas.
- Gestión de transacciones distribuidas.
- **Los microservicios introducen eventuales problemas de coherencia debido a su loable insistencia en la gestión de datos descentralizada.**
- **Con un monolito, puede actualizar un montón de cosas juntas en una sola transacción.**
- **Los microservicios requieren múltiples recursos para actualizarse, y las transacciones distribuidas están mal vistas (por una buena razón).**

# Casos de uso

- Cuando necesitemos o cuando queramos:
  - Reducir time to market
  - Exponer nuestros datos o servicios
  - Integrarnos con otros sistemas
  - Publicar servicios de alta disponibilidad y rendimiento
  - Centralizar la funcionalidad.



# En resumen,

- El **estilo** arquitectónico de microservicio es un enfoque para desarrollar una sola aplicación como un **conjunto de pequeños servicios**, cada uno **ejecutándose en su propio proceso** y comunicándose con mecanismos livianos, a menudo una API de recursos HTTP. Estos servicios se **basan en capacidades comerciales** y se implementan de **forma independiente** mediante maquinaria de implementación totalmente automatizada. Hay un **mínimo indispensable de gestión centralizada** de estos servicios, que pueden estar escritos en diferentes lenguajes de programación y utilizar diferentes tecnologías de almacenamiento de datos.
- -- James Lewis y Martín Fowler (2014)

# Microservicios Netflix OSS

- Separación de responsabilidades
  - Modularidad, encapsulamiento,
  - Escalabilidad
- Escalamiento horizontal
  - Partición de cargas de trabajo
- Virtualización y elasticidad
  - Automatización de operaciones
  - Aprovisionamiento bajo demanda

# NETFLIX

# OSS



# Patterns in Microservices Architecture

## API Gateway

1. Choose to build the application as a set of micro-services.
2. Decide how the application client's will interact with the micro services.
3. With a monolithic application there is just one set of (typically replicated, load-balanced) endpoints.
4. In a micro services architecture, however each micro services exposes a set of what are typically fine-grained endpoints.

## Service Registry

1. Service registry helps to determine the location of service instances to send request to the corresponding service
2. Here, we have used the Netflix Eureka to register a service that are available to be registered in service registry server and it can be identified through the router.

## Service Discovery

1. In a monolithic application, services invoke one another through language-level method or procedure calls.
2. But, in a modern micro services based application typically runs in a virtualized environments where the number of instances of a service and their locations changes dynamically.
3. Each service can be identified using router that are registered with service registry server.

# Micro servicios Netflix OSS



EUREKA    ARCHAIUS    HYSTRIX    TURBINE



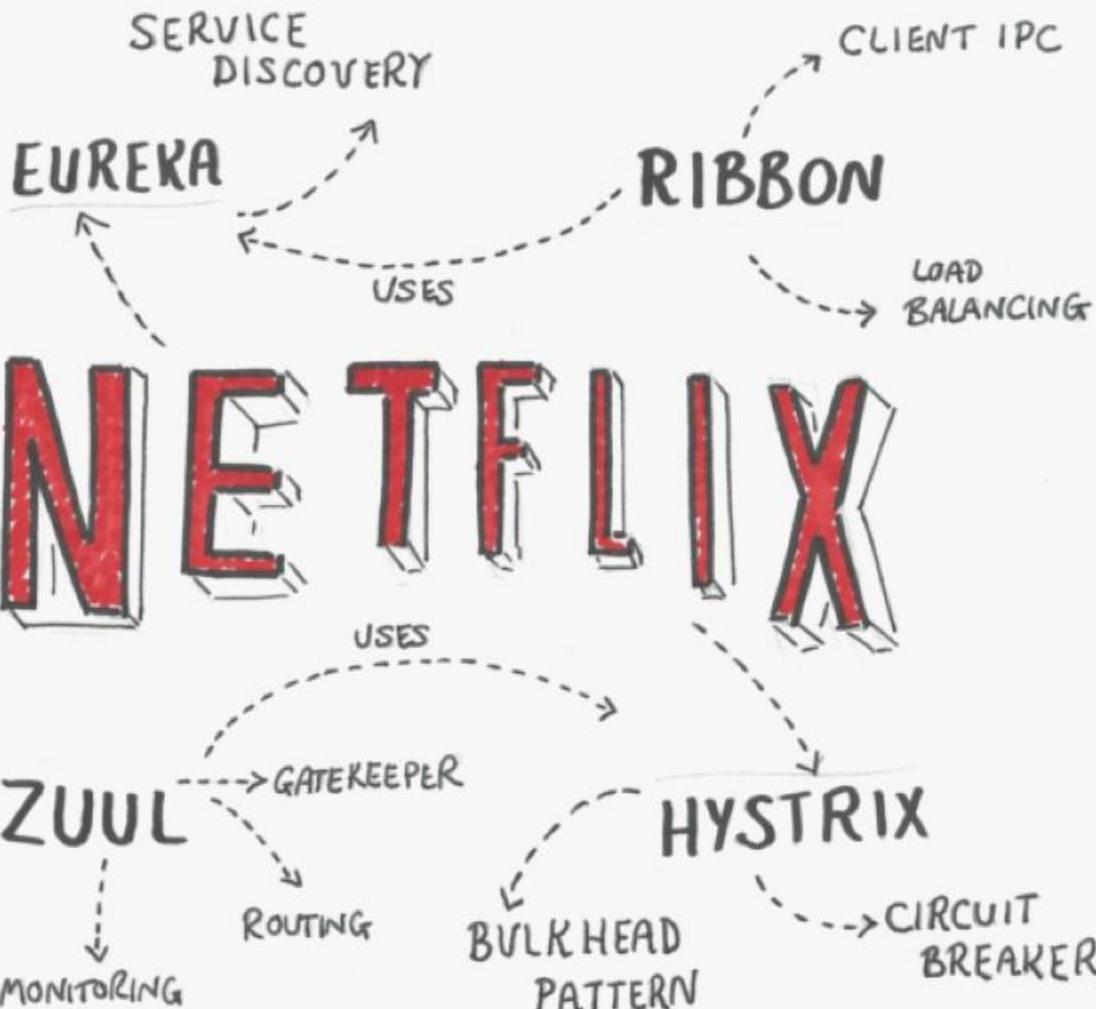
ZUUL    BLITZ4J    RIBBON

| Operations Component              | Netflix, Spring, ELK                     |
|-----------------------------------|--|
| Service Discovery server          | Netflix Eureka                           |
| Dynamic Routing and Load Balancer | Netflix Ribbon                           |
| Circuit Breaker                   | Netflix Hystrix                          |
| Monitoring                        | Netflix Hystrix dashboard and Turbine    |
| Edge Server                       | Netflix Zuul                             |
| Central Configuration server      | Spring Cloud Config Server               |
| OAuth 2.0 protected API's         | Spring Cloud +<br>Spring Security OAuth2 |
| Centralised log analyses          | Logstash, Elasticsearch, Kibana (ELK)    |

<https://netflix.github.io/>



Eureka



Ego Iván Oliveros Acosta @scalapp.co



HYSTRIX



# Microservicios Netflix OSS

- **Eureka**
  - Servidor para el registro y la localización de los microservicios.
  - Balanceo de carga y tolerancia a fallos.
  - Su función principal es registrar las diferentes instancias de microservicios existentes.
- **Cómo funciona Eureka**
  - Cada microservicio, durante su arranque, se comunica con el servidor Eureka para notificar que está disponible y luego continuará notificando su estado cada 30 segundos.



## Eureka

# Microservicios Netflix OSS

- **Qué aporta Eureka**
  - Abstracción de la localización física de los microservicios
  - Conocimiento del estado del ecosistema de microservicios actualizado en todo momento mediante un dashboard
  - Configuración como clúster para aumentar la tolerancia a fallos.
  - Soporte a multirregión
  - Integración con los servicios de AWS

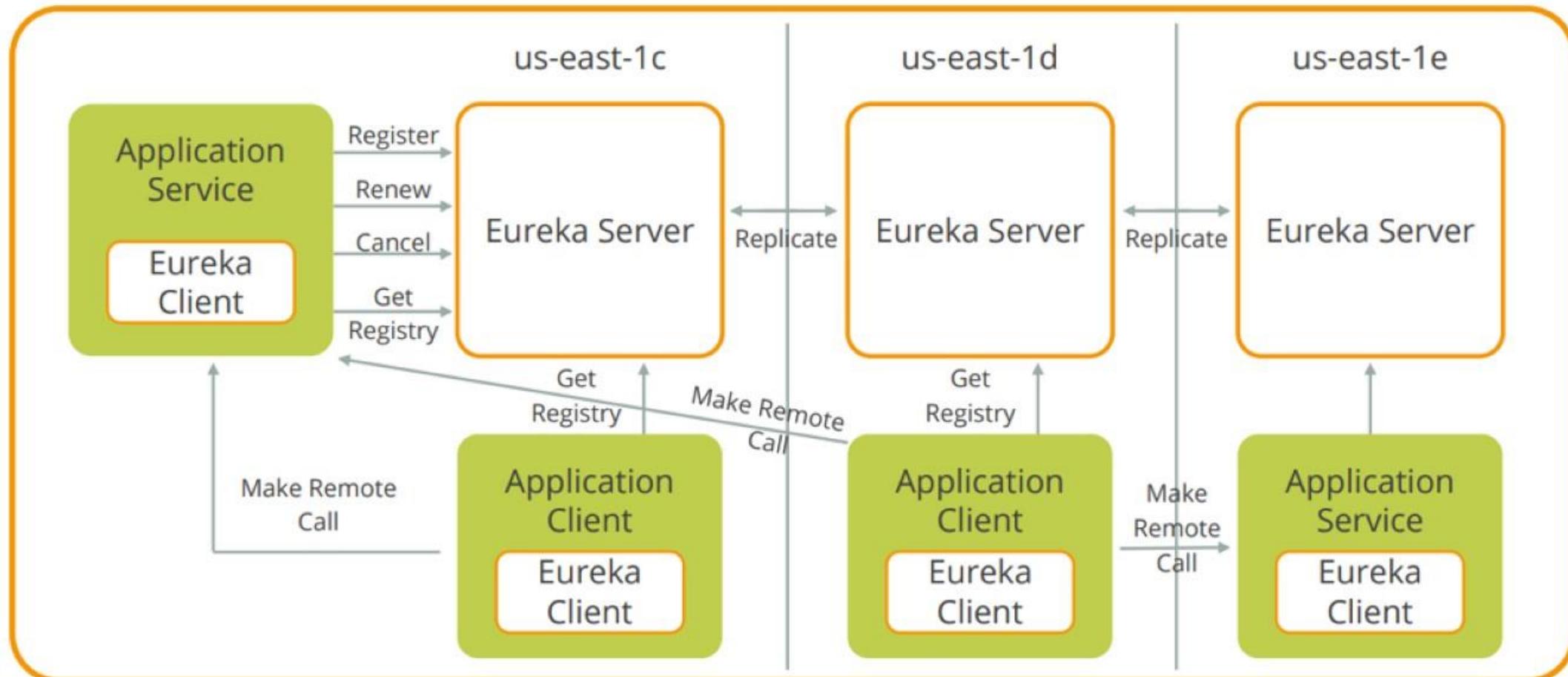


## Eureka

# Microservicios Netflix OSS



Qué aporta Eureka



## Registry: Autodescubrimiento con Eureka

Servidor Eureka

```
@SpringBootApplication
@EnableEurekaServer
@EnableDiscoveryClient
public class EurekaServerApplication {

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

}
```

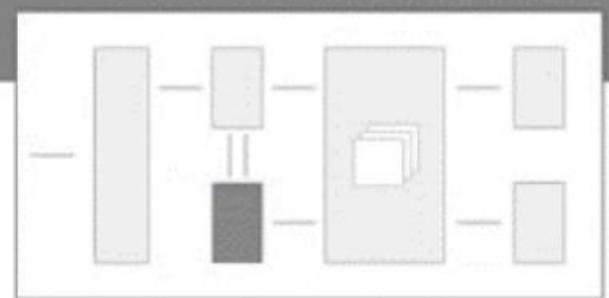
Cliente Eureka

```
@Configuration
@EnableAutoConfiguration
@EnableEurekaClient
@RestController
public class EurekaClientApplication {

    @RequestMapping("/")
    public String home() {
        return "Hello World";
    }

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

}
```



As long as Spring Cloud Netflix and Eureka Core are on the classpath any Spring Boot application with `@EnableEurekaClient` will try to contact a Eureka server on `http://localhost:8761` (the default value of `eureka.client.serviceUrl.defaultZone`):

# Microservicios Netflix OSS

- **Zuul**
  - Se puede definir como un proxy inverso.
  - Permite enrutar y filtrar nuestras peticiones.
  - Actúa como un punto de entrada a nuestros servicios.
  - Se encarga de solicitar una instancia de un microservicio concreto de Eureka.
  - Enruta hacia el servicio que consumimos.



# Microservicios Netflix OSS

- **Cómo funciona Zuul**
  - Será configurado como el punto de entrada al ecosistema de microservicios.
  - Es el encargado de enrutar y balancear las peticiones que se reciban de los microservicios.



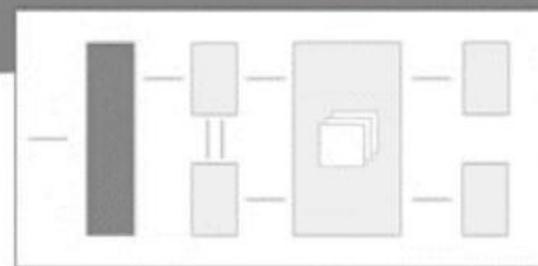
# Microservicios Netflix OSS

## Qué aporta Zuul

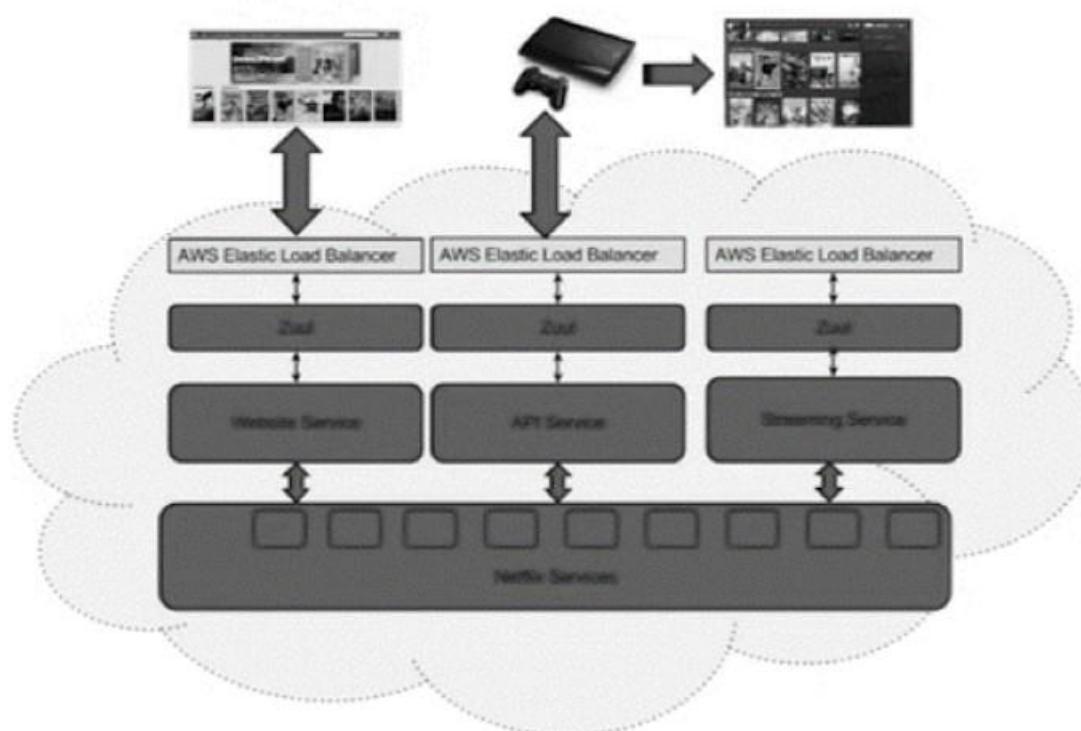
- Proporciona un sistema que reacciona rápidamente cambiando el comportamiento ante diferentes situaciones.
- Proporciona filtros para gestionar diferentes situaciones:
  - Filtros de autenticación
  - Filtros de seguridad
  - Filtros de monitoreo
  - Filtros para enrutados dinámico
  - Filtros para test de carga
  - Filtros para gestión de recursos
  - Filtros para la gestión de multirregión



## Edge Service: Zuul



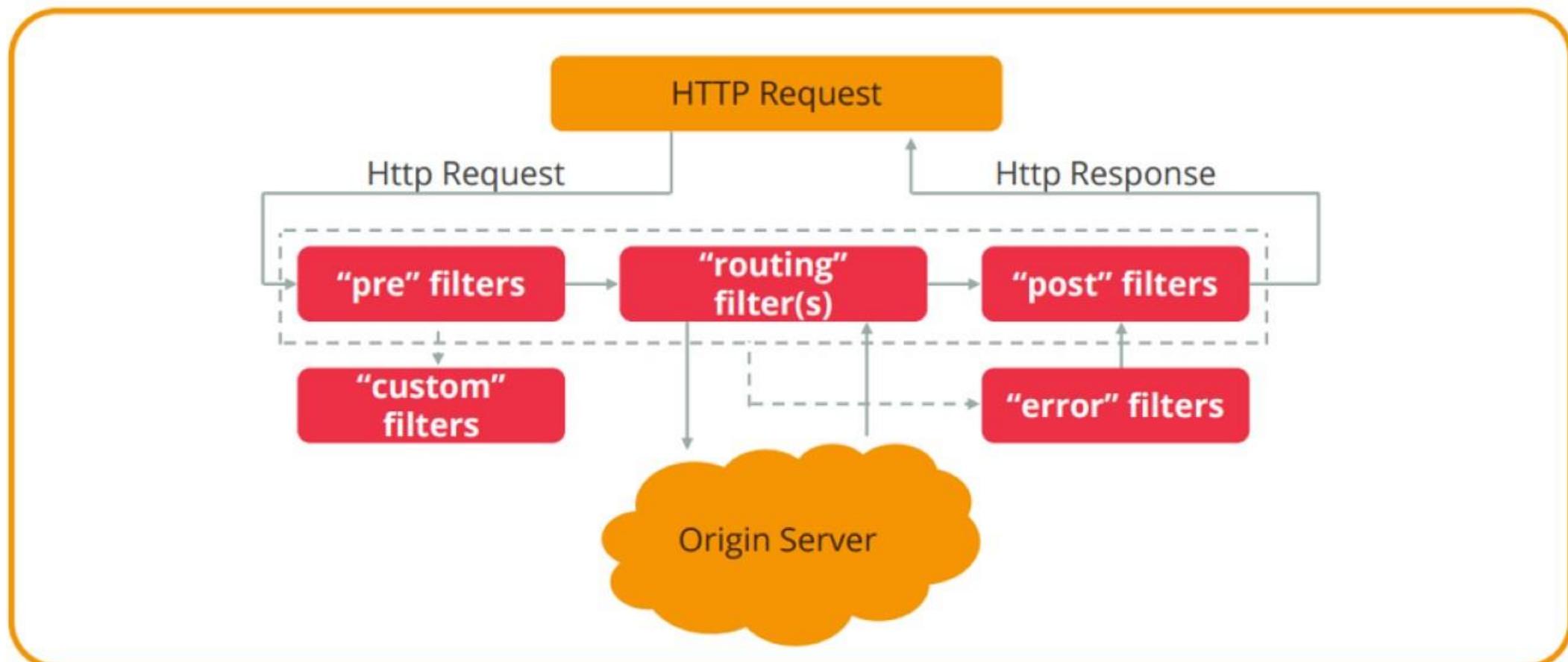
- Zuul proporciona una puerta de entrada única a todo nuestro ecosistema de microservicios para peticiones desde dispositivos móviles o sitios web. Como edge service, Zuul está construido para permitir enruteado dinámico, monitorización, seguridad y robustez.
- Zuul aparte de un edge service permite añadir filtros en tiempo real que permiten hacer un balanceo personalizado en función de un criterio propio sin parar el servicio.



# Microservicios Netflix OSS



Qué aporta Zuul



# Microservicios Netflix OSS

- **Hystrix**
  - Implementa el patrón CircuitBreaker.
  - Permite gestionar las interacciones entre servicios en sistemas distribuidos con lógica de latencia y tolerancia a fallos.
  - Ofrece librería para aislar puntos de acceso a sistemas remotos.
  - Mejora la fiabilidad global del sistema.
- **Cómo funciona Hystrix**
  - Encapsula las peticiones a sistemas “externos” para gestionar aspectos tales como timeout, estadísticos y propagación de errores.

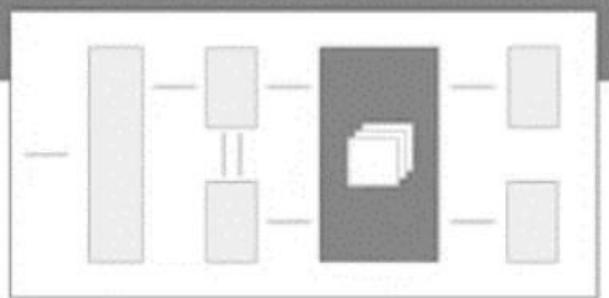
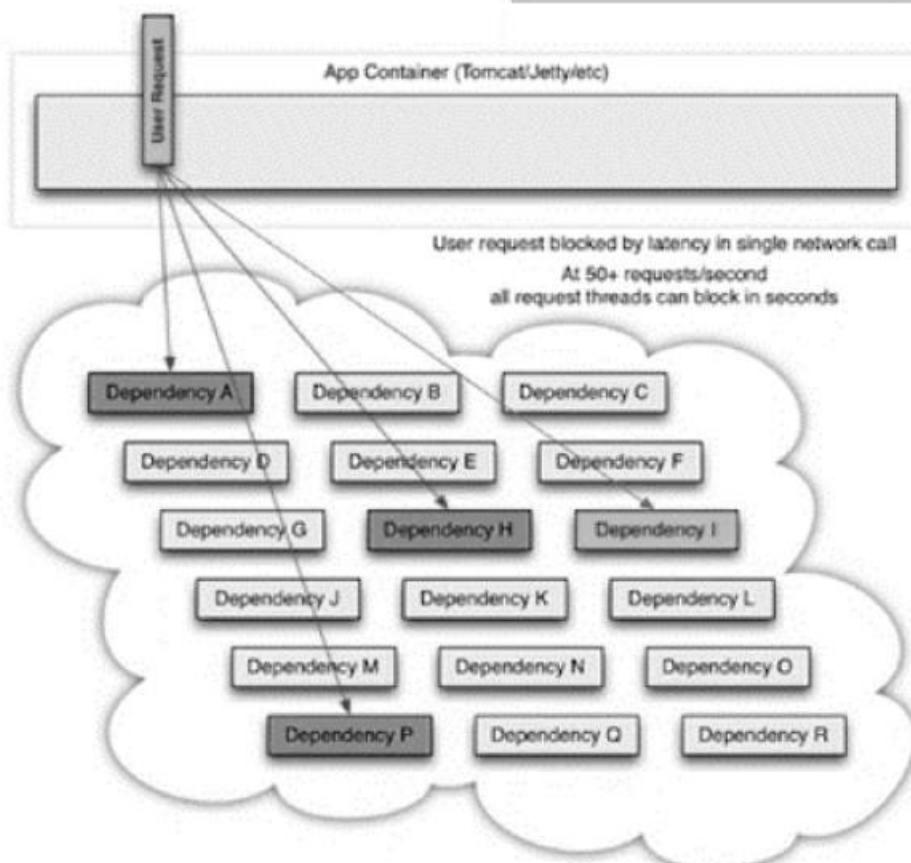
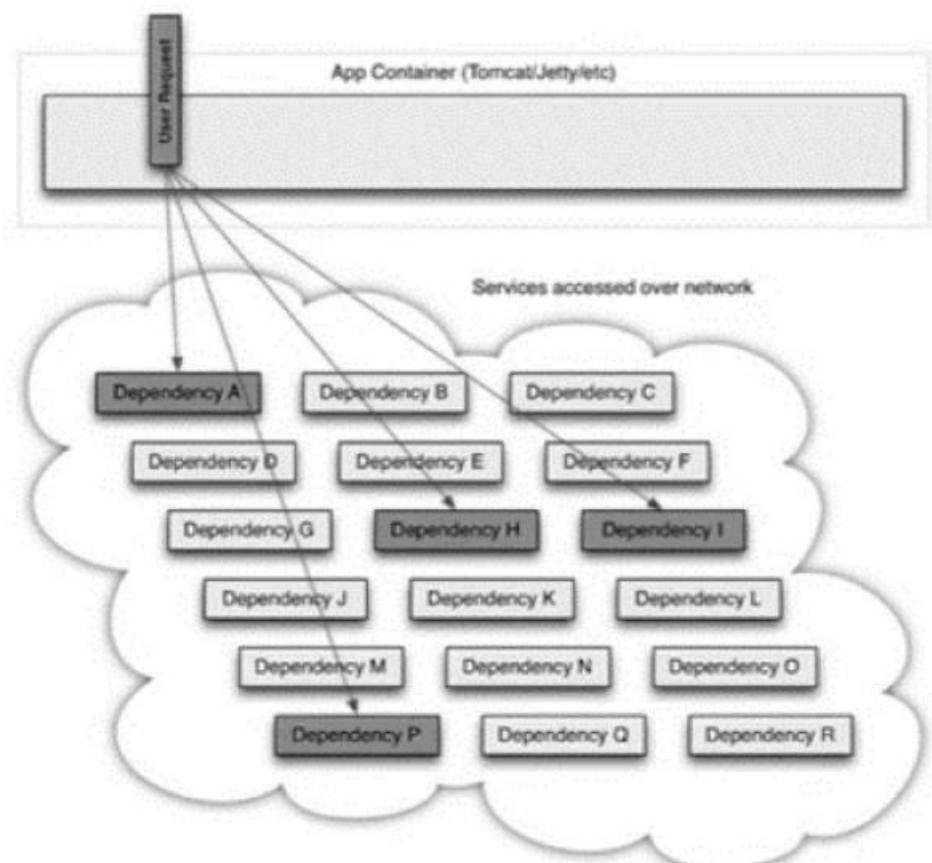


# Microservicios Netflix OSS

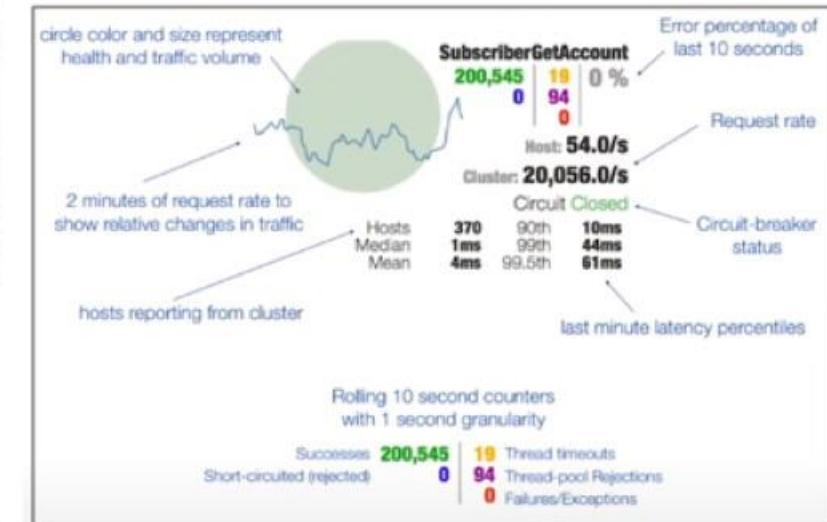
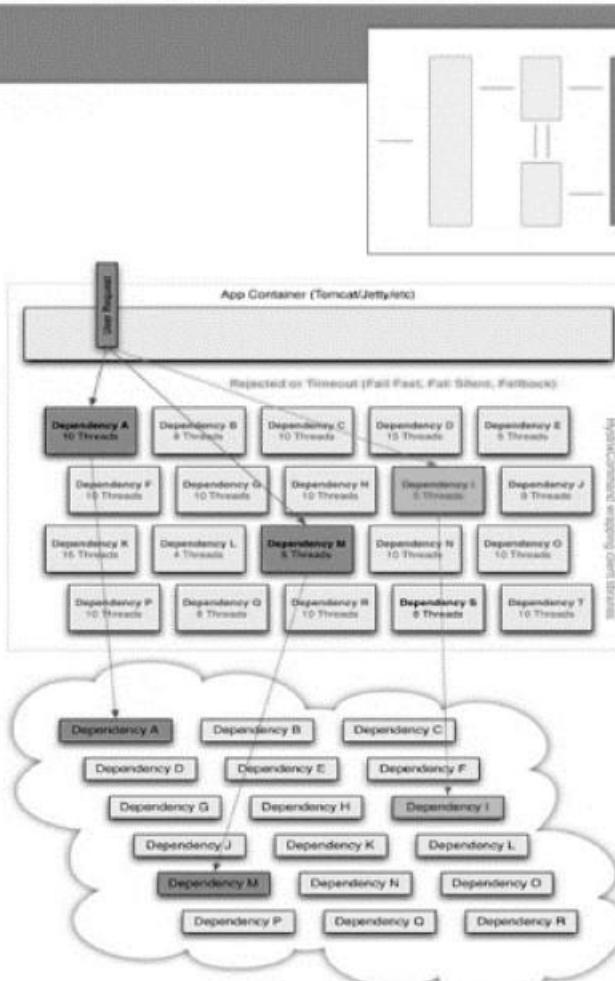
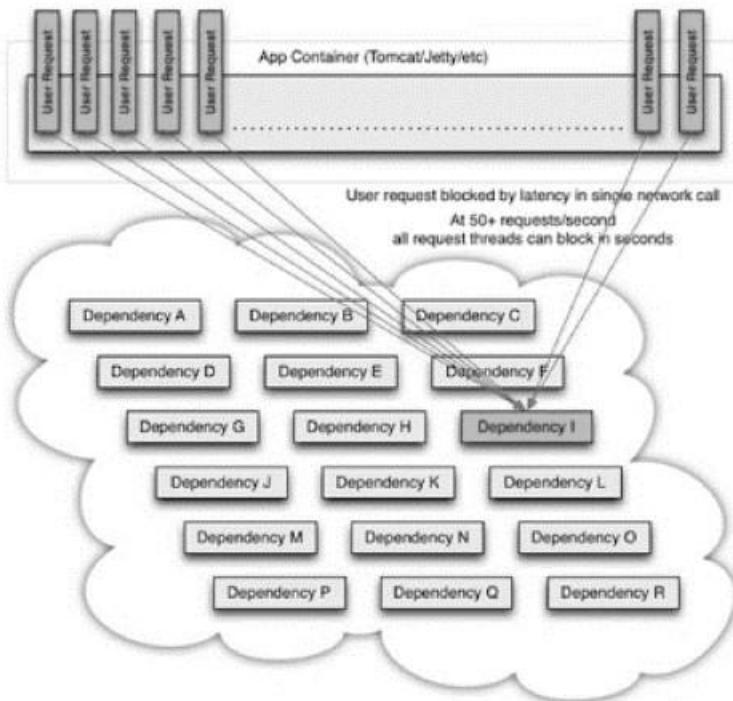
- **Qué aporta Hystrix**
  - Encapsula las peticiones.
  - Cancela las peticiones que exceden el timeout.
  - Gestiona pool de hilos para cada petición a sistema “externo”.
  - Gestiona la propagación de errores en cascada.
  - Proporciona un dashboard que integra las métricas capturadas.



## Circuit Breaker: Hystrix



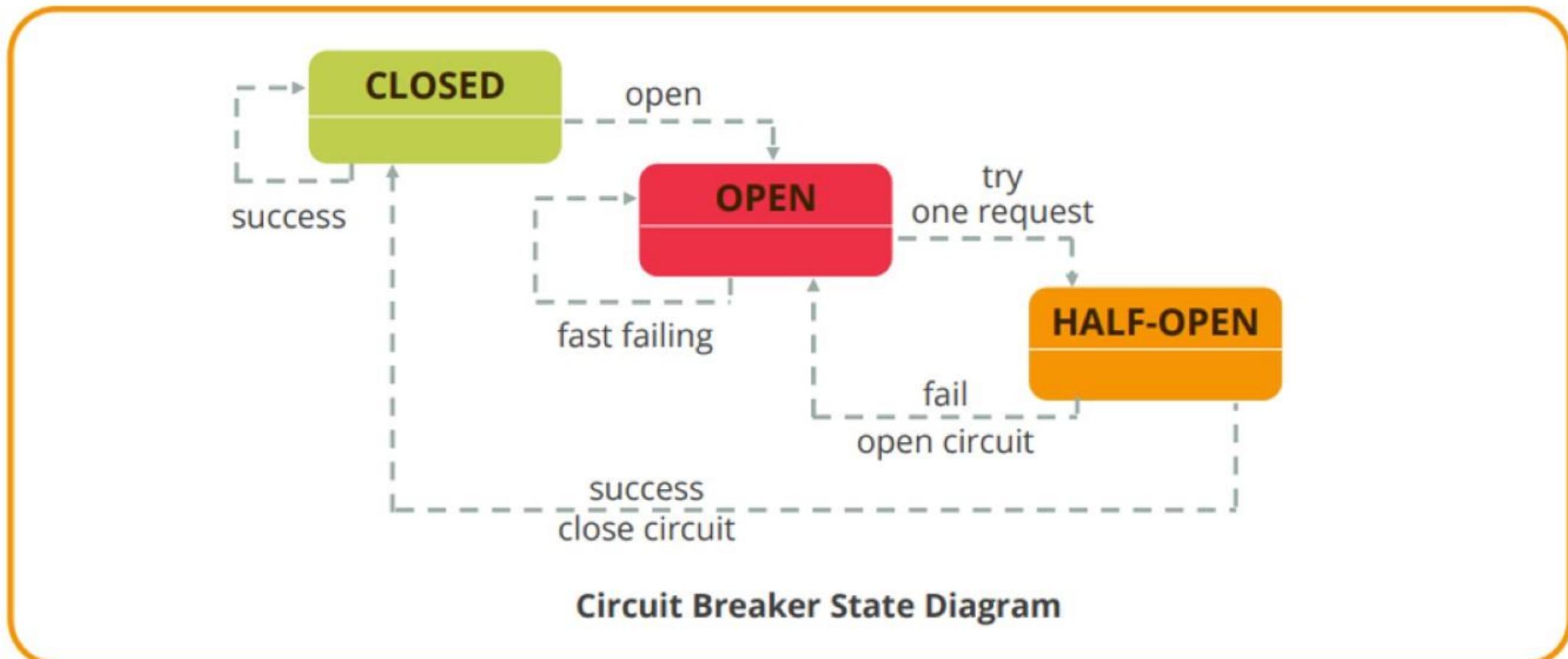
## Circuit Breaker: Hystrix



# Microservicios Netflix OSS



Qué aporta Hystrix



# Microservicios Netflix OSS

- **Ribbon**
- Es una librería diseñada para la comunicación entre procesos en la nube que realiza balanceo de carga en el lado del cliente.
- Tiene funcionamiento integrado con Eureka para el descubrimiento de las diferentes instancias de un microservicio.



# Microservicios Netflix OSS

## Cómo funciona Ribbon

- Identifica el microservicio por el nombre con que se registra en Eureka sin ser necesario identificar la máquina, o ip o puerto donde está el microservicio.
- Identifica cuántas instancias existen de un mismo microservicio y en qué máquinas.
- Ejecuta el algoritmo de balanceo de carga Round Robin para determinar qué instancia de microservicio invocar.



# Microservicios Netflix OSS

## Qué aporta Ribbon

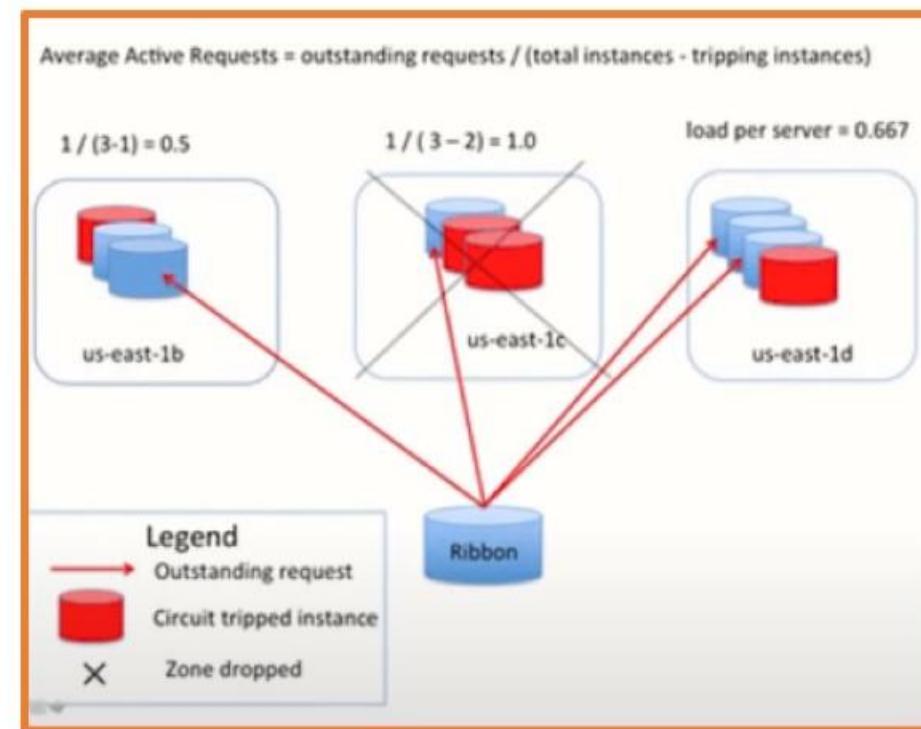
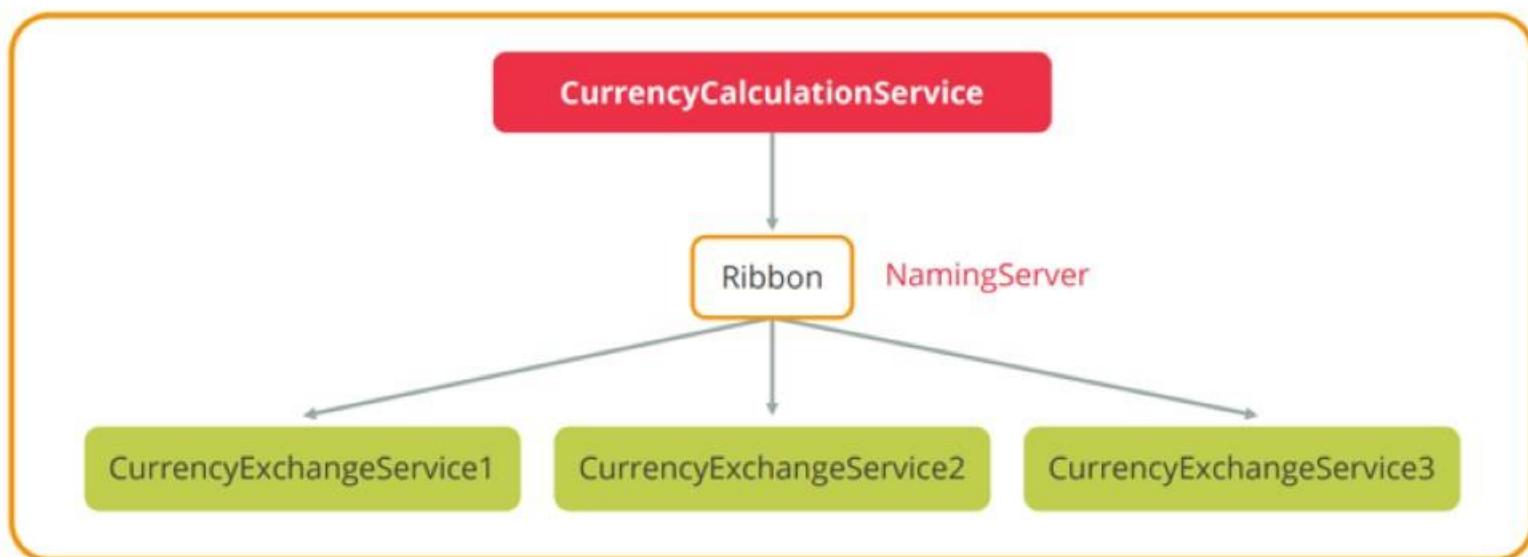
- Proporciona abstracción del número de instancias existentes.
- Posee diferentes implementaciones de algoritmos de balanceo.
- Permite la gestión de zonas en entornos Cloud.
- Distingue diversas zonas.
- Mediante la integración con Eureka detecta qué instancias de microservicios están caídas.
- Se integra con Hystrix para encapsular las peticiones.



# Microservicios Netflix OSS

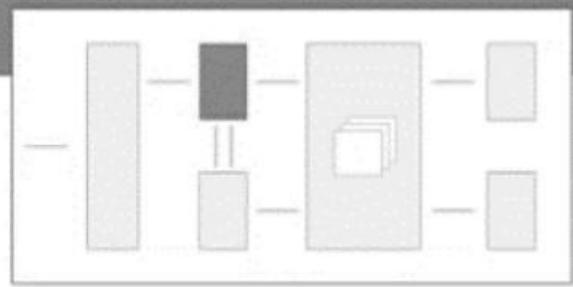


## Qué aporta Ribbon

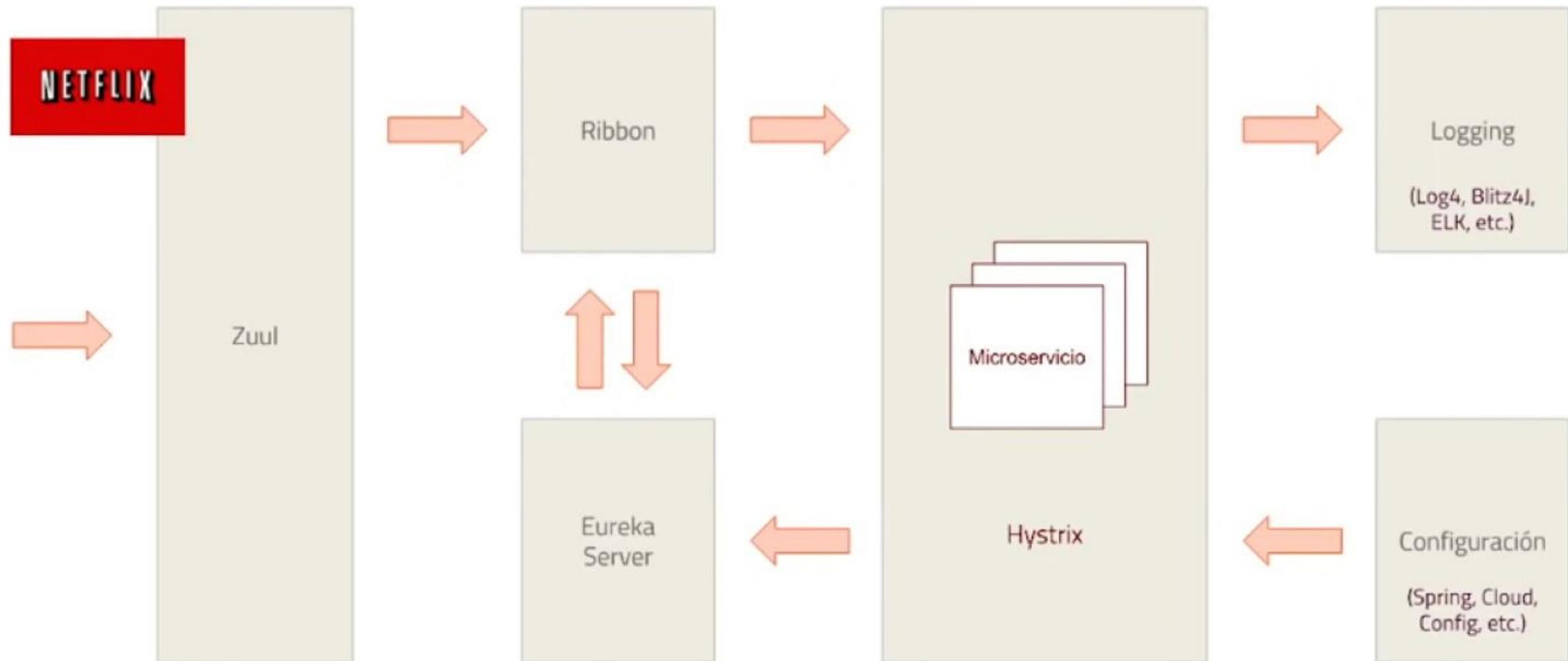


## Load Balancer: Ribbon

Ribbon es un balanceador de carga en el lado cliente. Se encarga de comunicarse con Eureka para saber cuántos y dónde están levantados los diferentes microservicios y así aplicar un algoritmo Round Robin para balancear la carga entre las diferentes instancias de los microservicios levantados.



## Vista general en una arquitectura basada en microservicios: Netflix



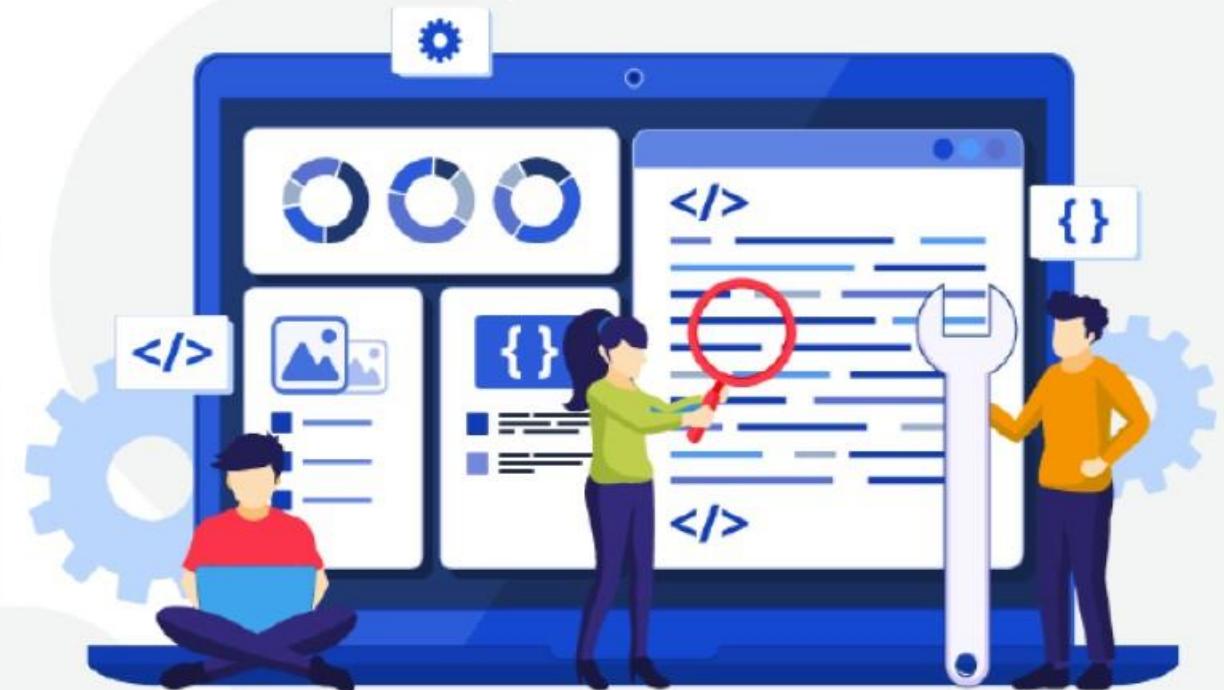


CICLO 4a

[FORMACIÓN POR CICLOS]

## Desarrollo de **APLICACIONES WEB**

**Introducción a  
microservicios  
con Spring Boot**



**UNIVERSIDAD  
DE ANTIOQUIA**

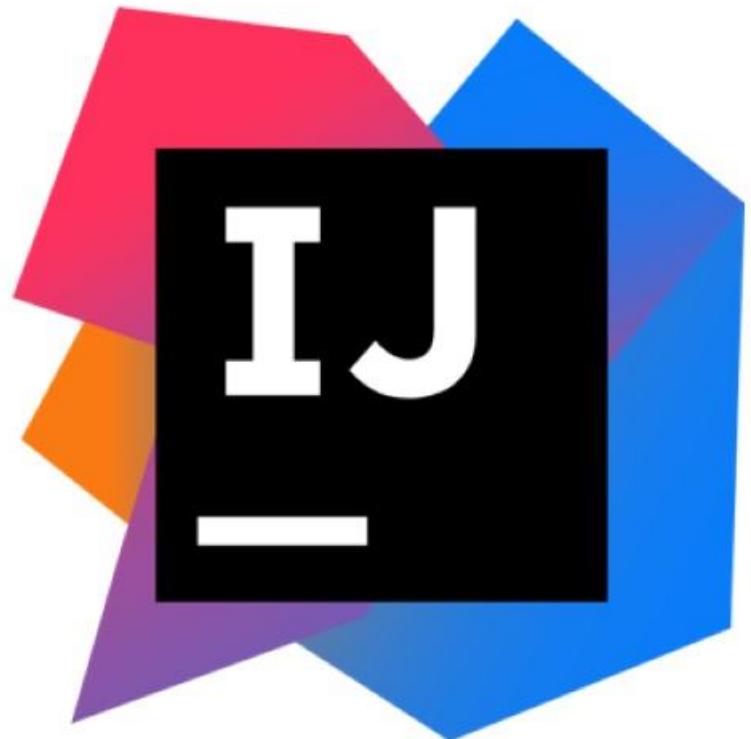
Facultad de Ingeniería

Diego Iván Oliveros Acosta @scalapp.co



- Windows /macOS/ Linux
- Ultimate: For web and enterprise development (Free 30-day trial)
- Community: For JVM and Android development
- **Learning or teaching programming:** IntelliJ IDEA Edu with special support for learners and educators

<https://www.youtube.com/user/intellijideavideo>



- Cuando pretendemos crear microservicios sobre un IDE en particular, como por ejemplo IntelliJ IDE, **requerimos un plug-in** para tener un asistente.
- **El plug-in** que utilizaremos será “**Spring Assistant**”, el cual podemos referenciar para instalar directamente desde el **IDE** mediante la opción:  
**File/Settings/Plugins**
- y le damos la opción de “Install”.



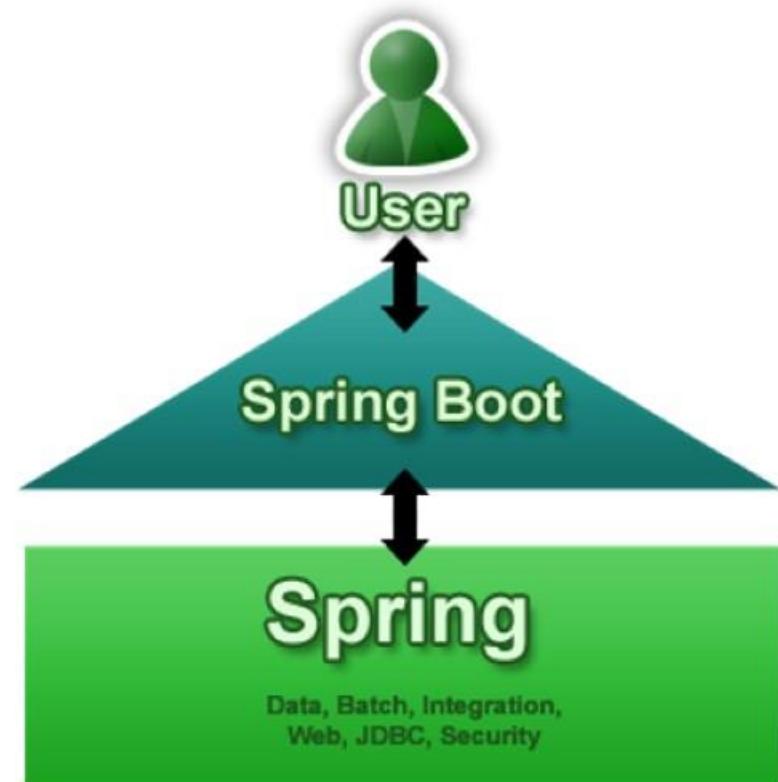
**Spring Boot®**

<https://spring.io/projects/spring-boot>

<https://adoptium.net/?variant=openjdk11>

# Microservicios con Spring Boot

- **El objetivo de Spring Boot:** es proporcionar un conjunto de herramientas para construir de manera rápida y ágil aplicaciones Spring, fáciles de crear, configurar y mantener en el tiempo.
- **Spring Boot:** facilita la creación de aplicaciones basadas en **Spring**, autónomas y del nivel de producción para ejecutar.
- **Poca configuración:** Básicamente es posible poner en funcionamiento una aplicación de Spring con muy poco trabajo





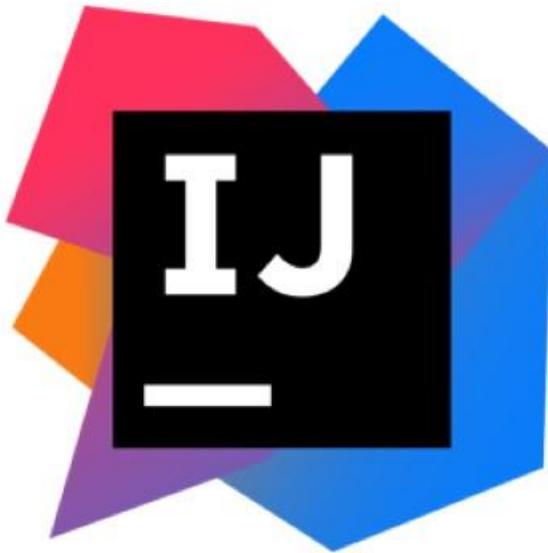
```
gradle myTask  
  
$ gradle :mySubproject:taskName  
$ gradle mySubproject:taskName  
  
$ gradle test  
  
gradle test deploy //Ejecutando multiples tareas  
gradle dist --exclude-task test gradle dist -x test  
gradle test --continue  
gradle build //assembling all outputs and running all checks  
gradle tasks --all  
gradle myTask --scan
```

<https://gradle.org/>

# Microservicios con Spring Boot

## Requisitos previos

- Para sacar el máximo provecho de la información de este curso debemos estar familiarizados y tener instalado lo siguiente:
  - Java Development Kit 1.8 (JDK)
  - IntelliJ IDEA para 2020.2 o superior
  - Gradle 6.0 o superior



- Para instalar el **Plug-in de “Spring Assistant”** sobre el IDE de IntelliJ IDEA diríjase al siguiente link:  
<https://plugins.jetbrains.com/plugin/10229-spring-assistant/versions>
- Presione el botón “Get” y ajuste la compatibilidad según su versión de IntelliJ IDEA y presione “Download” en la última versión o la que prefiera.
- Con esto tendrá un archivo descargado con la extensión .zip que contendrá el plugin.

Framework integration

# Spring Assistant



Development Team @ 1Ton Technologies

# Microservicios con Spring Boot

JET BRAINS Marketplace Edu Courses Themes Plugin Ideas Build Plugins Sign In Q

Framework integration

## Spring Assistant

★★★★★ ★  
Development Team @ 1Ton Technologies

Overview Versions Reviews

Android Studio  
IntelliJ IDEA Community  
IntelliJ IDEA Educational  
IntelliJ IDEA Ultimate

Compatibility with IntelliJ IDEA Community ▲

Get Compatible with IntelliJ IDEA (Ultimate, Community, Educational), Android Studio

Stable EAP Nightly

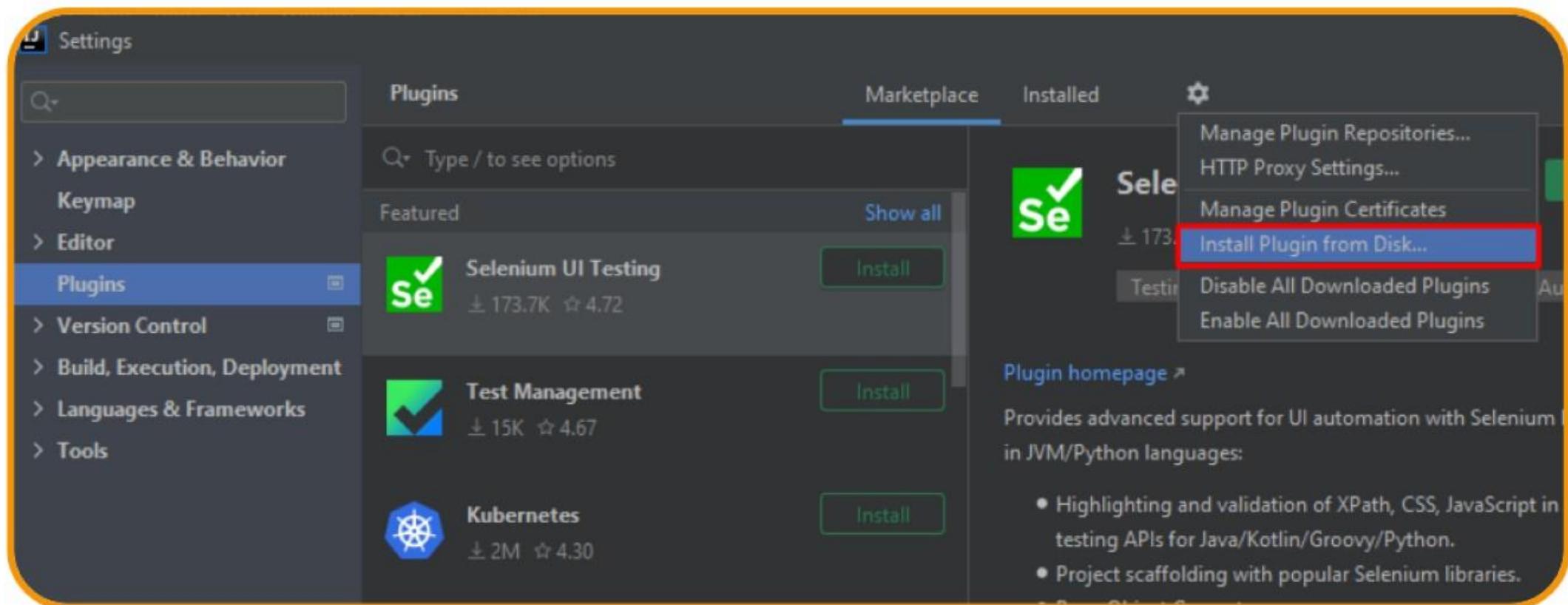
### Version History

| Version | Compatibility with IntelliJ IDEA Ultimate ▾  | Update Date  |
|---------|--|--------------|
| 0.12.0  | 2017.2 — 2019.3.5  | Apr 11, 2018 |
| 0.11.0  | To have full functionality you have to accept <a href="#">Plugin Marketplace Agreement</a> . | Download X   |

Diego Iván Oliveros Acosta @scalapp.co

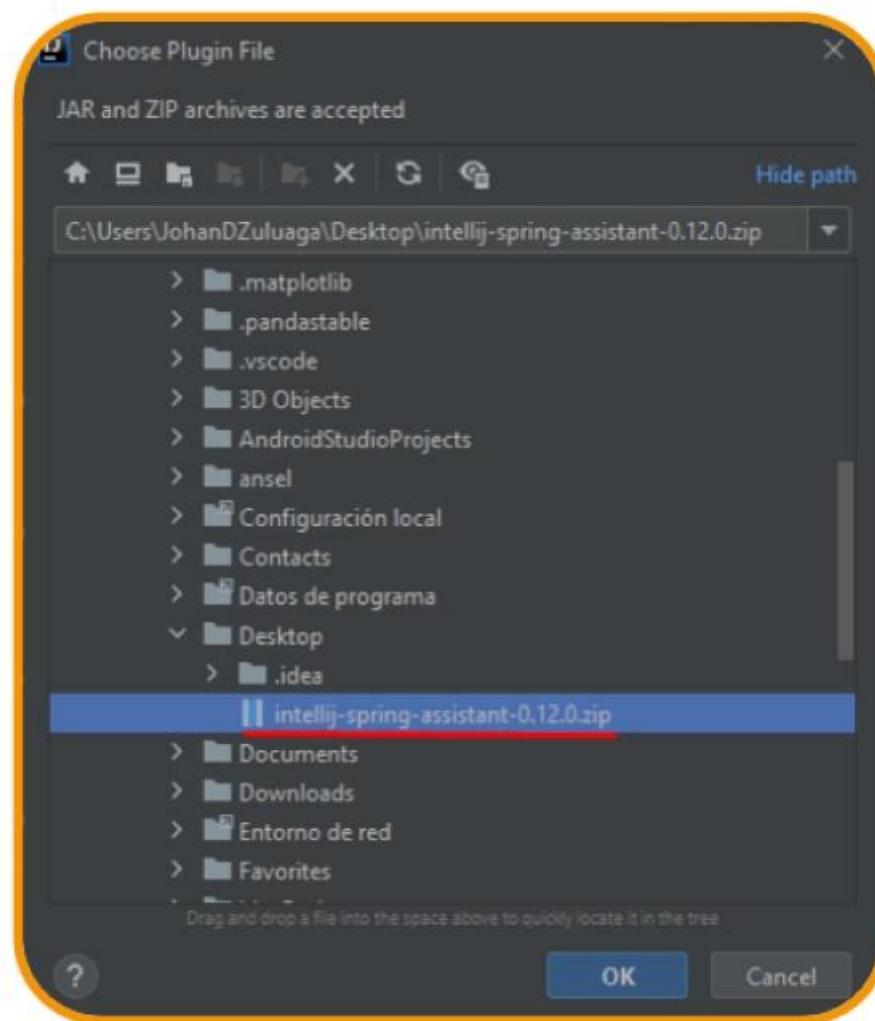
# Microservicios con Spring Boot

- Diríjase a IntelliJ y presione las teclas Ctrl + Alt + s, luego presione el botón del engranaje y la opción “Install Plugin from Disk...”

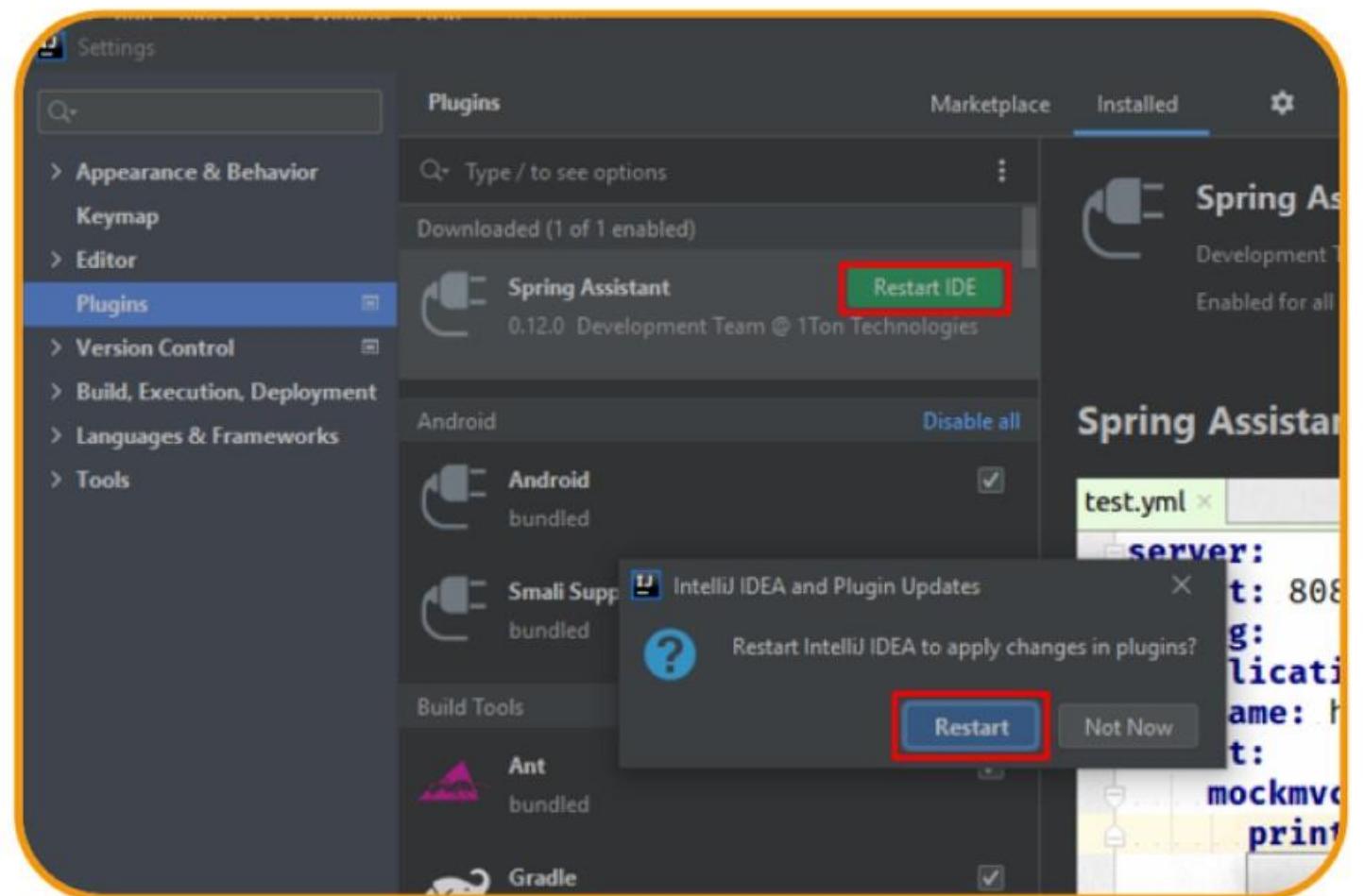


# Microservicios con Spring Boot

- Busque el archivo descargado  
y presione “OK”

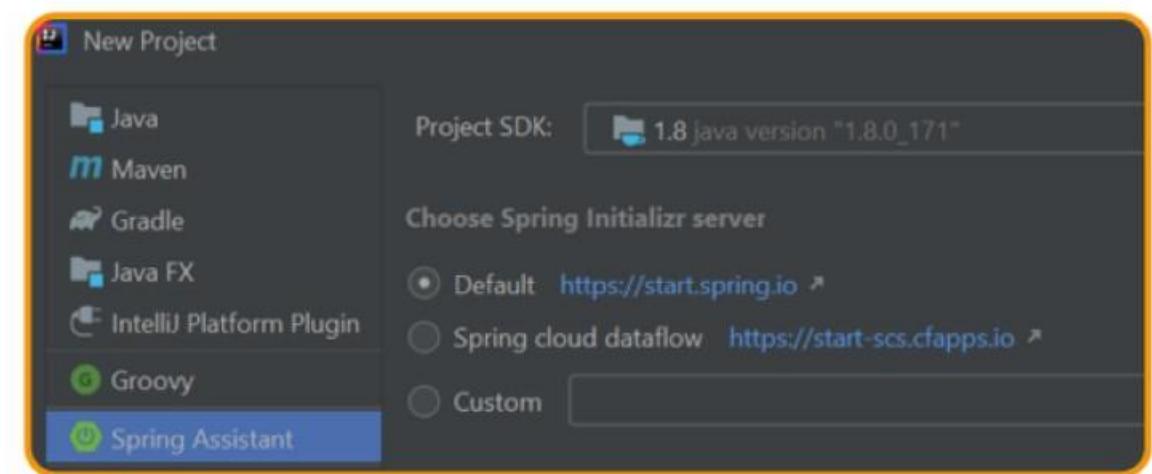
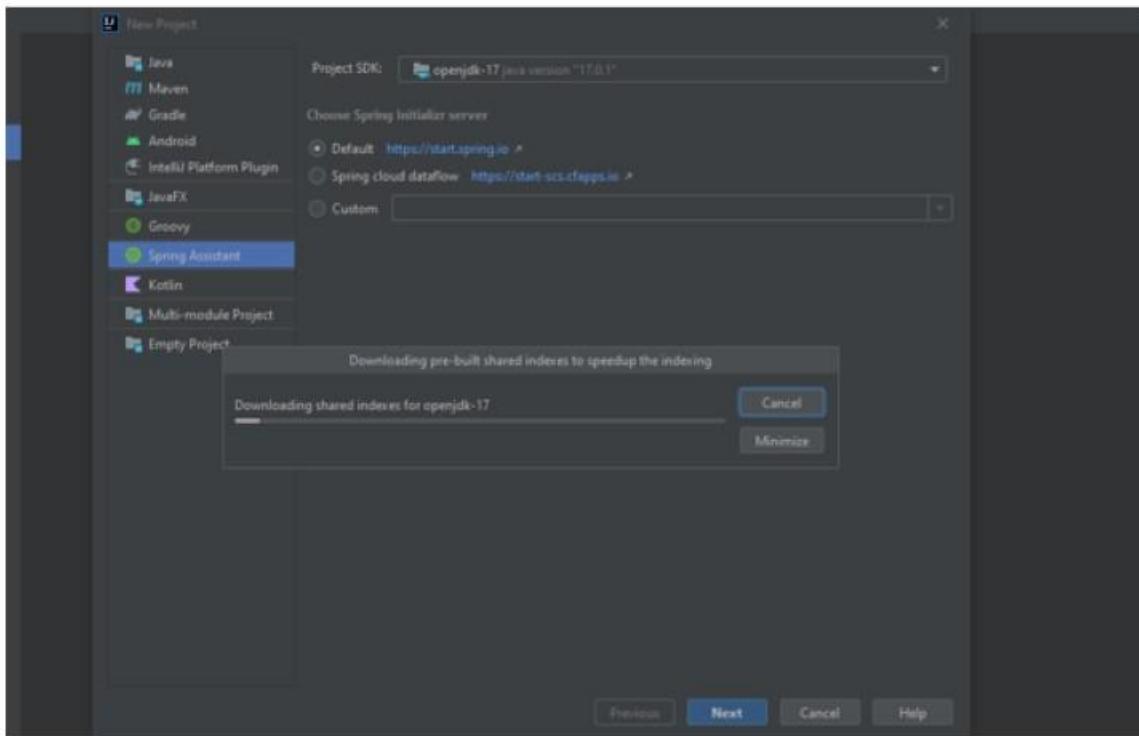


- Presione “Restart IDE” y luego “Restart”



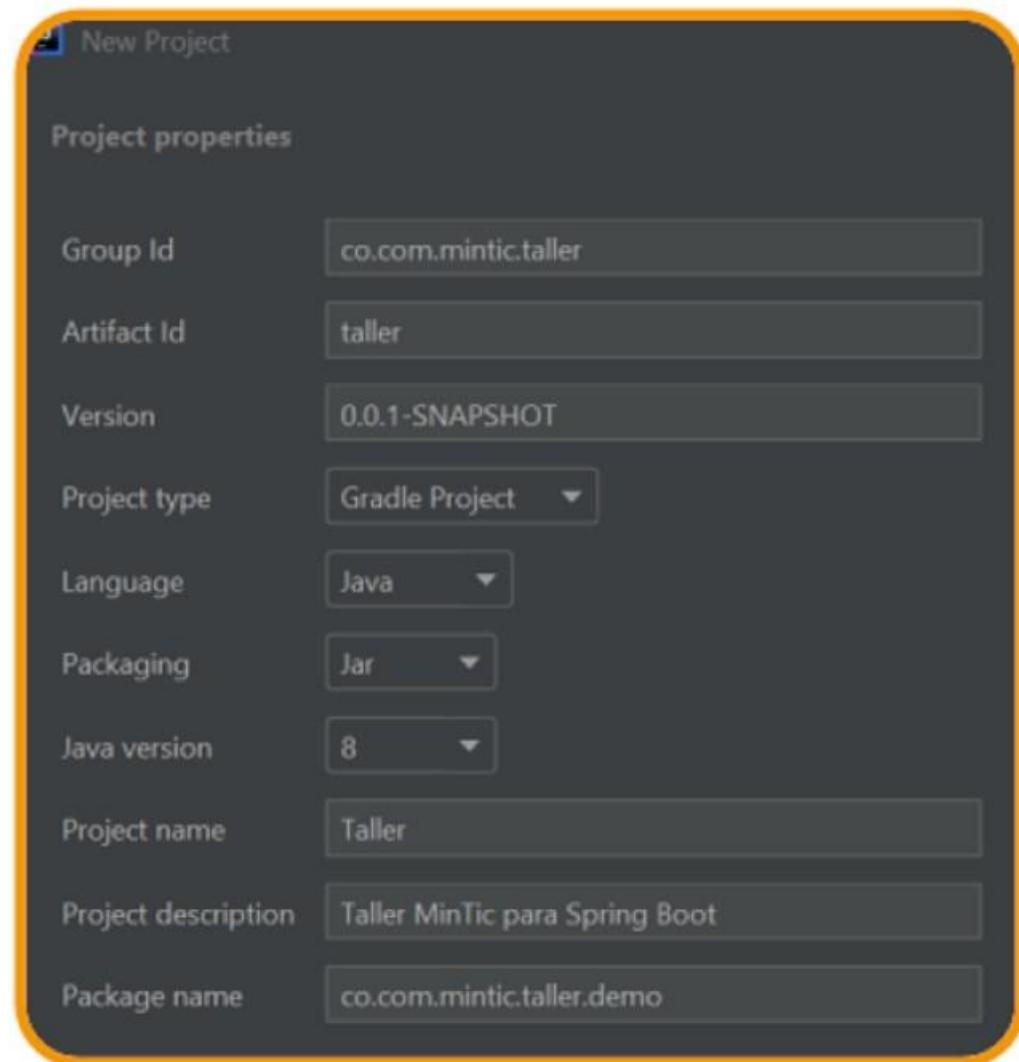
# Microservicios con Spring Boot

- Para crear un nuevo proyecto de Spring Boot sobre el IDE de IntelliJ seleccionamos la opción File/New/ Project. Seleccionamos luego “Spring Assistant” y después “Next”:



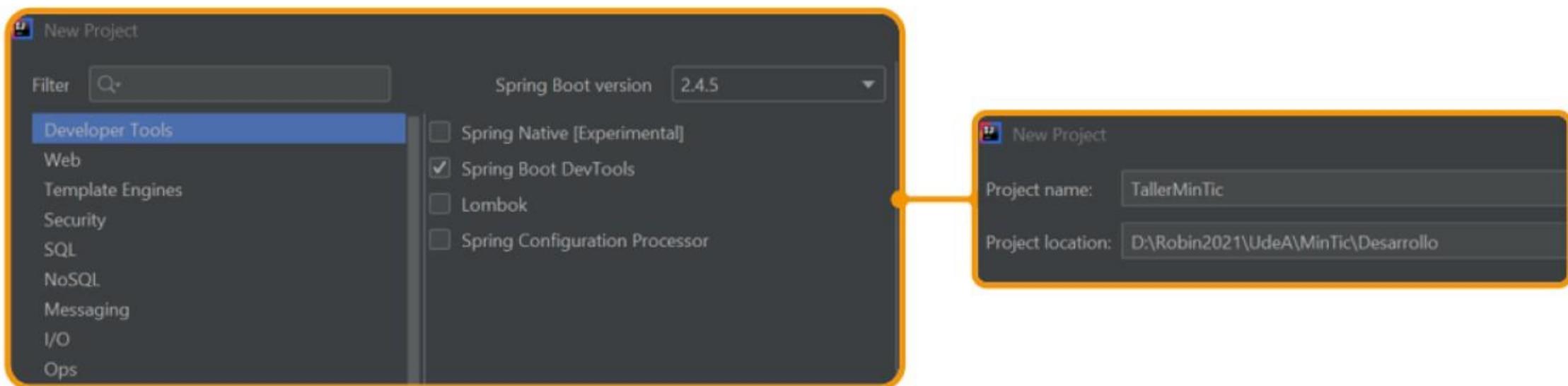
# Microservicios con Spring Boot

- Llenamos las propiedades del nuevo proyecto tal como se requiere para un taller de ejemplo y procedemos a dar “Next”:



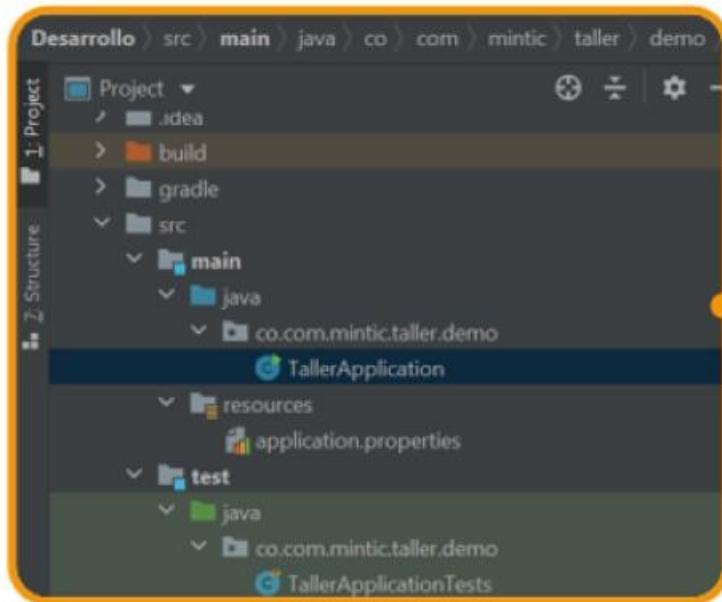
# Microservicios con Spring Boot

- Seleccionamos la opción de herramientas de desarrollo “Spring Boot DevTools” y luego continuamos con “Next”. Por último, finalizamos el asistente poniendo el nombre del nuevo proyecto en la ruta en que quedará:



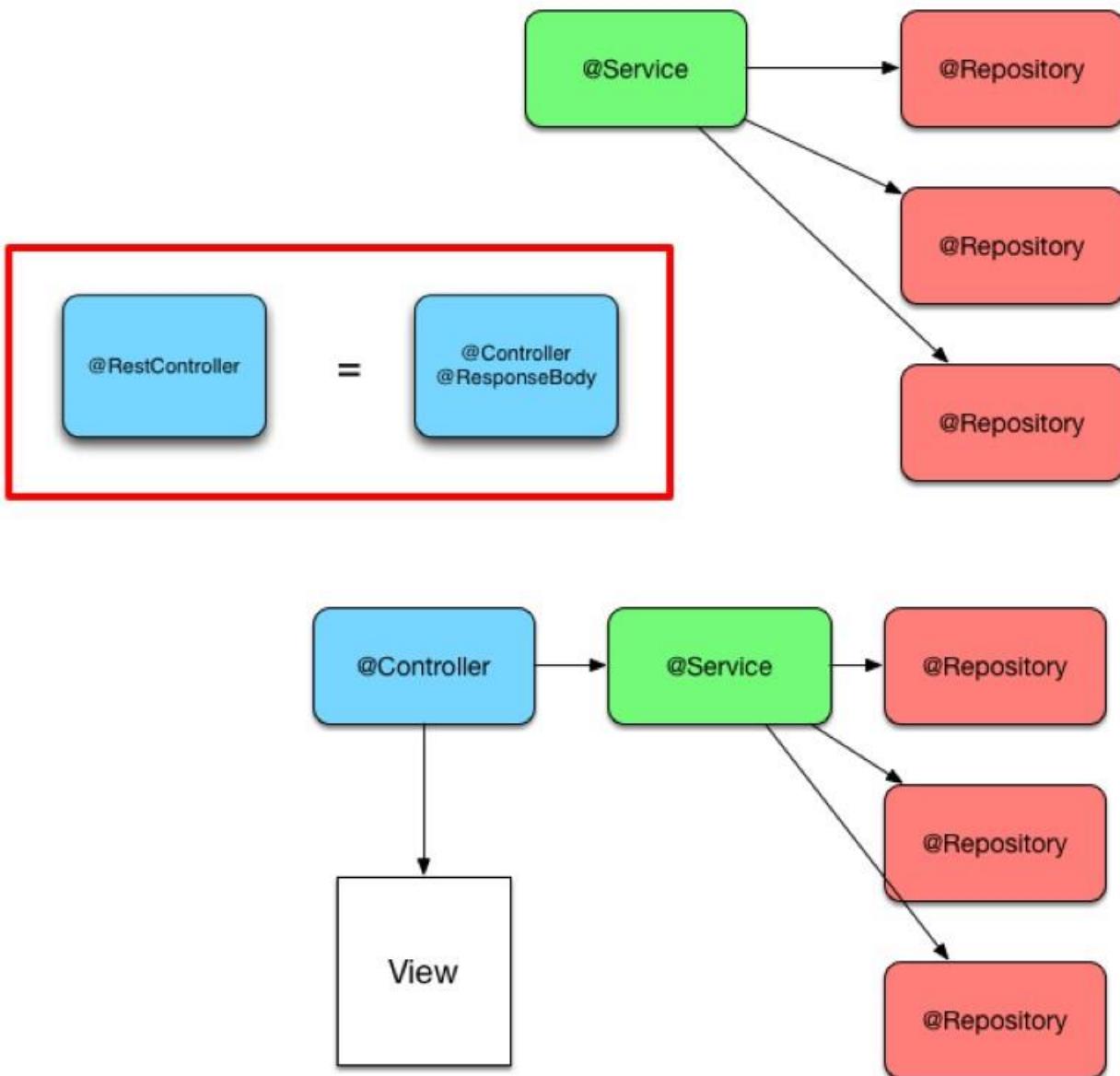
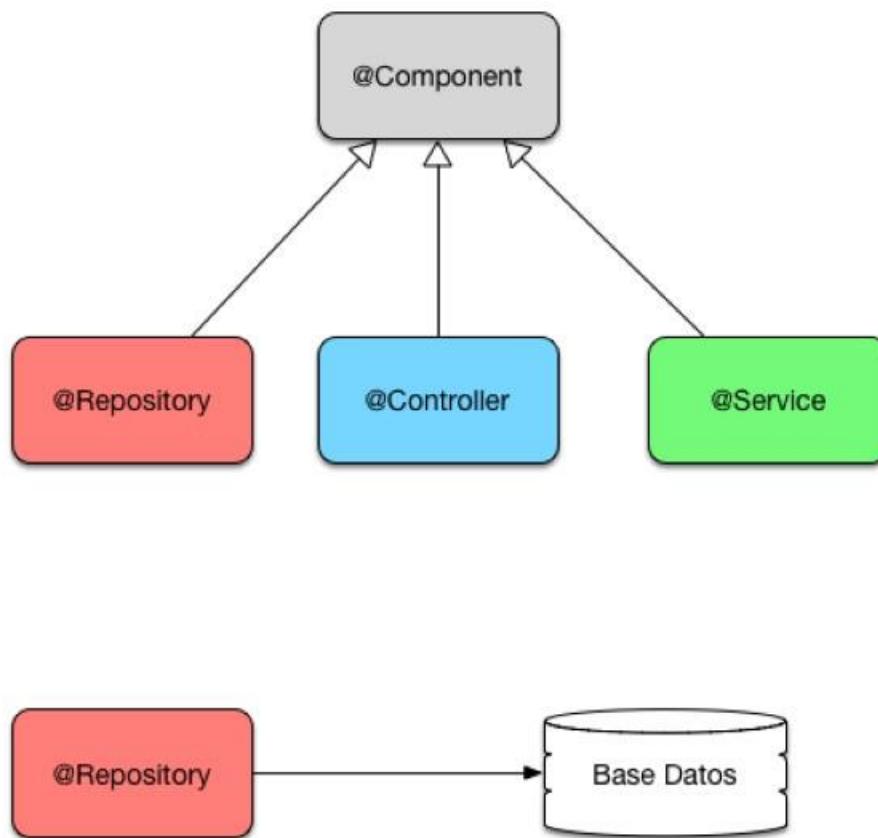
# Microservicios con Spring Boot

Una vez finalizado el paso a paso de “Spring Assistant”, abrimos el nuevo proyecto para verificar el arquetipo inicial para la construcción de los microservicios basados en el framework de Spring Boot y revisamos las librerías de Spring Boot en el archivo build.gradle:



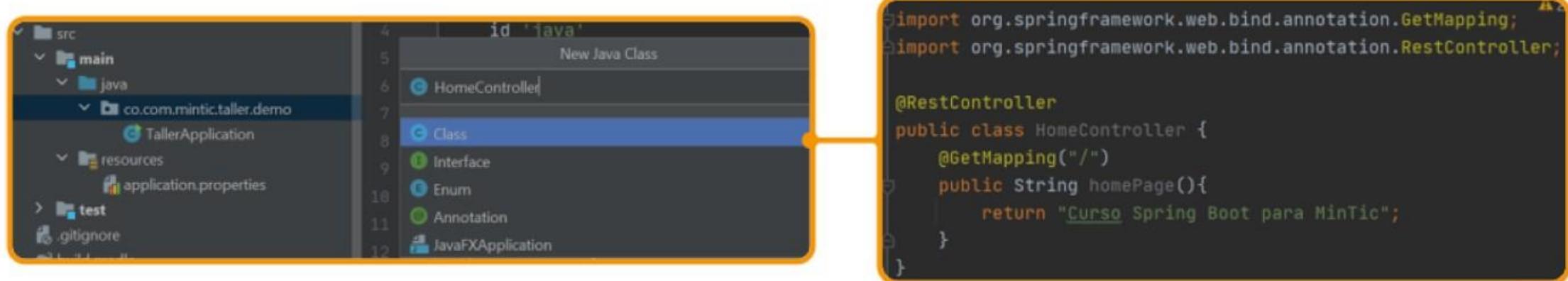
```
dependencies {  
    implementation 'org.springframework.boot:spring-boot-starter-web-services'  
    implementation 'org.springframework.boot:spring-boot-starter-webflux'  
  
    implementation group: 'org.springframework.boot', name: 'spring-boot-starter-parent', v  
    implementation group: 'org.springframework.boot', name: 'spring-boot-starter-web', vers  
  
    developmentOnly 'org.springframework.boot:spring-boot-devtools'  
    testImplementation 'org.springframework.boot:spring-boot-starter-test'  
    testImplementation 'io.projectreactor:reactor-test'  
}
```

# Spring estereotipos y anotaciones



# Microservicios con Spring Boot

- Creamos un controlador utilizando las anotaciones básicas en una nueva clase, denominada HomeController:
- @RestController
- @GetMapping



The screenshot shows a Java file structure in an IDE. The project tree on the left shows a package structure: src/main/java/co/com/mintic/taller.demo. Inside this package, there is a TallerApplication.java file and an application.properties file in resources. A new Java class, HomeController, is being created. The code editor on the right displays the following code:

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

@RestController
public class HomeController {
    @GetMapping("/")
    public String HomePage(){
        return "Curso Spring Boot para MinTic";
    }
}
```

# Microservicios con Spring Boot

- Ejecutamos la aplicación desde la clase TallerApplication.java para enviar el mensaje desde la clase controladora hacia el browser y ver el mensaje. Así podemos ver que se encuentra adecuadamente el servidor web de Spring Boot:

The diagram illustrates the execution flow from code to browser. On the left, a code editor window shows the `TallerApplication.java` file. A red square highlights the `main` method. An orange arrow points from this method to a browser window on the right. The browser window displays the URL `localhost:8080`, with the page content reading "Curso Spring Boot para MinTic".

```
@SpringBootApplication
public class TallerApplication {

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

# Microservicios con Spring Boot

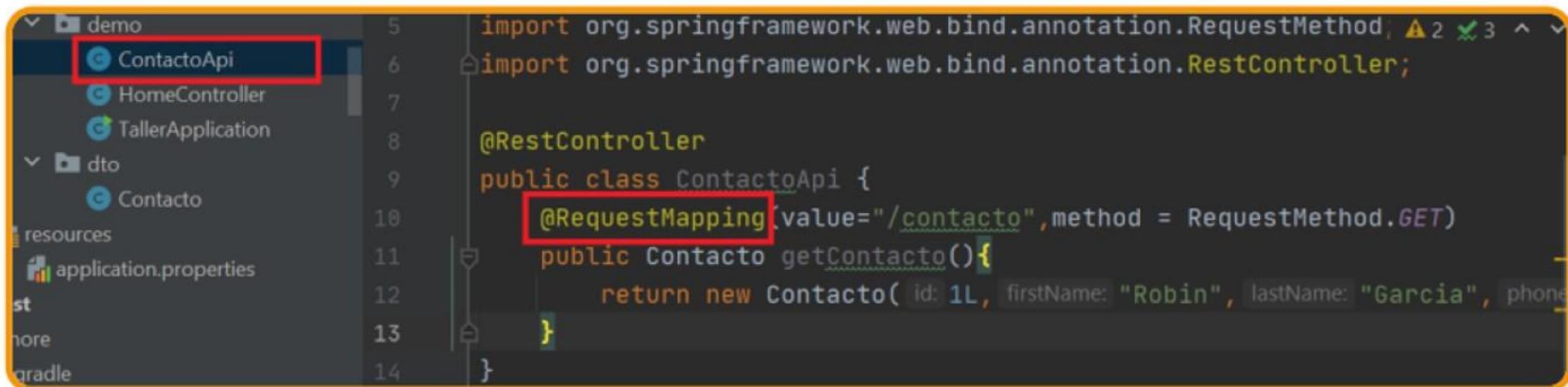
- Ahora creamos su primer servicio REST. Para ello creamos un paquete llamado “dto” y una clase llamada “Contacto”, y ponemos las propiedades del contacto y un constructor de la clase:

The screenshot shows a Java code editor with a file structure on the left. The file structure includes a 'dto' package containing a 'Contacto' class, a 'resources' folder with an 'application.properties' file, a 'test' folder, a '.gitignore' file, and build scripts ('build.gradle', 'gradlew', 'gradlew.bat', 'HELP.md', 'settings.gradle'). External libraries are listed under 'External Libraries'. The code editor displays the 'Contacto' class definition:

```
public class Contacto {  
    private Long id;  
    private String firstName;  
    private String lastName;  
    private String phoneNumber;  
    private String email;  
  
    public Contacto(Long id, String firstName, String lastName, String  
        this.id=id;  
        this.firstName=firstName;  
        this.lastName = lastName;  
        this.phoneNumber = phoneNumber;  
        this.email =email;
```

# Microservicios con Spring Boot

- Creamos una clase denominada ContactoApi para retornar la información del contacto en formato JSON sobre el browser.
- Utilizamos la anotación @RequestMapping, que permiten tener un microservicio básico y funcional con un método que retorne una variable del tipo Contacto con la información configurada por defecto:



The screenshot shows a code editor with the following file structure and code content:

```
demo
  ContactoApi (highlighted with a red box)
  HomeController
  TallerApplication
dto
  Contacto
resources
  application.properties
```

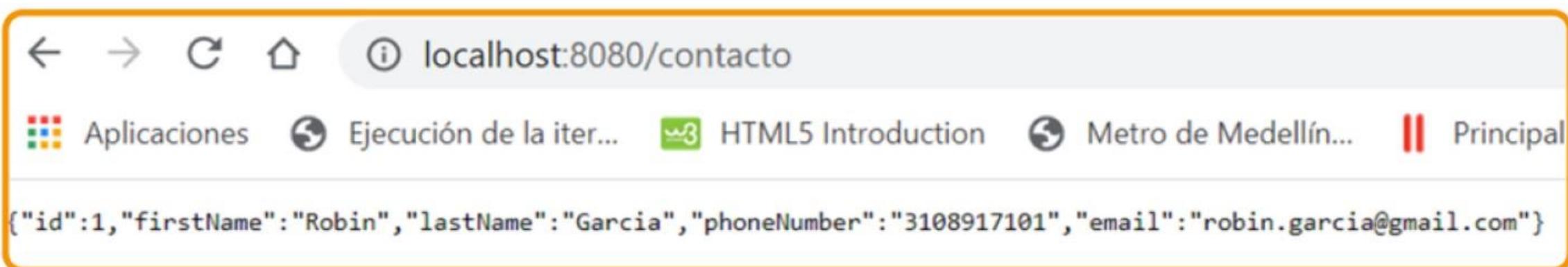
```
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class ContactoApi {
    @RequestMapping(value="/contacto", method = RequestMethod.GET)
    public Contacto getContacto() {
        return new Contacto( id: 1L, firstName: "Robin", lastName: "Garcia", phone
    }
}
```

The file `ContactoApi.java` is highlighted with a red box. The line `@RequestMapping` is also highlighted with a red box.

# Microservicios con Spring Boot

- Ejecutamos su aplicación construida con el arquetipo de Spring Boot y vamos a la URL local para observar la información del contacto que retornamos en el método getContacto de la clase ContactoApi en formato JSON sobre el browser:



# Ejemplo de uso de API REST con Node y Express

- Este tutorial te permitirá seguir el paso a paso para crear una aplicación sencilla usando API REST con Node y Express, y te aclarará el concepto de comunicación que se usa en microservicios.

# Los microservicios

- se consideran como un estilo de arquitectura software para el diseño de aplicaciones complejas. Según Martin Fowler y James Lewis ([link](#)) , los microservicios se basan en un conjunto de pequeños servicios, cada uno de ellos ejecutándose de forma autónoma y comunicándose entre sí mediante mecanismos livianos, generalmente a través de peticiones REST sobre HTTP por medio de sus APIs.

# Agenda

- Historia – contexto
- ¿Qué son microservicios?
- ¿Qué es REST?
- Introducción a los servicios REST
- Microservicios con Spring Boot
- Microservicios con Netflix OSS
- Referencias

# Objetivos

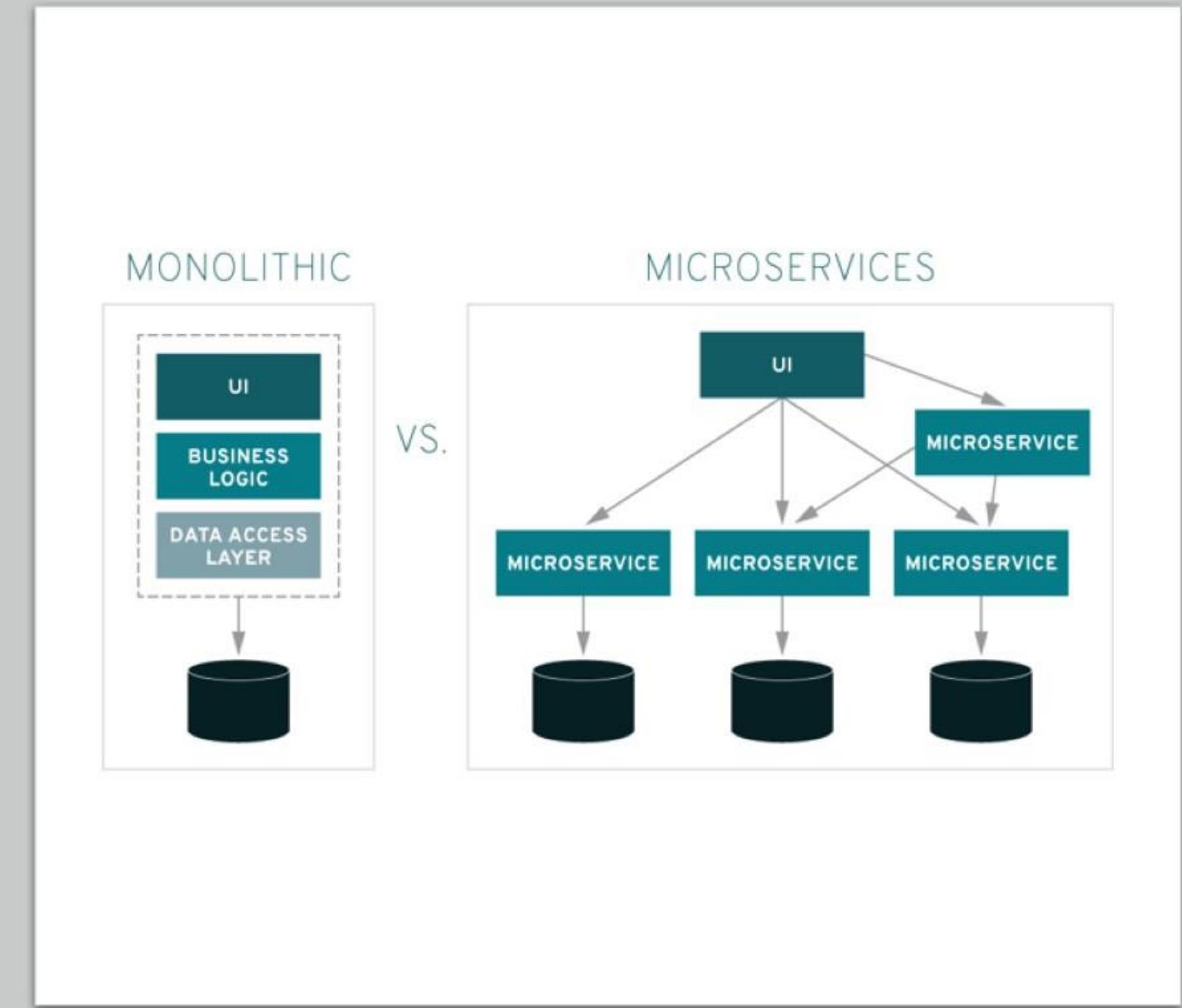
- Construir arquitecturas basadas en microservicios que nos permita atomizar las reglas del negocio.
- Dar más flexibilidad y eficiencia a la hora de implementar y desplegar artefactos funcionales dentro de la arquitectura empresarial corporativa.

# Temas a desarrollar (Microservicios)

- Introducción a los servicios REST
- Microservicios con Spring Boot
- Microservicios con Netflix OSS

# Vocabulario

- Arquitectura
- Sistema distribuido
- Protocolo
- Framework
- API
- Servicio – micro servicio
- Estándar



# El manejo de los **microservicios** en las definiciones de las **nuevas** arquitecturas de software

- Ha venido creciendo vertiginosamente en los últimos tiempos y se ha hecho parte del concepto general de los sistemas distribuidos.
- Los sistemas distribuidos utilizan protocolos como REST
- Estos son más livianos y de fácil acceso, para el envío de información, conectando diferentes sistemas e interfaces con diversas arquitecturas en un sinnúmero de lenguajes de programación.

# Principales ventajas de un microservicio

- **Arquitectura y lenguaje libre**
- **Se ejecutan de manera independiente y autónoma**
- **Infraestructuras IT más flexibles y adaptables.**

# Principales ventajas de un microservicio

- **Multiplataforma:** permite crear servicios y aplicaciones que pueden ser usadas por cualquier dispositivo o cliente que entienda HTTP
- **Competitivo:** Es increíblemente más simple y convencional que otras alternativas de comunicaciones hacia y desde otras interfaces expuestas por cualquier sistema.
- Podríamos considerar **REST** como un **framework** para construir aplicaciones web respetando **HTTP**.

# Desventajas

- Alto consumo de memoria
- Inversión de tiempo inicial
- Complejidad en la gestión
- Perfiles especializados
- Heterogeneidad
- Complejidad en la realización de pruebas
- Costos de implantación o implementación

# Servicios REST

- **REST** (representational state transfer) es un estilo de arquitectura de software para sistemas de hipermédia distribuidos como, por ejemplo, la web.
- **REST** se refiere a una colección de principios para el diseño de arquitecturas en red. Estos principios nos dicen cómo son definidos los recursos y cómo son transmitidos desde y hacia cualquier interfaz en un dominio sobre el protocolo HTTP sin capas adicionales.

# Servicios REST

- REST no es un estándar, pero se basa en los estándares:
  - HTTP
  - URL
  - XML/HTML/GIF/JPEG
  - MIME:text/html

# REST

- **REST** es cualquier interface entre sistemas que usa HTTP.
- **Obtiene datos** o genera operaciones sobre los datos.
- Usa todos los formatos posibles:
  - XML
  - JSON

# Stateful vs stateless

- ¿Qué es un estado?
- ¿Cuánto tiempo se esté registrando y cómo se debe almacenar esa información?
- **Apátrida**
- **Con estado**

# Principios de REST

- **Escalabilidad** de la interacción con los componentes:
  - Crecimiento exponencial sin degradar el rendimiento
  - Variedad de clientes que pueden acceder
  - Uso en los dispositivos móviles
- **Generalidades de interfaces. Gracias al protocolo HTTP:**
  - Cualquier cliente puede interactuar con cualquier servidor HTTP.
  - No posee ninguna configuración adicional para lograr conexión.

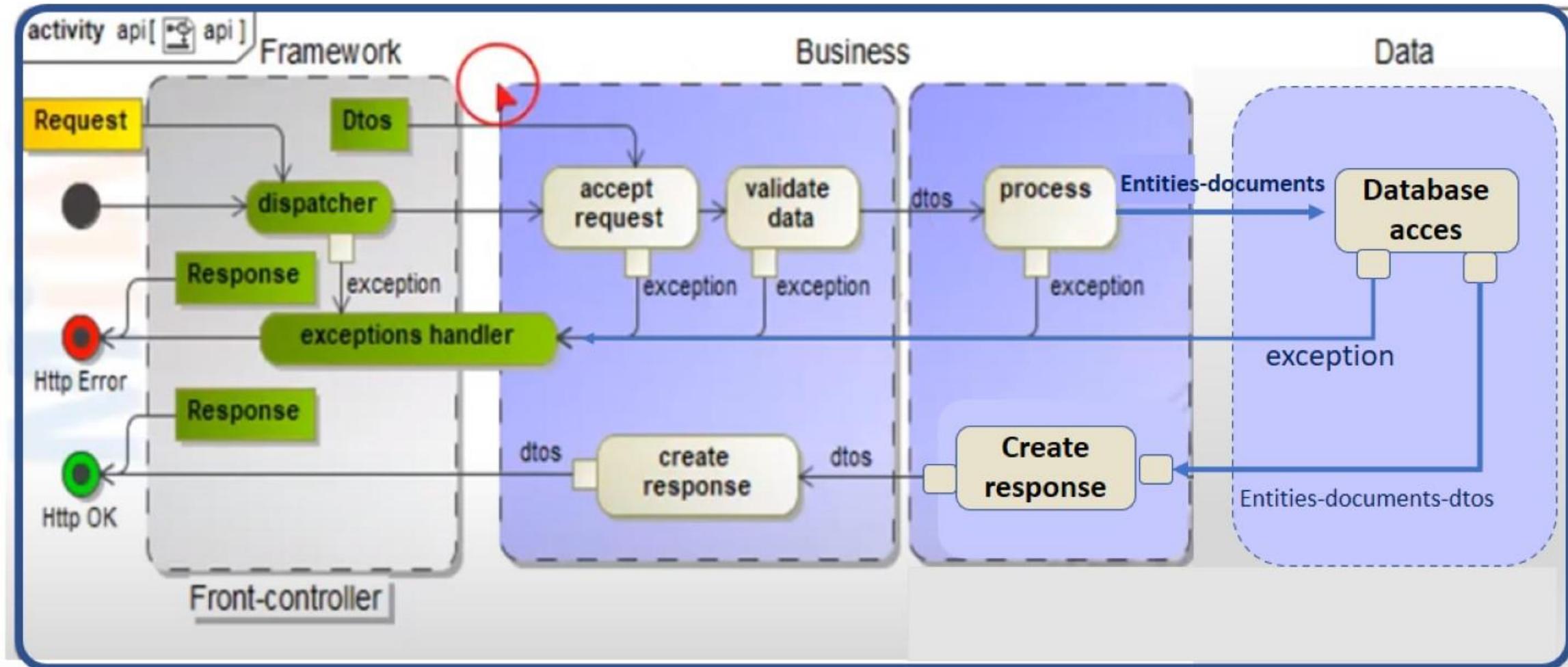
# Principios de REST

- **Compatibilidad** con componentes intermedios:
  - Proxy para web
  - Cache, para mejorar el rendimiento
  - Firewalls
  - Gateway

# Términos en REST

- URI (uniform resource identifier) son recursos identificados como colecciones e interfaces que sirven de puntos de acceso e identifican los recursos que queremos operar.
- Ejemplo: **Empleado** (una URI por empleado)
  - TodosLosEmpleados(Listado de los empleados)
  - Cada página, archivo, es un recurso al que accedemos o podemos modificar/eliminar

## Flujo de datos API REST



# Términos en REST

- **URL** (uniform resource locator) es un tipo de URI que, además de identificar de forma única el recurso, nos permite localizarlo para acceder a él.
- Estructura URL:
  - {protocolo}://{dominio o hostname}[:puerto (opcional)]/{ruta del recurso}?{consulta de filtrado}
- Ejemplo URL:
  - <https://cursomintic.com:101/2021/conceptos-sobre-apis-rest/>

# Términos en REST

- **Formatos.** Son una representación como un documento HTML, XML, una imagen, etc.
- Ejemplo de un formato de representación en XML para **Empleado**:
  - <employee xmlns='HTTP://example.org/my-example-ns/'>
  - <name>Full name goes here.</name>
  - <title>Persons title goes here.</title>
  - <phone-number>Phone number goes here.</phone-number>
  - </employee>

# Acciones REST

- Los métodos o acciones están diseñadas para operar con datos atómicos dentro de una transacción similar a las operaciones CRUD.
- REST se diseña alrededor de transferencias atómicas de un estado más complejo.
- Es la transferencia de un documento estructurado de una aplicación a otra

| Acción | HTTP   | SQL    | Copy&Paste    | Unix Shell |
|--------|--------|--------|---------------|------------|
| Create | PUT    | Insert | Pegar         | >          |
| Read   | GET    | Select | Copiar        | <          |
| Update | POST   | Update | Pegar después | >>         |
| Delete | DELETE | Delete | Cortar        | Del/rm     |

# Métodos soportados en REST

- Cuando hablamos de los **métodos** o acciones, es la forma en que se van a referenciar los recursos (URI) para ser accedidos de diferentes maneras.
- Ejemplo:

| HTTP   | CRUD     | Descripción                              |
|--------|----------|--|
| POST   | CREATE   | Crear un nuevo recurso                   |
| GET    | RETRIEVE | Obtener la representación de una recurso |
| PUT    | UPDATE   | Actualizar un recurso                    |
| DELETE | DELETE   | Eliminar un recurso                      |

# Código de estado devueltos en REST

- Son los códigos de estado del recurso que podrían ser devueltos con cualquier método o acción enviado desde el cliente.
- Los códigos son respuestas estandarizadas para informar al cliente sobre el resultado de una solicitud o acción.

2 - 4 - 5

# Código de estado devueltos en REST

## Códigos de estado para la finalización satisfactoria de la solicitud REST

| Código de estado  | Descripción   |
|-------------------|---|
| 200 Correcto      | La solicitud se ha completado satisfactoriamente.                                     |
| 201 Creado        | La solicitud se ha completado satisfactoriamente; se ha creado un recurso nuevo.      |
| 204 Sin contenido | La solicitud se ha completado satisfactoriamente pero el contenido no está disponible |

# Código de estado devueltos en REST

## Códigos de estado para situaciones de error esperado

| Código de estado                 | Descripción   |
|----------------------------------|---|
| 400 Solicitud incorrecta         | La solicitud REST contiene parámetros que no son válidos o que están ausentes.  |
| 401 No autorizado                | El interlocutor no está autorizado a realizar la solicitud.   |
| 403 Prohibido                    | El interlocutor no está autorizado a completar la solicitud.  |
| 404 No encontrado                | El recurso solicitado no existe.  |
| 406 No aceptable                 | Se ha solicitado un tipo de contenido o de codificación de contenido no soportado.  |
| 409 Conflicto                    | Existe un conflicto con el estado actual del recurso. La acción solicitada no se puede llevar a cabo en el recurso en su estado actual. |
| 415 Tipo de soporte no soportado | La solicitud contiene un tipo de contenido o una codificación de contenido desconocidos.  |

# Código de estado devueltos en REST

## Códigos de estado para errores inesperados

| Código de estado                 | Descripción   |
|----------------------------------|---|
| 500 Error interno del servidor   | Se ha producido un problema; se proporciona más información en el rastreo de la pila.                         |
| 501 No implementado              | La solicitud no tiene soporte en la API de REST de IBM® Business Process Manager                              |
| 503 Servicio no disponible       | Las solicitudes federadas no se han podido entregar a todos los destinos de la federación.                    |
| 504 Tiempo de espera de pasarela | Una respuesta federada no está completa ya que faltan respuestas de algunos de los destinos de la federación. |

# Referencias

- <https://lms.misiontic2022udea.com/course/view.php?id=531>
- [www.scalapp.co](http://www.scalapp.co)
- <https://www.redhat.com/en/topics/api/what-is-a-rest-api>
- <https://www.ibm.com/docs/es/bpm/8.5.6?topic=apis-status-codes>
- <https://datatracker.ietf.org/doc/html/rfc6838>
- <https://www.iana.org/assignments/media-types/media-types.xhtml>
- [https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics\\_of\\_HTTP/MIME\\_types](https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/MIME_types)
- Máster en Ingeniería Web. Universidad Politécnica de Madrid.
- Arquitectura y Patrones para Aplicaciones Web.
- Capítulo 14: Arquitectura de capa de negocio. API Rest con Spring

# Referencias:

- <https://plugins.jetbrains.com/plugin/10229-spring-assistant/versions/nightly>

**Ciclo4 misión tic 2022** Introducción a microservicios con Spring Boot

- [https://lms.misiontic2022udea.com/pluginfile.php/81115/mod\\_resource/content/9/2021\\_000127\\_DW\\_Semana5\\_MinTic\\_Microservicios%20Spring%20Boot\\_VFinal.pdf](https://lms.misiontic2022udea.com/pluginfile.php/81115/mod_resource/content/9/2021_000127_DW_Semana5_MinTic_Microservicios%20Spring%20Boot_VFinal.pdf)
- [www.scalapp.co](http://www.scalapp.co)
- <https://escuelafullstack.com/slides/curso-de-microservicios-con-spring-boot-13>
- <https://www.oracle.com/java/technologies/downloads/#jdk17-windows>
- <https://www.jetbrains.com/idea/download/other.html>

# Referencias:

- 5 Trucos del IntelliJ IDEA

<https://www.youtube.com/watch?v=DOTL82UiQjo>

- IntelliJ IDEA | Full Course | 2020

<https://www.youtube.com/watch?v=yefmcX57Eyg>

[https://www.jetbrains.com/idea/features/editions\\_comparison\\_matrix.html](https://www.jetbrains.com/idea/features/editions_comparison_matrix.html)

<https://docs.oracle.com/en/java/javase/17/index.html>

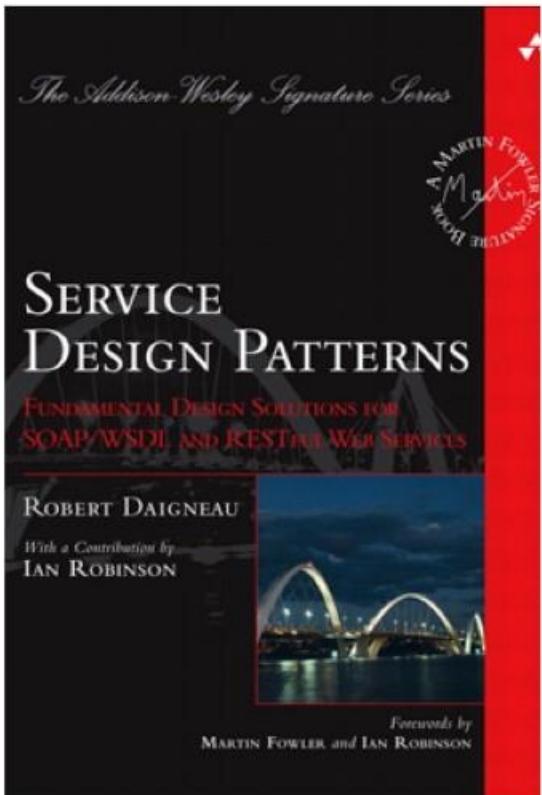
<https://gradle.org/releases/>

<https://gradle.org/guides/>

[https://gradle.com/training/?\\_ga=2.146957936.690131247.1638469529-173266704.1638469529](https://gradle.com/training/?_ga=2.146957936.690131247.1638469529-173266704.1638469529)

<https://gradle.org/release-candidate/>

# Referencias



- <https://spring.io/projects/spring-cloud-netflix>
- <https://www.youtube.com/watch?v=uWWKQhpGWPw&t=167s>
- Service Design Patterns
- Fundamental Design Solutions for SOAP/WSDL and RESTful Web Services
- **by Robert Daigneau, with Ian Robinson**
- <https://www.youtube.com/watch?v=wgdBVIX9ifA> 2014

# Referencias

- **Building Microservices: Designing Fine-Grained Systems 1st Edición**
- **Microservices Patterns: With examples in Java**
- **Microservice Patterns and Best Practices: Explore patterns like CQRS and event sourcing to create scalable, maintainable, and testable microservices**
- <https://www.youtube.com/watch?v=uWWKQhpGWPw&t=167s>

