

# Sesión 05

## Configuración y Resiliencia

Instructor:

**ERICK ARÓSTEGUI**

earostegui@galaxy.edu.pe



# 8

## NET

### MICROSERVICES ARCHITECTURE

# ÍNDICE

**01**

Servidor de configuración

---

**02**

Registro y discovery de microservicios.

---

**03**

Steeltoe

---

**04**

Resiliencia y alta disponibilidad de microservicios.

---

**05**

Principales patrones de resiliencia (Circuit Breaker, Retry Design y Bulkheads Design).

---

01



# Servidor de configuración

# Configuración de servicios mediante configuración distribuida

**¿Qué tiene de diferente administrar la configuración en una aplicación nativa de la nube?**

## Configuración: no distribuido vs distribuido



**De uno o un puñado  
de archivos de  
configuración**

## Configuración: no distribuido vs distribuido



**De uno o un puñado  
de archivos de  
configuración**



**A...**



**Muchos, muchos  
archivos de  
configuración**

## Configuración: no distribuido vs distribuido



Herramientas de gestión de configuración al rescate,  
¿verdad?

e.g. Chef/Puppet/Ansible



## Configuración: no distribuido vs distribuido

**Funcionará ... pero no es ideal en la nube**



## Configuración: no distribuido vs distribuido



**Orientado al  
despliegue**

## Configuración: no distribuido vs distribuido



**Orientado al  
despliegue**



**Basado en PUSH  
generalmente no es lo  
suficientemente  
dinámico**

## Configuración: no distribuido vs distribuido



**Orientado al  
despliegue**



**Basado en PUSH  
generalmente no es lo  
suficientemente  
dinámico**



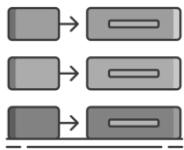
**Basado en PULL  
agrega latencia con  
sondeo temporal**

P: Si las herramientas de administración de configuración no resuelven nuestro problema, ¿qué lo hace?

P: Si las herramientas de administración de configuración no resuelven nuestro problema, ¿qué lo hace?

**R: Servidor de configuración**

## Servidor de Configuración de Aplicaciones



Almacén de clave / valor  
centralizado, dinámico y  
dedicado (puede distribuirse)



Fuente con acceso de  
autorización



Revisión de cuentas



Versionado

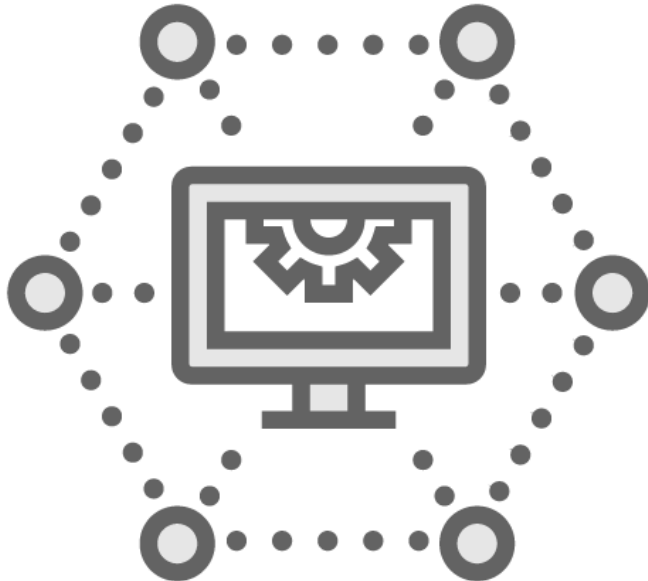


Soporte de criptografía

# Administrar la configuración de la aplicación con **Spring Cloud**



## Administrar la configuración con



- Spring Cloud Consul
- Spring Cloud Zookeeper
- **Spring Cloud Config**

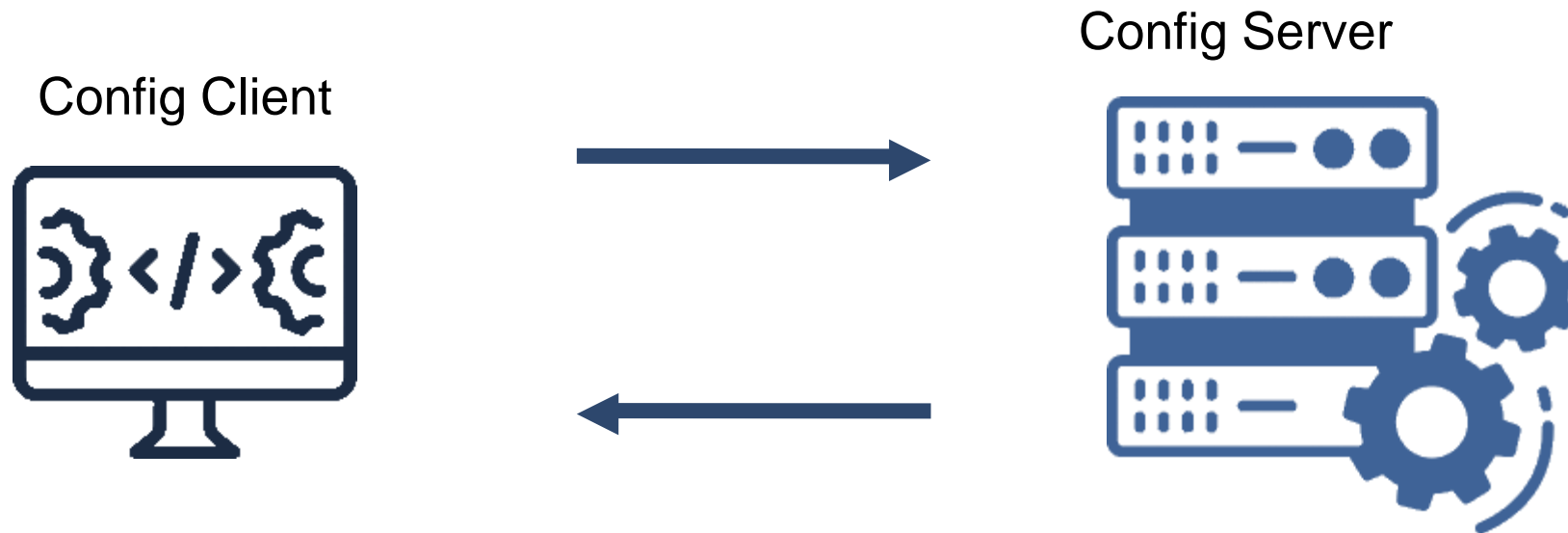
# Spring Cloud Config

Spring Cloud Config proporciona soporte del servidor y del lado del cliente para la configuración externa en un sistema distribuido.

Documentación de referencia :

<https://cloud.spring.io/spring-cloud-config/reference/html/>

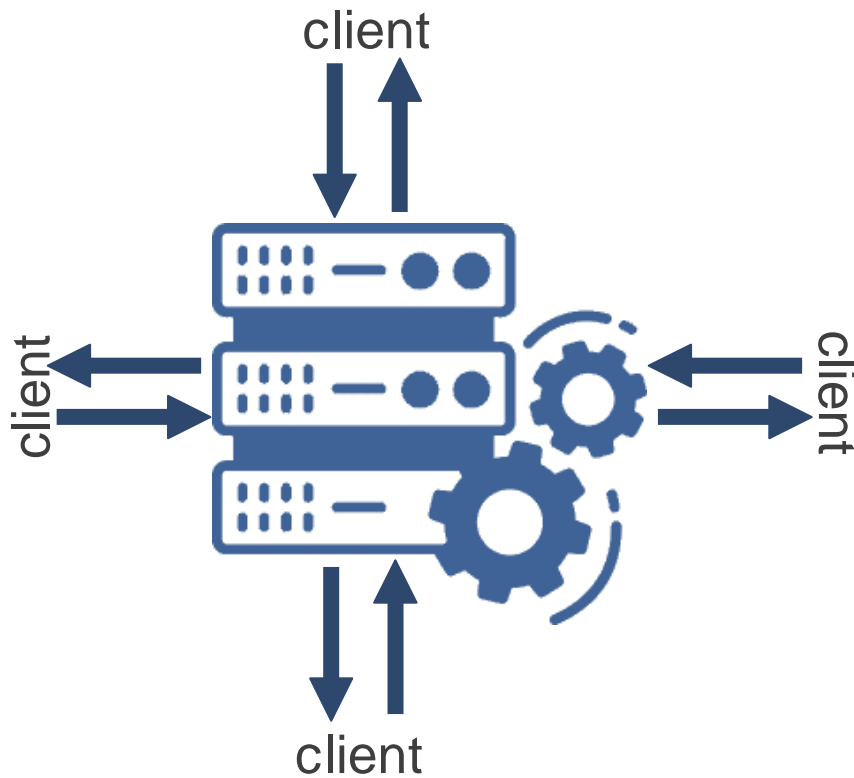
## Integración con aplicaciones



- **Embebido en la aplicación**
- **Spring Environment abstraction**
  - e.g. @Inject Environment

- **Standalone (puede embeberse)**
- **Spring PropertySource abstraction**
  - e.g. classpath:file.properties

## Spring Cloud Config Server



Acceso HTTP REST

Formatos de salida

- **JSON (default)**
- Properties
- YAML

Almacenamiento Backend

- **Git (default)**
- SVN
- Filesystem

**Enviroments** de configuración

## Spring Cloud Config Server

**¡No olvides asegurar tu servidor de configuración!**

Fácil de configurar **Spring Security**



## REST Endpoint Parameters

**{application}**

**maps to**  
spring.application.name  
**on client**

**{profile}**

**maps to**  
spring.profiles.active  
**on client**

**{label}**

función del lado del  
servidor para referirse al  
conjunto de archivos de  
configuración por nombre

## REST Endpoint



Endpoint

---

GET /{application}/{profile}[/{label}]



Ejemplo

---

- /myapp/dev/master
- /myapp/prod/v2
- /myapp/default

## REST Endpoint



Endpoint

---

`/ {application}-{profile}.(yml | properties)`



Ejemplo

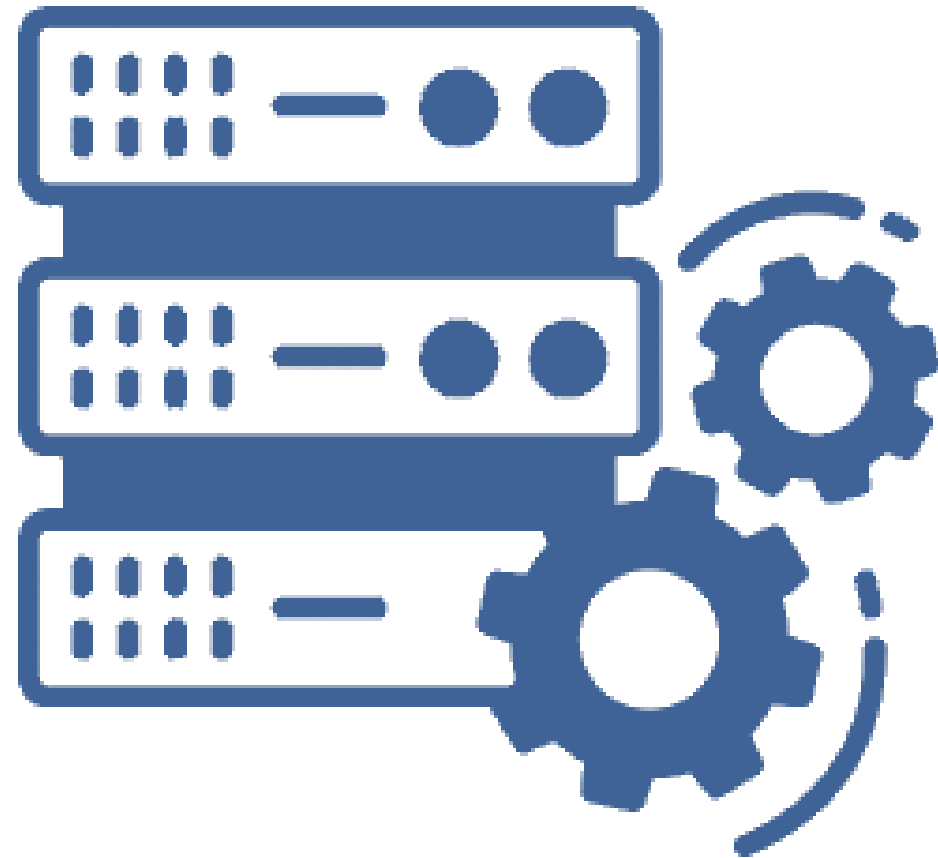
---

- `/myapp-dev.yml`
- `/myapp-prod.properties`
- `/myapp-default.properties`

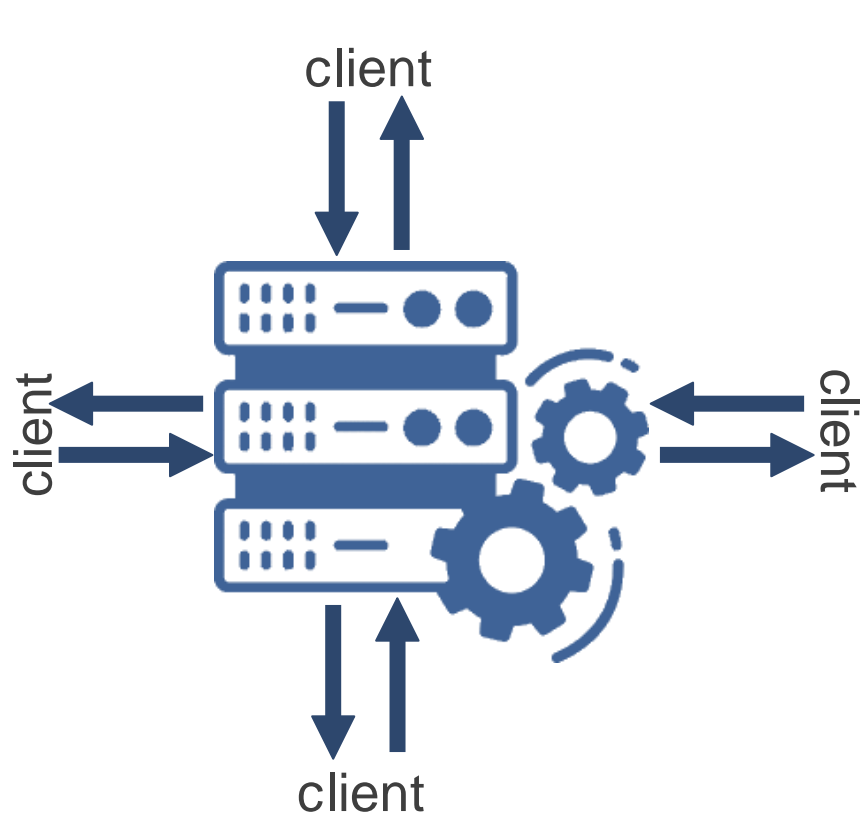


# Creando e iniziando un config server

DEMO

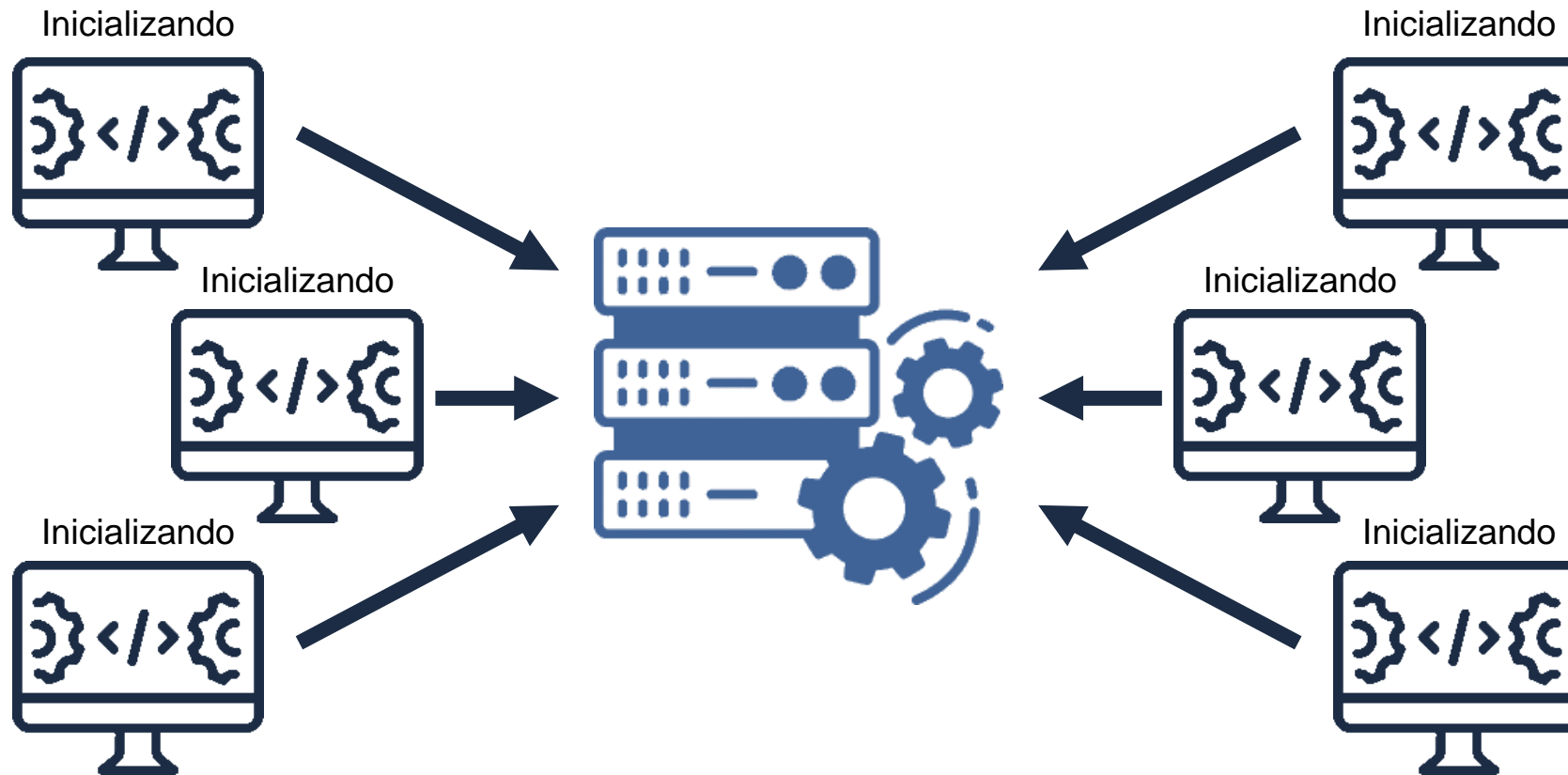


## Spring Cloud Config Client

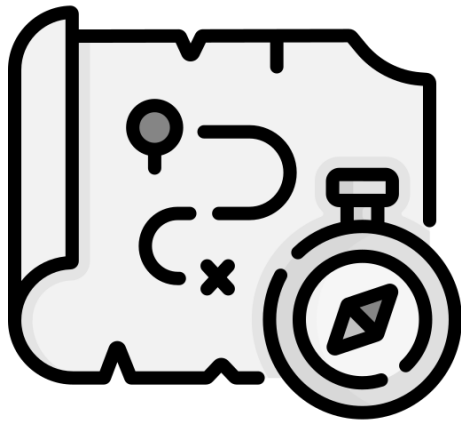


Inicialización y carga de configuración de aplicaciones

## Recuperando configuración: inicio de la aplicación



## Bootstrapping



### Config first

Especifique la ubicación del servidor de configuración



### Discovery first

Descubre la ubicación del servidor de configuración

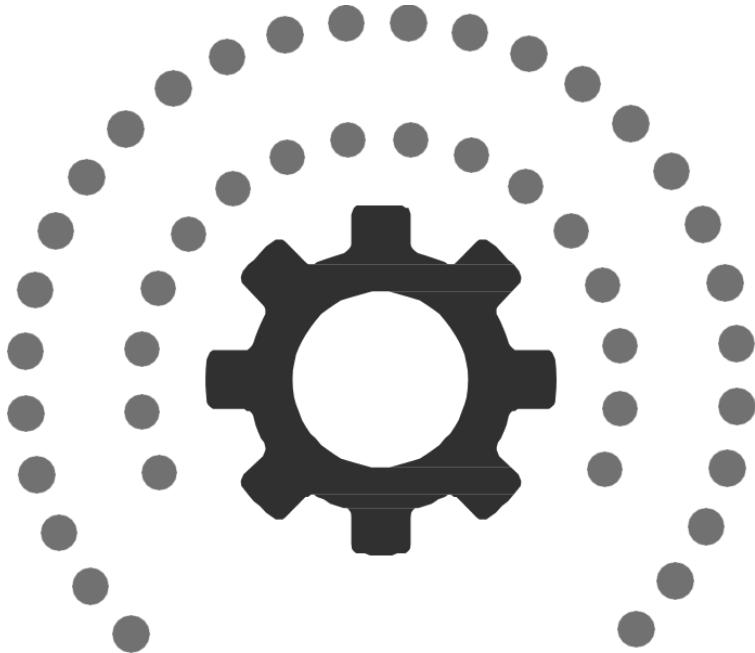
# Inicialización de un servicio que usa config client

DEMO



# Actualización de la configuración en tiempo de ejecución

## Actualización de Config Client

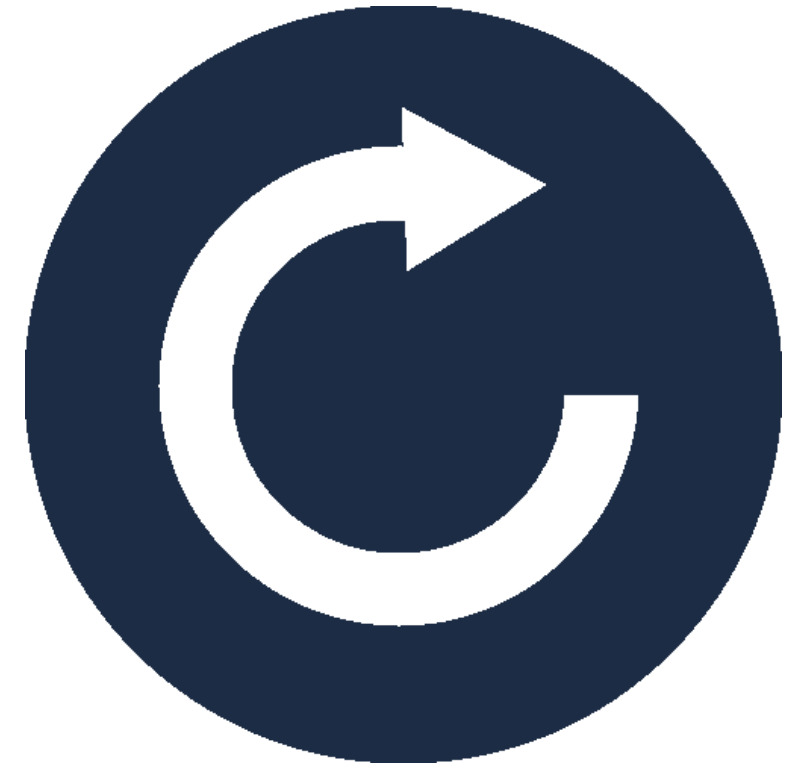


**Refresh**

@ConfigurationProperties

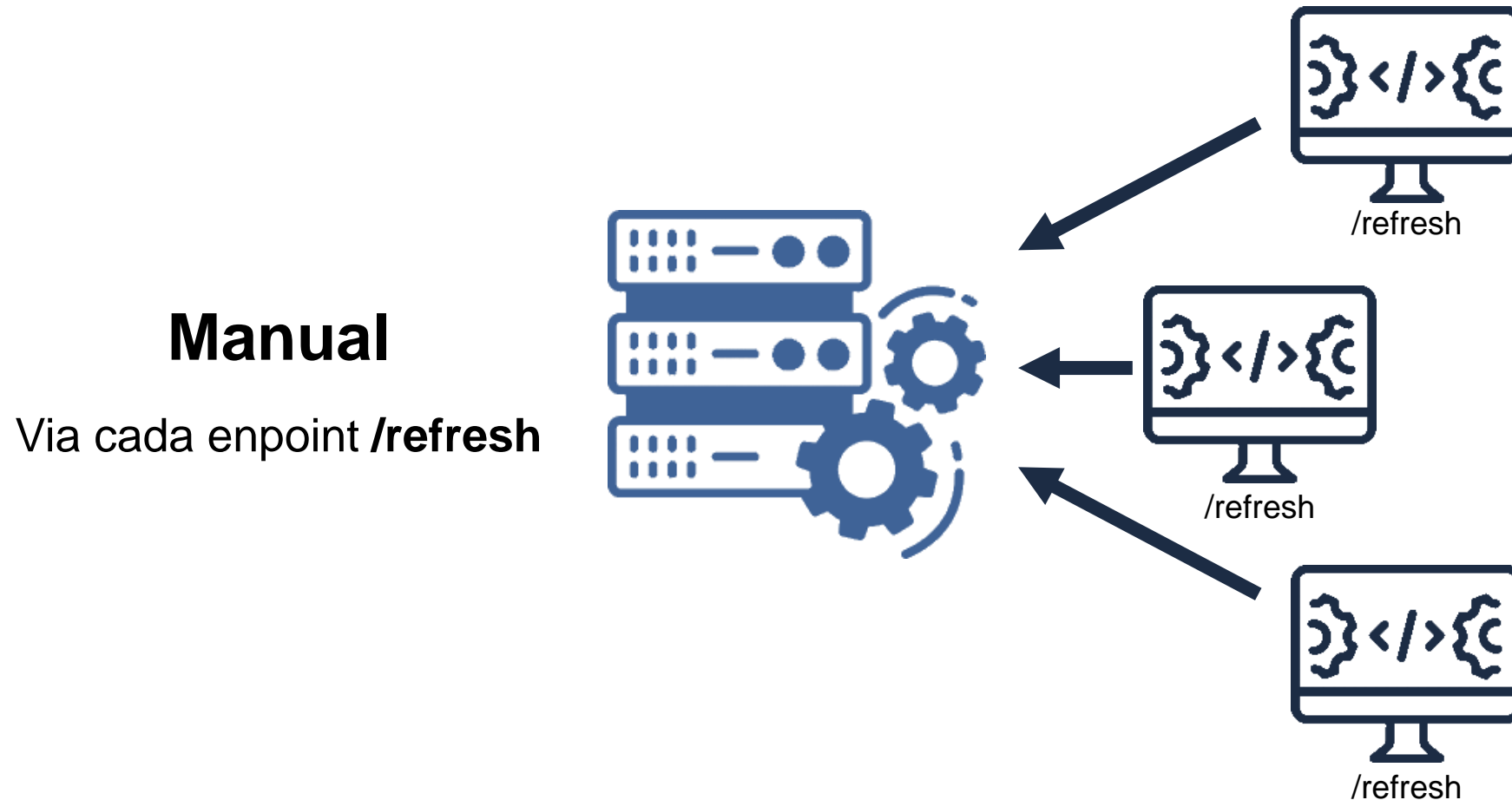
Notifique las aplicaciones para actualizar la configuración

**/refresh**  
with  
**spring-boot-actuator**





## Obteniendo configuración: actualización explícita



## Notifique las aplicaciones para actualizar la configuración



**/refresh**

with  
spring-boot-actuator

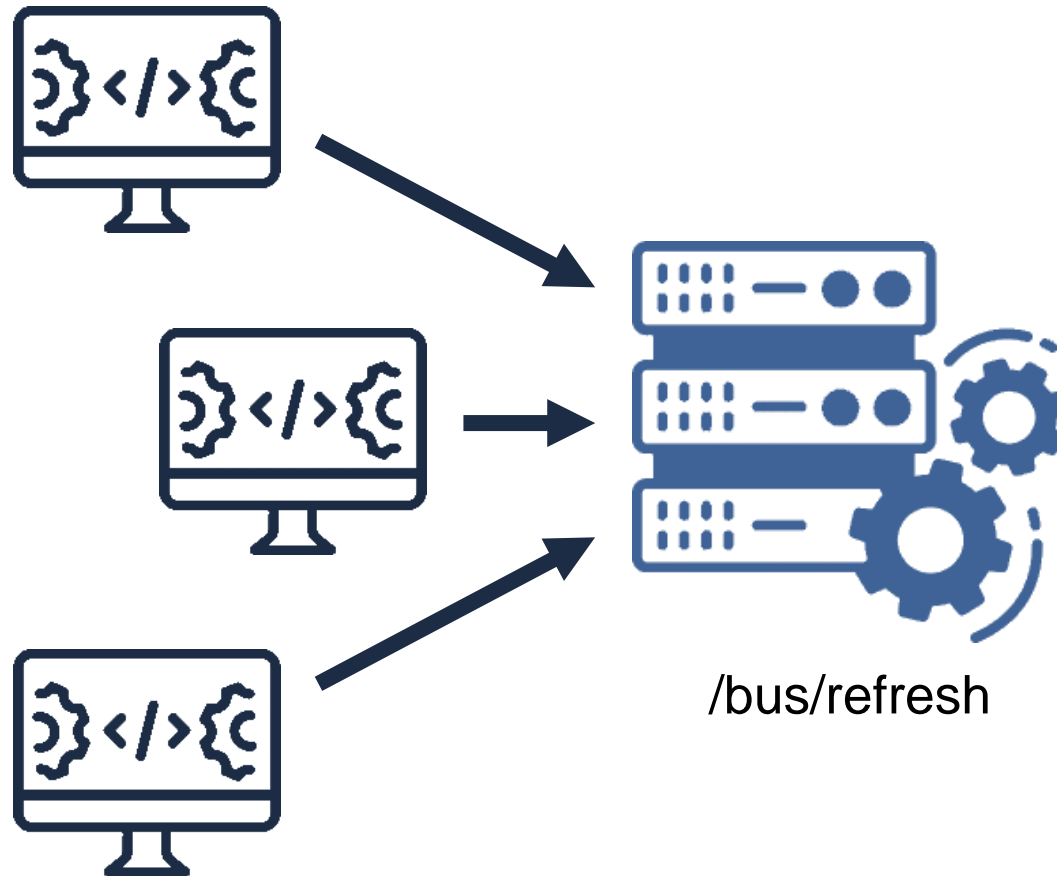


**/bus/refresh**

with  
spring-cloud-bus

# → Servidor de configuración

## Recuperación de la configuración: actualización dinámica por push

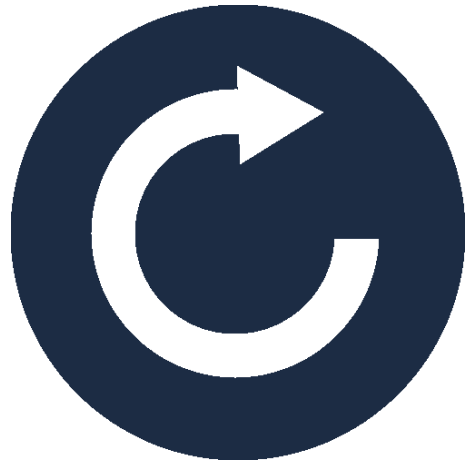


**Automático**

Via broadcasting Spring

Cloud Bus

## Notifique las aplicaciones para actualizar la configuración



**/refresh**

with  
spring-boot-actuator



**/bus/refresh**

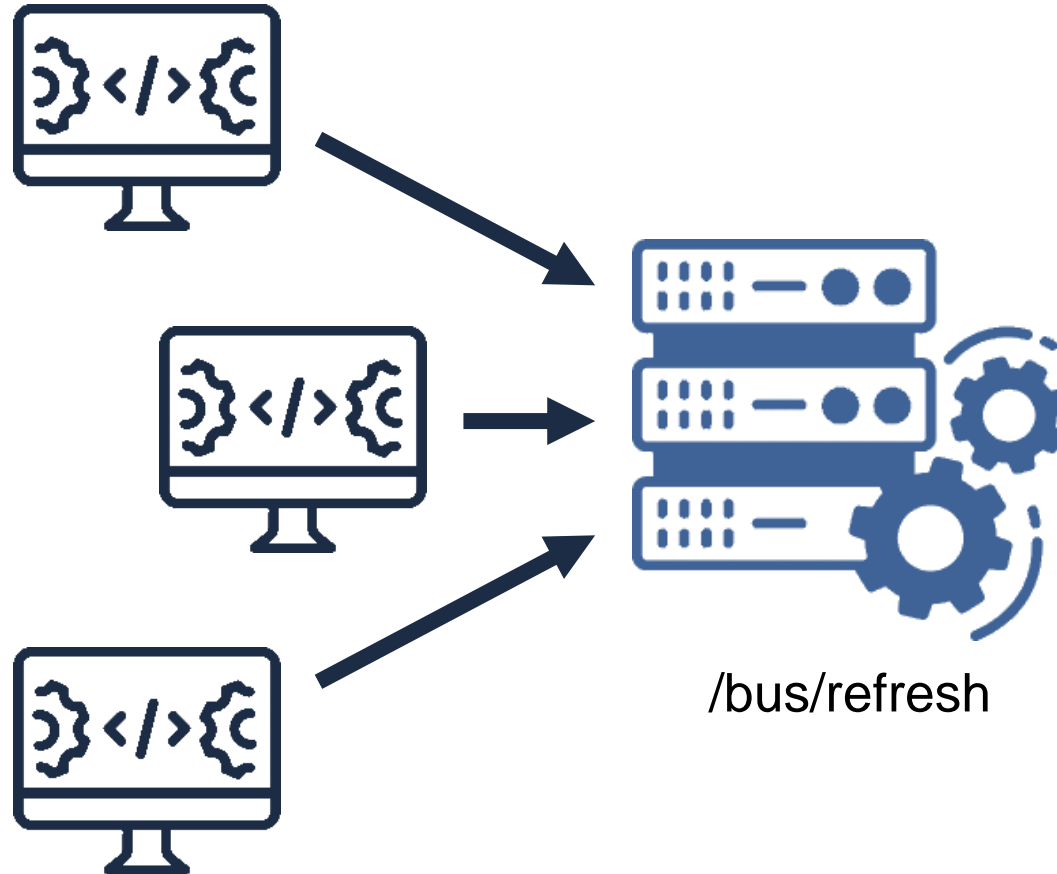
with  
spring-cloud-bus



**VCS + /monitor**

with  
spring-cloud-config- monitor &  
spring-cloud-bus

## Recuperando configuración: Actualización inteligente

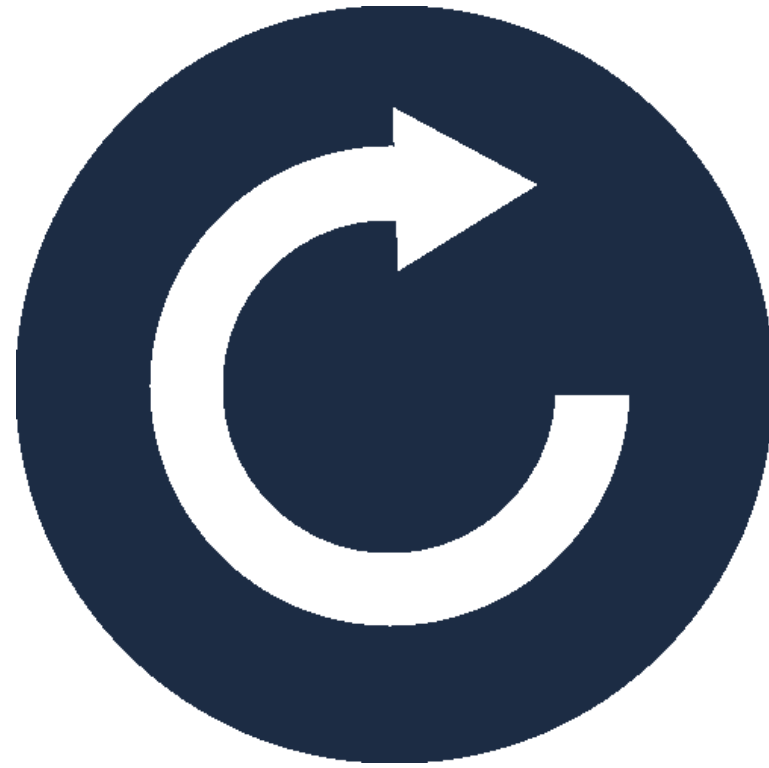


**Automático e  
Inteligente**

Via post commit hooks  
Spring Cloud Config Monitor  
& Spring Cloud Bus  
broadcasting

# Actualización de configuración al vuelo

DEMO



## ¿Qué características son compatibles?



Configuración  
encriptada en  
reposo y / o en  
vuelo



Endpoint de  
encriptamiento  
de  
configuración



Endpoint de  
descripta  
iento de  
configuración



Cifrado y descifrado  
con claves  
simétricas o  
asimétricas.

# 02

## Registro y discovery de microservicios.



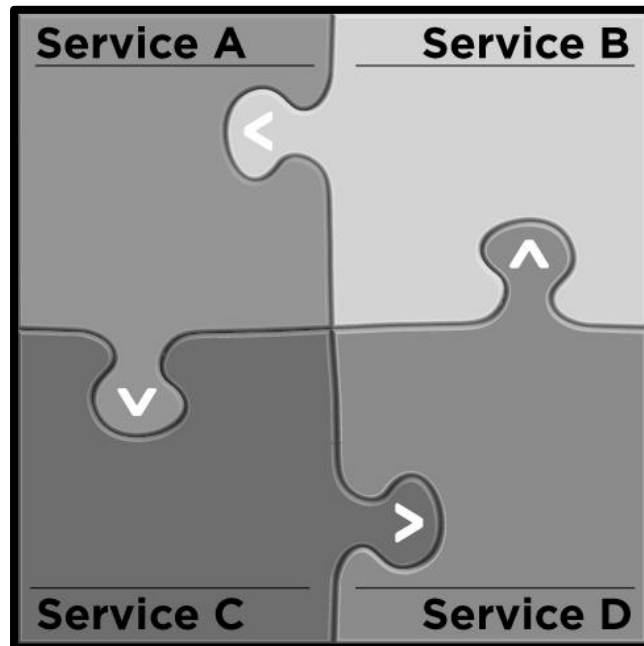


# Encontrar servicios utilizando el **Services Discovery**

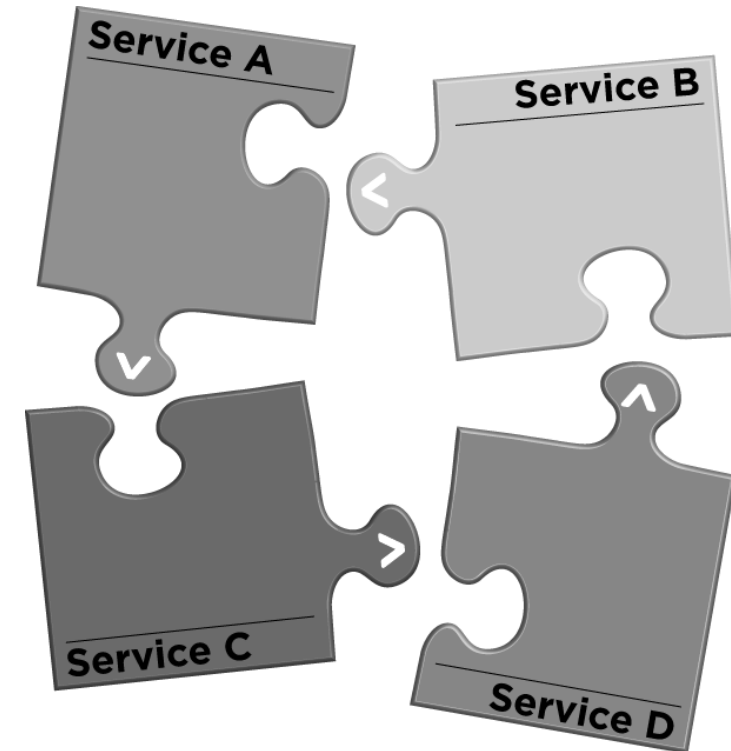
¿Qué es **Service Discovery** y por qué lo necesitamos?

# → Registro y discovery de microservicios

## Cambios en la forma en que desarrollamos software



Desde una aplicación  
monolítica



Para servicios desplegables  
individualmente

**El problema: ¿cómo un servicio ubica a otro?**



Servicio de  
aplicación A

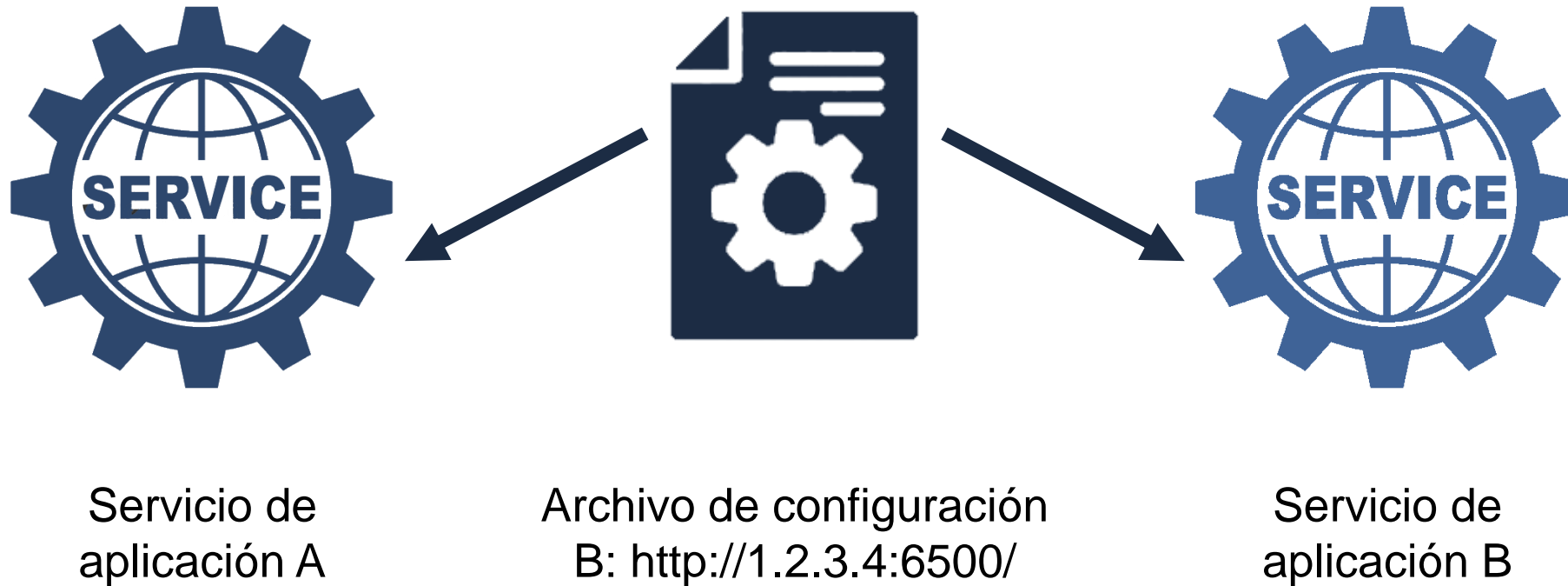


¿Localización?

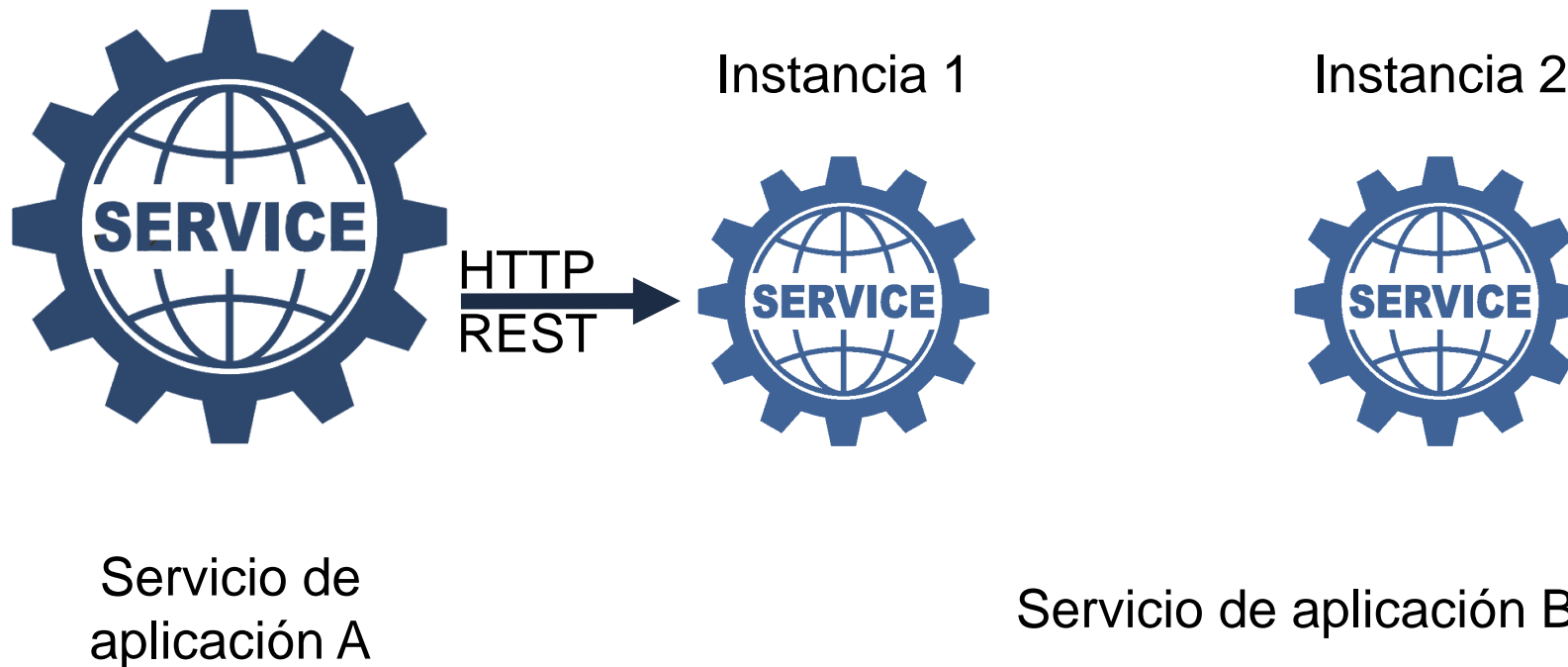


Servicio de  
aplicación B

## El enfoque simple: a través de la configuración

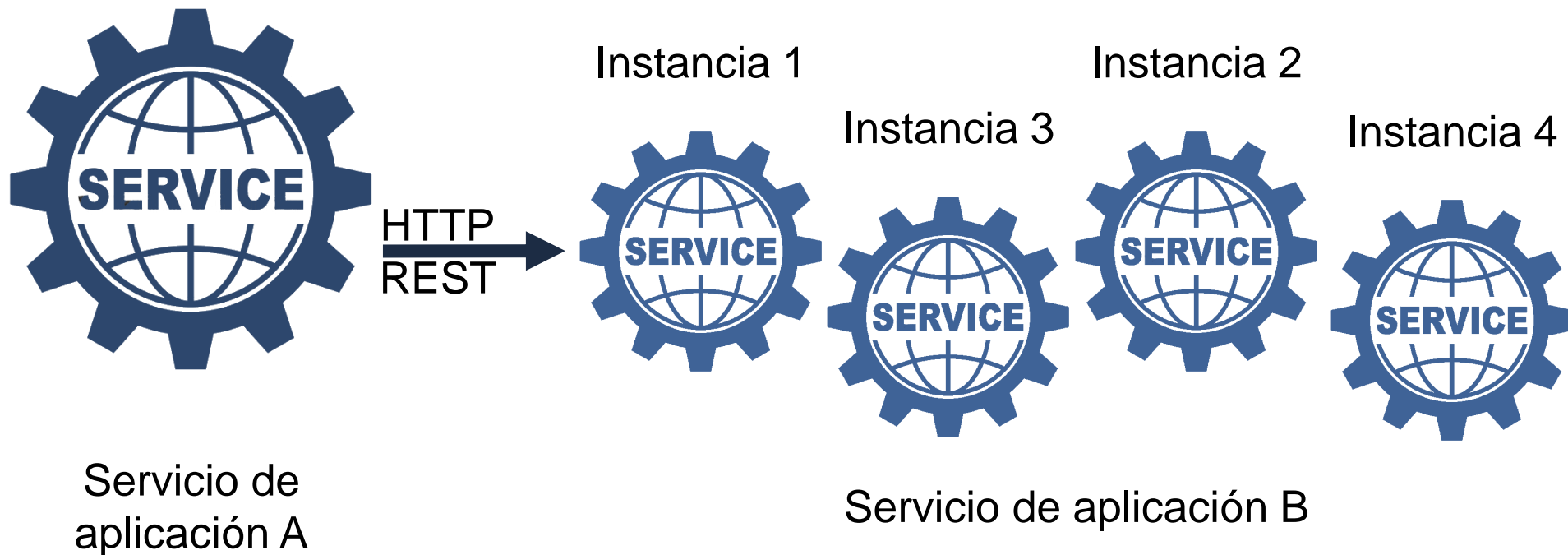


## Multiple Instancias

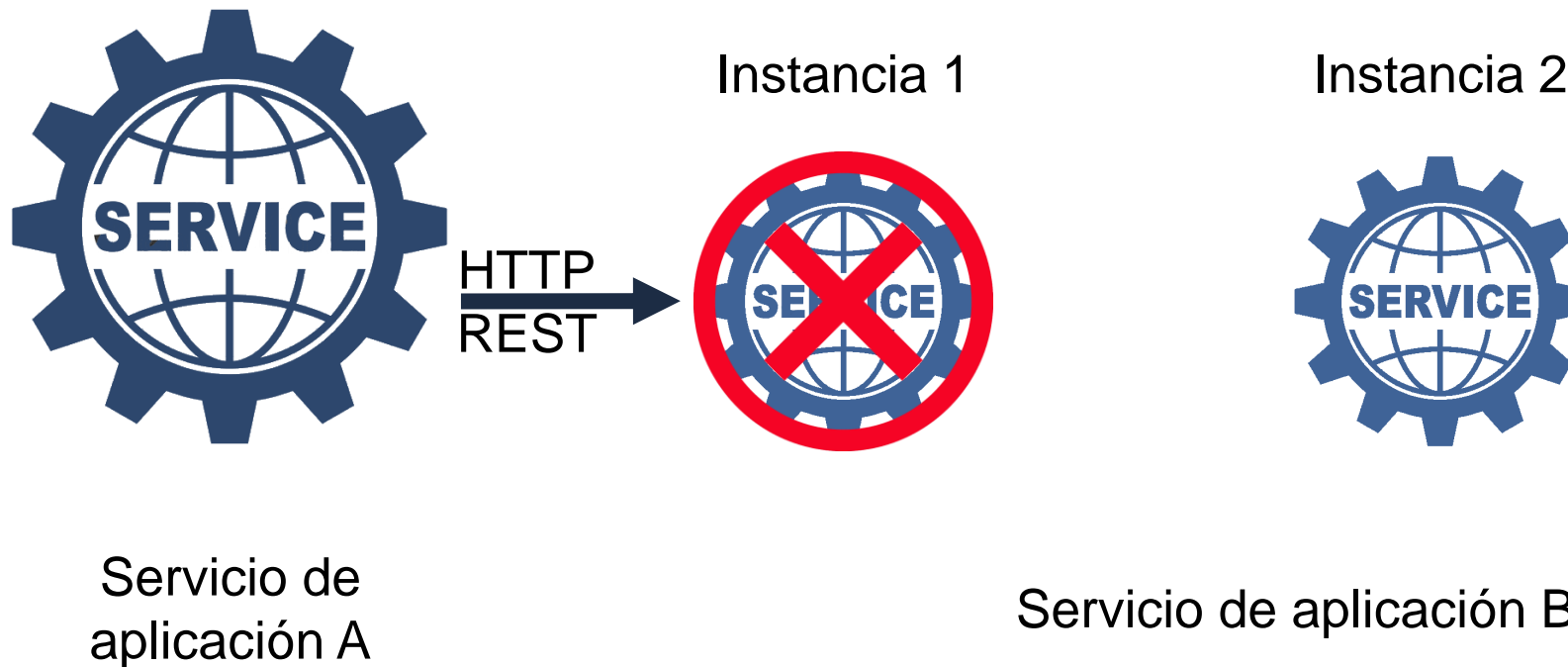


# → Registro y discovery de microservicios

**Las instancias van y vienen en respuesta a la demanda**



## Las instancias fallan





## El enfoque simple: a través de la configuración

¡El enfoque simple es  
**demasiado estático**  
(congelado en el tiempo)  
para la nube!



## Service Discovery : a través de la configuración



El descubrimiento de servicios proporciona

- Una forma para que un servicio se registre
- Una forma para que un servicio se desregistre
- Una manera para que un cliente encuentre otros servicios
- Una forma de verificar el estado de un servicio y eliminar instancias caídas

# Discovering Services con Spring Cloud

## Service Discovery : a través de la configuración

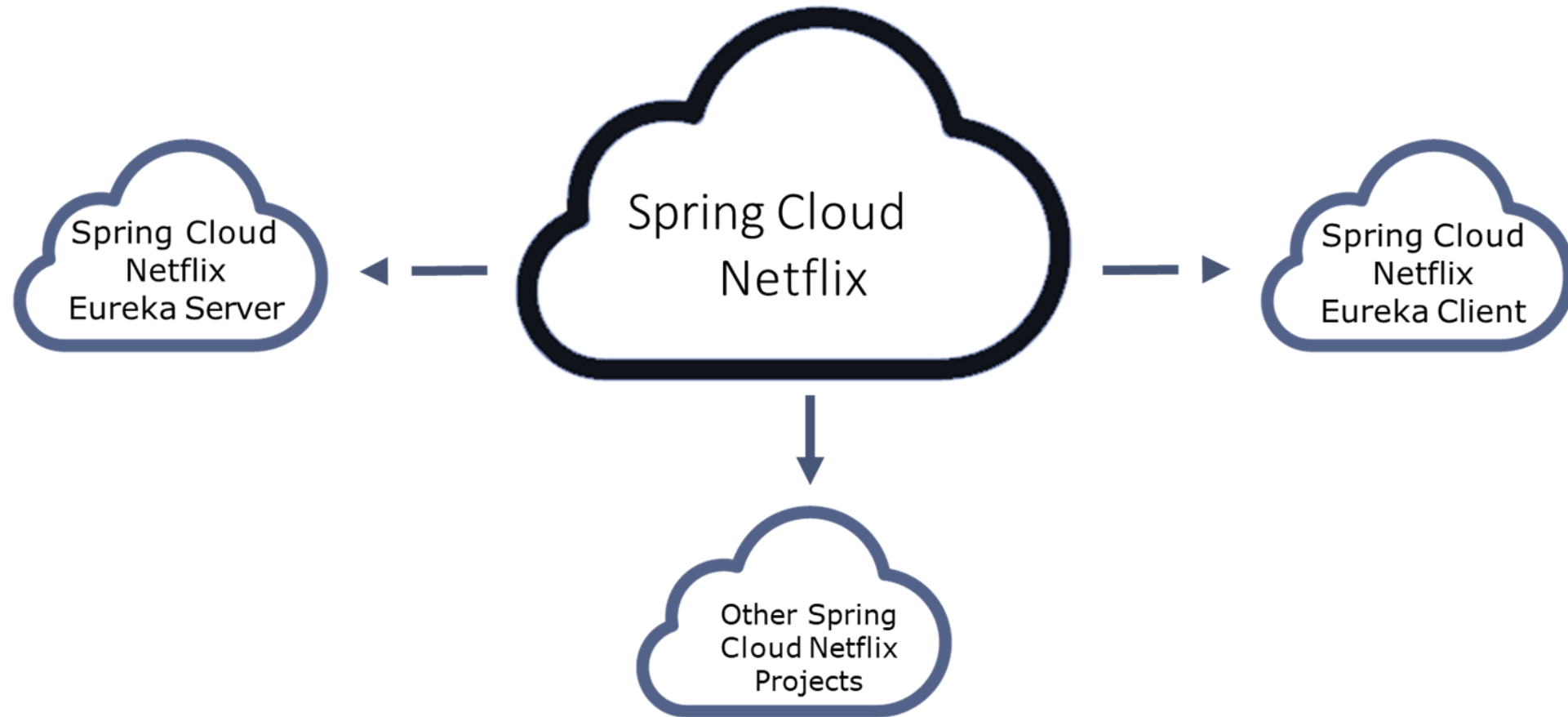


Discover services con:

- Spring Cloud Consul
- Spring Cloud Zookeeper
- Spring Cloud Netflix

Netflix OSS + Spring + Spring Boot  
=  
**Spring Cloud Netflix**

# → Registro y discovery de microservicios



## Componentes clave en Service Discovery

Discovery Server



Service

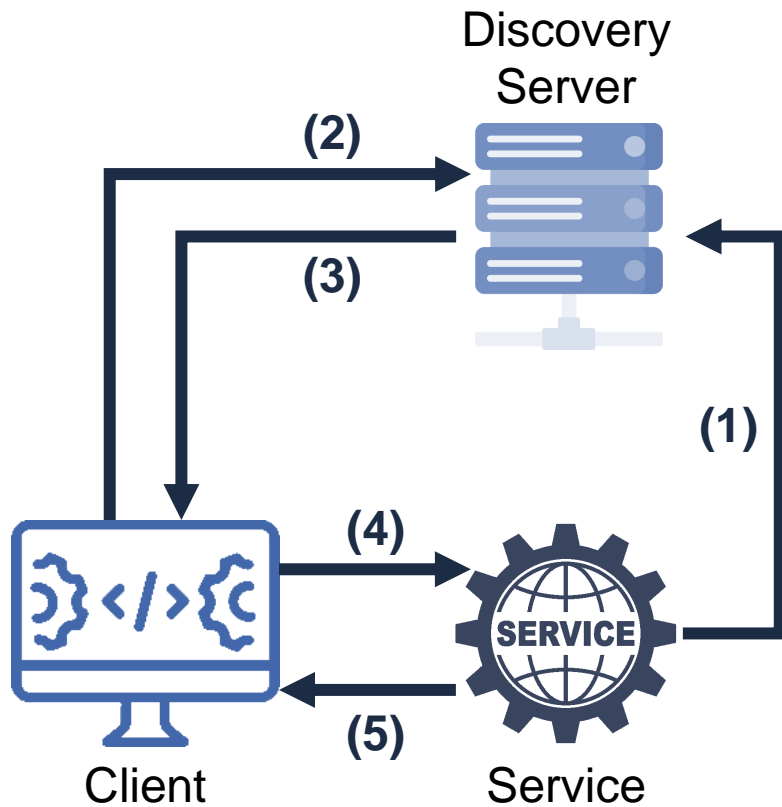


Client



# → Registro y discovery de microservicios

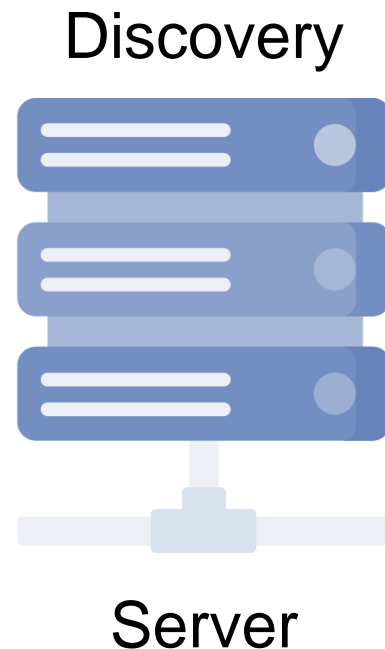
## Service Discovery : a través de la configuración



1. El servicio registra la ubicación
2. El cliente busca la ubicación del servicio
3. El servidor de descubrimiento devuelve la ubicación
4. El cliente solicita servicio en la ubicación
5. El servicio envía respuesta



## Service Discovery : a través de la configuración



Un registro de ubicaciones de servicio gestionado activamente

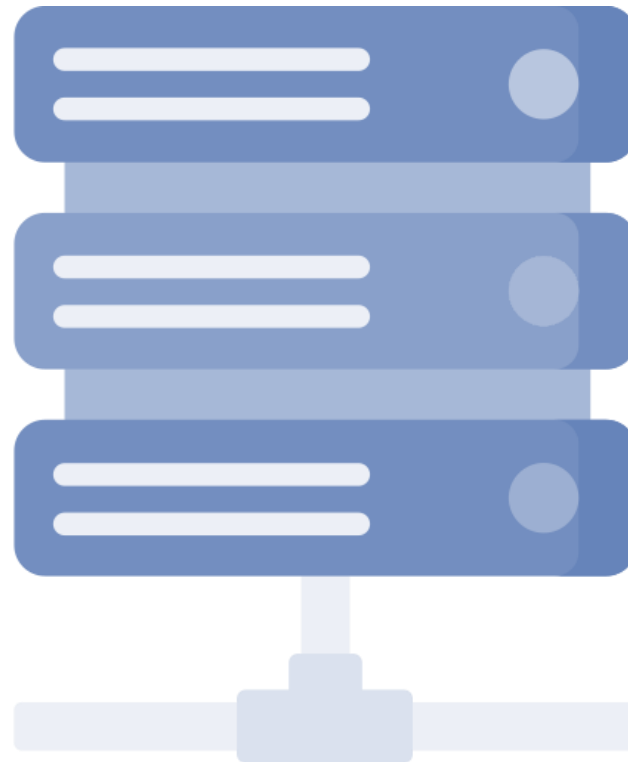
Un solo origen para una o más instancias

Proyecto Spring Cloud:

- **Servidor Spring Cloud Eureka**

# Creando e iniziando un discovery server

DEMO



## Servicio de que se conecta a un Discovery Server

Application



Service

- Proporcionar algunas funciones de la aplicación.
- El receptor de request
- Una dependencia de otros servicios de una o más instancias
- Uso del **discovery client**
  - Registrarse
  - Darse de baja

## Cliente que consume a un Discovery Server

Application

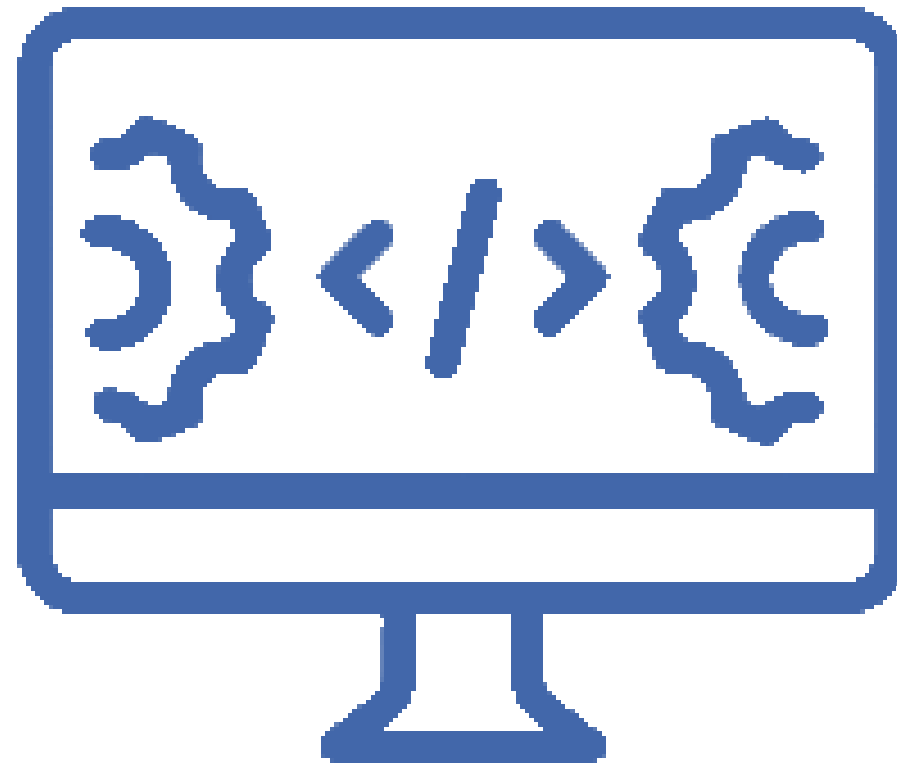


Client

- Llama a otro servicio de aplicación para implementar su funcionalidad
- El emisor de las solicitudes depende de otros servicios Usuario del cliente de descubrimiento
- Encuentra ubicaciones de servicio

# Creando un cliente que puede describir servicios

DEMO



# Spring Cloud Eureka Dashboard

## Cliente que consume a un Discovery Server

Application



Client

Habilitado por defecto

- **`eureka.dashboard.enabled=true`**

Muestra metadatos útiles y estado del servicio

## Servidor Eureka: ¿Estan saludables mis servicios?



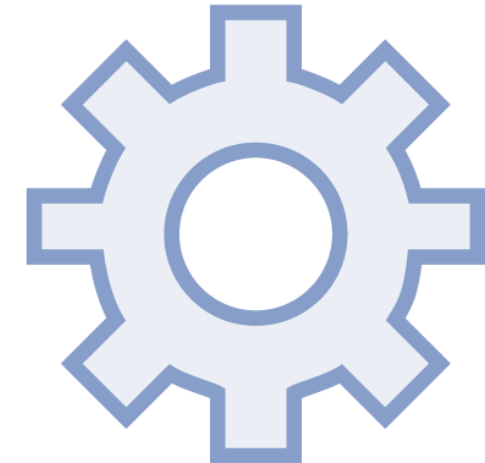
Comprueba  
regularmente el  
estado de los  
servicios.



Los clientes envían  
latidos cada 30  
segundos  
(predeterminado)



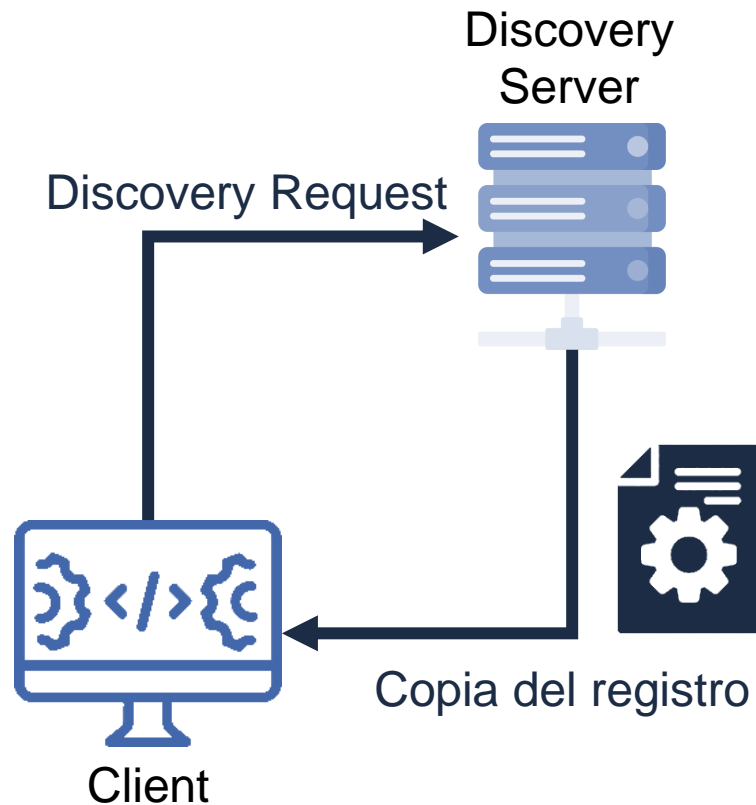
Servicios eliminados  
después de 90  
segundos sin latidos  
(predeterminado)



Puede personalizar la  
configuración para  
usar  
**/health endpoint**  
**eureka.client.**  
**healthcheck.enable**



## Servidor Eureka: ¿Estan saludables mis servicios?



1. El registro se distribuye (almacena en caché localmente en cada cliente)
2. Los clientes pueden operar sin servidor de descubrimiento
3. Obtiene deltas para actualizar el registro

# Spring Cloud Eureka AWS Support

## Spring Cloud Eureka AWS Support

AWS-specific instance data

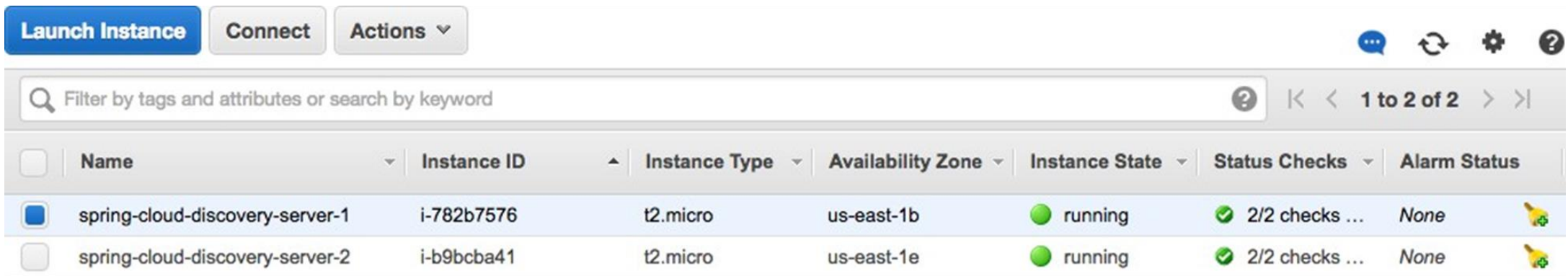
Multi-zone aware

Multi-region aware

Elastic IP Binding

## EC2 Dashboard

us-east-1



<input type="checkbox"/>	Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Status
<input checked="" type="checkbox"/>	spring-cloud-discovery-server-1	i-782b7576	t2.micro	us-east-1b	● running	✓ 2/2 checks ...	None
<input type="checkbox"/>	spring-cloud-discovery-server-2	i-b9bcba41	t2.micro	us-east-1e	● running	✓ 2/2 checks ...	None

```
eureka.client.availability-zones.us-east-1=us-east-1b,us-east-1e
```

Availability Zones Configuration in `application.properties`

```
eureka.client.availability-zones.[region]=[az1],[az2],[az3]
```

## Elastic IP Dashboard

us-east-1

Allocate New Address

Actions ▾

↺

⚙

?

Filter

VPC addresses ▾

🔍 Search Elastic IPs...

✕

⏪ < 1 to 2 of 2 Elastic IPs > ⏩

<input type="checkbox"/>	Address	Allocation ID	Instance ID	Network Interface ID	Scope	Private Address
<input type="checkbox"/>	34.192.167.121	eipalloc-bdf9e782	i-782b7576	eni-2d3231d2	vpc	172.31.52.243
<input type="checkbox"/>	34.193.24.166	eipalloc-59e3fd66	i-b9bcba41	eni-f1e0c418	vpc	172.31.33.117

```
eureka.client.service-url.us-east-1b=
```

```
http://ec2-34-192-167-121.compute-1.amazonaws.com:8761/eureka/
```

```
eureka.client.service-url.us-east-1e=
```

```
http://ec2-34-193-24-166.compute-1.amazonaws.com:8761/eureka/
```

### Service URL Configuration in `application.properties`

```
eureka.client.service-url.[zone]=http://[eip-dns]/eureka
```

- \*Use EIP DNS name. Do not use IP (as of version Eureka 1.4)

## Eureka Dashboard: AWS Multi-zone Discovery Servers

### DS Replicas

ec2-34-192-167-121.compute-1.amazonaws.com

ec2-34-193-24-166.compute-1.amazonaws.com

### Instances currently registered with Eureka

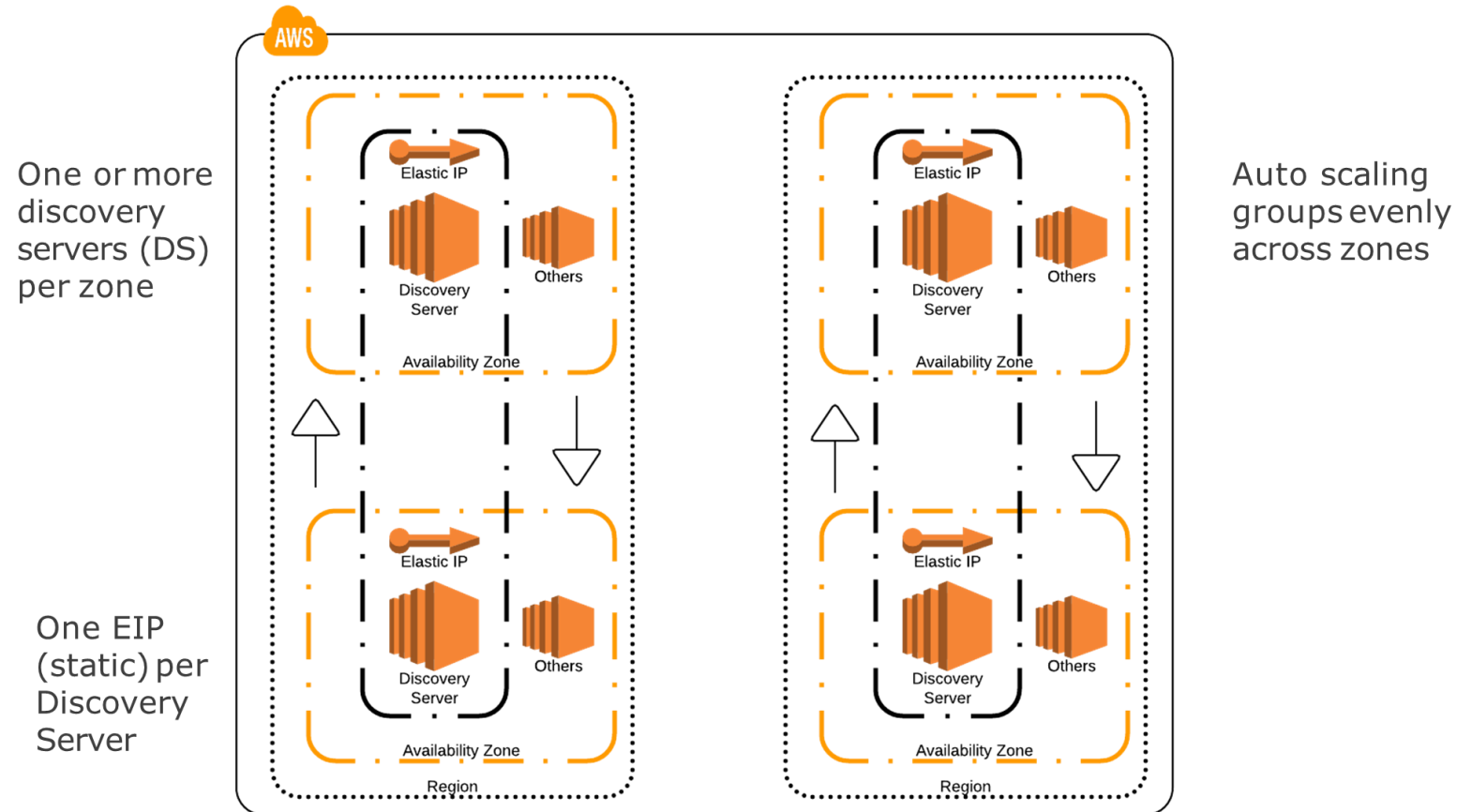
Application	AMIs	Availability Zones	Status
DISCOVERY-SERVER-1	ami-40d28157 (1)	us-east-1b (1)	UP (1) - i-782b7576
DISCOVERY-SERVER-2	ami-40d28157 (1)	us-east-1e (1)	UP (1) - i-b9bcba41

## Eureka Dashboard: AWS Instance Data

Instance Info	
Name	Value
public-ipv4	34.192.167.121
public-hostname	ec2-34-192-167-121.compute-1.amazonaws.com
instance-id	i-782b7576
instance-type	t2.micro
ami-id	ami-40d28157
ipAddr	172.31.52.243
status	UP
availability-zone	us-east-1b

# → Registro y discovery de microservicios

## Eureka Dashboard: AWS Instance Data



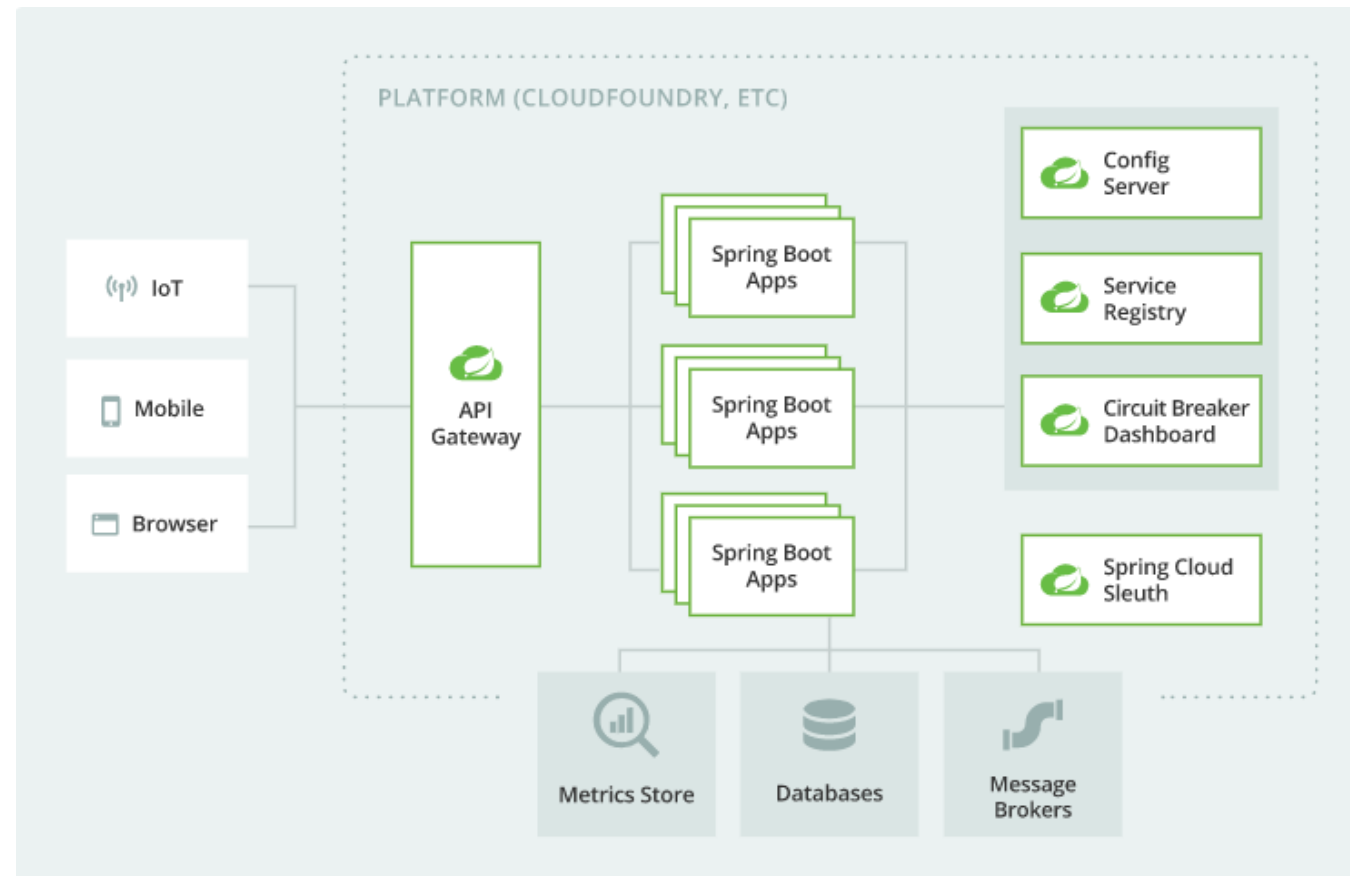


03



Steeltoe

# Spring Cloud



## ¿Qué es Steeltoe?



Steeltoe es un proyecto de código abierto destinado a **desarrollar aplicaciones de microservicios .NET** nativas de la nube. Este proyecto proporciona bibliotecas que siguen patrones de desarrollo similares de bibliotecas de microservicios conocidas y probadas **como Netflix OSS, Spring Cloud** y otras.

Las bibliotecas de Steeltoe se crean sobre las API de .NET. Steeltoe le **permite trabajar con .NET Core/.NET 6** con **Steeltoe 3.x** y .NET Framework 4.x con Steeltoe 2.x.

<https://github.com/SteeltoeOSS/Steeltoe/wiki>

## Características



### **Application Configuration**

- Config Server Provider
- Random Value Provider
- Placeholder Provider
- Cloud Foundry Provider
- Kubernetes Providers
- Hosting Extensions

### **Circuit Breakers**

### **Distributed Tracing**

### **Dynamic Logging**

### **Management**

- Using Endpoints (Actuators)

### **Messaging**

- RabbitMQ

### **Network File Sharing**

## Características



### Security

- JWT for ASP.NET Core
- Redis Key Storage Provider
- SSO Openid
- SSO OAuth2
- Mutual SSL
- CredHub Api Client

### Service Connectors

- MySQL Database
- Microsoft SQL Database
- Redis

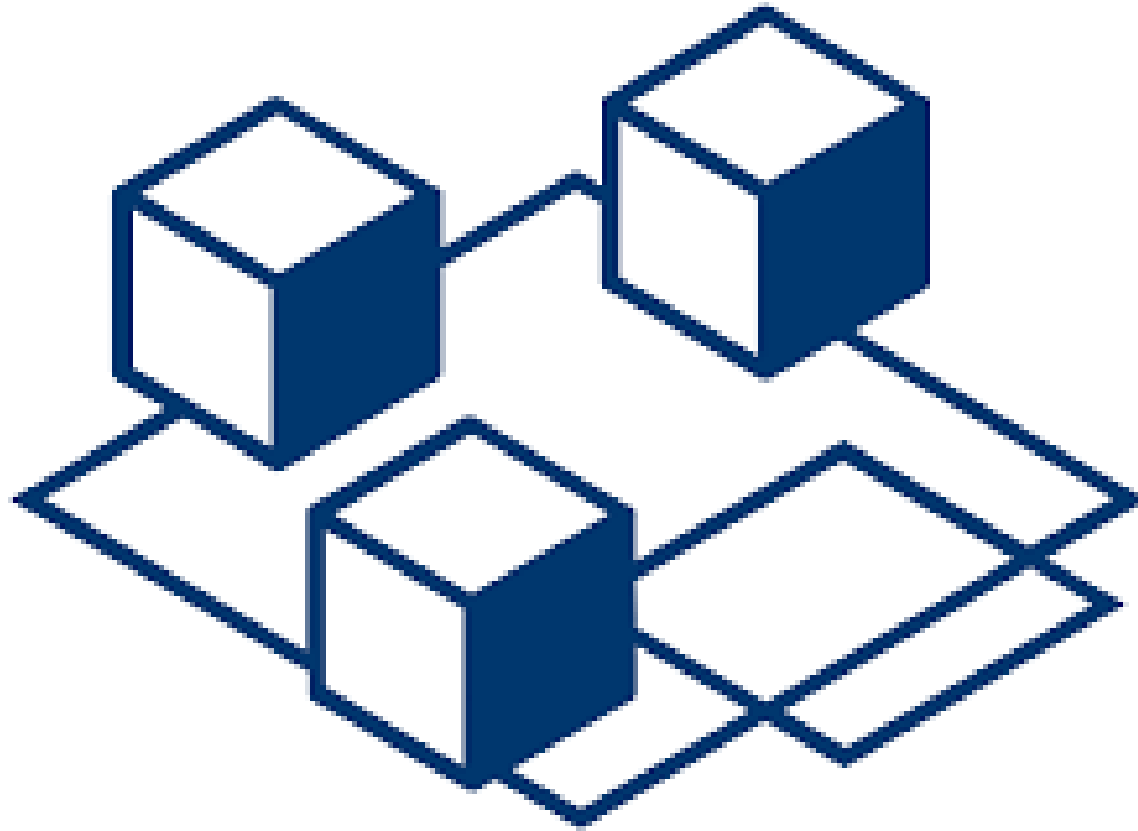
### Service Discovery

- Hashicorp Consul Registry
- Eureka Registry

### Stream

# Servicios con Steeltoe

DEMO



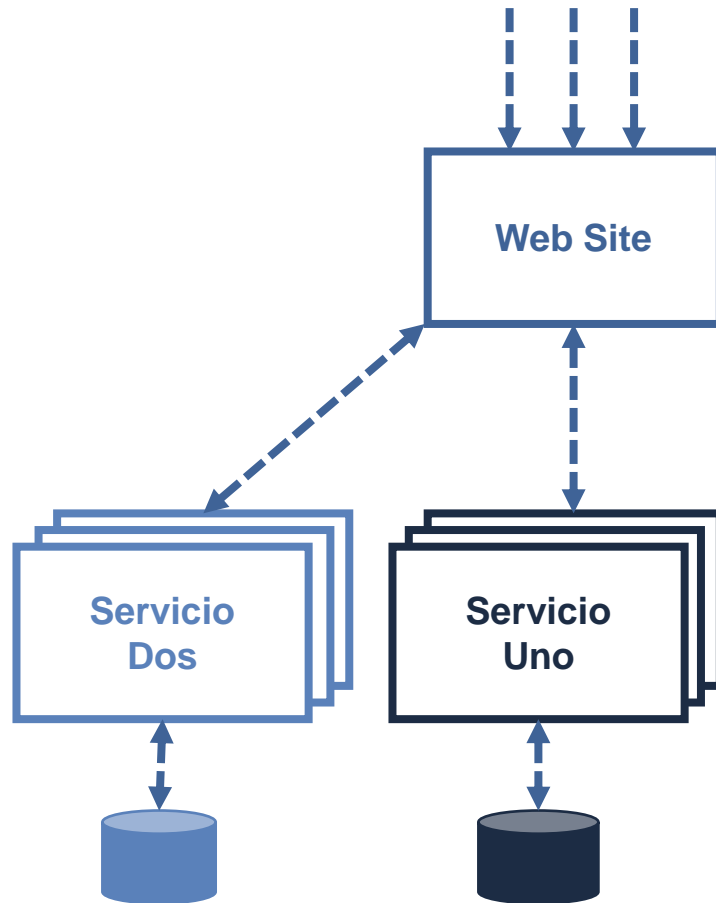
04



Resiliencia y alta disponibilidad de  
microservicios.

# → Resiliencia y alta disponibilidad de microservicios

## Introducción



### Necesidad de resiliencia

- Arquitectura distribuida
- Comunicación a través de una red
- Se requiere tolerancia a fallas
- Evite fallas en cascada
- Evite agotar las reservas de recursos

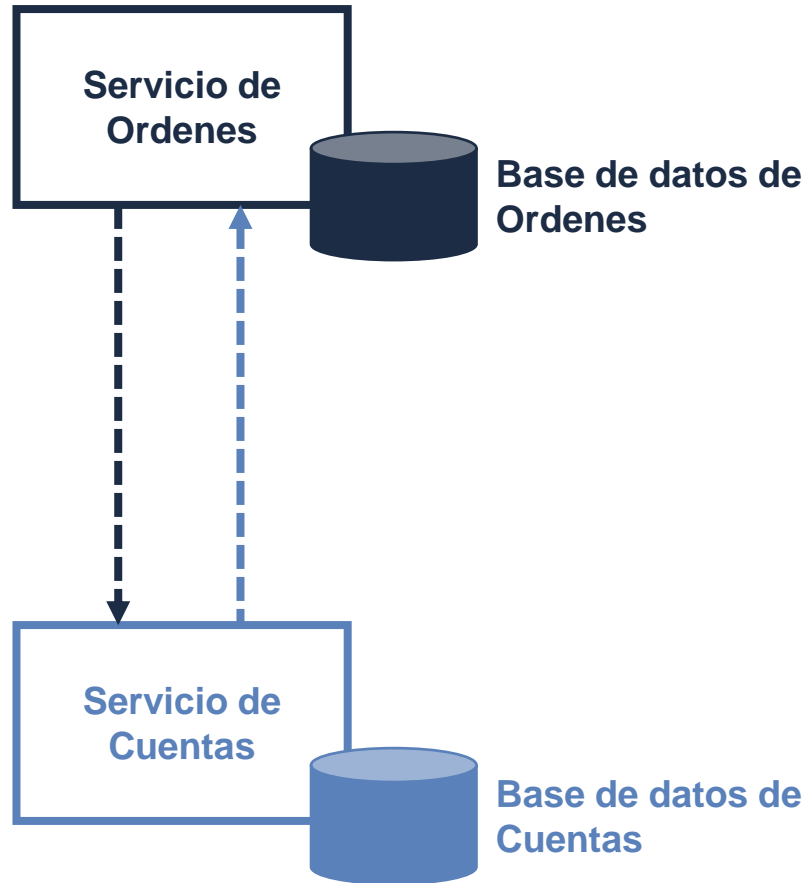
### Tipos de fallas

- Hardware
- Infraestructura
- Capa de comunicación
- Dependencias
- Microservicios



# → Resiliencia y alta disponibilidad de microservicios

## Resiliencia ¿Cómo?



### Diseño para fallas conocidas

- Mira las causas de fallas anteriores
- Diseñe un solo punto de fallas
- Usar el patrón bulkhead

### Acepta el fracaso

- Disyuntores/reintentos con monitorización

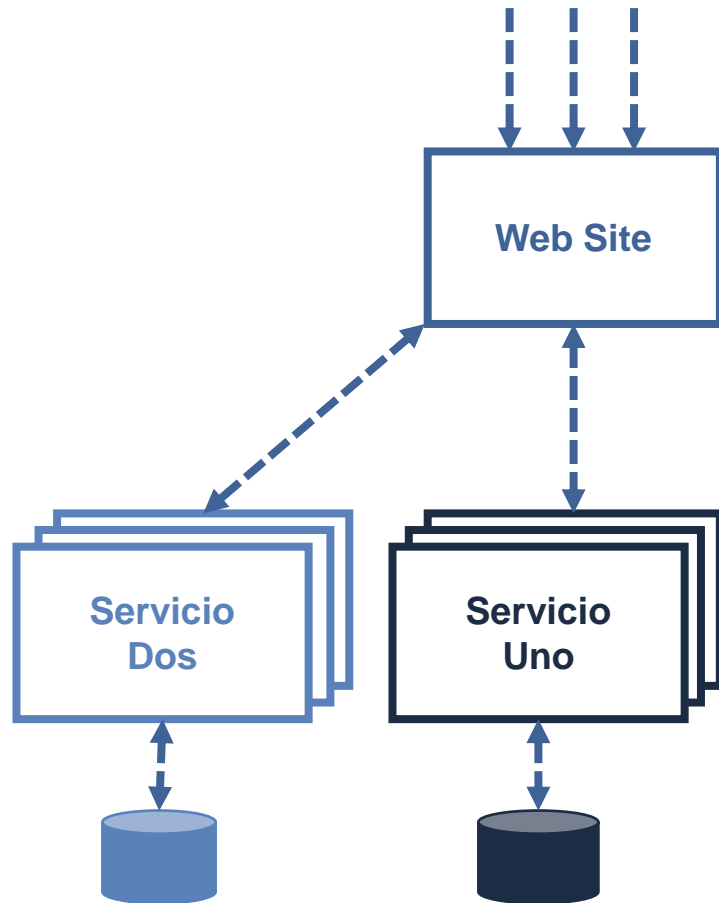
### Fallar rápido

- Usar patrones de diseño de resiliencia
- Falla rápido para la solicitud entrante
- Uso de tiempos de espera en llamadas salientes

### Funcionalidad degradada

- Circuit breakers
- Usar cachés

## Patrones y enfoques



### Patrones de diseño para la resiliencia

- Timeouts
- Circuit Breaker
- Retry
- Bulkhead

### Enfoque del desarrollo de software

- Diseño para fallas conocidas
- Acepta los fracasos
- Fallar rápido
- Funcionalidad degradada

05



Principales patrones de resiliencia  
(Circuit Breaker, Restry Desig y  
Bulkheads Design).

# → Principales patrones de resiliencia

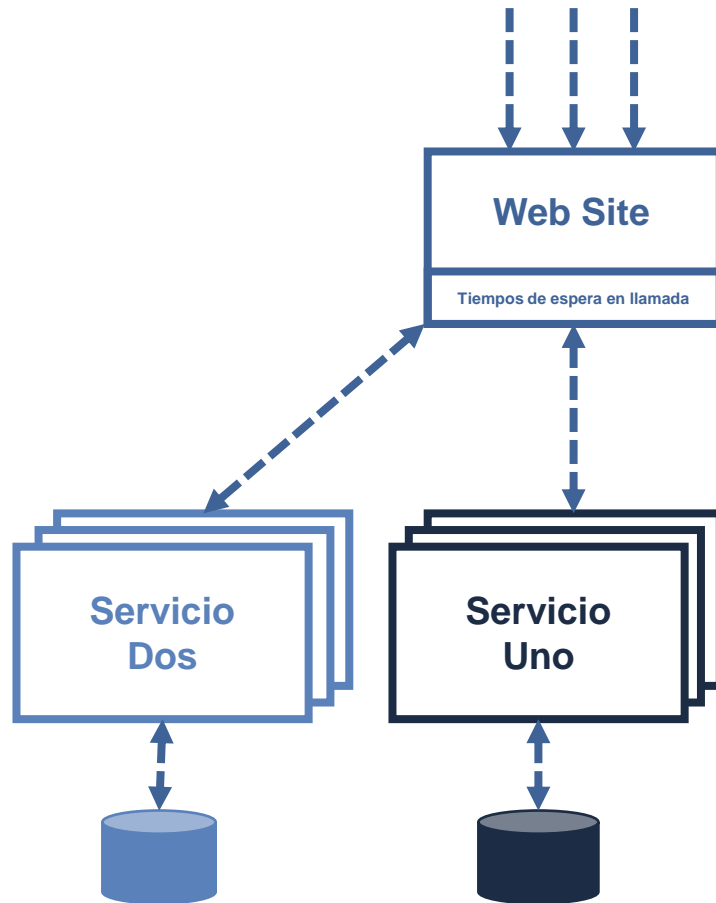
## Timeouts

### Código de llamada del cliente

- Configurar explícitamente el valor del tiempo de espera
- No confíe en los valores de tiempo de espera predeterminados
- Valor configurable
- Puede usarse en reintentos

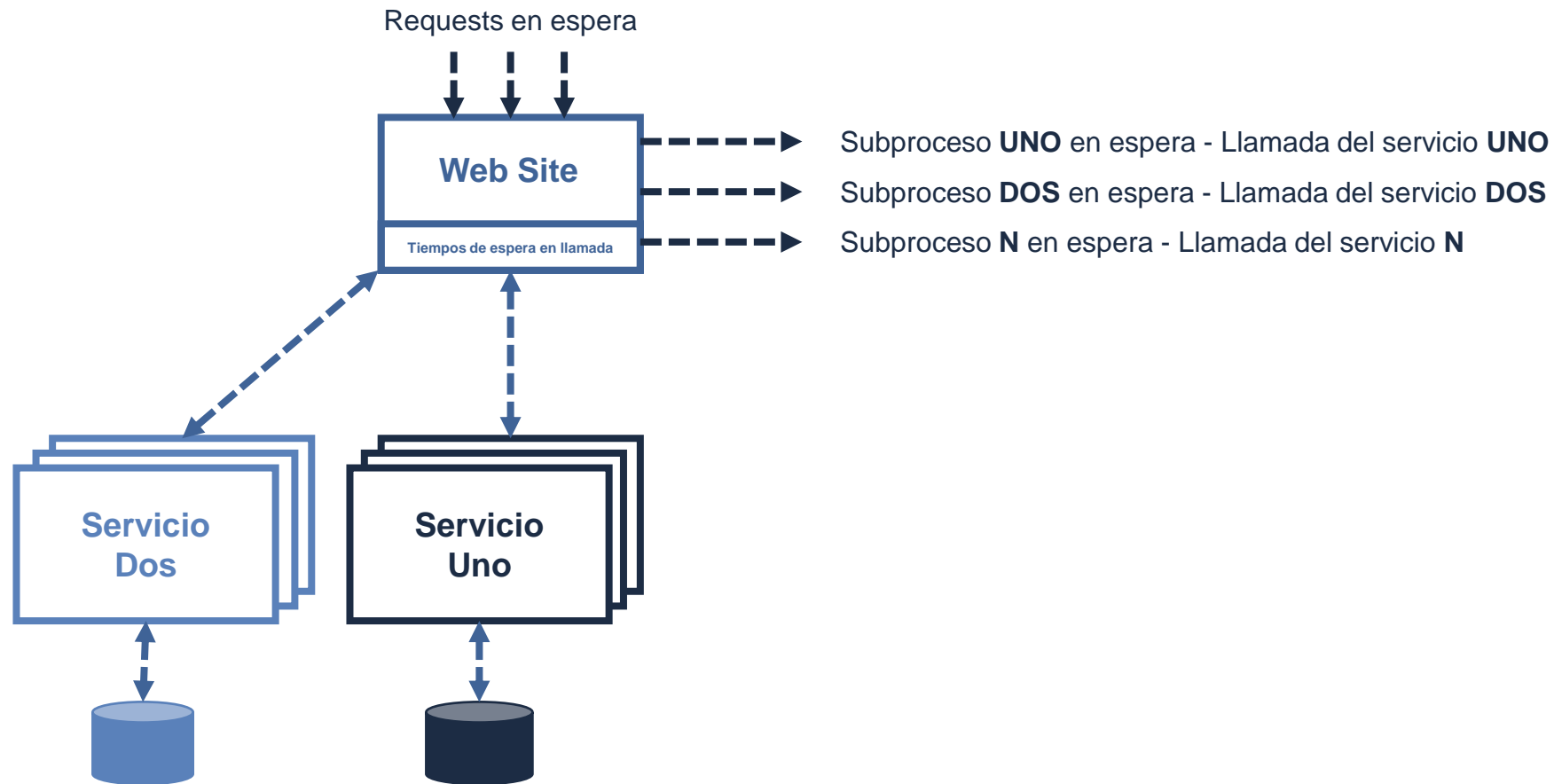
### Ventaja de los Timeouts

- Evita esperar una respuesta para siempre
- Libera el hilo de llamada
- Le permite manejar fallas de llamadas
- Compatible con casi todas las tecnologías cliente



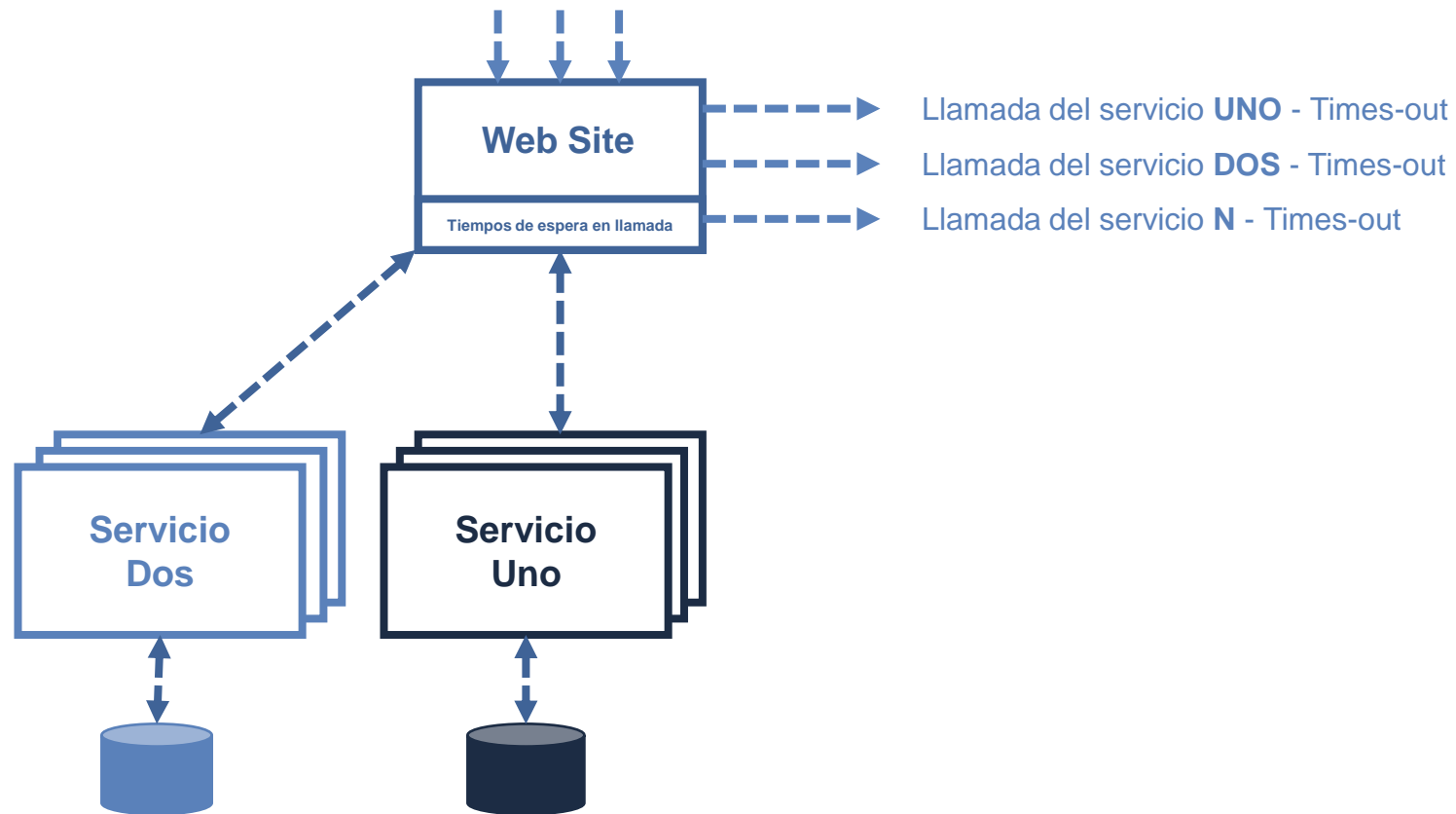
# → Principales patrones de resiliencia

## Degradado de funcionalidad



# → Principales patrones de resiliencia

## Timeouts al rescate



## ¿Qué es Circuit Breaker?



Un disyuntor es un interruptor eléctrico operado automáticamente **diseñado para proteger un circuito eléctrico** de daños causados por el exceso de corriente de una sobrecarga o cortocircuito.

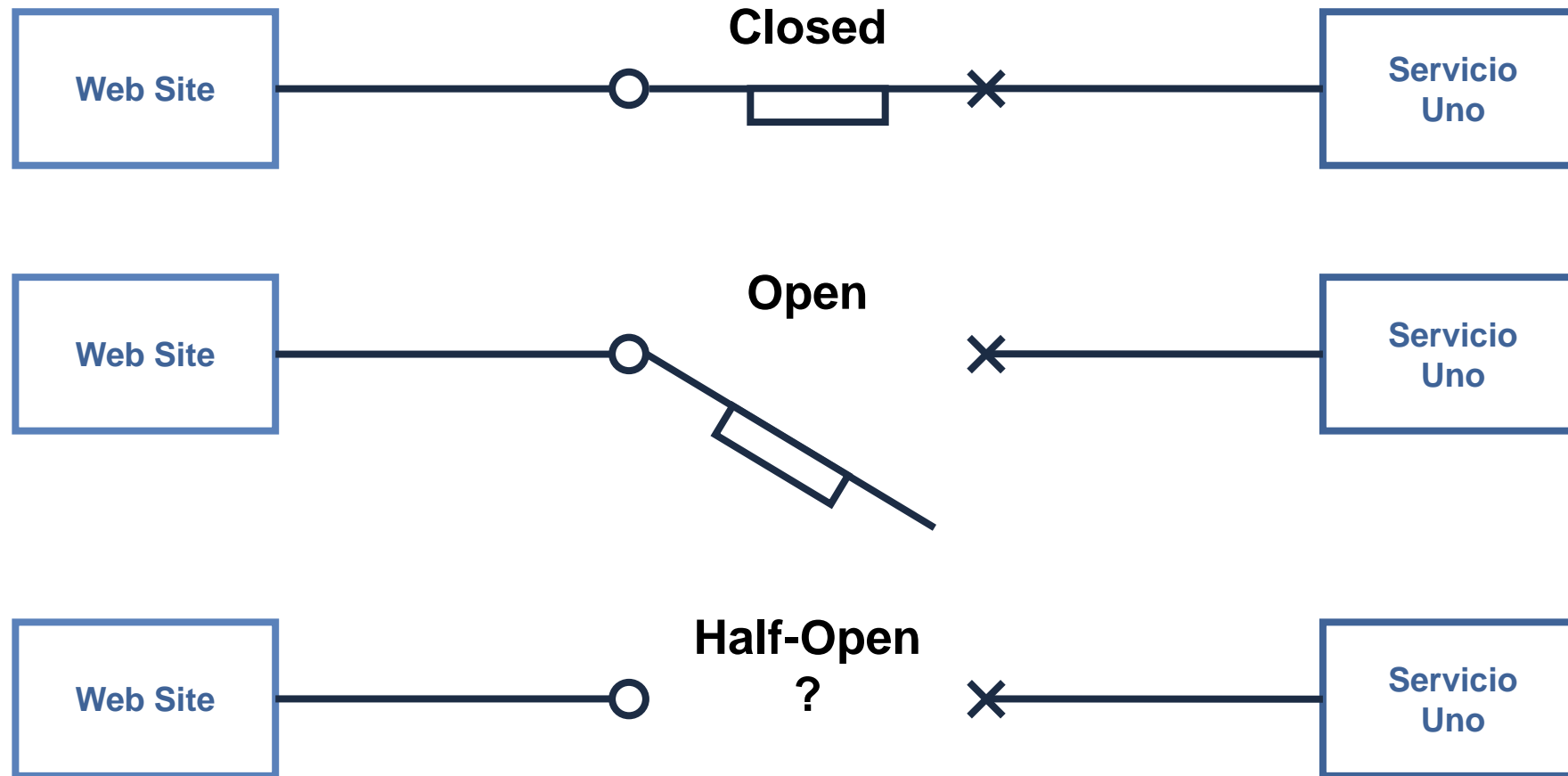
Su **función básica es interrumpir el flujo de corriente después de detectar un fallo**. A diferencia de un fusible que funciona una vez y luego debe ser reemplazado, un disyuntor se puede restablecer (ya sea manual o automáticamente) para reanudar el funcionamiento normal.

El patrón de disyuntor **se utiliza para detectar fallos y encapsula la lógica de evitar que un fallo se repita constantemente**, durante el mantenimiento, fallas temporales del sistema externo o dificultades inesperadas del sistema.



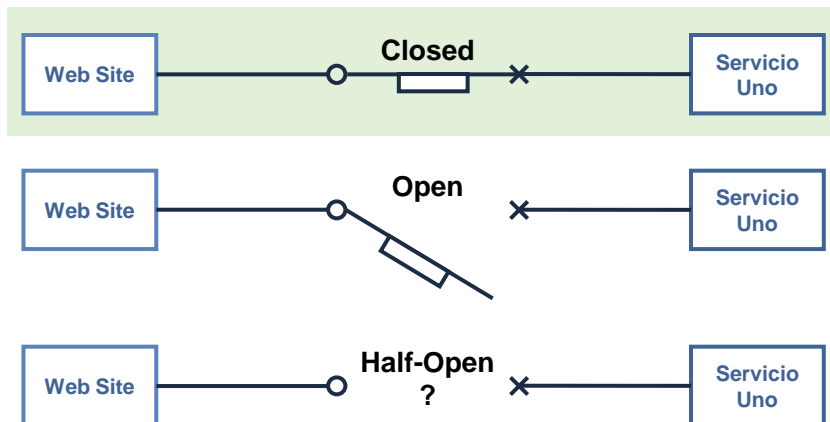
# → Principales patrones de resiliencia

## Circuit Breaker stages



# → Principales patrones de resiliencia

## Circuit Breaker stages



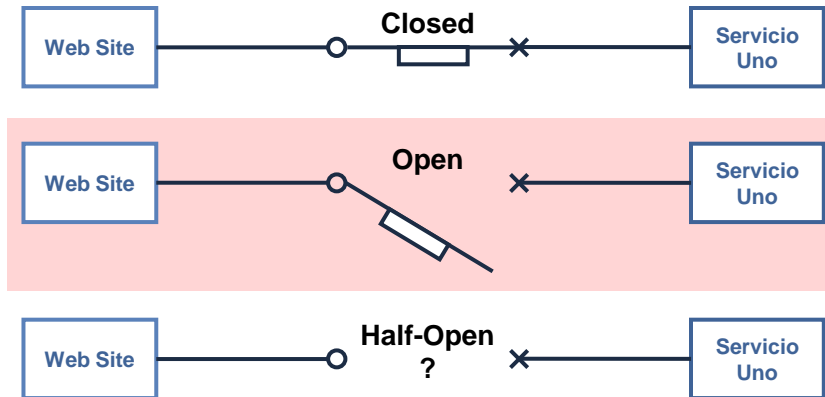
### CERRADO

Inicialmente, el disyuntor entra en un estado CERRADO y espera las solicitudes de cliente. Recibe solicitudes de cliente y realiza una llamada al Servicio. **Si se realiza correctamente** y recibe la respuesta de ese servicio, **restablecerá el recuento de errores a 0** y enviará esa respuesta al usuario final.

**Si no puede recibir la respuesta de ese servicio, aumentará el recuento de errores en uno** y comprobará si ese recuento es mayor que el umbral de error predefinido. **Si el recuento de errores es mayor que el umbral de error**, el componente Circuit Breaker se activa o entra en el estado OPEN

# → Principales patrones de resiliencia

## Circuit Breaker stages



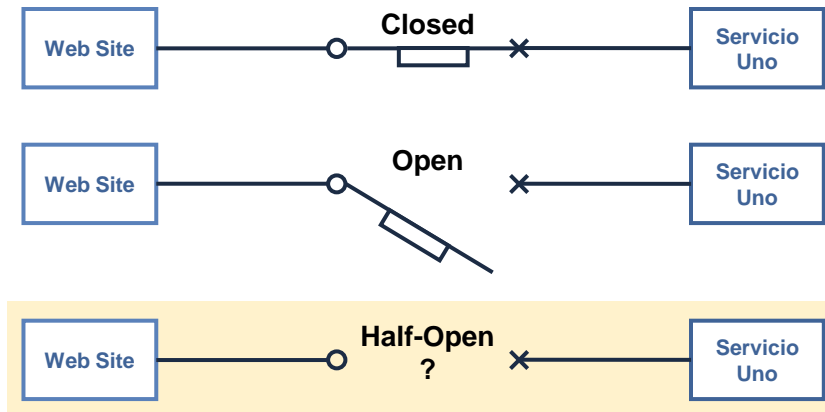
### OPEN

El componente Circuit Breaker entra en este estado cuando el **conteo de errores es mayor que el umbral de falla**. Cuando está en este estado, **no hace una llamada al Servicio**. Cuando está en este estado, si el cliente le envía alguna solicitud, no hará una llamada al Servicio; **sólo envía una excepción y espera algún tiempo**, es decir, el valor de tiempo de espera especificado.

**Una vez que expire el valor de Tiempo de espera, entrará en el estado HALF-OPEN**

# → Principales patrones de resiliencia

## Circuit Breaker stages



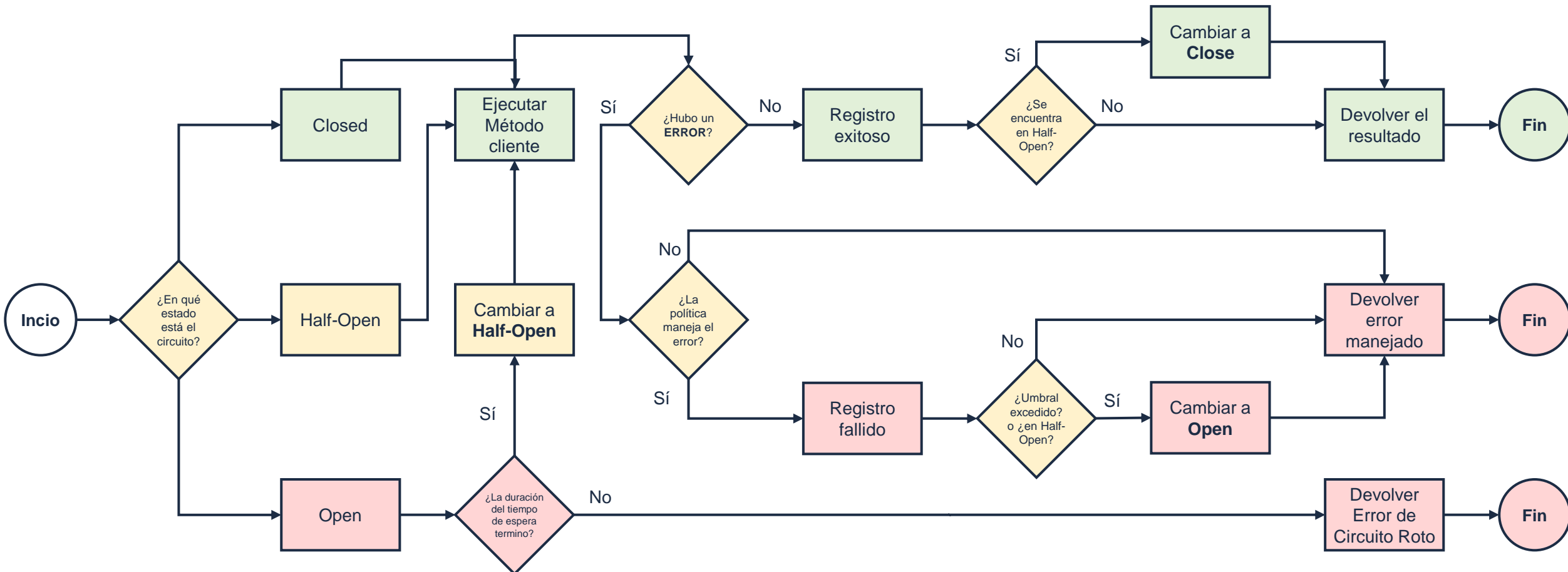
### HALF-OPEN

El componente entra en este estado **cuando finaliza el tiempo de espera predefinido. Envía la primera solicitud** al Servicio. Si recibe una respuesta de **éxito, entrará en el estado CERRADO** para procesar más solicitudes de cliente.

Antes de recoger la primera solicitud de cliente, **restablecerá el recuento de errores a 0** de nuevo. Si recibe una **respuesta de error, volverá al estado OPEN** y esperará a que expire un valor de tiempo de espera predefinido.

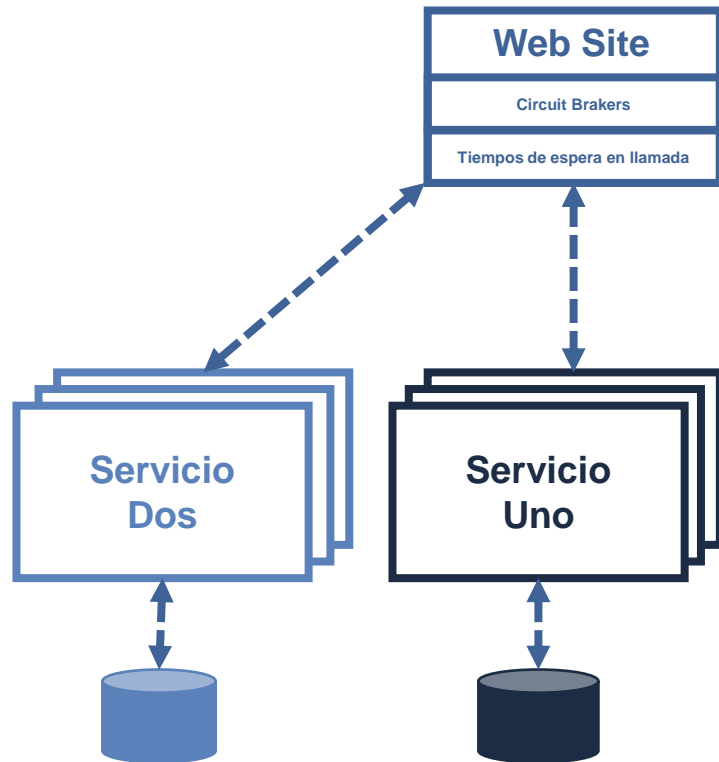
# Principales patrones de resiliencia

## Circuit Breaker stages



# → Principales patrones de resiliencia

## Estrategias de implementación de Circuit Breaker



Estrategia	Implementación	Necesidad
Black Box	<ul style="list-style-type: none"> <li>Proxies</li> <li>Service Mesh               <ul style="list-style-type: none"> <li>Istio</li> <li>Linkerd</li> <li>Dapr (*)</li> </ul> </li> </ul>	Fallar rápido
White Box	Librerías <ul style="list-style-type: none"> <li>Hystrix</li> <li>Resilience4J</li> <li>Polly</li> </ul>	Fallbacks que se basan en lógica de negocio

# → Principales patrones de resiliencia

## Reintentos (Retry)

### Ideal para fallas transitorias

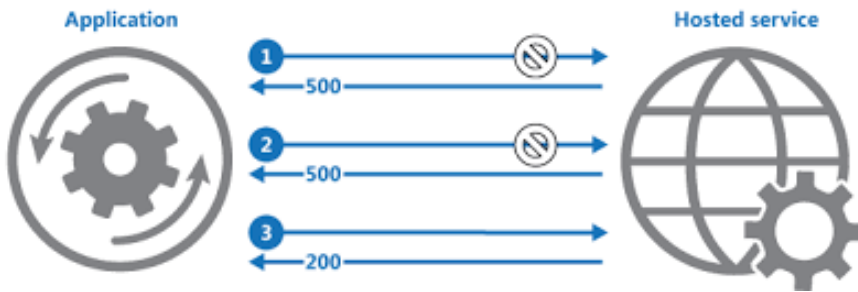
- Pérdida momentánea de conectividad
- Indisponibilidad temporal del servicio
- Tiempos de espera cuando el servicio está ocupado
- El servicio acelera las solicitudes aceptadas
- Errores autocorregidos

### Estrategias

- Reintentar inmediatamente
- Reintentar después de una espera
- Cancelar

Registrar y monitorear ocurrencias

Usar junto con disyuntores



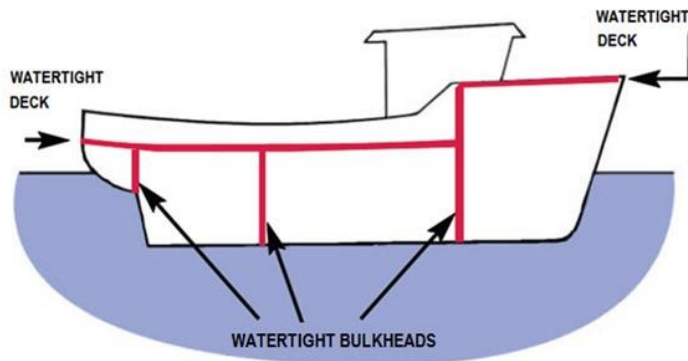
# → Principales patrones de resiliencia

## Bulkheads

### ¿Qué son los Bulkheads?

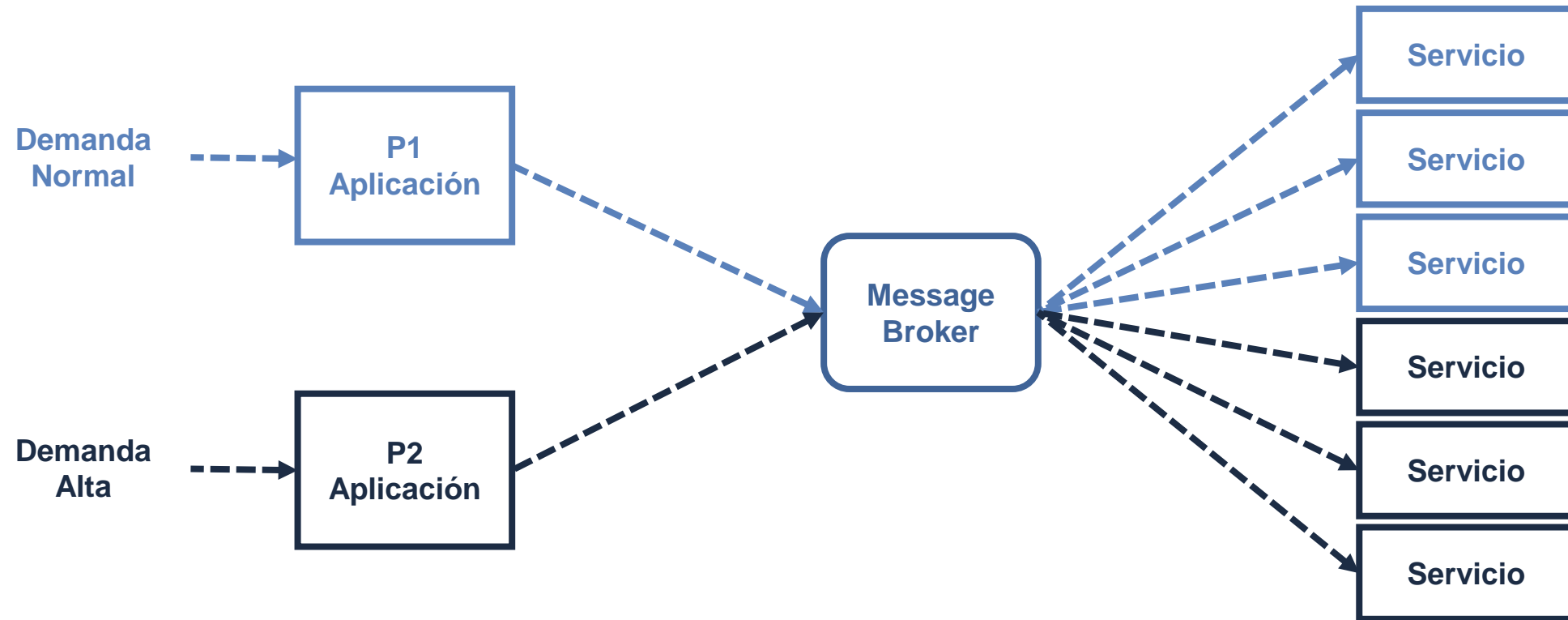
Bulkheads es una forma de particionar una aplicación. Proporcionan una manera de simultaneidad enlazada y para limitar una serie de acciones simultáneas. El término Bulkheads se origina en el envío y se refiere a la partición de partes de un barco.

La biblioteca de Polly explica: **Un Bulkheads es una pared dentro de un barco que separa un compartimento de otro, de tal manera que el daño a un compartimiento no hace que todo el barco se hunda.**



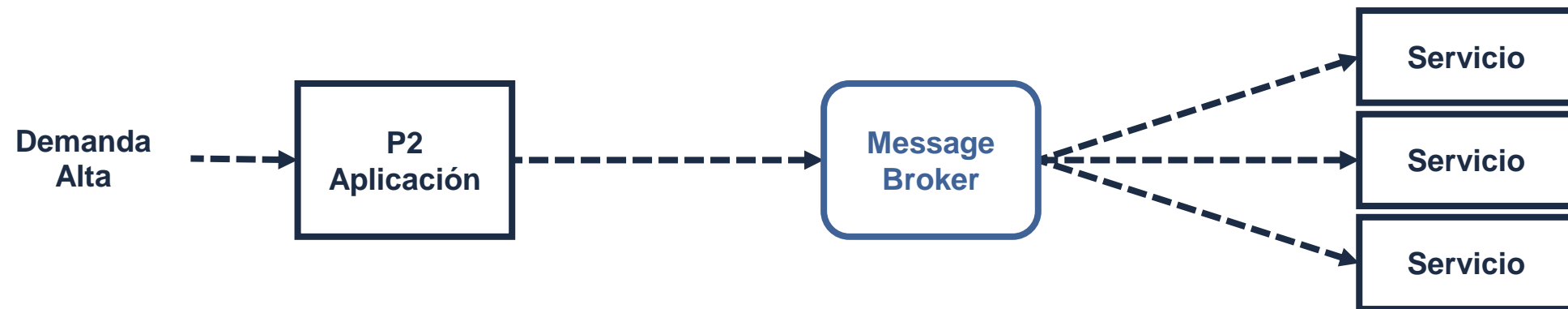
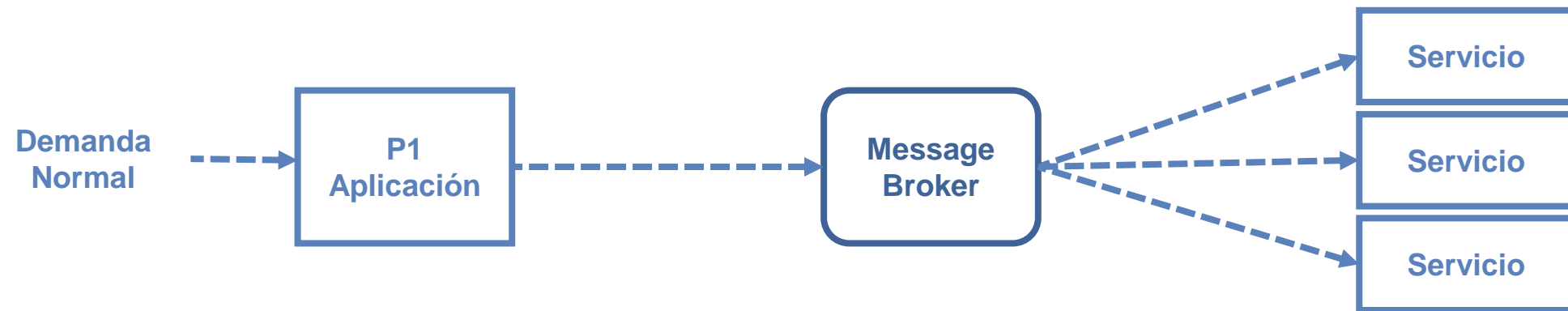


## Bulkheads : Separación por criticidad

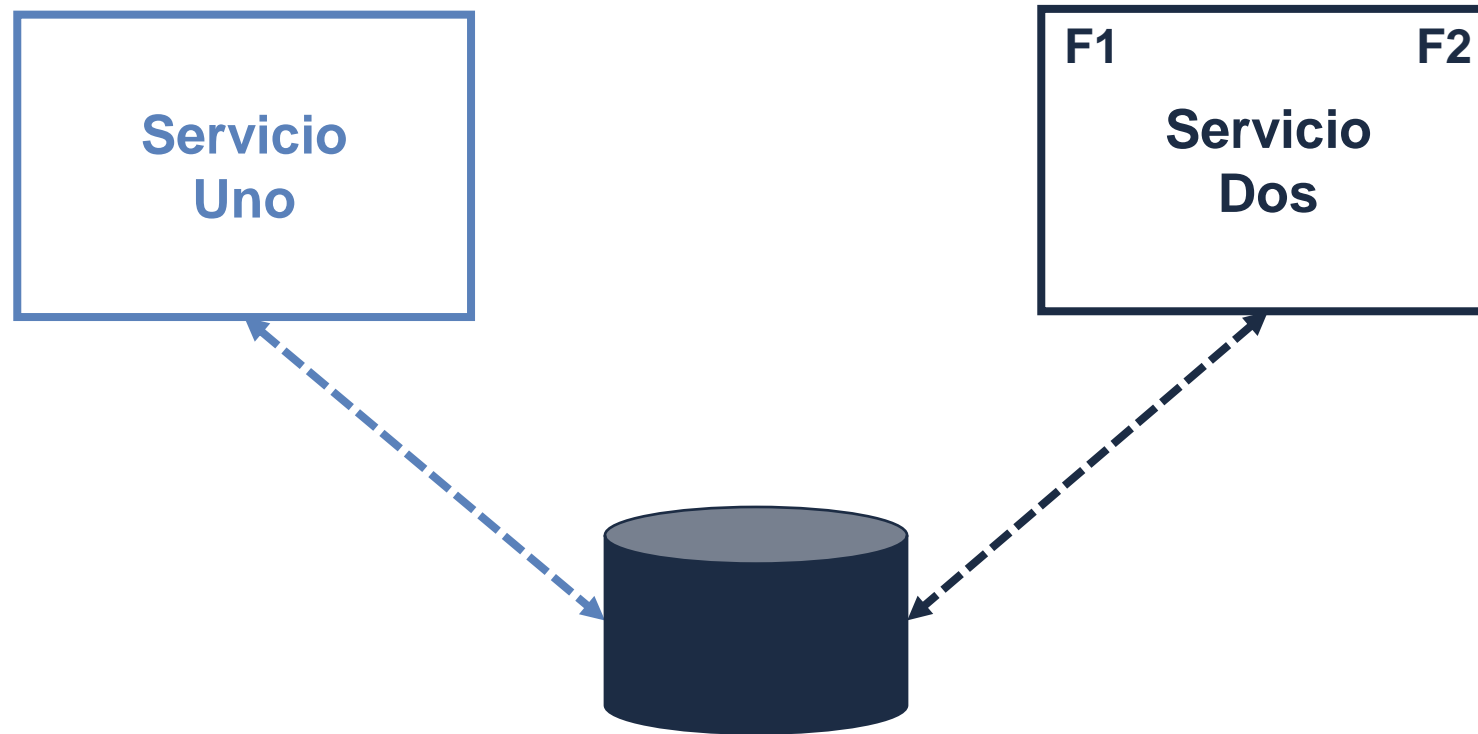


# → Principales patrones de resiliencia

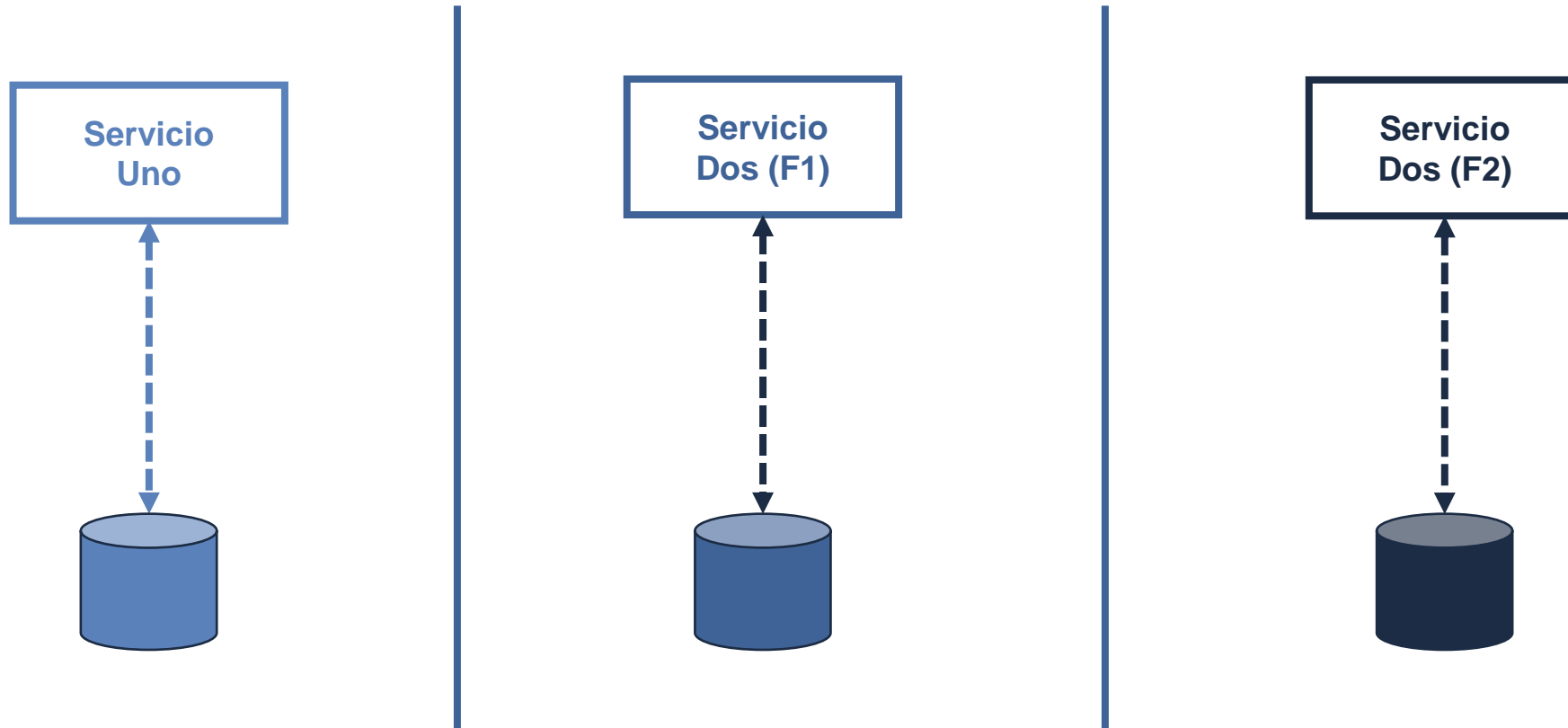
## Bulkheads : Separación por criticidad



## Bulkheads : Aislar microservicios

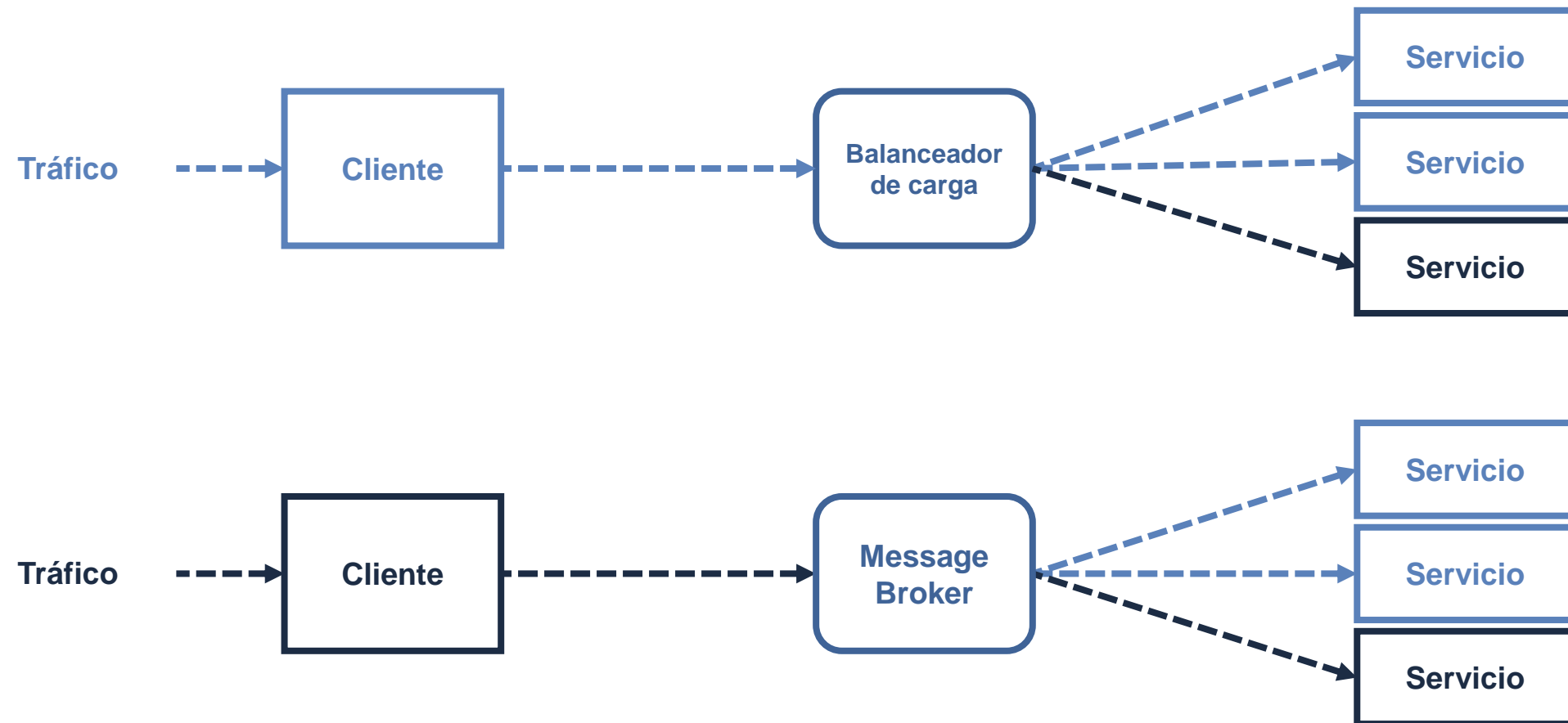


## Bulkheads : Aislar microservicios



# → Principales patrones de resiliencia

## Bulkheads : Redundancia



DEMO



# Polly

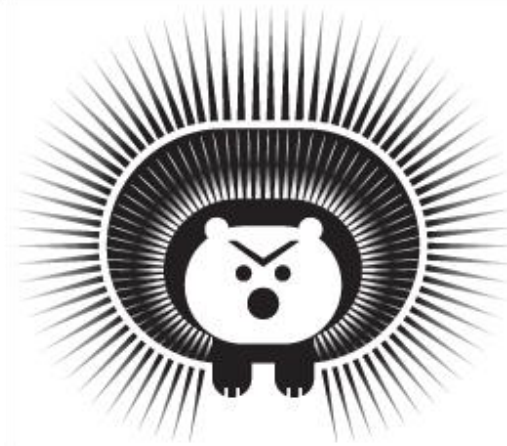


Polly es una biblioteca de control de errores transitorios y resiliencia de .NET que permite a los desarrolladores expresar directivas como Retry, Circuit Breaker, Timeout, Bulkhead Isolation, and Fallback de una manera fluida y segura para subprocessos. Polly tiene como destino .NET 4.0, .NET 4.5 y .NET Standard 1.1.

<http://www.thepollyproject.org/>

# Polly

DEMO



**HYSTRIX**  
DEFEND YOUR APP

Hystrix es una biblioteca que le ayuda a controlar las interacciones entre estos servicios distribuidos mediante la adición de tolerancia de latencia y lógica de tolerancia a errores. Hystrix hace esto aislando los puntos de acceso entre los servicios, deteniendo los errores en cascada en ellos y proporcionando opciones de reserva, todo lo cual mejora la resistencia general del sistema.

<https://github.com/Netflix/Hystrix/>

## Service mesh

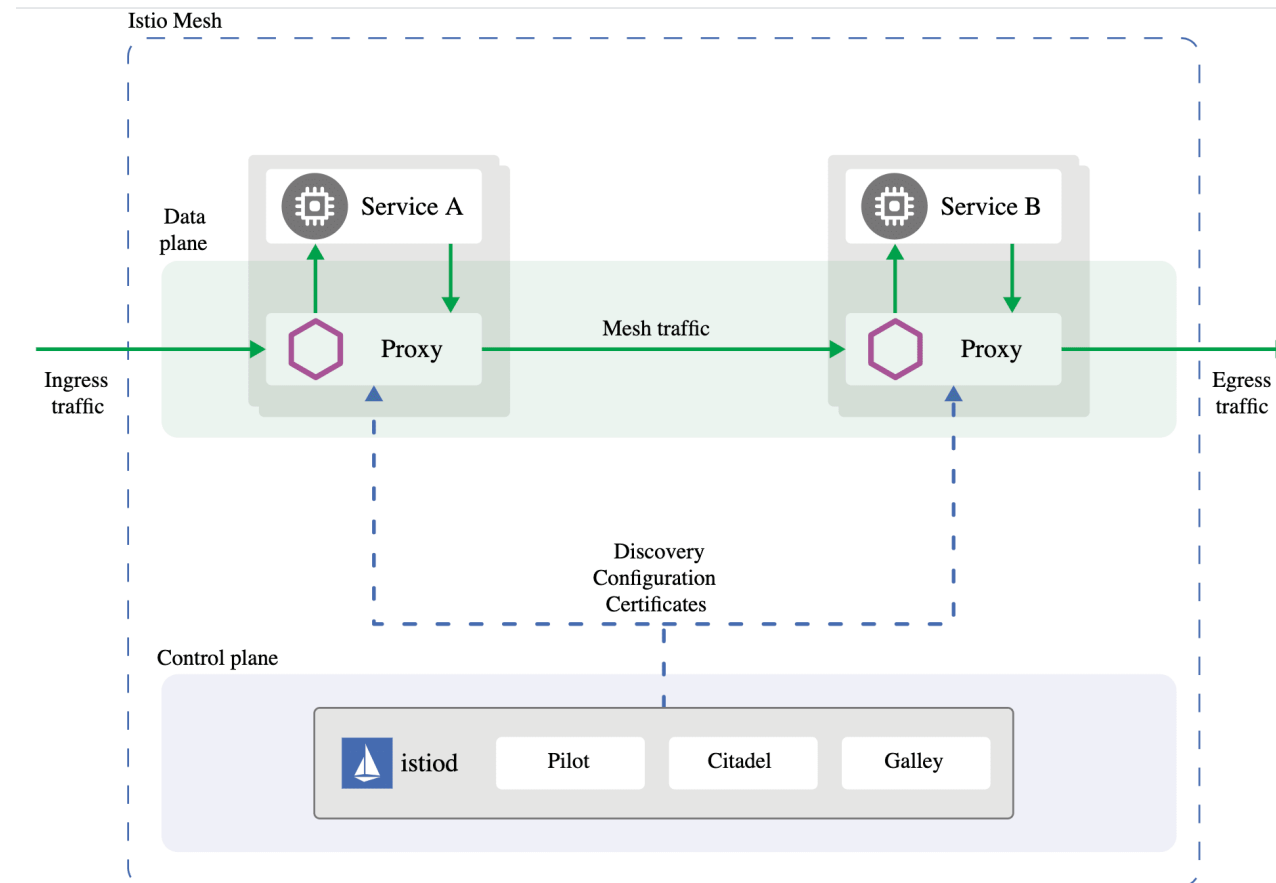
Un Service mesh es una **capa de infraestructura configurable** para las aplicaciones basadas en microservicios. Hace que la **comunicación entre instancias de servicio sea flexible, confiable y rápida.**

El mesh proporciona **Servicio de descubrimiento, balanceo de carga**, encriptación, autenticación y autorización, soporte para el patrón **Circuit Breaker** y otras capacidades.



# → Principales patrones de resiliencia

## Service Mesh





**GRACIAS**  
**POR SU PREFERENCIA**

