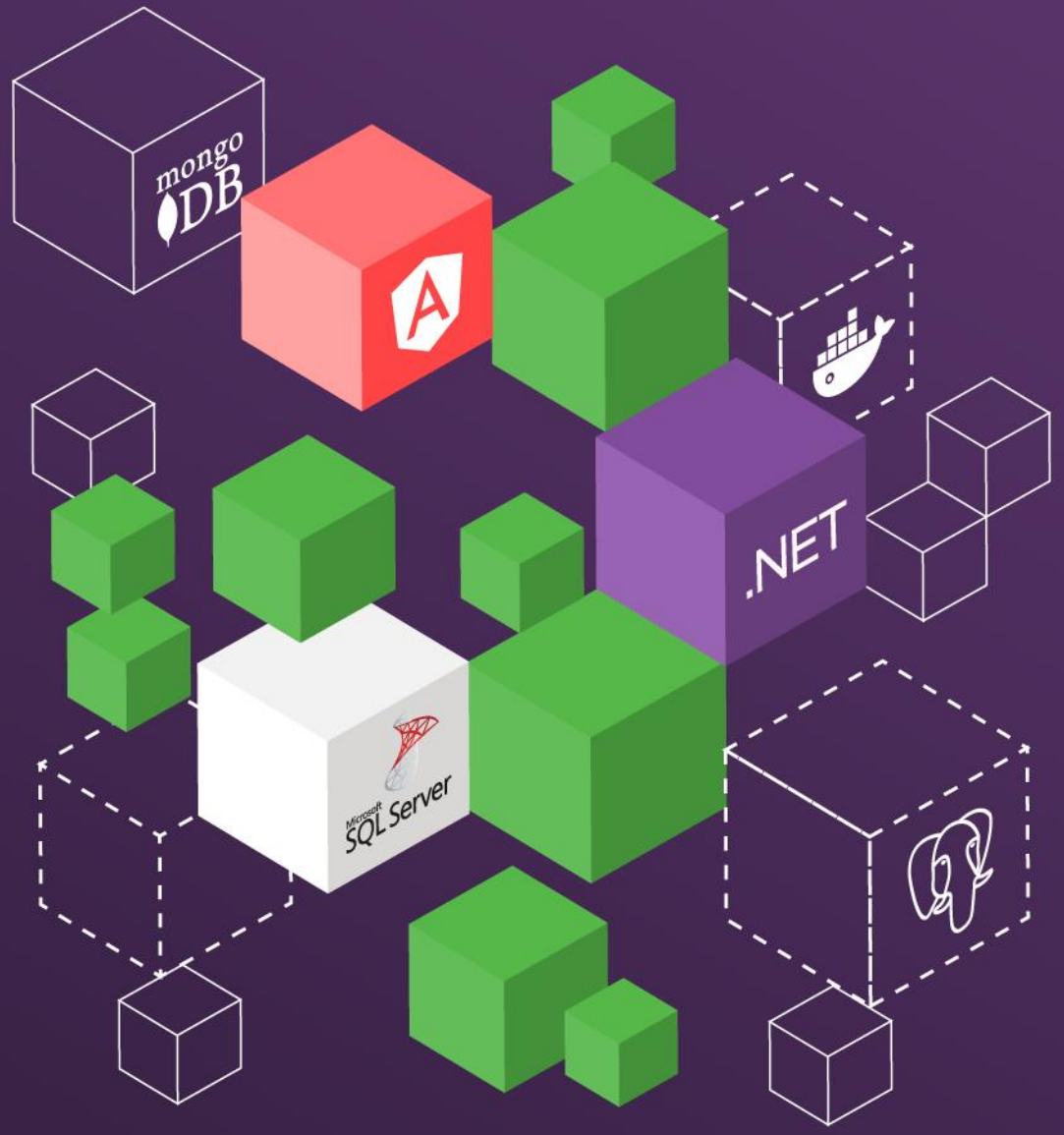


BIENVENIDOS
AL CURSO:

Arquitectura de Microservicios en Net

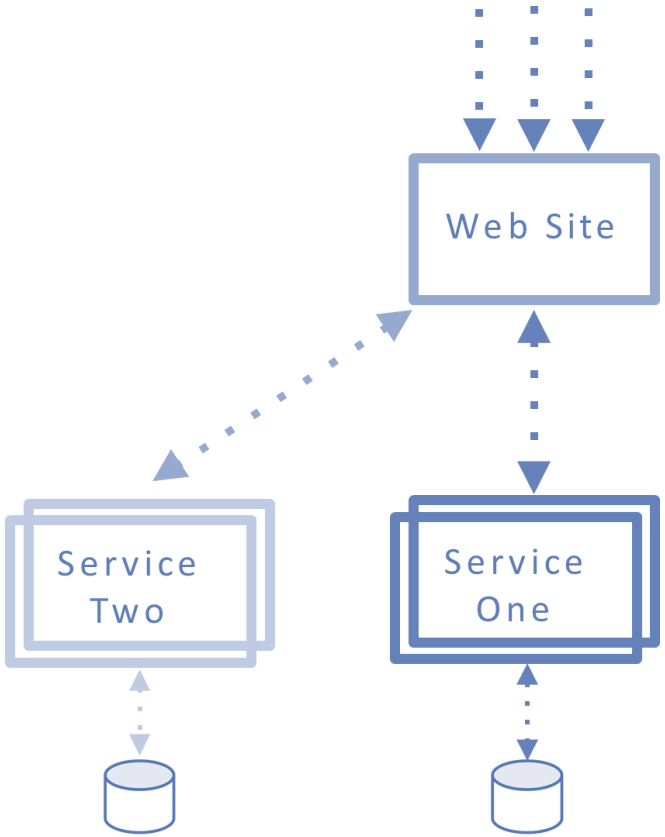
SESIÓN 05

GALAXY
TRAINING

ÍNDICE

Resiliencia y alta disponibilidad de microservicios.

Introducción



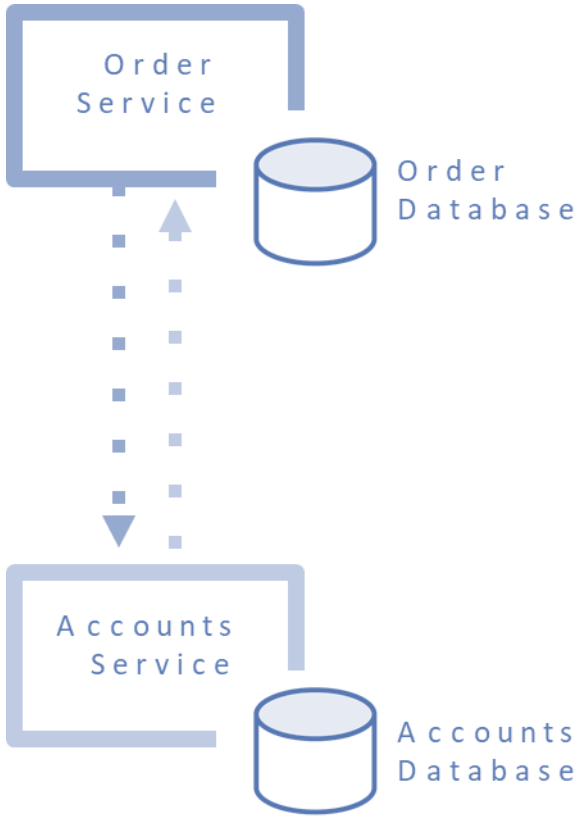
Necesidad de resiliencia

- 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 ¿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

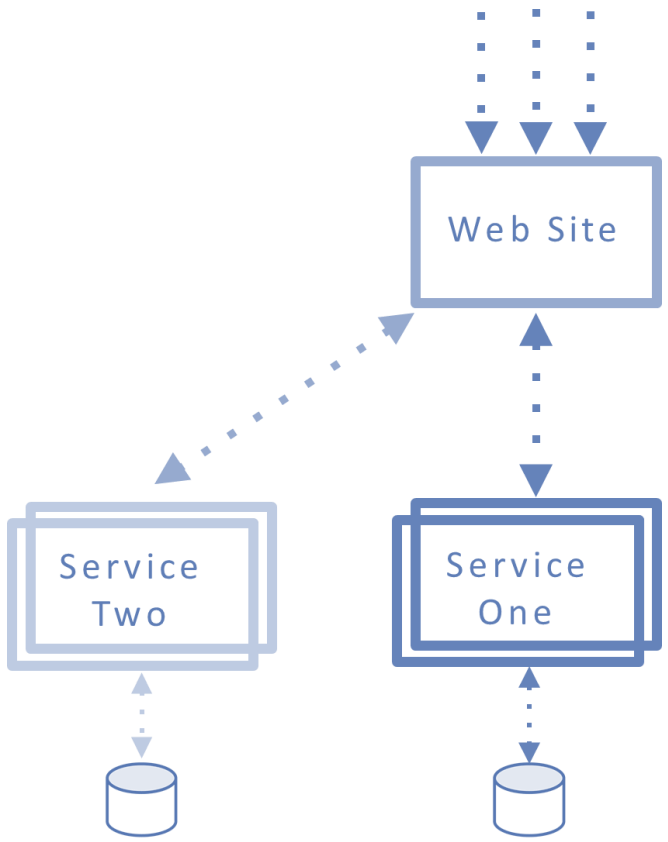
Fallar rápido

- 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

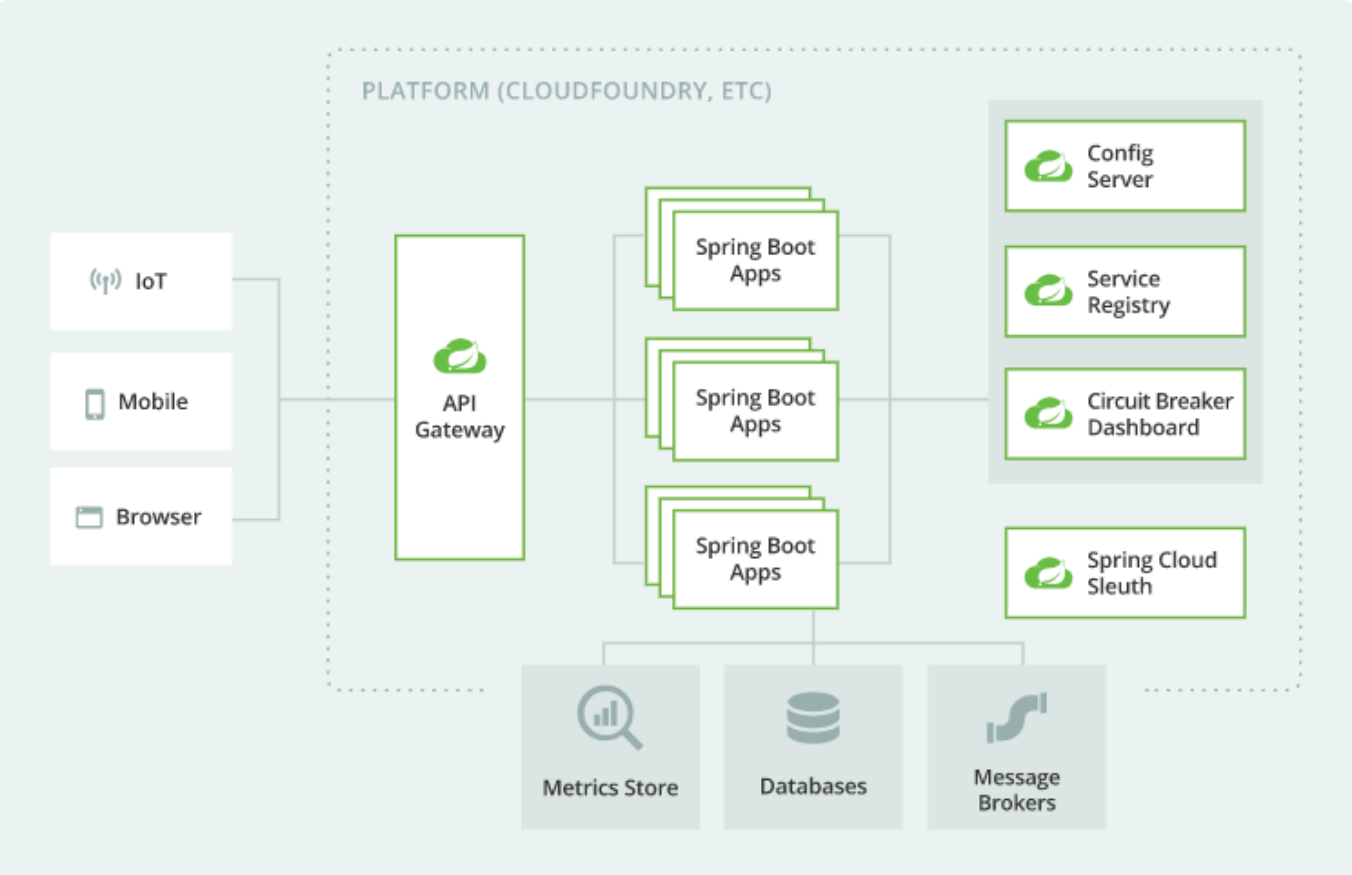
- Timeouts
- Circuit Breaker
- Retry
- Bulkhead

Enfoque del desarrollo de software

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

Steeltoe

Spring Cloud



Steeltoe

Steeltoe es un proyecto de código abierto destinado a desarrollar aplicaciones de microservicio .NET nativas en 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 otros.

Las bibliotecas Steeltoe se crean sobre las API de .NET, siguiendo la especificación de .NET Standard 2.0. Por lo tanto, Steeltoe permite trabajar con .NET Core y .NET Framework 4.x.

https://github.com/SteeltoeOSS/Steeltoe

Steeltoe

Application Configuration
- Spring Config Server
- Cloud Foundry Provider
- Placeholder Provider
- Random Value Provider

Circuit Breakers

Distributed Tracing

Dynamic Logging

Management
- Endpoints (Actuators)

Messaging
- RabbitMQ

Network File Sharing

Observability

Steeltoe

Security

- Store key ring in Redis
- Resource Protection using JSON Web Tokens (JWT)
- CredHub Client
- Cloud Foundry SSO with OAuth2 provider
- Cloud Foundry SSO with OpenID Connect provider

Service Connectors

- MySQL Database
- Microsoft SQL Database
- PostgreSQL Database
- Mongo Database
- Redis Cache
- Apache Geode/GemFire Cache
- RabbitMQ Messaging
- OAuth2 Client

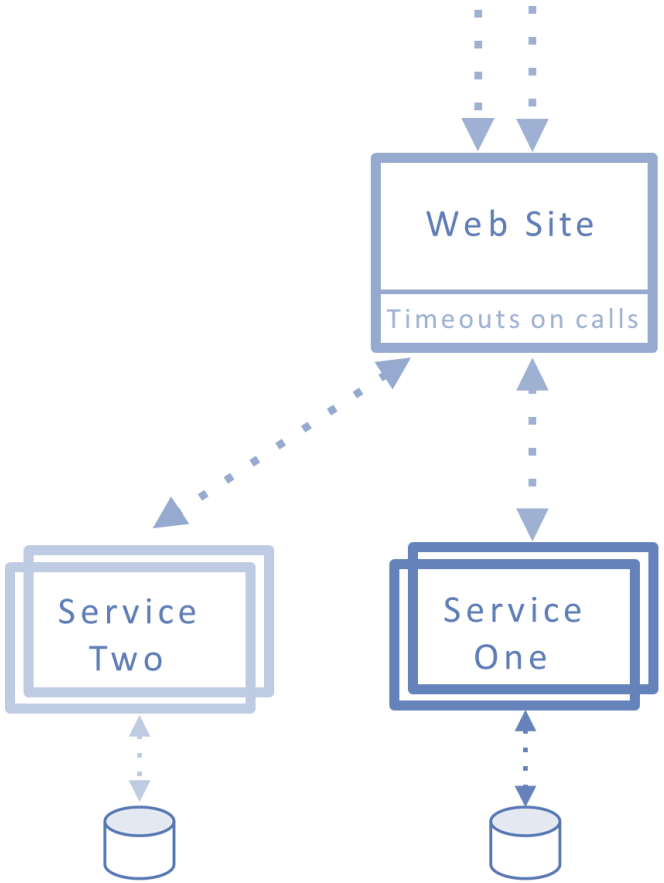
Service Discovery

Hashicorp Consul Registry

Eureka Registry

Patrones para implementación de aplicaciones resilientes: Circuit Breaker, Restry Desig y Bulkheads Design.

Timeouts



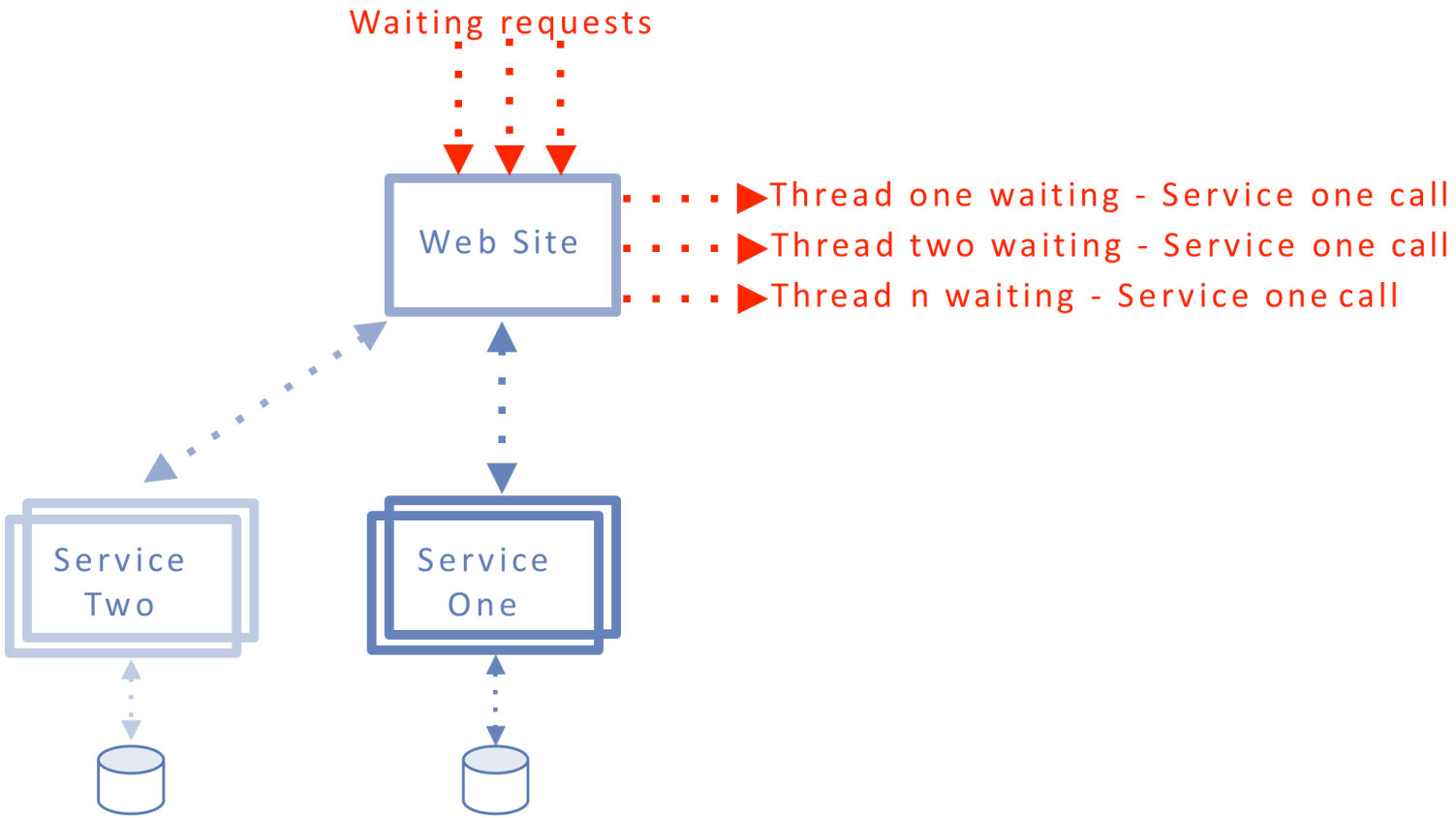
Código de llamada del cliente

- 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

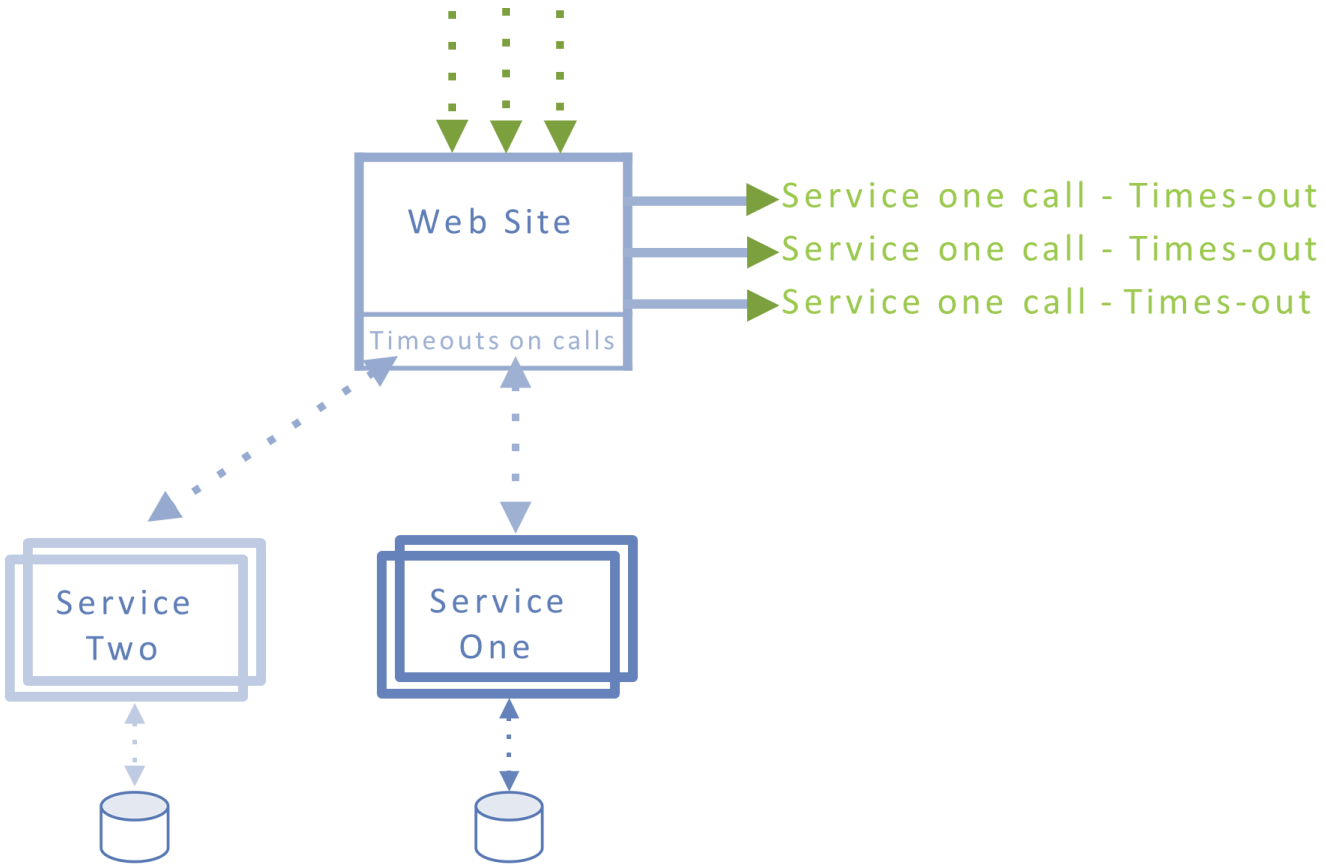
- Evita esperar una respuesta para siempre
- Libera el hilo de llamada
- Le permite manejar fallas de llamadas
- Compatible con casi todas las tecnologías cliente

Degradado de funcionalidad



Patrones : Circuit Breaker, Retry Design y Bulkheads Design.

Timeouts al rescate



Patrones : Circuit Breaker, Retry Design y Bulkheads Design.

Circuit Breaker

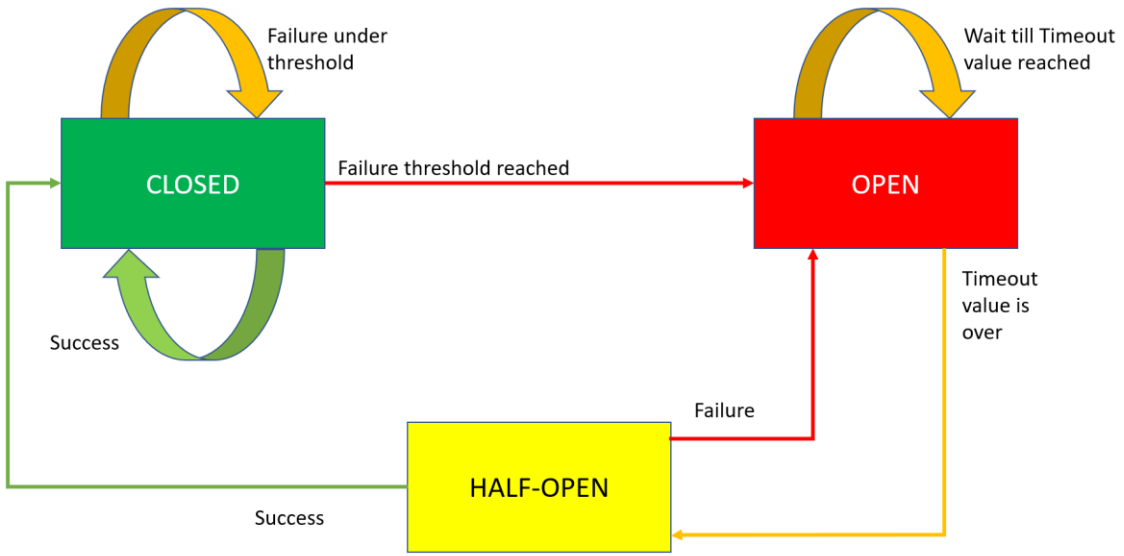
¿Qué es Circuit Breaker?

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.

Patrones : Circuit Breaker, Retry Design y Bulkheads Design.

Circuit Breaker stages



CERRADO

Inicialmente, el disyuntor entra en un estado CERRADO y espera las solicitudes de cliente.

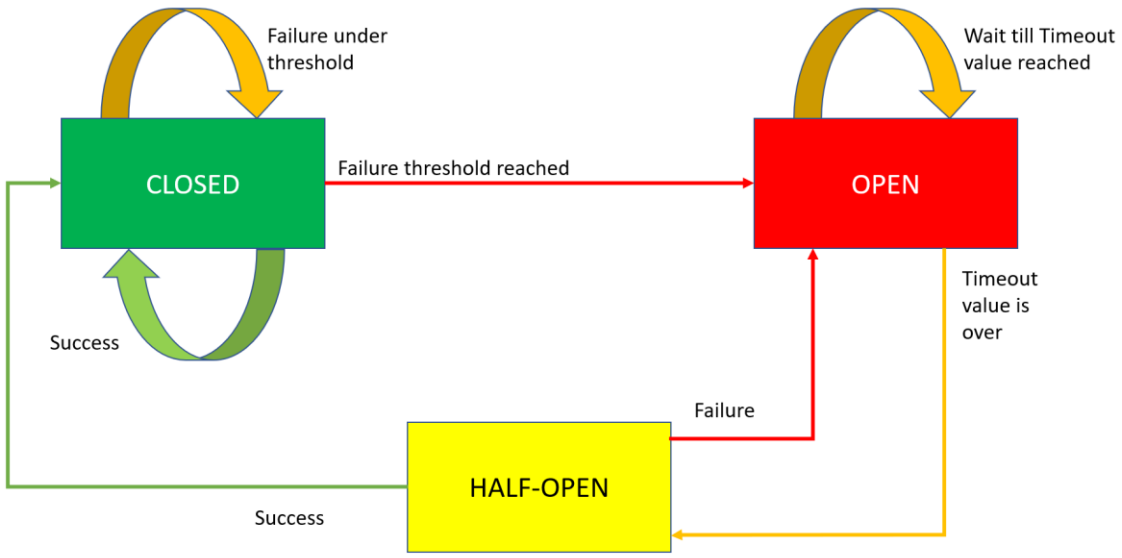
Recibe solicitudes de cliente y realiza una llamada al Servicio

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

Circuit Breaker stages



OPEN

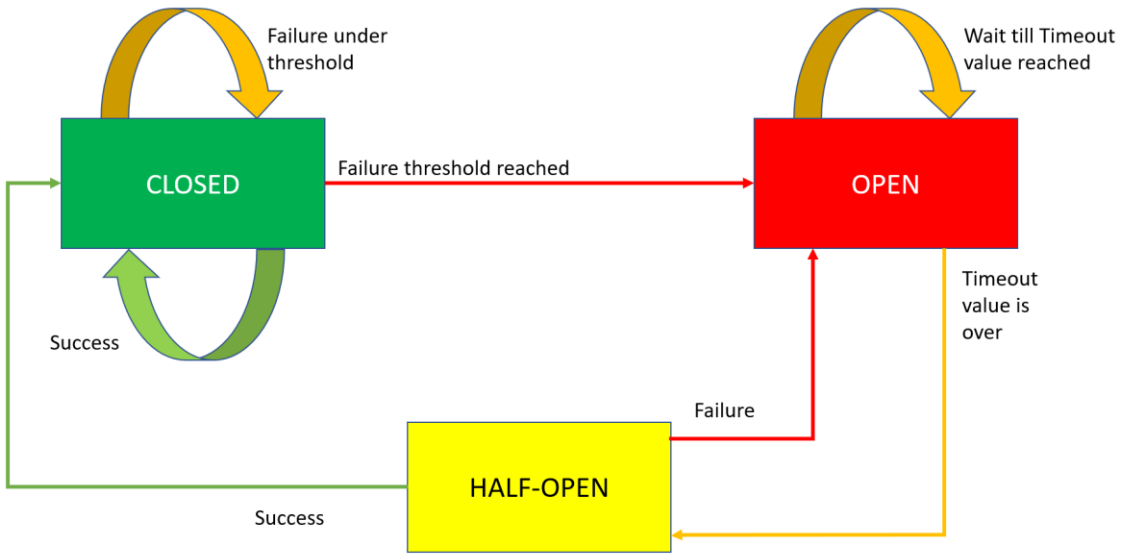
El componente Circuit Breaker entra en este estado cuando el conteo de errores es mayor que el umbral de falla.

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

Circuit Breaker stages



HALF-OPEN

El componente entra en este estado cuando finaliza el tiempo de espera predefinido.

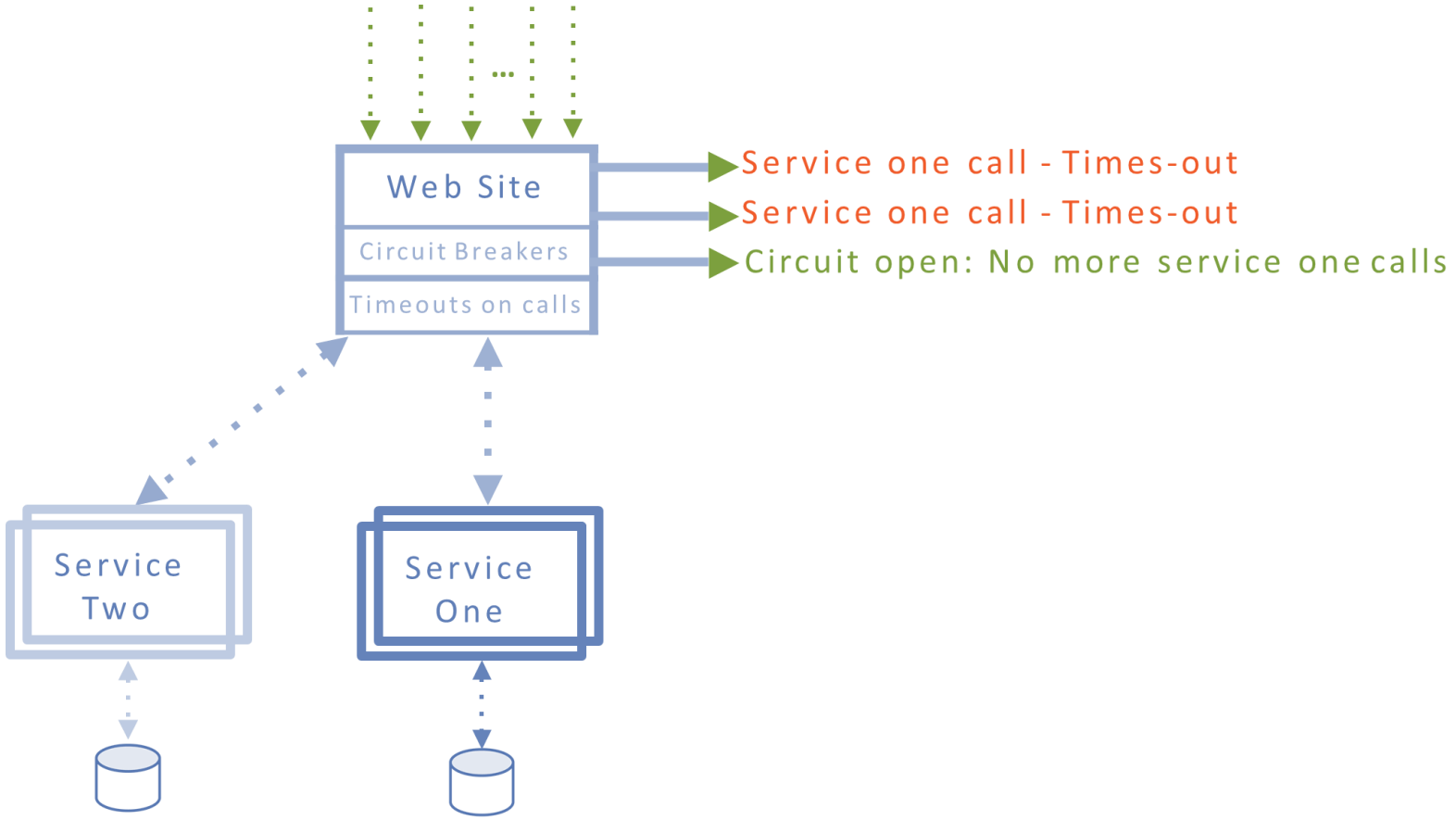
Envía la primera solicitud al Servicio.

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.

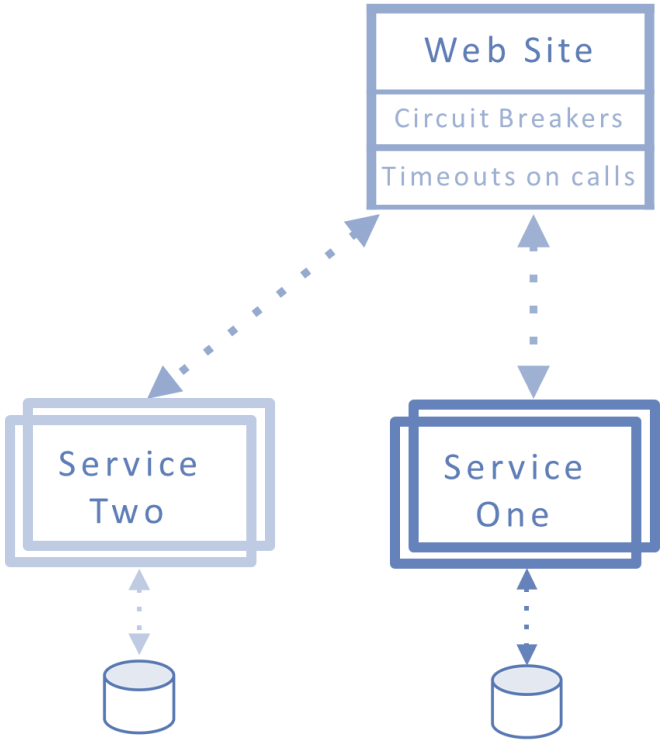
Si recibe una respuesta de error, volverá al estado OPEN y esperará a que expire un valor de tiempo de espera predefinido.

Circuit Breaker, manejo de fallos



Patrones : Circuit Breaker, Retry Design y Bulkheads Design.

Implementando Circuit Breaker



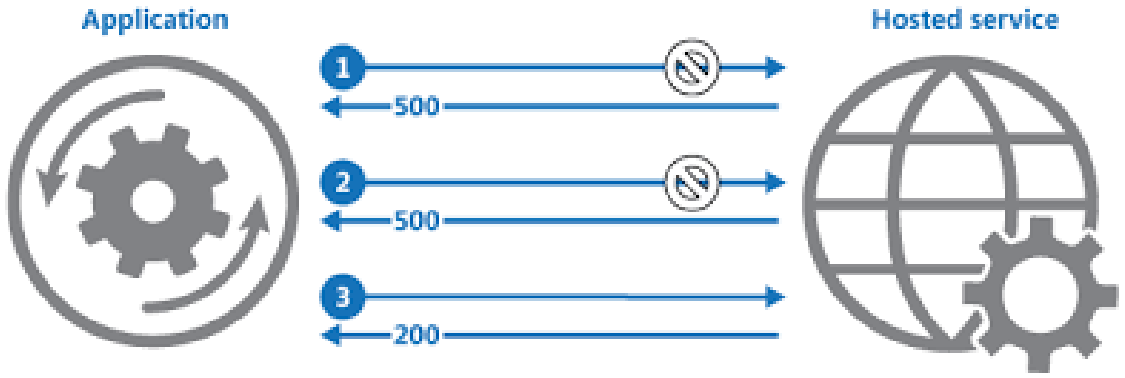
Desarrolla tu propia librería

- Muchos ejemplos de código abierto
- Hacerlo thread safe
- Hacerlo configurable
  - Failure/timeout threshold
  - Duración del estado OPEN

Bibliotecas de terceros

- Thepollyproject.org
- Hystrix
- y muchos más…

Reintentos (Retry)



Ideal para fallas transitorias

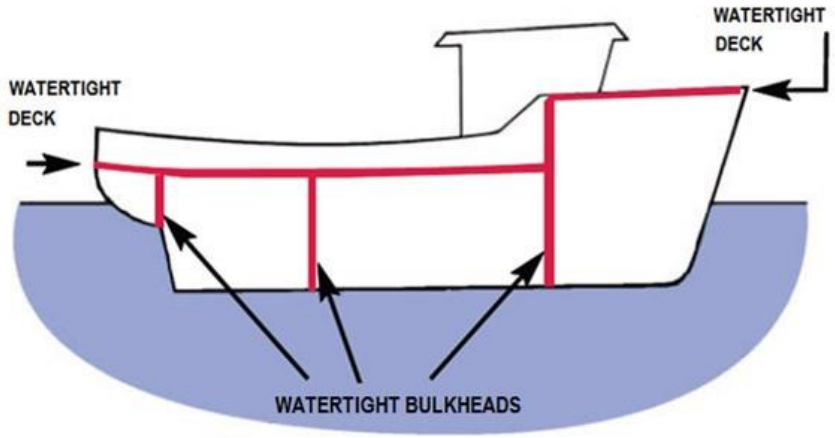
- Pérdida momentánea de conectividad
- Indisponibilidad temporal del servicio
- Tiempos de espera cuando el servicio está ocupado
- El servicio acelera las solicitudes aceptadas
- Errores autocorregidos

Estrategias

- Reintentar inmediatamente
- Reintentar después de una espera
- Cancelar

Registrar y monitorear ocurrencias
Usar junto con disyuntores

Bulkheads



¿Qué son los Bulkheads?

Bulkheads es una forma de particionar una aplicación.

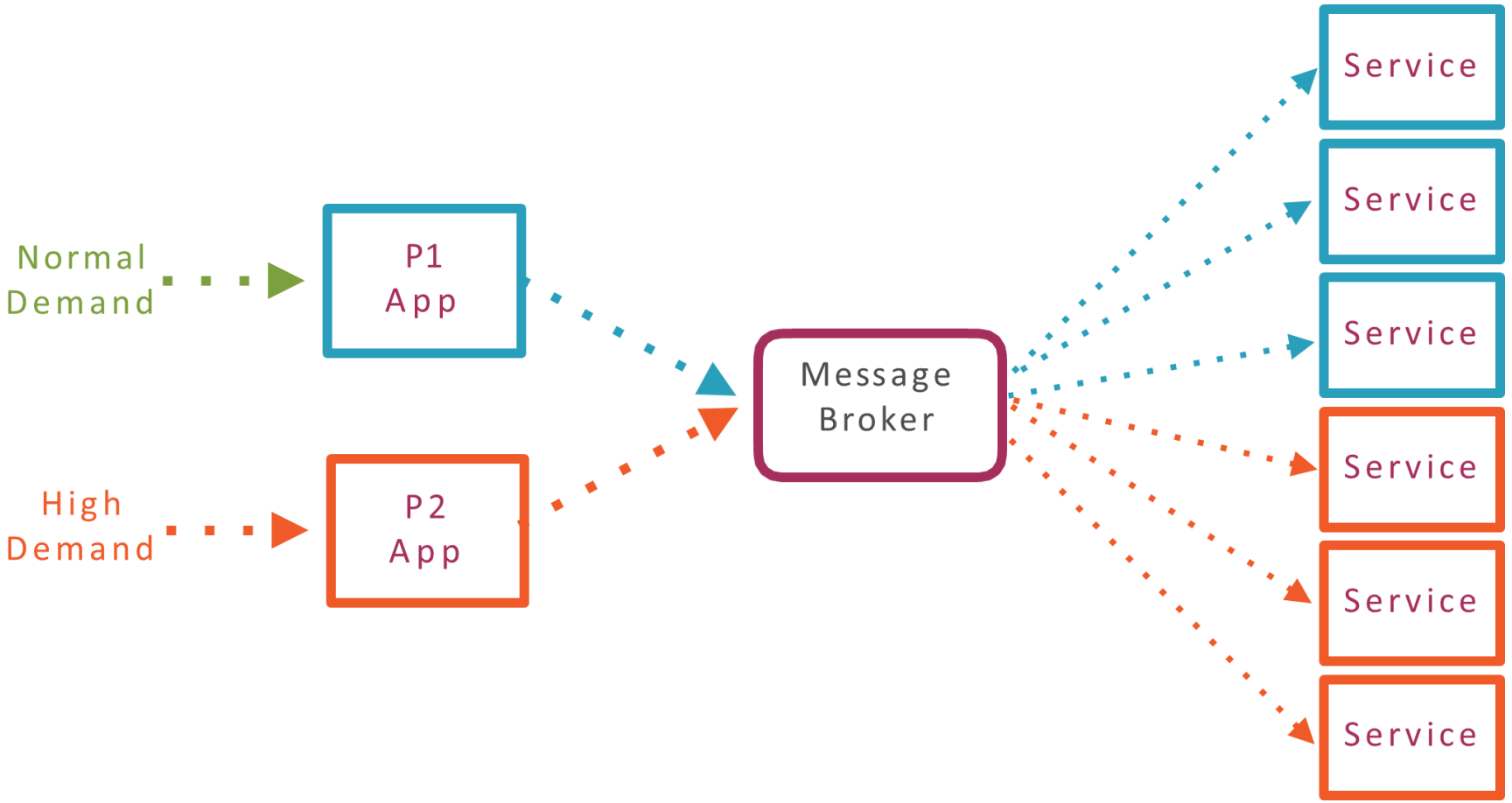
Proporcionan una manera de simultaneidad enlazada y para limitar una serie de acciones simultáneas.

El término Bulkheads se origina en el envío y se refiere a la partición de partes de un barco.

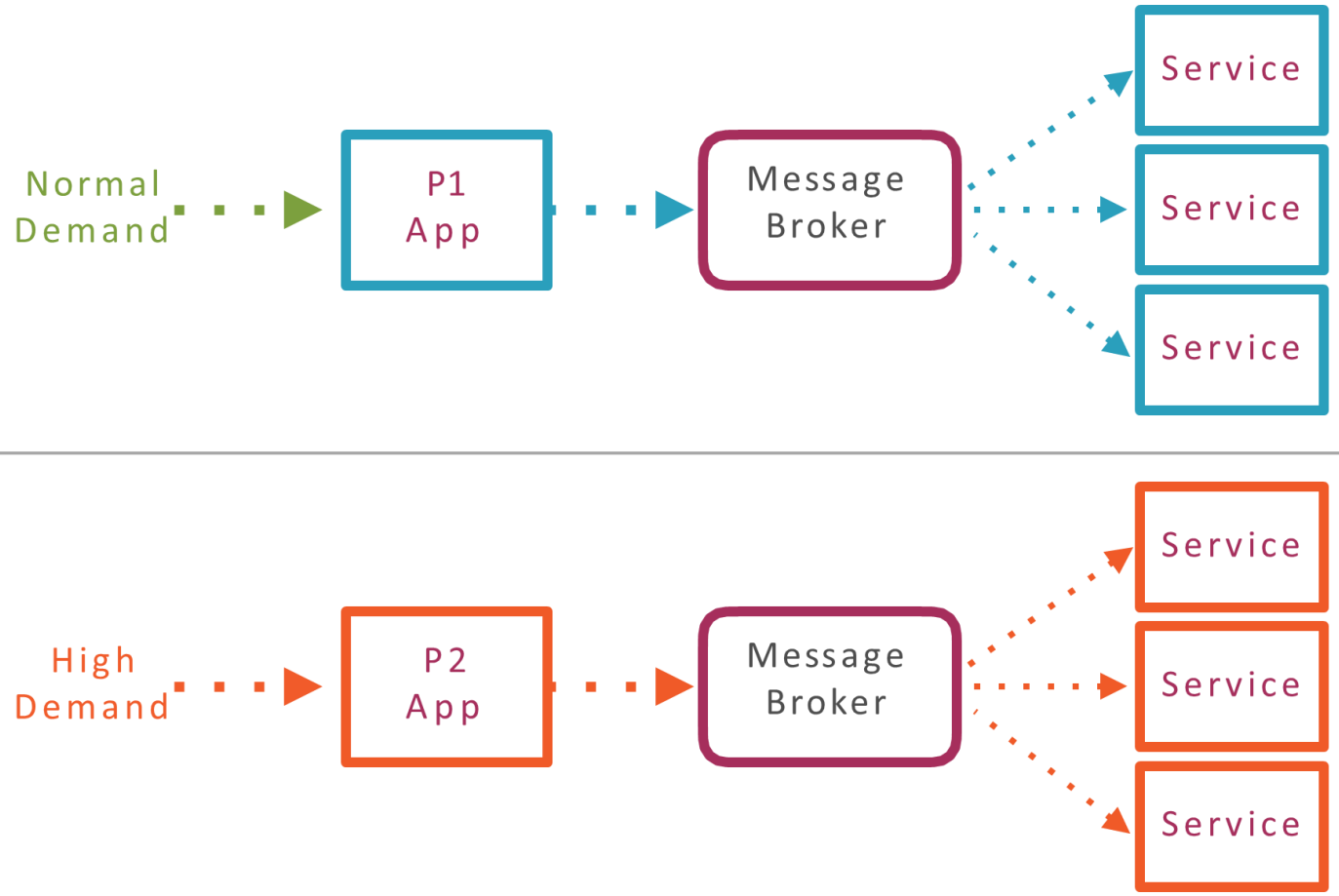
La biblioteca de Polly explica:

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.

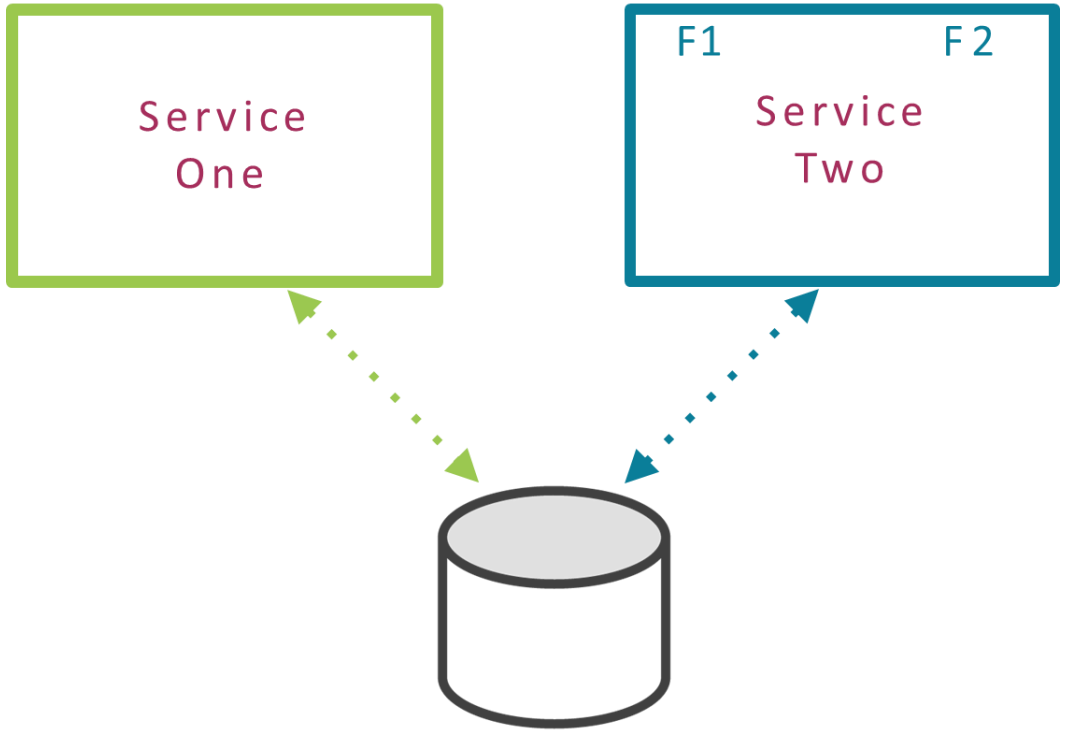
Bulkheads : Separación por criticidad



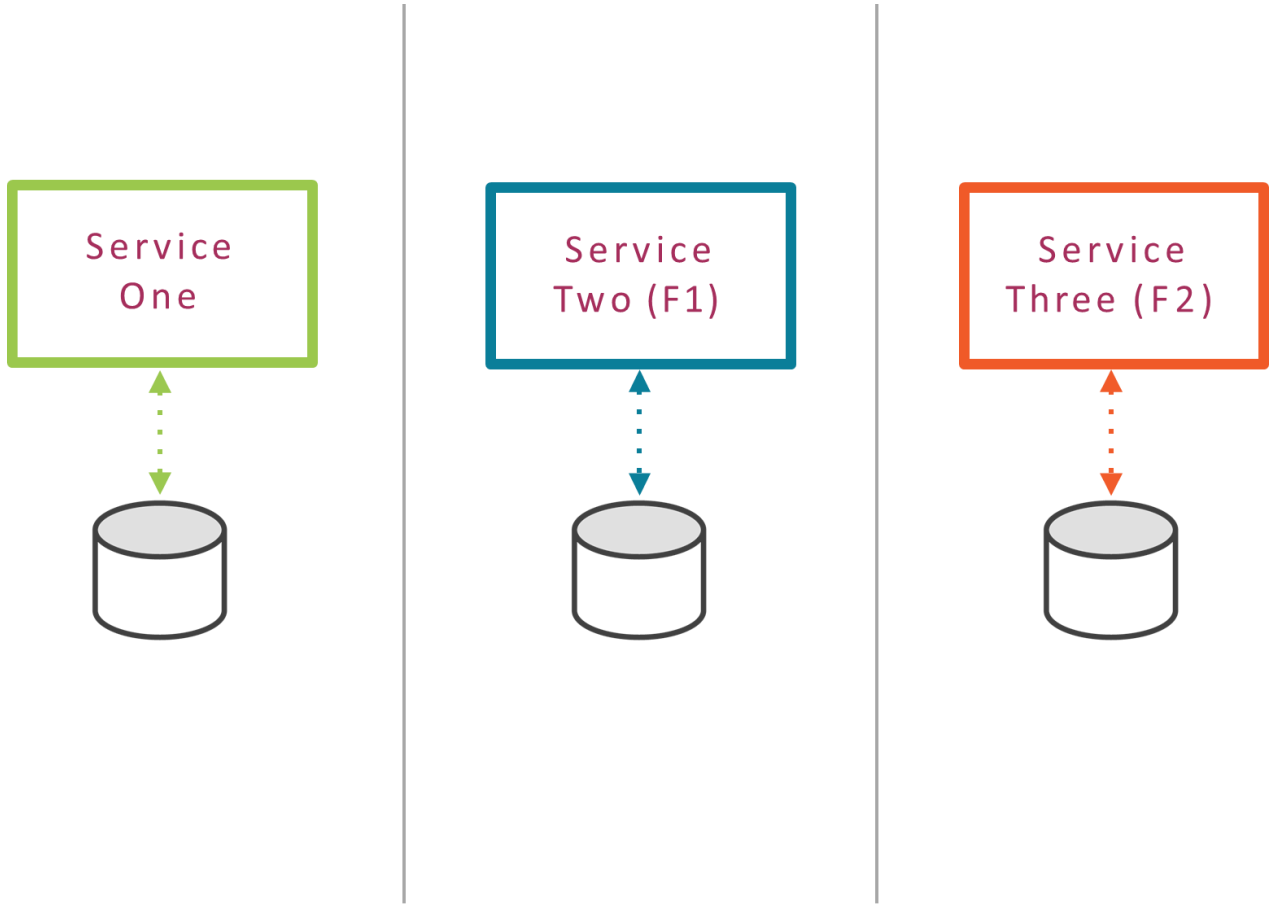
Bulkheads : Separación usando criticidad



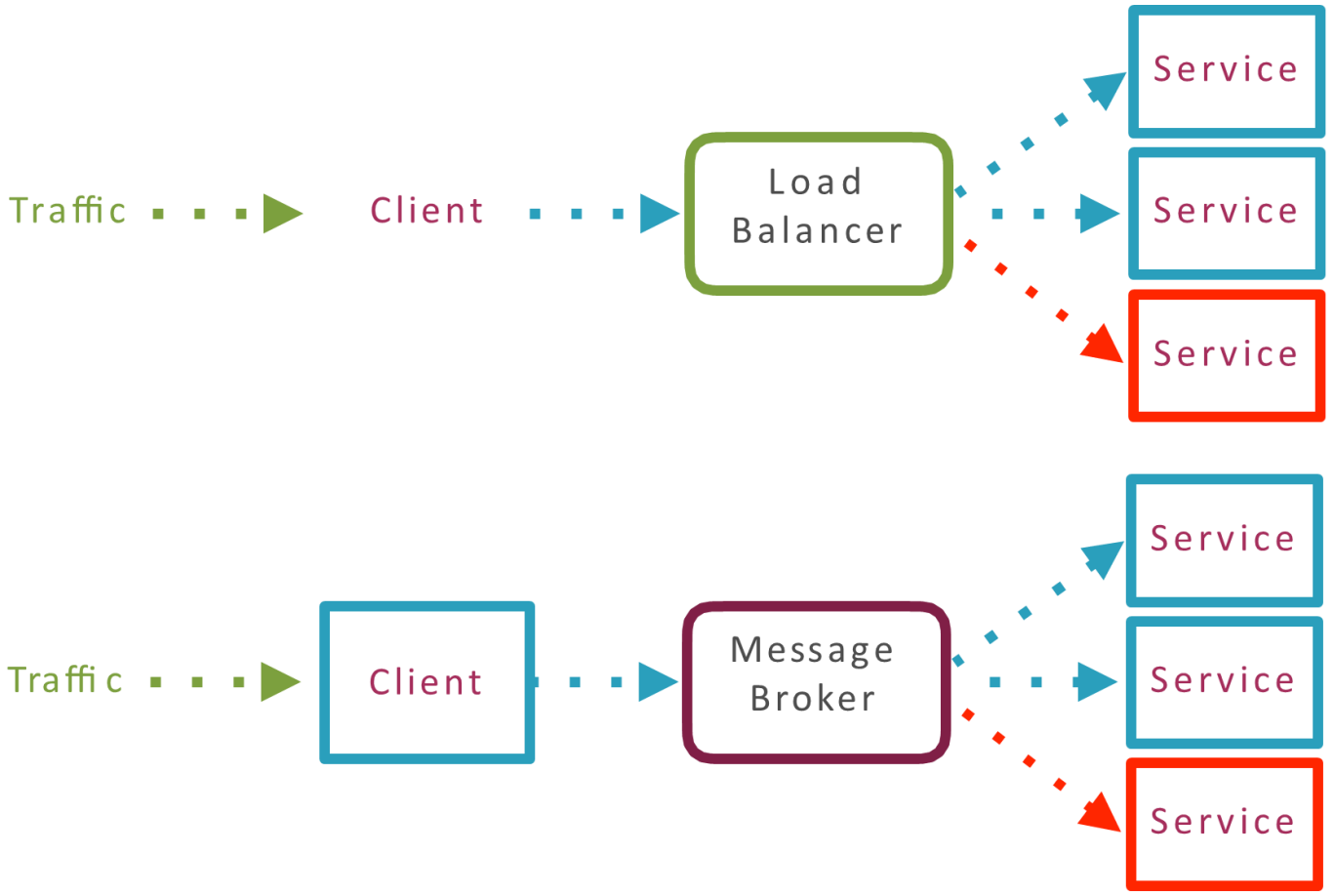
Bulkheads : Aislar microservicios



Bulkheads : Aislar microservicios



Bulkheads : Redundancia



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 subprocesos. Polly tiene como destino .NET 4.0, .NET 4.5 y .NET Standard 1.1.

http://www.thepollyproject.org/

Patrones : Circuit Breaker, Retry Design y Bulkheads Design.

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/

Patrones : Circuit Breaker, Retry Design y Bulkheads Design.

Configuración de servicios mediante configuración distribuida

¿Qué es un servidor de configuración?

¿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

a …

Muchos, muchos archivos de configuración

¿Qué es un servidor de configuración?

Configuración: no distribuido vs distribuido

Herramientas de gestión de configuración al rescate, ¿verdad?

e.g. Chef/Puppet/Ansible

¿Qué es un servidor de configuración?

Funcionará ... pero no es ideal en la nube

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

¿Qué es un servidor de configuración?

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

R: Servidor de configuración

Servidor de Configuración

Servidor de Configuración de Aplicaciones

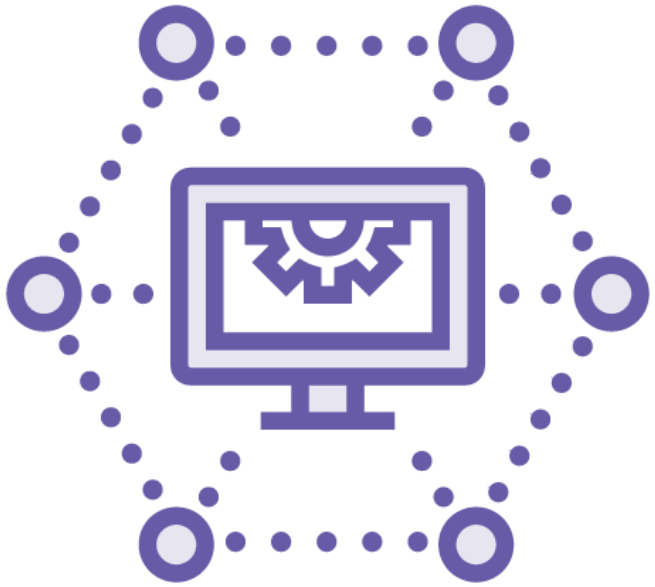
- Almacén de clave / valor centralizado, dinámico y dedicado (puede distribuirse)
- Fuente con acceso de autorización
- Revisión de cuentas
- Versionado
- Soporte de criptografía

¿Qué es un servidor de configuración?

Administrar la configuración de la aplicación con Spring Cloud

¿Qué es un servidor de configuración?

Servidor de Configuración



Administrar la configuración con :

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

¿Qué es un servidor de configuración?

Spring Cloud Config

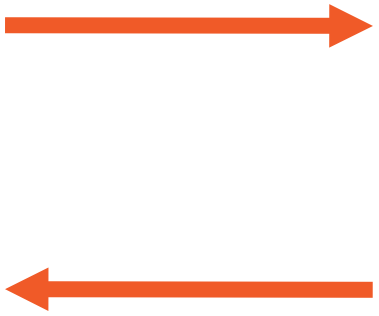
Spring Cloud Config proporciona soporte del servidor y del lado del cliente para la configuración externa en un sistema distribuido.

Documentación de referencia :
https://cloud.spring.io/spring-cloud-config/reference/html/

¿Qué es un servidor de configuración?

Integración con aplicaciones Spring

Config Client



Config Server

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

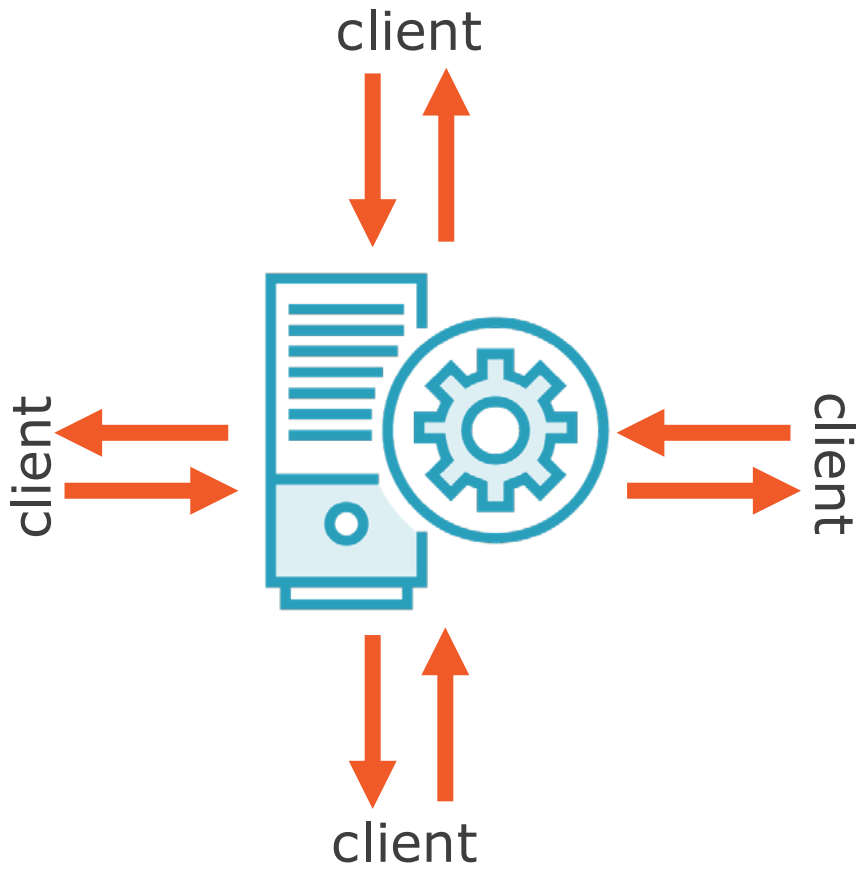
- Standalone (puede embeberse)
- Spring `PropertySource` abstraction
  - e.g. `classpath:file.properties`

¿Qué es un servidor de configuración?

Spring Cloud Config Server

¿Qué es un servidor de configuración?

Servidor de Configuración



Acceso HTTP REST

Formatos de salida

- JSON (default)
- Properties
- YAML

Almacenamiento Backend

- Git (default)
- SVN
- Filesystem

Enviroments de configuración

¿Qué es un servidor de configuración?

Servidor de Configuración

¡No olvides asegurar tu servidor de configuración!

Fácil de configurar Spring Security

¿Qué es un servidor de configuración?

Spring Cloud Config Server: REST Endpoints

¿Qué es un servidor de configuración?

REST Endpoint Parameters

`{application}`	`{profile}`	`{label}`
maps to `spring.application.name` on client	maps to `spring.profiles.active` on client	server side feature to refer to set of config files by name

¿Qué es un servidor de configuración?

REST Endpoint

Endpoint

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

Example

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

¿Qué es un servidor de configuración?

REST Endpoint

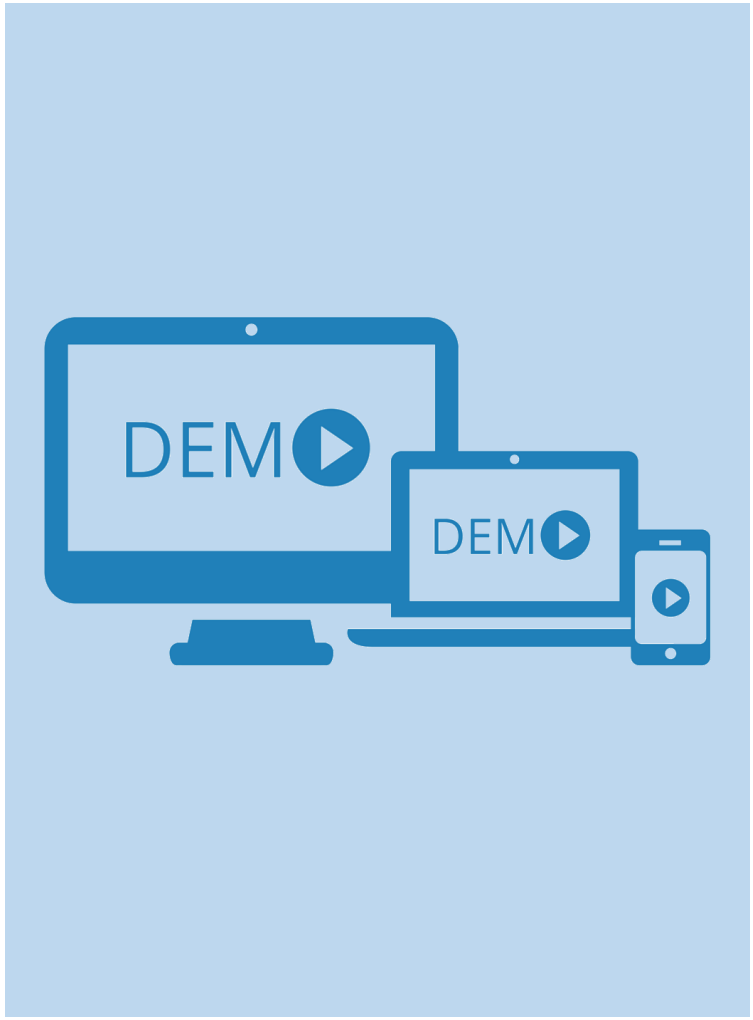
Endpoint

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

Example

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

¿Qué es un servidor de configuración?



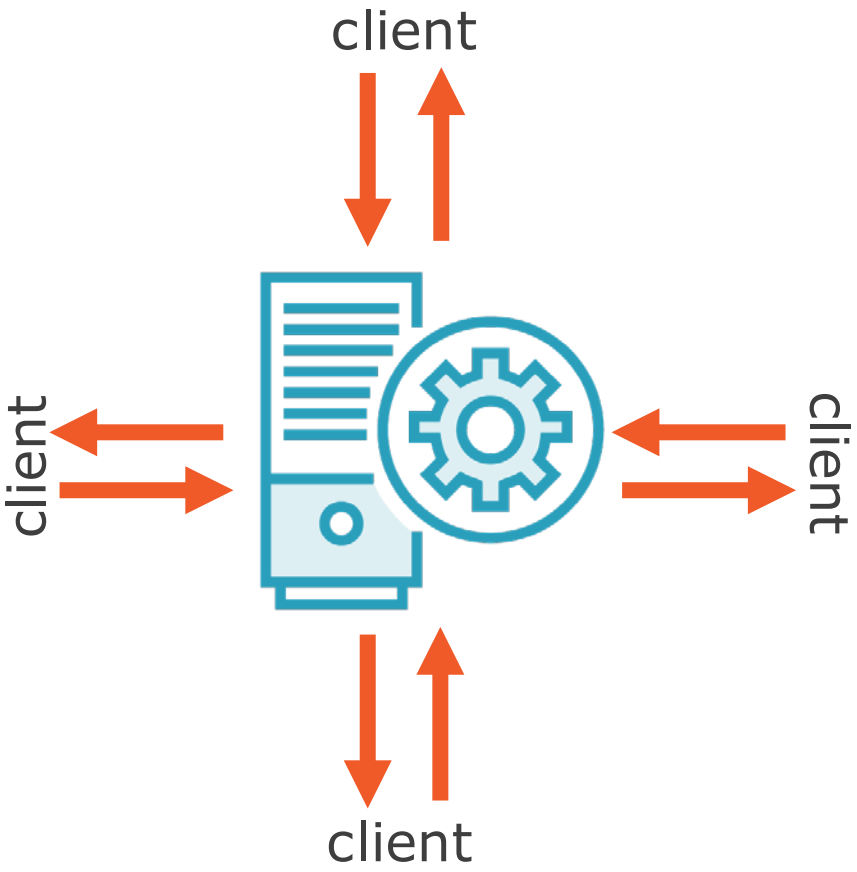
Creando e indiciando un config server

¿Qué es un servidor de configuración?

Spring Cloud Config Client

¿Qué es un servidor de configuración?

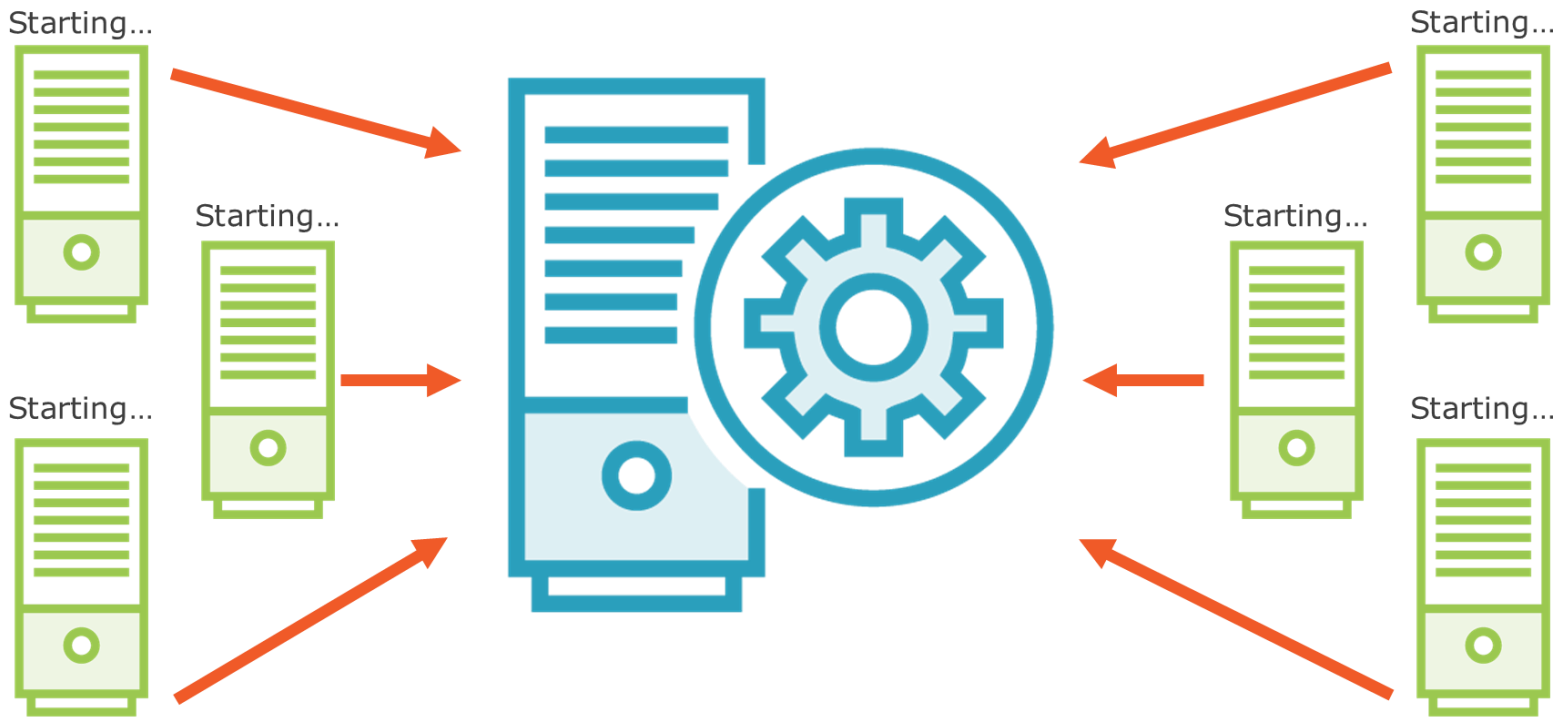
Config Client



Inicialización y carga de condfiguracion de aplicaciones

¿Qué es un servidor de configuración?

Recuperando configuración: inicio de la aplicación

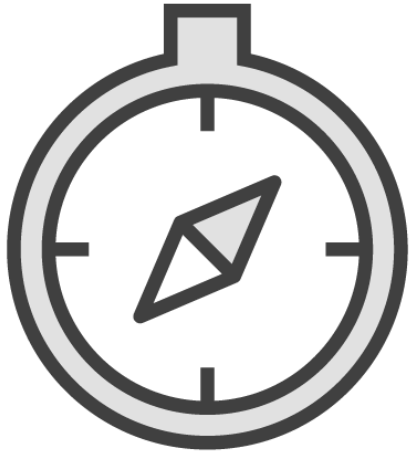


¿Qué es un servidor de configuración?

Bootstrapping con bootstrap.properties or bootstrap.yml

Config first

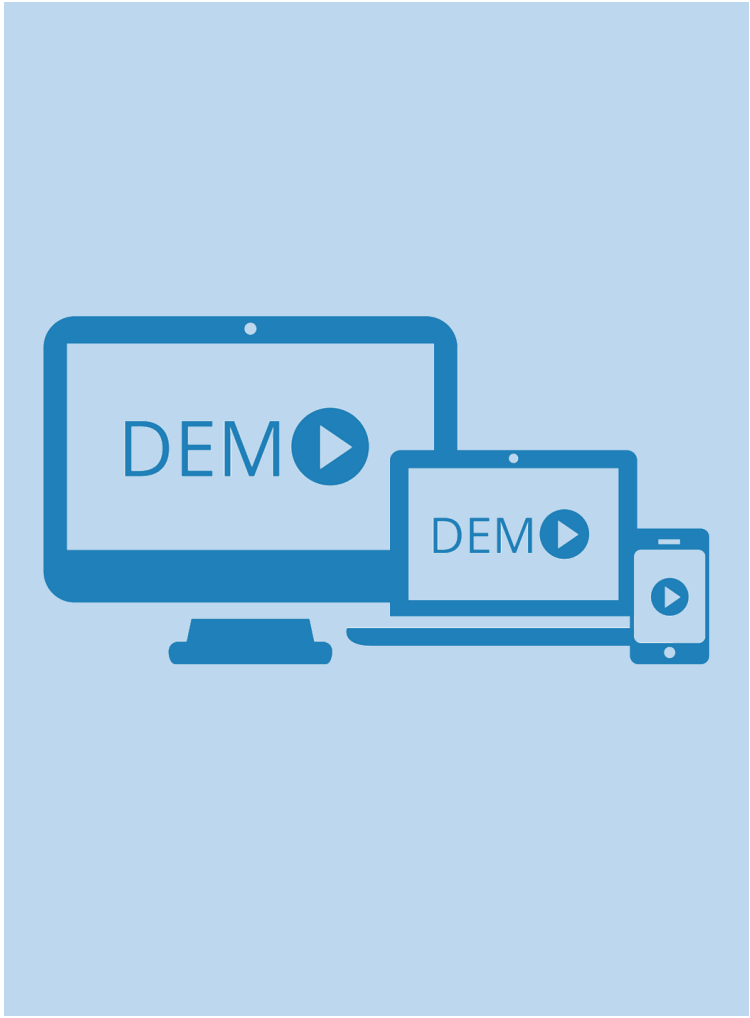
Especifique la ubicación del servidor de configuración



Discovery first

Descubre la ubicación del servidor de configuración

¿Qué es un servidor de configuración?



Inicializacion de un servicio que usa config client

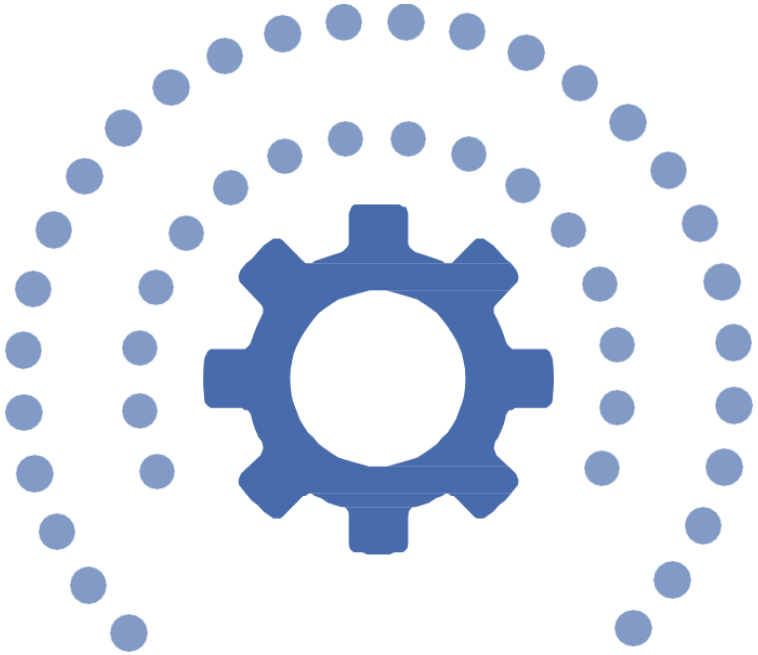
Actualización de la configuración en tiempo de ejecución

¿Qué es un servidor de configuración?

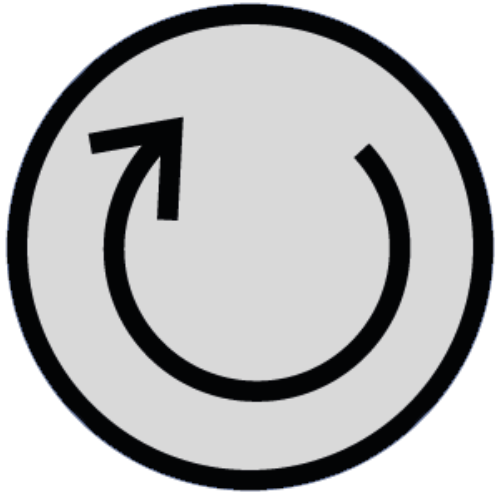
Actualización de Config Client

Refresh

@ConfigurationProperties



Notifique las aplicaciones para actualizar la configuración

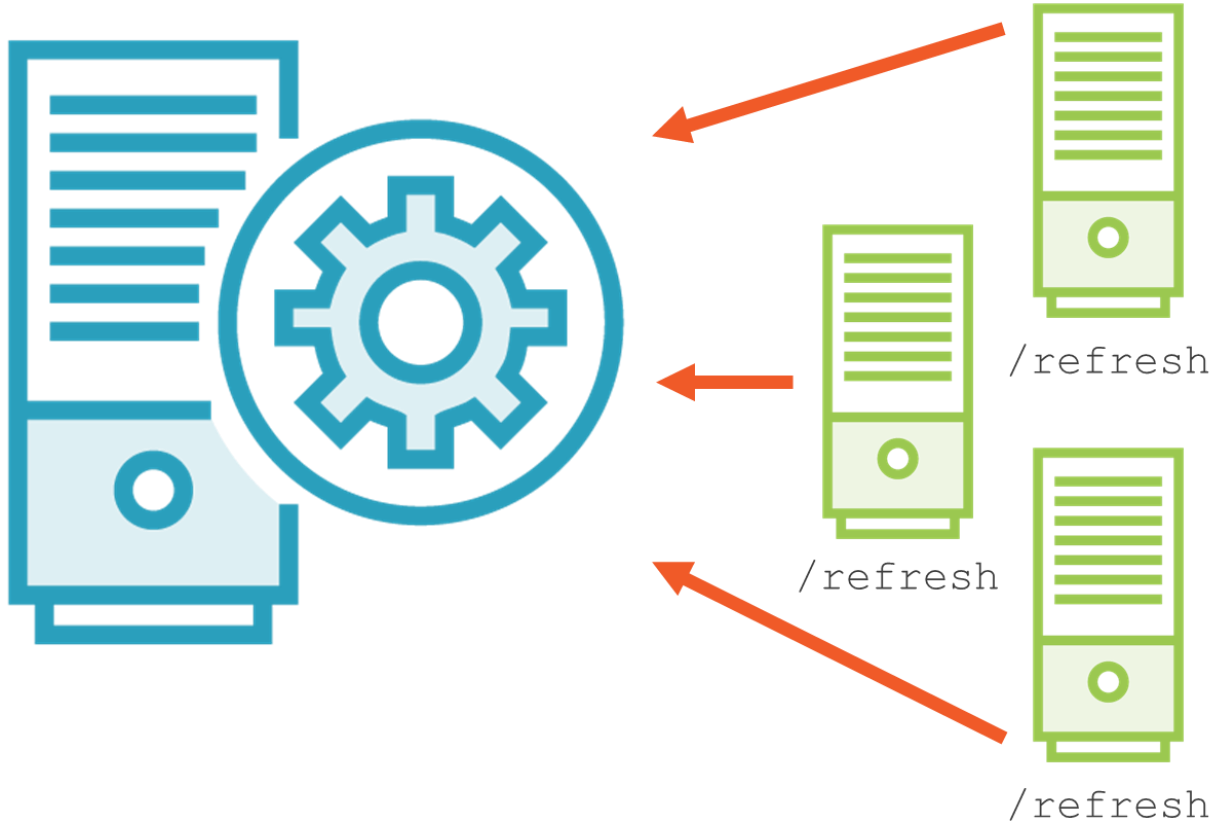


`/refresh`
with
`spring-boot-actuator`

Obteniendo configuración: actualización explícita

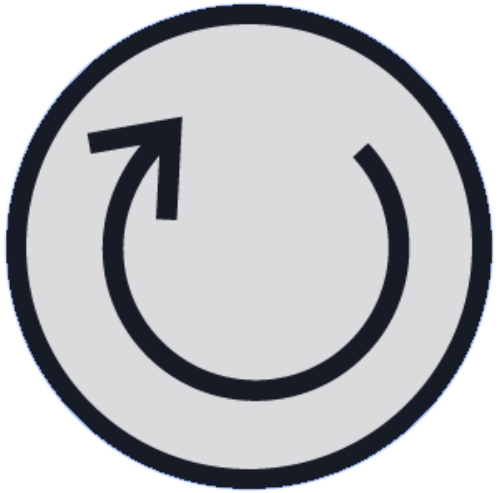
Manual

via each `/refresh` endpoint

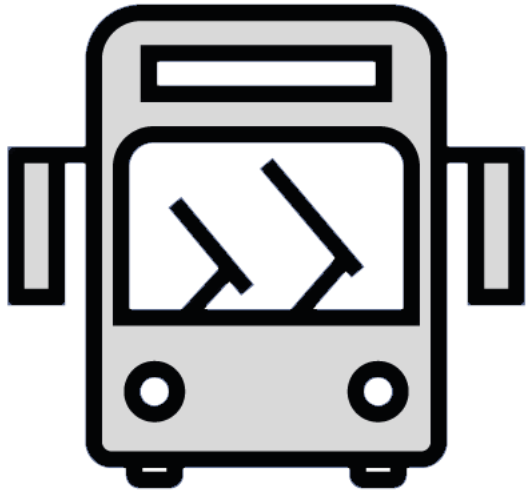


Construyendo un Config Server personalizado

Notifique las aplicaciones para actualizar la configuración

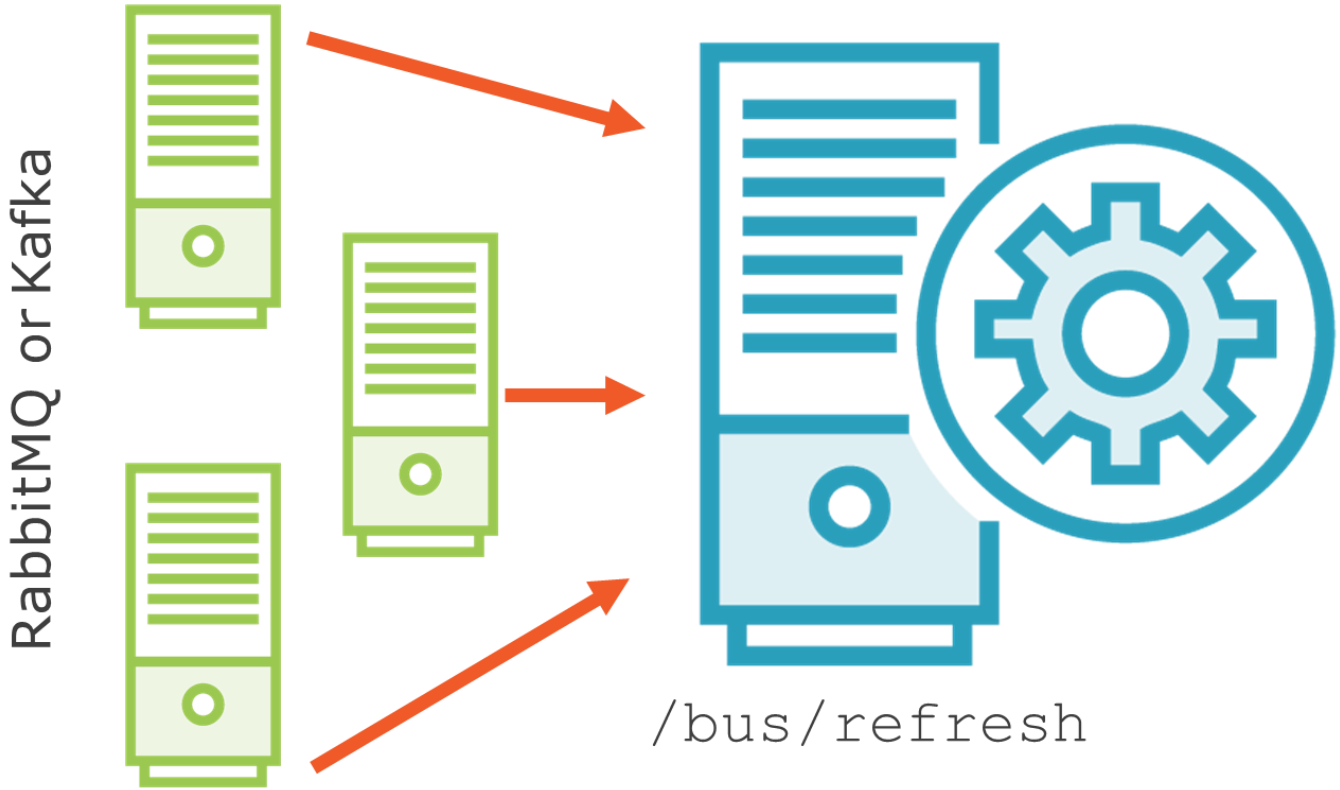


`/refresh`
with
`spring-boot-actuator`



`/bus/refresh`
with
`spring-cloud-bus`

Recuperación de la configuración: actualización dinámica pro push

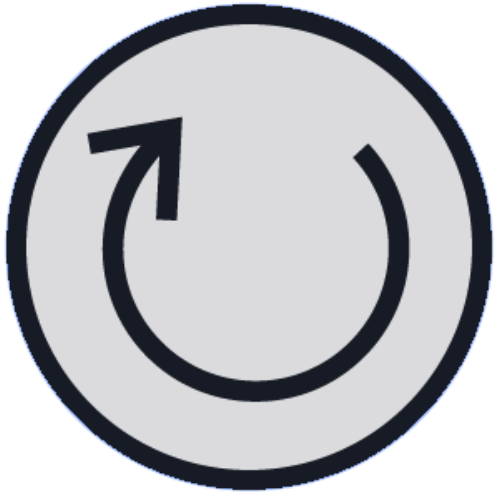


Automatic

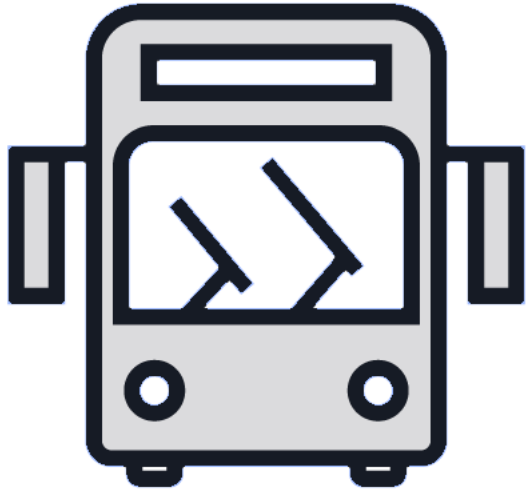
via Spring Cloud Bus broadcasting

Construyendo un Config Server personalizado

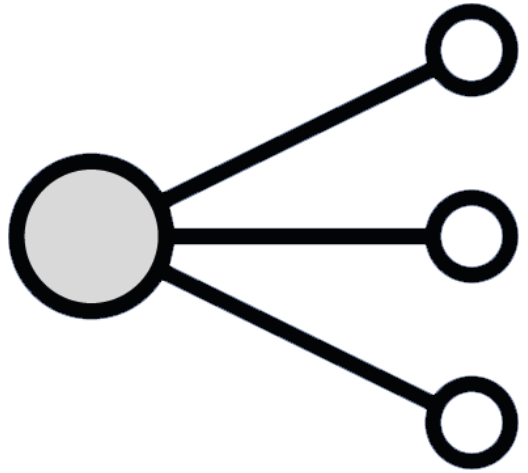
Notifique las aplicaciones para actualizar la configuración



`/refresh`
with
`spring-boot-actuator`



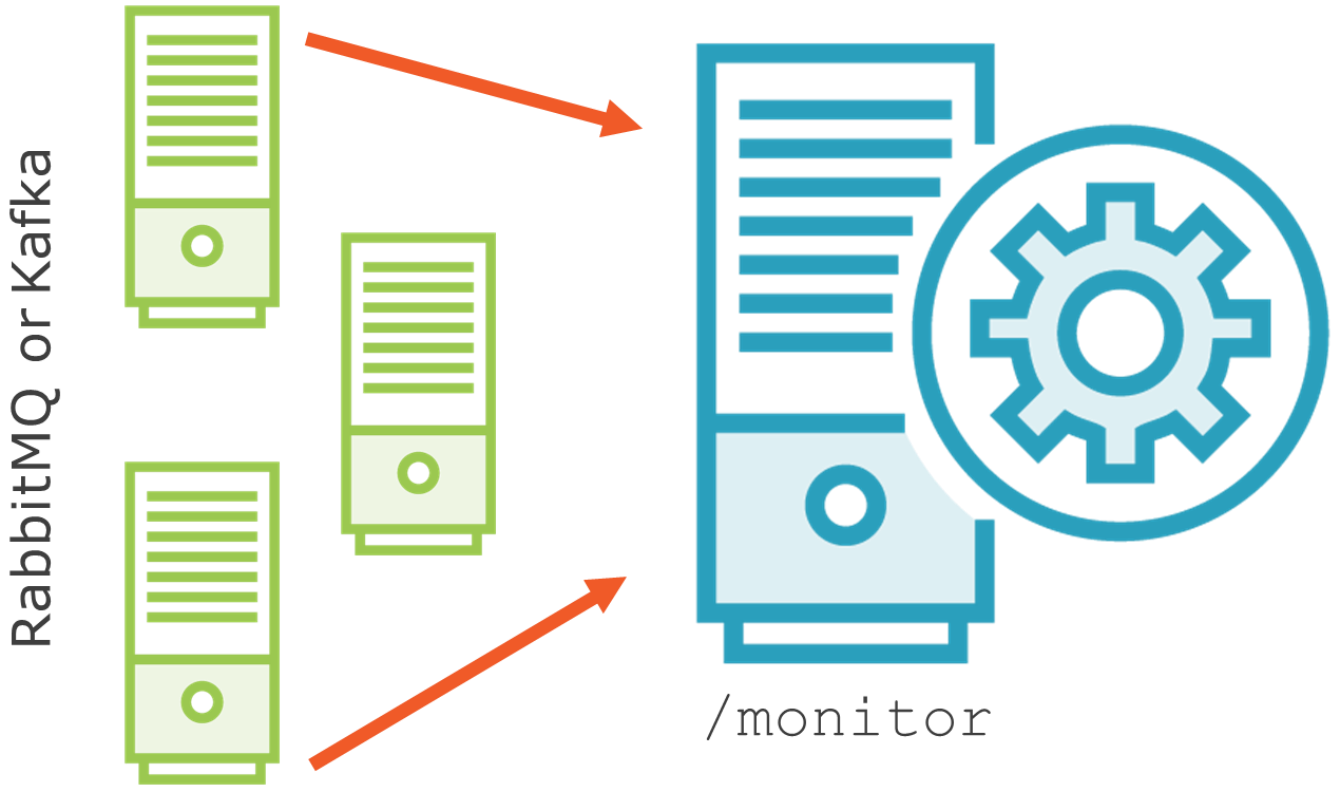
`/bus/refresh`
with
`spring-cloud-bus`



VCS + `/monitor`
with
`spring-cloud-config-monitor` &
`spring-cloud-bus`

Construyendo un Config Server personalizado

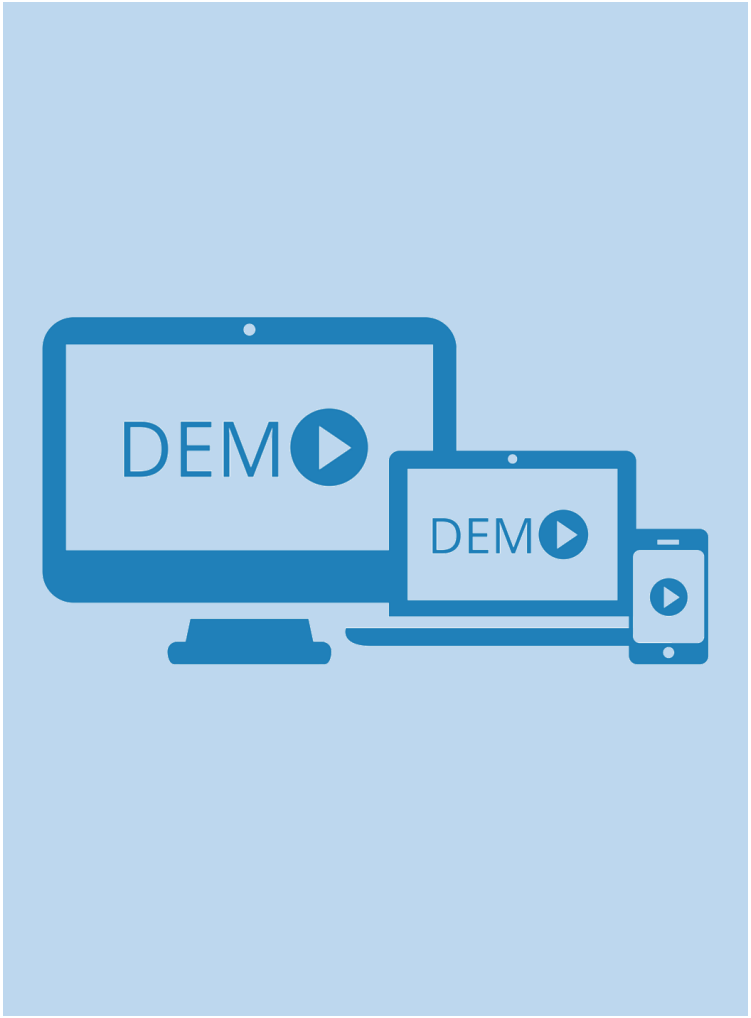
Recuperando configuración: Actualización inteligente



Automatic
& Smart

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

Construyendo un Config Server personalizado

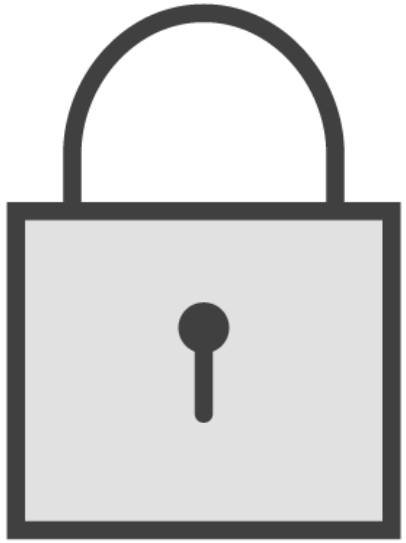


Actualización de configuracion al vuelo

Encriptamiento y desencriptamiento de la configuración

¿Qué características son compatibles?

Configuración encriptada en reposo y / o en vuelo



Endpoint de encriptamiento de configuración

Endpoint de desencriptamiento de configuración

Cifrado y descifrado con claves simétricas o asimétricas.

Construyendo un Config Server personalizado

GRACIAS

POR SU PREFERENCIA

GALAXY
TRAINING