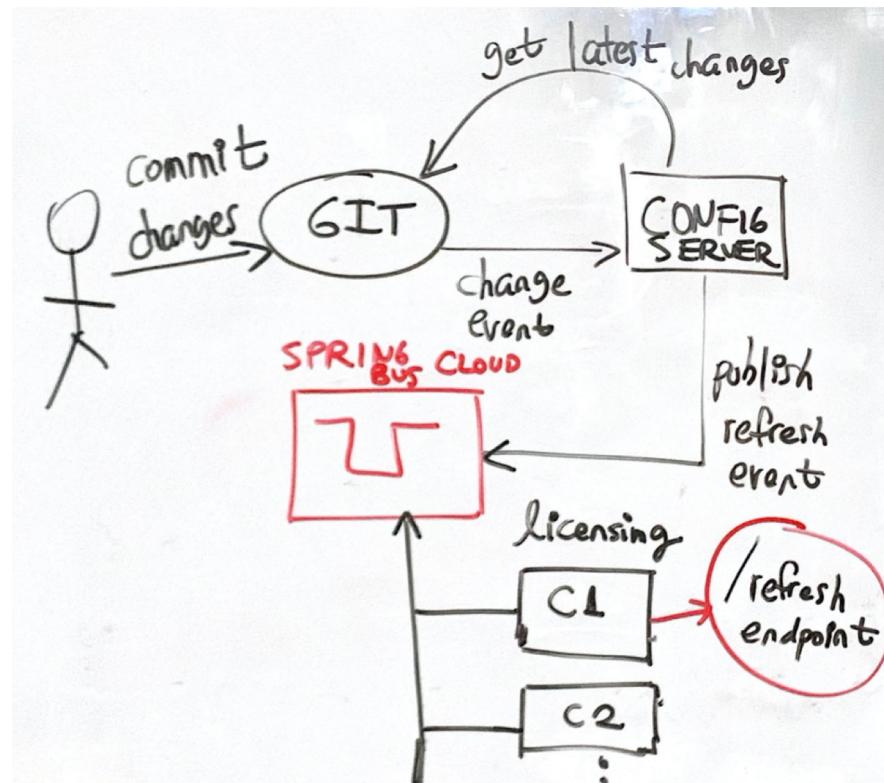


Arquitectura e Implementación de Microservicios

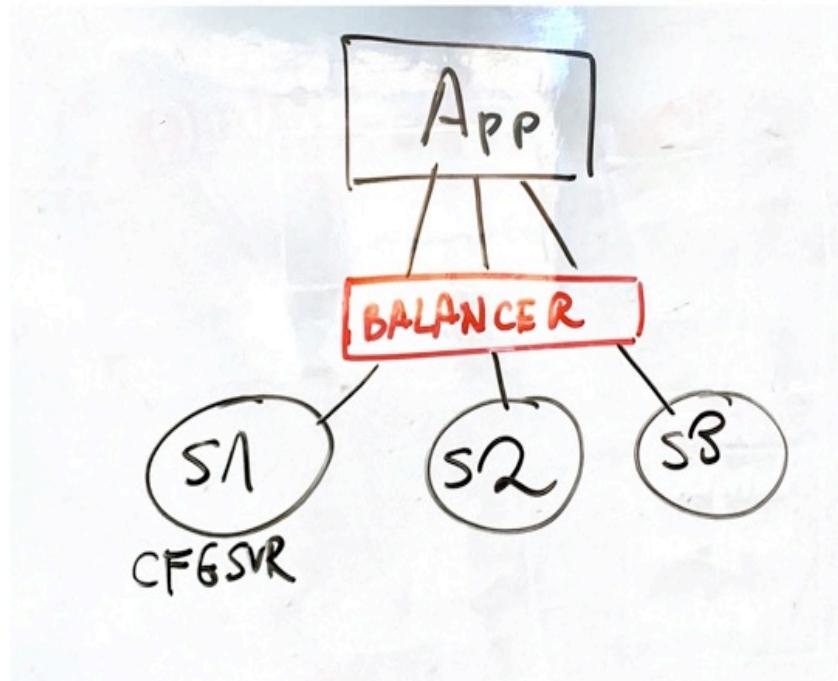
Sesión 3

Spring Cloud Bus

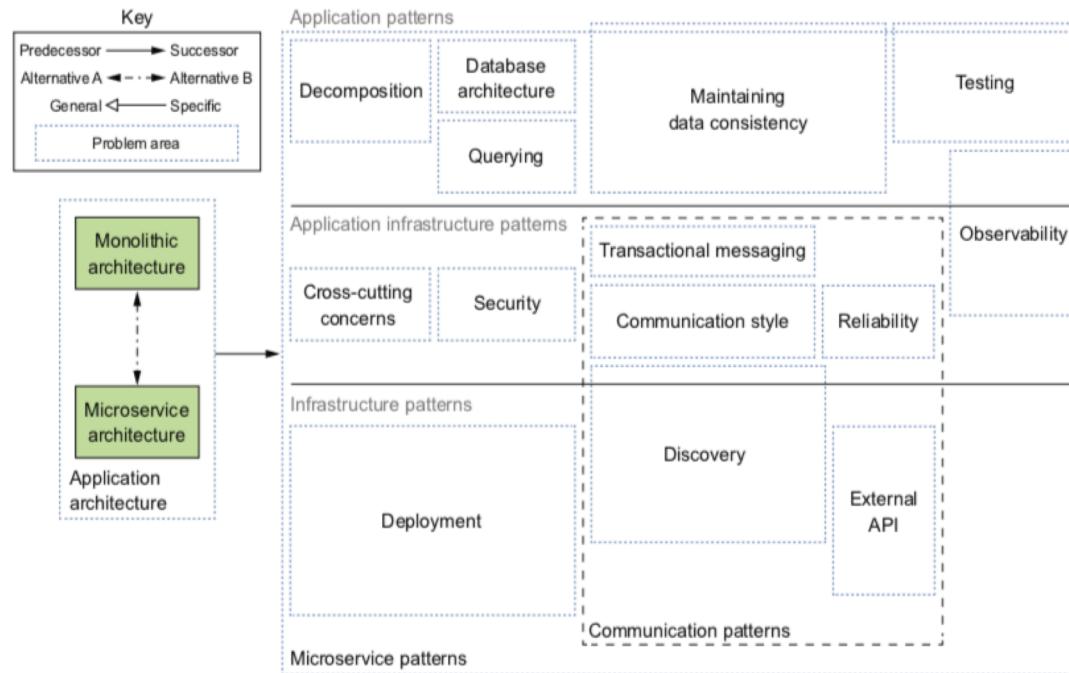


Service Discovery

Alta disponibilidad de config server



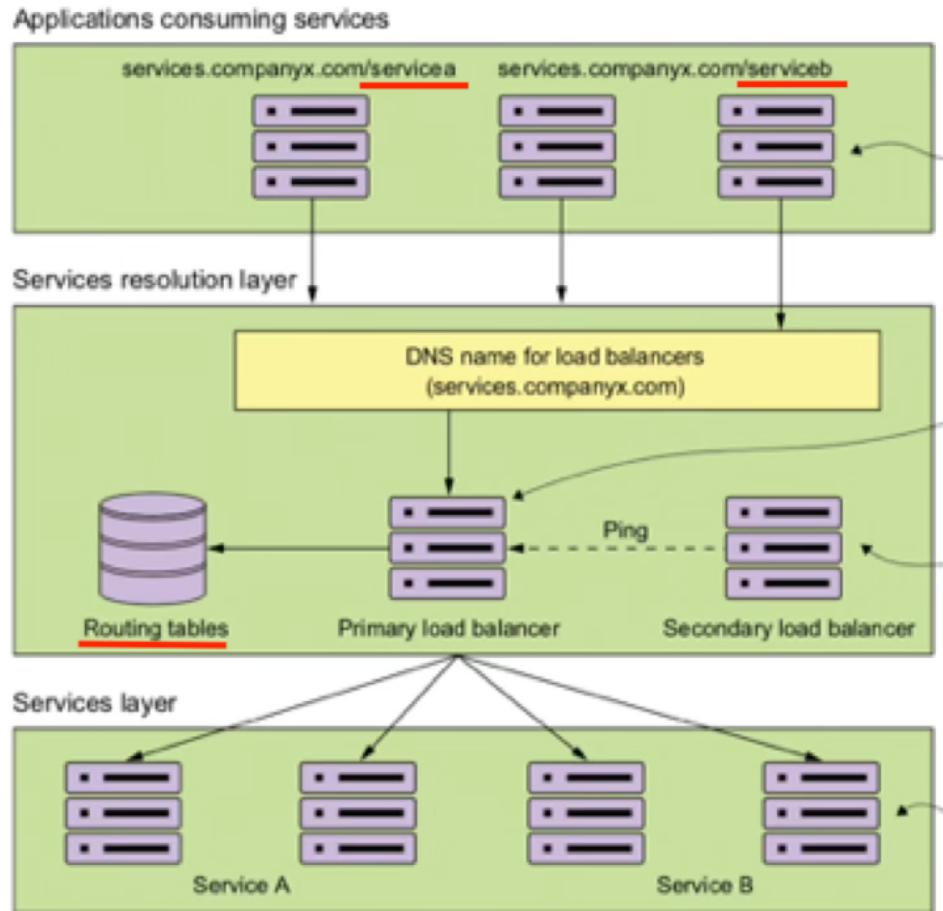
Patrones de diseño de microservicios



Service Discovery

- Necesaria dirección física
 - Formalmente: service Discovery
 - Simple: Propiedades con direcciones remotas
- Critico microservicios:
 - Escalar arriba y abajo instancias
 - Ocultar ubicación física
 - Servicios añadidos/removidos pool servicios
 - Escalamiento:
 - Monolitos, agrandando hardware: vertical
 - Mejor, agregando servidores: horizontal
 - Microservicios = escalamiento
 - Service Discovery oculta deployments
 - Incrementa Resilencia
 - Si Servicio no saludable, instancia removida

Solucion tradicional

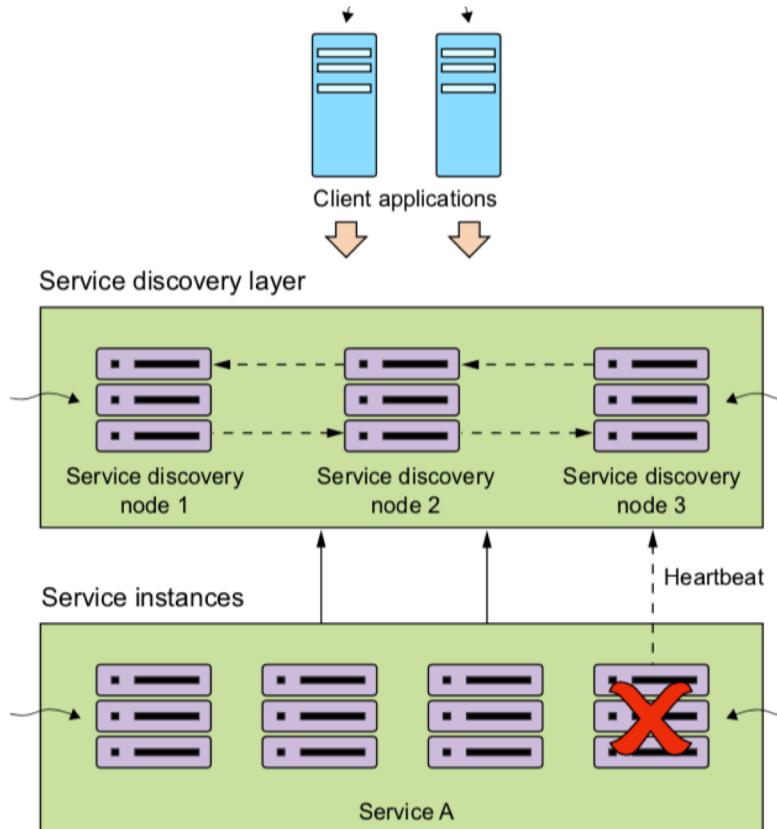


- No cloud: dns y balanceador de carga
- Instancias: app servers
 - Estatico
 - persistente
- Trabaja Bien On premise
 - Mal en cloud

En Microservicios

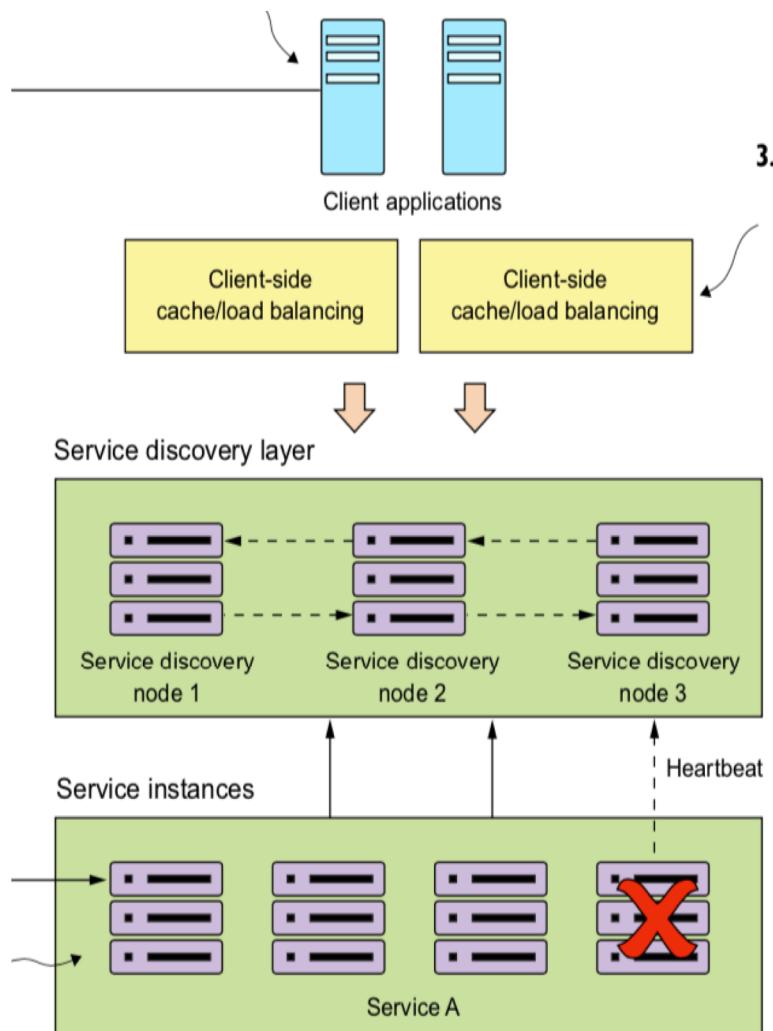
- Service Discovery debe ser:
 - Altamente disponible
 - Otros Nodos deben operar
 - balancear la carga
 - A todas las instancias
 - Resiliente
 - Informacion de servicios en cache
 - Si service Discovery cae, se usa cache
 - Tolerante fallos
 - instancias no saludable=remover

arquitectura



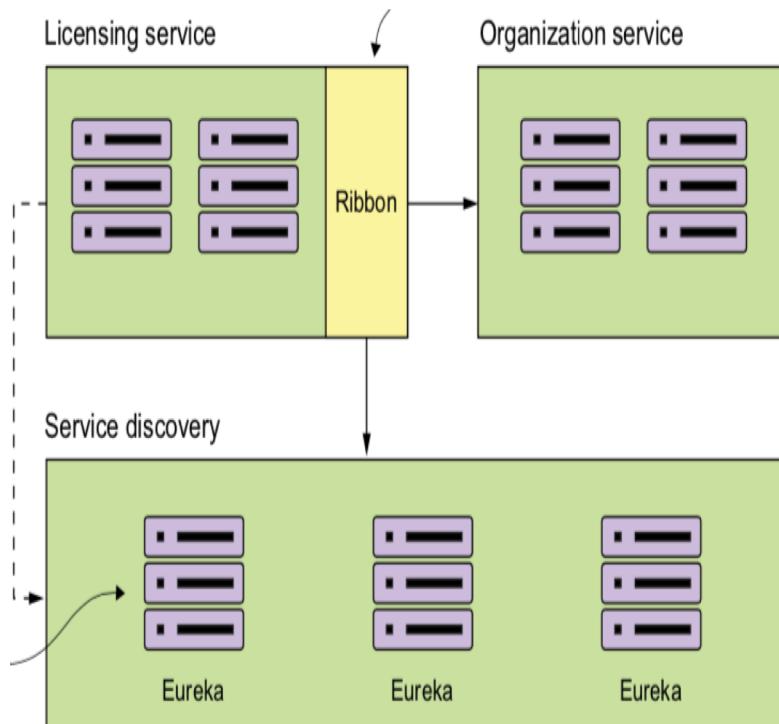
- En inicio registrar ip y puerto
 - Mismo serviceid
 - Serviceid identifica servicio
- Si no heartbeat , removido del pool
- Servicio registrado = listo
- Service Discovery en cada llamada
 - Fragil , dependiente del motor

Agregar Balanceo carga lado cliente



- Cliente lee todas las instancias
 - Cache local
- en cada llamada, lee cache
 - Algoritmo balanceo round robin
- Periodicamente Refresca cache
- Hay Riesgo llamar instancia no saludable
 - Cache local invalidado y refrescado

Service Discovery con eureka



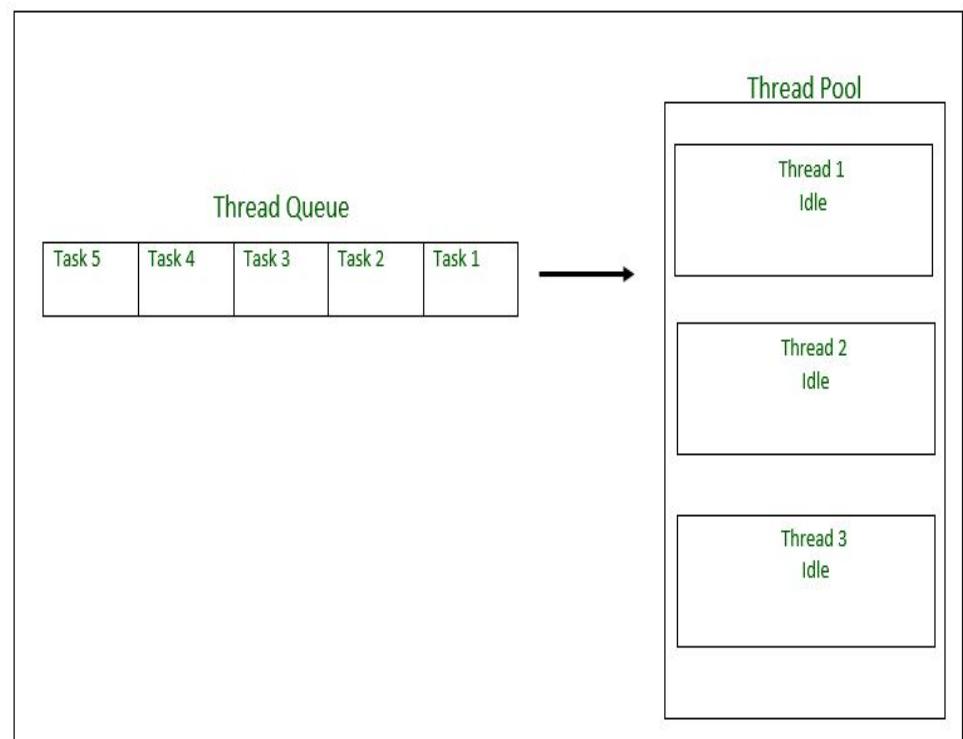
- Spring cloud + Netflix eureka
- Netflix ribbon
- Service Registry: Informacion de localización de instancias
- Ribbon lee instancias desde Eureka
 - Cachea localmente
 - Periodicamente Refresca cache local

Laboratorio

Patrones de resiliencia con Netflix Hystrix

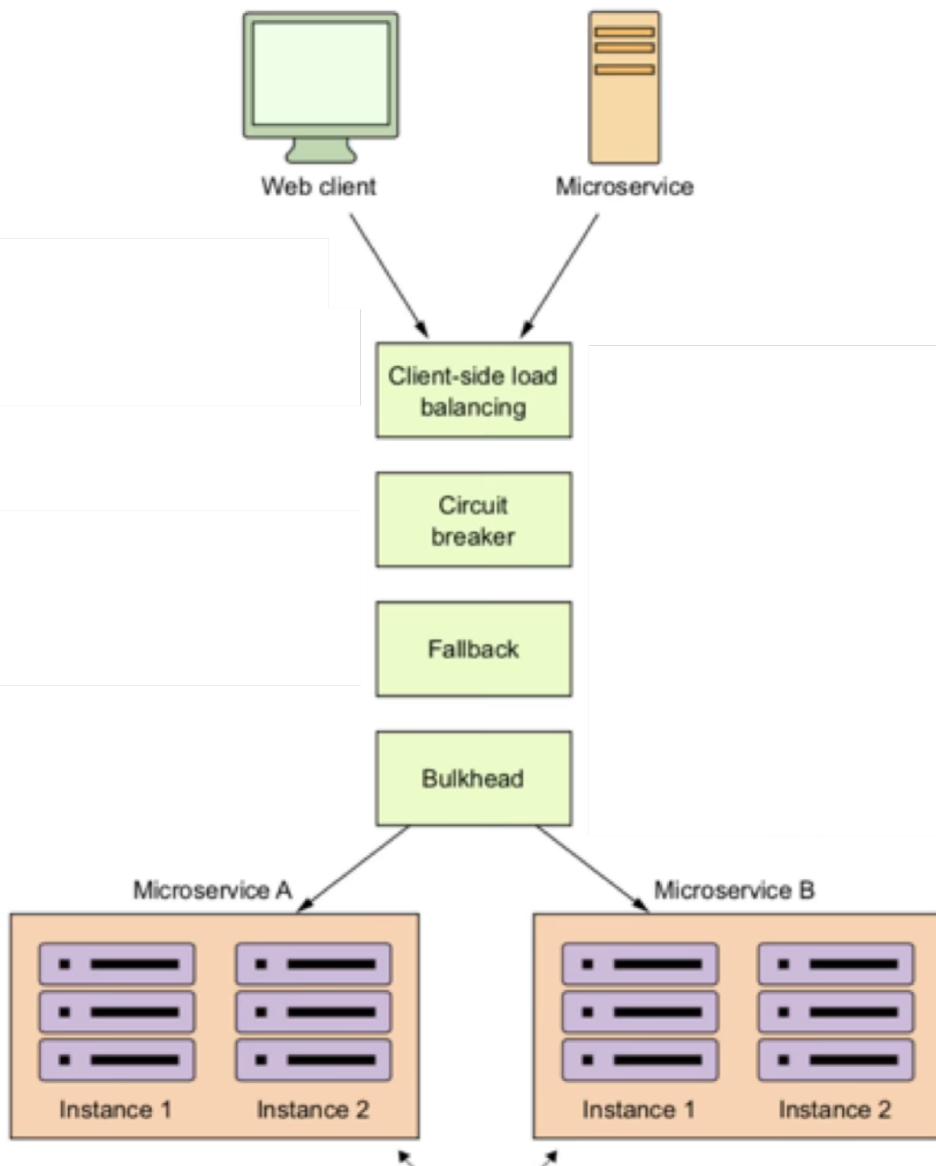
Definiciones previas

- programación multithread
 - Ejecucion concurrente con máxima cpu
 - Parte programa=thread
 - Thread=proceso dentro proceso
- Thread pool
 - Reusa threads
 - Usa thread luego devuelve



Patrones de resiliencia

- Proteger cuando recurso fallando
- Permitir fail-fast para no consumir recursos
- Prevenir problema extienda



- Balanceo cliente

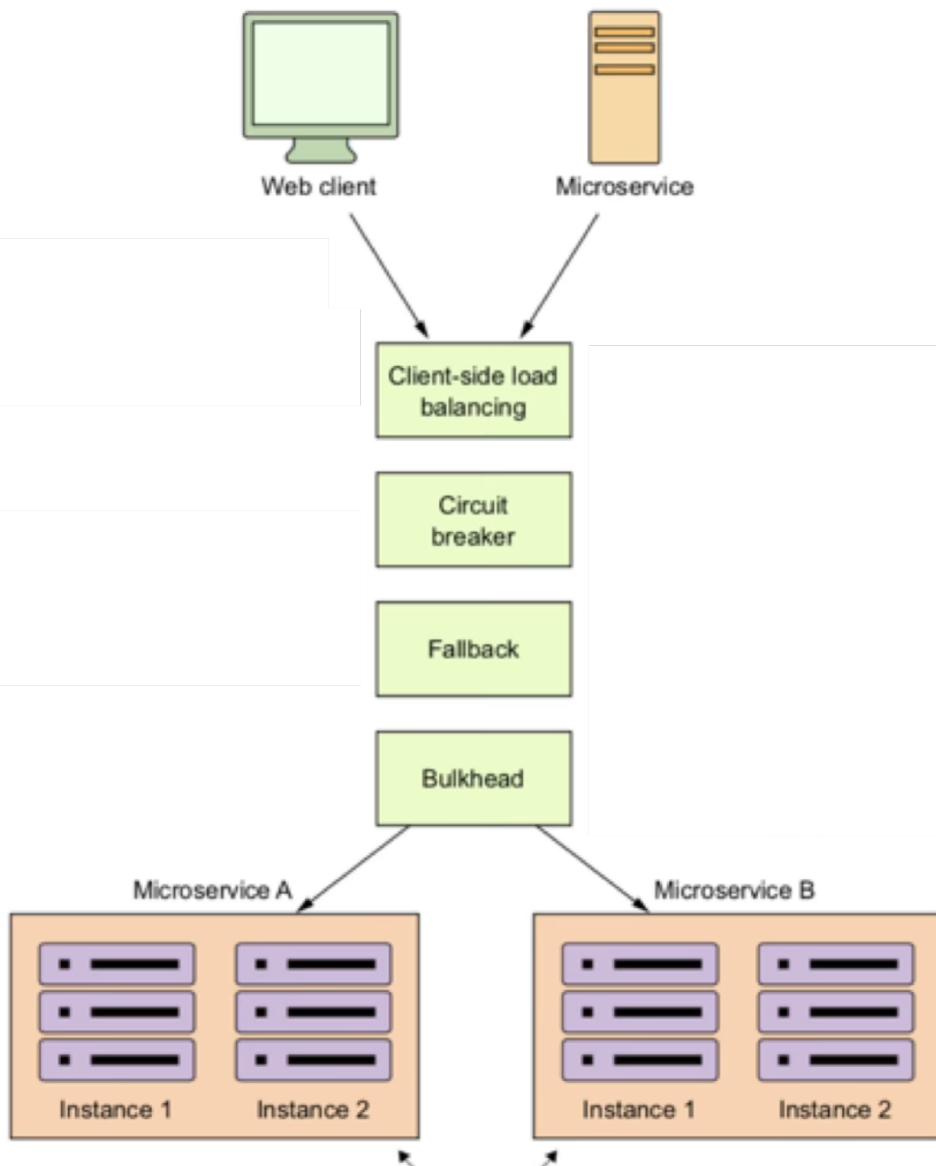
- Detecta errores
- Remueve instancias, previene errores

- Circuit breaker

- Circuitos eléctricos:
 - Si demasiada corriente, corta conexión
 - Previene componentes quemados

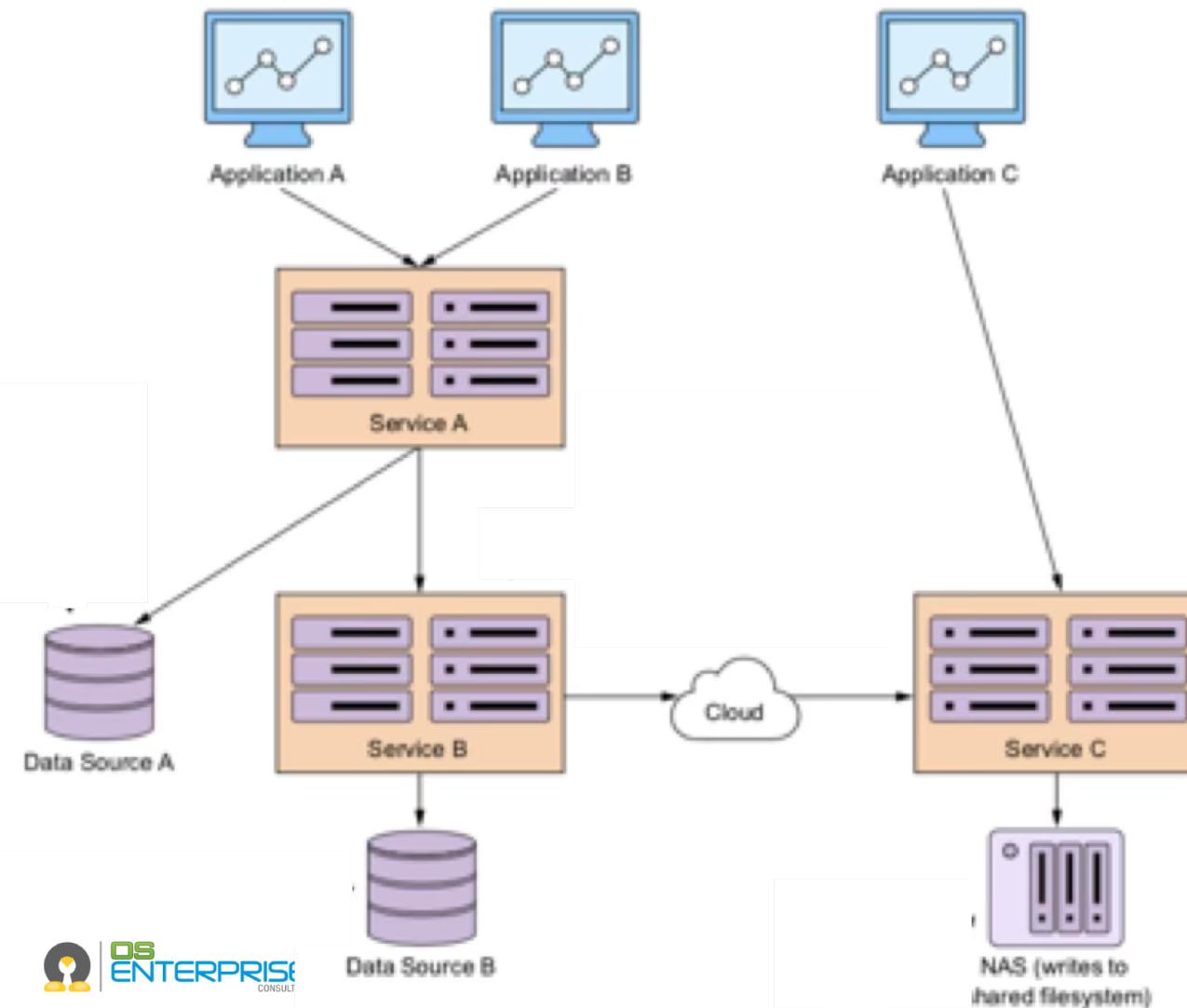
- Software

- Monitorea servicios remotos
 - Si toma mucho, corta llamada
- Monitorea total llamadas
 - Si fallan, fail-fast llamada

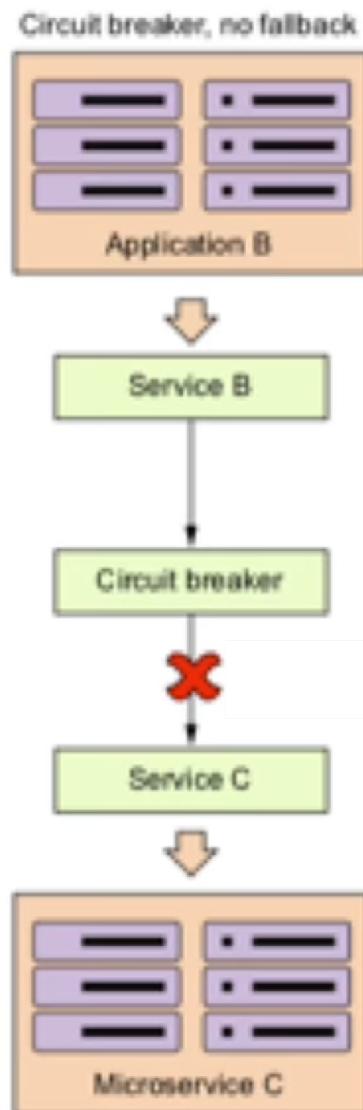


- **Fallbacks o plan b**
 - Llamada falla, código alternativo
 - Otra base de datos, encola request
 - Se Notifica al cliente
- **Bulkhead o particiones**
 - Diseño particionado
 - Separar recurso propio thread-pool
 - Riesgo 1 recurso tumbe aplicación anterior
 - Si servicio lento. Thread-pool satura
 - Recurso usa propio thread-pool
 - Otros servicios no se saturan

ejemplo



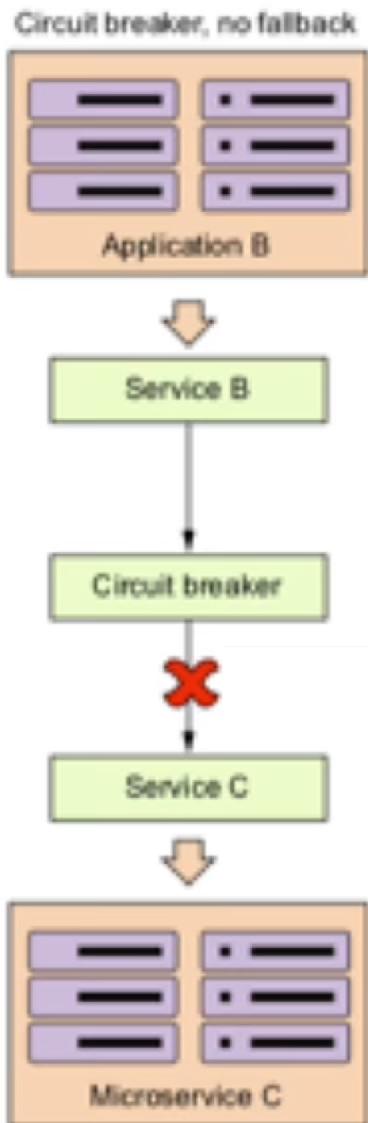
- Las 3 apps dejan de responder
- Evitado con circuit breaker.
 - C fallara rápidamente
 - Solo partes de B fallaran



Con Circuit breaker

- Intermediario entre apps y servicios
- B delega invocación
 - Envuelve llamada en thread-pool
- B no espera directamente
 - CB monitorea llamada
 - la mata si demora mucho
- B recibe error
 - CB, cuenta numero de fallas
 - Suficientes errores, CB activa
 - Llamadas a C fallan

Beneficios



- B no espera timeout
- B devuelve error o fallback
- C puede recuperarse
- Algunas llamadas pasan
Si éxito, CB desactiva

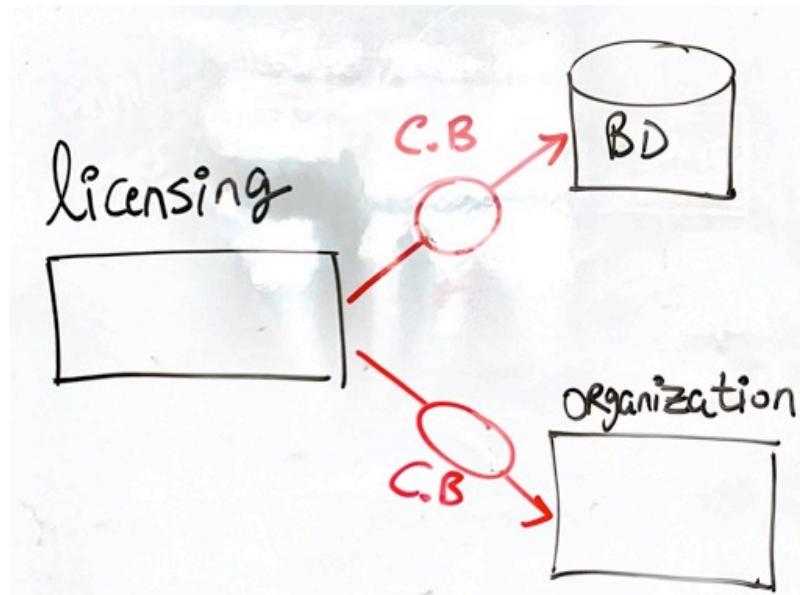
Hystrix

- implementa circuit breakers, fallbacks y bulkhead

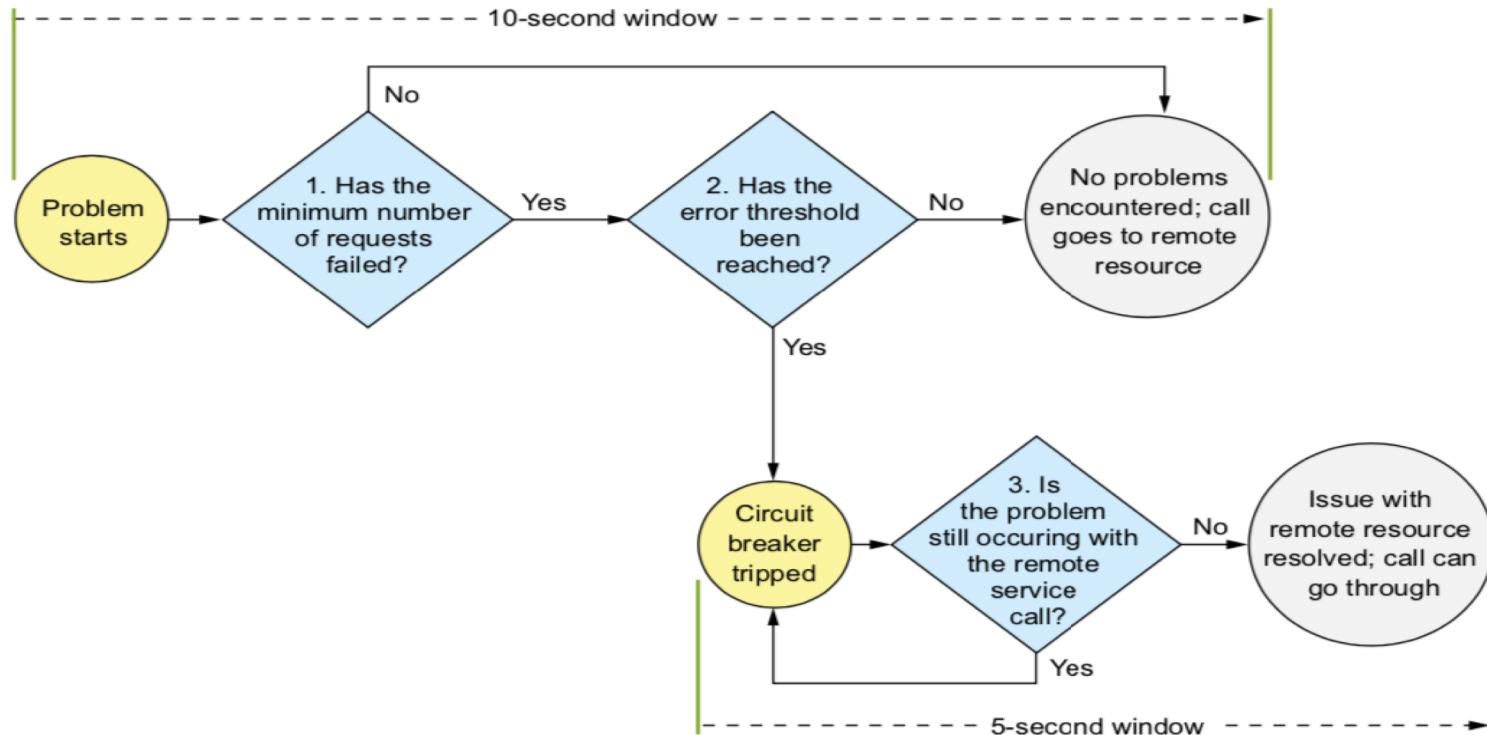


Laboratorio

Circuit breakers



Circuit breaker details



- Tiempo monitoreo: 10'
- Volumen límite fallas: 20
- Porcentaje máximo fallas: 50%
- Tiempo pausa: 5'