Charla Microservicios

Capgemini

¿Quiénes somos?

pablo.jimenez-martinez@capgemini.com

armen.mirzoyan-denisov@capgemini.com

Equipo de la SDO (Solution Delivery Office) en Capgemini.

- Ayudamos en el diseño / arquitectura de las aplicaciones y en la solución de las piezas más complejas
- Damos soporte en el arranque de los proyectos y, sobre todo, cuando saltan riesgos y problemas.
- Hacemos formaciones y mentoring para los compañeros del área de Capgemini Spain.

© Capgemini 2025. Todos los derechos reservados |

Introducción

Piezas clave microservicios

Retos de los microservicios

Ejercicios → Necesitáis un IDE instalado

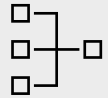
© Capgemini 2025. Todos los derechos reservados |

Introducción

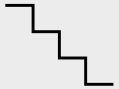
Todo tiende a fraccionarse / especializarse

1950 1970 1990 2010

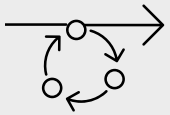
METODOLOGÍA



Programación Estructurada

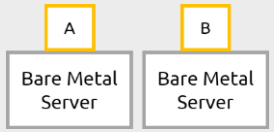


Waterfall

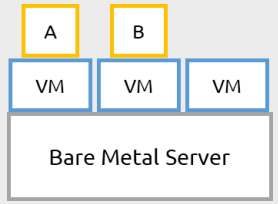


Programación Agile

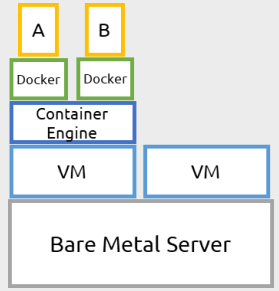
INFRAESTRUCTURA



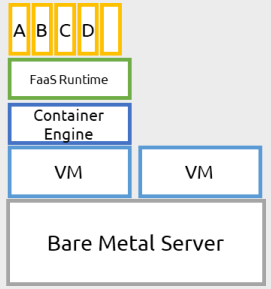
Dedicated Server



Virtualization

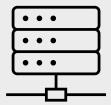


Containerization

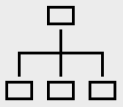


Serverless, FaaS

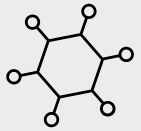
DESARROLLO



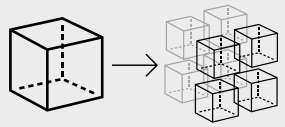
Monolith



N-Tier



SOA



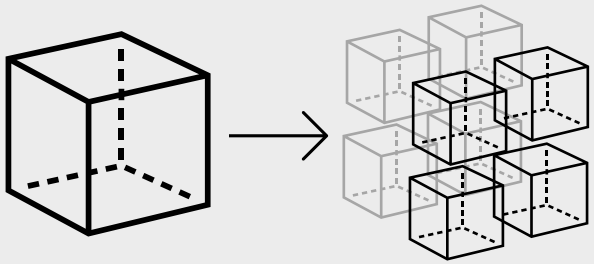
Microservices

Serverless

© Capgemini 2025. Todos los derechos reservados |

Introducción

¿Qué son los microservicios?

¿  ?

© Capgemini 2025. Todos los derechos reservados |

Introducción

¿Qué son los microservicios?

In short, the microservice architectural style is an **approach to developing a single application as a suite of small services**, each running in its own process and **communicating with lightweight mechanisms**, often an HTTP resource API. These services are built around business capabilities and independently deployable by fully automated deployment machinery. There is a bare minimum of centralized management of these services, **which may be written in different programming languages and use different data storage technologies.**

Martin Fowler

© Capgemini 2025. Todos los derechos reservados |

Introducción

¿Qué son los microservicios?

Pequeños desarrollos con responsabilidad única, implementados en el lenguaje y almacenamiento óptimo para su funcionalidad, que se interrelaciona con otras piezas con protocolos ligeros y que, en conjunto, funcionan como un sistema más complejo.

Martin Fowler

© Capgemini 2025. Todos los derechos reservados |

Introducción

¿Beneficios e inconvenientes?

© Capgemini 2025. Todos los derechos reservados |

Introducción

¿Beneficios e inconvenientes?

Beneficios

- Altamente escalable
  - **Optimización de recursos**
- Software pequeño
  - **Muy rápidos de arrancar y construir**
- Equipos en paralelo
  - **Entregas de producto más rápidas**
- Cada servicio en el lenguaje más adecuado
  - **Mejor aprovechamiento de la tecnología**

© Capgemini 2025. Todos los derechos reservados |

Introducción

¿Beneficios e inconvenientes?

Beneficios

- Altamente escalable
  - **Optimización de recursos**
- Software pequeño
  - **Muy rápidos de arrancar y construir**
- Equipos en paralelo
  - **Entregas de producto más rápidas**
- Cada servicio en el lenguaje más adecuado
  - **Mejor aprovechamiento de la tecnología**

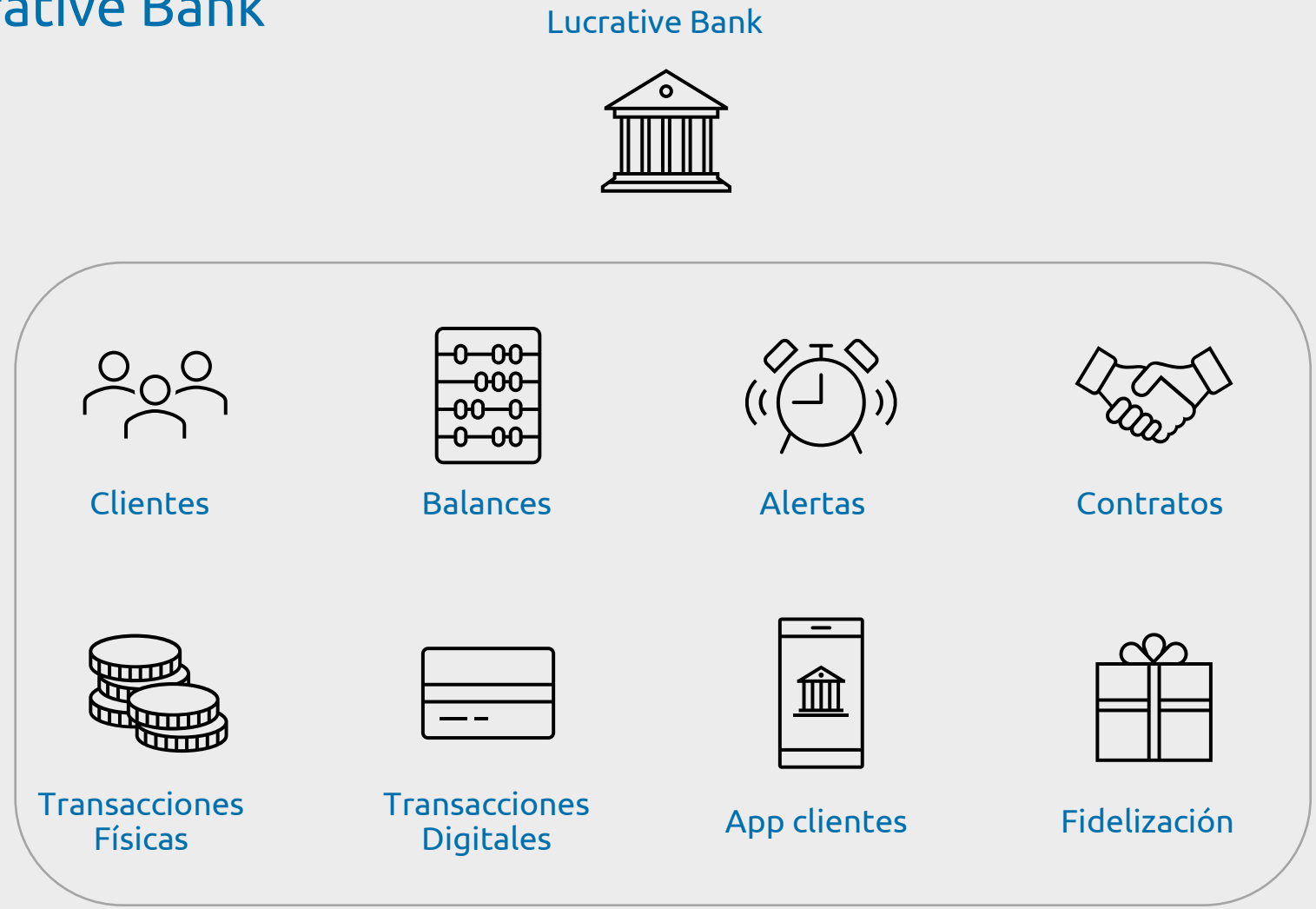
Inconvenientes

- Más complejos de desarrollar
  - **Necesitas infraestructura local**
  - **Necesitas coordinación**
  - **Necesitas una buena arquitectura**
  - **Necesitas un buen análisis**
- Más caros de desarrollar
  - **Son complejos de estructurar**
- Son muy “nuevos”
  - **Hay problemas de arquitectura que aún no están resueltos correctamente**

© Capgemini 2025. Todos los derechos reservados |

Introducción

Ejemplo: Lucrative Bank

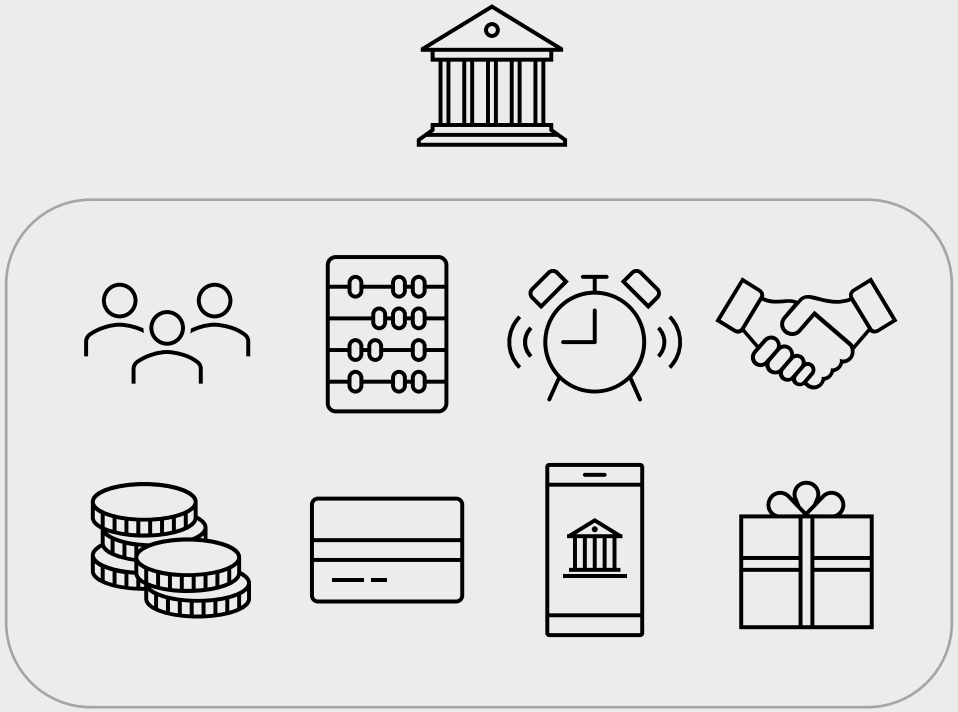


© Capgemini 2025. Todos los derechos reservados |

Introducción

Ejemplo: Lucrative Bank - Monolito

¡¡ BlackFriday !!

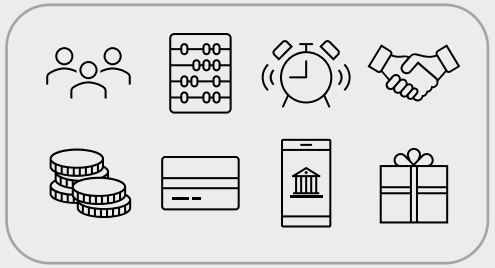


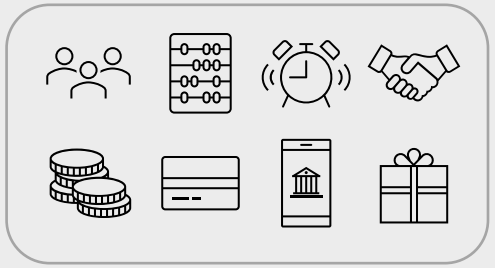
© Capgemini 2025. Todos los derechos reservados |

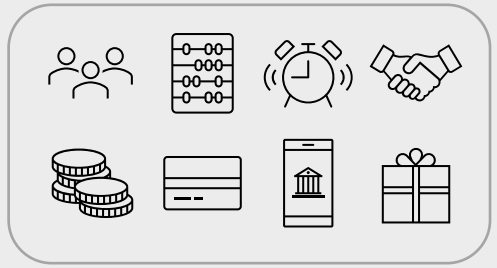
Introducción

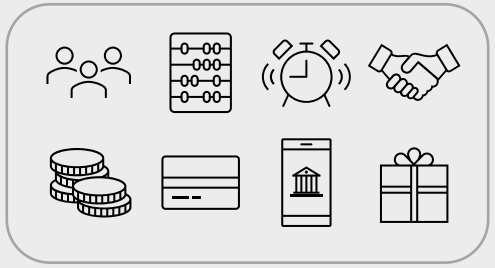
Ejemplo: Lucrative Bank - Monolito

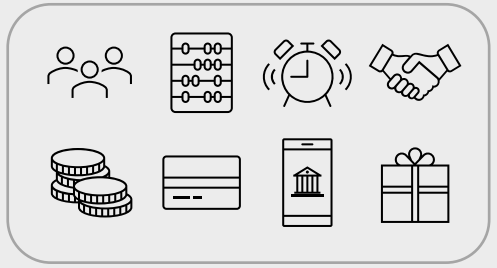
¡¡ BlackFriday !!











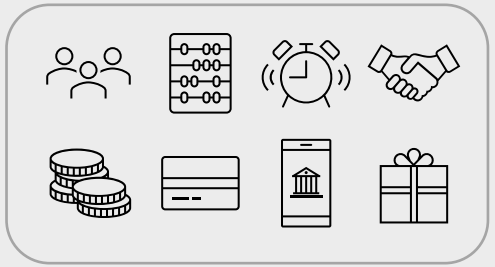
© Capgemini 2025. Todos los derechos reservados |

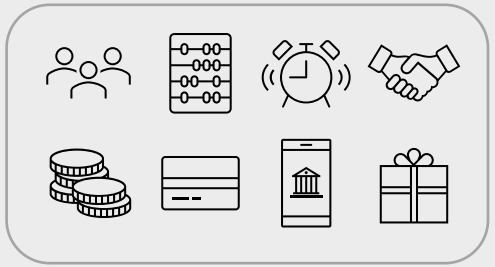
Introducción

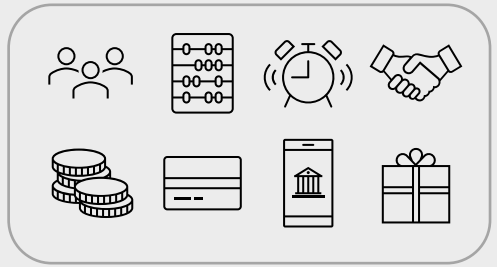
Ejemplo: Lucrative Bank - Monolito

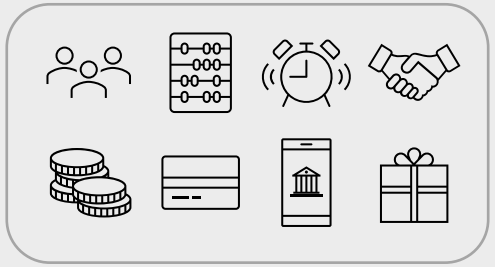
¡¡ BlackFriday !!

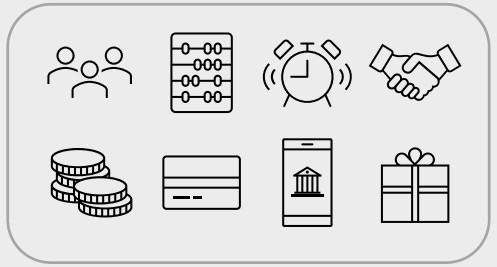
x5









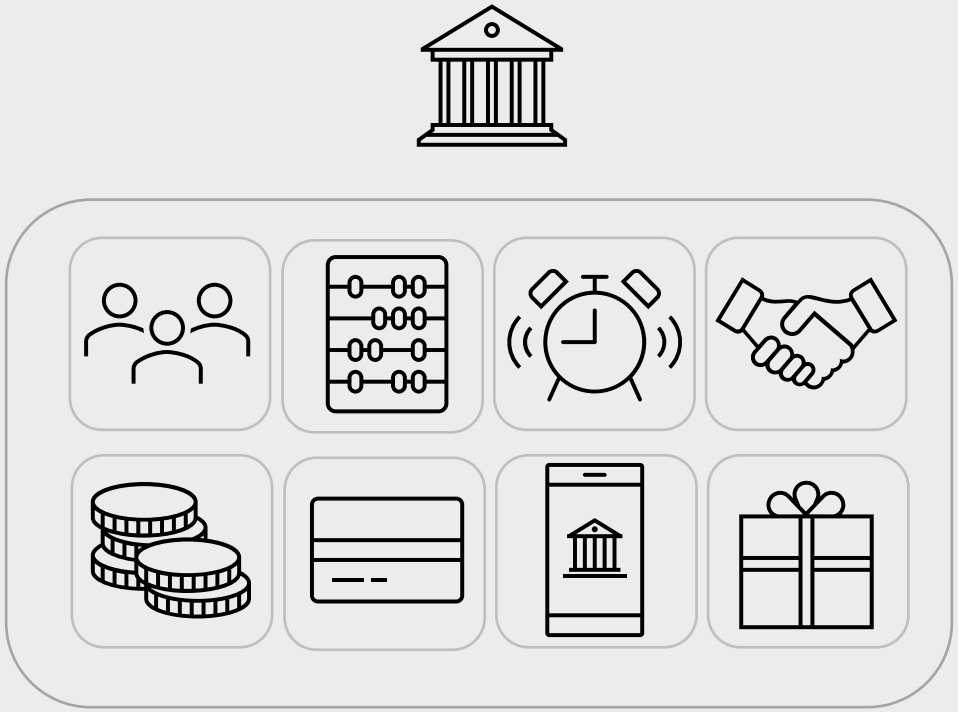


© Capgemini 2025. Todos los derechos reservados |

Introducción

Ejemplo: Lucrative Bank - Microservicios

¡¡ BlackFriday !!

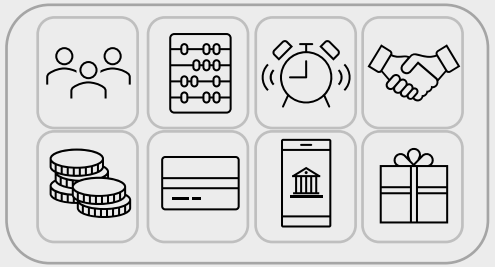


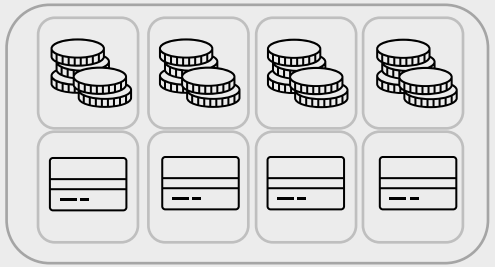
© Capgemini 2025. Todos los derechos reservados |

Introducción

Ejemplo: Lucrative Bank - Microservicios

¡¡ BlackFriday !!



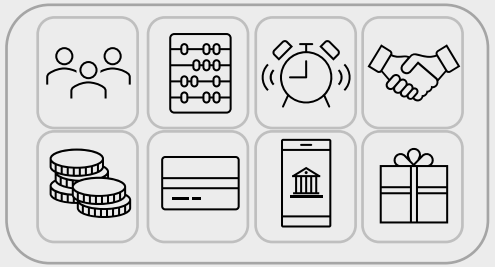


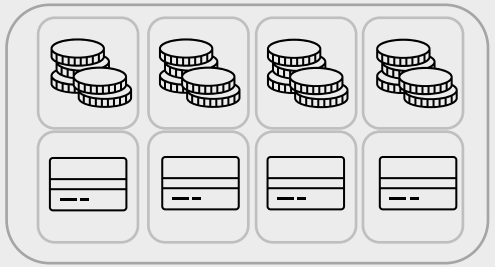
© Capgemini 2025. Todos los derechos reservados |

Introducción

Ejemplo: Lucrative Bank - Microservicios

¡¡ BlackFriday !!
2x



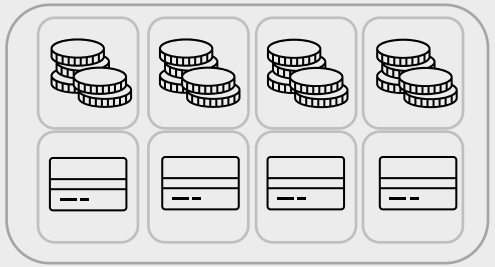


© Capgemini 2025. Todos los derechos reservados |

Introducción

Ejemplo: Lucrative Bank - Microservicios

¡¡ BlackFriday !!
2x



© Capgemini 2025. Todos los derechos reservados |

Introducción

Conclusión

- Microservicios
  - **Pequeñas piezas desarrolladas de forma aislada, en su propia tecnología y almacenamiento de datos, que se comunican entre sí para conformar un sistema más complejo.**
- ¿Para qué?
  - **Sistemas más escalables, más ajustados a las tecnologías, desarrollos más paralelizables.**
- Handicaps
  - **Más complejos y por tanto más caros de desarrollar, retos inexplorados***
- ¿Cuándo? → Depende
  - **¿Mi aplicación es sencilla o tiene una arquitectura compleja / muy grande?**
  - **¿Necesito alta escalabilidad solo de algunas partes de mi aplicación? ¿Evolución rápida?**
  - **¿Quiero trabajar en paralelo con muchos equipos o no tengo prisa por la entrega?**
  - **¿Tengo dinero para invertir en expertos / tengo expertos en arquitectura?**

© Capgemini 2025. Todos los derechos reservados |

Introducción

Piezas clave microservicios

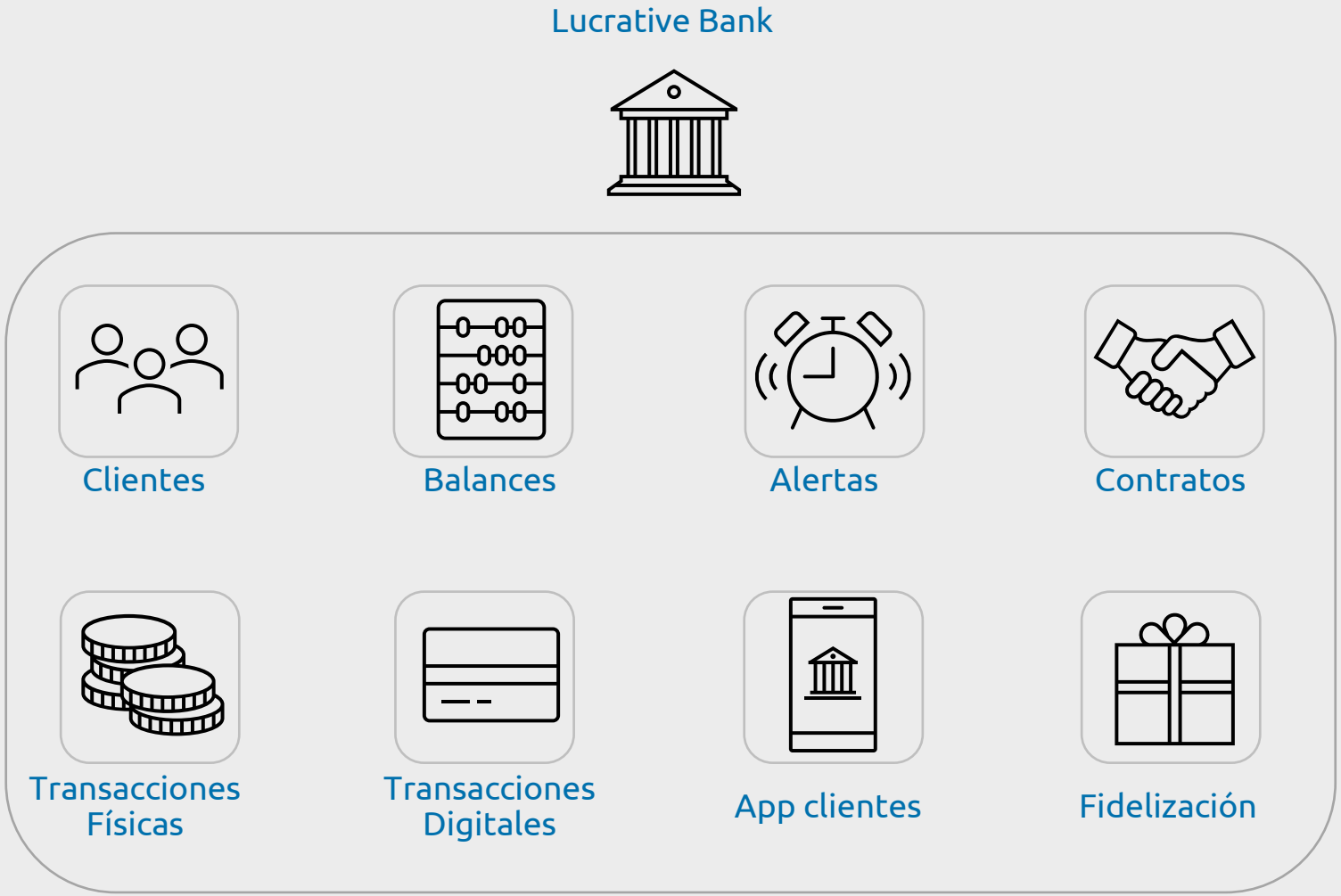
Retos de los microservicios

Ejercicios

© Capgemini 2025. Todos los derechos reservados |

Piezas clave en la arquitectura de microservicios

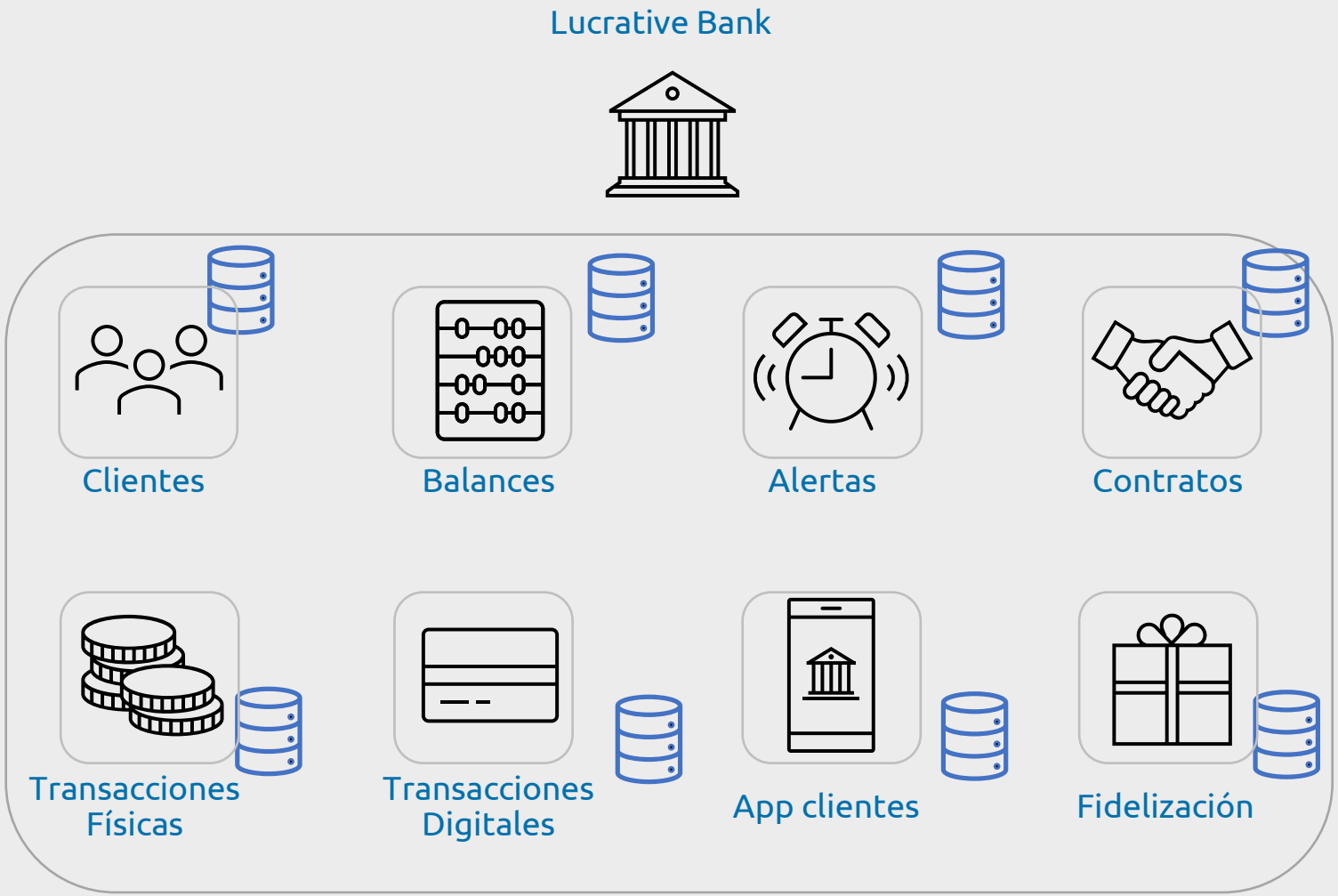
Gobierno de microservicios



© Capgemini 2025. Todos los derechos reservados |

Piezas clave en la arquitectura de microservicios

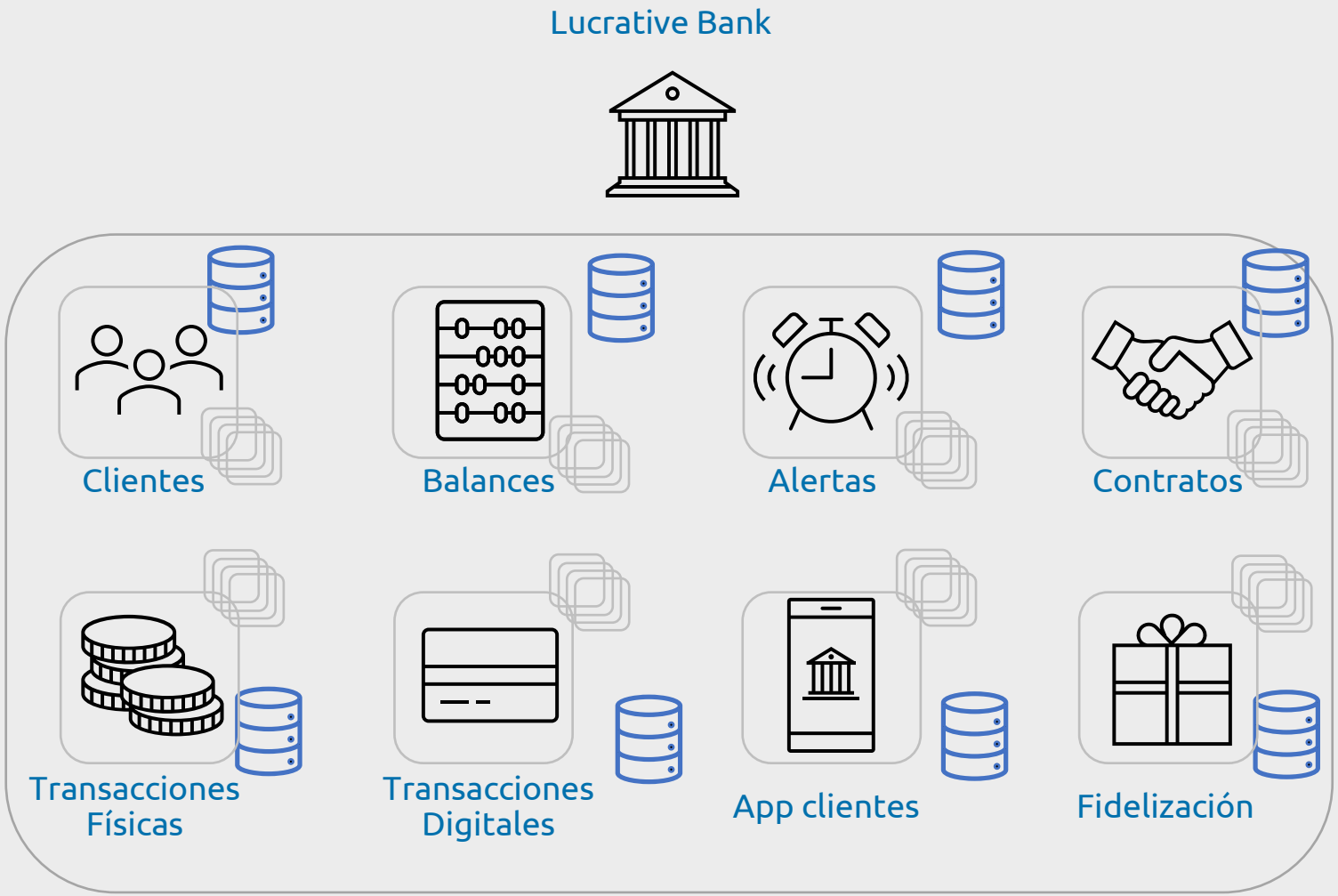
Gobierno de microservicios



© Capgemini 2025. Todos los derechos reservados |

Piezas clave en la arquitectura de microservicios

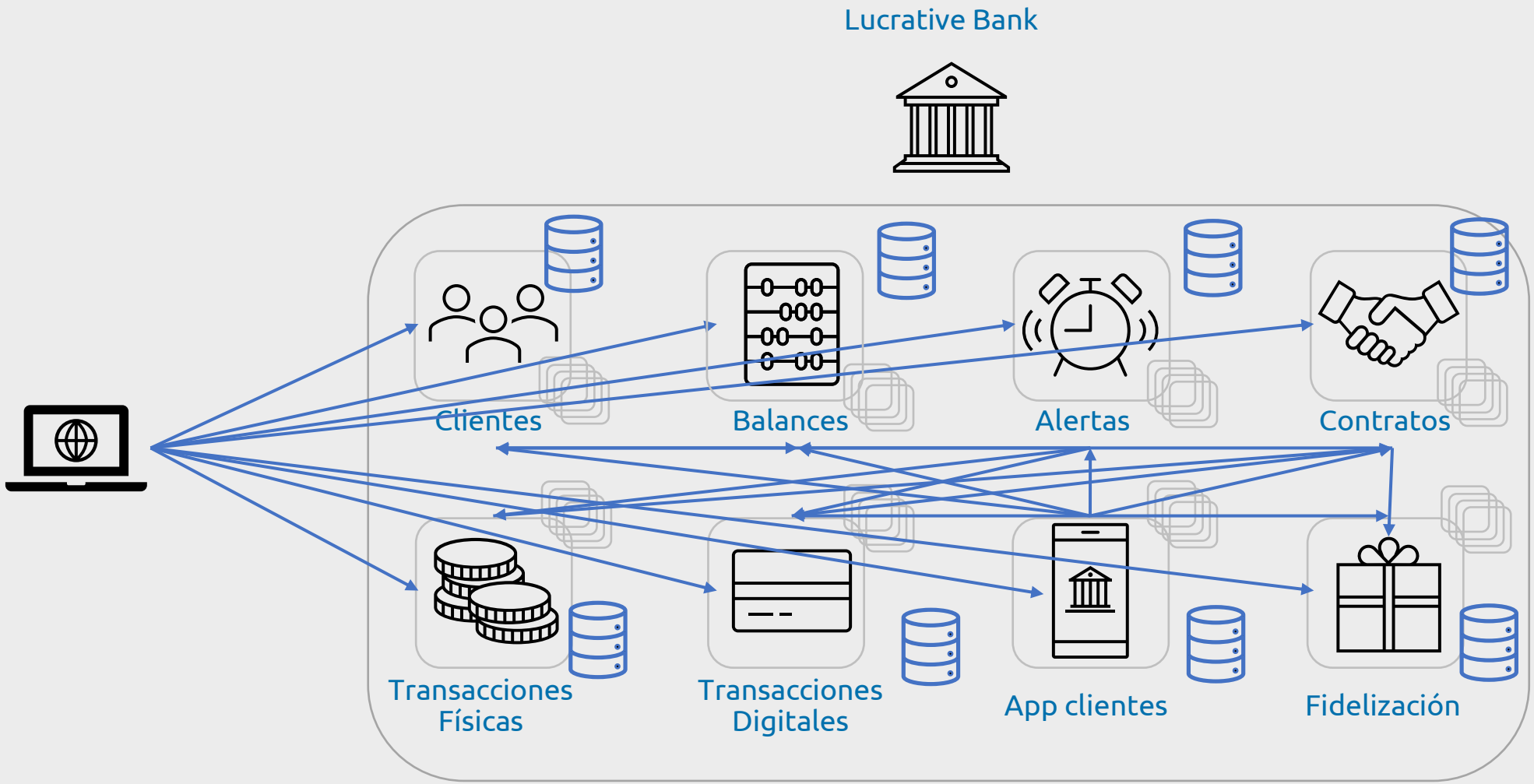
Gobierno de microservicios



© Capgemini 2025. Todos los derechos reservados |

Piezas clave en la arquitectura de microservicios

Gobierno de microservicios

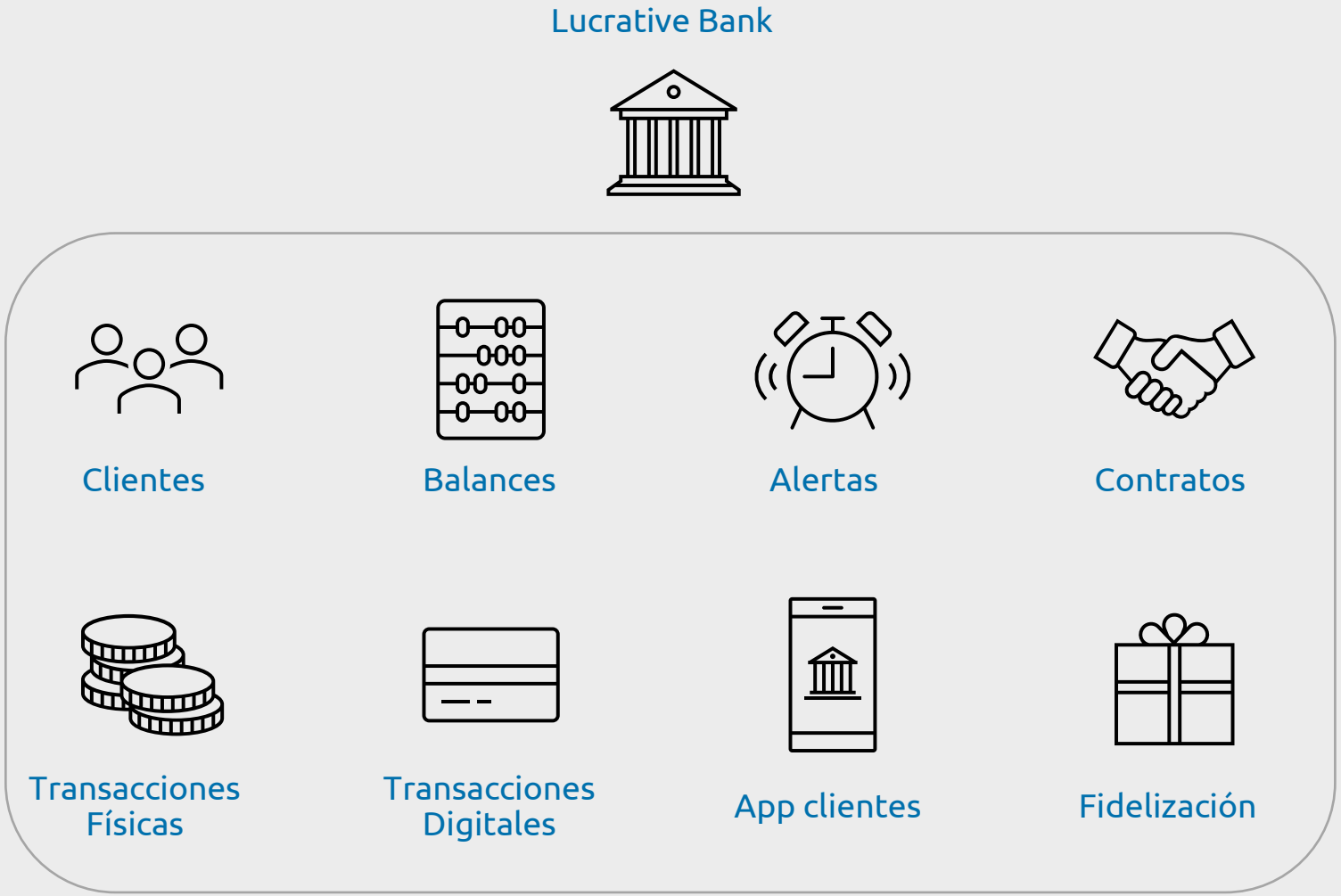


© Capgemini 2025. Todos los derechos reservados |

Piezas clave en la arquitectura de microservicios

Gobierno de microservicios

- Configuration Service
- Discovery Service
- Load Balancing
- Circuit Breaker
- Edge Service
- Log Management
- Authentication



© Capgemini 2025. Todos los derechos reservados |

Piezas clave en la arquitectura de microservicios

Configuration Service

- Configuration Service
- Discovery Service
- Load Balancing
- Circuit Breaker
- Edge Service
- Log Management
- Authentication

- Existe mucha configuración que es común entre todos los microservicios
  - **BBDD, URL aplicaciones de terceros, claves encriptación**
- Duplicar, en general, es mala idea
  - **MUY MALA IDEA**
- Necesitamos
  - **Un lugar para guardar configuración**
  - **Una forma para leer esa información en arranque**
  - **Una forma para cambiar esa información en caliente**

Spring Cloud Config Server

Provider Config Server

© Capgemini 2025. Todos los derechos reservados |

Piezas clave en la arquitectura de microservicios

Discovery Service

- Configuration Service
- **Discovery Service**
- Load Balancing
- Circuit Breaker
- Edge Service
- Log Management
- Authentication

- A menudo vamos a querer hacer llamadas entre microservicios
- Los microservicios hemos visto que pueden ser escalables y se pueden arrancar en diferentes servidores
- Vamos a necesitar un catálogo
  - **Conocer que servicios tenemos disponibles**
  - **Donde están ubicados (IP / Port)**

Spring Cloud Eureka

Kubernetes

© Capgemini 2025. Todos los derechos reservados |

Piezas clave en la arquitectura de microservicios

Load Balancing

- Configuration Service
- Discovery Service
- **Load Balancing**
- Circuit Breaker
- Edge Service
- Log Management
- Authentication

- Cuando voy a hacer una llamada entre microservicios y el destino tiene muchas instancias, voy a querer poder elegir la instancia que más me beneficie
- Necesitamos
  - **Un algoritmo que dada una entrada (catálogo de servicios + estadísticas de usuario + métricas de servicios), sepa decidir que instancia es la mejor**

Spring Cloud Ribbon / Feign Client

Kubernetes / Balancer

© Capgemini 2025. Todos los derechos reservados |

Piezas clave en la arquitectura de microservicios

Circuit Breaker

- Configuration Service
- Discovery Service
- Load Balancing
- **Circuit Breaker**
- Edge Service
- Log Management
- Authentication

- Cuando hago una llamada entre microservicios
  - **¿Qué pasa si la llamada da error?**
  - **¿Qué pasa si el microservicio no está disponible?**
- Necesitamos
  - **Un algoritmo que sepa decidir que es lo que hacemos en ese caso**
  - **Una metodología y arquitectura bien definida. Programación orientada a fallos.**

Spring Cloud Hystrix

Retry ¿?

© Capgemini 2025. Todos los derechos reservados |

Piezas clave en la arquitectura de microservicios

Edge Service

- Configuration Service
- Discovery Service
- Load Balancing
- Circuit Breaker
- **Edge Service**
- Log Management
- Authentication

- ¿Y que pasa cuando quiero hacer una llamada fuera de mi ecosistema o cuando alguien de fuera quiere llamarme? ¿como lo redirijo donde corresponda?
- Necesitamos
  - **Un servidor que haga de Proxy entre infraestructuras / arquitecturas**
  - **Deseable que tenga balanceo de carga**

Spring Cloud Gateway

Edge Server / Proxy

© Capgemini 2025. Todos los derechos reservados |

Piezas clave en la arquitectura de microservicios

Log Management

- Configuration Service
- Discovery Service
- Load Balancing
- Circuit Breaker
- Edge Service
- **Log Management**
- Authentication

- Ocurrió un problema en una funcionalidad que pasa por 7 microservicios ¡¿qué hago?! ¿miro los 7 logs?
- Necesitamos
  - **Algo que sea capaz de recolectar los logs**
  - **Algo que sea capaz de darle un formato (cada microservicio puede ser de una tecnología)**
  - **Algo que sea capaz de buscar / mostrar logs**

Sleuth + Zipkin + ELK

Otras mil herramientas

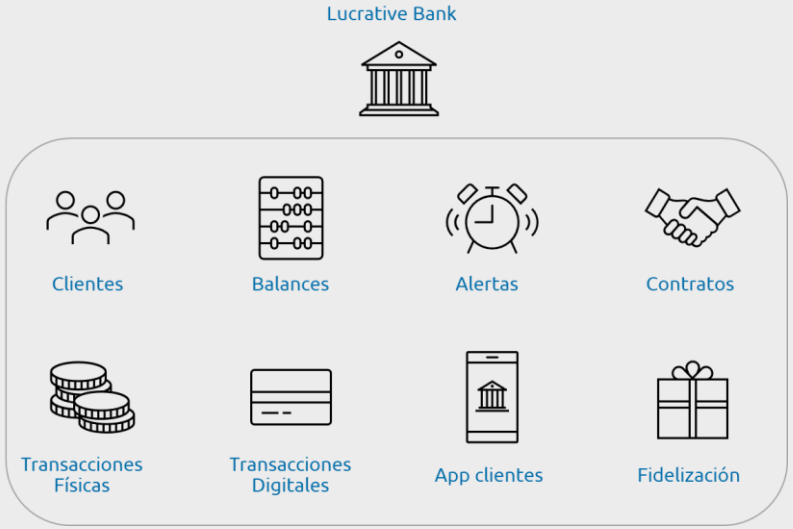
© Capgemini 2025. Todos los derechos reservados |

Piezas clave en la arquitectura de microservicios

Authentication and Authorization

- Configuration Service
- Discovery Service
- Load Balancing
- Circuit Breaker
- Edge Service
- Log Management
- **Authentication**

- Si tengo un usuario que utiliza los 8 microservicios, ¿tiene que hacer login en los 8 para usarlos?
- Necesitamos
  - **Si uso sesiones, tengo que replicarlas en todos los servicios**
  - **Mejor no usar sesiones, usar tokens tipo JWT, oAuth, etc.**

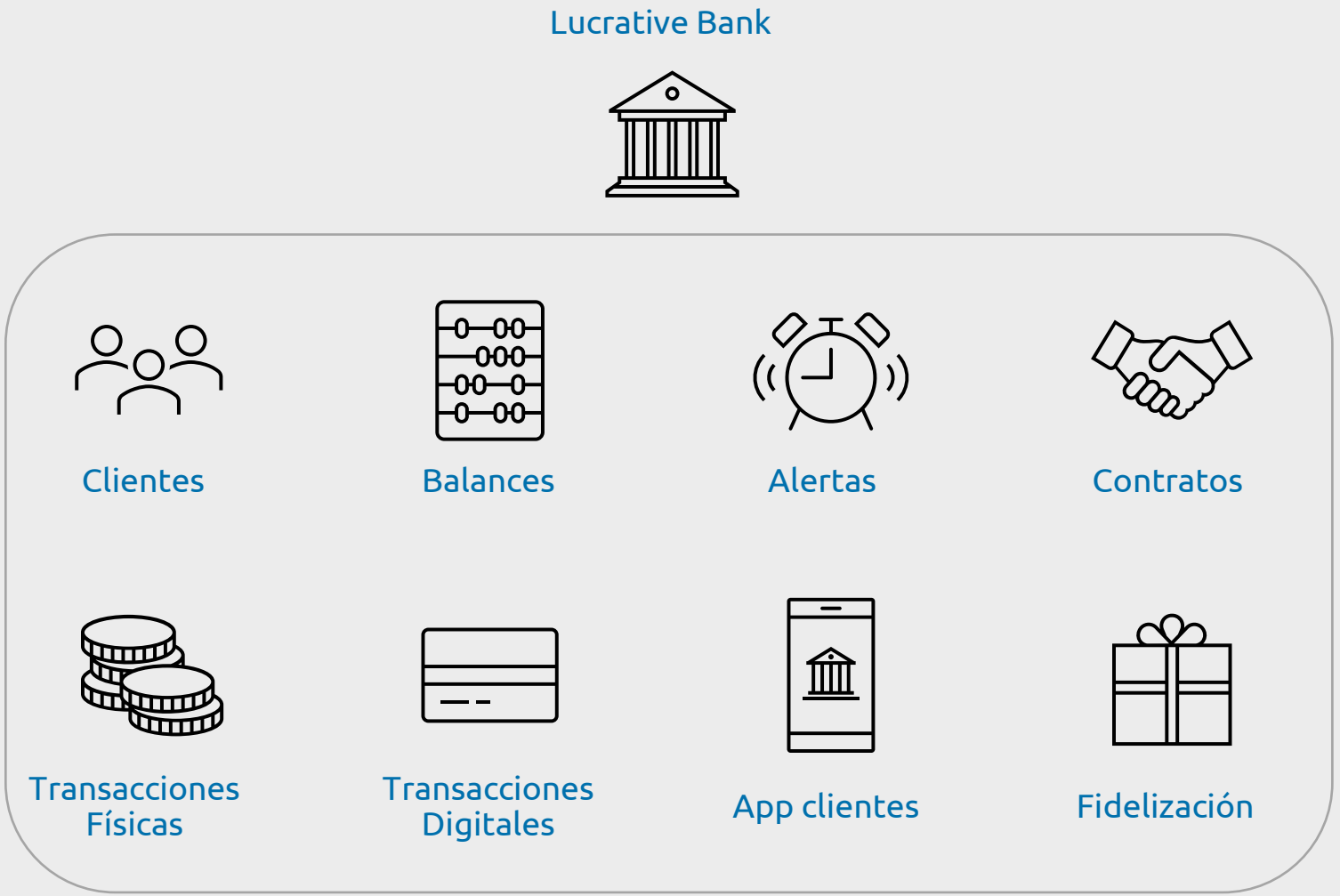


© Capgemini 2025. Todos los derechos reservados |

Piezas clave en la arquitectura de microservicios

Resumen

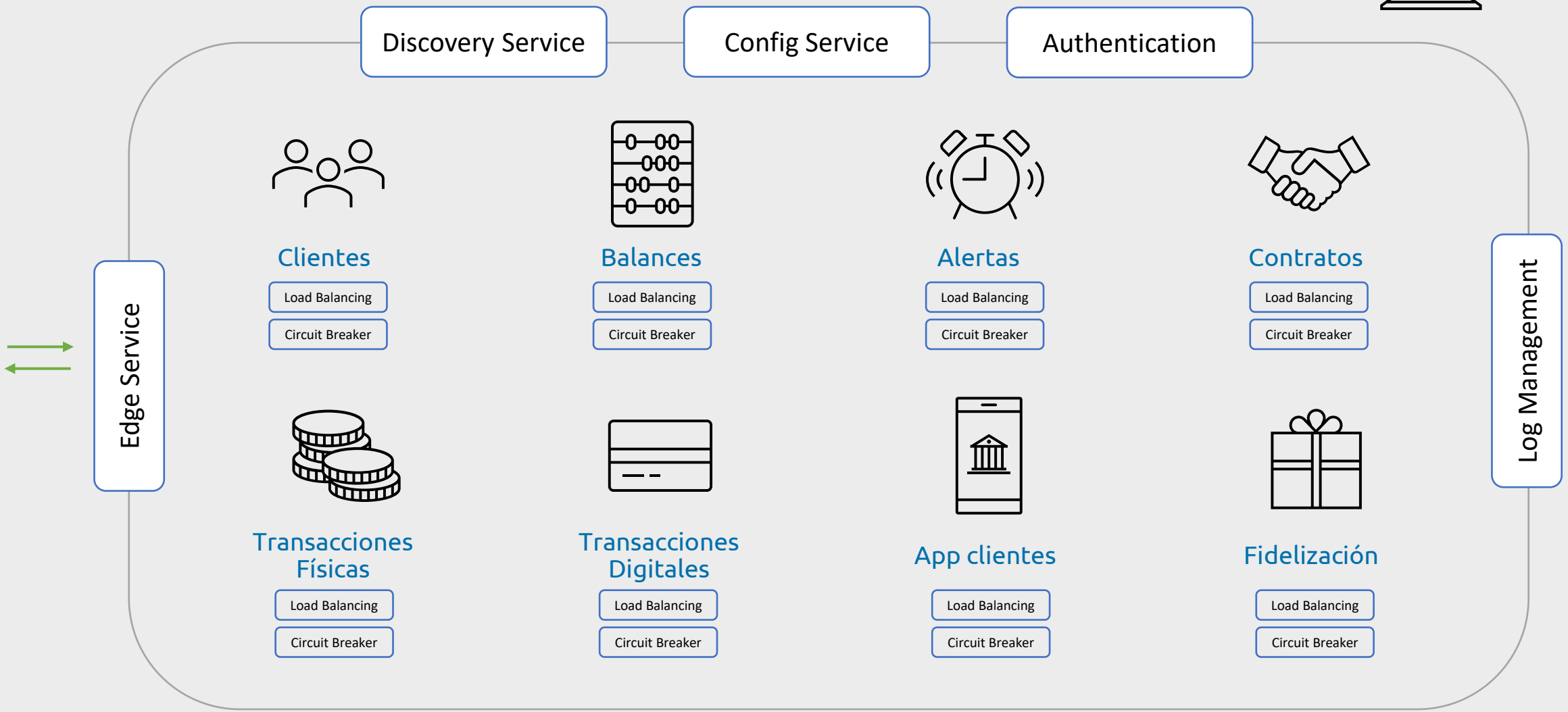
- Configuration Service
- Discovery Service
- Load Balancing
- Circuit Breaker
- Edge Service
- Log Management
- Authentication



© Capgemini 2025. Todos los derechos reservados |

Piezas clave en la arquitectura de microservicios

Resumen



© Capgemini 2025. Todos los derechos reservados |

Introducción

Piezas clave microservicios

Retos de los microservicios

Ejercicios

© Capgemini 2025. Todos los derechos reservados |

Retos de los microservicios

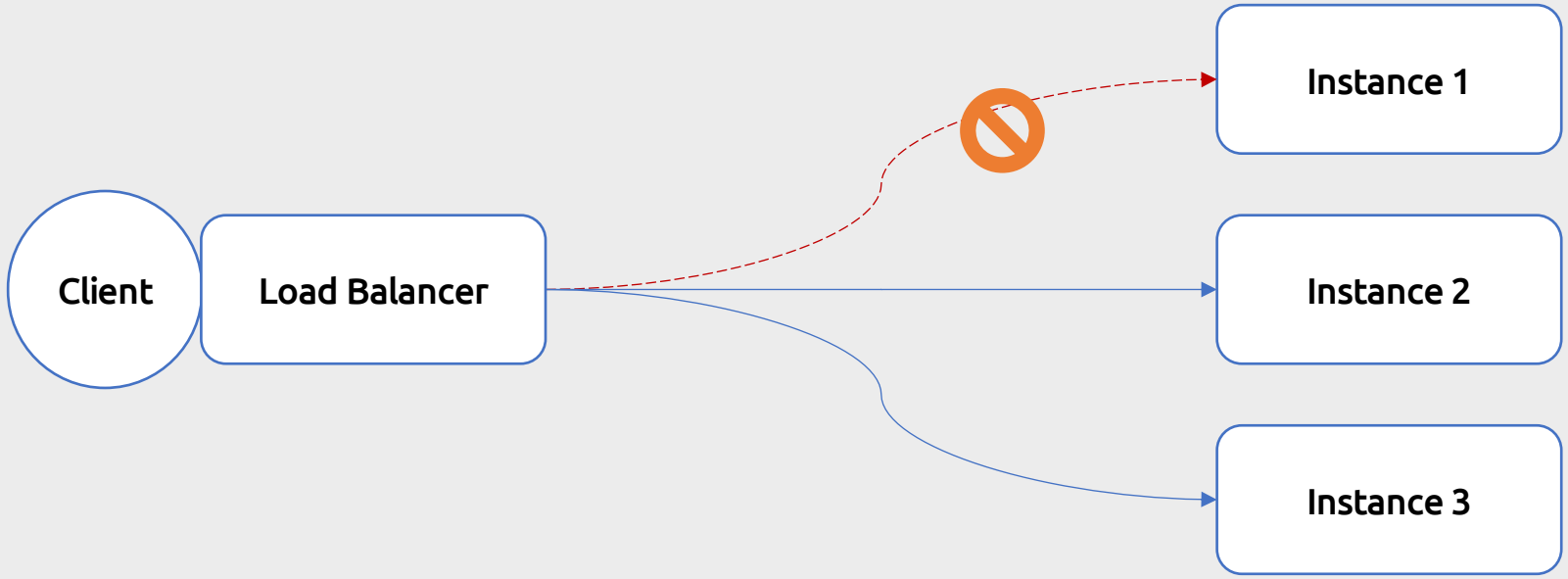
Principales problemas técnicos / funcionales que nos podemos encontrar

- Fallo en la comunicación síncrona – vía API Rest
  - **Hay muchísimas llamadas entre microservicios, debemos prepararnos para lo peor**
- Sincronización de información entre sistemas
  - **En muchas ocasiones queremos mantener una sincronización de datos/estados entre varios sistemas, Event Driven Architecture.**
- Particionamiento del dato
  - **Ojo que ahora tenemos múltiples microservicios con múltiples orígenes de datos**
  - **¿Cómo particionamos el dato? ¿Quién es el propietario? ¿Cómo creamos transacciones distribuidas?**

© Capgemini 2025. Todos los derechos reservados |

Retos de los microservicios

Fallo comunicación síncrona.

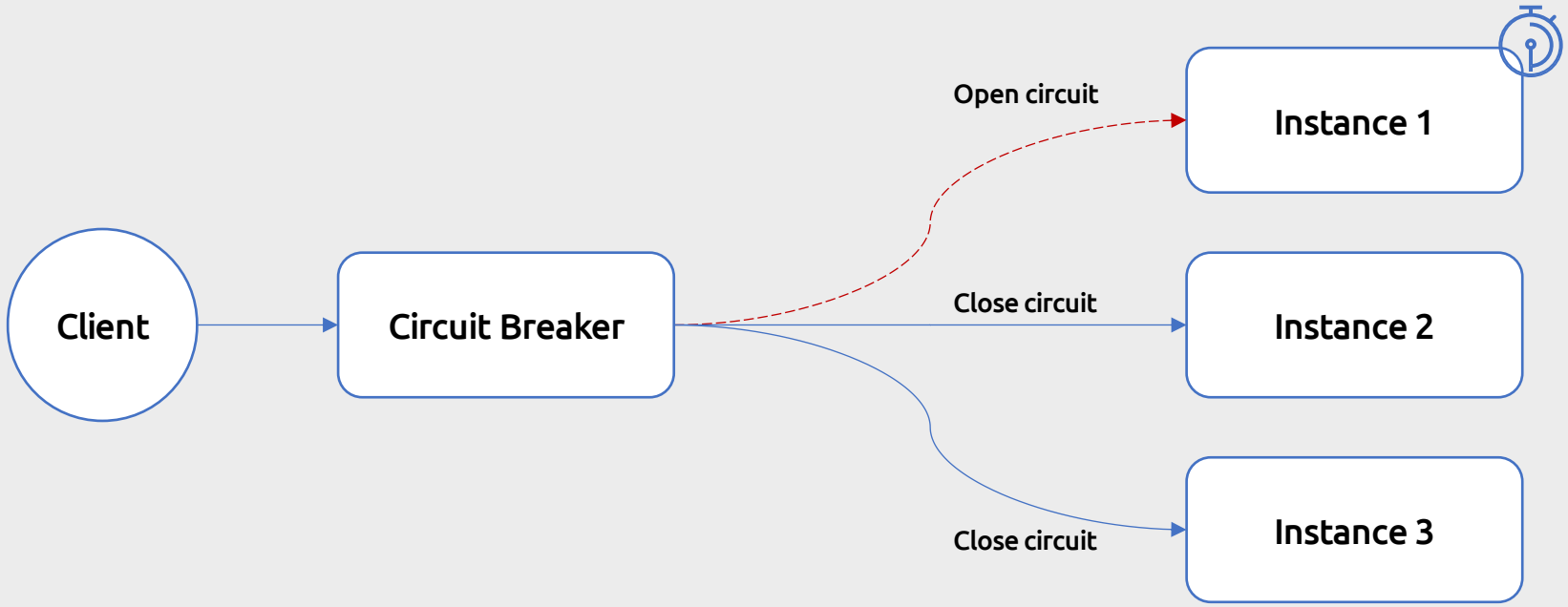


© Capgemini 2025. Todos los derechos reservados |

Retos de los microservicios

Fallo comunicación síncrona.

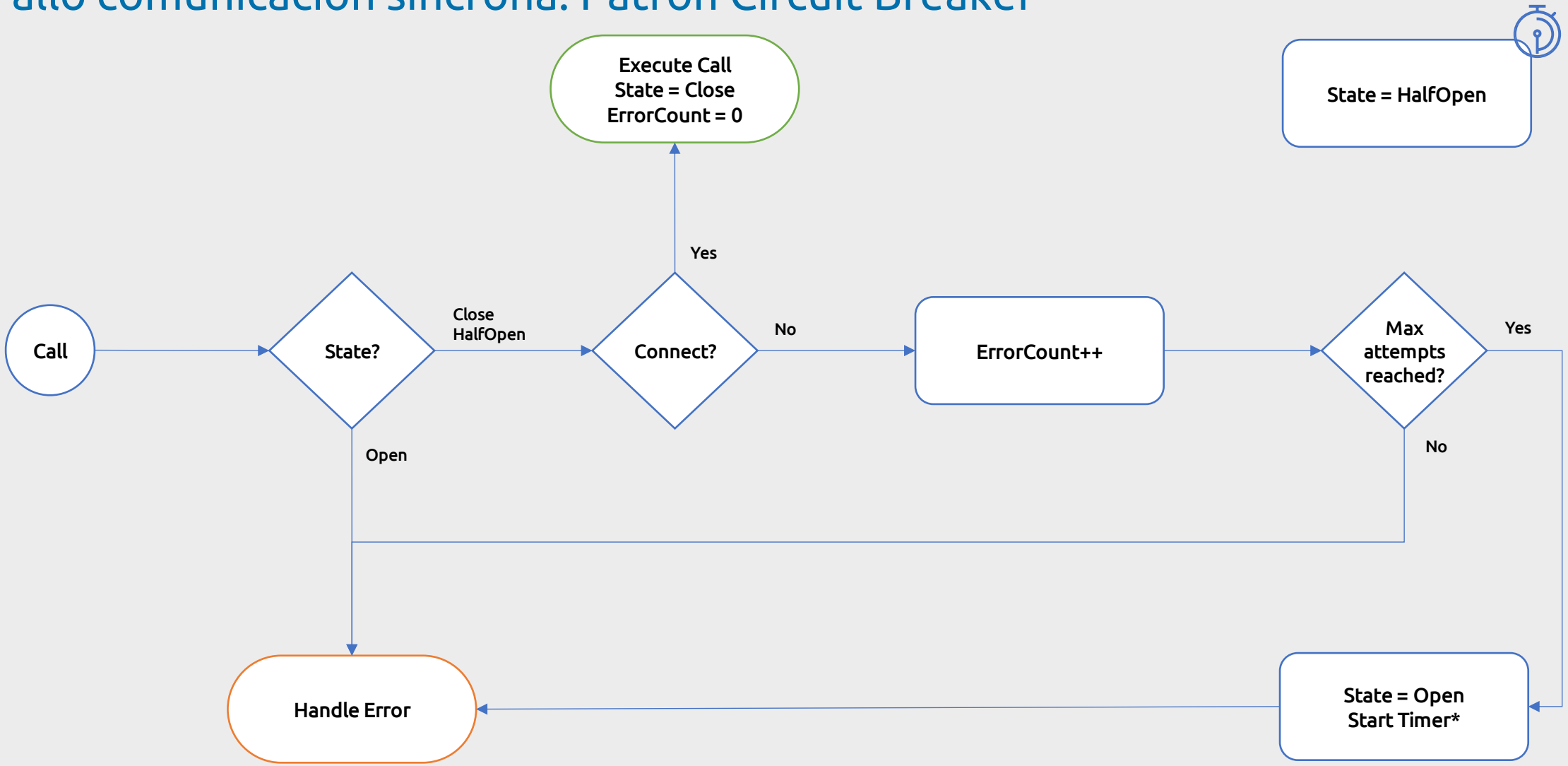
Patrón Circuit Breaker



© Capgemini 2025. Todos los derechos reservados |

Retos de los microservicios

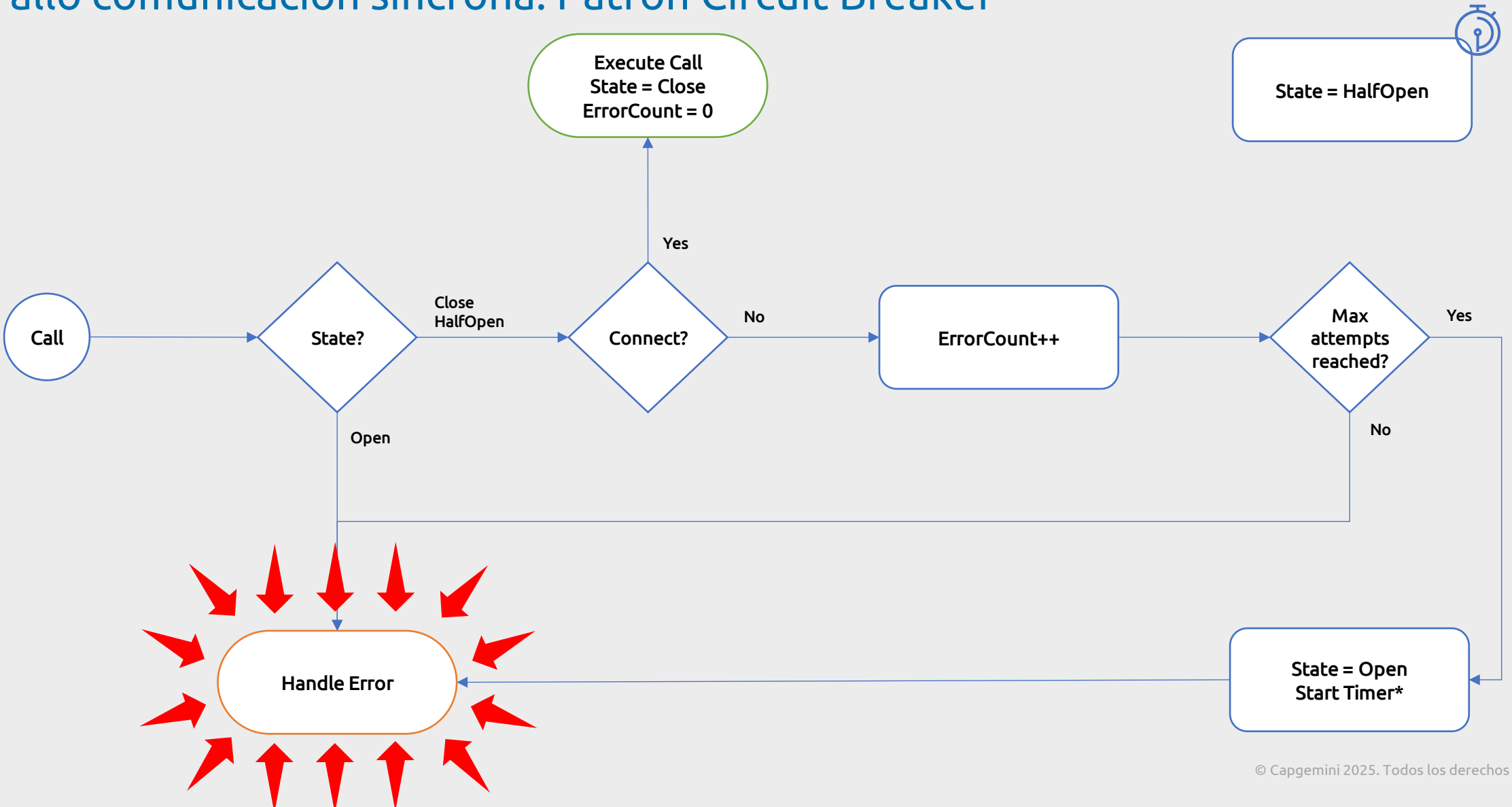
Fallo comunicación síncrona. Patrón Circuit Breaker



© Capgemini 2025. Todos los derechos reservados |

Retos de los microservicios

Fallo comunicación síncrona. Patrón Circuit Breaker



© Capgemini 2025. Todos los derechos reservados |

Retos de los microservicios

Fallo comunicación síncrona. Patrón Circuit Breaker – Springboot

```
@FeignClient(name = "test", fallback = Fallback.class)
protected interface TestClient {

    @RequestMapping(method = RequestMethod.GET, value = "/hello")
    Hello getHello();

    @RequestMapping(method = RequestMethod.GET, value = "/hellonotfound")
    String getException();

}
```

© Capgemini 2025. Todos los derechos reservados |

Retos de los microservicios

Fallo comunicación síncrona. Patrón Circuit Breaker – Springboot

```java
@FeignClient(name = "test", fallback = Fallback.class)
protected interface TestClient {

    @RequestMapping(method = RequestMethod.GET, value = "/hello")
    Hello getHello();

    @
    S
}
```

```java
@Component
static class Fallback implements TestClient {

    @Override
    public Hello getHello() {
        throw new NoFallbackAvailableException("Boom!", new RuntimeException());
    }

    @Override
    public String getException() {
        return "Fixed response";
    }

}
```

025. Todos los derechos reservados |

Retos de los microservicios

Sincronización entre sistemas. Event-Driven Architecture

- ¿Para qué la utilizamos?
    - **Localización agnóstica de los componentes que se comunican**
    - **Modelo de comunicación abanico → un publicador y n suscriptores**
    - **Asincronía**

- ¿Qué aporta?
    - **Bajo acoplamiento → Evitamos Load Balancer**
    - **Alta escalabilidad**
    - **Tolerancia a particionado → Evitamos problemas de red**

© Capgemini 2025. Todos los derechos reservados |

Retos de los microservicios

Sincronización entre sistemas. Event-Driven Architecture

- Tipos de arquitecturas basadas en eventos
  - **Arquitecturas para avisar → Notificaciones**
  - **Arquitecturas para sincronizar sistemas**
    - **Event-Carried State Transfer**
    - **Event Sourcing**

    } **Command Query Responsability Segregation (CQRS)**

- A tener en cuenta
  - **Eventual Consistency**
    - **La información tarda en propagarse**
    - **No existe un único estado actual**
    - **Alta disponibilidad**
  - **Sistemas más complejos y más piezas de infraestructura**

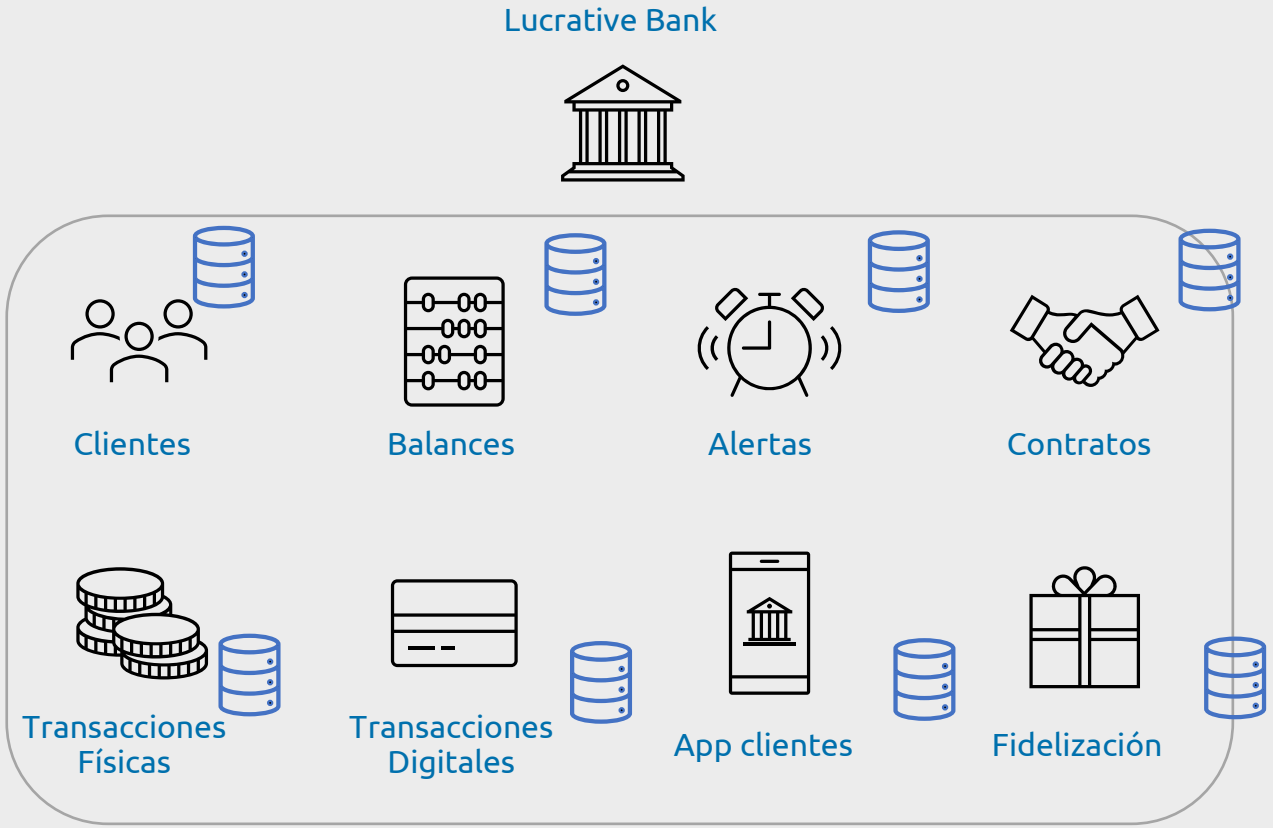
© Capgemini 2025. Todos los derechos reservados |

Retos de los microservicios

Particionamiento del dato

- Como segmentamos los datos
  - **Problema puramente funcional**

- Como accedemos a los datos
  - **Problema puramente técnico**
    - **Como leemos el dato particionado**
    - **Como escribimos el dato particionado**

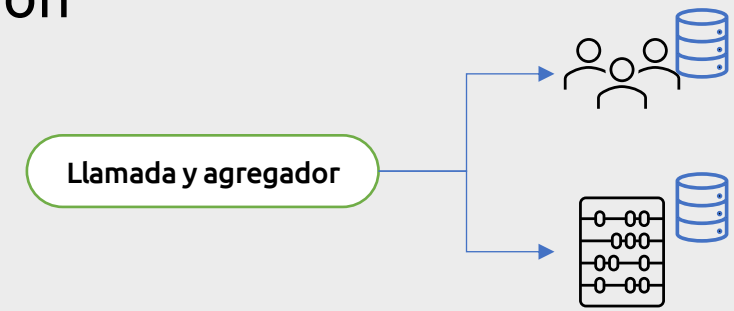


© Capgemini 2025. Todos los derechos reservados |

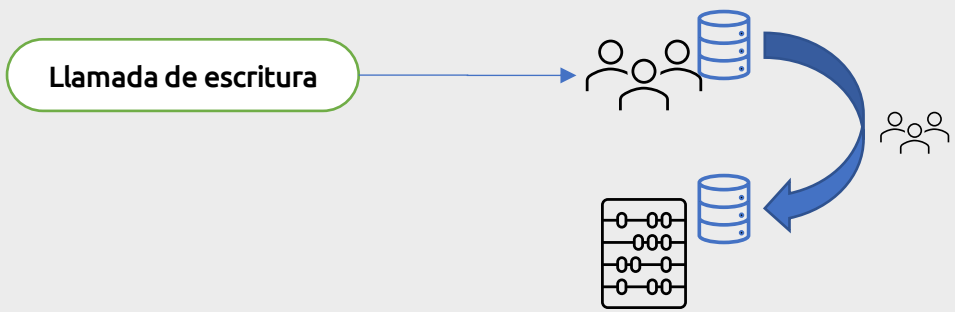
Retos de los microservicios

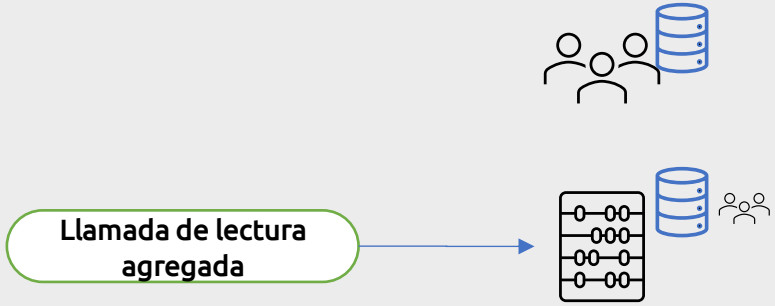
Particionamiento del dato. Lectura del dato en multi-sistemas.

- Llamadas a varios sistemas y un agregado de información
  - **El que llama debe coordinar y agregar datos**
  - **Los tiempos de respuesta se suman**



- CQRS y consolido en mi BBDD lo que necesite
  - **Al escribir se propaga y consolida en todos los sistemas donde haga falta**
  - **Al leer se lee ya agregado en la BBDD local**



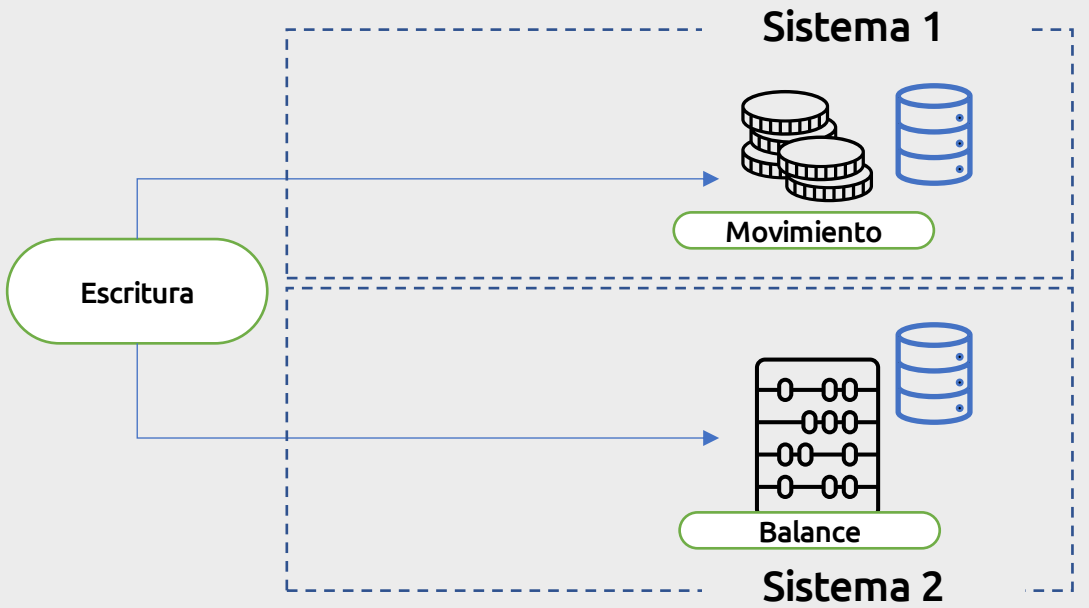


© Capgemini 2025. Todos los derechos reservados |

Retos de los microservicios

Particionamiento del dato. Escritura del dato, transacciones distribuidas.

Problema Dual Write

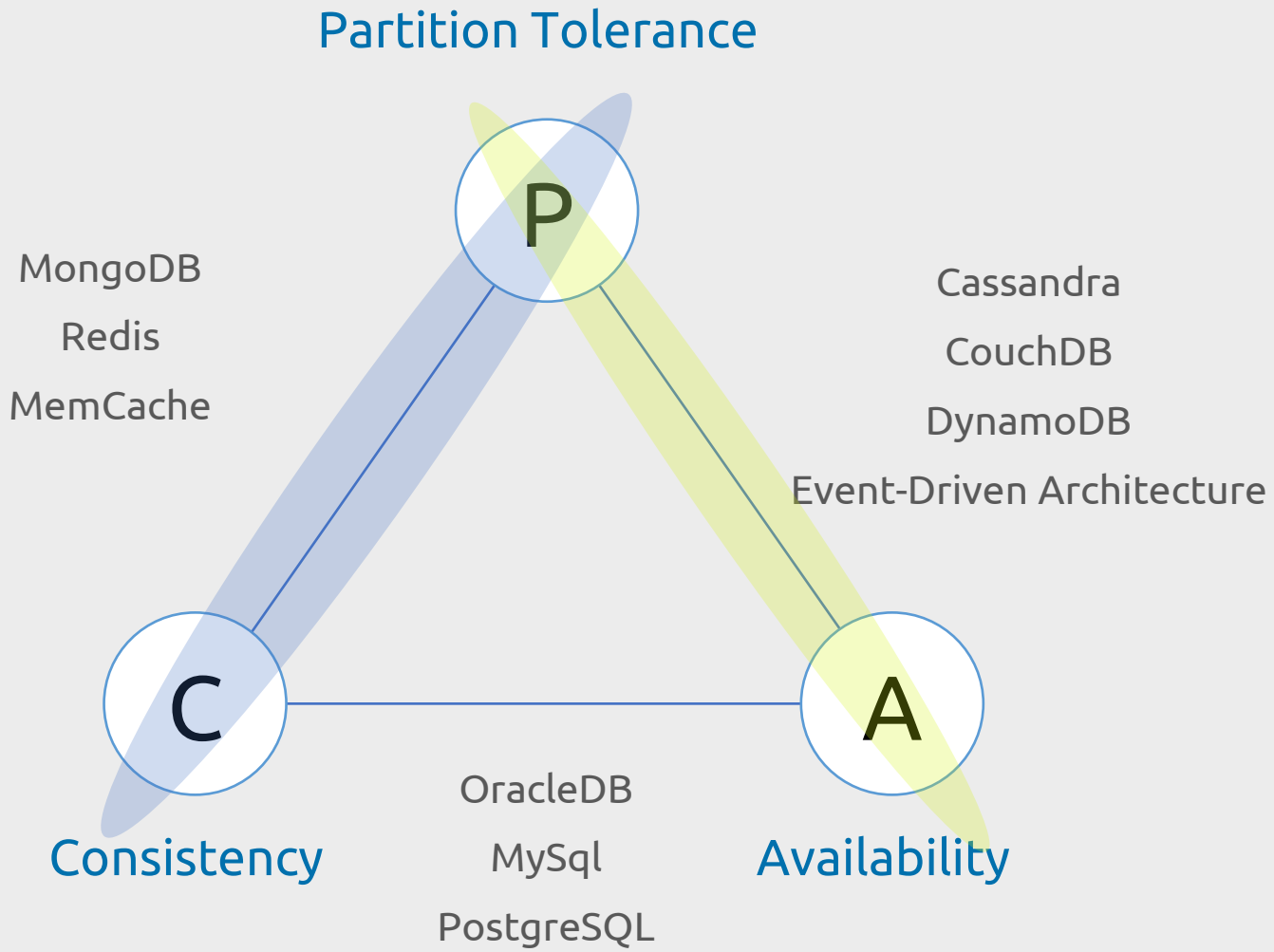


1. Escribo movimiento
2. Persisto movimiento
3. Escribo balance
4. Persisto balance → Falla

© Capgemini 2025. Todos los derechos reservados |

Retos de los microservicios

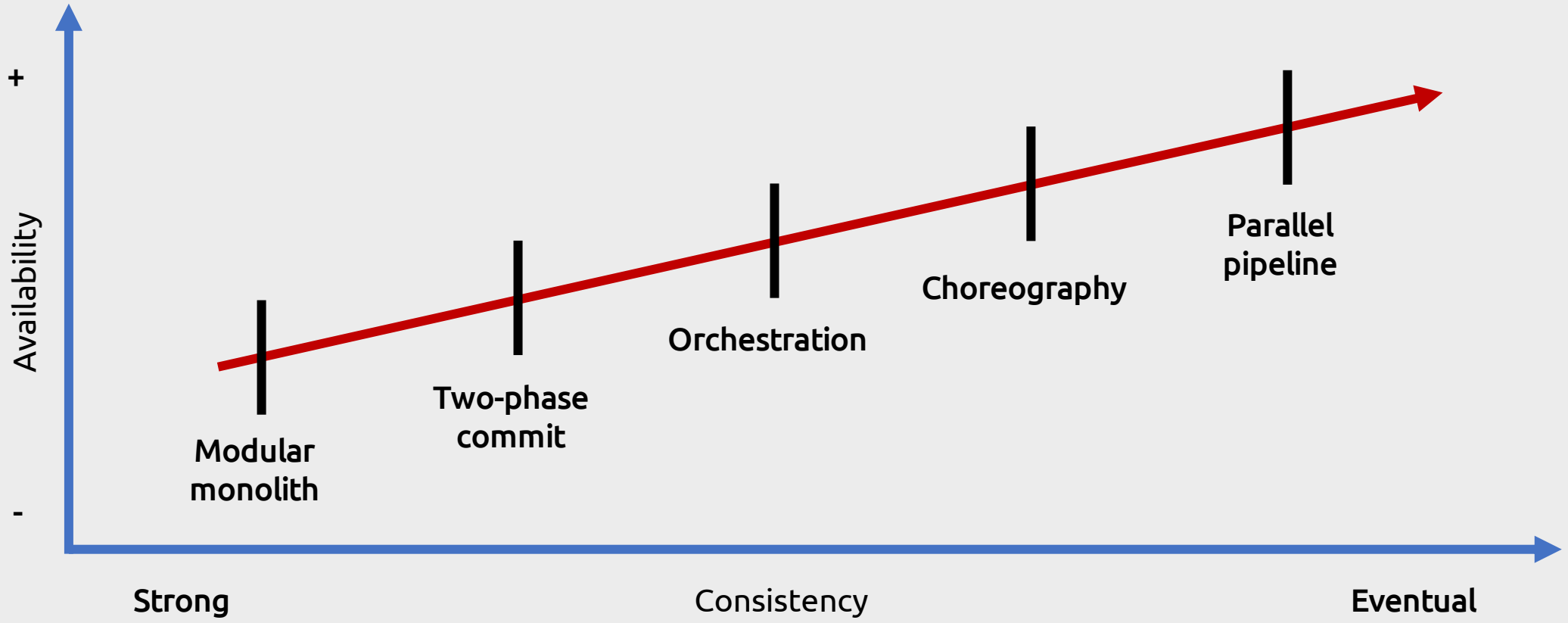
Particionamiento del dato. Teorema de CAP



© Capgemini 2025. Todos los derechos reservados |

Retos de los microservicios

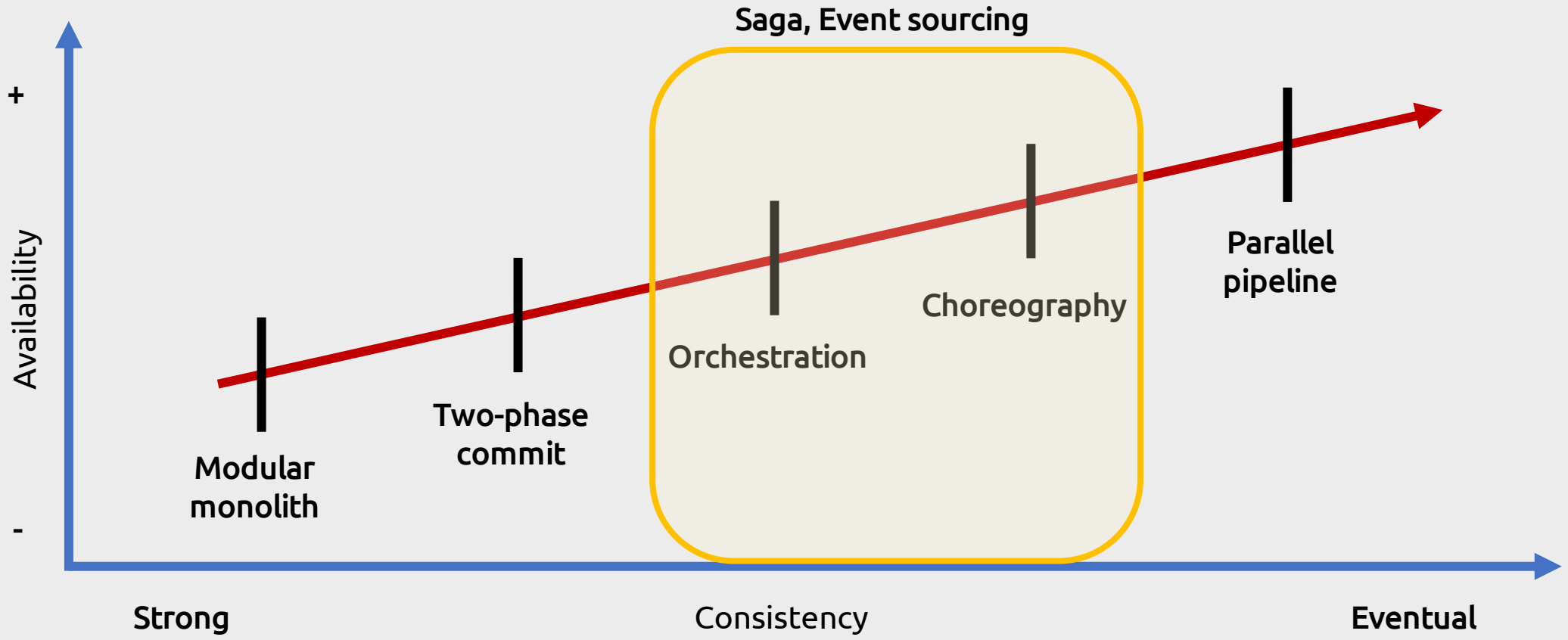
Particionamiento del dato. Escritura del dato, transacciones distribuidas.



© Capgemini 2025. Todos los derechos reservados |

Retos de los microservicios

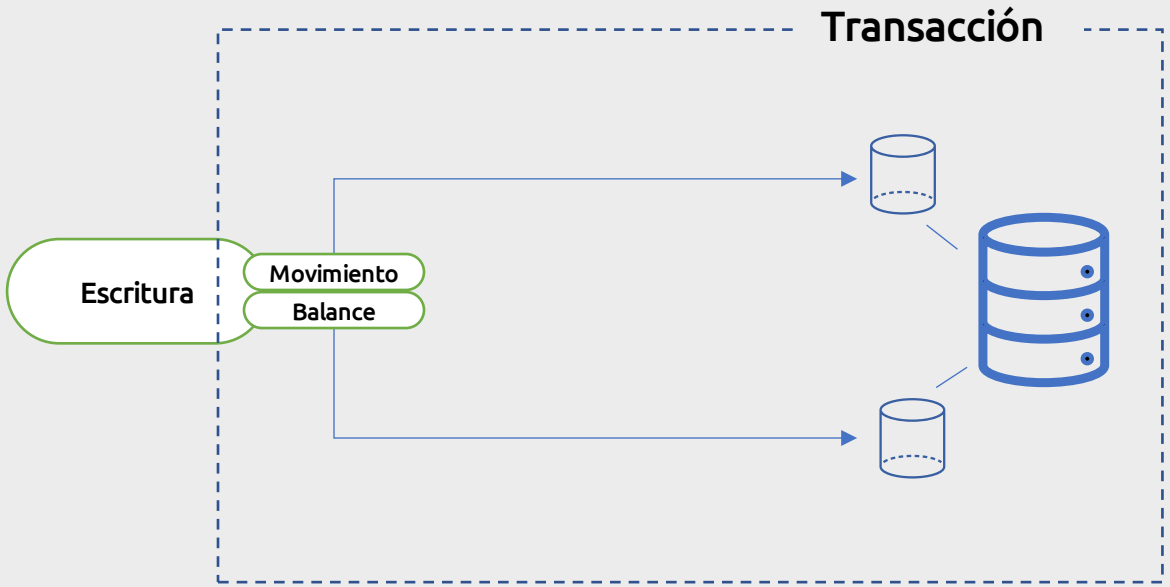
Particionamiento del dato. Escritura del dato, transacciones distribuidas.



© Capgemini 2025. Todos los derechos reservados |

Retos de los microservicios

Transacciones distribuidas - Modular Monolith

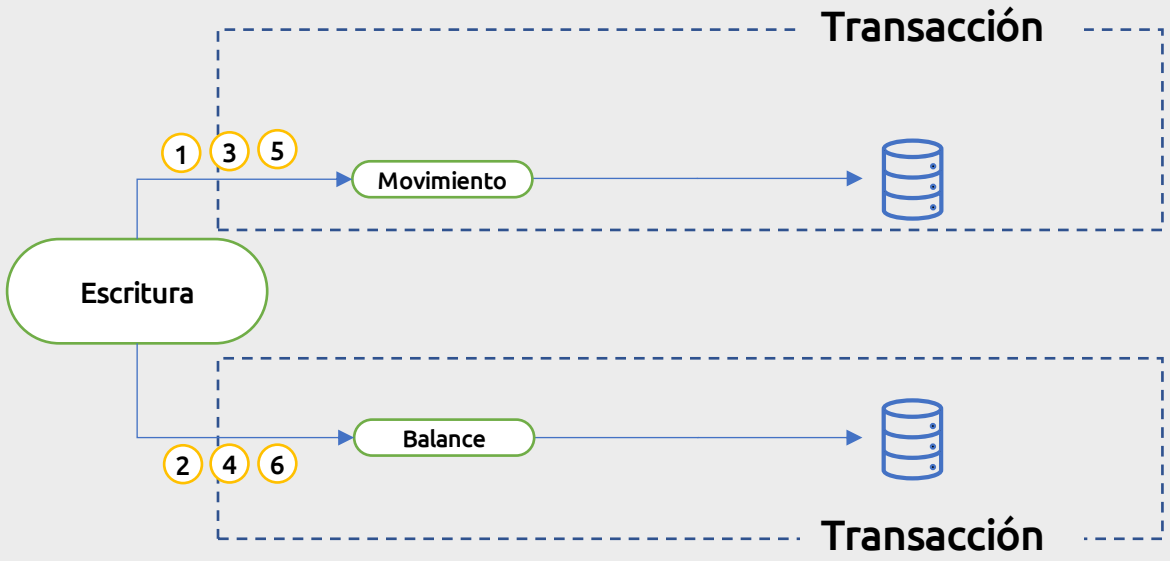


- Esquemas dentro de la misma BBDD
- Ejecución de la transacción dentro del mismo runtime / servicio
- Cuando prima más la consistencia de datos y el acoplamiento de servicios, a la escalabilidad del módulo
- **No es malo tener monolitos / microlitos en tu arquitectura**, pueden convivir con microservicios

© Capgemini 2025. Todos los derechos reservados |

Retos de los microservicios

Transacciones distribuidas - Two-phase commit



1. Write
2. Write

3. Can commit?
4. Can commit?

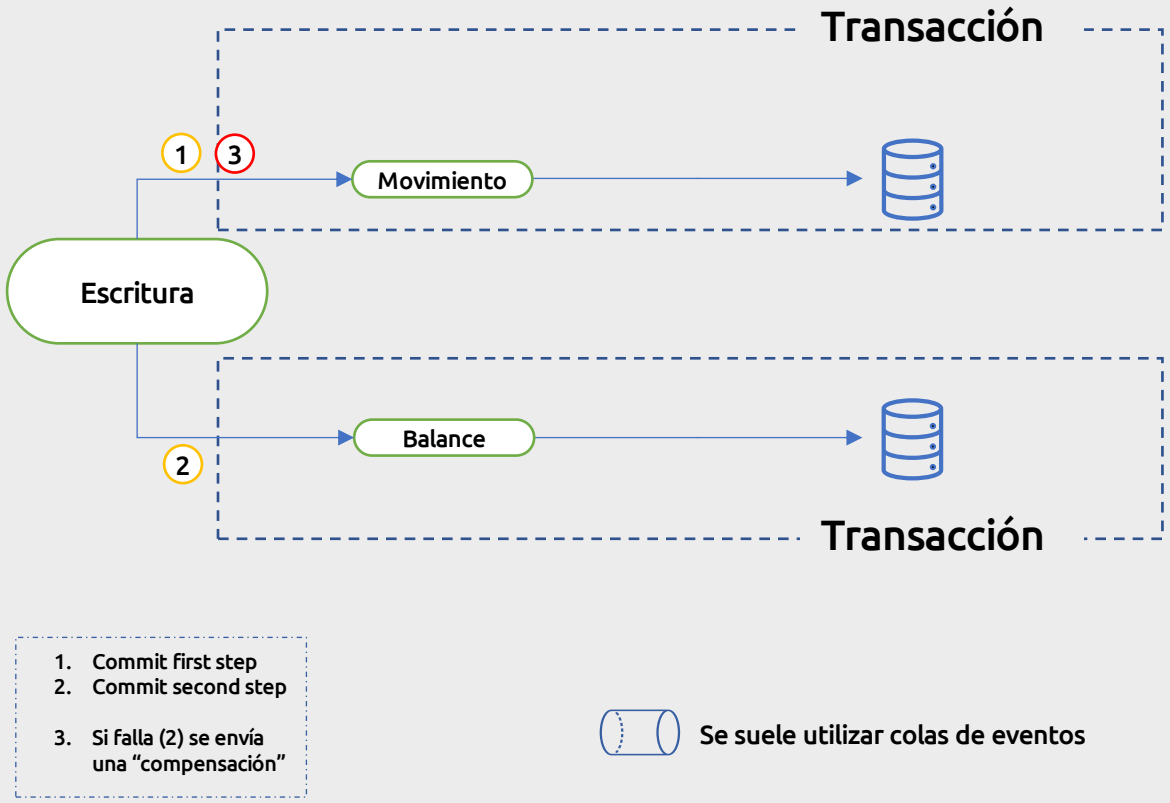
5. Commit
6. Commit

- BBDD físicamente separadas
- Un agente coordinador con capacidad de transacciones distribuidas - XA
- La BBDD debe soportar este tipo de transacciones XA
- Hay muchas librerías que gestionan este tipo de transacciones
- Necesitamos un coordinador y un bloqueo de datos durante un lapso de tiempo
- Cuando prima más la consistencia de datos y el acoplamiento de servicios, a la escalabilidad del módulo

© Capgemini 2025. Todos los derechos reservados |

Retos de los microservicios

Transacciones distribuidas - Orchestration

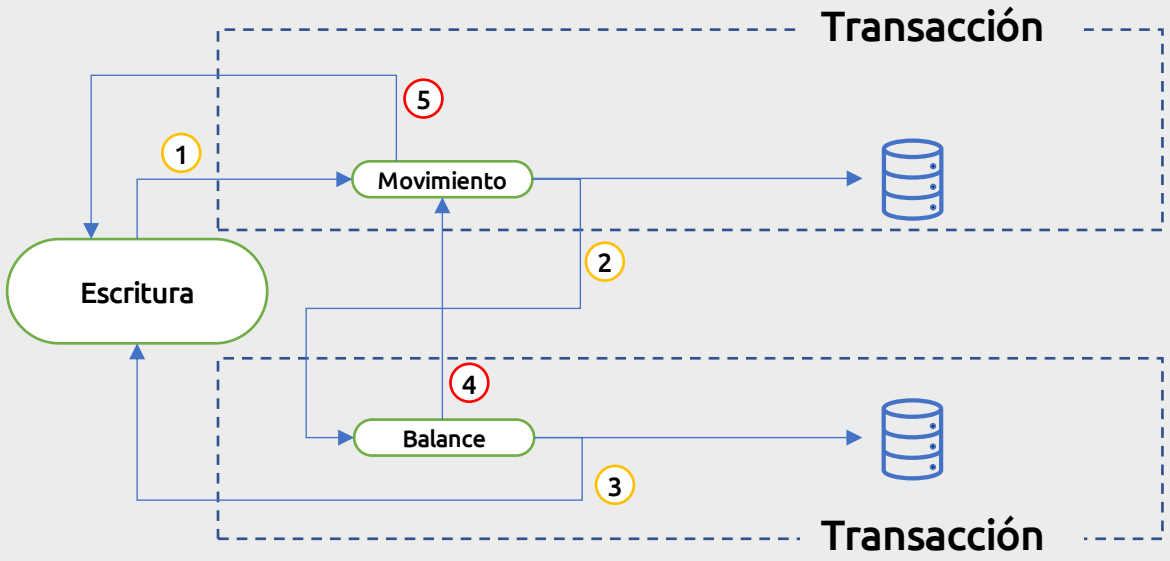


- BBDD físicamente separadas y sin XA
- Existe un agente coordinador de los steps
  - Si un step es OK el agente lanza el siguiente
  - Si un step es KO, lanzará una acción de compensación en los anteriores steps
  - Cuando todo termina OK / KO, cierra la transacción original
- El sistema es menos consistente, hay momentos en los que se está propagando la información.
- Para conocer el estado de la "transacción" puedes preguntar al agente coordinador, él conoce el estado global

© Capgemini 2025. Todos los derechos reservados |

Retos de los microservicios

Transacciones distribuidas - Choreography



1. Commit first step
2. Commit next step
3. Finish process

4. Si falla (2) se envía una "compensación" al anterior step
5. Al fallar (2) se envía una "compensación" hacia atrás

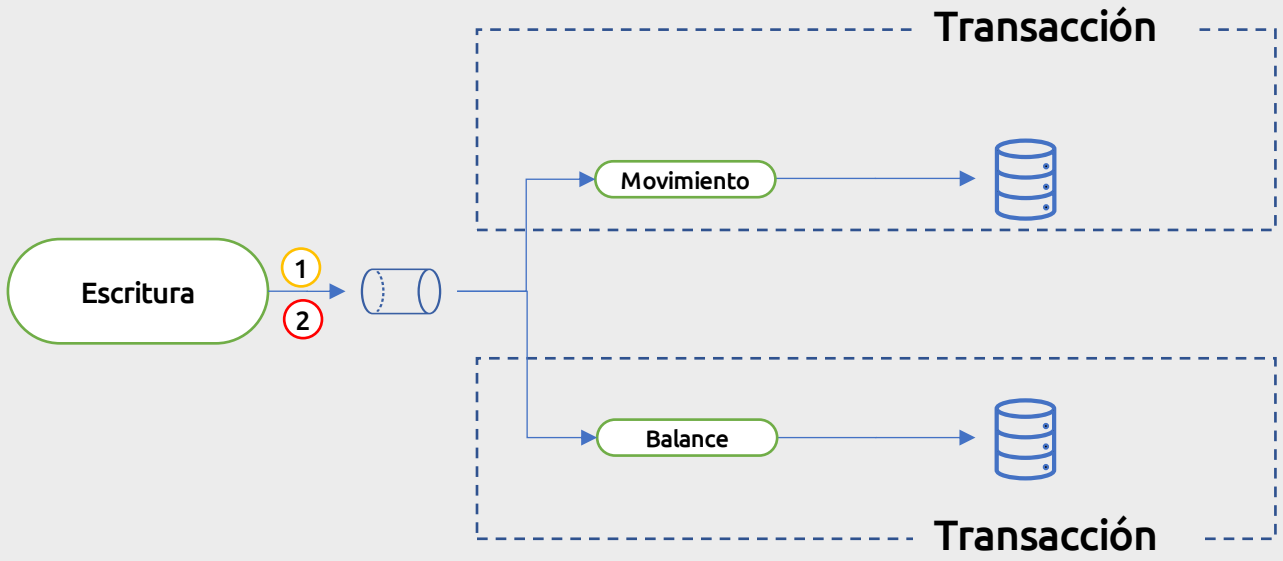
Se suele utilizar colas de eventos

- BBDD físicamente separadas y sin XA
- NO existe un agente coordinador de los steps, se coordinan entre ellos
  - Cada step conoce al siguiente y al anterior, la lógica está distribuida
  - Si un step es OK, este lanza el siguiente
  - Si un step es KO, este lanza compensación al anterior
  - El último step cierra la transacción original
- El sistema es mucho menos consistente, hay momentos en los que se está propagando la información hacia atrás o hacia delante.
- **No es posible conocer el estado ni alcance de la "transacción" a menos que mires en todas.** Es muy difícil de trazar un error.

© Capgemini 2025. Todos los derechos reservados |

Retos de los microservicios

Transacciones distribuidas - Parallel pipeline



1. Commit all steps
2. Si falla algo, se envía acción de "compensación" a todos los steps

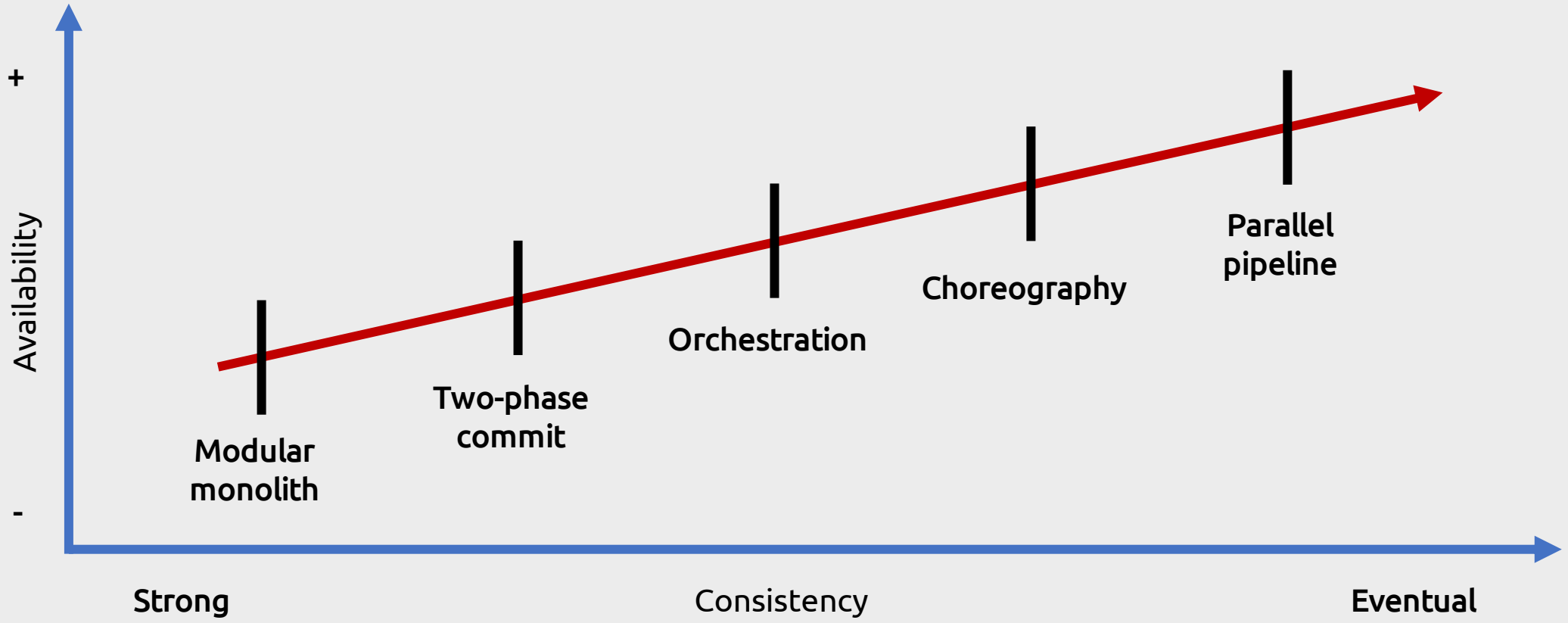
Se deben utilizar colas de eventos

- BBDD físicamente separadas y sin XA.
- NO existe un agente coordinador de los steps, se lanzan los procesos en paralelo.
- Solo se puede usar si hay desacoplamiento temporal (los procesos no dependen unos de otros).
- La arquitectura y la lógica de orquestación es más simple, la consistencia del dato es muy baja pero es muy escalable y paralelizable.
- No es posible conocer el estado de la "transacción" a menos que preguntes a todos. Es muy difícil de trazar un error.

© Capgemini 2025. Todos los derechos reservados |

Retos de los microservicios

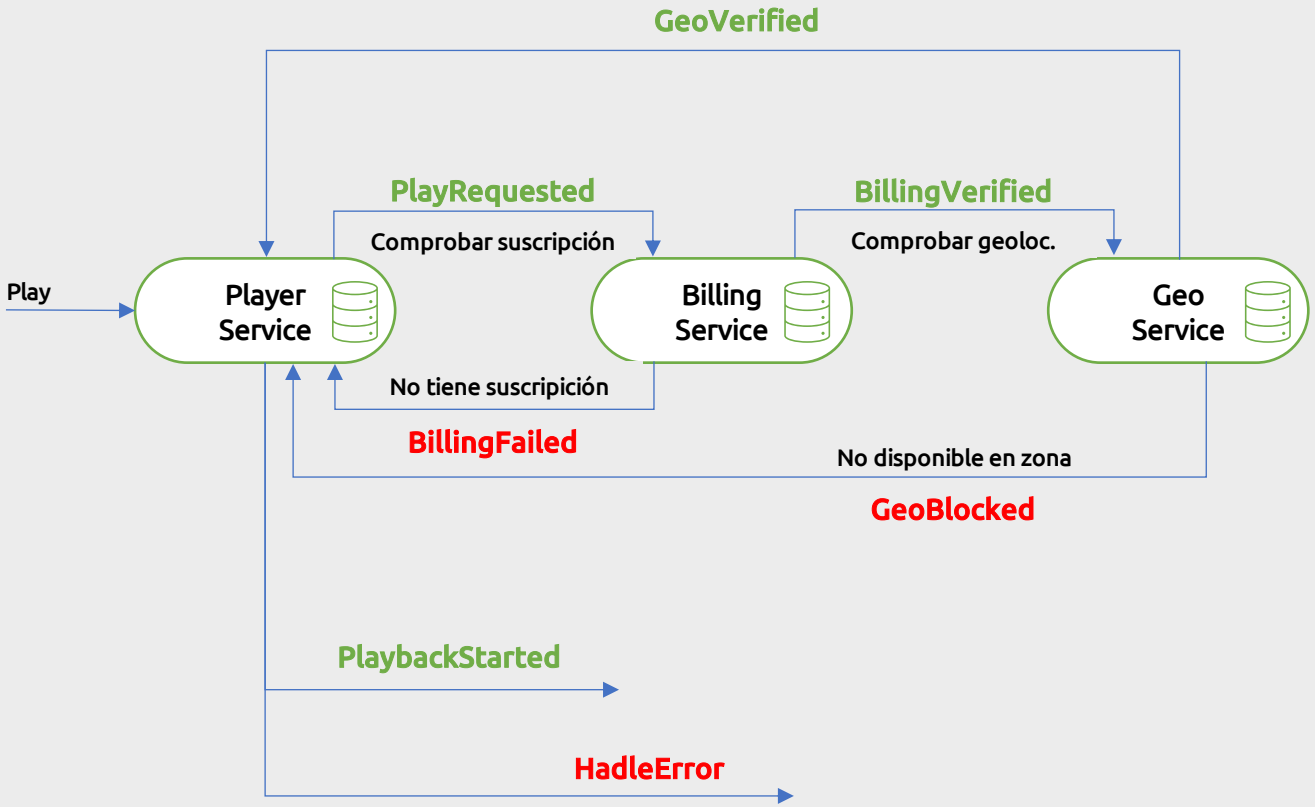
Particionamiento del dato. Escritura del dato, transacciones distribuidas.



© Capgemini 2025. Todos los derechos reservados |

Retos de los microservicios

Particionamiento del dato. Choreography en Netflix.



© Capgemini 2025. Todos los derechos reservados |

Retos de los microservicios

Principales problemas técnicos / funcionales que nos podemos encontrar

- Fallo en la comunicación síncrona – vía API Rest
  - **Hay muchísimas llamadas entre microservicios, debemos prepararnos para lo peor**
- Sincronización de información entre sistemas
  - **En muchas ocasiones queremos mantener una sincronización de datos/estados entre varios sistemas, Event Driven Architecture.**
- Particionamiento del dato (lectura de dato, escritura de dato)
  - **Ojo que ahora tenemos múltiples microservicios con múltiples orígenes de datos**
  - **¿Cómo particionamos el dato? ¿Quién es el propietario? ¿Cómo creamos transacciones distribuidas?**

© Capgemini 2025. Todos los derechos reservados |

Ahora a jugar…

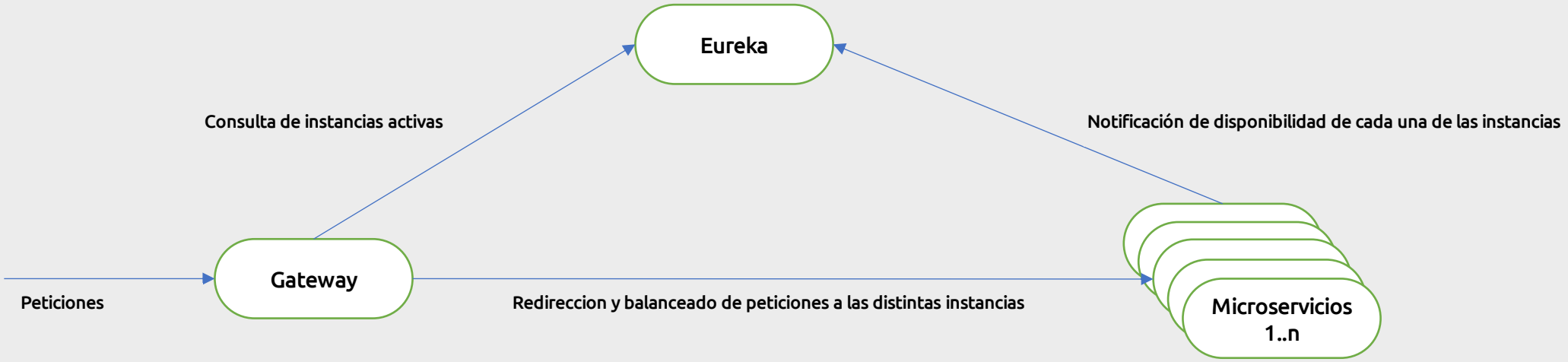
Ejercicio práctico 1

Ecosistema Microservicios

© Capgemini 2025. Todos los derechos reservados |

Ejercicio 1

Ecosistema sencillo de microservicios



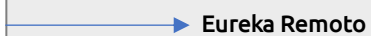
GiHub: https://github.com/ccsw-csd/micro-simple

© Capgemini 2025. Todos los derechos reservados |

Ejercicio 1

Ecosistema sencillo de microservicios - Segunda parte

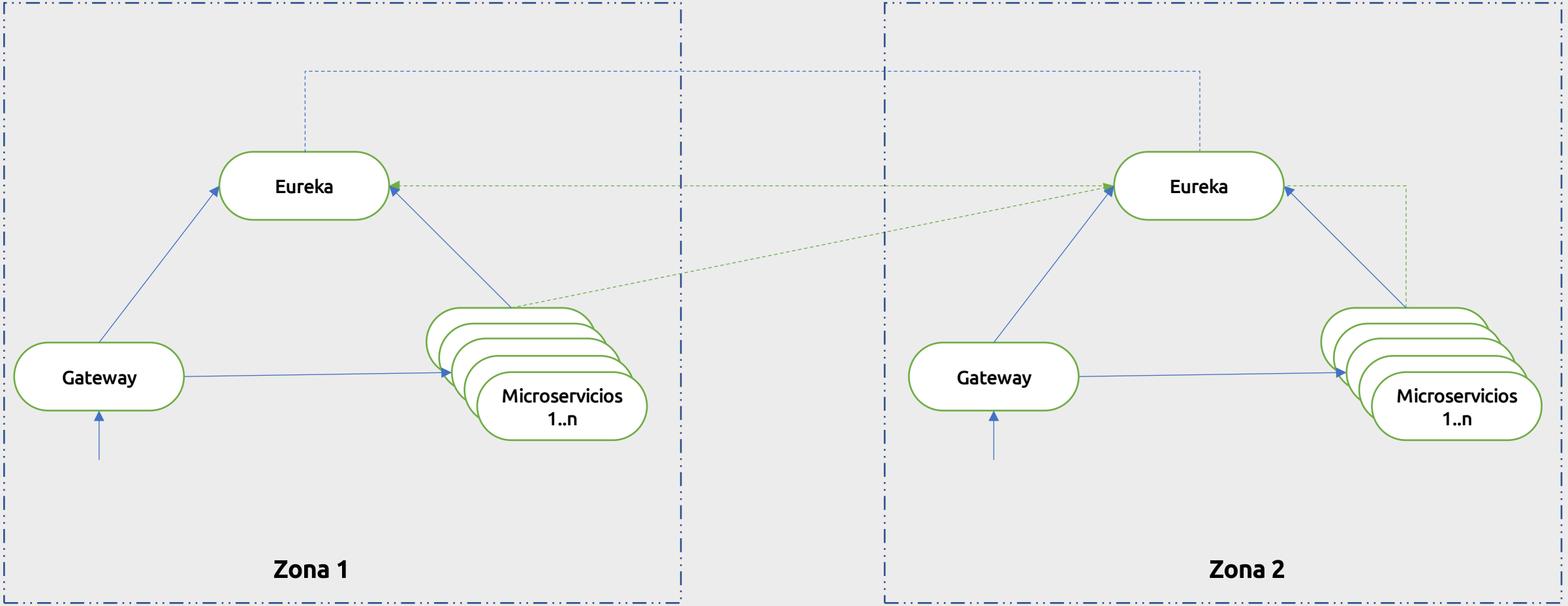
```
spring:
  application:
    name: spring-cloud-eureka-client
server:
  port: 8090
eureka:
  client:
    serviceUrl:
      defaultZone: ${EUREKA_URI:http://frparccsw:8761/eureka }
  instance:
    preferIpAddress: true
```



© Capgemini 2025. Todos los derechos reservados |

Ejercicio 1

Alta disponibilidad – Zonas de disponibilidad



© Capgemini 2025. Todos los derechos reservados |

Ahora a jugar…

Ejercicio práctico 2

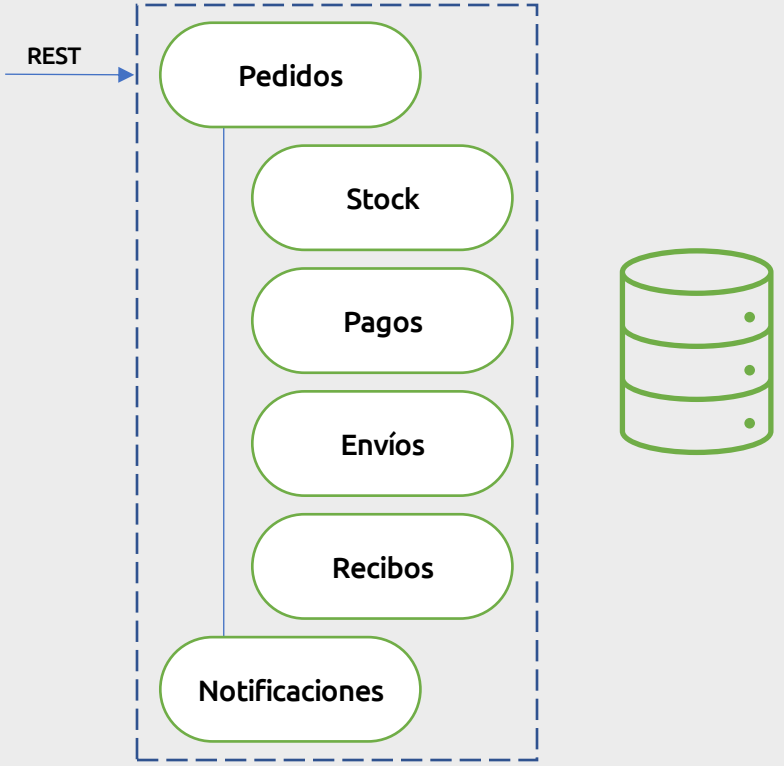
Transacciones distribuidas

© Capgemini 2025. Todos los derechos reservados |

Carrito de compra

Ejemplo de una transacción de compra - Monolito

- Al hacer una compra online se producen los siguientes estados:
  - Se registra el pedido
  - Se comprueba que hay stock suficiente
  - Se realiza el pago/cobro
  - Se valida y registra el envío
  - Se genera la factura
  - Se notifica la compra completada o errónea
- En un sistema monolítico la transacción es única, se validan todos los pasos, se escriben en las tablas correspondientes y se realiza un commit único y atómico.
- Pero… ¿y si queremos realizar un escalado de algunas de las piezas? ¿y si queremos utilizar microservicios?



© Capgemini 2025. Todos los derechos reservados |

Carrito de compra

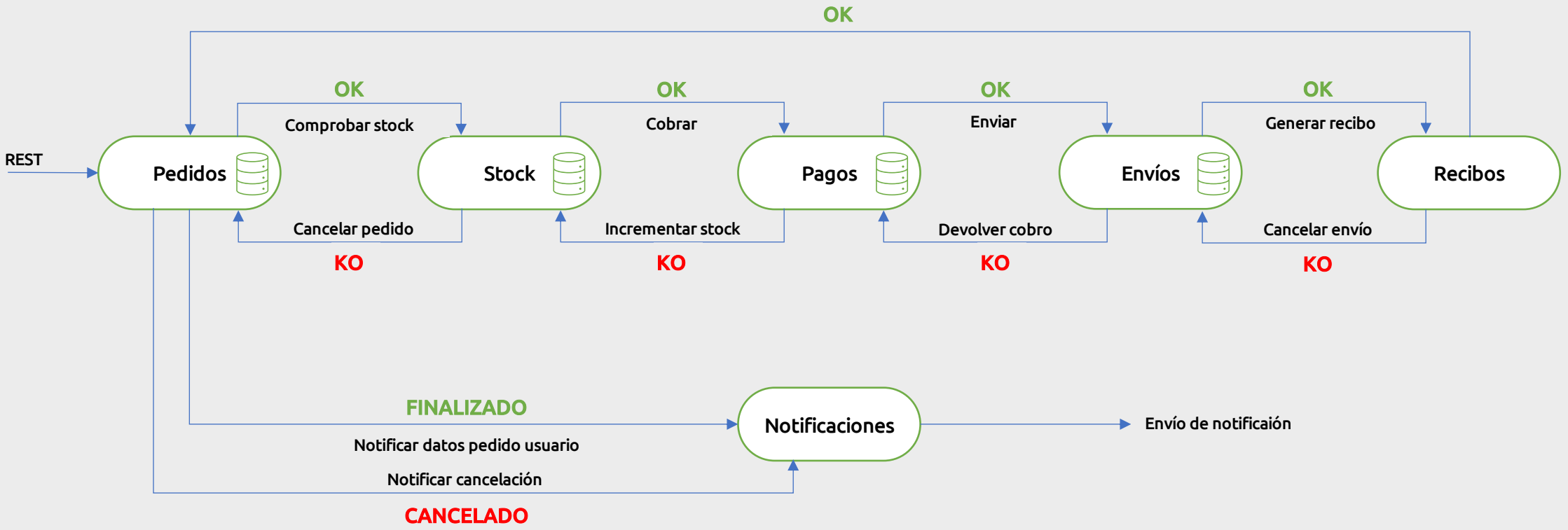
Ejemplo de una transacción de compra - Microservicios

- En un sistema de microservicios idealmente cada microservicio se encarga de un ámbito funcional (pedidos, stock de producto, pagos, envíos logísticos, recibos / facturación, avisos / notificaciones).
- Cada microservicio puede estar o no escalado, y estar implementado en diferentes lenguajes. Además, generalmente cada microservicio escribe en su propia fuente de datos.
- Entonces, cuando hago una compra online con este sistema, ¿como gestiono las llamadas entre servicios?
- Y peor aun, si falla por ejemplo la pasarela de pago, pero ya he registrado el pedido y he decrementado el stock ¿qué tengo hago con la transacción en curso? No me sirve hacer un rollback, porque esos sistemas ya hicieron su commit …

© Capgemini 2025. Todos los derechos reservados |

Carrito de compra

Transacciones distribuidas – Choreography con Kafka



© Capgemini 2025. Todos los derechos reservados |

Carrito de compra

Transacciones distribuidas – Choreography con Kafka

team-1

- Moises Rodriguez
- Cristina Diez
- Alejandro Valenzuela
- Celtia Portas
- Marcos De La Fuente
- Alejandro Pinto

team-2

- César Ferrández
- Laura Dinorah Garcia
- Clara Maria Herranz
- Juan Francisco Berenguer
- Sergio Contreras
- María Gracia

team-3

- Ruben Gonzalez
- Cristo Suarez
- Rocio Nogales
- Noelia Cremades
- Jorge Domínguez
- Angel Nacher

team-4

- Ana Valero
- Enrique Fletes
- Iván Risueño
- Mónica Rodríguez
- Jose Luis Huelva
- Cristina Lopez
- Marina Isabel Martínez

© Capgemini 2025. Todos los derechos reservados |

Carrito de compra

Microservicios

Pedidos – ShopOrder (shop_order)

- Endpoint recepción pedido (REST)
  - Recibe cliente, email, dirección, tarjeta de crédito, producto y cantidad
  - Generar identificador único de pedido (UUID)
  - Crear pedido en BBDD (Tabla de pedidos)
  - Asignar id del grupo
  - Realizar reserva producto (Enviar a **Stock**)
- Finalizar pedido
  - Actualizar pedido en BBDD
  - Notificar finalización (Enviar a **Notificaciones**)
- Cancelar pedido
  - Borrar pedido en BBDD
  - Notificar cancelación (Enviar a **Notificaciones**)

Stock – Stock (stock)

- Tabla de productos (producto, stock, precio)
- Crear productos
- Realizar reserva de producto
  - Comprobar stock (Si no hay suficiente o el producto no existe, Cancelar pedido)
  - Decrementar stock
  - Informar precio
  - Realizar pago (Enviar a **Pagos**)
- Cancelar reserva producto
  - Incrementar stock
  - Cancelar pedido (Enviar a **Pedidos**)

© Capgemini 2025. Todos los derechos reservados |

Carrito de compra

Microservicios

Pagos – Payment (payment)

- Tabla de pagos (UUID, cliente, tarjeta, total)
- Realizar pago de pedido
  - Comprobar tarjeta de crédito (Si no hay tarjeta de crédito, Cancelar reserva producto)
  - Calcular pericio total a pagar
  - Almacenar pago en la BBDD
  - Informar importe pagado
  - Realizar envió (Enviar a **Envíos**)
- Cancelar pago de pedido
  - Eliminar pago en la BBDD
  - Cancelar reserva producto (Enviar a **Stock**)

Envíos – Shipment (shipment)

- Tabla de envíos (UUID, cliente, dirección, fecha)
- Realizar envío de producto
  - Comprobar dirección (Si no hay dirección de envío, Cancelar pago pedido)
  - Calcular fecha envío como el momento de la petición
  - Almacenar envío en la BBDD
  - Informar fecha envío
  - Generar recibo (Enviar a **Recibos**)
- Cancelar envío producto
  - Eliminar envío de la BBDD
  - Cancelar pago pedido (Enviar a **Pagos**)

© Capgemini 2025. Todos los derechos reservados |

Carrito de compra

Microservicios

Recibos – Invoice (invoice)

- Generar recibo
  - Generar recibo (UUID + Cliente + Producto)
  - Comprobar recibo
  - Si la longitud de recibo supera 60 caracteres, Cancelar envío producto (Enviar a **Envíos**)
  - Informar el recibo generado
  - Finalizar pedido (Enviar a **Pedidos**)

Notificaciones – Notification (notification)

- Envío de notificación de confirmación
  - Crear cuerpo del mensaje con el recibo generado
  - Muestra en consola
- Envío de notificación de cancelación
  - Crear cuerpo del mensaje con el UUID del pedido eliminado
  - Obtener motivo de cancelación
  - Muestra en consola

© Capgemini 2025. Todos los derechos reservados |

Carrito de compra

Mensaje

```java
public class ShopOrderRequest {

    private String groupId;

    private Boolean success;

    private ShopOrderDataRequest data;
}
```



```java
public class ShopOrderDataRequest {

    private String uuid;

    private String customer;

    private String email;

    private String address;

    private String credit;

    private String product;

    private Integer quantity;

    private Double price;

    private Double paid;

    private String shipment;

    private String invoice;

}
```

© Capgemini 2025. Todos los derechos reservados |

Carrito de compra

Consumer

```
@Component
public class KafkaConsumer {

  @Value("${spring.kafka.consumer.group-id}")
  private String groupId;

  ObjectMapper mapper = new ObjectMapper();

  @KafkaListener(topics = "XXX")
  public void listener(String message) {

    try {
      System.out.println("Message has been received: " + message);
      ShopOrderRequest request = mapper.readValue(message, ShopOrderRequest.class);

      if(groupId.equals(request.getGroupId())) {
        //TODO: Funcionalidad
      }

    } catch (JsonProcessingException e) {
      System.out.println("Error parsing request");
    }
  }

}
```

© Capgemini 2025. Todos los derechos reservados |

Carrito de compra

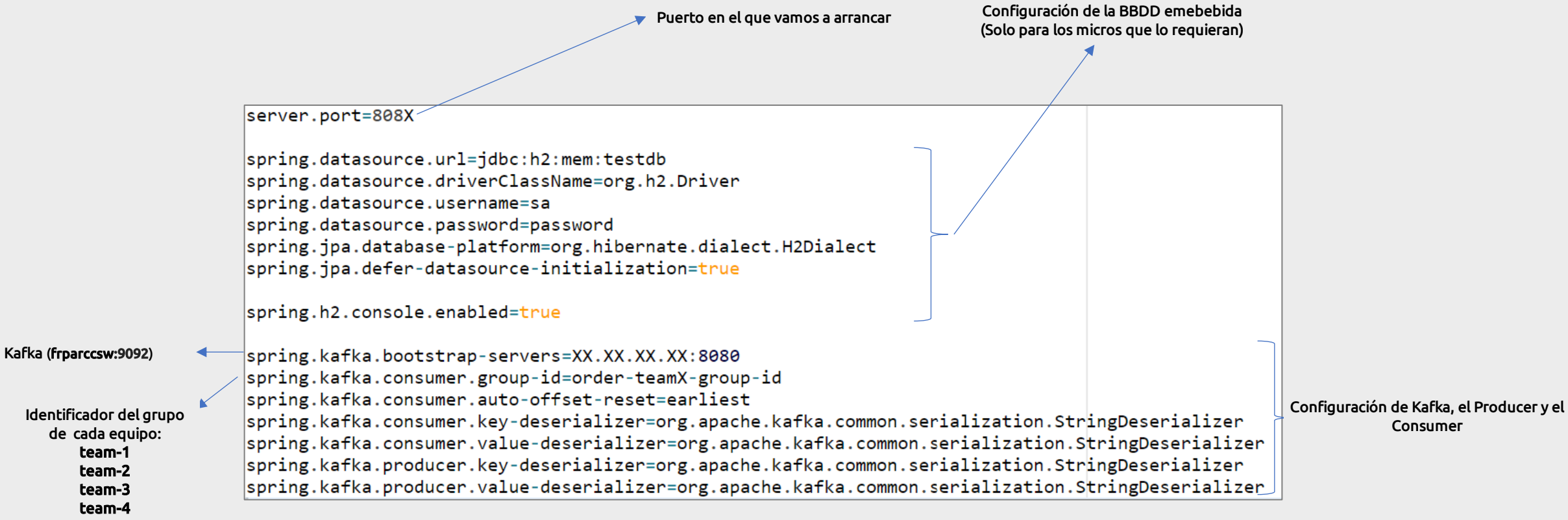
Producer

```
@Component
public class KafkaProducer {

  @Autowired
  private KafkaTemplate<String, String> kafkaTemplate;

  ObjectMapper mapper = new ObjectMapper();

  public void sendMessage(String topic, ShopOrderRequest request) {

    try {
      String message = mapper.writeValueAsString(request);

      CompletableFuture<SendResult<String, String>> future = kafkaTemplate.send(topic, message);

      future.whenComplete((result, ex) -> {
        if (ex == null) {
          System.out.println("Message has been sent: " + message);
        } else {
          System.out.println("Something went wrong with the message: " +  message);
        }
      });

    } catch (JsonProcessingException e) {
      System.out.println("Error parsing request");
    }
  }
}
```

© Capgemini 2025. Todos los derechos reservados |

Carrito de compra

Propiedades

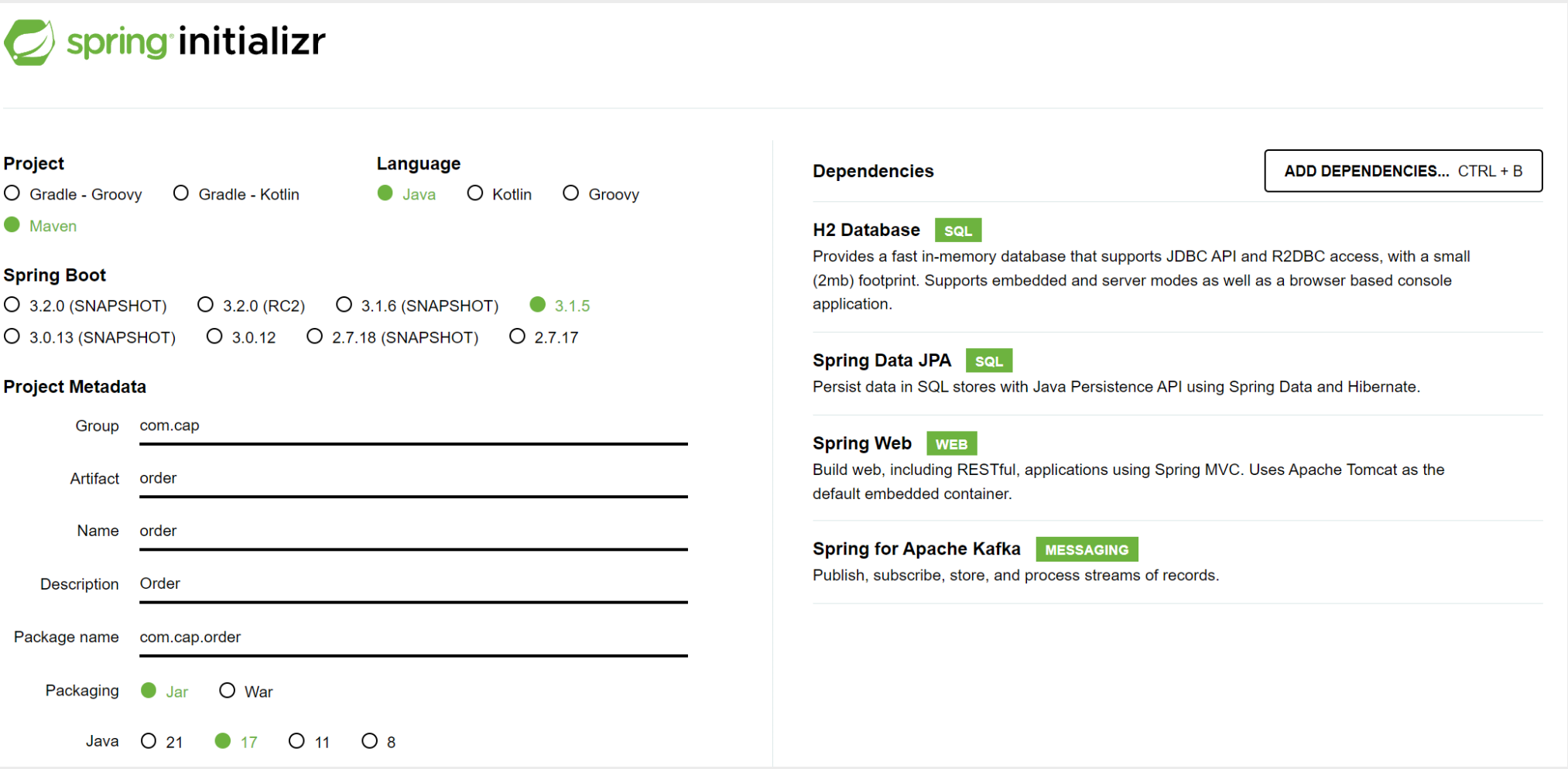


GiHub: https://github.com/ccsw-csd/shop-cart-public

© Capgemini 2025. Todos los derechos reservados |

Carrito de compra

Creación microservicio



© Capgemini 2025. Todos los derechos reservados |

Valora la sesión

Ayúdanos a mejorar

- qué te ha parecido
- qué te ha costado de entender
- qué te hemos aclarado
- cualquier sugerencia

© Capgemini 2025. Todos los derechos reservados |

© Capgemini 2025. Todos los derechos reservados |