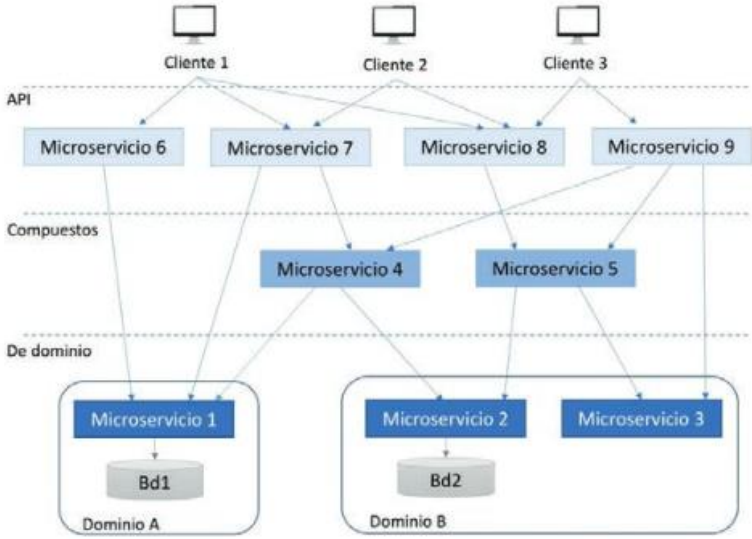


Figura 2.1. Tipos de microservicios

Tema 2 Principios de Diseño de Microservicios

Indice

- Descomposición de aplicaciones en servicios.
- Diseño de servicios autónomos y cohesivos.
- Principios de responsabilidad única y separación de preocupaciones.
- Patrones de diseño comunes en microservicios.

Descomposición de Aplicaciones en Servicios

Análisis del Dominio:

Identificación de los dominios de negocio y subdominios.

Uso de técnicas como Domain-Driven Design (DDD) para modelar el dominio.

Contextos delimitados: definición y aplicación.

Estrategias de Descomposición:

Descomposición por capacidad de negocio.

Descomposición por subdominio.

Descomposición por flujo de trabajo.

Descomposición por verbo (acciones)

Granularidad del Servicio

Equilibrio entre servicios pequeños y grandes.

Consideraciones sobre el acoplamiento y la cohesión.

Evitar el "microservicio atómico" y el "monolito distribuido"

Diseño de Servicios Autónomos y Cohesivos

Autonomía del Servicio:

- Independencia en el despliegue y la escalabilidad.
- Gestión de datos propia (base de datos por servicio).
- Comunicación a través de APIs bien definidas.

Cohesión del Servicio

- Agrupación de funcionalidades relacionadas dentro de un servicio.
- Minimización de las dependencias externas.
- Que el servicio tenga un unico motivo para cambiar.

Acoplamiento Débil

- Reducción de las dependencias entre servicios.
- Uso de APIs y contratos bien definidos.
- Comunicación asíncrona para desacoplar servicios.

Principios de Responsabilidad Única y Separación de Preocupaciones

Responsabilidad Única (SRP)

Cada servicio debe tener una única razón para cambiar.

Evitar la sobrecarga de funcionalidades en un solo servicio.

Separación de Preocupaciones

Aislamiento de diferentes aspectos funcionales en servicios separados.

Mejora de la mantenibilidad y la escalabilidad.

Ejemplo, un microservicio de autentificación, no deberia tener la responsabilidad de la logica de negocio de pedidos.

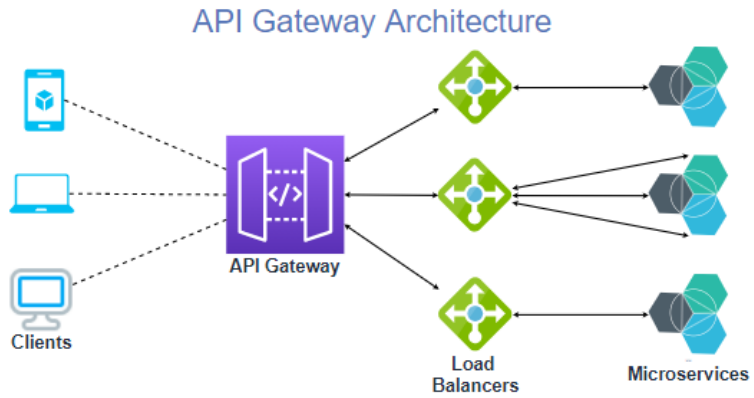
Patrones de Diseño Comunes en Microservicios

API Gateway

Punto de entrada único para los clientes.

Gestión de rutas, autenticación y autorización.

Agregación de respuestas de múltiples servicios.



API Gateway

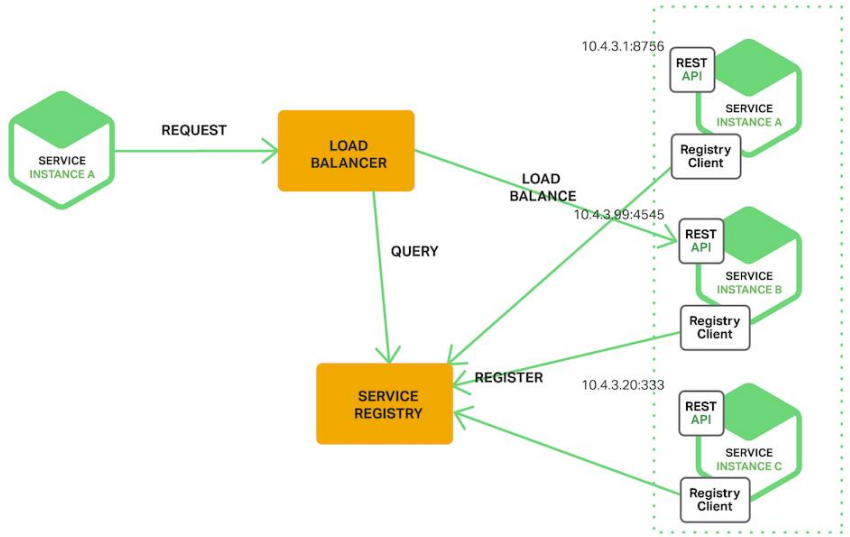
- **Escenario:** Una aplicación de comercio electrónico con microservicios para productos, pedidos y clientes.
- **Ejemplo:**
  - Un API Gateway recibe todas las solicitudes de los clientes (web, móvil).
  - Si la solicitud es "/productos", el Gateway la enruta al microservicio de productos.
  - Si es "/pedidos", la enruta al microservicio de pedidos.
  - El Gateway también puede manejar la autenticación, autorización y agregación de datos de múltiples microservicios.
  - De esta manera el cliente solo se comunica con un solo punto de entrada.

Service Registry/Discovery

Registro y descubrimiento dinámico de servicios.

Facilitación de la comunicación entre servicios.

Ejemplos: Eureka, Consul, etcd.

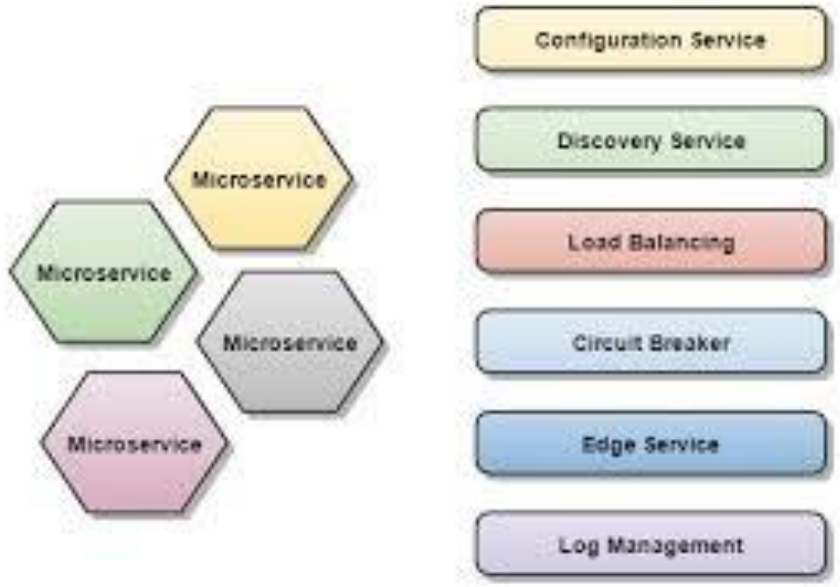


2. Service Registry/Discovery

- **Escenario:** Una aplicación que necesita que los microservicios se descubran dinámicamente.
- **Ejemplo:**
  - El microservicio de pedidos necesita comunicarse con el microservicio de clientes.
  - En lugar de codificar la dirección del microservicio de clientes, el microservicio de pedidos consulta un registro de servicios (como Eureka o Consul).
  - El registro de servicios proporciona la dirección actual del microservicio de clientes, permitiendo que la comunicación sea dinámica y resistente a cambios en la infraestructura.

Circuit Breaker

- Prevención de fallos en cascada.
- Monitoreo de la disponibilidad de los servicios.
- Ejemplo: Hystrix, Resilience4j.

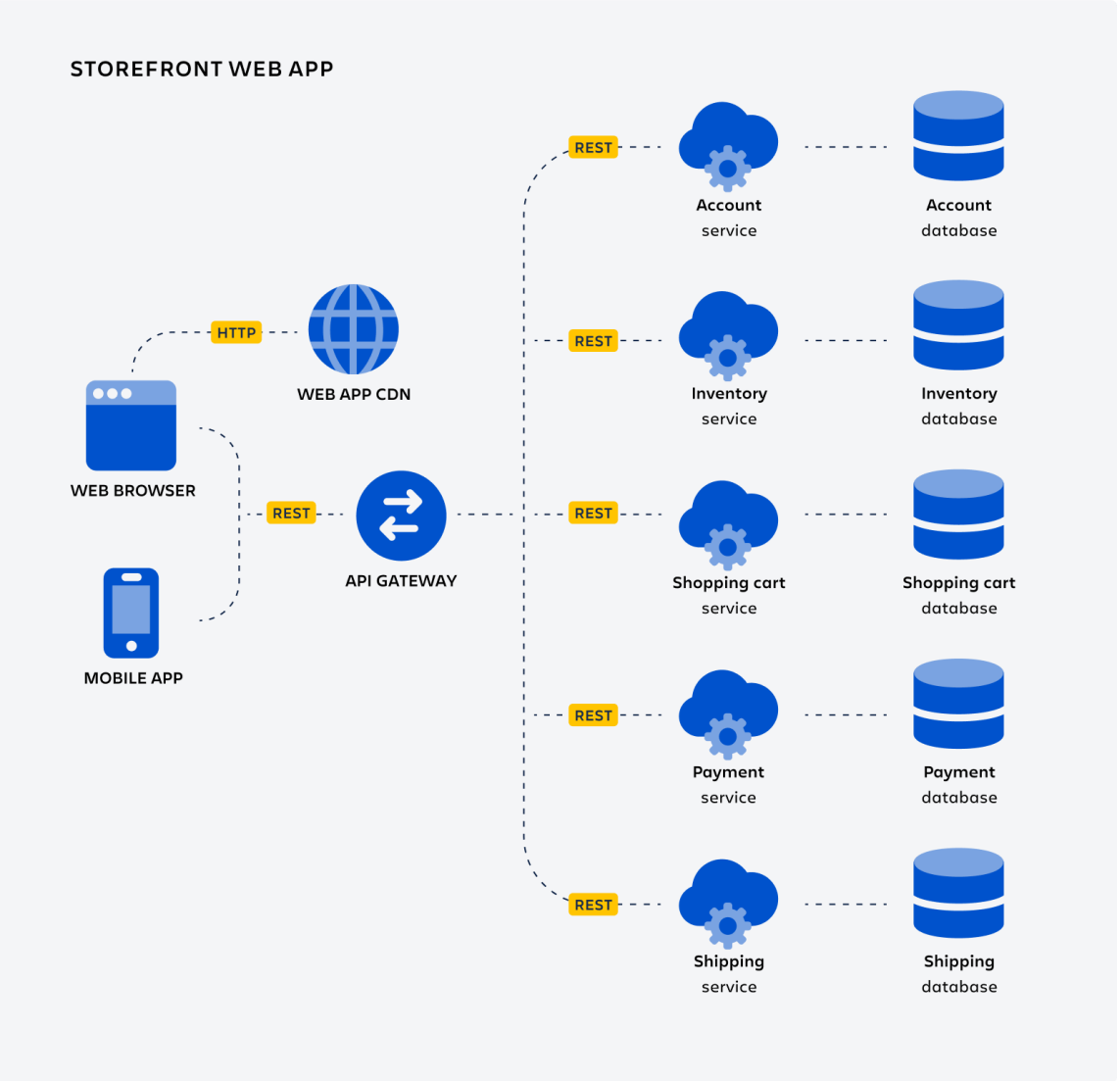


Circuit Breaker

- **Escenario:** Un microservicio de pedidos depende de un microservicio de pagos que a veces falla.
- **Ejemplo:**
  - Si el microservicio de pagos falla repetidamente, el Circuit Breaker en el microservicio de pedidos "abre el circuito".
  - Durante un tiempo, el microservicio de pedidos deja de intentar llamar al microservicio de pagos, evitando sobrecargar un servicio ya fallido.
  - Después de un tiempo, el Circuit Breaker "cierra el circuito" y permite nuevos intentos, pero con precaución.
  - De esta manera se evita el fallo en cascada.

Base de datos por servicio

- Cada microservicio debe tener su propia persistencia de datos, de esta manera se aumenta el desacoplamiento.

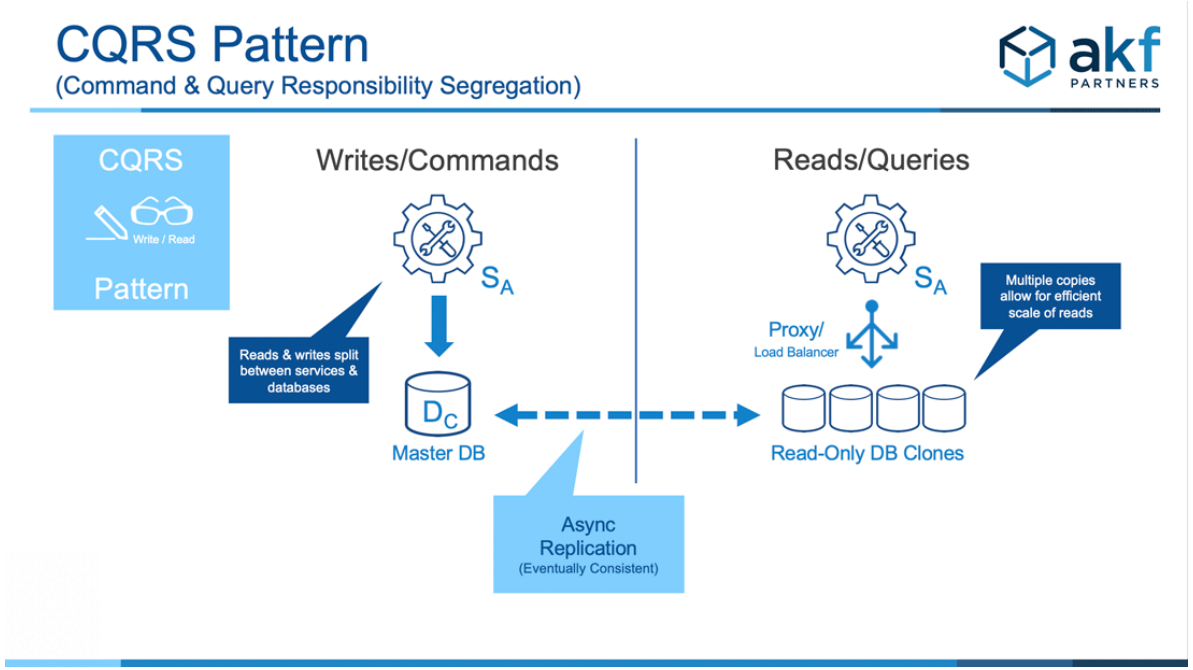


4. Base de datos por servicio

- **Escenario:** Una aplicación de gestión de clientes.
- **Ejemplo:**
  - El microservicio de "clientes" tiene su propia base de datos, donde almacena información sobre los clientes.
  - El microservicio de "pedidos" tiene su propia base de datos, donde almacena información sobre los pedidos.
  - De esta manera cada microservicio tiene el control total de sus datos, y no hay dependencia entre las bases de datos de los microservicios.

CQRS (Command Query Responsability Segregation)

- Separar las operaciones de lectura de las de escritura, de esta manera se optimizan los recursos, y se pueden escalar de manera independiente.



CQRS (Command Query Responsability Segregation)

- **Escenario:** Una aplicación de informes financieros con alta carga de lectura.
- **Ejemplo:**
  - Los comandos (creación, actualización) se manejan por un lado, actualizando una base de datos optimizada para escritura.
  - Las consultas (lectura de informes) se manejan por otro lado, consultando una base de datos optimizada para lectura (posiblemente una réplica de lectura o un almacén de datos separado).
  - De esta manera, se optimizan los recursos para cada tipo de operación, y se mejora el rendimiento general.