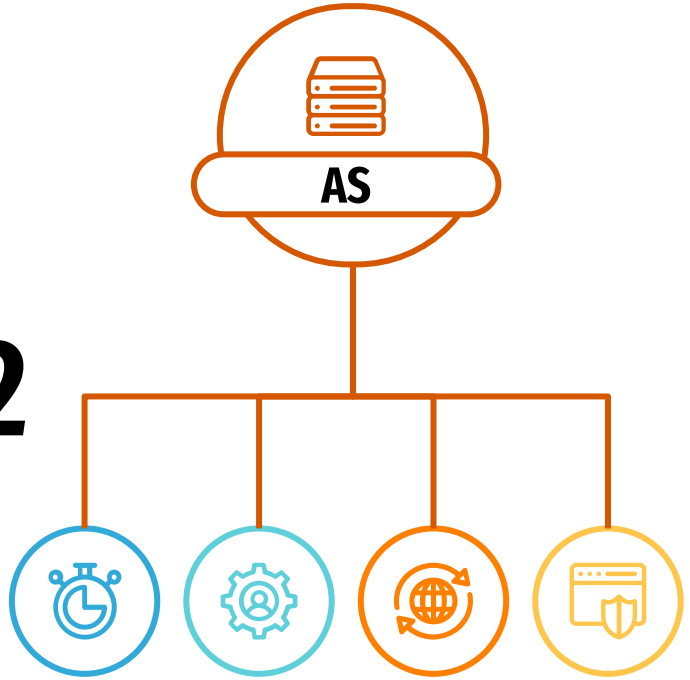
# Estilos Arquitectonicos 2

MSc. Marco Tulio Aldana Prillwitz

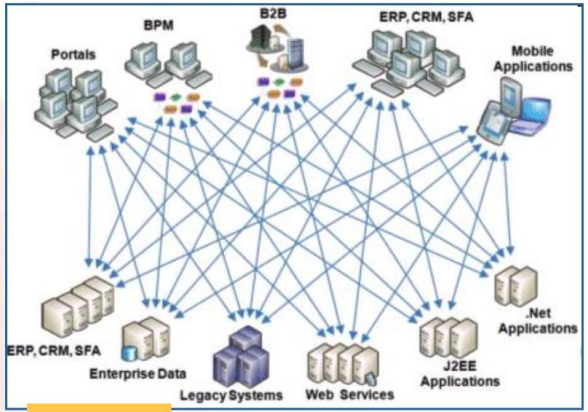


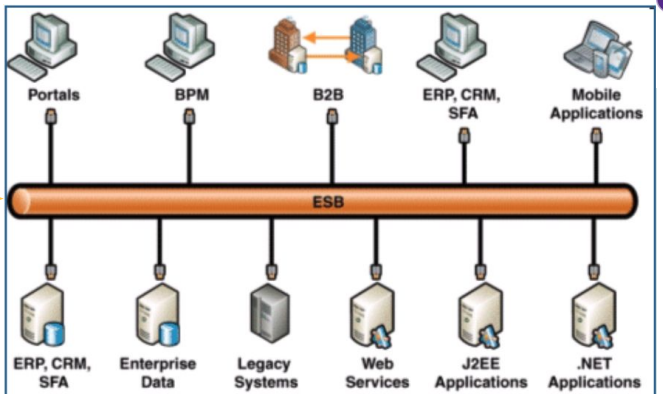

FACULTAD DE INGENIERÍA
UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

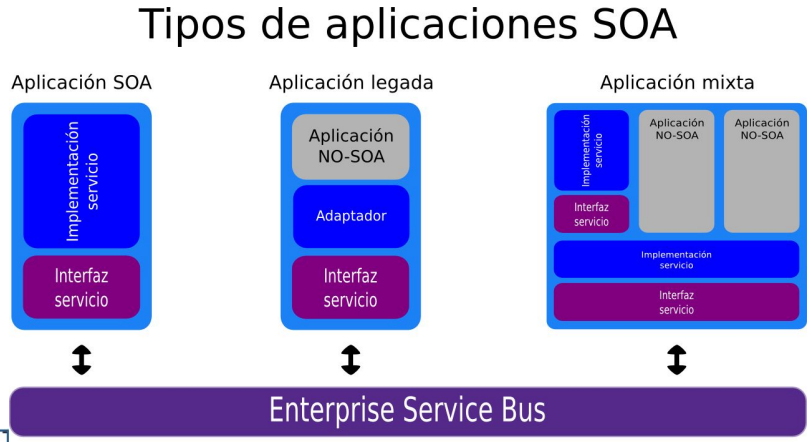
# Integración empresarial

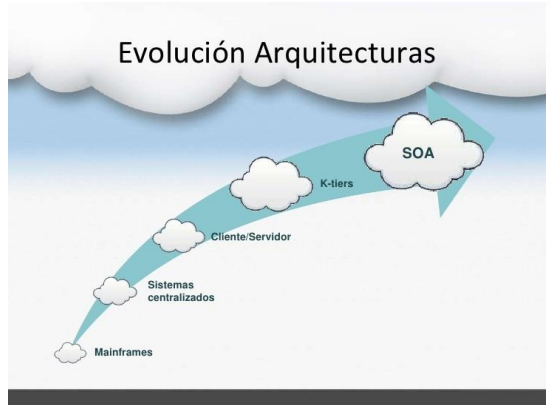
- La integración de las aplicaciones y los datos es fundamental para ofrecer nuevos servicios y experiencias a los clientes
- Por lo general, un equipo gestiona una tecnología de integración monolítica para toda una empresa, pero las aplicaciones son cada vez más complejas: se encuentran distribuidas y deben ampliarse y modificarse rápidamente para ir a la par de los mercados competitivos.
- Estos nuevos desafíos requieren un enfoque capaz de repetirse que utilice equipos ágiles y tecnologías de integración desarrolladas en la nube.

# Introducción a microservicios



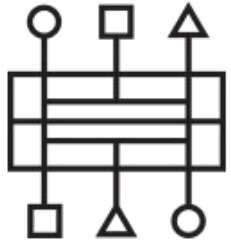
# ¿Qué se integra? Y ¿Cómo se integra?

Messaging

Enterprise integration patterns

Application connectors

Data streams



APIs

# ¿Qué se integra? Y ¿Cómo se integra?

- Mensajería : La mensajería es una forma de comunicación entre los diferentes componentes de una arquitectura de aplicaciones distribuida.
- Conectores : Son elementos de la arquitectura que determinan las reglas de interacción entre los componentes. Son conexiones estándares personalizadas para determinadas API, de modo que se pueden utilizar para integrar rápidamente los extremos nuevos.
- Flujos de datos : Con ellos, las aplicaciones pueden incorporar o utilizar datos que circulan de forma constante, independientemente de su transmisión
- Patrones de negocio : Se trata de conjuntos de soluciones que no dependen de la tecnología y que resuelven los problemas comunes de integración.
- API : Es un conjunto de herramientas, definiciones y protocolos que se utilizan para desarrollar el software de las aplicaciones.

# Enterprise Integrators Patterns

Se refiere a un conjunto de patrones utilizados para diseñar y desarrollar soluciones de integración de sistemas empresariales. Estos patrones proporcionan un lenguaje común y una guía para abordar los desafíos de integración que surgen al conectar sistemas heterogéneos y distribuidos en un entorno empresarial.

EIP fue creado por Gregor Hohpe y Bobby Woolf y se presenta en el libro "Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions" (Patrones de Integración Empresarial: Diseño, Construcción e Implementación de Soluciones de Mensajería).

Estos patrones se centran en el intercambio de mensajes y se aplican a sistemas que utilizan diferentes tecnologías y protocolos de comunicación, como servicios web, mensajería asíncrona, ESB (Enterprise Service Bus), entre otros.

# Enterprise Integrators Patterns

Message Channel: Proporciona un medio para el intercambio de mensajes entre componentes.

Message Router: Enruta los mensajes a diferentes destinos según ciertos criterios.

Message Filter: Filtra mensajes basados en criterios específicos.

Message Translator: Transforma mensajes de un formato a otro para facilitar la comunicación entre sistemas.

Content Enricher: Enriquece los mensajes agregando información adicional de fuentes externas.

Splitter: Divide un mensaje en partes más pequeñas para su procesamiento individual.

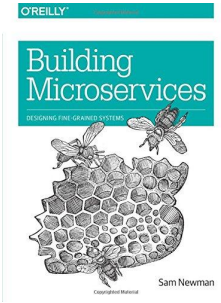
Aggregator: Combina múltiples mensajes relacionados en uno solo.

Event-Driven Consumer: Procesa los mensajes en respuesta a eventos específicos.

# ¿Qué es un micro servicio?

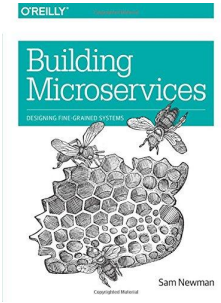
- Un micro servicio es la función principal de una aplicación que se ejecuta de forma independiente de los demás servicios

- *Martin Fowler "En pocas palabras, el estilo arquitectónico de micro servicios es un enfoque para el desarrollo de una sola aplicación como un conjunto de servicios pequeños, cada uno ejecutándose en su propio proceso y comunicándose con mecanismos ligeros, a menudo una API de un recurso HTTP. Estos servicios son construidos alrededor de capacidades comerciales y son desplegables independientemente por maquinaria de despliegue completamente automatizada. Hay un mínimo indispensable de gestión centralizada de estos servicios, que puede estar escrito en distintos lenguajes de programación y usar diferentes tecnologías de almacenamiento de datos."*

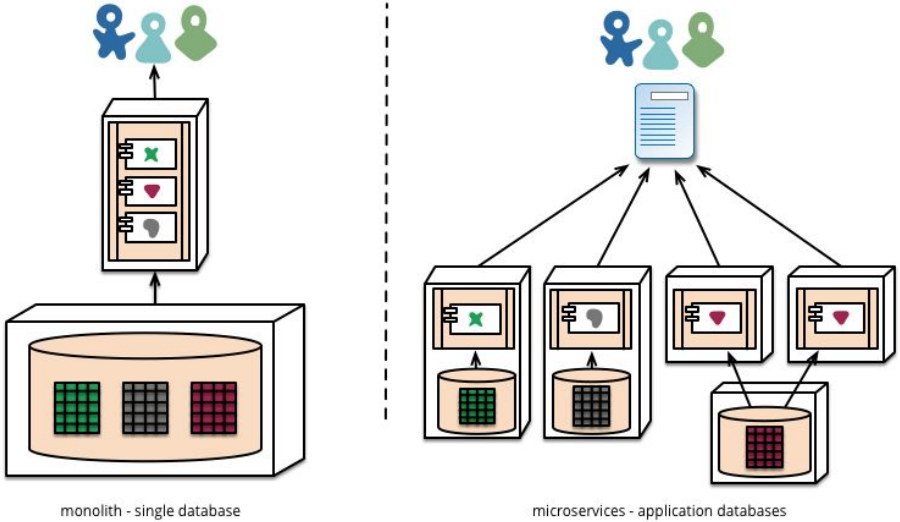


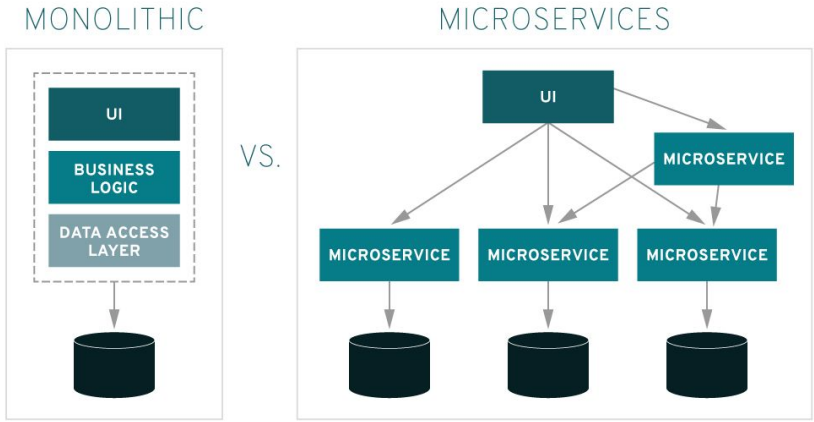
# ¿Qué es un micro servicio?

- Los microservicios son una arquitectura de desarrollo de software que estructura una aplicación como un conjunto de servicios pequeños, independientes, autónomos y altamente especializados.
- En lugar de construir una aplicación como un único monolito, los microservicios dividen la funcionalidad en componentes más pequeños y manejables, cada uno de los cuales es un servicio independiente.
- Aspecto claves :
  - Independencia y autonomía
  - Intercomunicación
  - Despliegue independiente
  - Escalabilidad
  - Tecnología Heterogénea
  - Resiliencia y Tolerancia a Fallos
  - Gestión de datos centralizado



# Introducción a microservicios

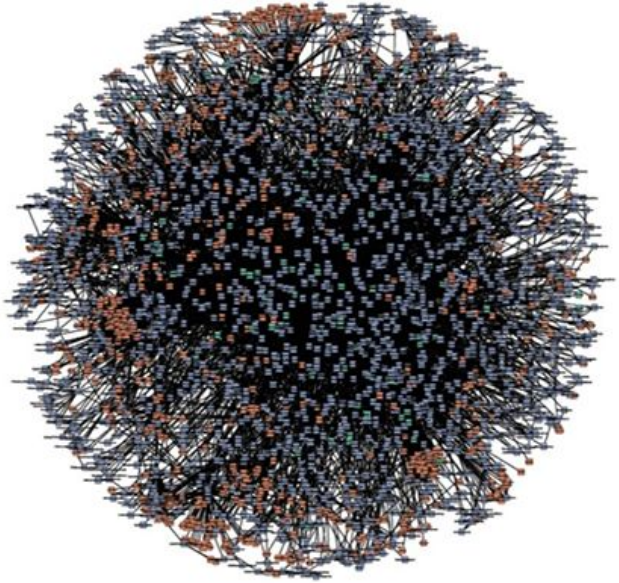


# Introducción a microservicios

- Tradicionalmente la arquitectura de las aplicaciones es monolítica
- La arquitectura de micro servicios en cambio, se caracteriza por modularizar las aplicaciones en muchos programas diferentes que se comunican entre sí, para en conjunto cumplir con los requerimientos del sistema.
- Cada micro servicio se encarga de implementar un componente de la aplicación entera. Se llaman micro servicios, para comunicar la idea que son muchos servicios pequeños que conversan.
- **La regla de oro de los micro servicios**, es que siempre sea posible hacer un cambio a un micro servicio y desplegarlo de **manera independiente**, sin afectar el resto de la aplicación

# Diagrama de arquitectura de microservicios

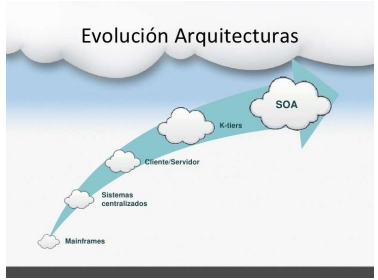


amazon.com

NETFLIX

# Microservicios ¿evolución de SOA?

- La arquitectura de micro servicios es una forma específica de implementación de SOA.
- La idea de SOA, es solamente romper un monolito en servicios que colaboran, con el beneficio teórico que se pueda reemplazar la implementación de un servicio de manera transparente para el resto de la aplicación.
- En cambio, la arquitectura de micro servicios surgió de uso real, como una manera bien definida de implementar SOA **bien** que permite evitar sus desventajas más comunes.
- ¿Cuáles son las desventajas de SOA?



# Beneficios de arquitectura de microservicios

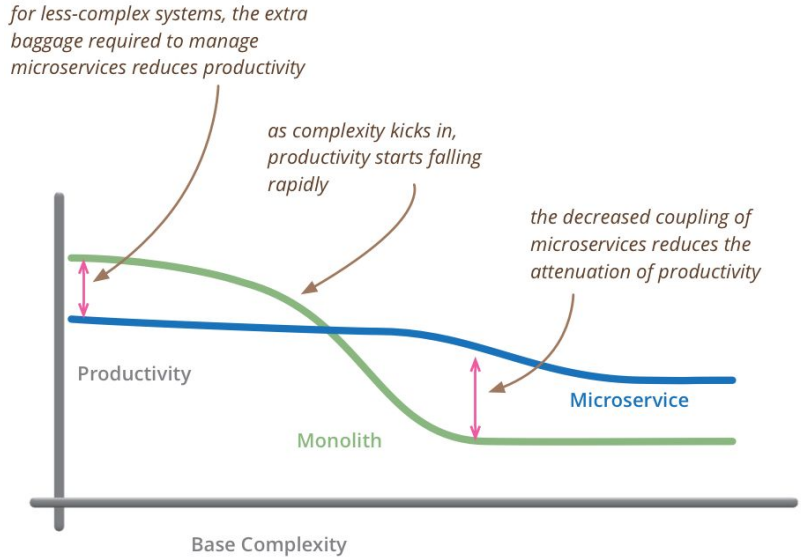
- Permite una programación políglota : Si un micro servicio necesita mucha velocidad, lo puedes escribir en Rust o Go, en caso contrario lo puedes escribir a un alto nivel de abstracción usando Ruby, Python o Node.js.
- Más fácil escalar la aplicación
- Más fácil de entender
- Más robustez
- El acoplamiento es mejor con la estructura de la organización

# Desafíos de arquitectura de microservicios

- Mayor complejidad de operación
- Difícil cambiar la arquitectura entre los servicios
- La depuración de errores es compleja
- Los sistemas distribuidos crean problema a la mayoría de equipos de desarrollo

# Cuándo usar microservicios

- Cuando la aplicación tiene una complejidad alta es bueno revisar la posibilidad de implementar la arquitectura de micro servicios



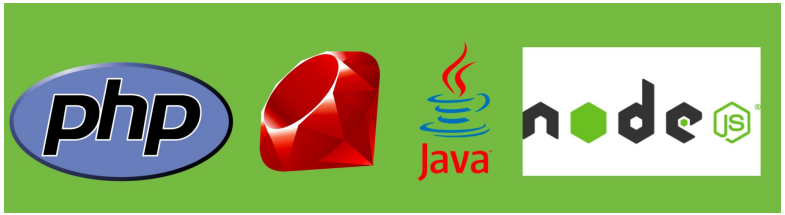

*but remember the skill of the team will outweigh any monolith/microservice choice*

# Cuándo usar microservicios

- Dividir un sistema monolítico grande en muchos servicios pequeños
- Optimizar servicios para una sola función
- Comunicarse a través de REST API y message brokers
- Aplicar CI/CD por servicio
- Aplicar alta disponibilidad por servicio (HA) / decisiones sobre clústeres

# Cuándo usar microservicios

- Desarrollo sencillo
- Implemente una necesidad de negocio específica
- Mayor control
- Servicios pequeños e independientes
- Cambios rápidos independientes, mejor “time to market”
- Deploy por servicio
- Equipos especializados geo distribuidos
- Independiente a la tecnología



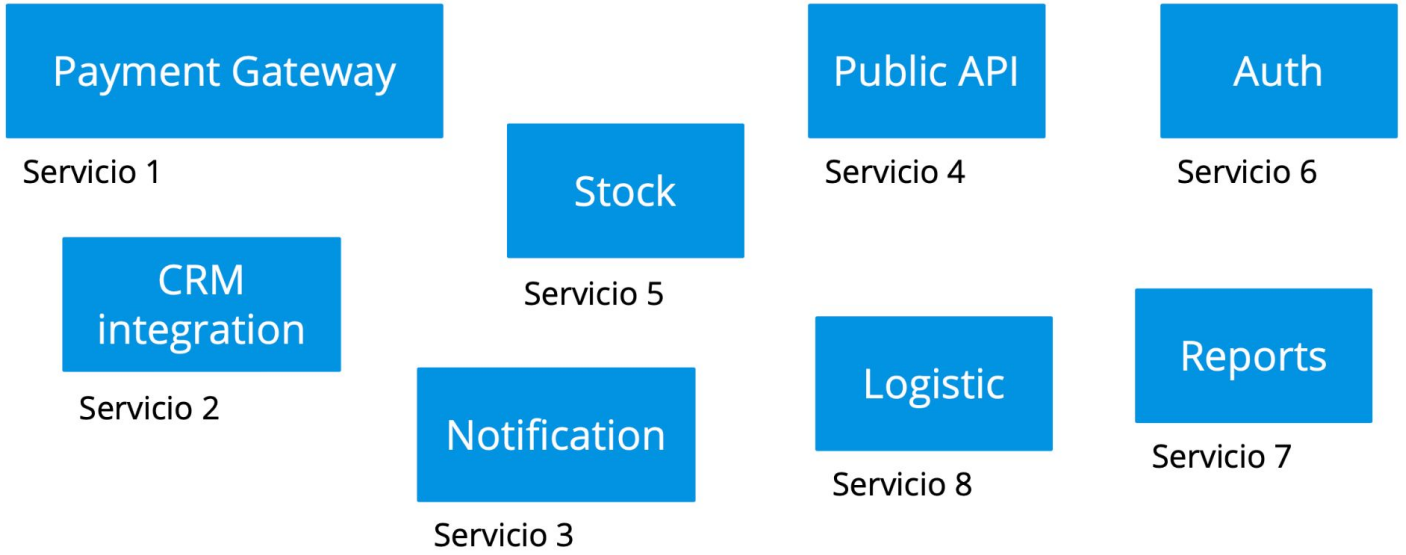
# Cuándo usar microservicios

- Principios SOLID
  - Single responsibility, Interface Segregation
  - Liskov Substitution, Dependency inversion
- DB Distribuidas
  - Cada servicio con su DB, no acoplar servicios por DB
  - Sincronización de datos y gestión de consistencia de datos
  - API's documentadas
    - RAML / Api Blueprint
  - Comunicación sencilla
    - Restful, SNS/SQS
    - Redis, AMQP, Zero MQ, RabbitMQ

# Cuándo usar microservicios

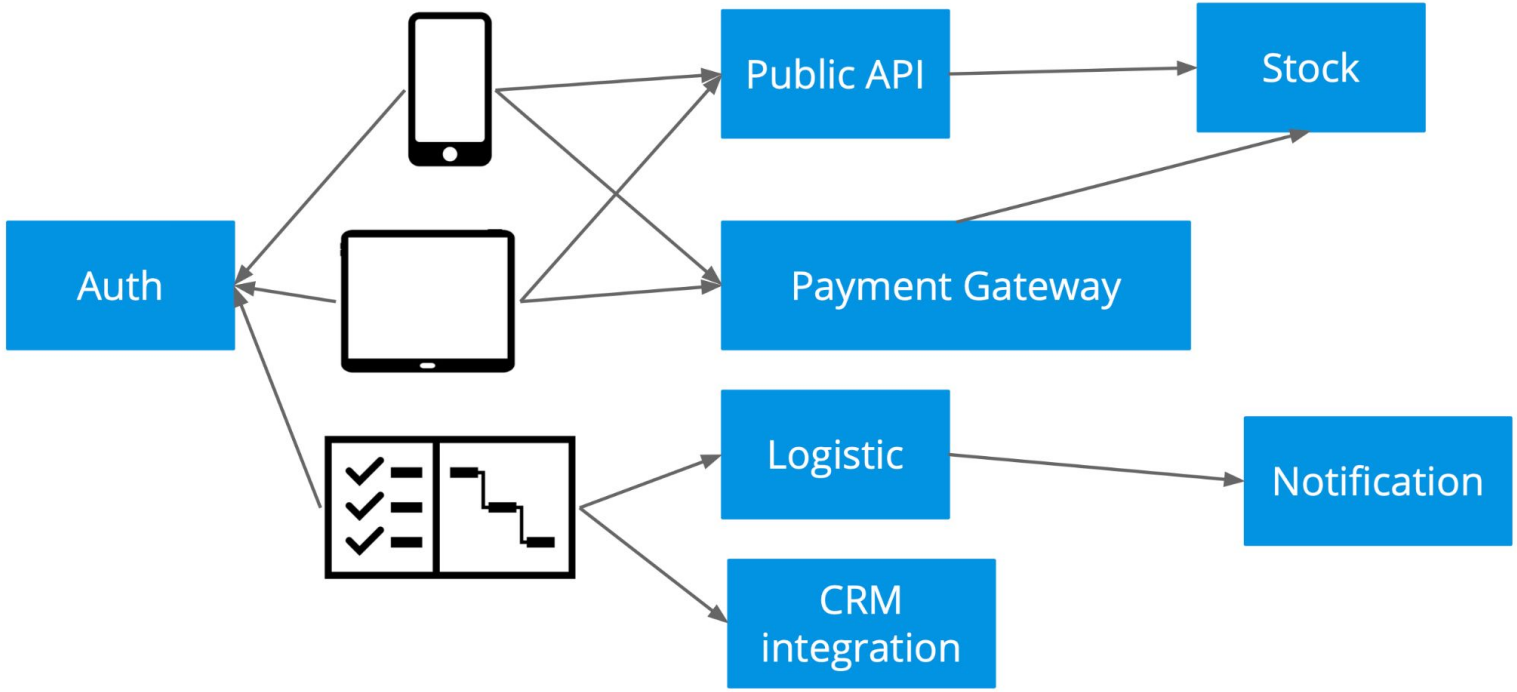
- Monitoreo
    - Centralizar logs (Loggly, Logstash, Grafana, Kibana)
    - Centralizar monitoreo (Zabbix, New Relic, Grafana, Kibana)
- Automatizar deploy
    - 1 deploy - ndeploy
    - AWS / Vagrant /Docker
    - Puppet / Chef

# Cuándo usar microservicios




E-commerce

# Cuándo usar microservicios



# Aplicación Monolítica y Microservicios

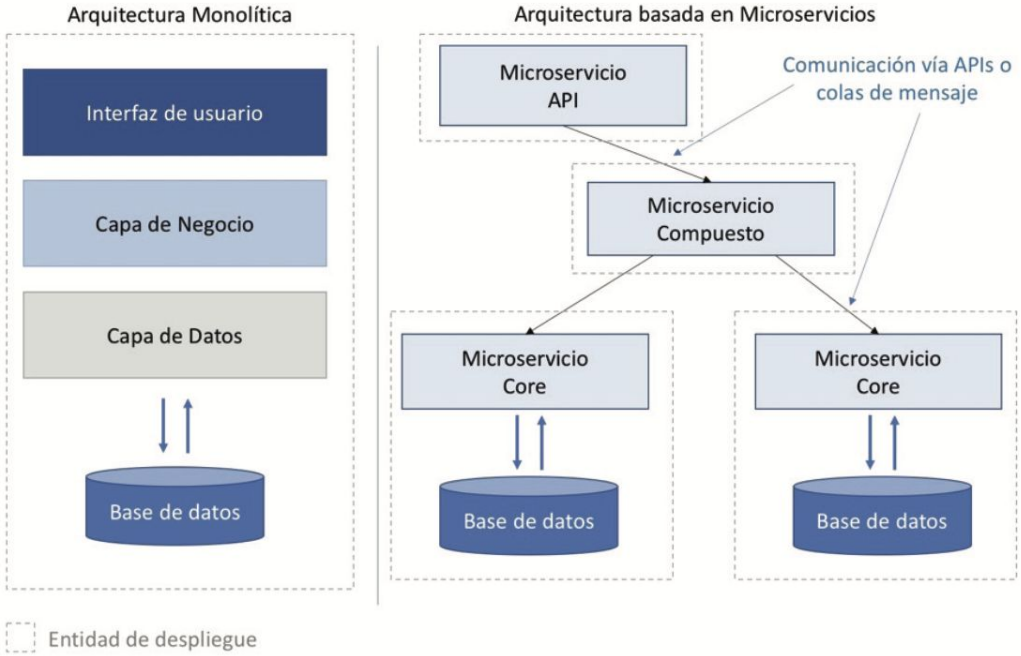



**Figura 2.4.** Arquitectura monolítica vs Microservicios

# Cubo de Escalabilidad

- Desarrollado por Martin Abbott y Michael Fisher en su libro de “Art of Scalability”.
- Considera 3 dimensiones: la duplicación (X) la descomposición funcional (Y) y la partición de datos (Z).
- El objetivo del cubo es que variando a través de los ejes conseguimos distintas estrategias de escalado de aplicaciones.

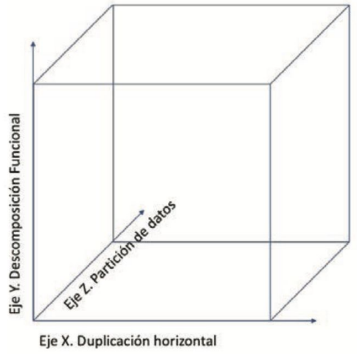



**Figura 2.3.** Cubo de escalabilidad

# Cubo de Escalabilidad

- Duplicación horizontal o duplicación, es la estrategia más fácil, tras un balanceador de carga se ejecutan varias copias de la aplicación, de manera que dicha carga se reparte entre ellas.
- Este escalado es capaz de soportar un mayor nivel de carga.

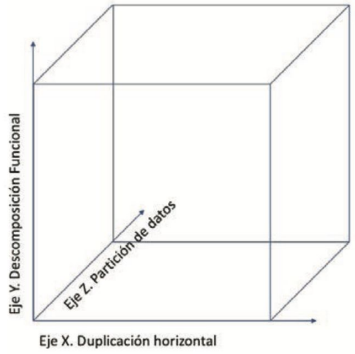



**Figura 2.3.** Cubo de escalabilidad

# Cubo de Escalabilidad

- Partición de datos o balanceo en función de la información. En cada servidor se ejecuta la aplicación completa pero el balanceo particiona los datos en rangos de manera que a cada servidor le corresponde manejar un determinado conjunto de datos.
- Se puede realizar segmentación por zona geográfica, tipo de cliente u otro tipo de estructura que sea funcional para la organización.

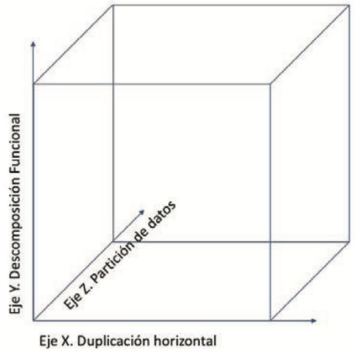



**Figura 2.3.** Cubo de escalabilidad

# Cubo de Escalabilidad

- Partición de datos o balanceo en función de la información
- Ventajas :
  - Cada servidor maneja menos información
  - Reduce el consumo de memoria
  - Aumenta el rendimiento, transacciones más rápidas
  - Posibles caídas de servidor afectan a menos usuarios
- Desventajas :
  - Incremento de la complejidad, particionado.
  - No resuelve los problemas monolíticos de la aplicación.

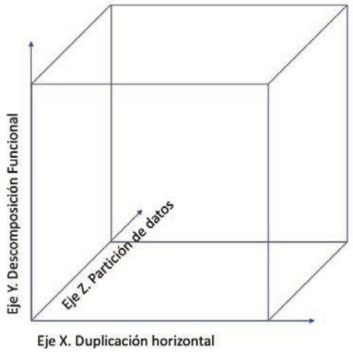



**Figura 2.3.** Cubo de escalabilidad

# Cubo de Escalabilidad

- Escalado funcional base del principio de los microservicios.
- Consiste en la descomposición de la aplicación en unidades funcionales independientes. De manera que las distintas unidades pueden evolucionar sin necesidad de afectar al resto.
- Además el testeo, el despliegue y otras actividades del ciclo de vida se llevan a cabo de manera separada.

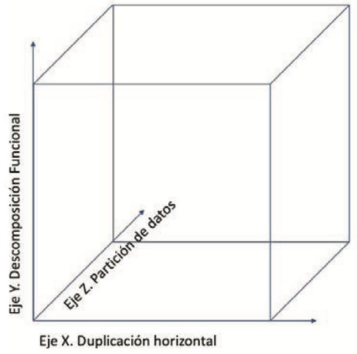
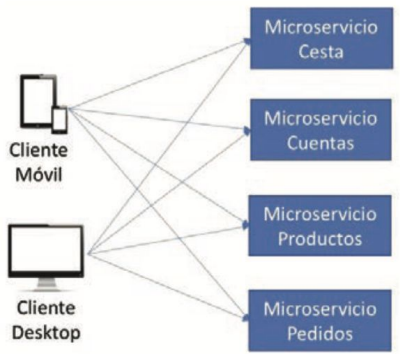



**Figura 2.3.** Cubo de escalabilidad

# Comunicación entre microservicios

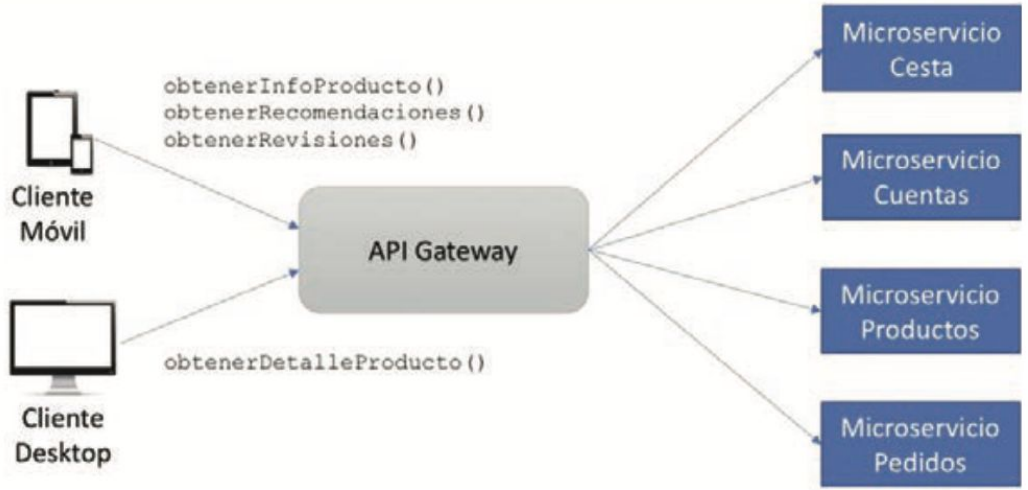
- En un entorno distribuido estas características serán de vital importancia, tratando de mantener lo más sencillo el modelo de comunicación.
- Comunicación directa entre el cliente y el microservicio.  Cliente a endpoint.
- Tipos de comunicación :
    - Por clase de protocolo,
        - sincrónico (http/https), API, GraphQL
        - Asincrónico (amqp), OpenMQ, RabbitMQ




    - Por número de receptores.
        - Permite enviar mensajes o eventos desde un microservicio emisor a múltiples microservicios receptores a través de un sistema de mensajería. Esto brinda desacoplamiento, escalabilidad, resiliencia y flexibilidad al sistema.

# Comunicación entre microservicios

- En un entorno distribuido estas características serán de vital importancia, tratando de mantener lo más sencillo el modelo de comunicación.
- Comunicación a través de un API Gateway
- Estilos de comunicación :
  - HTTP
  - REST



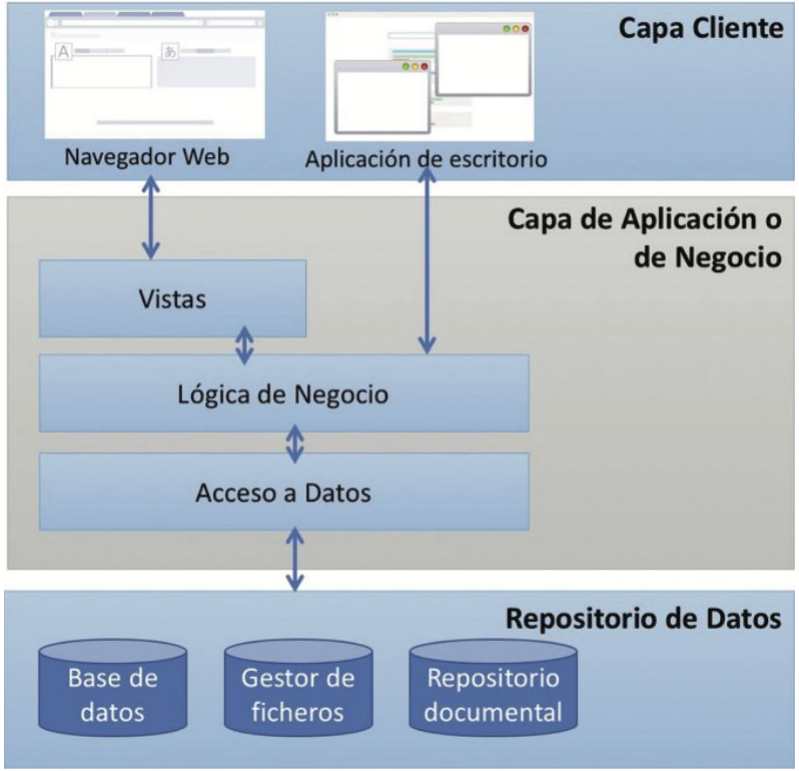
# Micro servicios

- Se construyen alrededor de funcionalidades de negocio muy concretas (autenticación, directorio, correo, CRUD de ítems, etc.) y que se comunican entre sí mediante mecanismos de interacción relativamente sencillos, como puedan ser llamadas *RESTfull* a un API o algún tipo de RPC (*Remote Procedure Communications*).

- Cada micro servicio se despliega de manera independiente al resto, lo que posibilita que cada uno evolucione por separado, e incluso que estén construidos en tecnologías diferentes, siempre y cuando expongan sus funcionalidades a través de un API bien documentada y conocida.

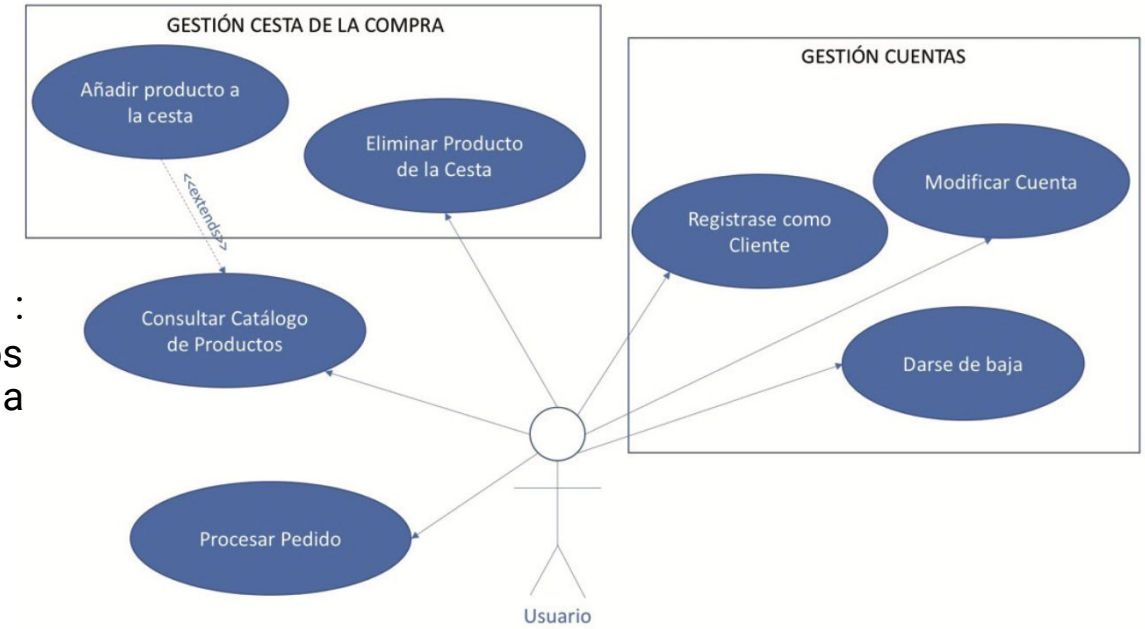
# Evolución a microservicios E-Commerce

- ¿Qué criterio seguir al momento de dirigir la evolución de una aplicación con arquitectura monolítica a micro servicios?



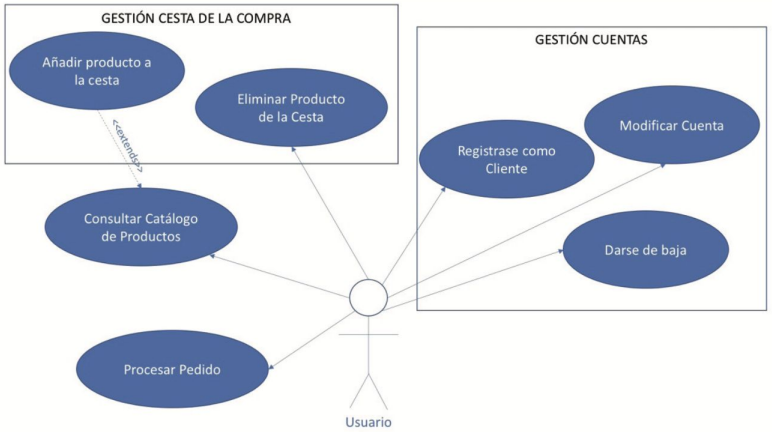
# Evolución a microservicios E-Commerce

- Diagrama UML de Casos de Uso : Muestra la interacción entre los distintos roles o tipos de usuario y la aplicación que se está diseñando



# Evolución a microservicios E-Commerce

- Se puede utilizar para esta aplicación una descomposición basada en sustantivos, agrupamos en cada micro servicio las funcionalidades relacionadas con una entidad en particular (sustantivo), respetando el SRP Single Responsibility Principle de Robert C. Martin.




# Ventajas de la evolución a microservicios E-Commerce

- El código fuente de cada micro servicio es mucho menor que el mismo código en “modo monolítico”, lo que mejora la comprensión y la gestión del mismo por parte del desarrollador.

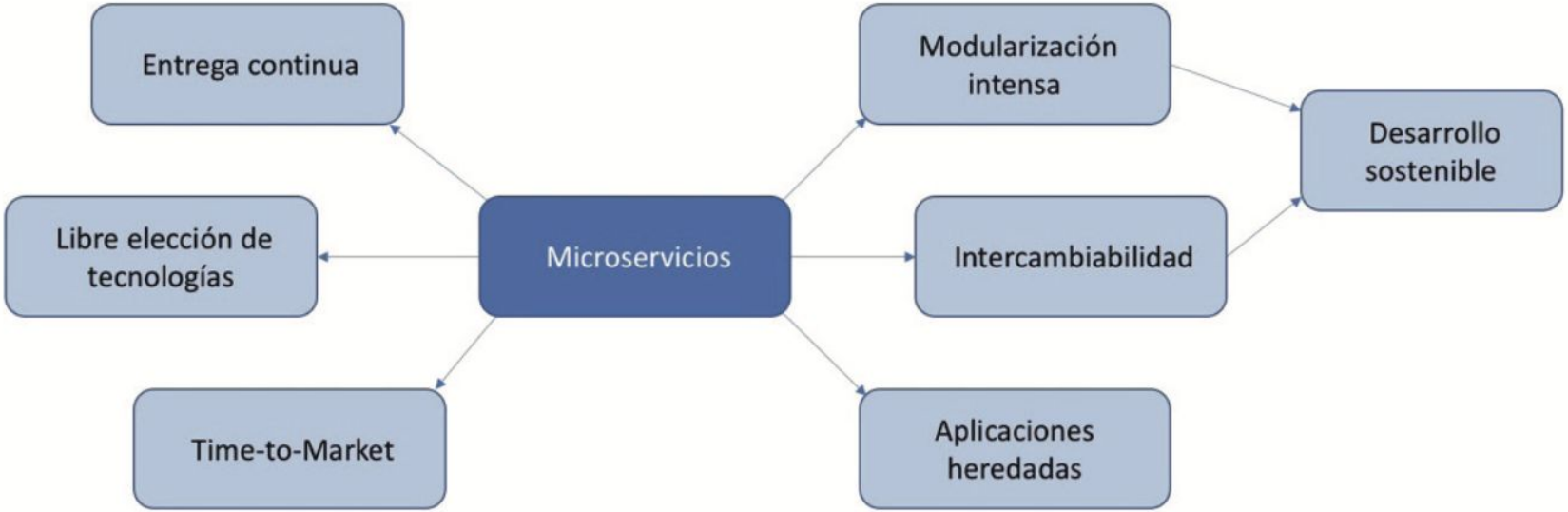
- Cada micro servicio se despliega de manera independiente al resto, por lo resulta más fácil desplegar nuevas versiones con relativa frecuencia y dota a la cadena de despliegue de mayor autonomía y velocidad.

- Cada equipo de desarrollo es responsable de todo el ciclo completo de un micro servicio: requisitos, codificación, pruebas y puesta en producción.

- Aumenta la tolerancia a fallos, ya que un micro servicios no puede hacer que falle la totalidad del sistema.

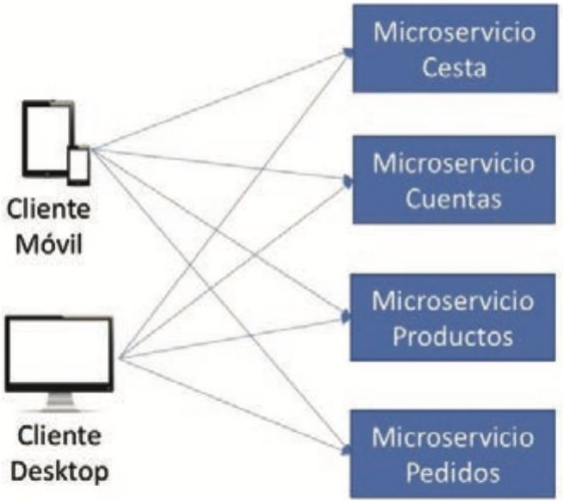
- Cada micro servicio puede estar desarrollado en una tecnología diferente, con una persistencia distinta, etc.

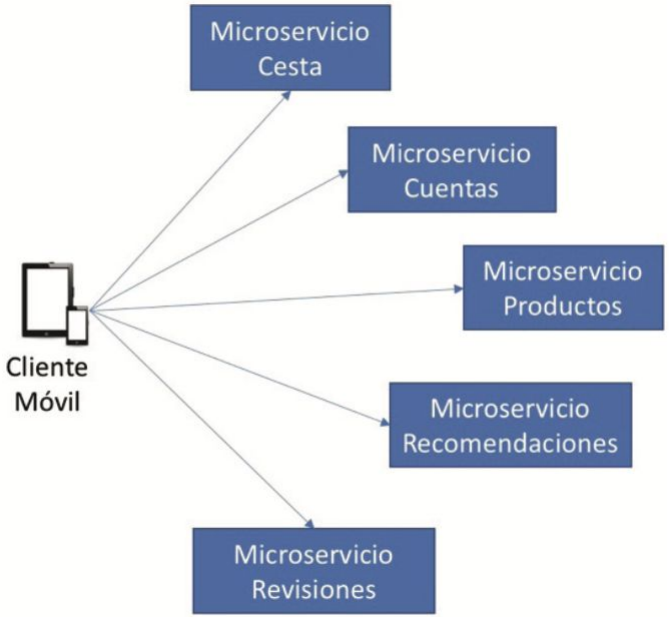
# Características de los micro servicios



# Evolución a microservicios E-Commerce

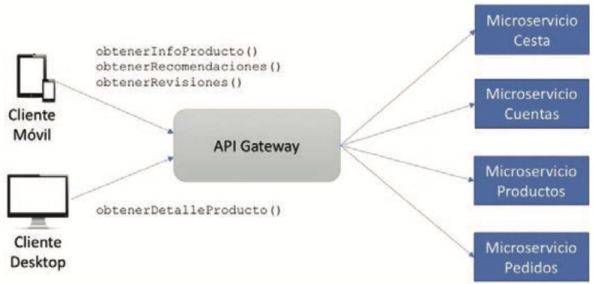
- **https://servicename.api.company.name/**



# Evolución a microservicios E-Commerce

- La mayor ventaja de un esquema de comunicación basado en API Gateway es que encapsula la estructura interna de la aplicación. En lugar de invocar a servicios específicos, los clientes simplemente se comunican con la pasarela, que proporciona a cada tipo de cliente un API específica, reduciendo así la latencia entre la aplicación y el cliente y, además, simplificando el código del cliente
- El riesgo es que el API Gateway se convierta en un cuello de botella



# Evolución a microservicios E-Commerce

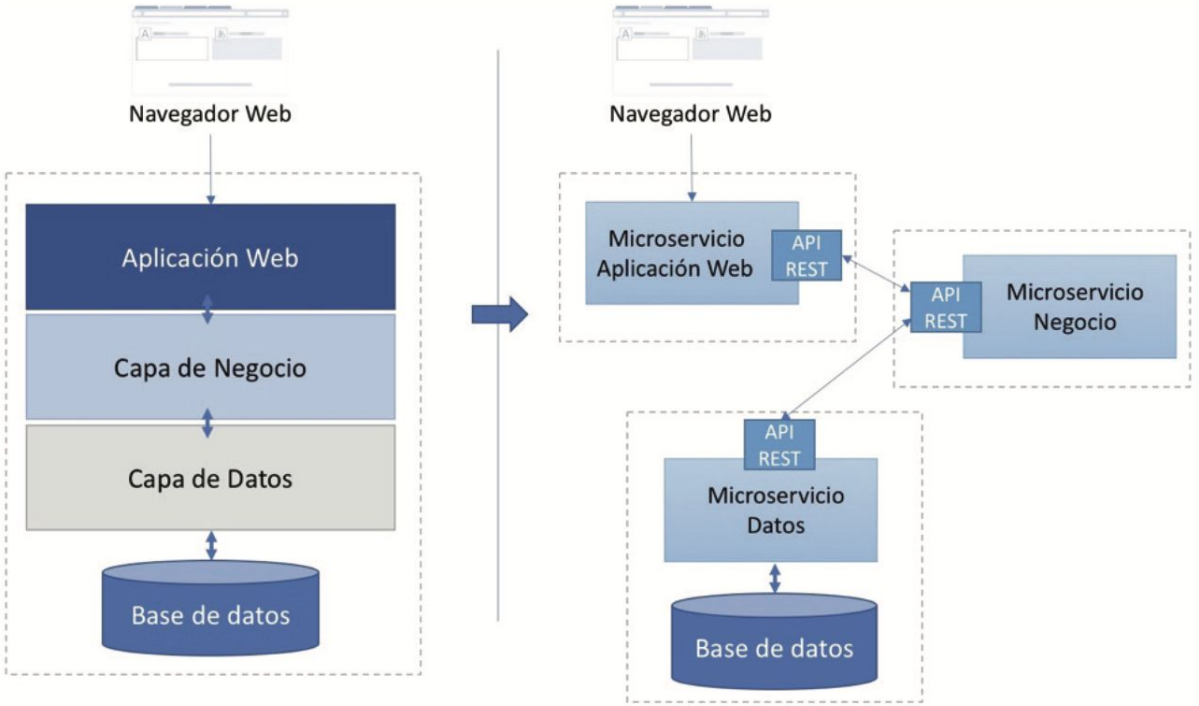



**Figura 2.4.** Evolución por distribución del back-end y el front-end.

# Evolución a microservicios E-Commerce

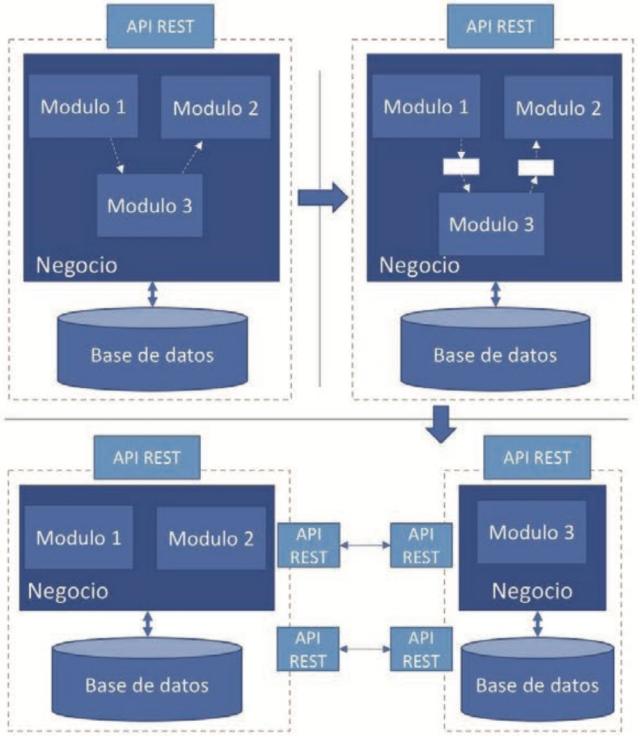



**Figura 2.5.** Extracción de microservicios

# Bibliografía


- "Software Architecture in Practice" (Arquitectura de Software en la Práctica), Len Bass, Paul Clements, Rick Kazman
- "Pattern-Oriented Software Architecture: A System of Patterns" (Arquitectura de Software Orientada a Patrones: Un Sistema de Patrones), Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, Michael Stal
- "Domain-Driven Design: Tackling Complexity in the Heart of Software" (Diseño Dirigido por Dominios: Abordando la Complejidad en el Corazón del Software), Eric Evans
- "Clean Architecture: A Craftsman's Guide to Software Structure and Design" (Arquitectura Limpia: Guía del Artesano para la Estructura y Diseño de Software), Robert C. Martin (Uncle Bob)
- "Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions" (Patrones de Integración Empresarial: Diseño, Construcción e Implementación de Soluciones de Mensajería, Gregor Hohpe, Bobby Woolf
- "Microservices Patterns: With Examples in Java" (Patrones de Microservicios: Con Ejemplos en Java), Chris Richardson