El futuro digital es de todos | MinTIC

UNIVERSIDAD EL BOSQUE | MisionTIC2022

Ciclo 4A

Semana 4

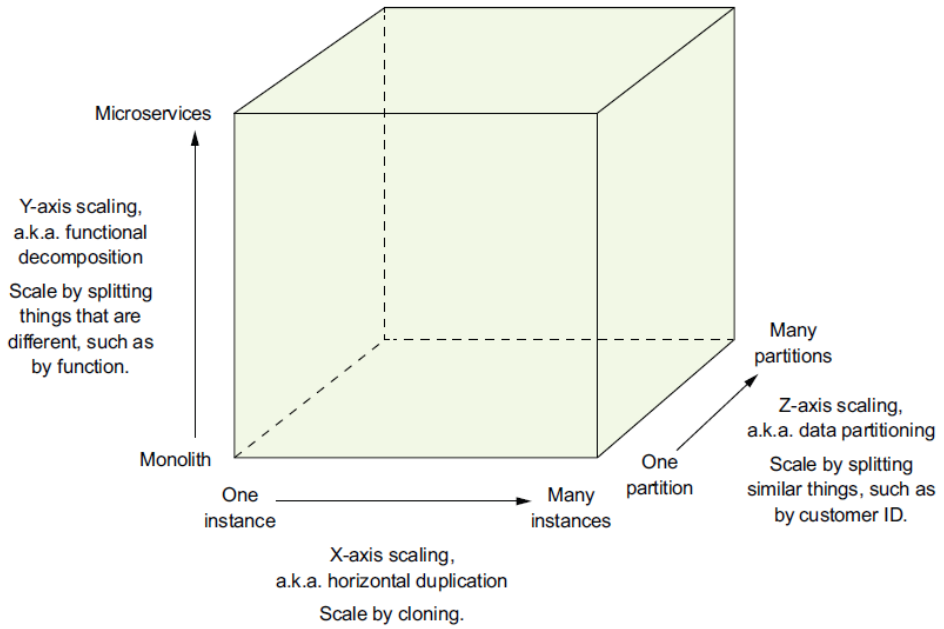
DAO, DTO, manejo de excepciones y documentación de código

Lectura 2 - Arquitectura de Microservicios

Convenio de asociación no. 777 de 2021 suscrito entre el Fondo Único de Tecnologías de la Información y las Comunicaciones y la Universidad el Bosque.

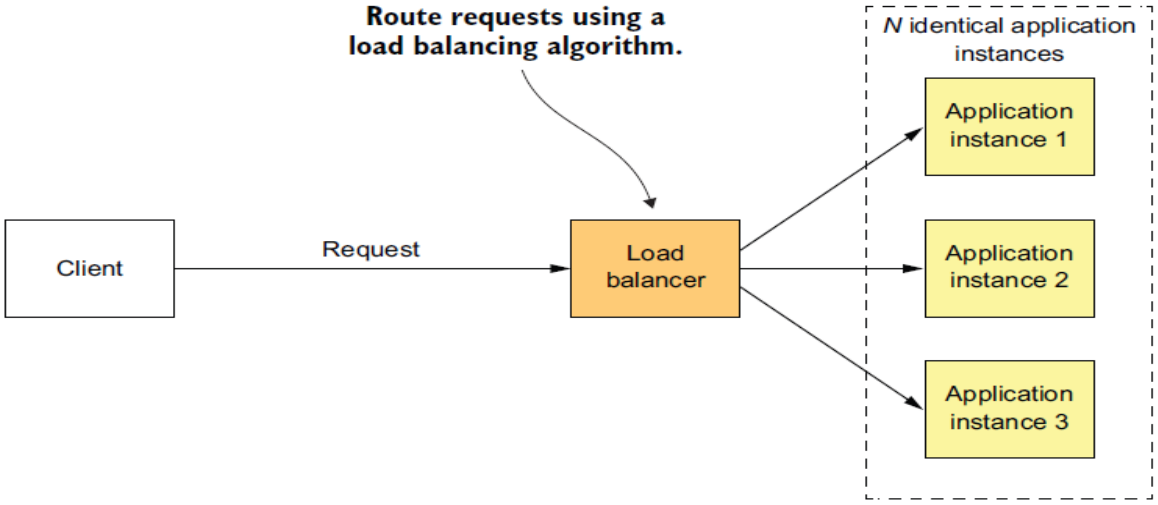
Arquitectura de Microservicios

En el libro "The Art of Scalability (Addison-Wesley,2015)" de Martin Abbott y Michael Fisher se describe un cubo como modelo de escalabilidad tridimensional "scale cube":

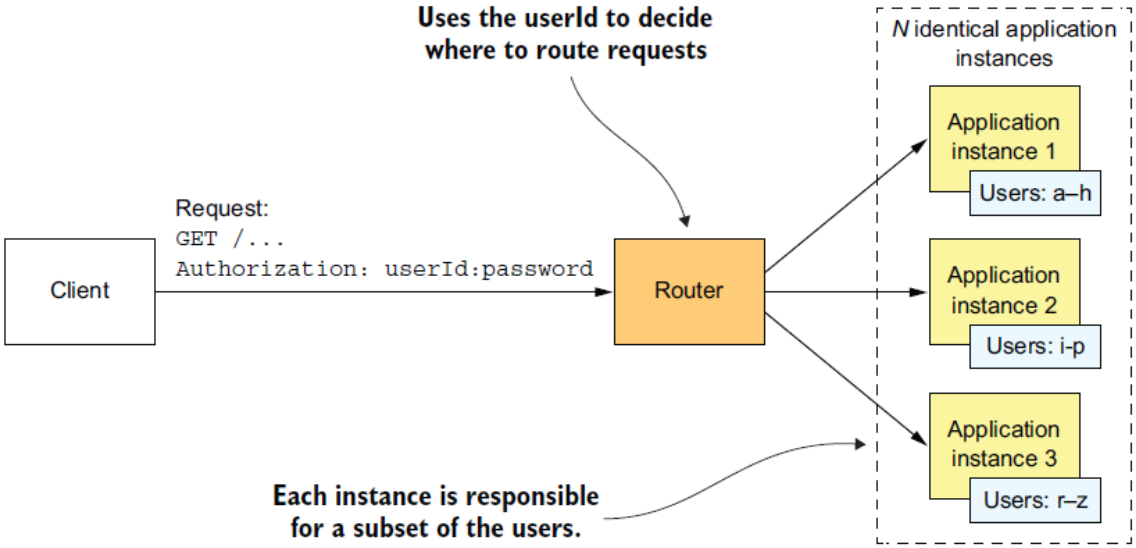


El modelo define tres formas de escalar una aplicación: X, Y y Z.

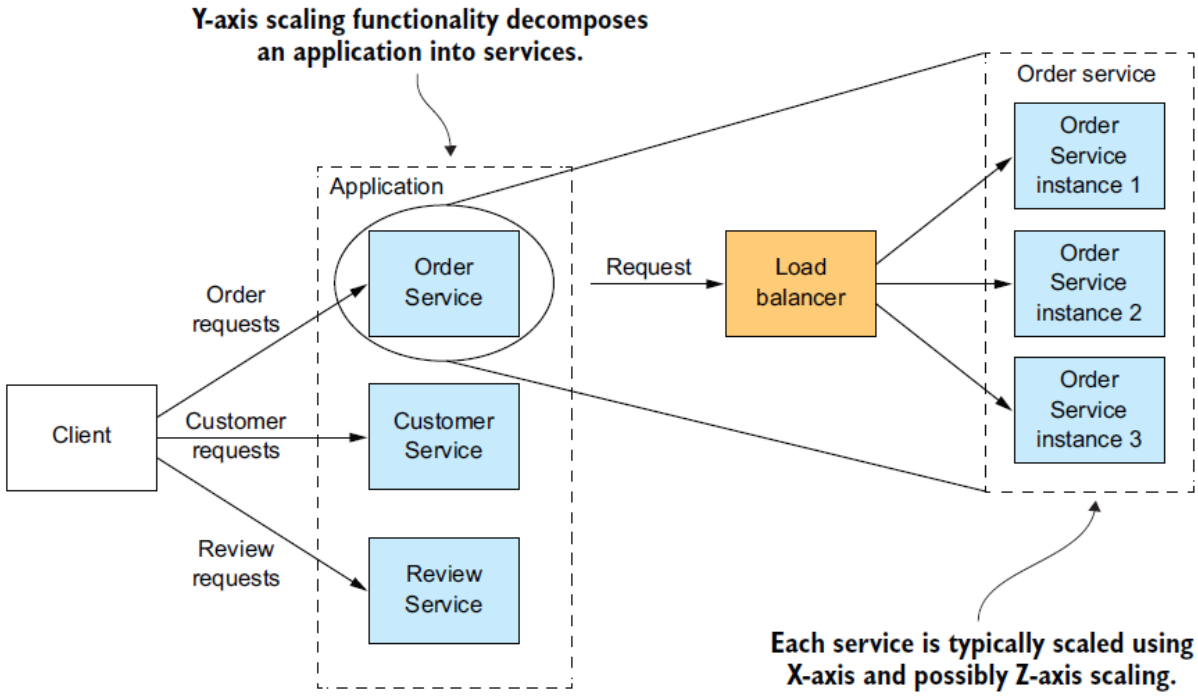
Escalar en X es una forma común de escalar una aplicación monolítica, en la que se corren múltiples instancias de la aplicación detrás de un balanceador de carga, el cual distribuye las peticiones entre las n instancias idénticas de la aplicación, mejorando su capacidad y disponibilidad.



Escalar en Z también corre múltiples instancias de la aplicación, pero a diferencia del escalamiento en X, cada instancia es responsable de solamente un subconjunto de los datos. El enrutador en frente de las instancias usa un atributo de la petición para enrutar a la instancia apropiada, como por ejemplo el userId, en cuyo caso cada instancia de la aplicación es responsable de un subconjunto de los usuarios. Este escalamiento permite manejar grandes incrementos en las transacciones y en los volúmenes de datos.



Tanto escalar en X como en Z mejora la capacidad y disponibilidad de la aplicación, pero ninguno de los dos enfoques soluciona el problema del incremento en la complejidad de la aplicación y en su desarrollo. Para solucionarlo se requiere escalar en Y, dividiendo una aplicación monolítica en un conjunto de servicios.



De acuerdo con lo anterior, dando una definición de alto nivel de la arquitectura de microsdervicios, se puede decir que es un estilo de arquitectura que descompone funcionalmente una aplicación en un conjunto de servicios. En ella no se hace referencia al tamaño, lo que importa es que cada servicio tiene un conjunto centrado y cohesivo de responsabilidades.

Los microservicios como una forma de modularidad: La modularidad es esencial cuando se desarrollan aplicaciones grandes y complejas que requieren varios desarrolladores. Deben ser descompuestas en módulos que son desarrollados y entendidos por diferentes personas. La arquitectura de microservicios usa los servicios como la unidad de modularidad. Cada servicio tiene un API que es una frontera impermeable que es difícil de violar, lo que trae como resultado, que es mucho más fácil preservar la modularidad de la aplicación en el tiempo y también la habilidad de desplegarlos y escalarlos independientemente.

Cada servicio tiene su propia base de datos: Una característica clave de esta arquitectura es que los servicios son débilmente acoplados y se comunican solamente vía APIs. Una forma de lograr ese bajo acoplamiento es que cada servicio tenga su propio almacén de datos. Así, al momento del desarrollo los

desarrolladores pueden cambiar un esquema de servicio sin tener que coordinar con otros desarrolladores trabajando en otros servicios. En ejecución, los servicios son aislados unos de otros, de tal forma que un servicio no bloqueará a otro por estar trabajando registros en la base de datos.

Comparando microservicios y SOA: A muy alto nivel, hay algunas similaridades, ambos son estilos de arquitectura que estructuran un sistema como un conjunto de servicios, pero si se mira más profundo se encuentran diferencias significativas.

Aspecto	SOA	Microservicios
Comunicación entre servicios	Conductos inteligentes, como Enterprise Service Bus (ESB), usando protocolos pesados, como SOAP y otros estándares WS*.	Conductos no inteligentes, como message broker, o comunicación directa servicio a servicio, usando protocolos livianos como REST o gRPC.
Datos	Modelo de datos global y bases de datos compartidas	Modelo de datos y base de datos por servicio.
Servicio típico	Aplicación monolítica más grande.	Servicio más pequeño.

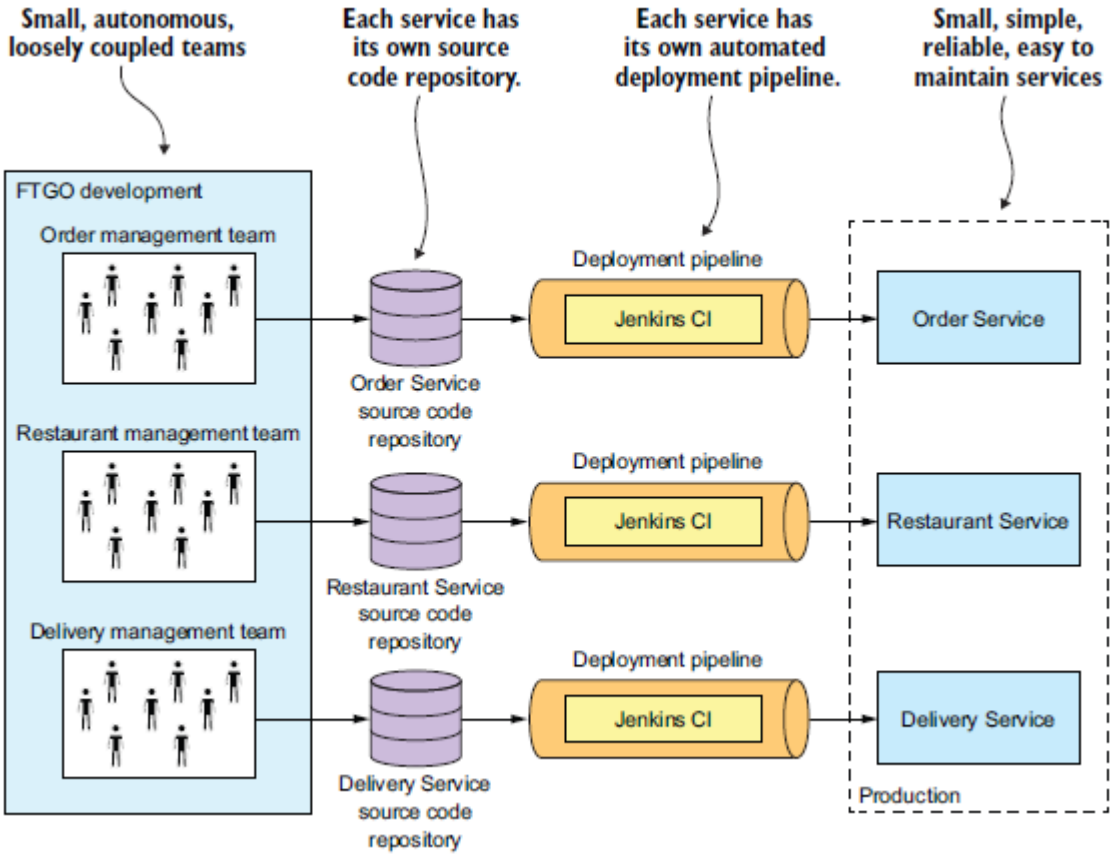
SOA y la arquitectura de microservicios usualmente usan diferentes pilas de tecnología. Las aplicaciones SOA típicamente usan tecnologías pesadas y otros estándares WS*. Frecuentemente utilizan un ESB, un conducto inteligente que contiene lógica de negocios y de procesamiento de mensajes para integrar los servicios. Las aplicaciones construidas usando arquitectura de microservicios tienden a usar tecnologías livianas de código abierto.

Beneficios de la arquitectura microservicios:

- **Permite el continuo desarrollo y despliegue de aplicaciones grandes y complejas.**

El despliegue y desarrollo continuos es parte del DevOps, un conjunto de prácticas para el despliegue de software rápido, frecuente y confiable. Hay tres formas que la arquitectura de microservicios permite un despliegue/desarrollo continuo:

- Las pruebas automatizadas son una práctica clave para un despliegue/desarrollo continuo. Debido a que cada servicio es relativamente pequeño, las pruebas automatizadas son mucho más fáciles de escribir y ejecutar. Como resultado, la aplicación tendrá menos errores.
- Cada servicio puede ser desplegado independientemente de otros servicios. Si los desarrolladores responsables de un servicio necesitan desplegar un cambio que es local a ese servicio, ellos no necesitan coordinar con otros desarrolladores, ellos pueden desplegar sus cambios. Como resultado, es mucho más fácil desplegar cambios frecuentemente en producción.
- La organización de ingeniería se puede estructurar como una colección de equipos pequeños, cada uno responsable solo del despliegue y desarrollo de uno o más servicios relacionados y cada equipo puede desarrollar, desplegar y escalar sus servicios independientemente de los otros equipos. Como resultado, la velocidad de desarrollo es mucho mayor.



- **Los servicios son pequeños y fácilmente mantenidos.**

Como cada servicio es relativamente pequeño, el código es más fácil de entender para un desarrollador y esa base de código pequeña no ralentiza el IDE, lo que hace que los desarrolladores sean más productivos. Y cada servicio normalmente inicia mucho más rápido que un gran servicio monolítico, lo que también hace que los desarrolladores sean más productivos y acelera las implementaciones.

- **Los servicios son independientemente desplegables y escalables.**

Cada servicio puede ser desplegado y escalado independientemente de los otros servicios utilizando la clonación en X y la partición en Z. Además, cada servicio puede ser desplegado en el hardware que mejor se adapta a sus requisitos de recursos. Esto es bastante diferente que cuando se utiliza una arquitectura monolítica, donde los componentes con muy diferentes requisitos de recursos (por ejemplo, uso intensivo de CPU frente a uso intensivo de memoria) deben desplegarse juntos.

- **Permite que los equipos sean autónomos.**

- **Tiene un mejor aislamiento de las fallas.**

Por ejemplo, una pérdida de memoria en un servicio solo afecta a ese servicio. Otros servicios seguirán atendiendo solicitudes normalmente. En comparación, un componente que falle en una arquitectura monolítica derribará todo el sistema.

- **Permite una fácil experimentación y adopción de nuevas tecnologías.**

Por último, pero no menos importante, la arquitectura de microservicio elimina cualquier compromiso a largo plazo con una pila de tecnología. En principio, al desarrollar un nuevo servicio, los desarrolladores son libres de elegir el idioma y los marcos de trabajo que mejor se adapten a ese servicio. En muchas organizaciones, tiene sentido restringir las opciones, pero el punto clave es que no está limitado por decisiones pasadas. Además, debido a que los servicios son pequeños, reescribirlos usando mejores lenguajes y tecnologías se vuelve práctico. Si falla la prueba de una nueva tecnología, puede desecharse ese trabajo sin arriesgar todo el proyecto. Esto es bastante diferente a cuando se utiliza una arquitectura monolítica, donde sus elecciones iniciales de tecnología limitan severamente su capacidad para utilizar diferentes lenguajes y marcos de trabajo en el futuro.

Desventajas de la arquitectura de microservicios:

Ciertamente, ninguna tecnología es una solución milagrosa, y la arquitectura de microservicios tiene una serie de inconvenientes y problemas importantes:

- **Encontrar el conjunto de servicios adecuado es un desafío.**

Un desafío con el uso de la arquitectura de microservicios es que no hay un algoritmo bien definido para descomponer un sistema en servicios. Como ocurre con gran parte del desarrollo de software, es una especie de arte. Para empeorar las cosas, si se descompone un sistema incorrectamente, se construirá un monolítico distribuido, un sistema que consta de servicios acoplados que deben implementarse juntos. Un monolítico distribuido tiene los inconvenientes de tanto la arquitectura monolítica como la arquitectura de microservicios.

- **Los sistemas distribuidos son complejos, lo que hace difícil el desarrollo, las pruebas y la implementación.**

Otro problema con el uso de la arquitectura de microservicios es que los desarrolladores deben lidiar con la complejidad adicional de crear un sistema distribuido. Los servicios deben utilizar un mecanismo de comunicación entre procesos. Esto es más complejo que una simple llamada a un método. Además, un servicio debe diseñarse para manejar fallas parciales y hacer frente a que el servicio remoto no esté disponible o presente una alta latencia. La implementación de casos de uso que abarcan varios servicios requiere el uso de técnicas no tan familiares. Cada servicio tiene su propia base de datos, lo que hace que sea un desafío la implementación de transacciones y consultas que abarcan servicios. Una aplicación basada en microservicios no puede recuperar datos de varios servicios mediante consultas sencillas. En cambio, debe implementar consultas utilizando la composición de APIs o vistas CQRS.

Los IDE y otras herramientas de desarrollo se centran en la creación de aplicaciones monolíticas y no brindan soporte explícito para el desarrollo de aplicaciones distribuidas. Escribir pruebas automatizadas que involucran múltiples servicios es un desafío. Todos estos son problemas que son específicos de la arquitectura de microservicios. Consecuentemente, los desarrolladores deben tener habilidades sofisticadas de desarrollo y despliegue de software para para utilizar los microservicios con éxito.

- **La implementación de funciones que abarcan varios servicios requiere una coordinación cuidadosa.**

Otro desafío con el uso de la arquitectura de microservicios es que la implementación de funciones que abarcan múltiples servicios requiere una cuidadosa coordinación entre los diversos equipos de desarrollo. Debe crear un plan de implementación que ordene las implementaciones de servicios basado en las dependencias entre servicios. Eso es bastante diferente a una arquitectura monolítica, donde puede fácilmente desplegar actualizaciones a múltiples componentes atómicamente.

- **Es difícil decidir cuándo adoptar la arquitectura de microservicios.**

Otro problema con el uso de la arquitectura de microservicio es decidir en qué punto durante el ciclo de vida de la aplicación debe utilizar esta arquitectura. Al desarrollar la primera versión de una aplicación, a menudo no se tienen los problemas que esta arquitectura resuelve. Además, el uso de una arquitectura distribuida y elaborada ralentizará el desarrollo. Ese puede ser un gran dilema para las startups, donde el mayor problema es por lo general, cómo evolucionar rápidamente el modelo de negocio y la aplicación que lo acompaña. El uso de la arquitectura de microservicio hace que sea mucho más difícil iterar rápidamente.

Es casi seguro que la puesta en marcha debería comenzar con una aplicación monolítica. Más adelante, sin embargo, cuando el problema es cómo manejar la complejidad, es cuando tiene sentido descomponer funcionalmente la aplicación en un conjunto de microservicios.

Como se puede ver, la arquitectura de microservicios ofrece muchos beneficios, pero también tiene algunos inconvenientes importantes. Debido a estos problemas, adoptar una arquitectura de microservicios no debe tomarse a la ligera. Pero para aplicaciones complejas, como una aplicación web que atiende consumidores o aplicación SaaS, suele ser la elección correcta. Sitios bien conocidos como Netflix, eBay, Amazon.com, Groupon y Gilt han evolucionado de una arquitectura monolítica a una arquitectura de microservicios.