Arquitecturas de desarrollo

Mateo Arcila Toro

February 17, 2022

1 ¿Qué es una arquitectura de microservicios?

La arquitectura de microservicios es un método de desarrollo de aplicaciones software que funciona como un conjunto de pequeños servicios que se ejecutan de manera independiente y autónoma, proporcionando una funcionalidad de negocio completa. En ella, cada microservicio es un código que puede estar en un lenguaje de programación diferente, y que desempeña una función específica. Los microservicios se comunican entre sí a través de APIs, y cuentan con sistemas de almacenamiento propios, lo que evita la sobrecarga y caída de la aplicación.

Los microservicios han creado infraestructuras IT más adaptables y flexibles. Porque si se quiere modificar solamente un servicio, no es necesario alterar el resto de la infraestructura. Cada uno de los servicios se puede desplegar y modificar sin que ello afecte a otros servicios o aspectos funcionales de la aplicación.

Veamos qué ventajas y desventajas tiene la aplicación de una arquitectura de microservicios frente a otros tipos de arquitectura. Hay que tener en cuenta estos puntos a la hora de identificar qué tipo de arquitectura software será la mejor para un determinado proyecto u organización.

1.1 Ventajas

- Modularidad: al tratarse de servicios autónomos, se pueden desarrollar y desplegar de forma independiente. Además un error en un servicio no debería afectar la capacidad de otros servicios para seguir trabajando según lo previsto.
- Escalabilidad: como es una aplicación modular, se puede escalar horizontalmente cada parte según sea necesario, aumentando el escalado de los módulos que tengan un procesamiento más intensivo.
- Versatilidad: se pueden usar diferentes tecnologías y lenguajes de programación. Lo que permite adaptar cada funcionalidad a la tecnología más adecuada y rentable.
- Rapidez de actuación: el reducido tamaño de los microservicios permite un desarrollo menos costoso, así como el uso de "contenedores de software" permite que el despliegue de la aplicación se pueda llevar a cabo rápidamente.
- Mantenimiento simple y barato: al poder hacerse mejoras de un solo módulo y no tener que intervenir en toda la estructura, el mantenimiento es más sencillo y barato que en otras arquitecturas.
- Agilidad: se pueden utilizar funcionalidades típicas (autenticación, trazabilidad, etc.) que ya han sido desarrolladas por terceros, no hace falta que el desarrollador las cree de nuevo.

1.2 Desventajas

- Alto consumo de memoria: al tener cada microservicio sus propios recursos y bases de datos, consumen más memoria y CPU.
- Inversión de tiempo inicial: al crear la arquitectura, se necesita más tiempo para poder fragmentar los distintos microservicios e implementar la comunicación entre ellos.

- Complejidad en la gestión: si contamos con un gran número de microservicios, será más complicado controlar la gestión e integración de los mismos. Es necesario disponer de una centralización de trazas y herramientas avanzadas de procesamiento de información que permitan tener una visión general de todos los microservicios y orquesten el sistema.
- Perfil de desarrollador: los microservicios requieren desarrolladores experimentados con un nivel muy alto de experiencia y un control exhaustivo de las versiones. Además de conocimiento sobre solución de problemas como latencia en la red o balanceo de cargas.
- No uniformidad: aunque disponer de un equipo tecnológico diferente para cada uno de los servicios tiene sus ventajas, si no se gestiona correctamente, conducirá a un diseño y arquitectura de aplicación poco uniforme.
- Dificultad en la realización de pruebas: debido a que los componentes de la aplicación están distribuidos, las pruebas y test globales son más complicados de realizar.
- Coste de implantación alto: una arquitectura de microservicios puede suponer un alto coste de implantación debido a costes de infraestructura y pruebas distribuidas.

2 Arquitectura SOA

La arquitectura orientada a los servicios (SOA) es un tipo de diseño de software que permite reutilizar sus elementos gracias a las interfaces de servicios que se comunican a través de una red con un lenguaje común.

Un servicio es una unidad autónoma de una o más funciones del software diseñada para realizar una tarea específica, como recuperar cierta información o ejecutar una operación. Contiene las integraciones de datos y código que se necesitan para llevar a cabo una función empresarial completa y diferenciada. Se puede acceder a él de forma remota e interactuar con él o actualizarlo de manera independiente.

En otras palabras, la SOA integra los elementos del software que se implementan y se mantienen por separado, y permite que se comuniquen entre sí y trabajen en conjunto para formar aplicaciones de software en distintos sistemas.

2.1 Ventajas

-
- Comercialización más rápida y mayor flexibilidad: la posibilidad de reutilizar los servicios agiliza y simplifica el proceso de ensamblaje de las aplicaciones. Los desarrolladores no tienen que empezar siempre desde cero, tal como sucede con las aplicaciones monolíticas.
- Uso de la infraestructura heredada en los mercados nuevos: la SOA permite que los desarrolladores tomen las funciones de una plataforma o un entorno y las ajusten e implementen en otros nuevos.
- Reducción de los costos gracias a una mayor agilidad y un desarrollo más eficiente
- Mantenimiento sencillo: dado que todos los servicios son autónomos e independientes, se puede modificar y actualizar cada uno cuando sea necesario, sin afectar al resto.
- Escalabilidad: la SOA posibilita la ejecución de los servicios en varios lenguajes de programación, servicios y plataformas, lo cual aumenta la escalabilidad de forma considerable. Además, utiliza un protocolo de comunicación estandarizado con el que las empresas pueden disminuir la interacción entre los clientes y los servicios, lo cual permite ajustar las aplicaciones con menos presiones e inconvenientes.
- Mayor confiabilidad: la SOA genera aplicaciones más confiables, ya que es más fácil depurar servicios pequeños que un código de gran volumen.
- Gran disponibilidad: las instalaciones de la SOA están disponibles para todos.

2.2 Funciones de la SOA

- Proveedor de servicios Se encarga de crear servicios web, ofrecerlos a un registro de servicios disponibles y gestionar sus condiciones de uso.
- Agente o registro de servicios Se encarga de brindar información acerca del servicio a quien lo solicite, y puede ser público o privado.
- Usuario del servicio o persona que lo solicita Buscará un servicio en el registro o por medio del agente, y se conectará con el proveedor para recibirlo.

3 Arquitectura Monolítica

El estilo arquitectónico monolítico consiste en crear una aplicación autosuficiente que contenga absolutamente toda la funcionalidad necesaria para realizar la tarea para la cual fue diseñada, sin contar con dependencias externas que complementen su funcionalidad. En este sentido, sus componentes trabajan juntos, compartiendo los mismos recursos y memoria. En pocas palabras, una aplicación monolítica es una unidad cohesiva de código.

Un monolítico podrías estar construido como una sola unidad de software o creada a partir de varios módulo o librerías, pero lo que la distingue es que al momento de compilarse se empaqueta como una solo pieza, de tal forma que todos los módulos y librerías se empaquetarán junto con la aplicación principal.

3.1 Características

En esta sección analizaremos las características que distinguen al estilo Monolítico del resto. Las características no son ventajas o desventajas, si no que más bien, nos sirven para identificar cuando una aplicación sigue un determinado estilo arquitectónico.

Las características se pueden convertir en ventajas o desventajas solo cuando analizamos la problemática que vamos a resolver, mientras tanto, son solo características:

1. Son aplicaciones autosuficientes (no requieren de nada para funcionar).
2. Realizan de punta a punta todas las operaciones para terminar una tarea.
3. Son por lo general aplicaciones grandes, un que no es un requisito.
4. Son por lo general silos de datos privados, es decir, cada instalación administra si propia base de datos.
5. Todo el sistema corre sobre una solo plataforma.

3.2 Ventajas

- Fácil de desarrollar: Debido a que solo existe un componente, es muy fácil para un equipo pequeño de desarrollo iniciar un nuevo proyecto y ponerlo en producción rápidamente.
- Fácil de escalar: Solo es necesario instalar la aplicación en varios servidores y ponerlo detrás de un balanceador de cargar.
- Pocos puntos de fallo: El hecho de no depender de nadie más, mitiga gran parte de los errores de comunicación, red, integraciones, etc. Prácticamente los errores que pueden salir son por algún bug del programador, pero no por factores ajenos.
- Autónomo: Las aplicaciones Monolíticas se caracterizan por ser totalmente autónomas (auto suficientes), lo que les permite funcionar independientemente del resto de aplicaciones.
- Performance: Las aplicaciones Monolíticas son significativamente más rápidas debido que todo el procesamiento lo realizan localmente y no requieren consumir procesos distribuidos para completar una tarea.

- Fácil de probar: Debido a que es una sola unidad de código, toda la funcionalidad está disponible desde el inicio de la aplicación, por lo que es posible realizar todas las pruebas necesarias sin depender de nada más.

3.3 Desventajas

- Anclado a un Stack tecnológico: Debido a que todo el software es una sola pieza, implica que utilicemos el mismo Stack tecnológico para absolutamente todo, lo que impide que aprovechemos todas las tecnologías disponibles.
- Escalado Monolítico: Escalar una aplicación Monolítica implica escalar absolutamente toda la aplicación, gastando recursos para funcionalidad que quizás no necesita ser escalada (en el estilo de Microservicios analizaremos como solucionar esto).
- El tamaño sí importa: sin albur, las aplicaciones Monolíticas son fácil de operar con equipo pequeños, pero a medida que la aplicación crece y con ello el equipo de desarrollo, se vuelve cada vez más complicado dividir el trabajo sin afectar funcionalidad que otro miembro del equipo también está moviendo.
- Versión tras versión: Cualquier mínimo cambio en la aplicación implicará realizar una compilación del todo el artefacto y con ello una nueva versión que tendrá que ser administrada.
- Si falla, falla todo: A menos que tengamos alta disponibilidad, si la aplicación Monolítica falla, falla todo el sistema, quedando totalmente inoperable.
- Es fácil perder el rumbo: Por la naturaleza de tener todo en un mismo módulo es fácil caer en malas prácticas de programación, separación de responsabilidades y organización de las clases del código. Puede ser abrumador: En proyectos muy grandes, puede ser abrumador para un nuevo programador hacer un cambio en el sistema.