Monolitos, SOA y Microservicios. ¿Cuál es mejor?

Caranqui Yaguarshungo Kevin Fernando

11 de septiembre de 2023

1. Introducción

En la evolución del desarrollo de software, diversas arquitecturas y enfoques han surgido para abordar los desafíos inherentes a la creación de aplicaciones empresariales robustas y escalables. Entre estos enfoques se encuentran los Monolitos, la Arquitectura Orientada a Servicios (SOA) y los Microservicios, cada uno de los cuales ofrece su propio conjunto de ventajas y desventajas en la construcción de sistemas informáticos Los Monolitos representan una abordaje tradicional en el diseño de software, donde una aplicación se concibe como una única entidad unificada y autónoma. Si bien esta metodología simplifica la gestión inicial del código y la implementación, a medida que el software crece, puede presentar desafíos de escalabilidad y flexibilidad. Por otro lado, SOA se basa en la reutilización de componentes de software a través de interfaces de servicio estandarizadas, permitiendo la comunicación entre aplicaciones de manera eficiente. Sin embargo, su dependencia de estándares y su complejidad pueden plantear obstáculos.

Finalmente, los Microservicios representan un enfoque más moderno en el desarrollo de software, donde la aplicación se compone de servicios autónomos e independientes que se comunican a través de interfaces definidas. Esto permite equipos de desarrollo más pequeños y una mayor agilidad, pero también presenta desafíos como la gestión de un gran número de servicios y pruebas complicadas en un entorno distribuido.

2. ¿Qué son Monolitos, SOA y Microservicios?

2.1. Monolitos

También conocida como arquitectura monolítica, los monolitos son un enfoque tradicional en el diseño de software en el que un programa se construye como una entidad unificada y autónoma, independiente de otras aplicaciones. El término "monolito.ª menudo sugiere algo grande y rígido, lo cual es una representación precisa de las arquitecturas monolíticas en el desarrollo de software. En una arquitectura monolítica, toda la aplicación está interconectada en una única entidad informática con una base de código que abarca todos los aspectos del negocio.

Para realizar cambios en una aplicación de este tipo, se requiere modificar y actualizar toda la estructura, lo que implica acceder a la base de código, compilar y desplegar una versión actualizada del servicio. Esto puede hacer que las actualizaciones sean un proceso restrictivo y lento.

Inicialmente, los enfoques monolíticos pueden ser útiles al comienzo de un proyecto, ya que simplifican la gestión del código y la implementación, permitiendo el despliegue de todas las funciones de la aplicación de una sola vez.

2.1.1. Ventajas

Las arquitecturas monolíticas brindan ventajas distintas a las organizaciones, dependiendo de varios factores. Cuando se opta por desarrollar con una arquitectura monolítica, la principal ventaja radica en la velocidad de desarrollo, gracias a la simplicidad de tener una aplicación construida sobre una única base de código.

1. **Implementación sencilla:**La implementación se simplifica al tratarse de un único archivo o directorio ejecutable.

2. **Desarrollo más ágil:**Es más sencillo desarrollar una aplicación que se compila a partir de una sola base de código.

3. **Mejor rendimiento:**En un entorno con una base de código y repositorio centralizados, una API suele desempeñar la misma función.

4. **Pruebas simplificadas:**Dado que una aplicación monolítica es una unidad única y centralizada, las pruebas integrales pueden llevarse a cabo de manera más rápida que en una aplicación distribuida.

5. **Depuración sencilla:**Con todo el código ubicado en un único lugar, resulta más fácil rastrear solicitudes y localizar problemas o incidencias.

2.1.2. Ventajas

Las aplicaciones monolíticas pueden demostrar su eficacia inicialmente, pero a medida que crecen en tamaño, surgen desafíos significativos en términos de escalabilidad. Realizar incluso pequeñas modificaciones en una sola función implica la necesidad de compilar y probar la totalidad de la plataforma, lo cual entra en conflicto con la agilidad que buscan los desarrolladores en la actualidad.

1. **Desarrollo más lento:** En el caso de aplicaciones grandes el proceso de desarrollo se torna más complejo y más lento.

2. **Limitaciones en la escalabilidad:** Presenta una gran incapacidad para que los componentes sean escalables.

3. **Fiabilidad en riesgo:** Si se presenta un error en algún módulo, podría afectar la disponibilidad de toda la aplicación.

4. **Dificultad en la adopción de nuevas tecnologías:** Cualquier cambio en el framework o el lenguaje de programación impacta en toda la aplicación, lo que a menudo se traduce en cambios costosos y demorados.

5. **Falta de flexibilidad:** Los monolitos se ven restringidos por las tecnologías que se emplean en su construcción.

2.2. Service-Oriented architecture (SOA)

La Arquitectura Orientada a Servicios (SOA) es un enfoque de diseño de software que posibilita la reutilización de componentes de software mediante interfaces de servicio que utilizan un lenguaje de comunicación común a través de una red.

Un servicio se define como una entidad autónoma de funcionalidad de software o un conjunto de funcionalidades diseñadas para llevar a cabo tareas específicas, como la obtención de información determinada o la ejecución de operaciones particulares. Contiene tanto el código como las integraciones de datos necesarios para ejecutar una función empresarial completa y discreta y puede ser accedido, interactuado o actualizado de manera remota e independiente.

2.2.1. Ventajas

1. **Mayor rapidez de Lanzamiento:** La capacidad de reutilizar servicios simplifica y acelera la creación de aplicaciones.

2. **Aprovechamiento de infraestructura heredada:** SOA facilita que los desarrolladores aprovechen la funcionalidad de una plataforma o entorno existente y la adapten y amplíen a nuevos mercados o entornos.

3. **Escalabilidad mejorada:**Dado que todos los servicios son autónomos e independientes, pueden ser modificados y actualizados según sea necesario sin que ello afecte a otros servicios.

4. **Mantenimiento sencillo:**Permite que los servicios se ejecuten en múltiples entornos, plataformas y lenguajes de programación, la capacidad de escalabilidad aumenta significativamente.

2.2.2. Desventajas

1. **Dependencia de estándares:**SOA se apoya en la implementación de estándares, y en su ausencia, la comunicación entre aplicaciones se convierte en un proceso prolongado y requiere una cantidad significativa de programación.
2. **Complejidad creciente:** Conforme avanza la implementación de SOA, se vuelve cada vez más desafiante y costoso cumplir con los protocolos y lograr una comunicación efectiva entre los servicios.
3. **Mayor criticidad de los servicios:** A medida que un servicio de negocio se integra más en la definición de los procesos de negocio, su importancia y criticidad aumentan.

2.3. Microservicios

Los microservicios representan una perspectiva en la arquitectura y la estructura del desarrollo de software en la cual el software se compone de pequeños servicios autónomos que se comunican a través de interfaces de programación de aplicaciones (API) claramente definidas. Estos servicios son administrados por equipos más pequeños e independientes.

Estas arquitecturas, comúnmente llamadas microservicios, simplifican la capacidad de expandirse y agilizan el proceso de creación de aplicaciones. Como resultado, promueven la innovación y aceleran la introducción de nuevas funcionalidades en el mercado.

2.3.1. Ventajas

1. **Equipo de Trabajo mínimo:** Los microservicios permiten que equipos de desarrollo sean más pequeños y manejables, lo que facilita la colaboración y la toma de decisiones ágil. Cada equipo puede centrarse en un servicio específico, lo que agiliza el proceso de desarrollo y promueve una mayor eficiencia en el trabajo.
2. **Escalabilidad:** Brinda la capacidad para escalar de manera individual.
3. **Funcionalidad modular, módulos independientes:** Ya que los microservicios se basan en la descomposición de una aplicación en módulos indepiendientes, facilita la gestión y el mantenimiento de cada función de manera aislada.
4. **Libertad del desarrollador de desarrollar y desplegar servicios de forma independiente:** Los equipos de desarrollo pueden trabajar de manera independiente en sus servicios, lo que brinda a los desarrolladores la libertad de elegir las tecnologías y herramientas que mejor se adapten a sus necesidades.
5. **Uso de contenedores permitiendo el despliegue y el desarrollo de la aplicación rápidamente:** La adopción de contenedores, como Docker, en arquitecturas de microservicios, simplifica el despliegue y el desarrollo de la aplicación.

2.3.2. Desventajas

1. **Alto consumo de memoria:** Pueden consumir más memoria en comparación con una aplicación monolítica. Esto puede requerir recursos adicionales y un diseño cuidadoso para optimizar el uso de memoria.
2. **Necesidad de tiempo para poder fragmentar distintos microservicios:** La fragmentación de una aplicación en microservicios requiere tiempo y planificación cuidadosa.
3. **Complejidad de gestión de un gran número de servicios:** A medida que una aplicación se divide en más microservicios, la gestión de estos servicios se vuelve más compleja.
4. **Necesidad de desarrolladores para la solución de problemas como latencia en la red o balanceo de cargas:** Los microservicios a menudo requieren la atención de desarrolladores para abordar problemas como la latencia en la red o el balanceo de cargas.
5. **Pruebas o testeos complicados al despliegue distribuido:** La naturaleza distribuida de los microservicios puede complicar las pruebas y el monitoreo.

3. Conclusiones

En resumen, los Monolitos, la Arquitectura Orientada a Servicios (SOA) y los Microservicios representan tres enfoques distintos en el desarrollo de software, cada uno con sus propias ventajas y desventajas.

- **Monolitos:** Son sistemas de software construidos como una única entidad, lo que inicialmente simplifica la gestión y el despliegue. Sin embargo, a medida que crecen, pueden volverse difíciles de escalar y actualizar, lo que va en contra de la agilidad deseada en el desarrollo.
- **SOA:** Se centra en la reutilización de componentes de software a través de interfaces de servicio, permitiendo una mayor rapidez en el desarrollo y el aprovechamiento de la infraestructura existente. Aunque ofrece ventajas significativas, la dependencia de estándares y la complejidad de gestión pueden ser desafíos.
- **Microservicios:** Dividen las aplicaciones en servicios autónomos, lo que facilita la escalabilidad y el desarrollo ágil. Esto permite una rápida innovación y tiempos de comercialización más cortos, pero también conlleva desafíos como la gestión de un gran número de servicios y pruebas más complicadas.

Dado a esto, la elección entre estos enfoques depende de las necesidades y objetivos específicos de desarrollo de software de una organización, así como de la etapa del proyecto y de su capacidad para gestionar las complejidades y desafíos asociados a cada enfoque.