Consultas de microservicios

1

¿Qué son los microservicios?

Los microservicios son módulos ligeros con acople suelto que pueden servir como componentes básicos de las aplicaciones complejas basadas en la nube. aunque los microservicios individuales pueden operar de forma independiente, tienen un acople suelto en una interfaz unificada.

la arquitectura de microservicios se considera un reemplazo moderno y flexible del modelo de desarrollo más tradicional de la arquitectura monolítica.

en una arquitectura monolítica, generalmente hay un servidor virtualizado o físico asignado a cada aplicación y esos servidores siempre están en ejecución. la disponibilidad y el escalamiento de la aplicación dependen completamente del hardware subyacente, que es una entidad fija.

los microservicios, por el contrario, pueden operar varias instancias en un solo servidor o en varios servidores, a medida que los recursos escalan de forma dinámica para admitir las demandas de la carga de trabajo. los microservicios individuales suelen estar en contenedores para mejorar la portabilidad y la escalabilidad.

Esas características incluyen:

* **Componentes.** Los microservicios suelen estar formados por componentes de software discretos que son actualizables y reemplazables de forma individual. Esta arquitectura tiene implicaciones para la [tecnología de administración de la nube](https://www.intel.la/content/www/xl/es/cloud-computing/cloud-management-technology.html) porque cada uno de los microservicios debe estar aprovisionado, monitoreado y actualizado por separado.
* **Servicios.** Los componentes comprenden servicios que están disponibles para comunicarse a pedido, pero pueden no estar activos de forma continua entre las solicitudes o las llamadas.
* **Implementación independiente.** En su mayor parte, los componentes de servicios individuales operan de forma independiente uno de otro dentro del marco de microservicios. Si se cambia o actualiza un componente, hay poco impacto en otros servicios y componentes, especialmente cuando se compara con una arquitectura monolítica más tradicional.
* **Seguridad.** La comunicación entre los microservicios suele estar cifrada con seguridad de la capa de transporte mutuo (mTLS) para proteger los datos del malware y las intrusiones mientras está en tránsito.
* **Contenerización.** Los microservicios suelen implementarse en [contenedores](https://www.intel.la/content/www/us/en/cloud-computing/containers.html) para lograr escalabilidad y portabilidad adicionales.

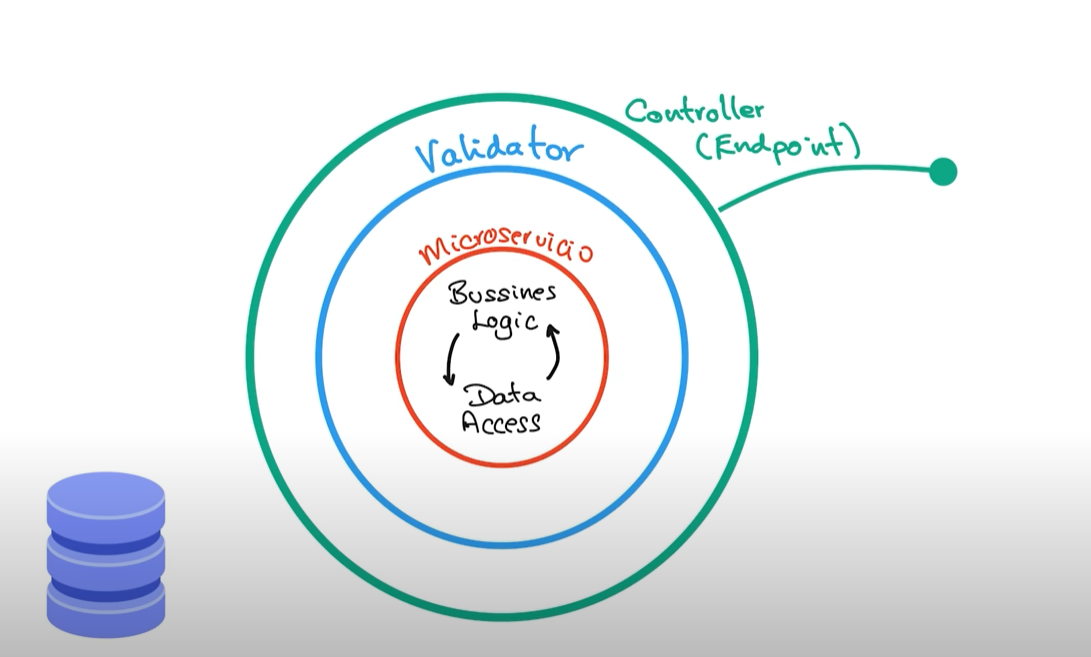
La implementación de microservicios suele seguir uno de tres patrones:

* **Nativo de la nube.** Algunos servicios y aplicaciones establecidos de alto volumen comienzan como microservicios y permanecen en la nube. Según International Data Corporation (IDC), alrededor del 56 % de los microservicios son nativos de la nube, mientras que el 44 % restante se originaron como aplicaciones antiguas.
* **Refactorización y cambio.** Estas implementaciones comienzan en las instalaciones o en un centro de datos en el perímetro y se refactorizan para adaptarse a la arquitectura de microservicios basados en la nube. La refactorización puede incluir reasignar bases de datos y otros recursos asociados con la arquitectura monolítica, de modo que se combinen con los microservicios correspondientes.
* **Lift and shift.** Algunas organizaciones migran sus aplicaciones a una arquitectura de microservicios, sin refactorización, en una simple transición “lift and shift”.

Los arquetipos de microservicios son atractivos para:

* aplicaciones y sitios web con sistemas de datos gigantescos de rápido crecimiento;
* equipos de expertos de desarrollo que se encuentran dispersos, a menudo internacionalmente;
* sitios web con componentes pequeños;
* equipos de desarrolladores que quieran cambiar una aplicación o web monolítica a una más sostenible en el tiempo.

2 estructura básica de un microservicio



Principios Clave para el Diseño

1. **Negocio como Servicio:** Cada microservicio debe alinearse con una capacidad de negocio específica.
2. **Autonomía:** Los servicios deben ser independientes y poder desplegarse y escalarse por separado.
3. **Tamaño Adecuado:** Los servicios deben ser lo suficientemente pequeños para ser manejables, pero lo suficientemente grandes para tener un propósito claro.
4. **Comunicación Ligera:** Utilizar protocolos ligeros como HTTP o mensajería asincrónica.
5. **Gestión de Datos:** Cada servicio debe tener su propia base de datos.
6. **Descentralización:** La toma de decisiones debe ser descentralizada en cada equipo.

Pasos para Diseñar un Sistema de Microservicios

1. **Identificar los Bounded Contexts:** Utilizar técnicas de Domain-Driven Design (DDD) para identificar los límites de cada servicio.
2. **Definir APIs:** Crear APIs claras y bien documentadas para cada servicio.
3. **Implementar Servicios:** Desarrollar cada servicio de forma independiente, utilizando la tecnología más adecuada.
4. **Gestionar la Comunicación:** Implementar un mecanismo de comunicación entre servicios (REST, gRPC, mensajería).
5. **Gestión de Errores:** Implementar mecanismos de tolerancia a fallos y recuperación de errores.
6. **Observabilidad:** Implementar herramientas de monitoreo y login para entender el comportamiento del sistema.

Consideraciones Adicionales

* **Gestión de la Configuración:** Utilizar herramientas de gestión de configuración para centralizar la configuración de los servicios.
* **Descubrimiento de Servicios:** Implementar un mecanismo para que los servicios se descubran entre sí.
* **Seguridad:** Implementar medidas de seguridad robustas para proteger los servicios.
* **Pruebas:** Realizar pruebas exhaustivas en cada servicio y en el sistema completo.

Herramientas y Tecnologías

* **Orquestación:** Kubernetes, Docker Swarm
* **Comunicación:** API Gateway, Kafka
* **Base de Datos:** MongoDB, Cassandra, PostgreSQL
* **Lenguajes:** Java, Go, Node.js, Python

Desafíos Comunes y Soluciones

* **Complejidad:** La gestión de muchos servicios puede ser compleja. Solución: Herramientas de orquestación y automatización.
* **Consistencia de Datos:** Mantener la consistencia de datos entre servicios puede ser difícil. Solución: Event Sourcing, Sagas.
* **Testing:** Probar sistemas distribuidos es complejo. Solución: Pruebas de contrato, pruebas de integración.
* **Dependencia:** Los servicios pueden volverse demasiado dependientes entre sí. Solución: Diseño cuidadoso de las APIs y límites de contexto.

3

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.

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 estos. 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.