**MICROSERVICIOS**

Los microservicios son tanto un estilo de arquitectura como un modo de programar software. Con los microservicios, las aplicaciones se dividen en sus elementos más pequeños e independientes entre sí. A diferencia del enfoque tradicional y monolítico de las aplicaciones, en el que todo se compila en una sola pieza, los microservicios son elementos independientes que funcionan en conjunto para llevar a cabo las mismas tareas. Cada uno de esos elementos o procesos es un microservicio. Este enfoque de desarrollo de software valora el nivel de detalle, la sencillez y la capacidad para compartir un proceso similar en varias aplicaciones. Es un elemento fundamental de la optimización del desarrollo de aplicaciones hacia un modelo nativo de la nube.

Los microservicios son un enfoque arquitectónico y organizativo para el desarrollo de software donde el software está compuesto por pequeños servicios independientes que se comunican a través de API bien definidas. Los propietarios de estos servicios son equipos pequeños independientes.

Las arquitecturas de microservicios hacen que las aplicaciones sean más fáciles de escalar y más rápidas de desarrollar. Esto permite la innovación y acelera el tiempo de comercialización de las nuevas características.

**Arquitectura monolítica en comparación con la arquitectura de microservicios**

Diagrama

Descripción generada automáticamenteCon las arquitecturas monolíticas, todos los procesos están estrechamente asociados y se ejecutan como un solo servicio. Esto significa que, si un proceso de una aplicación experimenta un pico de demanda, se debe escalar toda la arquitectura. Agregar o mejorar las características de una aplicación monolítica se vuelve más complejo a medida que crece la base de código. Esta complejidad limita la experimentación y dificulta la implementación de nuevas ideas. Las arquitecturas monolíticas aumentan el riesgo de la disponibilidad de la aplicación porque muchos procesos dependientes y estrechamente vinculados aumentan el impacto del error de un proceso.

Con una arquitectura de microservicios, una aplicación se crea con componentes independientes que ejecutan cada proceso de la aplicación como un servicio. Estos servicios se comunican a través de una interfaz bien definida mediante API ligeras. Los servicios se crean para las capacidades empresariales y cada servicio desempeña una sola función. Debido a que se ejecutan de forma independiente, cada servicio se puede actualizar, implementar y escalar para satisfacer la demanda de funciones específicas de una aplicación.

**Beneficios empresariales de los microservicios**.

**Implementables de forma independiente**

Tal vez la característica más importante de los microservicios es que, debido a que los servicios son más pequeños e independientemente implementables, ya no requieren una legislación para cambiar una línea de código o añadir una nueva característica en una aplicación.

Los microservicios prometen a las organizaciones un antídoto contra la frustración que provoca que la aplicación de pequeños cambios requiera grandes cantidades de tiempo. No necesita un doctorado en ciencias de la computación para ver o entender el valor de un enfoque que facilita más la velocidad y la agilidad.

Pero la velocidad no es el único valor de diseñar servicios de esta manera. Un modelo organizativo emergente común es reunir equipos multifuncionales en torno a un problema empresarial, servicio o producto. El modelo de microservicios encaja perfectamente con esta tendencia, ya que permite a una organización crear equipos pequeños y multifuncionales en torno a un servicio o una colección de servicios y hacer que funcionen de forma ágil.

El acoplamiento suelto de microservicios también crea un grado de aislamiento de fallas y una mejor resiliencia en las aplicaciones. Y el pequeño tamaño de los servicios, combinado con sus claros límites y patrones de comunicación, hace que sea más fácil para los nuevos miembros del equipo entender la base de código y contribuir con ella rápidamente, un claro beneficio en términos de la velocidad y moral de los empleados.

**La herramienta adecuada para el trabajo**

En los patrones de arquitectura tradicionales de n niveles, una aplicación normalmente comparte un lote común, con una gran [base de datos relacional](https://www.ibm.com/mx-es/topics/relational-databases) que da soporte a toda la aplicación. Este enfoque tiene varios inconvenientes obvios. El más significativo es que cada componente de una aplicación debe compartir un recurso común, un modelo de datos y una base de datos, incluso si hay una herramienta clara y mejor para el trabajo para determinados elementos. Esto perjudica a la arquitectura y es frustrante para los desarrolladores, que son conscientes en todo momento de que hay una forma mejor y más eficaz crear estos componentes.

Por el contrario, en un modelo de microservicios, los componentes se implementan de forma independiente y se comunican a través de una combinación de REST, streaming de eventos y Message Brokers, por lo que es posible que el lote de cada servicio individual esté optimizado para ese servicio. La tecnología cambia todo el tiempo, y es mucho más fácil y menos costoso que una aplicación formada por múltiples servicios más pequeños evolucione con la tecnología a medida que este avance.

**Escalado preciso**

Con los microservicios, los servicios individuales se pueden implementar individualmente, pero también se pueden escalar individualmente. Hecho correctamente, los microservicios requieren menos infraestructura que las aplicaciones monolíticas porque permiten el escalado preciso de sólo los componentes que lo requieren, en lugar de toda la aplicación, como es el caso de aplicaciones monolíticas.

**También existen algunos desafíos**

Los beneficios significativos de los microservicios vienen con desafíos significativos. Cambiar de una aplicación monolítica a los microservicios implica una mayor complejidad de gestión: muchos más servicios, creados por muchos más equipos, implementados en muchos más lugares. Los problemas en un servicio pueden causar, o ser causados por, problemas en otros servicios. Los datos de registro (utilizados para supervisión y resolución de problemas) son más voluminosos y pueden ser inconsistentes entre los servicios. Las nuevas versiones pueden causar problemas de compatibilidad con versiones anteriores. Las aplicaciones requieren más conexiones de red, lo que significa más oportunidades para que ocurran problemas de latencia y conectividad. Un enfoque de DevOps (como se lee más abajo) puede abordar muchos de estos problemas, pero la adopción de DevOps tiene retos propios.

Sin embargo, estos desafíos no impiden que los no usuarios adopten el uso de microservicios, o que los usuarios aumenten sus compromisos de microservicios.

**Tecnologías y herramientas clave**

Mientras que casi cualquier herramienta o lenguaje moderno puede ser utilizado en una arquitectura de microservicios, hay un puñado de herramientas básicas que se han vuelto esenciales y prácticamente definitivas para los microservicios:

**Contenedores, Docker y Kubernetes**

Uno de los elementos clave de un microservicio es que, en general, es bastante pequeño. (No hay una cantidad arbitraria de código que determine si algo es o no un microservicio, pero "micro" está incluido por algo en el nombre).

Cuando Docker inició la era moderna de contenedores en 2013, también introdujo el modelo de computación que se asociaría más estrechamente con los microservicios. Como los contenedores individuales no tienen la sobrecarga de su propio sistema operativo, son más pequeños y ligeros que las tradicionales máquinas virtuales y pueden girar hacia arriba y hacia abajo más rápidamente, lo que los convierte en una combinación perfecta para los servicios más pequeños y ligeros que se encuentran dentro de las arquitecturas de microservicios.

Con la proliferación de servicios y contenedores, la orquestación y la gestión de grandes grupos de contenedores rápidamente se convirtieron en uno de los principales retos. Kubernetes, una plataforma de orquestación de contenedores de código abierto, se ha convertido en una de las soluciones de orquestación más populares porque realiza ese trabajo muy bien.

**Gateways de API**

Los microservicios a menudo se comunican a través de API, especialmente cuando se establece el estado por primera vez. Si bien es cierto que los clientes y los servicios pueden comunicarse entre sí directamente, los gateways de API suelen ser una capa intermedia útil, especialmente a medida que el número de servicios en una aplicación crece con el tiempo. Una gateway de API actúa como un proxy inverso para los clientes mediante solicitudes de direccionamiento, distribuyendo las solicitudes por varios servicios y proporcionando seguridad y autenticación adicionales.

Existen varias tecnologías que se pueden utilizar para implementar gateways de API, incluidas las plataformas de gestión de API, pero si la arquitectura de microservicios se implementa mediante contenedores y Kubernetes, el gateway se implementa normalmente mediante Ingress o, más recientemente, Istio.

**Mensajería y streaming de eventos**

Aunque la práctica recomendada sería el diseño de servicios sin estado, el estado existe y los servicios deben tenerlo en cuenta. Y aunque una llamada de API es a menudo una forma efectiva de establecer inicialmente el estado de un determinado servicio, no es una manera particularmente efectiva de mantenerse al día. El enfoque "¿ya llegamos?" de sondeo constante para mantener los servicios actualizados no resulta práctico.

En su lugar, es necesario acoplar las llamadas de API que establecen el estado con la mensajería o el streaming de eventos para que los servicios puedan emitir cambios en el estado y otros stakeholders puedan escuchar esos cambios y ajustarse a ellos. Es probable que este trabajo sea el más adecuado para un message broker de uso general, pero hay casos en los que una plataforma de streaming de eventos como, por ejemplo, Apache Kafka, puede ser una buena solución. Y con la combinación de microservicios con la arquitectura impulsada por eventos, los desarrolladores pueden crear sistemas distribuidos, altamente escalables, tolerantes a errores y ampliables que pueden consumir y procesar grandes cantidades de eventos o información en tiempo real.

**Sin servidor**

Las arquitecturas sin servidor llevan algunos de los patrones de nube y microservicios principales a su conclusión lógica. En el caso de sin servidor, la unidad de ejecución no es solo un pequeño servicio, sino una función, que a menudo puede ser solo unas pocas líneas de código. La línea que separa una función sin servidor de un microservicio es borrosa, pero comúnmente se entiende que las funciones son incluso más pequeñas que un microservicio.

El aspecto en el cual las arquitecturas sin servidor y las plataformas de funciones como servicio (FaaS) comparten afinidad con los microservicios es que ambas están interesadas en crear unidades de implementación más pequeñas y escalar con precisión en relación a la demanda.

**Patrones comunes**

Dentro de las arquitecturas de microservicios, hay muchos patrones comunes y útiles de diseño, comunicación e integración que ayudan a abordar algunos de los retos y oportunidades más comunes, incluyendo los siguientes:

* Patrón back-end-for-frontend (BFF). Este patrón inserta una capa entre la experiencia del usuario y los recursos a los que recurre la experiencia. Por ejemplo, una aplicación utilizada en un desktop tendrá diferentes tamaños de pantalla, visualización y límites de rendimiento que un dispositivo móvil. El patrón BFF permite a los desarrolladores crear y dar soporte a un tipo de programa de backend por interfaz de usuario con el uso de las mejores opciones para esa interfaz, en lugar de intentar dar soporte a un programa de backend genérico que funcione con cualquier interfaz, pero que pueda afectar negativamente al rendimiento de frontend.
* Entidad y patrones agregados. Una entidad es un objeto distinguido por su identidad. Por ejemplo, en un sitio de comercio electrónico, un objeto de producto puede distinguirse por nombre de producto, tipo y precio. Un agregado es un grupo de entidades relacionadas que deben tratarse como una unidad. Por lo tanto, para el sitio de comercio electrónico, un pedido sería un grupo (agregado) de productos (entidades) solicitados por un comprador. Estos patrones se utilizan para clasificar datos de forma significativa
* Patrones de descubrimiento de servicios. Estas aplicaciones y servicios de ayuda se encuentran entre sí. En una arquitectura de microservicios, las instancias de servicio cambian dinámicamente debido al escalado, las actualizaciones, la anomalía de servicio e incluso el término del servicio. Estos patrones proporcionan mecanismos de descubrimiento para hacer frente a esta transición. El equilibrio de carga puede utilizar patrones de descubrimiento de servicios mediante comprobaciones de estado y fallos del servicio como desencadenantes para reequilibrar el tráfico.
* Patrones de microservicios del adaptador. Piense en los patrones de adaptador como si fueran los adaptadores de enchufes que utiliza al viajar a otro país. La finalidad de los patrones de adaptador es ayudar a convertir las relaciones entre clases u objetos que de otro modo son incompatibles. Una aplicación que depende de API de terceros puede necesitar utilizar un patrón de adaptador para garantizar que la aplicación y las API puedan comunicarse.
* Patrón de aplicación Strangler. Estos patrones ayudan a gestionar la refactorización de una aplicación monolítica en aplicaciones de microservicios. El nombre Strangler (estrangulador) se refiere a cómo una vid (microservicios) lentamente y con el tiempo supera y estrangula un árbol (una aplicación monolítica).