Engineer

Microservicios

Arquitectura de servicios pequeños desplegados por separado

Qué es

Los microservicios son un estilo arquitectónico que divide una aplicación en servicios pequeños. Cada uno es un programa propio que se ejecuta como un proceso separado y puede desplegarse sin los demás. James Lewis y Martin Fowler, cuyo artículo de 2014 dio al estilo su descripción más conocida, lo resumen como construir aplicaciones en «conjuntos de servicios desplegables de forma independiente». Enumeran los rasgos habituales: comunicación por mecanismos ligeros, a menudo una API HTTP de recursos; organización en torno a capacidades de negocio; despliegue completamente automatizado; gestión central mínima; y libertad para que cada servicio use su propio lenguaje de programación y su propio almacenamiento. Advierten expresamente de que no hay una definición formal, sino características que comparten la mayoría de los sistemas con esa etiqueta.

La palabra es más reciente que la idea. Una nota al pie del mismo artículo documenta que un taller de arquitectos de software cerca de Venecia debatió el término en mayo de 2011 y que el mismo grupo eligió «microservices» en mayo de 2012. Adrian Cockcroft, de Netflix, ya llamaba al enfoque una forma de SOA de grano fino. Lewis y Fowler no afirman que el estilo sea nuevo: sitúan sus raíces, como mínimo, en los principios con los que se construyó Unix.

La alternativa con la que suele compararse es el monolito: una aplicación de servidor que se construye y despliega como una unidad. Toda la lógica de una solicitud se ejecuta en un proceso. Con cierto cuidado, la aplicación completa puede ejecutarse y probarse en el portátil de un desarrollador, y escala ejecutando más copias detrás de un equilibrador de carga. La fricción llega después. Un cambio pequeño exige reconstruir y desplegarlo todo; cuesta mantener una buena estructura modular durante años y la parte que necesita más capacidad solo puede escalarse junto con todo lo demás.

Lewis y Fowler definen un componente como una unidad de software que puede sustituirse y actualizarse por separado. Una biblioteca se enlaza con un programa y se llama en memoria. Un servicio es un componente externo al proceso, accesible por red mediante una solicitud web o una llamada a procedimiento remoto. La diferencia importa para el acoplamiento. En la mayoría de los lenguajes, solo la documentación y la disciplina impiden que otro código acceda al interior de una biblioteca; un servicio solo admite llamadas remotas explícitas, de modo que es mucho más difícil eludir su interfaz. Aun así, un servicio puede comprender más de un proceso, como una aplicación junto con una base de datos que solo ella utiliza.

«Micro» dice menos de lo que parece. Lewis y Fowler señalan que el nombre dirige la atención al tamaño y provoca discusiones sobre qué cuenta como micro. Los servicios más grandes de los que habían oído hablar seguían el modelo del equipo de dos pizzas de Amazon, es decir, no más de una docena de personas; en el extremo pequeño, vieron un equipo de media docena a cargo de media docena de servicios.

Utilice el concepto en una decisión

Aplique «¿Qué son los microservicios?» a una decisión real. Podemos ayudarle a relacionar el concepto con el sistema que está construyendo o comprando. La parte que el usuario no ve: dónde se guardan los datos, qué reglas se aplican y cómo pueden comunicarse otros sistemas con ella.

Explore nuestros servicios Póngase en contacto →

Compruebe los supuestos detrás del término

Utilice la explicación y los trabajos enlazados para formular preguntas sobre su proyecto. Podemos examinar los supuestos con usted antes de elegir una herramienta. Diseño y operación de 6 flujos ETL/ELT — 6 flujos ETL en tres motores de bases de datos

Diseño y operación de 6 flujos ETL/ELT Póngase en contacto →

Por qué importa

La primera ventaja es publicar con un calendario propio. Cuando una aplicación se divide en servicios, muchos cambios solo requieren volver a desplegar uno: un equipo puede publicar sin esperar a los demás. Lewis y Fowler aclaran que no es absoluto: los cambios de interfaz de un servicio deben seguir coordinándose con los servicios que lo llaman.

La segunda es que el software puede seguir la organización, en lugar de luchar contra ella. En 1968, Melvin Conway sostuvo en Datamation que las organizaciones que diseñan sistemas están limitadas a producir diseños que reproducen sus estructuras de comunicación. Lewis y Fowler lo aplican a la división habitual de una aplicación grande por capas técnicas, con un equipo para

la interfaz, otro para la lógica del servidor y otro para la base de datos. Hasta un cambio simple se convierte entonces en un proyecto entre equipos y la lógica acaba donde algún equipo tenga acceso. Los microservicios invierten el enfoque. Cada servicio cubre una capacidad de negocio, como pedidos, existencias o pagos, desde la interfaz hasta el almacenamiento; pertenece a un equipo multidisciplinar con las competencias necesarias para toda esa porción. Ese equipo debería responsabilizarse del servicio durante todo su ciclo de vida, en lugar de entregarlo al acabar el proyecto. La versión de Amazon de la regla es que quien construye un servicio también lo opera en producción.

La tercera es que la lógica permanece en los servicios. Lewis y Fowler resumen la preferencia como extremos inteligentes y conductos simples. Los servicios se comunican mediante solicitudes HTTP a API de recursos o mediante un bus de mensajes ligero que solo encamina mensajes; las decisiones se toman en los servicios de ambos extremos. Es un alejamiento deliberado del bus de servicios empresariales, que encamina, transforma y aplica reglas de negocio en medio de la red, en lugar de en sus extremos.

Los datos se dividen de la misma manera. Cada servicio gestiona su propia base de datos y servicios distintos pueden usar sistemas de bases de datos distintos si conviene a su trabajo. El diseño guiado por el dominio de Eric Evans aporta el razonamiento: un sistema grande contiene varios modelos y cada uno necesita un contexto delimitado explícitamente, en términos de organización del equipo, de las partes de la aplicación que gobierna y de elementos físicos como bases de código y esquemas de base de datos. Las visiones de un cliente desde ventas y desde soporte pueden diferir sin que nadie tenga que reconciliarlas en una tabla.

Los costes forman parte de la definición y Lewis y Fowler los exponen con claridad. Una llamada remota cuesta mucho más que una llamada en memoria, por lo que las interfaces entre servicios deben tener un grano más grueso. Mover código a través de una frontera de servicio es difícil: hay que coordinar cambios de interfaz, añadir capas de compatibilidad hacia atrás y asumir pruebas más complejas. Las transacciones distribuidas son difíciles de implementar; los servicios se coordinan sin ellas, aceptan que la consistencia quizá solo sea eventual y reparan errores con operaciones compensatorias. Cualquier llamada puede fallar. Si los servicios no encajan bien, la complejidad no ha desaparecido: se ha trasladado a las conexiones, donde es más difícil verla. Lewis y Fowler recogen un argumento que consideran razonable: construir primero un monolito, darle módulos claros y dividirlo solo cuando empiece a causar problemas. Añaden una salvedad: una interfaz que funciona bien dentro de un proceso rara vez es buena entre servicios.

Cómo se aplica en la práctica

La nota Vender la última entrada exactamente una vez recurre a un servicio solo cuando una base de datos deja de bastar; entonces formula la regla que sostiene el diseño: cada artículo tiene exactamente un responsable de sus existencias, el servicio de inventario, y ningún otro componente decide. Ese servicio se particiona por organización cliente y artículo, y las organizaciones cliente más grandes reciben celdas propias. Solo la reserva de existencias debe ser síncrona. El pago, el pedido y el recibo son sistemas separados, sin una transacción que los abarque. La nota rechaza la confirmación en dos fases porque bloquea cuando cualquier participante va lento y los proveedores de pagos no participan; utiliza una saga: una secuencia de pasos locales, cada uno con otro que lo deshace.

Cada paso escribe su mensaje en una tabla de salida, u outbox, dentro de la misma transacción que el cambio anunciado. En PostgreSQL, una sentencia que modifica datos dentro de `WITH` puede pasar su resultado a una inserción, de modo que el servicio de inventario reserva una unidad y registra el mensaje con una sola sentencia:

```
-- inventory service, its own database
WITH held AS (
    UPDATE inventory
    SET available = available - 1
    WHERE tenant_id = 't-7' AND item_id = 'item-881'
      AND available >= 1
    RETURNING item_id
)
INSERT INTO outbox (order_id, item_id, event, units)
SELECT 'A-17', item_id, 'StockReserved', 1 FROM held;
-- sold out: no row matches, so no message is written
```

La captura de cambios de datos publica después las filas de la tabla de salida; cada consumidor es idempotente, con el pedido como clave, porque los mensajes llegan al menos una vez. En este esquema, ningún otro servicio toca la tabla `inventory` : los demás conocen las existencias a través de los mensajes o consultando al servicio de inventario.

La misma nota también sitúa la separación en el orden de trabajo. En su lista de prioridades, dar al inventario un servicio y un responsable propios ocupa el cuarto lugar: después de medir, de convertir la escritura en una sola sentencia

condicional y de reducir la carga con una sala de espera, una puerta de admisión y una respuesta «agotado» en caché. Las soluciones más baratas van primero.

El caso del portafolio Una plataforma marítima construida en capas claras documenta la contribución arquitectónica de la fundadora en NextMariner y muestra la otra cara de la decisión. Era un producto joven cuyos requisitos iban a cambiar mucho; el objetivo era que modificar una regla de negocio no exigiese tocar a la vez la interfaz, la API y la base de datos. Se eligieron tres capas, cada una con una función: una interfaz Next.js para la presentación, una API Go deliberadamente ligera que encamina, valida, filtra por seguridad y orquesta, y toda la lógica de negocio en funciones PostgreSQL. Los límites se documentaron y se exigieron en las revisiones. Lewis y Fowler señalan que mantener fronteras modulares dentro de una aplicación exige mucha disciplina y que la separación más explícita de los servicios facilita mantener claros los límites de los equipos. Esta plataforma conservó esa disciplina sin dividir su lógica de negocio en servicios separados.

Dónde surgen los problemas

Las cadenas de llamadas síncronas multiplican la indisponibilidad. Si responder a una solicitud exige que un servicio llame a otro y este a un tercero, solo tiene éxito mientras los tres estén disponibles y espera a cada uno por turno. Lewis y Fowler lo llaman efecto multiplicativo de la indisponibilidad y citan la regla de The Guardian de una llamada síncrona por solicitud de usuario. Mantenga el recorrido síncrono tan corto como en la nota: solo la reserva espera respuesta y todo lo posterior es un mensaje. Dé a cada llamada restante un tiempo de espera y un cortacircuitos, los patrones que Lewis y Fowler señalan en el libro Release It!, para aislar un servicio que falla en lugar de esperarlo.

Servicios que siempre cambian juntos. Separar por fronteras equivocadas produce servicios que no pueden publicarse de forma independiente: cada funcionalidad afecta a tres y se publican al unísono. Lewis y Fowler dan la prueba: si dos servicios cambian juntos continuamente, deberían fusionarse. Un fallo relacionado consiste en convertir, una por una, llamadas a métodos en llamadas remotas; deja servicios que conversan sin parar por la red y rinden mal. Trace fronteras alrededor de capacidades de negocio y contextos delimitados, diseñe interfaces de grano grueso y haga que cada servicio tolere cambios en aquellos a los que llama, de modo que un servicio proveedor pueda evolucionar sin hacer fallar a sus consumidores. Lewis y Fowler consideran el versionado un último recurso.

Dos servicios comparten una base de datos. Si el servicio de pedidos lee y escribe directamente las tablas de inventario, el esquema se convierte en una interfaz sin responsable; ningún equipo puede cambiarlo por sí solo y aparece un segundo componente que modifica los datos de existencias sin que las comprobaciones del servicio de inventario lo detecten. Dé a cada dato un responsable, como hace la nota con las existencias, y haga que los demás servicios lo consulten o lean sus mensajes. Si una actualización abarca varios servicios, use una saga con pasos compensatorios en lugar de una transacción distribuida.

Nadie puede seguir una solicitud. Una solicitud que atraviesa varios servicios deja un fragmento de su historia en cada log. Lewis y Fowler dan un papel central a la monitorización en tiempo real: cifras técnicas como las solicitudes por segundo a la base de datos se vigilan junto a cifras de negocio como pedidos por minuto. También advierten de que los servicios que reaccionan a eventos mutuos producen comportamientos emergentes que la monitorización debe detectar pronto. Trace todas las solicitudes. En el modelo de OpenTelemetry, una traza es el recorrido de una solicitud por el sistema, formada por tramos que comparten un identificador de traza e indican su padre; la propagación del contexto permite unir tramos de servicios distintos en una traza. Pase el contexto en cada llamada y cada mensaje, incluidos los que pasan por la tabla de salida, o la traza terminará en el primer salto asíncrono.

Lecturas adicionales

- Microservicios — James Lewis y Martin Fowler, 2014: las nueve características, la historia del nombre, los recuadros sobre tamaño y llamadas síncronas y los motivos para actuar con cautela.
- ¿Cómo inventan los comités? — el artículo de Melvin Conway de 1968 en Datamation, donde la estructura de un sistema reproduce la comunicación de la organización que lo diseñó.
- Referencia de diseño guiado por el dominio — el resumen de patrones de Eric Evans, incluidos el contexto delimitado y los mapas de relaciones entre contextos.
- OpenTelemetry: trazas — tramos, identificadores de traza, tramos padre y propagación del contexto: las piezas que hacen visible una solicitud entre servicios.
- PostgreSQL: consultas WITH — sentencias que modifican datos dentro de `WITH`, que permiten cambiar una fila y registrar el cambio en otro lugar con una sentencia.

Glosario

Elija según sus restricciones reales

Describa el flujo de trabajo, las interfaces existentes y los usuarios que necesitan el cambio. Podemos definir una entrega comprobable y un traspaso que permita a su equipo mantener lo entregado.

Vea cuánto cuesta Póngase en contacto →

Haga práctica la terminología

Pregunte cómo se aplica «¿Qué son los microservicios?» a su organización. Basta un ejemplo concreto de su sistema actual para iniciar la conversación.

Explore nuestros servicios

welcome@engineer.company +45 50 10 61 84 Copenhague, Dinamarca

Llévese esta página

Tarjeta de contacto

Creado con esmero y cariño por D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Derechos de autor © 2025 – actualidad · Engineer ApS · Todos los derechos reservados.

Impreso desde https://engineer.company/es/glossary/microservices/