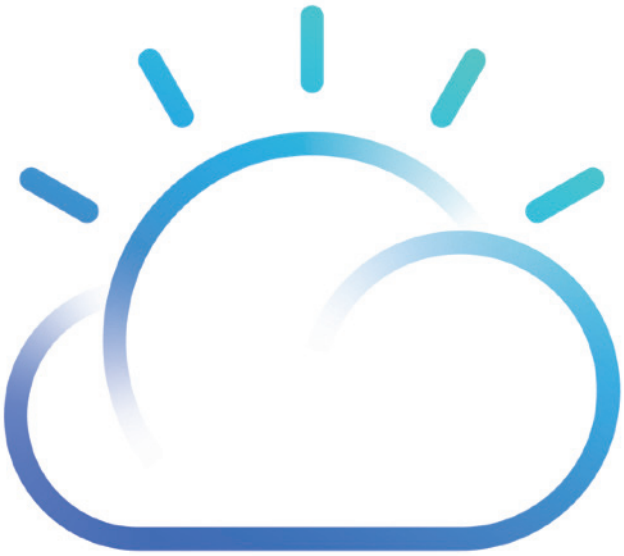
IBM Cloud

Guía para punto de vista de microservicios

Entender los microservicios



Roland Barcia – Ingeniero distinguido de IBM, CTO Microservices, NYC IBM Cloud Garage
Kyle Brown – Ingeniero distinguido de IBM, CTO Cloud Architecture
Richard Osowski – Miembro sénior del personal técnico de IBM, Microservices Adoption

Contenido

Visión general

En un estilo de arquitectura de aplicación de microservicios, una aplicación está compuesta por componentes discretos, conectados en red, denominados *microservicios*. El estilo arquitectónico de microservicios es una evolución del estilo arquitectónico SOA (Arquitectura orientada a los servicios). Las aplicaciones construidas mediante los servicios SOA tienden a enfocarse en problemas de integración técnica, y el nivel de los servicios implementados es a menudo el de las interfaces de programación de aplicaciones (API) técnicas de grano fino. En contraste, el enfoque de microservicios implementa claras capacidades empresariales a través de API empresariales de grano grueso.

La mayor diferencia entre los dos enfoques es la manera en la que son desplegados. Durante muchos años, las aplicaciones han sido empaquetadas de manera monolítica —un equipo de desarrolladores construye una aplicación grande que hace todo lo que requiere una necesidad de la empresa. Una vez construida, la aplicación se despliega múltiples veces en una gran cantidad de servidores de aplicación. En el estilo arquitectónico de los microservicios, los desarrolladores construyen y empaquetan independientemente varias aplicaciones más pequeñas que contribuyen a implementar partes de la aplicación total.

Una explicación simplista sería decir que la arquitectura de microservicios se basa en dividir grandes aplicaciones de silo en piezas totalmente desacopladas, más manejables.

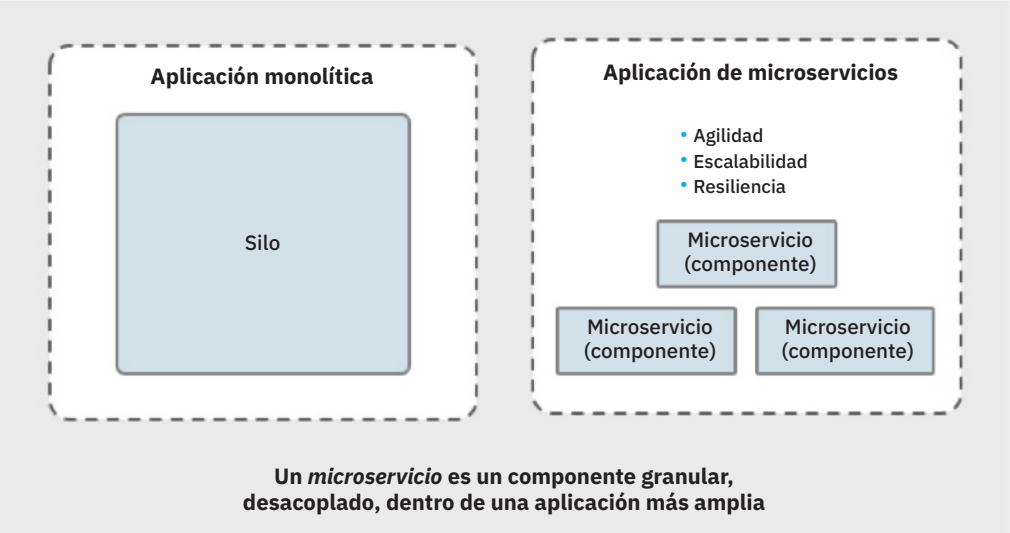


Figura 1: Aplicación monolítica contra microservicios

Hay cinco reglas simples que impulsan la implementación de aplicaciones construidas mediante la arquitectura de microservicios:

1. **Separar grandes monolitos en muchos servicios pequeños** – Un solo servicio accesible en la red es la unidad a desplegar más pequeña para una aplicación de microservicios. Cada servicio ejecuta su propio proceso. Esta regla se denomina un servicio por contenedor. Contenedor se refiere a un contenedor “Docker” o cualquier otro mecanismo liviano de despliegue, tal como un tiempo de ejecución de “Cloud Foundry”.

2. **Optimizar los servicios para una función única** – En un enfoque monolítico SOA tradicional, el tiempo de ejecución de una aplicación única realiza múltiples funciones empresariales. En un enfoque de microservicios, hay una sola función empresarial por servicio. Esto hace que cada servicio sea más pequeño y simple para escribir y mantener. Esto se denomina como el Principio de responsabilidad única (SRP, por sus siglas en inglés).

3. **Comunicarse a través de las API REST y los corredores de mensajes** – Uno de los inconvenientes del enfoque SOA es que hay numerosas normas y opciones para implementar los servicios SOA. El enfoque de microservicios limita estrictamente los tipos de conectividad en red que un servicio puede implementar para lograr la máxima simplicidad. De manera similar, los microservicios tienden a evitar el estrecho acoplamiento introducido por la comunicación implícita a través de una base de datos. Toda la comunicación entre un servicio y otro debe hacerse a través de la API del servicio o al menos debe usar un patrón de comunicación explícito tal como el 'Claim Check Pattern' [Hohpe y Woolf].

4. **Aplicar CI/CD por servicio** – En una gran aplicación compuesta por muchos servicios, los diferentes servicios evolucionan a velocidades diferentes. Cada servicio tiene una única y continua tubería de permisos de integración/entrega que avanza a un ritmo natural. Esto no es posible con el enfoque monolítico, donde cada aspecto del sistema es liberado forzosamente a la velocidad de la parte del sistema que se mueve más lentamente.

5. **Aplicar decisiones de alta disponibilidad (HA) / "clustering" por servicio** – Cuando se construyen grandes sistemas, el "clustering" no es un enfoque de "talla única". El enfoque monolítico de escalar todos los servicios del monolito al mismo nivel hace que se usen en exceso algunos servidores y se usen menos otros o, lo que es peor, genera la anulación de algunos servicios por otros que monopolizan todos los recursos compartidos disponibles, tales como los "thread pools". La realidad es que en un sistema grande, no todos los servicios necesitan ser ampliados, muchos pueden ser desplegados en un número mínimo de servidores para conservar los recursos. Otros requieren ser ampliados a cantidades muy grandes.

La combinación de estas cinco reglas y sus beneficios son las razones principales por las que la arquitectura de microservicios se ha hecho tan popular.

La arquitectura de microservicios se ha convertido en un estándar de facto para desarrollar aplicaciones comerciales de gran escala. En "Microservices a definition of this new architectural term" (Microservicios una definición de este nuevo término arquitectónico), Martin Fowler define así a los microservicios:

"En resumen, el estilo arquitectónico de microservicios es un enfoque para desarrollar una única aplicación como un conjunto de pequeños servicios, cada uno de ellos ejecutándose en su propio proceso y comunicándose con mecanismos ligeros, a menudo una API de recurso HTTP. Estos servicios se construyen alrededor de capacidades empresariales y para ser desplegados independientemente por maquinaria de despliegue totalmente automatizada. Hay un mínimo absoluto de gestión centralizada de estos servicios, que pueden ser escritos en diferentes lenguajes de programación y usan diferentes tecnologías de almacenamiento de datos".

Una de las diferencias clave entre los microservicios y los paradigmas como SOA y API es el enfoque en los componentes desplegados y en funcionamiento. Los microservicios se enfocan en la granularidad de los componentes desplegados en lugar de hacerlo en las interfaces, como se muestra en la figura siguiente.

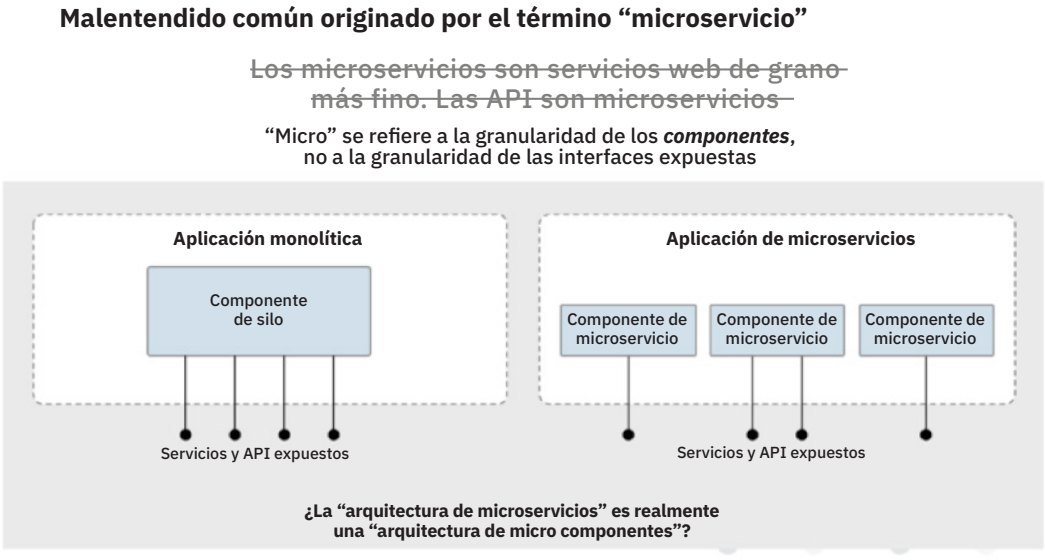


Figura 2: Diferencia entre una API y un microservicio

Muchas organizaciones, impulsadas por operaciones, construyen aplicaciones monolíticas en las que los casos y las funciones de uso se despliegan en una aplicación única y grande. Si bien es posible eludir a operaciones, cambiar estas aplicaciones para que se basen en microservicios requiere un gran esfuerzo. Son tres los factores que impulsan el desarrollo de los microservicios:

1. **La manera en que los equipos de desarrollo son organizados:** El desarrollo de microservicios se hace mejor con un enfoque de ingeniería que se concentre en descomponer una aplicación en módulos de una sola función con interfaces bien definidas, que sean desplegadas independientemente y operadas por pequeños equipos que se apropien de todo el ciclo vital del servicio. Los microservicios aceleran la entrega al minimizar la comunicación y coordinación entre las personas a la vez de reducir el alcance y el riesgo del cambio.

2. **Cómo se construyen las aplicaciones:** Las aplicaciones basadas en microservicios hacen algunas suposiciones acerca de la manera en que son construidas y el entorno en el que funcionan. El entorno es generalmente denominado *aplicaciones nativas de la nube o aplicaciones de factor 12*. Una arquitectura basada en microservicios aprovecha las fortalezas y acomoda los retos de un entorno de nube estandarizado, incluyendo conceptos tales como “elastic scaling” (ampliación elástica), “immutable deployment” (despliegue inmutable), “disposable instances” (instancias desechables) y “less predictable infrastructure” (infraestructura menos predecible).

3. **Cómo se entregan y ejecutan las aplicaciones:** El uso de contenedores como una manera estándar de ejecutar las aplicaciones, impulsa el empaquetado y ejecución de las aplicaciones basadas en microservicios. Los contenedores no son una tecnología nueva. Los contenedores de Linux son una capacidad del sistema operativo que hace posible ejecutar múltiples sistemas Linux aislados (o contenedores) en un “host” Linux de control. Los contenedores Linux sirven como una alternativa liviana de las máquinas virtuales completas. Aunque los contenedores no son un concepto nuevo, los marcos tales como Docker han popularizado su uso al diseñar una manera de crear una imagen para los contenedores en funcionamiento. Esto le proporciona a usted una manera estándar de empaquetar una aplicación y todas sus dependencias de manera que pueda ser trasladada entre distintos entornos y ejecutada sin cambios. Otros marcos como “Cloud Foundry” usan los contenedores para ejecutar las aplicaciones pero abstraen la virtualización.

La figura siguiente muestra de qué manera la arquitectura de aplicación monolítica evoluciona hacia una basada en microservicios.

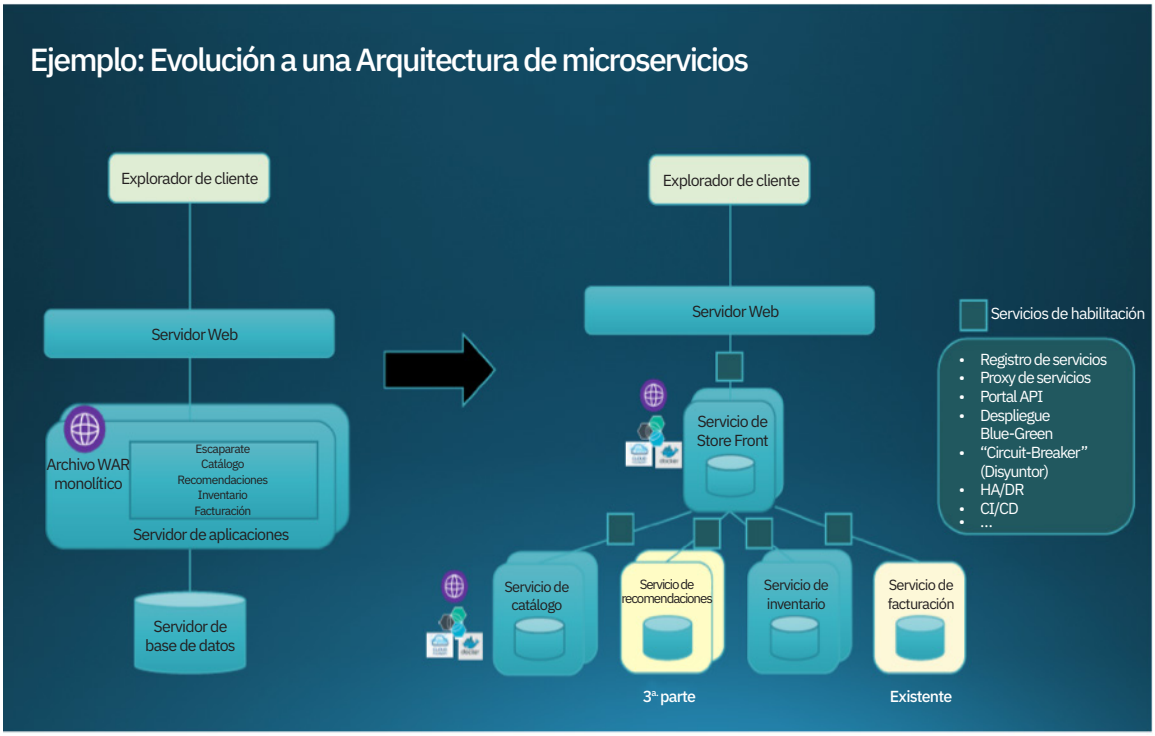


Figura 3: Diferencia entre las aplicaciones monolíticas y los microservicios

En la figura 3 se muestra un único archivo empresarial que aloja a todos los componentes de un sitio web de comercio minorista. El archivo de la aplicación contiene todas las funciones empresariales, tales como catálogo e inventario, así como la lógica del sitio web, la lógica de la empresa y la lógica de persistencia para cada componente. Además, la aplicación comparte una única base de datos, que a menudo contiene modelos de datos estrechamente acoplados y es compartida con otras aplicaciones.

Obsérvese en el lado derecho de la imagen que las capacidades empresariales son ahora desplegadas en aplicaciones separadas, con sus propios datos. Este nuevo estilo de arquitectura introduce preocupaciones relacionadas con la comunicación, la propiedad de los datos, la sincronización y la resiliencia de los datos de los microservicios.

El capítulo "The Characteristics of a Microservice Architecture" (Las características de una arquitectura de microservicios) de James Lewis y Martin Fowler trata sobre aspectos claves de las aplicaciones basadas en microservicios.

Usando el ejemplo de referencia anterior, un resumen de los aspectos incluye:

Separación en componentes a través de servicios: Este aspecto es tratado previamente en este artículo.

Organizadas alrededor de las capacidades empresariales: Hemos comentado previamente la noción de desarrollo de equipos y la organización alrededor de las capacidades empresariales requiere cambiar la manera en que pensamos acerca de los equipos de desarrollo.

Consideremos la manera en que los equipos son frecuentemente organizados por roles, como se indica a continuación:

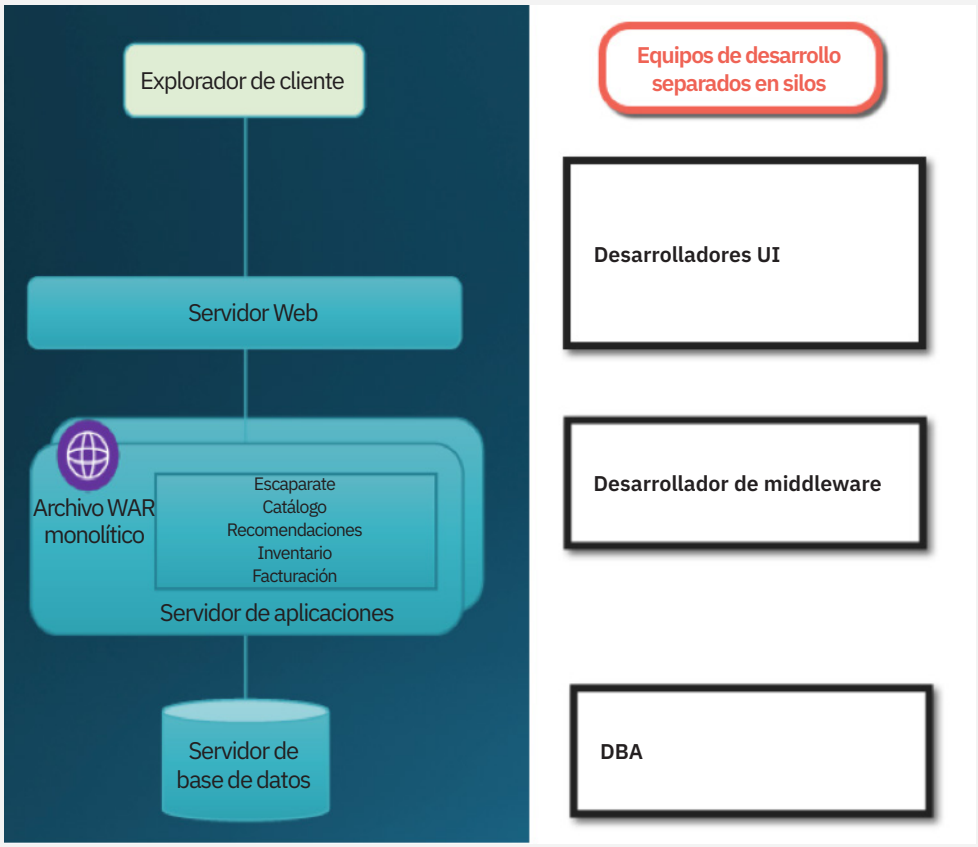


Figura 4: Equipos de desarrollo en silos

Comparemos eso con un enfoque de equipos basado en microservicios, que se organiza alrededor de las capacidades empresariales, como muestra la figura 5.

Figura 5: Equipos de desarrollo basados en las capacidades empresariales

Resolver problemas empresariales con microservicios

La mayoría de las compañías construyen las aplicaciones nuevas, nativas de la nube, mediante un enfoque basado en microservicios. Sin embargo, muchas compañías también están observando la necesidad de refactorizar las aplicaciones monolíticas existentes para moverse más rápidamente.

Migrar aplicaciones monolíticas

La figura 6 muestra un ejemplo de migración de una arquitectura monolítica a una arquitectura basada en microservicios. En este caso, un comerciante minorista quiere hacer la transición a microservicios para moverse más rápido, aprender más acerca de los clientes e introducir características modernas tales como los pagos.

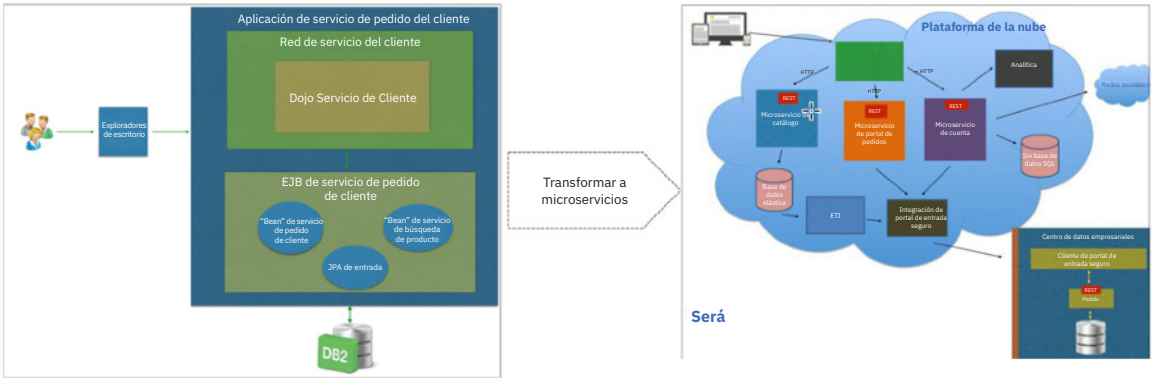


Figura 6. Migración a una arquitectura basada en microservicios

El minorista del escenario anterior siguió estas etapas para pasar de una arquitectura monolítica a una de microservicios:

1. **Determinar cómo manejar el catálogo-** El primer problema empresarial que tuvieron que resolver fue cómo gestionar el catálogo de artículos. Los clientes no podían encontrar los datos de los productos y el minorista no podía exponer las cosas a otros sitios. El equipo decidió construir un único microservicio para el catálogo de la empresa adoptando los pasos siguientes:

    - Importó los datos en una búsqueda elástica para proporcionar nuevas maneras de buscar los datos e identificar nuevos patrones de datos
    - Vinculó su sitio web existente con la nueva búsqueda
    - Con este piloto, introdujeron un nuevo modelo CI/DC como cimiento.

El equipo convenció a la gerencia a hacer la transición a un modelo de microservicios y expandir la empresa.

2. **Aprender más acerca del cliente** - Para aprender más acerca del cliente, el equipo creó un microservicio de cuenta:

    - Primero, analizaron cómo cambiar la empresa de ser una organización enfocada en el cliente a una enfocada en el inventario.. Diseñaron un nuevo modelo de cliente y usaron una base de datos NOSQL como Mongo DB o Cloudant, que les proveyó un modelo de datos no estructurado. Sabían que, con el correr del tiempo, los datos de los clientes se enriquecerían con base en la analítica, el marketing y los datos cognitivos.
    - Los datos existentes sobre los clientes se usaron para hacer el seguimiento de los pedidos.

3. **Crear una nueva experiencia de usuario** - El equipo creó una nueva “front end” (interfaz de interacción directa) para plataformas móviles y de red y una nueva aplicación móvil nativa. Con una moderna interfaz de usuario y nuevo catálogo, crearon una nueva experiencia para el usuario final. El nuevo catálogo se integró con la lógica de pedidos existente, que incluía al núcleo empresarial y era demasiado compleja para descomponer en ese momento.

4. **Microservicio de pedido** - El equipo se concentró en la creación de nuevas API de pedidos para móviles, además de su integración con las transacciones existentes. La empresa decidió crear un microservicio adaptador que accedía al sistema de registro (SOR) existente. Aprovecharon la oportunidad para integrarlo con nuevos y modernos pagos en esta capa del adaptador.

5. **Expandir la empresa** - El minorista expandió su modelo de empresa al agregar una nueva característica de subasta, extendiendo el nuevo modelo de microservicios.

Arquitectura de microservicios

Ahora es el momento de informarse sobre los detalles de una arquitectura basada en microservicios.

Capacidades de los microservicios

Los componentes de las aplicaciones se ejecutan como microservicios en la nube, comunicándose unos con otros a través del entramado de microservicios. Los distintos tipos de aplicaciones usan diferentes patrones. Ilustraremos un ejemplo red / móvil más adelante. La figura 7 muestra las capacidades requeridas para una arquitectura-basada en microservicios.

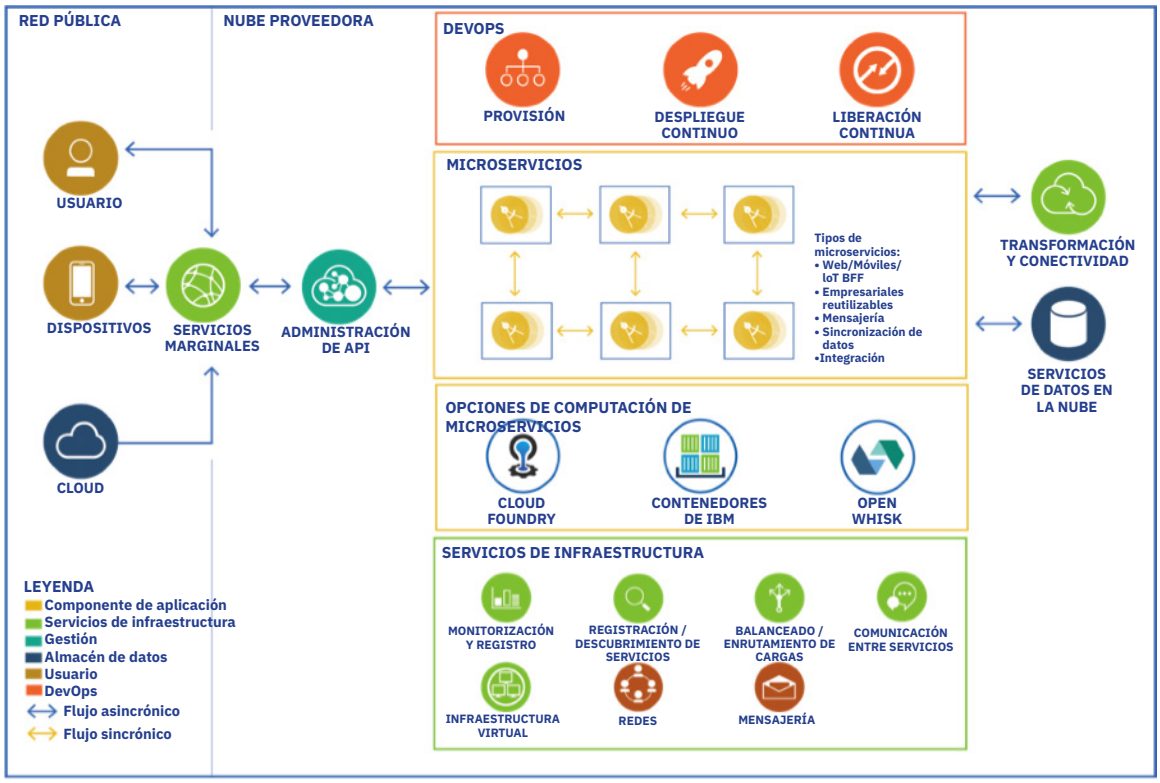


Figura 7. Capacidades de una arquitectura basada en microservicios

Leyendo el diagrama de izquierda a derecha, el paso siguiente es revisar las capacidades de los microservicios. Muchos microservicios son expuestos a través de API, y algunos de estos microservicios necesitan ser consumidos a través de un *Portal API*.

Un Portal API puede ser tan sencillo como un proxy de puntos finales. Algunos Portales API son más elaborados, con seguridad al primer contacto, monitorización y versión de API, o completos sistemas de gestión de API con un portal desarrollador para consumo de terceros.

DevOps

Es necesario desarrollar una sólida estrategia de automatización mediante el uso de DevOps para crear una arquitectura de microservicios exitosa, que abarque aprovisionamiento, despliegue continuo y liberación continua. Su estrategia debería incluir lo siguiente:

- **Provisioning (Aprovisionamiento)** - Una arquitectura de microservicios exitosa requiere un aprovisionamiento automatizada para los entornos de aplicación. La capa “Platform as a Service” (Plataforma como servicio) generalmente provee este servicio a través de servicios gestionados, tales como “Container as a Service (CaaS)” (Contenedor como servicio) en IBM® Cloud®. Si se ejecuta una infraestructura de contenedores sobre una capa de “Infrastructure as a Service (IaaS)” (Infraestructura como Servicio) usando motores de orquestación Docker, como Kubernetes o Docker Datacenter (DDC), se debe automatizar la manera de aprovisionamiento y actualización de esos entornos usando aprovisionamiento automatizado en un entorno de máquinas virtuales.

- **Despliegue continuo** - En el entorno de desarrollo se requiere un proceso automatizado de construcción y despliegue para las imágenes Docker o las aplicaciones de microservicios. Si se tiene una estrategia-de nube múltiple, la automatización de construcción y despliegue necesita abstraer las diferencias en la manera en que se hacen las cosas, como las políticas de ampliación automática o de nube.

- **Liberación continua** - Un entorno de DevOps admite una fuerte cultura de desarrollo impulsado por pruebas y la automatización de pruebas. Esto incluye las pruebas de unidades, pruebas automatizadas funcionales y de desempeño, y pruebas de convalidación.

Opciones de computación con microservicios

Existen varias opciones de computación para ejecutar los microservicios:

- **Contenedores Docker** - Estos contenedores proporcionan la mayor portabilidad en entornos de nube -y on-premises. El uso de contenedores Docker requiere solidez de DevOps.

- **Cloud Foundry** - Esta PaaS de fuente abierta provee abstracción para la manera en que los microservicios funcionan y se comunican entre sí y para la virtualización, tal como los contenedores. Si bien Cloud Foundry ofrece cierto nivel de portabilidad, requiere una pila más completa para funcionar en el entorno de la nube.

- **Funciones** - Opción de computación basada en eventos con el más alto nivel de abstracción y facilidad de uso. Los desarrolladores despliegan sencillos manejadores de eventos para responder a eventos que emiten los componentes de la nube dentro de un "hub" (concentrador) de mensajes central. Toda la virtualización es abstraída.

- **Máquinas virtuales o "bare metal"** - Se pueden crear aplicaciones basadas en microservicios y ejecutarlas en máquinas virtuales (VM). Esto requiere mucho más aprovisionamiento y DevOps para tener éxito. Las VM y "bare metal" ofrecen una mayor flexibilidad a costo de la facilidad.

La figura 8 resume estas opciones:

Figura 8. Opciones de computación de microservicios

Servicios de infraestructura

El entorno de nube que rodea a los microservicios proporciona el entramado y los servicios necesarios para conexión en redes, mensajería, comunicación de microservicios, registros y monitorización, virtualización, descubrimiento y proxy de servicios, características de resiliencia y mucho más.

Arquitectura de aplicaciones

A continuación se muestra un ejemplo de una arquitectura de aplicación de microservicios.

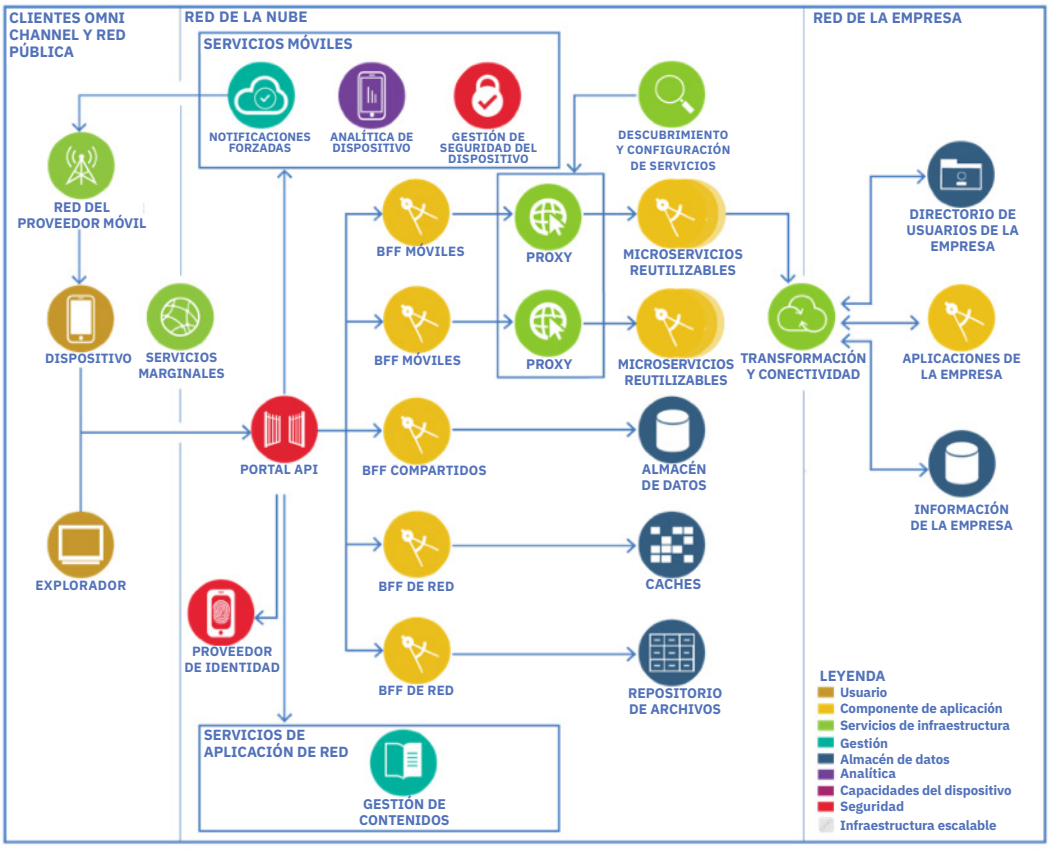


Figura 9. Arquitectura de aplicación de microservicios

Para resumir los componentes de esta arquitectura:

- Las aplicaciones OmniChannel de este ejemplo contienen ambas una aplicación nativa iOS y una aplicación de red basada en Angular. El diagrama las representa como un dispositivo y un explorador.

- Las aplicaciones móviles usan el servicio IBM Mobile Analytics for Cloud (Analítica móvil para la nube de IBM) para recopilar analítica de dispositivos para operaciones y negocios.

- Ambas aplicaciones del cliente hacen llamadas API a través de un Portal API. El Portal API, IBM API Connect, proporciona un "OAuth Provider" para implementar la seguridad de las API.

- Las API se implementan como patrones de microservicios Node.js, que son denominados "Backend" para "Frontends" (BFF). Otros nombres posibles incluyen "Experience API (xAPI) o "Iteration API". En esta capa, los desarrolladores de "front end" generalmente escriben lógica de "back end" para su "front end". El "Inventory BFF" se implementa usando el marco Express. El "Social Review BFF" se implementa usando el marco "API Connect LoopBack". Estos microservicios se ejecutan en IBM Cloud como aplicaciones "Cloud Foundry".

Lea un estudio de caso de cliente

Obtenga los detalles técnicos

- Los "Node.JS BFF" invocan otra capa de microservicios Java reutilizables. En un proyecto de vida real, esto será generalmente escrito por un equipo diferente. Estos microservicios reutilizables se escriben en Java usando "Spring Boot". Se ejecutan dentro de contenedores de IBM usando Docker.

- Los Node BFF y los microservicios de Java se comunican entre sí por medio de un entramado de microservicios. Ejemplos de ello son Istio y Netflix OSS.

- Los microservicios de Java pueden interactuar con las bases de datos de la nube y las capas de integración.

Pila de microservicios

Es importante definir su pila de microservicios. A continuación se incluye un ejemplo de una pila con algunas de las elecciones ya hechas. Debe definir las diferentes capas, incluso la aplicación, entramado y DevOps. Consultar la Microservices Decision Guide (Guía de decisiones sobre microservicios) de IBM para comprobar de qué manera IBM determinó estas tecnologías habilitadoras de microservicios.

Figura 10. Pila de microservicios

Aplicación

Un lenguaje y una pila de tiempo de ejecución escriben su aplicación. Entre los ejemplos se incluyen pilas modernas, ligeras "Java Platform, Enterprise Edition (Java EE) como MicroProfile, WebSphere Liberty, pilas Java alternativas como Spring, o pilas Node.js como StrongLoop.

Se espera que bibliotecas de lenguaje específico manejen los elementos de aplicaciones de microservicios tales como la ruptura o el seguimiento de circuitos.

Entramado de los microservicios

Un entramado de microservicios es una pila de capacidades que provee un conjunto de capacidades necesarias:

- Enrutamiento y descubrimiento es una capacidad básica para la comunicación entre microservicios atenta a la nube. Una instancia de microservicios podría ser autoescalada y tener “clusters” dinámicos por ejemplo, y usted debería descubrir el microservicio por nombre y luego encaminarse a una instancia de ejecución. Los marcos como Zuul / Eureka proveen esto como parte de la pila Netflix OSS. Istio es una opción preferida que tiene base políglota y proporciona control de grano más fino del enrutamiento de las llamadas de servicio y control a la capa subyacente. La figura siguiente muestra un ejemplo de la arquitectura Istio.

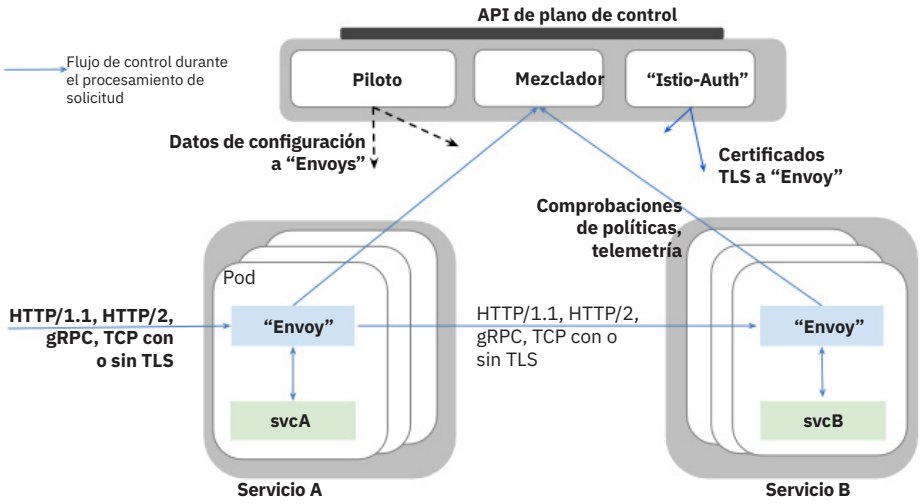


Figura 11. Arquitectura Istio

- Para registros y analítica, se requiere una pila que capture los registros de “streaming” y los reenvíe a varios lugares. Hay varios marcos disponibles para esa función.

- Se requiere disponer de tableros de monitorización para varios datos, incluso información sobre tiempo de ejecución, aplicación y disyuntores. Al usar la pila de registro, estas herramientas proporcionan una vista unificada.

- Las capas de mensajería y seguridad también son importantes.

DevOps

Es necesario tener un fuerte conjunto de herramientas de DevOps para completar los requerimientos de aprovisionamiento,orquestación, construcción y despliegue, y operaciones.

Gestión de contenedores

Es necesario un entorno de orquestación y tiempo de ejecución de contenedores tal como el “IBM Cloud Container Service”, “Kubernetes” o “Docker Data Center” para soportar una pila de microservicios.

Resiliencia en las arquitecturas basadas en microservicios

Con tales componentes de grano fino, se debe tener consciencia de todos los puntos de resiliencia en una arquitectura de microservicios. Esto incluye alta disponibilidad, “failover”, DR, ruptura de circuitos y asolación.

La figura 12 ilustra la resiliencia en las arquitecturas basadas en microservicios y muestra el uso de un balanceador de cargas global y de la redundancia multisitios.

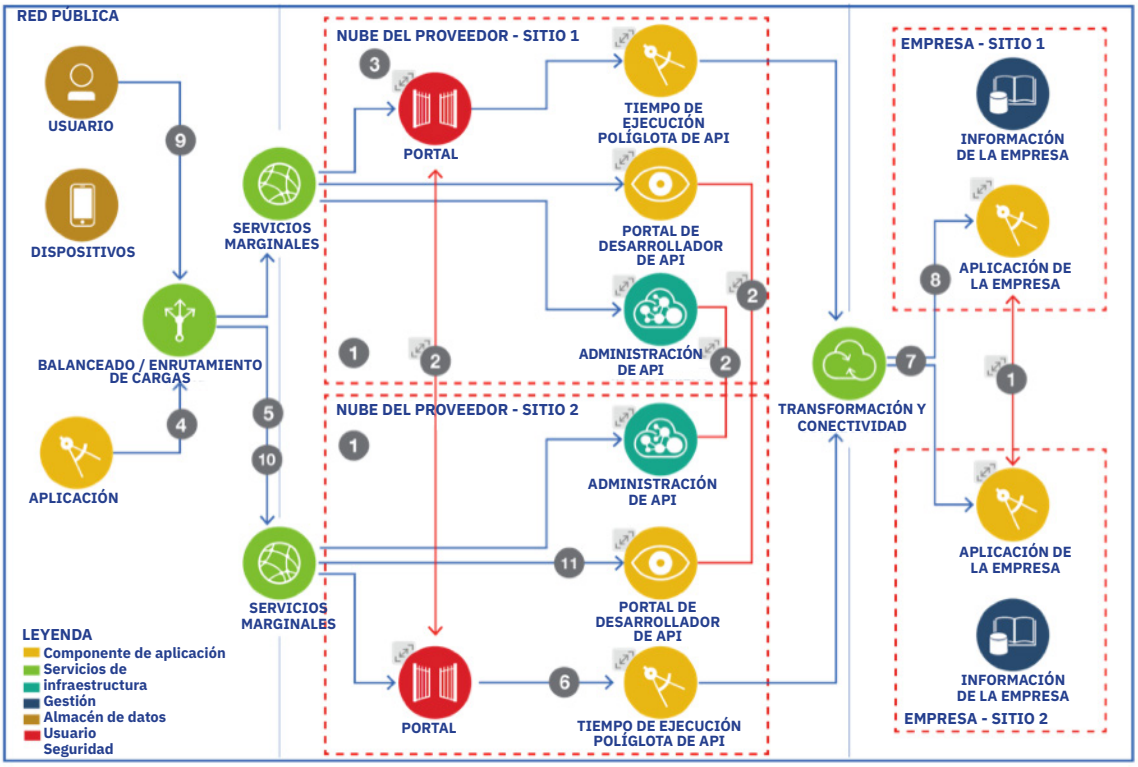


Figura 12. Resiliencia en una arquitectura de microservicios

Patrones de alta disponibilidad y recuperación en caso de desastre para los microservicios

Consideraciones de resiliencia

- “Back ends” de datos redundantes
- “Back ends” de datos replicados
- Desplegar múltiples veces por región
- Desplegar en múltiples regiones (tres centros de datos)
- Balanceo de cargas global

Cuando se trata con la resiliencia, es importante hacer algunas distinciones entre la alta disponibilidad (HA) y la recuperación de desastres (DR).

La HA asegura que los servicios estén disponibles para los usuarios finales cuando se realizan en el sistema actividades de mantenimiento como despliegue de actualizaciones, reinicio de las máquinas virtuales de “hosting”, y aplicación de parches de seguridad al SO de “hosting”.

La HA generalmente no se aplica a problemas importantes inesperados, como la pérdida completa del sitio debido a grandes cortes de energía, terremotos, graves fallos de hardware o pérdida de la conectividad en todo el sitio. En tales casos, si los servicios tienen estrictos objetivos de nivel de servicio (SLO), se debería hacer redundante a toda la pila de aplicaciones (infraestructura, servicios y componentes de aplicaciones) recurriendo al menos a dos regiones diferentes de IBM Cloud. Esto se define típicamente como arquitectura de Recuperación de desastres (DR).

Hay muchas opciones para implementar soluciones de DR. A fines de simplificar, es posible agrupar las diferentes opciones en tres categorías principales:

Activa/Pasiva, Activa/Stand by, y **Activa/Activa**

Activos/Pasivos

Esta opción mantiene activa a toda la pila de aplicaciones en un lugar, mientras otra pila de aplicaciones es desplegada en una ubicación diferente, pero se mantiene inactiva o sin funcionar. En caso de que la falta de disponibilidad del sitio principal se prolongue, se activará la pila de aplicaciones en el sitio de respaldo. Generalmente, eso requiere restaurar las copias de seguridad tomadas en el sitio principal. Este enfoque no se recomienda si la pérdida de datos es un problema o si la disponibilidad del servicio es crítica, ya que el objetivo de tiempo de recuperación (RTO) es menor que unas pocas horas.

Activos/Reposo

Con esta opción, la pila total de aplicaciones está activa tanto en las ubicaciones primaria como de respaldo; sin embargo, solo el sitio principal atenderá las transacciones de los usuarios. El sitio de respaldo almacena una réplica del estatus de la ubicación principal a través de la replicación de datos como replicación de base de datos o replicación del disco. En caso de que la falta de disponibilidad del sitio principal se prolongue, todas las transacciones de los clientes se enrutarán al sitio de respaldo. Este enfoque proporciona buen objetivo de punto de recuperación (RPO) y RTO (minutos), pero es significativamente más caro que el enfoque Activo/Pasivo debido a la duplicación del despliegue. Esto es un desperdicio de recursos porque los activos en "standby" no pueden usarse para mejorar la escalabilidad y procesamiento.

Activos/Activos

En este caso, ambas ubicaciones están activas y las transacciones de los clientes se distribuyen a ambas regiones, según políticas predeterminadas como "round-robin", balanceo de cargas y ubicación. En el caso de que un sitio falle, el otro sitio atenderá a todos los clientes. Con esta configuración, es posible lograr RPO y RTO cercanos a cero. El inconveniente es que ambas regiones deben tener el tamaño suficiente para manejar la carga completa, aunque solo se use la mitad de sus capacidades cuando ambas ubicaciones están disponibles. En ese caso, el servicio "Auto-Scaling for IBM Cloud" asigna recursos según las necesidades, de manera similar a la aplicación de ejemplo BlueCompute.

Consideraciones de escalabilidad y rendimiento

El agregado de resiliencia generalmente implica tener despliegues redundantes, que también se pueden usar para mejorar el rendimiento y la escalabilidad. Eso se cumple para el caso de la opción Activa/Activa, descrita en la sección anterior. En el caso de aplicaciones globales, es posible redirigir las transacciones de los usuarios a la ubicación más cercana para mejorar el tiempo de respuesta y la latencia, mediante el uso de soluciones de enrutamiento globales desde Akamai o Dyn.

Implementación de proyectos de microservicios

El siguiente paso lógico es comprender cómo implementar los proyectos basados en microservicios en su organización.

Usted puede construir un sistema de microservicios a partir de cero o desde un sistema monolítico existente. La mayoría de los clientes de IBM comienzan con su experiencia de microservicios buscando una actualización de sus monolíticos existentes, así que esta sección se enfoca en la evaluación e implementación de los microservicios cuando se trabaja a partir de una arquitectura monolítica existente.

Si bien la construcción de proyectos de microservicios sin una aplicación previa es relevante o importante, es conveniente usar una estrategia que comience por arquitectura monolítica al construir microservicios. En resumen, esto significa que debe construir su aplicación en cualquier forma en la que pueda convalidar su idea primero.

Luego, aplicará los principios de este artículo para ampliar y evolucionar su monolito inicial en un proyecto de microservicios. No tiene valor crear microservicios arquitectónicamente puros que no ofrezcan un aporte de valor a la empresa.

Un excelente relato de primera mano sobre la implementación de una aplicación de microservicios con la estrategia de comenzar por la arquitectura monolítica ha sido documentado por el equipo Game On! en The Chronicles.

Evolución de aplicaciones existentes

A esta altura, usted conoce que hay muchos conceptos que necesita comprender al emprender una transformación basada en microservicios. En esta sección, describiremos tres áreas que necesita comprender para implementar un proyecto de microservicios exitoso: su empresa, su cultura y conjunto de destrezas, y su tecnología.

Entender y definir las necesidades de su empresa

¿Por qué está pensando en hacer la transición a microservicios? Para muchas empresas, se requieren prácticas más eficientes de desarrollo de software y operativas para entregar valor a la empresa más rápidamente —y para entregar una mejor experiencia de usuario.

Antes de que pueda comprender el impacto de un proyecto de microservicios en sus aplicaciones e infraestructura existentes, debe entender qué partes de su empresa se están moviendo demasiado lentamente para producir resultados satisfactorios. En muchos casos, los sistemas de contratación (SOE) de la organización son los que hacen las cosas más lentas. Estos sistemas están disponibles a través de muchos canales, incluso Internet, móviles y API. La falta de velocidad es la razón principal para pasar a una arquitectura basada en microservicios.

Antes de que usted pueda adoptar un enfoque orientado a los microservicios, debe debe saber qué es lo que no está llegando al mercado suficientemente rápido. Primero, necesita identificar qué piezas de la aplicación necesitan mejoras y modificaciones para hacerlas más rápidas. A partir de allí, podrá detectar qué partes del monolito existente deben ser tomadas como objetivos para su evolución a microservicios.

En este paso, resulta valioso usar los artefactos de diseño y arquitectura, tales como los flujos de experiencia de usuario o los diagramas de arquitectura. El equipo puede identificar y priorizar rápidamente las secciones del monolito, como se muestra en la figura 13.

Figura 13. Puntos álgidos del monolito existente, usando la escala Rojo-Amarillo-Verde para las prioridades

Este proceso de identificación no necesita ser exhaustivo y debería ser de naturaleza iterativa. Los objetivos clave son:

- Identificar las distintas funciones empresariales que su monolito está proveyendo
- Comprender la velocidad relativa y la complejidad requeridas para cambiar esas funciones empresariales
- Comprender el deseo de la empresa para lograr ciclos de retroalimentación más rápidos para funciones empresariales específicas.

Entender su cultura y conjunto de destrezas

Si bien no es específica de las arquitecturas basadas en microservicios, durante una transformación digital es crucial tener una comprensión completa de los equipos, cultura y conjuntos de destrezas de una organización.

Al diseñar monolitos, es típico que la mayoría de las organizaciones se construyan en silos, con la participación necesaria a lo largo del ciclo vital del desarrollo del software. A menudo, esto crea límites bien definidos, con roles y responsabilidades restrictivas a lo largo de estos límites. En la figura 14 se muestra una típica estructura institucional monolítica.

Figura 14. Estructuras institucionales de TI tradicionales

En comparación, las arquitecturas de microservicios solo pueden tener éxito cuando los equipos tienen el poder para hacerse cargo de todo el ciclo vital de desarrollo y operaciones del software. Para hacerse cargo de todo el ciclo vital de DevOps, los equipos necesitan miembros con diferentes roles y responsabilidades.

Estos equipos interfuncionales se construyen para empoderar las arquitecturas basadas en microservicios. En lugar de estar distribuidos en miembros de equipos en silos, todos los roles y responsabilidades están contenidos en el mismo equipo. Todos —desde diseño a desarrollo y operaciones— trabajan estrechamente juntos y , a menudo, comparten ubicaciones. Las estructuras físicas de los equipos están fuera del alcance de este artículo, pero los límites de los equipos virtuales tienen más posibilidades de éxito para transformar una empresa cuando se forman de manera interfuncional.

Esto permite que la empresa identifique con claridad las experiencias de diseño, comprenda las posibles líneas cronológicas de entrega y minimice los gastos operativos, ya que todas las partes interesadas de diseño, desarrollo y operaciones están representadas en el equipo. En la figura 15 se muestra una configuración óptima de equipo de DevOps.

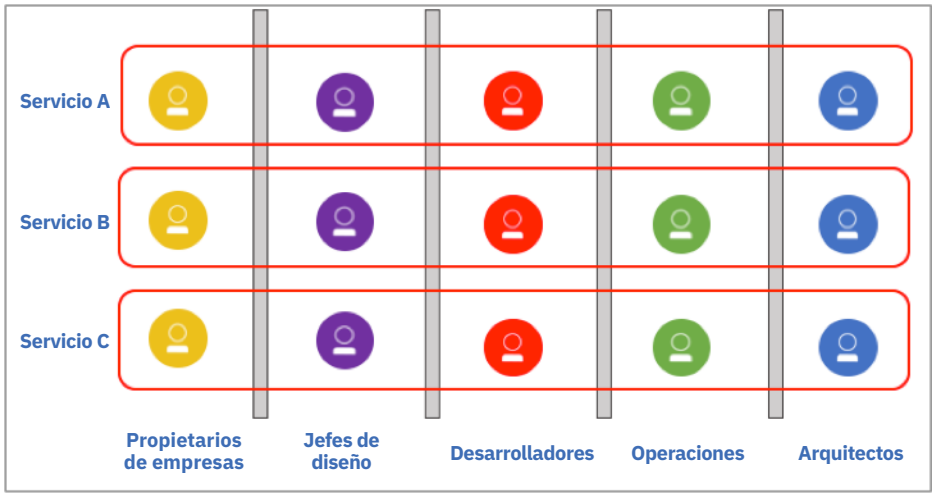


Figura 15. Equipos de DevOps optimizados para entregar éxito en microservicios

Un equipo interfuncional también respaldará el rápido crecimiento de las destrezas individuales en todo el equipo. Cuando un equipo se hace cargo de todas las responsabilidades del microservicio, desde diseño a operaciones y a datos de tiempo de ejecución, los miembros individuales del equipo no son relegados a una sola tarea. A menudo, los ingenieros de "front-end" desarrollan destrezas de administración de bases de datos, mientras que los miembros del equipo orientados a operaciones aprenden más acerca de las diferencias en marcos de interfaces de usuario. La expansión de los conjuntos de destrezas de esta manera, ayuda a toda la organización de TI a tener éxito con los microservicios —es mucho más fácil construir nuevos equipos compuestos por miembros de equipo con formación integral que buscar especialistas para desempeñar roles muy específicos.

A menos que usted aborde el problema de la empresa y la cultura y conjuntos de destrezas de su equipo, no podrá implementar eficazmente la tecnología de microservicios y seguirá conservando los mismos procesos y estructuras.

Entender la tecnología

El análisis correcto de las pilas tecnológicas existentes varía mucho de una organización a otra, pero el enfoque simplificado que describimos ayuda a asegurar el éxito, tanto inicial como continuo de sus proyectos de microservicios. Comenzar por partes y definir éxitos progresivos e iterativos es un enfoque mucho más factible y fructífero que tratar de transformar todo a la vez.

Figura 16. Enfoque iterativo a las evoluciones de microservicios

La primera fase de comprender su tecnología es identificar los servicios de grano grueso que están presentes en el monolito existente. A menudo se puede alinear eto con principios de diseño impulsados por dominios (DDD) que le permitirán aplicar principios de diseño bien definidos a aplicaciones existentes que nunca fueron construidas en base a esos principios. La identificación de estos servicios de grano grueso le ayudarán a entender la complejidad de las estructuras de datos, el nivel de acoplamiento entre los componentes actuales y los equipos que son responsables de los nuevos servicios de grano grueso. Una revisión exitosa le brindará una comprensión clara de los límites de datos, tanto dentro de un servicio determinado como entre los servicios.

Una vez que haya identificado los servicios de grano grueso, deberá crear un plan para convertirlos en microservicios de grano fino. Estos microservicios, basados en su trabajo previo, deberían trabajar todos con información similar, gestionar sus propios datos y comprender qué datos necesitan para leer y escribir a otros servicios. A partir de aquí, podrá identificar e implementar la resiliencia, escalabilidad y agilidad de los microservicios de grano fino individuales.

Como se mencionó previamente, las API y los microservicios no pueden compararse uno a uno, sino que son, simplemente, dos partes de un todo mayor. Una vez que usted tiene una mejor comprensión de sus microservicios de grano fino, también tendrá una mejor comprensión de sus interfaces, incluso de cuales interfaces están en el camino crítico, cuáles son opcionales y cuáles ya no se necesitan. Si usted no puede trazar un mapa de una interfaz o API existente a uno de sus microservicios de grano grueso o grano fino, es muy probable que la misma no sea necesaria.

Dimensionar el esfuerzo de microservicios

Una vez completado todo el trabajo de análisis y planificación, es el momento de definir las líneas cronológicas, las velocidades de entrega y los resultados esperados. Estos variarán ampliamente, pero no será un proceso totalmente nuevo con respecto a los proyectos de transformación de algo existente a digital.

El trabajo pesado de comprender la empresa, comprender la estructura de los equipos y comprender la tecnología, ayudará ahora a asegurar que la organización está preparada para comprender la totalidad de la evolución a microservicios de cualquier monolito dado—ya sea que se encuentre en un alcance de prueba-de-concepto, alcance piloto o alcance de evolución a gran escala.

Patrones de microservicios

Patrones de desarrollo para los microservicios

En desarrollo, los patrones son útiles para aplicar soluciones conocidas a tipos comunes de requerimientos, y esto es válido también para los microservicios. Uno de los principios de diseño de microservicios de Martin Fowler es que los microservicios están "Organizados alrededor de las capacidades de la empresa".[2] Esto deriva directamente del descubrimiento de que el simple hecho de que usted pueda distribuir algo, no significa que deba hacerlo.

Estos patrones son usados comúnmente para microservicios:

- El **patrón fachada (facade pattern)** define una API externa específica para un sistema o subsistema. El subtexto de este patrón es que esta API es impulsada por la empresa

- **Los patrones de entidad (Entity) y agregado (Aggregate)** son útiles para identificar conceptos específicos de la empresa que se relacionan directamente con microservicios, para equipos de desarrollo que no están acostumbrados a diseñar en términos de interfaces de empresa.

- **El patrón de servicios** ofrece una manera de mapear operaciones que no corresponden a una sola entidad o agregado en el enfoque basado en entidad que se requiere para los microservicios.

- **El patrón Adaptor Microservices (Microservicios adaptadores)** es útil en el mundo corporativo, donde en muchos casos los equipos de desarrollo no tienen control descentralizado sobre los datos de patrones. En un microservicio adaptador, usted adapta entre dos API diferentes. Una API es una API orientada a la empresa construida con RESTful o técnicas de mensajería ligeras, con las mismas técnicas impulsadas por dominio que un microservicio tradicional. La segunda API es la API heredada o servicio tradicional SOAP basado en WS-*.

- **Patrón Strangler Application** aborda el hecho de que las empresas y las aplicaciones nunca viven en realidad en un entorno sin restricciones. Los programas que más pueden beneficiarse por los microservicios son las grandes aplicaciones monolíticas que se pueden refactorear. El patrón brinda un enfoque para gestionar la refactorización. El patrón "Strangler Application" es descripto con más detalle en una sección posterior de este artículo.

Patrones de operaciones para los microservicios

Si bien los microservicios permiten cambiar y desplegar un servicio único más rápidamente, también es cierto que hacen que resulte más esforzado gestionar y mantener un conjunto de servicios en comparación con una aplicación monolítica correspondiente. Estos patrones de operaciones para microservicios que estaban originalmente desarrollados para gestión convencional de aplicaciones es válido para el aspecto operativo de DevOps.

- **El patrón "Service Registry" (Registro de servicio)**

hace posible cambiar la implementación de los servicios posteriores, y le brinda a usted la posibilidad de elegir la ubicación del servicio para variar en distintas etapas de su tubería de DevOps. Esto se logra evitando la codificación dura de puntos finales específicos de microservicio en su código. Sin "Service Registry", su aplicación fracasará rápidamente a medida que los cambios de código comiencen a propagarse hacia arriba a través de una cadena de llamadas de microservicios.

- **Los patrones "Correlation ID" y "Log Aggregator"**

logran una mejor aislación, mientras que a la vez hacen posible eliminar los fallos de los microservicios más fácilmente. El patrón "Correlation ID" permite rastrear la propagación a través de una cantidad de microservicios escritos en un número de distintos lenguajes. El patrón "Log Aggregator" complementa a "Correlation ID" permitiendo que los registros de una cantidad de diferentes microservicios sean agregados en un único repositorio pasible de búsqueda. Juntos, estos patrones permiten la solución eficiente y comprensible de fallos de microservicios, independientemente del número de servicios o de la profundidad de cada pila de llamadas.

- **El patrón "Circuit Breaker" (Disyuntor)**

ayuda a evitar pérdidas de tiempo al manejar fallas en las etapas posteriores si usted sabe que ya están ocurriendo. Para hacer esto, usted planta una sección de código *"circuit breaker" (disyuntora)* en llamadas de servicios anteriores que detecta cuándo un servicio posterior está funcionando mal y evita tratar de llamarlo. El beneficio de este enfoque es que cada llamada falla rápidamente. Usted puede proveer una experiencia general mejor a sus usuarios y evitar gestionar erróneamente recursos como "threads" y "connection pools" cuando sabe que las llamadas a secciones posteriores están destinadas a fallar.

Enfocarse en el “Strangler Pattern”

El “Strangler Pattern” de Fowler se basa en una analogía a una enredadera que estrangula al árbol al que está enroscada. La idea es que usted use la estructura de una aplicación web — construida con Identificadores de Recursos Uniformes (URI) que mapean funcionalmente a diferentes aspectos de un dominio empresarial—para dividir una aplicación en diferentes dominios funcionales y reemplazarlos con una nueva implementación basada en microservicios, un dominio por vez.— Estos dos aspectos forman aplicaciones separadas que viven lado a lado en el mismo espacio URI. A lo largo del tiempo, la nueva aplicación refactoreada “estrangula” o reemplaza a la aplicación original hasta que, finalmente, usted puede anular la aplicación monolítica.

Los pasos del “Strangler Pattern” son transformar, coexistir y eliminar:

- **Transformar** - Crear un nuevo sitio paralelo, por ejemplo, en IBM Cloud o en su entorno existente, pero basado en enfoques más modernos.
- **Coexistir** - Dejar al sitio existente donde está durante un tiempo. Redireccionar gradualmente desde el sitio existente al nuevo sitio para la nueva funcionalidad implementada.
- **Eliminar** - Quitar la antigua funcionalidad del sitio existente, o simplemente dejar de mantenerla, a medida que el tráfico es reenviado eludiendo esa porción del sitio.

La ventaja de aplicar este patrón es que crea valor incremental mucho más rápidamente que si uno tratara de hacer todo en una sola y gran migración. También le proporciona un enfoque incremental para adoptar los microservicios—si usted comprueba que el enfoque no funciona en su entorno, dispone de una manera sencilla de cambiar de dirección.

¿Cuándo funciona el patrón de "Strangler Application" y cuándo no funciona?

- **Monolito de red o basado en API -** Partir de un monolito existente de red o basado en API es el primer requerimiento para la aplicación exitosa del patrón. El propósito del "strangler pattern" es brindarle a usted una manera de pasar fácilmente de la funcionalidad nueva a la vieja y viceversa. Si su aplicación es una aplicación de red, entonces su estructura URL le da a usted un marco para elegir cómo y qué partes del sistema se implementarán. Sin embargo, cualquier aplicación basada en un conjunto fijo de API, tal como el conjunto de API SOAP que está transformando a REST o incluso un conjunto de colas implementadas en un sistema de mensajería, le permitirá aplicar el patrón. Por otra parte, las pesadas aplicaciones de clientes o las numerosas aplicaciones móviles nativas no se adecuan bien a este enfoque, ya que no tienen necesariamente una estructura que le permita desarmar fácilmente la aplicación.

- **Estructura URL estandarizada (verdadero uso de las URL) -** Aunque las aplicaciones de red trabajan todas de acuerdo a estándares impuestos por la estructura de la red, por ejemplo, HTTP y HTML, usted puede usar una amplia variedad de arquitecturas de aplicación para implementar las aplicaciones de red. Dentro de este enfoque, hay mucho margen de acción, lo que puede complicar su intento de desarmar la aplicación. Por ejemplo, cuando hay una capa intermedia debajo de los pedidos del servidor, por ejemplo, un abordaje de portal, usted puede tener un problema si usa URL para dividir la aplicación. La decisión de conmutar y enrutar no se hace al nivel del explorador, sino más profundamente en la aplicación, lo que es más complicado.

- **Meta UI -** Cuando la UI está basada en un proceso empresarial, o construida al pasar, se hace difícil separar el código para la UI y la lógica empresarial en microservicios diferentes. Este enfoque funciona, pero el tamaño de las partes (ver abajo) es mayor y debe implementarse todo a la vez.

Como no aplicar el “Strangler Application Pattern”

El “Strangler Application Pattern” no es un remedio universal. Usted no querrá aplicarlo en todos los casos o especialmente en estos:

- No lo aplique a una página por vez. La “astilla” más pequeña (ver la sección de gestión de liberación siguiente) es un microservicio único. Ese microservicio necesita ser completo y autosuficiente y, lo que es más importante, necesita la propiedad absoluta de los datos que gestiona. Usted desea evitar tener dos métodos diferentes de acceso de datos que funcionen a la vez con sus datos para evitar problemas de coherencia.

- No aplique todo el patrón al mismo tiempo. Si lo hace, no estará aplicando realmente el “Strangler Pattern” y se encontrará nuevamente en el enfoque Big Bang.

Como aplicar mejor el “Strangler Application Pattern”

Por lo tanto, si una sola página es demasiada pequeña y una aplicación completa es demasiado grande, ¿cuál es el nivel correcto de granularidad para aplicar el patrón “Strangler Application”? Para tener éxito, debe entrelazar dos aspectos diferentes del refactoreo de su aplicación:

- Refactorear su “back-end” al diseño de microservicios (la **parte interior**)

- Refactorizar su “front-end” para acomodar los microservicios y hacer los nuevos cambios funcionales que estén impulsando el refactorización (la **parte exterior**)

Comencemos con la parte interior:

1. Comience por identificar los contextos unidos en el diseño de su aplicación Un contexto unido es otro patrón del libro de Eric Evans “Domain-Driven Design” (Diseño impulsado por los dominios). Según el libro, un contexto unido, “define el contexto dentro del cual se aplica un modelo”.[3] Un contexto unido es un marco conceptual compartido que restringe el significado de una cantidad de entidades dentro de un conjunto mayor de modelos empresariales. En una aplicación para aerolíneas, la reserva de vuelos es un contexto unido, mientras que el programa de lealtad de la aerolínea es un contexto unido diferente. Aunque ambos puedan comparti9r términos tales como “vuelo”, la manera en que esos términos se usan y definen es muy diferente.

2. Elija el contexto unido más pequeño y menos costoso para refactorear. Ordene sus otros contextos unidos según la complejidad, desde los menos complejos a los más complejos. Comience con los contextos unidos menos complejos para probar el valor de su refactoreo y resuelva los problemas que encuentre al adoptar el proceso —antes de que aborde tareas de refactoreo más complejas y potencialmente más costosas.

3. Planifique conceptualmente los microservicios dentro del contexto aplicando los patrones "Entity", "Aggregate" y "Service" de *"Domain-Driven Design"* como se describió antes. En este punto, estará tratando de lograr comprender qué microservicios existen probablemente de manera que puede usar esa aproximación en el siguiente conjunto de pasos.

Luego, pase a la parte exterior:

1. Analice las relaciones entre las pantallas de su UI existente. En particular, busque flujos de gran escala que vinculen varias pantallas juntas, estrechamente. Si está construyendo un sitio para una aerolínea, un flujo podría ser la reserva de un pasaje, que está comprendido por varias pantallas relacionadas que proporcionan información para completar el proceso de reserva. Un flujo diferente podría centrarse en inscribirse para el programa de lealtad de la aerolínea. Comprender el conjunto de flujos le ayudará a pasar al siguiente paso de refactorización.

2. Mientras examina su UI existente o nueva UI, busque los aspectos que corresponden a los microservicios identificados en la parte interior. Identifique cuáles flujos corresponden a cuáles microservicios. El resultado de este paso es una lista de flujos en un lado de una página y una lista de los microservicios que puede implementar cada flujo en el otro lado.

3. Dimensione su trozo con base en la suposición de que los cambios de UI deben ser coherentes en sí mismos. Suponga que uno o más flujos son el tamaño mínimo de cambio que puede liberarse a un cliente, pero el trozo en sí mismo puede ser más grande que un flujo. Por ejemplo, usted podría considerar que toda la lealtad de los clientes sea un trozo único, aunque pueda estar hecho por dos o tres flujos separados.

4. Elija si liberará un trozo completo por vez o un trozo como una serie de astillas.

Referencias

- **IBM Cloud Garage Method, Microservices Architecture Center**
- **End-to-End Reference Implementation (GitHub)**
- **Refactoreo a microservicios**
- **E/S de patrones de microservicios**
- **Explorar los microservicios: Game-On (GitBook)**

[1, 2]”Martin Fowler, *“Microservices a definition of this new architectural term” (Microservicios una definición de este nuevo término arquitectónico)* *https://martinfowler.com/articles/microservices.html*

[3]Eric Evans, *“Domain-Driven Design: Tackling Complexity in the Heart of Software” (Diseño impulsado por los dominios: abordaje de la complejidad en el corazón del software), 2003*

Fin.