OpenLegacy
Digital to the Core

Acelerando el viaje digital de sistemas heredados a microservicios modernos

Zeev Avidan & Hans Otharsson

Acelerando el Viaje Digital de Sistemas Heredados a Microservicios Modernos

Zeev Avidan
Hans Otharsson

ISBN-13: 978-1987762822

Edición 1-B

Tabla de Contenidos

Sobre los autores

Zeev Avidan - Zeev tiene décadas de experiencia con sistemas heredados y su integración. Inició como arquitecto mainframe en 1996 en donde también diseñó e implementó la Arquitectura Orientada a Servicios (SOA). Desde ese entonces Zeev trabajó en la integración Mainframe para SRL y fundó una compañía de servicios con un énfasis especial en la integración y la performance de la IBM i series (AS/400). En el 2011, Zevv comenzó a brindar servicios de consultoría de sistemas heredados e integración a instituciones financieras y gubernamentales. Además, brinda cursos en estas áreas y gracias a su gran experiencia en la materia evalúa las tecnologías y el potencial de start-ups al servicio de empresas de Venture Capital. Zevv se unió a OpenLegacy en el 2014 y actualmente ocupa el cargo de Gerente de Producto.

Aunque su interés en los microservicios es reciente y crece día tras día, Zevv ha estado relacionado con el concepto desde los primeros días del EAI y SOA. Durante los últimos 20 años ha visto de primera mano las dificultades y desafíos de la integración y los sistemas heredados, y cree que los microservicios son una forma de simplificar y resolver problemas del pasado.

Hans Otharsson - Hans es un líder global en materia de programas de transformación de sistemas heredados con décadas de experiencia en el desarrollo, mejora, mantenimiento, troubleshooting y transformación de las llamadas “aplicaciones heredadas” y sus entornos asociados. Su viaje global lo ha llevado a estar en innumerables ambientes y escenarios de negocios, donde su enfoque “directo al grano” le ha permitido brindar a sus clientes un cambio real y una transformación impulsada por los negocios. Su capacidad para evaluar rápidamente una situación y determinar si una organización puede agregar valor – y ofrecer sugerencias para otras alternativas en caso de ser necesarias – lo han convertido en un asesor de confianza para numerosas organizaciones globales. Hans posee muchos años de experiencia con la “modernización de sistemas heredados”, ocupando puestos de ejecutivo senior en Consist Software Solutions y Ateras. También fundó ModernWiser, una consultora que ayuda a las organizaciones a entender sus opciones de modernización de sistemas heredados. En Software AG, Hans fue el responsable de todos los Servicios Profesionales de Venta & Entrega en Norte América y Canadá.

En su puesto actual de Jefe de Operaciones en OpenLegacy, Hans es el responsable de las operaciones corporativas y el éxito del cliente. Estos roles aprovechan las fortalezas de Hans como lo son la calidad de entrega, el cliente primero y una sólida experiencia en

la industria – que se reflejan en su mantra “medimos nuestro éxito a través del éxito de nuestros clientes”.

Introducción

Este libro responde a la pregunta: “¿Cómo se puede acelerar la entrega de servicios digitales innovadores desde tecnologías monolíticas heredadas de una manera que no agregue mayor complejidad y más capas?” En resumen, ¿Cómo puede TI proveer soluciones en función de las demandas del negocio?

Para responder a esa pregunta describiremos las últimas tecnologías y enfoques que involucran las interfaces de programación de aplicaciones (APIs) y los microservicios, y concluiremos con muchos ejemplos de cómo estos han funcionado en otras organizaciones (capítulo 8). Como por ejemplo, el banco que entregó un nuevo sistema de procesamiento de pagos 50% más rápido que otros sistemas mainframe comunes. O la empresa de seguros que finalmente pudo competir con los motores de comparación de precios online.

Nuestra discusión es algo técnica pero está escrita con la intención de clarificar y desmitificar estos conceptos tanto para TI como para las audiencias empresariales. Nuestro objetivo es facilitar conversaciones valiosas alrededor de estos temas, construyendo un

puente entre estos grupos basado en objetivos comunes y entendimiento.

Aunque la innovación y el time-to-market han sido siempre importantes, el mercado millennial genera un nuevo sentido de urgencia. Los millennials son digitalmente impacientes. Ellos no aceptan procesos bancarios tradicionales. Ellos no van a visitar una sucursal, llenar un formulario para luego esperar días y revisar sus correos. A pesar de ello, en muchos bancos en muchas partes del mundo este sigue siendo el proceso de abordaje. Si no puedes ofrecer servicios bancarios a través de un dispositivo móvil, los millennials descargarán alguna app de un competidor y seguirán su camino.

Si no puedes ofrecer servicios bancarios a través de un dispositivo móvil, los millennials descargarán alguna app de un competidor y seguirán su camino.

Las demandas tecnológicas crecen a la misma escala que las capacidades tecnológicas. Cuando la tecnología se vuelve más rápida los consumidores quieren las cosas más rápido. Como resultado, la mayoría de los líderes en las organizaciones requieren de una innovación más rápida, mientras que los líderes técnicos aun enfrentan muchos de los mismos desafíos técnicos.

Hoy en día, un “microservicio moderno” es un aspecto fundamental de la integración de sistemas heredados que consiste en reducir y evitar capas. Consiste también en un acceso rápido al sistema de grabación evitando la innecesaria capa intermedia. Es la realización de lo que décadas atrás esperábamos que nos brindaría la Arquitectura Orientada a Servicios (SOA) y es lo que imaginamos cuando inicialmente creamos los sistemas de grabación heredados back-end. En resumen, el microservicio moderno es el enfoque correcto que justo se encuentra en el lugar y en el momento adecuado.

Sin embargo, hoy en día existe una gran confusión alrededor de los microservicios. Preguntas comunes incluyen cuestiones tan básicas como “qué tan diferentes son de los APIs” y “¿por qué la definición del vendedor suena diferente a esta de aquí?

La verdad es que es fácil confundirse. Los microservicios han cambiado mucho a lo largo de los años y los vendedores se refieren a ellos de muchas formas. Entonces, ¿qué son?, ¿debería importarte? y, sí es así, ¿qué puedes hacer al respecto?

En primer lugar, hablaremos de “sistemas heredados”, “monolitos heredados”, y “aplicaciones monolíticas”. Una definición acertada de estos entornos heredados es que son sistemas desarrollados en el pasado y que aún proveen valor empresarial. Estos fueron construidos utilizando la mejor tecnología de la época pero a

menudo no cumplen con los requerimientos de integración y las necesidades empresariales de hoy en día. Son vistos como complejos silos de lógica empresarial estrechamente acoplada que requieren numerosas capas de abstracción para así reducir y desmitificar, de manera que los recursos DevOps modernos puedan acceder a esos sistemas empresariales.

A diferencia de las aplicaciones monolíticas comunes, los microservicios son pequeños servicios desplegados de manera independiente, enfocados en una funcionalidad empresarial específica.

A diferencia de las aplicaciones monolíticas comunes (capítulo 1), los microservicios son pequeños servicios desplegados de manera independiente, enfocados en una funcionalidad empresarial específica. Los programas escritos en un estilo de microservicios son populares porque son fáciles de entender, desarrollar y testear. El desarrollar una aplicación compuesta por servicios individuales significa que equipos diferentes pueden trabajar en paralelo. Esto hace que sea más fácil y rápido lanzar una colección de microservicios relacionados.

Desafortunadamente, muchos departamentos de TI con sistemas heredados, ESB o SOA, aún luchan por ser tan ágiles como sea necesario en este mundo cada vez más globalizado y digitalizado.

Te puedes preguntar, “Pero, espera un momento – ¿no se suponía que los ESBs y SOA hacían eso? Sí y en muchos casos tuvieron éxito. Desafortunadamente, muchos departamentos de TI con sistemas heredados, ESB o SOA, aún luchan por ser tan ágiles como sea necesario en este mundo cada vez más globalizado y digitalizado. Los microservicios proveen el beneficio de la SOA de muchas formas y a su vez eliminan muchas de las desventajas (Capítulo Dos).

Los microservicios se alinean bien con los procesos ágiles que soportan el desarrollo y la entrega continuos. Estos aspectos son necesarios en empresas donde se requieren actualizaciones frecuentes de datos y programas (Capítulo Tres). Estos se adecuan de forma natural a DevOps.

Los microservicios fueron originalmente un esfuerzo por llevar la modularidad a un nuevo nivel. Operaban principalmente como componentes “tras bastidores”. Los microservicios modernos de

hoy, son servicios frecuentemente orientados al cliente y altamente integrados, utilizados para crear combinaciones de funcionalidades de aplicaciones que eran imposibles. En gran parte, aprovechan las capacidades de los contratos de API para interactuar con una variedad de sistemas back-end. Lo más importante es que los microservicios modernos pueden evitar capas de complejidad existente y ser implementados mucho más rápido que enfoques anteriores (Capítulo Seis).

Este libro es para cualquier líder TI, DevOps o empresarial dentro de una organización que se encuentre considerando los microservicios como una manera de aprovechar los datos heredados de forma rápida y eficientemente en las tecnologías modernas. Esperamos que tu viaje digital te lleve a la eficiencia de TI, a ciclos más rápidos, mayor escalabilidad y diferenciación competitiva.

Esperamos que tu viaje digital te lleve a la eficiencia de TI, ciclos más rápidos, mayor escalabilidad y diferenciación competitiva.

Capítulo Uno:

De Aplicación Monolítica a Microservicios

Si tu organización no puede innovar lo suficientemente rápido para satisfacer las demandas empresariales y competitivas, considera un enfoque de microservicios para tus aplicaciones monolíticas. Si te estás preguntando "¿qué es una aplicación monolítica?", es posible que estés usando una en este momento.

Normalmente el departamento de TI desarrolla una aplicación – usualmente una aplicación de un solo núcleo o una serie de aplicaciones relacionadas – y combina todos los elementos en un sistema como se muestra en la Figura 1.

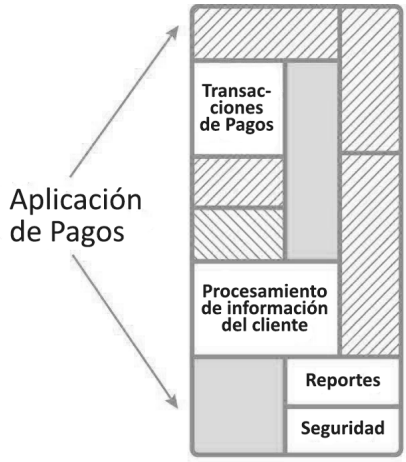


Figura 1. Ejemplo de una aplicación monolítica en donde múltiples elementos y funcionalidades se combinan en una aplicación.

Por ejemplo, la funcionalidad se mezcla en una gran aplicación; es posible que tengas una aplicación de pago que incluya la información del cliente,

transacciones de pago y funcionalidades adicionales como informes y seguridad. A veces este enfoque tiene sus ventajas y algunas aplicaciones son más fáciles de desarrollar de esta manera. Por otro lado, los monolitos crean mucha ineficiencia y desafíos en la usabilidad de código.

Lanzamientos Lentos – El Enemigo de la Velocidad

Si bien este enfoque monolítico puede haber funcionado en el pasado, el mundo ha cambiado. La mayoría de organizaciones tienen numerosos equipos de desarrollo, todos tratando de crear, actualizar y testear su código antes de su lanzamiento (Figura 2).

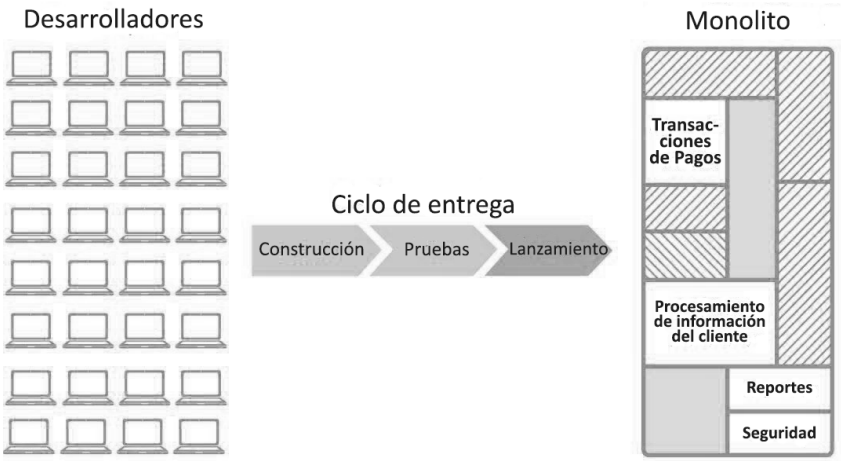


Figura 2. Las aplicaciones monolíticas son en su mayoría incompatibles con los entornos Agile y DevOps de hoy en día. Los ciclos de desarrollo, testeo y lanzamiento suelen ser mucho más largos de lo que se requiere en términos de velocidad y escala.

Lanzar a producción una o dos veces al año solía ser la norma, pero en una economía global de rápido ritmo como la actual esto ya no es funcional para la mayoría de las empresas. Las empresas dependen de los enfoques Lean y Agile cuando necesitan ciclos de lanzamiento más rápidos – sin embargo, la aplicación monolítica es la enemiga de dicha velocidad.

Problemas con Código Compartido y Datos

Una aplicación puede ser monolítica si existe acoplamiento de software, como código compartido o datos compartidos (ver Figura 3).

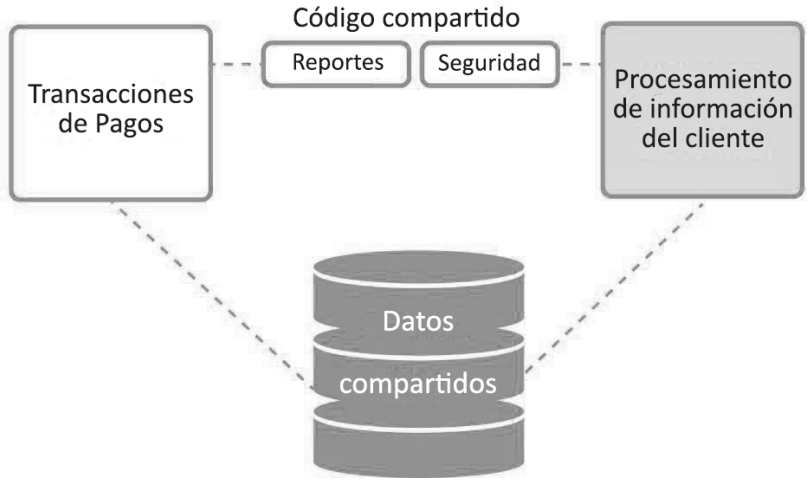


Figura 3. En una aplicación monolítica típica, el mismo código o datos pueden ser compartidos por diferentes módulos. Por lo tanto, cualquier cambio en ese código o datos requiere del testeo de todo el monolito.

El código compartido significa que los datos rutinarios centrales son compartidos por muchos módulos en la aplicación. Por ejemplo, una compañía de tarjetas de crédito probablemente tenga una rutina central de validación de dígitos que verifica la validez de los dígitos en una tarjeta de crédito. Casi todos los módulos en la aplicación pueden utilizar esta rutina – ya sea en el proceso de aprobación, el sistema de alertas o la declaración del cliente – porque cada parte necesita verificar que el número de tarjeta sea válido. En consecuencia, si realizas un cambio a esta rutina central habrás realizado un cambio en todas y cada una de las funcionalidades de la aplicación. Si algo sale mal con este módulo entonces la aplicación fallará.

> Este "castillo de cartas" interconectado es precisamente la razón por la cual a la mayoría de organizaciones les resulta difícil innovar con sistemas heredados.

Los datos compartidos plantean un problema similar. Por ejemplo, muchos componentes de la aplicación monolítica utilizan la información del cliente. Compartir el código y los datos resulta en un acoplamiento muy fuerte. Aunque resulta más fácil mantener una base de código o una base de datos, es más difícil efectuar y administrar cambios.

Dado que los cambios pueden ser impredecibles, las empresas necesitan testeos extensos que consumen mucho tiempo – a

menudo toman meses – con la finalidad de asegurarse que los cambios en el código sean lo suficientemente estables para pasar a la producción. De hecho muchas organizaciones evitan corregir errores porque les preocupa las consecuencias que ello pueda traer.

Este "castillo de cartas" interconectado es precisamente la razón por la cual a la mayoría de organizaciones les resulta difícil innovar con sistemas heredados.

Sin Escalabilidad

Con el fin de mejorar la escalabilidad, las empresas han tratado de separar el monolito en componentes de aplicaciones principales que se ejecutan en diferentes imágenes del servidor o plataformas. A pesar de ello, muchas aplicaciones están tan acopladas con código y datos que no pueden ser fácilmente modernizadas. No es posible desplegar un componente de funcionalidad de una aplicación en un lugar y otra parte de la funcionalidad en otro. Esto le pone un límite a la escalabilidad.

Los problemas de escalabilidad, la falta de agilidad, los largos tiempos de lanzamiento, una lenta innovación, los desafíos complejos y la gestión de riesgo son retos bien conocidos del monolito (Figura 4).

Figura 4. Desafíos comunes del monolito de aplicaciones heredadas

¿Qué es un Microservicio?

Los microservicios son un estilo de arquitectura de aplicaciones que tiene poco más de una década y cuyo interés y adopción ha crecido rápidamente (Figura 5).

Figura 5. El interés en la arquitectura de microservicios ha crecido rápidamente desde el 2012 como una forma de resolver problemas comunes en los monolitos de aplicaciones. Fuente: Google Trends

La idea de los microservicios nació de las experiencias con la arquitectura orientada a servicios (SOA). A diferencia de la SOA, los microservicios estructuran una aplicación como una colección de servicios independientes cuyo alcance es limitado y que se comunica utilizando protocolos muy eficientes. En esencia, los microservicios son un estilo formal de arquitectura para desacoplar funcionalidades empresariales de un monolito. A su vez, cuando encapsulas una Interfaz de Programación de Aplicaciones (APIs) en el microservicio, estás evitando muchos de los inconvenientes de los diseños monolíticos.

Por ejemplo, en los sistemas de registro actuales los datos y procesos están a menudo bloqueados, como la información del cliente y el proceso de verificación de tarjeta de crédito que se utilizan en tus aplicaciones monolíticas. Los microservicios incluyen APIs que intencionalmente tienen limitada su funcionalidad. Por ejemplo, un sistema de pago no existiría dentro de un microservicio. En su lugar, el sistema de pagos estaría compuesto por una malla de microservicios – uno para obtener los detalles del cliente, uno para transferir dinero, etc. Cada servicio se encuentra acoplado de manera flexible permitiendo que la aplicación general funcione aun cuando un servicio esté caído. Lo anterior permite que existan diferentes servicios en diferentes servidores y diferentes nubes o permite también que los distintos componentes se escalen de forma independiente (Figura 6).

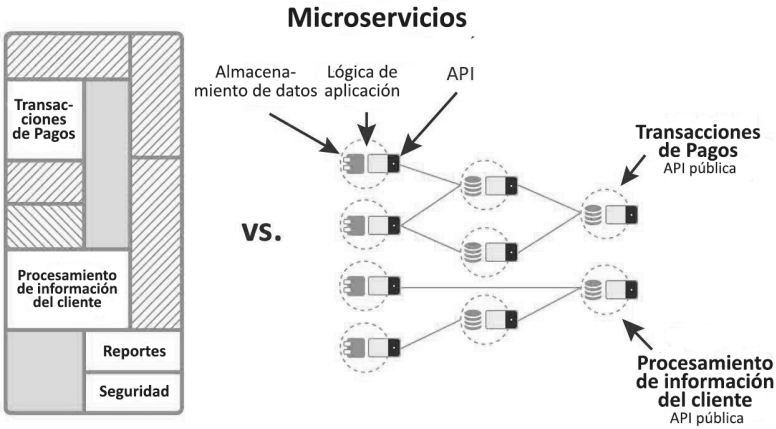


Figura 6. Mientras que una aplicación monolítica típica está estrechamente acoplada, los microservicios desacoplan

funcionalidades empresariales independientes en servicios separados de tal modo que ninguna funcionalidad interfiera con las otras.

Algunas Veces Es Necesario Dividir

La modularidad se ve mejorada al dividir una aplicación en distintos microservicios más pequeños y esto hace que la aplicación sea más fácil de entender, desarrollar y testear. Esto también permite un desarrollo paralelo al permitir que pequeños equipos de trabajo autónomos desarrollen y desplieguen sus servicios de manera independiente. Finalmente, los microservicios permiten que la función de un servicio individual emerja a través de mejora continua permitiendo una entrega y despliegue continuos.

Velocidad de Desarrollo de Microservicios en Aplicaciones Heredadas

El principal beneficio de los microservicios no es únicamente la velocidad de desarrollo sino que también la concurrencia de desarrollo. Los desarrolladores pueden trabajar en distintas partes de la misma aplicación al mismo tiempo y esto tiene enormes beneficios para empresas que trabajan con aplicaciones heredadas.

El desarrollo de aplicaciones modernas requiere de un ritmo acelerado, pero usualmente trabajar con aplicaciones heredadas es lento y laborioso. Los microservicios eliminan esa carga de

trabajo. A través de estos es posible exponer funcionalidades empresariales, si consideras la aplicación heredada como un almacén de datos, en donde encapsulas la lógica de la aplicación (funcionalidades empresariales) dentro de un API. De este modo, es posible utilizar la salida de COBOL de una aplicación heredada en un formato que es comprensible para una API REST.

> El principal beneficio de los microservicios no es únicamente la velocidad de desarrollo sino que también la concurrencia de desarrollo.

Los desarrolladores pueden crear otros microservicios que representan características relacionadas con la aplicación heredada y conectarlas a la aplicación sin necesidad de escribir o modificar algún código heredado. De esta manera es posible que un equipo de desarrollo agregue rápidamente nuevas características a la aplicación heredada y que otro equipo trabaje en el mantenimiento de la base de código heredada. Todo esto sin necesidad de que un equipo deba comunicar o ralentizar su trabajo en función de las demandas del otro. Los microservicios permiten un rápido desarrollo de proyectos que involucren aplicaciones heredadas.

Monitoreo de Microservicios

Antes de la era de los microservicios era común que los desarrolladores creasen un único monolito con cientos de APIs y después ejecutaran esos APIs en un único contenedor. El

inconveniente de esto fue discutido anteriormente – la caída de un único componente puede llevar a la caída de toda la aplicación. Por otro lado, una ventaja (extremadamente débil) de esto era la facilidad de monitorear la condición de la aplicación pues todo se encontraba en un solo lugar.

En la era de los microservicios modernos es posible tener una aplicación compuesta por cientos de servicios. Algunos de estos servicios pueden no estar en el mismo servidor o en la misma nube. Si bien la aplicación resultante será extremadamente tolerante a los fallos, vale la pena preguntarse cómo monitorear los microservicios antes de hacer la transición a ellos. Considera para ello buscar un proveedor de software o soluciones, como OpenLegacy, que esté equipado para realizar analíticas y monitoreo de cientos de microservicios que se ejecutan en paralelo.

Con estos beneficios en mente no es sorpresa que los analistas líderes en la industria estén sugiriendo a las organizaciones prestar más atención a los microservicios. Forrester explica que los microservicios juegan un importante rol en el futuro de la arquitectura de soluciones[1] y destaca como principales beneficios una entrega más rápida de software, una mayor resiliencia operativa y escalabilidad y un mejor mantenimiento. Gartner indica que la arquitectura de microservicios permite una agilidad y escalabilidad sin precedentes[2].

[1] *Forrester, Microservices Have An Important Role In The Future Of Solution Architecture, July 2015*

[2] *Gartner, Innovation Insight for Microservices, January 2017*

Capítulo Dos:

Microservicios & Arquitectura Orientada a Servicios (SOA)

Cuando se analizan las ventajas de una arquitectura de microservicios en comparación con aplicaciones monolíticas es fácil preguntarse por qué nadie adoptó este enfoque desde el inicio. Después de todo, más del 90% de las aplicaciones empresariales actuales en el mundo son monolíticas.

Cuando estas se crearon, el diseño monolítico era simplemente el mejor enfoque disponible para satisfacer las demandas tecnológicas y empresariales. Sin embargo, las necesidades empresariales han evolucionado y la demanda por una mayor agilidad ha aumentado. A los desarrolladores y al equipo de TI se les ha encomendado la tarea de dejar estas arquitecturas monolíticas.

El enfoque resultante, al menos en los últimos años, ha sido el de una arquitectura orientada a servicios (SOA). A pesar de que la SOA resolvió muchos problemas, no los ha resuelto todos. El término “microservicios” es un subconjunto de la terminología SOA (Figura 1).

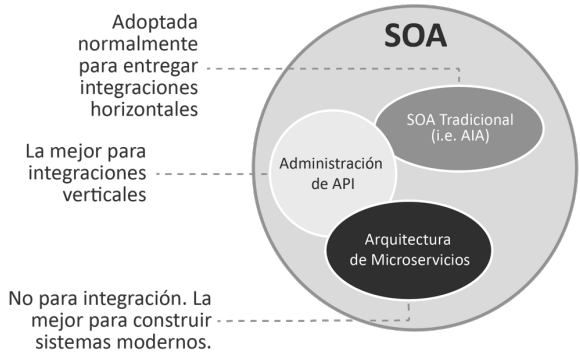


Figura 1. Los Microservicios y APIs se pueden pensar como un subconjunto de la Arquitectura Orientada a Servicios.

El Desafío de la Integración SOA

La promesa de las iniciativas SOA fue extender el alcance de las funcionalidades centrales de la empresa y reducir los gastos internos y la complejidad que traían los monolitos. Las SOA lograrían este objetivo dividiendo las funcionalidades centrales de un monolito en servicios web, utilizando protocolos como el protocolo simple de acceso a objetos (SOAP) y el lenguaje de marcas extensible (XML).

El SOAP fue construido para comunicaciones de aplicación universal. Debido a que está basado en XML, una SOA diseñada con SOAP puede, en teoría, ser utilizada para crear una capa de integración agnóstica. En lugar de luchar por integrar varios

sistemas propietarios, estos protocolos abrirían monolitos en diferentes sistemas operativos para que puedan trabajar juntos.

Una capa de integración agnóstica permite a los administradores del sistema conectar piezas de un monolito a un bus de servicio empresarial (ESB) para lograr una SOA ágil y plug-and-play. Este enfoque fue exitoso en algunos casos. Hay muchas "SOA heredadas" en el mundo que aún proporcionan valor. Sin embargo, en estos casos el valor que brinda la SOA se encuentra tras bastidores de la comunicación servidor-servidor que es la que mayormente ayuda a los desarrolladores. Cuando se trata de servir a los clientes – y a sus demandas rápidamente cambiantes – la SOA no siempre está a la altura del trabajo.

El problema es que el ESB supervisa los mensajes que logran la integración del servicio para que lleguen a su destino. Esta comunicación no es tan simple como los proveedores de SOA podrían haberlo prometido.

Cuando se trata de servir a los clientes – y a sus demandas rápidamente cambiantes – la SOA no siempre está a la altura del trabajo.

Tomemos por ejemplo un enfoque de integración SOA con una aplicación central bancaria. Esto implica un mensaje que va desde la aplicación central en tu mainframe hasta un servidor de una sucursal. Si la lógica empresarial decide que el mensaje es relevante únicamente durante un día hábil, los administradores deben decidir si este se debe mover a una cola, registrarse o ser ignorado cuando pase el día. Este es un escenario común para aplicaciones bancarias centrales en una SOA, pero ¿qué tan bien funciona?

En este ejemplo (y en otros similares), el ESB debe saber si el día hábil ya pasó o no para poder tomar la decisión correcta sobre a dónde enviar el mensaje. Esto significa que la integración requiere de un algoritmo. Aun cuando es un simple algoritmo, el ESB no puede simplemente confiar en una regla universal para enviar mensajes entre el monolito y la integración externa.

Cuando los administradores introducen una nueva lógica empresarial en la integración de monolito, lo que suponía ser una capa de servicio agnóstica se convierte en una nueva capa de aplicación y esto aumenta la complejidad. En lugar de simplemente integrar el monolito a un nuevo servicio digital, la empresa termina con un monolito más grande – uno que incluye el mainframe y todas las nuevas pilas de integración que han sido agregadas. A pesar de las promesas de la SOA, los problemas de integración resultan en mayores tiempos de mantenimiento, mayor complejidad del código y software, y el crecimiento continuo (no

la eliminación) de las aplicaciones monolíticas. La primera regla de una integración efectiva es “smart end-points and dumb pipes”. La creación de lógica y capas en la capa de servicio rompe esa regla, le suma a la complejidad general y le agrega otra aplicación heredada a tu portafolio.

Intentar optimizar el enfoque SOA solo dará como resultado monolitos más grandes. El tomar un nuevo enfoque con arquitecturas de microservicios ayudará a cumplir la promesa original de la SOA (Figura 2).

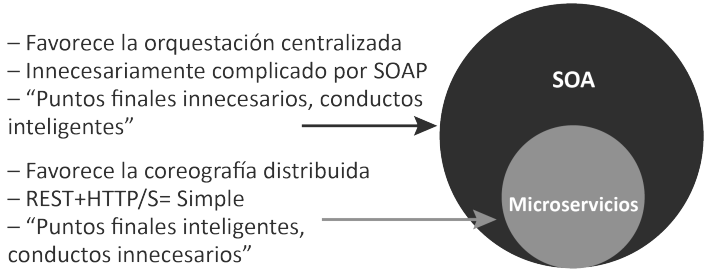


Figure 2. Los microservicios cumplen con la promesa original de la SOA.

Mejorando las SOA con los Microservicios

Durante dos décadas, los DSI han intentado hacer la transición de los monolitos, adoptando enfoques tradicionales a la integración,

sólo para darse cuenta de que han duplicado sus inversiones en sistemas heredados. Todas estas pilas de integración terminan acopladas a sistemas heredados y sólo resultan en más trabajo para el equipo de TI – y en una organización menos ágil.

El paso de integración de la SOA nunca tuvo la intención de incluir lógica empresarial. El tratar de forzar lógica empresarial a este enfoque conduce a soluciones temporales y a un esfuerzo extra para lograr un resultado menor que el ideal.

El factor clave aquí es cómo incorporar microservicios sin efectos adversos. Afortunadamente, la arquitectura de microservicios puede ser forjada desde las SOA introduciendo nuevos principios.

Uno de estos principios es el mapeo contextual. Cuando se reemplaza una SOA existente, los equipos deben considerar el tamaño y el alcance del nuevo microservicio y aplicar los límites contextuales apropiados. Por ejemplo, las SOA que generalmente buscan una integración con servicios digitales deberán dividirse en dominios más pequeños para simplificar la operación.

Otro principio clave es la idea de una arquitectura share-nothing. Muchas integraciones SOA generan dependencias que crean complejidad en la pila tecnológica. Los microservicios evitan estas dependencias entre servicios. Es importante que aquellos que busquen trasladarse de una SOA existente hacia los microservicios miren la lista de dependencias y que trabajen hacia una funcionalidad independiente.

En última instancia, el objetivo debe ser refactorizar los monolitos de una manera tal que se desplace la pila de TI hacia los microservicios. Adoptando el enfoque correcto puedes aprovechar al máximo tanto los microservicios como los APIs dentro de tu SOA.

Capítulo Tres:

Microservicios vs. APIs

Ya hemos explicado como diferenciar las SOAs de los microservicios pero diferenciar las APIs de los microservicios es otra cosa. Las empresas se están haciendo la siguiente pregunta: ¿nos permitirán los microservicios crear productos más valiosos? También se están preguntando si es necesario dominar las APIs antes de buscar un enfoque de microservicios. Tal vez esa ni siquiera sea la mejor pregunta. Tanto los CEOs como los compradores de tecnología utilizan las terminologías de microservicio y API de forma intercambiable. Sin embargo, la diferencia no es simplemente una pregunta académica (Figura 1).

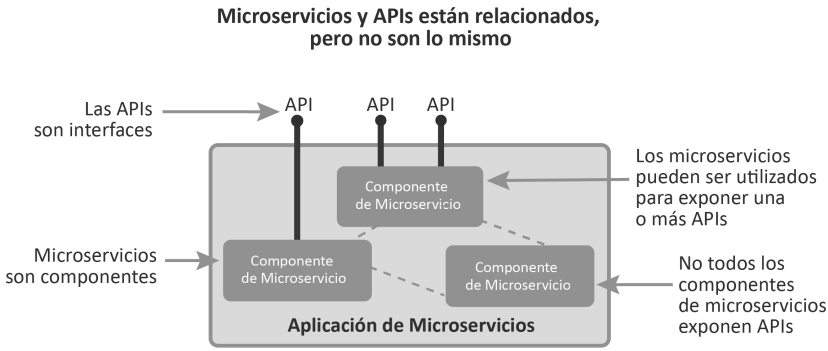


Figura 1. Todos los microservicios incluyen APIs, pero no todas las APIs son microservicios.

La diferencia entre los microservicios y las APIs es importante para empresas que están comenzando a planificar su participación en una era más digital. Puedes implementar microservicios sin exponer APIs y puedes crear APIs sin utilizar microservicios. La opción que elijas depende en mayor medida de las necesidades de tu empresa.

Entonces, tratemos de ser claros en cuanto a las diferencias y similitudes entre las APIs y los microservicios. El concepto fundamental detrás de una arquitectura de microservicios es dividir tu aplicación o aplicaciones en muchos servicios pequeños. Típicamente, cada uno de estos servicios tendrá su(s):

- Responsabilidad distintiva relacionada con la empresa.
- Proceso de Ejecución
- Base de Datos
- Versiones
- API
- UI (Interfaz de Usuario)

El concepto fundamental detrás de una arquitectura de microservicios es dividir tu aplicación o aplicaciones en muchos servicios pequeños.

Un microservicio debe exponer una API bien definida (Figura 2).

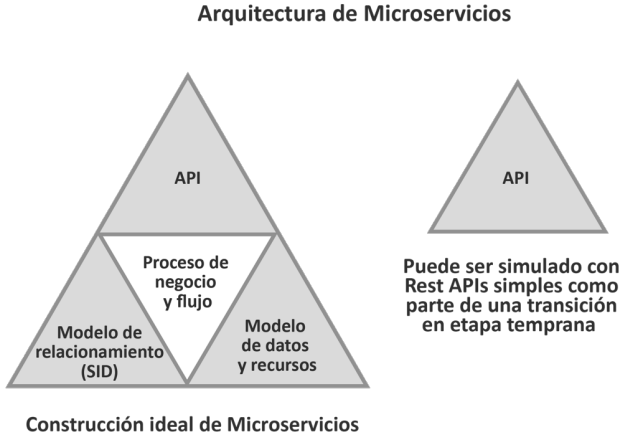
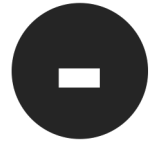


Figura 2. Hay ocasiones donde una API es suficiente – y preferible – y ocasiones donde un microservicio es mejor. Un microservicio expondrá una API, pero también incluye otros elementos.

Los microservicios son la manera en que quieres diseñar tu solución, mientras que la API es lo que el consumidor ve, teniendo en mente que puedes exponer una API sin una arquitectura de microservicios.

Una regla general a seguir es que mantengas tus servicios pequeños y que tengas muchos de ellos en lugar de crear servicios más grandes. Luego, debes entender que un microservicio puede utilizar una transferencia de estado representacional (REST), una cola de mensajes o cualquier otro método para comunicarse entre sí, entonces una REST es ortogonal (independiente) al tema de los microservicios (Tabla 1).

Tabla 1. Cuándo elegir microservicios, APIs o ambos.

API	Microservicios	Estado de la Aplicación
		Tu aplicación es aún lenta de cambiar y difícil de integrar. El tiempo para su comercialización será lento y esta no es realmente un activo en tu Iniciativa de Transformación Digital.
		Tu aplicación es aún lenta de cambiar pero ahora es fácil de integrar. Esto significa que tu tiempo de comercialización y tu habilidad para acceder a esta aplicación heredada es fácil y rápido. Ahora tu aplicación puede ser un colaborador eficaz en tu viaje de Transformación Digital.

		Un nirvana de integración – tu aplicación es fácil de cambiar y rápida de integrar. Lo más importante es que tu habilidad para agregar nuevos productos y servicios a tu aplicación heredada es la óptima.

Microservicios: una Arquitectura Optimizada para APIs

Desde la perspectiva de un desarrollador, los microservicios son un enfoque que mejora el rendimiento, y por ende el valor, de las APIs y las aplicaciones que estos crean. Recuerda que en el fondo, una API sigue siendo un contrato. La API recibe una cierta entrada y entrega una cierta salida. Dependiendo de si la API está construida con una arquitectura de microservicios, esa salida llegará de cierta manera. Los microservicios imponen un conjunto de reglas sobre las APIs que las hace más simples, más modulares, y más funcionales.

Microservicio	**API**
Una función	Muchas funciones
Un almacén de datos	Muchos almacenes de datos
Comunicaciones simples	Tuberías complejas

Una Función vs. Muchas Funciones

Una API estándar puede estar construida para comportarse de distintas formas. Por ejemplo, esta puede aceptar un tipo de entrada como un número de teléfono de un cliente y puede devolver el ID y la dirección del cliente. Si a la misma API se le da el mismo ID del cliente esta podría devolver su número de tarjeta de crédito y teléfono, y otras salidas distintas dependiendo de qué otra información recibe.

Los microservicios limitan estrictamente el tipo de información que una API puede devolver. Una API de microservicios va a devolver únicamente una o dos piezas de información dependiendo de la entrada. Otro microservicio distinto maneja el resto de cosas. La razón de esto radica en la forma en que los microservicios se relacionan con sus almacenes de datos.

Un almacén de datos vs. Muchos almacenes de datos

Para que una API común devuelva tantos tipos de datos, el componente que la implementa deberá conectarse a muchos almacenes de datos distintos. Aquí es donde las APIs pueden comenzar a ralentizar el proceso de desarrollo. Si uno de esos almacenes de datos es actualizado, la información que se devuelve podría cambiar. Si más de una API está conectada al mismo almacén de datos entonces cualquier cambio las afectará a todas.

Los microservicios devuelven únicamente uno o dos tipos de datos y la mejor práctica es conectarlos con un único almacén de datos. De esta manera, un equipo de desarrollo deberá testear una única API antes de enviar los cambios a producción. Esto elimina las disputas multidisciplinarias y previene la duplicación de esfuerzos.

Comunicaciones Simples vs. Tuberías Complejas

Una vez que su información es recuperada, las APIs deben hacer algo con ella. Algunas veces esto significa simplemente transmitir la información al usuario final pero es más frecuente que esta sea transmitida a otro sistema automatizado. En una aplicación ordinaria, la API puede llamar a cualquier almacén de datos o a cualquier otra API según lo determine la lógica de dicha aplicación. Esto necesariamente hace que la aplicación esté más estrechamente unida, sea menos modular y más monolítica. En una arquitectura de microservicios, la lógica de la aplicación vinculada a una API en particular puede llamar únicamente a otras APIs. Esta comunicación simplificada previene que la aplicación se caiga cuando se cambia o se remueve un microservicio. La flexibilidad agregada simplifica y optimiza el proceso de desarrollo.

Una implementación API se mantiene por sí misma. Esta puede aceptar cualquier comando, ser conectada a cualquier base de datos y hacer llamadas a cualquier otra aplicación o servicio. Sin embargo, esta aparente flexibilidad puede hacer a las aplicaciones más monolíticas, menos flexibles y más difíciles para trabajar. En

contraste, los microservicios encapsulan las APIs con una serie de reglas y componentes que, en última instancia, liberan a las aplicaciones de las restricciones monolíticas.

Una API simple en un marco de microservicios puede ser creada en un día utilizando tecnologías probadas.

En resumen, las empresas se están empezando a dar cuenta que el simple hecho de tener APIs no las convierte en una organización especialmente innovadora o con visión a futuro. Si el crear e implementar una API toma cuatro meses, entonces no existe una velocidad inherente en el desarrollo. Los microservicios cambian la conversación. Una API simple en un marco de microservicios puede ser creada en un día utilizando tecnologías probadas.

Dependiendo de sus requerimientos, las empresas que en este momento adopten los microservicios tendrán una ventaja enorme frente a empresas que dependen de las APIs tradicionales. Los usuarios de microservicios podrán desarrollar aplicaciones con más valor y agregar funcionalidades más rápidamente que sus competidores. En otras palabras, si aún no has considerado los microservicios, prepárate para que eso cambie muy pronto.

Capítulo Cuatro:

Anatomía de la Arquitectura de Microservicios

Un microservicio está construido de una manera específica que incorpora tres partes: la API, una unidad de lógica de aplicación y un almacén de datos (Figura 1). La arquitectura de microservicios permite cierta variación dentro de este diseño mientras se adhiera a los siguientes preceptos.

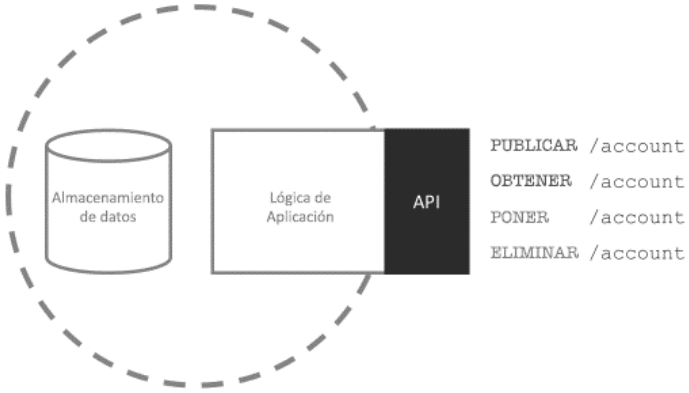


Figura 1. Un microservicio incluye tres partes: un almacén de datos, una lógica de aplicación y una API.

Anatomía de los microservicios: La API

Cada microservicio incluye una API pero esta debe estar escrita de una manera específica. La arquitectura de microservicios favorece a tareas muy limitadas y específicas, y es por ello que la API

incorporada tendrá usualmente un único rol, representando un contrato público. Por ejemplo, el contrato para un agente de escritorio puede involucrar una API que proporciona el nombre, número de teléfono y la dirección de un cliente cuando se le brinda el ID del cliente.

La arquitectura de microservicios favorece a tareas muy limitadas y específicas, y es por ello que la API incorporada tendrá usualmente un único rol, representando un contrato público.

Algunas APIs pueden tener una función más amplia – digamos que el cliente olvida su ID, entonces el agente puede recuperarlo al proporcionar en su lugar el número de teléfono del cliente. Sin embargo, ir mucho más allá de ello representa por lo general una mala práctica.

Además, la API deberá en sí misma retener funciones anteriores cuando se le den nuevas. En otras palabras, el equipo de desarrollo puede decidir mejorar la funcionalidad de su API pero no puede reemplazarla. El hacerlo podría interrumpir flujos de trabajo u ocasionar fallas en cascada.

Anatomía de los microservicios: Lógica de la Aplicación

En los microservicios, la aplicación o el componente de lógica empresarial agregan una medida de inteligencia a la API. Por

ejemplo, imagina una plataforma de comercio electrónico construida con microservicios. Un cliente hace clic en “hacer un pedido”. Esto le indica a un microservicio que verifique si el cliente que hace el pedido tiene una cuenta, información de facturación y dirección válidas. Una vez que esto finaliza, el microservicio debe pasar cierta cantidad de información:

- ¿Pudo la aplicación validar al cliente?
- En caso negativo, ¿por qué no pudo?
- En caso afirmativo, llama al siguiente microservicio en la secuencia para crear una orden.

Anatomía de los microservicios: Almacén de Datos

Las mejores prácticas de la arquitectura de microservicios sostienen que los microservicios nunca comparten datos – un microservicio encapsula su almacén de datos y otras APIs nunca llamarán a ese almacén de datos directamente. Esto le permite a los desarrolladores realizar cambios en sus almacenes de datos sin que ello afecte a otros microservicios, acelerando en gran medida el proceso de desarrollo y testeo.

Otra regla fundamental en el almacenamiento de datos en los microservicios es utilizar la base o bases de datos que mejor se acoplen al uso del microservicio. Esto le permite a los desarrolladores, por ejemplo, utilizar tanto la base de datos SQL como la NoSQL dentro de la misma aplicación, reteniendo los

beneficios de la NoSQL sin renunciar a las transacciones ACID (atomicidad, consistencia, aislamiento, durabilidad) y a otros aspectos positivos de las bases de datos relacionales. Esto se refleja en el lado de la API con la programación políglota que permite que la lógica de aplicación y las APIs se escriban en cualquier lenguaje que agregue la combinación correcta de funcionalidad y eficiencia.

Los Microservicios son como una Mini-Aplicación

En resumen, se puede considerar que un microservicio incorpora los aspectos de la arquitectura cliente-servidor de tres niveles comúnmente utilizada en aplicaciones web como comercio electrónico: una capa de presentación, una capa de negocio y una capa de datos.

- **Capa de presentación** - es el mecanismo que se utiliza para enmarcar la comunicación en una arquitectura de tres niveles. La API, en el paradigma de microservicios, es el contrato que define el mecanismo de comunicación.

- **La Capa de negocio** - funciona igual dentro de una arquitectura de tres niveles y una arquitectura de microservicios. Esta coordina el inicio y final de las tareas y actividades junto con la gestión de la comunicación con otros servicios.

- **La Capa de Datos** - gestiona el empaquetado y la exposición de la información (datos) a la lógica de la Capa de Negocio, que a su vez es transmitida a la API (Contrato de Presentación).

Aunque la aplicación de tres niveles contiene capas de separación horizontal, los microservicios agregan divisores verticales, exponiéndose a otros microservicios únicamente a través de la capa lógica de presentación. Esta es la mayor diferencia anatómica entre los microservicios y las aplicaciones monolíticas. Al aislarse a sí mismos del resto de la aplicación, los microservicios pueden ser administrados a lo largo de toda su vida por un solo equipo, eliminando la posibilidad de una lógica de negocios mal ubicada, esfuerzos duplicados y otros aspectos negativos.

Capítulo Cinco:

Buenas Prácticas de Microservicios

Los enfoques hacia los microservicios han evolucionado hasta volverse menos complejos de lo que eran hace algunos años. En lugar de módulos con preocupaciones de conectividad y transmisión de mensajes, los microservicios modernos son aplicaciones de datos que pueden ser creadas más fácilmente que en el pasado y luego usarse de manera independiente o combinada en aplicaciones utilizando APIs. Estos hacen que los datos importantes y procesos se encuentren disponibles de nuevas formas sin interrumpir los sistemas de registro a los que acceden.

Cuando inviertes en microservicios modernos puedes esperar; escalar con facilidad, ciclos de lanzamiento rápidos, mayor agilidad, innovación más rápida y menor riesgo. Estas motivaciones se encuentran detrás de las siguientes siete reglas de los microservicios:

1. Evitar capas donde sea posible.
2. Accede únicamente a APIs públicas.
3. Utiliza la herramienta adecuada para el trabajo.
4. Asegura todos los niveles.
5. Sé un buen ciudadano pero ten una buena vigilancia.
6. No se trata únicamente de tecnología.

7. Automatiza todo.

Regla Número 1: Evita capas donde sea posible.

Muchos lectores pueden tener experiencia de primera mano o percibir que la integración de los microservicios es algo complejo de hacer. Mientras que en el pasado la integración de microservicios requería navegar a través de las complejas capas de tu arquitectura existente (Figura 1), los microservicios evitan estas arquitecturas para conectarse directamente con tus sistemas monolíticos heredados, como mainframes, sistemas de rango medio y bases de datos (Figura 2).

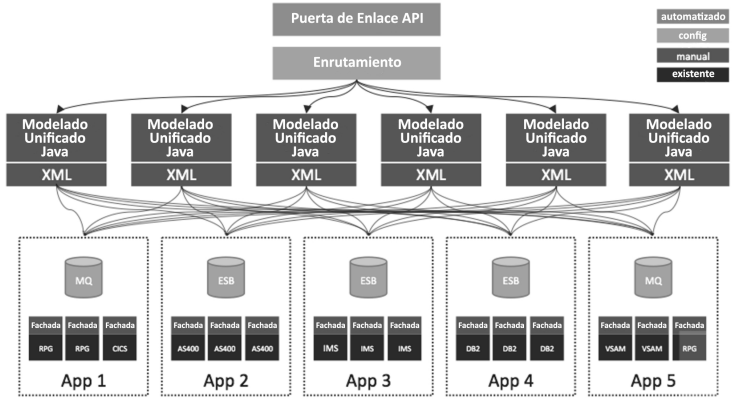


Figura 1. Típicamente, la integración de microservicios ha sido difícil de hacer debido en gran medida a un esfuerzo manual complejo.

Algunos softwares (como OpenLegacy) consumen la lógica del sistema heredado para poder determinar el mejor método de conexión al sistema. Luego, en el tiempo de ejecución, esa

información es utilizada con conectores preintegrados para conectarse automáticamente con el sistema heredado y evitar tantas capas como sea posible (Ver Regla Número 3 – Poliglota Back-end).

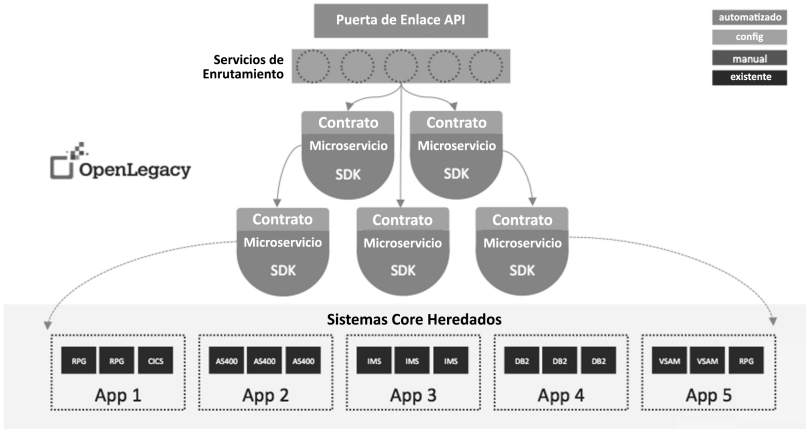


Figura 2. Algunos proveedores (como OpenLegacy) evitan capas de arquitectura para conectarse directamente con los sistemas monolíticos heredados existentes. Además, muchos procesos manuales están automatizados, simplificando y acelerando así el proceso.

Los microservicios se pueden crear muy rápidamente, sin complejidad y sin necesidad de habilidades de programación especiales como COBOL o RPG, o cambios invasivos a los sistemas subyacentes.

Dado que el valor de los microservicios no se puede alcanzar sin antes haberlos creado, este es un punto que no podemos pasar por

alto. En nuestra experiencia, el mayor obstáculo para las organizaciones que adoptan microservicios o APIs para monolitos heredados es el tiempo y el esfuerzo para su creación.

Regla Número 2: Accede únicamente a APIs públicas

La creación de nuevas aplicaciones empresariales utilizando APIs Públicas está cambiando fundamentalmente la forma en que se crea y se entrega software al mercado. La API pública, creada con REST o SOAP, tiene un contrato que gobierna su acceso, entonces la regla es que debes únicamente interactuar con el contrato del microservicio. Esto es importante porque para que un microservicio esté acoplado o combinado, este debe utilizarse de una manera específica para preservar su aspecto modular. Cuando utilizas el contrato de servicio evitas problemas que pueden surgir al leer la base de datos del servicio o la cola de mensajes directamente. Si evitas el contrato, estarás dependiendo de los atributos físicos del servicio y puedes encontrar problemas cuando haya un cambio en el código del microservicio (Figura 3).

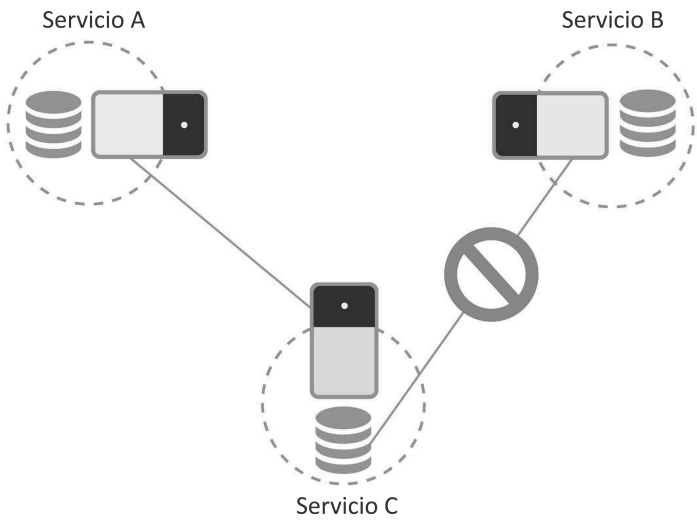


Figure 3. Interactúa únicamente con el contrato de la API del microservicio, de lo contrario podrías encontrar problemas si hay un cambio en el código del microservicio.

¿Cómo cambia y evoluciona un microservicio? Existe un proceso para cambiar y hacer crecer el microservicio. Esto se maneja a través de versiones del contrato, de modo tal que las organizaciones pueden tener algunas aplicaciones que usan una versión del contrato mientras que otras aplicaciones usarán un contrato diferente.

Por ejemplo, los microservicios pueden dividir una aplicación en componentes funcionales que son combinados con otros microservicios para crear una API externa. Al crear este tipo de "red de funcionalidades", estas combinaciones utilizan un SDK

para acceder al sistema heredado como un mainframe. Estos microservicios se usan para crear nuevas aplicaciones empresariales. Los nuevos contratos que utilizan no están en el mainframe sino que están dentro del mismo microservicio que accede al mainframe.

Tener una red de funcionalidades facilita el cambio, la mejora y la adición de una nueva lógica empresarial. Los microservicios modernos crean una capa de protección y abstracción por encima del sistema heredado. Al mismo tiempo brindan a los usuarios la flexibilidad y agilidad sin cambiar la aplicación heredada.

Regla número 3: Utiliza la herramienta adecuada para el trabajo

Muchas organizaciones están comenzando a adoptar el enfoque de "la mejor herramienta para el trabajo" al desarrollar una colección integrada de productos y tecnología compatibles. Esto puede incluir una lista de "debe usarse" cuando hay una herramienta principal que debes usar para solventar una necesidad específica. También hay una "lista de disponibles" que es una lista de soluciones alternativas aprobadas. Los programadores pueden elegir las herramientas y productos que mejor se adapten a sus necesidades – base de datos SQL, sistema de archivos secuenciales o sistema de datos en memoria. Esto se conoce como persistencia políglota (Figura 4).

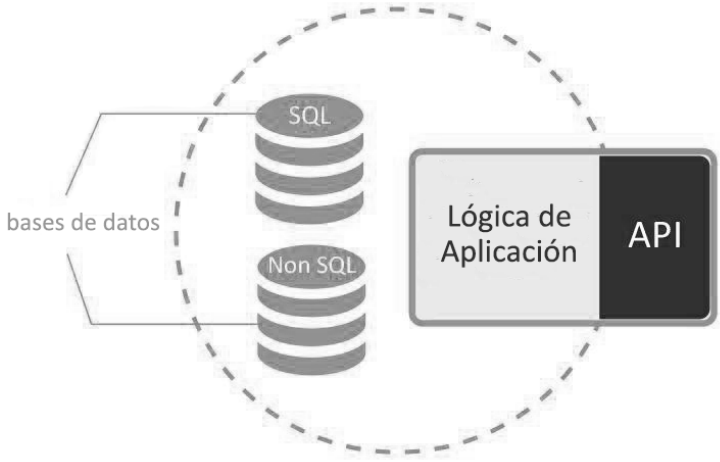


Figura 4. La persistencia políglota le permite a los desarrolladores encontrar la herramienta adecuada para el trabajo.

Tomar la decisión correcta al elegir un lenguaje de programación es tan importante como elegir la opción de gestión de datos adecuada. Elegir el mejor lenguaje para completar la tarea se conoce como programación políglota. El hecho de que una organización se haya comprometido con un lenguaje de programación específico no significa que los programadores deban realizar todos los procedimientos en dicho lenguaje, esto mientras invoquen a los microservicios de forma apropiada utilizando sus contratos. Por ejemplo, aunque Java sea el estándar de la empresa para algunas aplicaciones, C o javascript pueden ser la mejor opción (Figura 5).

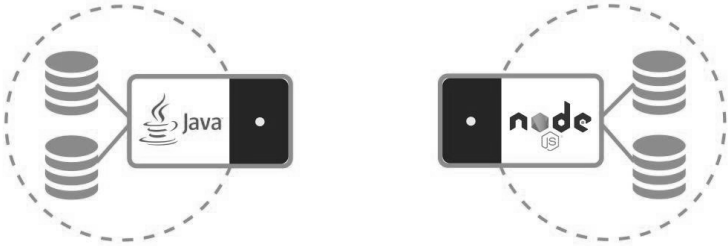


Figura 5. La Programación políglota significa que los programadores pueden elegir el lenguaje de programación adecuado para el trabajo.

El back-end políglota completa la trifecta de las arquitecturas políglotas. Esto le permite a los microservicios acceder flexiblemente a uno o más back-ends en el microservicio dependiendo de las necesidades de la aplicación que se está desarrollando (como se describió en la regla número uno). El back-end políglota no toma la forma de una capa de integración, pero es en su lugar una implementación de microservicio pura sin washing de microservicios – la práctica de vender un producto como microservicio cuando en realidad este no cumple con las características de microservicio (Figura 6).

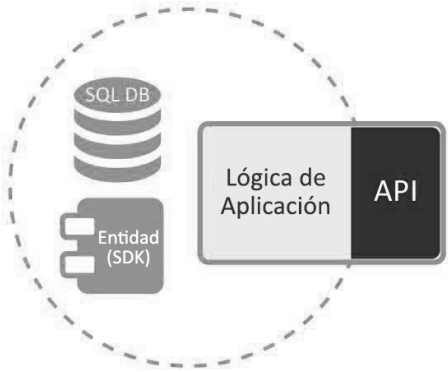


Figura 6. La capacidad de back-end políglota es una característica poderosa ya que el back-end puede ser una variedad de fuentes como un mainframe, un sistema de rango medio o una base de datos relacional en una plataforma distribuida. El soporte para diversos back-ends hace posible crear microservicios con habilidades significativas para integrar datos y procesos de fuentes previamente dispares.

Regla 4: Asegura todos los niveles

Dado que los microservicios son unidades ejecutables separadas, estos no disfrutan de algunos de los beneficios de un marco de seguridad que son compartidos por todos los componentes en una aplicación monolítica. Dado que a menudo no existe un mecanismo de seguridad compartida, los desarrolladores compensan esta falta ejecutando el microservicio detrás de una puerta de enlace API[1], la cual proporciona una gran funcionalidad mientras desempeña el papel de un firewall.

Cuando se implementa una puerta de enlace API, muchas organizaciones asumen que cualquier cosa detrás de dicha puerta de enlace está segura. Una vez que la amenaza cibernética penetra la puerta de enlace no existen mecanismos de seguridad adicionales que la desafíen. Los microservicios requieren una defensa a fondo que no esté limitada a una capa de seguridad. Las comunicaciones entre microservicios deben estar protegidas utilizando SSL[2]. Para la identidad de usuario y el control de acceso debe usarse OAuth[3]. Es importante manejar JSON[4] adecuadamente, protocolo que reemplazó al XML pero que tiene capacidades de escritura de datos débiles. JSON tiene funcionalidades limitadas para ayudar con la validación de datos, lo que significa que tiene vulnerabilidades que deben abordarse a través de la lógica en el microservicio (Figura 7).

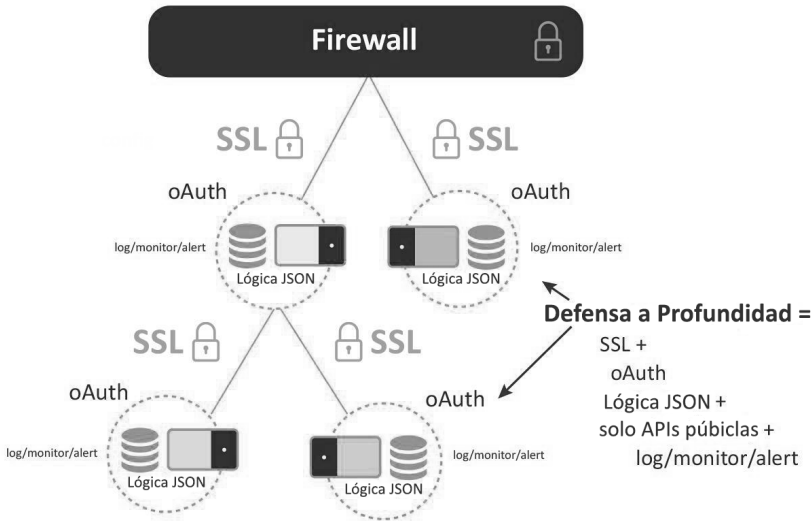


Figura 7. Debido a su naturaleza única, los microservicios requieren múltiples niveles de seguridad.

Otra estrategia de seguridad para todos los niveles es nunca permitir que los microservicios se ejecuten en la red pública porque normalmente esta no es segura. Adicionalmente, los programadores deben adoptar ciertas guías como registrar cada evento significativo, implementar monitoreo automatizado y generar alertas cuando sea necesario. Los programadores no deben reinventar la rueda en términos de seguridad, desarrollo y soluciones de gestión de sistemas. En su lugar, deben utilizar herramientas confiables y frameworks con los que se encuentren familiarizados. Como por ejemplo:

- Apache Log4j - Un framework de registro rápido, confiable y flexible, escrito en Java.
- oAuth - Protocolo abierto para permitir autorización segura para REST APIs, aplicaciones web, móviles y de escritorio.
- EhCache – Sistema de cache distribuido en Java ampliamente utilizado y open source.
- Angular - Plataforma de desarrollo open source para aplicaciones web.
- Freemarker - Motor de plantillas open source en Java. Las plantillas tienen formato estructurado, creadas por Freemarker, en donde los programadores ingresan datos cuando generan entidades.

Además de las otras consideraciones de seguridad discutidas – firewall, SSL, JSON, uso de una red pública, registro, monitoreo y alertas – algunos proveedores (como OpenLegacy) tienen una característica llamada seguridad en servicio. Esta agrega otra capa de seguridad a cada microservicio en una aplicación al basarse en la autenticación LDAP y oAuth. Dicha capa proporciona seguridad de estructura de datos al restringir el acceso a API fields específicos de acuerdo al nivel de seguridad. También agrega seguridad de contenido de datos al restringir el acceso a valores de datos de acuerdo al nivel de seguridad.

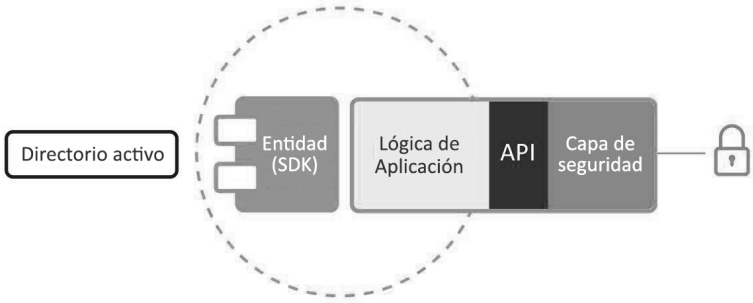


Figura 8. La seguridad debe implementarse al nivel del servicio.

Regla 5: Sé un buen ciudadano pero ten una buena vigilancia

Ser un buen ciudadano del ecosistema de microservicios significa que cuando escribes un nuevo microservicio que utiliza el contrato de otro microservicio, debes estar consciente del uso actual de dicho microservicio. Debes acercarte al equipo que soporta ese microservicio para hacerles saber tus planes y necesidades. Si

planeas utilizarlo extensamente podrías impactar su SLA[5] y ellos podrían tener que tomar medidas que pueden incluir aumentar el tamaño del pool de datos o habilitar el almacenamiento en caché[6] (Figura 9).

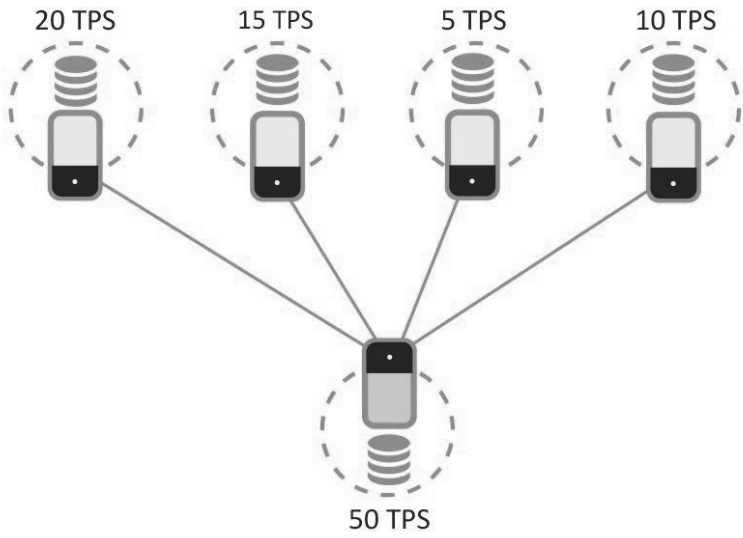


Figura 9. Ser un buen ciudadano significa que los desarrolladores deben trabajar juntos cuando sus microservicios se utilizan de manera conjunta como parte de una nueva funcionalidad para la empresa.

Debes ser un buen ciudadano pero también necesitas muy buenas herramientas de vigilancia. Debes medir los SLAs, recopilar registros y seguimientos y aplicar el acelerador a cargas de trabajo excesivas. También es importante recopilar tanto las métricas internas (‘¿qué servicios estuvieron involucrados y cuál es su

tiempo de respuesta?') como las métricas de experiencia de usuario ('¿cuánto tiempo hay entre el clic y los datos?').

Cuando tu microservicio interactúa con aplicaciones y datos mainframe, debes tomar acciones para proteger el monolito. Cada microservicio, no solo mantiene los registros y brinda la habilidad de seguimiento, sino que también existe un mecanismo de almacenamiento en caché que mejora la performance de las búsquedas al mantener en la memoria los datos utilizados con mayor frecuencia. Cuando se envían muchas solicitudes a un host, algún software puede limitar el acceso al monolito. El throttling puede limitar el número de solicitudes que un cliente puede enviar a las APIs de la aplicación por unidad de tiempo (Figura 10).

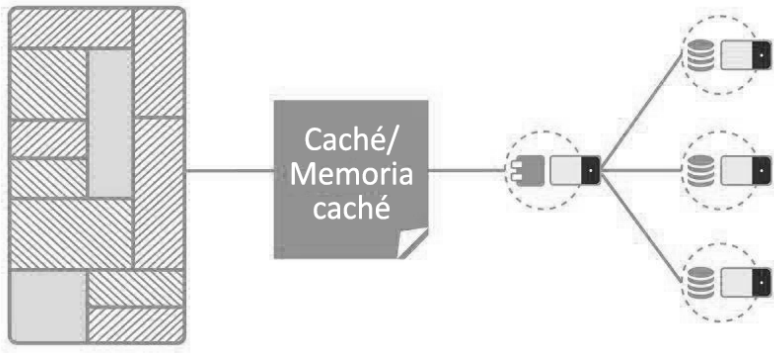


Figure 10. Tanto el almacenamiento en caché como el throttling son herramientas utilizadas para minimizar el impacto del microservicio en el monolito.

Regla 6: No se trata únicamente de tecnología

Antes de desarrollar microservicios, considera la organización de tu equipo. Los microservicios prosperan cuando te organizas como un equipo multidisciplinar[8] en donde cada equipo tiene diferentes habilidades y es lo más autosuficiente posible. Martin Fowler dice que "equipos funcionales en silo llevan a arquitecturas de aplicaciones en silo" (Figura 11). Los equipos multidisciplinarios trabajan mejor con APIs de microservicios porque estos están organizados alrededor de las capacidades que crean y gestionan (Figura 12).

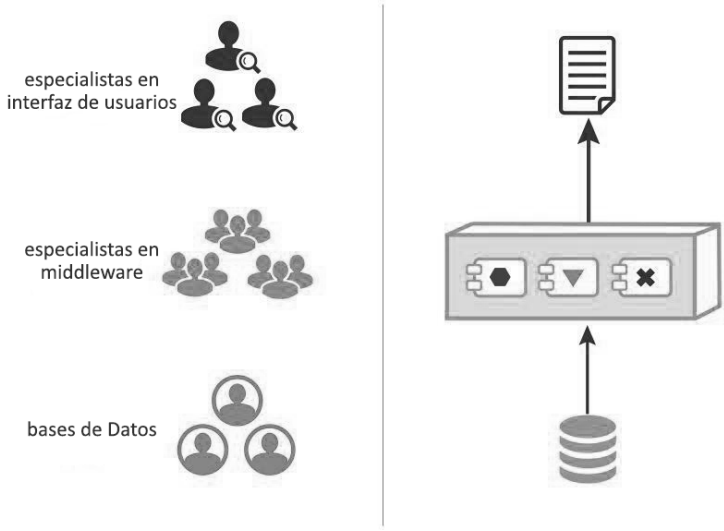


Figura 11. Equipos funcionales en silo llevan a arquitecturas de aplicaciones en silo.

Antes de desarrollar APIs de microservicios, considera el conjunto de habilidades de tu equipo. Java es una habilidad importante a tener en tu equipo, pero también se necesitan otras habilidades de lenguajes y middleware. Por ejemplo, algunas herramientas generan microservicios en Java de manera automática y las habilidades en Java se necesitan únicamente cuando quieres cambiar la salida estándar de una API de microservicio. Sorprendentemente, debido a que la aplicación heredada y el acceso a datos se manejan de forma automática, el equipo de implementación de microservicios no tiene que tener habilidades en sistemas heredados. Esto significa que el número de equipos multidisciplinarios no está limitado al número de personas con habilidades en sistemas heredados y por ende el mercado laboral es más amplio.

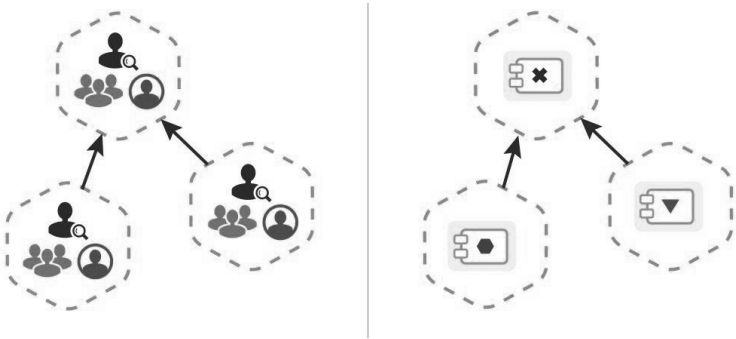


Figura 12. Es importante organizar a los equipos alrededor de las funcionalidades empresariales que desarrollan y despliegan como microservicios para que el desarrollo sea rápido y adaptable.

Los enfoques ágiles[9] para el desarrollo se ajustan de forma natural a los microservicios. El equipo multidisciplinario puede rápidamente crear las APIs públicas que resultan de sus microservicios. DevOps[10] también se ajusta bien a los microservicios puesto que tanto los desarrolladores como los técnicos pertenecen al mismo equipo multidisciplinario. El estar en un equipo también rompe las barreras organizacionales que impiden el uso de herramientas y procedimientos comunes.

Regla 7: Automatiza todo

La automatización es una forma comprobada de mejorar la calidad y reducir el riesgo en TI. Gartner[11] escribe que la automatización es la próxima frontera de TI. Un buen candidato para la automatización es el testeo debido a que, durante el desarrollo de los microservicios, repetir las pruebas de manera manual es costoso y consume mucho tiempo. El testeo de software automatizado puede reducir el tiempo de ejecución de pruebas repetitivas de días a horas, brindando resultados más completos y consistentes.

La automatización conduce a un entorno de TI mucho más robusto así como también a una mayor oportunidad de cambiar software a bajo riesgo. Algunos proveedores de microservicios están inherentemente más automatizados que otros. Por ejemplo, algunos proveedores hacen posible que las unidades desplegables – los propios microservicios – sean entidades estándar de Java. No hay

nada propietario en ellas – son exactamente lo que un programador Java experimentado crearía.

Adicionalmente, con este tipo de microservicios, soluciones como Jenkins y Maven están disponibles para que las uses. Con Jenkins, puedes automatizar muchas partes del proceso de desarrollo de software al hacer uso de pasos de integración continua y entrega continua. Con Maven puedes administrar la construcción, los informes y la documentación de un proyecto. Jenkins y Maven son solo ejemplos. Cuando elijas un proveedor o herramienta de microservicios, busca una solución en donde no se necesiten soluciones de despliegue propietario, testeo o control de versiones y se recomienda que uses cualquier solución open source de mejores prácticas que consideres mejor.

Tabla 1. Muestra de un Stack Tecnológico de Herramientas que se usan para crear, administrar y desplegar microservicios (utilizando OpenLegacy como ejemplo).

Herramienta	**Descripción**
Spring	Spring es un framework de aplicación utilizado para crear sistemas y aplicaciones, basadas en JVM, que sean simples, portables, rápidas y flexibles.

SonarQube	SonarQube, una herramienta de calidad de código continua, brinda la capacidad de mostrar el estado de una aplicación e indicar los problemas que pueden haberse introducido recientemente.
Jenkins	Jenkins es un servidor open source de automatización que ofrece cientos de plugins para apoyar la creación, despliegue y automatización de cualquier proyecto.
Maven	Apache Maven es un software de gestión de proyecto y una herramienta de comprensión que se puede utilizar para crear y gestionar cualquier proyecto basado en Java.
Git	Git es un sistema open source de control de versiones distribuido, diseñado para manejar desde proyectos pequeños a proyectos muy grandes con rapidez y eficiencia.
Sonatype Nexus	Sonatype Nexus es una herramienta para organizar, almacenar y distribuir componentes de software.

Cloud Foundry	Cloud Foundry es una plataforma de aplicaciones en la nube, open source, para desarrollar y desplegar aplicaciones empresariales en la nube. Automatiza, escala y administra las aplicaciones en la nube a lo largo de su ciclo de vida.

Es importante utilizar un stack tecnológico estándar porque las tecnologías están probadas. También existe una buena cantidad de experiencia y documentación disponible sobre cómo configurarlos apropiadamente para asegurar la seguridad y una alta performance. Algunas situaciones de desarrollo requieren de una base de datos más rápida o más robusta. O podría haber requerimientos contractuales o regulatorios para algunos tipos de hardware, sistemas operativos y software de servidor. Estos son casos poco comunes y la mayoría de microservicios funcionan bien en un stack tecnológico estándar.

Las mejores prácticas en este capítulo son de sentido común, pero algunas requerirán cambios culturales y tecnológicos en la organización de TI. “Accede únicamente a APIs públicas” y “asegura todos los niveles” se enfocan en medidas prudentes para proteger los recursos de datos. “Utiliza la herramienta adecuada para el trabajo” y “no se trata únicamente de tecnología” posiblemente requieran de cambios organizacionales para tener un

impacto. “Sé un buen ciudadano pero ten una buena vigilancia” y “automatiza todo” se enfocan en una mezcla de medidas organizacionales y tecnológicas que son fáciles de implementar siguiendo los procedimientos de TI.

[1] *API gateway – API management, Wikipedia*

[2] *SSL – What is SSL?, SSL.com*

[3] *oAuth – oAuth, TechTarget.com*

[4] *JSON – JSON, JSON.org*

[5] *SLAs – Service-Level Agreement, Wikipedia*

[6] *Caching _ Cache (Computing, Wikipedia*

[7] *Throttle – Throttling Process (Computing), Wikipedia*

[8] *Cross functional team – Want to Develop Great Microservices? Reorganize Your Team, TechBeacon*

[9] *Agile approaches – Agile In a Nutshell*

[10] *DevOps – The AgileAdmin, What is DevOps*

[11] *Gartner – Automation: The Next Frontier for IT, May 2016*

Capítulo Seis:

Costo/Beneficio de los Microservicios

Nos estamos enfocando en cómo los microservicios pueden permitir que las aplicaciones heredadas sean activos – y no baches – en tu viaje digital. Los microservicios como tales pueden beneficiar tanto a organizaciones TI como a organizaciones empresariales.

A pesar de los beneficios de una estrategia de microservicios, es importante ser cauto al querer implementar una, puesto que la adopción de una arquitectura de microservicios es similar a la adopción de cualquier disciplina de software que sea relativamente nueva. Si estás creando y desplegando microservicios, necesitarás el ambiente y staff apropiados. La siguiente no es una lista extensa, pero estas son algunas cosas a tomar en cuenta cuando se esté pensando en microservicios, similar a cualquier tecnología o metodología nueva.

Desarrollo de la Arquitectura de Microservicios

Determinar el costo y valor de los proyectos de microservicios no es tan diferente a otros proyectos, pero existen factores únicos a tomar en cuenta. Por ejemplo, aquí hay algunos de los gastos en los que se puede incurrir al empezar:

1. **Costos de Personal:** No todos los desarrolladores estarán familiarizados con la arquitectura de microservicios.
2. **Gastos Organizacionales:** La arquitectura de microservicios funciona mejor cuando es administrada por pequeños equipos multidisciplinarios.
3. **Herramientas:** Contenerización y otras tecnologías de apoyo.

Como regla general, dependiendo de dónde estás comenzando, los costos iniciales pueden ser preocupantes desde el punto de vista presupuestario pero los beneficios posteriores serán significativos. La adopción de una arquitectura de microservicios pagará rápidamente esos costos al devolver un gran valor comercial y técnico.

Mantenibilidad & Costos Operativos

La primera parte de generación de valor viene en la forma de ventajas de mantenimiento. Supongamos que empiezas ejecutando monolitos de aplicaciones en lugar de un desarrollo Greenfield. El mantenimiento de estas aplicaciones toma tiempo ya que están construidas a partir de dependencias entrelazadas.

Por ejemplo, imagina que hay una interrupción en una aplicación monolítica – el administrador de inicio de sesión falla. Todas las otras partes de la aplicación dependen del administrador de inicio de sesión, así que cuando este está caído todo está caído. Con una

aplicación que se comporta de esta manera es difícil soportar un número creciente de clientes, y aunque existen soluciones alternativas, como por ejemplo failover (conmutación por error) y la creación de instancias, estas suelen ser costosas.

> Por sí misma, la reducción en gastos de mantenimiento debería ser suficiente para pagar, en unos pocos años, los costos iniciales de los microservicios.

El tiempo que toma probar, actualizar y mantener los monolitos de aplicaciones significa que el mantenimiento se ha convertido en una gran parte del presupuesto tradicional de TI. Una muestra del presupuesto de TI en empresas de salud indica que el 70% de los gastos[1] corresponden al manejo de la empresa – llegando a un 73% en el 2017. Esto deja poco espacio para la innovación.

Regla general – Una arquitectura de microservicios con menos dependencias de aplicaciones y APIs simples reducirá de inmediato el tiempo y el dinero que se gasta en el mantenimiento de aplicaciones. Se ha demostrado que el ahorro en gastos de mantenimiento de aplicaciones es más que suficiente para solventar en pocos años los gastos iniciales.

El matrimonio entre la Calidad y la Velocidad

Las dependencias (reductores de velocidad) inherentes a cualquier aplicación monolítica inhibirán la innovación. Los monolitos de aplicaciones no suelen llevarse muy bien con nuevas técnicas de desarrollo – como Agile y DevOps – que hacen énfasis en la velocidad. Cada actualización que se haga en una parte de la aplicación será reflejada en las otras partes, así que cualquier actualización deberá ser testeada a fondo.

Para mitigar este problema existen herramientas de testeo automáticas, pero al igual que las soluciones para mitigar las fallas en monolitos, estas son caras y difíciles de escalar.

Por otro lado, los microservicios permiten a los desarrolladores aumentar la velocidad de desarrollo sin tener que sacrificar la calidad. Esto resulta en una ventaja competitiva – estos podrán refinar su aplicación más rápido que aquellos que no hayan adoptado una estrategia de microservicios. Los clientes y proveedores externos se volverán leales a estas aplicaciones mientras que los usuarios internos finales serán más productivos.

CALIDAD

Así es como funciona: DevOps, Agile y otras prácticas de desarrollo modernas dependen en gran medida del testeo automático. La idea es brindarle a los desarrolladores y al personal de QA la habilidad de configurar varios entornos de pruebas con

tan solo unos cuantos clics, y después dejar que un programa de testeo automatizado (por ejemplo Jenkins) haga la mayor parte del trabajo. Si se hace correctamente, los microservicios deberían requerir cero cambios en tus aplicaciones heredadas, limitando así la necesidad del testeo de la aplicación monolítica que consume mucho tiempo y es costoso.

Los microservicios hacen que el proceso de testeo sea mucho más limpio. Están construidos de forma más simple por lo que es más fácil revisar su código. Como resultado, es también más simple realizar pruebas unitarias. Por definición los microservicios son pequeños, simples, rápidos y fáciles de escribir, por lo tanto son igualmente fáciles de probar.

VELOCIDAD

El valor de la velocidad es diferente para cada organización pero es fácil apreciar el beneficio de un aumento del 90% de servicios entregados por año, o el poder sacar 20 nuevos servicios cada cinco semanas.

Regla General: Velocidad de Desarrollo + Calidad de Desarrollo = Ventaja Competitiva. Por ejemplo:

- Cuando una organización aseguradora aprovecha los microservicios para competir con los motores de comparación de precios online, entonces es parte de un canal digital de rápido

crecimiento que es usado por millones de compradores.

- Un banco que pueda ofrecer pagos de cuentas y depósitos desde un dispositivo móvil, como resultado de los microservicios, es capaz de capturar a las generaciones más jóvenes de clientes bancarios, que pueden ofrecer valor de por vida y agregar millones en depósitos.

Camina, después corre

Encuentra un socio que trabaje contigo en un caso de uso empresarial que sea urgente y convincente, en donde la arquitectura de microservicios pueda traer valor inmediato. Define el criterio de éxito de una prueba de concepto de bajo riesgo que te dé la habilidad de visualizar "lo que es posible" y los puntos de datos para evaluar con confianza las referencias de potencial y costo/valor necesarias para que inicies tu viaje digital.

[1]*Gartner IT Budget: Enterprise Comparison Tool, March 2017*

Capítulo Siete:

Consejos para iniciar con Microservicios

Lo primero que se debe considerar es evitar caer en el escenario de "balas de plata" – no existe una respuesta o un enfoque para todo. Con cualquier estrategia de microservicios existen innumerables opciones y alternativas. Primero debes evaluar los problemas de la empresa que buscas resolver, a dónde ves que tu mercado se dirige, cómo tus clientes quieren interactuar contigo, cuáles podrían ser tus problemas operacionales, qué impactos generan los problemas de soporte, etc.

Basándose en estos pasos, podrías darte cuenta que el mejor enfoque hacia la Transformación Digital puede ser evolutivo o revolucionario – es probable que cualquiera de las dos vías requiera de una combinación de recursos y desarrollo interno, tecnología open source, proveedores de servicios y tecnología.

No se recomienda iniciar un proyecto de microservicios – incluso cualquier proyecto de desarrollo de software – sin antes tener un plan detallado de cómo debería lograrse. Sin embargo, puede ser que los microservicios se alejen más de las normas de desarrollo que otras formas de arquitectura de software. Esto es porque el desarrollo de microservicios no solo significa aprender nuevos

frameworks de programación – significa fundamentalmente reorganizar las unidades funcionales de desarrollo dentro de una organización.

Vamos a cubrir dos áreas de enfoque principales de microservicios: La primera es fundamental – cómo sentar las bases de trabajo para los microservicios dentro de una organización. La segunda es de proceso – las mejores formas que hemos encontrado para que las empresas creen, construyan e implementen microservicios específicos desde cero. Con estas bases a mano, las empresas tienen más probabilidades de pasar del papel a la realidad su enfoque de microservicios.

Antes de comenzar: Precondiciones para los Microservicios

Las investigaciones muestran que muchas empresas fallan en su intento de reemplazar las arraigadas arquitecturas heredadas por infraestructura digital – usualmente después de haber gastado grandes cantidades de dinero durante largos períodos de tiempo. Por otro lado, se supone que los microservicios son exactamente lo contrario – rápidos, baratos y exitosos. Sin embargo, las empresas deben cumplir ciertas precondiciones para garantizarse el éxito:

Primera Precondición: Habilitación Tecnológica

Un desafío clave para las organizaciones es el imperativo de tomar la salida de la infraestructura de software heredada construida en la década de los 80s y traducirla en una entrada que el último modelo

de teléfono móvil pueda entender. Los microservicios pueden ayudar a facilitar este proceso, pero una de las claves de este enigma es saber que el hacer que el backend heredado esté más disponible, para los usuarios móviles y basados en navegadores, aumentará la carga de trabajo en una infraestructura ya sensible.

Hay otra cuestión que toma en cuenta el futuro. Los microservicios – y las tecnologías que los soportan – no son estáticos. El crear una infraestructura de microservicios significa anticipar tendencias en tecnologías de la información evitando a la vez la ampliación de la misión. En otras palabras, los desarrolladores deben crear una arquitectura de microservicios que sea actualizable pero que también esquive la inevitabilidad de una pila de middleware compleja.

La idea de que todos los microservicios deben estar diseñados con una serie de estándares comunes va de la mano con la idea de que todos los microservicios deben ser “a prueba del futuro”.

Segunda Precondición: Enfoque basado en estándares

La idea de que todos los microservicios deben estar diseñados con una serie de estándares comunes va de la mano con la idea de que todos los microservicios deben ser “a prueba del futuro”. Muchos desarrolladores externos (que comúnmente se utilizan para crear arquitecturas de microservicios) no valoran este enfoque, y como

resultado, crean frameworks ciegos a las realidades de la programación para infraestructuras heredadas. Cuando no se aplican estándares a un microservicio, será difícil, sino imposible, reutilizarlos y mantenerlos en aplicaciones adicionales.

Tercera Precondición: Preparándose para la velocidad
Para un equipo de desarrollo es fácil crear un microservicio de manera rápida, pero el que sea fácil depende en su mayoría del equipo. Un equipo capaz de crear microservicios de manera rápida necesita estar equipado con la tecnología adecuada y una serie de estándares apropiados. Pero eso no es todo.

La Ley de Conway dice que las organizaciones crean software que ayuda a la estructura de su organización. Los microservicios son discretos y desplegables de manera independiente, entonces una organización que crea microservicios debe estar conformada por pequeños equipos que puedan trabajar en proyectos independientes con una organización liberal y sin un control jerárquico estricto. Por lo tanto, una arquitectura de microservicios es complemento de un framework Agile o DevOps.

Agile, DevOps y Microservicios.
Al año 2016, aproximadamente el 66% de las compañías usaban Agile, pero su uso está lejos de ser integral a un nivel industrial. Las organizaciones que regularmente lidian con el hardware y software heredado tendrán a menudo dificultades para adaptar el

rápido ritmo de Agile a la lenta realidad del desarrollo de sistemas heredados.

- J.P. Morgan Chase – el décimo banco más grande del mundo – inició la adopción de Agile apenas en el 2015.
- Muchas organizaciones de la salud aún consideran a Agile como un equivalente de Waterfall.
- Grandes organizaciones gubernamentales aún luchan con el concepto de adoptar métodos Agile a hardware heredado.

Muchas de las organizaciones que utilizan Agile en la teoría podrían no estar utilizándola en la práctica. Agile se puede definir como la habilidad de una organización para lograr un alto ritmo de cambio. Si una organización aún despliega lanzamientos una vez cada cuatro meses entonces no es Agile.

Utilizar microservicios puede ayudar a las organizaciones a lograr ser completamente Agile. Los microservicios están diseñados para ser pequeños y flexibles, es por ello que cuando equipos Agile comienzan a trabajar en proyectos de microservicios su velocidad aumenta. Esto puede ser suficiente para llevar a una organización de un ritmo de lanzamiento mensual a un ritmo de lanzamiento semanal – en otras palabras, transformar a una organización en una que logre ser Agile en la práctica y no solo en palabras.

Los proveedores de software de Microservicios/API que le permiten a una organización lograr escalabilidad y velocidad con

aplicaciones heredadas pueden considerarse habilitadores de DevOps. Por ejemplo, OpenLegacy:

1) Genera automáticamente APIs/microservicios desde aplicaciones heredadas en una fracción de tiempo y con estándares abiertos.
2) Genera objetos de Java estándar que pueden ser desplegados como servicios digitales para dispositivos móviles, la Web o la nube.
3) Es compatible con todas las herramientas DevOps estándar y se incorpora fácilmente al ecosistema DevOps.
4) Se ajusta al diseño basado en testeo al generar unidades de pruebas automáticas.
5) Permite el despliegue al generar código que puede ser desplegado como cualquier otro – nada propietario.
6) La consola de gestión puede usarse para controlar tanto APIs como microservicios y escalar hacia arriba o hacia abajo, cosa que se ajusta bien con los procesos DevOps.

Desarrollo de microservicios una vez cumplidas las precondiciones

Una vez que una organización tiene los prerrequisitos tecnológicos y estructurales que pueden soportar un ciclo de desarrollo mucho más rápido, es hora de comenzar a realmente crear microservicios. Este es un proceso de tres fases – descubrir una necesidad de un microservicio, crear el microservicio y ponerlo en producción. Por ejemplo, en OpenLegacy, esto se puede resumir en Descubrir,

Construir, y Realizar (Figura 1).

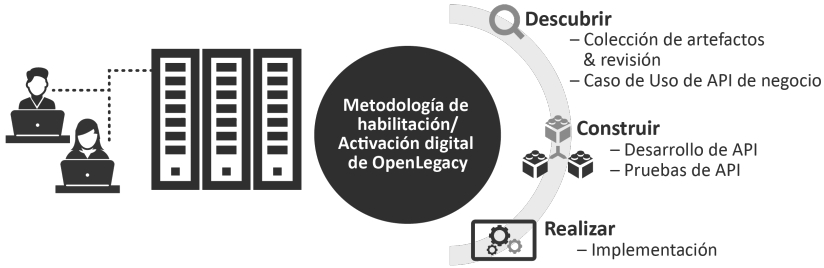


Figura 1. Una vez que se cumplen las precondiciones, una manera de comenzar incluye el enfoque de Descubrir, Construir y Realizar.

Primera etapa: Descubrir

"Descubrir" en este caso significa documentación. El trabajo no inicia hasta que los desarrolladores puedan articular, en un documento de texto, la lógica que sostiene su microservicio, describiendo para qué es, cómo funciona y si existe algún vacío.

Como ejemplo, imagina un microservicio que consulta los registros de un hospital cuando los pacientes y médicos desean consultar los resultados de sus pruebas. El documento de descubrimiento puede incluir:

- Usuarios: En este caso, los pacientes que deseen consultar sus registros y los médicos que deseen hacer lo mismo.
- Descripción: Esta aplicación le permite a los usuarios consultar sus registros hospitalarios. Para ello, esta debe

unir un cierto número de sistemas, como un portal web, una biblioteca de registros, y/o un mainframe.

- Secuencia: El usuario se autentica a través del portal web, navega hasta la página de registros y selecciona un registro, esto hace funcionar el microservicio y brinda una respuesta al usuario.
- Precondición: El usuario se debe autenticar como un paciente o un médico, debe estar autorizado para ver los registros y debe tener registros para consultar.
- Postcondición: La aplicación provee un registro médico que es descargado en PDF. Si el microservicio no puede hacerlo, este muestra un error para informar al usuario.

El documento de descubrimiento es el eje central del microservicio. Muchos de los errores, inconsistencias y deficiencias de las aplicaciones se pueden rastrear hasta sus orígenes en el papel, independientemente de su código. Los puntos anteriores representan el bosquejo más breve de un documento de descubrimiento real – la versión real debe ser revisada exhaustivamente en búsqueda de vacíos.

Segunda Etapa: Construir

La fase de "Construir" dentro del desarrollo de un microservicio comprende tanto la escritura como el testeo del código. Por su naturaleza los microservicios son rápidos y fáciles de crear, utilizando a menudo herramientas de testeo automáticas.

Generalmente, testear un microservicio es más complicado que escribirlo. Una vez que el microservicio se ha escrito, son necesarias al menos tres fases de testeo:

1. **Desarrollo y Pruebas de Tiempo de Desarrollo**
   Las pruebas de desarrollo las realiza un desarrollador durante e inmediatamente después de que se ha escrito el microservicio. En el caso de los microservicios, estas toman generalmente la forma de testeos unitarios. Por otro lado, las Pruebas de Tiempo de Desarrollo, tomarán la forma de análisis estadístico, con distintas herramientas de testeo que apuntan a diferentes partes del microservicio.
2. **Testeo QA**
   El testeo QA de microservicios involucra el testeo positivo/negativo de varias áreas. Por ejemplo, en funcionalidad – ¿trabaja la aplicación como se espera? En seguridad y autorización – ¿quién puede usar el microservicio y cómo reacciona este a usuarios no autorizados? Finalmente, el testeo se realiza en la lógica de negocio de la aplicación para medir el rendimiento de la aplicación bajo un cierto número de escenarios.
3. **Testeo PTE**
   El Entorno de Pruebas Público (PTE por sus siglas en inglés) simula el rendimiento de un microservicio en un entorno de producción. Dentro de las preguntas esenciales están; cómo la aplicación se comporta ante tráfico normal,

si esta se comporta bien ante cargas de tráfico inusual y cómo su rendimiento se traduce a un SLA.

Tercera Etapa: Realizar

Una vez que el descubrimiento y la creación se han completado, todo lo que resta es poner el microservicio en producción. Esto es fácil de hacer – posiblemente el paso más fácil dentro del bucle descubrir-crear-realizar. Por su naturaleza, los microservicios están diseñados para ser fáciles de desplegar.

Dado que los microservicios son mini-aplicaciones, pueden ser diseñados y puestos en producción sin una extensa colaboración entre equipos. El poner a un microservicio en producción no genera el riesgo de traerse abajo cualquier otra parte de la aplicación.

Es por ello que la realización de un microservicio va más allá que la simple producción. Más bien, pide a los desarrolladores que reflexionen sobre las lecciones aprendidas de la construcción y el testeo de la aplicación. El incorporar estas lecciones es un aspecto de mejora continua. Crear tu primer microservicio puede haber sido un proceso inesperadamente fácil. El incorporar las lecciones aprendidas en su construcción significa que tu segundo microservicio será aún más rápido y simple.

Ponlo todo junto: Comienza pequeño & Piensa en grande

Para los evangelistas de microservicios dentro de grandes organizaciones el desafío no es sentar las bases, comprometerse con un cambio institucional o programar nuevos servicios. Más bien, el desafío es obtener la aceptación de los directores para poder iniciar esos cambios.

> Por su naturaleza, los microservicios están diseñados para ser desplegados fácilmente.

Desde la perspectiva de un único individuo esta tarea podría parecer desalentadora, pero hay una serie de pasos concretos que incluso una sola persona puede tomar con el fin de generar un cambio.

Sin pruebas – pruebas de que el concepto del microservicio es viable – no hay forma de que los encargados de tomar decisiones se comprometan a un cambio institucional a gran escala. Por lo tanto, el trabajo del evangelista es crear pruebas, idealmente a través del compromiso con un exitoso proyecto de microservicios a pequeña escala. Afortunadamente, dentro de una organización grande con mucha infraestructura heredada, siempre hay oportunidad para que una persona o un equipo pequeño de personas creen un microservicio que sea viable.

Imagina una función específica – algo pequeño, conveniente y preferiblemente que no sea crítica – que pueda ser mejorada con un microservicio. Lleva dicha función a través del proceso de diseño, construcción y realización descrito anteriormente. Esto representará el caso de prueba para un enfoque de microservicios. Si funciona, ese logro servirá como un punto de apoyo: a partir de allí puedes escalar.

En la mayoría de organizaciones, crear una cultura que soporte los microservicios – y que después cree los microservicios – es un proceso iterativo. El mejor enfoque es empezar con un pequeño logro y crecer a partir de allí. Aunque el punto de inicio sea pequeño y el viaje largo, el resultado será una unidad de negocios más optimizada y enfocada que finalmente pueda solventar las demandas de una economía digital en crecimiento.

Una última cosa: Elegir un proveedor de microservicios

Una estrategia de monolito a microservicios implicará impactos tecnológicos, impactos de proceso e impactos en la cultura corporativa – es fácil afirmar que uno es Agile y flexible en lugar de realmente serlo. Entonces, elige un proveedor que se convierta en un socio, que tenga un historial comprobado de entendimiento sobre dónde te encuentras, lo que tienes y a dónde quieres ir.

Tecnología del proveedor

No todos los productos de microservicios son creados de igual forma, y algunos ni siquiera cumplen los estrictos requerimientos para ser microservicios. Esto es lo que Gartner llama "microservices washing" (lavado de microservicios) – el acto de vender un producto con microservicios superficiales que no cumple con la definición de microservicio.

Hasta hace poco, Las SOAs y ESBs eran las estrategias primarias de integración para aplicaciones de interacción directa con el cliente. A medida que esta arquitectura fue perdiendo impulso, muchos proveedores sintieron la necesidad de cambiarse a una nueva arquitectura como los microservicios.

En teoría, este cambio se debería ver como una estrategia de microservicios idealizada – una capa de microservicios que expone las características a clientes y socios, una capa que conecta estas características con unidades de negocios, y una tercer capa que conecta unidades de negocios con aplicaciones heredadas (Figura 2).

Sin embargo, en la práctica, estas compañías nunca encontraron realmente la forma de deshacerse de la arquitectura SOA que subyace a sus productos. El resultado es usualmente algo que se ve como un ESB encapsulado por un envoltorio de microservicios.

Por ejemplo, imagina un microservicio que expone información del cliente como una API. La API necesita transmitir todos los datos del cliente pero solo a algunos agentes, debido a reglas de conformidad. Este microservicio contiene un ESB. Cuando este recibe una solicitud de datos del cliente, el ESB se pone en contacto con el mainframe, pero no directamente. Este debe primero contactar a otro microservicio que verificará si el agente solicitante está o no autorizado para recibir dichos datos.

Esto viola los buenos principios de diseño arquitectónico de microservicios. Este genera una multicapa de dependencia debido a su integración, creando en esencia un monolito. Si una parte se cae, todo el servicio se cae (Figura 3).

Las APIs se deben conectar a mainframes heredados de formas simples, inteligentes y modernas. Por ejemplo, OpenLegacy hace esto posible al crear conectores precompilados que se interconectan con aplicaciones heredadas con el fin de crear un Objeto en Java, cuya salida pueda ser leída por una API REST estándar, una página web basada en un navegador, o incluso un servicio web SOA (Figura 4).

El resultado es una herramienta que cualquier desarrollador puede usar sin necesidad de aprender o modificar el código fuente heredado (por ejemplo, COBOL). Los desarrolladores, en lugar de preocuparse por largos períodos de integración o por el código

heredado, pueden crear APIs e implementar nuevas funcionalidades robustas en solo unas horas – lo suficientemente rápido para satisfacer el creciente ritmo de la demanda de los clientes.

Figura 2. Según Gartner, las organizaciones que exageran la forma en que sus microservicios funcionan caen en el llamado "microservices washing" (lavado de microservicios). Este diagrama a menudo representa la forma en como ellos dicen que funcionan.

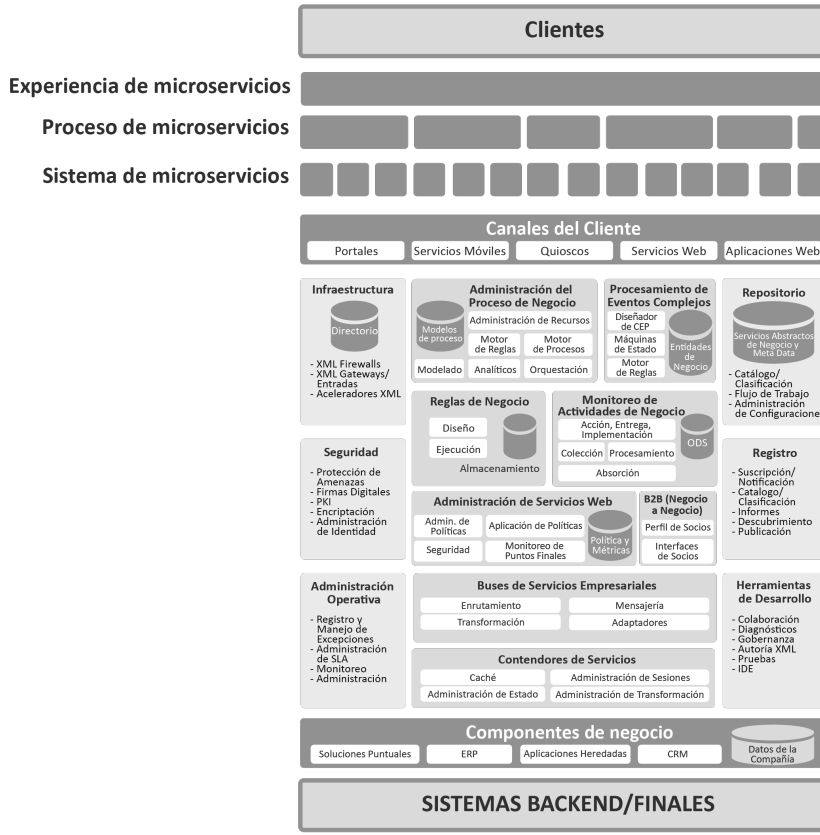


Figura 3: En un escenario de "microservices washing", la arquitectura de microservicios real es de hecho más complicada de lo que se describe. Este diagrama es a menudo como realmente funciona.

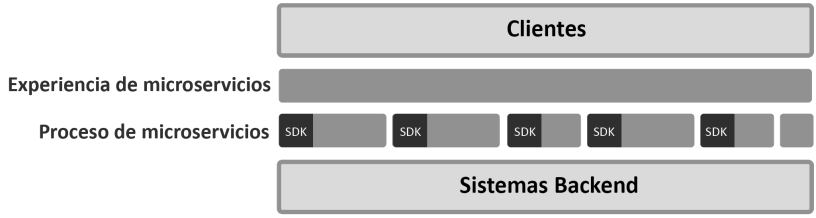


Figura 4: Por ejemplo, los "microservicios modernos" de Open Legacy eliminan capas y complejidad con el fin de acelerar la innovación con monolitos heredados. Así es como realmente funciona.

Lista de chequeo de proveedor & Consideraciones

Socio o Proveedor	Un socio debe comprender más de un aspecto de los desafíos de tu negocio. Un socio se preocupa por aquello que te preocupa. Tendrán problemas, pero un socio se mide por cómo caminan juntos y resuelven esos problemas. Un socio es transparente con sus clientes y responde rápidamente a sus preguntas. Un socio se hará responsable de los errores si los comete.
Modelo de Suscripción	Se trata de flexibilidad, minimizando el riesgo inicial y el capital de inversión mínimo. En un modelo de suscripción, los suscriptores están siempre actualizados o pueden acceder a las últimas versiones, están alineados con la adopción constante de computación y software en la nube como un servicio, y pueden manejar el timebox para acomodarse a los plazos del proyecto actual

	y a las necesidades de uso.
Modelo de Soporte	Busca un proceso y metodología de soporte definidos. Comprende los términos estándar y premium asociados al Tiempo de Respuesta, Tiempo de Mitigación y Tiempo de Resolución. Examina el proceso de solicitud de mejora, la comunidad de desarrolladores, centro de conocimiento, grupos de usuarios y la filosofía principal de contacto.
Incorporación	¿Cómo adoptas esta nueva tecnología? ¿Cuál es el plan de estudios de capacitación? y ¿puede este ser adaptado a tus necesidades? ¿Qué hay disponible para un proceso de autoaprendizaje, acceso y contenido dentro del Knowledge Center? ¿Tu socio proporciona servicios de Inicio Rápido y Tutorías en el Trabajo?
Estándares Abiertos	¿Por qué Estándares Abiertos? Otra pregunta que podrías hacerte es ¿es necesario re-inventar la rueda? Los Estándares Abiertos permiten una adopción más amplia con la comunidad, además de reducir el costo, que es la mayor barrera para la adopción. Los Estándares Abiertos eliminan las barreras innecesarias, reducen los supuestos de implementación y aumentan la adopción y la cooperación entre equipos. Los Estándares Abiertos invitan a la cooperación y a pensar fuera de la caja, todo ello impulsando la innovación.
Arquitectura Subyacente	¿Pone tu socio en práctica lo que dice – están usando herramientas estándar de la industria dentro de su arquitectura? ¿Son flexibles y

	adaptables a nuevas u otras tecnologías? ¿Te brindan acceso a las personas que pueden explicarte su arquitectura subyacente y por qué seleccionaron ciertos estándares? Lo más importante ¿Te brindan una clara y definitiva Hoja de Ruta de la Plataforma?
Enfoque central	Identifica un socio que entienda tu problema empresarial o técnico. No necesariamente todos tus problemas, pero asegúrate de que entienden al menos un punto de negocios difícil y que sean capaces de resolver, y muy bien, dicho punto. El socio ideal no pretenderá hacer más de lo que realmente hace pero es honesto y está dispuesto a hacer lo que hace muy bien.

[1]*Gartner Blog Network, What A Microservice is Not, January, 2017*

Capítulo Ocho:

Impacto Empresarial de los Microservicios & APIs

Los beneficios de los microservicios y las APIs se extienden más allá del departamento de TI y es importante mantener más involucrada dentro de la discusión al área de negocios de la organización.

El punto es que si necesitas innovar más rápido pero tus equipos de TI argumentan que el proyecto es “complejo, costoso y requiere mucho tiempo”, debido a los “sistemas heredados” de tu organización, entonces los microservicios pueden ser la respuesta.

La ironía de los sistemas heredados es que estos tienden a impactar a algunas de las compañías más antiguas y exitosas del mundo. Los monolitos heredados como IBM, UNISYS, HP, Digital, Siemens, Honeywell, Tandem, Stratus y otros, comenzaron a impactar positivamente las operaciones de negocios en los años 50. Almacenamiento de datos y métodos de acceso como VSAM, IDMS, IMS, DB2, Oracle, y ADABAS mostraron ser revolucionarios en la habilidad para almacenar y facilitar un acceso rápido a datos a través de aplicaciones críticas para la misión y el negocio, escritas en COBOL, Assembler, Fortran, ADS, RPG, y Natural.

Por lo tanto, las compañías más establecidas – las grandes marcas – son las que a menudo tienen las mayores dificultades para permanecer a la "vanguardia" dentro de sus industrias, a menos de que hayan encontrado maneras de acelerar la innovación dentro de sus monolitos heredados. Algunas organizaciones se han acostumbrado tanto al status quo, que ni siquiera creen que es posible reducir los retrasos y la velocidad de entrega de canales y aplicaciones digitales innovadoras.

Cuando tienes sistemas heredados y debes atender a clientes y prospectos que dependen en gran medida de servicios digitales ofrecidos a través de dispositivos móviles o la Web, necesitas encontrar una manera para cerrar esa brecha entre la tecnología antigua y las demandas modernas.

Cuando tienes sistemas heredados y debes atender a clientes y prospectos que dependen en gran medida de servicios digitales ofrecidos a través de dispositivos móviles o la Web, necesitas encontrar una manera para cerrar esa brecha entre la tecnología antigua y las demandas modernas.

No queremos parecer un anuncio promocional, pero podemos hablar con propiedad sobre los casos de uso empresariales en los

que personalmente hemos estado involucrados a lo largo de los años y como parte de nuestras funciones en OpenLegacy.

Uno de los aspectos más sorprendentes de nuestro trabajo es que incluso las organizaciones globales más grandes no han podido resolver su desafíos en sistemas heredados por medio de desarrollos internos, tecnología middleware o iniciativas SOA/ESB. Todas estas inversiones, aunque fueron excelentes decisiones en su momento, a menudo no son capaces de satisfacer las demandas de la "nueva economía digital" – aun cuando en sus esfuerzos han utilizado cientos de personas, millones de dólares y muchos años.

La parte empresarial de la organización podría no interesarse ni un poco en cómo la tecnología funciona: solo buscan ser competitivos, ágiles y crear nuevos o mejorados canales de ingresos. El área de TI se preocupa por reducir la complejidad de acceso y el aprovechamiento de sus sistemas. En alguna parte de la intersección de estos objetivos es donde los "microservicios modernos" entran a ser parte de la conversación.

A menudo, nuestro enfoque de microservicios reduce sus tiempos de creación de semanas a minutos u horas y el despliegue del proyecto de meses a semanas.

Aunque los microservicios pueden beneficiar a cualquier empresa

que utilice sistemas heredados, hemos destacado a continuación algunos ejemplos.

Un portal RR.HH. moderno y optimizado

La compañía de telecomunicaciones Bezeq había comenzado a notar la edad de su portal de recursos humanos, desarrollado en .NET. Usado por empleados para todo, desde solicitudes de vacaciones y reportes de gastos hasta aprobaciones de viaje y reservaciones de salas de reuniones, el portal no era capaz de brindar la interfaz y funcionalidad más actualizada y el desarrollo de nuevas funcionalidades tomaba mucho tiempo e involucraba múltiples equipos de desarrollo.

Con OpenLegacy, un nuevo conjunto de APIs soporta las aplicaciones móviles utilizadas por técnicos de campo para enviar de forma rápida y sencilla reportes de gastos, a través de la subida de fotos desde el teléfono del empleado. Además, la solución de microservicios fue capaz de adaptarse a las necesidades cambiantes del sistema de reporte de costos de la empresa. En tres días, OpenLegacy desarrolló 8 API REST sobre el módulo de Gestión de Capital Humano de SAP para el reporte de gastos, proporcionando una flexibilidad muy necesaria.

Incorporando clientes 3 veces más rápido

Un grupo asesor de inversiones mexicano que administra varias decenas de billones de dólares, buscaba adaptar el proceso de

incorporación de sus clientes para cumplir con sus expectativas cambiantes. El desafío era incorporar funcionalidades como el autoservicio y el acceso móvil de los clientes, a la vez que se cumplía con todas las regulaciones gubernamentales. El plan era aprovechar su back-end de SAP CRM existente, complementado con nueva tecnología API – y por ello recurrieron a OpenLegacy.

Se tardó 15 minutos en crear una API simple, mientras que las transacciones más complejas tomaron menos de una semana en implementarse. La implementación de todas las APIs para el proyecto tomó alrededor de un mes, comparado con los nueve meses sin OpenLegacy. ¿El resultado? Un tiempo de comercialización más rápido, un autoservicio implementado por completo, un proceso de incorporación de clientes optimizado – y un aumento de ingresos incremental de hasta un 10%.

Entrega de microservicios en la nube en días en lugar de meses
Un importante banco con sede en los Países Bajos tenía la misión de adaptar nuevas tecnologías para ayudar a sus clientes a realizar operaciones bancarias de manera más rápida, más fácil y más inteligente. Al asociarse con startups FinTech, se encuentran explorando áreas como la banca abierta, inteligencia artificial, cadena de bloques, economía circular, todo ello en conjunción con sus actividades relacionadas con el cumplimiento de la PSD2.

El banco subcontrataba casi todas sus actividades de TI a tres

proveedores – uno para el despliegue de aplicaciones, otro para el soporte y mantenimiento de aplicaciones, y otro para la infraestructura TI. La fuerza laboral subcontratada consistía en cientos de desarrolladores externos y personal de TI en varios continentes. Esto significaba que la introducción de cualquier cambio o nueva funcionalidad era costosa y requería mucho tiempo – tomando hasta tres meses para el lanzamiento de nuevos software, servicios e incluso APIs basadas en microservicios.

Con nuestro Conector IMS DC, OpenLegacy ayudó al banco a exponer APIs directamente desde el sistema core mainframe z/OS; y creó y desplegó nuevas APIs de microservicios encapsulando tres workflows core del mainframe en un día.

Finalmente, OpenLegacy ayudó a desplegar APIs de microservicios en días en lugar de meses, facilitó un desarrollo ágil y permitió un cambio cultural en toda la organización. Esta incomparable velocidad permite una banca abierta, presencia en múltiples canales y una transformación digital.

Mejorando la eficiencia del personal interno

Antes: A pesar de que una compañía de seguros pudo modernizar la mayoría de sus ofertas, su oferta de seguro vehicular fue dejada como una aplicación heredada. Como resultado, los agentes se veían forzados a cambiar entre un navegador web y una anticuada "pantalla verde", lo cual impactaba la productividad y la capacidad

de respuesta.

Después: La compañía aseguradora pudo exponer un servicio de su sistema de gestión de reclamos AS/400 que presentaba todos los reportes relacionados a una queja específica dentro de las aplicaciones web principales de seguros vehiculares. La prueba de concepto inicial se completó en tan solo cinco días. En modo de producción, los agentes de seguro lograron ahorros de tiempo de hasta un 30%.

“OpenLegacy nos permitió conectar nuestras aplicaciones IBM i y AS/400 a nuestro portal para agentes de seguros sin necesidad de cambiar nuestras aplicaciones COBOL, lo cual hubiera sido un enorme y costoso dolor de cabeza. No podíamos creer que OpenLegacy fuera capaz de cumplir con todas nuestras restricciones de seguridad, rendimiento y diseño.” Director de TI de Servicios de Seguros.

> “OpenLegacy nos permitió conectar nuestras aplicaciones IBM i y AS/400 a nuestro portal para agentes de seguros sin necesidad de cambiar nuestras aplicaciones COBOL, lo cual hubiera sido un enorme y costoso dolor de cabeza. No podíamos creer que OpenLegacy fuera capaz de cumplir con todas nuestras restricciones de seguridad, rendimiento y diseño.”

Acelerando la innovación bancaria en un 50%

Antes: A pesar de querer ser un banco innovador y preparado para FinTech, un importante banco latinoamericano estaba siendo sofocado por los sistemas heredados resultantes de una fusión, distribuidos entre dos países y con algunos de los costos operativos más grandes de cualquier banco en la región. La programación mainframe con COBOL era hecha en Colombia, la programación Java era hecha en Panamá y el mantenimiento de la infraestructura estaba a cargo de un integrador de sistemas globales externo. Una complejidad excesiva demoraba los tiempos de salida al mercado de los nuevos productos y servicios basados en mainframe, requiriendo usualmente seis meses o más para su despliegue. Entre sus principales prioridades se encontraba un servicio de Procesamiento de Pagos para sus clientes comerciales.

Después: El “viaje digital” del banco está basado en el software de integración y gestión de API habilitado para microservicios de OpenLegacy. El primer proyecto incluyó capacitación, creación y despliegue de 12 nuevas APIs en solo 8-9 semanas y con tan solo cuatro desarrolladores Java. El tiempo total para el despliegue de su nuevo servicio de Procesamiento de Pagos fue de 90 días, 50% más rápido que cualquier proyecto mainframe típico. El banco consideró estos lapsos de tiempo como excepcionales, considerando que también se trasladaron de IBM BlueMix a Amazon Lambda a mitad del proyecto. Además, el servicio se ejecuta entre un 20% y un 30% más rápido que aquellos sin microservicios construidos previamente, y una prueba de estrés DevOps concluyó que OpenLegacy y Lambda juntos superaron sus objetivos al manejar 60,000 solicitudes simultáneas.

Banco crea seis APIs globales en 2 semanas

Antes: Las demandas de servicios globales digitales de un importante banco global llevaron a una acumulación de más de 100 APIs fundacionales, necesarias para crear nuevas aplicaciones y experiencias para sus clientes.

Alrededor de 200 desarrolladores estuvieron trabajando en el proyecto por más de un año, utilizando un producto popular para puertas de enlace y orquestación de APIs. El producto no se especializaba en aplicaciones heredadas (core) ejecutadas en AS/400 y plataformas mainframe, y a pesar de su popularidad, no

logró solventar las necesidades del banco.

Específicamente, el producto no pudo generar APIs exponiendo programas RPG, logró únicamente gestionar y exponer APIs existentes. Por lo tanto, los desarrolladores del banco tuvieron que escribir código AS/400 extra en RPG para poder exponer funcionalidades. En otros casos, tuvieron que cambiar código existente; una práctica invasiva para aplicaciones usadas y probadas por muchos años. Más allá de agregar nuevas funcionalidades, se invirtió tiempo considerable en la prueba, en la certificación y personalización regional, lo cual ralentizó aún más las cosas. Irónicamente, una herramienta que se suponía debía reducir los tiempos de desarrollo y esfuerzos, terminó creando un esfuerzo manual adicional.

Otro requerimiento crítico para el éxito del proyecto era crear una API Global: Una API unificada con la misma experiencia de usuario final para todos los países, regiones, o entorno técnico subyacente. Desde un punto de vista empresarial, esto requeriría la extracción de una lógica y funcionalidad comunes que pudieran servir como un punto de lanzamiento único para nuevos productos y servicios, capaces de construirse y desplegarse de manera unificada y global.

Después: Los consultores de integración de API de OpenLegacy desarrollaron una arquitectura alternativa, mostrando que era

posible eliminar todas las capas intermedias y la lógica de canal AS/400. En su lugar, el banco podía exponer las transacciones AS/400 directamente y moverlas a una aplicación Java en dónde el banco desarrolla nuevas aplicaciones y lógica.

Ahora el banco puede ejecutar y orquestar transacciones fuera de su entorno heredado. Esto fue posible gracias a la arquitectura de producto simplificada de OpenLegacy, que cuenta con un número mínimo de capas y conectividad directa (en línea recta) con la transacción CBS.

Como resultado, se crearon seis APIs globales clave en tan solo dos semanas y este nuevo enfoque permitió al banco acelerar las innovaciones "omni-canal" para webs móviles o la nube, para así servir a los clientes donde y cuando ellos lo elijan. Además, la simplicidad y la automatización permitieron sorprendentes mejoras de rendimiento de APIs de hasta siete veces.

"En dos días, OpenLegacy creó APIs para transacciones CBS estándar, incluyendo pagos y otros productos financieros, en comparación con las 5-7 semanas requeridas al utilizar el producto anterior de orquestación API y la arquitectura TI existente." Ejecutivo de importante Banco.

> “En dos días, OpenLegacy creó APIs para transacciones CBS estándar, incluyendo pagos y otros productos financieros, en comparación con las 5-7 semanas requeridas al utilizar el producto anterior de orquestación API y la arquitectura TI existente.”

Estos son algunos ejemplos de algunas industrias y hay muchos más en (www.openlegacy.com/case-studies). A medida que organizaciones establecidas por todo el mundo enfrentan la creciente economía digital e influencia de los millennials, y su impaciencia digital, nunca ha existido un mejor momento para considerar los microservicios y las APIs.

Independientemente del proveedor y del enfoque que elijas, esperamos que *tu* viaje digital te lleve a la eficiencia de TI, a ciclos más rápidos, mayor escalabilidad y a una diferenciación competitiva.

Acerca de OpenLegacy

La integración impulsada digitalmente de OpenLegacy permite a las organizaciones con sistemas legados lanzar nuevos servicios digitales de manera más rápida y eficiente que nunca. Se conecta directamente incluso a los sistemas legados más complejos, evitando la necesidad de capas adicionales de tecnología. Luego genera automáticamente API en minutos, integrando rápidamente esos activos en nuevas e interesantes mejoras. Finalmente, los entrega como microservicios estándar o funciones sin servidor, brindando a las organizaciones velocidad y flexibilidad mientras reduce drásticamente los costos y recursos. Con OpenLegacy, las compañías líderes de la industria lanzan nuevas aplicaciones, funcionalidades y actualizaciones en días en lugar de meses, lo que les permite convertirse en realmente digitales hasta su esencia. Para más información visita www.openlegacy.com