E-BOOK

ENSEÑARLE A BAILAR A UN ELEFANTE

La evolución deliberada en equipos, procesos y aplicaciones

Burr Sutter, *Director de Experiencia del Desarrollador*
Deon Ballard, *Marketing de Contenido*

¿QUÉ SIGNIFICA EL CAMBIO?
VER AL ELEFANTE EN LA HABITACIÓN
UNA VISIÓN DARWINISTA DE LA TRANSFORMACIÓN DIGITAL
"LA DISTRIBUCIÓN CONTINUA NO ES POSIBLE": LEY DE CONWAY, DEVOPS Y CULTURA
- La cultura primero
- DevOps como el primer paso

DISEÑAR UNA ARQUITECTURA DE APLICACIONES: ANALIZAR LOS MICROSERVICIOS
- Resistencia al diseño, deudas técnicas y estrategia
- Definición de microservicios y monolitos
- Falacias de la informática distribuida
- Repensar una aplicación significativa

CONDUCIR CON RAPIDEZ (DE FORMA RESPONSABLE)
- Autoservicio, automatización y CI/CD
- Implementaciones avanzadas e innovación

CÓMO ENSEÑARLE A BAILAR A UN ELEFANTE
- Elija su etapa
- Defina sus principios operativos
- Estrangular su monolito

CONCLUSIÓN

facebook.com/redhatinc
@RedHatIberia
Red Hat EMEA

es.redhat.com

Las aplicaciones se han trasladado fuera del departamento de TI. Existe una verdad elemental que dice que todas las empresas del hoy son empresas de software, y la capacidad para proveer nuevos servicios y funciones rápidamente a los clientes es uno de los diferenciadores competitivos clave que puede ofrecer una empresa. La agilidad de TI es una piedra que arrojan los Davids emergentes para derrocar a los masivos Goliats.

Érase una vez, hace muchas generaciones atrás (contadas en años tecnológicos), los departamentos de TI eran departamentos internos, enfocados en mantener la infraestructura y los servicios dentro de la compañía. Es posible que algunas empresas hayan tenido servicios de cara al exterior, en especial, servicios web, pero esto continuaba siendo un área restringida y pequeña. La TI no era un departamento estratégico o geerador de ingresos; era un entorno de soporte considerado como un centro de costos.

Uno de los resultados de un entorno centrado en la infraestructura era que los desarrolladores no captaban el sentido de lo que hacía su código. Los ciclos de entrega eran largos y los cambios lentos. Un desarrollador podía trabajar en algún proyecto y someter al código a pruebas u operaciones, y lo lanzaría meses después. Debido a estos prolongados tiempos de entrega, los ingenieros perdían la alegría de ser desarrolladores: de crear algo y ver cómo funciona en el mundo real.

Uno de los cambios grandes y poderosos con la transformación digital y los cambios relacionados con la cultura y la tecnología, como DevOps, es que se vuelve a introducir la alegría de crear un código. Los desarrolladores pueden crear algo y luego ver cómo funciona en la práctica. Este es un giro poderoso puesto que reintroduce la inmediatez de crear un código. Ver las aplicaciones en vivo proporciona a los desarrolladores un bucle de retroalimentación que les permite rediseñar y mejorar su código y lograr que sus proyectos prosperen.

El elefante se encuentra en el mismo lugar en el que está hoy su empresa. Estas son etapas de cambio entre los entornos tradicionales y los entornos modernos impulsados por microservicios y DevOps. Algunas organizaciones tienen el privilegio de comenzar desde cero, pero para muchas empresas el desafío es enseñar al enorme elefante a bailar como una ágil bailarina.

¿QUÉ SIGNIFICA EL CAMBIO?

La transformación digital es un cambio estratégico para la empresa. Permite a las empresas pivotar sus servicios básicos a medida que las presiones competitivas cambian, o surjan las nuevas regulaciones, y a la vez realizar actualizaciones tan pronto como se detecten las vulnerabilidades.

Sin embargo, no existe una definición común de la transformación digital que sea distinta a "las cosas que cambian". El término "transformación digital" a veces se usa para las nuevas arquitecturas, como los microservicios, o los nuevos procesos, como DevOps, o las nuevas tecnologías, como los contenedores y las interfaces de programación de aplicaciones (API).

Cuando algo puede significar algo, efectivamente, no significa nada. La transformación digital no es algo específico que se puede obtener. Es algo que toda organización debe definir particularmente para sí misma.

No existe un solo patrón de arquitectura o una sola plataforma tecnológica que funcione sin errores en cada entorno individual. Las organizaciones que son exitosas con la transformación digital son aquellas que comprenden más claramente sus objetivos y su cultura, y cuya transformación es diferente a la de las otras. Por ejemplo:

- Walmart, un Black Friday, implementó un código mientras 200 millones de personas estaban en línea.[1]
- Amazon implementa actualizaciones del código a cada segundo (50 millones al año) en cientos de aplicaciones y millones de instancias de nube.[2]
- Etsy realiza 60 implementaciones por día con una aplicación monolítica.[3]
- Netflix implementa cientos de veces al día una arquitectura distribuida compleja, con un solo cambio de código, desde la introducción hasta la producción en tan solo 16 minutos.[4]

Cada una de estas empresas trabaja con estructuras de equipo diferentes y complejas, tecnologías subyacentes, bases de código y arquitecturas. El punto es que no se centraban solo en un patrón en particular o en un solo tipo de tecnología; todo funcionaba correctamente. Todas comenzaron con evaluaciones de sus equipos, su actual deuda técnica y sus estrategias comerciales, y luego, de forma deliberada y coherente, orientaron sus empresas en la dirección correcta. Y obtuvieron resultados.

Ese es el proceso de aprendizaje para que el elefante baile. Cualquiera sea el lugar en el que se encuentra su empresa ahora, usted puede trasladarla hacia donde necesita que esté (deuda técnica, fallos de diseño... y demás) con una estrategia deliberada y clara. Esto significa que debe tener una clara comprensión de lo que desea lograr y qué tan lejos está de su posición actual.

VER AL ELEFANTE EN LA HABITACIÓN

Evaluar su panorama técnico actual y definir una estrategia comercial son tareas casi sencillas. Obtener este tipo de perspectiva es complejo. Hay una parábola muy conocida sobre seis hombres ciegos que se encuentran con un elefante y cada uno toca una parte diferente, intentando identificar al tacto de qué bestia se trata. Uno toca el tronco y piensa que es una lanza; otro toca uno de los lados y piensa que es una pared; otro toca las orejas y piensa que es un ventilador. Lo interesante es que la parábola tiene varios finales diferentes, según cuál sea la fuente. En algunos, los hombres son incapaces de aceptar las afirmaciones de los otros y tienen una disputa. En otros, un hombre vidente se acerca y explica la apariencia en general y unifica sus percepciones.

Los distintos finales son probablemente la parte más realista de la parábola. El mensaje de la historia es que cada persona tiene distinto punto de vista, información limitada y suposiciones en base a esa perspectiva. El resultado de esas perspectivas diferentes representa la comunicación

1 *O'Maidin, Cian. "Why Node.js Is Becoming The Go-To Technology In The Enterprise." NearForm, 10 de marzo de 2014, www.nearform.com/blog/node-js-becoming-go-technology-enterprise/. Se accedió el 1.º de septiembre de 2017.*

2 *McKendrick, Joe. "How Amazon Handles a New Software Deployment Every Second." ZDNet, 24 de marzo de 2015, www.zdnet.com/article/how-amazon-handles-a-new-software-deployment-every-second/.*

3 *"The Great Microservices Vs Monolithic Apps Twitter Melee." High Scalability, 28 de julio de 2014, http://highscalability.com/blog/2014/7/28/the-great-microservices-vs-monolithic-apps-twitter-melee.html.*

4 *Bukoski, Ed, et al. "How We Build Code at Netflix." The Netflix Tech Blog, 9 de marzo de 2016, https://medium.com/netflix-techblog/how-we-build-code-at-netflix-c5d9bd727f15.*

intrínseca y las relaciones internas del equipo; algunos equipos pueden superar sus perspectivas, otros terminan desintegrándose. Lo que vemos depende de lo que somos, dónde estamos, qué es lo que observamos, qué sabemos y qué *no* sabemos.

Esto se complica aún más por el hecho de que la mayoría de las organizaciones existentes tienen más de un elefante. Netflix es un unicornio, no solo por su avanzada arquitectura de microservicios y sus entornos de implementación y tests basados en contenedores. Tiene la capacidad para diseñar sus procesos de equipo y su pila de tecnología basada en su estrategia comercial, sin deudas técnicas. Eso fue un comienzo.

Cualquier organización con aplicaciones heredadas (y equipos heredados) tiene más de un elefante en su habitación, como los siguientes:

- Estructuras de equipos y patrones de comunicación actuales.
- Procesos de implementación, pruebas, diseño y lanzamiento.
- Deuda técnica y aplicaciones básicas existentes.
- Diferentes objetivos o visiones estratégicas entre los departamentos.

Todos estos son elefantes, y cada parte interesada puede ver potencialmente a esos elefantes desde distintas perspectivas.

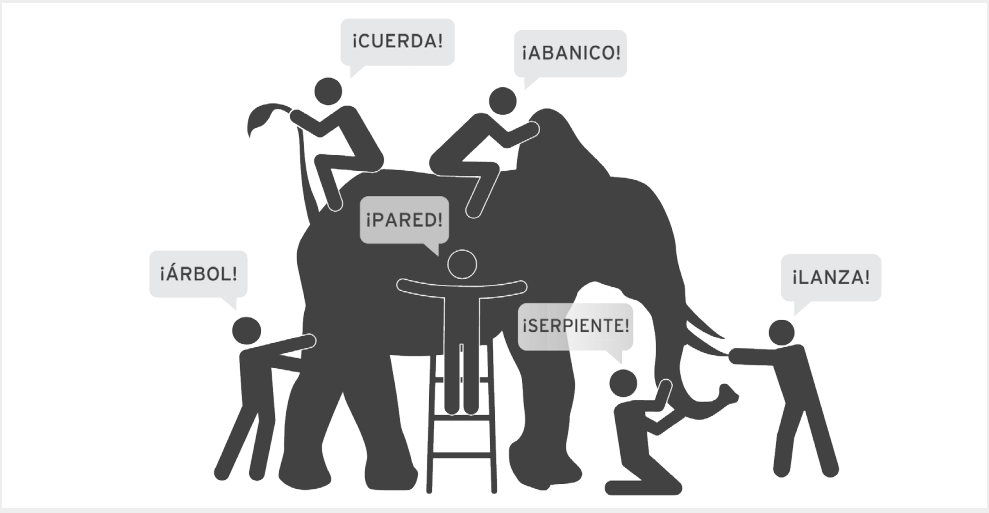


Figura 1. La alegoría de seis hombres ciegos y un elefante

UNA VISIÓN DARWINISTA DE LA TRANSFORMACIÓN DIGITAL

Existe una tendencia a considerar a la TI o a las iniciativas de transformación digital como un concepto binario: usted hace una cosa o hace otra totalmente diferente. Es posible que no sea efectivo por dos razones: primero, porque a veces se puede optar por hacer más de una cosa o elegir una solución híbrida. Segundo, porque con frecuencia no se trata de cambiar un factor; distintos tipos de cambios pueden requerir cambios esenciales diferentes o depender de cambios culturales y de procesos.

Nada sucede en un vacío.

Puede ser más constructivo buscar una transformación digital como una secuencia, con diferentes etapas en el proceso, y que cada una permita la evolución hacia la etapa siguiente.

Figura 2. La etapas del darwinismo digital

DevOps y el cambio de los procesos

La base de la evolución digital es DevOps. Las aplicaciones, como en la estrategia comercial, son un reflejo de los equipos y de la comunicación involucrada en su creación. DevOps, o los cambios de proceso similares, involucra a más partes interesadas en discusiones sobre la implementación y ofrece perspectivas más amplias acerca de cómo las operaciones mantienen el software y la infraestructura (y cómo los clientes y los partners usan en realidad esas aplicaciones). Crea un bucle de retroalimentación más fuerte entre los equipos y requiere líneas abiertas de comunicación. Esta comunicación abierta es la base para cualquier otra etapa de la evolución.

Infraestructura de autoservicio

Esta etapa es un cambio centrado en la tecnología que presenta eficiencias que se asocian normalmente a plataformas modernas de tecnología. Los contenedores y catálogos de autoservicio permiten a los grupos de desarrolladores, pruebas y operaciones poner en marcha entornos consistentes y de manera muy rápida y, en algunas organizaciones, permite llevar el tiempo de lanzamiento de nuevas instancias de días a minutos. ¿Por qué debería un tecnólogo esperar días para obtener un recurso informático?

¿LOS MICROSERVICIOS SON UN ESTADO NECESARIO?

La última etapa de la evolución digital son los microservicios, dado que mantenerlos requiere una gran complejidad. Pero, ¿tiene que ser esa la última etapa de evolución de su organización?

No necesariamente.

Si los otros aspectos de la evolución están establecidos, una arquitectura monolítica aún puede lanzar y usar técnicas de implementación avanzadas, CI/CD e infraestructura escalable y distribuida.

Los microservicios solo pueden ser considerados cuando el tamaño del equipo es grande y necesita realizar los lanzamientos más rápido todas las semanas o según una programación variada. Un equipo o una base de código pequeños no necesitan ser reducidos a una base de código de microservicios.

Automatización del diseño y orquestación

La automatización del diseño es un cambio de dos aspectos. Existe un ángulo tecnológico, con motores de implementación avanzados, como Red Hat® Ansible o Puppet, pero también requiere un cambio en el proceso. Muchas organizaciones tienen establecidos procesos estrictos en relación con la gestión de riesgos y cambios; si estos procesos no se adaptan a metodologías más ágiles, no sería posible aprovechar la nueva tecnología.

Canales de integración continua/distribución continua (CI/CD)

La distribución continua es un compromiso para introducir cambios al software de forma rápida e iterativa. La idea de un canal es que existen procesos y tecnologías establecidos para reducir el riesgo de un código sin calidad (o roto) desde su creación hasta su implementación. Este nivel muestra la consolidación de los pasos previos: DevOps y comunicación abierta entre los equipos, procesos establecidos en relación con pruebas y diseños, y pruebas e implementación automatizadas. Cuando todas esas etapas están consolidadas, recién es posible crear el código rápidamente. Y este es el canal.

Vías de implementación avanzadas

Una vez que se establecen los procesos e infraestructuras para las implementaciones rápidas, entonces es posible comenzar a usar los sistemas de implementación como una vía para mitigar cualquier riesgo durante las actualizaciones, evaluar la efectividad de la funcionalidad y proveer un área práctica para probar las nuevas ideas. Esto puede incluir tener entornos separados y equilibrio de las cargas de trabajo entre estos durante las implementaciones (implementaciones azul-verde), usar dos entornos diferentes para probar la interacción del usuario (pruebas A/B), o implementar actualizaciones en pequeños porcentajes de usuarios y aumentar ese número de forma segura (implementaciones "canary" para evitar desbordamiento de buffer).

Microservicios (o sistemas distribuidos)

Un microservicio es una aplicación pequeña que realiza una función discreta y única. La arquitectura general de aplicaciones podría tener que realizar decenas o cientos de funciones diferentes, y cada una de esas funciones serían definidas y orquestadas en un microservicio. Una arquitectura de microservicios (o cualquier arquitectura informática distribuida) es al mismo tiempo compleja y sencilla. Los servicios individuales son mucho más simples y fáciles de mantener, añadir y retirar, aunque la arquitectura general sea más compleja. Cuando se hace correctamente, un "diseño basado en microservicios es la manifestación última de todo lo que usted conoce en relación con un buen diseño de aplicaciones".[5] Esta arquitectura altamente distribuida permite vías más fáciles para escalar, facilita la introducción de nuevos servicios o actualizaciones, y reduce el riesgo de una falla general del sistema. Esta elasticidad en la arquitectura es el motivo por el que los microservicios se encuentran ampliamente asociados con empresas sorprendentemente innovadoras, como Netflix y Google.

"LA DISTRIBUCIÓN CONTINUA NO ES POSIBLE": LEY DE CONWAY, DEVOPS Y CULTURA

Rachel Laycock dijo en una sesión general de DevNation 2016 que el "software de 'puesta en marcha' es muy difícil".[6] Estaba compartiendo una historia sobre una falla al implementar la distribución continua en una gigantesca firma de servicios financieros que padecía durante semanas la implementación de las actualizaciones (incluso las actualizaciones críticas). La firma pensó que la solución era mudarse a la distribución continua. Después de seis meses de intentar el cambio de los procesos, regresó y le informó a uno de sus vicepresidentes que no tendrían distribución continua.

5 *Cotton, Ben. "From Monolith to Microservices." 3 de enero de 2017, https://www.nextplatform.com/2017/01/03/from-monolith-to-microservices/.*

6 *Laycock, Rachel. "Continuous Delivery." Sesión vespertina. Red Hat Summit - DevNation 2016, 1.º de julio de 2016, San Francisco, California. https://www.youtube.com/watch?v=y87SUSOfgTY*

"Distribuir de forma rápida y continua es CI/CD. Dar soporte a lo que distribuye es DevOps."[9]

BURR SUTTER

Parte del problema son la tecnología y la arquitectura. La empresa de servicios financieros tenía una base de códigos con más de 70 millones de líneas de código y la apariencia apócrifa de una arquitectura de barro. Pero el software era la parte más fácil, era el problema más obvio. El verdadero elefante en la habitación era la incapacidad de la organización para cambiar las conductas de sus diversos departamentos. Esto bloqueaba sus esfuerzos por cambiar.

La cultura primero

Un aspecto crítico para destacar en el camino de la evolución (software) darwinista es que no se trata solo de un cambio de tecnología. Alterna entre cambios de procesos y personas y cambios de infraestructura, y los cambios de la cultura son infinitamente más importantes. Como Laycock concluye en su sesión:[7]

"Puede crear el diagrama de arquitectura más bello que desee. Una vez que involucre a estos elementos: personas, procesos, debe crear el entorno (cultural) que da soporte a la distribución continua y la disciplina de la arquitectura, porque un cambio de proceso o estructura no es un cambio realmente duradero. Las personas no siguen las reglas. Entonces, el verdadero elefante en la habitación es la Ley de Conway. ¿Qué significa exactamente? Las estructuras organizativas heredadas destruirán su hermosa arquitectura *a cada momento, todo el tiempo*".

La clave para ver al software o la tecnología como un cambio evolutivo es que la evolución es una función natural de su entorno. En una empresa, esa es la cultura. Estos cambios que son necesarios para dar soporte a un cambio evolutivo pueden ser impulsados por la administración, pero no pueden ser determinados por esta. Las personas deben desear el cambio. Es una cuestión de libre albedrío, no forzada.

De hecho, Gartner tiene números para esto: "El 90% de las organizaciones que intentan usar DevOps sin abordar específicamente sus bases culturales fracasarán".[8]

Cambiar la infraestructura o la arquitectura de las aplicaciones es fácil. Para cambiar efectivamente lo que produce, primero debe cambiar su cultura.

La ley de Conway establece que "cualquier organización que diseña un sistema producirá, inevitablemente, un diseño cuya estructura es una copia de la estructura de comunicación de la organización". Esta idea tiene dos interpretaciones relacionadas:

- Cambiar su arquitectura o infraestructura no cambiará nada, a menos que también cambie su estructura de comunicación.
- Cambiar su estructura de comunicación resultará en procesos y una infraestructura mejores, casi independientemente de la infraestructura.

La ley de Conway establece que "cualquier organización que diseña un sistema producirá, inevitablemente, un diseño cuya estructura es una copia de la estructura de comunicación de la organización".[10]

DevOps como el primer paso

La metodología ágil era un enfoque al diseño de software que trataba de involucrar a todas las partes interesadas: QA, administración del producto, desarrolladores e incluso documentación, para formar un grupo cohesivo. La idea era clarificar los objetivos mediante iteraciones cortas centradas en tareas específicas expresadas como objetivos de usuario (llamadas historias). Esto rompió con el tradicional método en cascada del software en desarrollo que era una especie de brigada contra incendios formada por un equipo que le pasaba los problemas a otro equipo.

7 *Laycock, Rachel. "Continuous Delivery." Sesión vespertina. Red Hat Summit - DevNation 2016, 1.º de julio de 2016, San Francisco, California. https://www.youtube.com/watch?v=y87SUSOfgTY.*

8 *"Gartner Highlights Five Key Steps to Delivering an Agile I&O Culture." 20 de abril de 2015, www.gartner.com/newsroom/id/3032517.*

9 *DevNation Federal, 8 de junio de 2017, Washington, DC, https://www.youtube.com/watch?v=tQOo2qaUc6w&t=1s*

10 *Conway, Melvin E. (Abril de 1968), "How do Committees Invent?", Datamation*

"Los diseños del equipo son el primer bosquejo de su arquitectura".

– MICHAEL NYGARD, RELEASE IT!

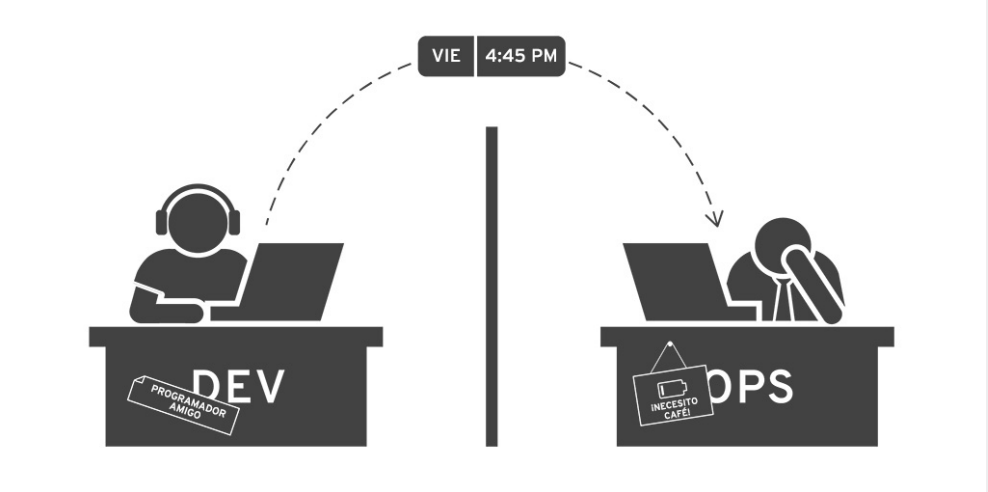


Figura 3. La pared de la división, antes de DevOps

Sin embargo, esto solo abordaba la mitad del ciclo de vida real de una aplicación. Una vez que algo fue desarrollado, fue llevado a operaciones para ser implementado y mantenido (frecuentemente en una ventana de mantenimiento de fin de semana).

El problema es que operaciones no siempre sabe lo que la aplicación debe hacer, lo cual significa que pueden implementarla ineficientemente. Es posible que los desarrolladores no tengan idea del verdadero entorno operativo y creen una aplicación que no funciona en el entorno de producción. Y luego, al intentar reducir el riesgo del cambio, muchas organizaciones establecen un proceso oneroso de gestión para intentar explicar y justificar cualquier cambio.

DevOps es un giro cultural que intenta romper la separación entre desarrolladores, operaciones y partes interesadas de la empresa. Las separaciones entre esos grupos son reales, aunque artificiales. Un equipo puede incluir más personas que comparten una función laboral; DevOps intenta redefinir al equipo para incluir a todos aquellos que comparten el ciclo de vida de una aplicación y abrir la comunicación entre esos grupos.

El cambio cultural es más profundo que DevOps o Agile u otras metodologías. Es un compromiso para integrar verdaderamente a cada persona en el mismo equipo. Si cambia sus patrones de comunicación, puede cambiar sus resultados.

Los cambios importantes pueden comenzar con pasos muy simples. Los cambios culturales sustentan todos los cambios tecnológicos y del proceso. Si se esfuerza por diseñar una cultura DevOps, intente dos cosas:

- Haga que sus desarrolladores pasen un fin de semana con operaciones, vean una implementación de producción y aprendan de lo que tienen que superar.
- Registre la cantidad de pasos o tickets de servicios que necesita realizar un desarrollador para solicitar un nuevo sistema virtual.

Ver cómo otros equipos funcionan, in situ, puede ser un gran impulso para animar a los equipos a generar el cambio en sus procesos o para abrir la comunicación.

LO MÁS DESTACADO DEL INFORME PUPPET'S STATE OF DEVOPS

- 2555 veces más rápidos son los tiempos de entrega
- 200 veces más de implementaciones
- 24 veces más rápido para recuperarse de los fallos
- 3 veces más baja la tasa de fallos por cambios
- 22% menos el tiempo de rectificación

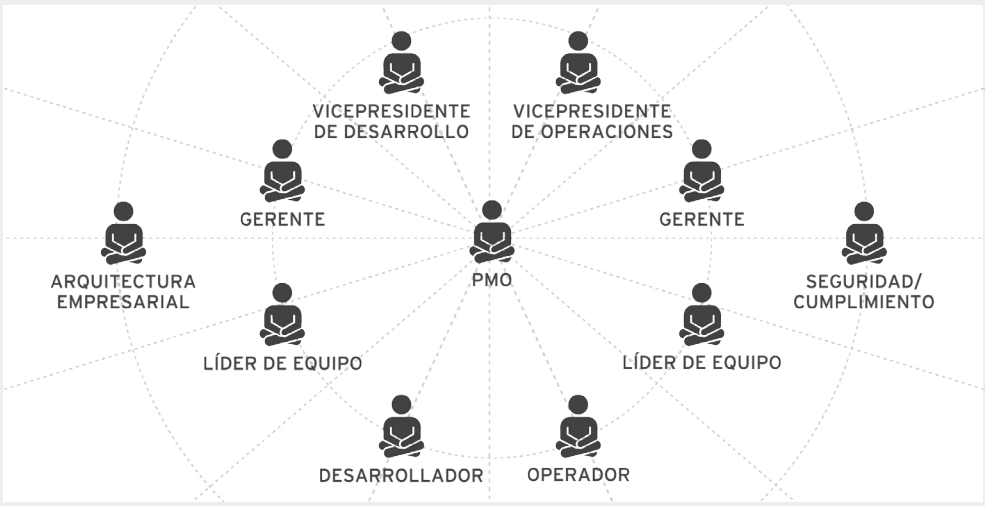


Figura 4. Visión completa de las vías de comunicación entre los equipos

El informe State of DevOps de Puppet muestra cuán efectivo puede ser cambiar la estructura del equipo (y la comunicación).[11] Las investigaciones revelan la experiencia de los equipos DevOps:

- 2.555 veces más rápido son los tiempos de entrega.
- 200 veces más de implementaciones.
- 24 veces más rápido para recuperarse de los fallos.
- 3 veces más baja la tasa de fallos por cambios.
- 22 % menos el tiempo de rectificación.

La velocidad es uno de los beneficios clave de DevOps. Al observar el proceso de lanzamiento completo, todos esos equipos diferentes necesitan involucrarse para realizar el lanzamiento de las aplicaciones y según lo claro que sean los procesos, la infraestructura y la base de códigos, ese proceso de lanzamiento puede implicar mucho esfuerzo.

11 Kim, Gene, et al. "State of DevOps Report" Puppet, 2016, https://puppet.com/resources/whitepaper/2016-state-of-devops-report.

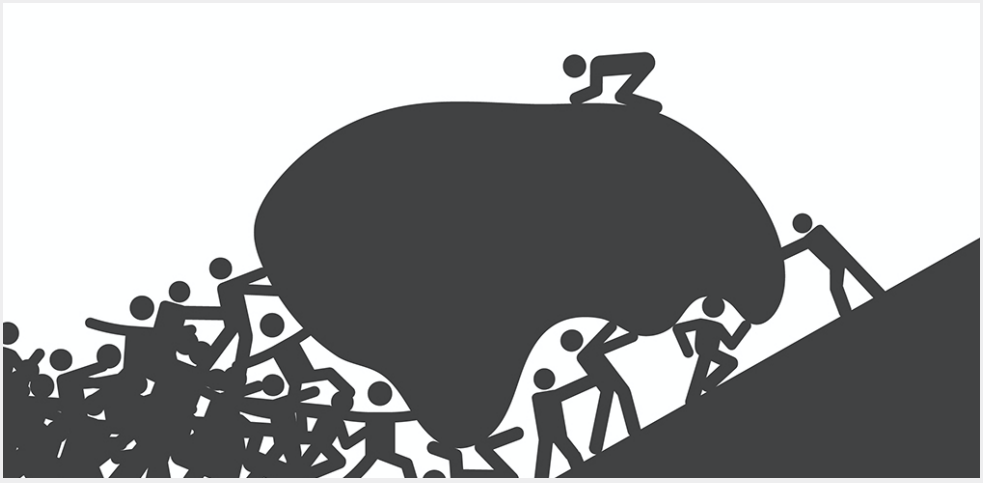


Figura 5. Esfuerzo del equipo para lograr el lanzamiento de una sola aplicación

Este esfuerzo era posible cuando los lanzamientos no se hacían con mucha frecuencia. Una vez que el software se convierte en un factor que impulsa la empresa, todo ese proceso de lanzamiento debe suceder varias veces al año (en las empresas altamente innovadoras como Amazon, ese proceso de lanzamiento puede suceder cientos de veces al día). En ese caso, el esfuerzo hercúleo de múltiples equipos para mudar la aplicación a producción se debe repetir una y otra vez.

Figura 6. Esfuerzo para lograr un solo lanzamiento y muchos más por realizar

DevOps cambia el enfoque de un impulso de emergencia, de escritorio y totalmente práctico a una iteración más sostenible entre la planificación del grupo, el desarrollo, las pruebas y la implementación. Este es el motivo por el que los equipos DevOps ven un crecimiento sorprendente en la productividad, porque el ciclo de vida total es repetible.

Así su arquitectura de aplicaciones sea, en última instancia, un monolito más perfeccionado o microservicios distribuidos, siempre es necesario cambiar la estructura del equipo y la comunicación. La comunicación debe fluir sin problemas antes de que se planifique y después de que se implemente una aplicación. Un equipo por servicio puede ser desglosado en un silo por equipo, excepto que haya creado y habilitado una cultura que de soporte a la comunicación abierta y a la retroalimentación.[12]

DISEÑAR UNA ARQUITECTURA DE APLICACIONES: ANALIZAR LOS MICROSERVICIOS

Una nueva arquitectura de aplicaciones es la etapa final de la evolución digital, pero frecuentemente es el elefante más notorio en la habitación y uno de los más fáciles de identificar. Observar la arquitectura es provechoso, incluso si se debe priorizar el cambio de las plataformas y los procesos.

Resistencia al diseño, deudas técnicas y estrategia

Un gran impedimento para el cambio es que las organizaciones observan sus aplicaciones actuales y no pueden ver la forma de trabajar en ellas. Esta es una razón por la que muchas iniciativas de transformación digital consideran un enfoque de "deshacerse y reemplazar"; anque aveces parece más fácil comenzar desde cero.

El problema, sin embargo, es que la deuda técnica es consecuencia del diseño. Sacar una aplicación sin tener una visión clara de lo que va a reemplazarla significa que eventualmente, la misma arquitectura impenetrable volverá a emerger.

"La esperanza", explicó Rachel Laycock, "no es un método de diseño. Usted tiene que tener toda la intención".[13]

Sea lo que usted esté desarrollando es su deuda técnica. Con las aplicaciones muy nuevas, los equipos frecuentemente empiezan a implementarlas sin un diseño claro, aunque esto no es necesariamente algo malo. Según la descripción de la resistencia al diseño y la deuda técnica de Martin Fowler, señala que comenzar sin un diseño claro permite con frecuencia que la innovación inicial sea mucho mayor y más rápida.[14] Sin embargo, existe un punto en el que un diseño bueno logra una trayectoria estable y equilibra la falta de diseño.

---

12 Cotton, Ben. "From Monolith to Microservices." 3 de enero de 2017, https://www.nextplatform.com/2017/01/03/from-monolith-to-microservices/.

13 Laycock, Rachel. "Continuous Delivery." Sesión vespertina. Red Hat Summit - DevNation 2016, 1.º de julio de 2016, San Francisco, California. https://www.youtube.com/watch?v=y87SUSOfgTY

14 Fowler, Martin. "Design Stamina Hypothesis." 20 de julio de 2007, https://martinfowler.com/bliki/DesignStaminaHypothesis.html.

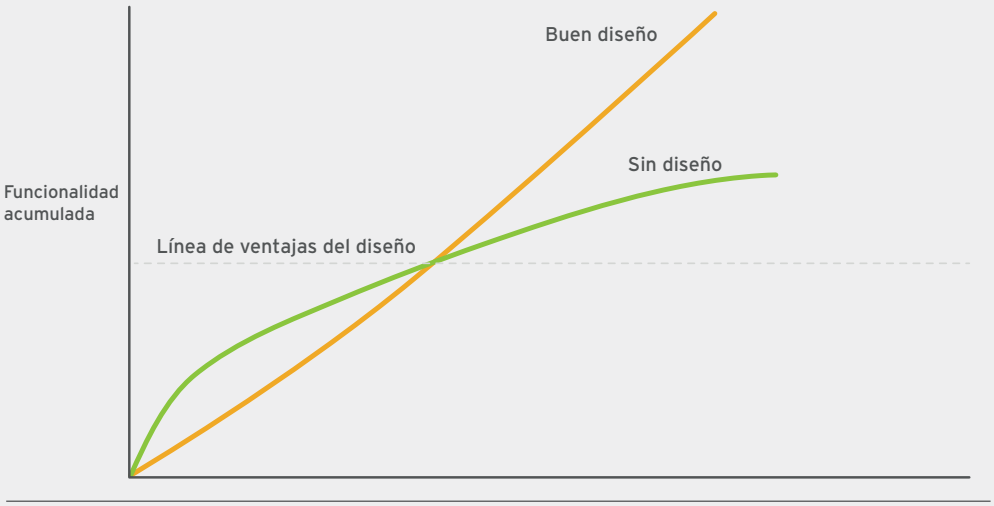


Figura 7. Gráfico conceptual de la teoría de resistencia al diseño

Antes de comenzar la planificación de su arquitectura, debe tener un claro entendimiento de sus prioridades y objetivos estratégicos. Rob Zuber publicó en Information Week:[15]

"Si no tiene una comprensión clara de su empresa, su producto o la posición que ocupa en la lista: lanzar todo en forma prematura en un montón de servicios y desconociendo su manejo, sin duda el resultado no sera el que esperaba... Así que este es un buen momento para pensar sobre la arquitectura, pero también es importante hacerlo de una forma gradual y que permita probar ideas y validarlas, en lugar de sortear obstáculos y crear de una manera totalmente diferente. El resultado será el fracaso y se convertirá en un proyecto de nueve meses que terminará con usted informando a su vicepresidente de ingeniería que, si bien fue un gran aprendizaje, al final, el equipo decidió no embarcarse en esto. Eso no es lo que usted desea. Usted desea que le proporcionen valor y generar impacto en sus clientes y la empresa a través de procesos planificados y bien analizados".

Stephen Elliot escribió en una perspectiva reciente de IDC que antes de definir una arquitectura, debe identificar quién es el cliente o usuario final, qué es lo que valora ese cliente, cuáles son los resultados que usted desea y cómo medirá ese éxito.[16]

En otras palabras, usted no debe comenzar con la forma en que desea lograr algo o con la arquitectura que desea usar. Debe comenzar con la definición del objetivo estratégico y luego, diseñar la arquitectura que le de soporte. Esto le da prioridad a la aplicación.

Definición de microservicios y monolitos

Hoy existen tres grandes arquitecturas de aplicaciones basadas en las relaciones entre los servicios: monolitos (con conexión directa), microservicios (sin conexión) y, si bien están cada vez más en desuso, las arquitecturas orientadas a los servicios (con conexión flexible).

15 Zuber, Rob. "Transitioning to Microservices: The Story of Two Monoliths." InformationWeek, 25 de mayo de 2017, https://www.informationweek.com/devops/transitioning-to-microservices-the-story-of-two-monoliths-/a/d-id/1328972.

16 Elliot, Stephen. "Enabling DevOps with Modern Application Architecture." El punto de vista de IDC, diciembre de 2016.

PLANIFICACIÓN DELIBERADA

- ¿Quiénes son sus clientes o usuarios?
- ¿Qué intentan hacer?
- ¿Qué infraestructura tiene usted?
- ¿Cuál es el ciclo de vida de esa infraestructura?
- ¿Qué servicios o funciones se requieren para un flujo de trabajo individual?
- ¿Cuál es el ciclo de vida para ese flujo de trabajo?
- ¿Cuál es su vía de implementación y con qué frecuencia necesita ser implementado?
- ¿A qué funciones de la empresa afecta?

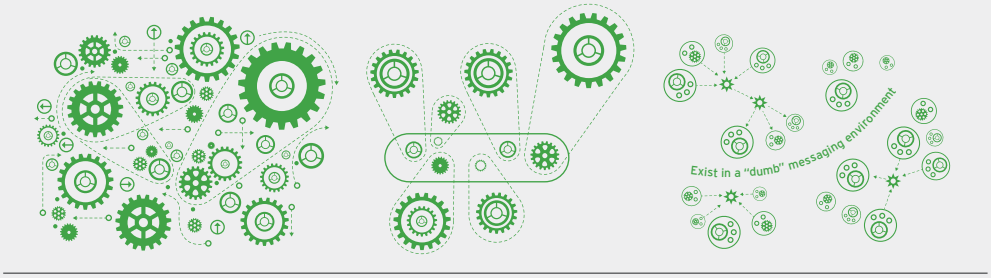


Figura 8. Acoplamiento en arquitecturas de microservicios monolíticas y orientadas al servicio

En términos sencillos, un monolito es una pila de aplicaciones individual que contiene todas las funciones comprendidas en esa única aplicación. Este tiene conexión directa, tanto en la interacción entre servicios como en la manera en que están desarrollados y distribuidos. Actualizar o escalar un aspecto individual de la aplicación monolítica tiene implicaciones para toda la aplicación y su infraestructura subyacente.

El escalado dinámico y la conmutación por error son problemas potenciales de un monolito. Estos tienden a ser abordados con simples diseños de escalabilidad, como un escalado horizontal (duplicando esa función en un clúster) o escalado vertical (instancias de duplicación y expansión de hardware). Un problema de escalabilidad que no es considerado con frecuencia, es el de los equipos de desarrollo y operaciones. Si cada 6 a 9 meses se necesita un equipo completo de 50 personas para lanzar una aplicación monolítica, entonces es posible mejorar el escalado si tenemos cinco aplicaciones más pequeñas con cinco equipos separados y ofrecer actualizaciones individuales cada par de semanas.

La arquitectura monolítica es probablemente la arquitectura de aplicaciones más antigua debido a su simplicidad inicial y a las relaciones más claras entre servicios e interdependencias. Esta arquitectura también refleja más una infraestructura de TI limitada y básica con un desarrollo y procesos de liberación más rígidos.

Debido a que los monolitos son un estilo de arquitectura más antiguo, frecuentemente se los asocia con las aplicaciones heredadas. En comparación, las arquitecturas más modernas intentan lanzar servicios por función o por capacidad comercial para proporcionar más flexibilidad. Esto es especialmente común con las interfaces orientadas al cliente, como las API, las aplicaciones móviles o las web, que tienden a ser más pequeñas y requerir actualizaciones más frecuentes para cumplir con las expectativas del cliente.

Una de las definiciones más recientes de una arquitectura distribuida es la de microservicios. Existen algunas similitudes con otros diseños modulares, como las arquitecturas orientadas al servicio (SOA), pero los microservicios varían de una conexión flexible entre servicios a servicios independientes. La definición de un servicio individual es generalmente clara y los servicios se pueden añadir, actualizar o eliminar de una arquitectura más grande con facilidad. Esto tiene

beneficios tanto de escalabilidad dinámica como de tolerancia a fallos: los servicios individuales se pueden escalar como sea necesario sin requerir una infraestructura pesada o pueden tener conmutación por error sin impactar en otros servicios. Tal como escribió Ben Cotton: "un diseño basado en microservicios es la manifestación última de todo lo que usted conoce en relación con un buen diseño de aplicaciones".[17]

La fluidez de la arquitectura de microservicios significa que esta se encuentra fuertemente asociada con tecnologías dinámicas como contenedores y nubes, lo que permite aumentar y eliminar las instancias individuales con facilidad, incluso de forma programada.

La inmediatez de la informática distribuida tiene beneficios directos, tanto para la organización como para los equipos que pueden ver el impacto de su trabajo, como por ejemplo:

- Mayor tolerancia a los fallos e interrupciones de servicios reducidas.
- Protocolos simples como JSON/REST y HTTP/OAuth para una integración más simple.
- Servicios políglotas que permiten flexibilidad de los desarrolladores.
- Menor tiempo de comercialización para las nuevas aplicaciones y funciones.
- Recuperación de datos simplificada y servicios compartidos, sin tener que usar buses de mensajería pesados o conversión.[18]

En Coding the Architecture, se establece que una arquitectura monolítica es opuesta a una arquitectura de microservicios cuando el tiempo de ejecución para una aplicación es en sí mismo monolítico y estático.[19] Muchas aplicaciones tienen numerosos paquetes o módulos que pueden ser independientes unos de otros en la manera que fueron creados o implementados, pero la misma aplicación interactúa como una entidad individual.

La pregunta subyacente es cuál estilo de arquitectura es el más apropiado. Un acto reflejo sería asumir que lo nuevo es siempre lo mejor, pero lo importante es dar un paso atrás y evaluar qué es lo ideal o lo que se ajusta más para los resultados comerciales que desea. Los ingenieros de Etsy y Netflix tuvieron una interesante discusión por Twitter sobre la necesidad de los microservicios para la implementación continua y la autonomía de los desarrolladores. Los ingenieros de Etsy señalaron que tenían equipos de desarrollo pequeños, ágiles y basados en la funcionalidad; y que sus implementaciones eran extremadamente rápidas (aproximadamente 60 al día) al mismo tiempo que ejecutaban una aplicación monolítica. Encontraron un sistema que funcionaba para ellos y para su cultura.

La arquitectura y las tecnologías cambian y evolucionan. "La práctica recomendada de ayer es el antipatrón de mañana", señaló Laycock en su discurso en una sesión de DevNation. "Los microservicios son una elección. Y son la respuesta. No son la solución; son una solución posible".[20]

---

17 *Cotton, Ben. "From Monolith to Microservices." 3 de enero de 2017, https://www.nextplatform.com/2017/01/03/from-monolith-to-microservices/.*

18 ***Lambert, Natal**ie. "Micro Services: Breaking down Software Monoliths." NetworkWorld, 22 de noviembre de 2017, https://www.networkworld.com/article/3143971/application-development/micro-services-breaking-down-software-monoliths.html.*

19 *Annett, Robert. "What Is a Monolith?" Coding the Architecture, 19 de noviembre de 2014, http://www.codingthearchitecture.com/2014/11/19/what_is_a_monolith.html.*

20 *Laycock, Rachel. "Continuous Delivery." Sesión vespertina. Red Hat Summit - DevNation 2016, 1.º de julio de 2016, San Francisco, California. https://www.youtube.com/watch?v=y87SUSOfgTY*

La pregunta para una organización no es *si podemos reemplazar nuestro monolito con microservicios*. La pregunta es *¿cuál es nuestro objetivo estratégico y qué debemos hacer para lograrlo?* La comprensión de la estructura de la comunicación y la cultura, las funciones comerciales y los flujos de trabajo necesarios influye en la manera en que los servicios se conectan, qué ciclos de vida tienen y finalmente, en la arquitectura de aplicaciones.

TABLA 1. COMPARACIÓN DE LA ARQUITECTURA MONOLÍTICA Y LA DE MICROSERVICIOS

	MONOLITOS	MICROSERVICIOS
Desarrollo	• Desarrollo inicial más rápido • Funciones difíciles de cambiar o añadir	• Diseño inicial crítico • Añadir o cambiar los servicios es más fácil
Flujos de trabajo de las aplicaciones	• Aplicaciones fáciles de adaptar en los flujos de trabajo • Función de implementación (por ej., autenticación o monitoreo) en una sola locación	• Servicios más difíciles de asignar en los flujos de trabajo • Es posible que las interdependencias o requerimientos entre servicios no sean claras
Capacitación y mantenimiento	• Arquitectura simple • Requerimientos de desarrollo rígidos para entorno y lenguaje	• Arquitectura flexible con más complejidad de diseño • Servicios políglotas con API estandarizadas o mensajería para conectar
Escalamiento	• Difícil de escalar; depende de la infraestructura del hardware • Escalamiento de toda la aplicación para picos de demanda de servicios individuales	• Servicios individuales fáciles de escalar sin impactar en la arquitectura general • Utiliza infraestructura definida por software (contenedores, nube) para respuesta dinámica
Actualizaciones, conmutación por error, tiempo de inactividad	• Todos los servicios tienen conexión directa • Los servicios se deben actualizar juntos; las versiones tienen conexión • Riesgo de fallo del sistema si hay un fallo de servicio individual	• Servicios sin conexión • Los servicios se pueden añadir o actualizar de forma independiente • El riesgo de fallo está limitado a un número más pequeño de servicios
Automatización	• La automatización es en gran parte innecesaria	• Se requiere automatización y orquestación

Falacias de la informática distribuida

Un ingeniero replicó que los microservicios convierten cada interrupción en un misterio criminal.[21] Ese es un resumen memorable del problema más importante con la informática distribuida: la complejidad dispersada.

21 @HonestStatusPage. Twitter, 7 de octubre de 2015, https://twitter.com/honest_update/status/651897353889259520?lang=en.

Aumento de costos y costos de transacción

Los cambios requeridos en infraestructuras para una informática realmente distribuida pueden hacerse a nivel de costos de capital importantes. Además, existen costos indirectos por capacitación o adquisición de nuevas habilidades, cambios de estructuras de equipo y migración de sistemas. Estos costos pueden resultar en ahorros futuros en indicadores como el tiempo de comercialización y tiempo de inactividad reducido (si la nueva arquitectura está implementada efectivamente), pero esos beneficios no son inmediatos.

Mayor complejidad

En lugar de un punto de fallo único (catastrófico), como con un monolito, una arquitectura de microservicios tiene cientos de posibles puntos de fallo diferentes. Al igual que un monolito, localizar ese fallo puede ser difícil porque es posible que la causa raíz no sea obvia; sin embargo, los microservicios añaden una complejidad extra porque las interdependencias entre los servicios son incluso menos aparentes.

Hasta los ciclos de lanzamiento más rápidos y la estructura de equipo más ágil implican complejidad. La comunicación efectiva es crítica para los microservicios. Cada arquitectura de software es una cuestión de equilibrar la complejidad inherente. Esa complejidad puede estar escondida en la misma aplicación (monolitos) o puede ser impulsada a las estructuras de comunicación del equipo (microservicios).

Análisis y diseño de sistemas

Diseñar una arquitectura de microservicios efectiva requiere de un análisis importante de los sistemas. Debe haber una comprensión de las interacciones entre servicios, las funciones comerciales y la experiencia del usuario. Este análisis de los sistemas también se debe extender a las estructuras de comunicación y a los procesos en la cultura de la organización. Gartner destaca que sin una comprensión del sistema en general, las arquitecturas de microservicios funcionarán en gran medida como el monolito esterotípico: "Al no optar por un enfoque holístico que considere a la arquitectura del software, la infraestructura de desarrollo y los procesos de desarrollo, no logrará resultados óptimos y usted continuará padeciendo muchos de los inconvenientes de un sistema de software monolítico".[22]

Limitaciones de recursos

Las limitaciones de recursos con los monolitos son obvias debido a los problemas de escalamiento. Cualquiera sea el sistema, tiene la capacidad de hardware suficiente para manejar los picos de cargas de los servicios más grandes y el resultado es una capacidad sin usar o el riesgo de no tener suficiente capacidad para manejar los picos de uso.

Con los microservicios, la arquitectura es flexible y ya que cada servicio individual es muy pequeño, es fácil escalar los nuevos servicios en recursos muy livianos y temporarios, como los contenedores o las instancias de nube.

Debido a que los requerimientos de recursos individuales para un servicio dado son pocos, es posible pasar por alto o minimizar las demandas únicas de recursos de la arquitectura en general. Esto conlleva algunas suposiciones:

- La red siempre es confiable.
- No hay tiempos de espera.
- El ancho de banda es infinito.
- La red es segura.

---

22 Knoernschild, Kirk. "Refactor Monolithic Software to Maximize Architectural and Delivery Agility." Perspectivas claves de Gartner, 18 de mayo de 2017

- La topología de la infraestructura no cambia.
- Existe un solo administrador.
- No hay costos de transporte.
- La red es homogénea.

Existe otra limitación para los recursos, si bien afecta a un nicho de casos prácticos de informática. Las aplicaciones de rendimiento extremadamente alto pueden tener requerimientos que son muy exigentes para una arquitectura de microservicios, como patrones climáticos o el mapa de ADN.

Las falacias de microservicios son importantes para destacar solo porque ningún sistema es perfecto. Encontrar una tecnología perfecta (o arquitectura) que solucione sus problemas no debería ser el único objetivo. En su lugar, centrarse en su cultura y comunicación y perfeccionar sus procesos crearán una organización más madura y efectiva. Desde ese lugar, su organización tendrá la capacidad de diseñar una arquitectura funcional que cumpla sus objetivos específicos.

Repensar una aplicación importante

Gran parte de la discusión acerca de la transformación digital se centra en la infraestructura y la cultura. Esto tiene sentido ya que son consideraciones básicas. Sin embargo, la infraestructura y la cultura son los medios; el objetivo es crear una aplicación que sea útil a sus usuarios e importante para la organización.

Las aplicaciones importantes tienen ciertas características:

- Tienen capacidad de respuesta para los usuarios
- Son reflejo de la función o propósito comercial principal
- Se adaptan o reaccionan a los cambios dinámicos en el entorno
- Están conectadas en todos los entornos
- Son livianas y flexibles para que las funciones se puedan añadir y mantener

Cuando una aplicación cumple con esas características, es útil.

Tanto la arquitectura monolítica como la de microservicios pueden reflejar esas características. La arquitectura es una elección de diseño. Incluso con una arquitectura monolítica, cambiar su perspectiva sobre distintos elementos es fundamental para crear una aplicación moderna y ágil. Etsy puede ejecutar su distribución continua, decenas de actualizaciones por día, grandes cargas de usuarios e incluso aplicaciones móviles porque tiene un monolito majestuoso: una aplicación con una arquitectura deliberada.

Esta intencionalidad se debe reflejar tanto en una arquitectura monolítica como en una de microservicios en distintas áreas clave, que incluyen, entre otras:

- Integración.
- Gestión de datos y consistencia.
- Mensajería y comunicaciones de servicios.
- Procesos coherentes o patrones de flujo de trabajo.

A grandes rasgos, los entornos monolíticos tradicionales tendían a abordar estas cuestiones como problemas que necesitaban una solución o como un aspecto negativo de una aplicación. La integración, por ejemplo, algunas veces era considerada como una alternativa a unir distintas aplicaciones o fuentes de datos; era un apósito que unía partes distintas de un entorno. Incluso los elementos como los flujos de procesos eran considerados como una manera de impulsar la productividad sobre la intervención manual y los flujos de trabajo automatizados, pero no siempre era una decisión de diseño importante.

Con una aplicación moderna, los aspectos de la arquitectura, como la integración y los flujos de procesos no son secundarios, son fundamentales para el funcionamiento de la aplicación. Esto se ve claramente en las arquitecturas de microservicios (aunque también aplica a los monolitos majestuosos).

Integración y mensajería

Con los microservicios, el conflicto está en cómo conectar esos servicios de manera que la autonomía de estos se mantenga y, al mismo tiempo, se permita la libre comunicación. Esa es una cuestión de integración y mensajería. La integración define cómo esos servicios por separado se comunican entre sí. Esta es una elección de diseño esencial. Desde una perspectiva de infraestructura, los microservicios a veces se dividen en "contenedores más API" para cumplir con la necesidad principal de acoplar esos servicios, donde los contenedores son los servicios y las API son la conexión. (Un enfoque similar sería el de "contenedores más mensajería": Cualquier tecnología que proporcione una manera de transmitir datos entre servicios).

Esto es distinto, incluso, de las arquitecturas orientadas al servicio, ya que la mensajería y la integración no están centralizadas en buses y los datos no están convertidos entre servicios.

Flujos de procesos

Una aplicación moderna tiene capacidad de respuesta para sus usuarios. Esto es más visible en las aplicaciones móviles que se centran en el consumidor y que son simples e inmediatas. Cuando un cliente comienza una transacción, como revisar un saldo bancario o navegar para buscar un pasaje aéreo, se debe lanzar un flujo de trabajo inmediatamente. Ese flujo de trabajo se debe adaptar a la próxima decisión del usuario, si bien también debe cumplir con los protocolos de la organización. La gestión de procesos empresariales (BPM) tradicional era al principio una manera de automatizar tareas para lograr eficiencia, pero al disponer de esto en una arquitectura de aplicaciones moderna, la gestión de procesos empresariales se convierte en un elemento muy importante para entregar funciones y contribuir a la experiencia del usuario.

Datos

En algún momento, todos los datos tienen estado. Los datos se deben almacenar y deben ser accesibles a los servicios en la arquitectura (así estén distribuidos o sean monolíticos), por lo que debe haber un entendimiento de cómo los datos se mueven entre los servicios y qué tipo de datos son generados.[23] El desarrollador, Christian Posta, escribió que la parte más difícil sobre los microservicios es la gestión de datos, con el objetivo de tratar de definir los límites naturales y transaccionales entre servicios.[24] El conflicto aquí es entre la consistencia de los datos, la accesibilidad y la autonomía. Definir los dominios naturales permite informar los modelos de datos y las estructuras de almacenamiento.

---

23 Brown, Kyle. "Refactoring to Microservices, Part 2: What to Consider When Moving Your Data." IBM developerWorks, 4 de mayo de 2016, https://www.ibm.com/developerworks/cloud/library/cl-refactor-microservices-bluemix-trs-2/index.html.

24 Posta, Christian. "The Hardest Part About Microservices: Your Data." 14 de julio de 2016, http://blog.christianposta.com/microservices/the-hardest-part-about-microservices-data/.

"Las personas intentan copiar a Neflix, pero solo pueden copiar lo que ven. Copian los resultados, no el proceso".[27]

– ADRIAN COCKCROFT, EXARQUITECTO JEFE DE NUBE DE NETFLIX

CONDUCIR CON RAPIDEZ (DE FORMA RESPONSABLE)

Uno de los objetivos principales de la transformación digital es lanzar aplicaciones más rápidamente. Sin embargo, la velocidad solo consiste en mejoras de la eficiencia. Para poder transformar, la velocidad tiene un propósito: permite la innovación rápida, nuevas funciones y la habilidad de probar nuevas ideas.

Ron Kohavi, un distinguido ingeniero y Gerente General del Equipo de Experimentación de Microsoft para inteligencia artificial, destacó en un discurso dado en el Open House 2013, en la Universidad de Ciencias Informáticas y Tecnología de ingenierías de Minnesota, que menos de un tercio de las ideas de mejora influyen en las métricas para las cuales fueron diseñadas.[25]

La manera de evaluar una buena idea es mediante pruebas efectivas y exhaustivas, no solo de la calidad del código, sino también de la experiencia del usuario y sus preferencias. Este conocimiento solo se adquiere a través de la experiencia. Como lo describe Kohavi: "los datos superan la intuición".

Este es el propósito de la distribución continua y las técnicas de implementación avanzadas. La CI/CD es la plataforma para la implementación rápida; las técnicas de implementación son herramientas para la experimentación y el perfeccionamiento.

Detrás de esas dos etapas se encuentra el cambio cultural que impulsa la innovación y resiste los fallos y los riesgos. La innovación no es un destino o un punto fijo; es un proceso alimentado por la experimentación. Correr el riesgo de fallos en el camino de la innovación requiere una cultura de humildad.

Autoservicio, automatización y CI/CD

Una de las primeras reestructuras para la transformación digital es mudarse a una cultura DevOps, con equipos pequeños y dinámicos y comunicación en toda la organización. El paso siguiente es la tecnología: proveer una infraestructura que de soporte a ciclos rápidos de desarrollo.

Existen dos etapas de tecnología muy relacionadas con lo siguiente:

- Infraestructuras elásticas y de autoservicio, que es la capacidad de solicitar y recibir una instancia acorde a especificaciones exactas casi instantáneamente.
- Automatización u orquestación, que es la capacidad de crear automáticamente y gestionar múltiples instancias en un entorno completo.

Estas son tecnologías complementarias que no se pueden automatizar sin un entorno elástico, y gestionar cientos de instancias potenciales es complejo de realizar sin herramientas para hacerlo consistente y repetible.

Mejorar la tecnología en esta etapa de transformación puede incrementar la productividad de forma tangible. Con un cliente Red Hat, la presentación de un catálogo de autoservicios para que los desarrolladores puedan solicitar rápidamente sistemas virtuales redujo el tiempo para obtener un sistema solicitado de cinco días a unos 15 minutos, y cambió los procesos de manuales a automáticos.[26] Este cambio liberó los recursos desde el punto de vista de las operaciones y mejoró la productividad (y la moral) de los desarrolladores.

---

25 *"Online Controlled Experiments: Introduction, Insights, Scaling, and Humbling Statistics." SOBACO University of Minnesota, 18 de octubre de 2013, https://sobaco.umn.edu/content/online-controlled-experiments-introduction-insights-scaling-and-humbling-statistics.*

26 *"Red Hat Virtualization Increases Efficiency and Cost-Effectiveness." Estudio del impacto económico total de Forrester, 26 de enero de 2017, www.redhat.com/es/resources/virtualization-tei-forrester-analyst-paper.*

27 *https://twitter.com/kelseyhightower/status/641886057391345664*

"Los docker se describen mejor como los contenedores con una opinión, ya que su conjunto de herramientas fomenta y crea sobre los principios de una infraestructura inmutable en la que la configuración de las aplicaciones es corregida por el desarrollador y se implementa para producción con cambios mínimos en la configuración".[28]

– IDC

Si bien las tecnologías son complementarias, no son prescriptivas. Las infraestructuras elásticas pueden ser nubes (públicas o privadas), máquinas virtuales o contenedores. La automatización puede ser un componente en el sistema de infraestructura (como el proyecto Heat para orquestar con OpenStack®) o puede ser una herramienta externa, como Red Hat CloudForms o Kubernetes, con contenedores basados en Docker. Existen avenidas tecnológicas diferentes, según las habilidades y la infraestructura existentes de su organización.

Una razón por la cual los contenedores (como Docker o Red Hat OpenShift) se han asociado tanto a CI/CD es que proporcionan entornos de sistemas rígidos y repetibles, lo que implica que existen menos cuestiones al mudar entre entornos organizacionales totalmente diferentes. (Esa es la aterradora excusa, "En mi computadora portátil funcionaba", cuando el cambio de código se traslada de desarrollo a producción). Las máquinas virtuales o las instancias de nube pueden tener variaciones entre el sistema operativo subyacente y las versiones de los paquetes. Los contenedores deben tener configuraciones idénticas del sistema operativo y la imagen de contenedor seleccionada es la misma entre implementaciones.

Esa consistencia proporciona una buena base para la primera parte de CI/CD, la integración continua. Con la integración continua, los cambios de desarrollo se compilan constantemente y se crean con cada introducción, por eso los problemas son evidentes más rápidamente. Esta situación se combina con frecuencia con un conjunto de pruebas automatizadas para verificar la estabilidad o la funcionalidad. Este proceso continuo de pruebas (creadas) de introducción mantiene un código de mayor calidad.

Una vez recorrido el camino de la integración continua, su organización puede lograr la implementación continua y cambios en la producción con mayor rapidez. Esta velocidad beneficia tanto a desarrolladores como al personal de operaciones. Los desarrolladores y los líderes de la empresa tienen la satisfacción de ver el lanzamiento de nuevos productos al mercado más rápido. Las operaciones tienen la capacidad para implementar Common vulnerabilities and exposures (CVEs) críticas en un período de tiempo mucho menor y permiten un sistema más seguro y rendidor.

La palabra "continuo" puede tener significados un poco diferentes, según su ritmo de desarrollo y las necesidades de la empresa. Con un monolito majestuoso (procesos y tecnología sólidos con una arquitectura de aplicaciones más tradicional), un lanzamiento se puede realizar todas las semanas con una sola actualización monolítica, con solo los requerimientos para procesos de sprint ágiles que funcionan como una limitación.[29] Con los microservicios, cualquier servicio se puede actualizar con ciclos de sprint superpuestos, de tal manera que las actualizaciones de la arquitectura general se puedan realizar a diario.

Implementaciones avanzadas e innovación

Las etapas que conducen a la CI/CD tienen que ver con la velocidad y calidad con que se distribuyen las aplicaciones. Pero, tal como dice Kohavi, la mayoría de las ideas no logran llegar al destino deseado. Dio un ejemplo de esto, en su discurso de apertura, al proporcionar una serie de imágenes de diseños diferentes del motor de búsqueda de Bing y al pedirle al público que adivine, según sus instintos, qué versión prefieren los usuarios en realidad. Formuló cuatro preguntas consecutivas y descartó a cientos de personas del público; solo quedó de pie una sola persona.[30]

---

28 "The Emergence of Microservices as a New Architectural Approach to Building New Software Systems." en las series de INDUSTRY DEVELOPMENTS AND MODELS de IDC. Autor: Al Hilwa, junio de 2015.

29 Spazzoli, Raffaele. "The Fast-Moving Monolith: How We Sped-up Delivery from Every Three Months, to Every Week." Red Hat Developer's, 27 de octubre de 2016, https://developers.redhat.com/blog/2016/10/27/the-fast-moving-monolith-how-we-sped-up-delivery-from-every-three-months-to-every-week/.

30 "Online Controlled Experiments: Introduction, Insights, Scaling, and Humbling Statistics." SOBACO University of Minnesota, 18 de octubre de 2013, https://sobaco.umn.edu/content/online-controlled-experiments-introduction-insights-scaling-and-humbling-statistics.

Su intención era demostrar que los datos superan la intuición. El diseño de Bing terminó con extensivas pruebas para usuarios, y se descubrió que solo un tercio aproximadamente de sus ideas mejoraron la experiencia del usuario (y una cantidad similar la dañó seriamente).

Infraestructura de aplicaciones para experimentación

En el 2006, casi una década antes de que el término microservicios irrumpiera en la conciencia de los desarrolladores, el desarrollador Neal Ford definió un concepto llamado "programación políglota" en su blog Meme Agora.[31] Identificaba un cambio en los entornos de programación.

Muy al comienzo de las aplicaciones empresariales, las aplicaciones eran, verdaderamente, un lenguaje único. A medida que las aplicaciones comenzaron a cambiar hacia arquitecturas basadas en la web o cliente de servidor, había algunos lenguajes que eran específicos para ciertas acciones (como JavaScript para una interfaz de usuario web o SQL para las bases de datos), pero la aplicación básica aún se escribía en un lenguaje único y central.

La programación políglota de Ford expresa la idea de que no existirá más un lenguaje de programación central, pero en la arquitectura de aplicaciones general, los servicios o las funciones individuales podrán estar escritos en lenguajes totalmente diferentes que se adaptan mejor a las necesidades de ese servicio en particular.

La programación políglota es representativa de una de las necesidades básicas de un entorno de desarrollo ágil: los desarrolladores deben ser capaces de probar algo nuevo y que esté fuera del diseño básico de una aplicación. Esto es tan verdadero para las aplicaciones monolíticas efectivas como para las arquitecturas de microservicios.

Considerar la flexibilidad y las opciones en el entorno de desarrollo al crear una plataforma para experimentación es fundamental. Un entorno de experimentación debe dar soporte a lo siguiente:

- Lenguajes múltiples.
- Varios tiempos de ejecución.
- Entornos de implementación flexibles, como la nube (física o híbrida).
- Arquitecturas de aplicaciones intercambiables o flexibles. Esto permite que una aplicación se adapte a los cambios del entorno durante su ciclo de vida.
- Estándares abiertos o la capacidad de estandarizar de forma iterativa.

Los contenedores son un buen ejemplo de este soporte, si bien es posible lograr el mismo resultado con otras plataformas. Un contenedor individual es prescriptivo de forma inherente en relación con las bibliotecas, lo lenguajes y las versiones que contiene. Sin embargo, un catálogo de un contenedor puede tener cientos o miles de imágenes diferentes, y dar soporte a lenguajes y tiempos de ejecución distintos. Este tipo de plataforma permite a los desarrolladores encontrar el tiempo de ejecución exacto y el lenguaje que ejecutará el servicio de la manera deseada, y permite que los equipos operativos implementen efectivamente esos servicios.

Patrones de implementación para innovar

La técnicas de implementación avanzadas acercan la estructura y claridad a la innovación. Las metodologías de implementación desarrolladas crean un entorno que permite experimentación verdadera, retroalimentación y análisis. Una mejor experimentación conduce a una mayor innovación.

Estos son patrones de implementación comunes; cada uno o todos pueden ser apropiados, según la naturaleza de su aplicación y su entorno de usuario.

---

31 Ford, Neal. "Polyglot Programming." Meme Agora, 5 de diciembre de 2006, http://memeagora.blogspot.com/2006/12/polyglot-programming.html.

"Los datos superan la intuición".

RON KOHAVI, GERENTE GENERAL DE EXPERIMENTACIÓN DE INTELIGENCIA ARTIFICIAL, MICROSOFT UNIVERSIDAD DE MINNESOTA, DEPARTAMENTO DE CIENCIAS INFORMÁTICAS Y TECNOLOGÍA DE INGENIERÍAS, OPEN HOUSE 2013 [23]

Entornos azul-verdes

Un entorno azul-verde es una manera de mitigar el riesgo de implementar cambios. Una nueva creación pasa por todos los entornos del canal de CI/CD. Para producción, existen dos entornos idénticos (azul y verde), pero solo uno está activo. El cambio se implementa en el entorno inactivo en producción; una vez que se verifica el entorno, se cambia el router y el tráfico se traslada al entorno actualizado.

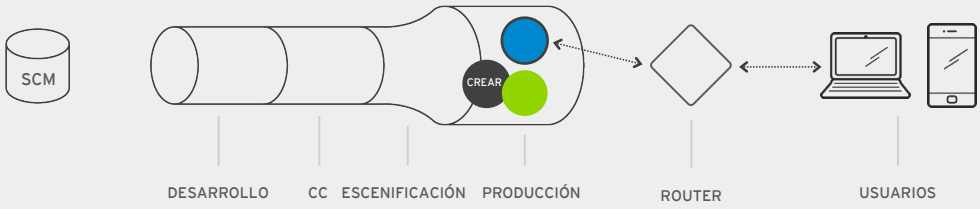


Figura 9. Canal de implementación azul-verde

Lanzamientos "canary"

Un lanzamiento "canary" es similar a una implementación azul-verde, excepto que el lanzamiento inicial solo es para un subconjunto de usuarios en el entorno (los "canarios" titulares en la mina de carbón). Mientras se recopilan los comentarios de los usuarios, ese subconjunto se puede incrementar progresivamente hasta que, eventualmente, se cambien todos los usuarios.

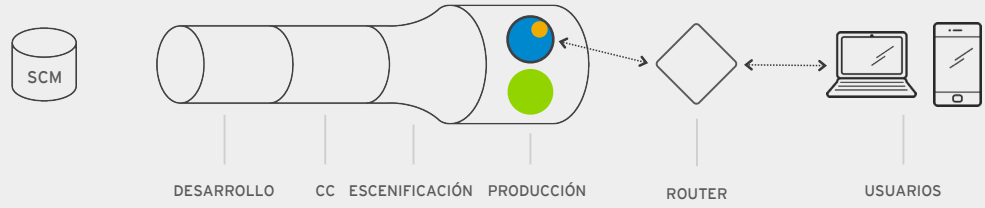


Figura 10. Canal de implementación para evitar el desbordamiento de buffer (Canary)

Esto se puede usar como parte de una técnica de pruebas para evaluar diferentes funcionalidades o diseños para aplicaciones en grupos pequeños en un entorno de producción activo, y con tráfico real y patrones de uso.

Pruebas A/B

Las pruebas A/B presentan usuarios con dos diseños diferentes y luego evalúa qué diseño funciona mejor, según las métricas deseadas. Esto puede ser tan explícito como tener usuarios que midan sus experiencias o proporcionen comentarios, pero también puede hacerse de forma más sutil. Por ejemplo, combinar pruebas A/B con lanzamientos "canary" se puede comparar con dos diseños potenciales, o incluso con funcionalidad escondida, y luego se evalúa cómo los usuarios interactúan con los diferentes diseños, con el entorno actual como punto de referencia.

Por ejemplo, en la figura 11, la aplicación móvil parece idéntica a los usuarios, pero los entornos A/B usan diferentes algoritmos para probar las recomendaciones del producto.

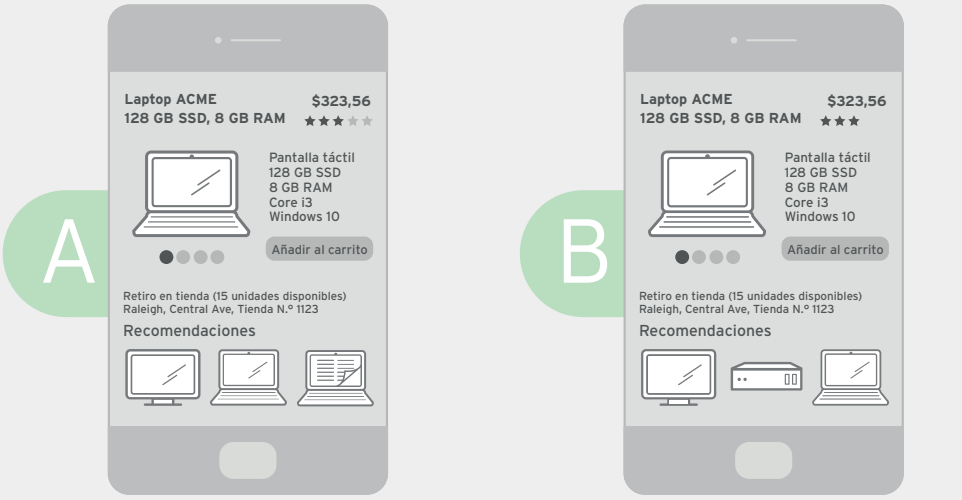


Figura 11. Pruebas de interfaz A/B

Una vez que el diseño dado es exitoso, entonces se puede lanzar a un conjunto más grande del entorno, como con un lanzamiento "canary".

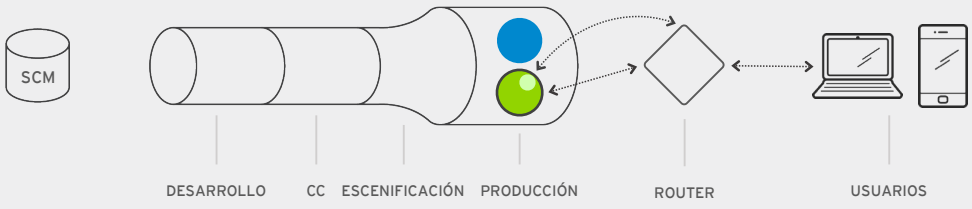


Figura 12. Canal de implementación de pruebas de interfaz A/B

Cuando se hace de forma efectiva, esto puede convertir un entorno de producción en un entorno de experimentación, lo que permite a los equipos crear diseños más innovadores y relevantes.

Este tipo de pruebas es lo principal de valorar los datos por sobre la intuición.

CÓMO ENSEÑARLE A BAILAR A UN ELEFANTE

Elija su etapa

Existen muchas etapas entre un entorno de cascada más tradicional y los microservicios totalmente distribuidos. Laycock habló sobre toda la arquitectura de software como el resultado del "conflicto entre la conexión y la cohesión".[32] Mientras comienza con la planificación de una estrategia de transformación digital, ese conflicto se representa en la infraestructura y la cultura que usted tiene actualmente.

Determine hacia qué dirección puede ir realmente su organización. Esto no significa que lo hará "fácilmente", ya que el objetivo de la transformación digital es cambiar significativamente la cultura, los procesos, la arquitectura y la tecnología. Significa comprender lo que intenta lograr con ese cambio y luego evaluar claramente lo que se necesitaría para avanzar hacia esa meta. Pregúntese a sí mismo:

- ¿Cuál es su equipo actual o división de grupo?
- ¿Cuáles son los patrones de comunicación entre esos grupos?
- ¿Quién está involucrado actualmente en los ciclos de planificación?
- Al observar la funcionalidad, ¿qué tan cerca se encuentra su arquitectura de aplicaciones a la arquitectura de aplicaciones deseada?
- ¿Cuál es el nivel de riesgos o tolerancia a los fallos en su organización?
- ¿Qué tan bien se comprenden sus flujos de materiales e información? (Esto es realizar un mapa de valor de su organización).
- ¿Con qué frecuencia necesita poder lanzar una actualización para satisfacer las necesidades operativas o del cliente?
- ¿Qué funcionalidad nueva es requerida por los objetivos empresariales o por las necesidades de desarrollo?

Definir sus principios operativos

Los cambios culturales sustentan todos los cambios del proceso, la tecnología o la arquitectura que su organización hará para la transformación digital.

Si bien es básico, crear un conjunto de principios fundamentales que sean respaldados por la gerencia y apoyados por los equipos puede ayudar a reforzar las iniciativas de transformación digital y unificar los equipos.

---

32 Laycock, Rachel. "Continuous Delivery." Sesión vespertina. Red Hat Summit - DevNation 2016, 1.º de julio de 2016, San Francisco, California. https://www.youtube.com/watch?v=y87SUSOfgTY

"Una sólida arquitectura de aplicaciones, prácticas de DevOps consolidadas y procesos ágiles, junto con un equipo de gestión de datos centralizado, crean un entorno de microservicios eficaz. Este entorno puede reducir el tiempo de implementación en un 75%".

– GARTNER[33]

Estos principios pueden ser simples, pero son útiles para ser claros y explícitos en cuanto a las actitudes y conductas más importantes para el cambio cultural, especialmente, si algunos de estos —como incentivar el riesgo y la experimentación— son contrarios a la cultura actual. Por ejemplo:

- Las fallas suceden.
- Experimentar es bueno.
- La organización (las personas) está primero.
- Practicar la mejora continua.
- Comprometerse a un aprendizaje permanente.
- Ser siempre confiable.
- Ser transparente.

Figura 13. Menos esfuerzos para un lanzamiento bien orquestado

33 *Innovation Insights for Microservices , Anne Thomas y Aashish Gupta, 27 de enero de 2017*

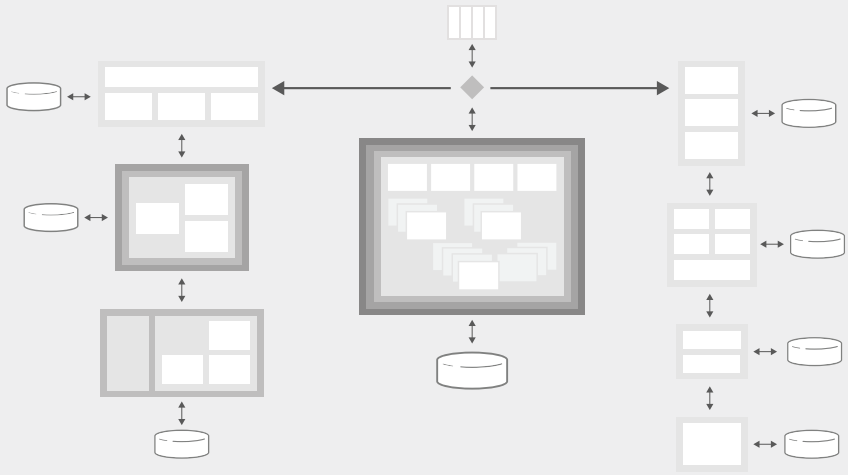


Figura 14. Desglosar los servicios individuales para "estrangular" el monolito

Estrangular su monolito

En algún momento, la transformación digital afectará su aplicación y arquitectura actuales. Si actualmente tiene una aplicación monolítica, existen dos maneras relacionadas para comenzar a abordar esa deuda técnica:

- Desglosar los servicios existentes que requieren actualizaciones o ser reemplazados (estrangulamiento).
- Crear nuevas funciones en servicios separados e independientes (carecimiento).

Esto no requiere un compromiso total con los microservicios. Key Bank, cliente de Red Hat, necesitaba reducir los tiempos de lanzamiento de un trimestre a una semana, y lo logró mientras aún tenía una aplicación monolítica.[34] La lógica principal de las aplicaciones puede permanecer en un monolito, pero puede haber dominios lógicos en los que los servicios se pueden separar, más frecuentemente para las interfaces de usuario. Por ejemplo, crear una capa separada para los servicios, como un sistema API, una interfaz móvil y otras interfaces de usuario, permite que los servicios orientados externamente o al cliente tengan iteraciones más rápidas o ciclos de vida más separados que la aplicación principal, y esto posibilita contar con más innovación y menos riesgo empresarial.

Gartner recomienda este enfoque gradual de las arquitecturas distribuidas, ya que es al mismo tiempo "iterativo y centrado en áreas de valor".[35]

Cualquiera sea la etapa final de la evolución digital para su organización, hay tres áreas básicas que deben cubrirse en su enfoque:

- Agilidad en el diseño de arquitectura.
- Experimentación.
- Automatización.

---

34 Spazzoli, Raffaele. "The Fast-Moving Monolith: How We Sped-up Delivery from Every Three Months, to Every Week." Red Hat Developers, 27 de octubre de 2016, developers.redhat.com/blog/2016/10/27/the-fast-moving-monolith-how-we-sped-up-delivery-from-every-three-months-to-every-week/.

35 Olliffe, Gary. "How to Design Microservices for Agile Architecture." Perspectivas claves de Gartner, 30 de enero de 2017.

Diseñar arquitecturas para una agilidad futura

Ya sea que su enfoque se centre en los procesos optimizados para mejorar su monolito o en la creación de microservicios, su base de arquitectura tiene que ser ágil. Con frecuencia, la agilidad significa ser híbrido. Gartner recomienda iniciar los proyectos (incluso los nuevos) como monolitos y lanzar los microservicios a medida que se desarrollan.[36] Gartner aclara que "su reacción inicial a esta situación podría ser considerada como una pérdida de esfuerzos de desarrollo. En resumen, nuestra investigación sugiere lo contrario: el enfoque primario en el monolito reducirá riesgos, mejorará la productividad inicial y asegurará que su aplicación se desconecte y descomponga en el conjunto correcto de microservicios".[37]

Con la teoría de la resistencia al diseño en mente, se debe crear un proceso de desarrollo que enfatice la claridad y la simplicidad. El código debe ser fácil de entender. La funcionalidad y el propósito deben ser claros. Mientras la aplicación se desarrolla, puede evolucionar en arquitecturas más distribuidas. Contar con un buen desarrollo y procesos de implementación adecuados mantendrá el camino ágil.

Deje a un lado el tiempo y el presupuesto para la experimentación

Debe haber un lugar en el presupuesto para la experimentación con nuevas tecnologías y funciones de las aplicaciones. Por ejemplo, IDC recomienda dejar de lado el 2% de su presupuesto que tiene para la TI solo para experimentar con tecnologías de contenedores.[38]

Gartner afirma que la tecnología no es un objetivo en sí misma, por lo que intentar un cambio para "implementar en la nube" o "hacer microservicios" fracasará debido a la falta de claridad en lo que se supone que esos cambios deben lograr.[39]

Sin embargo, los objetivos estratégicos digitales suelen estar respaldados por los cambios tecnológicos. Reducir el tiempo de comercialización puede requerir el translado a contenedores (por ejemplo), y las aplicaciones JavaTM EE pueden ejecutarse en plataformas de contenedores, como Red Hat JBoss® Enterprise Application Platform (EAP).

Separe los recursos que permiten a sus desarrolladores y grupos operativos identificar las tecnologías útiles y desarrollar sus habilidades para dar soporte a cualquier infraestructura que se implemente en última instancia.

Automatizar todo

La automatización tiene dos (entre otros) beneficios muy claros: la eficiencia mejorada mediante la eliminación de pasos manuales, y la consistencia y la reproducibilidad inherentes. Automatizar cada paso para el desarrollo (para empezar) y la implementación (a medida que los procesos se desarrollan) le dará a sus equipos bucles de retroalimentación acerca de los cambios en cada paso, mientras las partes interesadas cambian del desarrollo a las operaciones para los clientes. Este enfoque mejora la calidad general del código.

Como primer paso, defina una base de referencia para el estado actual de su organización como punto de partida para su estrategia de automatización. Los pasos incluyen los siguientes:

- Definir métricas relevantes.
- Visualizar o diagramar sus flujos de trabajo actuales.
- Identificar participantes clave en etapas diferentes.

---

36 Ibid.

37 Ibid.

38 Elliot, Stephanie, et al. "IDC TechBrief: Containers." IDC. Enero de 2017.

39 Knoernschild, Kirk. "Refactor Monolithic Software to Maximize Architectural and Delivery Agility." Perspectivas claves de Gartner, 18 de mayo de 2017.

CONCLUSIÓN

Con el tiempo, las aplicaciones empresariales tienden a recaer en el monolito estereotípico: opaco, complicado para actualizar y lento para incorporar nuevas funciones. Sin embargo, estas aplicaciones empresariales también proporcionan las operaciones que generan ingresos y que son fundamentales para la empresa. Este es el elefante en la habitación.

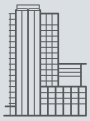
Ese elefante se puede entrenar para ser ágil y transformador, siempre que exista una visión clara de lo que debe ser ese estado final. Esta es la transformación digital entendida como un proceso evolutivo. No hay un resultado ideal; cada camino hacia la evolución refleja el propósito único y la personalidad de la misma organización.

Evalúe cada etapa de la evolución digital: DevOps, entornos flexibles o de autoservicios, automatización, canales de CI/CD, implementaciones avanzadas y microservicios. Diseñe su estrategia de transformación digital sobre el nivel de evolución que mejor se adapte a las necesidades de su empresa.

Concéntrese en construir su cultura y equilibrar los cambios tecnológicos con los correspondientes cambios de procesos para que su tecnología tenga el soporte total de sus equipos.

A medida que los procesos se desarrollan, comience a evaluar su aplicación y arquitectura. Aisle o desarrolle servicios independientes como sea necesario y cree una arquitectura ágil que se pueda adaptar al cambio o al surgimiento de las prioridades empresariales.

Por último, promueva una habilidad para innovar. Esto significa tener un poco de tolerancia al riesgo y los fallos (según los límites de sus objetivos empresariales y las necesidades del cliente). Requiere disciplina para apartar los recursos de tiempo, dinero e infraestructura. La experimentación está en la raíz de la innovación y establece una mejor opción para lograr el éxito de la transformación digital. También recaptura parte de la alegría inicial que atrajo a tantos desarrolladores y personal de operaciones a la tecnología en primer lugar: la habilidad de crear y ver esa creación.



ACERCA DE RED HAT, INC.

Red Hat es el proveedor líder mundial de soluciones open source empresarial, con un enfoque impulsado por la comunidad para la obtención de tecnologías cloud, Linux, middleware, almacenamiento y virtualización de alta fiabilidad y rendimiento. Red Hat también ofrece servicios de soporte, formación y consultoría. Como eje central de una red global de empresas, partners y comunidades open source, Red Hat ayuda a crear tecnologías competentes e innovadoras que liberan recursos para el crecimiento y preparación de los consumidores para el futuro de las TI. Conozca más en http://es.redhat.com.

ARGENTINA
Ingeniero Butty 240, 14º piso
Ciudad de Buenos Aires
Argentina
+54 11 4329 7300

CHILE
Avda. Apoquindo Nº 2827
oficina 701, Piso 7
Los Condes, Santiago, Chile
+562 2597 7000

COLOMBIA
Red Hat Colombia S.A.S
Cra 9 No. 115-06 Piso 19 Of 1906
Edificio Tierra Firme Bogota, Colombia
+571 5088631
+52 55 8851 6400

MÉXICO
Calle Río Lerma 232
Cuauhtémoc
06500 Ciudad de México
Mexico
+52 55 8851 6400

ESPAÑA
Torre de Cristal
Paseo de la Castellana 259C
Piso 17 Norte
28046 Madrid
+34 914148800

facebook.com/redhatinc
@RedHatIberia
Red Hat EMEA

es.redhat.com
#8980_0917