Informe de investigación

Temática: Arquitectura de Microservicios

Área: Tecnología

Responsables: Juan Ortellado

Índice

1. Objetivo de investigación

El uso de arquitecturas de microservicios ha crecido considerablemente en los últimos años debido a la necesidad de la industria de optimizar costos y agilizar procesos. Esto también está acompañado de avances tecnológicos y metodologías de trabajo que se acoplan muy bien con este tipo de arquitecturas.

El propósito de esta investigación es el de identificar las distintas características que se encuentran en las arquitecturas de microservicios, cuándo es conveniente o no aplicarlo y cómo hacerlo.

En este proceso de investigación, se identifican una multitud de fuentes de información al respecto, cada una de ellas con un enfoque diferente. Es así que surge la inquietud por unificar los criterios para su utilización, basándose principalmente en la investigación realizada por el Open Group. [1]

2. Introducción

En primer lugar comencemos identificando qué es una arquitectura. Open Group define la arquitectura como la estructura de componentes, sus relaciones y las guías y principios que gobiernan su diseño y evolución. Por otra parte, define el estilo arquitectónico como la combinación de características distintivas en las que se expresa una arquitectura.

En los comienzos del desarrollo de software, los sistemas seguían un estilo arquitectónico monolítico y a la actualidad todavía existen algunos sistemas que siguen este tipo de arquitectura. La arquitectura monolítica es similar a un gran contenedor con todos los componentes de software de una aplicación, que son ensamblados y empacados juntos. A modo de ejemplo, una arquitectura monolítica sigue un esquema similar al de la figura 1.

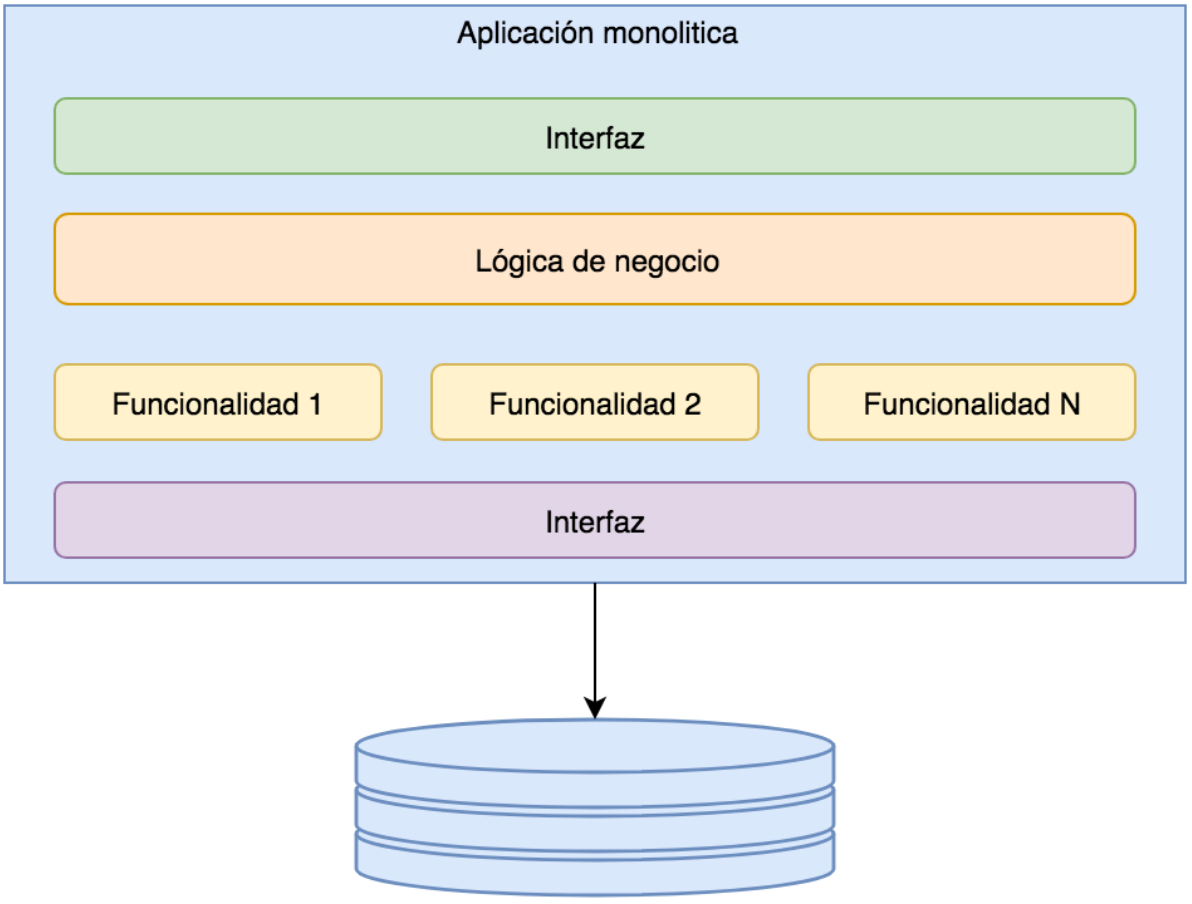

Figura 1. Arquitectura monolítica

Este tipo de arquitecturas presentan desafíos ante determinadas situaciones, por ejemplo al momento de actualizar una funcionalidad, o tienen fuertes dependencias con la tecnología sobre la que trabajan, entre otras cosas.

Más cercano en el tiempo, es posible encontrarse con arquitecturas de microservicios, para las cuales se define un microservicio como un servicio implementado con un único propósito, que es autocontenido y que es independiente de otras instancias de servicios. De acuerdo a esta definición, podemos identificar la arquitectura de microservicios con alguno de los esquemas definidos en las figuras 2 y 3.

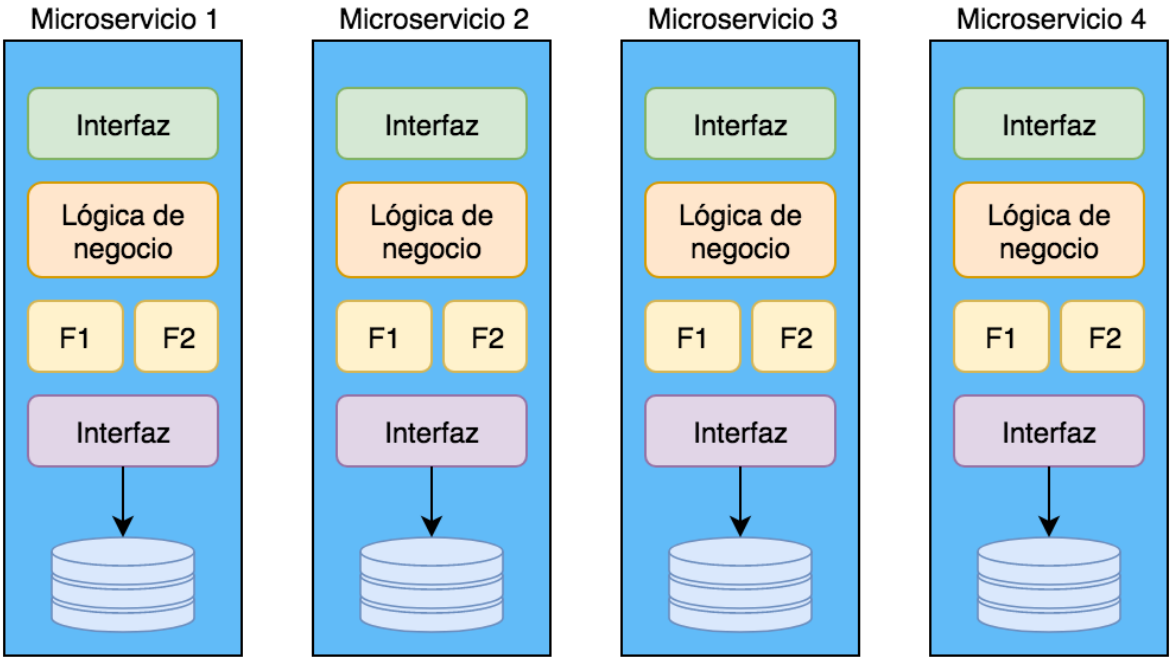

Figura 2. Microservicios con bases de datos independientes

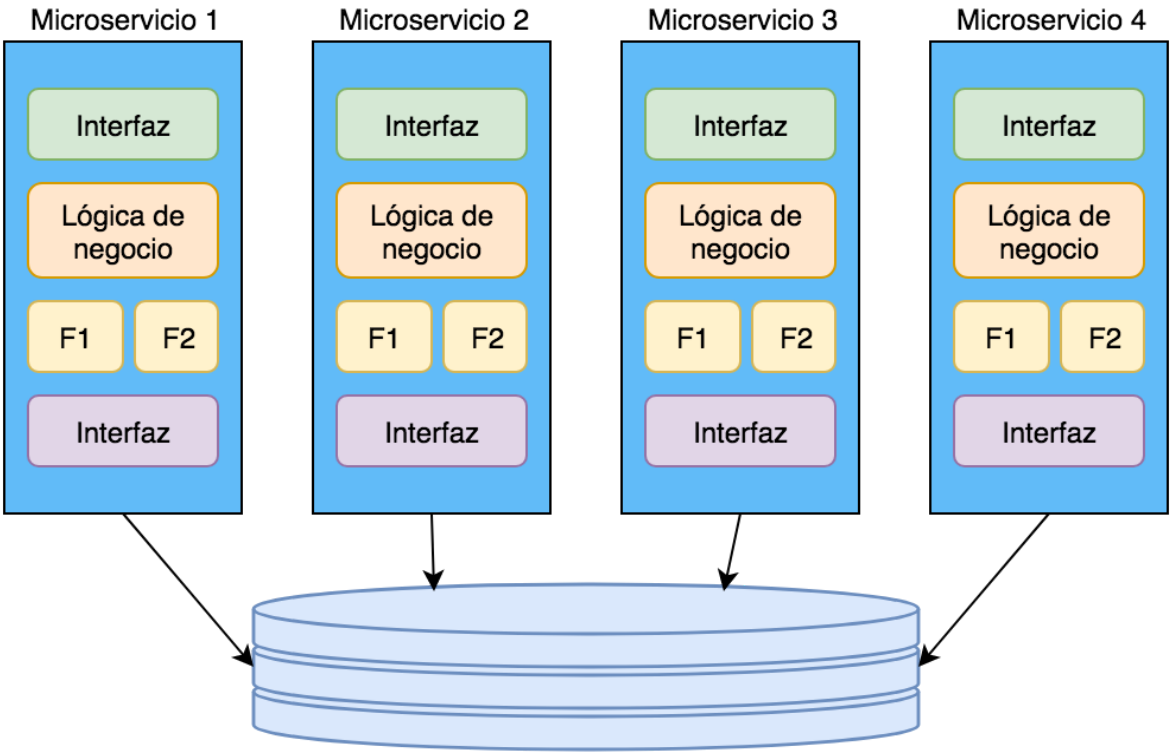

Figura 3. Microservicios con base de datos compartida

Claramente la diferencia entre una y otra es el manejo de la base de datos, pero de momento es preferible no entrar en esos detalles. El principio de este estilo arquitectónico se basa en definir y crear sistemas a través de pequeños servicios independientes y auto-contenidos, que se alinean a los propósitos de ese sistema. Los microservicios son el insumo primordial de una arquitectura de microservicios.

Luego de presentar brevemente estas arquitecturas, parece necesario entender qué relación hay entre ellas. Esta explicación[7] tiene origen en los 80s, donde obviamente no existían la tecnologías que se tienen hoy y por tanto las formas de desarrollar y operar las soluciones eran completamente diferentes. En esa época empezaban a aparecer necesidades de distribución debido a la necesidad de escalabilidad de procesamiento y memoria. Esto generó una sobrecarga mayor sobre las aplicaciones que los beneficios que aportaba la distribución. Por este motivo, se aplicaron patrones de diseño que ayudaron a mejorar el funcionamiento y dado que se estaba muy atado a cada tecnología, fueron creados protocolos para interoperar fácilmente como por ejemplo SOAP en el año 1999 o REST en el 2000. Luego apareció la noción de un servidor de aplicaciones como un host para diferentes aplicaciones, donde un solo grupo de operaciones podría controlar, monitorear y mantener una "granja" de servidores de aplicaciones. Esto creó conflictos con los desarrolladores de aplicaciones, debido a que los entornos de prueba y desarrollo eran grandes, difíciles de crear, y requerían de la participación del equipo de operaciones. A menudo significaba que podía tomar meses para que se crearan los nuevos entornos, ralentizando proyectos e incrementando sus costos de desarrollo. Pronto, los equipos de desarrollo descubrieron que la obtención de la capacidad para generar e

implementar sistemáticamente sus propias aplicaciones, en entornos de desarrollo, prueba y producción que estaban más cercanos unos de los otros, no sólo era más rápida, sino menos propensa al error. Esto condujo a la siguiente observación, que acerca a una definición de microservicios:

"Siempre que sea posible, sus programas y sus entornos de ejecución deben estar totalmente autocontenidos."

Ahora bien, esta evolución vista a lo largo de la historia respecto a las arquitecturas, puede verse identificada en la ley de Conway[6], que indica a grandes rasgos que cualquier organización diseña sus sistemas con una estructura similar a la estructura de la misma organización. Esto quiere decir que si en una organización se tiene por ejemplo un equipo de interfaz de usuario, otro de bases de datos y otro para la lógica de negocio, probablemente el sistema diseñado siga una arquitectura en capas, dividido de forma muy similar a la de esa organización. Esto puede verse reflejado en la Figura 4.

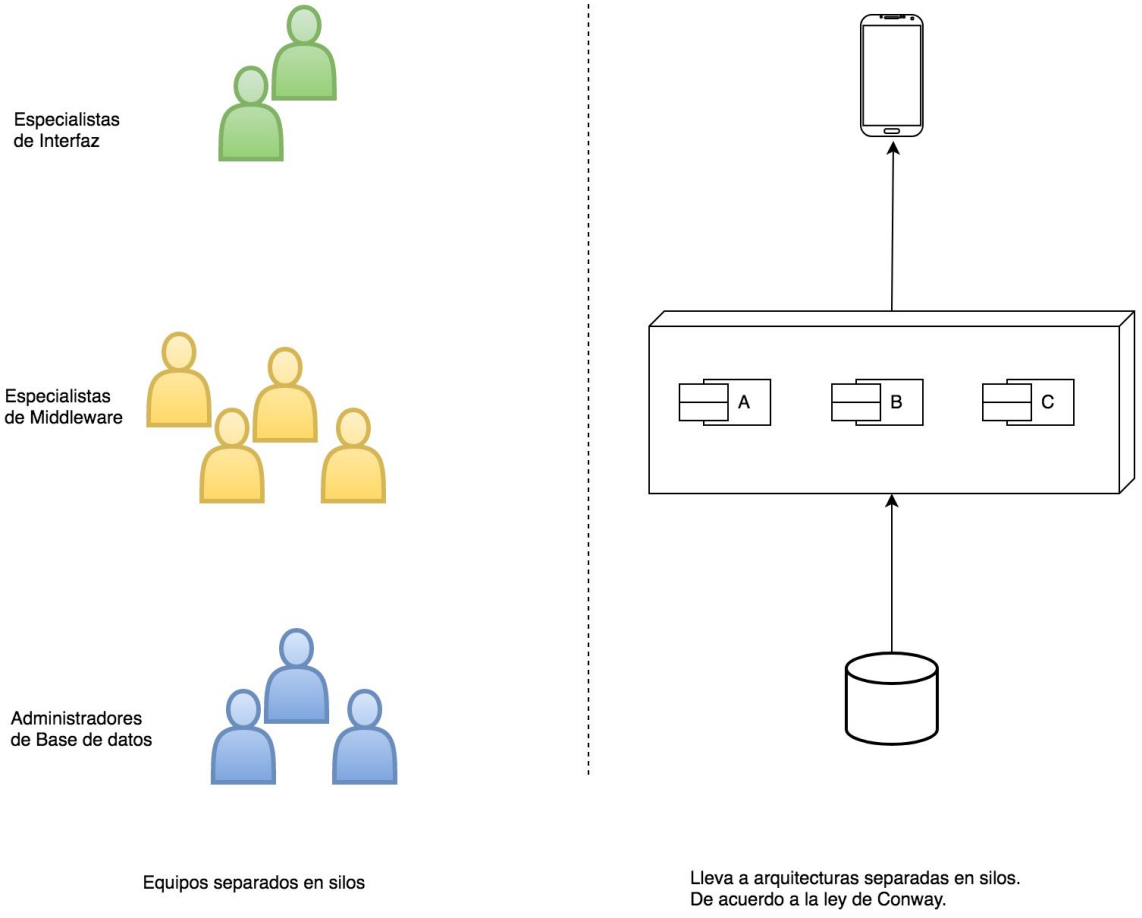

Figura 4. Equipos en silos, arquitecturas en silos.

Entendiendo la problemática planteada, se puede realizar una reestructura en los equipos de trabajo, provocando un cambio no solo en el diseño y desarrollo del sistema, sino también en la forma de trabajo en la organización.

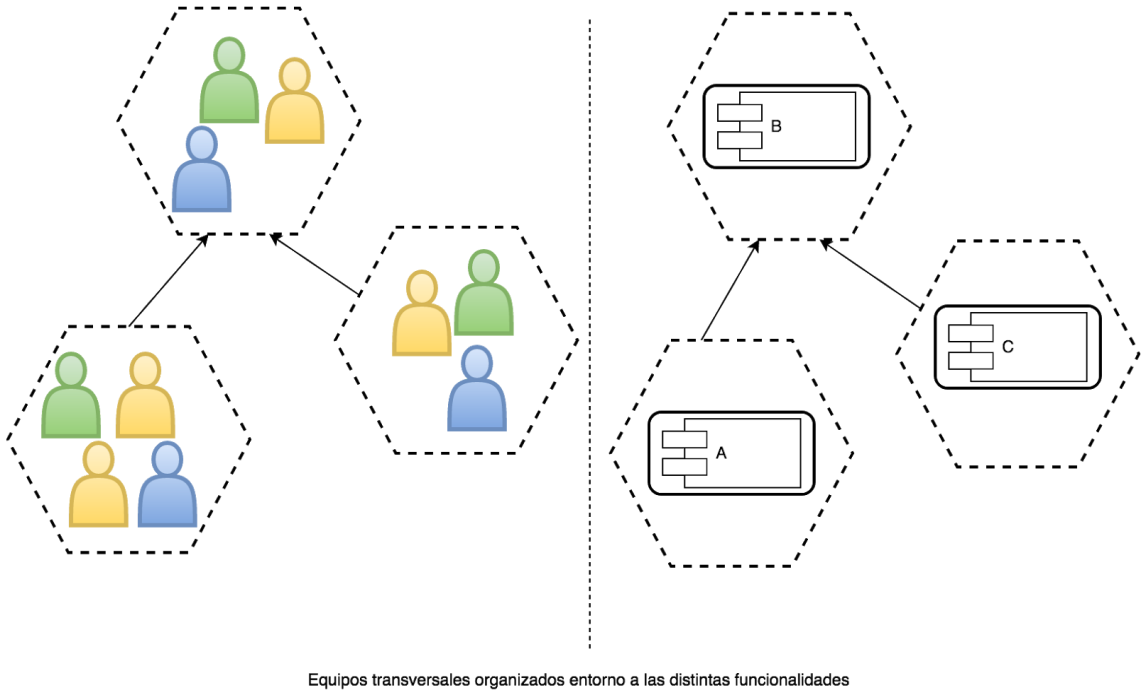

Figura 5. Cambio en los equipos de acuerdo a funcionalidades

3. Características principales

Para la creación de este tipo de arquitecturas, los componentes de software son fraccionados en servicios independientes (característica fundamental de una arquitectura de microservicios), con bajo acoplamiento y cada uno de ellos desplegado de forma independiente. La escalabilidad y resiliencia de la aplicación es lograda a través de la independencia de los servicios y las múltiples instancias funcionando en paralelo.

Para identificar a los microservicios, no existe una forma específica o pasos a seguir que deriven en la fragmentación correcta. Es importante en este proceso tener en cuenta el nivel de granularidad al que se llega al momento de identificar un microservicio, ya que esto es vital para el ciclo de vida de esos microservicios, así como para el funcionamiento de la aplicación en sí. Más adelante en el documento se profundizará en este aspecto.

Elasticidad

Cada microservicio puede necesitar escalar ya sea de forma horizontal o vertical. La forma de escalar horizontalmente consiste en clonar el microservicio de forma que otro servicio más pueda atender la carga. Ésta es por lo general la más utilizada y es recomendable contar con algún mecanismo que permita fácilmente a los servicios

escalar basados en su contexto. Una plataforma que ofrezca estas características puede simplificar mucho esta funcionalidad.

Dependencias

En entornos donde múltiples servicios deben colaborar para proveer una funcionalidad, su resolución mediante una arquitectura de microservicios puede ser una tarea difícil y más si tenemos en cuenta que cada uno de esos microservicios puede escalar de distinta forma. Por tal motivo es recomendable implementar un servicio de descubrimiento (service discovery), con el objetivo de que cada microservicio de la solución se registre contra él y éste luego pueda informar donde encontrar a cada dependencia.

Monitoreo

Uno de los aspectos más importantes de un sistema distribuido es el monitoreo y logging de forma de poder tomar acciones proactivamente, por ejemplo si un servicio está consumiendo recursos innecesariamente. El análisis de un bug puede volverse una tarea altamente difícil de llevar a cabo. En esto influye la granularidad definida para cada microservicio y también la forma en que se genera el log. Es altamente recomendable mantener el log centralizado y algún tablero con información del sistema que asista a ese análisis.

Resiliencia

Las fallas en el software existen sin importar que tanto se pruebe. La clave en este punto no es cómo evitar la falla sino cómo lidiar con ella. Esto aumenta con microservicios distribuidos en distintos sitios. Es importante que los microservicios puedan tomar acciones correctivas automáticamente asegurando que la experiencia de usuario no se vea impactada.

Reutilización

Un aspecto muy útil que presenta este estilo arquitectónico es que de acuerdo a la fragmentación que se realice de los microservicios, es posible que algunos de estos sean reutilizables en otros contextos. Probablemente esta adaptación no sea trivial, pero sin dudas en comparación con la implementación de esta funcionalidad, implica un gran ahorro.

DevOps

Continuous integration y Continuous delivery o deployment (CI/CD) es una práctica muy importante para que aplicaciones basadas en microservicios aumenten las probabilidades de triunfar. Esto es importante para que cualquier tipo de error pueda ser identificado en una etapa temprana del desarrollo y no se requiera la coordinación entre los distintos equipos de desarrollo y operaciones. Se puede obtener más información en el Anexo DevOps - CI/CD

Componentes, gobernanza y su interacción

Una de las principales características de esta arquitectura es que los componentes son desarrollados como servicios en lugar de, por ejemplo librerías, y típicamente implementados utilizando interfaces bien definidas. Este punto es de vital importancia ya que de no hacerlo así, ante un cambio en una interfaz, se pueden afectar otros servicios. Si bien se entiende que en algunos casos esto no se puede lograr, se alienta a que la definición de los microservicios minimice el impacto sobre las interfaces.

Las aplicaciones desarrolladas sobre microservicios deberán mantenerse desacopladas y totalmente independientes de ellos y cualquier orquestación que se deba realizar entre los microservicios, deberá ser realizada desde la aplicación misma y no desde uno de los microservicios, ya que se busca mantener un protocolo de interacción simple, que no tenga un microservicio central.

La gobernanza sobre este tipo de arquitectura es descentralizada o distribuida, permitiendo a los equipos enfocarse en determinadas plataformas o lenguajes de implementación para solucionar problemas particulares.

Granularidad

Determinar la granularidad correcta de los servicios podría considerarse como un arte, en lugar de un procedimiento científico, dado que no necesariamente hay una única solución. De cualquier manera, parece necesario definir algunas reglas para aproximarse al nivel de granularidad correcto. Si se define una granularidad baja, se termina definiendo una arquitectura monolítica, mientras que si es una granularidad alta, la arquitectura puede resultar en el anti-patrón conocido como nanoservicios. Por lo general se tiende a separar vinculando un microservicio con una capacidad de negocio puntual, aunque esto no siempre es así y puede llevar a problemas al momento de definirlos.

Open Group plantea la posibilidad de definir una línea de granularidad basada en las actividades de negocio, la capacidad del negocio a adaptarse a un cambio y otras consideraciones. Los servicios que se encuentren cercanos a esa línea son considerados microservicios, los que se encuentren muy por encima son considerados monolíticos y los que se encuentran muy por debajo son considerados nanoservicios.

4. Beneficios y contras

El estilo arquitectónico de microservicios presenta una serie de beneficios que viene dada por su propia estructura. En primer lugar podemos encontrar que la complejidad del desarrollo se ve disminuida dado que se termina trabajando sobre piezas individuales e independientes. Adicionalmente, ante una necesidad de un incremento o modificación de funcionalidades, se convierte en un trabajo mucho más simple, dado que esas funcionalidades se encuentran claramente identificadas y con bajo acoplamiento respecto a otros microservicios. En relación a lo mencionado anteriormente, permite una rápida respuesta ante cambios o entrada en funcionamiento en el mercado.

La arquitectura de microservicios permite que la tecnología con la que se desarrolla cada microservicio sea independiente del resto de las implementaciones de los otros microservicios dentro de un mismo sistema. Esto permite además, que la instalación de cada uno de ellos se pueda realizar de forma independiente y que cada uno de ellos escale de forma independiente, según las necesidades y restricciones de cada microservicio. Más adelante se verán algunas estrategias para la instalación que ayudan a una rápida respuesta y por qué la arquitectura de microservicios es útil en ese contexto.

Enfocándose en metodologías de trabajo, como se mencionaba en la sección anterior, se puede encontrar que este tipo de arquitectura funciona de gran manera con paradigmas como la Integración Continua (CI) y Despliegue Continuo (CD).

Potencialmente permite distribuir la carga de trabajo elásticamente ajustándose por microservicio a la demanda en cada momento. Lograr este tipo de comportamientos tiene un alto impacto sobre la experiencia de usuario.

Por otra parte, este tipo de arquitecturas presenta varios desafíos ante distintas problemáticas. La distribución de la propia arquitectura hace que identificar un error o un mal comportamiento sea un poco más difícil de lo normal y como se comentaba en la sección anterior es recomendable tener un buen mecanismo de monitoreo y logging para que la problemática disminuya.

Adicionalmente, en casos en los que se tienen dependencias se pueden agregar retardos entre llamadas que pueden influir en los tiempos de respuesta generales. Es recomendable analizar y realizar pruebas sobre las funcionalidades para minimizar estos retardos y que sean despreciables para el usuario final.

En cuanto a la configuración y administración, puede volverse un tanto complejo ya que se pasan a tener varios componentes individuales que deben ser configurados y operados también de forma individual, con el objetivo de que todo el sistema brinde las funcionalidades correctamente. Es recomendable tener una buena percepción del funcionamiento de los componentes. En caso de utilizar muchos ambientes, es bueno

tener presente que el mantenimiento de las configuraciones para cada uno de ellos puede volverse una tarea complicada por momentos.

En contextos donde se intercambien datos entre distintos servicios, surgen problemas de transaccionalidad, por lo que es recomendable ser cauteloso y tomar medidas para controlarlo. Esto en definitiva termina complejizando el desarrollo de cada servicio.

Si bien es posible desarrollar microservicios sin automatizaciones, la carga extra de trabajo generada para la construcción, despliegue, testeo, etc., puede ser muy alta, por lo que es recomendable automatizar muchas de estas tareas. La automatización de estos procesos para cada componentes no es una tarea simple, pero luego de confeccionada evita mucho trabajo extra.

A continuación se presenta una tabla que resume los beneficios y contras antes planteadas.

Beneficios de arquitecturas de microservicios	Contras de arquitecturas de microservicios
Libertad para usar diferentes tecnologías, incluso dentro del mismo sistema.	Se incrementa la dificultad para resolver problemas.
Cada microservicio se enfoca en una sola funcionalidad.	Se incrementan los retardos debido a las llamadas remotas.
Desarrollo y despliegue de unidades individuales.	Aumentan los esfuerzos para la configuración y operación.
Permite actualizaciones con un alto nivel de periodicidad.	Se complejiza la integridad transaccional.
Elasticidad vertical y horizontal por microservicio	Difícil para rastrear datos a lo largo de todo el sistema.

5. Estrategias de despliegue

Canary releasing

La estrategia Canary Releasing es una técnica utilizada para reducir el riesgo de introducir una nueva versión con errores en un ambiente productivo. Este nombre es atribuido por la técnica utilizada por los mineros hace mucho tiempo, cuando no tenían equipamiento para detectar gases tóxicos. La idea consistía en llevar un canario a las cuevas donde trabajaban, ya que estos son altamente susceptibles a los gases tóxicos. Si el canario se moría, entonces los mineros debían evacuar la cueva. Esta misma estrategia, y sin poner en juego la vida de ningún canario, es la que sigue Canary Releasing y consiste en aumentar gradualmente el tráfico a través de la nueva versión que se está instalando, mientras se monitorea ante un eventual problema. De esta forma, nuestro "canario" es la pequeña porción de tráfico inicial que hacemos pasar por la nueva versión de software y si en cualquier momento algo falla, se le quita el tráfico, ya que es altamente probable que la nueva versión provoque un error. Si por el contrario se obtiene un buen resultado, la estrategia es continuar aumentando el tráfico hasta que la nueva versión sea la única funcionando.

Las arquitecturas de microservicios tienen una estructura muy alineada con este tipo de estrategias, ya que tienen un nivel de granularidad que se ajusta al procedimiento, es decir, podemos considerar un microservicio como ese "canario" y por tanto aprovechar esta estructura para la puesta en el ambiente productivo y así analizar el comportamiento.

Blue-green releasing

La técnica de despliegue blue-green reduce el riesgo y el tiempo de indisponibilidad entre dos versiones de software que se identifican como "Blue" y "Green", permitiendo que en cualquier momento se pueda dejar funcionando únicamente uno de los ambientes, por ejemplo el "Blue".

A medida que se desarrolla el nuevo software, se realizan los despliegues y todas las pruebas necesarias sobre el ambiente que no se encuentra activo, continuando con el ejemplo, se realizaría sobre el ambiente "Green". Una vez que se determina que ese software cumple los requisitos, lo que se hace es cambiar el tráfico para que apunte a ese nuevo ambiente, el "Green" en lugar del "Blue". De esta forma se deja funcionando el nuevo software y en caso de fallas o un comportamiento inusual, se puede volver inmediatamente a la última versión, siguiendo el ejemplo el ambiente "Blue".

Esta técnica permite que rápidamente se puedan llevar a ambientes productivos pequeñas funcionalidades, representadas por cada microservicio. También es necesario tener en cuenta que esto es viable siempre que se tengan las herramientas para poder controlar el flujo del tráfico de acuerdo a las funcionalidades.

6. Conclusiones

El estilo de arquitectura de microservicios, es un enfoque para desarrollar una sola aplicación como un conjunto de pequeños servicios, cada uno ejecutándose en su propio proceso y comunicándose a través de distintos mecanismos. Esta independencia entre los servicios, minimiza el impacto identificado y aísla a los servicios que se encuentran en constante cambio del resto de la solución. El desarrollo y despliegue de un microservicio es independiente de cualquier otro servicio o microservicio. Dado que cada servicio es auto-contenido, el despliegue de servicios independientes para cada nueva actualización, se ve simplificada dado que no es necesario controlar dependencias entre ellos. Igualmente es sabido que existen arquitecturas de microservicios, en las cuales sus microservicios tienen dependencias y esto por el contrario complejiza el testeo, teniendo que agregar pruebas de integración entre ellos y teniendo especial cuidado en su despliegue dado que deben seguir un determinado orden.

7. Referencias

[1] “Microservices Architecture – Chris Harding et al - The Open Group” - https://publications.opengroup.org/w169

[2] “Microservices Architecture – Irakli Nadareishvili, Ronnie Mitra, Matt McLarty & Mike Amundsen - O’Reilly”

[3] “Achieving enterprise agility with microservices and API management – Manfred Bortenschlager - Red Hat”

[4] “Migrating to Microservice Databases - Edson Yanaga - O’Reilly”

[5] “Microservices - a definition of this new architectural term - Martin Fowler, James Lewis” - https://martinfowler.com/articles/microservices.html

[6] “Conway’s law” - https://en.wikipedia.org/wiki/Conway%27s_law

[7] “Una breve historia de los patrones de microservicios” - https://www.ibm.com/developerworks/ssa/cloud/library/cl-evolution-microservices-patterns/index.html

[8] D. Wells, “Extreme Programming: A Gentle Introduction” - http://www.extremeprogramming.org/

[9] M. Fowler, “Continuous Integration” - http://www.martinfowler.com/articles/continuousIntegration.html

[10] Cloud Foundry Documentation, “Using Blue-Green Deployment to Reduce Downtime and Risk” - https://docs.cloudfoundry.org/devguide/deploy-apps/blue-green.html

ANEXO DevOps - CI/CD

¿Qué es integración continua?

Comúnmente nos encontramos con proyectos que sufren uno o varios de estos síntomas:

- Los plazos no se cumplen
- Se exceden en el presupuesto
- No satisfacen los requerimientos para los cuales fueron creados
- Los objetivos se alcanzan, al costo de dejar un equipo frustrado

Surge así la necesidad de contar con metodologías y prácticas de desarrollo que nos permitan elevar los niveles de certeza respecto al cumplimiento de plazos, restricciones presupuestales y requerimientos del negocio.

Este es el contexto en el que surge el concepto de Integración Continua (CI por sus siglas en inglés) como práctica de desarrollo de software. Inicialmente surgida como una práctica en el contexto de “eXtreme Programming”[8] (XP), tiene por objetivo principal evitar los problemas de integración que suelen suceder en proyectos de software.

Martin Fowler, define CI de la siguiente forma:

“Integración Continua es una práctica de desarrollo de software en la cual los miembros del equipo integran su trabajo con frecuencia, usualmente, cada persona integra su trabajo una vez al día llevando a múltiples integraciones al día. Cada una de ellas verificada por un proceso de “build” automatizado (incluyendo la ejecución de las pruebas) para detectar errores de integración lo antes posible. Muchos equipos encuentran que este enfoque lleva a reducir significativamente los problemas de integración y permite desarrollar software de forma cohesiva rápidamente.” [9]

Aplicar las prácticas relacionadas con Integración Continua involucra mucho más que habilidades exclusivamente técnicas. Durante el ciclo de vida intervienen roles de desarrollo, operación y gestión de proyectos. En particular, el rol “DevOps” que ha surgido desde un tiempo a esta parte, toma una relevancia más que importante ya que muchas tareas antes consideradas exclusivas de operación, necesitan de conocimiento en ocasiones profundo de aspectos relacionados al desarrollo de software y vice-versa.