UNIVERSIDAD AUTÓNOMA DE CHIAPAS

FACULTAD DE CONTADURÍA Y ADMINISTRACIÓN
CAMPUS 1

Licenciatura en Ingeniería en Desarrollo y Tecnologías de Software

Act.1.2 Arquitectura Orientada a Servicios

6 “M”

Materia: Taller de Desarrollo 4

Docente: Luis Gutiérrez Alfaro

ALUMNO:
A 211387
Steven de Dios Montoya Hernández

TUXTLA GUTIÉRREZ, CHIAPAS
Sábado, 19 de Agosto de 2023, 23:59

Índice

Introducción:

La forma en que se desarrollan e implementan las aplicaciones de software ha cambiado significativamente con la introducción de la Arquitectura Orientada a Servicios (SOA). Ha surgido un nuevo enfoque llamado Arquitectura de microservicios para enfrentar varios problemas, como la escalabilidad, la flexibilidad y el mantenimiento en el desarrollo de software. Con un enfoque descentralizado y modular, los Microservicios fomentan la creación e implementación de componentes individuales de manera autónoma, lo que resulta en una mayor adaptabilidad y tiempos de respuesta más rápidos a los requisitos del mercado.

Requisitos

Realizar investigación:

Arquitectura Orientada a Servicios, anexando la siguiente estructura:
Arquitectura de Microservicios

Características
Beneficios .
Técnicas de integración de microservicios
Balanceo de carga
Despliegue
Arquitectura monolítica vs arquitectura de microservicios
Buenas prácticas para diseñar arquitecturas de microservicio

* Aplicación con el cliente.
* Servicios.
* Directorio de servicio
* Bus de Servicio

Arquitectura Orientada a Servicios

La Arquitectura Orientada a Servicios, conocida también como SOA por sus siglas en inglés (Service-Oriented Architectures), es un tipo de arquitectura de software que permite reutilizar sus elementos gracias a las interfaces de servicios que se comunican a través de una red con un lenguaje común. Define la utilización de servicios (programas concretos o tareas con una función específica) para dar soporte a los requisitos de negocio.

En conclusión, una arquitectura SOA permite integrar los elementos del software que se implementan y se mantienen por separado, permitiendo que se comuniquen entre sí y trabajen de forma conjunta para formar aplicaciones de software en distintos sistemas.

¿Qué características identifican una arquitectura SOA?

- Los servicios son autónomos. Cada servicio SOA se mantiene y desarrolla de forma independiente.

- Los servicios son distribuibles. Se pueden ubicar en cualquier parte sobre la red siempre que este soporte los protocolos de comunicación requeridos.

- Los servicios se pueden descomponer. Cada servicio SOA es independiente de los otros y puede ser reemplazado o actualizado sin romper con las aplicaciones que conecta.

- Los servicios no comparten clases. En una arquitectura SOA, los servicios comparten contratos y esquemas cuando se comunican, no clases internas.

- Los servicios son compatibles con políticas. Entiendo políticas como la definición de características como el transporte, protocolo o seguridad.

Beneficios

Ventajas de implementar SOA

- Permite alinear y acercar las áreas de tecnología y negocio.
- Permite el desarrollo de aplicaciones manejables y seguras.
- Proporciona una infraestructura y documentación común.
- Permite desarrollar servicios con la posibilidad de añadir nuevas funcionalidades.
- Mejora la agilidad y flexibilidad de las organizaciones.
- Permite centralizar todos los servicios en un modelo único.
- Facilita el descubrimiento y reúso de los servicios existentes.

Mayor agilidad empresarial, comercialización más rápida: la reutilización es clave. La eficiencia de ensamblar aplicaciones a partir de servicios reutilizables, es decir bloques de construcción, en lugar de reescribir y reintegrar con cada nuevo proyecto de desarrollo, permiten a los desarrolladores crear aplicaciones con mucha mayor rapidez en respuesta a nuevas oportunidades comerciales. El enfoque arquitectónico orientado al servicio admite escenarios para la integración de aplicaciones, la integración de datos y la automatización del estilo de orquestación de servicios de los procesos de negocio o flujos de trabajo. Esto acelera el diseño de software y el desarrollo de software al permitir que los desarrolladores dediquen mucho menos tiempo a la integración y mucho más tiempo *a la entrega y* mejora de sus aplicaciones.

Capacidad para aprovechar la funcionalidad heredada en nuevos mercados: una SOA bien lograda permite a los desarrolladores adoptar fácilmente una funcionalidad "bloqueada" en una plataforma o entorno informático y llevarla a entornos y mercados nuevos. Por ejemplo, muchas empresas han utilizado SOA para exponer la funcionalidad de los sistemas financieros basados en mainframe a nuevas aplicaciones web, lo que permite a sus clientes acceder a procesos e información que antes solo eran accesibles a través de la interacción directa con los empleados o business partners de la empresa.

Mejora en la colaboración entre la empresa y la TI: en una SOA, los servicios se pueden definir en términos comerciales (por ejemplo, "generar oferta de seguros" o "calcular ROI de equipamiento de capital"). Esto permite a los analistas de negocio trabajar más eficazmente con los desarrolladores en aspectos importantes, como el ámbito de un proceso de negocio definido por servicios o las implicaciones para los negocios de cambiar un proceso, lo que puede generar mejores resultados.

Técnicas de integración de microservicios

La integración de microservicios es un aspecto clave en la arquitectura de microservicios, y existen varias formas de lograrlo. Algunas técnicas comunes incluyen:

API Gateway: Utilizar un punto de entrada único para enrutar y redirigir las solicitudes a los microservicios correspondientes. Esto puede simplificar la gestión de las solicitudes, autenticación y seguridad.

Event-Driven Architecture: Implementar una arquitectura basada en eventos, donde los microservicios se comunican a través de eventos y mensajes asincrónicos. Esto permite la escalabilidad y la independencia entre los microservicios.

Coreografía y Orquestación: En la coreografía, los microservicios colaboran entre sí para lograr una funcionalidad completa, mientras que en la orquestación, un componente central coordina y controla las interacciones entre los microservicios.

Retroalimentación de API: Mantener actualizaciones de API y versiones compatibles para facilitar la comunicación entre microservicios cuando se realizan cambios.

Sincronización de Datos: Utilizar patrones como el patrón de sincronización de base de datos o el patrón de vista de base de datos para garantizar la consistencia de los datos entre los microservicios.

Contratos de Interfaz: Definir contratos de interfaz claros y documentados para que los microservicios se comuniquen entre sí de manera efectiva.

Contenedorización y Orquestación: Utilizar tecnologías como Docker para contener microservicios y Kubernetes para orquestar y gestionar su despliegue.

API Composition: Componer y combinar múltiples microservicios para crear una funcionalidad más amplia antes de presentarla al usuario final.
Circuit Breaker: Implementar patrones de circuit breaker para manejar fallas en las llamadas entre microservicios y prevenir la degradación del sistema.

Seguridad y autenticación: Implementar medidas de seguridad como autenticación, autorización y encriptación para proteger las comunicaciones entre microservicios.

Balanceo de carga

¿Qué es el load balancing o balanceo de carga? El load balancing o balanceo de carga se define como una opción a cargo de la distribución de las cargas de trabajo entre los componentes que pertenecen al sistema. De manera que el equilibrio de carga se enfoca en el reparto de las solicitudes sobre una fila de servidores.

¿Cómo funciona el balanceo de carga definido por software?

El balanceo de carga definido por software hace lo mismo que el balanceo de carga definido por hardware: distribuye el tráfico entre un grupo de servidores de acuerdo con el algoritmo elegido.

La distribución de cargas de trabajo entre varios servidores mediante el balanceo de carga puede hacer que la red sea más eficiente y fiable. El balanceo de carga aumenta la capacidad de una red, ya que utiliza los servidores disponibles de una manera más eficiente. Por consiguiente, hace que la red se ejecute más rápido gracias a que las cargas de trabajo no se quedan bloqueadas en un servidor sobrecargado mientras otros servidores no se están utilizando. El balanceo de carga también garantiza un tiempo de actividad ininterrumpida cuando un servidor falla, porque el tráfico del servidor que ha fallado se desvía a servidores que sí están funcionando.

¿Qué tipos de balanceo de carga existen?

Los balanceadores de carga basados en software se pueden instalar directamente en un servidor, o bien se pueden adquirir como balanceadores de carga como servicio (LBaaS). En el caso de los LBaaS, el proveedor de servicios se encarga de instalar, configurar y gestionar el software de balanceo de carga. El balanceador de carga basado en software puede estar ubicado en las instalaciones o fuera de ellas.

Al igual que los servidores, los dispositivos de balanceo de carga pueden ser físicos o virtuales. Tanto los dispositivos físicos (balanceo de carga definido por hardware) como los virtuales (balanceo de carga definido por software) evalúan las solicitudes de los clientes y el uso de los servidores en tiempo real, y envían las solicitudes a diferentes servidores en función de diversos algoritmos. El destino del tráfico depende de la política de balanceo de carga establecida por el administrador.

Métodos de balanceo de carga

Los balanceadores de carga utilizan uno de los siguientes métodos para determinar dónde enviar el tráfico de red:

- Algoritmo Round Robin: es el método más sencillo de balanceo de carga. Tan solo mueve las solicitudes por una lista de servidores disponibles por orden.

- Algoritmo de menos conexiones: se trata de un método un poco más sofisticado. Envía solicitudes a los servidores menos ocupados, es decir, a los servidores que estén procesando la menor cantidad de cargas de trabajo en un momento determinado.

- Algoritmo de menos tiempo: va un paso más allá, y elige los servidores en función de la velocidad de procesamiento más rápida y del menor número de solicitudes activas. Este enfoque puede integrar algoritmos de balanceo de carga ponderados que dan preferencia de forma sistemática a los servidores con más capacidad, recursos informáticos o memoria.

- Algoritmo basado en hash: en este último método, el dispositivo de balanceo de carga asigna una clave hash única a las direcciones IP de origen y destino del cliente y el servidor. De este modo se garantiza que, si el mismo usuario vuelve y hace otra solicitud, esta solicitud de usuario se dirigirá al mismo servidor que se estaba utilizando antes. Además, el servidor conservará todos los datos introducidos durante las sesiones anteriores.

¿Por qué se utiliza el balanceo de carga definido por software?

El balanceo de carga definido por software está ganando popularidad porque ofrece varias ventajas en comparación con el balanceo de carga definido por hardware:

- Escalabilidad: la mayor ventaja que ofrecen los balanceadores de carga definidos por software en comparación con los dispositivos de balanceo de carga definido por hardware es su escalabilidad. Los balanceadores de carga definidos por software pueden añadir o eliminar servidores virtuales según la demanda, respondiendo automáticamente y en tiempo real a las fluctuaciones en el tráfico de red.
- Flexibilidad: los balanceadores de carga definidos por software también son más flexibles que los balanceadores de carga definidos por hardware, ya que son compatibles con diversos entornos. Se pueden programar para que funcionen con sistemas operativos estándar de sobremesa, entornos de nube, servidores bare metal, servidores virtuales y contenedores. Los balanceadores de carga definidos por hardware no son tan flexibles porque no son programables.
- Coste: las organizaciones pueden ahorrar dinero con los balanceadores de carga definidos por software, especialmente si utilizan LBaaS. Aunque una organización de TI compre sus propios balanceadores de carga definidos por software, el coste se suele considerar un gasto operativo en lugar de una inversión en capital, que sería el caso de los balanceadores de carga definidos por hardware.
- Facilidad de implementación: los balanceadores de carga definidos por hardware pueden ser difíciles y caros de instalar, pero los balanceadores de carga definidos por software son fáciles de implementar según las necesidades, lo cual ahorra tiempo y dinero.
- Seguridad: por último, el software de balanceo de carga que se encuentra entre el cliente y el servidor ofrece una capa adicional de seguridad, ya que cuenta con la capacidad de rechazar paquetes sospechosos antes de que lleguen al servidor.

Despliegue

El despliegue de software resulta esencial para garantizar que las aplicaciones se entregan e instalan de forma correcta y eficaz. Algunas ventajas que te aporta este proceso son:

- Agiliza los tiempos de comercialización: un despliegue veloz ayuda a satisfacer las demandas de los clientes y a adelantarse a la competencia.
- Mejora la calidad: también garantiza que la aplicación se entrega en la configuración deseada, con todas las dependencias y ajustes necesarios del sistema. Esto contribuye a minimizar los errores, optimizar el rendimiento y la experiencia del usuario.
- Incrementa la seguridad: el proceso garantiza que la aplicación de software está protegida frente a vulnerabilidades que podrían ser aprovechadas por atacantes.
- Optimiza el control de los costos: al automatizar el proceso de despliegue, las organizaciones reducen los errores manuales, ahorran tiempo y bajan los gastos.

A pesar de sus numerosas ventajas, también es necesario conocer algunos obstáculos que pueden aparecer:

- Problemas de compatibilidad: hay que asegurar la compatibilidad entre los diferentes componentes de software y sistemas.
- Falta de estandarización: como el despliegue es posible hacerlo de muchas maneras, crea un proceso estandarizado para tu organización.
- Gestión de la configuración y control de versiones: es clave hacer un seguimiento de la versión y la configuración del software en todos los dispositivos. En ese sentido, las herramientas de monitoreo constituyen un gran aliado.
- Dependencias complejas de software: para gestionarlas adecuadamente, construye un CDMB con toda la estructura de relaciones.

¿Cuáles son las formas de realizar el deploy?

Como dijimos antes, el proceso de deploy de sistemas ha evolucionado mucho a lo largo de las últimas décadas. Hoy existen básicamente 3 formas de realizarlo:

1 – Manual

Un ejemplo de este proceso es el Protocolo de Transferencia de archivos, conocido como FTP. Se trata de un tipo de conexión que permite que dos computadores con acceso a Internet intercambien archivos. El proceso se realiza de forma manual, por lo cual necesita de una persona en el comando.

Otra forma de hacer un deploy manual es cuando necesitas cambiar un pequeño detalle en tu archivo JS y realizas el upload para la producción.

2 – Parcialmente automatizado

Un ejemplo de deploy parcialmente automatizado puede ser el push de branch master que se realiza en el repositorio Git, el cual opera un pequeño hook y actualiza el servidor de web hosting. Aunque necesita algunos comandos, el proceso ocurre de manera automática. Su ventaja es el control de la versión y el estado de cada deploy.

3 – Completamente automatizado

Es la tendencia más moderna en el desarrollo web. Este no solamente copia los cambios de forma automática en el servidor, sino también está íntimamente conectado con el concepto de integración continua.

Además de la copia, el deploy continuo ofrece muchas ventajas, pues con ello será posible trabajar los cambios, o integraciones, hechas por diferentes programadores.

La herramienta de deploy, en este caso, realiza todas las pruebas necesarias para que no existan problemas a la hora de juntar las integraciones de la producción. Las posibilidades son infinitas, desde:

- Actualizar bibliotecas;
- Probar la conectividad de los servidores;
- Simular visitas y entradas de datos.

Todo esto es hecho de forma automática y, en caso de que ocurra algún error, el deploy se puede revertir.

Una de las herramientas más famosas para el deploy automatizado es Jenkins. Incluyendo sus beneficios, se destaca:

1. Alto nivel de productividad;
2. Seguridad;
3. Calidad en el desarrollo de softwares

Arquitectura monolítica vs arquitectura de microservicios

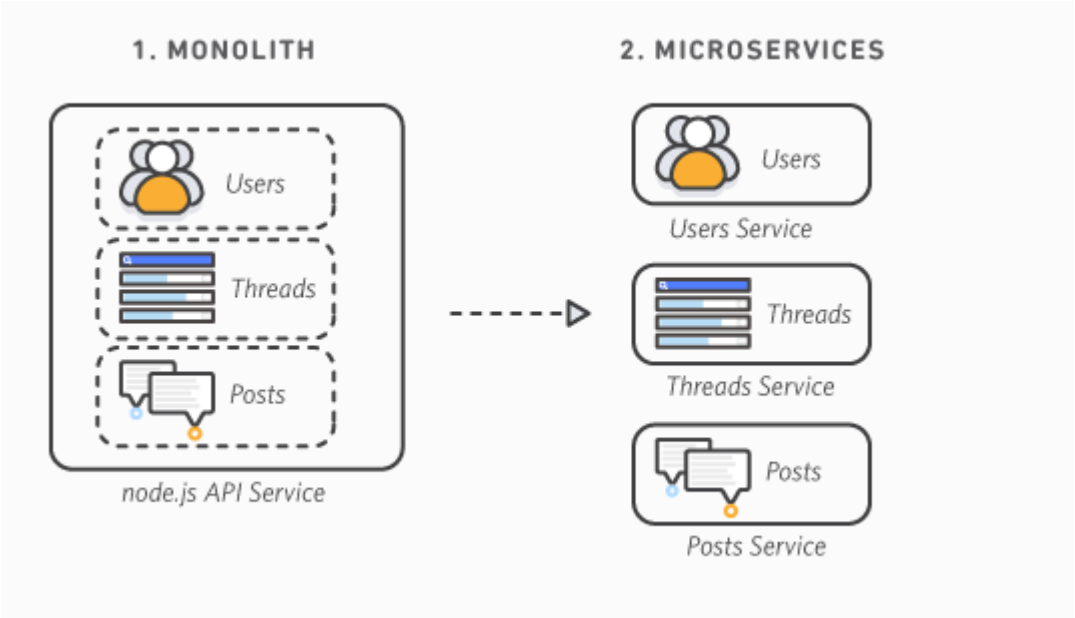
¿Cuál es la diferencia entre la arquitectura monolítica y la de microservicios?

Una arquitectura monolítica es un modelo de desarrollo de software tradicional que utiliza un código base para realizar varias funciones empresariales. Todos los componentes de software de un sistema monolítico son interdependientes debido a los mecanismos de intercambio de datos dentro del sistema. Modificar la arquitectura monolítica es restrictivo y lleva mucho tiempo, ya que los pequeños cambios afectan a grandes áreas del código base. Por el contrario, los microservicios son un enfoque arquitectónico que compone el software en pequeños componentes o servicios independientes. Cada servicio realiza una única función y se comunica con otros servicios a través de una interfaz bien definida. Como se ejecutan de forma independiente, se puede actualizar, modific

Diferencias clave: arquitectura monolítica frente a arquitectura de microservicios

Las aplicaciones monolíticas suelen constar de una interfaz de usuario del cliente, una base de datos y una aplicación del servidor. Los desarrolladores crean todos estos módulos en un único código base.

Por otro lado, en una arquitectura distribuida, cada microservicio funciona para lograr una única característica o lógica empresarial. En lugar de intercambiar datos dentro del mismo código base, los microservicios se comunican con una API.



Buenas prácticas para diseñar arquitecturas de microservicio.

1) Crea un almacén de datos separado para cada microservicio

Primero que nada, no uses el mismo almacén de datos para el backend cuando quieras implementar o realizar la transición a este tipo de arquitectura..

Mantener los datos separados puede hacer que la administración de datos sea más complicada, porque los sistemas de almacenamiento separados pueden tener errores o ralentizarse en los momentos de sincronización, volviéndose inconsistentes y limitando la escalabilidad.

2) Mantén el código en un nivel de madurez similar

Esta práctica recomienda mantener todo el código a un nivel similar de madurez y estabilidad, es decir, si necesita agregar o reescribir parte del código en este tipo de arquitectura implementada que funciona bien, el mejor enfoque suele ser crear un nuevo microservicio para el código nuevo o modificado, dejando el microservicio existente en su lugar.
De esta forma, puedes implementar y probar de forma repetida el nuevo código hasta que esté libre de errores y sea lo más eficiente posible, sin riesgo de fallas o degradación del rendimiento en el microservicio existente, este proceso se conoce como el principio de infraestructura inmutable.
Permitiendo así, luego de verificar que este nuevo microservicio es tan

estable como el original, fusionarlos nuevamente si realmente realizan una sola función en conjunto, o si hay otras eficiencias al combinarlos.

3) Hacer una compilación separada para cada microservicio

Esta es una de las mejores prácticas recomendadas al momento de diseñar una arquitectura este tipo para el desarrollo móvil o de software, ya que insta la realización de una compilación por separado para cada microservicio, de modo que puedas extraer archivos de componentes del repositorio en los niveles de revisión apropiados para él.

Esto a veces conduce a la situación en la que varios de este tipo extraen un conjunto similar de archivos, pero a diferentes niveles de revisión, eso puede hacer que sea más difícil limpiar la base de código retirando versiones antiguas de archivos (porque debes verificar con más cuidado que ya no se está utilizando una revisión), pero eso es una compensación aceptable por lo fácil que es agregar nuevos archivos a medida que construyes nuevos microservicios.

4) Implementar en contenedores

La implementación de microservicios en contenedores es importante porque significa que solo necesitas una herramienta para implementarlo todo.

Mientras el microservicio esté en un contenedor, la herramienta sabe cómo implementarlo, sin importar cuál sea el contenedor.

5) Tratar a los servidores como miembros intercambiables

Esta práctica recomienda tratar a los servidores, particularmente aquellos que ejecutan código orientado al cliente, como miembros intercambiables de un grupo.

Es decir, todos realizan las mismas funciones, por lo que no necesitas preocuparse por ellos individualmente, tu única preocupación es que existan suficientes para producir la cantidad de trabajo que necesitas, y puede usar la escala automática para ajustar los números hacia arriba y hacia abajo.

De esa manera, si uno deja de funcionar, se reemplaza automáticamente por otro.

6) Usa ‘defensa en profundidad’ para priorizar servicios clave
Luego de identificar cuáles son tus servicios más sensibles, aplicas varias capas de seguridad diferentes para que un atacante potencial que pueda explotar una de sus capas de seguridad todavía tenga que encontrar una manera de vencer a todas sus otras defensas en tus servicios críticos.
En general, esto es más fácil decirlo que hacerlo, pero hay varios recursos disponibles, la buena noticia es que este tipo de arquitecturas facilitan la adopción de esta estrategia de una manera muy granular y estratégica, al enfocar sus esfuerzos y recursos de seguridad en microservicios específicos, aumentando la diversificación de las capas de seguridad que deseas adoptar en cada uno de ellos.

7) Usa actualizaciones de seguridad automáticas
Cada vez que se actualiza una parte de tu sistema, debes asegurarte de detectar cualquier problema lo suficientemente pronto y con el mayor detalle posible.
Para ello, implementa esta práctica para asegurarte de que tu plataforma sea principalmente “atómica”, es decir, todo debe estar envuelto en contenedores para que probar tu aplicación con una biblioteca actualizada o una versión de idioma sea solo una cuestión de envolver un contenedor diferente a su alrededor, así, si la operación falla, revertir todo será bastante fácil y, lo más importante, puede automatizarse.

Conclusiones
A diferencia de la arquitectura de monolito, diseñar e implementar microservicios de manera correcta puede llegar a ser algo desafiante y difícil, pero dado que brinda una solución descentralizada para diferentes problemas para el desarrollo móvil y desarrollo de software, definir el conjunto de mejores prácticas no solo es importante sino también crucial.

Aplicación con el cliente:

Descomposición Funcional: Divide la aplicación en funciones y características independientes que puedan ser implementadas y desplegadas como microservicios separados.

APIs Claras: Define interfaces claras y estables para cada microservicio, utilizando APIs RESTful u otros protocolos según sea necesario.

Escalabilidad: Diseña para la escalabilidad horizontal, permitiendo agregar instancias de microservicios según la demanda, lo que mejora el rendimiento.

Servicios:

Límites de Contexto: Delimita claramente los contextos y responsabilidades de cada microservicio para evitar interdependencias excesivas.

Comunicación Ligera: Utiliza protocolos de comunicación ligera como HTTP/REST o gRPC para facilitar la interacción entre microservicios.

Persistencia Autónoma: Cada microservicio debe manejar su propia persistencia de datos, eligiendo la base de datos adecuada para sus necesidades.

Directorio de Servicio:

Descubrimiento de Servicio: Utiliza un servicio de descubrimiento (como Consul o Eureka) para que los microservicios puedan registrarse y descubrirse mutuamente de manera dinámica.

Balanceo de Carga: Asegúrate de que el directorio de servicio pueda realizar un balanceo de carga efectivo entre instancias de microservicios para optimizar el rendimiento.

Resiliencia: Configura el directorio de servicio para manejar la alta disponibilidad y la recuperación ante fallas.

Bus de Servicio:

Patrones de Mensajería: Utiliza patrones de mensajería como publicación/suscripción o colas de mensajes para facilitar la comunicación asincrónica entre microservicios.

Tolerancia a Fallos: Diseña el bus de servicio para manejar la tolerancia a fallos y garantizar que los mensajes no se pierdan incluso en situaciones de interrupción.

Monitorización: Implementa la monitorización y registro adecuados para supervisar el tráfico y la salud del bus de servicio.

Conclusión

La Arquitectura Orientada a Servicios (SOA, por sus siglas en inglés) ha transformado la forma en que las organizaciones desarrollan y despliegan aplicaciones. Dentro de este enfoque, la Arquitectura de Microservicios ha surgido como una alternativa innovadora que busca abordar los desafíos de escalabilidad, flexibilidad y mantenimiento en el desarrollo de software. Los microservicios representan un enfoque modular y descentralizado para la construcción de aplicaciones, donde los componentes individuales se desarrollan y despliegan de manera independiente, lo que permite una mayor agilidad y una respuesta más rápida a las demandas cambiantes del mercado.

Bibliografía:

***Arquitectura monolítica vs arquitectura de microservicios*. (s/f). Bing.** Recuperado el 18 de agosto de 2023, de https://www.bing.com/search?pglt=161&q=Arquitectura+monol%C3%ADtica+vs+arquitectura+de+microservicios&cvid=b6a13cc952f8492f812e9c31d02b4bfb&aqs=edge..69i57.579j0j1&FORM=ANAB01&PC=U531

***Deploy: Qué significa en el mundo de la programación*. (s/f). Hostgator.mx.** Recuperado el 18 de agosto de 2023, de https://www.hostgator.mx/blog/deploy-en-programacion/

Huenei IT Services. (2020, octubre 1). *Mejores prácticas en Microservicios*. Huenei IT Services; Huenei. https://www.huenei.com/mejores-practicas-en-microservicios/

***¿Qué es la arquitectura orientada a los servicios?* (s/f). Redhat.com.** Recuperado el 18 de agosto de 2023, de https://www.redhat.com/es/topics/cloud-native-apps/what-is-service-oriented-architecture

***¿Qué es la SOA (arquitectura orientada a servicios)?* (s/f). Ibm.com.** Recuperado el 18 de agosto de 2023, de https://www.ibm.com/mx-es/topics/soa

Vilchez, E. D. O. (1664241683000). *Arquitectura Orientada a servicios (soa)*. Linkedin.com. https://www.linkedin.com/pulse/arquitectura-orientada-servicios-soa-eber-daniel-or%C3%A9-vilchez/?originalSubdomain=es

***What is Software Load Balancing?* (2022, septiembre 20). VMware.** https://www.vmware.com/es/topics/glossary/content/software-load-balancing.html

Wrobel, M. (2023, junio 19). ¿Qué es el despliegue de software? Definición, alcance y buenas prácticas. *Invgate.com*. https://blog.invgate.com/es/despliegue-de-software

(S/f). Amazon.com. Recuperado el 18 de agosto de 2023, de https://aws.amazon.com/es/compare/the-difference-between-monolithic-and-microservices-architecture/