UNIVERSIDAD AUTÓNOMA DE CHIAPAS

FACULTAD DE CONTADURÍA Y ADMINISTRACIÓN
CAMPUS 1

ING. EN DESARROLLO Y TECNOLOGÍAS DE SOFTWARE

6 “M”

TALLER DE DESARROLLO 4

SUBCOMPETENCIA 1

ACT. 1.2 ARQUITECTURA ORIENTADA A SERVICIOS

ALUMNO: MARCO ANTONIO ZÚÑIGA MORALES – A211121

DOCENTE: DR. LUIS GUTIÉRREZ ALFARO

TUXTLA GUTIÉRREZ, CHIAPAS
SÁBADO, 19 DE AGOSTO DE 2023

ÍNDICE

Introducción

En mundo de la tecnología y la informática, donde la agilidad y la escalabilidad son esenciales, la Arquitectura Orientada a Servicios ha emergido como un paradigma revolucionario. En este contexto, la Arquitectura de Microservicios se alza como una poderosa respuesta a la necesidad de sistemas flexibles y eficientes. A través de este enfoque innovador, las organizaciones pueden descomponer sus aplicaciones en unidades más pequeñas y autónomas, conocidas como microservicios. Estos componentes, altamente especializados, trabajan en conjunto para formar sistemas complejos y, a la vez, se pueden desarrollar, desplegar y escalar de manera independiente.

Las características distintivas de esta arquitectura impulsan su adopción en diversos sectores. Su modularidad y autonomía permiten a los equipos de desarrollo enfocarse en áreas específicas de funcionalidad, lo que acelera el proceso de implementación y actualización. A medida que los microservicios se comunican a través de interfaces bien definidas, las técnicas de integración se convierten en un aspecto crucial para garantizar la cohesión del sistema. Aquí es donde el balanceo de carga juega un papel fundamental, asegurando que cada microservicio reciba una distribución equitativa de las solicitudes entrantes.

El despliegue ágil es otro rasgo destacado de esta arquitectura. En comparación con las arquitecturas monolíticas, donde cualquier cambio puede afectar a toda la aplicación, los microservicios se despliegan de manera independiente, lo que reduce el riesgo y la interrupción del servicio. Esta ventaja se conecta directamente con la discusión sobre Arquitectura monolítica vs arquitectura de microservicios, un dilema que las organizaciones enfrentan al decidir el enfoque de desarrollo más adecuado para sus necesidades.

A medida que exploramos las bondades de la Arquitectura de Microservicios, se torna esencial comprender las buenas prácticas para diseñar estas arquitecturas. Desde la definición de límites de microservicios hasta la gestión de la comunicación entre ellos, cada paso debe seguir principios sólidos para asegurar un sistema coherente y eficiente en el tiempo. En esta exploración, exploraremos en detalle las características, beneficios, técnicas de integración, balanceo de carga, despliegue y comparación con arquitecturas monolíticas, brindando una panorámica completa de cómo esta innovadora aproximación está transformando la forma en que concebimos y construimos aplicaciones en la era digital.

A continuación, en este documento se presentarán las características de los microservicios, así como sus beneficios y sus buenas prácticas, etc.

Desarrollo

Arquitectura Orientada a Servicios

La arquitectura orientada a servicios (SOA, por sus siglas en inglés) es un método de desarrollo de software que utiliza componentes de software llamados servicios para crear aplicaciones empresariales. Cada uno de estos servicios brinda una capacidad empresarial y, además, pueden comunicarse también con el resto de los servicios mediante diferentes plataformas y lenguajes. Los desarrolladores usan SOA para reutilizar servicios en diferentes sistemas o combinar varios servicios independientes para realizar tareas complejas.

Arquitectura de microservicios

Los microservicios (o arquitectura de microservicios) son un enfoque arquitectónico nativo de la nube en el que una sola aplicación se compone de muchos componentes o servicios más pequeños acoplados de forma flexible e independiente

Características de 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 remplazado o actualizado sin romper con las aplicaciones que conecta.

- **Los servicios no comparten clases.** En una arquitectura SOA, los servicios comparten y 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.

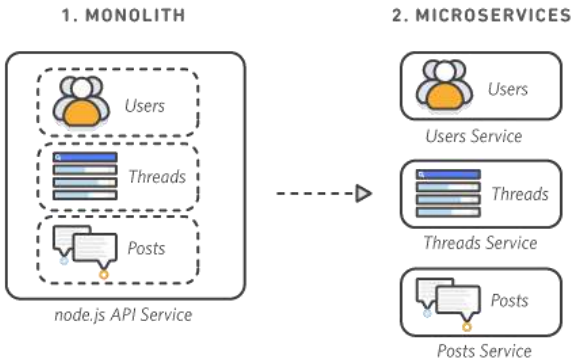
Beneficios SOA

La arquitectura orientada a servicios (SOA) ofrece varios beneficios por encima de las arquitecturas monolíticas tradicionales, donde todos los procesos se ponen en marcha como una unidad única. Algunos de los principales beneficios de la SOA incluyen los siguientes:

- **Reducción del plazo de comercialización**
  Los desarrolladores reutilizan servicios en diferentes procesos empresariales para ahorrar tiempo y dinero. Pueden crear aplicaciones en menos tiempo con SOA en lugar de escribir código y llevar a cabo integraciones desde cero.

- **Mantenimiento eficiente**
  Es más fácil crear, actualizar y corregir errores en servicios pequeños que en bloques grandes de código en aplicaciones monolíticas. La modificación de un servicio en SOA no afecta a la funcionalidad general del proceso empresarial.

- **Excelente capacidad de adaptación**
  La SOA se adapta de mejor manera a los avances tecnológicos. Puede modernizar sus aplicaciones de forma eficiente y rentable. Por ejemplo, las organizaciones de atención médica pueden utilizar la funcionalidad de sistemas de registro de salud electrónico antiguos en aplicaciones basadas en la nube que son más recientes.



Ventajas

- 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.

Técnicas de integración de microservicios

Algunas técnicas de la integración de microservicios son las siguientes:

- **API Gateway:** Un API Gateway es un punto de entrada único para las solicitudes de clientes hacia varios microservicios. Actúa como un proxy inverso y maneja funciones como enrutamiento, autenticación, autorización y transformación de datos. Ayuda a simplificar la comunicación para los clientes y mejora la seguridad y el rendimiento.
- **Comunicación Sincrónica:** Los microservicios pueden comunicarse a través de solicitudes HTTP síncronas, utilizando patrones REST o GraphQL. Esto es útil para operaciones que requieren respuestas inmediatas y directas.
- **Comunicación Asincrónica:** Los mensajes y las colas de mensajes (como RabbitMQ o Apache Kafka) permiten la comunicación asincrónica entre microservicios. Esto es útil para casos en los que la velocidad de respuesta no es crítica y para evitar acoplamientos fuertes.
- **Event-Driven Architecture:** Los eventos son emitidos por un servicio y pueden ser escuchados por otros servicios interesados. Esto permite la comunicación y coordinación sin acoplamientos directos. Eventos como "usuario registrado" o "pedido creado" pueden activar acciones en otros microservicios.
- **Coreografía y Orquestación:** En la coreografía, cada microservicio realiza su propia acción en respuesta a eventos y no hay un control centralizado. En la orquestación, un microservicio coordina las acciones de otros, generalmente a través de flujos de trabajo definidos.
- **Patrones de Mensajes:** Utiliza patrones de mensajes como "Request-Reply", "Publish-Subscribe", "Competing Consumers", entre otros, según los requerimientos de comunicación entre microservicios.
- **Contratos de API:** Define y documenta claramente los contratos de API para cada microservicio. Esto incluye la estructura de las solicitudes y respuestas, las rutas de acceso y los métodos admitidos.
- **Testing de Integración:** Realiza pruebas de integración que aseguren que los microservicios se comunican correctamente y que las respuestas se manejan adecuadamente. Puedes utilizar herramientas de pruebas como Postman o frameworks de pruebas automatizadas.
- **Monitorización y Observabilidad:** Implementa herramientas de monitorización y observabilidad para rastrear el rendimiento y los errores de los microservicios. Esto facilita la identificación de problemas de comunicación y mejora la resolución de problemas.
- **Gestión de Errores y Retrys:** Implementa manejo de errores y mecanismos de reintento en la comunicación entre microservicios. Esto garantiza una mayor tolerancia a fallos.
- **Versionado de APIs:** Cuando realices cambios en las APIs de los microservicios, asegúrate de tener en cuenta el versionado para que las

actualizaciones no rompan la compatibilidad con los consumidores existentes.

- **Seguridad:** Implementa medidas de seguridad como autenticación y autorización en las comunicaciones entre microservicios para proteger los datos y evitar accesos no autorizados.

Balanceo de carga

El balanceo de carga es una técnica crucial en la arquitectura de microservicios para distribuir la carga de trabajo de manera equitativa entre varios servidores o instancias para mejorar el rendimiento, la disponibilidad y la escalabilidad de una aplicación.

Tipos de balanceo de carga

- **Balanceo de carga basado en Round Robin:** Las solicitudes se distribuyen en orden secuencial a través de los servidores disponibles.

- **Balanceo de carga ponderado:** Se asignan ponderaciones diferentes a los servidores según su capacidad, de modo que los servidores más potentes manejen una carga mayor.

- **Balanceo de carga basado en IP hash:** Se utiliza la dirección IP del cliente para determinar a qué servidor se enviará la solicitud. Esto puede ser útil para mantener la persistencia en sesiones.

Despliegue

El despliegue de software (software deployment) es una tarea central para los equipos de IT en las organizaciones de cualquier tamaño. Todo el proceso de lanzamiento de las aplicaciones de software presenta algunos desafíos, ya que abarca una amplia gama de actividades: desde la creación y prueba, hasta su empaquetado y despliegue.

El despliegue de software es el proceso de entregar aplicaciones de software desde entornos de desarrollo a producción, poniéndolas a disposición de los usuarios finales. Debe garantizar que el software se configure, pruebe, lance e instale de forma coherente y controlada.

Importancia del despliegue

- **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.

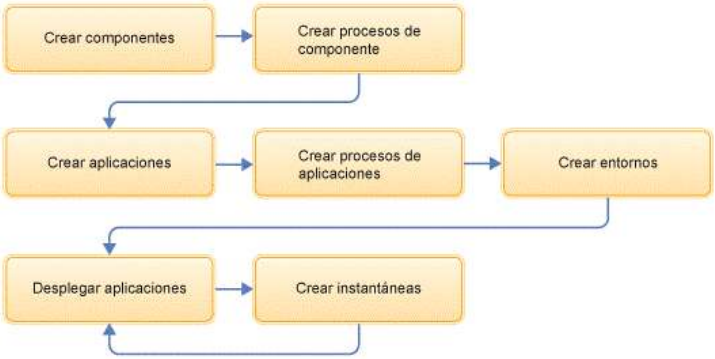
- **Incrementa la seguridad:** el proceso garantiza que la aplicación de software está protegida frente a vulnerabilidades que podrían ser aprovechadas por atacantes.

- **Optimiza el control de los costos:** al automatizar el proceso de despliegue, las organizaciones reducen los errores manuales, ahorran tiempo y bajan los gastos.

El modelado del despliegue de software de IBM® UrbanCode Deploy incluye la configuración de componentes y procesos de componente y la adición de estos componentes a aplicaciones. A continuación, utilice los procesos para desplegar los componentes en entornos.

Los pasos generales de este flujo de trabajo típico incluyen:

1. Importe las partes de la aplicación como componentes.
2. Cree procesos que desplieguen componentes.
3. Crear una aplicación para agrupar los componentes.
4. Crear un proceso de aplicaciones que ejecute cada proceso de componente.
5. Crear uno o varios entornos en los que desplegar los componentes.
6. Ejecutar el proceso de aplicaciones para desplegar los componentes.
7. Tome instantáneas de los despliegues en ejecución para poder guardar estas disposiciones y desplegarlas en otros entornos.



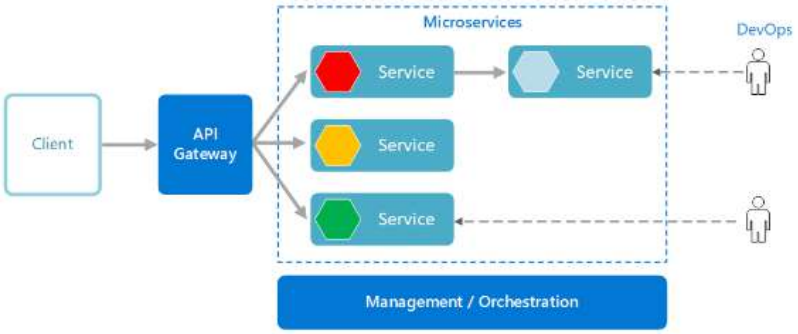
Arquitectura monolítica vs arquitectura de microservicios

Arquitectura monolítica

Una arquitectura monolítica es un modelo tradicional de un programa de software que se compila como una unidad unificada y que es autónoma e independiente de otras aplicaciones.

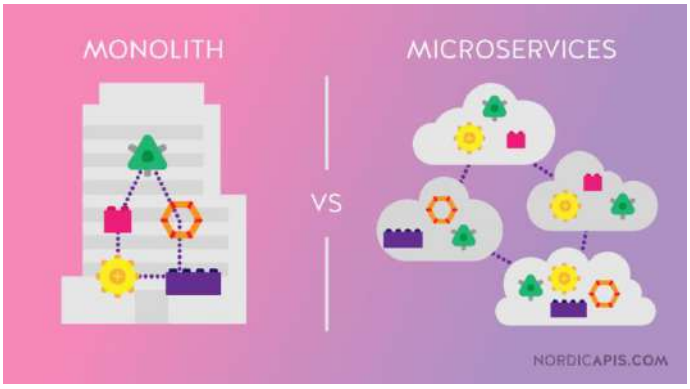
Arquitectura de microservicios

Una arquitectura de microservicios consta de una colección de servicios autónomos y pequeños. Cada uno de servicio es independiente y debe implementar una funcionalidad de negocio individual dentro de un contexto delimitado. Un contexto delimitado es una división natural de una empresa y proporciona un límite explícito dentro del cual existe un modelo de dominio.



Los microservicios son pequeños e independientes, y están acoplados de forma imprecisa. Un único equipo reducido de programadores puede escribir y mantener un servicio. Cada servicio es un código base independiente, que puede administrarse por un equipo de desarrollo pequeño. Los servicios pueden implementarse de manera independiente. Un equipo puede actualizar un servicio existente sin tener que volver a generar e implementar toda la aplicación. Los servicios son los responsables de conservar sus propios datos o estado externo. Esto difiere del modelo tradicional, donde una capa de datos independiente controla la persistencia de los datos.

Aspecto	Arquitectura Monolítica	Arquitectura de Microservicios
Estructura	Una sola aplicación y base de código	Varios servicios independientes
Escalabilidad	Escalado vertical de toda la aplicación	Escalado horizontal selectivo de servicios
Despliegue	Despliegue como una entidad única	Despliegue independiente de servicios
Comunicación	Componentes generalmente comunican dentro del mismo proceso	Comunicación a través de redes
Tecnologías	Tecnologías coherentes en toda la aplicación	Diferentes tecnologías por servicio
Acoplamiento	Mayor acoplamiento entre componentes	Bajo acoplamiento entre servicios
Desarrollo	Menos equipos de desarrollo	Equipos pequeños y especializados
Mantenimiento	Puede ser más simple en aplicaciones pequeñas	Requiere una gestión de servicios más compleja
Escalabilidad	Menos flexible, se escalan todos los componentes juntos	Mayor flexibilidad, escalado solo donde es necesario
Pruebas	Pueden ser más sencillas debido a la estructura monolítica	Pruebas más complejas debido a la comunicación entre servicios
Tiempo de Desarrollo	Puede ser más corto inicialmente	Puede requerir más tiempo debido a la separación
Evolución y Actualización	Cambios pueden ser más complejos y riesgosos	Cambios más manejables debido a la independencia
Costos	Menor costo inicial	Mayor costo en la implementación y gestión de infraestructura
Ejemplos	Aplicaciones pequeños o prototipos	Aplicaciones complejas y escalables



Buenas prácticas para diseñar arquitecturas de microservicio

Diseñar una arquitectura de microservicios efectiva requiere la consideración de varias buenas prácticas para asegurar que los servicios sean escalables, mantenibles y confiables. Algunas de las buenas prácticas para diseñar arquitecturas de microservicios son las siguientes:

1. **Definir límites de dominio claros:** Divide la aplicación en servicios basados en límites de dominio claros. Cada servicio debe ser responsable de una funcionalidad específica y acotada.

2. **Descomposición adecuada:** Descompón la aplicación en servicios lo suficientemente pequeños como para ser gestionables, pero no tan pequeños que resulten en una complejidad excesiva de comunicación entre ellos.

3. **Separación de responsabilidades:** Cada servicio debe ser responsable de una función o característica específica. Evita la duplicación de funcionalidad entre servicios.

4. **APIs bien definidas:** Define interfaces claras y estables (APIs) para cada servicio. Esto ayuda a reducir la dependencia entre los servicios y facilita futuros cambios.

5. **Comunicación eficiente:** Utiliza protocolos de comunicación ligeros y eficientes, como HTTP/REST o gRPC, para la interacción entre servicios. Considera el uso de colas de mensajes para tareas asincrónicas.

6. **Gestión de datos:** Cada servicio debe tener su propia base de datos si es necesario. Evita compartir bases de datos entre servicios para minimizar el acoplamiento.

7. **Aislamiento de errores:** Los servicios deben estar aislados unos de otros en términos de fallos. Un fallo en un servicio no debería afectar la disponibilidad de otros servicios.

8. **Escalabilidad individual:** Diseña servicios de manera que puedan escalarse de forma independiente. Esto permite que solo los servicios que necesitan más recursos sean escalados.

9. **Monitorización y registro:** Implementa mecanismos de registro y monitorización para cada servicio. Esto ayuda a identificar y resolver problemas de manera más eficiente.

10. **Circuit Breaker y manejo de fallos:** Utiliza patrones como Circuit Breaker para gestionar el flujo de tráfico y manejar fallos de manera adecuada para evitar la propagación de problemas.

11. **Automatización y CI/CD:** Implementa prácticas de integración continua (CI) y entrega continua (CD) para automatizar el despliegue y asegurar que los cambios se implementen de manera consistente.

12. **Cultura de equipo colaborativo:** Fomenta la comunicación constante entre equipos y colaboración en el diseño, desarrollo y operación de los microservicios.

13. **Seguridad:** Implementa medidas de seguridad como autenticación y autorización en cada servicio. Protege las comunicaciones y los datos sensibles.

14. **Documentación:** Mantén una documentación clara y actualizada para cada servicio, incluyendo su propósito, API, requisitos y dependencias.

15. **Pruebas adecuadas:** Realiza pruebas unitarias, de integración y de extremo a extremo para garantizar que los servicios funcionen correctamente individualmente y en conjunto.

16. **Considera el rendimiento:** Evalúa y optimiza el rendimiento de cada servicio, identificando cuellos de botella y optimizando las consultas y operaciones.

17. **Evolución continua:** Prepárate para la evolución constante. Los microservicios deben ser adaptables a medida que los requisitos cambian con el tiempo.

18. **Capacidad de rollback:** Asegúrate de que puedes revertir cambios en un servicio si algo sale mal durante una actualización.

Mejores prácticas para realizar microservicios

1. Crea un almacén de datos separas para cada microservicio
2. Mantén el código en un nivel de madurez similar
3. Hacer una compilación separada para cada microservicio
4. Implementar en contenedores
5. Tratar a los servidores como miembros intercambiables
6. Usa defensa en profundidad para priorizar servicios clave
7. Usa actualizaciones de seguridad automáticas

Aplicación con el cliente:
Aplicación con el cliente se refiere a la forma en que las aplicaciones consumen y utilizan los servicios proporcionados por los microservicios. Los clientes son componentes o sistemas que hacen solicitudes a los microservicios para obtener datos o realizar acciones. Los clientes pueden ser otras aplicaciones, interfaces de usuario, sistemas externos, etc.

Servicio:
Un servicio en el contexto de los microservicios se refiere a una unidad funcional autónoma que realiza una tarea específica. Cada microservicio se centra en una función de negocio y se desarrolla y despliega independientemente de los demás servicios. Los servicios pueden comunicarse entre sí a través de protocolos de comunicación, como HTTP o mensajería, para construir una aplicación completa.

Directorio de servicio:
Un directorio de servicios, también conocido como registro de servicios o Service Registry, es un componente en una arquitectura de microservicios que mantiene un registro actualizado de todos los servicios disponibles junto con su ubicación y detalles de conexión. Esto permite que otros servicios o componentes de la aplicación localicen y se comuniquen con los servicios necesarios sin tener que conocer su ubicación exacta.

Bus de servicio:
Un bus de servicios, también conocido como Service Bus, es una infraestructura que permite la comunicación y el intercambio de datos entre los diferentes servicios de una arquitectura de microservicios. Proporciona un canal centralizado para la comunicación entre los microservicios, lo que puede simplificar la gestión de las interacciones entre ellos. Puede incluir capacidades de enrutamiento, transformación de mensajes y manejo de colas para garantizar una comunicación eficiente y confiable entre los servicios.

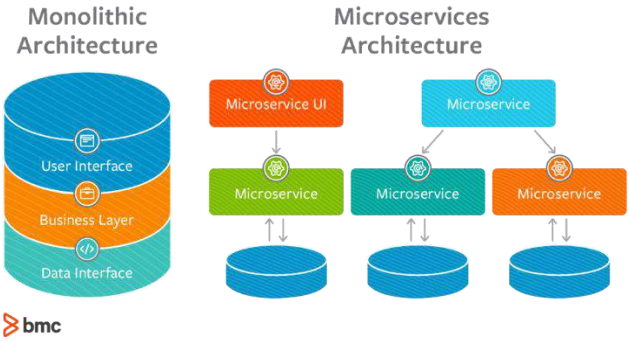
Conclusión

En conclusión, de acuerdo con la información investigada de páginas web y algunos libros vemos que la Arquitectura Orientada a Servicios ha evolucionado para dar paso a la Arquitectura de Microservicios, un enfoque que ha redefinido la manera en que diseñamos y construimos sistemas de software. Sus características fundamentales, como la modularidad y la autonomía de los microservicios, han brindado a las organizaciones la capacidad de adaptarse ágilmente a un entorno tecnológico en constante cambio.

Los beneficios intrínsecos de esta arquitectura son notables, desde una mayor escalabilidad hasta una mejor administración de recursos y una reducción de los riesgos asociados a implementaciones y actualizaciones. Las técnicas de integración, incluido el crucial balanceo de carga, han permitido a los microservicios comunicarse sin problemas, garantizando la coherencia y la eficiencia del sistema en su conjunto.

El proceso de despliegue independiente de los microservicios ha eliminado las limitaciones tradicionalmente asociadas con las arquitecturas monolíticas, ofreciendo una mayor flexibilidad y un menor impacto en la experiencia del usuario durante los cambios. La comparación entre estas dos arquitecturas resalta la ventaja que ofrece la Arquitectura de Microservicios en términos de agilidad y mantenibilidad.

Finalmente, las buenas prácticas en el diseño de arquitecturas de microservicio, desde la definición adecuada de los límites hasta la gestión efectiva de la comunicación entre microservicios, son esenciales para aprovechar al máximo las ventajas de este enfoque. Al adoptar estas prácticas, las organizaciones pueden asegurarse de que sus sistemas sean coherentes, eficientes y capaces de evolucionar de manera constante en un mundo tecnológico en rápida transformación. En conjunto, la Arquitectura de Microservicios no solo ha transformado la forma en que creamos aplicaciones, sino que también ha sentado las bases para un futuro tecnológico más adaptable y resiliente.



Referencias bibliográficas

Aguilar, Q. (2021, 29 julio). *Arquitecturas monolíticas o arquitectura de microservicios*. Cloud Computing y Servicios Gestionados IT. https://www.ilimit.com/blog/arquitecturas-monoliticas-o-arquitectura-de-microservicios-ventajas-e-inconvenientes/

Atlassian. (s. f.). *Comparación entre la arquitectura monolítica y la arquitectura de microservicios*. https://www.atlassian.com/es/microservices/microservices-architecture/microservices-vs-monolith#:~:text=Una%20arquitectura%20monol%C3%ADtica%20es%20un,e%20independiente%20de%20otras%20aplicaciones.

IBM documentation. (s. f.). https://www.ibm.com/docs/es/urbancode-deploy/6.2.4?topic=modeling-software-deployment

Martinekuan. (s. f.). *Estilo de arquitectura de microservicios - Azure Architecture Center*. Microsoft Learn. https://learn.microsoft.com/es-es/azure/architecture/guide/architecture-styles/microservices

¿Qué es AOS? - Explicación sobre la arquitectura orientada a servicios - AWS. (s. f.). Amazon Web Services, Inc. https://aws.amazon.com/es/what-is/service-oriented-architecture/

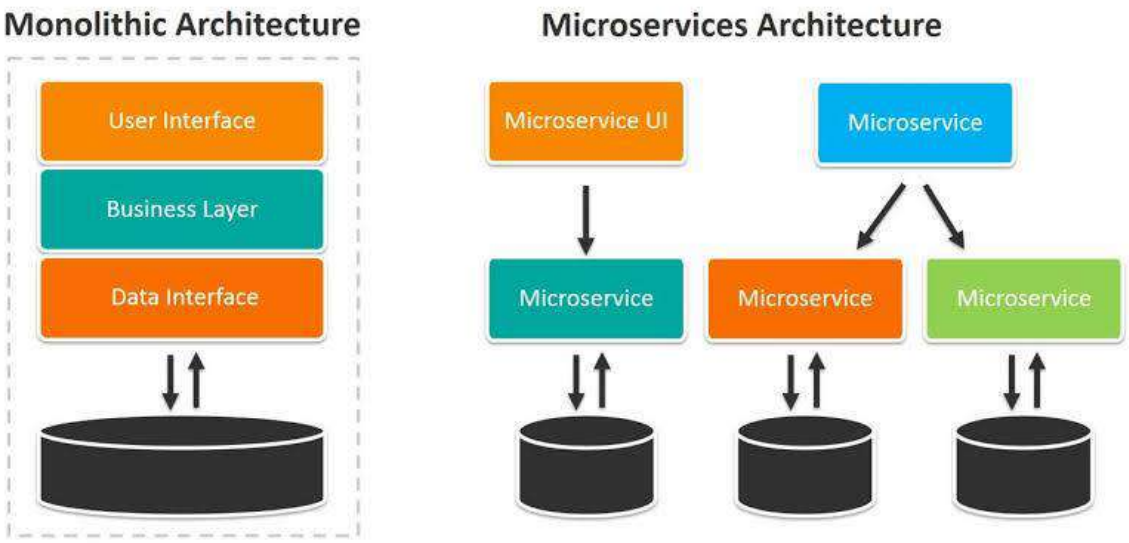
¿Qué es la arquitectura orientada a los servicios? (s. f.). https://www.redhat.com/es/topics/cloud-native-apps/what-is-service-oriented-architecture

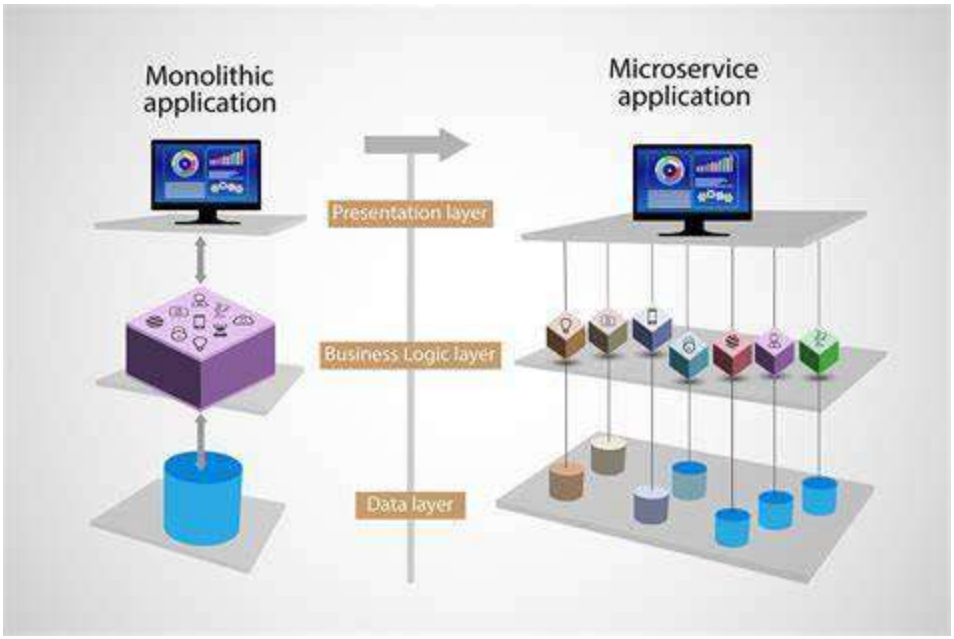
¿Qué es la SOA (arquitectura orientada a servicios)? | IBM. (s. f.). https://www.ibm.com/mx-es/topics/soa

Vilchez, E. D. O. (s. f.). ARQUITECTURA ORIENTADA a SERVICIOS (SOA). *es.linkedin.com*. https://es.linkedin.com/pulse/arquitectura-orientada-servicios-soa-eber-daniel-or%C3%A9-vilchez

Wrobel, M. (2023, 30 junio). ¿Qué es el despliegue de software? Definición, alcance y buenas prácticas. *Invgate*. https://blog.invgate.com/es/despliegue-de-software

Anexos


Monolithic Architecture
User Interface
Business Layer
Data Interface
Microservices Architecture
Microservice UI
Microservice
Microservice
Microservice
Microservice


Monolithic application
Microservice application
Presentation layer
Business Logic layer
Data layer