LENGUAJE AUDIOVISUAL

INVESTIGACION “MICROSERVICIOS”

DOCENTE: ING. LUIS

ESTUDIANTES: TAPIA ARCE PAMELA LIGIA
MAMANI HUANCA JOSUE ABRAHAM

30/09/2023

LIGIA
ptapiaarce@gmail.com

Estudiante: Pamela Ligia Tapia Arce

AQUITECTURA DE SOFTWARE

MICROSERVICIOS

Introducción:

Son un enfoque de desarrollo basado en un conjunto de componentes o servicios modulares que se intercomunican para dar como resultado una aplicación mayor, dentro de la cual lo primordial es la escalabilidad y la velocidad.

para la generación de una aplicación de servidor como un conjunto de servicios pequeños. Esto significa que una arquitectura de microservicios está orientada principalmente hacia el back-end, aunque el enfoque también se utiliza para el front-end. Cada servicio se ejecuta en su propio proceso y se comunica con otros procesos mediante protocolos como HTTP/HTTPS, WebSockets o AMQP. Cada microservicio implementa un dominio de un extremo a otro específico o una capacidad empresarial dentro de un determinado límite de contexto, y cada uno se debe desarrollar de forma autónoma e implementar de forma independiente. Por último, cada microservicio debe poseer su modelo de datos de dominio relacionado y su lógica del dominio (soberanía y administración de datos descentralizada), y podría basarse en otras tecnologías de almacenamiento de datos (SQL, NoSQL) y lenguajes de programación.

Objetivos

Objetivo principal

Demostrar las ventajas y desventajas de la arquitectura de los microservicios

Objetivo secundario

Mostrar su aplicabilidad

Desarrollo

a) **Definición:**

Es una arquitectura que divide una aplicación en una serie de servicios, que son implementados de manera independiente, las cuales se comunican a través de una API. Además, permite implementar nuevas funciones y hacer cambios mas rápido, con lo cual se ahorra tiempo ya que no se reescribe el código.

Estudiante: Pamela Ligia Tapia Arce

b) Características:

- **Varios servicios de componentes**
  - **i.** Ya que se componen de servicios de componentes individuales y poco vinculados que se pueden desarrollar, implementar, operar, cambiar y volver a implementar sin afectar al funcionamiento de los otros servicios o la integridad de la aplicación.
- **Fáciles de mantener y de probar**
  - **i.** Permiten a los equipos experimentar con nuevas funciones y revertirlas si estas no llegan a funcionar, con ello se facilita la actualización del código madre y acelera el tiempo de salida l mercado de nuevas funciones, simplificando el proceso de aislamiento y corrección de fallos y errores en los servicios individuales.
- **Pertenecen a equipos pequeños**
  - **i.** Adoptan las prácticas de la metodología ágil y de DevOps. De manera que logran trabajar independientemente y
- **Se organizan en torno a capacidades empresariales**
  - **i.** Los equipos son multifuncionales
- **La infraestructura es automatizada**
  - **i.** suelen utilizar prácticas de automatización de infraestructuras, como la integración continua (CI), la entrega continua (CD) y la implementación continua (también CD).

c) Beneficios

- Los componentes de la aplicación se pueden construir en diferentes lenguajes de programación
- Se pueden mantener los flujos individuales de desarrollo e implementación continuos
- Se pueden construir aplicaciones extremadamente escalables
- Es posible el uso de opciones de implementación de función como servicio nativas de la nube
- Frecuentemente resultan en costos operativos más bajos

Estudiante: Pamela Ligia Tapia Arce

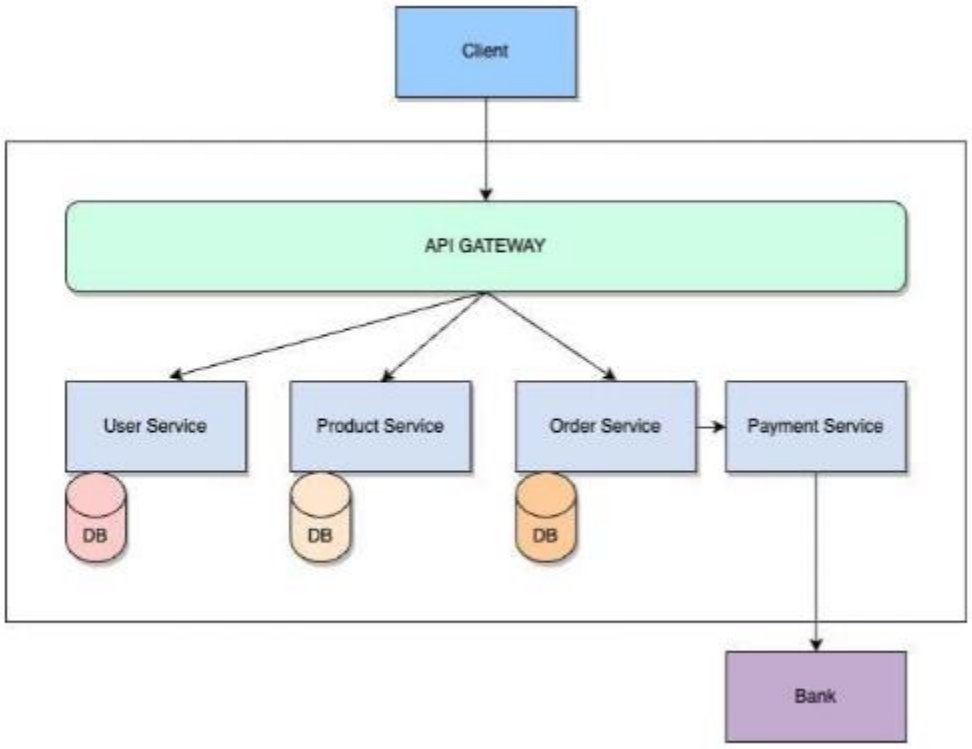
- El aislamiento y la estructura flexible permiten actualizaciones y mejoras compartimentadas

d) Tamaño

- crear libremente servicios acoplados para que tenga autonomía de desarrollo, implementación y escala, para cada servicio. Por supuesto, al identificar y diseñar microservicios, debe intentar que sean lo más pequeños posible, siempre y cuando no tenga demasiadas dependencias directas con otros microservicios. Más importante que el tamaño del microservicio es la cohesión interna que debe tener y su independencia respecto a otros servicios.

e) Arquitectura

- crear aplicaciones basadas en muchos servicios que se pueden implementar de forma independiente y que tienen ciclos de vida granulares y autónomos, lo que permite un mejor mantenimiento en sistemas complejos, grandes y altamente escalables.



Estudiante: Pamela Ligia Tapia Arce

- El Gateway/API: actúa como punto de entrada para las solicitudes de los clientes y enruta las peticiones a los microservicios correspondientes.

- User Service: el primer microservicio que se encargará de la gestión de usuarios.

- Product Service: será otro microservicio y se dedicará a atender todo lo relacionado con los productos.

- Order Service: gestionará las órdenes de compra, también como microservicio.

- Payment Service: como otro microservicio que procesa los pagos. Este microservicio es directamente invocado por el de pedidos, veremos más adelante que se puede hacer de varias maneras.

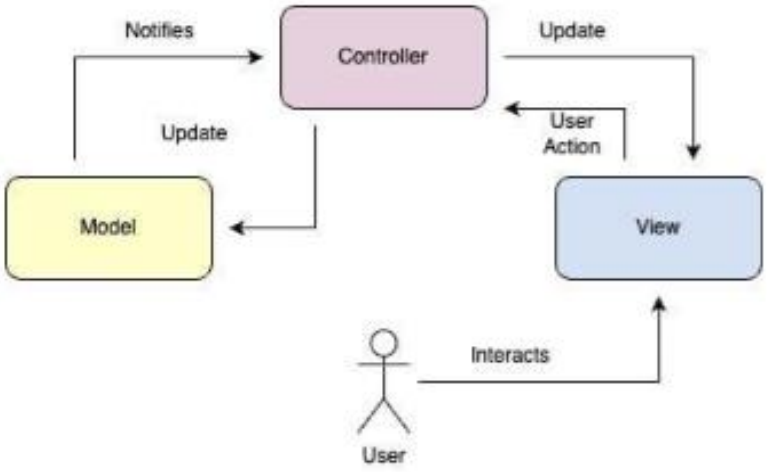
- BBDD: cada microservicio tiene su propia BBDD, también pueden compartirla y dividir schemas, tablas, colecciones, etc.

- Cliente: finalmente, el cliente o los clientes que interactúan con los microservicios a través del API Gateway.

f) Patrones base de microservicios

- **Patrón MVC**

Estudiante: Pamela Ligia Tapia Arce



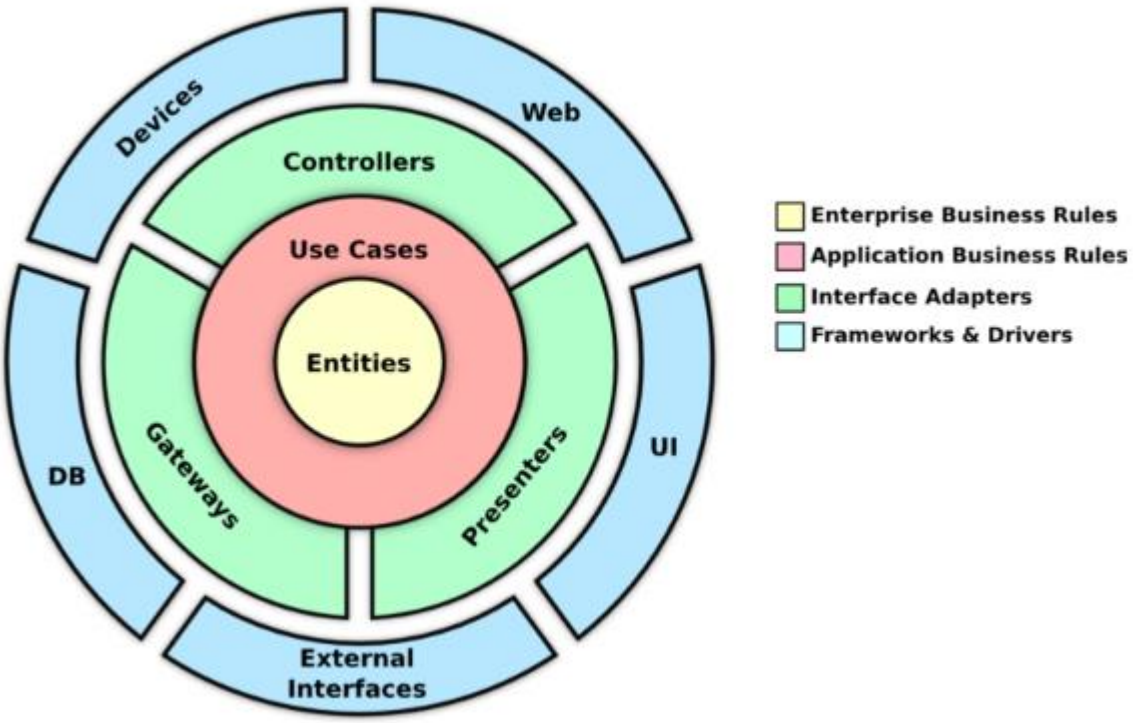
i. Modelo: representa la lógica de negocio y los datos subyacentes. Se encarga de gestionar el estado de la aplicación y proporciona métodos para acceder y manipular los datos.

ii. Vista: es la representación visual de los datos del Modelo. Muestra la información al usuario y se encarga de la presentación y la interfaz de usuario.

iii. Controlador: actúa como intermediario entre el Modelo y la Vista. Procesa las interacciones del usuario y decide cómo debe actualizarse el Modelo y la Vista en consecuencia.

- **Patrón Clean Architecture**

Estudiante: Pamela Ligia Tapia Arce



i. Entidades (Entities): esta capa contiene las entidades y reglas de negocio fundamentales de la aplicación. Representa el núcleo de la lógica de negocio, independiente de cualquier detalle técnico o interfaz de usuario.

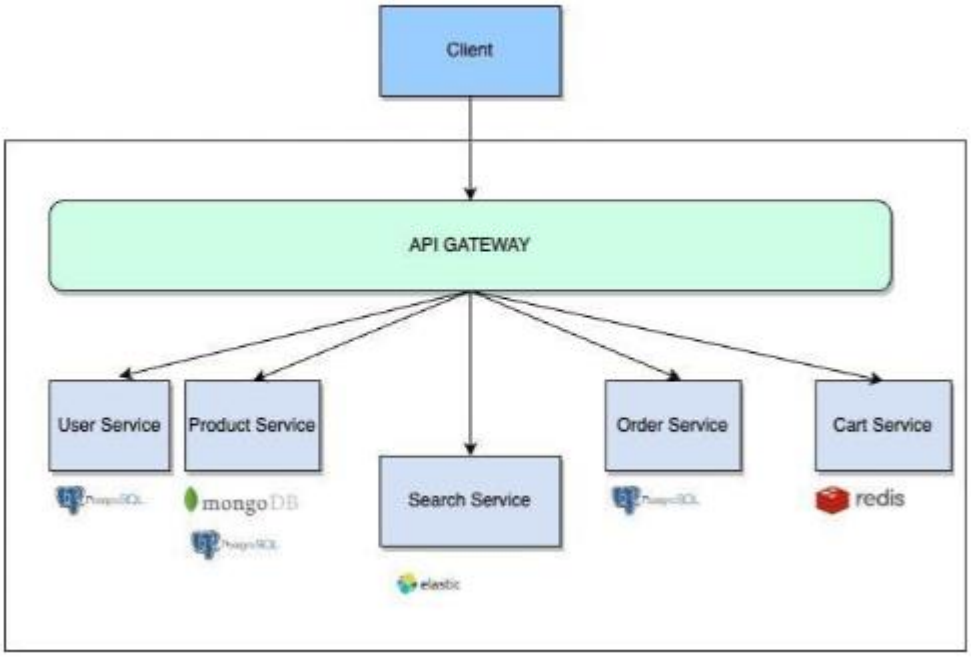
ii. Casos de uso (Use Cases): aquí se encuentran las interacciones de alto nivel que la aplicación puede realizar. Los casos de uso representan la lógica de la aplicación y utilizan las entidades del dominio para ejecutar flujos de trabajo y procesos.

iii. Adaptadores de interfaz de usuario (Interface Adapters): estos adaptadores se encargan de comunicarse con los casos de uso y traducir los datos y solicitudes de la interfaz de usuario a un formato comprensible para la lógica de negocio. Pueden incluir controladores web, presentadores, adaptadores para bases de datos, entre otros.

Estudiante: Pamela Ligia Tapia Arce

iv. Frameworks y herramientas externas (Frameworks and Drivers): esta capa contiene las herramientas y frameworks externos que la aplicación utiliza para interactuar con el mundo exterior, como bases de datos, servicios web, etc. Los adaptadores en esta capa se encargan de convertir los datos y solicitudes en un formato adecuado para el resto de la aplicación.

- **Database Microservice**



i. Independencia tecnológica: cada microservicio puede utilizar la tecnología de base de datos que mejor se adapte a sus requerimientos, sin estar limitado por una única tecnología compartida.

ii. Bajo acoplamiento: los microservicios no comparten esquemas ni tablas en la base de datos, lo que evita la necesidad de cambios en la base de datos cuando se realiza una actualización en un microservicio.

iii. Escalabilidad individual: cada microservicio puede escalar su base de datos según sus propias necesidades, sin afectar a otros microservicios.

iv. Facilita la mantenibilidad: los cambios en un microservicio no impactan en otros microservicios, lo que facilita el mantenimiento y evolución individual de cada uno.

v. Resiliencia y tolerancia a fallos: si un microservicio falla, no afectará la integridad de la base de datos de otros microservicios.

vi. Cumplimiento de límites de dominio: cada microservicio puede definir sus propias restricciones y políticas de seguridad en su propia base de datos.

g) Implementación monolítica Vs microservicios

- En el enfoque monolítico tradicional, la aplicación se escala mediante la clonación de toda la aplicación en varios servidores o máquinas virtuales. En el enfoque de microservicios, la funcionalidad se aísla en servicios más pequeños, por lo que cada servicio puede escalarse de forma independiente. El enfoque de los microservicios permite modificaciones ágiles e iteraciones rápidas de cada microservicio, ya que puede cambiar áreas específicas y pequeñas de aplicaciones complejas, grandes y escalables.

Estudiante: Pamela Ligia Tapia Arce

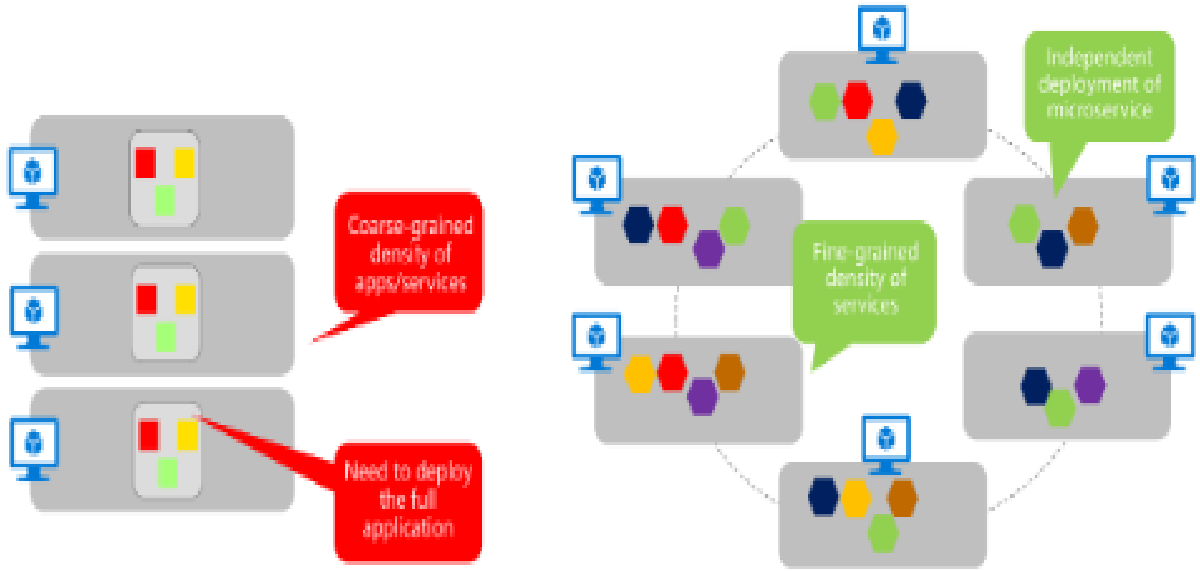


Figura 1[1]

h) Asociar DevOps con microservicios

- DevOps se puede explicar como una ideología centrada en herramientas. Básicamente, las herramientas y la tecnología se utilizan para simplificar el trabajo que deben realizar los desarrolladores. Junto con las herramientas mejoradas, la cooperación entre los equipos de operaciones y desarrollo ayuda a que los proyectos se completen de manera más rápida y eficiente.
- El nuevo modelo de flujo de trabajo de DevOps crea un cambio cultural importante. El solo uso de nuevas herramientas no hace que un plan de DevOps sea exitoso. En todo caso, las nuevas herramientas podrían inhibir el éxito si le hacen creer sólidamente que todo se implementará adecuadamente.
- Debido a la naturaleza de los microservicios, a menudo es necesario contar con procesos de DevOps sólidos para la implementación. El mayor uso de herramientas y prácticas compartidas más sólidas evita que los equipos se sientan abrumados por todas las piezas pequeñas y móviles. Reduce la interrupción o la carga de este cambio mientras continúa recibiendo los beneficios.

[1] Arquitectura monolítica vs microservicios

Estudiante: Pamela Ligia Tapia Arce

i) Basado en eventos

- Cuando las personas comienzan a experimentar con microservicios, a menudo utilizan de forma predeterminada técnicas familiares, por ejemplo, RESTful API. REST opera en un tipo de comunicación de solicitud-respuesta. El problema con este enfoque sincronizado es que deberá esperar una respuesta; los servicios se vuelven dependientes unos de otros. Si un servicio se ejecuta más lento o no responde, significa que el servicio que lo llamó se ejecutará más lento o fallará. Este acoplamiento puede significar perder algunos de los beneficios de una arquitectura de microservicios, creando una estructura más interdependiente similar a un estilo de arquitectura orientada a servicios (SOA).
- Si diseña sus servicios utilizando un modelo basado en eventos, puede asegurarse de que algunas partes de su aplicación continúen funcionando, mientras que otras partes podrían no estar involucradas, en lugar de que toda la aplicación deje de responder. Tome Netflix como ejemplo. En ocasiones, puede notar que el botón "continuar viendo" no aparece. Esto se debe a que un servicio específico no está disponible. Sin embargo, eso no significa que todo Netflix se detenga. Los usuarios aún podrán buscar programas y ver vistas previas, por lo que otros aspectos del servicio de Netflix aún están disponibles, aunque es posible que un servicio no lo esté.
- Los desarrolladores que adoptan completamente un enfoque de microservicios se dan cuenta de que la verdadera escalabilidad se habilita con una estructura flexible y una arquitectura basada en eventos. Un servicio puede ser asíncrono, realizar una acción, transmitir un mensaje y continuar con su función principal sin tener que esperar una respuesta de otro servicio.

j) Casos de uso

- **Adoptar opciones de implementación nativas de la nube:** aproveche la función sin servidor y la función como servicio para operaciones más eficientes y escalables.

Estudiante: Pamela Ligia Tapia Arce

- **Migrar la funcionalidad de las aplicaciones tradicionales:** descomponga los servicios de las grandes aplicaciones monolíticas para que se puedan mantener y escalar de forma independiente.
- **Aprovechar la arquitectura moderna de aplicaciones:** adopte patrones de aplicaciones de microservicio de estructura flexible y basados en eventos, con la capacidad de aprovechar diferentes lenguajes de programación según las necesidades del caso de uso. Por ejemplo, opte por funciones computacionalmente pesadas, Node.js para aplicaciones web rápidas, etc.

k) Aplicación:

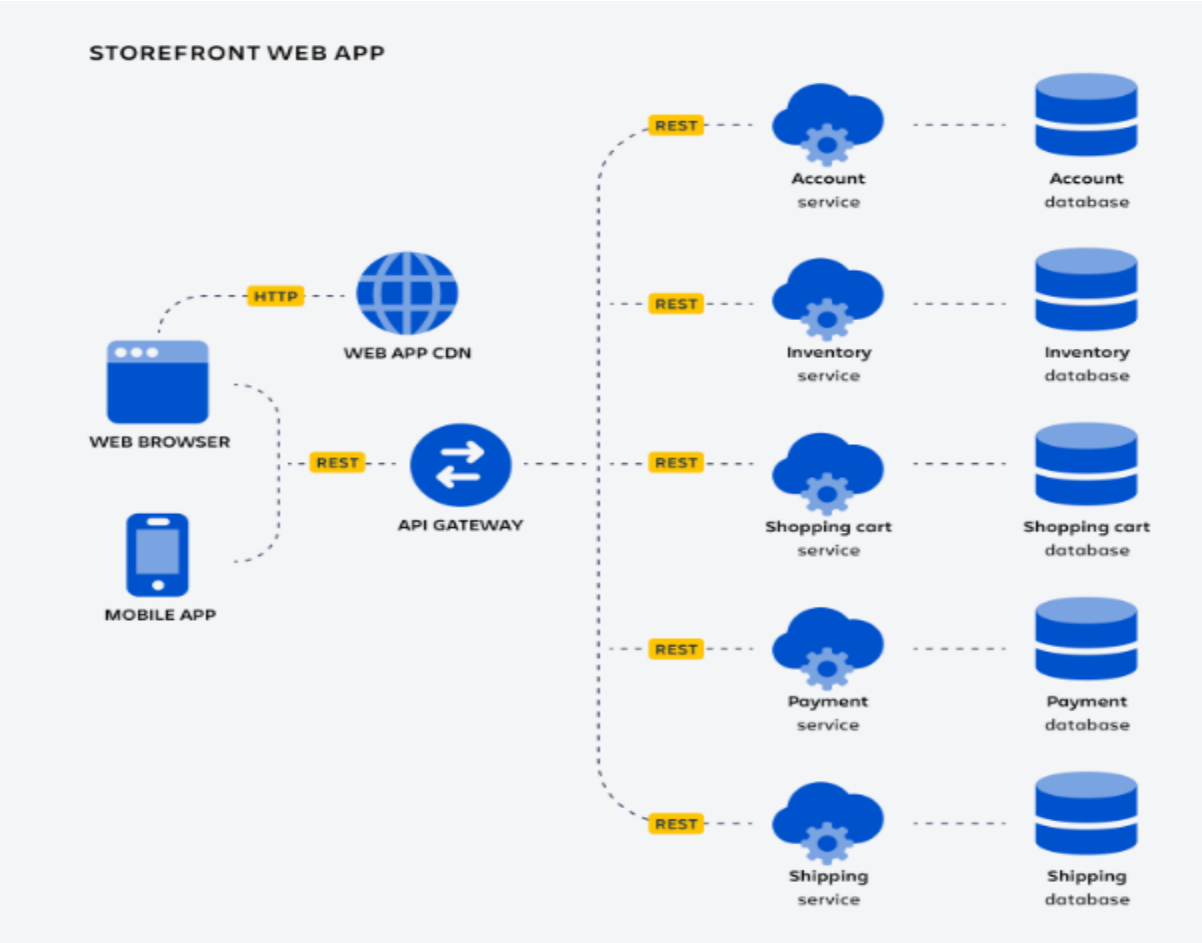


Figura2[2]

[2] Ejemplo de ecomers

Estudiante: Pamela Ligia Tapia Arce

Conclusión

La arquitectura de microservicios ha ganado popularidad debido a que sus características modulares conducen a la flexibilidad, escalabilidad y esfuerzo de desarrollo reducido. Su flexibilidad de implementación y el aumento de las opciones de implementación nativas de la nube sin servidor y su función como servicio (como AWS Lambda y Microsoft Azure Cloud Functions) crearon el entorno perfecto para que los microservicios prosperen en el panorama de TI actual. Estas plataformas en la nube permiten que los microservicios y las funciones se escalen de la inactividad a un gran volumen y viceversa, mientras que los clientes pagan solo por la capacidad informática que utilizan.

A medida que las empresas buscan continuamente ser más ágiles, reducir los obstáculos y mejorar los tiempos de entrega de las aplicaciones, la arquitectura de microservicios sigue aumentando en popularidad.

Imágenes

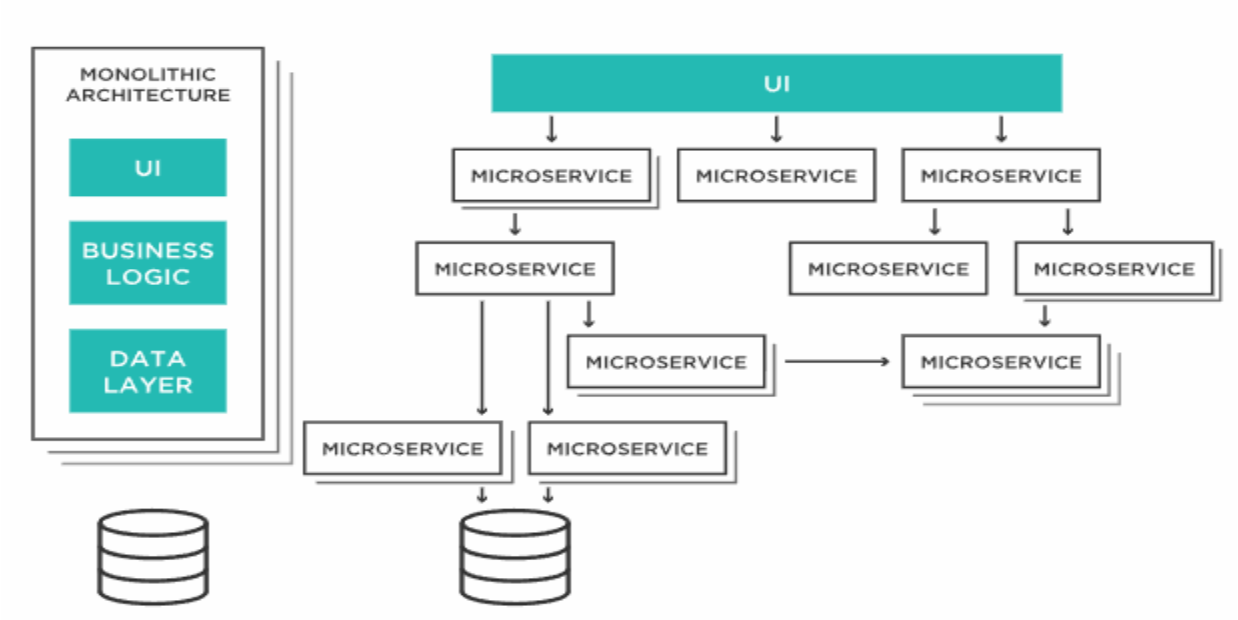


figura 3[3]

[3] Arquitectura monolítica vs microservicios

Estudiante: Pamela Ligia Tapia Arce

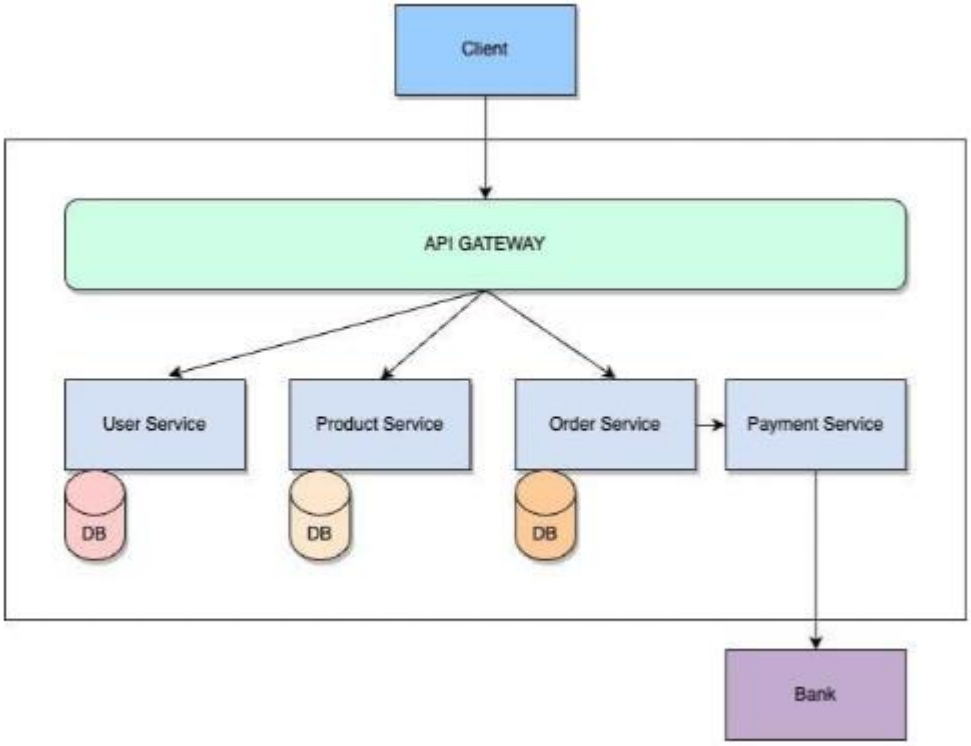


Figura4 [4]

Bibliografía

.NET, A. d. (s.f.). *Arquitectura de microservicios - .NET*.

Arquitectura de microservicios: qué es, v. y. (s.f.). *Arquitectura de microservicios: qué es, ventajas y desventajas* .

Center, E. d.-A. (s.f.). *Estilo de arquitectura de microservicios - Azure Architecture Center* .

microservicios, A. d. (s.f.). *Arquitectura de microservicios*.

microservicios?, ¿. e. (s.f.). *¿Qué es una arquitectura de microservicios?*

Patrones de arquitectura de microservicios, ¿. s.-P. (s.f.). *Patrones de arquitectura de microservicios, ¿qué son y qué ventajas… - Paradigma (paradigmadigital.com)*.

uso, Q. s. (s.f.). *Qué son Microservicios y ejemplos reales de uso* .

[4] Arquitectura de servicios sencilla

Estudiante: Pamela Ligia Tapia Arce