UNIVERSIDAD AUTÓNOMA DE CHIAPAS

FACULTAD DE CONTADURÍA Y ADMINISTRACIÓN
CAMPUS 1

Licenciatura en Ingeniería en Desarrollo y Tecnologías de Software

Act. 1.1 Definir los siguientes conceptos: Microservicios

6 “M”

Materia: Taller de Desarrollo 4

Docente: Luis Gutiérrez Alfaro

ALUMNO:
A 211387
Steven de Dios Montoya Hernández

TUXTLA GUTIÉRREZ, CHIAPAS
Viernes 18 de Agosto del 2023

API:

Las API son mecanismos que permiten a dos componentes de software comunicarse entre sí mediante un conjunto de definiciones y protocolos. Por ejemplo, el sistema de software del instituto de meteorología contiene datos meteorológicos diarios. La aplicación meteorológica de su teléfono “habla” con este sistema a través de las API y le muestra las actualizaciones meteorológicas diarias en su teléfono.

API significa “interfaz de programación de aplicaciones”. En el contexto de las API, la palabra aplicación se refiere a cualquier software con una función distinta. La interfaz puede considerarse como un contrato de servicio entre dos aplicaciones. Este contrato define cómo se comunican entre sí mediante solicitudes y respuestas. La documentación de su API contiene información sobre cómo los desarrolladores deben estructurar esas solicitudes y respuestas.

La arquitectura de las API suele explicarse en términos de cliente y servidor. La aplicación que envía la solicitud se llama cliente, y la que envía la respuesta se llama servidor. En el ejemplo del tiempo, la base de datos meteorológicos del instituto es el servidor y la aplicación móvil es el cliente.

Las API pueden funcionar de cuatro maneras diferentes, según el momento y el motivo de su creación.

Arquitectura:

Las arquitecturas de microservicios proporcionan un sistema moderno altamente escalable y distribuido

Los microservicios son una forma de arquitectura de software donde una aplicación se divide en un conjunto de servicios pequeños y autónomos, es decir: cada microservicio se puede implementar y ejecutar de forma independiente.

Esta arquitectura de microservicios ganó popularidad frente a la arquitectura monolítica para el diseño de aplicaciones, pues en el enfoque tradicional todos los elementos estaban contenidos en una sola aplicación. En cambio, los microservicios permiten un mayor nivel de flexibilidad y escalabilidad, además de tener mayores niveles de automatización de los centros de datos.

AWS:

Con la arquitectura de microservicios AWS Lambda, puedes desarrollar, implementar, operar y escalar cada componente sin afectar el funcionamiento de otros servicios. Además, cada uno de ellos está diseñado para un conjunto de capacidades y se orienta a solucionar un problema determinado.

En ese sentido, se diferencia de los enfoques monolíticos en que los procesos no se ejecutan como un solo servicio, haciendo mucho más fácil agregar o mejorar las características a medida que crece la base de código o surgen nuevas ideas.

Adicionalmente, reduce el riesgo de la disponibilidad de la aplicación, propio de las arquitecturas monolíticas porque muchos procesos dependientes y estrechamente relacionados incrementan las posibilidades e impacto del error en las operaciones.

Entre las ventajas de este enfoque cabe destacar:

- Facilita la innovación de las aplicaciones y acelera el tiempo de comercialización de las nuevas características.
- Acorta los tiempos del ciclo de desarrollo.
- Reducción de costos en la construcción y ejecución de sistemas de software.
- Escalado flexible
- Implementación sencilla, sin servidores que administrar.

Back End:

El Back-End en el contexto de microservicios se refiere a la implementación de la lógica de negocio, procesamiento de datos y gestión de la base de datos para cada microservicio individual. Aquí hay algunas consideraciones clave para el Back-End en microservicios:

Independencia y Escalabilidad: Cada microservicio puede ser desarrollado, probado, implementado y escalado de manera independiente. Esto permite un despliegue ágil y facilita la escalabilidad horizontal para manejar cargas variables.

Comunicación entre Microservicios: Los microservicios se comunican entre sí generalmente a través de protocolos de red, como HTTP/HTTPS o mensajería asíncrona. Las API bien definidas son esenciales para establecer cómo se comunican y comparten datos.

Base de Datos: Puedes optar por una base de datos por microservicio o utilizar bases de datos compartidas. Las bases de datos NoSQL, como MongoDB, y las bases de datos en memoria, como Redis, son populares en este enfoque.
Seguridad: Cada microservicio debe tener medidas de seguridad adecuadas para proteger los datos y la comunicación. Puedes implementar autenticación y autorización a nivel de microservicio y a nivel de API.

Monitorización y Depuración: Es importante tener herramientas de monitorización para rastrear el rendimiento y la salud de los microservicios. Los registros y las métricas son fundamentales para identificar problemas y realizar mejoras.

Orquestación y Gestión: En el caso de tener muchos microservicios, es útil utilizar herramientas de orquestación, como Kubernetes, para gestionar y automatizar su implementación y escalado.

Testing: Debido a la naturaleza distribuida de los microservicios, es esencial realizar pruebas exhaustivas, incluyendo pruebas de integración y de extremo a extremo.

Desarrollo y Mantenimiento: El desarrollo y mantenimiento de microservicios puede ser complejo debido a la cantidad de componentes involucrados. Se requiere un enfoque disciplinado en el diseño de las interfaces y la documentación.

Bifurcación:

Una bifurcación se refiere a la división de un servicio o componente en dos o más servicios independientes. Esta acción puede realizarse por diversas razones, como mejorar la escalabilidad, la separación de preocupaciones, la reutilización de código o para facilitar un desarrollo más ágil. Sin embargo, es importante planificar y gestionar cuidadosamente las bifurcaciones para evitar complicaciones no deseadas en la arquitectura.

Escalabilidad:

La **escalabilidad en microservicios** se refiere a la capacidad de estos para manejar un aumento en el tráfico y la carga de trabajo. Los microservicios permiten un escalado más eficiente que las arquitecturas de aplicaciones monolíticas. Para escalar una aplicación basada en microservicios de manera eficaz, los equipos también deben realizar un seguimiento de los objetivos de rendimiento y eficiencia. Un sistema de monitoreo eficaz junto con una estrategia de escalado puede ayudar a mantener un rendimiento óptimo para una aplicación basada en microservicios.

Flexibilidad:

La flexibilidad es una característica central de la arquitectura de microservicios. Consiste en la capacidad de desarrollar, desplegar y escalar componentes individuales de manera independiente. Cada microservicio se trata como una entidad autónoma, lo que permite cambios y actualizaciones sin afectar otras partes del sistema. Esto facilita el desarrollo paralelo, el despliegue selectivo, la escalabilidad granular y la adaptación a tecnologías específicas. Sin embargo, esta flexibilidad también conlleva la necesidad de gestionar la comunicación entre microservicios y la complejidad de la infraestructura.

La flexibilidad en la arquitectura de microservicios es una característica que permite a las aplicaciones de software adaptarse y evolucionar de manera ágil en un entorno cambiante. Se basa en el principio de dividir una aplicación en componentes pequeños y autónomos, conocidos como microservicios, que pueden ser desarrollados, desplegados y escalados de forma independiente. Esta flexibilidad ofrece varias ventajas clave:

Desarrollo Independiente: Los equipos de desarrollo pueden trabajar en microservicios diferentes al mismo tiempo, lo que agiliza el proceso de desarrollo y permite una innovación más rápida en diferentes áreas de la aplicación.

Despliegue Continuo: Los microservicios se pueden desplegar de manera independiente, lo que facilita la implementación continua de nuevas características, correcciones de errores y mejoras sin afectar a todo el sistema.
Escalabilidad Eficiente: La flexibilidad de los microservicios permite escalar componentes específicos en función de la demanda. Esto mejora la eficiencia en el uso de recursos y evita el sobreaprovisionamiento.

Tecnologías Diversas: Cada microservicio puede ser desarrollado utilizando tecnologías y lenguajes de programación adecuados para su funcionalidad específica. Esto permite utilizar las mejores herramientas para cada tarea.

Resiliencia a fallos: Debido a la independencia de los microservicios, un fallo en uno de ellos no necesariamente afectará a otros, lo que mejora la resiliencia del sistema en general.

Escalabilidad Organizativa: Los equipos pueden trabajar en microservicios específicos y tomar decisiones de diseño de manera autónoma, lo que permite una mayor escalabilidad organizativa y fomenta la creatividad.

Modularidad y Mantenimiento: Los microservicios separan áreas funcionales, lo que facilita el mantenimiento, la comprensión y la resolución de problemas a medida que la aplicación crece.

Adaptación a Requisitos Cambiantes: La flexibilidad inherente a los microservicios permite a las aplicaciones adaptarse rápidamente a los cambios en los requisitos y demandas del mercado.

Framework:

En programación, un framework es un marco de trabajo que tiene como objetivo facilitar la solución de problemas que pueden surgir al programar. Los frameworks aceleran el proceso de programar facilitando tareas como la organización del código o el trabajo en equipo dentro de un proyecto, por ejemplo.
En otras palabras, el objetivo de un framework en programación es facilitar la tarea de los programadores.

También hay que destacar que muchos de estos frameworks no sólo facilitan la organización del trabajo, sino que también ofrecen recursos desarrollados por otros programadores. Estos recursos pueden ser informes o códigos que pueden usarse para solucionar los problemas más habituales que se dan al llevar a cabo determinadas tareas.

Front End:

El enfoque de Front-End en una arquitectura de microservicios se centra en la forma en que se diseñan, desarrollan y gestionan las interfaces de usuario de las aplicaciones distribuidas. Aquí hay algunos aspectos clave a considerar al trabajar con el Front-End en microservicios:

Independencia de Microservicios: Cada microservicio puede tener su propia interfaz de usuario si es necesario. Esto significa que los equipos de Front-End pueden trabajar de manera independiente en las interfaces de usuario de cada microservicio, lo que facilita la paralelización del desarrollo.

Comunicación Asíncrona: En una arquitectura de microservicios, las interacciones entre el Front-End y los microservicios a menudo implican comunicación asíncrona a través de API. Las llamadas a los microservicios pueden realizarse a través de HTTP/HTTPS o mediante mensajería.

API Gateway: Un API Gateway es un componente que puede actuar como punto de entrada único para las solicitudes del Front-End. Puede redirigir las solicitudes a los microservicios correspondientes y manejar tareas como el enrutamiento y la autenticación.

Single Page Applications (SPA): Las SPAs son un enfoque popular en el desarrollo Front-End para aplicaciones de microservicios. Estas aplicaciones se cargan en una única página web y utilizan AJAX para actualizar contenido de manera dinámica sin necesidad de recargar la página.

Micro Front-Ends: Similar a los microservicios, los micro front-ends dividen la interfaz de usuario en componentes pequeños e independientes. Cada componente puede ser desarrollado, probado y desplegado por separado.

Gestión de Estado: Puedes utilizar bibliotecas como Redux, MobX o Context API (en React) para gestionar el estado de la aplicación, especialmente cuando interactúas con múltiples microservicios.

Caching: El uso de cachés en el Front-End puede mejorar el rendimiento al reducir las solicitudes repetitivas a los microservicios. Sin embargo, es importante considerar la coherencia de los datos en la caché.

Testing: Las pruebas de Front-End en microservicios deben abordar tanto las interacciones de la interfaz de usuario como las llamadas a los microservicios. Las pruebas de integración y de extremo a extremo son cruciales.

Monitorización y Rendimiento: La monitorización del Front-End y del rendimiento de las solicitudes a los microservicios es esencial para garantizar una experiencia de usuario fluida.

Adaptabilidad: Dado que los microservicios pueden cambiar y evolucionar independientemente, el Front-End debe ser capaz de adaptarse a las actualizaciones de los microservicios sin problemas.

IaaS:

IaaS, que significa "Infraestructura como Servicio" (del inglés, Infrastructure as a Service), es un modelo de implementación en la nube que proporciona a los usuarios recursos de infraestructura de TI virtualizados a través de Internet. En lugar de adquirir y mantener hardware y software físicos en sus propias instalaciones, las organizaciones pueden alquilar estos recursos según sea necesario, lo que brinda flexibilidad, escalabilidad y reducción de costos.

La infraestructura como servicio (IaaS) es un tipo de servicio de informática en la nube que ofrece recursos esenciales de proceso, almacenamiento y redes a petición que son de pago por uso. IaaS es uno de los cuatro tipos de servicios en la nube, junto con el software como servicio (SaaS), la plataforma como servicio (PaaS) y la tecnología sin servidor.

La migración de la infraestructura de tu organización a una solución de IaaS ayuda a reducir el mantenimiento de los centros de datos locales, a ahorrar dinero en los costes de hardware y a obtener información empresarial en tiempo real. Las soluciones de IaaS ofrecen la flexibilidad necesaria para escalar y reducir verticalmente los recursos de TI a petición. También ayudan a aprovisionar rápidamente nuevas aplicaciones y a aumentar la confiabilidad de la infraestructura subyacente.

IaaS permite evitar el coste y la complejidad de comprar y administrar servidores físicos e infraestructura de centro de datos. Cada recurso se ofrece como un componente de servicio aparte, y solo pagas por el tiempo que necesites un recurso concreto. Un proveedor de servicios informáticos en la nube como Azure administra la infraestructura, mientras que tú compras, instalas, configuras y administras tu propio software (sistemas operativos, middleware y aplicaciones).

Microservicios:

Los microservicios son un enfoque arquitectónico y organizativo para el desarrollo de software donde el software está compuesto por pequeños servicios independientes que se comunican a través de API bien definidas. Los propietarios de estos servicios son equipos pequeños independientes.

Las arquitecturas de microservicios hacen que las aplicaciones sean más fáciles de escalar y más rápidas de desarrollar. Esto permite la innovación y acelera el tiempo de comercialización de las nuevas características.

Arquitectura monolítica en comparación con la arquitectura de microservicios
Con las arquitecturas monolíticas, todos los procesos están estrechamente asociados y se ejecutan como un solo servicio. Esto significa que, si un proceso de una aplicación experimenta un pico de demanda, se debe escalar toda la arquitectura. Agregar o mejorar las características de una aplicación monolítica se vuelve más complejo a medida que crece la base de código. Esta complejidad limita la experimentación y dificulta la implementación de nuevas ideas. Las arquitecturas monolíticas aumentan el riesgo de la disponibilidad de la aplicación porque muchos procesos dependientes y estrechamente vinculados aumentan el impacto del error de un proceso.

Con una arquitectura de microservicios, una aplicación se crea con componentes independientes que ejecutan cada proceso de la aplicación como un servicio. Estos servicios se comunican a través de una interfaz bien definida mediante API ligeras. Los servicios se crean para las capacidades empresariales y cada servicio desempeña una sola función. Debido a que se ejecutan de forma independiente, cada servicio se puede actualizar, implementar y escalar para satisfacer la demanda de funciones específicas de una aplicación.

PaaS:
PaaS es una manera de disponibilizar plataformas de desarrollo de aplicaciones por medio de internet, como un servicio. Esas plataformas empresariales en nube son extremadamente seguras y ayudan a su empresa a conducir todos los departamentos con aplicaciones personalizadas, creadas por desarrolladores, departamentos de TI, usuarios de negocios o hasta mismo todos ellos, que pueden ser fácilmente adaptados de acuerdo con las necesidades, crecimiento y cambios de su negocio.

La Plataforma como Servicio es un modelo comprobado para el desarrollo e implementación de aplicaciones sin las complicaciones de mantener una infraestructura completa de hardware y software en su empresa. Las soluciones PaaS son simples, seguras, dimensionables y cuentan con los más recientes recursos y funcionalidades, evitando la necesidad de constantes actualizaciones.

Plataforma como servicio (PaaS) es un entorno de desarrollo e implementación completo en la nube, con recursos que permiten entregar todo, desde aplicaciones sencillas basadas en la nube hasta aplicaciones empresariales sofisticadas habilitadas para la nube. Compras los recursos que necesitas a un proveedor de servicios en la nube, a los que tienes acceso a través de una conexión segura a Internet, pero solo pagas por el uso que haces de ellos.

Al igual que IaaS, PaaS incluye infraestructura (servidores, almacenamiento y redes), pero también middleware, herramientas de desarrollo, servicios de inteligencia empresarial (BI), sistemas de administración de bases de datos, etc. PaaS está diseñado para sustentar el ciclo de vida completo de las aplicaciones web: compilación, pruebas, implementación, administración y actualización.

Servicio:
En el contexto de la arquitectura de microservicios, un "servicio" se refiere a una unidad independiente y autónoma de funcionalidad de software que se enfoca en realizar una tarea específica o proporcionar una característica particular de una aplicación más grande. Cada servicio es autocontenido y opera como una entidad separada, lo que permite modularidad, escalabilidad y mantenimiento eficiente en el desarrollo de software.

Los servicios de nube son infraestructuras, plataformas o sistemas de software que los proveedores externos alojan y ponen a disposición de los usuarios a través de Internet.

SaaS (de las siglas en inglés de "Software as a Service") o software como servicio, es un término que se usa para describir cuando los usuarios "rentan" o usan prestado un software en línea, en lugar de comprarlo e instalarlo en sus propios equipos de hardware.

Servidor:
Un servidor en microservicios es una aplicación de servidor que se compone de muchos servicios pequeños e independientes que se comunican entre sí.

En el contexto de la arquitectura de microservicios, el término "servidor" se utiliza para referirse a una instancia o implementación de un microservicio específico. Un servidor de microservicios es una entidad independiente que aloja y ejecuta un componente de la aplicación, y se encarga de manejar solicitudes y proporcionar respuestas a través de una interfaz de programación de aplicaciones (API) bien definida.

Cada servidor de microservicios es responsable de una funcionalidad específica y opera de manera autónoma, lo que permite modularidad, escalabilidad y despliegue independiente en el entorno de microservicios. Cada servidor puede ser desarrollado, probado y actualizado por separado, lo que facilita la entrega continua y permite a los equipos de desarrollo trabajar de manera eficiente y colaborativa en diferentes partes de la aplicación.

SOAP:
SOAP (Simple Object Access Protocol), es un protocolo que nos permitirá realizar servicios web sin estado, a través de TCP y con un formato XML.

Ventajas SOAP
Entre sus ventajas podemos encontrar que al funcionar a través del protocolo de transporte TCP, se pueden utilizar diferentes protocolos de aplicación como: HTTP, SMTP o JMS. También nos brinda la posibilidad de generar cliente/servidor en distintos lenguajes de programación. Y está ampliamente estandarizado, por lo cual hay reglas concretas para formar el mensaje, el contrato entre cliente/servidor o el formato de los datos a enviar, siempre XML.

Debido a estas cualidades, SOAP es ampliamente utilizado en entornos empresariales. Donde es requerida la existencia de un contrato claro entre cliente y servidor, y además la seguridad en las comunicaciones es muy importante.
SOAP está diseñado para permitir la integración entre diferentes sistemas y lenguajes de programación, lo que significa que se puede automatizar el intercambio de información entre diferentes sistemas y plataformas.

XML:
El lenguaje de marcado extensible (XML) permite definir y almacenar datos de forma compatible. XML admite el intercambio de información entre sistemas de computación, como sitios web, bases de datos y aplicaciones de terceros. Las reglas predefinidas facilitan la transmisión de datos como archivos XML a través de cualquier red, ya que el destinatario puede usar esas reglas para leer los datos de forma precisa y eficiente.

En los microservicios, la comunicación entre servicios se basa comúnmente en la transferencia de datos en formato JSON debido a su simplicidad, ligereza y facilidad de análisis en diferentes lenguajes de programación. JSON se ha convertido en el formato preferido para intercambiar datos entre los microservicios debido a su capacidad para representar estructuras de datos de manera concisa y su amplia adopción en muchas tecnologías modernas.

Aunque XML todavía puede ser utilizado en algunos casos, como en sistemas heredados o en integraciones con tecnologías que utilizan XML, su uso no es tan común en la mayoría de las implementaciones de microservicios actuales.

En resumen, aunque XML es un formato de intercambio de datos válido y ha sido ampliamente utilizado en el pasado, en el contexto de arquitectura de microservicios, JSON se ha convertido en la opción preferida para la comunicación entre servicios debido a su simplicidad y eficiencia.

Web Services:
Un Web Service, o Servicio Web, es un método de comunicación entre dos aparatos electrónicos en una red. Las aplicaciones escritas en varios lenguajes de programación que funcionan en plataformas diferentes pueden utilizar web services para intercambiar información a través de una red.

La principal diferencia entre microservicios y servicios web es que los microservicios se refieren a un enfoque del desarrollo de aplicaciones en el que una gran aplicación se construye como un conjunto de componentes o servicios modulares, mientras que los servicios web se refieren a un conjunto de estándares o protocolos que permiten que varias aplicaciones se comuniquen entre sí a través de la World Wide Web

Los microservicios se pueden desarrollar, implementar y escalar de forma independiente, mientras que los servicios web se basan en la infraestructura de la aplicación subyacente. Los microservicios usan protocolos livianos como REST o gRPC, mientras que los servicios web tradicionalmente usan SOAP o XML-RPC.

Web:
Web es una palabra inglesa que significa red o telaraña. Se designa como ‘la web’ al sistema de gestión de información más popular para la trasmisión de datos a través de internet. La web es el diminutivo de world wide web o www, cuyas tecnologías para su funcionamiento (HTML, URL, HTTP) fueron desarrolladas en el año 1990 por Tim Berners Lee.

Balanceador:
Un balanceador es un dispositivo que se utiliza para distribuir el tráfico de red entre varios servidores. Su función es igualar el trabajo de los servidores para que las aplicaciones respondan eficazmente. Un balanceador de carga es una herramienta que direcciona a un cliente al servidor web que se encuentre con mayor disponibilidad entre los que cuentan con el mismo contenido. También existe el balanceador / agregador de anchos de banda, que se encarga de gestionar más de una salida a Internet para conseguir más anchos de banda en nuestras ubicaciones.

Un balanceador de carga en el contexto de la arquitectura de microservicios es una pieza fundamental de la infraestructura que distribuye el tráfico de las solicitudes entre múltiples instancias de un mismo servicio o entre diferentes servicios. El objetivo principal de un balanceador de carga es mejorar la disponibilidad, escalabilidad y rendimiento de una aplicación distribuida, asegurando que las solicitudes se manejen de manera eficiente y equitativa.

REST:
Sus siglas vienen de Representational State Transfer, es un estilo de arquitectura de software para realizar una comunicación cliente-servidor. Se apoya en el protocolo HTTP para la comunicación al servidor y los mensajes que se envían y reciben pueden estar en XML o JSON.

En el contexto de la arquitectura de micro servicios, REST se refiere a la forma en que los microservicios se comunican entre sí y con el mundo exterior utilizando los principios de REST. Esto implica el diseño y desarrollo de APIs (Interfaces de Programación de Aplicaciones) que siguen las convenciones REST para permitir la comunicación y la interacción entre los componentes del sistema.

Bibliografía

Microservicios y Arquitectura de Software:

Newman, S. (2018). Microservicios: Patrones para el diseño de aplicaciones. O'Reilly Media.

López, A., & Díaz, J. (2019). Arquitectura de Microservicios. Universidad Nacional de Colombia.

REST y Comunicación en Microservicios:

Castro, L. J. (2018). API REST: Fundamentos y diseño. Marcombo.

Reyes, E. (2019). Desarrollo de Aplicaciones Móviles y Web Services con Symfony 3. Universidad Politécnica de Valencia.

IaaS, PaaS y Despliegue en la Nube:

Martínez, J. M. G., & Molina, M. G. (2018). Cloud Computing: Servicios en la nube. Casos prácticos. Ediciones ENI.

García, R. R. (2019). Introducción a las Plataformas de Servicio en la Nube (PaaS). Universidad Politécnica de Madrid.