Centro Universitario de Ciencias Exactas e Ingenierías

CUCEI

Computación Tolerante a Fallas

A Netflix guide to Microservices

Michel Emanuel López Franco

Gutiérrez Galán Ruben Alejandro

Código: 214798315

Contenido

Introducción: 3
Microservicios: 3
Arquitectura de microservicios: 3
Ventajas de los microservicios: 3
Desventajas de los microservicios: 4
Usos comunes de los microservicios: 5
¿Qué es ELB?: 5
Características: 5
Tipos en AWS: 6
ZUUL: 6
Características: 6
Uso en Netflix: 6
Zuul 2: 6
¿Qué es un API?: 6
Principales características de una API: 7
Tipos de API: 7
Conclusion: 8
Bibliografía: 8

Introducción:

En este reporte se busca expandir un poco mas sobre los temas explorados en el material proporcionado por el profesor, hablaremos en la medida de lo posible acerca de los microservicios, cuales son los impactos que han tenido en el avance de las plataformas de stream, cuales son sus ventajas y desventajas y en que casos es mejor su uso y en cuáles no.

Microservicios:

Arquitectura de microservicios:

La arquitectura de microservicios es un enfoque para desarrollar una única aplicación como un conjunto de pequeños servicios, cada uno ejecutándose en su propio proceso y comunicándose mediante mecanismos livianos, a menudo una API de recursos HTTP. Aunque esta definición es técnica y precisa, puede que no proporcione suficiente contexto.

Para entenderlo mejor, imaginemos una aplicación como un conjunto de bloques de construcción independientes. Cada uno de estos bloques es un servicio pequeño y especializado que realiza una tarea específica. Estos servicios pueden ejecutarse de manera independiente y comunicarse entre sí de manera eficiente.

En lugar de tener una aplicación monolítica, donde todas las funciones están agrupadas y dependen unas de otras, la arquitectura de microservicios descompone la aplicación en partes más pequeñas y manejables. Cada servicio puede ser desarrollado, implementado y escalado de forma independiente, lo que facilita la adaptación a cambios y mejoras incrementales.

La comunicación entre estos servicios suele realizarse a través de interfaces livianas, como API RESTful, que utilizan el protocolo HTTP. Esto permite una integración más sencilla y flexible entre los distintos servicios.

Ventajas de los microservicios:

- Despliegue Independiente: Los microservicios pueden ser desarrollados, desplegados y escalados de forma independiente. Esto permite a los equipos de desarrollo actualizar o agregar nuevas características a un servicio sin afectar a la totalidad de la aplicación.
- Escalabilidad: Al ser unidades independientes, los microservicios pueden ser escalados de manera individual según la demanda. Esto mejora la eficiencia en el uso de recursos, ya que solo los servicios específicos que requieren escalabilidad se ven afectados.
- Mantenimiento Facilitado: La modularidad inherente de los microservicios facilita el mantenimiento. Cambios o actualizaciones en un servicio no afectan a los demás, lo que simplifica la corrección de errores y la introducción de mejoras.
- Tecnología y Lenguaje de Programación Diversificados: Cada microservicio puede ser desarrollado utilizando diferentes tecnologías o lenguajes de programación según sea más apropiado para su función específica. Esto permite a los equipos utilizar las mejores herramientas para cada tarea.
- Mejora en la Resiliencia: En un entorno de microservicios, si un servicio falla, no compromete la integridad de toda la aplicación. La arquitectura distribuida y la independencia de los servicios ayudan a crear sistemas más resilientes.

- Facilita la Innovación Continua: La capacidad de desplegar y actualizar partes específicas de una aplicación de manera rápida y sin afectar otras áreas facilita la innovación continua. Los equipos pueden experimentar con nuevas características de manera más eficiente.
- Facilita el Desarrollo por Equipos Pequeños: Los equipos pueden trabajar de manera más autónoma en servicios específicos, lo que facilita la colaboración y mejora la velocidad de desarrollo. Cada equipo puede enfocarse en la funcionalidad de su microservicio sin depender en gran medida de otros equipos.
- Facilita la Adopción de la Nube: La arquitectura de microservicios es compatible con entornos en la nube, lo que facilita la adopción de servicios en la nube y permite aprovechar las características y beneficios que ofrece.

Desventajas de los microservicios:

- Complejidad Operativa: Gestionar un sistema distribuido compuesto por varios microservicios puede ser más complejo que gestionar una aplicación monolítica. Se requiere una infraestructura robusta para gestionar la comunicación, la monitorización y el despliegue de servicios.
- Comunicación entre Servicios: La comunicación entre microservicios a menudo se realiza a través de la red, lo que puede aumentar la latencia. La gestión de la comunicación entre servicios y la prevención de cuellos de botella son desafíos importantes.
- Consistencia de Datos: Mantener la consistencia de los datos a lo largo de los microservicios puede ser un desafío. La gestión de transacciones distribuidas y la coherencia de los datos requieren cuidado y planificación.
- Testing Complejo: La prueba de sistemas distribuidos es más compleja que la prueba de una aplicación monolítica. Las pruebas de integración y las pruebas de extremo a extremo son esenciales para asegurar que todos los microservicios funcionen correctamente juntos.
- Seguridad: La gestión de la seguridad en una arquitectura de microservicios puede ser más desafiante debido a la diversidad de servicios y la necesidad de proteger las comunicaciones entre ellos.
- Desafíos en la Migración: La transición de una arquitectura monolítica a microservicios puede ser un proceso complicado y requiere una planificación cuidadosa. La migración de datos y la coordinación entre los servicios durante el proceso pueden ser desafiantes.
- Costos de Operación: Aunque la escalabilidad independiente puede ser una ventaja, también puede dar lugar a costos operativos más altos. La necesidad de gestionar y escalar varios servicios puede aumentar los costos de infraestructura y de mantenimiento.
- Dificultad en Entornos Pequeños: En proyectos pequeños o equipos con recursos limitados, la complejidad adicional de la arquitectura de microservicios puede superar sus beneficios. En estos casos, una arquitectura monolítica puede ser más adecuada.
- Gestión de Versiones: La gestión de versiones de los microservicios puede ser un desafío. Coordinar las actualizaciones de servicios sin interrumpir la funcionalidad general del sistema requiere una planificación cuidadosa.

Usos comunes de los microservicios:

- Escalabilidad y Despliegue Independiente: Permite escalar y desplegar servicios de manera independiente según la demanda. Esto es especialmente beneficioso en aplicaciones con componentes que tienen requisitos de escalabilidad diferentes.
- Desarrollo Ágil y Rápido: Facilita el desarrollo ágil y rápido al permitir que equipos pequeños trabajen de manera autónoma en servicios específicos. Esto mejora la velocidad de desarrollo y la capacidad de respuesta a los cambios.
- Entornos Empresariales Complejos: En grandes empresas con sistemas complejos, la arquitectura de microservicios puede ayudar a dividir la complejidad en partes más manejables y favorecer la evolución independiente de cada servicio.
- Integración de Tecnologías Diversas: Permite el uso de diferentes tecnologías y lenguajes de programación para cada microservicio, lo que es útil cuando se requiere la integración de sistemas heredados o tecnologías específicas.
- Aplicaciones Empresariales Escalables: En aplicaciones empresariales que necesitan escalar de manera eficiente, los microservicios ofrecen una solución para gestionar la carga y adaptarse a los cambios en la demanda sin afectar toda la aplicación.
- Plataformas en la Nube: La arquitectura de microservicios es compatible con entornos en la nube, lo que facilita la implementación y el escalado de servicios en plataformas como AWS, Azure o Google Cloud.
- Desarrollo de Aplicaciones Web y Móviles: Es adecuada para el desarrollo de aplicaciones web y móviles, donde la modularidad y la capacidad de respuesta a las demandas del usuario son cruciales.
- Ecosistemas de Software: Puede ser útil en la construcción de ecosistemas de software donde diferentes servicios realizan funciones específicas y pueden ser reutilizados en diferentes contextos.
- Empresas con Cultura DevOps: La arquitectura de microservicios se alinea bien con las prácticas de DevOps al permitir la entrega continua y la implementación automatizada de servicios independientes.
- Facilita la Innovación Continua: Permite a las organizaciones adoptar un enfoque de innovación continua al facilitar la introducción rápida y eficiente de nuevas características y mejoras.

¿Qué es ELB?:

ELB (Elastic Load Balancer):

Definición: Elastic Load Balancer es un servicio de balanceo de carga proporcionado por proveedores de servicios en la nube como Amazon Web Services (AWS). Su función principal es distribuir el tráfico de red de manera equitativa entre múltiples instancias (o servidores) para garantizar una distribución uniforme de la carga y mejorar la disponibilidad y la tolerancia a fallos.

Características:

- Balanceo de carga: Distribuye el tráfico de manera equitativa entre múltiples instancias para evitar la sobrecarga en un único servidor.
- Escalabilidad: Se adapta dinámicamente al aumento o disminución de la carga de trabajo.

- Disponibilidad: Proporciona alta disponibilidad al redirigir el tráfico lejos de instancias no saludables.

Tipos en AWS:

- Application Load Balancer (ALB): Opera a nivel de la capa de aplicación y es adecuado para aplicaciones web modernas.
- Network Load Balancer (NLB): Opera a nivel de la capa de transporte y es ideal para cargas de trabajo TCP/UDP.
- Classic Load Balancer: Proporciona balanceo de carga básico y soporta tanto aplicaciones web como aplicaciones no web.

ZUUL:

Definición: Zuul es un servidor de puerta de enlace (gateway) que se utiliza comúnmente en arquitecturas de microservicios. Desarrollado por Netflix, Zuul actúa como un punto de entrada único para manejar solicitudes y enrutamiento en un sistema distribuido.

Características:

- Enrutamiento Dinámico: Permite el enrutamiento dinámico de solicitudes a servicios específicos en función de reglas configurables.
- Filtrado: Ofrece capacidades de filtrado para modificar solicitudes o respuestas en función de ciertos criterios.
- Gestión de Reintentos: Puede gestionar automáticamente los reintentos de solicitudes en caso de fallos transitorios.
- Autenticación y Autorización: Proporciona funciones de autenticación y autorización centralizadas.

Uso en Netflix:

En Netflix, Zuul es parte integral de su arquitectura de microservicios y se utiliza para enrutar el tráfico entre diferentes servicios, aplicar filtros de seguridad y realizar otras funciones de gestión de solicitudes.

Zuul 2:

Existe una versión más reciente de Zuul llamada Zuul 2, que fue diseñada para ser más modular y extensible.

Tanto ELB como Zuul son herramientas poderosas en arquitecturas distribuidas, y su elección dependerá de los requisitos específicos del entorno y de las preferencias tecnológicas. Mientras que ELB se centra en el balanceo de carga y la alta disponibilidad en entornos de nube, Zuul está diseñado para gestionar el enrutamiento y la gestión de solicitudes en arquitecturas de microservicios.

¿Qué es un API?:

API (Interfaz de Programación de Aplicaciones):

Una Interfaz de Programación de Aplicaciones (API) es un conjunto de reglas y definiciones que permiten la comunicación y la interacción entre diferentes softwares. Esta interfaz establece cómo

los componentes de software deben interactuar entre sí, permitiendo que aplicaciones o servicios se comuniquen y compartan datos o funcionalidades de manera estructurada y estandarizada.

Principales características de una API:

- Puntos de Acceso (Endpoints):
  - Una API proporciona puntos de acceso bien definidos (URLs o URI) que permiten a los desarrolladores interactuar con los servicios o recursos que la API expone.
- Protocolos de Comunicación:
  - Define los protocolos y métodos de comunicación que se deben utilizar. Los protocolos comunes incluyen HTTP/HTTPS para servicios web y REST o SOAP para la transferencia de datos.
- Formatos de Datos:
  - Especifica los formatos de datos que se utilizan para la comunicación. Los formatos comunes son JSON (JavaScript Object Notation) y XML (eXtensible Markup Language).
- Operaciones Disponibles:
  - Detalla las operaciones y acciones que se pueden realizar a través de la API. Por ejemplo, una API de mapas podría proporcionar operaciones para buscar ubicaciones o calcular rutas.
- Autenticación y Autorización:
  - Define cómo se autentican los usuarios o aplicaciones para acceder a la API y cómo se gestionan los permisos (autorización) para realizar determinadas operaciones.
- Documentación:
  - Una buena API incluye documentación clara que explica cómo utilizarla, qué recursos están disponibles y cómo interactuar con ellos. La documentación suele incluir ejemplos y descripciones detalladas.
- Versionamiento:
  - Las API a menudo incluyen números de versión para permitir cambios y actualizaciones sin romper la compatibilidad con las versiones anteriores.

Tipos de API:

- APIs Web (Web APIs):
  - Permiten la comunicación entre sistemas a través de la web, utilizando protocolos estándar como HTTP.
- APIs de Bibliotecas:
  - Proporcionan funciones y procedimientos que los desarrolladores pueden utilizar directamente en sus aplicaciones.
- APIs de Sistemas Operativos:
  - Permiten a las aplicaciones interactuar con funciones del sistema operativo, como el sistema de archivos o la gestión de memoria.
- APIs de Hardware:
  - Facilitan la interacción con hardware, como API de gráficos para interactuar con tarjetas gráficas.

Conclusion:

Los microservicios permitieron un mejor desarrollo de APIs web y algunas otras, permitió la descomposición de las arquitecturas monolíticas en componentes mas pequeños y por supuesto mas manejables, lo que a su vez permite dar un mejor mantenimiento y solución en caso de que falle.

La arquitectura de microservicios a su vez permitió escalar componentes individuales según sea su demanda, lo que proporciono flexibilidad y eficiencia en el uso de recursos. Se facilito el desarrollo ágil al permitir que equipos pequeños e independientes trabajen en microservicios específicos. Cada equipo siendo responsable de su propio servicio, lo que acelera la entrega de nuevas características, como podemos ver la implementación de microservicios es una gran ayuda en la programación moderna.

Bibliografía:

- InfoQ. (2017, February 22). *Mastering Chaos - A Netflix Guide to Microservices* [Video]. YouTube. https://www.youtube.com/watch?v=CZ3wIuvmHeM
- "Building Microservices" de Sam Newman
- "Microservices Patterns: With examples in Java" de Chris Richardson.
- "APIs: A Strategy Guide" de Daniel Jacobson, Greg Brail y Dan Woods.