UNIVERSIDAD AUTÓNOMA DE CHIAPAS

FACULTAD DE CONTADURÍA Y ADMINISTRACIÓN
CAMPUS 1

Licenciatura en Ingeniería en Desarrollo y Tecnologías de Software

Act. 2.5 Investigar sistemas de información con arquitectura basadas en microservicios.

6 “M”

Materia: Compiladores

Docente: Luis Gutiérrez Alfaro

ALUMNO:

A 211387
Steven de Dios Montoya Hernández
GIT:

TUXTLA GUTIÉRREZ, CHIAPAS

Sábado, 9 de Septiembre de 2023, 23:59

Arquitectura Basada en Microservicios.

Esta arquitectura ofrece una serie de ventajas, entre las que se incluyen:

- **Flexibilidad**: Los microservicios se pueden desarrollar, implementar y actualizar de forma independiente, lo que facilita la adaptación a los cambios en los requisitos del negocio.

- **Escalabilidad:** Los microservicios se pueden escalar de forma independiente, lo que permite a las aplicaciones soportar un mayor tráfico o carga de trabajo.

- **Mantenimiento:** Los microservicios son más fáciles de mantener que las aplicaciones monolíticas, ya que cada servicio es responsable de una función específica.

Arquitectura de microservicios también tiene algunas desventajas:

- **Complejidad:** La comunicación entre microservicios puede ser compleja, lo que puede dificultar el desarrollo y mantenimiento de la aplicación.

- **Integración:** La integración de microservicios de diferentes proveedores puede ser un desafío.

- **Seguridad**: La seguridad de los microservicios puede ser un desafío, ya que cada servicio es responsable de sus propios datos y lógica.

La arquitectura basada en microservicios es un enfoque de diseño de software en el que una aplicación se divide en pequeños componentes autónomos llamados "microservicios".

Cada microservicio tiene su propia funcionalidad y puede ser desarrollado, implementado y escalado de manera independiente. Estos microservicios se comunican entre sí a través de interfaces definidas y permiten una mayor agilidad en el desarrollo, escalabilidad y tolerancia a fallos. Aunque ofrece muchas ventajas, también puede introducir una mayor complejidad operativa debido a la gestión de múltiples componentes. En general, busca mejorar la flexibilidad y la eficiencia en el desarrollo y operación de aplicaciones.

Una API REST, o API RESTful, es una interfaz de programación de aplicaciones (API) que se ajusta a los límites de la arquitectura REST y permite la interacción con los servicios web de RESTful. El informático Roy Fielding es el creador de la transferencia de estado representacional (REST).

Una API REST se basa en el protocolo HTTP y utiliza métodos HTTP para realizar operaciones en recursos. Los recursos se identifican mediante URI y sus datos se representan en formatos estándar, como JSON o XML.

Las API REST son ampliamente utilizadas en el desarrollo web, ya que permiten que diferentes aplicaciones se comuniquen entre sí de forma sencilla y eficiente.
Algunos ejemplos de API REST incluyen:

API de Google Maps: Esta API permite a las aplicaciones obtener datos de mapas y direcciones de Google.

API de Facebook: Esta API permite a las aplicaciones interactuar con los servicios de Facebook, como la publicación de actualizaciones de estado y la conexión con amigos.

API de Twitter: Esta API permite a las aplicaciones interactuar con los servicios de Twitter, como la publicación de tweets y la búsqueda de usuarios.
Las API REST ofrecen una serie de ventajas, entre las que se incluyen:

Facilidad de uso: Las API REST son fáciles de usar y comprender, ya que se basan en protocolos estándar.

Escalabilidad: Las API REST son escalables, ya que pueden soportar grandes volúmenes de tráfico.

Flexibilidad: Las API REST son flexibles, ya que pueden adaptarse a diferentes necesidades.

Sin embargo, las API REST también tienen algunas desventajas, entre las que se incluyen:

Seguridad: Las API REST pueden ser vulnerables a ataques, por lo que es importante tomar medidas de seguridad para protegerlas.

Documentación: Las API REST requieren una buena documentación para ser utilizadas correctamente.

Ejemplos de API REST:

API de Redes Sociales: Por ejemplo, la API de Twitter permite a los desarrolladores acceder a la funcionalidad de Twitter, como publicar tweets, seguir a otros usuarios y obtener el feed de noticias de un usuario.

API de Comercio Electrónico: Un sitio web de comercio electrónico podría tener una API REST que permita a los desarrolladores acceder a información sobre productos, realizar pedidos y procesar pagos.

API de Mapas: Empresas como Google y Mapbox ofrecen API REST que permiten a los desarrolladores incorporar mapas interactivos en aplicaciones web y móviles.

API de Clima: Servicios como OpenWeatherMap ofrecen una API REST que permite a los desarrolladores obtener datos meteorológicos actuales y pronósticos para cualquier ubicación en el mundo.

API de Banca: Los bancos pueden ofrecer API REST que permitan a los desarrolladores acceder a información de cuentas, realizar transferencias y realizar otras operaciones bancarias en línea.

Los componentes básicos de Docker son:

Imagenes: Son archivos que contienen todo lo necesario para ejecutar una aplicación, incluyendo el código de la aplicación, las bibliotecas y dependencias del sistema operativo.

Contenedores: Son instancias en tiempo de ejecución de una imagen. Cada contenedor se ejecuta en su propio espacio aislado, con su propio sistema operativo, bibliotecas y archivos.

Registros: Son repositorios donde se almacenan las imágenes.

Docker Engine: Es el software que permite crear, ejecutar, gestionar y distribuir contenedores.
Arquitectura Docker

La arquitectura de Docker se basa en la siguiente estructura:

Docker Client: Es la interfaz que los usuarios utilizan para interactuar con el Docker Engine.

Docker Engine: Es el software que ejecuta los contenedores.

Docker Registry: Es el repositorio donde se almacenan las imágenes.
Imágenes.

Las imágenes son el componente central de Docker. Son archivos que contienen todo lo necesario para ejecutar una aplicación, incluyendo el código de la aplicación, las bibliotecas y dependencias del sistema operativo.

Las imágenes se crean a partir de un archivo de Dockerfile, que es un script que especifica los pasos necesarios para crear la imagen. El Dockerfile puede especificar el código de la aplicación, las bibliotecas, el sistema operativo y otras dependencias.

Contenedores

Los contenedores son instancias en tiempo de ejecución de una imagen. Cada contenedor se ejecuta en su propio espacio aislado, con su propio sistema operativo, bibliotecas y archivos.

Los contenedores son ligeros y portables, lo que los hace ideales para la implementación de aplicaciones en diferentes entornos.

Registros

Los registros son repositorios donde se almacenan las imágenes. Docker proporciona un registro público, Docker Hub, donde se pueden encontrar imágenes de aplicaciones populares.

Los usuarios también pueden crear sus propios registros privados para almacenar imágenes de aplicaciones internas.

Docker Engine

El Docker Engine es el software que permite crear, ejecutar, gestionar y distribuir contenedores.

El Docker Engine se ejecuta en un host, que puede ser un ordenador local, un servidor en la nube o un dispositivo IoT.

El Docker Engine proporciona una serie de comandos para interactuar con contenedores, incluyendo comandos para crear, iniciar, detener y eliminar contenedores.

Metodología:

1. Planificación

El primer paso es planificar la implementación de Docker. Esto incluye determinar las necesidades de la aplicación, el entorno de implementación y los recursos necesarios.

2. Instalación

El siguiente paso es instalar Docker. Esto se puede hacer en un ordenador local, un servidor en la nube o un dispositivo IoT.

3. Configuración

Una vez instalado Docker, es necesario configurarlo. Esto incluye configurar las redes, los volúmenes y otros recursos.

4. Creación de imágenes

El siguiente paso es crear imágenes de Docker. Esto se puede hacer a partir de un archivo de Dockerfile o a partir de una imagen existente.

5. Ejecución de contenedores

Una vez creadas las imágenes, se pueden ejecutar contenedores. Esto se puede hacer a partir de la línea de comandos o a través de una interfaz gráfica de usuario.

6. Gestión de contenedores

Los contenedores se pueden gestionar a través de la línea de comandos o a través de una interfaz gráfica de usuario. Esto incluye operaciones como la creación, el inicio, la parada y la eliminación de contenedores.

7. Almacenamiento de imágenes

Las imágenes se pueden almacenar en un registro público, como Docker Hub, o en un registro privado.

8. Despliegue

Una vez implementadas las aplicaciones en contenedores, se pueden desplegar en diferentes entornos. Esto se puede hacer manualmente o a través de una herramienta de automatización.

9. Monitorización

Los contenedores se pueden monitorizar para detectar problemas. Esto se puede hacer a través de herramientas de monitorización o a través de la línea de comandos.

10. Mantenimiento

Los contenedores se pueden actualizar y mantener para garantizar que funcionen correctamente. Esto se puede hacer a través de la línea de comandos o a través de una herramienta de automatización.

Conclusión:

En resumen, estas tres tecnologías y enfoques tienen un impacto significativo en el mundo del desarrollo de software y la gestión de infraestructura. La arquitectura basada en microservicios ofrece agilidad y escalabilidad, las API REST simplifican la comunicación entre aplicaciones, y Docker facilita la implementación y la gestión de contenedores. Combinadas, pueden potenciar la creación, entrega y operación eficientes de aplicaciones modernas y servicios en la nube.

Bibliografías:

Fowler, M. (2014). Microservices: Patterns, Languages, Architectures. Addison-Wesley Professional.
Newman, S. (2015). Implementing Microservices. O'Reilly Media.
Richardson, C. (2018). Microservices Patterns: Building Reliable, Scalable, and Resilient Systems (2.a ed.). Addison-Wesley Professional.

Richardson, L., & Amundsen, M. (2008). RESTful API Design and Implementation. O'Reilly Media.
O'Reilly Media. (2015). Designing RESTful APIs. O'Reilly Media.
Sonmez, J. (2017). Building RESTful APIs with Node.js. Packt Publishing.

Mouat, A. (2017). Docker in Action (2.a ed.). Manning Publications.
Mouat, A. (2016). Docker: Up & Running (3.a ed.). O'Reilly Media.
Packt Publishing. (2022). Docker: The Complete Reference. Packt Publishing.