UNIVERSIDAD AUTÓNOMA DE CHIAPAS

FACULTAD DE CONTADURÍA Y ADMINISTRACIÓN
CAMPUS 1

ING. EN DESARROLLO Y TECNOLOGÍAS DE SOFTWARE

6 “M”

TALLER DE DESARROLLO 4

SUB2

ACT. 2.5 INVESTIGAR SISTEMAS DE INFORMACIÓN CON ARQUITECTURA BASADAS EN MICROSERVICIOS

ALUMNO: MARCO ANTONIO ZÚÑIGA MORALES – A211121

DOCENTE: DR. LUIS GUTIÉRREZ ALFARO

TUXTLA GUTIÉRREZ, CHIAPAS
SÁBADO, 9 DE SEPTIEMBRE DE 2023

ÍNDICE

Resumen

La convergencia de la arquitectura basada en microservicios, las API REST y la arquitectura Docker representa una evolución fundamental en el desarrollo y la gestión de aplicaciones modernas. La arquitectura basada en microservicios descompone las aplicaciones en componentes autónomos y escalables, las API REST proporcionan una interfaz estandarizada para la comunicación entre estos componentes, y Docker permite la encapsulación y distribución eficiente de estos servicios en contenedores.

Introducción

En el mundo de la informática y el desarrollo de aplicaciones, tres conceptos clave se han destacado como pilares fundamentales: la arquitectura basada en microservicios, las API REST y la arquitectura Docker. Estos elementos, cada uno en su propio dominio, han revolucionado la forma en que se construyen, implementan y gestionan aplicaciones modernas, ofreciendo un enfoque modular, interoperable y altamente escalable que se ha convertido en la columna vertebral de muchas soluciones tecnológicas actuales.

A continuación, en esta investigación, se mostrará la esencia de cada uno de estos conceptos y, entenderemos cómo su combinación transforma el panorama tecnológico en una era caracterizada por la versatilidad y la eficiencia.

Desarrollo

Arquitectura Basada En Microservicios

Una arquitectura basada en microservicios es un enfoque de desarrollo de software que consiste en dividir una aplicación en un conjunto de servicios pequeños, autónomos y autosuficientes. Cada servicio se encarga de una única funcionalidad o responsabilidad de negocio, y se comunica con otros servicios a través de APIs.

Características

- **Los microservicios son pequeños y autónomos:**
  Cada servicio es independiente y tiene sus propias responsabilidades de negocio.

- **Los microservicios se comunican a través de APIs:**
  Los microservicios se comunican entre sí a través de APIs, que proporcionan una interfaz uniforme para la interacción entre servicios.

- **Los microservicios se pueden implementar de forma independiente:**
  Cada servicio se puede implementar de forma independiente, lo que permite a las organizaciones escalar o actualizar sus aplicaciones de forma flexible.

Ventajas

La arquitectura de microservicios ofrece una serie de ventajas sobre los enfoques tradicionales de desarrollo de software, como las arquitecturas monolíticas.

- **Flexibilidad y escalabilidad:**
  Los microservicios se pueden escalar de forma independiente, lo que permite a las organizaciones adaptar sus aplicaciones a las necesidades cambiantes del negocio.

- **Mantenimiento y evolución:**
  Los microservicios son más fáciles de mantener y evolucionar que las aplicaciones monolíticas, ya que cada servicio es independiente y puede ser actualizado o reemplazado sin afectar a otros servicios.

- **Innovación:**
  Los microservicios facilitan la innovación, ya que permiten a las organizaciones implementar nuevas funcionalidades o características de forma rápida y sencilla.

Desventajas

- **Complejidad:**
  La arquitectura de microservicios puede ser compleja de diseñar e implementar, ya que requiere un mayor nivel de coordinación entre los equipos de desarrollo.

- **Coste:**
  La arquitectura de microservicios puede ser más costosa que los enfoques tradicionales de desarrollo de software, ya que requiere una mayor infraestructura y recursos.

- **Seguridad:**
  La arquitectura de microservicios puede ser más vulnerable a los ataques, ya que cada servicio es un objetivo potencial.

Ejemplos de aplicaciones

- **Netflix:**
  La plataforma de streaming utiliza microservicios para gestionar la reproducción de vídeo, la recomendación de contenido y la gestión de usuarios.

- **Amazon:**
  El gigante del comercio electrónico utiliza microservicios para gestionar su plataforma de comercio electrónico, su servicio de almacenamiento en la nube y su servicio de inteligencia artificial.

- **Twitter:**
  La red social utiliza microservicios para gestionar la publicación de tweets, la búsqueda de contenido y la gestión de usuarios.

Definir y dar ejemplos Api Rest

Una API REST, o API RESTful, es una interfaz de programación de aplicaciones (API) que se adhiere a los principios de diseño de REST, o el estilo de arquitectura de transferencia de estado representacional. Por esta razón, las API REST a veces se denominan API RESTful.

Definición

Una API REST es una forma de conectar dos sistemas informáticos. Permite a un sistema obtener datos de otro sistema o ejecutar operaciones sobre esos datos. Las API REST se basan en el protocolo HTTP, que es el mismo protocolo que se utiliza para navegar por la web.

Principios de Rest

Las API REST se basan en los siguientes principios:

- **Recursos:**
  Los datos se representan como recursos. Cada recurso tiene un identificador único, que se utiliza para acceder a él.

- **Verbos HTTP:**
  Los verbos HTTP se utilizan para definir las operaciones que se pueden realizar sobre un recurso. Los verbos HTTP más utilizados son GET, POST, PUT y DELETE.

- **URIs:**
  Las URIs (Uniform Resource Identifiers) se utilizan para identificar los recursos.

- **Codificaciones de datos:**
  Las API REST utilizan diversos formatos de datos para representar los recursos, como XML, JSON o YAML.

Ventajas de las API REST

Las API REST ofrecen una serie de ventajas, entre las que se incluyen:

- **Facilidad de uso:**
  Las API REST son relativamente fáciles de usar y de implementar.

- **Escalabilidad:**
  Las API REST son escalables, lo que las hace adecuadas para aplicaciones que necesitan soportar un gran número de usuarios.

- **Seguridad:**
  Las API REST se pueden proteger mediante el uso de mecanismos de seguridad, como OAuth y HTTPS.

Desventajas de las API REST

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

- **Rendimiento:**
  Las API REST pueden ser menos eficientes que otros tipos de API, como las API SOAP.

- **Compatibilidad:**
  Las API REST no son compatibles con todos los lenguajes de programación y frameworks.

Ejemplos de API REST

- Algunos ejemplos de API REST son:
  API de Google Maps:Esta API permite a los desarrolladores obtener información sobre mapas y direcciones.

- **API de Facebook:**
  Esta API permite a los desarrolladores interactuar con los datos de los usuarios de Facebook.

- **API de Twitter:**
  Esta API permite a los desarrolladores acceder a los datos de Twitter, como los tweets y los usuarios.

Dockers

Docker es una plataforma de código abierto que permite a los desarrolladores crear, implementar, ejecutar, actualizar y gestionar contenedores. Un contenedor es un componente estandarizado y ejecutable que combina el código fuente de la aplicación con las bibliotecas y dependencias del sistema operativo (SO) necesarias para su ejecución.

Componentes básicos de Docker

Los componentes básicos de Docker son:

- **Imágenes:**
  Las imágenes son plantillas que contienen todo lo necesario para ejecutar una aplicación, incluido el código, las bibliotecas, los archivos de configuración y los datos.

- **Contenedores:**
  Los contenedores son instancias de imágenes que se ejecutan en un entorno aislado.

- **Registros:**
  Los registros son almacenes de imágenes que pueden ser públicos o privados.

- **Docker Engine:**
  También conocido como Docker Daemon, es el núcleo de Docker. Es un servicio que se ejecuta en segundo plano en el sistema operativo anfitrión y es responsable de la creación y gestión de contenedores. El Docker Engine incluye tres componentes clave:

    - **Dockerd:**
      El proceso principal que escucha las solicitudes de la API de Docker y administra los contenedores.

- **Containerd:**
  Un componente subyacente que administra el ciclo de vida de los contenedores, incluida la ejecución, el almacenamiento y la eliminación de imágenes y contenedores.

- **Runc:**
  Una interfaz que permite ejecutar contenedores de acuerdo con las especificaciones OCI (Open Container Initiative).

- **Imágenes de Docker**:
  Las imágenes son plantillas de solo lectura que contienen el sistema operativo, las bibliotecas y la aplicación necesarios para ejecutar un contenedor. Las imágenes son la base para crear contenedores y se almacenan en el registro de Docker, como Docker Hub.

- **Contenedores:**
  Son instancias en tiempo de ejecución de una imagen de Docker. Los contenedores son entornos aislados que encapsulan aplicaciones y sus dependencias, lo que facilita la ejecución de aplicaciones de manera consistente en diferentes entornos.

- **Dockerfile:**
  Es un archivo de texto que contiene instrucciones para construir una imagen de Docker. Define qué software se incluirá en la imagen y cómo se configurará.

- **Registro de Docker:**
  Es un repositorio de imágenes de Docker. Docker Hub es un registro público y popular, pero también puedes configurar registros privados para almacenar imágenes personalizadas.

Arquitectura Docker

La arquitectura de Docker sigue el modelo cliente-servidor. A continuación, se describen los componentes clave de la arquitectura de Docker:

1. **Cliente Docker**:
   Es la interfaz de línea de comandos o la API que permite a los usuarios interactuar con Docker. Los usuarios envían comandos al cliente Docker para crear, administrar y supervisar contenedores e imágenes.

2. **Docker Host:**
   Es el sistema en el que se ejecuta Docker Engine. Puede ser un servidor físico o una máquina virtual.

3. **Docker Daemon:**
   También conocido como Docker Engine, se ejecuta en el Docker Host y es responsable de administrar los contenedores y las imágenes. Escucha las solicitudes del cliente Docker y realiza las operaciones necesarias en el sistema operativo subyacente.

4. **API de Docker:**
   Permite que el cliente Docker y el Docker Daemon se comuniquen entre sí. El cliente envía solicitudes a través de la API, y el Daemon las procesa y toma medidas en consecuencia.

5. **Registro de Docker:**
   Es un repositorio de imágenes de Docker que se utiliza para almacenar y compartir imágenes. Los registros pueden ser públicos (como Docker Hub) o privados (configurados por el usuario).

6. **Red de Docker:**
   Docker proporciona capacidades de red que permiten la comunicación entre contenedores y con redes externas. Puedes crear redes personalizadas y conectar contenedores a ellas.

7. **Docker CLI (Interfaz de Línea de Comandos):**
   Es como tu teléfono para hablar con Docker. Puedes darle órdenes en texto desde tu computadora para crear, iniciar y manejar los contenedores. Es como enviar mensajes de texto a Docker para que haga cosas

8. **Volúmenes de Docker:**
   Son como carpetas secretas que solo algunas cajas mágicas pueden ver y usar. Puedes poner datos allí y las cajas mágicas pueden compartir estos datos sin que los demás lo sepan.

9. **Docker Compose:**
   Es como un director de orquesta que sabe cómo hacer que muchas cajas mágicas trabajen juntas en una gran actuación. Le dices qué cajas mágicas quieres que trabajen juntas y cómo, y él se encarga del resto. Es como un plan maestro para las cajas mágicas

Ejemplos de uso de Docker

Docker se puede utilizar para ejecutar una amplia gama de aplicaciones, incluidas:

- **Aplicaciones web:**
  Docker se puede utilizar para ejecutar aplicaciones web sin tener que instalarlas en el sistema operativo subyacente.

- **Bases de datos:**
  Docker se puede utilizar para ejecutar bases de datos sin tener que instalarlas en el sistema operativo subyacente.
- **Servicios:**
  Docker se puede utilizar para ejecutar servicios sin tener que instalarlos en el sistema operativo subyacente.

Ventajas de Docker

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

- **Portabilidad:**
  Los contenedores se pueden ejecutar en cualquier sistema operativo que admita Docker.

- **Eficiencia:**
  Los contenedores comparten el kernel del sistema operativo subyacente, lo que los hace más eficientes que las máquinas virtuales.

- **Seguridad:**
  Los contenedores están aislados entre sí, lo que ayuda a proteger las aplicaciones.

Desventajas de Docker

Docker también tiene algunas desventajas, entre las que se incluyen:

- **Complejidad:**
  Docker puede ser complejo de aprender y usar.

- **Escalabilidad:**
  La escalabilidad de Docker puede ser un desafío en entornos grandes.

Conclusión

En conclusión, de acuerdo con la información investigada en libros y sitios web vemos que La combinación de arquitectura basada en microservicios, API REST y la tecnología Docker ha revolucionado la forma en que se desarrollan, implementan y gestionan aplicaciones en la era moderna de la informática. Estos tres elementos trabajan en conjunto para ofrecer flexibilidad, escalabilidad y eficiencia en el mundo de la informática contemporánea.

La arquitectura basada en microservicios descompone las aplicaciones en componentes más pequeños y autónomos, lo que facilita el desarrollo y la escalabilidad. Cada microservicio se puede desarrollar, implementar y actualizar de manera independiente, lo que acelera el ciclo de desarrollo y permite una mayor agilidad.

Las API REST (Representational State Transfer) proporcionan un medio estándar y flexible para que los microservicios se comuniquen entre sí y con otras aplicaciones. Estas interfaces se basan en principios simples, como el uso de métodos HTTP estándar, lo que facilita la integración y la interoperabilidad de los microservicios.

Docker, por su parte, es una tecnología que permite encapsular aplicaciones y sus dependencias en contenedores. Estos contenedores son portátiles y se ejecutan de manera consistente en cualquier entorno que admita Docker, lo que simplifica la implementación y el escalado de microservicios. Docker también proporciona una forma eficiente de administrar y orquestar múltiples contenedores, lo que es esencial en arquitecturas de microservicios distribuidas.

Esta combinación de tecnologías es fundamental para la creación de aplicaciones modernas que pueden adaptarse rápidamente a las demandas cambiantes del mercado y escalar de manera eficiente para satisfacer las necesidades del negocio.

Referencia Bibliográfica

Docker, Inc. (2023). Docker documentation. Recuperado de https://docs.docker.com/

Docker Hub. (2023, 9 de agosto). Docker. https://hub.docker.com/

Fowler, M. (2014). Microservices: Building Scalable, Reliable Systems. MartinFowler.com. Retrieved from https://martinfowler.com/articles/microservices.html

Lewis, J., & Fowler, M. (2014). Microservices: A Definition of This New Architectural Style. MartinFowler.com. Retrieved from https://martinfowler.com/articles/microservices.html

Heroku. (s.f.). The Twelve-Factor App. Heroku.com. Retrieved from https://12factor.net/

Fowler, M., Parsons, R., & Evans, E. (2014). Microservices Architecture: Design, Building, and Deploying Scalable Applications. Addison-Wesley Professional.

Richards, M., & Kleppmann, M. (2019). Designing Microservices. O'Reilly Media.

Richardson, C. (2018). Microservices Patterns: Building Scalable, Reliable, and Extensible Systems. Addison-Wesley Professional.

Microservices.io. (n.d.). Microservices.io. Retrieved from https://microservices.io/

MartinFowler.com. (n.d.). MartinFowler.com. Retrieved from https://martinfowler.com/

Heroku.com. (n.d.). Heroku.com. Retrieved from https://heroku.com/

Fielding, R. T. (2000). Architectural styles and the design of network-based software architectures. University of California, Irvine.

IBM. (s. f.). REST APIs. IBM Cloud.

HubSpot. (s. f.). Qué es una API REST, para qué sirve y ejemplos. Blog de HubSpot.

Red Hat. (s. f.). REST API. Red Hat.

TechTarget. (s. f.). What is REST API (RESTful API)? TechTarget.

Atlassian. (s. f.). Arquitectura de microservicios | Atlassian. https://www.atlassian.com/es/microservices/microservices-architecture#:~:text=Una%20arquitectura%20de%20microservicios%20divide%20una%20aplicaci%C3%B3n%20en,r%C3%A1pida%20y%20frecuente%20de%20aplicaciones%20grandes%20y%20complejas.

Kinsta. (2023, 13 enero). Qué es Docker: una guía completa. Kinsta®. https://kinsta.com/es/base-de-conocimiento/que-es-docker/

¿Qué es Docker? | IBM. (s. f.). https://www.ibm.com/mx-es/topics/docker