



Dockerizando microservicios

UNJu - Desarrollo y Arquitecturas Avanzadas de Software

Ing. José Zapana



Introducción a Microservicios

- Los microservicios son un enfoque de diseño de software donde una aplicación se compone de pequeños servicios independientes que se comunican entre sí a través de APIs.
- Características Clave:
 - Desacoplamiento: Cada servicio es autónomo y maneja una funcionalidad específica del sistema.
 - Escalabilidad: Los servicios pueden escalarse de manera independiente según la demanda.
 - Implementación Independiente: Permite a los equipos desarrollar, desplegar y mantener cada servicio de forma aislada.
- Beneficios: Mejora la flexibilidad, la eficiencia en el desarrollo y facilita la adopción de tecnologías diversas.



Desventajas de los Microservicios

- Complejidad Operativa:
 - La gestión de múltiples servicios aumenta la complejidad en la implementación, monitoreo y mantenimiento.
 - Se requiere una infraestructura más sofisticada para manejar la orquestación y el despliegue continuo.
- Dificultad en la Comunicación:
 - La comunicación entre servicios se realiza a través de la red, lo que puede introducir latencia y aumentar el riesgo de fallos de red.
 - Es necesario diseñar un protocolo robusto para manejar la comunicación y posibles errores entre los servicios.
- Costos de Desarrollo y Mantenimiento:
 - Requiere equipos con habilidades especializadas en tecnologías y herramientas de orquestación, como Kubernetes y Docker.
 - La implementación de microservicios puede implicar costos adicionales para gestionar y escalar cada servicio de manera independiente.
- Pruebas y Depuración:
 - La prueba de una aplicación distribuida es más complicada en comparación con una aplicación monolítica.
 - Los errores pueden ser difíciles de rastrear debido a la naturaleza distribuida del sistema.



¿Cuándo es recomendable implementar Microservicios?

- Aplicaciones a Gran Escala:
 - Cuando el proyecto es lo suficientemente grande y complejo como para justificar la modularización.
 - Es ideal para aplicaciones que requieren alta disponibilidad y escalabilidad.
- Equipos de Desarrollo Grandes:
 - Cuando tienes equipos de desarrollo grandes que pueden trabajar de forma independiente en diferentes partes de la aplicación.
 - Permite a los equipos trabajar en paralelo sin interferencias, mejorando la productividad.
- Evolución Rápida del Producto:
 - Si tu aplicación necesita lanzar nuevas funcionalidades de manera rápida y frecuente.
 - Los microservicios permiten una implementación y un ciclo de desarrollo más rápidos.
- Necesidad de Tecnologías Diversas:
 - Cuando tu aplicación requiere el uso de diferentes lenguajes de programación o tecnologías para optimizar cada componente.
 - Los microservicios permiten utilizar la tecnología más adecuada para cada parte del sistema.



Tecnologías y herramientas

- Contenedores y Orquestación: docker y kubernetes
- Comunicación entre Servicios
 - RESTful APIs: Aprender a diseñar e implementar APIs RESTful para la comunicación entre microservicios.
 - Mensajería asíncrona: RabbitMQ o Apache Kafka para la comunicación basada en eventos entre servicios.
- Gestión de Configuración y Monitoreo
- Infraestructura y DevOps
 - CI/CD (Integración y Despliegue Continuo):
 - Infraestructura como código (IaC): Tecnologías como Terraform o Ansible
- Frameworks y Lenguajes de Programación
 - Spring Boot (Java): Ideal para desarrollar microservicios robustos y escalables.
 - Node.js: Muy utilizado para construir servicios rápidos y eficientes.
 - Python frameworks como Flask o FastAPI para microservicios ligeros.
- **6. Seguridad**
 - OAuth2.0 y JWT (JSON Web Tokens): Para la autenticación y autorización de servicios.
 - API Gateway: Herramientas como Spring Cloud Gateway para gestionar la seguridad, la autenticación y el enruteamiento de las APIs.
- **7. Bases de Datos**
 - Bases de datos distribuidas: bases de datos que soporten particionamiento y replicación, como Cassandra o MongoDB.
 - Bases de datos SQL y NoSQL: PostgreSQL o MySQL frente a NoSQL como Redis o DynamoDB.



Criterios para partir en microservicios

- Bounded Contexts (DDD) – lenguaje y reglas propias: Cada microservicio debe encapsular un contexto.
 - Ej.: “Reservas” (reglas de estado, confirmación/cancelación, compensaciones) es un contexto distinto a “Catálogo de vuelos”.
- Cohesión funcional y autonomía de cambio: Agrupa lo que cambia junto y separa lo que suele evolucionar con ritmos diferentes.
 - Ej.: Precios/Tarifas puede cambiar con frecuencia distinta a Reservas.
- Propiedad de datos (single source of truth): Cada servicio es dueño de sus datos y expone contratos
 - Ej.: “Clientes” es dueño del perfil; “Reservas” referencia clienteld.
- Escalabilidad y carga: Separar lo que tiene patrones de carga muy distintos o necesita escalar de otro modo
 - Ej.: Búsquedas intensivas o pricing podrían escalar distinto a Reservas.



Criterios para partir en microservicios II

- Integraciones externas / compliance: (pagos, GDS) o políticas específicas (seguridad, auditoría)
- Equipo y entrega independiente: Servicios que pueda mantener un equipo de forma autónoma, con su ciclo de vida
- Tamaño y simplicidad operativa: Definir servicios con valor de negocio y responsabilidades claras.

Regla informal: si un servicio no puede explicar su capability en una frase clara, está mal definido.



Antipatrones

- Entidad = microservicio. (Casi siempre mal.)
- Compartir BD entre servicios.
- Dividir antes de tener reglas claras. (Primero entender el dominio; después partir.)



Dockerfile

```
#inicia con la imagen base que contiene Java runtime
FROM openjdk:21-jdk-slim as build

# se agregar el jar del microservicio al contenedor
COPY target/ms-0.0.1-SNAPSHOT.jar ms-0.0.1-SNAPSHOT.jar

#se ejecuta el microservicio
ENTRYPOINT ["java","-jar","/ms-0.0.1-SNAPSHOT.jar"]
```

ms: nombre del microservicio



Construir una imagen docker

Construir la imagen a partir del proyecto

docker build . -t usuario/ms

Comandos útiles de docker

- ver las imágenes
 - **docker images**
- corremos la imagen para crear el contenedor
 - **docker run -p 8080:8080 usuario/ms**
- ver los contenedores que están corriendo
 - **docker ps**
- inicia dentro de otro contenedor
 - **docker run -p 8090:8080 usuario/ms**



Subir la imagen a docker hub

- Crear cuenta en <https://hub.docker.com/>
- Desde la consola
 - login -u "usuario" -p xxxxx
 - docker push docker.io/usuario/ms