Trabajo Práctico 4 - Arquitectura de Microservicios

1- Objetivos de Aprendizaje

- Familiarizarse con conceptos de Microservicios

2- Unidad temática que incluye este trabajo práctico

Este trabajo práctico corresponde a la unidad Nº: 2 (Libro Ingeniería de Software: Unidad 18)

3- Consignas a desarrollar en el trabajo práctico:

A continuación, se presentarán algunos conceptos generales de la arquitectura de microservicios y exploraremos un ejemplo completo y funcional de ejemplo de este tipo de sistemas.

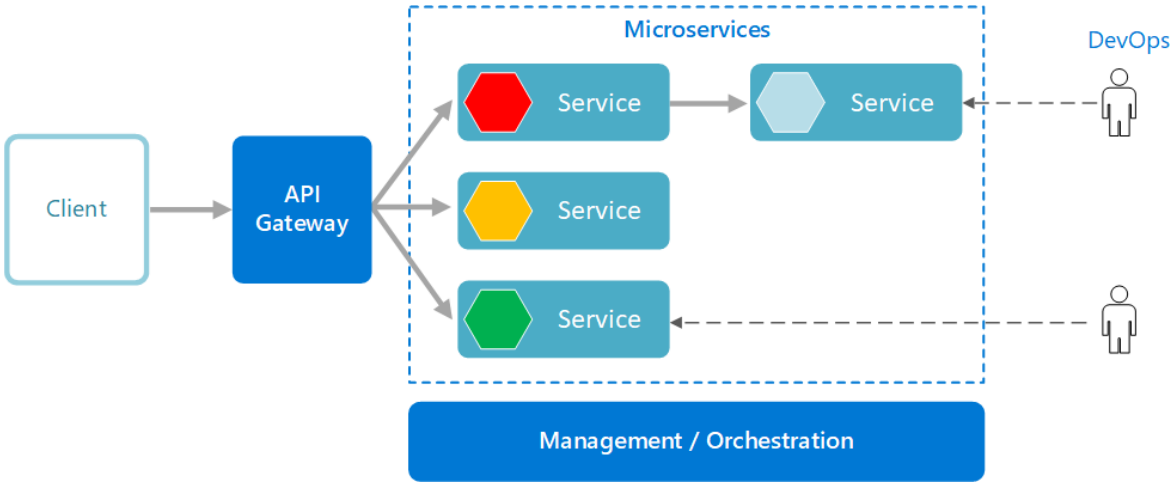
Arquitectura

Una arquitectura de microservicios consta de una colección de servicios autónomos y pequeños. Los servicios son independientes entre sí y cada uno debe implementar una funcionalidad de negocio individual.

¿Qué son los microservicios?

- Los microservicios son pequeños e independientes, y están acoplados de forma imprecisa. Un único equipo reducido de programadores puede escribir y mantener un servicio.
- Cada servicio es un código base independiente, que puede administrarse por un equipo de desarrollo pequeño.
- Los servicios pueden implementarse de manera independiente. Un equipo puede actualizar un servicio existente sin tener que volver a generar e implementar toda la aplicación.
- Los servicios son los responsables de conservar sus propios datos o estado externo. Esto difiere del modelo tradicional, donde una capa de datos independiente controla la persistencia de los datos.
- Los servicios se comunican entre sí mediante API bien definidas. Los detalles de la implementación interna de cada servicio se ocultan frente a otros servicios.

- No es necesario que los servicios compartan la misma pila de tecnología, las bibliotecas o los marcos de trabajo.



(Contenido e Imagen: https://docs.microsoft.com/es-es/azure/architecture/microservices/)

Administración e implementación.

Este componente es el responsable de la colocación de servicios en los nodos, la identificación de errores, el reequilibrio de servicios entre nodos, etc. Normalmente, este componente es una tecnología estándar, como Kubernetes, en lugar de algo creado de forma personalizada.

Puerta de enlace de API

La puerta de enlace de API es el punto de entrada para los clientes. En lugar de llamar a los servicios directamente, los clientes llaman a la puerta de enlace de API, que reenvía la llamada a los servicios apropiados en el back-end.

Entre las ventajas de usar una puerta de enlace de API se encuentran las siguientes:

- Desacoplan los clientes de los servicios. Los servicios pueden cambiar de versión o refactorizarse sin necesidad de actualizar todos los clientes.
- Los servicios pueden utilizar los protocolos de mensajería que no son fáciles de usar para un servicio web, como AMQP.
- La puerta de enlace de API puede realizar otras funciones transversales como la autenticación, el registro, la terminación SSL y el equilibrio de carga.

4- Desarrollo:

1- Instanciación del sistema

- Clonar el repositorio https://github.com/microservices-demo/microservices-demo

```
mkdir -p socks-demo
cd socks-demo
git clone https://github.com/microservices-demo/microservices-demo.git
```

- Ejecutar lo siguiente

```
cd microservices-demo
docker-compose -f deploy/docker-compose/docker-compose.yml up -d
```

- Una vez terminado el comando `docker-compose` acceder a http://localhost
- Generar un usuario
- Realizar búsquedas por tipo de media, color, etc.
- Hacer una compra - poner datos falsos de tarjeta de crédito ;)

2- Investigación de los componentes

1. Describa los contenedores creados, indicando cuales son los puntos de ingreso del sistema
2. Clonar algunos de los repositorios con el código de las aplicaciones

```
cd socks-demo
git clone https://github.com/microservices-demo/front-end.git
git clone https://github.com/microservices-demo/user.git
git clone https://github.com/microservices-demo/edge-router.git
.
.
```

3. ¿Por qué cree usted que se está utilizando repositorios separados para el código y/o la configuración del sistema? Explique puntos a favor y en contra.
4. ¿Cuál contenedor hace las veces de API Gateway?
5. Cuando ejecuto este comando:

```
curl http://localhost/customers
```

6. ¿Cuál de todos los servicios está procesando la operación?

7. ¿Y para los siguientes casos?

```
curl http://localhost/catalogue
curl http://localhost/tags
```

8. ¿Como perisisten los datos los servicios?
9. ¿Cuál es el componente encargado del procesamiento de la cola de mensajes?
10. ¿Qué tipo de interfaz utilizan estos microservicios para comunicarse?