Informe del Sistema Distribuido

Diseño e Implementación de una Red Social Distribuida

Victor Hugo Pacheco Fonseca C411
Jose Agustin del Toro C412
Asignatura: Sistemas Distribuidos

2025

Índice

1. Introducción

El presente informe describe el diseño e implementación de una red social construida bajo una arquitectura de microservicios distribuida. El objetivo principal es aplicar los conceptos fundamentales de los sistemas distribuidos, tales como particionamiento de datos, comunicación entre procesos, consistencia, replicación y demas

La aplicación se divide en múltiples servicios independientes que cooperan mediante comunicación interna HTTP, mientras que el cliente interactúa únicamente con un API Gateway que unifica y abstrae la infraestructura. Cada servicio gestiona su propia base de datos o fragmento (*shard*), lo que permite distribuir la carga, mejorar la disponibilidad

2. Arquitectura General del Sistema

La arquictectura es microservicios

2.1. Microservicios

El sistema se divide en los siguientes servicios independientes:

- **API Gateway**: punto de entrada único para el cliente.
- **Auth Service**: autenticación y firma/verificación de tokens JWT.
- **User Service**: administración de perfiles de usuarios.
- **Relation Service**: mantenimiento del grafo de seguidores.
- **Post Service**: almacenamiento distribuido de publicaciones.
- **Feed Service**: generación del feed mediante almacenamiento temporal distribuido.

3. Comunicación

3.1. Comunicación Cliente → Servidor

El cliente interactúa únicamente con el **API Gateway**, que actúa como intermediario y responsable de:

- Redirigir solicitudes a los microservicios correctos.
- Validar tokens JWT.
- Simplificar la visibilidad de la arquitectura interna.

Las comunicaciones se realizan mediante peticiones HTTP utilizando JSON como formato de transporte. El cliente nunca accede directamente a los microservicios.

3.2. Comunicación entre Servidores

La comunicación entre microservicios se realiza mediante **HTTP interno** dentro de la red del despliegue. Cada servicio expone endpoints internos que permiten:

- Consultar información alojada en otro servicio (p. ej., obtener seguidores).
- Notificar eventos relevantes, como una nueva publicación.
- Propagar cambios para mantener consistencia eventual.

Este enfoque mantiene independencia entre los servicios, respetando el estilo distribuido, sin necesidad de colas de mensajes o brokers.

4. Organización Interna del Sistema Distribuido

4.1. Procesos

- **Procesos sin estado**: API Gateway y Auth Service.
- **Procesos con estado**: User, Relation, Post y Feed Services.
- **Instancias replicadas**: cada servicio se puede duplicar para tolerancia a fallos.

4.2. Agrupación de Procesos

Cada microservicio se ejecuta en un contenedor autónomo, permitiendo:

- despliegue individual;
- fallos parciales sin impacto global;
- escalado independiente.

5. Sharding y Distribución de Datos

5.1. Modelo de Sharding

El sistema distribuye los datos en múltiples fragmentos mediante una función de dispersión aplicada al identificador del usuario. Esto permite distribuir:

- perfiles de usuarios;
- relaciones de seguidores;
- publicaciones asociadas al autor.

Cada shard opera como un nodo autónomo, con sus propias réplicas si se requieren.

5.2. Ventajas del modelo

- Distribución uniforme de la carga.
- Reducción de contención y colisiones.
- Escalado horizontal efectivo.

6. Consistencia y Replicación

6.1. Consistencia

El sistema implementa una combinación de:

- **Consistencia eventual global**: los microservicios intercambian información por HTTP, por lo que el estado del sistema converge con el tiempo.

6.2. Replicación

Cada shard puede replicarse en múltiples nodos para:

- mejorar disponibilidad;
- prevenir pérdida de datos;

La replicación sigue un modelo **primario–réplica**, donde el nodo principal acepta escrituras y las réplicas absorben lecturas.

7. Nombrado y Localización

El sistema utiliza un esquema de nombrado basado en **nombres lógicos** para identificar tanto a los microservicios como a los shards de base de datos. Estos nombres no dependen de direcciones físicas, lo que proporciona transparencia y flexibilidad en un entorno distribuido.

7.1. Nombrado

Cada microservicio se identifica mediante un nombre lógico, como `auth-service`, `user-service` o `post-service`. De igual forma, cada shard de la base de datos posee un nombre como `user_db_shard_0` o `post_shard_1`. El uso de estos nombres permite que la aplicación interactúe con los servicios sin conocer su ubicación real, cumpliendo así con el principio de separar identidad y localización.

7.2. Localización

La localización se resuelve mediante dos mecanismos complementarios:

- **DNS interno**: el entorno de ejecución traduce automáticamente los nombres lógicos de los microservicios a sus direcciones reales. Esto permite mover, replicar o reiniciar servicios sin modificar el código.
- **API Gateway**: actúa como punto de acceso único para el cliente y como sistema de localización indirecta. El cliente solo conoce al Gateway, y este reenvía cada solicitud al microservicio correspondiente según su nombre.

Este enfoque permite mantener independencia entre servicios, facilita el escalado y garantiza que los componentes puedan reorganizarse sin afectar al sistema global.

8. Tolerancia a Fallos

- El sistema permite que un servicio falle sin provocar la caída global.
- La replicación reduce la probabilidad de pérdida de información.
- La comunicación entre servicios se reintenta automáticamente si un nodo está temporalmente inaccesible.

9. Conclusiones

El sistema diseñado cumple con los principios fundamentales del Teorema CAP, que establece que un sistema distribuido no puede garantizar simultáneamente consistencia, disponibilidad y tolerancia a particiones.

En este caso:

- Se prioriza la **Disponibilidad** mediante microservicios independientes y replicados.
- Se prioriza la **Tolerancia a Particiones** usando comunicación interna por HTTP, que permite que los servicios continúen operando incluso ante fallos parciales.
- Se alcanza **Consistencia Eventual**, dado que las actualizaciones se propagan entre microservicios con un retraso inherente.

Por tanto, el proyecto implementa un sistema distribuido clasificado como **AP** según CAP: prioriza disponibilidad y tolerancia a fallos, sacrificando consistencia inmediata, lo cual es apropiado y común en sistemas sociales modernos.

El resultado es una arquitectura robusta, extensible y coherente con los objetivos de la asignatura de Sistemas Distribuidos.