Informe

Roger Fuentes Rodríguez
Kevin Manzano Rodríguez

Índice

1. Arquitectura

1.1. Organización del sistema distribuido

Implementaremos una arquitectura de microservicio, con los principales servicios: Autenticación (gestión de usuarios, autenticación e identificación), Grupos (gestión de los grupos de los usuarios, incluyendo roles y jerarquias), Agendas (gestión de eventos y reuniones), Notificaciones (envio de notificaciones en tiempo real a los usuarios), Base de datos Distribuidas (almacena la información).

1.2. Distribución de servicios en ambas redes Docker

Tendremos dos redes de Docker separadas, Cliente (frontend) y Servidor (Backend y Base de Datos), el Servidor alberga los servicios anteriormente mencionados.

2. Procesos

2.1. Tipos

1. Autenticación.
    a) Gestión de usuarios.
    b) Autenticación y Autorización.
2. Grupos.
    a) Gestión de grupos.
3. Agenda.
    a) Gestión de agenda y reuniones.
4. Notificaciones.
    a) envio de notificaciones en tiempo real.
5. Base de datos.
    a) Gestión de datos.

2.2. Organización en una instancia

Los procesos estarán en instancias separadas para cada microservicio, cada servicio tiene su propia instancia de contenedor Docker.

2.3. Tipo de patrón de diseño

El sistema utilizará Async, para los llamados HTTP, y para optimizar el envio de notificaciones Hilos para enviar las notificaciones de forma paralela.

3. Comunicación

Tendremos comunicación cliente-servidor a través de APIs RESTful, entre servidores utilizaremos ZMQ y entre procesos RPC.

4. Coordinación

Para la coordinación entre servidores utilizaremos PUB-SUB que nos brinda ZMQ y para el acceso exclusivo a recursos emplearemos locks. Con respecto a la toma de decisiones implementaremos un algoritmo de consenso de quorum.

5. Nombrado y Localización

Utilizaremos CHORD como vimos en clases, tendremos un anillo de tamaño 2^m y asignaremos los recursos a una clave utilizando la función de hash SHA-256, y en el momento de acceder a un recurso utilizaremos las fingers-tables almacenadas en los nodos.

6. Consistencia y Replicación

Dado que utilizaremos CHORD en la sección anterior, para garantizar nivel 2 de tolerancia a fallos implementaremos replicación múltiple (K-Réplicas) para distribuir las copias de los datos entre varios nodos (ej consecutivos) y una consistencia eventual combinado con reparación automática de réplicas.

7. Tolerancia a fallas

Utilizaremos mecanismos de health checks a los nodos vecinos de manera periódica para verificar si están activo. En caso de que un nodo falle se asegura que las claves asignadas a ese nodo sean rápidamente reasignadas (mecanismos de replicación). Para la recuperación de datos, las replicas almacenadas en otros nodos sincronizan su información una vez que el nodo se recupere.

8. Seguridad

En el transito tendremos TLS, para cifrar la comunicación entre los nodos del CHORD. Los nodos necesitan ser capaces de verificar la autenticidad de otros nodos para eso implementaremos una Autenticación basada en certificados. Para la Autenticación de los usuarios usamos JWT