Tarea 2 Sistemas distribuidos

Rodrigo Álvarez 201573587-5 - Manuel Sandoval 201573604-9

Enero 2020

1 Resumen

En el presente documento se señalarán las presentes diferencias técnicas entre dos implementaciones para una arquitectura, donde se creó un chat que consiste de n-clientes y un servidor central de coordinación de mensajes, mediante el uso del lenguaje Python y los frameworks gRPC[1] para implementar el sistema de mensajería usando el protocolo RPC(Remote Procedure Call) y RabbitMQ [2] como bróker de mensajería o gestor de colas, implementando así mensajes asíncronos. Finalmente se señalará una recomendación de estas herramientas para desarrollar la arquitectura y servicios solicitados.

2 Implementación con gRPC

2.1 Servidor

Para el servidor fue necesario definir previamente un archivo .proto el cual define los métodos que serán utilizados e implementados. Los métodos utilizados fueron: **RecibirMensaje, EnviarMensaje, DevolverUsuarios, ValidarUsuario, DevolverMensajes y Desconectarse**, además de los tipos de mensaje: **Usuario, Mensaje y Ack**. Estos representan los elementos que se usarán para desarrollar la arquitectura.

2.2 Cliente

El cliente realiza llamados y ejecuciones de los métodos señalados anteriormente según los necesite.
El cliente se conecta a la dirección del servidor, luego solicita un id único, si el id ingresado ya existe, se solicita uno nuevo hasta que el id ingresado se encuentre disponible.
A continuación, el cliente obtiene un menú con las opciones: **Chatear, Obtener clientes activos, Obtener mensajes enviados y 0 para salir.**. Las opciones **Obtener clientes y Obtener mensajes** ejecutan los servicios imprimen lo solicitado y vuelven al menú. **Al presionar 0** el usuario se desconecta del servidor dejando libre el id para ingresar posteriormente, si hay un error en la ejecución o se esta se cancela, el id queda tomado hasta que se reinicie el servidor. Finalmente **Chatear** verifica si hay mensajes nuevos para el cliente, y espera un input para enviar un mensaje, el cual debe ser de la forma **destinatario@mensaje**, cualquier formato distinto finalizará la ejecución.

3 Implementación con RabbitMQ

3.1 Servidor

El servidor es el encargado de manejar los mensajes y solicitudes realizadas por el cliente mediante 3 colas. La cola Login es la que se encarga de procesar las solicitudes de logeo al servidor por parte de

los clientes, la cola Chat que maneja el envío de mensajes entre usuarios y la cola View que se encarga de los otros servicios que solicitan los clientes, que son la lista de clientes conectados y el historial de mensajes enviados. Cada cola es manejada por una hebra, para que así puedan mantenerse los servicios ofrecidos en paralelo y no sean interrumpidos. El servidor almacena el id único del usuario para usarlo posteriormente como routing_key para la comunicación mediante el exchange que corresponda.

3.2 Cliente

Al conectarse al servidor el cliente debe introducir su nombre de usuario lo que ayudará para ser identificado facilmente por otros usuarios a la hora de enviar mensajes, además se le asigna un id único con el fin de poder usar dicho identificador como routing_key y asi enrutar los exchanges de las colas. Una vez que finaliza el proceso de login, el usuario puede utilizar los servicios ofrecidos, además de poder recibir mensajes por parte de los demás usuarios. Se utilizan dos hebras, las cuales se conectan al Broker de manera individual, una de ellas se encarga del login y posteriormente se utiliza para el manejo de los servicios del usuario, mientras que la otra se encarga solo de manejar los mensajes que recibe el usuario.

4 Comparación

Al utilizar gRPC, la definición de los métodos y mensajes que se usarán hace que el programador tenga mucho control sobre lo que se está comunicando y de la misma forma debe tener muy claro lo que se hará, dado que estas definiciones serán las utilizadas posteriormente.
Por otro lado, en RabbitMQ no se requiere una conexión directa entre cliente y servidor, lo que es conveniente para un chat, pues no es necesario para el cliente ser consciente de que hay un servidor contestando sus solicitudes, pudiendo además dividir el servidor en varios workers y que no todos los clientes envíen sus solicitudes a uno solo, permitiendo así distribuir la carga del servidor.

Ambas tecnologías permiten la comunicación entre programas en distintos lenguajes, no obstante, gRPC no ofrece la posibilidad de abstraerse de los tipos de datos de cada lenguaje, en cambio para RabbitMQ, es posible utilizar alguna notación como por ejemplo JSON para y así comunicar 2 programas.

Por lo anterior la recomendación sería hacer uso de RabbitMQ, no obstante, todas estas recomendaciones van de la mano de la arquitectura realizada. Es posible que alguna aplicación requiera siempre estar conectado al servidor y en ese caso gRPC sería más conveniente que RabbitMQ.

References

[1] *gRPC*

[2] *RabbitMQ*