Método de acceso al servidor
Para acceder al backend con ssh:

sh -i "allsshinstance.pem" root@ec2-13-58-214-7.us-east-2.compute.amazonaws.com

Para acceder a la BD con ssh:

ssh -i "allsshinstance.pem"
ubuntu@ec2-18-223-238-42.us-east-2.compute.amazonaws.com

Para acceder a la instancia de redis:

ssh -i "redis-test.pem" ubuntu@ec2-34-224-111-177.compute-1.amazonaws.com

Consideraciones
La documentación de todas las tecnologías utilizadas se encuentra al final del archivo

Requisitos

Parte mínima

Backend

RF1: logrado
El chat básico funciona, se envían mensajes y se registra el timestamp

RF2: logrado
El backend funciona como API

RF3: parcial
Se implementó el AutoScalingGroup con un LoadBalancer, pero no se añadió el header correspondiente

RF4: logrado
El servidor se encuentra en open-chat-api.tk

RF4 (https): logrado
Se usa SSL y se redirige de http a https

Frontend

RF5: logrado
Se implementó Cloudfront y S3

RF6: logrado
Se implementó una aplicación web mediante ReactJS, que realiza las llamadas a la api. Esta aplicación se encuentra en open-chat.ml

Sección variable

Caché

RF1: logrado
Se implementó una instancia AWS EC2 con redis-server, a la que se conecta el backend.

RF2: logrado
Se aplicó con el listado de chatrooms y los últimos 10 mensajes por chatroom. Más detalles en la documentación

RF3: logrado
La documentación se encuentra al final de este archivo.

Mensajes en tiempo real

RF1: logrado
Se implementó con websockets

RF2: logrado
Se envía un mail mediante AWS SES.

RF3: logrado
La documentación se encuentra al final de este archivo.

Documentación

Frontend
Se implementó a través de AWS Cloudfront y AWS S3. Toda la página se encuentra en un bucket y es servida a través de un CDN en su totalidad

CDN
El CDN se implementó con AWS Cloudfront y AWS S3, y actualmente todo el frontend se encuentra hosteado ahí, en un bucket llamado *openchat*. Desde ahí se consume el frontend, y los *assets* de este se consumen desde otro CDN, también con AWS Cloudfront y AWS S3, en un bucket llamado *e1g20* donde se encuentran las imágenes, como el favicon y el logo, y también el archivo css.

Backend y Load Balancer
El backend se implementó dentro de una instancia de AWS EC2 con docker-compose, desde la cual se conecta a Redis y a la BD.
Además, un load balancer se encarga de replicar la instancia en caso de que el uso de la CPU supere el 50% y, posteriormente, elige a qué instancia dirigir una llamada a la api.

BD
La base de datos se encuentra corriendo dentro de una instancia de EC2 separada del backend. De este modo, al escalar este último, la BD se mantiene única y los datos se mantienen consistentes entre instancias.

Mensajes en tiempo real
Se utilizaron websockets (socket.IO), de modo que los usuarios, al entrar a un chatroom, se conectan a un canal y, cuando se envía un mensaje por dicho chat, el backend notifica a todos los usuarios conectados a él.
Una restricción que posee es que puede mantener, a lo más, alrededor de 600 conexiones estables simultáneas. Pero, dado el tamaño actual de la aplicación, esto no es problema.

Notificaciones por mail
Se utilizó el servicio de AWS de Simple Email Sending (SES). Este almacena un template del correo, y desde el backend se conecta al servicio para enviar un mail según ese template. Se utilizó dado que es sencillo de implementar y ya que la aplicación completa se encuentra en AWS.
Una de las restricciones que tiene es el límite de 200 mails por día pero, dado el tamaño de la aplicación, es una cota superior lejana. También restringe la tasa de envío de mails a 1 por 1 segundo, aunque permite superarla por periodos cortos de tiempo (no más de 4 segundos). Pero dado el tamaño actual de la aplicación, es un límite razonable.
Otra restricción es que, en un principio, sólo permite enviar correos a cuentas validadas por AWS; y para cambiar este comportamiento se debe realizar una solicitud a través de la página. Esto, más que nada, impone la restricción de tiempo de esperar a que la solicitud sea resuelta.

Cache
Para la implementación de la capa de caché utilizamos Redis para almacenar información de la aplicación.
La infraestructura consiste en una instancia AWS EC2 con redis-server corriendo. Redis fue instalado y configurado manualmente. Tanto en local como en producción nuestro backend se conecta directamente a esta instancia que está siempre activa y recibiendo consultas.
Esta instancia fue montada en la cuenta de AWS Educate personal (raespinoza4@uc.cl) para que en el caso de que existiera un error en la configuración no se comiera los créditos del grupo.

Casos de uso:

- Salas: Cada vez que un usuario consulta por las salas del chat el backend primero revisa si están en caché, de no ser así realiza la consulta a la BD, le entrega el resultado al cliente y luego guarda en cache el resultado de la query. Las salas se guardan en forma de string (desde un json) y utilizamos una regla de expiración de 10 segundos. Elegimos esta regla de expiración para poder probar la aplicación y verificar que realmente se están obteniendo las rooms desde redis, luego de este

tiempo se pudieron haber creado nuevas salas y es necesario ir a buscar la información a la base de datos nuevamente.

- Mensajes: Se almacenan en caché los últimos 10 mensajes de cada sala de chat. Como método de reemplazo se eligió FIFO, de modo que, si se llena la caché y se quiere almacenar un nuevo valor, se sobreescribe el del más antiguo. Para este caso es bastante útil si queremos mostrar un set de últimos mensajes a los usuarios siempre que llegue uno nuevo el que salga de caché sea el primero que se envió dejando espacio al mensaje nuevo, lo que confirma nuestra selección de FIFO.