SLACK CHAT SYSTEM DESIGN DOC

Yaqian Qi

Overview

A chat system is a real-time communication platform that allows users to exchange text messages in a conversational manner. A typical chat system comprises various components such as clients, servers, message queues, and databases, which work together to enable seamless communication between users. This doc explores some user story and standards of building a chat system and proposes a high-level architecture of a chat system.

User Stories

- As a user, I want to be able to create an account so that I can log in and access the chat system.
- As a user, I want to be able to search for other users in the chat system so that I can connect and chat with them.
- As a user, I want to be able to send and receive messages in real-time so that I can communicate with other users.
- As a user, I want to be able to see when other users are online or offline so that I can know when they are available to chat.
- As a user, I want to be able to create groups and invite other users to join so that we can have group conversations.
- As a user, I want to be able to customize my profile (e.g. profile picture, status message) so that other users can learn more about me.
- As a user, I want to be able to block or report other users who are abusive or harassing so that the chat system remains a safe and welcoming environment.
- As a user, I want to be able to receive notifications (e.g. sound, visual) when I receive a new message so that I can stay informed and respond in a timely manner.

Requirements

- User authentication and authorization: Users should be able to sign up, log in, and log out securely.
- Real-time messaging: Users should be able to send and receive messages in real-time. Messages may include text, images, videos, and other types of media.
- Message storage: Messages should be stored for later retrieval. Depending on the requirements of the chat system, message storage may include searching, archiving, and/or deleting messages.

- Group messaging: Users should be able to create and join groups or channels for messaging with multiple participants. Groups may have different levels of access (e.g. public, private, invite-only) and may be moderated by designated users.
- Notifications: Users should be able to receive notifications for new messages, mentions, or other events within the chat system. Notifications may be delivered via email, mobile push notifications, or other channels.
- User profiles: Users should be able to view their own profiles, edit their profiles, and view other users' profiles. Profiles may include user avatars, status messages, and other information.

Non-functional Requirements

- Scalability: The chat system should be able to handle a large number of concurrent users and messages. It should be able to scale horizontally or vertically to accommodate increasing load.
- Availability: The chat system should be highly available and should not suffer from extended periods of downtime. It should have a high level of fault tolerance and be able to recover from failures quickly.
- Performance: The chat system should be fast and responsive. Users should be able to send and receive messages in real-time, without significant latency or delay.
- Security: The chat system should be secure and protect user privacy. It should use encryption to protect messages in transit and ensure that users can only see messages intended for them.
- Extensibility: The chat system should be easy to extend and customize. It should be possible to add new features or integrate with other systems as needed.
- User experience: The chat system should provide a seamless and intuitive user experience. It should be easy to use, with a clean and responsive interface that supports multiple platforms and devices.

Out of Scope

- This doc aims to provide high level core function architecture proposal. Derivative advanced features like access management, file sharing are out of scope.

ACCEPTANCE CRITERIA

- The system should allow users to create accounts and log in securely.
- Users should be able to send text messages to other users in real-time.
- The system should support group chat functionality, allowing multiple users to chat in the same conversation.
- Users should be able to view a list of their existing conversations and start new ones.

- The system should allow users to search for other users and add them to their contacts list.
- The system should be able to handle a large number of concurrent users and messages without experiencing performance issues.
- The system should be highly available and resilient, with backup and recovery mechanisms in place in case of failures.
- The system should have strong security measures in place to protect user data and prevent unauthorized access or hacking attempts.
- The system should be easy to use and intuitive, with a clean and modern user interface.

Design

On a small scale, all services could fit in one server. The number of concurrent connections that a server can handle will most likely be the limiting factor. This is a good starting point since we are building something from scratch. A chat system can be broken down into three major categories: stateless services, stateful services, and third-party integration.

Stateless Services

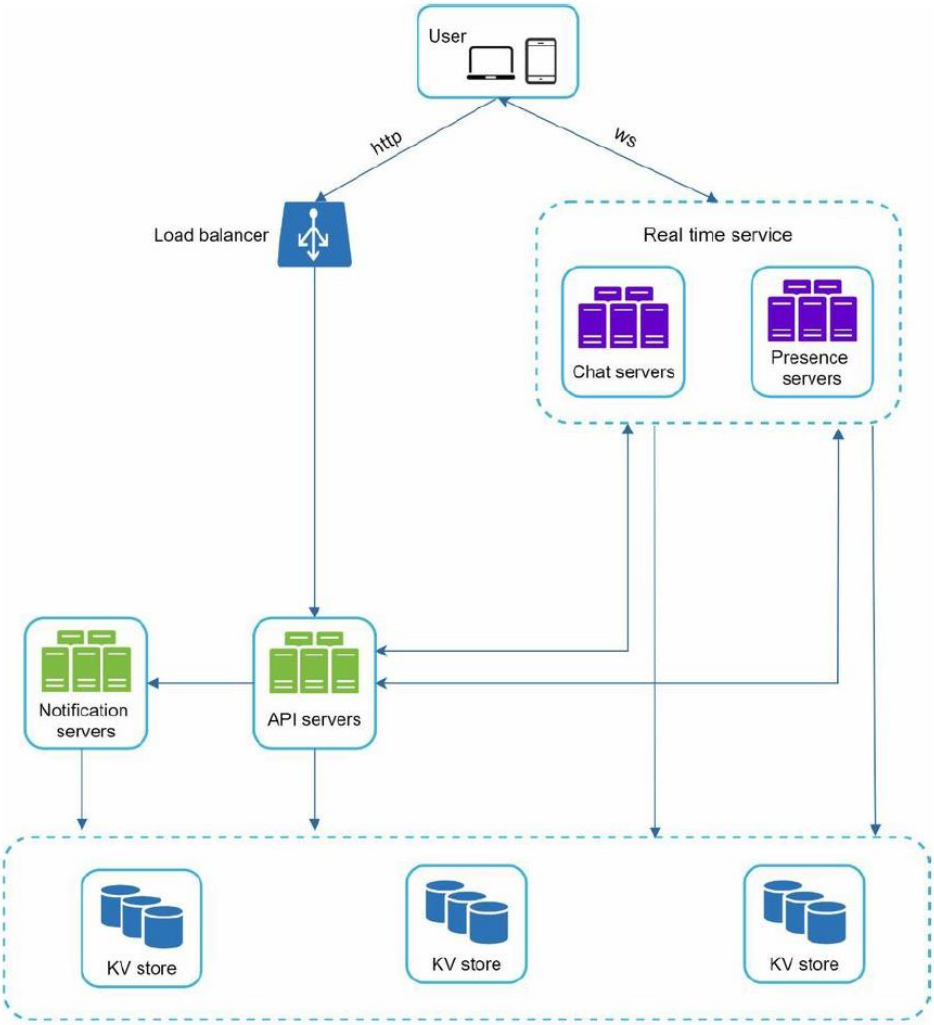
Stateless services are traditional public-facing request/response services, used to manage the login, signup, user profile, etc. These are common features among many websites and apps. Stateless services sit behind a load balancer whose job is to route requests to the correct services based on the request paths. These services can be monolithic or individual microservices.

Stateful Service

The only stateful service is the chat service. The service is stateful because each client maintains a persistent network connection to a chat server. In this service, a client normally does not switch to another chat server as long as the server is still available. The service discovery coordinates closely with the chat service to avoid server overloading.

Third-party integration

For a chat app, push notification is the most important third-party integration. It is a way to inform users when new messages have arrived, even when the app is not running. Proper integration of push notification is crucial.



Detailed Description of each Architecture Element:

Client Applications:
The client applications will be installed on the user's device (desktop or mobile). They will be responsible for sending and receiving messages, creating, and joining channels, and managing the user's account. The client will communicate with the server through RESTful APIs, WebSocket or other communication protocols.

Server:
The server will receive requests from the clients, validate the requests, and send responses. The server will handle user authentication and authorization, manage channels, and store message history. The server will be responsible for distributing messages to clients connected to a particular channel. It will also provide APIs for the clients to perform CRUD operations on users, channels, and messages.

Database:
The database will store user information, channel information, and message history. It will provide ACID guarantees for transactions and support a schema for the data stored. The database will be accessed by the server for storing and retrieving data. A NoSQL database can be used based on the specific requirements of the application.

API servers:
API servers handle everything including user login, signup, change profile, etc.

Wrap up

In this doc, a chat system architecture that supports both 1-to-1 chat and small group chat is presented. WebSocket is used for real-time communication between the client and server. The chat system contains the following components: chat servers for real-time messaging, presence servers for managing online presence, push notification servers for sending push notifications, key-value stores for chat history persistence and API servers for other functionalities.