Chat System

Understand the problem and establish design scope: know if the focus is one-to-one chat or group chat

Qu: Is chat 1:1 or group?
Ans: 1:1

Qu: mobile app or web or both?
Ans: both

Qu: app scale? Startup or massive?
Ans 50 mn daily active user

Qu: for group chat: what is group member limit?
Ans: max 100

Qu: what features app support
Ans: 1:1 chat, group chat, online indicator. Only support text message

Qu: end to end encryption required
Ans: not req for now

Qu: message size limit
Ans: yes, length < 100,000 chars

Qu: how long to store chat history
Ans. forever

Functional req

1. One to one chat
2. Small group (max 100 people)
3. Online presence
4. Multiple device support
5. Push notification

<u>Propose HLD and get buy in</u>

- client can be either mobile or web
- receive message from other client
- find right recipient for each message and relay message to recipients
- if recipient is not online, hold message for that recipient on server until she is online.

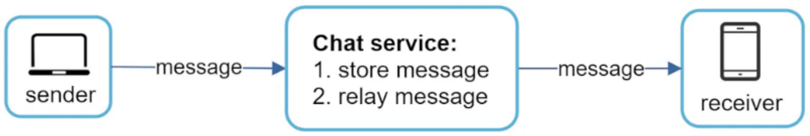


Figure 2

sender send message using time tested HTTP connection

- use keep-alive header to maintain persistent connection with chat service, also reduce TCP handshake

receiver side can use different techniques e.g. polling, long polling, and web socket

polling - client regularly ask server if there are available messages based on its polling frequency.

long polling - client hold the connection open until there are actually new message available or timeout threshold has been reached, once receive message, immediately send another new request to server, restart the process

drawback - server and receiver might not connect to same server since http is stateless, also don't know if client is disconnected. It is inefficient as well

web socket (commonly used) - bi-drectional asynchronous connection. Connect even if firewall is in place, because they use port 80 and 443

High level design

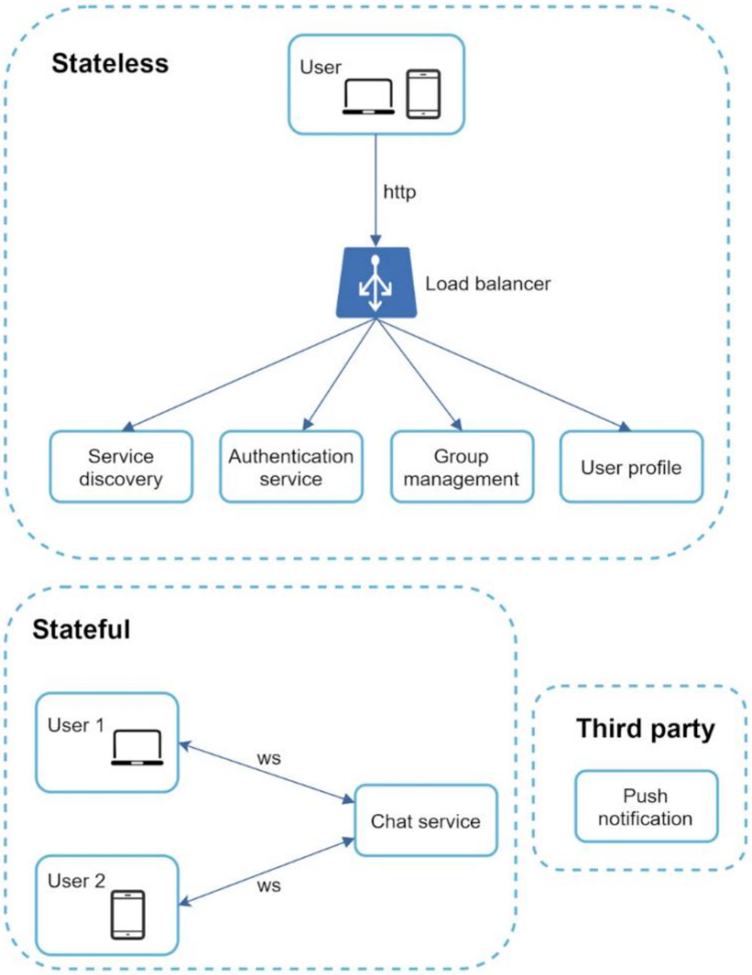
NOTE: only main communication channel is via web socket, other feature (signup, login, user profile) use traditional req/response method over HTTP

stateless service - load balancer route request to stateless service based on their request path. Service can be monolithic or micro service. Eg: Service discovery: give client list of DNS host name of chat server that client could connect to.

stateful service - only chat service - client maintain persistent network connection to chat server, and not switch as long as server is available

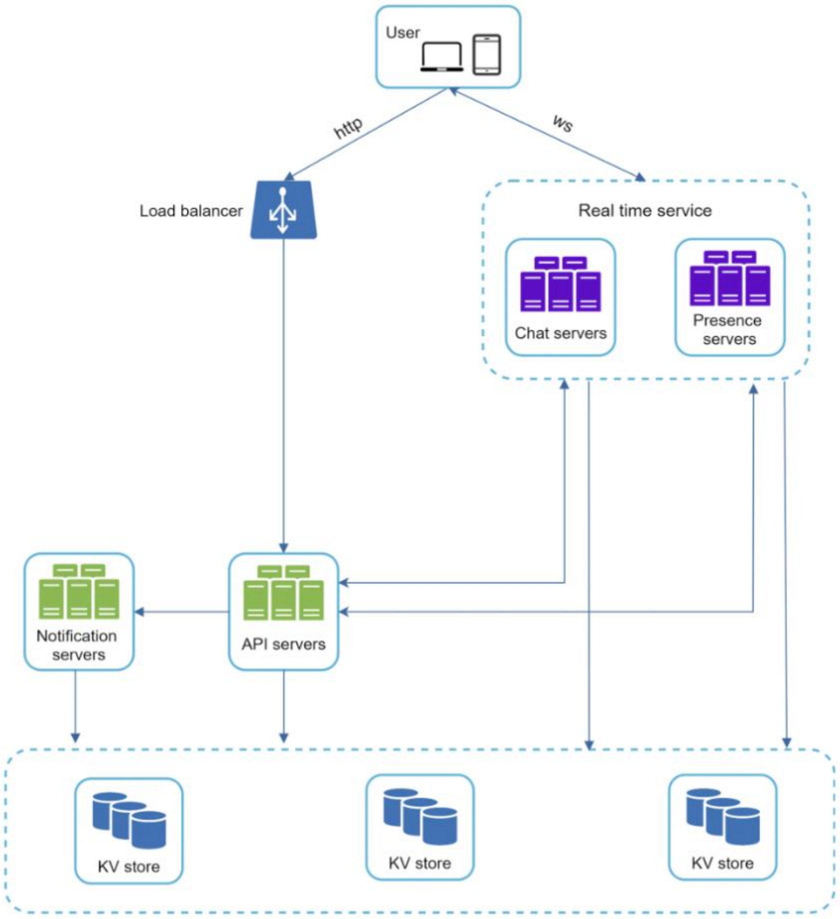
third party integration

eg push notification to let user know of new message even app is not running



scalability

1 mn concurrent user - per user = 10k memory - single server -> single point of failure, but lets start with singe server



chat server - for send/receive message
presence server - manage online / offline status
API server - handle everything (signup , login, change profile etc)
notification server - push notification
KV store - store chat history

storage

- generic data (user profile setting, user friend list) = relational db
- chat history - enormous data, not freq access, only use for search option -> use KV store (each horizontal scale, low latency to search, adapted by other app like fb(base), discord (Cassandra)

data models

message table for 1:1 chat - message_id (PK), and time sortable: 1. Use global 64 bit sequence number generator like snowflake 2. Local sequence number generator

message_id	Biting
message_from	bigint
message_to	bigint
content	text
created_At	timestamp

message table fr group - composite key (msg_id+channel_id)

message_id	Biting
channel_id	bigint
user_id	bigint
content	text
created_At	timestamp

<u>**Design deep dive**</u>

1. Service discovery
2. Message flow
3. Online offline indicator

service discovery

- recommend best chat server based on geography, server capacity etc. eg apache zookeeper
- user tries to login
- load balancer send login request to API server
- once backend authenticate user, service discover find best chat server for user
- user connect to chat server through web socket

message flow

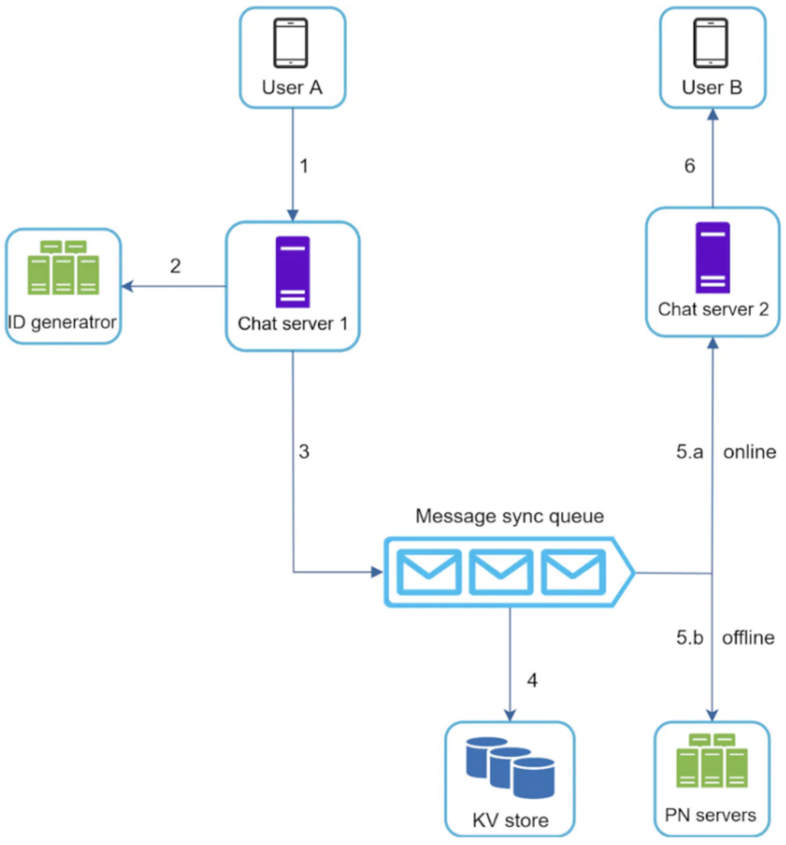
- user send chat message to server 1
- chat server 1 obtain message ID from ID generator
- chat server 1 send message to message sync queue

- message stored in key-value store

-if user B is online, message forwarded to chat server 2 where user B is connected.

- if user B is offline, push notification send from push notification PN server.

- chat server forward message to user B - there is persistent web socket connection between user B and server 2



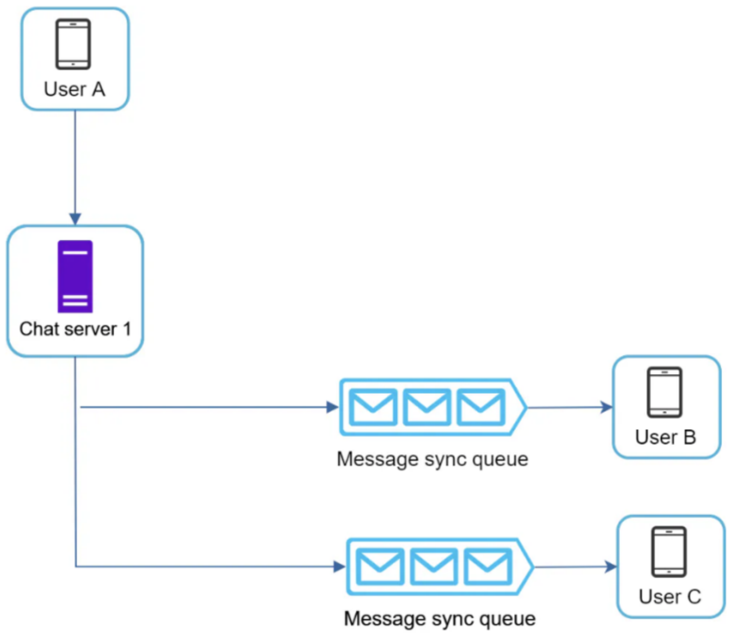
message sync across multiple device

- for both phone and laptop, maintain web socket connection with same chat server

- each device maintain variable = curr_max_message_id - to track latest message on device

- every device check if recipient id = currently logged in user id, and message id in kv store > curr_max_message_id -> then perform synchronization

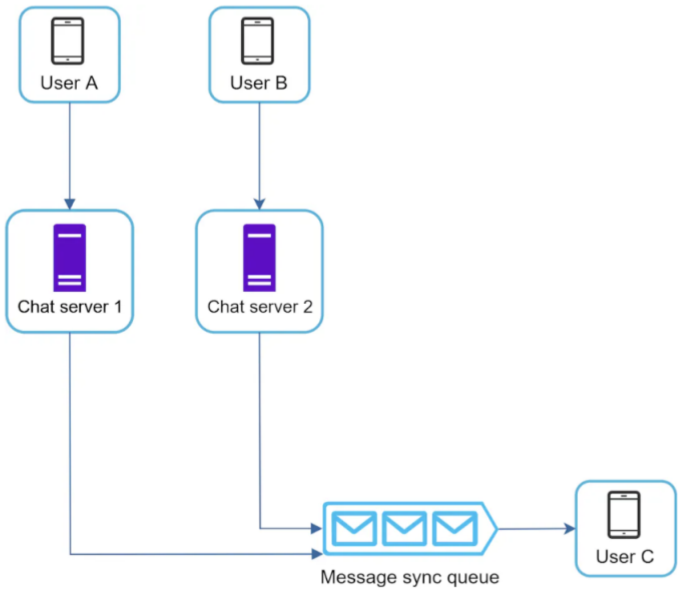
small group chat flow



eg in a group of 3 member, when A send message - copy to each member message sync queue

- However for large group, storing message copy in each member is not acceptable

- also NOTE, that each recipient can receive message from multiple sender



Online presence:

user login flow - once web socket connection built between client and real time service - A online status and last_active_at timestamp store in KB store -> presents indicator shows user is online once she logged in

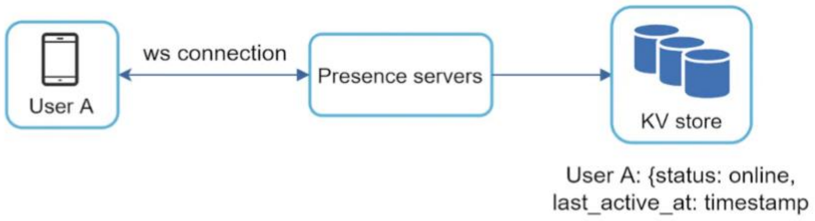
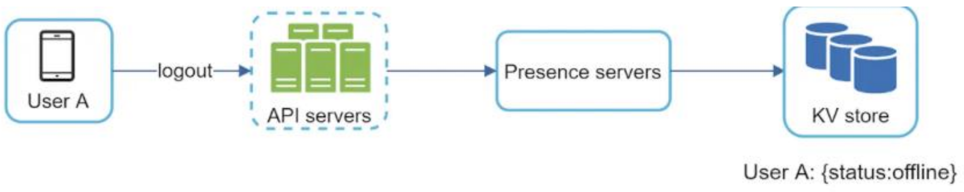


Figure 16

user logout flow



User disconnection

heartbeat mechanism - online client send heartbeat event to presence server eg every 5 second - if presence server receive heartbeat within certain time - online, else if not respond to 3 heartbeat - offline

Online status fanout

Presence server use publish subscribe model - where pair maintain a channel. When A status change - subscribed channel A-B, A-C, A-D will get online status update through real time web socket

Wrap up

1. Extend chat app to support media files eg photos and videos - compression, cloud storage, and thumbnails
2. End to end encryption - only sender and reader can read not other
3. Caching message on client side - to reduce data transfer
4. Improve load time - slack build geographically distributed network to cache user data, channel et. For better load time
5. Error handling

6. Chat server error - if server goes offline, zookeeper provide new chat server to establish new connection
7. Message resent mechanism - retry and queueing

Reference (Further reading)

https://www.codekarle.com/system-design/Whatsapp-system-design.html