Chat Application — System Design

Let discuss about how an chat application works and how can we scale the application so that there may be millions of users.

Let us write down our requirements from application,

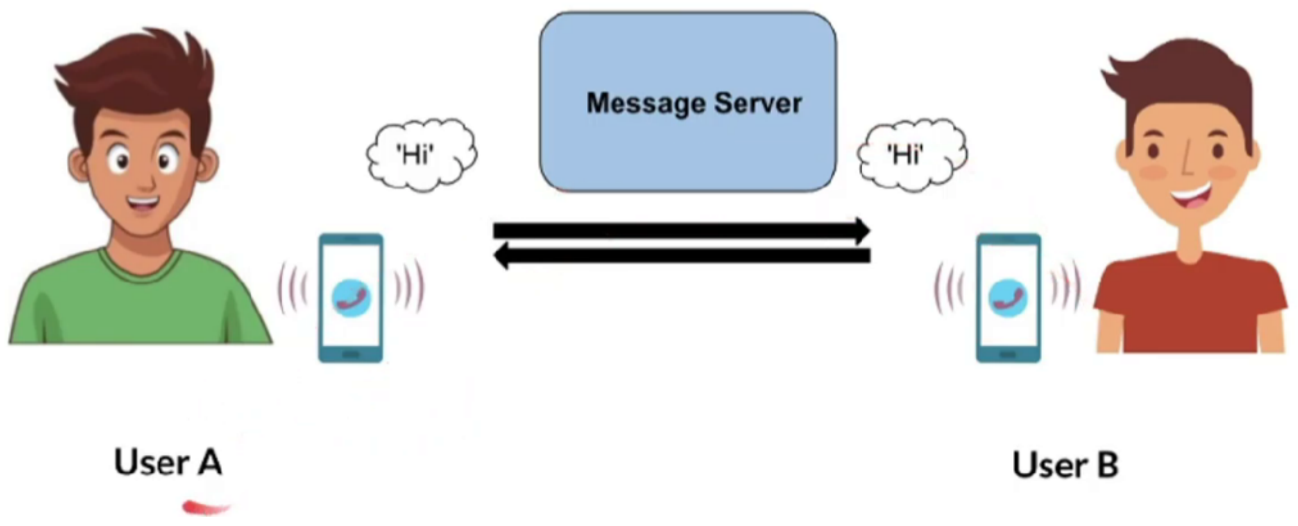
Functional Requirements.

- *One to one messaging* — User A should be send private message to User B.
- *Sent/Delivered/Seen Message updates* — User should be able see the sent messages status if sent, delivered, seen.
- *Last online* — If the user is offline then show what was the last time he was active.
- *Send media messages* — Share the multimedia messages(Audio, Video, Images)
- *Persistent chat* —Save the chat in our DB for life long.

Non — Functional Requirements.

- *Real time experience* — Message should be delivered instantly.
- *Consistent* — Service should be consistent.
- *Scalable* — As the users increases we should be able to scale the application.
- *Availability* — Chat service should be highly available.

Let us see how chat works,

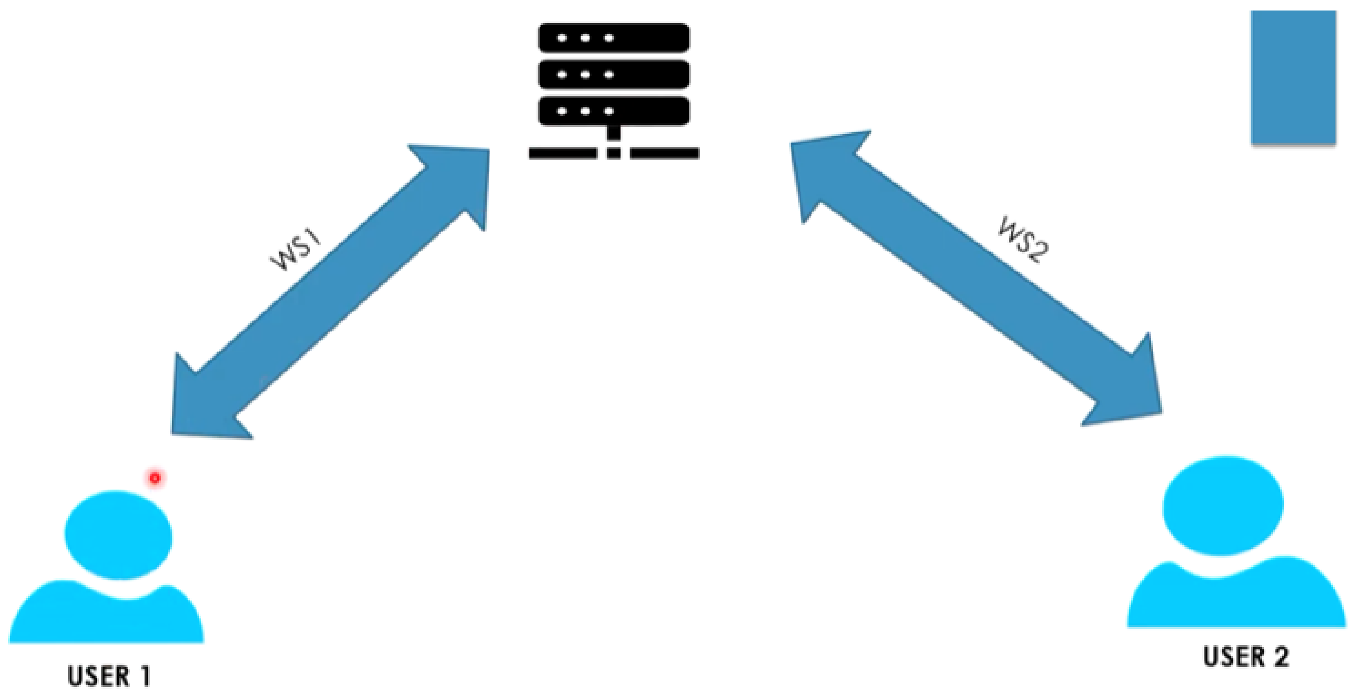


Simple Communication

User A sends a message "Hi" to User B via message server. Now there are different ways to implement this. Lets discuss all the approaches below,

- **HTTP —** Say you send message via HTTP request to server and instruct to send it to user B And User B sends a message by instructing server to deliver it to user A. But the challenge here is HTTP works only from client to server. so server can not send message received from user A to user B.
- **PUSH/PULL —** Say user A send a message to user b via chat server in this method the chat server can PUSH a message to user B or the user B can connect and check if there are any pending message for me.
- **Long Polling** — Say User A send a message to User B, But before sending the message user B connects to server and ask that if there are any messages for me, if no message then it waits for some time and disconnects and will reconnect in some time to check if there are any new message, once user A sends a message to B then it catches the message and replies back. But the challenge here is user b has to make a connection and then wait for certain time then time out if no messages are there again make a connection and check, this is costly operation for server.
- **Web Socket** — This works over single TCP connection. This mechanism has 2 way communications, Server can talk to client and client can also talk to server.

So from all the above approaches we can see that WebSocket is more efficient than others. Let us discuss how this works.



Web Socket Communication

Lets assume we have 2 users **User1** and **User2**, having web socket connection to server as **WS1** and **WS2.** Now if user1 wants to send message to user2 it can be done by sending a message to chat server via web socket1 and once server receives it delivers to user2 via web socket2. Now user 2 can also send messages in this way. So now we have a system where 2 users are able to chat with each other. Now say our application has 2000 users and there are 2000 web socket connections established how the server will handle this i.e how the server will know which user have which socket connection.
We can solve this by having a set or HashMap that will store user id and socket connection object. So now when user1 wants to send a message to user20 then the request goes to server and server will look into the table for user20 what is the connection object and then sends the message to user. With this approach we have satisfied our first functional requirement.

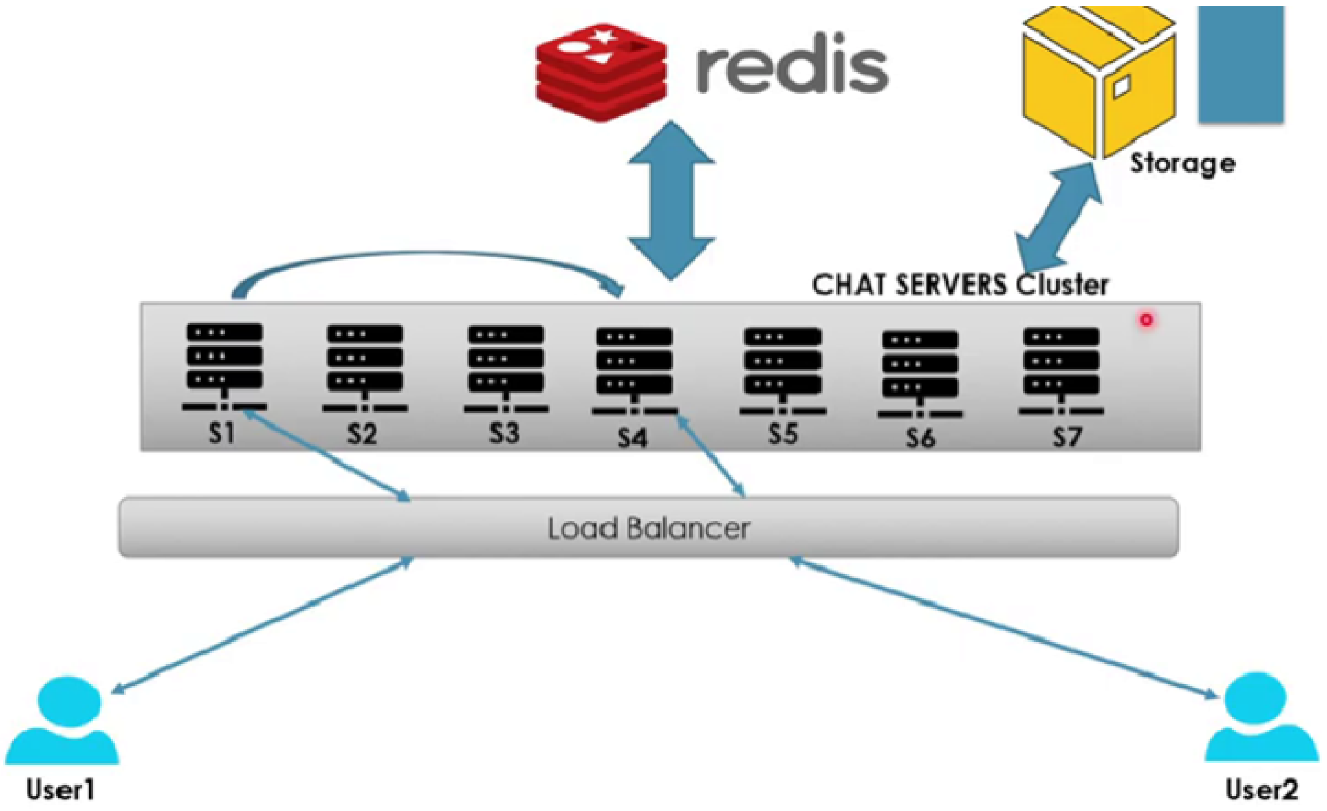
How do we get the message status ?

User1 send a message to u2 then chat server will check in the table for connection object if there is user2 entry it means that connection is active and it will send notification to u1 that it is **delivered** and it will send notification/message to u2 that you have one new message.

When u2 opens the message it will send notification to chat server that message is read and chat server will send it back to u1 that msg is **read**.

We will discuss about offline status in upcoming part.

Scale Chat System



Scale Chat System

Now we have some new components as you can see in the image above. Lets discuss what are they.

1. Load Balancer — A load balancer acts as the "traffic cop" sitting in front of your servers and routing client requests across all servers capable of fulfilling those requests in a manner that maximizes speed and capacity utilization and ensures that no one server is overworked, which could degrade performance.
2. Chat sever cluster — Multiple chat servers in single cluster

3. Redis Cache — Redis is an **open-source** data structure store that is primarily used as a **database, message broker,** or **cache**.
4. Storage DB — Mongo DB to save the user information.

Let us understand how this system works.

User1 makes a connection to our chat server and it will come to load balancer first and load balancer will forward the request to server. Now you may think on what basis load balancer will decide which server should I send the request to process. So there are some approach like forward in round robin way, Least Connections, Hashing Methods, Random Algorithm, etc. So based on what we configure it will send the request to server. Now User1 socket connection will be on server and User2 will be connection via socket on server2.

Now if user1 wants to sends a message to user 2 how it will know which server of chat cluster the user2 is connected to send message as each server will have its table of connected users and socket object. For this we can use **Redis Cache.**

In Redis Cache we store info about which user is connected to which server. So now in our example we want to send a message to user2. User1's Web Socket1 connection will reach server1 and then it will lookup in redis cache where my user2 is i.e. which server he is connected to and from that it will transfer the request to respective server of user2. When the user reconnected then again it will be assigned to some server and the redis cache will be updated.

When a user is connected to server in cluster we have something called heartbeat ping which will say the connection is still active. So from this we can see all the available online user and their last seen if they disconnect.

Media Message — WebSocket can accept media message in base64 format so that can be stored in server DB and then the recipient can get the data again. and to persist the messages we can store all chats in MongoDB.

How do we handle if the recipient is offline?

To handle this we can have one table that will have 3 columns to_user, from_user, time_stamp. Now user1 sends a message to user2 but user2 is offline and Now when user comes back online he will check in that table if there are any messages for me if yes then it will fetch them. Or there can be also push notifications.

In this article we have implemented all functional requirements and scaled our chat system.