# Detailed Design of WhatsApp

Learn about the design of the WhatsApp system in detail, and understand the interaction of various microservices.

We'll cover the following



## Detailed design

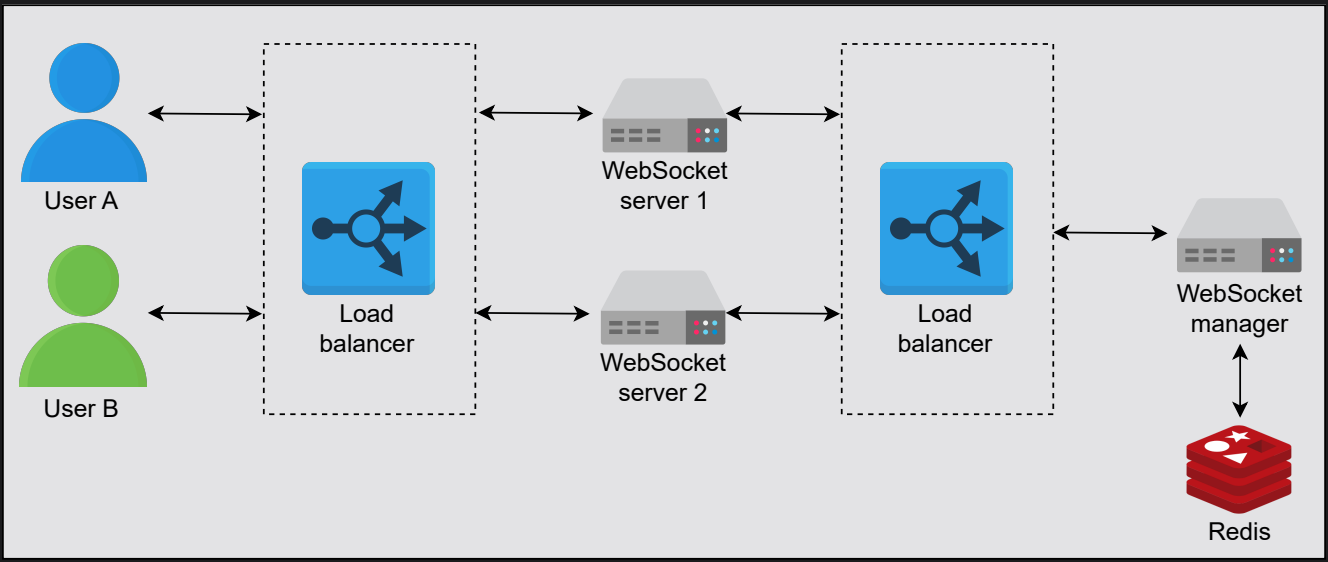
The high-level design discussed in the previous lesson doesn’t answer the following questions:

- How is a communication channel created between clients and servers?
- How can the high-level design be scaled to support billions of users?
- How is the user’s data stored?
- How is the receiver identified to whom the message is delivered?

To answer all these questions, let’s dive deep into the high-level design and explore each component in detail. Let’s start with how users make connections with the chat servers.

## Connection with a WebSocket server

In WhatsApp, each active device is connected with a **WebSocket server** via WebSocket protocol. A WebSocket server keeps the connection open with all the active (online) users. Since one server isn't enough to handle billions of devices, there should be enough servers to handle billions of users. The responsibility of each of these servers is to provide a port to every online user. The mapping between servers, ports, and users is stored in the WebSocket manager that resides on top of a cluster of the data store. In this case, that's Redis.




Two users are connected via WebSocket handlers

### Point to Ponder

**Question**

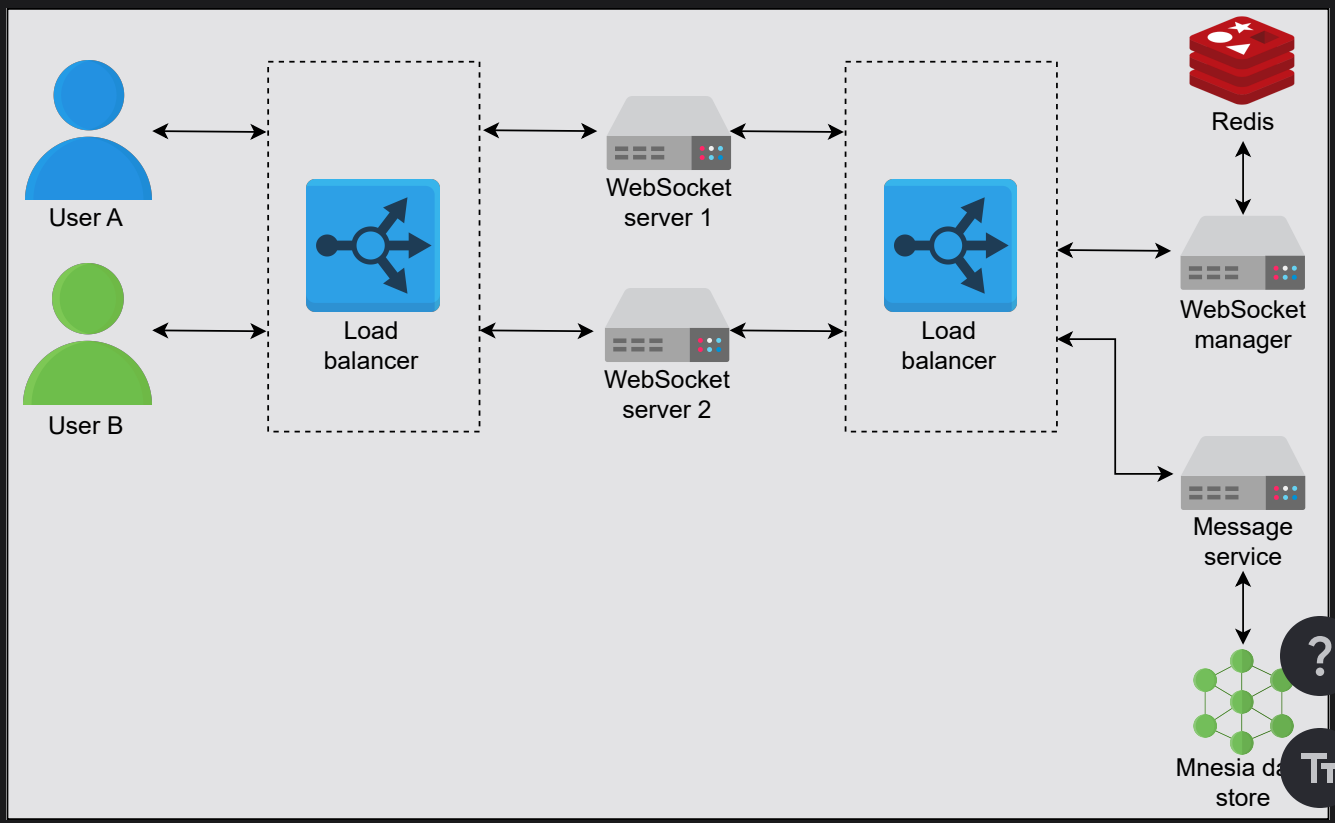
Why is WebSocket preferred over HTTP(S) protocol for client-server communication?

Show Answer

## Send or receive messages

The WebSocket manager is responsible for maintaining a mapping between an active user and a port assigned to the user. Whenever a user is connected to another WebSocket server, this information will be updated in the data store.

A WebSocket server also communicates with another service called message service. **Message service** is a repository of messages on top of the **Mnesia database cluster**↴. It acts as an interface to the Mnesia database for other services interacting with the databases. It is responsible for storing and retrieving messages from the Mnesia database. It also deletes messages from the Mnesia database after a configurable amount of time. And, it exposes APIs to receive messages by various filters, such as user ID, message ID, and so on.




WebSocket communicates with message service on top of Mnesia database cluster

Now, let's assume that user A wants to send a message to user B. As shown in the above figure, both users are connected to different WebSocket servers. The system performs the following steps to send messages from user A to user B:

1. User A communicates with the corresponding WebSocket server to which it is connected.
2. The WebSocket server associated with user A identifies the WebSocket to which user B is connected via the WebSocket manager. If user B is online, the WebSocket manager responds back to user A's WebSocket server that user B is connected with its WebSocket server.
3. Simultaneously, the WebSocket server sends the message to the message service and is stored in the Mnesia database where it gets processed in the first-in-first-out order. As soon as these messages are delivered to the receiver, they are deleted from the database.
4. Now, user A's WebSocket server has the information that user B is connected with its own WebSocket server. The communication between user A and user B gets started via their WebSocket servers.
5. If user B is offline, messages are kept in the Mnesia database. Whenever they become online, all the messages intended for user B are delivered via push notification. Otherwise, these messages are deleted permanently after 30 days.

Both users (sender and receiver) communicate with the WebSocket manager to find each other's WebSocket server. Consider a case where there can be a continuous conversation between both users. This way, many calls are made to the WebSocket manager. To minimize the latency and reduce the number of these calls to the WebSocket manager, each WebSocket server caches the following information:

- If both users are connected to the same server, the call to the WebSocket manager is avoided.

- It caches information of recent conversations about which user is connected to which WebSocket server.

Point to Ponder

**Question**

The data in the cache will become outdated if a user gets disconnected and connects with another server. Keeping this in mind, how long should a WebSocket server cache information?

Show Answer

## Send or receive media files

So far, we've discussed the communication of text messages. But what happens when a user sends media files? Usually, the WebSocket servers are lightweight and don't support heavy logic such as handling the sending and receiving of media files. We have another service called the **asset service**, which is responsible for sending and receiving media files.

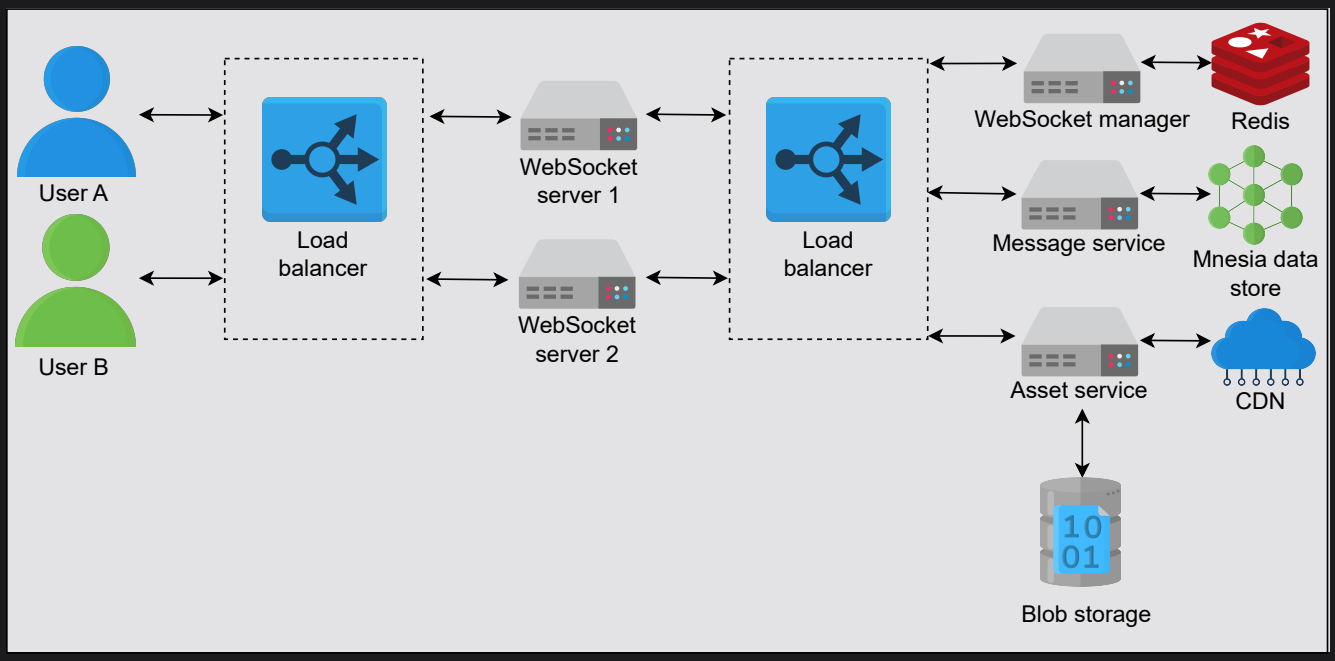
Moreover, the sending of media files consists of the following steps:

1. The media file is compressed and encrypted on the device side.
2. The compressed and encrypted file is sent to the asset service to store the file on blob storage. The asset service assigns an ID that's communicated with the sender. The asset service also maintains a hash for each file to avoid duplication of content on the blob storage. For example, if a user wants to

upload an image that's already there in the blob storage, the image won't be uploaded. Instead, the same ID is forwarded to the receiver.

3. The asset service sends the ID of media files to the receiver via the message service. The receiver downloads the media file from the blob storage using the ID.
4. The content is loaded onto a CDN if the asset service receives a large number of requests for some particular content.

The following figure demonstrates the components involved in sharing media files over WhatsApp messenger:




Sending media files via the asset service

## Support for group messages

WebSocket servers don't keep track of groups because they only track active users. However, some users could be online and others could be offline in a grou For group messages, the following three main components are responsible for delivering messages to each user in a group:

- Group message handler
- Group message service

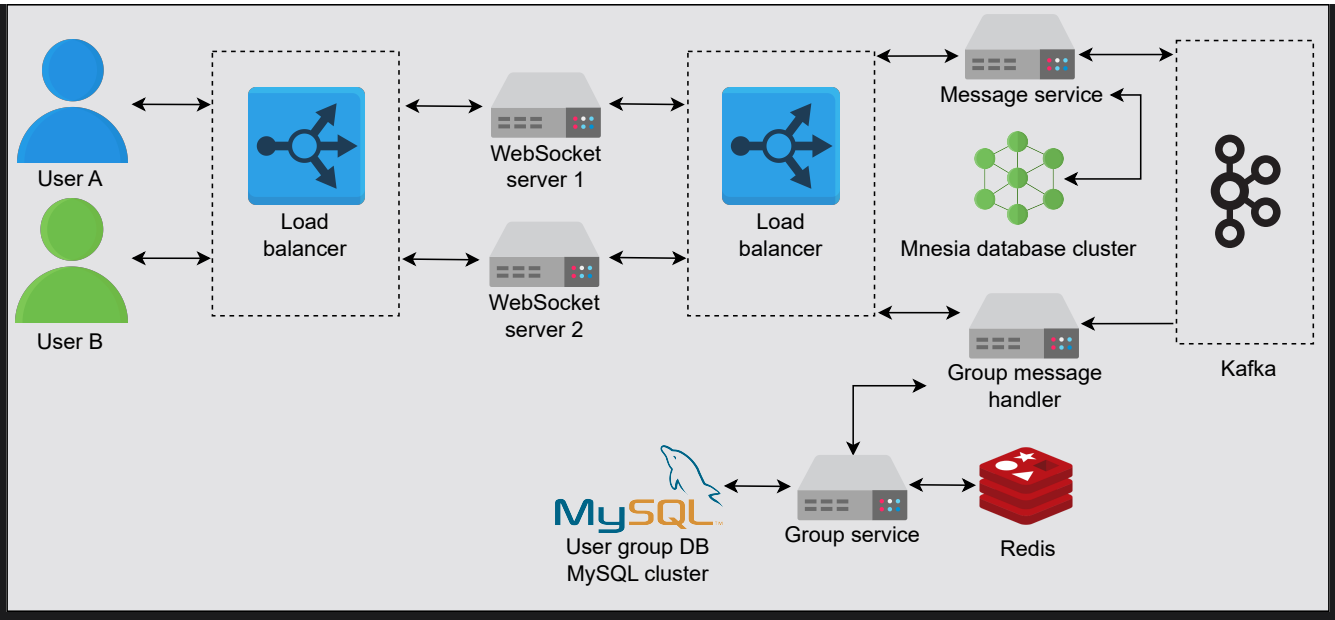
- Kafka

Let’s assume that user A wants to send a message to a group with some unique ID—for example, `Group/A`. The following steps explain the flow of a message sent to a group:

1. Since user A is connected to a WebSocket server, it sends a message to the message service intended for `Group/A`.
2. The message service sends the message to Kafka with other specific information about the group. The message is saved there for further processing. In Kafka terminology, a group can be a topic, and the senders and receivers can be producers and consumers, respectively.
3. Now, here comes the responsibility of the group service. The **group service** keeps all information about users in each group in the system. It has all the information about each group, including user IDs, group ID, status, group icon, number of users, and so on. This service resides on top of the MySQL database cluster, with multiple secondary replicas distributed geographically. A Redis cache server also exists to cache data from the MySQL servers. Both geographically distributed replicas and Redis cache aid in reducing latency.



5. In the last step, the group message handler follows the same process as a WebSocket server and delivers the message to each user.

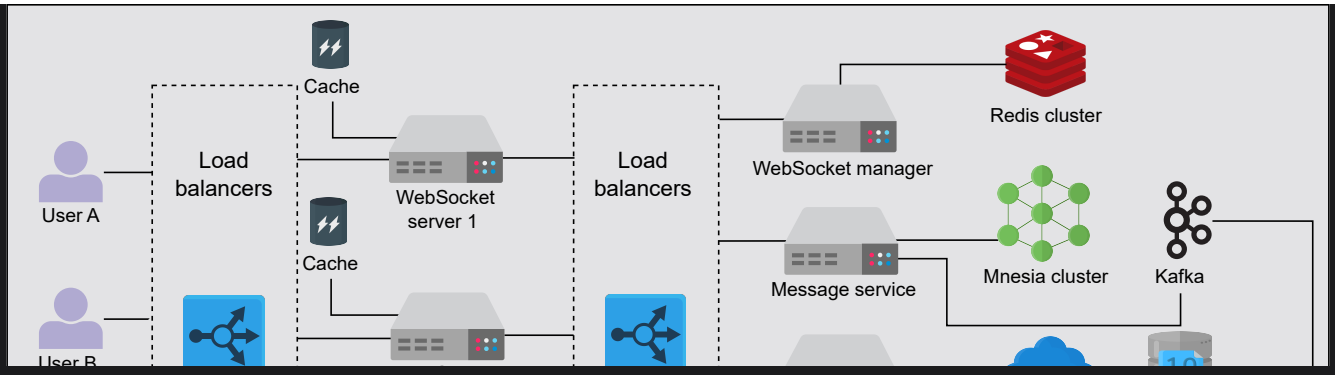



Components responsible for sending group messages

**Optional: How encryption and decryption work in WhatsApp**

## Put everything together

We discussed the features of our WhatsApp system design. It includes user connection with a server, sending messages and media files, group messages, and end-to-end encryption, individually. The final design of our WhatsApp messenger is as follows:


Cache
Load balancers
User A
WebSocket server 1
Cache
Load balancers
WebSocket manager
Redis cluster
Message service
Mnesia cluster
Kafka