WhatsApp System Design

Functional Requirements

- Send/receive chat messages
- Group chat (2 to 257 users)
- Track the status of sent message
- Use phone number to register new account
- Push notification
- Receive messages when online
- Analytics/Monitoring
- End to End Encryption
- Support for WhatsApp web
- Change the settings (Display/Privacy) of app
- Support media files as a message
- Backup of database
- Post the status on WhatsApp

Non-Functional Requirements

- Highly Available
- Scalable (load balance with increasing users)
- Fault Tolerant or Disaster Recovery
- Minimum Latency
- Consistency
- Durability

API Specification

- **registerAccount(API key, User Info)** – User Info includes phone number as user id
- **boolean validateAccount(API key, User Id, validation code)** – Validation code is sent to user as OTP which he enters and calls this API

- **int initiateDirectChatSession(API key, Sender User ID, Receiver User ID, Handshake Info)** – Handshake Info to ensure end to end encryption. API returns session ID
- **int sendMessage(API Key, session ID, msg type, msg)** – msg type can be a text msg or status msg. API returns msg ID
- **getMessageStatus(API key, msg ID)**
- **readNewMessages(API key, session ID, msg ID of last msg received)** – This will return list of all new messages after the pointer location to last msg
- **initiateGroupChatSession(API key, group Info)** – group Info includes group name, group description, etc. It returns group ID
- **addUserToGroup(API key, User ID, Group ID)**
- **removeUserFromGroup(API key, User ID, Group ID)**
- **promoteUserToAdmin(API key, Group ID, User ID)**
- **postStatus(API key, user id, status msg)** – Returns session ID
- **removeStatus(API Key, session ID)**
- **viewStatus(API key, user id of whom you want to see status)**
- **deleteGroup(API key, Group ID)**
- **deleteAccount(API key, User ID)**

➢ High Level Design

Routing Service is a proxy service that serves as a gateway of communication from user to other microservices.

When user comes online, the user creates a web-socket connection with routing service which further creates a similar connection with every micro service. API calling would follow the same communication mechanism.

Database layer would be part of every microservices

Microservices include Groups, Sessions, Fan Out, User, User Registration

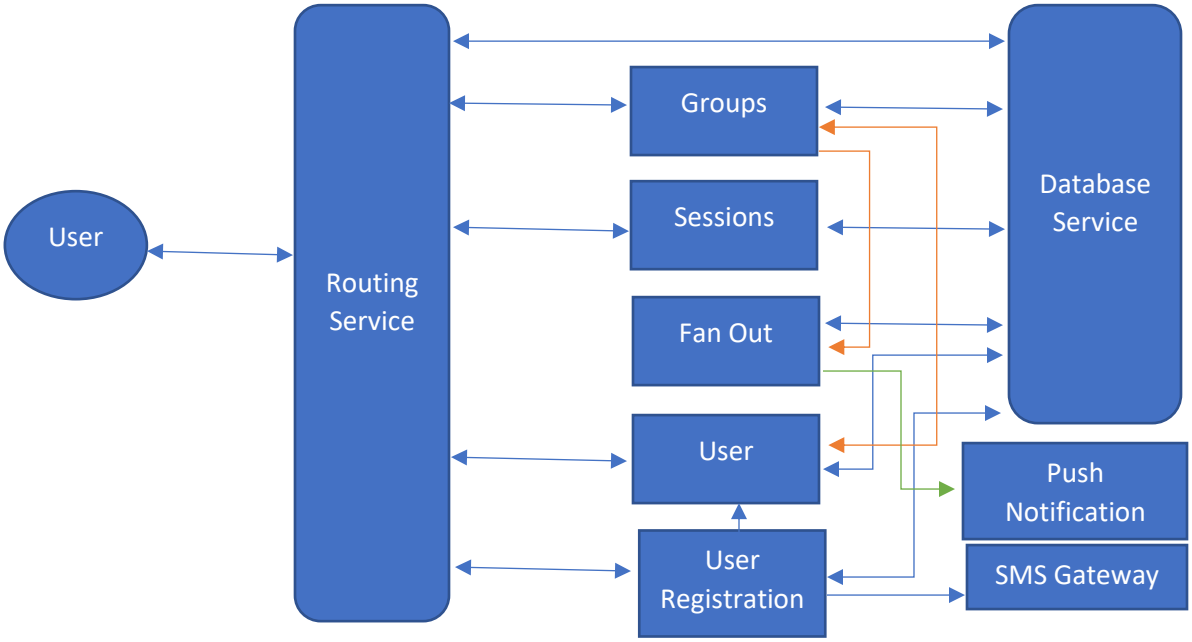


Fig. High-level architecture Diagram of the application

1. User and User Registration Service

Primary Key	User ID
	User details
	Validation code
	Creation Date
User Registration Schema	

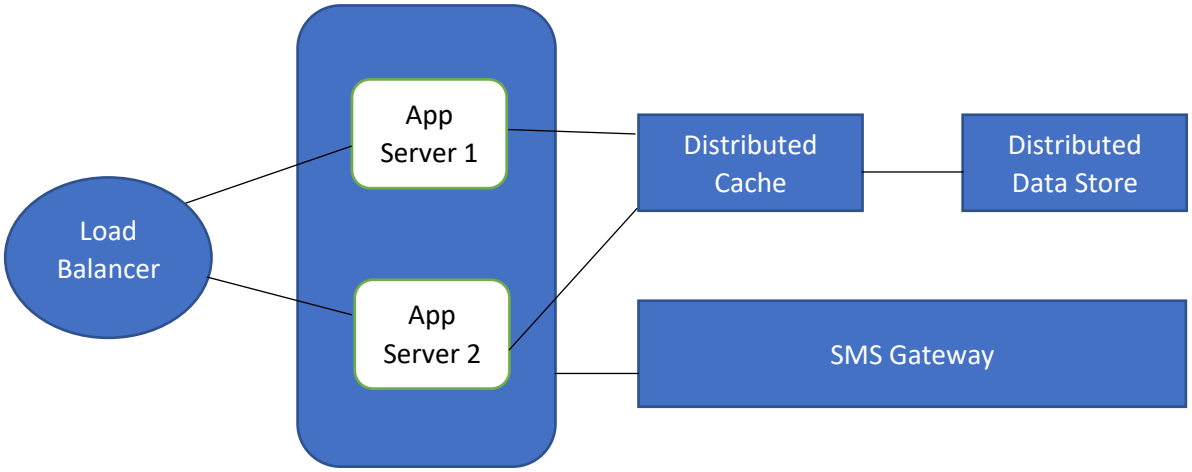


Fig. User and User Registration Service

Load Balancer is connected to the stateless app servers and the servers are storing the data in cache and data store via write-through mechanism. As load increases, we can add more application servers as they are stateless. The validation code generated upon creation of new tuple in user registration table, is sent to the user (user id as primary Key) via SMS Gateway. Upon validating the code received from user and the datastore value, user registration service requests user service to add the user and do the required operations. If the existing user with same user id but on different phone, then the API key token is replaced accordingly. Also, user registration table is time bound, so it expires in about 15 minutes. Therefore, the table needs to store creation time too. Database scheme for user registration is given above.

2. Group Service

Primary Key	Group ID	Primary Key and Secondary Index	Group ID and User ID
Group Schema	Group name		Creation Date
	Group details		User Type (Admin)
	member count	**Group Membership Schema**	
	Creation Date		

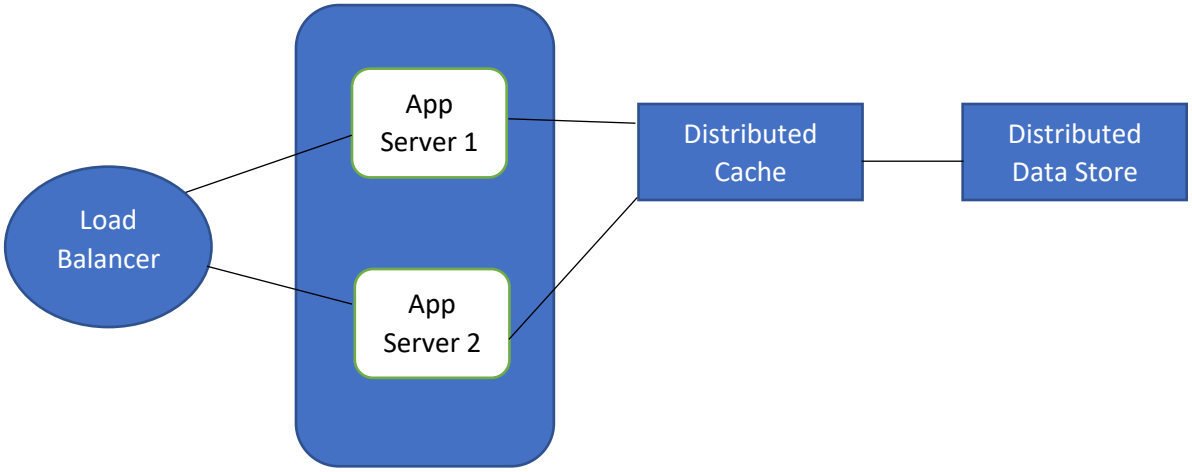


Fig. Group Service

Lookup for the queries for a group will be quick as we need to investigate the partition dedicated for that group. When user changes the phone, user can query group membership table using user id as secondary index to get information on all the groups he's been connected to earlier. Here, for secondary key mechanism, there can be two ways:

- **By Group ID (Local Index):** User Info will be spread across multiple partitions according to the groups he's connected with.

- **By User ID (Global Index):** User info will reside in one single partition.

3. Session Service

Primary Key	Session ID	Primary Key	Session ID and Timestamp
Session Schema	Creation Date		Creation Date
	Session Type (Private or Group)		Session Type (Group/Private)
			Sender ID
		Value is All in case of Group session	Recipient ID
		Returns Msg Id	Message Data
		Session Message Schema	

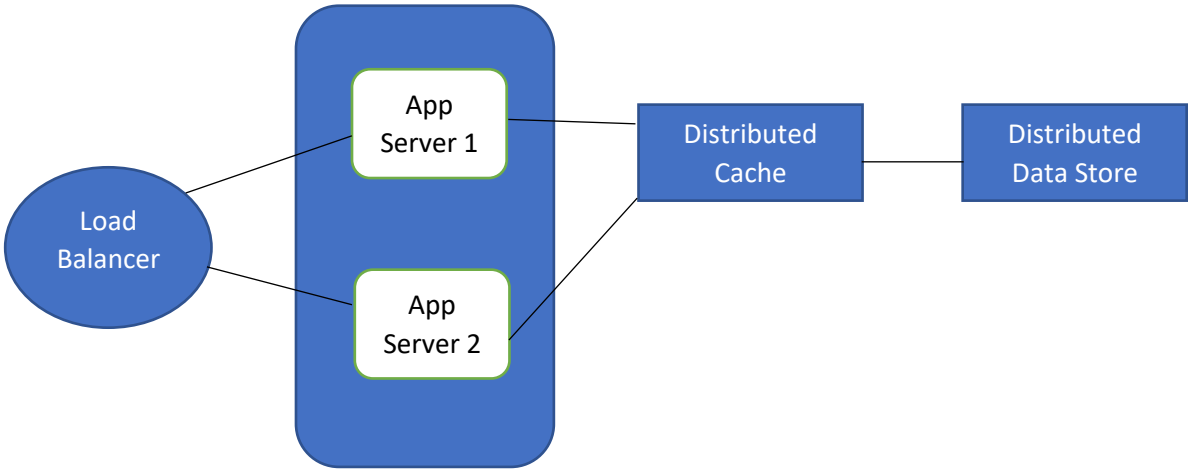


Fig. Session Service

Session service stores session information and message statistics for private chats and group chats. To enable the end to end encryption, the encryption keys are passed from one user to another.

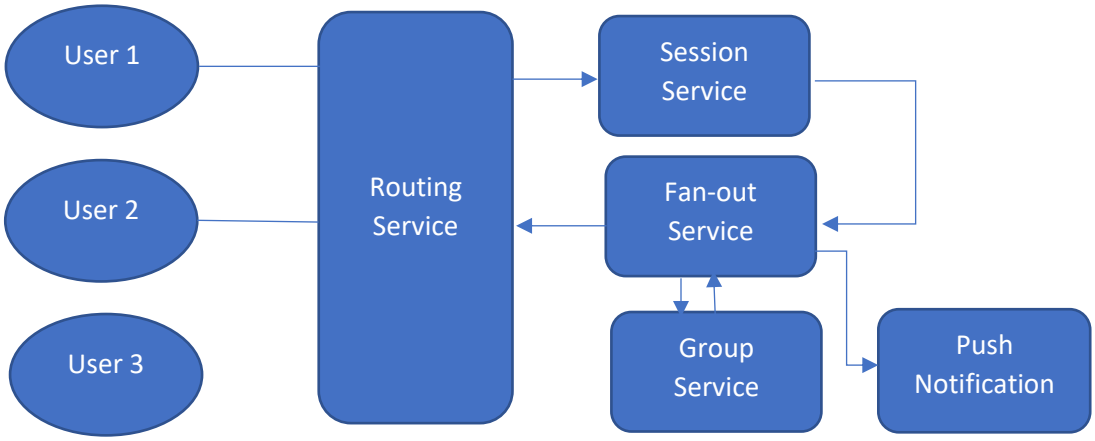
- **Dedicated/Private Chat Sessions:** Session id can be given as concatenation of user ids of involved persons
- **Group Chat Sessions:** Session id can be generated randomly with some prefix for identifying it is a group chat session. In a group chat session, if 10 users send message at same time, then timestamp will not be unique. However, it is a primary key. To resolve this, we can use epoch timestamp with precision of 1 second and append a counter value every 1 second. Alternatively, we can include a counter in Session schema and add a record in session message schema on a valid transaction only. On every transaction counter value will be incremented, and by so, for a timestamp, 9 users will not be able to get that and only 1 user will get a unique timestamp and can commit the transaction.

Everything about the session information (msgs related to that session) can be retrieved from a single partition as the tables will be sharded by session ids. If user id is used to shard the table, then the approach will not be efficient as to retrieve all the msgs in the group, you need to query across all the partitions of the users who are part of that group.

4. **Fan out service**

Fan out service receives msg from session service and then decides what needs to be done with the msg based on the msg if it is private or group. If private msg then it sends it to routing service and then to user, but if it is group msg then it sends to group service, identifies users in group and send to routing service and to all users. However, if the user is offline then there is no connection between routing service and the user and then the fan out service sends it to push notification service.

Use Case:



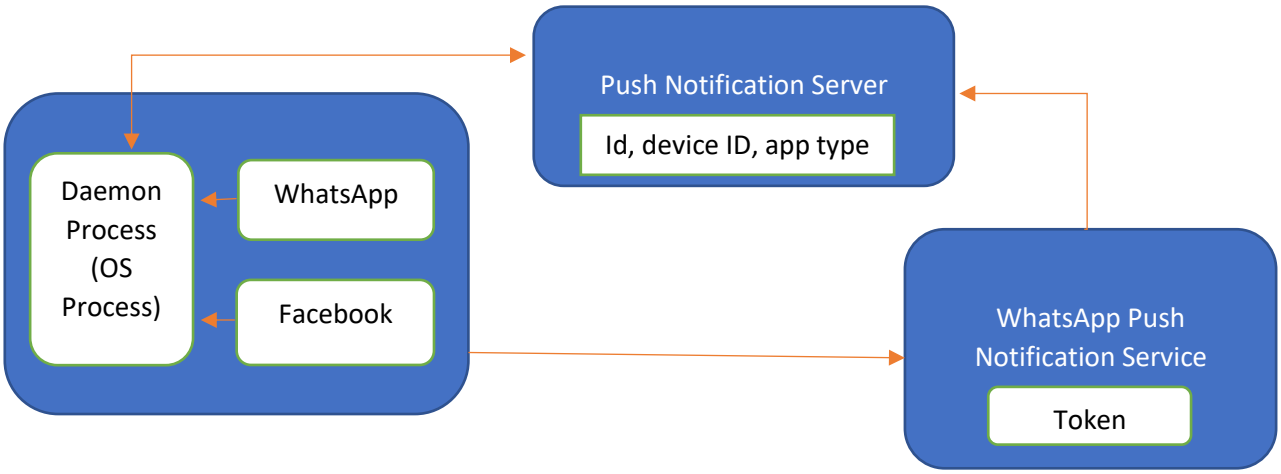
In this case, user 3 is offline. User 1 and user 2 are connected to WhatsApp routing service via web socket connection. User 1 sends message to group. It goes to routing service which sends it to session service which then returns session id, and other details. Then it forwards the msg to fan-out service which then queries the group service to get the list of members in the group and then it sends the msg to routing service, which sends message to the connected user, user 2. Once, the message is delivered to user 2, it returns a status

acknowledgement to routing service and then to session service. Similarly, when user 2 has read the msg, a status acknowledgement will be sent to session service and then to user 1 via fanout service and routing service.

For user 3, since it is offline, the routing service returns not found and then sends it to fan-out service which then sends msg to push notification service.
When user 3 comes online, then it connects to routing service, and queries session service for all the messages in the queue after the last message id received by user 3. Then the process for sending status for delivered message and read message will repeat.

Message ID	Sender	Recipient	Message type	Message
Id1	User1	ALL	Data	“Hello”
Id2	User2	User1	Status	“Id1 is delivered”
Id3	User2	User1	Status	“Id1 is read”
Id4	User3	User1	Status	“Id1 is delivered”
Id5	User3	User1	Status	“Id1 is read”

5. **Push Notification Service:**
   There will be Push Notification Servers (PNS) – APN (Apple Push Notification Server) by Apple, GCM/FCM (Google Cloud Messaging/Firebase Cloud Messaging) by Google and WNS (Windows Notification Service) by Microsoft.



Here, a daemon process is running by default which has the web socket connection with Push Notification service. This daemon process comes along with the OS of client (phone), say the daemon process on iOS phone has web socket connection to APN. Once WhatsApp application registers itself on Daemon process, it then calls Push Notification service to make an entry for this app. The entry would have id, device ID, app type and other details. This record will be identified by a token which is returned to daemon process and then it passes token to WhatsApp client application which then registers this token to WhatsApp push Notification Service.

Now, on receiving offline status of user from routing service to fanout service, the status message is passed to WhatsApp Push Notification Service, which then uses the token stored for that device and identifies the related record in Push Notification Server, which then passes the message to daemon process of the device. WhatsApp application would be triggered on receiving such message for showing the push notification on the client phone.