**📌 FB Messenger – High Level Design (HLD) Summary**

**✅ Core Functional Requirements**

* 1:1 chat and group chat
* Real-time messaging with read receipts
* Message history (persisted for years)
* Voice/video calls using WebRTC
* Online/offline delivery & consistency

**📈 Capacity and Bandwidth Estimation**

* DAU: 500 million
* Avg messages/user/day: 40 → 20 billion messages/day
* Message size: 100 bytes + 100 bytes metadata = 200 bytes
* Storage/day: ~2 TB → 3.65 PB for 5 years
* Write Throughput: 2 TB/day = ~25 MB/sec
* Sharding Need: With 4TB max per shard, 3.65PB / 4TB = ~900 shards

**🧠 Important Design Points**

* Use **Push model** for efficient message delivery (requires persistent open connection)
* Map users to connections: user\_id → socket\_connection
* Use **Load Balancer** + **Consistent Hashing** to balance sockets (~10K servers for 500M users)
* **Async writes** preferred over sync (events sent to message storing service)
* FB uses **MyRocks** (MySQL variant optimized for write-heavy use cases)
* **HBase** (wide column store) or **Cassandra** are solid NoSQL alternatives
* **WebRTC** used for real-time voice/video – switch to client-server for group calls

**🧩 Microservices / Components**

* Message Service (Send/Receive)
* Chat History Service (fetch/store messages)
* Group Management Service
* Presence Service (online/offline tracking)
* Notification Service (push notifications)
* WebRTC Gateway (media relays)
* Connection Manager (handles socket lifecycle)

**🗃️ Database Design – SQL / NoSQL Mix**

**✅ SQL Tables**

(Use MySQL/MyRocks with CQRS for write-heavy flows)

sql

CopyEdit

-- User Table

CREATE TABLE Users (

id BIGINT PRIMARY KEY,

name VARCHAR(100),

email VARCHAR(100),

phone VARCHAR(15),

created\_at TIMESTAMP

);

-- 1-to-1 Chat Table

CREATE TABLE Messages (

msg\_id BIGINT PRIMARY KEY,

sender\_id BIGINT,

receiver\_id BIGINT,

content TEXT,

content\_type VARCHAR(20),

timestamp TIMESTAMP,

status VARCHAR(20) -- Sent, Delivered, Read

);

-- Message Metadata Table (Optional)

CREATE TABLE MessageMeta (

msg\_id BIGINT PRIMARY KEY,

chat\_id BIGINT,

metadata JSON

);

-- Group Table

CREATE TABLE Groups (

group\_id BIGINT PRIMARY KEY,

name VARCHAR(100),

created\_by BIGINT,

created\_at TIMESTAMP

);

-- Group Membership Table

CREATE TABLE GroupMembers (

group\_id BIGINT,

user\_id BIGINT,

joined\_at TIMESTAMP,

PRIMARY KEY (group\_id, user\_id)

);

-- Group Messages Table

CREATE TABLE GroupMessages (

msg\_id BIGINT PRIMARY KEY,

group\_id BIGINT,

sender\_id BIGINT,

content TEXT,

content\_type VARCHAR(20),

timestamp TIMESTAMP

);

**✅ NoSQL Tables (HBase / Cassandra)**

**HBase Schema Example:**

* RowKey: user\_id#timestamp
* ColumnFamily: msg\_details
* Columns: msg\_id, receiver\_id, msg\_content, status

Partition by user\_id to ensure all messages from a user stay in one shard.

**☁️ Infrastructure Recommendations**

| **Component** | **Suggested Cloud Service** |
| --- | --- |
| Object Storage (media) | S3 / Azure Blob |
| DB (SQL) | AWS RDS MySQL / MyRocks |
| DB (NoSQL) | Amazon HBase / Cassandra / DynamoDB |
| Message Queue | Kafka / AWS SQS |
| Notification | Firebase / SNS |
| WebSockets / Push | AWS AppSync / Custom via EC2 / NGINX |
| Voice/Video Streaming | WebRTC + Media Servers (TURN/STUN) |
| Monitoring & Metrics | Prometheus + Grafana / CloudWatch |

**✨ Enhancements to Mention in Interview**

* Store message receipts and delivery events in a separate event table or stream
* Auto-delete or archive old messages for compliance
* E2E Encryption design pattern (e.g., envelope encryption)
* Deduplication logic using message\_id + timestamp
* Offline message queue per user with expiry (use Redis or Kafka topic)