Calibri Segoe UI

Social Media Platform System Design

Comprehensive Architecture Documentation

Subham Luitel

November 28, 2025

Document Overview

This document provides comprehensive system design documentation for a social media platform capable of handling 500 million daily active users. The design covers functional requirements, non-functional requirements, architecture components, data flow, scalability considerations, and technology stack selection to build a robust, scalable, and high-performance social networking service.

Contents

1 Introduction 3

2 Functional Requirements 3

3 Non-Functional Requirements 3

4 Assumptions 4

5 Back-of-the-Envelope Calculation 4
- 5.1 Storage Calculation . . . 4
  - 5.1.1 Daily Storage: . . . 4
  - 5.1.2 Metadata Storage Calculation: . . . 4
  - 5.1.3 5 Year Storage: . . . 5
- 5.2 QPS (Query Per Second) Calculation . . . 5
  - 5.2.1 Read: . . . 5
  - 5.2.2 Write: . . . 5

6 High Level Design Architecture (Overview) 5
- 6.1 System Architecture Diagram . . . 6

7 Architecture Design (Component / Service-level detail) 6
- 7.1 Component Details: . . . 6
- 7.2 Microservices: . . . 6
- 7.3 Databases: . . . 6

8 Data Flow 7
- 8.1 User Upload Post Flow: . . . 7
- 8.2 User View Feed Flow: . . . 7
- 8.3 User Like Post Flow: . . . 7
- 8.4 User Comment Flow: . . . 7
- 8.5 User Follow / Unfollow Flow: . . . 7
- 8.6 View Profile Flow: . . . 7

9 Initial API Design 7
- 9.1 User API: . . . 7
- 9.2 Post API: . . . 8
- 9.3 Feed API: . . . 8
- 9.4 Like API: . . . 8
- 9.5 Follow API: . . . 8

10 Flow Diagrams 8
- 10.1 Upload Post Flow Diagram . . . 8
- 10.2 View Feed Flow Diagram . . . 9

11 Scalability Considerations 9

12 Technology Stack Selection 10

13 CDN Strategy **10**
13.1 What to store in CDN: 10
13.2 CDN Architecture: 10
13.3 Image/Video Optimization: 10
13.4 CDN Caching Rules: 10

14 Caching Strategy **11**
14.1 What to Cache: 11
14.2 Cache Invalidation Strategies: 11

15 Load Balancing **11**
15.1 Types: 11
15.2 Algorithms: 11
15.3 Layers of LB: 11
15.4 Health Check: 12

16 Microservices **12**
16.1 User Services: 12
16.2 Post Service: 12
16.3 Feed Service: 12
16.4 Like Service / Comment Service: 12
16.5 Follow Service: 12
16.6 Media Service: 12

17 Monitoring & Logging **13**
17.1 What to Monitor: 13
17.2 What to Log: 13

18 Trade-offs and Bottlenecks **13**
18.1 Trade-offs: 13
18.2 Bottlenecks: 13

19 Security Consideration **14**

20 Additional Considerations **14**

1 Introduction

Executive Summary

This system design document outlines the architecture for a social media platform that supports core features including user registration, content sharing, social connections, and engagement features. The system is designed to handle massive scale with 500 million daily active users while maintaining low latency and high availability.

2 Functional Requirements

The system must support the following core functionalities:

- **Signup / Login**: User registration and authentication system
- **Upload Photo / Video**: Ability for users to create and share media posts
- **Follow / Unfollow**: Functionality to establish and remove social connections
- **Like and Comment**: Core engagement features on posts
- **View Feed**: A personalized timeline displaying posts from followed users
- **View Their and Other Profile**: Ability to see user profiles and their posts
- **Edit / Delete Post**: Management of user-generated content

3 Non-Functional Requirements

Key Requirements

- **Low Latency for Feed**: Fast loading times for the user feed
- **Handle Millions of DAU**: The system must support high concurrency
- **Image / Video Storage**: Efficient and scalable storage for large media files
- **Scalable**: The architecture must be able to grow with user demand
- **CAP Theorem**: We choose AP (Availability & Partition Tolerance)

Why AP?

CAP Theorem Decision

It's better to show stale data than showing an error. This choice ensures system availability even during network partitions.

Where to apply?

- **Feed Generation**
- **Likes/comment count**

- **Follow/unfollow operations**

4 Assumptions

System Assumptions

- 500 million DAU (Daily Active Users)
- 200 million posts per day
- Each User follows 200 people on average
- Average Feed Refresh: 10 times per day
- Average like: 100 per post
- Average comment: 10 per post
- Average Image Size: 500 KB
- Average Video Size: 5 MB
- Image : Video ratio = 4:1
- Posts are stored for a lifetime
- Read : Write ratio = 100:1
- Data is stored in multiple data centres
- Peak Traffic: 2x average Traffic
- Storage: 5-year retention policy

5 Back-of-the-Envelope Calculation

5.1 Storage Calculation

5.1.1 Daily Storage:

- **Posts per day**: 200 million
- **Image (80%)**: 160 million * 500 KB = ~80 TB/day
- **Video (20%)**: 40 million * 5 MB = ~200 TB/day
- **Total Media Storage**: ~280 TB/day

5.1.2 Metadata Storage Calculation:

- **Post Metadata**: 200 million * 1KB = 200 GB
- **Like Metadata**: 200 million * (100 likes * 100 bytes) = ~2 TB
- **Comment Metadata**: 200 million * (10 comments * 500 bytes) = ~1 TB
- **Total Metadata**: ~3 TB/day
- **Total Per Day**: 280 TB + 3 TB = ~283 TB

5.1.3 5 Year Storage:

Total Storage Requirement

283 TB/day * 365 days * 5 years = ∼**516 PB (Petabytes)**

5.2 QPS (Query Per Second) Calculation

5.2.1 Read:

- 500 million DAU * 10 feeds/day = 5 billion reads/day
- $5 \times 10^9 / 86,400 \approx 58,000$ req/sec
- Peak: ∼116,000 req/sec

5.2.2 Write:

- 200 million posts/day
- $2 \times 10^8 / 86,400 \approx 2,300$ posts/sec
- Peak: ∼4,600 posts/sec

6 High Level Design Architecture (Overview)

Architecture Components

- **Client Layer**: Mobile Apps (React Native/Flutter), Web Apps (React/Node.js)
- **Load Balancer (LB)**: Distributes incoming traffic
- **API Gateway**: Single entry point for all clients
- **Microservices**: Decomposed services (User, Post, Feed, Like/Comment, Follow, Search)
- **Databases**:
  - SQL: For User Data (Profiles, auth)
  - NoSQL DB: For post, feed, like, comments
  - Graph DB: For Follow/unfollow relationships
- **Object Storage**: S3 Bucket/Cloud storage for media files
- **Cache Layer**: Redis for frequently accessed data (feeds, profiles, trending posts)
- **CDN (Content Delivery Network)**: For global delivery of static media
- **Messaging Queue**: Kafka/RabbitMQ for asynchronous processing

6.1 System Architecture Diagram

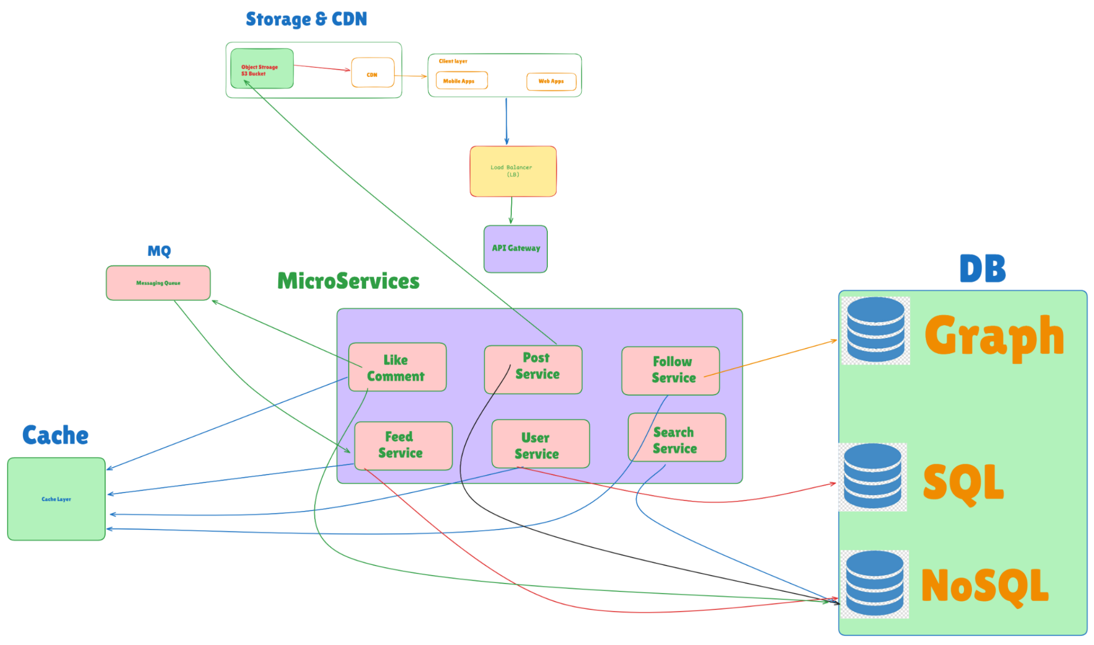


Figure 1: Overall System Architecture showing components and their relationships

7 Architecture Design (Component / Service-level detail)

7.1 Component Details:

- **Client Layer**: Handles user interaction
- **Load Balancer**: Routes requests to available servers
- **API Gateway**: Manages API routing, composition, and rate limiting

7.2 Microservices:

Service Architecture

- **User Service**: Manages user profiles and authentication
- **Post Service**: Handles creation, editing, and deletion of posts
- **Feed Service**: Generates and serves the user's timeline
- **Like/Comment Service**: Manages engagement metrics
- **Follow Service**: Handles social graph relationships
- **Search Service**: Enables content discovery

7.3 Databases:

- **SQL (PostgreSQL/MySQL)**: Stores structured user data

- **NoSQL (MongoDB/Cassandra)**: Stores semi-structured post and engagement data
- **Graph DB (Neo4j)**: Efficiently stores and traverses follower relationships
- **Cache (Redis/Memcached)**: Reduces latency for hot data
- **Object Storage (AWS S3)**: Stores media files durably
- **CDN (Cloudflare/AWS CloudFront)**: Caches media at edge locations
- **Message Queue (Apache Kafka)**: Decouples services for async tasks

8 Data Flow

Data Flow Patterns

8.1 User Upload Post Flow:

User → LB → API Gateway → Post Service → Object Storage (s3) & NoSQL DB (save metadata)

8.2 User View Feed Flow:

User → LB → API Gateway → Feed Service → Cache (if hit, return feed; if miss, fetch from Graph DB & NoSQL DB) → Cache it → Return feed

8.3 User Like Post Flow:

User → LB → API Gateway → Like Service → NoSQL DB (save like) → Message Queue (async increment count) → Cache (invalidate post cache)

8.4 User Comment Flow:

User → LB → API Gateway → Comment Service → NoSQL DB (save comment) → Message Queue (async increment count) → Cache (invalidate post cache)

8.5 User Follow / Unfollow Flow:

User → LB → API Gateway → Follow Service → Graph DB (update relationship) → Message Queue (trigger feed rebuild)

8.6 View Profile Flow:

User → LB → API Gateway → User Service → Cache (check profile) → If miss, SQL DB (user data) & NoSQL DB (user posts) → Cache (store profile) → Return profile

9 Initial API Design

9.1 User API:

```
POST /users/signup
POST /users/login
GET /users/{userId}
PUT /users/{userId}
```

Listing 1: User Management APIs

9.2 Post API:

```
POST /posts
GET /posts/{postId}
DELETE /posts/{postId}
PUT /posts/{postId}
```

Listing 2: Post Management APIs

9.3 Feed API:

```
GET /feed
```

Listing 3: Feed API

9.4 Like API:

```
POST /posts/{postId}/like
DELETE /posts/{postId}/like
```

Listing 4: Like Management APIs

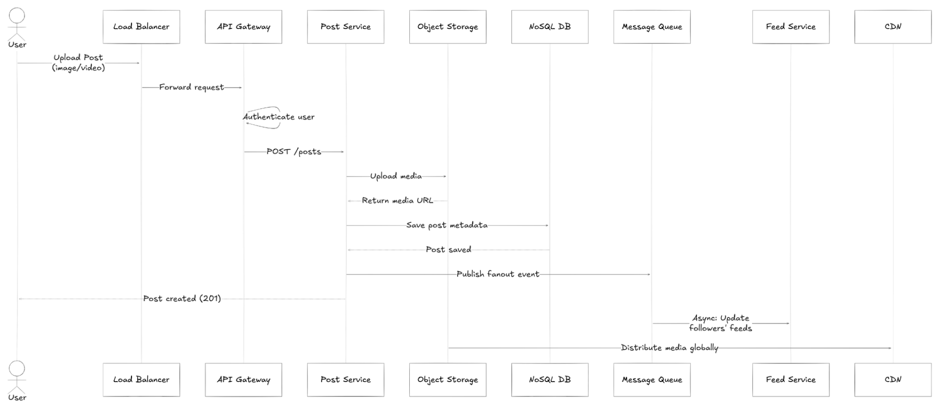
9.5 Follow API:

```
POST /users/{userId}/follow
DELETE /users/{userId}/unfollow
GET /users/{userId}/followers
GET /users/{userId}/following
```

Listing 5: Follow Management APIs

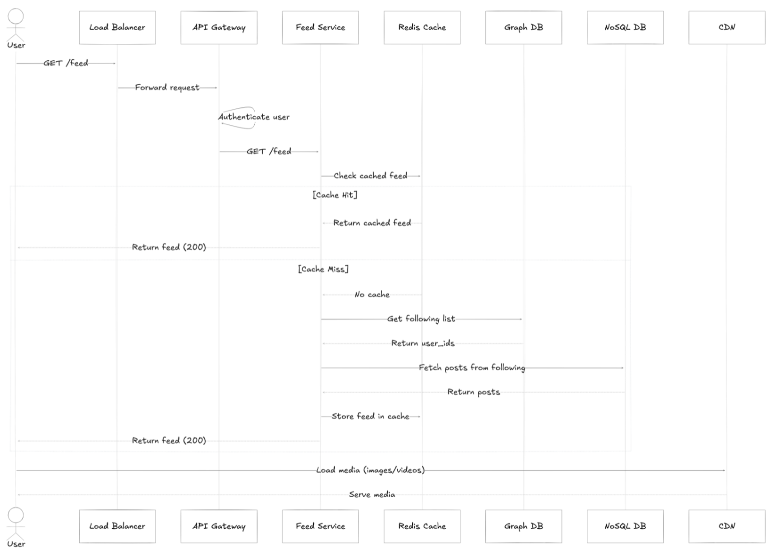
10 Flow Diagrams

10.1 Upload Post Flow Diagram



The diagram illustrates the sequence: User → Load Balancer → API Gateway → Post Service → (Object Storage & NoSQL DB) → Message Queue → CDN.

10.2 View Feed Flow Diagram



The diagram illustrates the sequence: User → Load Balancer → API Gateway → Feed Service → Cache? → (If Miss: Graph DB & NoSQL DB) → Return Feed.

11 Scalability Considerations

Scalability Strategy

- **Horizontal Scaling**: Auto-scaling application servers behind the load balancer
- **Database Scaling**:
  - SQL: Master-Slave replication
  - NoSQL: Sharding based on user_id or post_id
  - Graph DB: Distribute graph across multiple nodes
- **CDN for Media**: Distribute images/videos globally to reduce latency
- **Database Partitioning/Sharding**: Shard posts by user_id; keep replicated data across regions
- **Async Processing**: Use message queues for non-critical tasks (likes, comments)
- **Read Replicas**: Use multiple read replicas for heavy read operations; master handles only writes

12 Technology Stack Selection

Technology Choices

- **Client Side**: Mobile: React Native / Flutter; Web: React / Node.js
- **Load Balancer**: Nginx / HAProxy
- **API Gateway**: Kong / AWS API Gateway
- **Application Servers**: Node.js / Java Spring Boot
- **Databases**:
  - SQL: PostgreSQL / MySQL
  - NoSQL: MongoDB / Cassandra DB / Amazon DynamoDB
  - Graph: Neo4j
- **Cache**: Redis / Memcached
- **Object Storage**: AWS S3 / Google Cloud Storage
- **CDN**: Cloudflare / AWS CloudFront / Akamai
- **Message Queue**: Apache Kafka / Rabbit MQ
- **Cloud Provider**: AWS / Google Cloud

13 CDN Strategy

13.1 What to store in CDN:

- Images, Videos
- Static content (CSS, JS, fonts)
- Thumbnails

13.2 CDN Architecture:

User Request → CDN → Cache Hit: Serve from edge → Cache Miss: Fetch from origin → Cache → Serve to user

13.3 Image/Video Optimization:

- Multiple resolutions
- Compression
- Thumbnail generation
- Progressive download

13.4 CDN Caching Rules:

- **Images**: 30 days
- **Videos**: 30 days
- **Thumbnails**: 7 days

- **Static assets**: 1 year (versioned)

14 Caching Strategy

14.1 What to Cache:

- User profiles (popular)
- Popular posts
- Follower/following counts
- Post metadata (likes, comments count)

14.2 Cache Invalidation Strategies:

- Invalidate when user follows/unfollows someone
- Invalidate when a post is edited/deleted
- Update like/comment counts asynchronously
- Invalidate on profile updates

15 Load Balancing

15.1 Types:

- By IP/port
- By URL/Headers

15.2 Algorithms:

- **Round Robin**: Equal distribution
- **Weighted**: More traffic to powerful servers
- **IP Hash**: Same user → Same server

15.3 Layers of LB:

- **Global LB**: Route to nearest data center
- **Regional LB**: Across availability zones
- **Service LB**: Between different microservices

15.4 Health Check:

Health Monitoring

Regularly check servers; remove unhealthy ones; auto-add when recovered

16 Microservices

16.1 User Services:

User Service

User signup/login, profile management, authorization/authentication. **Database: SQL**

16.2 Post Service:

Post Service

Create/edit/delete post, upload data to S3, get post details. **Database: NoSQL**

16.3 Feed Service:

Feed Service

Generate user feed, fetch posts from followed users. **Database: NoSQL + Cache**

16.4 Like Service / Comment Service:

Engagement Services

Add/remove likes, comments; get comment and number of likes. **Database: NoSQL**

16.5 Follow Service:

Follow Service

Follow/unfollow users; get follower/following list. **Database: Graph DB**

16.6 Media Service:

Media Service

Handles images/videos; compress media; store in S3 bucket

17 Monitoring & Logging

17.1 What to Monitor:

- API speed (latency)
- Number of requests
- Error count
- CPU/memory/disk usage
- Cache hit vs miss

17.2 What to Log:

- API calls
- Errors
- Slow DB queries
- System warnings

18 Trade-offs and Bottlenecks

18.1 Trade-offs:

Architecture Trade-offs

- **Consistency vs Availability**: Chose availability first; tolerate stale data to keep the system up
- **SQL vs NoSQL vs Graph**: Different DBs for different needs; more complexity for better performance
- **Sync vs Async Processing**: Some actions run in background with a small delay for better speed and scaling

18.2 Bottlenecks:

Potential Bottlenecks

- **Feed Generation Bottleneck**: Hard for users with many followers. Solution: Use pre-computed feeds, caching, and limit feed size
- **Hot Users (Celebrities)**: Too many followers. Solution: Use a pull model for them
- **Storage Growth**: Huge media storage over years. Solution: Compress old files and move to cheaper storage
- **Single Point of Failure**: One service going down breaks the system. Solution: Use replicas and failover mechanisms

19 Security Consideration

Security Measures

- **Authentication & Access Control**: Login tokens, Role-based access, Token expiry
- **API Security**: Use HTTPS, Limit API calls (Rate Limiting), Validate inputs
- **Data Security**: Encrypt passwords, Protect sensitive user info, Keep data safe during transfer
- **Media Security**: Allow only safe file types, Scan uploads, Limit file size
- **DDoS Protection**: Limit requests per user/IP, Use firewall
- **Database Security**: Allow only trusted IPs, Use separate access for reading/writing, Regular backups

20 Additional Considerations

Additional Important Aspects

- **Idempotency**: Avoid duplicate actions (e.g., double likes/follows); ensure the same result even if a request repeats
- **Rate Limiting**: Limit how many requests a user/IP can send to protect the system from overload
- **Data Retention**: Keep new data in fast storage; move old data to cheap storage; delete very old logs; compress old files
- **Disaster Recovery**: Keep backups; replicate data across regions; have a plan to restore quickly if something fails
- **Content Moderation**: Auto-detect bad content; allow user reports; have a manual review process when needed