Certificate no: UC-12180dd8-e4e9-4b6f-aeba-25bfedf15adc
Certificate url: ude.my/UC-12180dd8-e4e9-4b6f-aeba-25bfedf15adc
Reference Number: 0004

CERTIFICATE OF COMPLETION

System Design Masterclass (2025)

Instructors **Sweet Codey, Suresh Gandhi | SDE-II @Microsoft, Ex-Amazon, Shubham Chandak | Bloomberg Engineering, Rohit Jain | SDE-2 @ Amazon**

Pavan B N

Date **April 17, 2025**
Length **9.5 total hours**

Social Sphere

This project aims to design a scalable **social network system** supporting **posts** (text, images, videos), **real-time messaging**, and **notifications**. The system will be built using a **microservices architecture** for modularity and independent scaling. It will use **NoSQL databases** for user content storage, ensuring fast access and scalability. **Horizontal scaling** will be implemented to handle growing traffic, and a **Content Delivery Network (CDN)** will be used to efficiently deliver media content.

For **real-time messaging**, the system will leverage **WebSockets** for instant communication, while **event-driven notification services** will handle email, SMS, and push notifications. The design prioritizes **high availability**, **fault tolerance**, and **performance** through **caching** and distributed systems. It also considers trade-offs between consistency and speed to ensure a responsive, user-friendly experience under heavy load.

This system design follows a structured 5-framework approach:

- ➢ **Designing Requirements:** Define functional (post text, images, videos, news feed, messaging, notifications) and non-functional (availability, scalability, latency) requirements.
- ➢ **Capacity Estimation:** Estimate storage, network throughput, and cache requirements based on DAU (500M) and MAU (2B) for efficient resource allocation.
- ➢ **API Design:** Design RESTful APIs for posting, reading feeds, messaging, and sending notifications, ensuring scalability and proper versioning.
- ➢ **High-Level Design:** Architect the system with components like load balancers, WebSockets, CDN, media storage, database clusters, and caching for scalability and reliability.
- ➢ **Deep Dive:** Select a NoSQL database for scalable data storage, model data for posts, users, and messages, and ensure efficient access and consistency.

Designing Requirements

Functional Requirements

- Users can create posts with text, images, or videos.
- Large video uploads allowed (up to 500MB).
- Infinite scrolling News Feed optimized and personalized.
- Real-time messaging between users using WebSocket's.
- Notifications delivered via:
  - Web Push Notifications (browser notifications)
  - Mobile App Push Notifications (Android/iOS)
  - SMS Alerts
  - Email Notifications

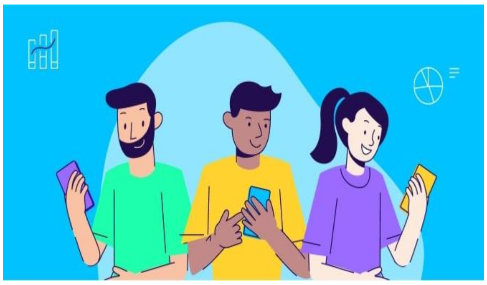
Non-Functional Requirements

- **High Availability:** 99.99% uptime (multi-region).
- **Scalability:** Must handle traffic spikes (festivals, global events).
- **Low Latency:** Feed response time < 300 milliseconds, messaging < 100 milliseconds.
- **Consistency:** Eventual consistency for feeds; strong consistency for messages.
- **Durability:** No data loss even on hardware failures.
- **Security:** End-to-end encryption, authentication, authorization, data encryption at rest and in transit.

Capacity Estimation

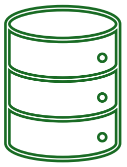
Assumptions

- **Daily Active Users (DAU):** 500 million
- **Monthly Active Users (MAU):** 2 billion
- Average posts per user per day: 2
- Average message exchanges per user per day: 20
- Average notifications per user per day: 5



Post Storage Estimation

- Text post average size = 1 Kilobyte
- Image average size = 2 Megabytes
- Video average size = 20 Megabytes



Storage

Out of all posts:

- 50% Text posts
- 30% Image posts
- 20% Video posts

Thus, for 1 Billion posts/day:

- **Text:** 50% × 1B × 1KB = 500 Million KB ≈ 500 Gigabytes
- **Images:** 30% × 1B × 2MB = 600 Million MB ≈ 600 Terabytes
- **Videos:** 20% × 1B × 20MB = 4 Billion MB ≈ 4 Petabytes

Total Storage/day ≈ 4.6 Petabytes

Messaging Storage Estimation

- Each message size = ~2 Kilobytes
- 10 Billion messages/day →
  10B × 2KB = 20 Terabytes/day

Notification Storage Estimation

- Lightweight (~0.5KB per notification)
- 2.5 Billion notifications/day →
  2.5B × 0.5KB = 1.25 Terabytes/day

Network Bandwidth Estimation

- Uploads (post photos/videos): heavy on media
- Downloads (feed, messaging): high concurrency, cached heavily

We need **hundreds of Gigabits/second** sustained bandwidth minimum.

Compute Estimation

- 100,000 Requests Per Second (peak)
- 5,000+ compute instances (depending on capacity planning)

API Design

Upload Media (Pre-Signed URL)

POST /v1/media/upload

Request Body

```
{
  "user_id": "12345",
  "timestamp": "2025-04-17T12:34:56Z",
  "media_type": "video",
  "file_name": "vacation.mov"
}
```

Response Body

```
{
  "pre_signed_url": "https://s3.amazonaws.com/socialsphere/uploads/vacation.mov?...",
  "post_id": "post_987654321"
}
```

Pre-Signed URL: A secure, time-limited link that lets clients upload files directly to Object Storage without needing to pass through backend servers

Create Post

POST /v1/posts

Request Body

```
{
  "user_id": "12345",
  "text": "Had a great trip!",
  "media_urls": [
    "https://cdn.socialsphere.com/uploads/vacation.mov"
  ],
  "timestamp": "2025-04-17T12:36:00Z"
}
```

Response Body

```
{
  "post_id": "post_987654321",
  "status": "created"
}
```

Get Feed

GET /v1/feed?user_id=12345

Response Body

```
{
  "feed": [
    {
      "post_id": "post_987654321",
      "user_id": "54321",
      "text": "Hello world!",
      "media_urls": [],
      "timestamp": "2025-04-17T11:00:00Z"
    },
    ...
  ],
  "next_page_token": "abc"
}
```

Send Message

POST /v1/messages

Request Body

```
{
  "from_user_id": "12345",
  "to_user_id": "67890",
  "text": "VGhpcyBpcyBhbiBlbmNyeXB0ZWQgbWVzc2FnZQ==",
  "timestamp": "2025-04-17T12:45:00Z"
}
```

- "text" is encrypted using AES-256-GCM.
- Encryption protects the core message while keeping system design **fast**, **secure**, and **scalable**.

Response Body

```
{
  "message_id": "msg_987654321",
  "status": "sent"
}
```

Notifications

GET /v1/notifications

Request Body

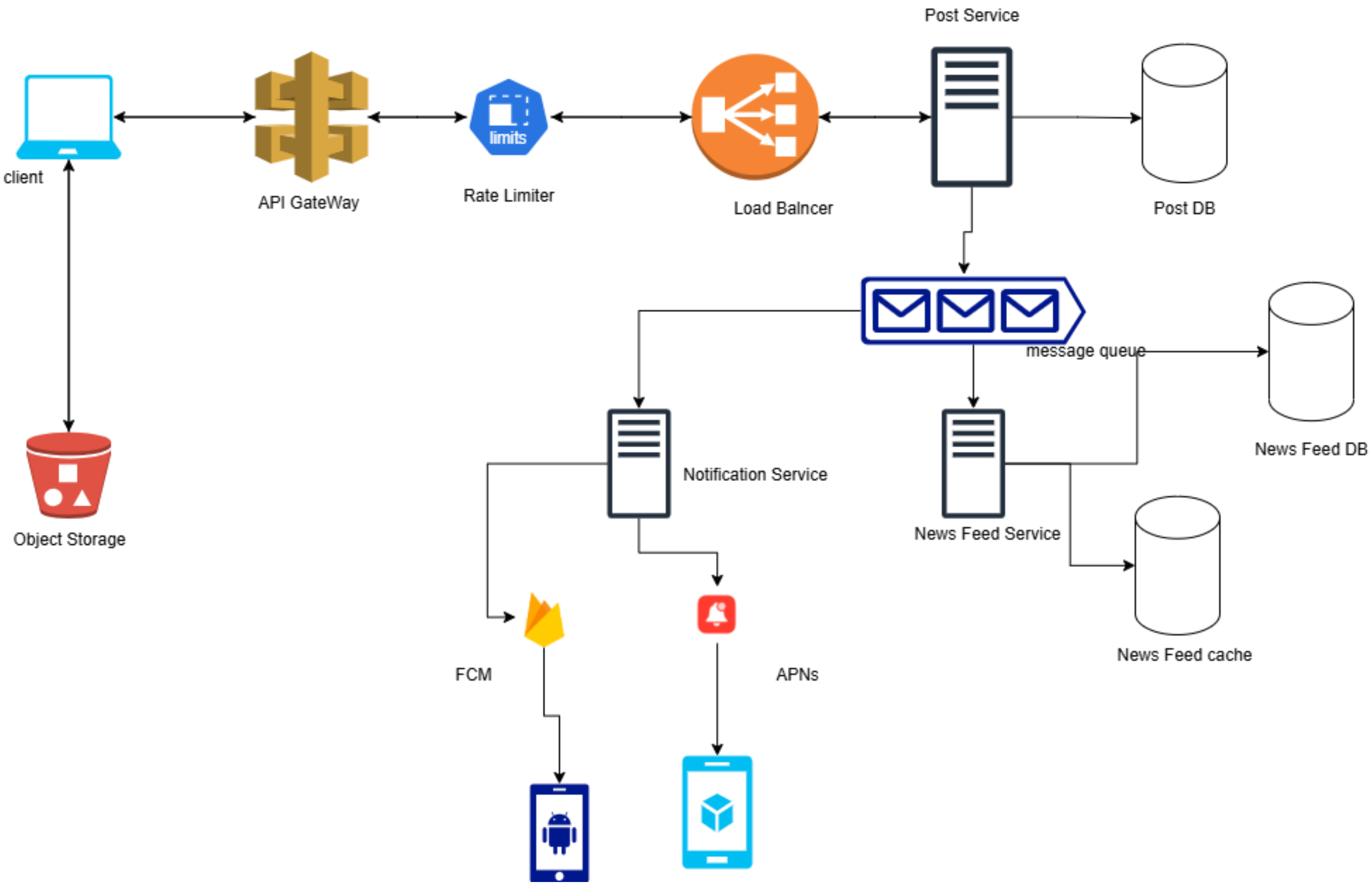
```
{
  "user_id": "12345",
  "notification_type": "push",
  "platform": "android",
  "limit": 20
}
```

Response Body

```
{
  "notifications": [
    {
      "notification_id": "notif_123",
      "title": "New friend request",
      "body": "John Doe sent you a friend request",
      "timestamp": "2025-04-17T11:30:00Z",
      "channel": "push"
    },
    ...
  ]
}
```

High Level Design

POST Image



1. User Devices

Users access the application through web browsers, Android apps, or iOS apps.

2. API Gateway

All incoming requests (such as posting content, sending messages, or fetching feeds) pass through the API Gateway. It manages routing, authentication, authorization, and rate limiting.

3. Authentication and Rate Limits

An Identity and Access Management (IAM) service verifies user identities and enforces usage limits to prevent abuse.

4. Load Balancer

The Load Balancer distributes incoming traffic evenly across multiple backend servers to ensure high availability and prevent server overload.

5. Backend Microservices

- **Feed Service**: Retrieves and displays the personalized news feed for users.
- **Post Service**: Handles creating, storing, and processing posts (text, images, videos).
- **Messaging Service**: Manages real-time communication between users via WebSockets.
- **Notification Service**: Sends notifications through app push notifications, email, and SMS.

6. Object Storage (S3/GCP Storage)

Media files such as images and videos are uploaded directly to object storage (e.g., AWS S3) using Pre-Signed URLs for secure, direct uploads without burdening application servers.

7. Message Queues (SQS/Kafka)

Asynchronous tasks (especially notifications) are handled using message queues to ensure reliable delivery and system decoupling.

8. Databases (NoSQL)

Each service uses a separate NoSQL database (e.g., DynamoDB, Cassandra) for scalability, flexible schema design, sharding, and high-performance read/write operations.

9. Notification Delivery

The Notification Service reads from message queues and sends alerts to users via Firebase Cloud Messaging (FCM) or Apple Push Notification Service (APNS).

10. Client Interaction

Mobile and web clients receive feeds, real-time messages, and notifications promptly, ensuring a seamless user experience.

Deep Dive into the System Design

Database Selection

Primary Choice: **NoSQL Databases** (e.g., Amazon DynamoDB, Apache Cassandra, or MongoDB).

Reasons:

- Ability to horizontally scale across massive data volumes.
- Flexible schema for user-generated content (posts, comments, messages).
- High write and read throughput for millions of active users.
- Native support for data sharding, replication, and eventual consistency models.

Graph Database (e.g., Neo4j) for friend relationships, social graphs, and feed generation optimization.

Data Modeling (NoSQL, JSON Examples)

Posts Collection

```
{
  "post_id": "post_987654321",
  "user_id": "12345",
  "text": "Had a great trip!",
  "media_urls": ["https://cdn.socialsphere.com/uploads/vacation.mov"],
  "timestamp": "2025-04-17T12:36:00Z",
  "likes": 0,
  "comments_count": 0  }
```

Messages Collection

```
{
  "message_id": "msg_987654321",
  "from_user_id": "12345",
  "to_user_id": "67890",
  "text": "Hey!",
  "timestamp": "2025-04-17T12:45:00Z",
  "status": "delivered"
}
```

Media Storage and Uploads

Service: Object Storage (e.g., **Amazon S3**)

Flow:

- Client requests a **Pre-Signed URL** from backend.
- Backend generates a temporary secure upload link tied to **user_id, post_id, timestamp**.
- Client uploads media directly to Object Storage via the Pre-Signed URL, bypassing backend servers to reduce load and improve performance.

Advantages:

- Reduces API server bottlenecks.
- Secure, time-bound uploads.
- Scalable media handling.

Media Processing and Streaming

Video Processing Pipelines:

- Triggered after successful upload.
- Transcodes videos into multiple qualities (e.g., 1080p, 720p, 480p) using **media processing services** (e.g., AWS Elastic Transcoder, AWS MediaConvert).
- Adaptive bitrate streaming using **HLS (HTTP Live Streaming)** format for smooth video playback across network conditions.

Delivery:

- Optimized delivery through **CDN** (e.g., CloudFront, Akamai) for minimal latency worldwide.

Notification Services

Event-Driven Architecture:

- Notifications (push, email, SMS) triggered by user actions (like comment, like, message).

Push Notifications:

- Mobile Push: **Firebase Cloud Messaging (FCM)** for Android, **Apple Push Notification Service (APNS)** for iOS.
- Web Push: **Service Workers** to send browser notifications.

Email/SMS:

- Email through providers like **SendGrid** or **Amazon SES**.
- SMS via services like **Twilio**.

Queue Usage:

- Use of **Kafka/SQS** for queuing notification events and ensuring high throughput and retry mechanisms.

Messaging System

Real-Time Chat:

- **WebSocket servers** handle persistent, bi-directional communication.
- Load-balanced across multiple WebSocket instances.

Message Storage:

- Messages stored in NoSQL database optimized for write-heavy operations.

Security:

- **End-to-End Encryption** planned using client-side encryption libraries for message payloads.

Security and Rate Limiting

Data Security:

- Encryption at rest (S3, NoSQL DBs) and encryption in transit (HTTPS, WSS).

Rate Limiting:

- Token bucket or leaky bucket algorithms to prevent abuse at API Gateway.

Authentication:

- OAuth 2.0 / OpenID Connect for user authentication and secure API access.

Caching and Search

Caching:

- Frequently accessed feeds, profile data cached using **Redis** or **Memcached**.

Full-Text Search:

- Search functionality powered by **Elasticsearch** (e.g., user search, hashtag search, content search).

Scalability Strategies

Horizontal Scaling:

- New server instances automatically added during traffic spikes.

Database Sharding:

- Partitioning user data across multiple nodes based on user_id.

Data Replication:

- Replica sets for high availability and fault tolerance.