Experiment 6

Student Name: Umang Kaushik **UID: 23BCS10712**
Branch: BE CSE **Section/Group: KRG-1-B**
Semester: 6th **Date of Performance:28th Feb**
Subject Name: SD **Subject Code: 23CSH-314**

1. **Aim:** To design a highly scalable video streaming platform similar to YouTube that supports heavy video uploading, adaptive bitrate streaming, and real-time social interactions with high availability and low latency.

2. **Objectives:**

- Design a distributed, fault-tolerant video streaming platform capable of handling massive scale.
- Support heavy, continuous video file ingestion without bottlenecking core services.
- Enable smooth video playback across diverse network conditions using Adaptive Bitrate Streaming (ABR).
- Facilitate low-latency, real-time social interactions (comments, likes) among concurrent viewers.

3. **Procedure:**

- Ingestion: The client requests a pre-signed URL from the Upload Service and uploads raw video chunks directly to Object Storage (e.g., S3).
- Asynchronous Trigger: Upon successful upload, an event is pushed to a message broker (Kafka) to decouple ingestion from processing.

- Transcoding: Worker nodes consume the event, fetch the raw video, and transcode it into multiple resolutions (1080p, 720p, etc.) and segments.

- Delivery: Transcoded segments are stored back in Object Storage and distributed globally via a Content Delivery Network (CDN) for fast, localized access.

- Social Interaction: Clients establish persistent WebSocket connections with the Social Service, which uses a Pub/Sub mechanism (e.g., Redis) and a high-write database (e.g., Cassandra) to broadcast and store live comments.

4. Functional Requirements:

- Video Upload: Users can reliably upload massive video files.

- Video Playback: Users can stream videos smoothly.

- Adaptive Bitrate Streaming (ABR): The system must automatically adjust video quality based on the user's real-time internet bandwidth.

- Social Engagement: Users must be able to post and view real-time comments and interactions.

- Metadata Management: The system must store and retrieve video titles, descriptions, and user profiles.

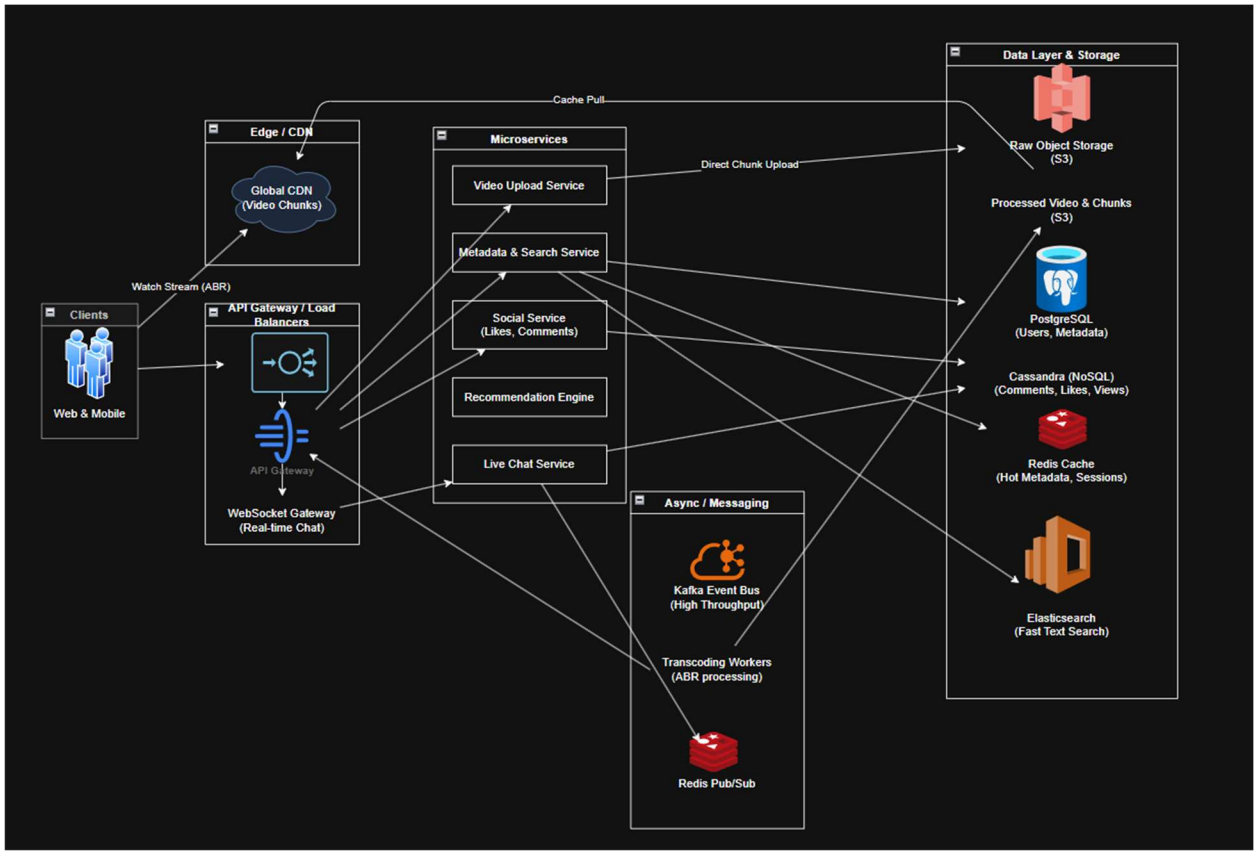
5. Non-Functional Requirements:

- High Scalability: Must seamlessly handle spikes in concurrent uploads and millions of simultaneous streams.
- High Availability: Streaming and core platform features should target 99.99% uptime.
- Low Latency: Minimal buffering time (Time-To-First-Byte) for video playback and near-instant delivery of social interactions.
- Reliability & Durability: Uploaded videos and user data must never be lost, requiring redundant storage strategies.

6. Outcome:

A highly decoupled, event-driven High-Level Design (HLD) that utilizes cloud-native patterns (presigned URLs, distributed queues, CDN caching, and WebSockets). This architecture ensures the platform remains responsive during heavy load, providing a seamless viewing and interactive experience for global users.

7. Required System Design:


Data Layer & Storage
Cache Pull
Edge / CDN
Global CDN
(Video Chunks)
Microservices
Direct Chunk Upload
Video Upload Service
Raw Object Storage
(S3)
Processed Video & Chunks
(S3)
Metadata & Search Service
Watch Stream (ABR)
Clients
API Gateway / Load Balancers
Social Service
(Likes, Comments)
PostgreSQL
(Users, Metadata)
Cassandra (NoSQL)
(Comments, Likes, Views)
Web & Mobile
Recommendation Engine
Redis Cache
(Hot Metadata, Sessions)
API Gateway
Live Chat Service
Async / Messaging
WebSocket Gateway
(Real-time Chat)
Kafka Event Bus
(High Throughput)
Elasticsearch
(Fast Text Search)
Transcoding Workers
(ABR processing)
Redis Pub/Sub