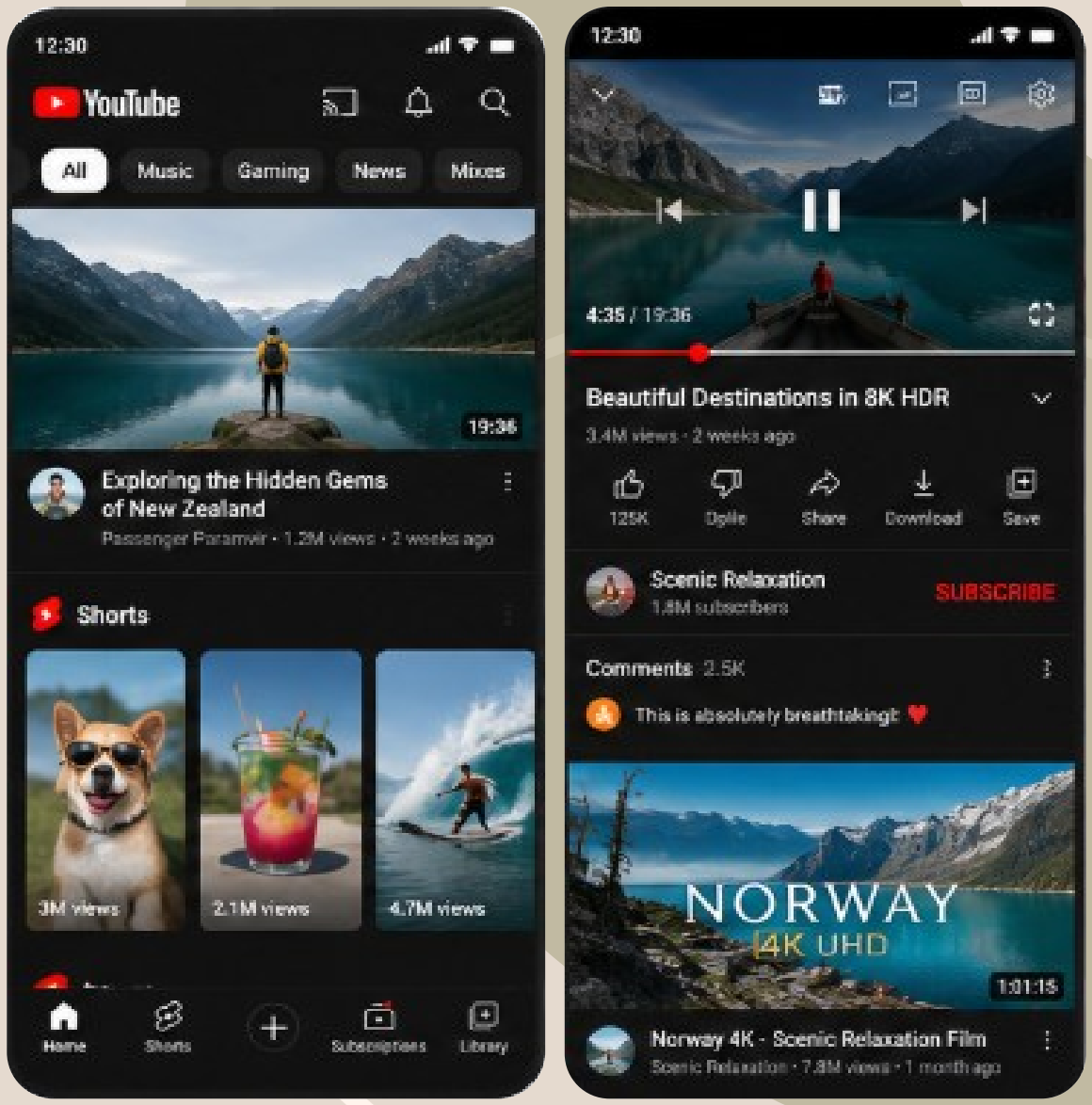
YouTube System Design

Let's Make YouTube Seamless

Outline

- About YouTube
- Problem Statement
- Functional Requirements
- Non-Functional Requirements
- System Assumptions
- User Growth Assumptions (MAU & DAU) QPS
- Estimation Storage
- Estimation 5-Year Growth Projection
- Entity Relationship Diagram (ERD)
- High-Level Architecture
- API Design
- Video Streaming Workflow
- Recommendation System



About YouTube

- YouTube is the world's largest video-sharing and streaming platform, allowing users to upload, watch, and share videos globally.
- It serves over 2.7 billion monthly active users and processes millions of video uploads and billions of daily views.
- The platform delivers personalized recommendations, real-time video streaming, and creator monetization at massive global scale.

Functional Requirements

- The system should allow users to create an account and log in securely.
- The system should allow creators to upload, update, and delete videos.
- The system should allow users to stream videos in real time from anywhere in the world.
- The system should support adaptive video quality based on the user's network speed.
- The system should allow users to search for videos, channels, and playlists using keywords.
- The system should provide personalized video recommendations based on user activity and interests.
- The system should allow users to like, dislike, comment on, and share videos.
- The system should allow users to subscribe to channels and receive notifications for new uploads.
- The system should maintain users' watch history and allow them to continue watching from where they left off.
- The system should allow users to create, update, and manage playlists.

Non Functional Requirements

- The system should support **800 million Monthly Active Users (MAU)** and **500 million Daily Active Users (DAU)**.
- The system should provide high availability with a target uptime of **99.99%**.
- The system should start video playback within **300 ms** under normal network conditions.
- Search results should be returned in **less than 100 ms**.
- The platform should support millions of concurrent video streams with low latency.
- The system should be horizontally scalable to handle future user and traffic growth.
- The system should ensure data durability by replicating video content and metadata across multiple regions.
- The system should secure user data using authentication, authorization, and encryption.
- The system should tolerate service failures without affecting the overall user experience.
- The system should optimize infrastructure costs by using caching, CDNs, and object storage for frequently accessed content.

Assumptions

Category	Assumption
Users	800 Million Monthly Active Users (MAU), 500 Million Daily Active Users (DAU), 40 Million Creators, 5 Million Active Creators uploading daily
Videos	3 Billion total videos, 15 Million new videos uploaded per day, Average video duration: 10 minutes, Average encoded video size: 150 MB (including multiple streaming resolutions)
User Activity	Each active user watches 10 videos/day, performs 3 searches/day, receives 5 recommendations/day, likes 2 videos/day, comments on 0.5 videos/day, subscribes to 0.05 channels/day, creates 0.02 playlists/day
Streaming	Videos are streamed using adaptive bitrate streaming with an average playback quality of 720p
Storage	Video content is retained for 5 years, while metadata, comments, likes, and watch history are retained indefinitely for analytics and recommendations
Growth	Users, creators, videos, and daily uploads are expected to grow by 10% annually

Traffic Estimation

Write QPS

Operation	Calculation	QPS
User Profile Updates	5% of DAU = 25M/day → 25M / 86,400	289
Video Uploads	15M uploads/day → 15M / 86,400	174
Playlist Create/Update	0.02 × 500M = 10M/day → 10M / 86,400	116
Likes	2 × 500M = 1B/day → 1B / 86,400	11,574
Comments	0.5 × 500M = 250M/day → 250M / 86,400	2,894
Channel Subscriptions	0.05 × 500M = 25M/day → 25M / 86,400	289
Watch History Logging	10 × 500M = 5B/day → 5B / 86,400	57,870
Video View Events	10 × 500M = 5B/day → 5B / 86,400	57,870
Recommendation Feedback (clicks, impressions)	5 × 500M = 2.5B/day → 2.5B / 86,400	28,935

Read QPS

Operation	Calculation	QPS
Video Streaming Requests	10 × 500M = 5B/day → 5B / 86,400	57,870
Search Requests	3 × 500M = 1.5B/day → 1.5B / 86,400	17,361
Recommendations	5 × 500M = 2.5B/day → 2.5B / 86,400	28,935
Video Metadata Fetch	10 × 500M = 5B/day → 5B / 86,400	57,870
Comments Fetch	10 × 500M = 5B/day → 5B / 86,400	57,870
Channel/Profile Fetch	2 × 500M = 1B/day → 1B / 86,400	11,574
Playlist Fetch	1 × 500M = 500M/day → 500M / 86,400	5,787
Watch History Fetch	1 × 500M = 500M/day → 500M / 86,400	5,787

Traffic Type	Average QPS	Peak QPS
Read	242,054	484,108
Write	159,011	318,022
Total	401,065	802,130

Data Base Schema – Storage Estimation

Entity	Estimated Row Size
User	734 B
Channel	879 B
Video Metadata	1,511 B
Playlist	644 B
Comment	532 B
Like	25 B
Subscription	24 B
Watch History	36 B
Notification	330 B

Entity	Row Size	Records	Storage
Users	734 B	800 M	587.2 GB
Channels	879 B	40 M	35.2 GB
Video Metadata	1,511 B	3 B	4.53 TB
Playlists	644 B	1 B	644 GB
Comments	532 B	30 B	15.96 TB
Likes	25 B	15 B	375 GB
Subscriptions	24 B	8 B	192 GB
Watch History (1 Year)	36 B	1.825 T	65.7 TB
Notifications (1 Year)	330 B	18.25 B	6.02 TB

Blob Storage for Videos

Total Video Storage

$3B \times 150MB = 450{,}000{,}000{,}000MB \approx$ **450 PB**

If we store **3 replicas** for durability:

$$450PB \times 3 = 1.35EB$$

Effective Video Storage Required = 1.35 Exabytes

Final Storage Summary

Category	Storage
Metadata Databases	≈ 94 TB
Video Files (S3/Object Storage)	≈ 450 PB
Replicated Video Storage (3×)	≈ 1.35 EB

10% Yearly Growth

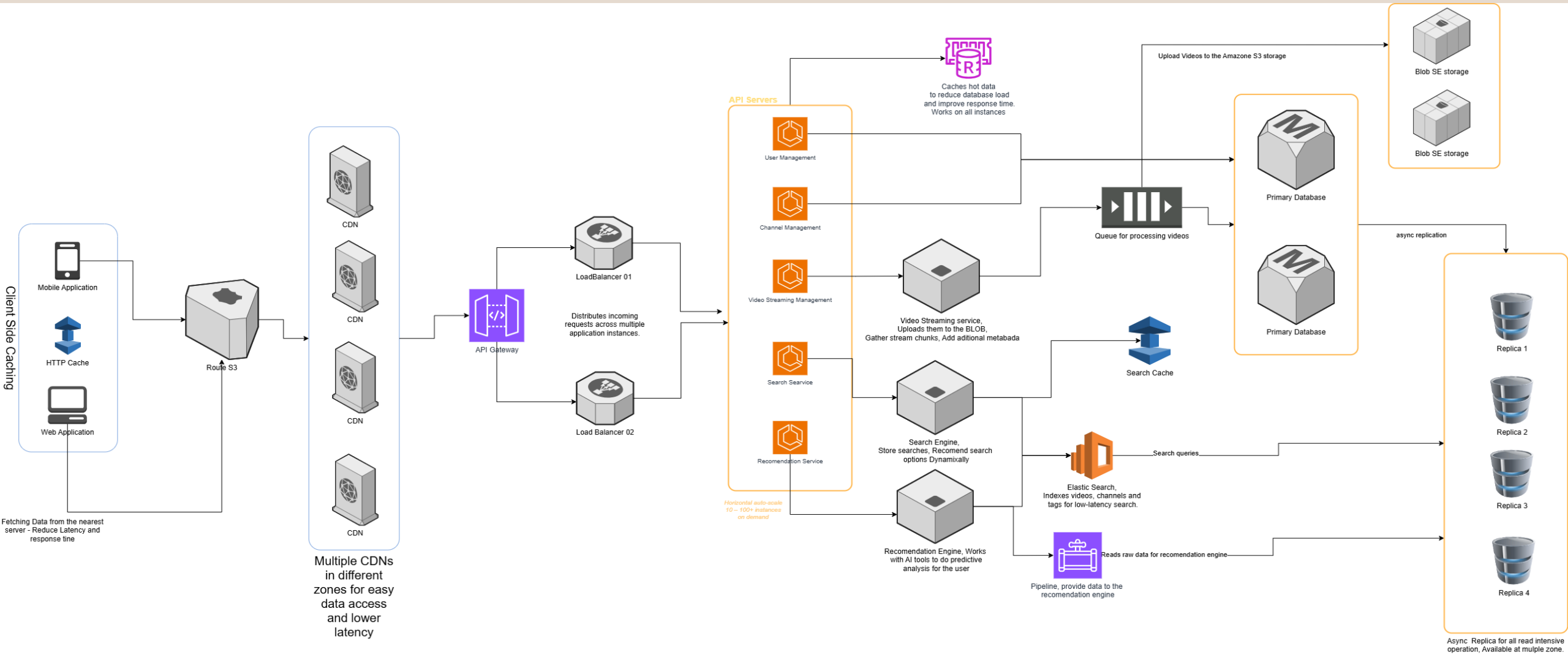
Year	Metadata	Videos	Total Storage
Year 0	94 TB	450 PB	≈ 450.09 PB
Year 1	103.4 TB	495 PB	≈ 495.10 PB
Year 2	113.7 TB	544.5 PB	≈ 544.61 PB
Year 3	125.1 TB	598.9 PB	≈ 599.03 PB
Year 4	137.6 TB	658.8 PB	≈ 658.94 PB
Year 5	151.4 TB	724.7 PB	≈ 724.85 PB

Year	MAU	DAU	Creators	Videos	Daily Uploads
Year 0	800 M	500 M	40 M	3.00 B	15.0 M
Year 1	880 M	550 M	44 M	3.30 B	16.5 M
Year 2	968 M	605 M	48.4 M	3.63 B	18.2 M
Year 3	1.06 B	665 M	53.2 M	3.99 B	20.0 M
Year 4	1.17 B	732 M	58.6 M	4.39 B	22.0 M
Year 5	1.29 B	805 M	64.4 M	4.83 B	24.2 M

Year	Average Read QPS	Peak Read QPS (×2)
Year 0	242,054	484,108
Year 1	266,259	532,518
Year 2	292,885	585,770
Year 3	322,174	644,348
Year 4	354,391	708,782
Year 5	389,830	779,660

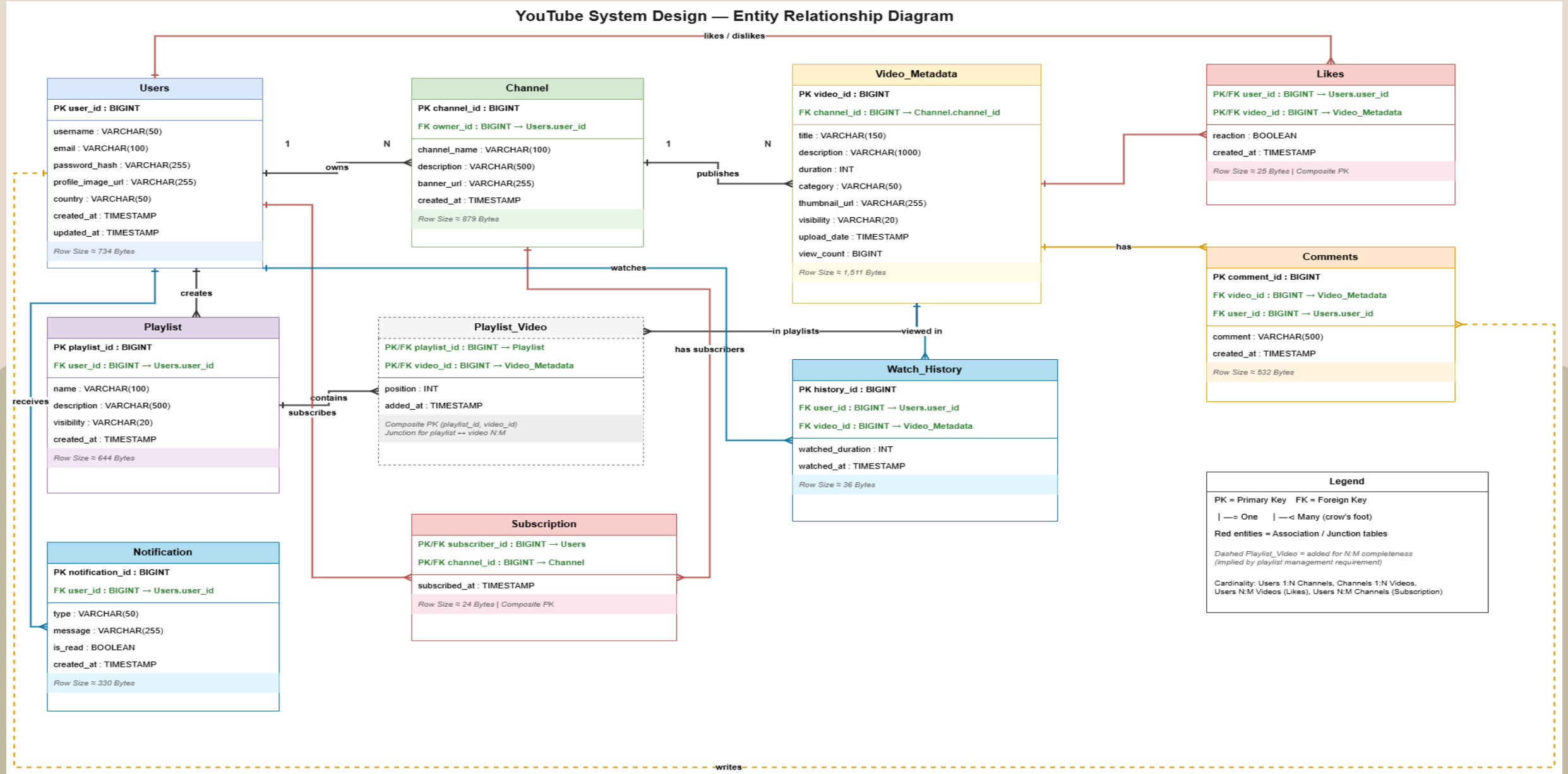
Year	Average Write QPS	Peak Write QPS (×2)
Year 0	159,011	318,022
Year 1	174,912	349,824
Year 2	192,403	384,806
Year 3	211,643	423,286
Year 4	232,807	465,614
Year 5	256,088	512,176

High Level Architecture Diagram



L→R Flow: User → Regional CDN Edges (CloudFront) → ALB Pool (10x) → API Fleet (ECS, 10–100+) → Streaming / Search / Recommendation → Redis (80% reads) + OpenSearch + Kafka → Multi-Master RDS (user_id hash) → Shards → Regional Read Replicas → S3 Blob Storage

ERD Diagram


YouTube System Design — Entity Relationship Diagram
likes / dislikes
Users
PK user_id : BIGINT
username : VARCHAR(50)
email : VARCHAR(100)
password_hash : VARCHAR(255)
profile_image_url : VARCHAR(255)
country : VARCHAR(50)
created_at : TIMESTAMP
updated_at : TIMESTAMP
Row Size ≈ 734 Bytes
1
N
owns
Channel
PK channel_id : BIGINT
FK owner_id : BIGINT → Users.user_id
channel_name : VARCHAR(100)
description : VARCHAR(500)
banner_url : VARCHAR(255)
created_at : TIMESTAMP
Row Size ≈ 879 Bytes
1
N
publishes
Video_Metadata
PK video_id : BIGINT
FK channel_id : BIGINT → Channel.channel_id
title : VARCHAR(150)
description : VARCHAR(1000)
duration : INT
category : VARCHAR(50)
thumbnail_url : VARCHAR(255)
visibility : VARCHAR(20)
upload_date : TIMESTAMP
view_count : BIGINT
Row Size ≈ 1,511 Bytes
Likes
PK/FK user_id : BIGINT → Users.user_id
PK/FK video_id : BIGINT → Video_Metadata
reaction : BOOLEAN
created_at : TIMESTAMP
Row Size ≈ 25 Bytes | Composite PK
has
Comments
PK comment_id : BIGINT
FK video_id : BIGINT → Video_Metadata
FK user_id : BIGINT → Users.user_id
comment : VARCHAR(500)
created_at : TIMESTAMP
Row Size ≈ 532 Bytes
watches
creates
Playlist
PK playlist_id : BIGINT
FK user_id : BIGINT → Users.user_id
name : VARCHAR(100)
description : VARCHAR(500)
visibility : VARCHAR(20)
created_at : TIMESTAMP
Row Size ≈ 644 Bytes
receives
contains
subscribes
Playlist_Video
PK/FK playlist_id : BIGINT → Playlist
PK/FK video_id : BIGINT → Video_Metadata
position : INT
added_at : TIMESTAMP
Composite PK (playlist_id, video_id)
Junction for playlist ↔ video N:M
in playlists
has subscribers
viewed in
Watch_History
PK history_id : BIGINT
FK user_id : BIGINT → Users.user_id
FK video_id : BIGINT → Video_Metadata
watched_duration : INT
watched_at : TIMESTAMP
Row Size ≈ 36 Bytes
Subscription
PK/FK subscriber_id : BIGINT → Users
PK/FK channel_id : BIGINT → Channel
subscribed_at : TIMESTAMP
Row Size ≈ 24 Bytes | Composite PK
Notification
PK notification_id : BIGINT
FK user_id : BIGINT → Users.user_id
type : VARCHAR(50)
message : VARCHAR(255)
is_read : BOOLEAN
created_at : TIMESTAMP
Row Size ≈ 330 Bytes
Legend
PK = Primary Key FK = Foreign Key
| —= One | —< Many (crow's foot)
Red entities = Association / Junction tables
Dashed Playlist_Video = added for N:M completeness
(implied by playlist management requirement)
Cardinality: Users 1:N Channels, Channels 1:N Videos,
Users N:M Videos (Likes), Users N:M Channels (Subscription)
writes

API Designs

1. Upload Video

Endpoint: POST /api/v1/videos

Authentication: Bearer JWT Token

Request

```
{
 "title": "System Design Tutorial",
 "description": "Complete YouTube System Design",
 "category": "Education",
 "visibility": "public",
 "tags": [
  "system-design",
  "backend",
  "architecture"
 ]
}
```

Response

```
{
 "video_id": "vid_987654321",
 "upload_url": "https://upload.youtube.com/signed-url",
 "status": "Upload Initialized"
}
```

4. Search Videos

Endpoint: GET /api/v1/search?q=system+design&type=video

Response

```
{
 "query": "system design",
 "total": 12548,
 "results": [
  {
   "video_id": "vid123",
   "title": "System Design Interview",
   "channel": "Tech Academy",
   "duration": 920,
   "thumbnail_url": "https://cdn.youtube.com/thumb.jpg"
  }
 ]
}
```

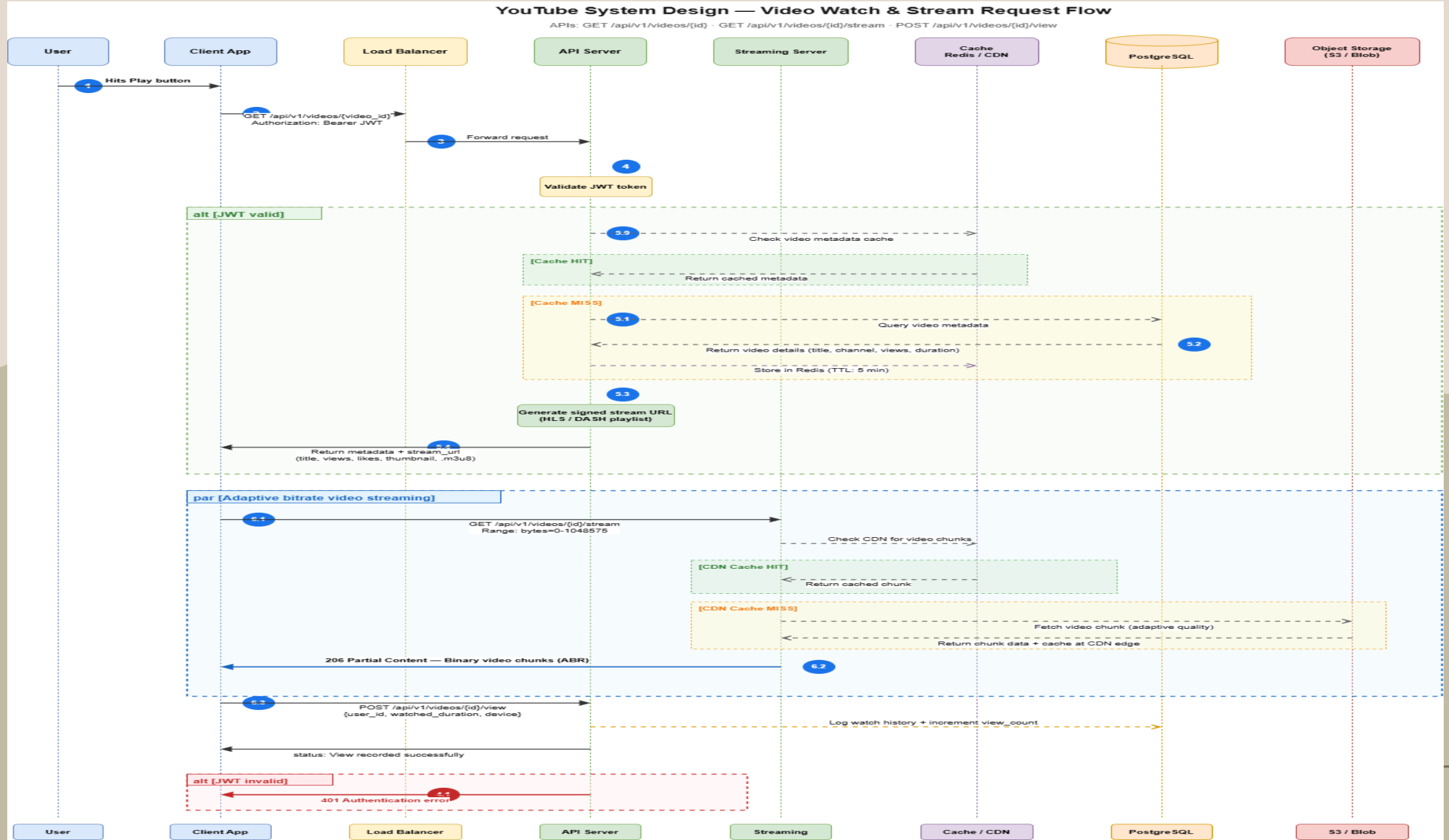
5. Recommendations

Endpoint : GET /api/v1/recommendations/{user_id}

Response

```
{
 "user_id": "user123",
 "recommendations": [
  {
   "video_id": "vid111",
   "title": "Distributed Systems",
   "score": 0.98
  },
  {
   "video_id": "vid222",
   "title": "Kafka Tutorial",
   "score": 0.96
  }
 ]
}
```

Watch and Stream Flow



Final Summary of Recommendation System:

Recommendation Techniques

- User Behavior Analysis – Learns user interests from watch history, search history, likes, subscriptions, and watch time.
- Content Similarity – Recommends videos with similar categories, tags, metadata, and semantic embeddings.
- Collaborative Filtering – Suggests videos liked by users with similar viewing patterns using machine learning embeddings instead of direct user-to-user comparisons.

Mode	Purpose
Real-Time Recommendation	Updates suggestions instantly based on recent user interactions (e.g., watch history, likes, searches).
Offline Batch Recommendation	Periodically retrains recommendation models using historical viewing patterns to improve long-term recommendation quality.

Key Design Principles Followed

- **Microservices Architecture** for modularity and independent scaling.
- **Event-Driven Communication** to decouple services and improve throughput.
- **Horizontal Scaling** to handle increasing traffic efficiently.
- **Caching First Strategy** to reduce latency and database load.
- **Stateless Services** to simplify scaling and failover.
- **Data Partitioning and Sharding** to support billions of records.
- **Asynchronous Processing** for compute-intensive workflows like video encoding.
- **High Availability and Disaster Recovery** through replication and multi-region deployment.
- **Security by Design** using authentication, encryption, and secure media delivery

Drive Link: https://drive.google.com/drive/folders/1jFpHevAoQ-Mz6l6QFJDX-bXpx48WTuBQ?usp=drive_link

The Detailed Document also shared that contains all calculations and diagrams