Design YouTube

Friday, March 22, 2024 8:44 PM

FUNCTIONAL REQUIREMENTS

1. upload a video
2. view / stream a video
3. search for a specific video
4. Like / dislike a video
5. Comments on a video
6. Build user profile.

Extended requirements
1. Mark video as a favorite
2. Build playlists
3. User subscriptions

NON FUNCTIONAL REQUIREMENTS

1. Available
2. Low latency
3. reliability

CAPACITY ESTIMATION.

Total users – 2 Billion.
500 million daily users
On an average 5 videos/day are viewed.

Traffic

(reads) 500M x 5 = 2500 x 10^6 ⇒ 2.5 billion videos/day.

(writes) upload : view = 1:200

$\frac{2.5}{200} \Rightarrow \frac{2500^M}{200} = 12.5M$ videos/day.

Storage:

assume 500 hrs worth of video/min.
1 min video ⟶ 50 Mb

500 (hrs) x 50 (Mb) x 60 (min) ⟶ 15,0000
= 1500 GB/min

Bandwidth:
Assume a cap of 10 mb/min.

Incoming (to server):

w.r.t 500 hrs worth of videos/min

500 (hrs) x 60 (min) x 10 (Mb/min)

= 30000 ⇒ mb/min

= 300 GB/min

= 5 GB/sec

Outgoing:

upload : views = 1:200
5x200 GB/sec = 1,000 GB/sec.
= 1 TB/sec

DATA MODELLING,

Users: user_id, name, email, loc
video: video_id, video_loc, user_id, created at, video_size, view count, permissions
comments: video_id, comment_id, user_id, created At, text, no. of likes

API

| upload video (API_dev_key, user_id, title, video_loc, description, video content, subtitles)
success/failure
video id.

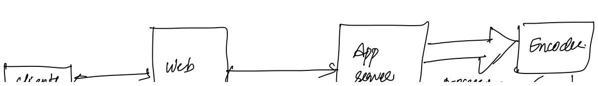
stream ← stream video (API_dev_key, video_id, resolution, offset, user id) — offset: when you forward and start.

List of videos ← search video (api_dev_key, searchstring, filters, location, userId, maxVideos to return)

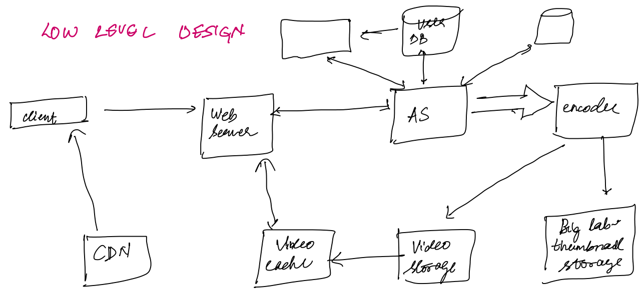
success/failure ← delete video (api_dev_key, video_id, userId)

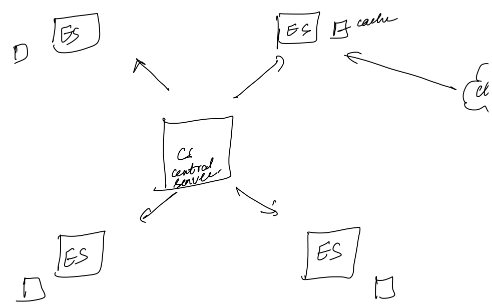
HIGH LEVEL DESIGN

read heavy system

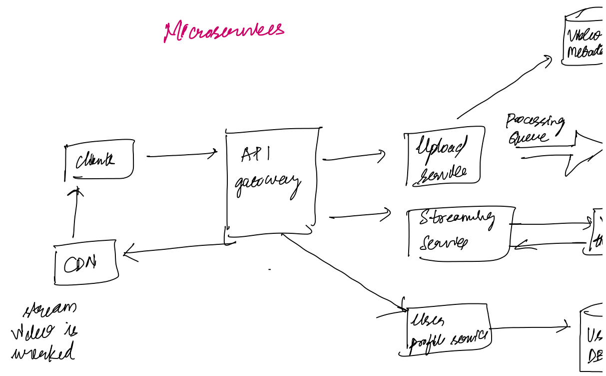


LOW LEVEL DESIGN





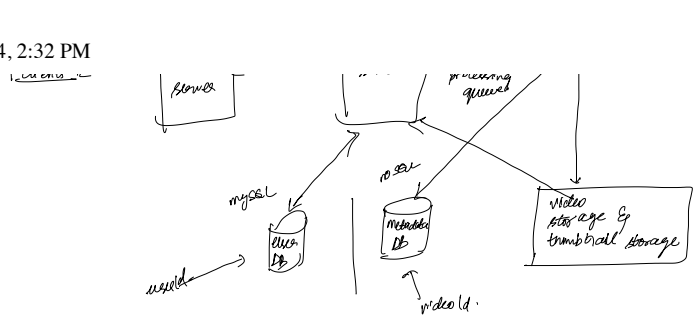
Microservices



(cache) Cache Eviction Technique

LFU

(Partitioning) userId? load imbalance (hot user)
videoId? video parts can be on multiple servers
If we can't decide use "consistent hashing"



video - [20 thumbnails/video]

video → HDFS
thumbnails → Google bigtable.
→ write/read traffic