SYSTEM DESIGN

"CONTINUE WATCHING"

in OTT platforms

{ Discussion focused on Hotstar Architecture }

What is "continue watching" - Functional Req.

1. You don't have to remember wherever you last left off the video.
2. Experience across the devices - Sync the content watched across the devices user is logged in.

Due to #2 above, we can't use IN-app cache, the data should be put somewhere at central location to sync across the devices.

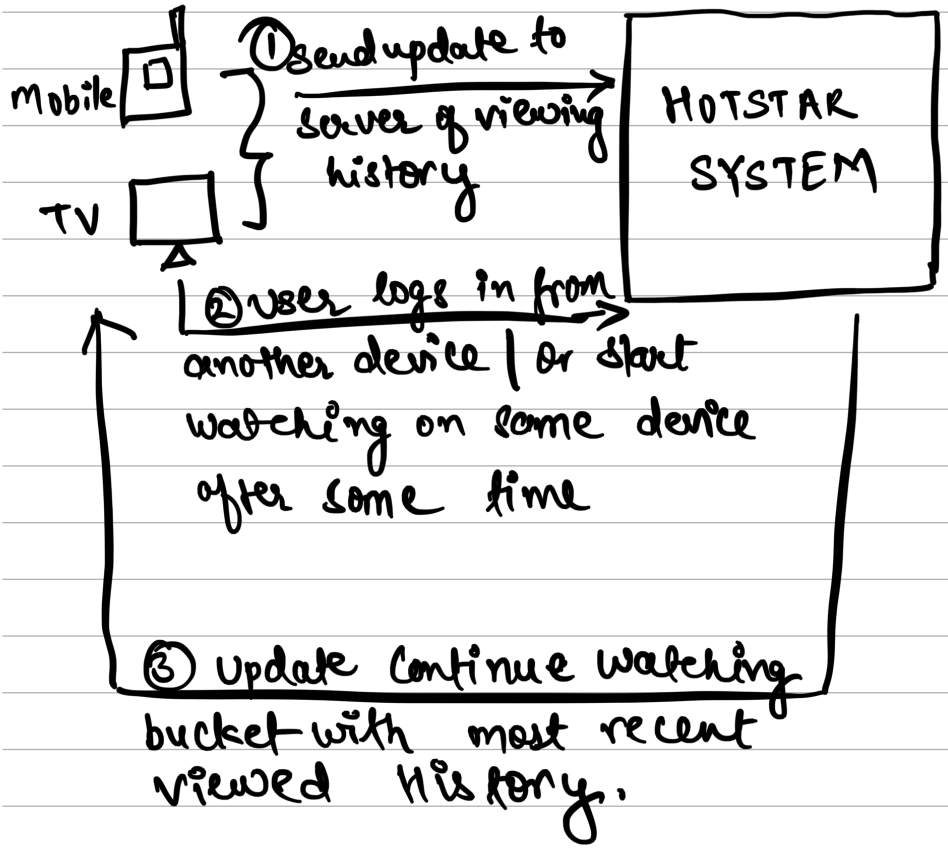
Non Functional Requirements:

1. low latency - should be synced across the devices in no time.
2. Availability
3. Durability
4. Eventual Consitency → related to #1
5. Less cost - Operations & money.

User Statistics (asper internet)

50M subscribers
300M active users → users watch avg 1B minutes of video/day
→ This requires ~200GB/day storage for continue watching feature

very High level view of Architecture →



→ Hotstar system should consist of database(s) which scales for 300M plus active users and maintains the watch history of every user.

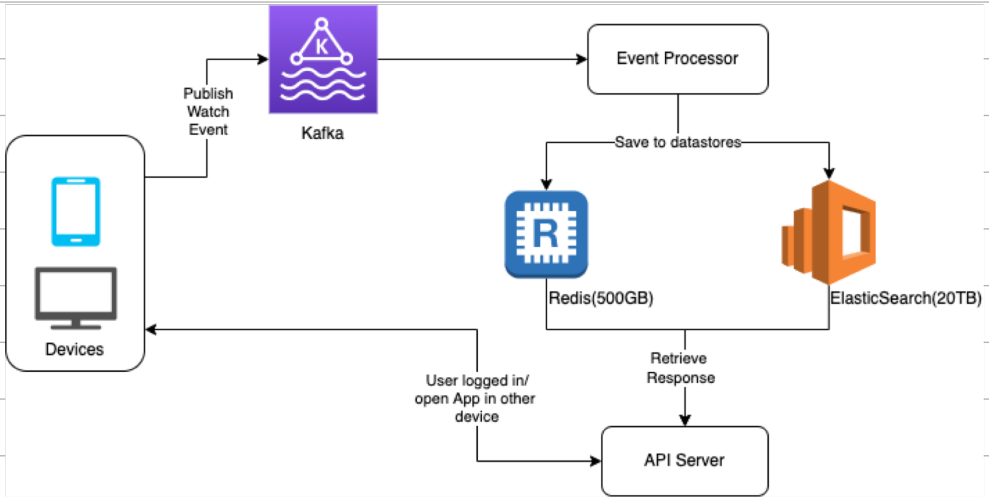
→ The database will definitely be write heavy.

→ The read latency should be minimum - users "Continue watching" bucket should be updated as soon they login (open the application).

→ Database should scale in heavy traffic times, request volume increases by 10 to 20 times within a minute.

Architecture 1

Use combination of Redis (key value store) and ElasticSearch (document store) for data storage and retrieval.



Database Model →

Redis

userId	list<content Id>
mandeep	{movie1, movie2}

Elastic Search

```
userId-content-id: {
    "field1" : xyz
    "field2" : pqr
}
```

→ Redis maintains list of content for a user.

→ Elastic Search Index contains metadata for userId-contentId

Example →

till what timestamp video is watched
which episode of series is in progress.

Bottlenecks in this design

① Redis latency was upto the mark, but there is requirement to horizontally scale the cluster as data size grew.
↳ increased cost every 3-4 months.

② ES latency ~200 milliseconds
↳ avg cost of Elastic Search was at higher end.
↳ its just a user experience feature

↳ Node maintanence issues and manual effort requirement to resolve issues.

↳ We can minimize this keeping cluster size small - Watch "How Discord Index Billions of messages" for complete analysis.

③ Two data stores for same use-case. - becomes difficult to maintain two data stores with different code base and query pattern at high scale.

↳ Extra administration cost.

Architecture Improvements →

① think if we can solve this with single database.

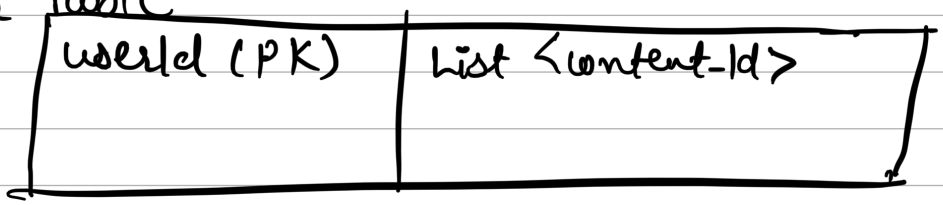
② Hotstar choose ScyllaDB as a replacement of Redis and ES cluster

- → low latency for read/write
- → Operational simplicity
- → low cost - compared to other NOSQL solutions available in Cloud.

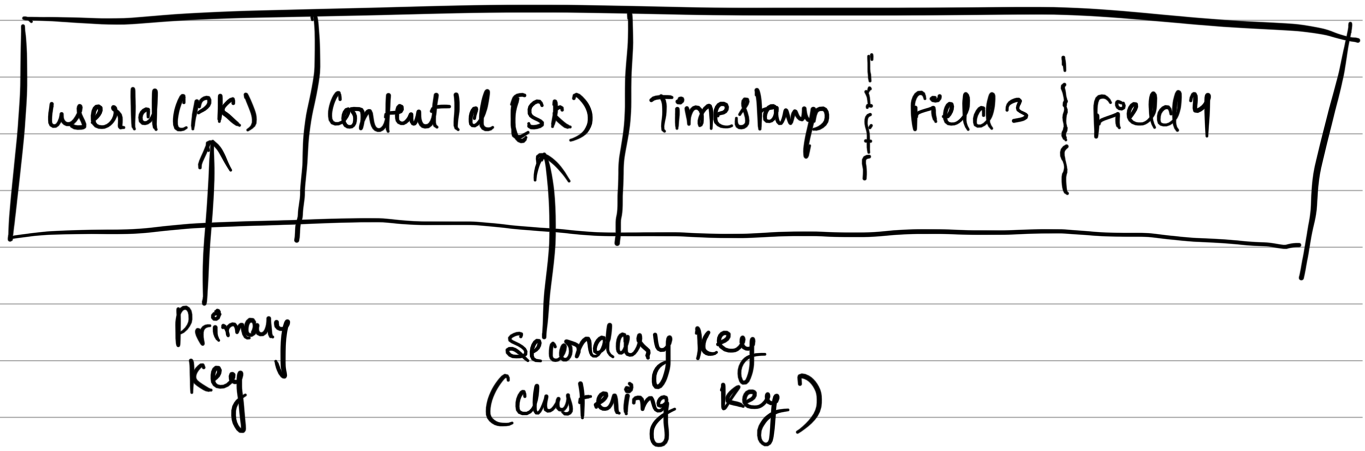
ScyllaDB Model

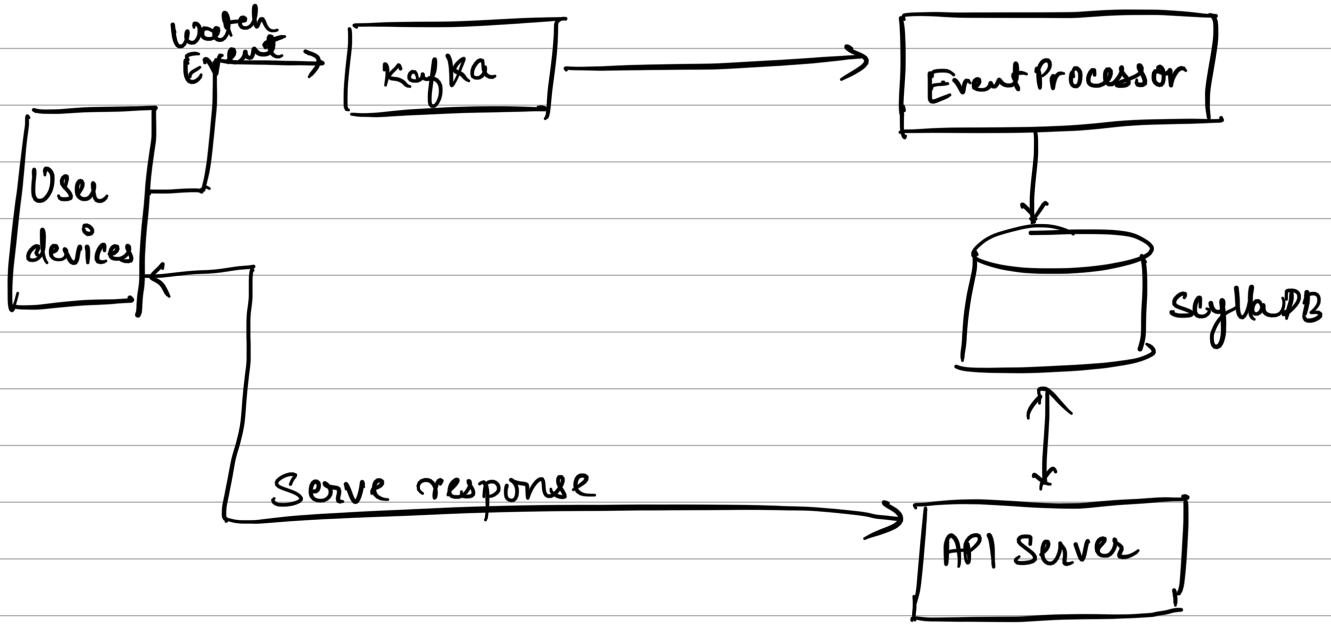
→ it is based on cassandra

① User Table



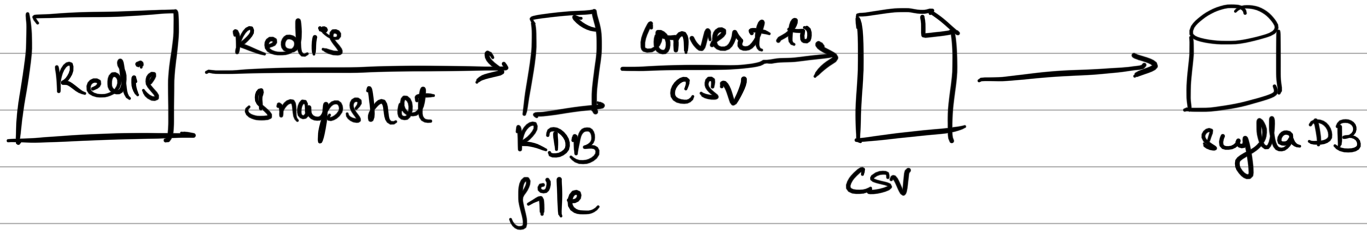
② User content Table

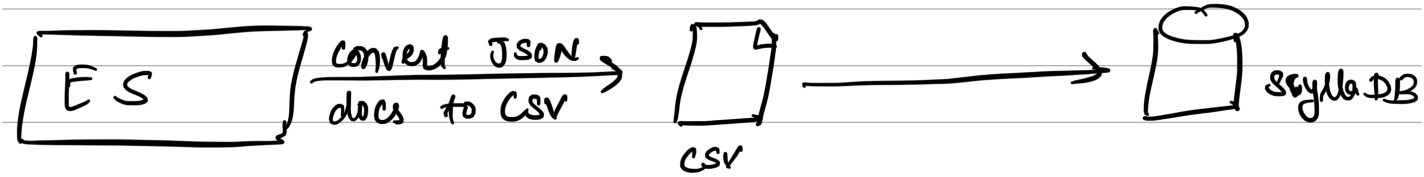




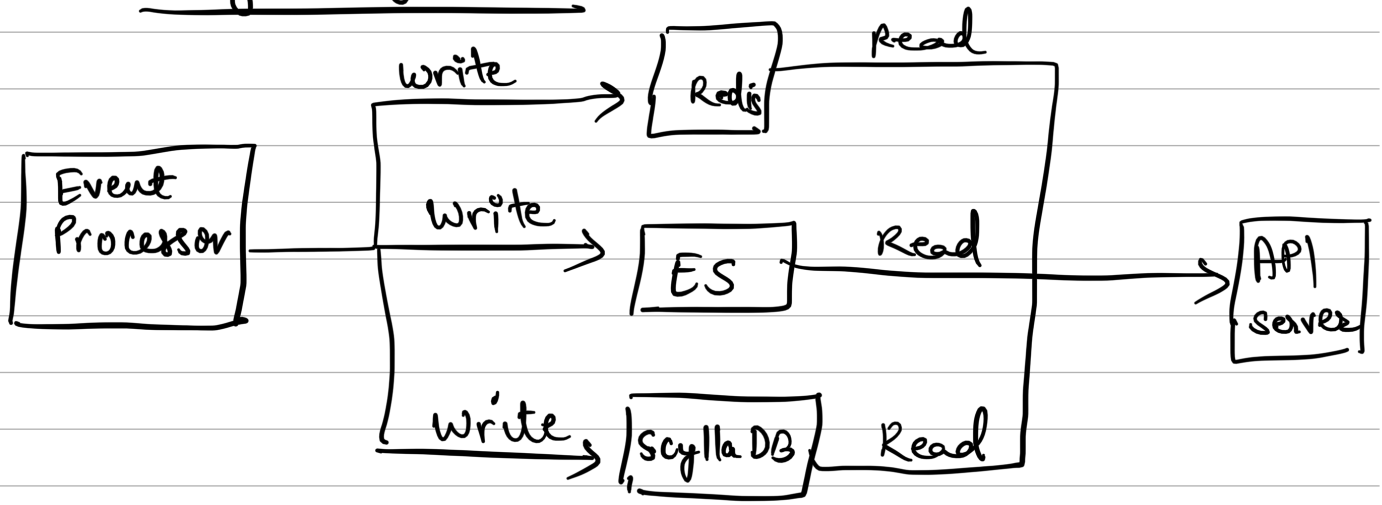
Migration from old to new { with NO DOWN TIME }

① Move old data from Redis / ES to ScyllaDB
② Run both system in parallel for confidence building
③ Remove old system.





During Migration :



Subscribe for more such content.

YouTube Channel - MsDeep Singh

Happy Learning ☺