DropBox/Google Drive

An application that saves the files and media.Here the user can upload, download, share and can do other options.

Features

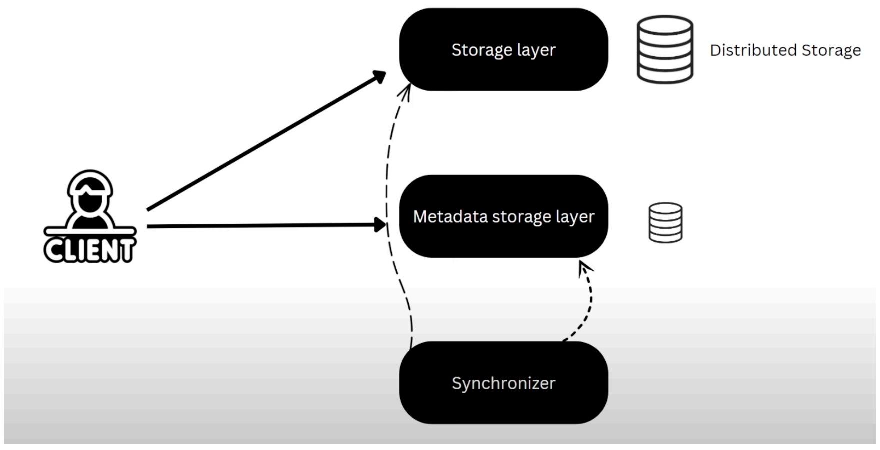
Functional	Non Functional
Upload	High Availability
Download	Low Latency [Sync Easily]
Share	Scalability
	ACID [Atomicity, Consistency, Isolation, Durability]
	Optimize data transfer by minimizing bandwidth. [If any change, do we have to upload the whole file or we can create a file and distribute it into smaller chunks and it uploads the chunks that are updated]

- **Atomicity** : If anything is updated, then the other will get updated at once not 1 by 1.
- **Consistency** : If we delete the file, the file data gets deleted for all at once.
- **Isolation** : If file is uploaded by **2 persons** at a single time, it should not be any issue.
- **Durability** : The file uploaded, should always be present.

Estimating the system Capacity

Total number of Users = 1 Billion
Average user stores = 20 Gb
Total Space = 20 Gb * 1 Billion = 20 ExtraByte [20 Billion Gb]

High Level Diagram [HLD]

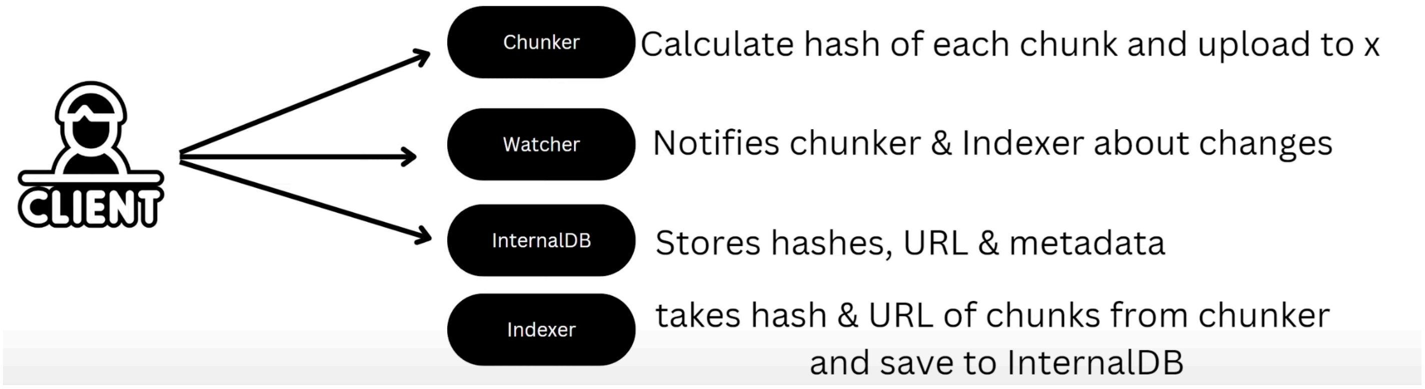


Caption

- **Storage Layer :** where the data is store. It can be Google cloud/Hadoop/Amazon S3 servers.
- **Metadata Storage Layer** : It stores the metadata related to the files.
- **Synchronizer :** Any changes in the files are closely watched by synchronizer, if any changes done in file , the metadata will also changes, so it synchronizes the changes to all the system an devices.
- **Client :** Client can perform the following tasks :
    - Upload the file
    - Download the files.
    - Detect any changes.
    - Break the file into smaller chunks, therefore optimize the file transfer.

A client can be divided into the following :

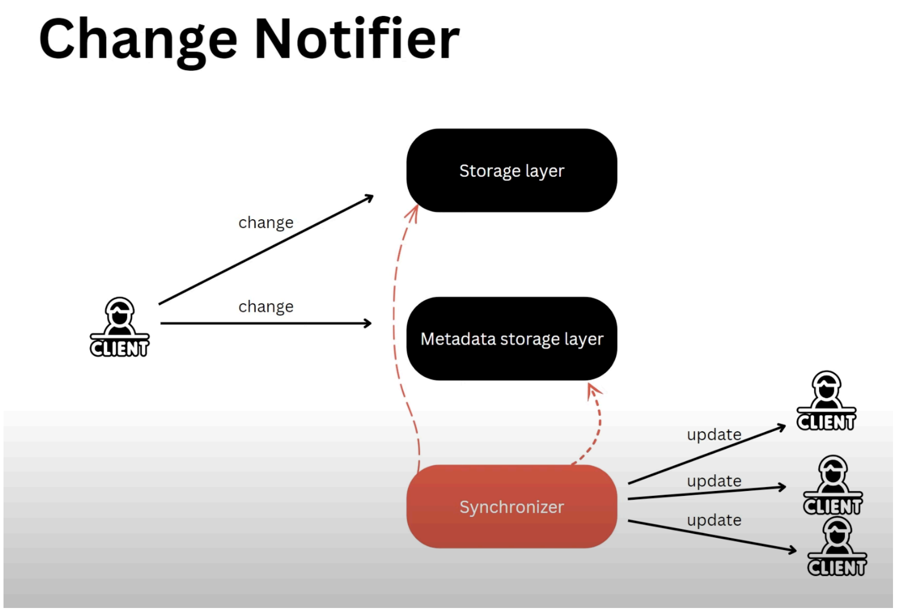
- **Watcher :** Notifies the chunker & indexer about the changes. It watches the incoming changes
- **Chunker :** Collects the file and divides the files into small chunks and gets the hash of every chunks and store the chunks or saves it to the internal DB
- **Indexer :** It takes the hashes from the chunker & url of the chunks and save them ot the internal DB
- **Internal DB :** It stores the hashes, URL & metadata of the files [chunk_id, chunk_order, URL]


CLIENT
Chunker
Calculate hash of each chunk and upload to x
Watcher
Notifies chunker & Indexer about changes
InternalDB
Stores hashes, URL & metadata
Indexer
takes hash & URL of chunks from chunker and save to InternalDB

Client Roles

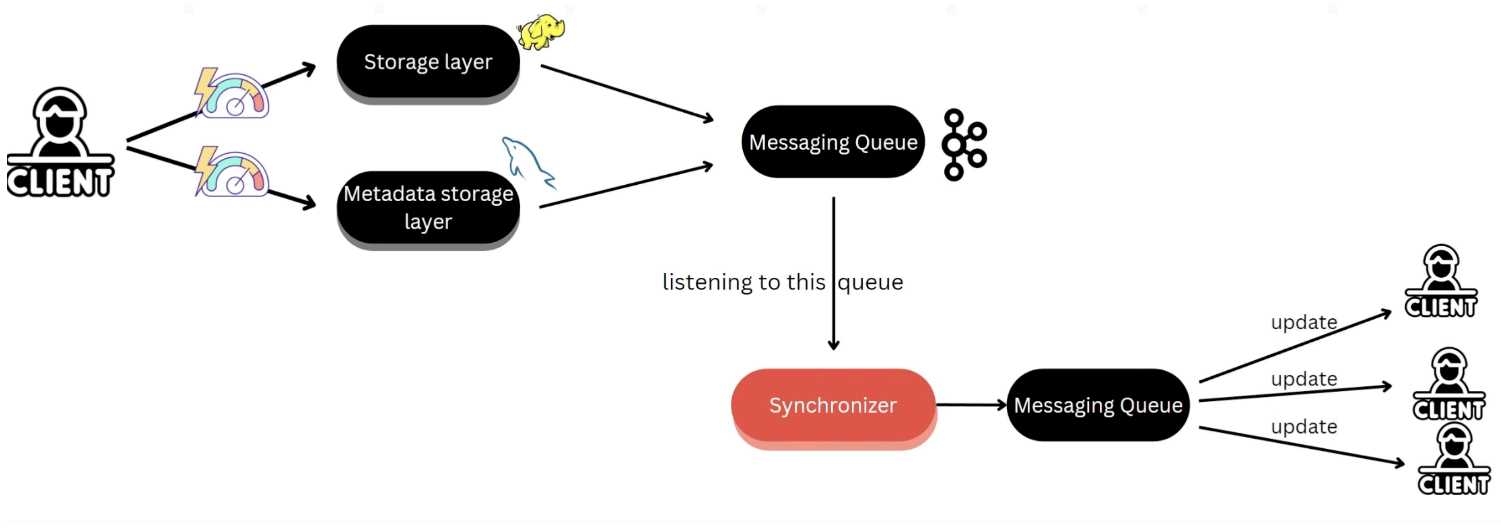
- **Storage layer :** Goolge Cloud / Amazon S3 / Hadoop
- **Meta Data Storage Layer :** Mysql [due to highly consistent]
    - Permissions
    - Device Lists
    - Version
    - Hashes
    - Information of chunks [chunk_id, chunk_order, URL]
- **Synchronizer :** It tells other clients that a client have updated some changes in the files.

Any Change by a client => Changes to the Storage Layer & the metadata layer is done, all these will be pushed to the **Messaging Queue** => Consumer of this queue is the **Synchroniser.**
Now Synchroniser, will send the new updates after the consumption and will send the new updates to the **Messaging Queue.**



Caption

Any updates int he Messaging Queue is listened by the Clients.



HLD

For Messaging Queue We will Use **Kafka**

Scale Up

- We can scale up the Metadata Storage Layer by Database Sharding.
- Synchroniser can be distributed into redundant services.
- In Metadata Storage Layer, data is stored in the MySql, we can distribute the sql server present on the user location.
  - We can get the user location, If US, then we can distribute out MySql server & put all the data related to US in that database.
- We can also use consistent hash partitioning.