Design Dropbox/Google Drive

Nikhil Gupta · Follow

7 min read · May 12, 2023

Overview

We all must have used Google drive or dropbox at some point in our life or some of us maybe active users of them at the moment of reading this article. Have you ever wondered how it works? How billions of data from millions of users across the world is managed by Google drive/Dropbox. In this article we will discuss how to design a cloud file hosting service like Google drive/Dropbox.

> *Note: Designing a Drive is a very complex task and everything may not be covered in this article. But we will have a starting point and feel free to extend the design.*

Requirements

Functional

1. User should be able to upload photo/files.
2. User should be able to create/delete directories on the drive.

3. User should be able to download files
4. User should be able to share the uploaded files.
5. The drive should synchronise the data between user all devices.
6. User should be able to upload files/photos even when internet is not available. As and when internet gets available, the offline files will be synced to online storage.

Non functional

1. **Availability —** Availability means what percentage of the time the system is available to take a user's request. We generally mentions availability as 5 Nine's, 4 Nine's, etc. 5 Nine's means 99.999% availability, 4 Nine means 99.99% availability and so on.
2. **Durability** — Durability means the data uploaded by user should be permanently stored in database even in case of database failure. Our System should ensure the files uploaded by user should be permanently stored on the drive without any data loss.
3. **Reliability** — Reliability means how many times the system gives expected output for the same input.
4. **Scalability —** With the growing number of user our system should be capable enough to handle the increasing traffic.
5. **ACID properties —** Atomicity, Consistency, Integrity and Durability. All the file operations should follow these properties.

Atomicity — It means that any operation executed on a file should be either complete or incomplete. It should not be partially complete, i.e. If a user uploads a file the end state of the operation should be either the file is 100% uploaded or it is not at all uploaded.

Consistency — The consistency should ensure that the data should be same before and after the operation is completed. Means the file that user uploads should be same before and uploading the file on the drive.

Isolation — It means that 2 operations running simultaneously should be independent and not affect each other's data.

Durability — Already explained above.

Capacity Estimation

Let's assume that we have 500M total users out of which 100M are daily active users.

Which means we will have on an overage 0.07M user's active per minute. Let's assume at peak we will have 1M active user's per minute.

Assuming on an average 1 user uploads 5 files, we will have 5 million uploads per minute.

If 1 upload is on an average 100KB then total data uploaded in a minute will be 100*5 = 500TB per minute.

This gives us a picture about how much data we have to store for all of our users.

API

1. **Upload file**

```
POST: /file
Request {
  filename: string,
  createdOnInUTC: long,
  createdBy: string,
  updatedOnInUTC: long,
  updatedBy: string
}

Response: {
  fileId: string,
  downloadUrl: string
}
```

The upload file will work in 2 steps. 1st the file metadata will be uploaded to the server and then the file upload process will start.

2. Download file.

```
GET: /file/{fileId}
Response: {
  fileId: string,
  downloadUrl: string
}
```

The url provided by the backend will be used to download the url, it can also be used to share the file to other user's.

3. Delete file.

```
DELETE: /file/{fileId}
```

4. Get Latest Snapshot.

```
GET: /folders/{folderId}?startIndex={startIndex}&limit={limit}

Response: {
  folderId: string,
  fileList: [
    {
      fileId: string,
      filename: string,
      thumbnail_img: string,
      lastModifiedDateInUTC: string
      creationDateInUTC: string
    }
  ]
}
```

The snapshot API will give the list of files present in the folder. It may happen that folder is having too many files, in that case we have to return the paginated result and load the data as user scroll's.

Design Consideration

1. **File storage for users** — We want highly available and durable system to store the user uploads. For this we can use AWS S3. S3 is highly available and durable service offered by AWS. S3 bucket does not have any limit to any storage capacity and the max object size it can store is 5TB which is way more than the most of the devices currently used.

2. **Storing user data and their uploads metadata—** To store the user data and their files metadata we can use a Non Relational database like AWS Dynamo DB which offers high availability and reliability.

3. **Offline Updates —** When a user's device is offline all the updates done by user will stored on their local device storage and once the user comes online the device will sync the updates on to the cloud.

4. **Uploading files —** The size of the file uploaded by user can be huge and to upload the file from any device on to the drive without any failure we have to upload it in parts. For this we can use multipart upload feature provided by S3. However, the multipart feature will be used in FileUploadService to upload the file to S3 bucket. But to transfer the file to FileUploadService, we need to some kind of small packet based uploader at client device application as well. For this we can build a custom library that will divide the large file into small packets and upload the packets to the service. We can design the library such that it should be able to upload multiple packets simultaneously and FileUploadService should be able to collate packets and then upload to S3 bucket.

5. **Downloading/sharing files —** The files will be shared via S3 Pre-signed Url. You can read more on sharing objects via Pre-signed url from the doc.

6. **Sync Between devices —** When a user makes a change on 1 of its device, the changes should be reflected on its other devices when user logs into other devices. There can be 2 ways to do it. 1) As soon as user updates from 1 device, the other devices should be updated. 2) Update other devices when the user logs into it. We will go with **second** approach as it will prevent unnecessary updates to other devices even when user is not using other devices. Now we have discussed the use case when user logs

into the device, what if user is online on 2 different devices. Then in that case we can use long polling. The device on which user is currently online will long poll the backend server and wait for any update. So, when a user makes an update on 1 device, the other device will get the update.

Database Design

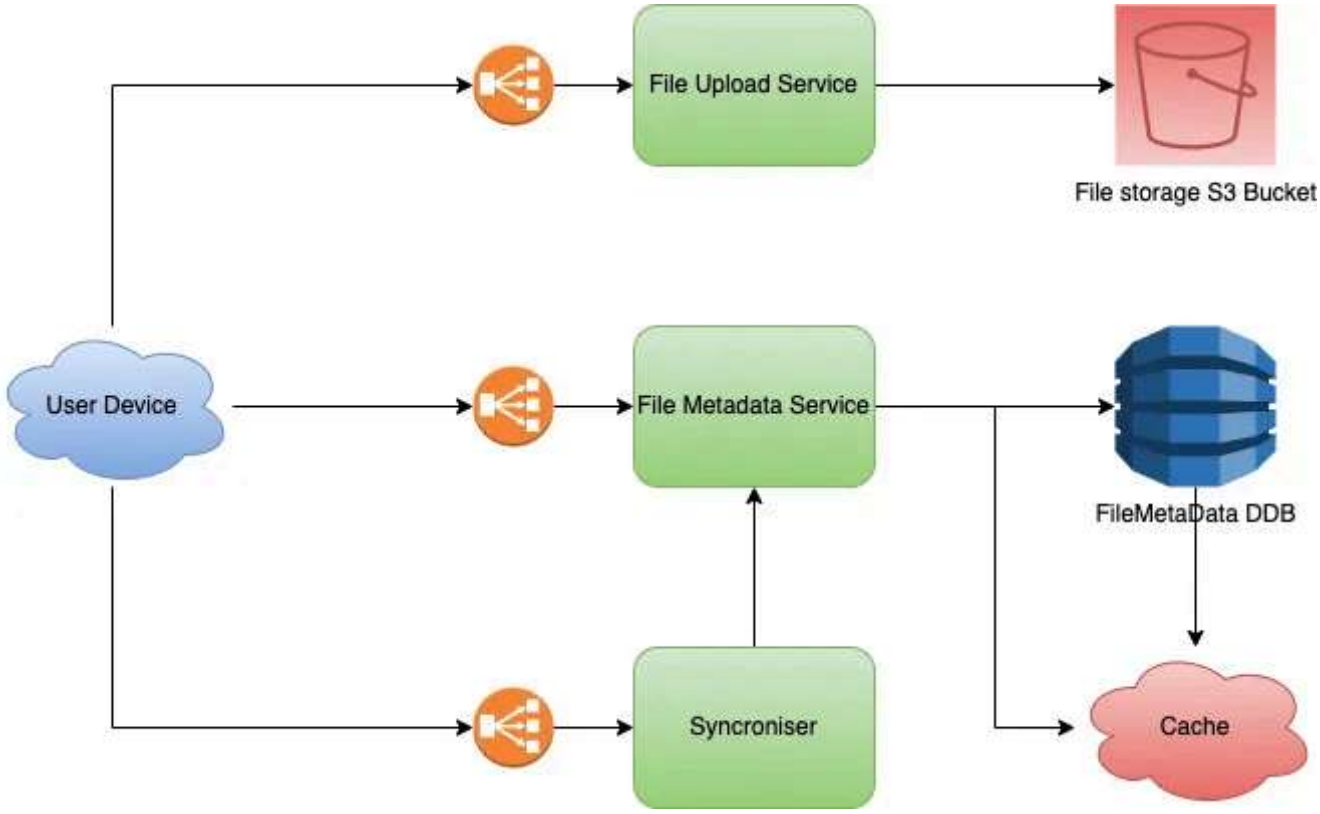
User table

```
userId: string
username: string
emailId: string
creationDateInUtc: long
```

Metadata table

```
fileId: string
userId: string
filename: string
fileLocation: string
creationDateInUtc: long
updationDateInUtc: long
```

HLD



Component Details

1. **FileMetaDataService** — This service will be responsible for adding/updating/deleting the metadata for the user uploaded files. The client devices will communicate with this service for metadata of files/folders.

2. **FileUploadService** — This service will be responsible for uploading the file to S3 bucket. User's device will stream the file to this service in chunks and once all the chunks get uploaded to the s3 bucket the upload will be completed.

3. **SynchronisationService** —There will 2 situations where we need synchronisation. 1) When user opens the application on its device, then in that case we will sync the user's that device from the Synchronisation service with the latest snapshot of the directory user currently viewing. 2) When the user is online from a device and there may be a possibility that user updates the drive from another device then in that case we need to

sync user's first device, So, for that we can use long polling to poll for any latest changes for that directory. Lazy sync with long polling.

4. **S3 Bucket** — We are using S3 bucket to store the user files/folders. Since, there is no limit to bucket size. We can create folder based on userId's and files/folders for each user can be stored in that user folder.
5. **Dynamo DB** — This Database will be used in storing user data, file metadata as per the database design explained above.
6. **Cache** — We are using Cache to reduce the latency for metadata retrieval, when client requests the metadata for file/folder, it will first look into the cache and if not found in cache then it will look into DDB, store in cache and return the result.
7. **Load Balancer** — We want our services to scale for million users and for that we need to scale our services horizontally. We will be using a load balancer to distribute the traffic to different hosts.
8. **User device** — User's can have multiple devices like mobile, Desktop, Tablet for accessing the drive. We need to ensure that all the user devices have same data and there should be data discrepancy.

Bottlenecks & future extension

1. Sharing folder to other users — We covered sharing files to other users, what if user wants to share the folder and other user should be able to view the folder and it's subfolders and files.
2. Permission based sharing control — This will allow user to share the file and configure what permission other permission should be allowed like read only, writing, etc.

3. Make user data secure — As of now we haven't talked about how we can secure the user data. Since, we do not have control over the data user is storing, we need to assume that every data user stores is very critical and should not be leaked to unauthorised person.

> *There can be lot of extensions and improvements possible for the current proposed design, feel free to drop your suggestions in the comment box. Hope you find this article helpful, keep watching the space and follow for more such future articles.*

Software Development | Software | Programming | Interview | Software Architecture

Written by Nikhil Gupta

1.5K Followers · 19 Following

Senior SDE @Microsoft. 8+ Years of Software Development experience.
www.linkedin.com/in/nikhilgupta240

Follow

Responses (2)

Soukhin Nayek

What are your thoughts?