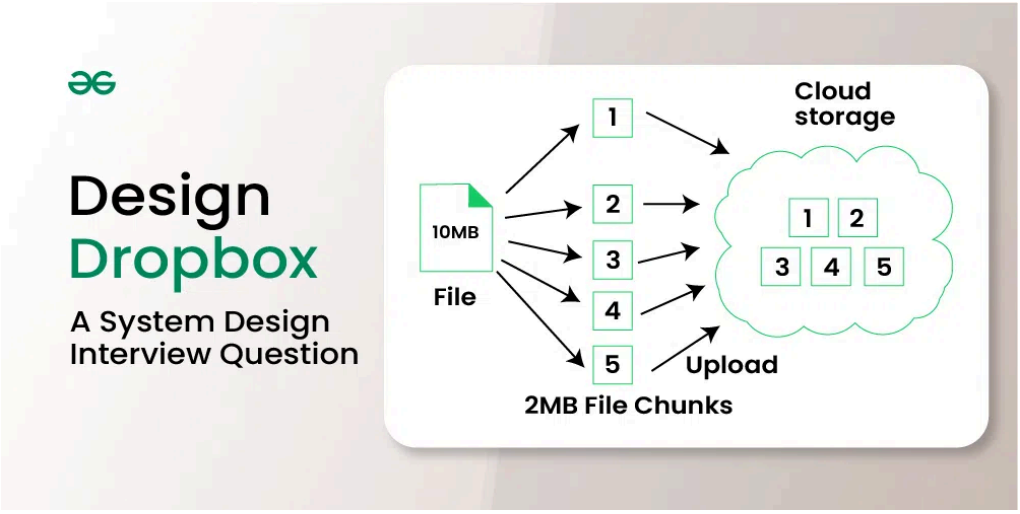
Design Dropbox – A System Design Interview Question

Last Updated : 14 Feb, 2024

System Design Dropbox, You might have used this file hosting service multiple times to upload and share files or images. System Design Dropbox is a quite common question in the system design round. In this article, we will discuss how to design a website like Dropbox.



Important Topics for the Dropbox System Design

- Requirements Gathering for Dropbox System Design
- Capacity Estimation for Dropbox System Design
- High-Level Design(HLD) of Dropbox System Design
- Low-Level Design(LLD) of Dropbox System Design
- Database Design for Dropbox System Design
- API Design for Dropbox System Design
- Scalabilty for Dropbox System Design

1. Requirements Gathering for Dropbox System Design

Functional Requirements:

- The user should be able to upload photos/files.
- The user should be able to create/delete directories on the drive.
- The user should be able to download files
- The user should be able to share the uploaded files.
- The drive should synchronize the data between user all devices.

Non Functional Requirements:

- **Availability:** Availability means what percentage of the time the system is available to take a user's request. We generally mention 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.
- **Durability:** Durability means the data uploaded by the user should be permanently stored in the database even in case of database failure. Our System should ensure the files uploaded by the user should be permanently stored on the drive without any data loss.
- **Reliability:** Reliability means how many times the system gives the expected output for the same input.
- **Scalability:** With the growing number of users our system should be capable enough to handle the increasing traffic.
- **ACID properties:** Atomicity, Consistency, Integrity and Durability. All the file operations should follow these properties.

2. Capacity Estimation for Dropbox System Design

Storage Estimations:

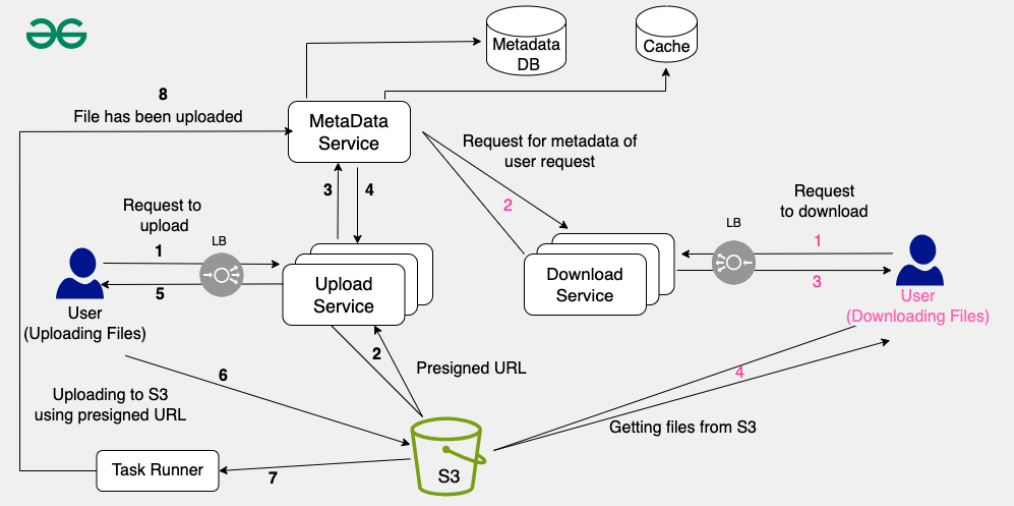
Assumptions:

The total number of users = ***500 million.***
Total number of daily active users = 100 million
The average number of files stored by each user = 200
The average size of each file = 100 KB
Total number of active connections per minute = ***1 million***

Storage Estimations:

Total number of files *= 500 million * 200 =* ***100 billion***
Total storage required *= 100 billion * 100 KB =* ***10 PB***

3. High-Level Design(HLD) of Dropbox System Design



3.1. User Uploading:

Users interact with the client application or web interface to initiate file uploads. The client application communicates with the Upload Service on the server side. Large files may be broken into smaller chunks for efficient transfer.

3.2. Upload Service:

Receives file upload requests from clients. Generates Presigned URLs for S3 to allow clients to upload directly. Coordinates the upload process, ensuring data integrity and completeness. After successful upload, it updates the Metadata Database with file details. Coordinates the upload process, breaking down large files into manageable chunks if necessary.

3.3. Getting Presigned URL:

The client application requests a Presigned URL from the Upload Service. The server generates the Presigned URL by interacting with the S3 service, creating a unique token for the specific upload operation. These URLs grant temporary, secure access to upload a specific file to a designated S3 bucket. Allows clients to bypass the server for direct communication with the storage layer.

3.4. S3 Bucket:

S3 serves as the scalable and durable storage backend. Presigned URLs allow clients to upload directly to S3, minimizing server involvement in the actual file transfer. The bucket structure may organize files based on user accounts and metadata.

3.5. Metadata Database:

Stores metadata associated with each file, including details like name, size, owner, access permissions, and timestamps. Enables quick retrieval of file details without accessing S3. Ensures that file metadata is consistent with the actual content in S3.

3.6. Uploading to S3 using Presigned URL and Metadata:

The client uses the Presigned URL to upload the file directly to the designated S3 bucket. Metadata associated with the file, such as file name and owner, is included in the upload process. This ensures that the file's metadata is synchronized with its corresponding data in S3.

3.7. Role of Task Runner:

After the file is successfully uploaded to S3, a task runner process is triggered. The task runner communicates with the Metadata Database to update or perform additional tasks related to the uploaded file. This may include updating file status, triggering indexing for search functionality, or sending notifications.

3.8. Downloading Services:

Clients initiate file download requests through the client application. The Download Service queries the Metadata Database for file details. The server's Download Service retrieves metadata from the Metadata Database. Metadata includes information such as file name, size, owner, and access permissions.

4. Low-Level Design(LLD) of Dropbox System Design

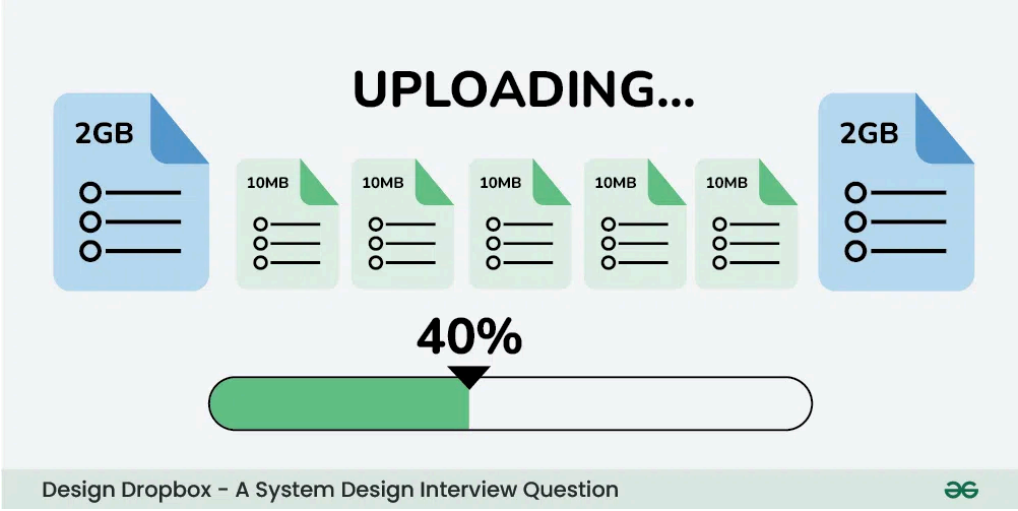
A lot of people assume designing a Dropbox is that all they just need to do is to use some cloud services, upload the file, and download the file whenever they want but that's not how it works. The core problem is "Where and how to save the files? ".

Suppose you want to share a file that can be of any size (small or big) and you upload it into the cloud.

Everything is fine till here but later if you have to make an update in your file then it's not a good idea to edit the file and upload the whole file again and again into the cloud. The reason is:

- **More bandwidth and cloud space utilization:**
    - To provide a history of the files you need to keep multiple versions of the files.
    - This requires more bandwidth and more space in the cloud. Even for the small changes in your file, you will have to back up and transfer the whole file into the cloud again and again which is not a good idea.
- **Latency or Concurrency Utilization:**
    - You can't do time optimization as well. It will consume more time to upload a single file as a whole even if you make small changes in your file.
    - It's also not possible to make use of concurrency to upload/download the files using multi threads or multi processes.

Let's discuss how we can solve this problem:

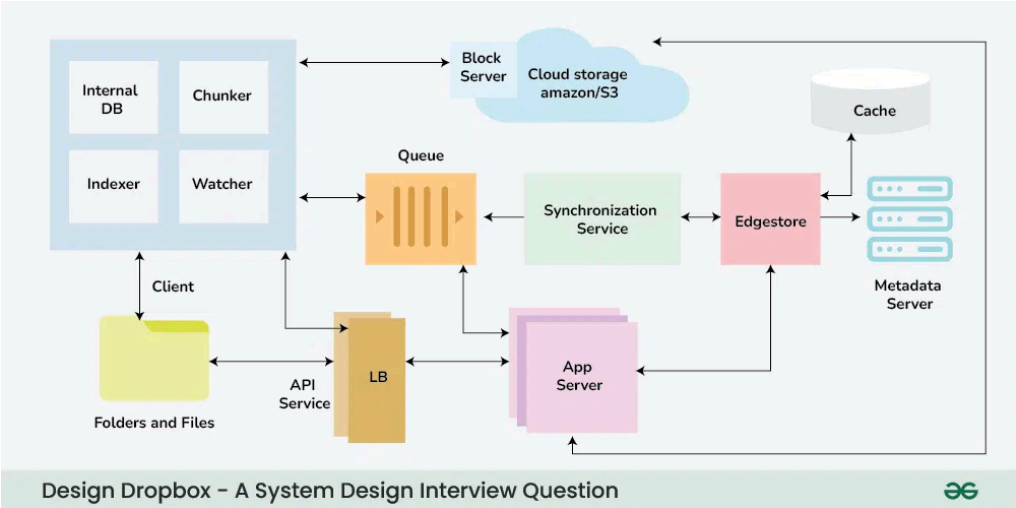


We can break the files into multiple chunks to overcome the problem we discussed above. There is no need to upload/download the whole single file after making any changes in the file.

- You just need to save the chunk which is updated (this will take less memory and time). It will be easier to keep the different versions of the files in various chunks.
- We have considered one file which is divided into various chunks. If there are multiple files then we need to know which chunks belong to which file.
- To keep this information we will create one more file named a metadata file. This file contains the indexes of the chunks (chunk names and order information).
- You need to mention the hash of the chunks (or some reference) in this metadata file and you need to sync this file into the cloud. We can download the metadata

file from the cloud whenever we want and we can recreate the file using various chunks.

Now let's talk about the various components for the complete **low level design** solution of the Dropbox.



Let's assume we have a client installed on our computer (an app installed on your computer) and this client has 4 basic components. These basic components are Watcher, Chunker, Indexer, and Internal DB. We have considered only one client but there can be multiple clients belonging to the same user with the same basic components.

- The client is responsible for uploading/downloading the files, identifying the file changes in the sync folder, and handling conflicts due to offline or concurrent updates.
- The client is actively monitoring the folders for all the updates or changes happening in the files.
- To handle file metadata updates (e.g. file name, size, modification date, etc.) this client interacts with the Messaging services and Synchronization Service.
- It also interacts with remote cloud storage (Amazon S3 or any other cloud services) to store the actual files and to provide folder synchronization.

4.1. Client Components

- **Watcher** is responsible for monitoring the sync folder for all the activities performed by the user such as creating, updating, or deleting files/folders.
    - It gives a notification to the indexer and chunker if any action is performed in the files or folders.
- **Chunker** breaks the files into multiple small pieces called chunks and uploads them to the cloud storage with a unique id or hash of these chunks.
    - To recreate the files these chunks can be joined together. For any changes in the files, the chunking algorithm detects the specific chunk which is

modified and only saves that specific part/chunk to the cloud storage.
    - It reduces bandwidth usage, synchronization time, and storage space in the cloud.
- **Indexer** is responsible for updating the internal database when it receives the notification from the watcher (for any action performed in folders/files).
    - It receives the URL of the chunks from the chunker along with the hash and updates the file with modified chunks.
    - Indexer communicates with the Synchronization Service using the Message Queuing Service, once the chunks are successfully submitted to the cloud Storage.
- **Internal databases** stores all the files and chunks of information, their versions, and their location in the file system.

4.2. Metadata Database

The metadata database maintains the indexes of the various chunks. The information contains files/chunks names, and their different versions along with the information of users and workspace.

- You can use RDBMS or NoSQL but make sure that you meet the data consistency property because multiple clients will be working on the same file.
- With RDBMS there is no problem with the consistency but with NoSQL, you will get eventual consistency.

Lets understand how we can efficientlt do relational database scaling

4.2.1 Relational Database Scaling:

Relational databases like MySQL may face scalability challenges as the data and traffic grow.

- Scaling can be achieved using techniques such as vertical scaling (increasing hardware capabilities) or horizontal scaling (adding more machines).
- However, horizontal scaling for relational databases often involves complexities, especially in scenarios with high read and write operations.

4.2.2 Database Sharding:

Database sharding is a horizontal partitioning technique where a large database is divided into smaller, more manageable parts called shards.

- Each shard is essentially a separate database instance that can be distributed across different servers or even different geographic locations.

- Sharding helps distribute the load, improve query performance, and enhance scalability.

4.2.3 Challenges with Database Sharding:

Managing multiple shards can become complex, especially when updates or new information needs to be added. Coordinating transactions across shards can be challenging. Maintenance, backup, and recovery operations become more intricate.

4.2.4 Edge Wrapper:

An edge wrapper is an abstraction layer that sits between the application and the sharded databases.

- It acts as an intermediary, providing a unified interface for the application to interact with the database system.
- The edge wrapper encapsulates the complexities of managing multiple shards and provides a simplified interface to the application.

4.2.5 Object-Relational Mapping (ORM):

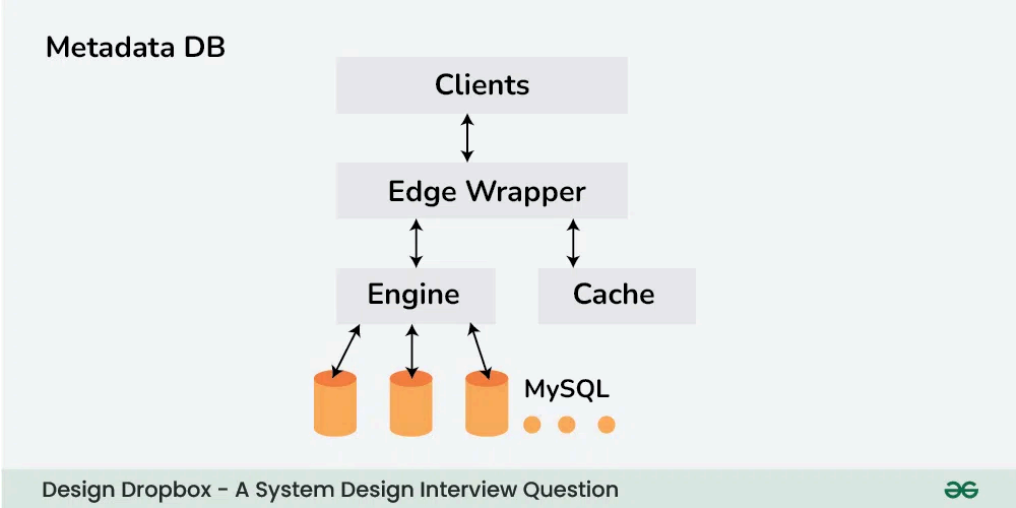
ORM is a programming technique that allows data to be seamlessly converted between the relational database format and the application's object-oriented format.

- It maps database tables to application objects, providing a convenient way to interact with the database using programming language constructs.
- ORM helps abstract away the intricacies of SQL queries and database schema, making it easier for developers to work with databases.

4.2.6 Edge Wrapper and ORM:

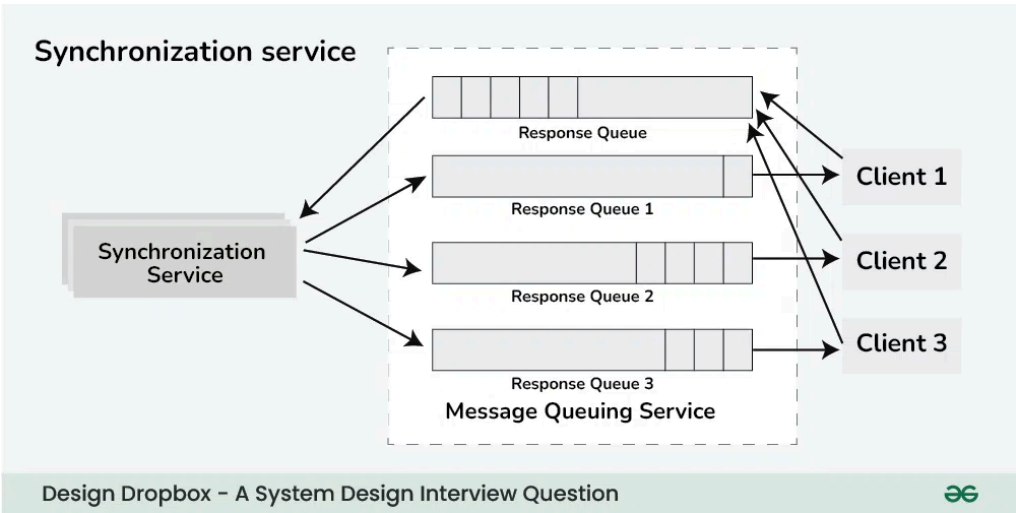
The edge wrapper integrates ORM functionality to provide a convenient interface for the application to interact with sharded databases.

- It handles tasks like routing queries to the appropriate shard, managing transactions across shards, and abstracting the underlying complexities of database sharding.
- ORM, within the edge wrapper, enables the application to interact with the database using high-level programming constructs, reducing the need for developers to write complex SQL queries.
- This combination of edge wrapper and ORM simplifies database management, making it easier to scale the application horizontally with sharded databases while maintaining a cohesive and developer-friendly interface.



4.3. Message Queuing Service

The messaging service queue will be responsible for the asynchronous communication between the clients and the synchronization service.



Below are the main requirements of the Message Queuing Service.

- Ability to handle lots of reading and writing requests.
- Store lots of messages in a highly available and reliable queue.
- High performance and high scalability.
- Provides load balancing and elasticity for multiple instances of the Synchronization Service.

There will be two types of messaging queues in the service.

- **Request Queue:**
    - This will be a global request queue shared among all the clients.
    - Whenever a client receives any update or changes in the files/folder it sends the request through the request queue.
    - This request is received by the synchronization service to update the metadata database.

- **Response Queue:**
    - There will be an individual response queue corresponding to the individual clients.
    - The synchronization service broadcast the update through this response queue and this response queue will deliver the updated messages to each client and then these clients will update their respective files accordingly.
    - The message will never be lost even if the client will be disconnected from the internet (the benefit of using the messaging queue service).
    - We are creating n number of response queues for n number of clients because the message will be deleted from the queue once it will be received by the client and we need to share the updated message with the various subscribed clients.

4.4. Synchronization Service

The client communicates with the synchronization services either to receive the latest update from the cloud storage or to send the latest request/updates to the Cloud Storage.

- The synchronization service receives the request from the request queue of the messaging services and updates the metadata database with the latest changes.
- Also, the synchronization service broadcast the latest update to the other clients (if there are multiple clients) through the response queue so that the other client's indexer can fetch back the chunks from the cloud storage and recreate the files with the latest update.
- It also updates the local database with the information stored in the Metadata Database. If a client is not connected to the internet or offline for some time, it polls the system for new updates as soon as it goes online.

4.5. Cloud Storage

You can use any cloud storage service like Amazon S3 to store the chunks of the files uploaded by the user. The client communicates with the cloud storage for any action performed in the files/folders using the API provided by the cloud provider.

5. Database Design for Dropbox System Design

To understand Database design one should understand

- Each user must have at-least one device.
- Each device will have at-least one object (file or folder). Once user registers, we create a root folder for him/her making sure he/she has at-least one object.
- Each object may have chunks. Only files can have chunk, folders can't have chunks.

- Each object may be shared with one or multiple users. This mapping is maintained in AccessControlList.

We need the following tables to store our data:

5.1 Users

Users

```
{
  user_id(PK)
  name
  email
  password
  last_login_at
  created_at
  updated_at
}
```

5.2 Devices

Devices

```
{
  device_id(PK)
  user_id(FK)
  created_at
  updated_at
}
```

5.3 Objects

Objects

```
{
    object_id(PK)
    device_id(PK,FK)
    object_type
    parent_object_id
    name
    created_at
    updated_at
}
```

5.4 Chunks

Chunks

```
{
    chunks_id(PK)
    object_id(PK,FK)
    url
    created_at
    updated_at
}
```

5.5 AccessControlList

AccessControlList

```
{
    user_id(PK,FK1)
    object_id(PK,FK2)
    created_at
    update_at
}
```

6. API Design for Dropbox System Design

6.1 Download Chunk

This API would be used to download the chunk of a file.

Request

```
GET /api/v1/chunks/:chunk_id
X-API-Key: api_key
Authorization: auth_token
```

Response

```
200 OK
Content-Disposition: attachment; filename="<chunk_id>"
Content-Length: 4096000
```

The response will contain `Content-Disposition` header as `attachment` which will instruct the client to download the chunk. Note that `Content-Length` is set as `4096000` as each chunk is of 4 MB.

6.2 Upload Chunk

This API would be used to upload the chunk of a file.

Request

```
POST /api/v1/chunks/:chunk_id
X-API-Key: api_key
Authorization: auth_token
Content-Type: application/octet-stream
/path/to/chunk
```

Response

```
200 OK
```

6.3 Get Objects

This API would be used by clients to query Meta Service for new files/folders when they come online. Client will pass the maximum object id present locally and the unique device id.

Request

```
GET /api/v1/objects?local_object_id=<Max object_id present locally>&device_id=<Unique Dev
X-API-Key: api_key
Authorization: auth_token
```

Response

```
200 OK
{
  new_objects: [
    {
      object_id:
      object_type:
      name:
      chunk_ids: [
```

```
        chunk1,
        chunk2,
        chunk3
      ]
    }
  ]
}
```

Meta Service will check the database and return an array of objects containing name of object, object id, object type and an array of `chunk_ids`. Client calls the Download Chunk API with these `chunk_ids` to download the chunks and reconstruct the file.

7. Scalabilty for Dropbox System Design

- **Horizontal Scaling**
    - We can add more servers behind the load balancer to increase the capacity of each service. This is known as Horizontal Scaling and each service can be independently scaled horizontally in our design.
- **Database Sharding**
    - Metadata DB is sharded based on `object_id`. Our hash function will map each `object_id` to a random server where we can store the file/folder metadata. To query for a particular `object_id`, service can determine the database server using same hash function and query for data. This approach will distribute our database load to multiple servers making it scalable.
- **Cache Sharding**
    - Similar to Metadata DB Sharding, we are distributing the cache to multiple servers. In-fact Redis has out of box support for partitioning the data across multiple Redis instances. Usage of Consistent Hashing for distributing data across instances ensures that load is equally distributed if one instance goes away.

8. Conclusion

In conclusion, the design of the Dropbox system incorporates a well-thought-out architecture that seamlessly handles user file uploads, downloads, metadata management, and storage using a set of key components.

Learn how to come up with systems that are scalable, efficient, and robust—ones that impress. Ready to elevate your tech skills? Enrol now and build the future!

anuu...

41

Previous Article

System Design Netflix | A Complete Architecture

Next Article

Designing Content Delivery Network (CDN) | System Design

Similar Reads

Design a system that counts the number of clicks on YouTube videos | System...

Designing a Click Tracking System for YouTube Videos involves architecting a comprehensive and efficient solution to monitor and analyze user interactions with...

15+ min read

Design Restaurant Management System | System Design

In the modern restaurant industry, delivering exceptional dining experiences requires more than just good cuisine. Restaurant Management Systems have emerged as the...

15 min read

Design BookMyShow - A System Design Interview Question

It's really easy to search for your favorite movie in a theatre, check the seat availability, and, book the ticket on the BookMyShow app within just 5-10 minutes without much...

11 min read

Difference between System Design and System Architecture

When it comes to software development and engineering there can be some confusion, between the two terms "System Design" and "System Architecture." Although they are...

3 min read

System Development Life Cycle vs. System Design Life Cycle

The System Development Life Cycle (SDLC) and System Design Life Cycle (SDLC) are two distinct but interconnected processes involved in the development of information...

4 min read

View More Articles

Article Tags : GBlog System Design System-Design

NextJS
Bootstrap
Web Design

Computer Science

Operating Systems
Computer Network
Database Management System
Software Engineering
Digital Logic Design
Engineering Maths
Software Development
Software Testing

System Design

High Level Design
Low Level Design
UML Diagrams
Interview Guide
Design Patterns
OOAD
System Design Bootcamp
Interview Questions

School Subjects

Mathematics
Physics
Chemistry
Biology
Social Science
English Grammar
Commerce
World GK

Python Interview Question
Django

DevOps

Git
Linux
AWS
Docker
Kubernetes
Azure
GCP
DevOps Roadmap

Inteview Preparation

Competitive Programming
Top DS or Algo for CP
Company-Wise Recruitment Process
Company-Wise Preparation
Aptitude Preparation
Puzzles

GeeksforGeeks Videos

DSA
Python
Java
C++
Web Development
Data Science
CS Subjects