Dropbox HLD (High-Level Design) Document

✅ Objective

Design a scalable Dropbox-like file syncing and storage system that supports:

- Multi-device sync

- File upload/download

- Chunked storage and deduplication

- Offline editing

- Sharing with permission control

- High consistency (ACID) in metadata

Core Components

1. Client Service

- Detect file changes (using Monitor)

- Break files into 4MB chunks (Chunker)

- Upload only changed chunks (deduplication)

- Handle retries with exponential backoff

- Offline editing with local metadata DB

2. Upload/Download Service

- Interact with Chunk Storage

- Connect to Metadata Service to fetch file info

- Upload large files in parallel chunk streams

3. Metadata Service

- Maintains global file/chunk metadata

- ACID-compliant SQL database

- Handles sync with client metadata

4. Synchronization Service

- Ensures all devices are synced

- Handles conflict resolution

- Publishes file change events (e.g., via Kafka/SNS)

5. FilePermission DB

- Stores file access control data (READ/WRITE)

- Shares files between users securely

6. Chunk Storage

- Stores all file data in deduplicated 4MB chunks

- Chunk hash used to detect duplicates

- Stored in S3 or blob store

7. Notification Service

- Push updates to connected clients

- Can use Kafka / Redis Streams / SNS

8. Background Sync/Worker Service

- Async retries for failed uploads/downloads

- Handles queued jobs for offline users

9. API Gateway

- Single entry point for clients

- Handles routing, auth, rate-limiting

Cloud Services

Use Case | Recommended Service

---------------------|-------------------------------

File Storage | AWS S3 / GCP Cloud Storage

Metadata DB | AWS RDS (PostgreSQL)

Cache (Hot Chunks) | Redis (AWS ElastiCache)

Notifications | Kafka / AWS SNS / Redis Stream

Background Jobs | AWS SQS + Lambda/ECS worker

API Gateway | AWS API Gateway / NGINX

Monitoring/Alerts | CloudWatch / Prometheus

Auth Service | Auth0 / OAuth2 / Cognito

Database Schema (SQL)

Users

CREATE TABLE Users (

user\_id VARCHAR PRIMARY KEY,

email VARCHAR UNIQUE,

name VARCHAR,

created\_at TIMESTAMP

);

Files

CREATE TABLE Files (

file\_id VARCHAR PRIMARY KEY,

filename VARCHAR,

owner\_id VARCHAR REFERENCES Users(user\_id),

size BIGINT,

created\_at TIMESTAMP

);

Chunks

CREATE TABLE Chunks (

chunk\_id VARCHAR PRIMARY KEY,

file\_id VARCHAR REFERENCES Files(file\_id),

sequence\_number INT,

chunk\_hash VARCHAR,

size INT,

storage\_path TEXT

);

FilePermissions

CREATE TABLE FilePermissions (

file\_id VARCHAR REFERENCES Files(file\_id),

user\_id VARCHAR REFERENCES Users(user\_id),

permission ENUM('READ', 'WRITE', 'OWNER'),

PRIMARY KEY(file\_id, user\_id)

);

SyncEvents

CREATE TABLE SyncEvents (

event\_id SERIAL PRIMARY KEY,

file\_id VARCHAR,

user\_id VARCHAR,

event\_type VARCHAR,

timestamp TIMESTAMP

);

Performance/Scaling Considerations

- Use consistent hashing for Metadata DB sharding (by file\_id)

- Global LRU cache for hot chunks

- CDN (Cloudflare/CloudFront) for large file distribution

- Deduplication inline on client (bandwidth saving) or server (faster UX)

- Retry with exponential backoff for failed syncs

- Partition metadata DB: 50TB across 25 shards (2TB each)

Security

- Encrypt chunks at rest (AES-256)

- HTTPS for in-transit data

- Token-based auth (OAuth2)

- Audit logs for access history

Conclusion

This Dropbox design is modular, scalable, and aligns with real-world production architecture. It balances user experience (offline edits, fast sync) with backend performance (deduplication, async workers).

Perfect for HLD interview discussions 🔥