Handling Large Files - Quick Reference Notes

1. Why Naive File Uploads Fail

- Uploads entire file in one request → fragile and slow.
- High memory usage (buffers full file in RAM).
- Long uploads cause timeout failures.
- No resume support if connection drops.
- Server restarts = upload lost.
- Large uploads block server resources.

2. Chunked Uploads (Recommended Basic Pattern)

- Split file into fixed-size chunks (4MB–64MB typical).
- Each chunk uploaded independently.
- If one chunk fails → retry only that chunk.
- Supports resumable uploads.
- Allows parallel uploads for faster speed.

Chunk Size Guidelines

- 4–8 MB → Mobile networks (unstable connections).
- 16–64 MB → Desktop/stable connections.
- Adaptive chunk sizing is ideal.

3. Resumable Uploads

- Server tracks uploaded chunks.
- Client queries missing chunks after failure.
- Upload resumes from last successful chunk.
- Use checksums to make uploads idempotent (safe retries).

4. Pre-Signed URLs (Direct Upload to Storage)

- Client uploads directly to S3/GCS/Azure.
- Application server handles only metadata.
- Reduces server load and bandwidth costs.
- URLs expire quickly for security.
- Supports chunked direct uploads.

5. Multipart Upload (Cloud Storage Built-in)

- Initiate upload session.
- Upload parts independently (5MB–5GB).
- Upload parts in parallel (4–8 threads typical).
- Complete upload by assembling parts.
- Use lifecycle rules to clean incomplete uploads.

6. Streaming Uploads (When File Size Unknown)

- Used for live video/audio or dynamic data.
- Uses HTTP chunked transfer encoding.
- No resumability support.
- Avoid for normal file uploads.

7. Download Optimizations

- Use HTTP Range Requests (206 Partial Content).
- Supports resumable downloads.
- Supports parallel downloads.
- Required for video seeking.
- Use CDN for global distribution.
- Set headers: Cache-Control, ETag, Accept-Ranges.

8. Storage Architecture Best Practices

- Use Blob Storage for large files (S3/GCS).
- Store metadata separately in database.
- Scale metadata and storage independently.
- Keep strong consistency for metadata.

9. Content-Addressable Storage

- Use file hash as storage key.
- Enables automatic deduplication.
- Ensures file integrity.
- Supports safe concurrent uploads.

10. Compression & Deduplication

- Compress text/log files.
- Do NOT compress already-compressed formats (JPEG, MP4, ZIP).
- Block-level deduplication reduces storage.
- Content-Defined Chunking improves incremental sync.

11. Complete Production Upload Flow

- Client splits file into chunks.
- Server generates upload session + pre-signed URLs.
- Parallel chunk uploads to storage.
- Resume failed uploads.
- Server verifies all chunks.
- Create metadata record with content hash.

12. Implementation Checklist

- Chunked uploads.
- Resumable support.
- Pre-signed URLs.

- Checksum validation.
- Parallel upload support.
- Range request support.
- CDN integration.
- Garbage collection for orphaned uploads.
- Retry logic with exponential backoff.
- End-to-end integrity verification.

Core Principle

- Break large problems into smaller independent pieces.
- Use parallelism instead of sequential processing.
- Design for failure and resumability.
- Separate metadata from file data.