### **1. Pre-compression Before Upload**

* **For PDFs**: Use tools or libraries to optimize file size by removing unnecessary metadata, compressing embedded images, and adjusting compatibility levels.
* **For Images**: Use libraries like sharp or imagemin to resize images, reduce resolution, and lower quality to a reasonable level (e.g., compress to 70–80% quality for JPEGs). This minimizes file size while maintaining acceptable quality.

### **2. Streaming File Uploads**

* When working with large files, stream them directly to avoid overloading memory or causing timeouts. Use FormData on the frontend to send files, and server-side tools like multer or Busboy to process them as streams.
* Streaming ensures that even large files are handled in small chunks, improving scalability and performance.

### **3. Chunked Uploads**

* Divide large files into smaller, manageable chunks (e.g., 5MB each) and upload these chunks sequentially or in parallel.
* On the backend, reassemble the chunks into a complete file. This approach is especially useful for unstable networks or large files that may cause issues during a single upload.

### **4. Use Content-Encoding in Requests**

* Compress files before uploading them to the API Gateway using methods like gzip or Brotli.
* Add the Content-Encoding header to the request to let the server know the payload is compressed. This reduces the data sent over the network but requires the API to handle decompression.

### **5. Optimize API Gateway Settings**

* Enable Payload Compression in API Gateway for specific content types (e.g., PDFs, images) to reduce response sizes for compressed payloads.
* Pair API Gateway with CloudFront for better performance and to take advantage of caching and additional compression features.

### **6. Use S3 for Direct Uploads**

* Generate pre-signed URLs to allow users to upload files directly to Amazon S3 without routing them through the API Gateway. This reduces the load on the gateway and ensures faster uploads.
* After uploading, the app can send metadata (like the S3 file path or other identifiers) to the backend via the API Gateway.

### **7. Client-Side Lazy Loading and Compression**

* Use Next.js’s next/image to automatically optimize image loading. This supports responsive image sizes, lazy loading, and modern formats like WebP.
* Resize or compress images client-side before uploading, ensuring only optimized versions are sent to the server.