1. Arquitectura del Sistema

La arquitectura básica del sistema puede seguir un patrón de **Cliente-Servidor**, donde:

- **Clientes:** Son los usuarios o aplicaciones que interactúan con el servidor para subir o descargar archivos.
- **Servidor:** Es un servidor Node.js con Express que maneja las solicitudes HTTP, gestiona el almacenamiento de archivos y responde a las peticiones de los clientes.

Componentes principales:

- **Servidor HTTP (Express):** Se encarga de manejar las solicitudes HTTP (GET, POST, etc.).
- **Gestión de Archivos (Multer):** Biblioteca para la gestión de cargas de archivos.
- **Almacenamiento de Archivos:** Los archivos pueden ser almacenados localmente en el servidor, en un sistema de archivos distribuido (como NFS), o en un servicio de almacenamiento en la nube (como AWS S3).

2. Protocolos

- **HTTP/HTTPS:** Utilizado para la comunicación entre el cliente y el servidor. HTTPS es crucial para asegurar la transmisión de datos.
- **REST:** El servidor puede seguir un enfoque RESTful para las rutas, lo que facilita la integración y escalabilidad.

3. Librerías y Llamadas al Sistema

Librerías principales:

- **Express:** Marco de trabajo para Node.js que facilita la creación de servidores HTTP.
- **Multer:** Middleware para gestionar la carga de archivos. Utiliza llamadas al sistema para almacenar los archivos en el disco.
- **fs (File System):** Módulo de Node.js para interactuar con el sistema de archivos.
- **dotenv:** Para manejar variables de entorno (por ejemplo, para configuraciones de producción).
- **bcrypt o crypto:** Si necesitas manejar la encriptación de datos sensibles.

Llamadas al sistema:

- **fs.writeFile, fs.readFile:** Para escribir y leer archivos en el sistema de archivos.
- **fs.mkdir:** Para crear directorios (si se usa un almacenamiento local).

4. Escalabilidad

Para escalar el sistema, se pueden seguir varias estrategias:

Escalado Vertical:

- **Aumentar recursos del servidor:** Añadir más CPU, memoria o almacenamiento a la máquina.

Escalado Horizontal:

- **Despliegue en Múltiples Servidores:** Utilizar un balanceador de carga (por ejemplo, Nginx o un servicio en la nube) para distribuir las solicitudes entre múltiples instancias de tu servidor Node.js.
- **CDN (Content Delivery Network):** Para servir archivos estáticos y reducir la carga en el servidor principal.
- **Almacenamiento en la nube:** Utilizar servicios como AWS S3 para almacenar archivos, lo que permite escalar el almacenamiento independientemente del servidor.

5. Alta Disponibilidad

Para asegurar que el sistema esté disponible incluso en caso de fallos:

- **Balanceadores de Carga:** Utilizar un balanceador de carga para distribuir el tráfico entre varios servidores.
- **Clústeres de Servidores:** Implementar una configuración de clúster en Node.js utilizando el módulo cluster para aprovechar múltiples núcleos de CPU.
- **Replicación de Base de Datos:** Si se usa una base de datos, configurarla con replicación para que exista una copia de seguridad en caso de fallo.
- **Implementación de Backup:** Configurar copias de seguridad regulares para los archivos almacenados.