Arquivos, se conectaria ao servidors utilizando SDK deles mesmos, usando endpoint com nome da bucket e a chave.
do front ele faz a chamada pro endpoit, e envia duas coisas, o arquivo e o token do usuario, valida, ve quem fez, deu certo, dai manda pro sdk do minio, ele faz o upload, e o minio retorna com mensagem de sucesso. O backend envia uma resposta de sucesso (201 Created) de volta para o frontend antes para o banco de dados tbm, talvez com o ID do novo documento ({ "id": 1234 }).
O frontend recebe essa resposta e exibe uma mensagem para o usuário

Usando Kafka (Processo Assíncrono)

Do front, a chamada para o endpoint é a mesma: ele envia o arquivo e o token do usuário para o backend.

O backend valida o token e faz o upload para o MinIO: ele recebe o arquivo, vê quem fez a chamada e usa a SDK do MinIO para salvar o arquivo no bucket. Até aqui, tudo igual.

O backend cria uma "ordem de serviço" para o Kafka: em vez de falar com o banco de dados, o backend cria uma pequena mensagem em JSON com os detalhes do que aconteceu (usuario_id, nome_do_arquivo_no_minio, etc.).

O backend envia a mensagem para um "canal" do Kafka: ele publica essa mensagem em um tópico, como se fosse um "canal de tarefas pendentes".

O backend responde IMEDIATAMENTE para o frontend: logo após enviar a mensagem para o Kafka (o que é quase instantâneo), ele responde ao frontend com um 202 Accepted. A mensagem é: "Recebido! Já estou cuidando disso". O usuário vê a mensagem de sucesso na tela na mesma hora, sem esperar.

Nos bastidores, um "Processador" entra em ação: um programa separado (um "consumer") está sempre "escutando" esse canal do Kafka. Ele pega a "ordem de serviço" que o backend acabou de enviar.

O Processador salva no banco de dados: usando as informações da mensagem, esse programa se conecta ao banco de dados e salva os detalhes do arquivo (o ID, o nome, etc.).

download

Cenário 1: O Download Direto (O Mais Comum e Eficiente)
Este é o fluxo padrão, sem Kafka. É o jeito mais rápido e escalável de entregar o arquivo para o usuário.

No front, o usuário clica em "Baixar": A aplicação faz uma chamada GET para o endpoint do backend, por exemplo: GET /api/arquivos/1234, enviando o token do usuário para autorização.

O backend valida a permissão: Ele verifica o token e checa no banco de dados se o usuário atual tem permissão para baixar o arquivo de id: 1234.

O backend pede ao MinIO um "link temporário": Aqui está o truque. Em vez de baixar o arquivo para o servidor do backend e depois reenviá-lo para o usuário (o que gastaria o dobro de banda), o backend usa a SDK do MinIO para gerar uma URL pré-assinada. É um link de download seguro, que aponta diretamente para o arquivo no MinIO e expira depois de alguns minutos.

O backend envia o link para o frontend: O backend responde à chamada do frontend com esse link temporário.

O navegador do usuário faz o download direto do MinIO: O frontend recebe o link e instrui o navegador a usá-lo. O download pesado acontece direto entre o MinIO e o navegador do usuário.

excluir

Passos 1 e 2 são idênticos: O usuário clica em "Excluir", o backend valida a permissão.

O backend envia uma "ordem de exclusão" para o Kafka: Em vez de fazer a exclusão, ele publica uma mensagem no tópico arquivos-para-deletar, contendo o ID do arquivo e do usuário.

O backend responde IMEDIATAMENTE para o frontend: Ele retorna um 202 Accepted. O frontend já pode remover o arquivo da interface do usuário, dando a impressão de que a exclusão foi instantânea.

Nos bastidores, um "Processador de Exclusão" age: Um serviço "consumer" pega a mensagem do Kafka.

O Processador executa a exclusão completa: Ele usa a SDK do MinIO para deletar o arquivo do bucket e, em seguida, deleta o registro do banco de dados, exatamente como nos passos 4 e 5 do cenário síncrono.

renomear

Cenário 2: Renomear o Arquivo Físico no MinIO (O Jeito Difícil)
Isso só é necessário se a estrutura de pastas e nomes no MinIO for importante para a lógica de negócio.

O backend usa a SDK do MinIO para COPIAR o objeto: Ele executa um comando que copia o arquivo bucket/nome_antigo.pdf para bucket/nome_novo.pdf.

O backend usa a SDK para DELETAR o objeto original: Se a cópia foi bem-sucedida, ele deleta o bucket/nome_antigo.pdf.

O backend atualiza o banco de dados: Ele atualiza o registro para apontar para o nome_novo.pdf.

Este processo é mais lento e arriscado. Poderíamos usar o Kafka aqui da mesma forma que na exclusão: o backend publicaria uma mensagem renomear-arquivo, e um consumer faria a operação COPIAR + DELETAR + ATUALIZAR DB nos bastidores, dando uma resposta instantânea ao usuário.

mas o seu código precisa garantir que o seu banco de dados seja sempre um espelho fiel do que existe no MinIO.

1. Desempenho Desastroso em Escala
Listar objetos de um bucket pode ser uma operação lenta,se houver milhares arquivos. Cada vez que um usuário abrisse a página de arquivos, seu backend teria que fazer essa chamada pesada ao MinIO, esperar a resposta e só então devolvê-la.

Comparação: Uma query SELECT em uma tabela de banco de dados com um índice na

coluna usuario_id é executada em milissegundos, mesmo com milhões de registros. É para isso que os bancos de dados foram otimizados.

2. Falta de Informações Essenciais (Metadados)
Ela só contém informações técnicas sobre o objeto:

object_name (o nome do arquivo no storage, ex: a1b2c3d4.pdf)
size (o tamanho em bytes)
last_modified (a data da última modificação)

Essa lista NÃO INFORMA:

Qual o nome original que o usuário deu ao arquivo? (ex: "Relatório de Vendas - Final.pdf")
Quem fez o upload? (o usuario_id)
A qual projeto ou pasta o arquivo pertence?
Qual o status do arquivo? (ex: "Aprovado", "Pendente")
Uma descrição do arquivo?
Sem a sua tabela no banco de dados, você perde todo esse contexto que é vital para a aplicação.

3. Buscas e Filtros se Tornam um Pesadelo
Imagine que o usuário queira ver "apenas os arquivos PDF que ele enviou na última semana".

Com a abordagem direta do MinIO: Você teria que:

Listar TODOS os arquivos do bucket (milhares deles).
Trazer essa lista gigante para a memória do seu servidor backend.
Fazer um loop (for) em cima dessa lista no seu código (Python, Java, etc.) para filtrar manualmente cada item.
Isso é extremamente ineficiente, consome muita memória e CPU, e não escala de forma alguma.

Com a abordagem do banco de dados: Você simplesmente executa uma query SQL

ou usando o microserviço da jade:

URL Pré-Assinada (Pre-signed URL), gerado pelo seu microsserviço, que aponta diretamente para o arquivo no MinIO. acelera o processo.

```
sequence
    participant Usuário
    participant API Principal
    participant Broker (RabbitMQ/Kafka)
    participant Seu Microsserviço
    participant MinIO

    Usuário->>+API Principal: POST /upload (com arquivo)
    API Principal->>+Broker: Publica mensagem { "fileId": "123" }
    Broker-->>-Seu Microsserviço: Entrega mensagem { "fileId": "123" }
    activate Seu Microsserviço
    Seu Microsserviço->>Seu Microsserviço: Processa o arquivo (converte, etc.)
    Seu Microsserviço->>+MinIO: Salva arquivo processado (putObject)
    MinIO-->>-Seu Microsserviço: Confirmação de salvamento
    deactivate Seu Microsserviço
    API Principal-->>-Usuário: Resposta { "status": "Upload recebido,
```

```
processando em segundo plano" }
```