**Cenário 1**

O MFE faz uma requisição para o BFF que reponde para o MFE  
  
   
  
Para explicar como o processamento, vamos usar a analogia de um **Restaurante Inteligente**.

No nosso cenário, o **Front-end** é o cliente sentado à mesa, e a **API Rest** é toda a estrutura do restaurante (garçom, cozinha e registros).

**1. O Pedido (A Requisição HTTP)**

O Front-end (celular ou navegador) envia uma mensagem chamada **Requisição HTTP**. Pense nisso como o cliente levantando a mão e fazendo um pedido específico: *"Quero ver os dados do CPF 123"*.

**2. O Garçom (O Controller)**

Em Java, a primeira coisa que recebe o pedido é o **Controller**.

* **O que ele faz:** Ele verifica se o pedido faz sentido. Se você pedir "pizza" em uma "churrascaria", ele já te responde na hora que não é possível (Erro 400 - Bad Request).
* **Anotação de Auditoria:** Assim que o Controller aceita o pedido, o sistema "abre uma ficha" de atendimento. É aqui que o **OpenTelemetry** começa a contar o tempo e anotar quem é o operador.

**3. O Chef de Cozinha (O Service)**

O Controller passa o pedido para o **Service**. É aqui que a "mágica" e as regras de negócio acontecem.

* **O que ele faz:** Se o serviço precisa de dados que ele não tem, ele "telefona" para outros setores (Microserviços externos).
* **Exemplo:** Ele liga para o setor de FGTS e pergunta: *"O CPF 123 tem saldo?"*.
* **Auditoria no Meio:** Cada vez que o Chef liga para alguém, o "estagiário" (OpenTelemetry) anota na ficha: *"Liguei para o setor de FGTS às 10:05 e eles confirmaram o saldo"*.

**4. Preparando o Prato (A Resposta)**

Depois de coletar todas as informações, o Service organiza tudo em um formato que o cliente entende (geralmente um texto formatado chamado **JSON**).

**5. Entrega e Fechamento (Response e Event Hub)**

O Controller recebe o "prato pronto" do Service e entrega para o Front-end.

* **O Momento Crítico:** Quando o garçom entrega o prato e o cliente fica satisfeito, a "ficha de atendimento" (a trilha de auditoria) é fechada.
* **Envio para o Azure:** O sistema pega essa ficha completa — com o horário que começou, as ligações feitas e o horário que terminou — e envia para o **Azure Event Hub** (o cofre de arquivos do restaurante).

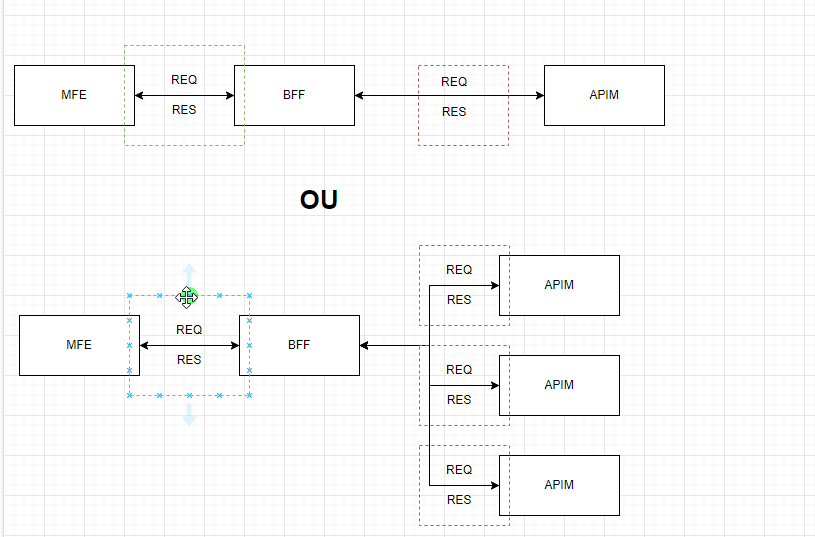
**🕵️ Onde e quando a mágica acontece no código?**

Para um programador Java, o controle disso é feito em três partes do código:

1. **Try (Tentar):** O código tenta executar o serviço. Se conseguir, no final ele grava: *"Sucesso!"*.
2. **Catch (Capturar Erro):** Se algo der errado (ex: o telefone do microserviço estava ocupado), o código pula para cá. O sistema anota: *"Erro: a cozinha não respondeu"*.
3. **Finally (Finalmente):** Independente de ter dado certo ou errado, este bloco é executado por último para garantir que a ficha de auditoria seja enviada para o Azure. Sem isso, o restaurante perderia o rastro do que aconteceu.

**Resumo do Fluxo para Iniciantes:**

1. **Front-end** chama o **Controller**.
2. **Controller** chama o **Service**.
3. **Service** (se precisar) chama outras **APIs**.
4. O **OpenTelemetry** anota cada passo silenciosamente.
5. O **Service** responde ao **Controller**.
6. O **Controller** responde ao **Front-end**.
7. **FIM:** O registro completo da "viagem" é despachado para o **Azure Event Hub**.

Cenário 2   
  
  
  
Para um iniciante em Java, entender o fluxo de uma API com auditoria pode parecer complexo, mas vamos quebrar isso em uma história lógica. Imagine que o seu sistema é como um **Banco Físico**.

Aqui está o passo a passo do que acontece quando o Front-end faz uma requisição:

**🏛️ 1. O Cenário: O Balcão do Banco**

1. **O Cliente (Front-end):** Chega no balcão e pede: *"Quero ver os contratos do CPF 123"*.
2. **O Atendente (Sua API REST/Controller):** Recebe o pedido. Ele não sabe as respostas de cabeça, então ele precisa consultar o arquivo central.
3. **O Registro (OpenTelemetry):** Assim que você começa a falar com o atendente, uma câmera de segurança (OTel) começa a gravar.

**🔄 2. O Fluxo Passo a Passo**

**Passo A: A Chegada (Request)**

O Front-end envia uma mensagem HTTP. O Java recebe isso no Controller.

* **Ação de Auditoria:** O sistema cria um "envelope" chamado **Span**. Dentro dele, ele anota: *"Início do atendimento para o CPF 123"*.

**Passo B: A Consulta (Service + Feign)**

O seu código (Service) chama um Microserviço externo usando uma ferramenta chamada **Feign**. É como se o atendente usasse um rádio para falar com a central.

* **Ação de Auditoria:** O "estagiário" (OpenTelemetry) anota no envelope: *"Pedindo dados para a Central de Contratos agora"*.

**Passo C: A Resposta da Central**

A central responde com os dados. O seu Java pega esses dados, organiza e prepara para entregar ao cliente.

* **Ação de Auditoria:** O sistema anota: *"Central respondeu com sucesso. Encontramos 2 contratos"*.

**Passo D: A Entrega e o Registro Final (Event Hub)**

O Java envia a resposta (JSON) de volta para o Front-end.

* **Ação de Auditoria:** O envelope (Span) é fechado. Agora, esse registro completo é enviado para o **Azure Event Hub**. É como se, ao fim do dia, a gravação da câmera fosse enviada para um cofre digital seguro na nuvem.

**📊 3. Diagrama do Fluxo de Informação**

**🕵️ 4. Em que momento a trilha é gravada?**

Essa é a dúvida mais comum! A trilha é **construída** durante todo o processo, mas ela é **enviada (gravada definitivamente)** em dois momentos principais:

1. **No Sucesso:** Quando o método termina e o return é executado. O sistema entende que a "viagem" acabou e despacha os dados para o Event Hub.
2. **No Erro (Try-Catch):** Se o microserviço cair ou o CPF for inválido, o Java entra no bloco catch. O OpenTelemetry captura o erro e envia a trilha imediatamente avisando: *"Atenção, a operação falhou no passo B pelo motivo X"*.

**Cenário 3... Complicando um pouco**

Num fluxo onde o **front-end chama uma API** que, por sua vez, **consulta 3 APIs**, **consolida os dados**, faz um **PUT em outra API** e **reconsulta as 3 APIs**, a auditoria precisa ser **centralizada, consistente e rastreável** ponta a ponta.

Para explicar esse fluxo mais complexo, imagine que o seu **BFF** é um **Organizador de Casamentos**. Ele precisa falar com vários fornecedores (APIs), tomar uma decisão importante e depois conferir se tudo mudou conforme o esperado.

Nesse cenário, a auditoria não é apenas um "carimbo" no fim; ela é um **diário de bordo** que registra cada passo crítico para que, se algo der errado no meio, você saiba exatamente onde parou.

## 🧭 O Fluxo da Operação (Orquestração)

O diagrama abaixo mostra a sequência de passos que o seu código Java executará:

### 1. Início: Abertura do Envelope (Request do Front)

* **Quando:** Assim que o Controller recebe a chamada do Frontend.
* **Onde:** Na camada de **Segurança/Filtro**.
* **O que é gravado:** O início da jornada. Registramos o "Intuito": "Operador X solicitou a consolidação de dados para o CPF Y". Criamos aqui o **Root Span** (o envelope principal da auditoria).

### 2. O Meio - Parte 1: Coleta de Dados (3 APIs)

* **Quando:** Durante a execução do Service, antes e depois de chamar cada uma das 3 APIs.
* **Onde:** Na camada de **Integração** (Feign Clients).
* **O que é gravado:** Registramos que buscamos dados nos fornecedores A, B e C. Se uma delas falhar, a auditoria já mostra: "Consegui os dados de A e B, mas C está fora do ar".

### 3. O Ponto Crítico: A Transformação (PUT API)

* **Quando:** **Imediatamente antes e imediatamente após** o comando PUT.
* **Onde:** Na camada de **Aplicação/Service**.
* **O que é gravado:** Este é o evento mais importante. Gravamos: "Dados consolidados com sucesso. Enviando atualização para o sistema central". Após o retorno, gravamos: "Sistema central atualizado".

### 4. O Meio - Parte 2: A Conferência (Re-consulta das 3 APIs)

* **Quando:** Após o sucesso do PUT.
* **Onde:** Novamente na camada de **Integração**.
* **O que é gravado:** Registramos a nova foto dos dados: "Após a atualização, os saldos nas APIs A, B e C agora são estes...". Isso prova que o PUT realmente funcionou no ecossistema completo.

### 5. O Término: Fechamento e Envio (Response para o Front)

* **Quando:** No momento do return final do Service para o Controller.
* **Onde:** Na saída da camada de **Aplicação**.
* **O que é gravado:** O resultado final consolidado e o tempo total de toda essa maratona. O OpenTelemetry fecha o "envelope" e despacha tudo para o **Azure Event Hub**.

## 🎓 Explicação: "O Diário de Bordo"

Para um programador Java iniciante, pense no **OpenTelemetry** como um diário que escreve sozinho conforme você caminha pelo código:

1. **Abertura do Diário:** Você entra no método. O Java cria um "ID de Rastreio".
2. **Anotações de Percurso:** Cada vez que você faz um client.chamarApi(), o diário anota: "Fui ali buscar tal coisa e voltei em 200ms".
3. **Anotação de Segurança:** Antes de fazer o PUT (que altera o dinheiro ou o contrato do cliente), você faz uma anotação especial: "Atenção: vou alterar o contrato agora!".
4. **Fechamento e Envio:** Quando o método chega no fim (ou se der um erro e cair no catch), o diário é fechado e enviado por um "mensageiro" (o OTel Collector) até a "central de arquivos" (Azure Event Hub).

Perguntas :Esse envio ao event hub será sempre no final ou catch   
  
A resposta curta é: **A informação é "fechada" no sucesso ou no erro (catch), mas o envio real para o Event Hub costuma ser feito em "lotes" logo em seguida.**

Para entender bem, precisamos diferenciar dois conceitos: **Gravar (notar o que aconteceu)** e **Enviar (despachar a mensagem para fora do app)**.

### 1. O Fluxo de Gravação (Dentro do Java)

No Java, utilizamos um bloco chamado try-catch-finally. Imagine que o OpenTelemetry é um estagiário com uma prancheta ao seu lado enquanto você trabalha:

1. **Início (try):** O estagiário abre uma nova folha (cria um *Span*) e anota a hora de início.
2. **Durante a execução:** Sempre que você chama uma das 3 APIs ou faz o PUT, você diz ao estagiário: "Ei, anota aí que eu acabei de consultar a API 1". Ele escreve isso na folha.
3. **No Sucesso:** Se tudo der certo até o final (após a re-consulta das 3 APIs), você diz: "Terminei com sucesso!". Ele fecha a folha.
4. **No Erro (catch):** Se algo explodir no meio (ex: a API de PUT deu erro), o código pula para o catch. Lá, você diz: "Anota que deu erro e o motivo foi X". Ele marca a folha com um carimbo vermelho de erro.
5. **O Fim (finally ou término do método):** É aqui que a "folha" (o Span) é considerada **pronta**.

### 2. O Fluxo de Envio (Para o Event Hub)

Agora, o estagiário não corre até o correio (Event Hub) para cada linha que ele escreve. Isso deixaria o restaurante (seu app) muito lento.

* **O Coletor (OTel Collector):** O OpenTelemetry guarda as folhas prontas em uma "caixinha de saída" dentro da memória do seu app.
* **Envio em Lote (Batch):** De tempos em tempos (ex: a cada 5 segundos ou a cada 100 folhas), o sistema pega esse monte de folhas e envia todas de uma vez para o **Azure Event Hub**.

### 3. Onde os eventos são gravados nesse fluxo complexo?

Conforme a sua especificação de orquestração (3 APIs -> Consolida -> PUT -> 3 APIs), os registros ocorrem em "camadas":

1. **O Evento Pai (Audit Trail Principal):** Começa no início de tudo e só termina no final de toda a jornada. Ele garante a trilha de auditoria completa exigida.
2. **Os Eventos Filhos (Sub-passos):**