# **Fundamentos do Streaming de Dados**

## **Módulo 5: Processamento de Dados em Tempo-Real**

#### 

### **Introdução**

Mais de 90% dos dados gerados globalmente são efêmeros e perdem seu valor analítico em minutos. Ainda assim, a maioria das empresas continua usando pipelines batch para processá-los.

Compreender o paradigma de dados em fluxo é essencial para construir soluções responsivas, escaláveis e orientadas a eventos. Streaming não é apenas uma alternativa ao batch: é uma mudança de mentalidade.

O UberEats depende de eventos em tempo real para atualizar localização de entregadores, estimar tempo de entrega, ajustar rotas e gerenciar promoções dinâmicas. Sem streaming, isso não é possível em escala.

**Objetivos de Aprendizado:**

* Entender a diferença entre dados em repouso e em fluxo
* Identificar limitações do modelo batch
* Introduzir os pilares do modelo streaming
* Relacionar a arquitetura moderna de dados com streaming

### **Fundamentos**

**Evolução Histórica:**

* Batch (Hadoop): Processamentos massivos, mas lentos.
* Lambda Architecture: tentativa de unificar batch + streaming
* Kappa Architecture: simplificação com foco exclusivo em eventos

**Problemas que o Streaming resolve:**

* Latência em decisões de negócio
* Sistemas que precisam de dados "frescos"
* Experiências digitais em tempo real

**Arquitetura e Componentes Centrais:**

* Fonte de dados (eventos)
* Sistema de ingestão (Kafka, Pulsar)
* Engine de processamento (Flink, Spark Streaming, Bytewax)
* Sinks (bancos, dashboards, microserviços)

**Princípios de Design:**

* Imutabilidade de eventos
* Processamento orientado a tempo
* Escalabilidade horizontal
* Tolerância a falhas

### **Tabela Comparativa: Batch vs Streaming**

| **Característica** | **Batch** | **Streaming Contínuo** |
| --- | --- | --- |
| Latência | Alta (minutos ou horas) | Baixa (milissegundos a segundos) |
| Volume de dados | Grande volume processado de uma vez | Eventos individuais ou em pequenos lotes |
| Frequência de processamento | Agendada (ex: diariamente) | Contínua, em tempo real |
| Arquitetura | Simples, mas menos responsiva | Complexa, mas responsiva |
| Casos de uso | Relatórios, BI estático | Monitoramento, alertas, automação |

#### 

#### 

#### 

#### 

### **Aprofundamento Técnico**

**Detalhamento técnico:**

* Comparativo: Batch x Micro-batch x Streaming Contínuo

| **Tipo de Processamento** | **Definição** | **Exemplos** |
| --- | --- | --- |
| Batch | Processa grandes volumes de dados de uma vez | Hadoop, Airflow |
| Micro-batch | Pequenos lotes em intervalos regulares | Spark Structured Streaming |
| Streaming Contínuo | Evento a evento, quase instantaneamente | Apache Flink, Bytewax |

**Semântica de entrega:**

| **Tipo de entrega** | **Comportamento** | **Risco** |
| --- | --- | --- |
| At-most-once | Evento processado no máximo uma vez | Perda de eventos |
| At-least-once | Evento pode ser processado mais de uma vez | Duplicidade |
| Exactly-once | Evento processado uma única vez, garantidamente | Maior complexidade |

**Trade-offs Relevantes:**

* Consistência vs Latência
* Precisão vs Desempenho

**Métricas e Benchmarks:**

* Throughput (eventos/s)
* Latência end-to-end
* Time to insight

**Casos de uso e contra-indicações:**

* Casos: fraude em tempo real, monitoramento de IoT, log analysis
* Contra-indicações: workloads esporádicos e não críticos

### **Implementação Prática (Conceitual)**

**Diagramas Arquiteturais:**

* Pipeline clássico: fonte > broker > processamento > sink
* Arquitetura Kappa com replay de eventos

**Decisões de Design:**

* Escolha do engine conforme caso de uso
* Garantias de entrega conforme criticidade

**Padrões e Anti-padrões:**

* Padrão: event sourcing
* Anti-padrão: usar streaming apenas como "mini-batch"

**Checklist de Validação:**

* Eventos bem definidos e imutáveis?
* Volume e velocidade justificam streaming?
* Infraestrutura suportar latência baixa?

#### 

### **Otimização e Performance**

**Técnicas de Otimização:**

* Backpressure management
* Batching em pontos estratégicos

**Monitoramento:**

* Lag de consumo
* Erros de entrega e reprocessamento

**Best Practices:**

* Versionamento de eventos
* Isolamento entre fontes e consumidores

#### 

#### 

#### 

### **Integração e Ecossistema**

**Relacionamento com outras tecnologias do módulo:**

* Kafka (ingestão)
* Spark/Flink (processamento)
* Trino/Pinot (consulta em tempo real)

**Dependências e Pré-requisitos:**

* Conhecimento de arquitetura distribuída
* Familiaridade com logs/eventos

**Impacto Downstream:**

* Sistemas analíticos passam a ser mais responsivos
* Habilita acionamento de alertas, execuções e automações

**Roadmap Futuro:**

* Processamento incremental
* Serverless streaming (ex: Cloud Run, AWS Lambda + Kafka)

#### 

### **Casos Avançados**

**Cenários Complexos:**

* Joins entre streams com estados grandes
* Gerenciamento de estado distribuído

**Edge Cases:**

* Eventos duplicados
* Reordenação por delays de rede

**Escalabilidade:**

* Repartição (partitioning)
* Checkpoints para recuperação

### **Conclusão e Reflexão**

Streaming é um novo paradigma orientado a eventos, que exige mudança de mentalidade e arquitetura. Ele resolve problemas reais de latência e atualização constante.

**Validação dos Objetivos:**

* Compreendeu o papel e a evolução do streaming
* Diferenciou batch de streaming com clareza
* Visualizou aplicações práticas como UberEats