Data Engineering

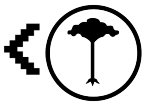
Roadmap

- Introdução
- Cloud Functions
- Beam/Dataflow
- Kafka
- Elasticsearch
- BigTable

Antes de começar

- Instalar git:
  - apt install git
- Instalar python 3:
  - apt install python3
- Instalar gcloud:
  - apt install curl
  - curl https://sdk.cloud.google.com | bash
  - exec -l $SHELL
  - gcloud init
    - us-central1-a

Cloud Function

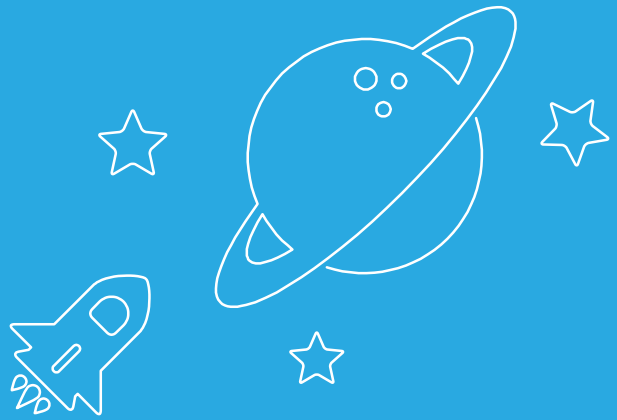


Function as a Service

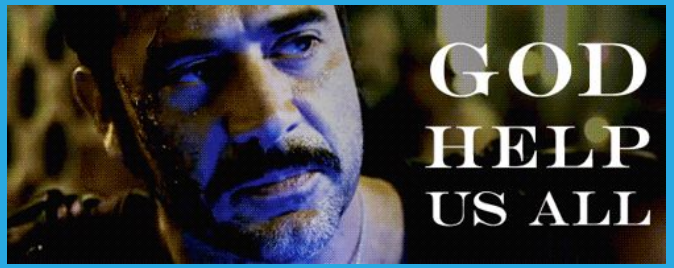
- Serverless
- Triggers
  - Storage
  - PubSub
  - HTTP
- Várias linguagens:
  - JS
  - Python
  - Go

Function as a Service: prática

- Pros:
  - Fácil deploy
  - Escalável
- Cons:
  - Barato



Demo



Beam

- Dataflow model
  - Akidau, 2015
  - “Modern data processing is a complex and exciting field.”
  - Junta dois papers:
    - FlumeJava
    - MillWheel
  - Batchs existentes
    - Alta latência
  - Streaming existentes
    - Tolerância a falhas, escalabilidade, latência
    - Complexidade de janelamento

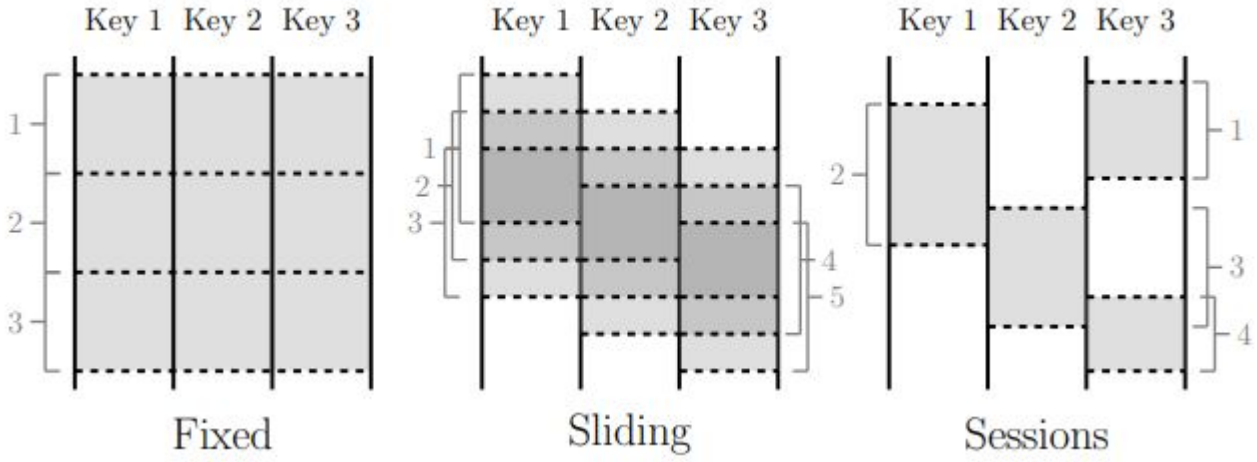
- Dataflow model
  - Propõe um modelo de programação simples para processamento de dados
    - Event time
    - Processing time

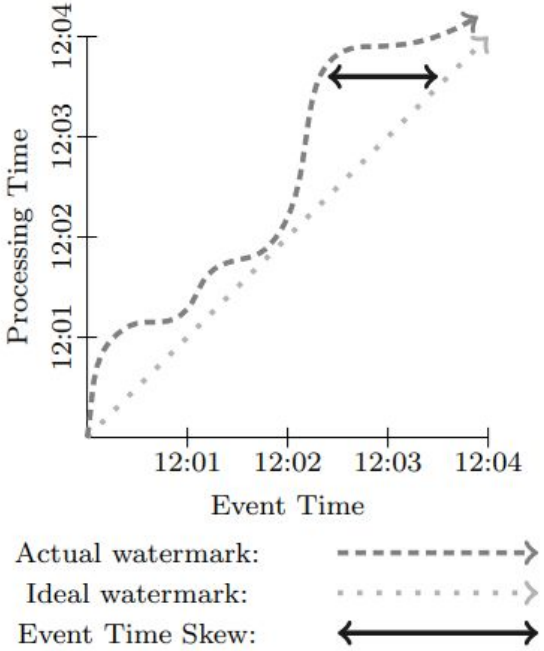
- Dataflow model
  - Divide pipelines em 4 dimensões
    - **Quais** resultados computados?
    - **Onde** serão computados? (em event time)
    - **Quando** serão materializados? (em processing time)
    - **Como** refinar dados recentes mais tarde?

- Streaming vs Batch
  - Streaming -> Unbounded
  - Batch -> Bounded
  - Unbounded: processado em batch systems
  - Streaming systems: capazes de processar batches

- Windows
  - Não é só um sistema operacional
  - Divisão do dataset para processamento
  - Aligned: se aplica a todo dataset
  - Unaligned: se aplica a um subset

Windows





- Watermark
  - Limite inferior de todos os event times processados pelo pipeline

- Primitivas
  - ParDo

$$(fix, 1), (fit, 2)$$

$$\downarrow \; ParDo(\; ExpandPrefixes)$$

$$(f, 1), (fi, 1), (fix, 1), (f, 2), (fi, 2), (fit, 2)$$

  - GroupByKey

$$(f, 1), (fi, 1), (fix, 1), (f, 2), (fi, 2), (fit, 2)$$

$$\downarrow \; GroupByKey$$

$$(f, [1, 2]), (fi, [1, 2]), (fix, [1]), (fit, [2])$$

- AssignWindows
- MergeWindows
- **GroupByKeyAndWindow**

$(k_1, v_1, 13{:}02, [0, \infty)),$
$(k_2, v_2, 13{:}14, [0, \infty)),$
$(k_1, v_3, 13{:}57, [0, \infty)),$
$(k_1, v_4, 13{:}20, [0, \infty))$

$\downarrow$ *AssignWindows(Sessions(30m))*

$(k_1, v_1, 13{:}02, [13{:}02, 13{:}32)),$
$(k_2, v_2, 13{:}14, [13{:}14, 13{:}44)),$
$(k_1, v_3, 13{:}57, [13{:}57, 14{:}27)),$
$(k_1, v_4, 13{:}20, [13{:}20, 13{:}50))$

$\downarrow$ *DropTimestamps*

$(k_1, v_1, [13{:}02, 13{:}32)),$
$(k_2, v_2, [13{:}14, 13{:}44)),$
$(k_1, v_3, [13{:}57, 14{:}27)),$
$(k_1, v_4, [13{:}20, 13{:}50))$

$\downarrow$ *GroupByKey*

$(k_1, [(v_1, [13{:}02, 13{:}32)),$
$(v_3, [13{:}57, 14{:}27)),$
$(v_4, [13{:}20, 13{:}50))]),$
$(k_2, [(v_2, [13{:}14, 13{:}44))])$

$\downarrow$ *MergeWindows(Sessions(30m))*

$(k_1, [(v_1, [\mathbf{13{:}02}, \mathbf{13{:}50})),$
$(v_3, [13{:}57, 14{:}27)),$
$(v_4, [\mathbf{13{:}02}, \mathbf{13{:}50}))]),$
$(k_2, [(v_2, [13{:}14, 13{:}44))])$

$\downarrow$ *GroupAlsoByWindow*

$(k_1, [([\mathbf{v_1}, \mathbf{v_4}], [13{:}02, 13{:}50)),$
$([\mathbf{v_3}], [13{:}57, 14{:}27))]),$
$(k_2, [([\mathbf{v_2}], [13{:}14, 13{:}44))])$

$\downarrow$ *ExpandToElements*

$(k_1, [v_1, v_4], \mathbf{13{:}50}, [13{:}02, 13{:}50)),$
$(k_1, [v_3], \mathbf{14{:}27}, [13{:}57, 14{:}27)),$
$(k_2, [v_2], \mathbf{13{:}44}, [13{:}14, 13{:}44))$

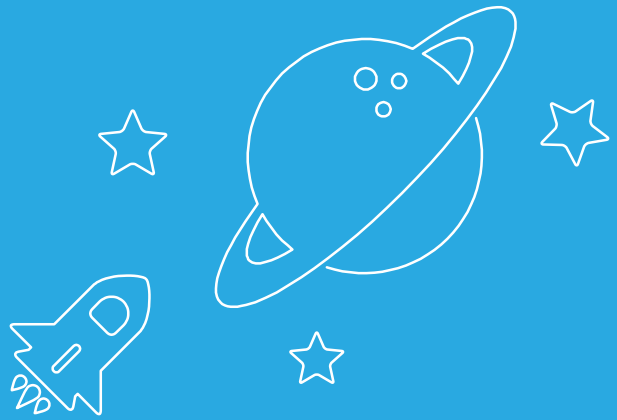
- Triggers
  - Determina quando um GroupByKeyAndWindow ocorre
  - **Window**: determina **onde** os dados serão agrupados (**event time**)
  - **Trigger**: determina **quando** os resultados dos agrupamentos serão emitidos (**processing time**)

- Triggers
  - Late data: atraso no event time em relação ao processing time
  - Refinamentos:
    - Descarte: resultados futuros **não dependem** de dados passados
    - Acúmulo: resultados futuros **dependem** de dados passados
    - Acúmulo + Retração:
      - resultados futuros **dependem** de dados passados
      - resultados passados **dependem** de dados futuros

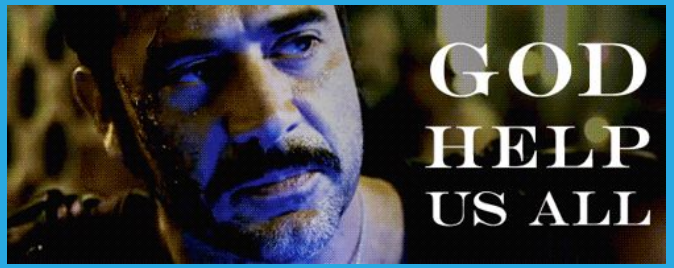
- APIs
  - Java
  - Scala (Scio)
  - Python
  - Go (experimental)

- Runners
  - https://beam.apache.org/documentation/runners/capability-matrix/

- Google Dataflow
  - Um dos runners do beam
  - Gerenciado
  - Escalável
  - Templates



Demo



Kafka

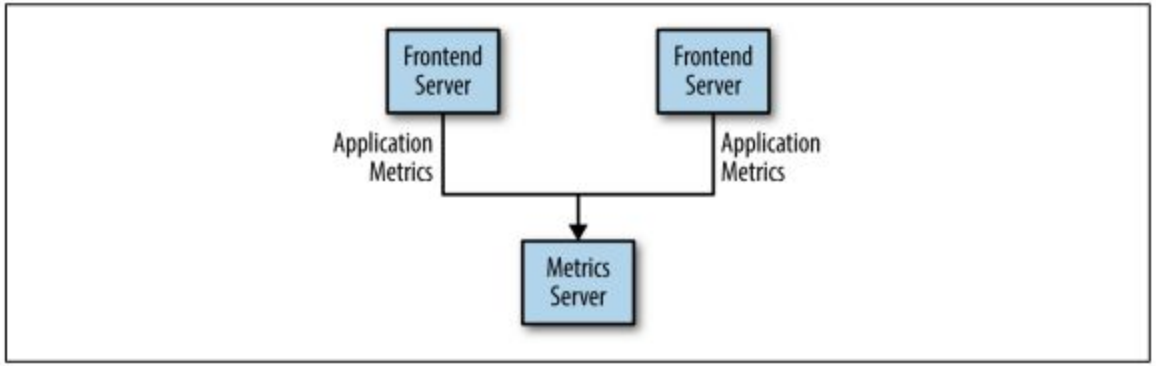


Figure 1-1. A single, direct metrics publisher

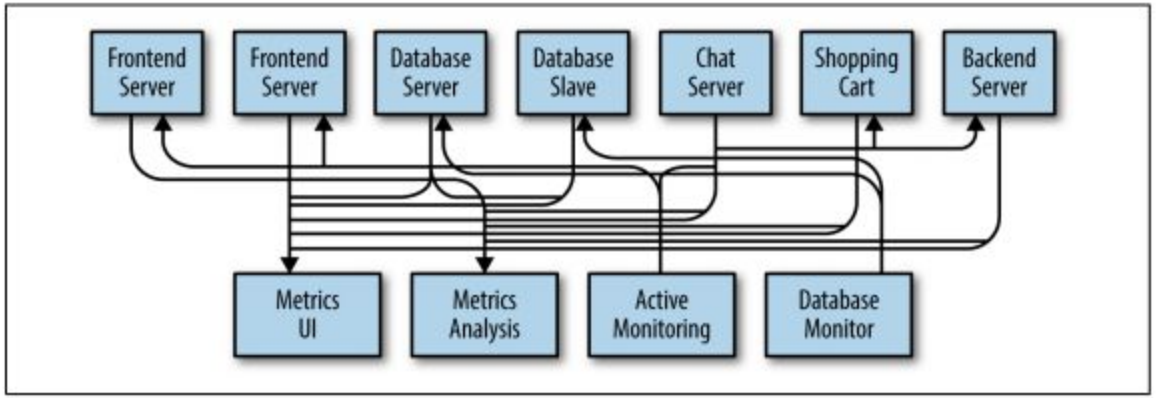


Figure 1-2. Many metrics publishers, using direct connections

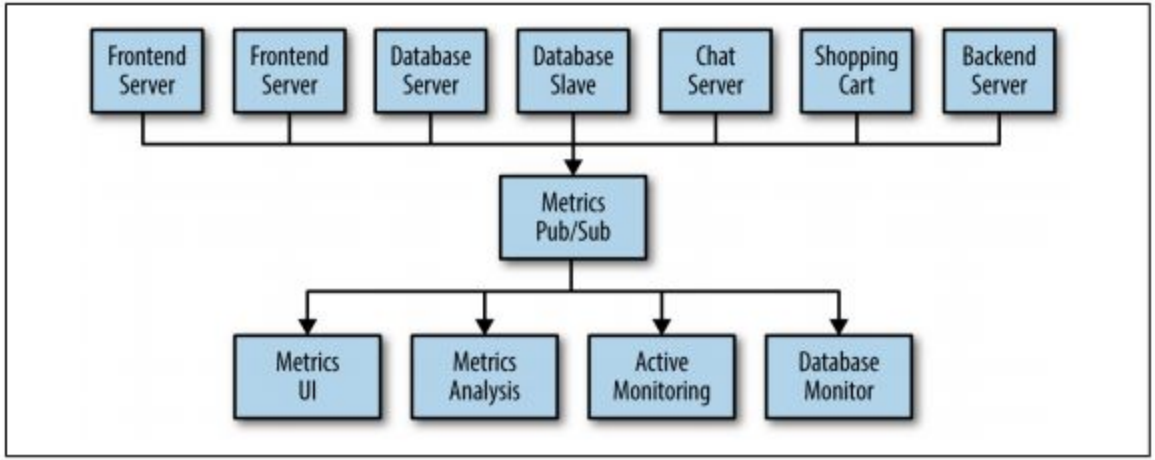


Figure 1-3. A metrics publish/subscribe system

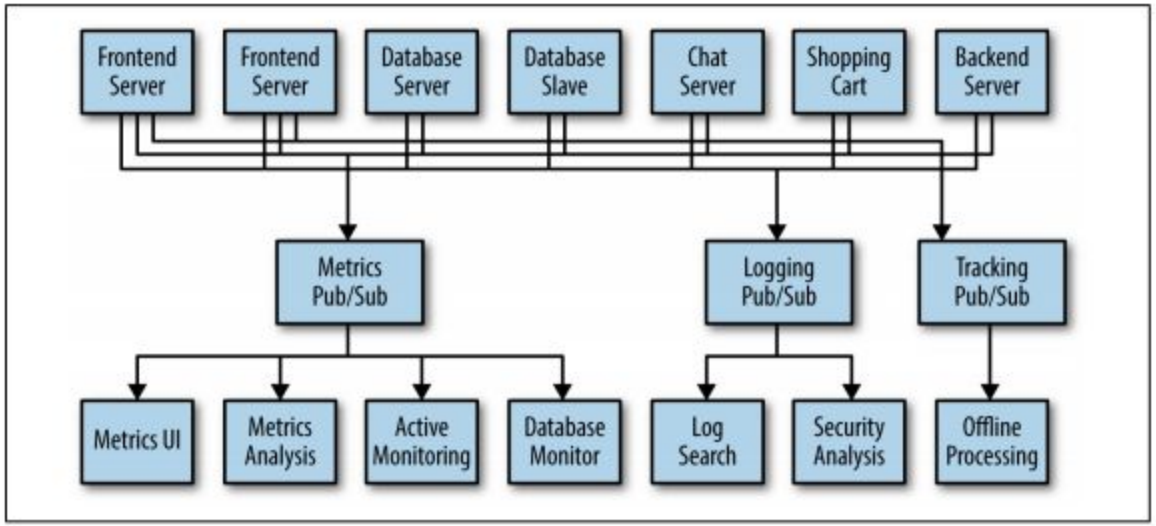


Figure 1-4. Multiple publish/subscribe systems

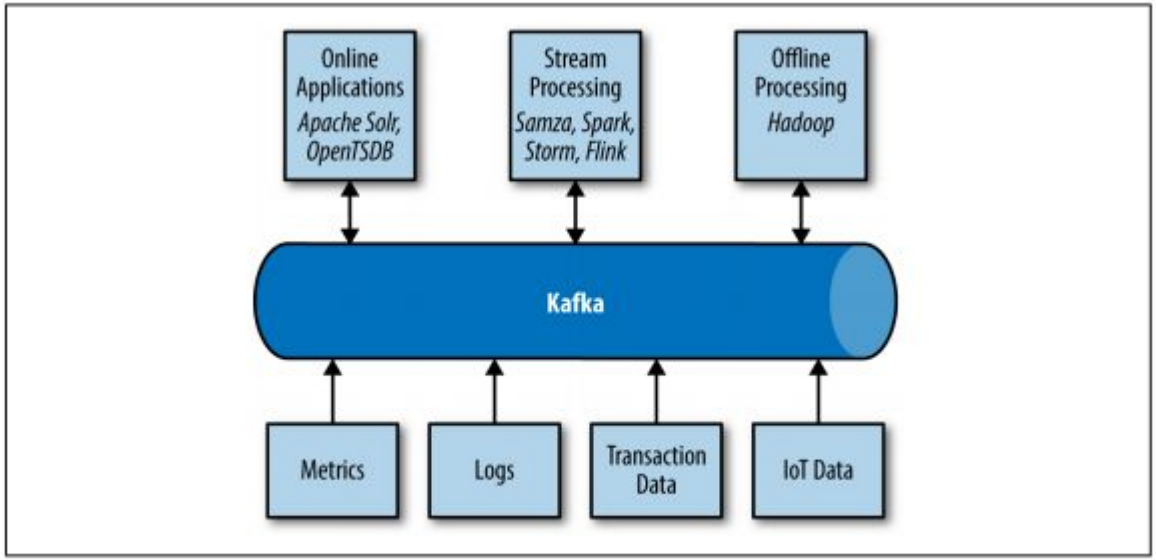


Figure 1-9. A big data ecosystem

- Por que Kafka?
  - Retenção
  - Escalabilidade
  - Ordem

- Por que Kafka?
  - Reliable Data Delivery
    - Dados de várias criticidades
  - Garantias
    - Ordem
    - Commits
      - Uma vez commitado, dado não será mais perdido
      - Somente serão lidos dados commitados

- Por que não Kafka?
  - Reliable Data Delivery
    - Acaba gerando muitas arquiteturas ruins
  - Garantias
    - Depende de uma série de configs

- Plataforma distribuída de streaming
  - O que é streaming?
    - Unbounded == infinite
  - Streams
    - Ordenados
    - Imutáveis
    - Repetíveis (replayable)

- Plataforma distribuída de streaming
  - Request-response: um sistema espera outro
  - Batch: dados processados de tempos em tempos
  - Streaming
    - Não é necessário espera
    - Processamento contínuo

- Streaming: conceitos
  - Tempo
    - Event time
    - Processing time
    - **Log append time (tempo no Kafka)**
  - Estado
    - Interno: acessado apenas por um sistema
    - Externo: disponível para outros sistemas

- Streaming: conceitos
  - Dualidade tabela-stream
    - Table: coleção de registros
    - Stream: cadeias de eventos que modificam algo
  - Table != Stream
  - Tabela pode ser convertida em stream (CDC)
  - Stream pode ser convertida em tabela (materialização)
  - Diferentes tipos de redundância

Modelos de streaming: Single-event processing

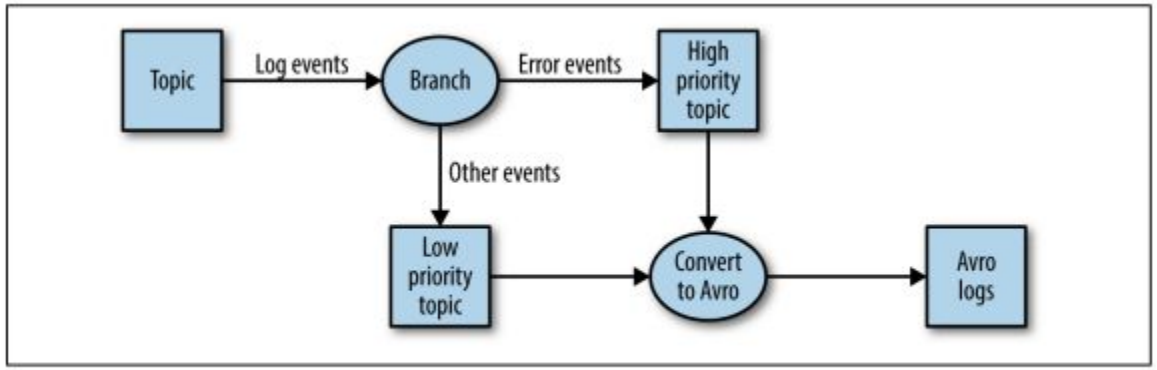


Figure 11-3. Single-event processing topology

Modelos de streaming: Local state

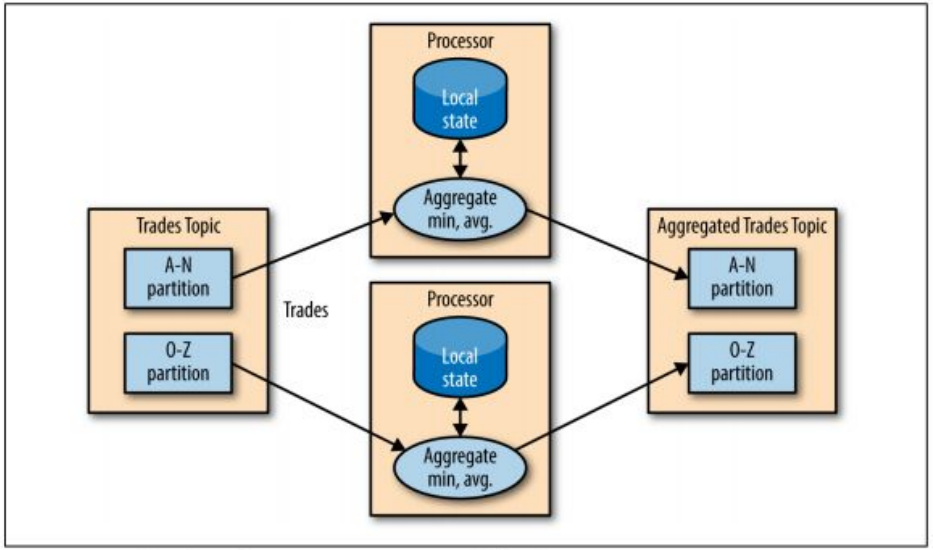


Figure 11-4. Topology for event processing with local state

- Modelos de streaming: Local state
- Problemas:
  - Estado deve caber em memória
  - Persistência
  - Rebalanceamento

Modelos de streaming: Multiphase

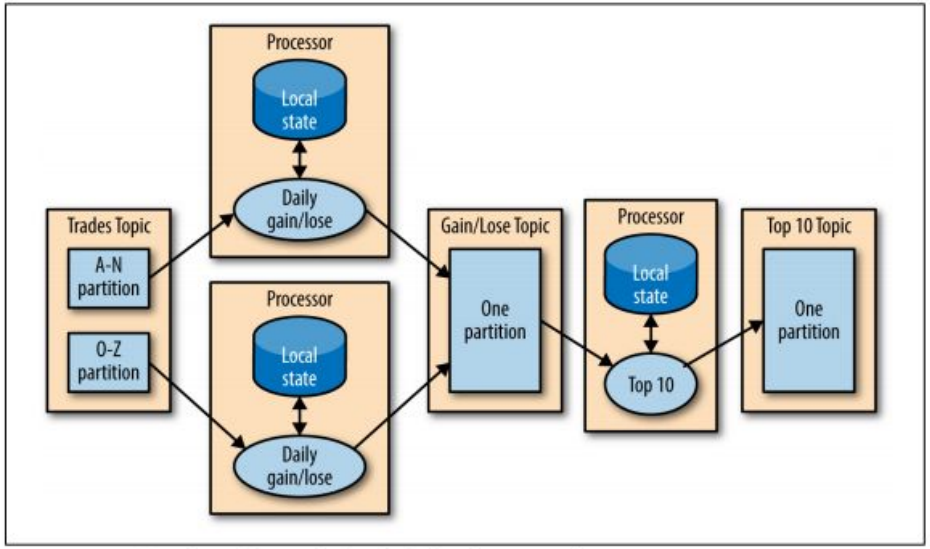


Figure 11-5. Topology that includes both local state and repartitioning steps

- Modelos de streaming: Stream-table Join

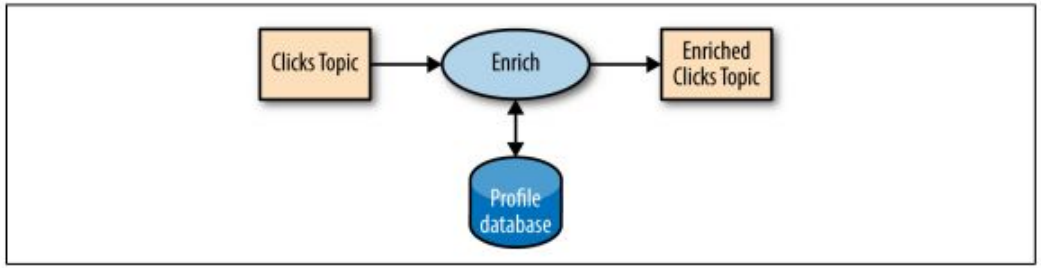


Figure 11-6. Stream processing that includes an external data source

- Modelos de streaming: Stream-table Join
- Problemas:
  - Latência
  - Escalabilidade

Modelos de streaming: Stream-table Join

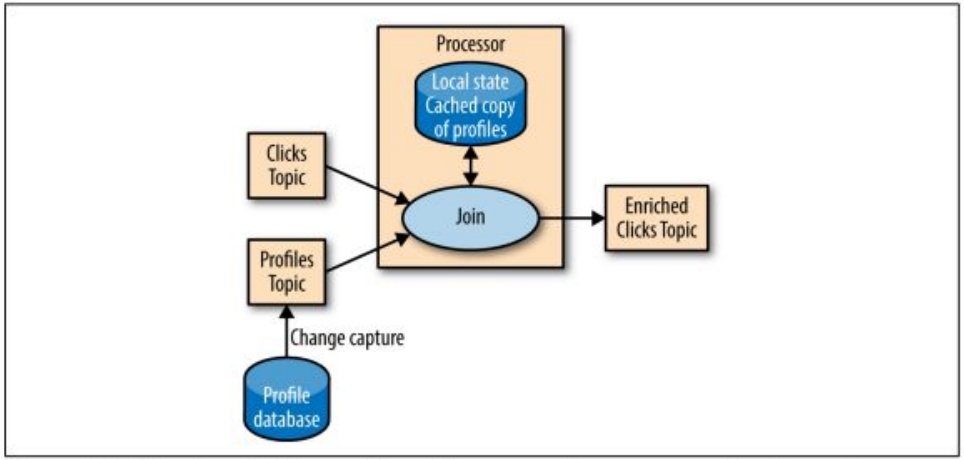
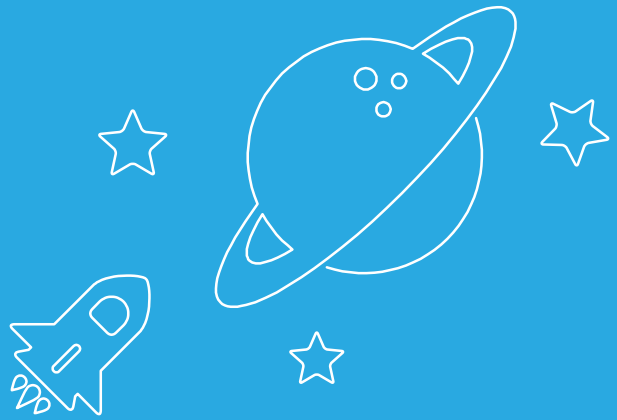
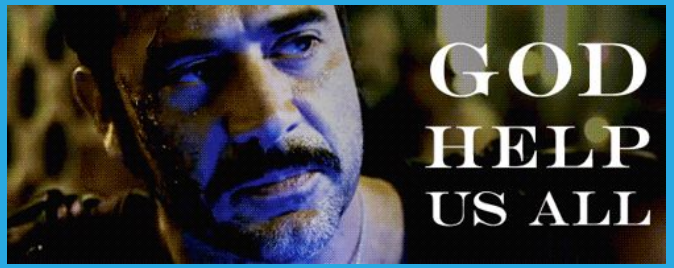


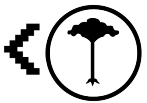
Figure 11-7. Topology joining a table and a stream of events, removing the need to involve an external data source in stream processing



Demo



Elasticsearch



Motor de busca

- Motor de busca e análise distribuído
- Construído sobre Apache Lucene
- Usos:
  - Busca
  - Logs (ELK)

Motor de busca

- Rápido
- Distribuído
- Ecossistema rico
- Simples

ELK

- Elastic Stack:
  - Logstash / Beats
  - Elasticsearch
  - Kibana

Elasticsearch - conceitos

- Near Realtime
  - Indexação -> Busca: ~1sec

Elasticsearch - conceitos

- Índices
  - Coleção de documentos
  - Índice invertido

Elasticsearch - conceitos

- Node
  - Máquina pertence a um cluster

Elasticsearch - conceitos

- Documentos
  - Unidade básica
  - JSON

Elasticsearch - conceitos

- Shards
  - Subdivisão física do índice
  - Escalabilidade
- Réplicas
  - Cópias de shards em nós
  - Disponibilidade
  - Escalabilidade

- Ciclo de vida
  - Ingestão
  - Indexação
  - Busca
  - Visualização

BigTable

NoSQL Gerenciado

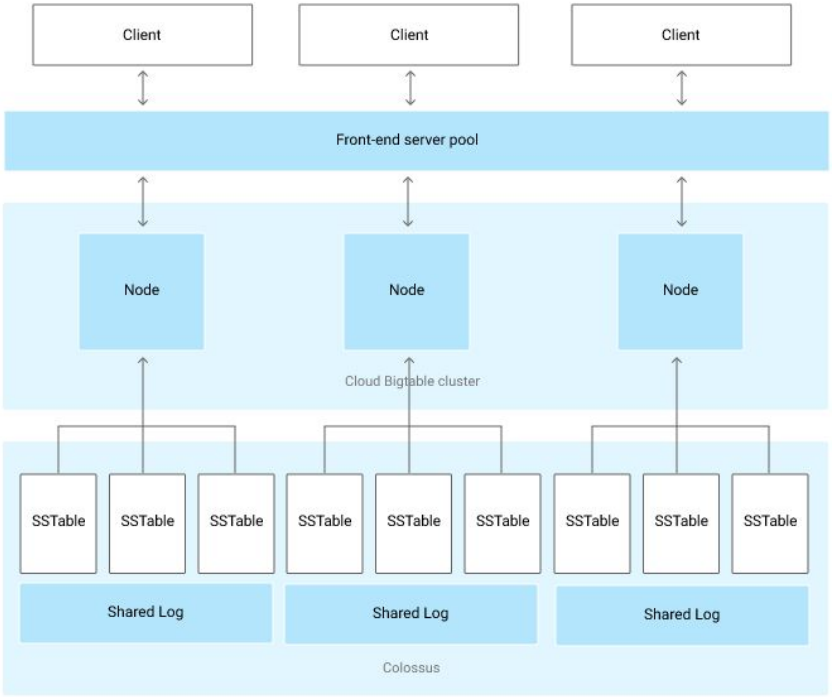
- NoSQL
- Baixa latência
- Escalável

Modelo de armazenamento

- Tabela
  - Conjunto chave-valor
  - Linhas e colunas
  - Versões

Modelo de armazenamento

- Arquitetura



That’s all folks