INSTITUTO DE GESTÃO E TECNOLOGIA
DA INFORMAÇÃO

Processamento de Fluxos Contínuos de Dados

José Carlos Barbosa

2022

Processamento de Fluxos Contínuos de Dados

Bootcamp: Engenheiro(a) de Dados Cloud

José Carlos Barbosa

Sumário

Capítulo 1. Introdução a Stream Processing Applications

O Stream processing é uma tecnologia Big Data usada para coletar, armazenar e gerenciar fluxos contínuos de dados, detectando de forma rápida, chegando até a milissegundos, os dados gerados ou recebidos. Possui, também, diversas nomenclaturas como: Real-time Stream Analytics, Real-time Analytics, Stream Analytics, Event Processing e Complex Event Processing.

Contextualização histórica

Os primeiros protótipos de pesquisa e produtos comerciais começaram a surgir no final dos anos 90, começando com bancos de dados ativos que proviam consultas condicionais nos dados armazenados nesses bancos. No início, os frameworks de Stream processing eram limitados a usos em pesquisas acadêmicas ou aplicações de nicho, como o mercado de ações. No entanto, o aumento da adoção de tecnologias de Stream processing, nos últimos anos, foi bastante motivada pela disponibilização de projetos Open Source de Stream processing maduros, como o Yahoo S4 e o Apache Storm, que foram introduzidos como “Parecidos com o Hadoop, mas em real-time” e se tornaram parte do movimento Big Data.

Streaming lifecycle

Quando se fala de ciclo de vida de processamento de dados em real-time, leva-se em consideração a continuidade e a limitação do tempo. O processamento foca o fluxo de Big Data que é ingerido em real-time e processado com latência mínima, divididos em cinco fases conceituais que incluem:

- Ingestão de dados:

- Fase em que os dados são ingeridos a partir de diferentes fontes de dados. Muitos paradigmas de processamento em real-time usam um armazenamento para a ingestão de mensagens para servirem como um buffer para as mensagens.

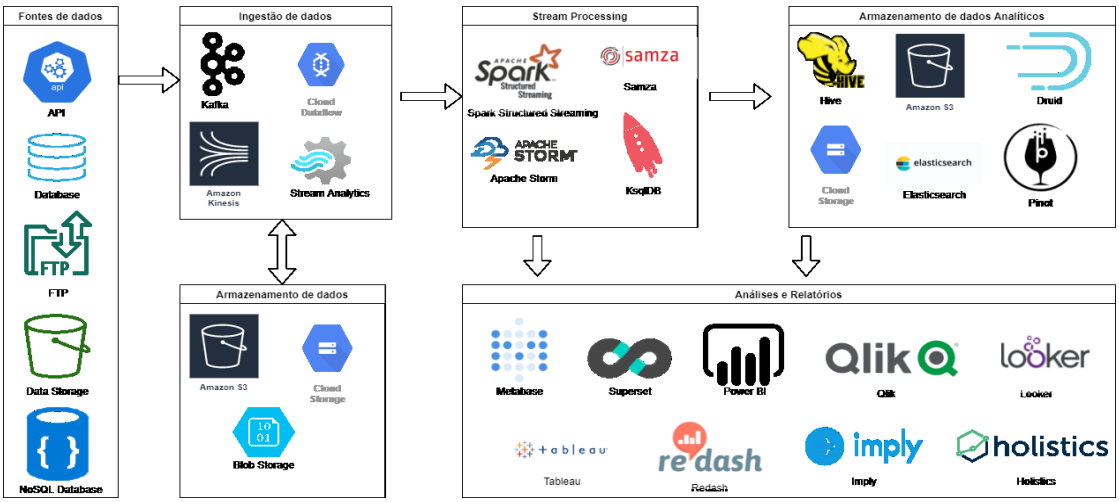
- Armazenamento de dados:
    - Fase em que ocorrem as operações de armazenamento dos dados. Na maioria das aplicações de real-time Big Data, esse armazenamento também é usado como armazenamento analítico ao final da etapa de Stream processing.

- Stream processing:
    - Fase em que os fluxos de dados em real-time são processados e estruturados para real-time analytics e para tomada de decisões.

- Armazenamento de dados analíticos:
    - Fase em que os dados processados e estruturados são armazenados e disponibilizados para consultas.

- Análises e relatórios:
    - Fase em que os dados são disponibilizados com implicações e insights para uma efetiva tomada de decisão.

Figura 1 – Streaming Life Cycle.



Data Producers

Os Data Producers são as fontes de dados da nossa arquitetura em real-time. As mais comuns são:

- API.
- Data Storages.
- FTPs.
- Database SQL.
- Database NoSQL.

Ingestão de dados

Etapa em que usamos as ferramentas de ingestão de dados em streaming. As mais usadas são:

- Kafka.

- Google Cloud Dataflow.
- Amazon Kinesis.
- Azure Stream Analytics.

Stream Processing

Etapa em que realizamos o processamento dos dados em real-time. As ferramentas mais usadas são:

- Spark Structure Streaming;
- KsqlDB.

Armazenamento de dados

Etapa em que os dados frios são armazenados. É muito comum este tipo de dado ser armazenado em Data Storages, como:

- Amazon S3.
- Google Cloud Storage.
- Azure Blob Storage.

Armazenamento de dados analíticos

Etapa em que os dados analíticos são armazenados. Podem ser armazenados em Data Storages e consultados por ferramentas de consulta ou disponibilizados em Data Warehouses. Os mais comuns são:

- Apache Hive.
- Apache Druid.

- Apache Pinot.
- Elasticsearch.
- Amazon S3.
- Google Cloud Storage.
- Azure Blob Storage.

Análises e relatórios

Etapa em que os dados são disponibilizados em ferramentas de visualização para facilitar a análise e tomada de decisão. As ferramentas mais utilizadas são:

- Metabase.
- Superset.
- Power BI.
- Tableau.
- Redash.
- Qlik.

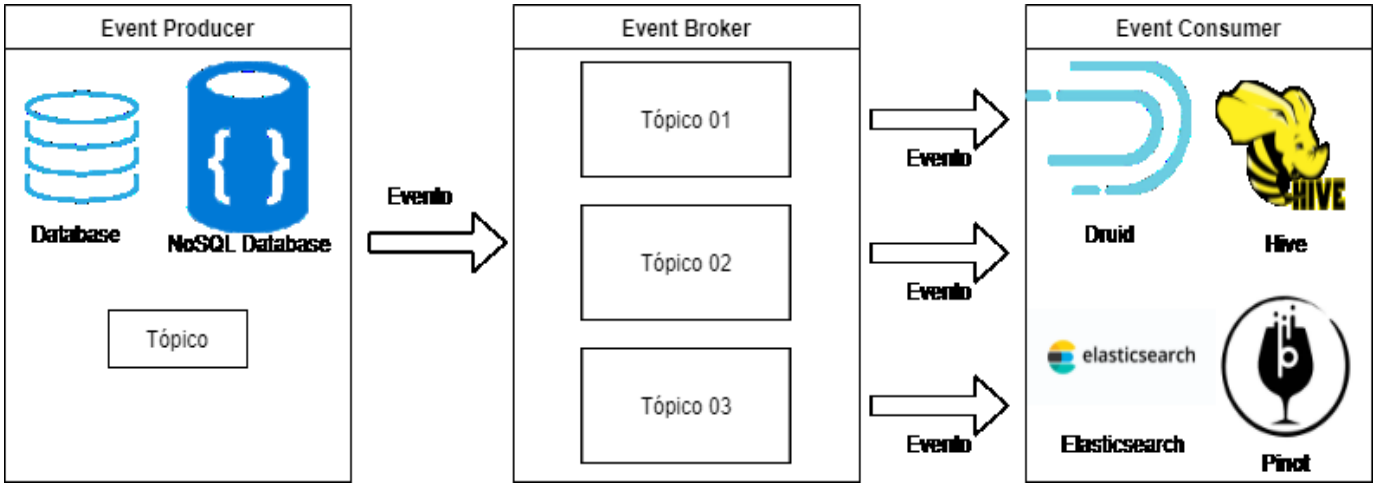
Capítulo 2. Arquitetura de sistemas de Stream processing

Neste capítulo, teremos um overview sobre duas arquiteturas de sistemas de Stream processing muito utilizadas atualmente: a arquitetura orientada a eventos e a arquitetura orientada a microsserviços.

Arquitetura orientada a eventos

A arquitetura orientada a eventos é um modelo de arquitetura de software para o design de aplicações que permite que uma organização detecte eventos ou momentos importantes e atue nelas em real-time ou near real-time. Em um sistema orientado a eventos, os componentes de captura, comunicação, processamento e persistência de eventos formam a estrutura básica da solução e, por usar uma quantidade mínima de acoplamentos, se torna uma boa opção para as arquiteturas de aplicações distribuídas e modernas.

Figura 2 – Arquitetura orientada a eventos.



Um produtor detecta ou percebe um evento e o representa como uma mensagem. Após um evento ser detectado, ele é transmitido do produtor para o consumidor por meio dos tópicos no Event Broker, em que os eventos são processados

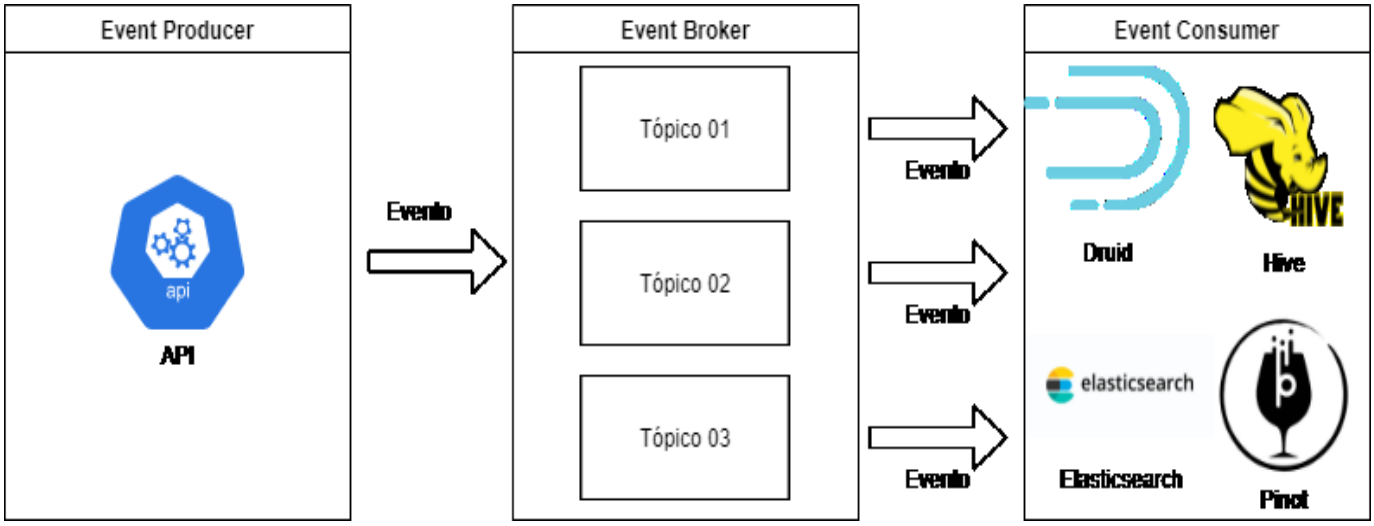
de maneira assíncrona e, em seguida, enviados em sentido downstream para os consumidores certos.

Ao adotar uma arquitetura orientada a eventos, as organizações conquistam um sistema flexível capaz de se adaptar a mudanças e tomar decisões em tempo real.

Arquitetura de microsserviços

Microsserviços são um padrão de arquitetura que estrutura uma aplicação com uma coleção de serviços pequenos e acoplados que operam juntos para atingirem um objetivo em comum. Como eles trabalham de forma independente, eles podem ser adicionados, removidos e atualizados sem interferirem em outras aplicações.

Figura 3 – Arquitetura de microsserviços.



Seus principais benefícios são a facilidade no deploy e no teste, maior produtividade, flexibilidade e escalabilidade.

Arquiteturas de sistemas de Stream processing usando microsserviços permitem comunicação em real-time e a disponibilização dos dados para os consumidores na forma de eventos antes mesmo deles serem requisitados.

Capítulo 3. Padrões de projetos de fluxos contínuos de dados

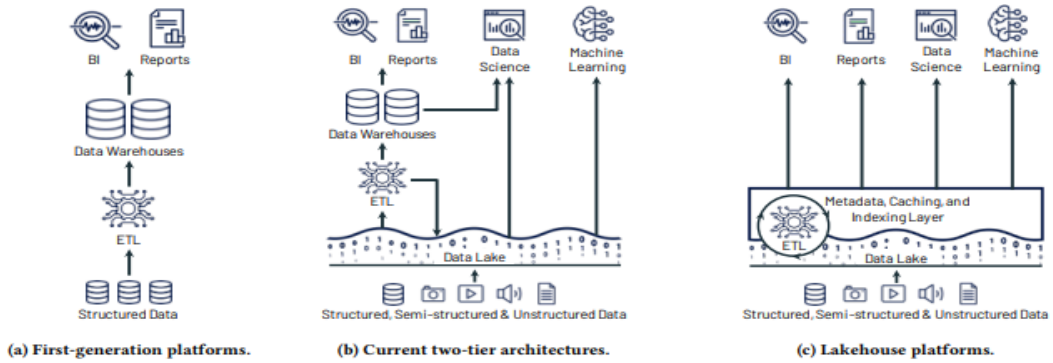
Neste capítulo, veremos os padrões mais utilizados em projetos de fluxos contínuos de dados.

Data Lake vs. Data Lakehouse

O Data Lake surgiu para lidar com diversos desafios enfrentados na arquitetura Data Warehouse, dentre eles, o alto custo envolvido em computação e armazenamento, além do grande crescimento de datasets completamente não estruturados, que não podiam ser armazenados ou consultados em Data Warehouses. Para isso, começou-se a ingerir os dados brutos em Data Storages de baixo custo, movimento esse que começou com o Apache Hadoop, usando o Hadoop File System (HDFS) como armazenamento.

Isso trouxe diversos benefícios, como a democratização dos dados, a possibilidade de lidar com dados estruturados, semiestruturados, não estruturados e a abertura posterior para a computação em nuvem, mas trouxe, consigo, uma série de desvantagens, como a desorganização dos dados, a dificuldade de gerenciamento, controle dos dados e problemas com qualidade e segurança dos dados.

Figura 4 – Evolução das arquiteturas de plataforma de dados.



Fonte: http://cidrdb.org/cidr2021/papers/cidr2021_paper17.pdf

O Data Lakehouse surge para unir as vantagens das arquiteturas anteriores, Data Warehouse e Data Lake, e para lidar com os problemas enfrentados nas arquiteturas anteriores. Por garantir transações ACID (atomicidade, consistência, isolamento e durabilidade), gera logs com metadados, permite versionamento de dados e enfrenta dificuldades apresentadas anteriormente, como problemas com governança e qualidade dos dados.

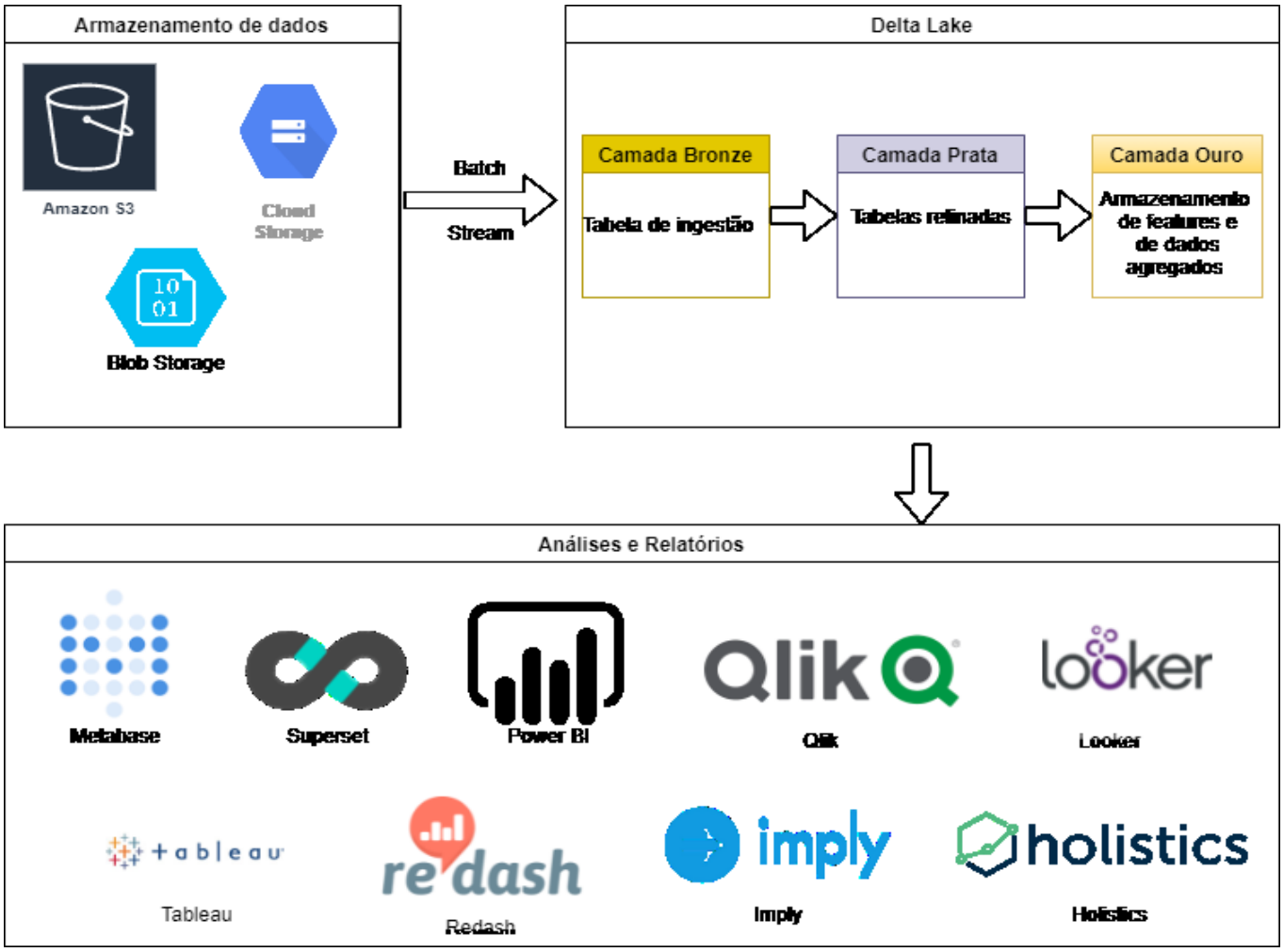
Delta Architecture

O Delta Lake é um framework para a aplicação do Data LakeHouse, criado pela DataBricks e otimizado para a integração com o Apache Spark. Dentre suas vantagens estão:

- Gerenciamento de dados confiáveis do Data Lake:
    - Geração de metadados das tabelas delta;
    - Transações ACID (atomicidade, consistência, isolamento e durabilidade);
    - Versionamento (Time Travel);
    - Possibilidade de auditoria;
    - Evolução de Schem.
- Formato aberto.
- Usabilidade em batch e streaming.
- Updates e deletes.
- Suporte para Machine Learning e Data Science.
- Performance SQL (nova engine de consulta DataBricks (Delta Engine).

O Delta Lake se caracteriza, dentro da arquitetura, na hora da escrita dos dados após o processamento. A escrita é realizada no formato delta e, assim, são gerados os logs com os metadados e as tabelas delta, que podem ser consumidas diretamente por ferramentas de consulta ou de visualização.

Figura 5 – Arquitetura Delta.



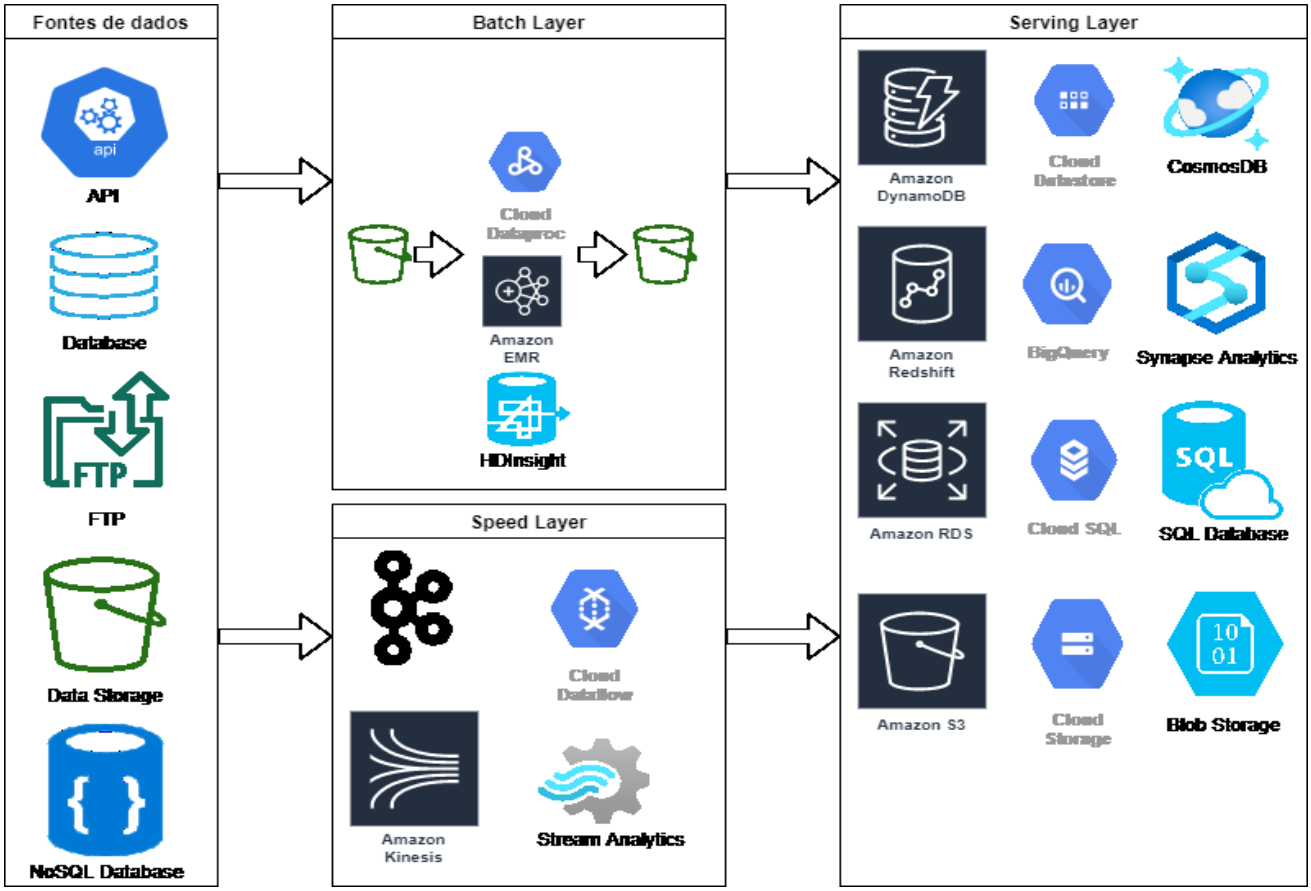
Capítulo 4. Arquitetura de Projetos de Dados

Neste capítulo, teremos um overview sobre os principais tipos de arquiteturas em projetos de dados.

Lambda Architecture (Cloud Agnostic)

A arquitetura Lambda é uma forma de processar grandes quantidades de dados através de uma abordagem híbrida contendo uma via de processamento em batch e outra de processamento em stream. Seus principais benefícios são a Escalabilidade flexível, Alta disponibilidade e Agilidade em reagir a mudanças nos cenários de negócios.

Figura 6 – Arquitetura Lambda.



A arquitetura Lambda é dividida em três camadas:

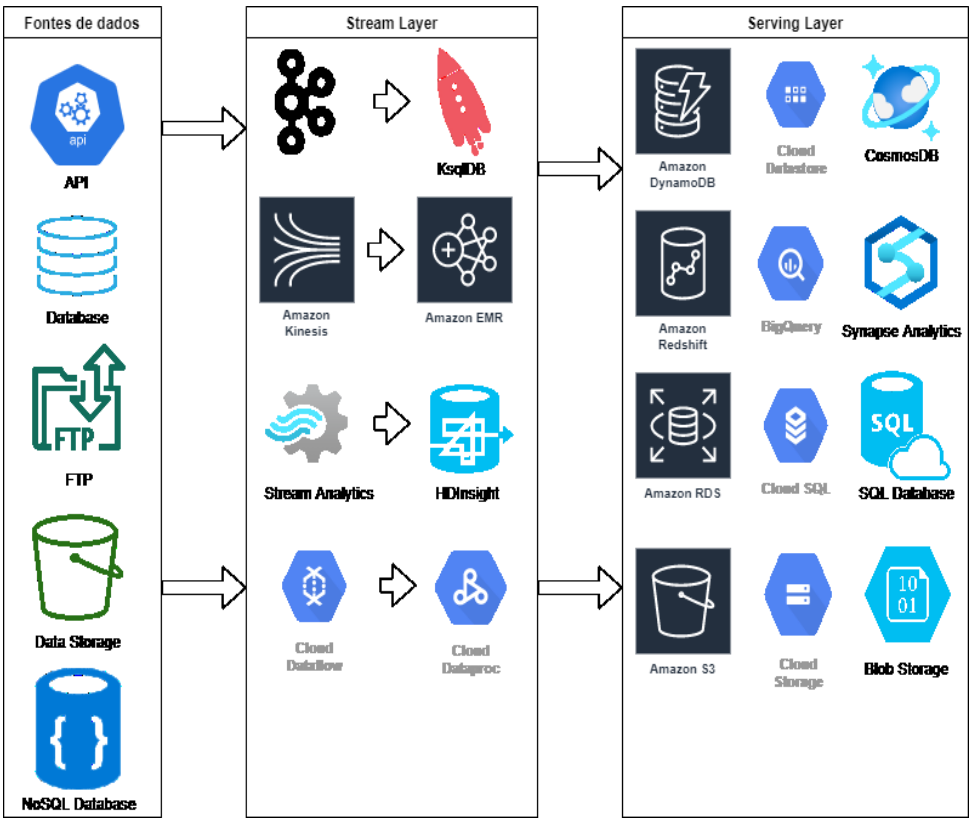
- Camada Batch (Batch Layer):
    - Carga realizada em um cronograma predeterminado, geralmente uma ou duas vezes por dia.
    - Podem manter dados históricos.
- Camada Rápida (Speed Layer):
    - Dados disponibilizados em real-time ou near real-time.
    - Mantém apenas os dados mais recentes.
- Camada de disponibilização (Serving Layer):
    - Recebe o output da camada batch e da camada rápida.

Kappa Architecture (Cloud Agnostic)

A arquitetura Kappa é uma arquitetura de software usada para processamento de dados em stream, ela é considerada uma alternativa mais simples da arquitetura Lambda, pois realiza tanto o processamento em Batch como o processamento em Stream usando um único stack de tecnologia.

Os dados mais recentes são disponibilizados em real-time ou near-real time após inseridos na engine de mensagens, enquanto os dados históricos podem ser disponibilizados em um horário posterior através de uma carga em batch.

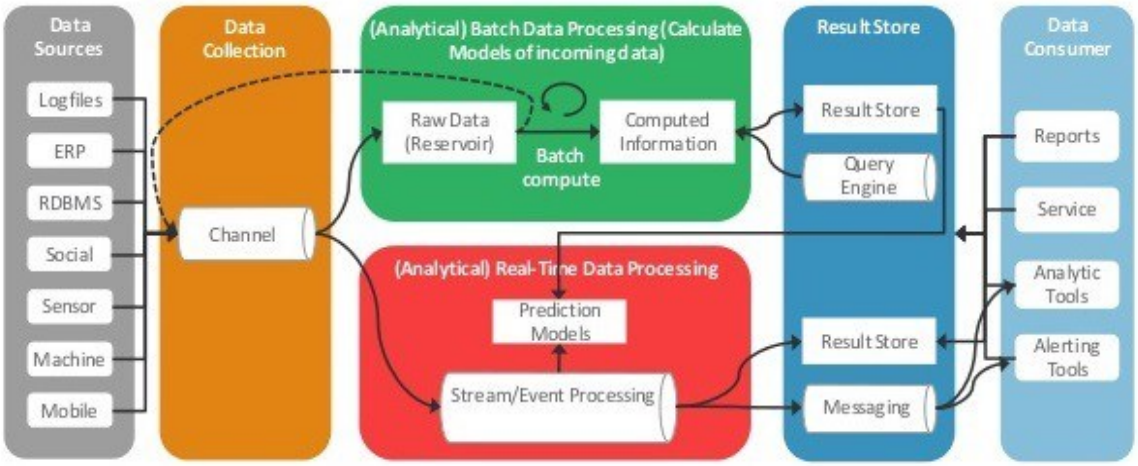
Figura 7 – Arquitetura Kappa.



Unifield Architecture (Cloud Agnostic)

A Arquitetura Unifield combina Machine Learning e processamento de dados. Possui um processamento em streaming com novas camadas sendo adicionadas para realizar o Machine Learning e o treinamento dos modelos. A sua principal vantagem é a capacidade de combinar análise de dados com soluções envolvendo Machine Learning.

Figura 8 – Unifield Architecture.



Fonte: https://www.yanboyang.com/bigdataarchitectures/

Capítulo 5. Coleta, Armazenamento e Processamento de Dados de Fluxos Contínuos

Neste capítulo, teremos um overview dos processos de coleta, armazenamento e processamento de dados em arquiteturas de fluxos contínuos demonstrados nas aulas práticas.

Data Producers em arquitetura de microsserviços com API para inserir dados em real-time no PostgreSQL

Iniciamos com a ingestão de dados em real-time através de uma API, que utiliza a biblioteca Faker para gerar dados falsos e popular um banco de dados PostgresQL. Essa API continua gerando amostras de dados de x linhas para cada coluna e insere no banco de dados até que sua execução seja cancelada. Ao final da execução, o número de linhas geradas é especificado.

- Pré-requisitos:
    - Faker: 8.10.0 https://pypi.org/project/Faker/
    - Python 3.9.1 https://www.python.org/downloads/
    - python-dotenv 0.19.0 https://pypi.org/project/python-dotenv/
- Usabilidade:
    - Crie um arquivo .env com as seguintes credenciais:
        - DATABASE_USER
        - PASSWORD
        - HOST

- PORT
- DATABASE
- TABLE

- Execute o arquivo python:
  - python -m <nome_do_arquivo> -n <numero_de_linhas>

- Resultado esperado:

Figura 9 – Script sendo executado.

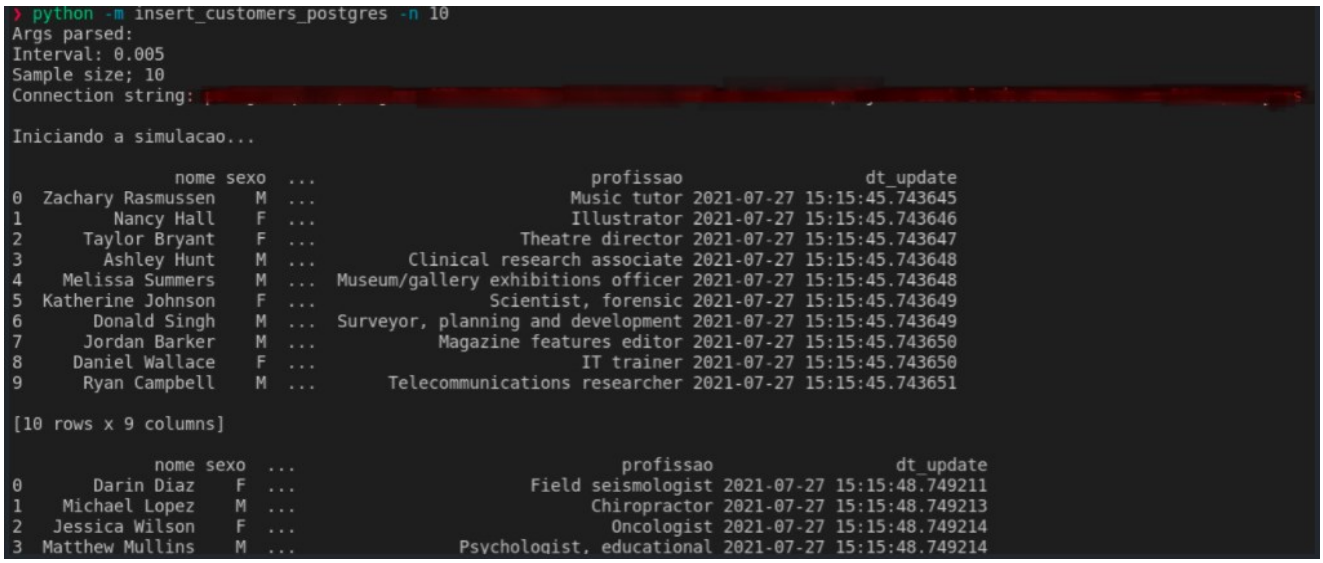
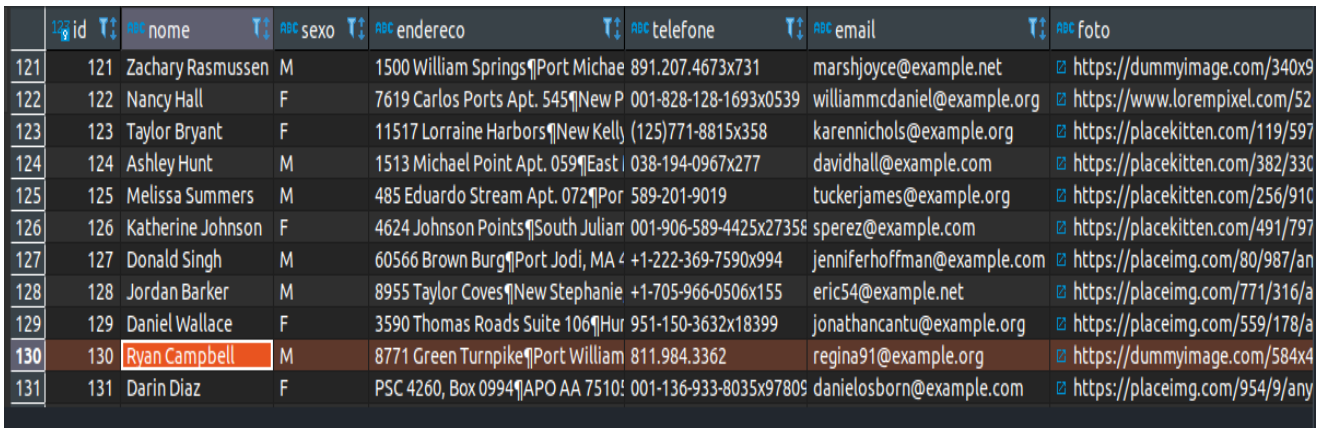


Figura 10 – Dados no Banco de dados PostgreSQL.



	id	nome	sexo	endereco	telefone	email	foto
121	121	Zachary Rasmussen	M	1500 William Springs¶Port Michae	891.207.4673x731	marshjoyce@example.net	https://dummyimage.com/340x9
122	122	Nancy Hall	F	7619 Carlos Ports Apt. 545¶New P	001-828-128-1693x0539	williammcdaniel@example.org	https://www.lorempixel.com/52
123	123	Taylor Bryant	F	11517 Lorraine Harbors¶New Kell	(125)771-8815x358	karennichols@example.org	https://placekitten.com/119/597
124	124	Ashley Hunt	M	1513 Michael Point Apt. 059¶East	038-194-0967x277	davidhall@example.com	https://placekitten.com/382/330
125	125	Melissa Summers	M	485 Eduardo Stream Apt. 072¶Por	589-201-9019	tuckerjames@example.org	https://placekitten.com/256/910
126	126	Katherine Johnson	F	4624 Johnson Points¶South Julian	001-906-589-4425x27358	sperez@example.com	https://placekitten.com/491/797
127	127	Donald Singh	M	60566 Brown Burg¶Port Jodi, MA 4	+1-222-369-7590x994	jenniferhoffman@example.com	https://placeimg.com/80/987/an
128	128	Jordan Barker	M	8955 Taylor Coves¶New Stephanie	+1-705-966-0506x155	eric54@example.net	https://placeimg.com/771/316/a
129	129	Daniel Wallace	F	3590 Thomas Roads Suite 106¶Hur	951-150-3632x18399	jonathancantu@example.org	https://placeimg.com/559/178/a
130	130	Ryan Campbell	M	8771 Green Turnpike¶Port William	811.984.3362	regina91@example.org	https://dummyimage.com/584x4
131	131	Darin Diaz	F	PSC 4260, Box 0994¶APO AA 7510	001-136-933-8035x9780	danielosborn@example.com	https://placeimg.com/954/9/any

Como funciona o Kafka Connect para conexão ao PostgreSQL

Em seguida, utilizaremos o Strimzi para rodar o nosso Kafka no Kubernetes e criar o Kafka Connect para a conexão com o PostgreSQL.

- Pré-requisitos:
    - Docker: 20.10.7 https://docs.docker.com/engine/install/
        - Python 3.9.1 https://www.python.org/downloads/
- Usabilidade:
    - Construa o strimzi operator:
        - docker build -t strimzi-operator-<namespace> .
    - Tag a imagem do strimzi operator:
        - docker tag our-image-kafka-connect-strimzi:2.7 YOURdockerRepository/our-image-kafka-connect-strimzi:2.7

- Envie a imagem para o Dockerhub:
    - docker push YOURdockerRepository/our-image-kafka-connect-strimzi:2.7
- Configuração no Cluster:
    - Criar namespace
        - k create namespace <namespace>
    - Adicionar Helm:
        - helm repo add strimzi https://strimzi.io/charts/
        - helm repo update
        - helm install kafka strimzi/strimzi-kafka-operator --namespace <namespace> --version 0.22.0
        - helm ls -n <namespace>
    - Adicionar Yaml [kafka-ephemeral.yml]:
        - k apply -f kafka-ephemeral.yml -n <namespace>
    - Adicionar Yaml [kafka-connect.yml]:
        - k apply -f kafka-connect.yml -n <namespace>
    - Adicionar Yaml [ingest-src-postgres-customers-json.yml]:
        - k apply -f ingest-src-postgres-customers-json.yml -n <namespace>

- Resultado esperado:

Figura 11 – Pods criados.

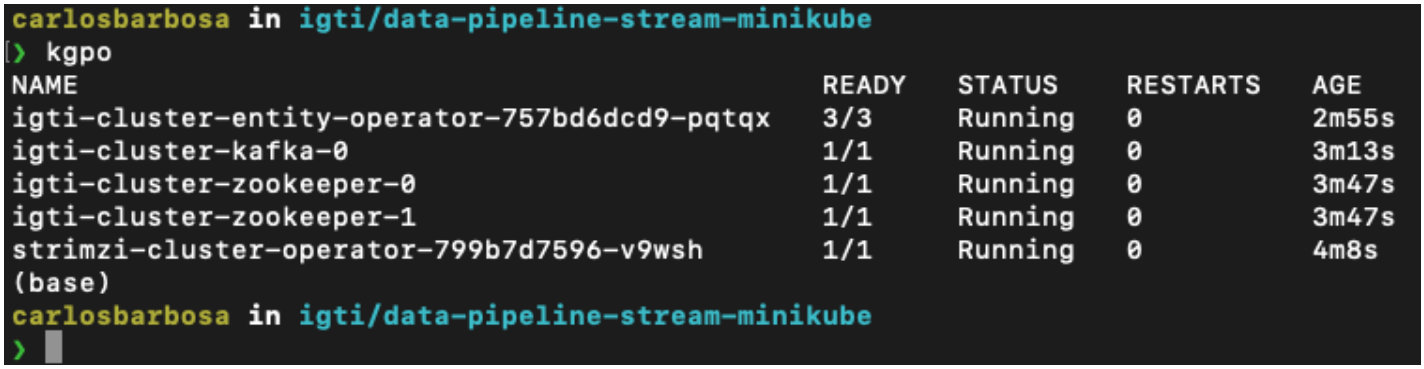


Figura 12 – Tópico criado.

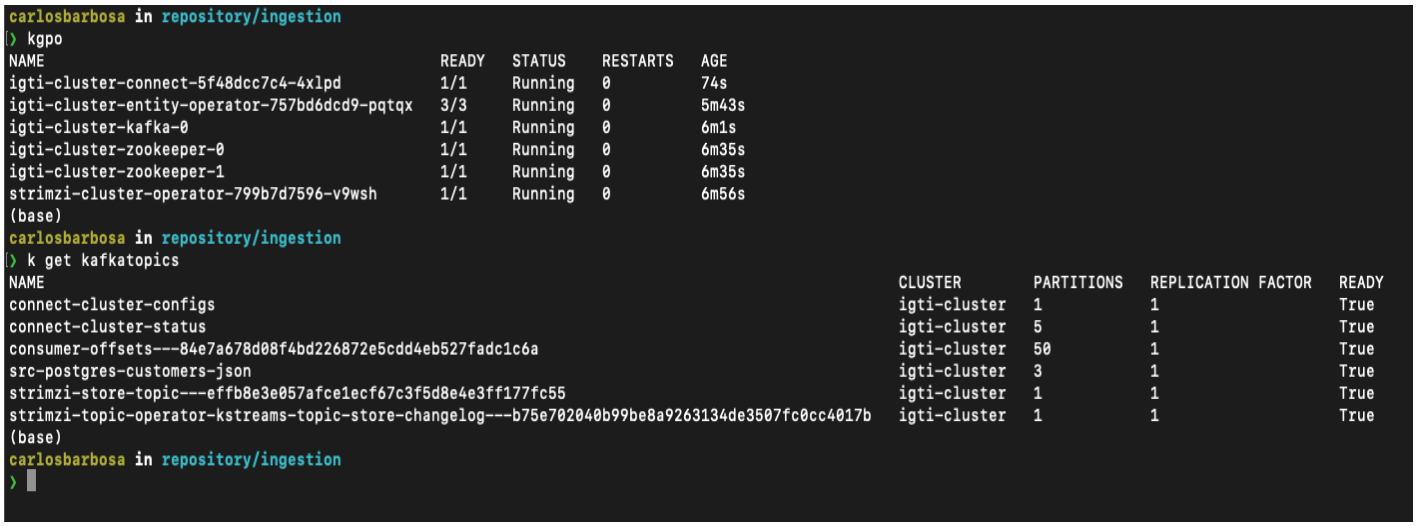


Figura 13 – Tópico consumido.

```
kubectl exec igti-cluster-kafka-0 -c kafka -i -t -- \
  bin/kafka-console-consumer.sh \
    --bootstrap-server localhost:9092 \
    --property print.key=true \
    --from-beginning \
    --topic src-postgres-customers-json
```

Figura 14 – Dados no Kafka.

```
carlosbarbosa in repository/ingestion
) kubectl exec igti-cluster-kafka-0 -c kafka -i -t -- \
  bin/kafka-console-consumer.sh \
    --bootstrap-server localhost:9092 \
    --property print.key=true \
    --from-beginning \
    --topic src-postgres-customers-json
{"schema":{"type":"int32","optional":false},"payload":2413238}  {"schema":{"type":"struct","fields":[{"type":"int32","optiona
ield":"endereco"},{"type":"string","optional":true,"field":"telefone"},{"type":"string","optional":true,"field":"email"},{"ty
"type":"string","optional":true,"field":"profissao"},{"type":"int64","optional":true,"name":"org.apache.kafka.connect.data.Ti
],"optional":false},"payload":{"id":2413238,"nome":"Brianna Scott","sexo":"M","endereco":"26609 Young Fields\nTiffanyview, MN
ation administrator","dt_update":1627568689787,"messagetopic":"src-postgres-customers-json","messagesource":"postgresql"}}
{"schema":{"type":"int32","optional":false},"payload":2413242}  {"schema":{"type":"struct","fields":[{"type":"int32","optiona
ield":"endereco"},{"type":"string","optional":true,"field":"telefone"},{"type":"string","optional":true,"field":"email"},{"ty
"type":"string","optional":true,"field":"profissao"},{"type":"int64","optional":true,"name":"org.apache.kafka.connect.data.Ti
],"optional":false},"payload":{"id":2413242,"nome":"Bradley Young","sexo":"F","endereco":"2999 Greer Circle\nLake Heatherport
ic interpreter","dt_update":1627568689787,"messagetopic":"src-postgres-customers-json","messagesource":"postgresql"}}
{"schema":{"type":"int32","optional":false},"payload":2413243}  {"schema":{"type":"struct","fields":[{"type":"int32","optiona
ield":"endereco"},{"type":"string","optional":true,"field":"telefone"},{"type":"string","optional":true,"field":"email"},{"ty
"type":"string","optional":true,"field":"profissao"},{"type":"int64","optional":true,"name":"org.apache.kafka.connect.data.Ti
],"optional":false},"payload":{"id":2413243,"nome":"Rachel Garcia","sexo":"F","endereco":"224 Elizabeth Underpass\nSarahside,
neering geologist","dt_update":1627568689787,"messagetopic":"src-postgres-customers-json","messagesource":"postgresql"}}
```

Processamento em tempo real com KsqlDB e Spark Structure Streaming

Agora, utilizaremos o KsqlDB para realizar o Stream processing dos dados que se encontram no kafka hub.

- Pré-requisitos:
    - Docker: 20.10.7 https://docs.docker.com/engine/install/
    - Docker img strimzi operator
- Usabilidade:
    - Adicionar Yaml [example-kafka-strimzi-operator-k8s-v1beta2.yml]
        - k apply -f template/ -n <namespace>

- Resultado esperado:

Figura 15 – KsqlDB pod

Figura 16 – Conexão com o KsqlDB - parte 1.

```
❯ k exec <nome do pod> -n <namespace> -i -t --bash ksql
```

Figura 17 – Conexão com o KsqlDB - parte 2.

```
carlosbarbosa in data-pipeline-stream-minikube/repository
❯ kgpo
NAME                            READY   STATUS    RESTARTS   AGE
ksqldb-server-856d59df74-4vjtm   1/1     Running   0          40s
(base)
carlosbarbosa in data-pipeline-stream-minikube/repository
❯ k exec ksqldb-server-856d59df74-4vjtm -n processing -i -t -- bash ksql
OpenJDK 64-Bit Server VM warning: Option UseConcMarkSweepGC was deprecated in version 9.0 and will likely be removed in a future release.

                  ===========================================
                  =       _              _ ____  ____       =
                  =      | | _____  __ _| |  _ \| __ )      =
                  =      | |/ / __|/ _` | | | | |  _ \      =
                  =      |   <\__ \ (_| | | |_| | |_) |     =
                  =      |_|\_\___/\__, |_|____/|____/      =
                  =                   |_|                   =
                  =  Event Streaming Database purpose-built =
                  =        for stream processing apps       =
                  ===========================================

Copyright 2017-2020 Confluent Inc.

CLI v0.12.0, Server v0.12.0 located at http://localhost:8088
Server Status: RUNNING

Having trouble? Type 'help' (case-insensitive) for a rundown of how things work!

ksql> show topics;

 Kafka Topic                                          | Partitions | Partition Replicas
-------------------------------------------------------------------------------------------
 __strimzi-topic-operator-kstreams-topic-store-changelog | 1          | 1
 __strimzi_store_topic                                | 1          | 1
 connect-cluster-configs                              | 1          | 1
 connect-cluster-offsets                              | 25         | 1
 connect-cluster-status                               | 5          | 1
 default_ksql_processing_log                          | 1          | 1
 src-postgres-customers-json                          | 3          | 1
-------------------------------------------------------------------------------------------
ksql>
```

Data Storage

Finalmente, faremos a ingestão dos dados para um Data Storage através de uma conexão sink.

- Pré-requisitos:
    - Docker: 20.10.7 https://docs.docker.com/engine/install/
    - Docker img strimzi operator
- Usabilidade:
    - Adicionar Yaml [example-kafka-strimzi-k8s-v1beta1.yml]
        - k apply -f template/ -n <namespace>

Figura 18 – Conexão Sink.

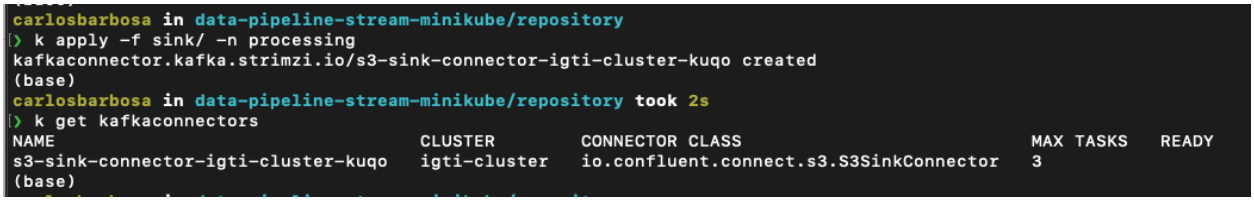


Figura 19 – Kafka Connector

Capítulo 6. Arquitetura e pipelines de dados para tempo real usando Kafka para armazenamento

A seguir, faremos uma breve introdução sobre quatro das principais ferramentas usadas para armazenamento de dados para análise em real-time:

Apache Druid - O que é? Para que serve? Como usar?

- O que é?
    - O Apache Druid é um banco de dados de alta performance para análises em real-time. Seu maior valor é a redução do tempo para obter insights e tomadas de decisão.
- Para que serve?
    - O Druid foi criado para fluxos de trabalho onde consultas rápidas e ingestões realmente importam, lidando com consultas operacionais e alta concorrência com maestria. O Druid pode ser considerado uma alternativa Open Source para Data Warehouse para uma variedade de casos.
- Como usar?
    - Seguindo a documentação, temos o Quickstart para o Druid de forma local e pelo Docker:
        - Local - https://druid.apache.org/docs/latest/tutorials/index.html
        - Docker - http://druid.apache.org/docs/latest/tutorials/docker.html

Apache Pinot - O que é? Para que serve? Como usar?

- O que é?
    - O Pinot é um Data Store OLAP distribuído e voltado para arquiteturas que usam streaming de dados em real-time, construído para suportar real-time analytics escaláveis com baixa latência.
- Para que serve?
    - O Pinot foi construído pelos engenheiros do LinkedIn e Uber para suportar real-time analytics escaláveis e com baixa latência. A performance sempre se mantém constante baseada no tamanho do seu cluster e numa quantidade de consultas por segundo esperada.
- Como usar?
    - Seguindo a documentação, temos o seguinte Getting Started para o Pinot:
        - https://docs.pinot.apache.org/#get-started

Apache Hive - O que é? Para que serve? Como usar?

- O que é?
    - O Hive é um software de Data Warehouse construído em cima do Apache Hadoop.
- Para que serve?
    - O Hive facilita a leitura, escrita e o gerenciamento de grandes Datasets armazenados em ambientes distribuídos usando SQL.

- Como usar?
    - Seguindo a documentação, temos o seguinte Getting Started para o Hive:
        - https://cwiki.apache.org/confluence/display/Hive/GettingStarted

ElasticSearch - O que é? Para que serve? Como usar?

- O que é?
    - O ElasticSearch é um engine RESTful distribuído de pesquisa e análise.
- Para que serve?
    - O ElasticSearch pode ser usado para armazenar, consultar e gerenciar dados.
- Como usar?
    - Seguindo a documentação, temos o seguinte guia de instalação para o ElasticSearch:
        - https://www.elastic.co/guide/en/elasticsearch/reference/current/install-elasticsearch.html

Referências

APACHE DRUID. Disponível em: https://druid.apache.org/. Acesso em: 08 mar. 2021.

APACHE DRUID. Disponível em: https://github.com/apache/druid. **GitHub**. Acesso em: 08 mar. 2021.

APACHE HIVE. Disponível em: https://github.com/apache/hive. **GitHub**. Acesso em: 04 ago. 2021.

APACHE HIVE. Disponível em: https://hive.apache.org/. Acesso em: 08 mar. 2021.

APACHE PINOT. Disponível em: https://github.com/apache/pinot. **GitHub**. Acesso em: 08 mar. 2021.

APACHE PINOT. Disponível em: https://pinot.apache.org/. Acesso em: 08 mar. 2021.

ARMBRUST, Michael; GHODSI, Ali; XIN, Reynold; ZAHARIA, Matei. **Lakehouse**: A New Generation of Open Platforms that Unify Data Warehousing and Advanced Analytics. Disponível em: http://cidrdb.org/cidr2021/papers/cidr2021_paper17.pdf. Acesso em: 08 mar. 2021.

AWS. **How SmartNews Built a Lambda Architecture on AWS to Analyze Customer Behavior and Recommend Content.** Disponível em: https://aws.amazon.com/pt/blogs/big-data/how-smartnews-built-a-lambda-architecture-on-aws-to-analyze-customer-behavior-and-recommend-content/. Acesso em: 08 mar. 2021.

CONFLUENT. **Event Driven microservices**. Disponível em: https://www.confluent.io/resources/event-driven-microservices/?utm_medium=sem&utm_source=google&utm_campaign=ch.sem_br.nonbrand_tp.prs_tgt.technical-research_mt.xct_rgn.latam_lng.eng_dv.all_con.event-

driven-architecture&utm_term=event%20driven%20architecture&creative=&device=c&placement=&gclid=CjwKCAjwjJmIBhA4EiwAQdCbxsV-kamLdLoXTpQPxBTWHR0Ui2wCOZ82CtPdx59NP2IIpxBkIMsbQRoCIdcQAvD_BwE. Acesso em: 08 mar. 2021.

CONFLUENT. **Stream Processing Fundamentals**: A Confluent Online Talk Series. Disponível em: https://www.confluent.io/online-talks/stream-processing-fundamentals/?utm_medium=sem&utm_source=google&utm_campaign=ch.sem_br.nonbrand_tp.prs_tgt.dsa_mt.dsa_rgn.latam_lng.eng_dv.all_con.online-talks&utm_term=&creative=&device=c&placement=&gclid=CjwKCAjwjJmIBhA4EiwAQdCbxmMiWgb8motUeGzip9jxOOC6NrklaS3os1RTQNrZAzLeRf9eI2c4ERoCntYQAvD_BwE. Acesso em: 08 mar. 2021.

DATABRICKS. **Lambda Architecture**. Disponível em: https://databricks.com/glossary/lambda-architecture. Acesso em: 08 mar. 2021.

ELASTIC. Disponível em: https://github.com/elastic/elasticsearch. **GitHub**. Acesso em: 08 mar. 2021.

ELASTIC. Disponível em: https://www.elastic.co/pt/. Acesso em: 08 mar. 2021.

GURCAN, Fatih; MUHAMMET, Berigel. **Real-Time Processing of Big Data Streams**: Lifecycle, Tools, Tasks, and Challenges. Disponível em: https://www.researchgate.net/publication/329565393_Real-Time_Processing_of_Big_Data_Streams_Lifecycle_Tools_Tasks_and_Challenges. Acesso em: 08 mar. 2021.

HAZELCAST. **What is the Kappa Architecture.** Disponível em: https://hazelcast.com/glossary/kappa-architecture/. Acesso em: 08 mar. 2021.

HUESKE, Fabian; VASILIKI, Kalavri O'Reily. **Chapter 1. Introduction to Stateful Stream Processing.** Disponível em: https://www.oreilly.com/library/view/stream-processing-with/9781491974285/ch01.html. Acesso em: 08 mar. 2021.

HUSSAIN, Sajjad. 4 big data architectures, Data Streaming, Lambda architecture, Kappa architecture and Unifield architecture. **Medium**, 18 jan; 2021. Disponível em: https://medium.com/dataprophet/4-big-data-architectures-data-streaming-lambda-architecture-kappa-architecture-and-unifield-d9bcbf711eb9. Acesso em: 08 mar. 2021.

PERERA, Srinath. A Gentle Introduction to Stream Processing. **Medium Stream Processing**, 4 abr. 2021. Disponível em: https://medium.com/stream-processing/what-is-stream-processing-1eadfca11b97. Acesso em: 08 mar. 2021.

RED HAT. **What is event driven architecture**. Disponível em: https://www.redhat.com/pt-br/topics/integration/what-is-event-driven-architecture. Acesso em: 08 mar. 2021.

TIBCO. **What is event-driven Architecture**. Disponível em: https://www.tibco.com/reference-center/what-is-event-driven-architecture. Acesso em: 08 mar. 2021.

YAN, Boyang. **4 big data architectures, Data Streaming, Lambda architecture, Kappa architecture and Unifield architecture.** Disponível em: https://www.yanboyang.com/bigdataarchitectures/. Acesso em: 08 mar. 2021.