**FACULDADE DE TECNOLOGIA DE SÃO JOSÉ DOS CAMPOS**

**FATEC PROFESSOR Jessen Vidal**

**JASON DAVIN OLIVEIRA CARNEIRO**

**PLATAFORMA DE ENGENHARIA DE DADOS PARA EXTRAÇÃO, TRANSFORMAÇÃO E DISPONIBILIZAÇÃO DE DADOS EM LOTE E EM TEMPO REAL**

São José dos Campos

2021

**JASON DAVIN OLIVEIRA CARNEIRO**

**PLATAFORMA DE ENGENHARIA DE DADOS PARA EXTRAÇÃO, TRANSFORMAÇÃO E DISPONIBILIZAÇÃO DE DADOS EM LOTE E EM TEMPO REAL**

Trabalho de Graduação apresentado à Faculdade de Tecnologia de São José dos Campos, como parte dos requisitos necessários para a obtenção do título de Tecnólogo em Banco de Dados.

**Orientador: Me. Giuliano Araujo Bertoti**

São José dos Campos

2021

**Dados Internacionais de Catalogação-na-Publicação (CIP)**

**Divisão de Informação e Documentação**

CARNEIRO, Jason Davin Oliveira

Plataforma de Engenharia de Dados para Extração, Transformação e Disponibilização de Dados em Lote e em Tempo Real.

São José dos Campos, 2021.

54f.

Trabalho de Graduação – Curso de Tecnologia em Banco de Dados.

FATEC de São José dos Campos: Professor Jessen Vidal, 2021.

Orientador: Me. Giuliano Araujo Bertoti.

1.Engenharia de Dados. 2. Plataforma de Dados. 3. Código Aberto. I. Faculdade de Tecnologia. FATEC de São José dos Campos: Professor Jessen Vidal. Divisão de Informação e Documentação. II. Plataforma de Engenharia de Dados para Extração, Transformação e Disponibilização de Dados em Lote e em Tempo Real.

**REFERÊNCIA BIBLIOGRÁFICA**

CARNEIRO, Jason Davin Oliveira. **Plataforma de Engenharia de Dados para Extração, Transformação e Disponibilização de Dados em Lote e em Tempo Real.** 2021. 54f. Trabalho de Graduação - FATEC de São José dos Campos: Professor Jessen Vidal.

**CESSÃO DE DIREITOS**

NOME(S) DO(S) AUTOR(ES): Jason Davin Oliveira Carneiro

TÍTULO DO TRABALHO: Plataforma de Engenharia de Dados para Extração, Transformação e Disponibilização de Dados em Lote e em Tempo Real

TIPO DO TRABALHO/ANO: Trabalho de Graduação/2021.

É concedida à FATEC de São José dos Campos: Professor Jessen Vidal permissão para reproduzir cópias deste Trabalho e para emprestar ou vender cópias somente para propósitos acadêmicos e científicos. O autor reserva outros direitos de publicação e nenhuma parte deste Trabalho pode ser reproduzida sem a autorização do autor.

|  |  |
| --- | --- |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Jason Davin Oliveira Carneiro  Rua Penedo, 170  12237-070, São José dos Campos - SP |  |

**JASON DAVIN OLIVEIRA CARNEIRO**

**PLATAFORMA DE ENGENHARIA DE DADOS PARA EXTRAÇÃO, TRANSFORMAÇÃO E DISPONIBILIZAÇÃO DE DADOS EM LOTE E EM TEMPO**

Trabalho de Graduação apresentado à Faculdade de Tecnologia de São José dos Campos, como parte dos requisitos necessários para a obtenção do título de Tecnólogo em Banco de Dados.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Me. José Walmir Gonçalves Duque – FATEC – São José dos Campos**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Tecnólogo Samuel Machado**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Me. Giuliano Araujo Bertoti – FATEC – São José dos Campos**

**\_\_\_\_\_/\_\_\_\_\_/\_\_\_\_\_**

**DATA DA APROVAÇÃO**

**AGRADECIMENTOS**

Agradeço aos professores Giuliano Araujo Bertoti e Jorge Tadao Matsushima pela confiança depositada em mim, e pelo empenho empregado na realização deste trabalho. Ao excelente profissional e colega de trabalho, Rubens Minoru Bueno, pela introdução e acompanhamento pelo campo de engenharia de dados.

Agradeço a todos os professores da Faculdade de Tecnologia de São José dos Campos – Professor Jessen Vidal pela contribuição e compartilhamento de conhecimento dos diversos temas e técnicas apresentados neste trabalho.

**RESUMO**

A cada ano que passa, o avanço e alcance das tecnologias proporcionam às empresas e instituições uma capacidade maior de geração e processamento de dados. De acordo com a empresa Statista, a quantidade de informações geradas e consumidas desde 2010 vem crescendo exponencialmente, resultando uma estimativa de que em 2025 haverá cerca de 181 Zetabytes de dados gerados em todo o mundo. Em consequência, tecnologias que lidam com essa demanda também vêm sendo aprimoradas, surgindo um campo de estudo específico chamado engenharia de dados, onde novas técnicas e ferramentas para lidar com a coleta, armazenamento, processamento e disponibilização de informações vem sendo implementadas. Além de novas ferramentas e técnicas, há o surgimento de serviços em nuvem não gratuitos que oferecem um ferramental conjunto com propósito de gerenciar dados e, além disso, oferecer feedbacks de qualidade entre outras funcionalidades. Desta forma, o objetivo deste trabalho é desenvolver uma plataforma nativa em nuvem capaz de armazenar, processar e disponibilizar dados utilizando apenas ferramentas de código aberto. Com o uso da plataforma construída, foram implementados estudo de casos utilizando soluções reais e conhecidas no mercado, como por exemplo, a aplicação Clickstream.

**Palavras-Chave**: Engenharia de Dados; Plataforma de Dados; Nativo em Nuvem; Código aberto.

**ABSTRACT**

Over the years, the advancement and reach of technologies have provided companies and institutions a greater capacity for data generation and processing. According to the company Statista, the amount of information generated and consumed since 2010 has been growing exponentially, resulting in an estimate that in 2025 there will be around 181 Zetabytes of data generated worldwide. As a result, technologies that deal with this demand are also being improved, resulting in a specific field of study called data engineering, where new techniques and tools to deal with the collection, storage, processing and availability of information are being implemented. In addition to new tools and techniques, there is the emergence of non-free cloud services that offer a joint tooling for the purpose of managing data, in addition, offering quality feedback and other functionalities. Thus, the objective of this paper is to develop a native cloud platform to store, process and make available data using only open source tools. Case studies were implemented using the platform that was developed, with real solutions known in the market, such as the Clickstream application.

**Keywords**: Data Engineering; Data platform; Cloud native; Open source.

**LISTA DE FIGURAS**

[Figura 1 - Comparação desempenho formatos Apache Parquet e CSV. 17](#_Toc89624610)

[Figura 2 - Arquitetura do Sistema. 19](#_Toc89624611)

[Figura 3 – Implementação de uma DAG. 20](#_Toc89624612)

[Figura 4 - Representação gráfica de uma DAG. 21](#_Toc89624613)

[Figura 5 –Implementação *hook* para comunicação com HDFS. 21](#_Toc89624614)

[Figura 6 – Exemplo da invocação do *hook* que comunica com HDFS. 22](#_Toc89624615)

[Figura 7 - Dado persistido no HDFS. 22](#_Toc89624616)

[Figura 8 - Arquitetura do HDFS. 23](#_Toc89624617)

[Figura 9 - Persistência de uma tabela Hive. 23](#_Toc89624618)

[Figura 10 - Tabela Hive persistido no HDFS. 24](#_Toc89624619)

[Figura 11 - Estrutura da tabela Hive. 25](#_Toc89624620)

[Figura 12 - *Script* de envio de mensagens para tópicos Kafka. 26](#_Toc89624621)

[Figura 13 - Configuração do container Trino. 27](#_Toc89624622)

[Figura 14 - Configuração do catálogo de dados do Trino. 27](#_Toc89624623)

[Figura 15 - Definição do container do SQL Pad. 28](#_Toc89624624)

[Figura 16 - Resultado da consulta efetuada pelo SQL Pad. 28](#_Toc89624625)

[Figura 17 - API geradora de dados aleatórios. 30](#_Toc89624626)

[Figura 18 - Script da DAG ETL. 31](#_Toc89624627)

[Figura 19 - *Script* de agregação dos dados. 33](#_Toc89624628)

[Figura 20 - Operador de extração de dados. 34](#_Toc89624629)

[Figura 21 - Script de conversão de JSON para Parquet. 35](#_Toc89624630)

[Figura 22 - Arquivo manifesto do componente StatefulSet do Namenode HDFS 36](#_Toc89624631)

[Figura 23 - *Script* de iniciação da aplicação Clickstream. 37](#_Toc89624632)

[Figura 24 - *Script* de processamento da aplicação Clickstream. 38](#_Toc89624633)

[Figura 25 - Script de criação de esquema. 39](#_Toc89624634)

[Figura 26 - Script de postagem de eventos na API. 40](#_Toc89624635)

[Figura 27 - Requisição HTTP na API geradora de dados. 41](#_Toc89624636)

[Figura 28 - Representação gráfica da DAG ETL. 42](#_Toc89624637)

[Figura 29 - Painel de logs do Airflow. 43](#_Toc89624638)

[Figura 30 - Aplicação web do Spark e histórico de execuções. 44](#_Toc89624639)

[Figura 31 - Camada de dados brutos particionado pela data. 45](#_Toc89624640)

[Figura 32 - Dado bruto persistido na camada de dados brutos. 45](#_Toc89624641)

[Figura 33 – Dados da tabela insights persistido no formato Apache Parquet. 46](#_Toc89624642)

[Figura 34 - Dados da tabela grouped\_insights persistido no formato Apache Parquet. 46](#_Toc89624643)

[Figura 35 - Consulta SQL na tabela insights. 47](#_Toc89624644)

[Figura 36 - Consulta SQL na tabela grouped\_insights. 47](#_Toc89624645)

[Figura 37 - Resultado script de criação do esquema dos eventos Avro. 48](#_Toc89624646)

[Figura 38 - Resultado script de postagem de eventos na API. 48](#_Toc89624647)

[Figura 39 - Métricas de execução da aplicação Clickstream - Parte 1. 49](#_Toc89624648)

[Figura 40 - Métricas de execução da aplicação Clickstream - Parte 2. 49](#_Toc89624649)

[Figura 41 - Dados processados pelo Clickstream persistidos no Data Lake. 50](#_Toc89624650)

[Figura 42 - Tabela Hive contendo os dados persistidos pela aplicação Clickstream. 50](#_Toc89624651)

**SUMÁRIO**

[1. INTRODUÇÃO 10](#_Toc89624652)

[1.1. Objetivos do Trabalho 10](#_Toc89624653)

[1.2. Conteúdo do Trabalho 10](#_Toc89624654)

[2. FUNDAMENTAÇÃO TÉCNICA 12](#_Toc89624655)

[2.1. Plataforma de Dados 12](#_Toc89624656)

[2.2. Computação distribuída 12](#_Toc89624657)

[2.3. Data Lake 12](#_Toc89624658)

[2.4. Big Data 13](#_Toc89624659)

[2.5. ETL 13](#_Toc89624660)

[2.6. Apache Airflow 14](#_Toc89624661)

[2.7. Apache Spark 14](#_Toc89624662)

[2.8. Hadoop Distributed File System 15](#_Toc89624663)

[2.9. Apache Hive 15](#_Toc89624664)

[2.10. Event Streaming 15](#_Toc89624665)

[2.11. Apache Kafka 16](#_Toc89624666)

[2.12. Trino 16](#_Toc89624667)

[2.13. Apache Parquet 16](#_Toc89624668)

[2.14. Apache Avro 17](#_Toc89624669)

[2.15. Confluent Schema Registry 17](#_Toc89624670)

[2.16. Spark Structured Streaming 17](#_Toc89624671)

[2.17. Docker 18](#_Toc89624672)

[2.18. Plataforma Clickstream 18](#_Toc89624673)

[2.19. Kubernetes 18](#_Toc89624674)

[3. DESENVOLVIMENTO 19](#_Toc89624675)

[3.1. Arquitetura do Sistema 19](#_Toc89624676)

[3.2. Gerenciamento de pipelines com DAGs 20](#_Toc89624677)

[3.3. Armazenamento de dados 21](#_Toc89624678)

[3.4. Definição de metadados 23](#_Toc89624679)

[3.5. Ingestão de dados em tempo real 25](#_Toc89624680)

[3.6. Análise dos dados 27](#_Toc89624681)

[3.7. Estudo de casos 28](#_Toc89624682)

[3.7.1. ETL em lote 29](#_Toc89624683)

[3.7.2. ETL em tempo real 36](#_Toc89624684)

[4. RESULTADOS 41](#_Toc89624685)

[4.1. ETL em lote 41](#_Toc89624686)

[4.2. ETL em tempo real (Clickstream) 47](#_Toc89624687)

[5. CONSIDERAÇÕES FINAIS 51](#_Toc89624688)

[5.1. Contribuições e conclusões 51](#_Toc89624689)

[5.2. Trabalho Futuros 51](#_Toc89624690)

[REFERÊNCIAS 53](#_Toc89624691)

# 1. INTRODUÇÃO

Nos dias atuais, é cada vez mais comum empresas coletarem quantidades imensas de dados, que são gerados por usuários, ferramentas de auditoria que geram logs ou soluções que envolvem *IoT* (Internet das Coisas). Porém, para lidar com essa quantidade imensa de dados não é uma tarefa trivial. Dependendo da área de negócio, dados devem ser armazenados em lugares de alta disponibilidade, consistente e seguro.

Além da preocupação em armazenar os dados, é entendível que estes servirão para algum propósito, como por exemplo, entregar uma experiência personalizada para usuários, analisar e monitorar técnicas antifraude entre outras coisas. Neste momento que a área de processamento massivo de dados entra em questão.

O processamento de quantidades massivas de dados lida com alguns aspectos não tão triviais, por exemplo, alta capacidade computacional, computação distribuída e tolerante a falhas, o que caracteriza lidar com técnicas e ferramentas, conhecidas hoje como Big Data.

## 1.1. Objetivos do Trabalho

O objetivo geral deste trabalho é desenvolver uma plataforma de dados que proporcione a escrita de *scripts* capazes de extrair, transformar e armazenar dados em lote e em tempo real utilizando ferramentas de código aberto disponibilizadas pelo mercado de tecnologia.

Para a realização deste trabalho foram estabelecidos alguns objetivos específicos que envolvem a criação de uma plataforma de dados:

* Plataforma capaz de fazer processamentos de dados em lote e em tempo real;
* Utilizar apenas ferramentas de código aberto;
* Tornar a plataforma extensível e personalizável;
* Capaz de fazer análises históricas.

## 1.2. Conteúdo do Trabalho

Este trabalho está dividido entre os seguintes capítulos:

No Capítulo 2 é feita a fundamentação das tecnologias e dos conceitos para compreensão deste trabalho.

O Capítulo 3 apresenta detalhadamente o desenvolvimento da plataforma de extração, transformação e disponibilização de dados, assim como, os estudos de casos, como uma aplicação Clickstream.

No Capítulo 4 são apresentados os resultados alcançados a partir dos estudos de casos, e o painel de gerenciamento da infraestrutura da plataforma nativa em nuvem.

O Capítulo 5 apresenta as considerações finais deste trabalho a partir da análise dos resultados obtidos, e também, breves sugestões de trabalhos futuros.

# 2. FUNDAMENTAÇÃO TÉCNICA

Neste capítulo serão abordados os conceitos básicos que definem uma plataforma de dados e seus requisitos, e algumas das tecnologias que compõe a plataforma construída neste trabalho.

## 2.1. Plataforma de Dados

Segundo Digital (2021), Plataforma de Dados é um conjunto ferramental e de técnicas que buscam extrair dados de diversas fontes, alimentar e enriquecer tais dados, que por fim são armazenados em uma outra base de dados. A finalidade principal de uma plataforma de dados é centralizar os dados de uma organização, de modo que esteja em conformidade com leis de proteção de dados, aplicando inteligência para alavancar a sua estratégia.

Comumente, plataformas de dados incluem ferramentas de extração e transformação de dados que por fim são armazenados em tecnologias de armazenamento de dados em larga escala, conhecidos como *data lake*.

## 2.2. Computação distribuída

Computação distribuída é o termo para referir-se a sistemas que possuem múltiplos componentes de softwares executando em múltiplos computadores, mas executando apenas um sistema. Sistemas distribuídos podem constituir-se de múltiplos computadores localizados geograficamente no mesmo local conectados por uma rede local ou podem estar geograficamente distribuídos conectados por uma rede ampla. O objetivo da computação distribuída é fazer com que esses múltiplos computadores se comportem como apenas um. Alguns benefícios da computação distribuída incluem escalabilidade, podendo adicionar mais computadores a rede e redundância, onde, se um computador parar de funcionar, o sistema continuará executando (IBM, 2021a).

## 2.3. Data Lake

*Data lake* é termo que se refere a um tipo de repositório de dados que armazena quantidades massivas de dados estruturados ou não estruturados. Comumente, armazena-se dados brutos, ou dados não tratados, que quando são coletados não há uma finalidade para tal, e sim, definida posteriormente. Este dado pode ser utilizado por cientistas e analistas de dados ou aplicações que envolvem aprendizado de máquinas por diversas vezes e para diversos propósitos, o que facilita a sua reutilização (REDHAT, 2021a).

Algumas das principais diferenças entre *data lake* e *data warehouse*, que é outro tipo de ferramenta de armazenamento de dados para Big Data, é que *data lakes* armazenam dados não estruturados que são estruturados posteriormente (conhecido como "*schema on read*"). Também não são acoplados às ferramentas de alto custo computacional para otimização de leitura e controle de acesso concorrente à dados, o que minimiza gastos com hardware e software e flexibiliza a capacidade de armazenamento de dados.

## 2.4. Big Data

Big Data, refere-se à manipulação ou gerenciamento de uma larga quantidade de dados ou de dados muito complexos. Big Data tornou-se conhecido pelos "3 Vs" de variedade, velocidade e volume. Volume refere-se à quantidade massiva de dados, variedade à ampla gama de formatos não padronizados, velocidade refere-se à necessidade de processar com rapidez e eficiência (REDHAT, 2021a).

O principal objetivo de Big Data é oferecer informações em tempo real que podem ser usadas para alavancar a estratégia da organização. O processamento de informações em tempo real é o principal objetivo das organizações que buscam agregar valor aos seus clientes de forma consistente e contínua, o que se encaixa em um dos pilares de computação de ponta.

## 2.5. ETL

ETL, ou Extrair (*Extract*), Transformar (*Transform*) e Armazenar (*Load*) é um processo que extrai, transforma e armazena dados de múltiplas fontes para um *data warehouse* ou outro repositório unificado de dados (IBM, 2021b).

No processo de extração, o dado é copiado de uma fonte e movido para outro local intermediário, conhecido como *staging area*. O dado pode ser extraído de uma fonte de dados estruturado ou não estruturado, por exemplo, banco de dados transacionais SQL ou não SQL, sistemas ERP, e-mails, API REST e outros.

No processo de transformação o dado bruto pode ser transformado em um formato útil que posteriormente é utilizado para análises. O processo de transformação pode envolver filtros, limpezas, duplicações, validações, cálculos, traduções, agregações, anonimizações para governança de dados entre outras coisas. Essa etapa de transformação normalmente ocorre em um ambiente computacional diferente da fonte, para assim, limitando o impacto do desempenho nos sistemas de origem e reduz a probabilidade de corrupção de dados.

No processo de armazenamento, o último passo, o dado é movido da *staging area* para um *data warehouse*, por exemplo. Em fluxos ETL, inicialmente envolve o armazenamento de todo o dado, seguindo por carregamentos de alterações de dados incrementais periódicos. Também, pode haver a necessidade, menos comum, do reprocessamento dos dados o que acaba envolvendo o carregamento total dos dados novamente.

Algumas ferramentas conhecidas na comunidade de engenharia de dados são Airflow, Luigi, Argo, Kubeflow e outros, aos quais abordam cada passo do processo ETL de forma separada.

## 2.6. Apache Airflow

Apache Airflow é uma plataforma de código aberto para criar, agendar e monitorar fluxos de trabalho. Airflow é uma ferramenta principalmente utilizada para fins ETL seguindo o conceito de DAGs (*Directed Acyclic Graphs*). Assim, as tarefas atribuídas aos fluxos são executadas por trabalhadores seguindo uma dependência especificada (APACHE, 2021 a).

A ferramenta Airflow permite a integração com diversas outras ferramentas que auxiliam no processo de ETL, como por exemplo, Apache Spark para transformação de dados de forma distribuída.

## 2.7. Apache Spark

Apache Spark, é um mecanismo de processamento de dados em grande escala. Spark utiliza *caching* em memória, a grande diferença do seu concorrente Hadoop, e execução de consulta otimizada para consultas rápidas em dados de qualquer tamanho. Alguns componentes disponibilizados pelo Spark, como por exemplo Spark Streaming e Spark Core, possibilitam processamento de dados em lotes e em tempo real (APACHE, 2021b).

## 2.8. Hadoop Distributed File System

Hadoop Distributed File System ou HDFS, segundo IBM (2021c), é um sistema de arquivos distribuído que lida com grandes conjuntos de dados executados em hardware comum. Ele é usado para dimensionar um único *cluster* Apache Hadoop para centenas (e até milhares) de nós. HDFS é um dos principais componentes do Apache Hadoop.

Com HDFS os dados podem ser particionados e armazenados em diferentes nós/servidores. O particionamento em diversos nós gera a replicação de dados, o que implica na redundância também. A redundância pode ser aumentada ou diminuída, conforme o caso de uso da aplicação. Portanto, a redundância oferece múltiplos benefícios, sendo que o mais óbvio é a alta disponibilidade, um dos principais pilares de computação distribuída.

## 2.9. Apache Hive

Apache Hive é uma ferramenta de *data warehouse* que facilita a leitura, escrita e gerenciamento de grandes conjuntos de dados que residem em armazenamento distribuído e consultados usando a sintaxe SQL (APACHE, 2021c).

Hive disponibiliza algumas ferramentas como:

* Acesso a dados via instruções SQL, portanto, permite tarefas de extração, transformação e armazenamento;
* Mecanismo de armazenamento de dados estruturados em uma variedade de formatos;
* Acesso à arquivos armazenados diretamente no Apache HDFS ou HBase;
* Execução de consultas via Apache Spark ou MapReduce;
* Componente para armazenamento de metadados, ou descritores de arquivos

Apache Hive foi projetado para maximizar a escalabilidade (escalar horizontalmente com mais máquinas adicionadas dinamicamente ao *cluster* Hadoop), desempenho, extensibilidade, tolerância a falhas e acoplamento fraco com seus formatos de entrada.

## 2.10. Event Streaming

O *event streaming* é um termo relacionado a prática de capturar dados em tempo real de fontes de eventos como bancos de dados, sensores, dispositivos móveis, serviços em nuvem e aplicativos de software na forma de fluxos de eventos para persistir, processar e reagir a estes eventos em tempo real e retrospectivamente. O *streaming* de eventos, portanto, garante um fluxo contínuo e interpretação dos dados para que as informações certas estejam no lugar certo, na hora certa (APACHE, 2021d).

Algumas das principais ferramentas de *event streaming* no mercado, nos dias de hoje, são Apache Kafka, AWS Kinesis, ActiveMQ, RabbitMQ entre outros.

## 2.11. Apache Kafka

Apache Kafka é uma plataforma de *streaming* de eventos distribuídos de código aberto para *pipelines* de dados de alto desempenho, análise de *streaming*, integração de dados e aplicativos de missão crítica. Kafka é comumente utilizado no processo ETL na etapa de extração via transmissão de eventos de diversas fontes para uma área intermediária, conhecida como *staging area* (APACHE, 2021e).

## 2.12. Trino

Trino é um mecanismo de consulta SQL distribuído projetado para consultar grandes conjuntos de dados distribuídos. Trino é utilizado principalmente para trabalhar com terabytes ou petabytes de dados armazenados em fontes diversas TRINO, 2021).

Trino foi projetado para manipular análises e *data warehousing*: análises de dados, agregações de quantidades massivas de dados. Essas cargas de trabalho são sempre classificadas como *Online Analytical Processing* (OLAP).

## 2.13. Apache Parquet

Apache Parquet é um formato de armazenamento colunar disponível no ecossistema Hadoop, independentemente da escolha da estrutura de processamento de dados, modelo de dados ou linguagem de programação.

Apache Parquet permite armazenar arquivos contendo a definição do *schema* no próprio arquivo com alguns metadados. Além disso, é possível armazenar a declaração do particionamento de dados. Com o benefício de armazenamento colunar do Apache Parquet, os custos de armazenamento, leitura e processamento dos dados reduzem significativamente, pois os dados são compactados e a leitura é feita apenas nas colunas necessárias para retornar uma consulta (APACHE, 2021f).

A Figura 1 ilustra a comparação na leitura de 1 TB de dados em um arquivo CSV e no formato Parquet.

Figura 1 - Comparação desempenho formatos Apache Parquet e CSV.

Tabela

Descrição gerada automaticamente

Fonte: Adaptada de Spicer (2017).

## 2.14. Apache Avro

Apache Avro é formato binário de armazenamento de dados. Apache Avro resolve alguns dos principais problemas quando se trata de evolução de estrutura dos dados. Avro armazena no próprio arquivo ou mensagem a estrutura do dado, então, aplicações antigas conseguem ler estruturas novas e, aplicações novas conseguem ler estruturas antigas. Além disso, Avro é armazenado em formato binário, o que o torna compacto e eficiente (IBM, 2021c).

## 2.15. Confluent Schema Registry

Confluent Schema Registry é uma camada provedora de metadados. Disponibiliza uma API para armazenar a definição de *schemas* (estrutura dos dados) no formato Avro, JSON ou Protobuf. Com Schema Registry torna-se capaz armazenar versões de cada *schema*, o que permite visualizar o histórico de versões dos *schemas* (CONFLUENT, 2021).

## 2.16. Spark Structured Streaming

Spark Structured Streaming é um mecanismo de processamento em tempo real que utiliza Spark SQL para criar fluxos de processamento de dados da mesma maneira que fluxo de processamento em lote são criados. Spark Structured Streaming é responsável por adicionar todos os dados em um fluxo de *streaming* e adicionar ao final de uma DataFrame ou Dataset, que é componente do mecanismo Spark SQL (APACHE, 2021g).

## 2.17. Docker

Docker é uma plataforma de código aberto que permite desenvolvedores virtualizar componentes e aplicações utilizando a técnica de containers e imagens. Docker permite empacotar diversas aplicações e suas respectivas configurações em um único artefato - a imagem, que pode ser implantada e replicada diversas vezes minimizando o custo de manutenção (DOCKER, 2021).

## 2.18. Plataforma Clickstream

Plataforma Clickstream, ou software de Clickstream é um produto que coleta todo o fluxo da jornada online de usuários em uma plataforma ou website (POWIS, 2021). Normalmente plataformas de Clickstream são focados em um único website e geralmente mostram o progresso dos usuários desde o momento em que ingressam no website até o momento que fecham a compra de um produto. Dados gerados por esse tipo de plataforma são comumente utilizados para análise do comportamento de usuários (POWIS, 2019).

## 2.19. Kubernetes

Kubernetes é um sistema de código aberto para orquestração de containers na nuvem, que disponibiliza diversas funcionalidades para automatização de atividades envolvendo gerenciamento da infraestrutura de containers, como por exemplo, escalabilidade horizontal de recursos, orquestração de componentes de persistência de dados, balanceamento de carga, gerenciamento de segredos etc. Kubernetes baseia-se em 15 anos de experiência do Google gerenciando e executando aplicações em produção combinado com as melhores práticas do mercado (KUBERNETES, 2021).

# 3. DESENVOLVIMENTO

Neste capítulo serão abordados o processo de construção e o desenvolvimento da plataforma de dados, tanto quanto o de definição da arquitetura da plataforma.

## 3.1. Arquitetura do Sistema

A Figura 2 mostra a arquitetura a partir da qual foi implementada a plataforma de dados. Na arquitetura, destaca-se 4 camadas - Airflow juntamente com Spark definindo a camada de processamento, ou seja, onde ocorre o gerenciamento de *pipelines* ETL em lote. Hadoop (HDFS) e Hive Metastore como camada de armazenamento ou *data lake*, tanto os dados/informações em si, como também o catálogo de dados. Trino compõe a camada de análise, onde é possível executar instruções utilizando-se da sintaxe SQL para a exploração analítica dos dados. Por fim, a aplicação escrita em Javascript (Node.js), Apache Kafka e Spark Streaming compõem a camada de ingestão e processamento de dados em tempo real.

Figura 2 - Arquitetura do Sistema.

Diagrama, Linha do tempo

Descrição gerada automaticamente

Fonte: Elaborado pelo Autor (2021).

## 3.2. Gerenciamento de pipelines com DAGs

Para definir, executar e gerenciar tarefas previamente agendadas utilizou-se a ferramenta Apache Airflow. Tal ferramenta permite-nos escrever diferentes *scripts* na linguagem Python que se denominam tarefas. Um conjunto de tarefas compõem o que se chama na literatura de pipelines ETL, as DAGs. As DAGs nos permitem definir fluxos de execução complexos, não limitando-se apenas a execução de códigos locais, mas também a invocação de aplicação remotas e a chamada de APIs externas entre outros tipos de soluções.

A Figura 3 mostra a implementação de uma DAG simples, onde a primeira tarefa é uma função Python que recebe dois números e retorna a soma deles. A segunda tarefa é outra função Python que apenas mostra o texto “ok” e retorna o mesmo.

Figura 3 – Implementação de uma DAG.

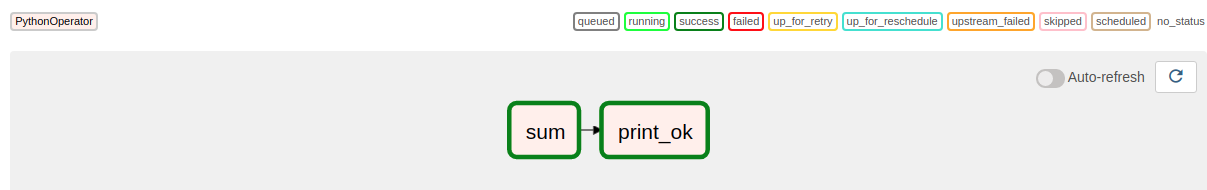
Texto

Descrição gerada automaticamente

Fonte: Elaborada pelo autor (2021).

A Figura 4 mostra de forma visual como as tarefas anteriormente definidas formam um grafo acíclico direcionado.

Figura 4 - Representação gráfica de uma DAG.



Fonte: Elaborada pelo autor (2021).

## 3.3. Armazenamento de dados

Para compor e alimentar o *data lake*, utilizou-se a ferramenta Apache Hadoop (HDFS). Para a inserção de dados no HDFS via tarefas definidas no Airflow, fez-se necessário a criação de um *hook* - um *script* personalizado que se comunica internamente com uma API do HDFS. A Figura 5 mostra a implementação do código *hook* para a comunicação com a API do HDFS.

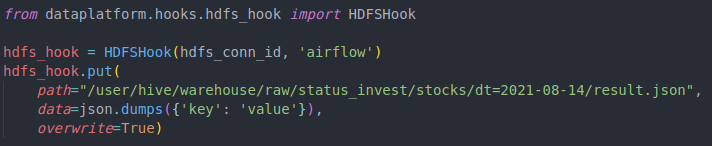
Figura 5 –Implementação *hook* para comunicação com HDFS.



Fonte: Elaborado pelo autor (2021).

Por fim, para inserir os dados no HDFS fez-se necessário apenas invocar o *hook* e passar quais os dados a serem inseridos. A Figura 6 ilustra um exemplo da invocação do *hook* e a inserção de um dado no formato JSON e a Figura 7 mostra o dado já persistido no HDFS.

Figura 6 – Exemplo da invocação do *hook* que comunica com HDFS.



Fonte: Elaborado pelo autor (2021).

Figura 7 - Dado persistido no HDFS.

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

Fonte: Elaborado pelo autor (2021).

O pilar de distribuição de dados se dá pela forma como o HDFS é implementado. A arquitetura é definida por dois componentes, sendo eles, Namenode e Datanode.

Namenode caracteriza-se pela interface com o cliente, onde manipula-se os dados e comporta-se como um orquestrador. Tal componente responsabiliza-se pelo recebimento das chamadas de leitura e escrita de dados e define em qual nó (Datanode) o dado será persistido ou será consultado para disponibilização. Namenode armazena somente metadados, como por exemplo, nome e caminho de arquivos, número de blocos escritos, número de réplicas e outras informações relacionadas aos Datanodes.

Datanodes, como citado anteriormente, são orquestrados pelo Namenode, e sua função é persistir e disponibilizar bloco de dados que são requisitados pelo Namenode. A Figura 8 ilustra a arquitetura do HDFS.

Figura 8 - Arquitetura do HDFS.

Diagrama

Descrição gerada automaticamente

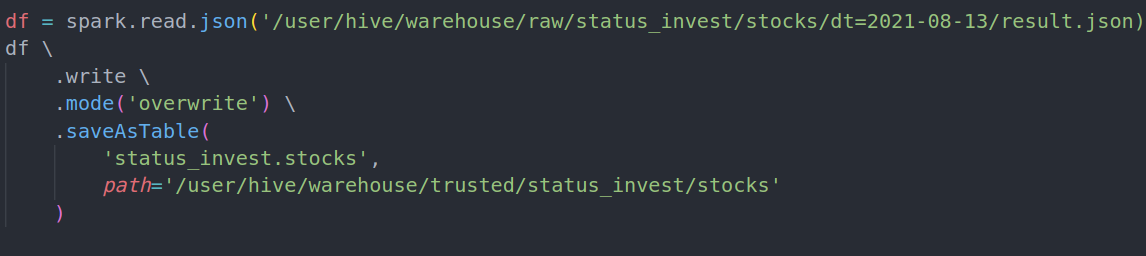
Fonte: Adaptada de Ray (2021)

## 3.4. Definição de metadados

Para a estruturação e definição dos dados, ou seja, a definição de banco de dados, as tabelas e suas respectivas colunas, fez-se necessário a utilização da ferramenta Apache Hive juntamente com Apache Spark para a definição de *scripts* que recebem um caminho de arquivo em um formato conhecido, como por exemplo JSON, e o converte para outro formato conhecido, Apache Parquet, para posterior consulta via uma ferramenta de análise de dados.

A Figura 9 mostra um trecho de código que recebe o caminho de um arquivo no formato JSON e o persiste novamente em outro caminho no formato Parquet.

Figura 9 - Persistência de uma tabela Hive.



Fonte: Elaborado pelo autor (2021).

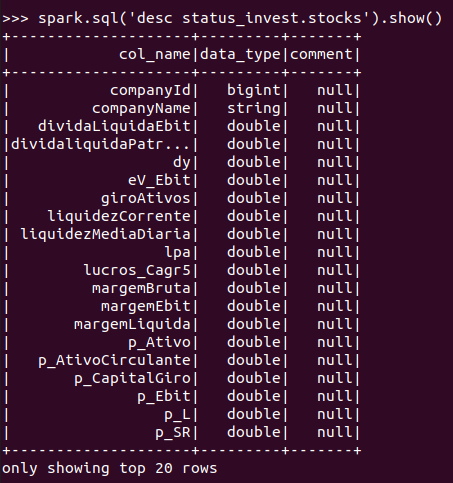
O *script* não somente persiste o dado em outro formato, mas também cria os metadados de definição dos dados, como por exemplo, a estrutura da tabela. A criação dos metadados é feita internamente pela API do Apache Spark. As Figuras 10 e 11 mostram o dado persistido no HDFS e a estrutura da tabela criada, respectivamente.

Figura 10 - Tabela Hive persistido no HDFS.

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente Fonte: Elaborado pelo autor (2021).

Figura 11 - Estrutura da tabela Hive.



Fonte: Elaborado pelo autor (2021).

## 3.5. Ingestão de dados em tempo real

Uma plataforma de dados que faz processamento de dados em tempo real necessita de uma camada para ingestão de dados em tempo real. Para prover a capacidade de ingerir dados em tempo real, criou-se uma aplicação escrita em Javascript utilizando a plataforma Node.js para prover uma API HTTP onde é passado o título e o dado da mensagem no corpo da requisição HTTP.

A Figura 12 apresenta a classe Javascript implementada para receber o título da mensagem, que condiz com o nome do tópico Kafka onde será postada a mensagem, e o corpo da mensagem.

Figura 12 - *Script* de envio de mensagens para tópicos Kafka.



Fonte: Elaborada pelo autor (2021).

É importante notar que, antes da postagem da mensagem no tópico Kafka é feita a codificação da mensagem para um formato conhecido, Apache Avro. Internamente, após a codificação da mensagem, também, é validado se a estrutura da mensagem está em conformidade com um esquema registrado no repositório de esquemas, o então Confluent Schema Registry. Só então a mensagem é de fato postada no Kafka.

## 3.6. Análise dos dados

Utilizou-se o componente Trino com o intuito de disponibilizar aos usuários a possibilidade de consultar os dados persistidos no *data lake* usando uma sintaxe conhecida, como por exemplo SQL.

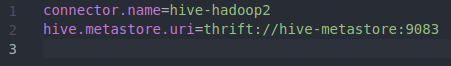
Para disponibilizar o ambiente provido do componente Trino, usou-se a ferramenta Docker para subir um *container* com a imagem do Trino e um arquivo de configuração para expor os dados armazenados como um catálogo de dados. As Figuras 13 e 14, ilustram a configuração do ambiente com Trino e do catálogo de dados, respectivamente.

Figura 13 - Configuração do container Trino.



Fonte: Elaborado pelo autor (2021)

Figura 14 - Configuração do catálogo de dados do Trino.

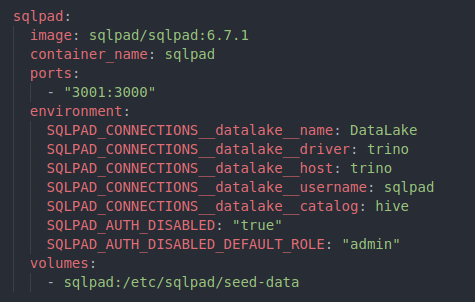


Fonte: Elaborado pelo autor (2021).

Por fim, para disponibilizar uma interface intuitiva para o usuário realizar consultas SQL, usou-se o componente SQL Pad que interage com o catálogo de dados definido no Trino.

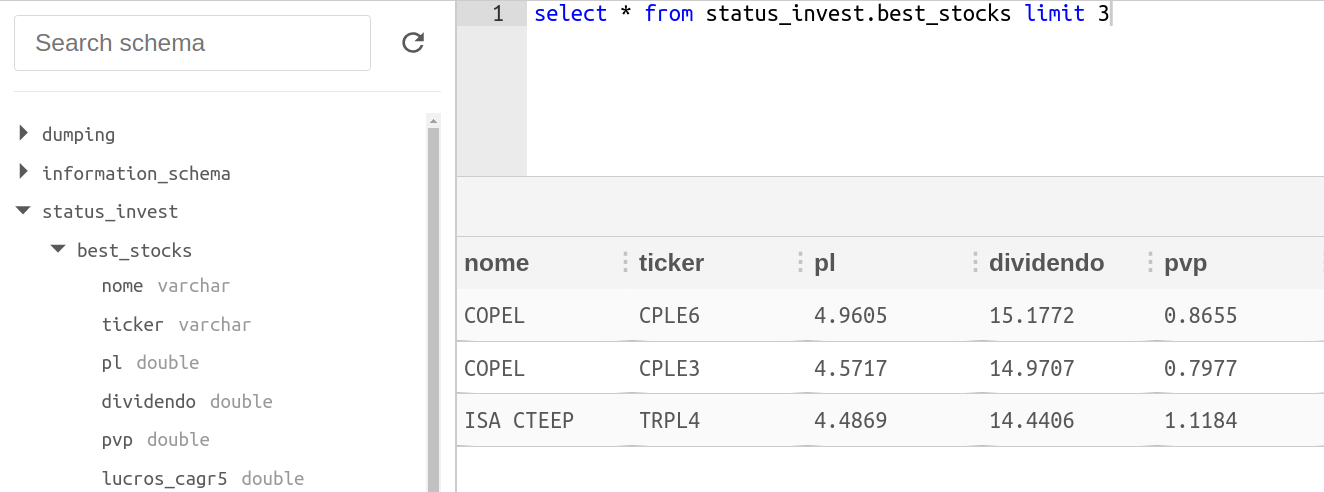
Para disponibilizar o ambiente provido do componente SQL Pad, foi usada, também, a ferramenta Docker para subir um container com a imagem contendo o SQL Pad. As Figuras 15 e 16 mostram a definição do container do SQL Pad e o resultado de uma consulta realizada através do SQL Pad, respectivamente.

Figura 15 - Definição do container do SQL Pad.



Fonte: Elaborado pelo autor (2021).

Figura 16 - Resultado da consulta efetuada pelo SQL Pad.



Fonte: Elaborado pelo autor (2021).

## 3.7. Estudo de casos

Nesta seção serão apresentados estudo de casos envolvendo fluxos ETL em lote, e em tempo real.

## 3.7.1. ETL em lote

Para a demonstração de um fluxo ETL em lote de ponta-a-ponta, implementou-se um fluxo completo de extração, de transformação e de persistência utilizando uma API escrita na linguagem Javascript, e *scripts* de extração e transformação com a biblioteca PySpark

Com o objetivo de expor dados para extração, criou-se uma API para gerar um conjunto de dados aleatórios simulando um evento de clique, visualização ou submissão de um formulário por parte de um usuário.

A Figura 17 apresenta o código de implementação da API geradora de dados aleatórios, que na chamada da API é passado o parâmetro nomeado como date e um limitador da quantidade de registros por chamada, nomeado como count. A partir da linha 28 a 35 implementou-se o método responsável por gerar um objeto JavaScript contendo os campos username, page e event\_name com valores aleatórios. O campo event\_time é a conversão do parâmetro date para o formato de Timestamp. Da linha 19 a 26 foi implementado o método responsável por gerar N objetos JavaScript utilizando o método anteriormente citado, onde N corresponde ao parâmetro count. Por fim, o método envia uma resposta de volta ao cliente invocador da API HTTP.

Figura 17 - API geradora de dados aleatórios.

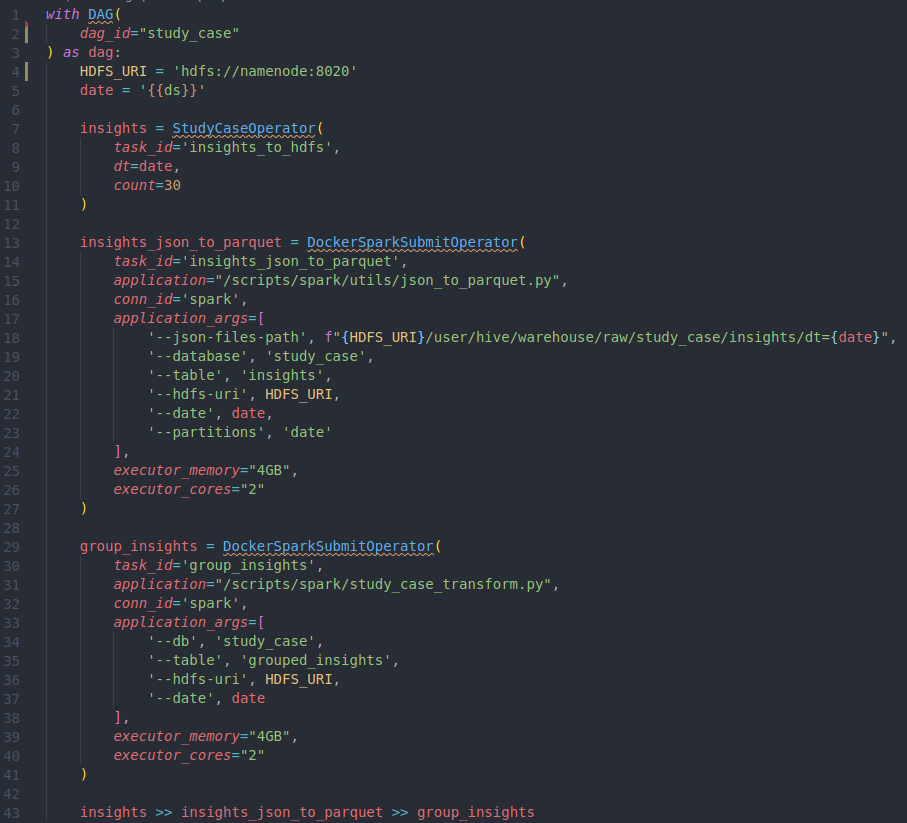
Texto

Descrição gerada automaticamente

Fonte: Elaborado pelo autor (2021).

Para extrair os dados da API anteriormente citada, criou-se um *script* de DAG contendo 3 tarefas para definição do fluxo ETL, conforme mostra a implementação do *script* ilustrada na Figura 18.

Figura 18 - Script da DAG ETL.



Fonte: Elaborado pelo autor (2021).

A partir da linha 7 a 11, como ilustrado na Figura 18, é definida a primeira tarefa nomeada como insights\_to\_hdfs. Na primeira tarefa é utilizado um operador personalizado - StudyCaseOperator, responsável por fazer uma chamada a API geradora de dados aleatórios e persistir a resposta da chamada no *data lake*. É importante ressaltar que os dados da resposta são persistidos em uma camada de dados brutos, sem nenhuma transformação. A segunda tarefa, implementada da linha 13 a 27 nomeada como insights\_json\_to\_parquet, é responsável por transformar o dado bruto extraído da API no formato JSON para o formato colunar Parquet, como também a criação de uma tabela no catálogo de dados do Hive Metastore. Nessa tarefa é utilizada outro operador personalizado, responsável por submeter a execução de *scripts* para o *cluster* Spark.

A terceira e última tarefa nomeada como group\_insights, definida da linha 29 a 41 (Figura 18) é responsável por realizar transformações no dado já catalogado, aplicando transformações com intuito de extrair informações analíticas. Essa tarefa também utiliza o mesmo operador da tarefa anterior, que faz submissão de *scripts* para o *cluster* Spark.

Na Figura 19 é ilustrada a implementação do *script* de transformação. Na linha 4 a 8 foi implementado o código para receber como argumentos a data ao qual o script irá extrair os dados. Na linha 10 a 26 do script é iniciado uma instância da aplicação Spark como também a instrução SQL que aplica as transformações nos dados. Para simular a obtenção de alguma informação analítica do dado, o *script* aplica uma agregação dos dados de uma determinada data, especificada pela cláusula WHERE date = '%s', e por fim, agrupa pelo campo event\_name e faz a contagem de cada tipo de evento. Na linha 28 a 36 a instrução é de fato executada, e o resultado da transformação é persistido no *data lake*, porém em outra camada - de dados analíticos.

Figura 19 - *Script* de agregação dos dados.

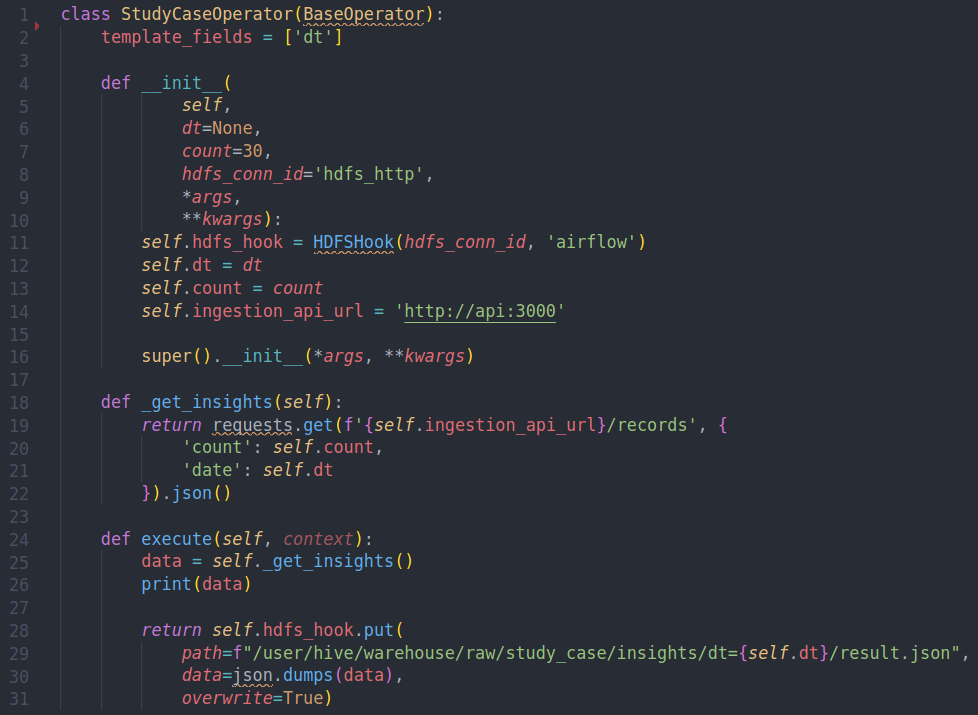


Fonte: Elaborado pelo autor (2021).

A Figura 20 apresenta a implementação da classe operador personalizado anteriormente citado, nomeado como StudyCaseOperator. Na linha 4 a 16 é definido o construtor da classe operadora que define as variáveis iniciais contendo a data de execução do *script*, o contador de objetos e o endereço da API geradora de dados aleatórios. Na linha 18 a 22 é definido o conjunto de códigos que realiza a chamada HTTP à API geradora de dados, passando como parâmetro a data e o contador de objetos definidos no construtor da classe. É utilizado o pacote requests nativo do Python.

Para a implementação do código executor da classe operadora, na linha 24 a 31 (Figura 20) foi implementado o método execute que realiza a chamada HTTP a API e persiste o resultado da chamada no formato JSON na camada de dados brutos do HDFS.

Figura 20 - Operador de extração de dados.

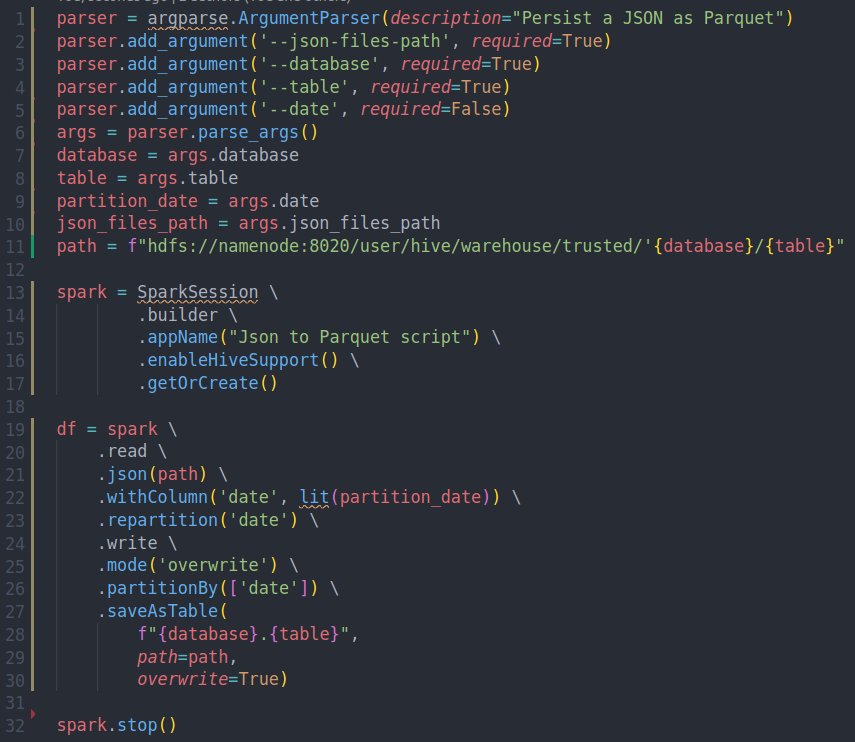


Fonte: Elaborado pelo autor (2021).

Na Figura 21 é ilustrado o *script* de transformação de dados no formato JSON para o formato Parquet, citado anteriormente na implementação da segunda tarefa ETL. Na linha 1 a 17 são definidas as variáveis contendo o caminho do arquivo JSON, o nome do banco de dados e tabela onde o dado Parquet será persistido, a data de execução do script e a instância da aplicação Spark.

Na linha 19 a 22 inicia-se a leitura do arquivo JSON, como também a adição de uma nova coluna contendo a data de execução do script, anteriormente definida. A partir da linha 23 a 30 inicia-se a persistência dos novos arquivos no formato Parquet, como também a especificação do nome do banco de dados, tabela e o caminho de onde os arquivos serão persistidos. Por fim, na linha 32, a aplicação Spark é finalizada.

Figura 21 - Script de conversão de JSON para Parquet.



Fonte: Elaborado pelo autor (2021).

Para disponibilizar uma plataforma *cloud native* (nativa em nuvem) e atingir as boas práticas de orquestração de containers do mercado, todos os componentes da arquitetura deste trabalho foram implantados utilizando a ferramenta Kubernetes. A Figura 22 apresenta o arquivo manifesto que implementa o componente StatefulSet do Namenode HDFS.

Figura 22 - Arquivo manifesto do componente StatefulSet do Namenode HDFS

Texto

Descrição gerada automaticamente

Fonte: Elaborado pelo autor (2021).

## 3.7.2. ETL em tempo real

Para a demonstração de um fluxo ETL em tempo real de ponta-a-ponta, implementou-se um fluxo de Clickstream utilizando a biblioteca PySpark. O objetivo do experimento foi enviar dados aleatórios para uma API HTTP, implementada na seção 3.5. A partir da API, postaram-se os dados em um tópico Kafka, onde conectou-se uma aplicação Clickstream para receber os eventos e agrupá-los em uma janela de tempo definida de 5 segundos. A cada janela de tempo, os eventos foram agrupados e, por fim, contabilizado a quantidade de eventos de cada usuário na respectiva janela de tempo.

As Figuras 23 e 24 apresentam o código da aplicação Clickstream separado em duas partes. O código da aplicação foi implementado utilizando a biblioteca PySpark juntamente com o framework Spark Structured Streaming.

Na Figura 23 que ilustra a primeira parte da aplicação, no trecho de código da linha 5 a 13 é definido a estrutura dos dados que são transmitidos no tópico Kafka utilizando a sintaxe de definição do formato Avro. Da linha 18 a 22 é instanciado a aplicação Spark que será executada. Da linha 24 a 30 é instanciado um objeto DataStreamReader onde são lidos e processados cada evento que é transmitido no tópico Kafka especificado na linha 29.

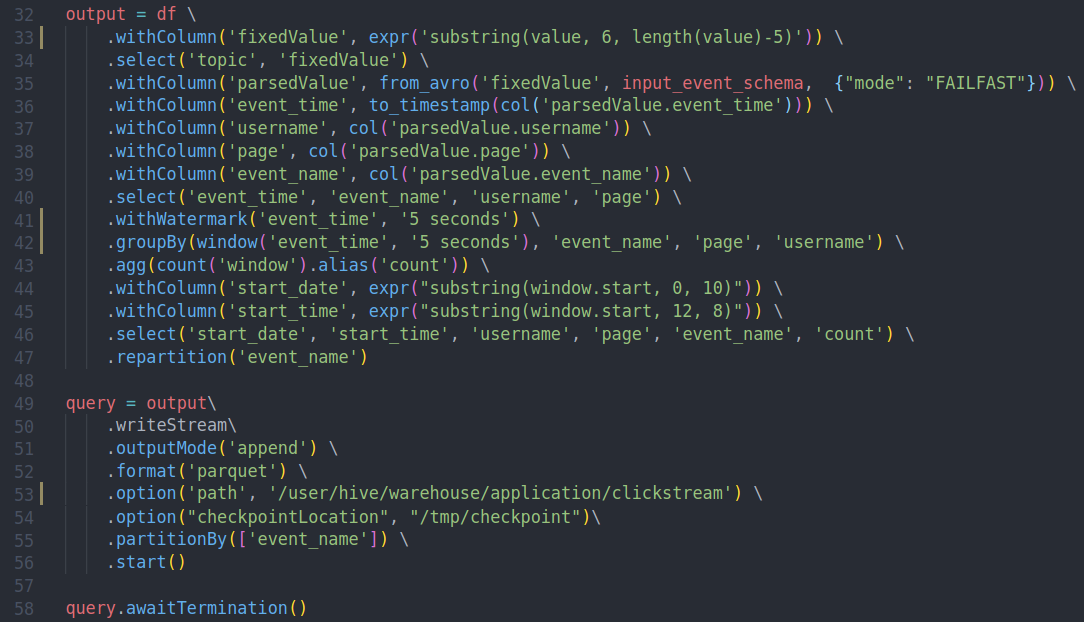
Figura 23 - *Script* de iniciação da aplicação Clickstream.



Fonte: Elaborado pelo autor (2021).

Na Figura 24, no trecho de código apresentado da linha 32 a 40 são mostradas algumas transformações no evento recebido pelo DataStreamReader. Na linha 33 é removido caracteres mágicos e extraído a mensagem Avro convertida para JSON. Da linha 35 a 40 cada campo do objeto JSON é anexado a sua respectiva coluna contendo o nome do campo. No trecho de código da linha 41 a 47 é aplicado a função de janelamento, onde cada janela contém eventos dentro de um período de 5 segundos. Tais eventos são agrupados pelas colunas event\_name, page e username que, por fim, é aplicada uma contagem da quantidade de eventos de cada grupo, adicionando-se o campo start\_date e start\_time contendo o ano-mês-dia e hora-minuto-segundo da janela, respectivamente. Por fim, da linha 49 a 56 é criado uma instância do objeto DataStreamWriter, onde os dados anteriormente processados são preparados para serem persistidos no *data lake*, no formato Parquet e no caminho de arquivo especificado na linha 53.

Figura 24 - *Script* de processamento da aplicação Clickstream.

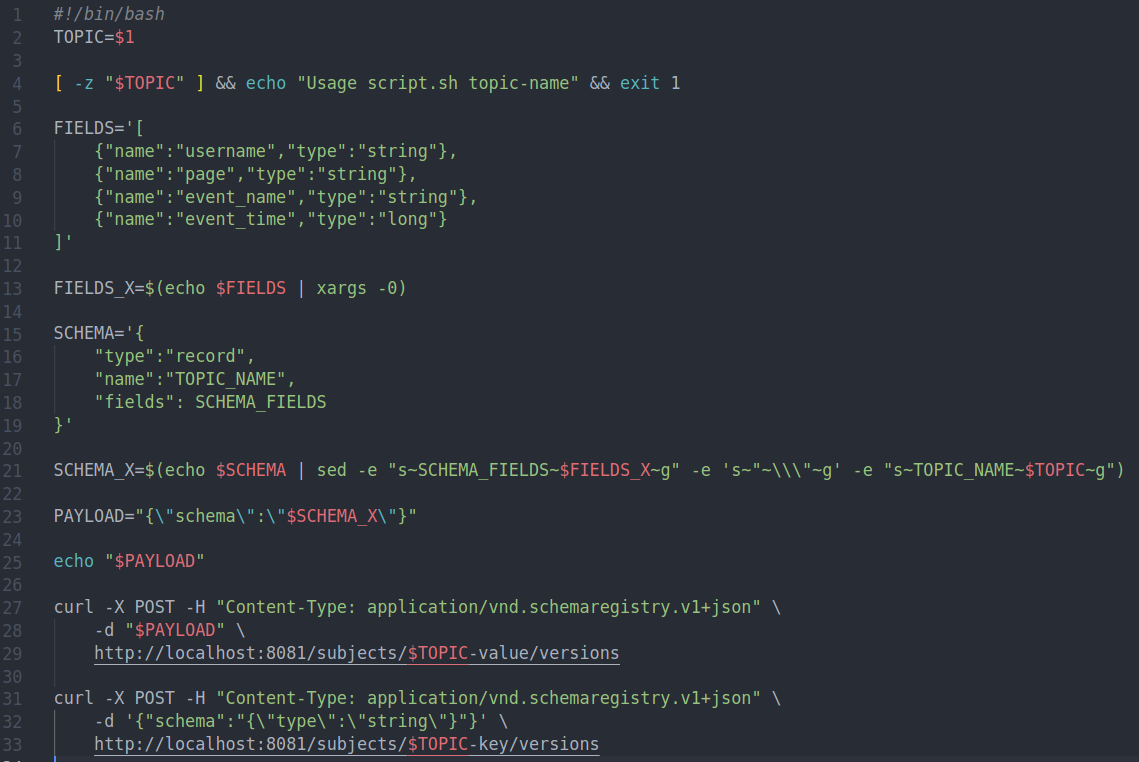


Fonte: Elaborado pelo autor (2021).

Para a definição e criação da estrutura de dados dos eventos postados no tópico Kafka anteriormente citado, necessitou-se a criação de um *script* para a criação da estrutura do dado no repositório de esquemas - Schema Registry.

Na Figura 25 é ilustrado o *script* de criação de esquemas no formato Avro. No trecho de código da linha 5 a 23, é definida a estrutura do dado e o corpo da requisição de criação de esquema a ser enviado para a API do Schema Registry. Da linha 27 a 33 é realizada a chamada criação de esquema a API, passando como corpo da requisição a estrutura definida anteriormente.

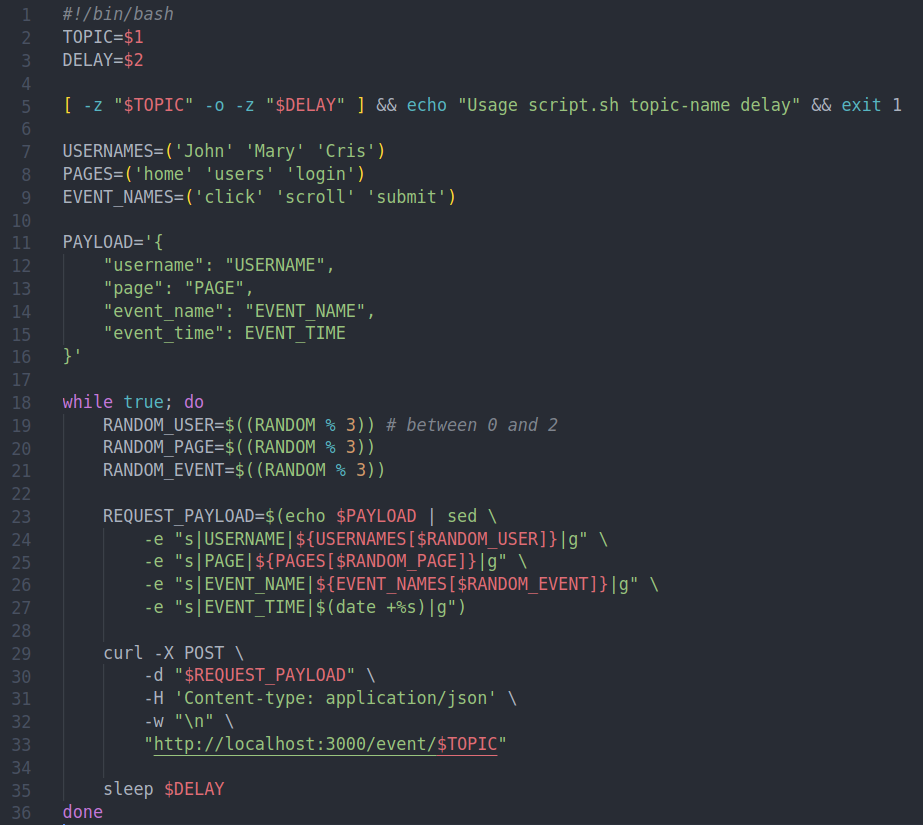
Figura 25 - Script de criação de esquema.



Fonte: Elaborado pelo autor (2021).

Para fazer a postagem dos eventos na API HTTP, citada na seção 3.5, criou-se um *script* ilustrado na Figura 26 com a utilização do formato do dado anteriormente definido. No trecho de código da linha 7 a 9 são definidos conjuntos de dados a serem sorteados por um número aleatório. Da linha 11 a 16 é definido o modelo do corpo da requisição a ser enviado a API citada na seção 3.5. Da linha 18 a 36 é definido um *loop* infinito, onde são sorteados aleatoriamente os dados anteriormente definidos. Com os dados sorteados, os campos do modelo do corpo da requisição são preenchidos e por fim, é feito a postagem da mensagem a API.

Figura 26 - Script de postagem de eventos na API.



Fonte: Elaborado pelo autor (2021).

# 4. RESULTADOS

Neste capítulo serão apresentados os resultados do desenvolvimento de uma plataforma para extração, transformação e disponibilização de dados dos estudos de casos anteriormente apresentados.

## 4.1. ETL em lote

Nesta seção serão apresentados os resultados obtidos a partir da criação de um fluxo ETL em lote implementado na seção 3.7.1.

A Figura 27 apresenta o resultado da chamada HTTP feita na API geradora de dados aleatórios através do navegador. Na chamada são passados os parâmetros de data (date) e quantidade de objetos (count). A resposta retorna os dados necessários para realizar as transformações posteriores.

Figura 27 - Requisição HTTP na API geradora de dados.

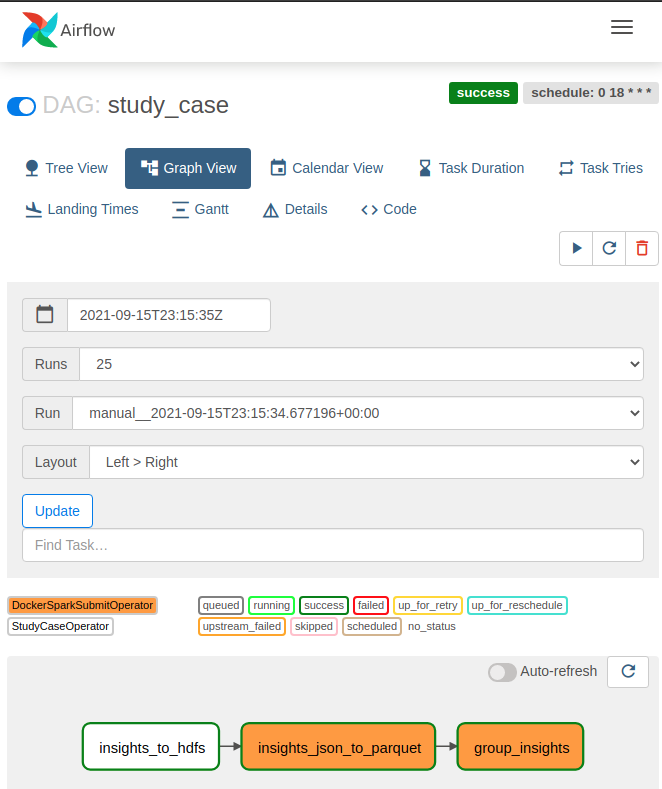
[Texto

Descrição gerada automaticamente](https://github.com/jasondavindev/open-dataplatform/blob/main/doc/images/api_random_data.png)

Fonte: Elaborado pelo autor (2021).

Na Figura 28 é ilustrado a representação gráfica da DAG no Airflow implementada na seção 3.7.1. Na figura é possível visualizar as 3 tarefas que compõem o fluxo ETL e que todas as tarefas foram executadas com sucesso, representado pela borda na cor verde como também o marcador *success*.

Figura 28 - Representação gráfica da DAG ETL.

[](https://github.com/jasondavindev/open-dataplatform/blob/main/doc/images/airflow_run_dags.png)

Fonte: Elaborado pelo autor (2021).

Navegando pelo painel de tarefas do Airflow, é possível visualizar os *logs* das execuções de tais tarefas. Na Figura 29 são ilustrados os *logs* da primeira tarefa, onde é apresentado a resposta retornada da chamada à API de geração de dados como também uma mensagem mencionando onde o arquivo resultado foi persistido.

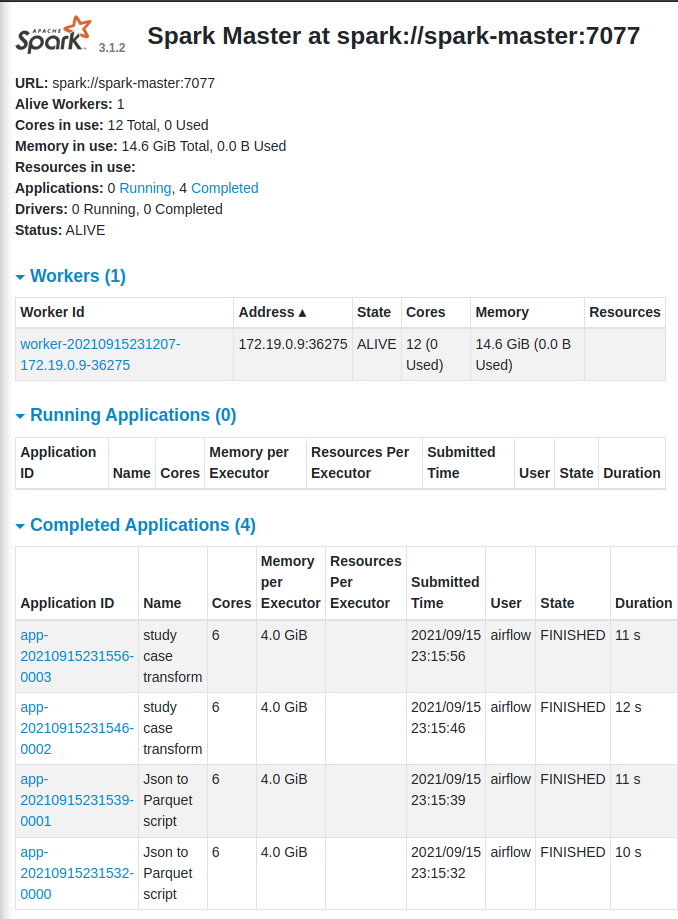
Figura 29 - Painel de logs do Airflow.

[](https://github.com/jasondavindev/open-dataplatform/blob/main/doc/images/airflow_logs.png)

Fonte: Elaborado pelo autor (2021).

Na página web contendo o painel do Spark, é possível visualizar o histórico das aplicações executadas como também suas respectivas métricas, as aplicações em execução e as instâncias executoras do Spark, os *workers*. Na Figura 30 é ilustrado o histórico de aplicações executadas a partir das invocações feita pelo Airflow.

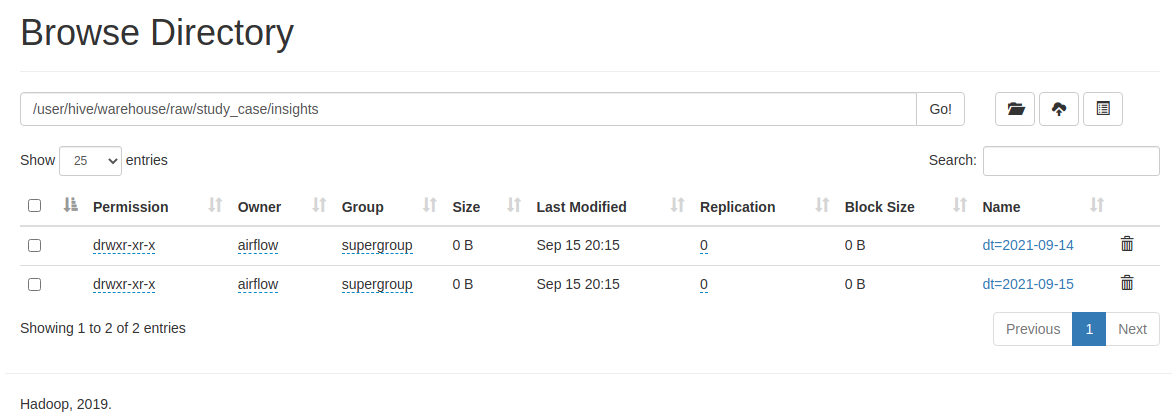
Figura 30 - Aplicação web do Spark e histórico de execuções.

[](https://github.com/jasondavindev/open-dataplatform/blob/main/doc/images/spark_ui_jobs.png)

Fonte: Elaborado pelo autor (2021).

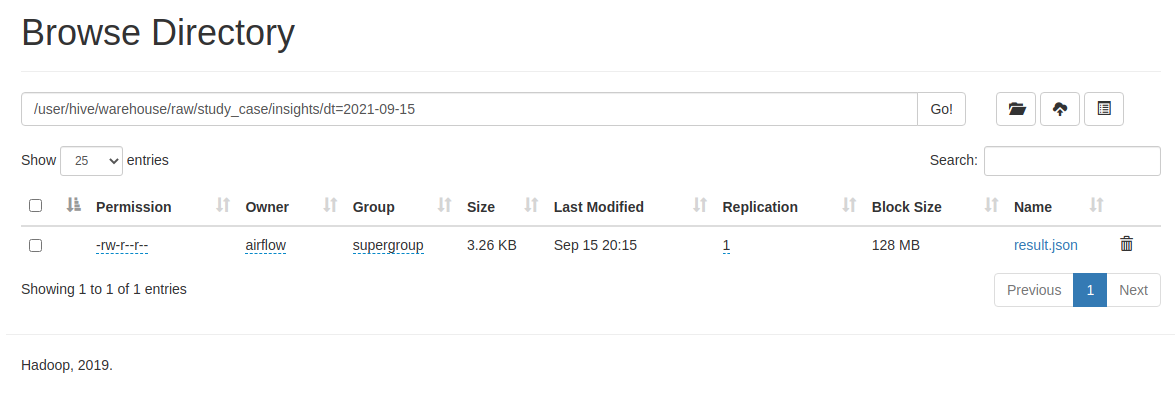
Como resultado da primeira tarefa, os dados extraídos da API são persistidos no Data Lake, na camada de dados brutos. Nas Figuras 31 e 32 são ilustradas as partições, ou seja, os dados da execução de cada dia em específico, como também o arquivo result.json contendo os dados extraídos por uma execução.

Figura 31 - Camada de dados brutos particionado pela data.

[](https://github.com/jasondavindev/open-dataplatform/blob/main/doc/images/hdfs_raw_layer.png)

Fonte: Elaborado pelo autor (2021).

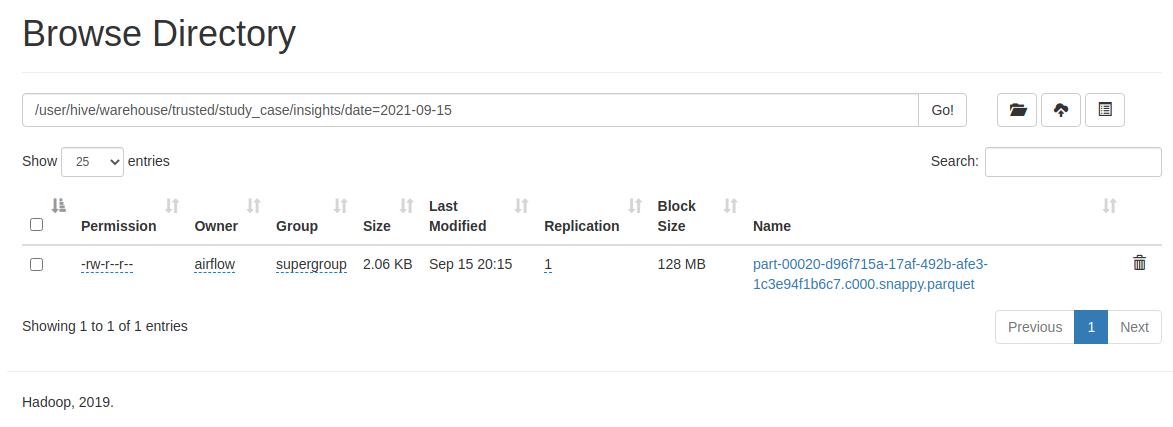
Figura 32 - Dado bruto persistido na camada de dados brutos.

[](https://github.com/jasondavindev/open-dataplatform/blob/main/doc/images/hdfs_partition_raw_layer.png)

Fonte: Elaborado pelo autor (2021).

Na Figura 33 são ilustrados os dados persistidos pela tarefa de transformação do dado bruto para o formato Parquet. É possível observar que os dados estão persistidos em outra camada, nomeada como trusted e também que o caminho do arquivo leva o nome da tabela Hive que foi criada "study\_case.insights".

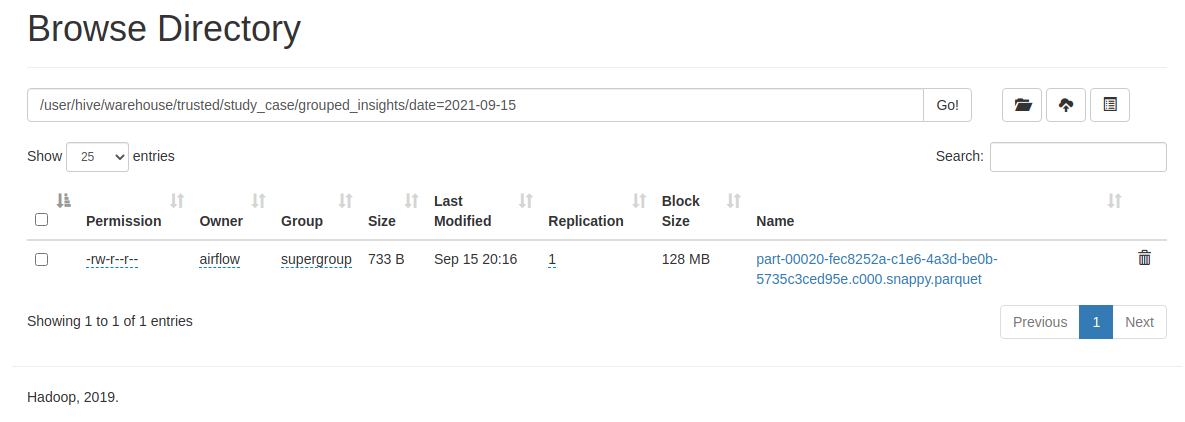
Figura 33 – Dados da tabela insights persistido no formato Apache Parquet.

[](https://github.com/jasondavindev/open-dataplatform/blob/main/doc/images/hdfs_partition_trusted_layer.png)

Fonte: Elaborado pelo autor (2021).

Como resultado da execução da terceira tarefa, a de agregação dos dados, um novo arquivo Parquet é persistido no *data lake*. Na Figura 34 são ilustrados os dados persistidos na mesma camada trusted, porém, em outro caminho de arquivo que leva o nome da tabela Hive que foi criada, study\_case.grouped\_insights.

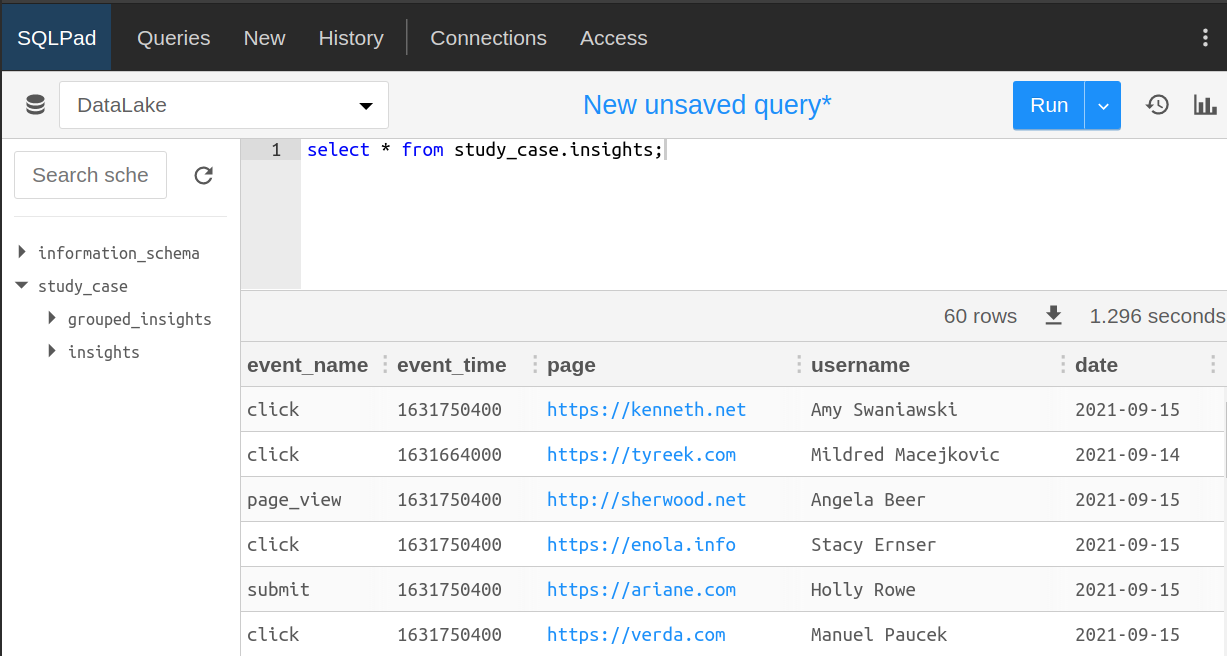
Figura 34 - Dados da tabela grouped\_insights persistido no formato Apache Parquet.

[](https://github.com/jasondavindev/open-dataplatform/blob/main/doc/images/hdfs_trusted_grouped_insights.png)

Fonte: Elaborado pelo autor (2021).

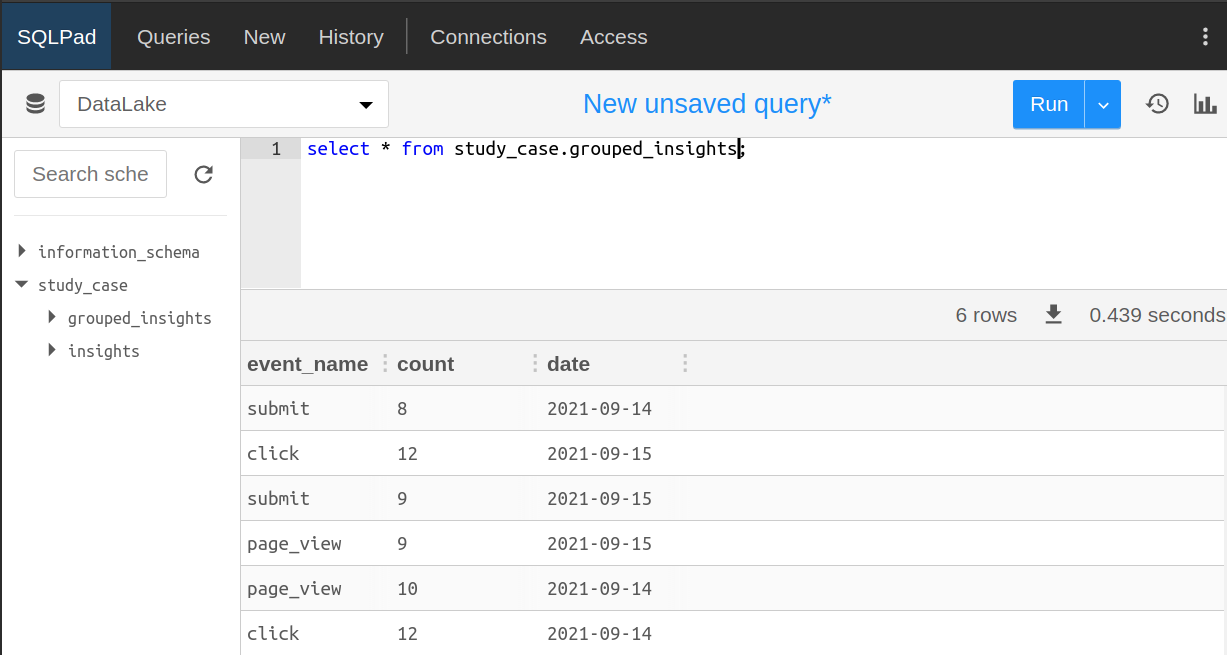
Na Figura 35 e 36 são ilustradas as instruções SQLs de consulta aos dados persistidos nas tabelas criadas pela segunda e terceira tarefa, respectivamente. As tabelas levam o nome de insights e grouped\_insights, onde a tabela insights contém apenas os dados convertidos do formato JSON para Parquet. A tabela grouped\_insights contém os dados agrupados pelos campos anteriormente citados na seção 3.7.1. As consultas foram feitas através da ferramenta SQL Pad.

Figura 35 - Consulta SQL na tabela insights.

[](https://github.com/jasondavindev/open-dataplatform/blob/main/doc/images/sql_pad_insights_table.png)

Fonte: Elaborado pelo autor (2021).

Figura 36 - Consulta SQL na tabela grouped\_insights.

[](https://github.com/jasondavindev/open-dataplatform/blob/main/doc/images/sql_pad_grouped_insights_table.png)

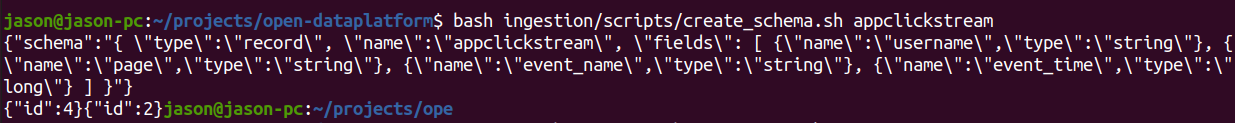
Fonte: Elaborado pelo autor (2021).

## 4.2. ETL em tempo real (Clickstream)

Nesta seção serão apresentados os resultados obtidos a partir da criação de um fluxo ETL em tempo real implementado na seção 3.7.2, nomeada como Clickstream.

Na Figura 37 é apresentada a execução do *script* para criação do esquema no formato Avro dos eventos que são postados na API apresentada anteriormente na seção 3.5. Na figura observa-se que o corpo completo do esquema é mostrado como saída da execução do *script* como também os identificadores no repositório de esquemas. O esquema criado foi nomeado como appclickstream.

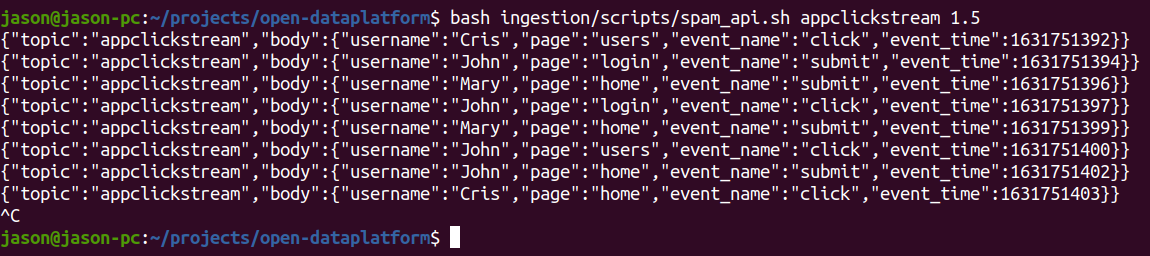
Figura 37 - Resultado script de criação do esquema dos eventos Avro.

[](https://github.com/jasondavindev/open-dataplatform/blob/main/doc/images/script_create_schema.png)

Fonte: Elaborado pelo autor (2021).

Na Figura 38 é ilustrado o resultado da execução do *script* que realiza chamadas HTTP na API anteriormente citada para a postagem dos eventos no respectivo tópico Kafka. O script foi invocado passando como parâmetro, o nome do esquema criado anteriormente e, como segundo parâmetro, um número indicando o tempo de espera entre cada chamada a API. O s*cript* realiza infinitas chamadas, mas para fins de testes a execução foi interrompida.

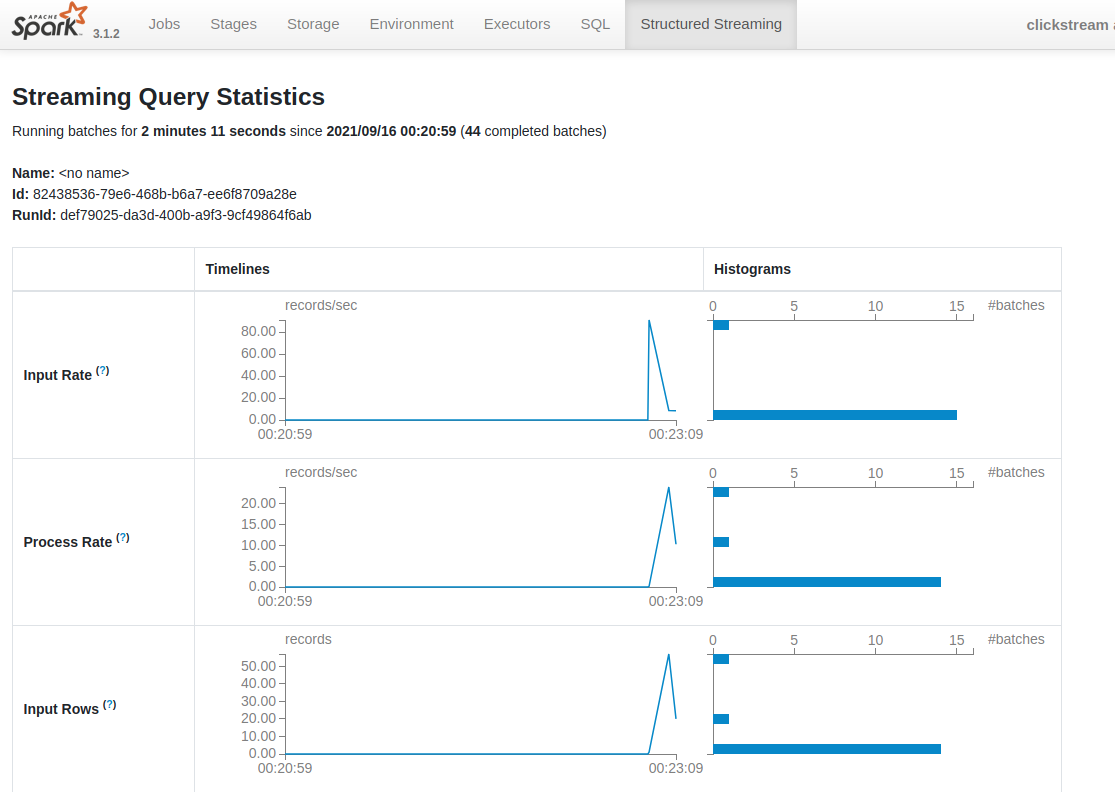
Figura 38 - Resultado script de postagem de eventos na API.

[](https://github.com/jasondavindev/open-dataplatform/blob/main/doc/images/script_spam_api.png)

Fonte: Elaborado pelo autor (2021).

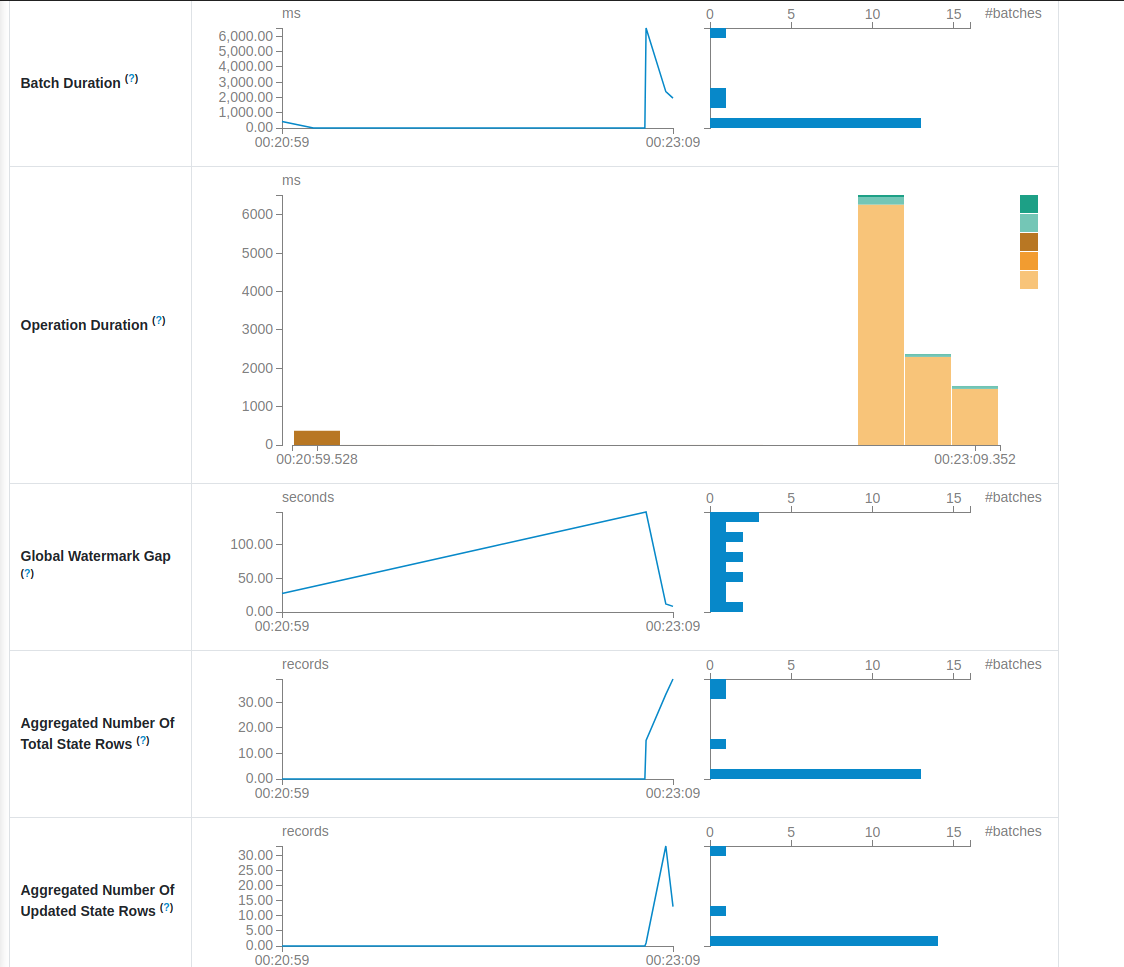
A partir das postagens realizadas na API HTTP consequentemente no tópico Kafka, a aplicação Spark Structured Streaming passa a realizar o processamento dos eventos que são entregues pelo tópico Kafka. A Figura 39 e 40 apresentam algumas métricas fornecidas pelo próprio Spark, como por exemplo, o número de eventos e linhas processadas, a duração de cada registro processado etc.

Figura 39 - Métricas de execução da aplicação Clickstream - Parte 1.

[](https://github.com/jasondavindev/open-dataplatform/blob/main/doc/images/app_clickstream_pt1.png)

Fonte: Elaborado pelo autor (2021).

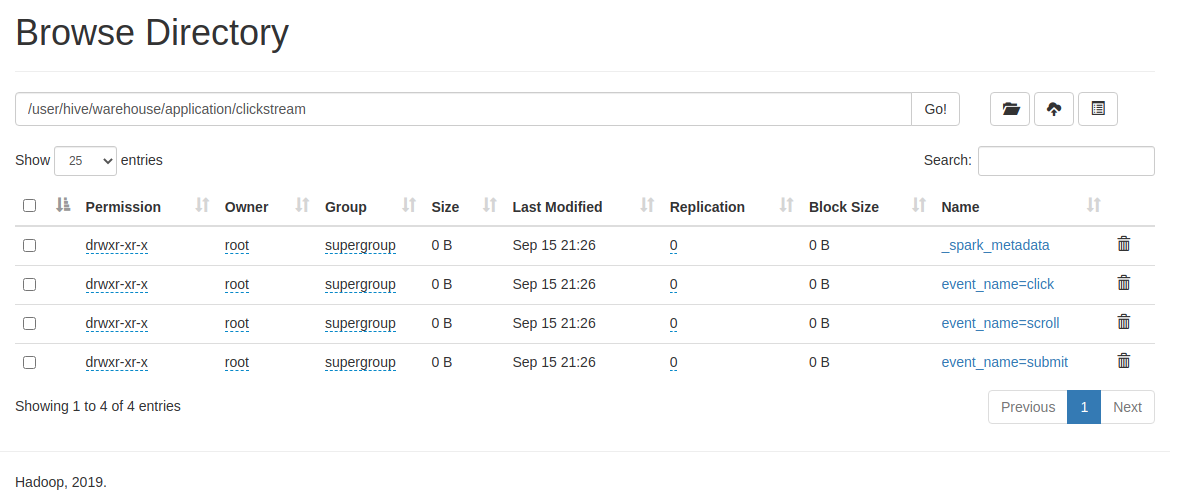
Figura 40 - Métricas de execução da aplicação Clickstream - Parte 2.

[](https://github.com/jasondavindev/open-dataplatform/blob/main/doc/images/app_clickstream_pt2.png)

Fonte: Elaborado pelo autor (2021).

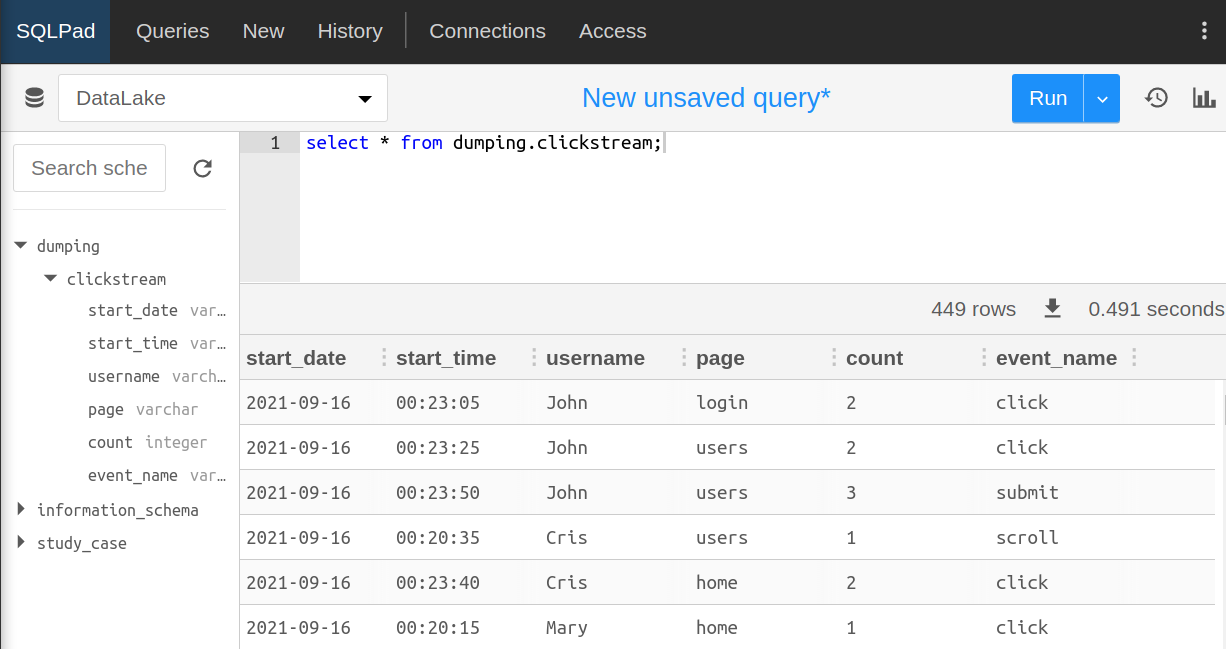
As Figuras 41 e 42 apresentam respectivamente os dados processados e persistidos pela aplicação Clickstream no Data Lake e a consulta efetuada para consultar os dados na tabela Hive criada, "dumping.clickstream". É possível observar que foi criada uma partição para cada tipo de evento, sendo eles *click*, *scroll* e *submit*.

Figura 41 - Dados processados pelo Clickstream persistidos no Data Lake.

[](https://github.com/jasondavindev/open-dataplatform/blob/main/doc/images/clickstream_hdfs.png)

Fonte: Elaborado pelo autor (2021).

Figura 42 - Tabela Hive contendo os dados persistidos pela aplicação Clickstream.

[](https://github.com/jasondavindev/open-dataplatform/blob/main/doc/images/clickstream_table.png)

Fonte: Elaborado pelo autor (2021).

# 5. CONSIDERAÇÕES FINAIS

Neste capítulo serão apresentadas as contribuições e conclusões deste trabalho, assim como os trabalhos futuros.

## 5.1. Contribuições e conclusões

A plataforma de extração, transformação e disponibilização de dados em lote e em tempo real apresentada neste trabalho cumpre com o objetivo de disponibilizar uma plataforma onde engenheiros de dados podem implementar scripts e aplicações para realizar extrações e transformações massivas em dados disponibilizados, por exemplo, por banco de dados, APIs HTTP, arquivos de texto etc.

Com o uso restrito e total de apenas ferramentas de código aberto neste trabalho, foi possível construir uma plataforma com zero custo financeiro, como por exemplo, a não necessidade da compra de licenças de softwares. Além de que, com a utilização das ferramentas aplicadas neste trabalho permitiu a possibilidade da implementação de códigos personalizados nas próprias ferramentas, como também a contribuição de tais códigos para o projeto oficial finalizando como contribuição para a comunidade de código aberto. Neste trabalho também foi possível implementar uma plataforma nativa para nuvem e pronta para ser portada para um ambiente produtivo com a ferramenta Kubernetes, aproveitando-se dos principais benefícios de portabilidade e gerenciamento de aplicações na nuvem que esta ferramenta provê.

A fundamentação técnica e desenvolvimento apresentado neste trabalho contribuiu para decisões técnicas aplicadas em um time de engenharia de dados, que se utilizou como base o conteúdo apresentado na seção 3.7.2 para a construção de uma aplicação Clickstream juntamente com o componente de persistência de dados e metadados apresentados nas seções 3.3 e 3.4. Como resultado, foi possível implementar uma ferramenta de captura de dados em tempo real indicando o comportamento de usuários suportando decisões de negócio por parte de times de análise de dados e marketing.

## 5.2. Trabalho Futuros

Este trabalho não encerra as pesquisas no desenvolvimento de uma plataforma para extração, transformação e disponibilização de dados em tempo real e lote, mas abre a oportunidade para os seguintes trabalhos futuros:

* Aplicar práticas e técnicas para atingir a separação de contextos de dados utilizando-se camadas com escopos definidos dentro do *data lake*.
* Utilizar ferramentas, como por exemplo Apache Ranger, para o gerenciamento de níveis de acessos aos dados persistidos no *data lake*.
* Com relação a aplicações em tempo real que a latência e o tempo de processamento são requisitos críticos, a ferramenta Spark Strucuted Streaming pode se tornar um gargalo. Outra ferramenta especialista em processamento em tempo real, como por exemplo Apache Flink, pode ser uma candidata a substituição.
* É interessante aplicar um novo formato de dados, como por exemplo Apache Hudi ou Delta Lake, para a possibilidade de operações *upsert* (atualização-inserção) e *update* (alteração), já que o formato Apache Parquet não atende à essas necessidades.

# REFERÊNCIAS

AIRFLOW, Apache. **Apache Airflow Documentation**. Disponível em: <https://airflow.apache.org/docs/apache-airflow/stable/index.html/> Acesso em: 06/04/2021.

CONFLUENT. **Schema Registry Overview**. Disponível em: <https://docs.confluent.io/platform/current/schema-registry/index.html> Acesso em: 08/05/2021.

DIGITAL, Tail. **O que é Customer Data Platform (CDP) e porque você precisa de uma**. Disponível em: <https://blog.tail.digital/o-que-e-customer-data-platform-e-porque-voce-precisa-de-um/> Acesso em: 30/03/2021.

DOCKER. **Docker overview**. Disponível em: <https://docs.docker.com/get-started/overview/> Acesso em: 04/10/2021.

HIVE, Apache. **Apache Hive - Apache Software Foundation**. Disponível em: <https://cwiki.apache.org/confluence/display/Hive/Home/> Acesso em: 06/04/2021.

IBM. **What is Avro?** Disponível em: <https://www.ibm.com/analytics/hadoop/avro> Acesso em: 13/04/2021.

IBM. **What is distributed computing**. Disponível em: <https://www.ibm.com/docs/ko/txseries/8.1.0?topic=overview-what-is-distributed-computing/> Acesso em: 08/05/2021.

IBM. **What is ETL (Extract, Transform, Load)?** Disponível em: <https://www.ibm.com/cloud/learn/etl>/Acesso em: 06/04/2021.

IBM. **What is HDFS?** Disponível em: <https://www.ibm.com/analytics/hadoop/hdfs>/Acesso em: 12/04/2021.

KAFKA, Apache. **Apache Kafka Documentation**. Disponível em: <https://kafka.apache.org/documentation/> Acesso em: 06/04/2021.

KAFKA, Apache. **Apache Kafka**. Disponível em: <https://kafka.apache.org/> Acesso em: 06/04/2021.

KUBERNETES. **Kubernetes Overview**. Disponível em: <https://kubernetes.io/> Acesso em: 11/10/2021.

PARQUET, Apache. **Apache Parquet**. Disponível em: <https://parquet.apache.org/>Acesso em: 13/04/2021.

POWIS, Owen. **What is Clickstream data?** Disponível em: <https://www.wordtracker.com/blog/keyword-research/what-is-clickstream-data/> Acesso em: 22/05/2021.

RAY, Amit. **Hadoop architecture in big data Yarn HDFS and MapReduce**. Disponível em: <https://amitray.com/hadoop-architecture-in-big-data-yarn-hdfs-and-mapreduce/> Acesso em: 15/08/2021.

REDHAT. **Understanding big data**. Disponível em: <https://www.redhat.com/en/topics/big-data>/Acesso em: 30/03/2021.

REDHAT. **What is a data lake?** Disponível em: <https://www.redhat.com/en/topics/data-storage/what-is-a-data-lake/> Acesso em: 30/03/2021.

SHEN, Catherine. **Why you should use Avro and schema registry for your streaming application**. Disponível em: <https://catherine-shen.medium.com/why-you-should-use-avro-and-schema-registry-for-your-streaming-application-2f24dcf017c8/> Acesso em: 13/04/2021.

SPARK, Apache. **Apache Spark - Unified Engine for large-scale data analytics**. Disponível em: <http://spark.apache.org/> Acesso em: 06/04/2021.

SPARK, Apache. **Structured Streaming Programming Guide**. Disponível em: <https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html/> Acesso em: 25/05/2021.

SPICIER, Thomas. **Apache Parquet: How to be a hero with the open-source columnar data format**. Disponível em: <https://blog.openbridge.com/how-to-be-a-hero-with-powerful-parquet-google-and-amazon-f2ae0f35ee04> Acesso em: 13/04/2021.

STATISTA. **Volume of data/information created, captured, copied, and consumed worldwide from 2010 to 2025**. Disponível em: <https://www.statista.com/statistics/871513/worldwide-data-created/> Acesso em: 05/11/2021.

TRINO. **Overview - Trino Documentation**. Disponível em: <https://trino.io/docs/current/overview.html/> Acesso em: 06/04/2021.

TRINO. **Use Cases - Trino Documentation**. Disponível em: <https://trino.io/docs/current/overview/use-cases.html/> Acesso em: 06/04/2021.