Universidade Estadual de Londrina

JENNIFER DO PRADO DA SILVA

PROPOSTA DE UMA ARQUITETURA DE GERÊNCIA DE DADOS PARA TAREFAS DE APRENDIZADO DE MÁQUINA E ANÁLISE DE DADOS NA AGRICULTURA

LONDRINA
2024

JENNIFER DO PRADO DA SILVA

PROPOSTA DE UMA ARQUITETURA DE GERÊNCIA DE DADOS PARA TAREFAS DE APRENDIZADO DE MÁQUINA E ANÁLISE DE DADOS NA AGRICULTURA

Trabalho de Conclusão de Curso apresentado ao curso de Bacharelado em Ciência da Computação da Universidade Estadual de Londrina para obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Daniel dos Santos Kaster

LONDRINA

2024

Ficha de identificação da obra elaborada pelo autor, através do Programa de Geração Automática do Sistema de Bibliotecas da UEL

Sobrenome, Nome.
Título do Trabalho : Subtitulo do Trabalho / Nome Sobrenome. - Londrina, 2017.
100 f. : il.

Orientador: Nome do Orientador Sobrenome do Orientador.
Coorientador: Nome Coorientador Sobrenome Coorientador.
Dissertação (Mestrado em Ciência da Computação) - Universidade Estadual de Londrina, Centro de Ciências Exatas, Programa de Pós-Graduação em Ciência da Computação, 2017.
Inclui bibliografia.

1. Assunto 1 - Tese. 2. Assunto 2 - Tese. 3. Assunto 3 - Tese. 4. Assunto 4 - Tese. I. Sobrenome do Orientador, Nome do Orientador. II. Sobrenome Coorientador, Nome Coorientador. III. Universidade Estadual de Londrina. Centro de Ciências Exatas. Programa de Pós-Graduação em Ciência da Computação. IV. Título.

JENNIFER DO PRADO DA SILVA

PROPOSTA DE UMA ARQUITETURA DE GERÊNCIA DE DADOS PARA TAREFAS DE APRENDIZADO DE MÁQUINA E ANÁLISE DE DADOS NA AGRICULTURA

Trabalho de Conclusão de Curso apresentado ao curso de Bacharelado em Ciência da Computação da Universidade Estadual de Londrina para obtenção do título de Bacharel em Ciência da Computação.

BANCA EXAMINADORA

Orientador: Prof. Dr. Daniel dos Santos Kaster
Universidade Estadual de Londrina

Prof. Dr. Segundo Membro da Banca
Universidade/Instituição do Segundo Membro da Banca – Sigla instituição

---

Prof. Dr. Terceiro Membro da Banca
Universidade/Instituição do Terceiro Membro da Banca – Sigla instituição

Londrina, 25 de Abril de 2024.

Este trabalho é dedicado aos meus pais e a minha irmã, que apesar de enfrentarem suas próprias lutas, nunca deixaram de me ajudar com as minhas.

AGRADECIMENTOS

Como agradecimento principal, gostaria de citar minha irmã gêmea Stefanie do Prado da Silva, que foi meu porto seguro em todos os momentos da minha vida e não se mostrou diferente durante a construção deste projeto, me auxiliou, cuidou de mim quando precisei, enfrentou problemas ao meu lado, me fortaleceu e me motivou a continuar persistindo mesmo com tantas dificuldades no caminho, me viu chorar, me viu sorrir, me viu ansiosa e irritada, mas nunca desistiu de permanecer ao meu lado.

Não menos importante, também gostaria de agradecer a minha mãe, Maria de Fátima Ferreira do Prado e ao meu pai, Henrique Tadeu da Silva, que fizeram tudo que foi possível para que o meu desejo de fazer Ciência da Computação em uma Universidade Pública se tornasse realidade. Agradeço imensamente por todo o esforço dedicado a minha causa e a confiança depositada em mim, por todas as viagens cansativas que tivemos que enfrentar, as noites em claro, as preocupações, as orientações, os investimentos, os cuidados com minhas alimentações e tudo que foi dado a mim, que me permitiu chegar até aqui.

Por fim, também quero agradecer ao meu professor, Daniel dos Santos Kaster, por me orientar neste Trabalho de Conclusão de Curso e se empenhar para sempre resolver minhas dúvidas e estar presente em todos os momentos que precisei e aos meus amigos, Isabela Hara Bando e Pedro Eduardo Garbossa de Almeida, por me acompanharem nesta jornada acadêmica, me auxiliando, divertindo, animando e sempre estando presentes até o fim.

"A ciência de hoje é a tecnologia de amanhã.
(Edward Teller)

SILVA, J. P.. **PROPOSTA DE UMA ARQUITETURA DE GERÊNCIA DE DADOS PARA TAREFAS DE APRENDIZADO DE MÁQUINA E ANÁLISE DE DADOS NA AGRICULTURA**. 2024. 70f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) – Universidade Estadual de Londrina, Londrina, 2024.

RESUMO

É possível observar que atualmente a Inteligência Artificial está ganhando força e rapidamente se tornando o pilar da inovação, com capacidade de identificar padrões, resolver problemas, realizar previsões para automatizar tarefas e proporcionar uma compreensão mais abrangente em relação a abundância de dados disponíveis. Na construção de sistemas de Aprendizado de Máquina, é preciso realizar coleta, transformação e organização dos dados antes de induzir os modelos de IA, sendo estes dados de diferentes tipos, fontes e escalas. Visando esta etapa, o objetivo deste projeto é apresentar uma arquitetura geral de ferramentas para coleta, limpeza e pré-processamento dos dados, filtrando dados qualificados e representativos para tarefas e problemas de Aprendizado de Máquina com aplicações voltadas a área da agricultura. Espera-se que os resultados apresentem uma arquitetura com ferramentas de código aberto integradas, assim como casos de uso, capazes de implementar *pipelines* de situações reais relacionadas a tarefas de suporte ao Aprendizado de Máquina na agricultura.

Palavras-chave: Integração de feramentas. Gerência de dados. Agricultura.

SILVA, J. P.. **PROPOSAL FOR A GENERAL DATA MANAGEMENT ARCHITECTURE FOR MACHINE LEARNING AND DATA ANALYSIS TASKS WITH APPLICATIONS IN AGRICULTURE**. 2024. 70p. Final Project (Bachelor of Science in Computer Science) – State University of Londrina, Londrina, 2024.

ABSTRACT

It is possible to observe that Artificial Intelligences (AIs) are currently gaining strength and quickly becoming the pillar of innovation, with the ability to identify patterns, solve problems, perform solutions to automate tasks and provide a more comprehensive understanding of the abundance of data available. In the construction of Machine Learning (ML) systems, it is necessary to collect, transform and organize the data before inducing the AI models, these data being of different types, sources and scales. Aiming at this stage, the objective of this project is to present a general architecture of tools for data collection, cleaning and pre-processing, filtering constructed and representative data for ML tasks and problems with applications outside the area of agriculture. It is expected that the results presented are an architecture with integrated open source tools, as well as use cases, capable of implementing pipelines of real situations related to tasks supporting Machine Learning in agriculture.

Keywords: Tool integration. Data management. Agriculture.

LISTA DE ILUSTRAÇÕES

Figura 1 – Funcionalidade do Apache Sqoop 18
Figura 2 – Arquitetura do Flume 19
Figura 3 – Arquitetura do Kafka 20
Figura 4 – Arquitetura do NiFi 21
Figura 5 – Exemplo de um Grafo Acíclico Direcionado 23
Figura 6 – Exemplo de um Grafo Cíclico Direcionado 24
Figura 7 – Gráfico percentual do tempo gasto por cientistas de dados 27
Figura 8 – Estrutura Board Support Package (BSP) em um SO Ångström 29
Figura 9 – Arquitetura lógica de alto nível BOP. 30
Figura 10 – Processo de enriquecimento no BOP. 31
Figura 11 – DAG criado dinamicamente com base nos dois arquivos de configuração. 33
Figura 12 – Tarefas reais a serem executadas organizadas em Grupos de Tarefas. . 33
Figura 13 – Arquitetura de gerência de dados para tarefas de aprendizado de máquina e análise de dados na agricultura 36
Figura 14 – Tarefa para iniciar o download em lote dos arquivos do MapBiomas . . 43
Figura 15 – Grupo de processadores para ingerir mapas de cobertura MapBiomas. . 43
Figura 16 – Fluxo dos arquivos do MapBiomas ingeridos em lote no Apache NiFi. . 44
Figura 17 – Fluxo de trabalho para extrair área de interesse do mapa anual de cobertura no Apache Airflow. 46
Figura 18 – Mapa anual de cobertura e uso da terra do Brasil produzido pelo MapBiomas. 47
Figura 19 – Camadas dos municípios no mapa do Brasil produzido pelo IBGE. . . . 48
Figura 20 – Camadas das áreas de interesse extraídas do mapa do Brasil. 48
Figura 21 – Área de interesse do mapa anual de cobertura e uso da terra do Brasil. 49
Figura 22 – Configurações do serviço de controlador DBCPConnectionPoll no Apache NiFi. 51
Figura 23 – Configurações do serviço de controlador DatabaseRecordSink no Apache NiFi. 52
Figura 24 – Grupo de processadores para streaming de dados meteorológicos. . . . 52
Figura 25 – Fluxo dos dados meteorológicos ingeridos em tempo real no Apache NiFi. 54
Figura 26 – Nome e tipo de dados das colunas na tabela dados_meteorologicos. . . 55
Figura 27 – Tarefa programada no Apache Airflow para inserir dados processados na tabela dados_meteorologicos. 56
Figura 28 – Fluxo de trabalho da DAG gera_mapa_diario. 58
Figura 29 – Mapa diário do fator meteorológico de temperatura média tempmedar2m. 59
Figura 30 – Mapa diário do fator meteorológico de umidade média umidrelmed2m. 60

LISTA DE TABELAS

Tabela 1 – Indicadores de Performance de Ferramentas de Ingestão de Dados . . . 22
Tabela 2 – Nome e tipo de dados das colunas na tabela dados_estacoes 57

SUMÁRIO

1 INTRODUÇÃO 13

2 FUNDAMENTAÇÃO TEÓRICO-METODOLÓGICA 15
2.1 Integração entre sistemas 15
2.2 Ingestão de Dados 16
2.2.1 Tipos de Ingestão de Dados 16
2.2.1.1 Ingestão de dados em lote 17
2.2.1.2 Ingestão de dados em tempo real 17
2.2.2 Ferramentas de Ingestão de Dados 17
2.2.2.1 Apache Sqoop 17
2.2.2.2 Apache Flume 18
2.2.2.3 Apache Kafka 19
2.2.2.4 Apache NIFI 20
2.3 Gestão de fluxo de trabalho 22
2.3.1 Tipos de fluxo de trabalho 23
2.3.1.1 Grafo Acíclico Direcionado 23
2.3.1.2 Grafo Cíclico Direcionado 23
2.3.2 Ferramentas de gerenciamento de fluxo de trabalho 24
2.3.2.1 Apache Airflow 24
2.3.2.2 Kedro 25
2.3.2.3 Luigi 25
2.4 Tarefas de preparação de dados para aprendizado de máquina ponta a ponta 26

3 TRABALHOS RELACIONADOS 28
3.1 ISOBlue HD 28
3.2 Billion Object Platform (BOP) 29
3.3 Polly 31
3.4 Estrutura de processamento de dados do Sentinel-2 32

4 ARQUITETURA 35
4.1 Integração 36
4.2 Ingestão de dados 37
4.3 Gestão de fluxo de trabalho 39
4.4 Ferramentas de Geoprocessamento 40

5 ESTUDO DE CASO 42

5.1 Área de interesse no MapBiomas 42
5.2 Mapa diário de dados meteorológicos 50

6 CONCLUSÃO 61

REFERÊNCIAS 63

1 INTRODUÇÃO

Em 2017, o site de publicidade inglês de notícias e assuntos internacionais, *The Economist*, afirmou que o recurso mais valioso do mundo não é mais o petróleo, mas os dados [1]. Diante desta afirmação, é comum observar que empresas de todo porte, instituições e organizações vem lidando com uma quantidade de dados significativamente grande, de todo tipo e de uma variedade de fontes, tornando cada vez mais evidente que a transformação de dados é frequentemente necessária [2].

É comum observar inúmeras aplicações do aprendizado de máquina em diversas áreas do conhecimento, como por exemplo na área da saúde, tecnológica, biológica e agrária. No caso desta última, inúmeras mudanças e inovações tecnológicas ocorreram nos anos atuais, com isso é possível observar que a inteligência artificial (IA) e o aprendizado de máquina são cada vez mais usados para previsão na agricultura [3] [4], uma das principais áreas em que os grandes volumes de dados e tipos de dados complexos são predominantes.

Em uma de suas matérias, no ano de 2019, a empresa ucraniana de TI, *Sciforce*, afirmou que o aprendizado de máquina está em toda parte durante todo o ciclo de cultivo e colheita [5]. Pode-se destacar os estudos realizados por pesquisadores que recorrem aos métodos de aprendizado de máquina e tecnologia de visão computacional para a identificação de doenças e pragas agrícolas [6]. Essas ideias e estudos enfocam que a agricultura pode sim se beneficiar do aprendizado de máquina para um melhor desempenho e resultados em suas atividades.

Em sua maioria, as tarefas de aprendizado de máquina direcionadas a área agrícola envolvem uma grande quantidade de dados. Com isso, a agricultura passa a ser cada vez mais dependente de soluções baseadas em dados e informações, para geração de diferenciais e elaboração de estratégias de impulsionamento do setor [7]. Isso acontece devido ao crescente volume e variedades de dados no campo. As técnicas convencionais de processamento de dados são incapazes de atender às demandas cada vez maiores na nova era da agricultura inteligente, o que é um importante obstáculo para extrair informações valiosas dos dados de campo [3].

Trabalhar com dados agrícolas requer levar em consideração que alcançar alta precisão e interpretabilidade é um desafio [8]. Esse grande volume de dados exigirá abordagens inteligentes a fim de evitar que se tornem apenas um enorme acúmulo [9]. Tal afirmação se direciona aos inúmeros tipos de dados existentes no meio agrícola que, se não preparados, podem prejudicar na previsão de uma dada variável dependente, na acurácia e até mesmo na otimização do processo, ao invés de oferecer *insights* que contribuam no acompanhamento desde a produção até o consumidor final.

Visando este panorama no aprendizado de máquina, muitas empresas se adaptaram e se especializaram para oferecer ferramentas úteis, no quesito de preparação dos dados, capazes de solucionar os problemas existentes no conjunto de dados, como por exemplo o *Azure Databricks* da empresa *Microsoft*, uma plataforma unificada que disponibiliza recursos para ingestão, preparação, análise e monitoramento de dados [10], dividindo o cenário com outras companhias especialistas, assim como o serviço *Oracle Cloud Infrastructure Data Integration*, da empresa *Oracle*, uma interface gerenciada, sem servidor e nativa da nuvem, responsável por extrair, carregar, transformar, limpar e remodelar dados [11], a interface *Amazon SageMaker Data Wrangler*, da empresa *Amazon*, a qual proporciona recursos para limpeza, exploração, visualização e processamento de dados tabulares e de imagem em grande escala [12] e outros sistemas disponíveis das diversas empresas no mercado da tecnologia. Estes serviços oferecem uma gama de recursos favoráveis para a preparação de dados, mas são plataformas limitadas, ja que cobram pelos recursos consumidos ou entregam pacotes de recursos por um valor preestabelecido. Além deste impasse relacionado a monetização, pode ser destacado também a limitação que o cliente tem em relação aos recursos próprios da empresa fornecedora do serviço, privando seus usuários e fazendo com que estes sejam incapazes de aplicar ao seu conjunto de dados, processos de preparação de dados provenientes de companhias concorrentes.

Neste contexto, este projeto propõe apresentar uma arquitetura geral para o processamento de grandes volumes de dados, com ferramentas integradas de código aberto que sejam compatíveis entre as demais, próprias para a coleta e limpeza de dados, bem como extrair informações relevantes destes, de preparar e gerenciar os diversos tipos de dados agro meteorológicos existentes e integrar as ferramentas selecionadas de maneira estratégica, de modo que seja possível desfrutar o máximo de suas funcionalidades e manter informações válidas, relevantes e precisas para aplicações em tarefas e soluções agrícolas baseadas em Aprendizado de Máquina.

2 FUNDAMENTAÇÃO TEÓRICO-METODOLÓGICA

Em um processo de desenvolvimento de integrações entre diferentes ferramentas e sistemas, focadas em trabalhar com *BigData*, é essencial se aprofundar em conceitos de integrações, ingestão de dados, orquestração de tarefas e outros, para estabelecer o embasamento teórico e as metodologias empregadas no desenvolvimento de uma arquitetura de dados robusta e eficiente. Este capítulo, visa descrever os conceitos fundamentais, as tecnologias envolvidas e as estratégias metodológicas adotadas, proporcionando uma compreensão clara do contexto em que o projeto está inserido e das soluções tecnológicas aplicadas para resolver problemas específicos na área de análise de dados.

2.1 Integração entre sistemas

A definição de integração de sistemas pode ser descrita como a capacidade de realizar conexões de dados, aplicativos, *APIs* e dispositivos [13], ou seja, diferentes ferramentas conversarem entre si e trocarem dados, reunindo ou incorporando partes em um todo, de maneira que a integração desses elementos contribuam para melhorar a eficiência, intensificar a produtividade, criar rotinas mais inteligentes com comunicação eficiente, e a garantir agilidade dos processos [14].

Para formar uma integração entre sistemas, alguns fatores devem ser considerados neste processo, como definir objetivos, mapear os processos internos, e definir quais integrações são necessárias. Para atuar na formação deste processo, é essencial contar com o suporte de plataformas tecnológicas [15].

A integração entre sistemas mantém a sincronização entre os sistemas sempre que ocorrer modificações de dados ou eventos, vinculando sistemas a nível funcional, permite criar uma integração orientada a eventos e mensagens de forma dinâmica e extremamente adaptáveis com transferências de dados em alta velocidade. As integrações apresentam ações controladas por eventos, que ocorrem quando um acionador, ou evento, dispara um procedimento ou uma solicitação. Também conta com integrações de *APIs*, mapeamento de dados entre os sistemas para definir como estes dados serão transferidos, facilitando a exportação, agrupamento e análise destas informações [16] [17].

Da mesma forma, existem diversos softwares e plataformas especializadas que realizam integrações entre sistemas, como um exemplo, é válido mencionar a plataforma *TensorFlow Extended(TFX)*, baseada em *TensorFlow* e implementada no *Google*, própria para implantação e gerenciamento de *pipelines* de produção de aprendizado de máquina [18]. Esta realiza a integração de componentes, como, por exemplo, para gerar modelos baseados em dados de treinamento, módulos para analisar e validar dados e modelos e in-

fraestrutura para servir modelos em produção, tudo em uma unica plataforma, padroniza os componentes, simplifica a configuração da plataforma e reduz o tempo de produção da ordem de meses para semanas, ao mesmo tempo que proporciona estabilidade da plataforma, minimizando interrupções [19]. Também é importante citar a plataforma de integração *IBM Cloud Pak® for Integration*, que apresenta recursos para aumentar a velocidade e a qualidade do sistema, com gerenciamento de *API*, integração de aplicativos, *streaming* de eventos e outras funcionalidades, além de possuir conectores inteligentes préconstruídos e recursos de automação, como integração de dados com tecnologia de IA, processamento de linguagem natural(PNL, pela sigla em inglês) e ferramentas de baixo código [17].

2.2 Ingestão de Dados

Nos últimos anos, diversos sistemas, dispositivos e maquinários apresentaram um crescimento exponencial de informações geradas, em diferentes contextos da sociedade, após passarem por inúmeros avanços tecnológicos. Isso possibilitou a produção em massa, em um tempo reduzido, de dados em uma escala sem precedentes, beneficiando análises minuciosas de dados e tomada de decisões baseadas em dados. Dentre essas informações geradas em grande escala, há diversos conjuntos de dados definidos como dados multiestruturados. Como exemplo desse tipo de dado, têm-se os textos em larga escala, imagens, vídeos e áudios [20]. Dessa forma, os dados precisam ser organizados para haver a aplicação plena e precisa destes, em diferentes destinos, além de serem facilmente acessíveis e compreensíveis para o usuário[21][22][23][24].

Para solucionar este problema, diversas empresas e organizações começaram a aplicar a ingestão de dados. Na ingestão de dados é realizado a coleta e o transporte de dados de diversas fontes para um meio de armazenamento centralizado, onde estes podem ser acessados pelos usuários, compartilhados para outros sistemas e plataformas e analisados ao fim do processo. Esta camada de ingestão de dados é etapa principal de qualquer arquitetura analítica e muitos sistemas de relatórios e análises dependem de dados consistentes e acessíveis [25][26].

2.2.1 Tipos de Ingestão de Dados

Atualmente, há dois métodos de ingestão de dados: Ingestão de Dados em Lote e Ingestão de Dados em tempo real. Os requisitos mínimos a serem atendidos e as restrições que devem ser respeitadas de cada negócio vão definir qual estrutura da etapa de ingestão de dados será aplicada em um projeto específico.

2.2.1.1 Ingestão de dados em lote

A forma mais implantada de ingestão de dados é o processamento em lote. Nesse contexto, a camada de ingestão coleta e reúne dados de diversas fontes em intervalos regulares e os envia posteriormente ao sistema de destino. Esses conjuntos de dados, podem ser processados com base em uma ordem lógica qualquer, ao ser realizado a ativação de condições específicas pré-determinadas ou em um agendamento simples estabelecido pelo usuário. Quando a coleta de dados em tempo real não é crucial para sistema em questão, a opção utilizada é o processamento em lote, visto que é mais simples e acessível de ser implementada do que a ingestão em tempo real [27][28].

2.2.1.2 Ingestão de dados em tempo real

A ingestão de dados em tempo real, também conhecida como processamento de fluxo ou *streaming*, difere da ingestão citada anteriormente por não requerer qualquer fase de agrupamento. Os conjuntos de dados alvos, são coletados da origem, manipulados e carregados imediatamente, após sua criação ou reconhecimento pela camada de ingestão de dados. Embora mais custoso, visto que é requerido o monitoramento constante das fontes de dados e a coleta imediata de novas informações, essa forma de ingestão é apropriada para análises que demandam dados continuamente atualizados, em um intervalo de tempo relativamente curto ou até mesmo em tempo real.

Muitas vezes, os dados são adquiridos sequencialmente, como um fluxo infinito e crescente. Esses dados de fluxo em tempo real precisam de processamento sequencial, em que a fonte de dados é particionada ao longo dos limites temporais em segmentos ou janelas finitas. Como exemplo, têm-se os dados do mercado de ações, sensores ou feeds do *Twitter*. Ao invés de esperar pela coleta total dos dados, em intervalos longos periódicos, a análise de streaming possibilita a identificação de padrões e a tomada de decisões com base nesses dados coletados, à medida que os dados começam a chegar. Quando os dados a serem ingeridos não são estacionários e apresentam padrões diferentes ao longo do tempo, as análises em tempo real se adaptam. Em casos de dados brutos em grande escala, onde o armazenamento destes torna-se questionável, a análise de streaming permite a persistência de apenas representações menores e mais direcionadas [29][30].

2.2.2 Ferramentas de Ingestão de Dados

2.2.2.1 Apache Sqoop

O *Apache Sqoop* é uma ferramenta projetada para facilitar a transferência eficiente de grandes volumes de dados entre o Hadoop e bancos de dados relacionais ou *mainframes*. Como ilustrado na figura 1, o *Sqoop* pode extrair dados de diversos sistemas de gerenciamento de banco de dados relacional (RDBMS), como *Oracle*, *MySQL* ou *mainframes*, e

transferi-los para o sistema de arquivos distribuídos *Hadoop (HDFS)*. Posteriormente, os dados podem ser processados usando *Hadoop MapReduce*, e os resultados exportados de volta para o RDBMS de origem [31].

O *Sqoop* utiliza conectores JDBC para estabelecer a conexão com os RDBMS. Para isso, requer que o *Java* esteja instalado e que o jar do *driver* JDBC esteja no caminho de classe do tempo de execução e depende do esquema do banco de dados relacional para interpretar os dados importados. Além dessas particularidades, ele também suporta *HSQLDB* (versão 1.8.0+), *MySQL* (5.0+), *Oracle* (10.2.0) e *PostgreSQL* (8.3+) e ainda pode ser usado com outros bancos de dados relacionais, como banco de dados *IBM DB2* e versões. Ao aproveitar o poder do *MapReduce* para processamento paralelo, o *Sqoop* gera vários arquivos no *HDFS* durante o processo de importação, os quais podem ser formatados como arquivos de texto delimitados, binários *Avro* ou arquivos de sequência, contendo os dados importados [32][33].

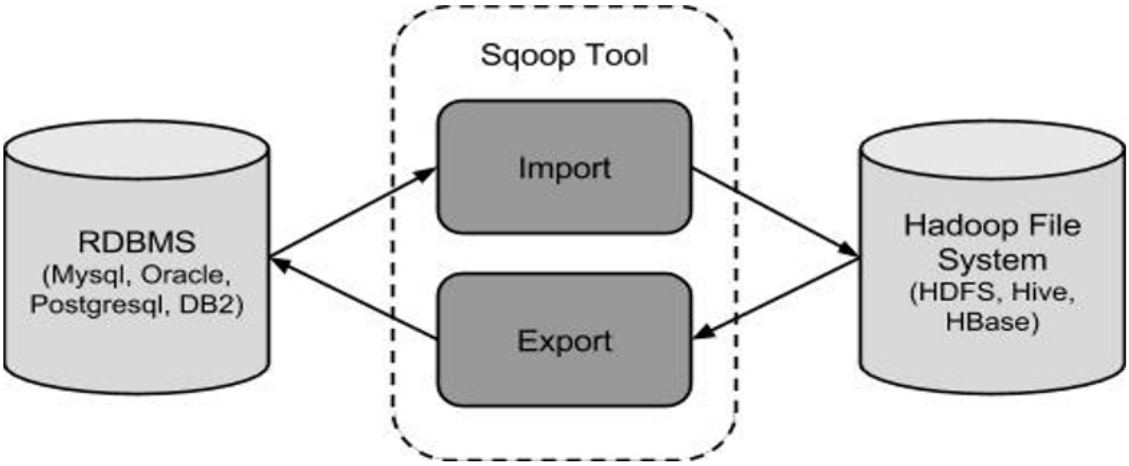

Figura 1 – Funcionalidade do Apache Sqoop. (Adaptado: 34)

2.2.2.2 Apache Flume

O *Apache Flume* é um serviço distribuído confiável, acessível e eficiente e foi projetado para importar, reunir e processar grandes volumes de dados, fornecendo uma plataforma para análise de dados em tempo real. Ele apresenta alguns desafios na tolerância a falhas com consistência precisa e é principalmente aplicado em modelo de dados para análise em tempo real. Desempenha responsabilidades essenciais no refinamento e visualização de dados. O *Flume* oferece uma estrutura para a coleta e análise de dados provenientes de uma rede de sensores, garantindo escalabilidade e alto desempenho no *Hadoop Distributed File System (HDFS)*[35].

O fluxo de dados no *Flume* é comparável a um *pipeline* responsável por capturar dados da origem e entregá-los ao destino. Na arquitetura *Flume*, ilustrada na figura 2, os dados passam por transformações da origem até o destino por meio do agente *Flume*. Esse agente, executado como um processo JVM, hospeda os componentes essenciais durante todo o percurso dos dados, incluindo *Source*, *Channel* e *Sink*. O *Source* recebe dados dos geradores associados e os encaminha para o *Channel*, que atua como uma ponte entre o *Sink* e o *Source*. O *Sink* é a entidade encarregada de enviar os dados ao destino [36].

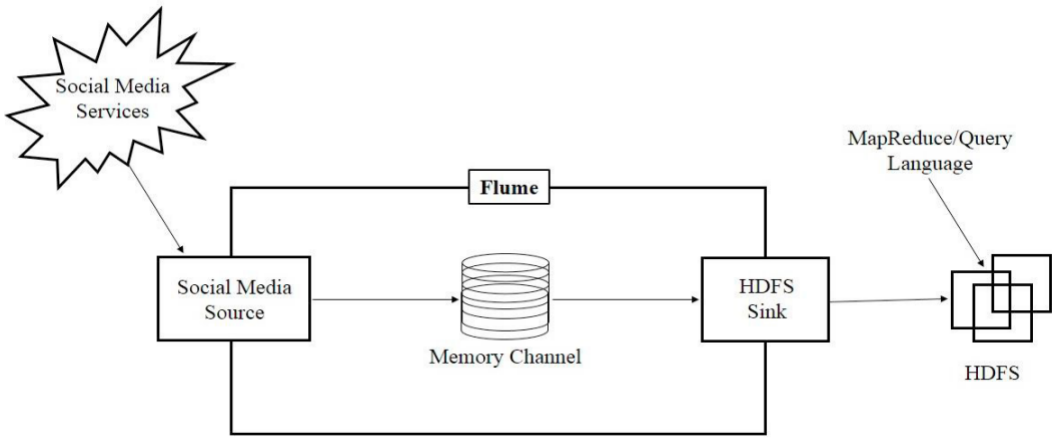

Figura 2 – Arquitetura do Flume. (Fonte: 37)

2.2.2.3 Apache Kafka

É um sistema *open source* que trabalha de forma distribuída para streaming de dados, ele é capaz de processar grandes volumes de dados em uma latência excepcionalmente baixa, visto que todo o processo ocorre na memória para evitar latência de acesso dos discos rígidos. Dentre suas funcionalidades, vale citar as seguintes: transferir uma informação de um sistema para o outro, armazenar essa informação para consultar posteriormente e transformar essa informação e transportá-la para outro sistema [38].

O *Kafka* possibilita trabalhos no formato de filas, assim como o *HabbitMQ* e *Amazon Simple Queue Service*. Ele possui mais recursos além de mensageria, que se trata do envio de dados de um lado para outro. O *Kafka* também apresenta esquema de escalabilidade, permitindo que os dados sejam armazenados por tempo indeterminado ou em um limite de tempo, pré-determinado pelo usuário, em que esses dados serão persistidos. Além dessas características, esta ferramenta dispõe de grande disponibilidade e tolerância a falhas, visto que é estruturado para manter cópias dessas informações [39][40].

Em sua arquitetura, ilustrada na figura 3, o *Kafka* apresenta três elementos principais: o *broker*, o consumidor e o produtor. Mesmo que o consumidor e o produtor sejam desenvolvidos em linguagens de programação distintas, o sistema opera de maneira efi-

ciente, conectando diversas plataformas. O *broker* desempenha a função de servidor no *Kafka* e é crucial para assegurar a tolerância a falhas, sendo este o recurso mais importante do *Kafka*. O produtor envia mensagens ao consumidor por meio do *broker*, que atua como um canal para a diferenciação das mensagens. O *Kafka* apresenta um canal de dados em tempo real de alto desempenho e para fins de gerenciamento e coordenação, o *broker* *Kafka* utiliza o *ZooKeeper* para notificar o produtor e o consumidor sobre a falha ou até mesmo inclusão de qualquer *broker* [41][42].

O *Apache Kafka* possui uma configuração que permite o usuário conectar sistemas externos nele, por meio de seus conectores. Logo após a realização de uma conexão, o *Kafka* inicia automaticamente a leitura nos conectores, se houverem. Esse recurso ainda permite a seleção de informações hospedadas no *Kafka* e o envio destas para outros sistemas de forma totalmente integrada, utilizando poucas linhas de código. O *Kafka* é aplicável para grandes e pequenas estruturas, neste ultimo caso, conforme o desenvolvimento da estrutura, o *Kafka* consegue auxiliar no crescimento de forma que seja possível escalar infinitamente [34][43].

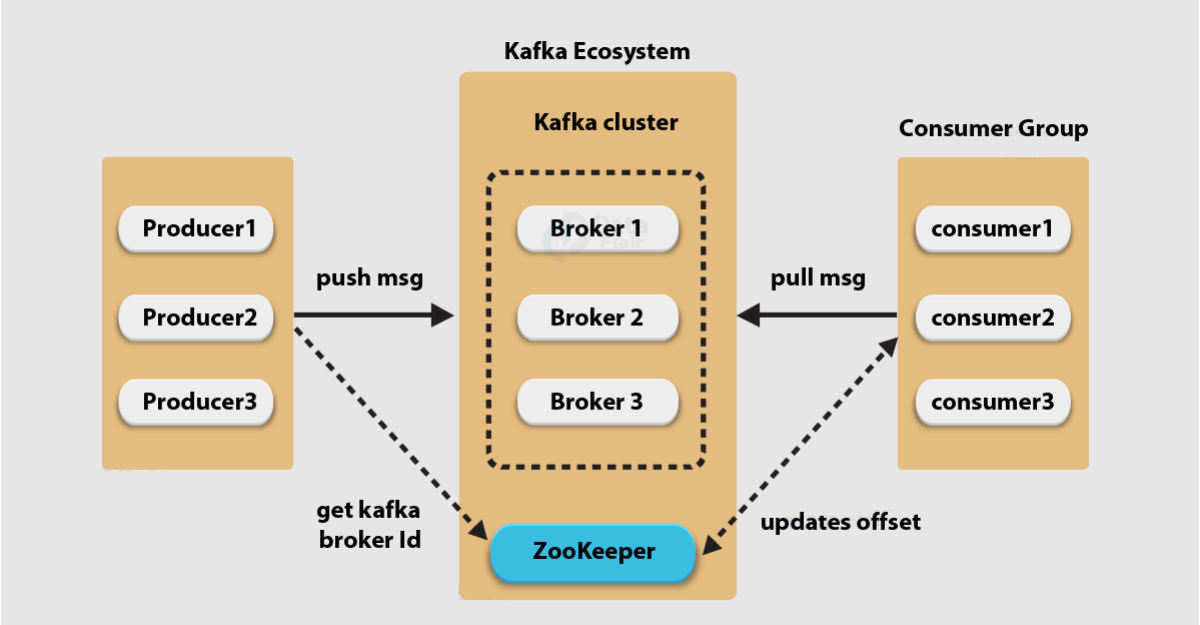

Figura 3 – Arquitetura do Kafka. (Fonte: 44)

2.2.2.4 Apache NIFI

O *NIFI* é um sistema de fluxo de dados, em tempo real e em lote, projetado para coletar, transformar, processar e rotear dados. Sua arquitetura segue o conceito de programação baseada em fluxo, visando automatizar e gerenciar o fluxo de dados entre sistemas [45].

Desenvolvido em *Java*, o *NIFI* opera em uma *Máquina Virtual Java (JVM)* em um sistema operacional *host*, também é capaz de ser executado em contêineres *Docker* e em um *cluster*, cada nó no *cluster NIFI* completa as mesmas tarefas, mas interagem com conjuntos diferentes de dados. O *cluster* é gerenciado pelo coordenador do *cluster*, eleito pelo *Zookeeper* [46].

A arquitetura do *NIFI*, apresentada na figura 4, é composta por diversos componentes, incluindo o Servidor *Web*, responsável por hospedar comandos baseados em *HTTP* e permitir que os usuários acessem o *NIFI* por meio de uma interface *web*. Outros componentes essenciais são o Controlador de Fluxo, encarregado de fornecer e agendar *threads* para execução, o Repositório *FlowFile*, onde o *NIFI* registra as atualizações de *status* dos arquivos de fluxo, o Repositório de Conteúdo, que armazena o conteúdo dos *flowfiles*, e o Repositório de Proveniência, que contém dados relacionados aos eventos de proveniência[47].

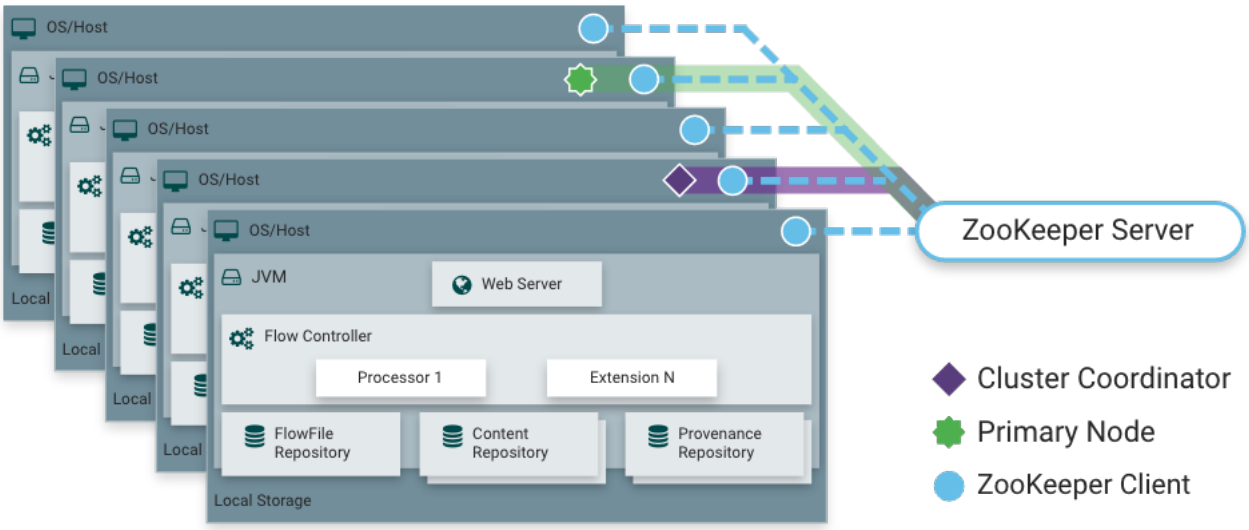

Figura 4 – Arquitetura do NiFi. (Fonte: 47)

Na tabela 1, comparações foram realizadas entre as ferramentas de ingestão de dados mencionadas, baseadas em diversos indicadores, incluindo tipo de carregamento, arquitetura e o tipo de dados que podem ser ingeridos. Todas as ferramentas descritas são de código aberto e escritas em *Java*. O *Sqoop* destaca-se como uma escolha sólida para a ingestão de dados em lote de sistemas de gerenciamento de banco de dados relacionais (RDBMS), suportando compactação de dados, assim como as demais ferramentas e oferecendo um modo bidirecional para interação com o HDFS, permitindo a importação e exportação de dados. O *NIFI* é uma opção recomendada para estabelecer fluxos de dados entre sistemas diversos. Ele é capaz de realizar a ingestão tanto em lotes quanto em fluxo, proporcionando a priorização de eventos e o carregamento dos dados de forma orientada

e não orientada a eventos. Quanto ao *Kafka* e o *Flume*, ambos lidam com fluxos de dados e armazenam mensagens por mais tempo em relação as demais ferramentas, mas o *Kafka* é especialmente usado para mensagens. Destaca-se por sua alta escalabilidade, permitindo a fácil adição de diferentes números de consumidores para atender às necessidades específicas do usuário.

Tabela 1 – Indicadores de Performance de Ferramentas de Ingestão de dados. (Adaptado de: [34][39])

Indicador	**Flume**	**NiFi**	**Kafka**	**Sqoop**
Primeira linguagem	Java	Java	Java	Java
Licença	Código aberto	Código aberto	Código aberto	Código aberto
Natureza básica	Para streaming de dados fontes gerados continuamente	Para a criação de fluxo de dados entre diferentes sistemas.	Para mensagens de Streaming de dados	Para RDBMS que possui JDBC(Java EE Database Connectivity) como o oracle
Tipo de dados	Dados em tempo real	Dados em tempo real e em lote	Dados em tempo real	Dados em lote
Escalabilidade	Médio	Médio	Alto	Médio
Tipo de carregamento	Orientado a eventos	Ambos (evento e não-evento)	Orientado a eventos	Não orientado a eventos
Arquitetura	Baseado em agente	Baseado em fluxo	Baseado na topologia de processo	Baseado em conector
Priorização de evento	Suportado	Suportado pelo conceito (processador de coletor de failover)	Programável	Não possui
Duração de mensagens	Alto	Baixo	Alto	Baixo

2.3 Gestão de fluxo de trabalho

Os sistemas de gerenciamento de fluxo de trabalho têm despertado considerável interesse nos últimos anos, devido à sua capacidade de integrar aplicações distribuídas e heterogêneas em diferentes ambientes de processamento. Mesmo apresentando algumas limitações, as ferramentas existentes de gerenciamento de fluxo de trabalho possuem um processamento rápido e estruturado de tarefas, eficiência de implantação para diversas aplicações e vários recursos disponíveis para o usuário orquestrar seus trabalhos de forma organizada [48] [49].

Um gerenciador de fluxo de trabalho se trata de um sistema responsável por administrar processos e tarefas repetitivas, que precisam ocorrer em uma ordem específica. Ele pode coordenar desde uma série simples de tarefas individuais até um processo de negócios, considerado mais complexo, por apresentar vários fluxos de trabalho, sistemas de informação, dados e padrões de atividade. Um fluxo de trabalho é geralmente visualizado

em um diagrama e possui diversas formas, isso devido à dificuldade de planejamento e repetitividade que os processos podem apresentar [50].

A maioria dos sistemas de gerenciamento de fluxo de trabalho disponibilizam recursos para criar linhas de processamento estruturados como um Grafo Acíclico Direcionado, conhecido como *DAG* do inglês *Directed Acyclic Graphs*, ou como um Grafo Cíclico Direcionado, conhecido como *DCG* do inglês *Directed Cyclic Graphs*.

2.3.1 Tipos de fluxo de trabalho

2.3.1.1 Grafo Acíclico Direcionado

Nos fluxos de trabalho baseados em *DAGs*, ilustrado na figura 5, os nós do grafo representam as tarefas a serem executadas e as arestas indicam as dependências entre essas tarefas. Existem dois tipos principais de dependências a serem consideradas: dependências de dados, em que a saída de uma tarefa serve como entrada para as tarefas subsequentes, e dependências de controle, em que certas tarefas precisam ser concluídas antes do início de uma tarefa individual ou de um conjunto de tarefas. O *DAG* pode ser empregado para representar um conjunto de programas, nos quais a entrada, saída ou execução de um, ou mais programas está condicionada à conclusão bem-sucedida de um ou mais programas específicos [49][51].

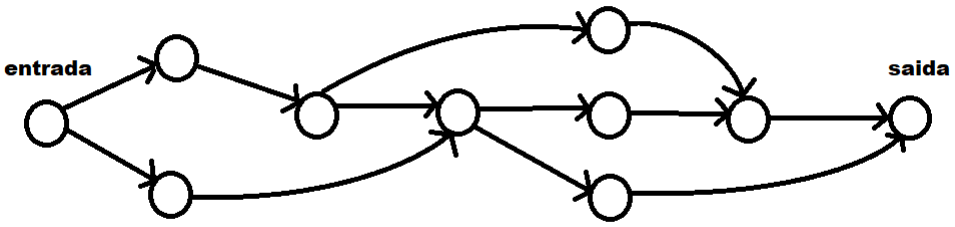

Figura 5 – Exemplo de um Grafo Acíclico Direcionado.

2.3.1.2 Grafo Cíclico Direcionado

Os fluxos de trabalho baseados em *DCG*, como o exemplo ilustrado na figura 6, são caracterizados por ciclos que denotam um *loop*, seja implícito ou explícito, ou mecanismos de controle de iteração. Nesse contexto, o grafo de fluxo de trabalho geralmente descreve uma rede de tarefas, onde os nós representam processos, instâncias de componentes de software ou objetos de controle mais abstratos. Neste tipo de fluxo, os processos podem ser executados uma ou mais vezes, dependendo das condições atendidas [49].

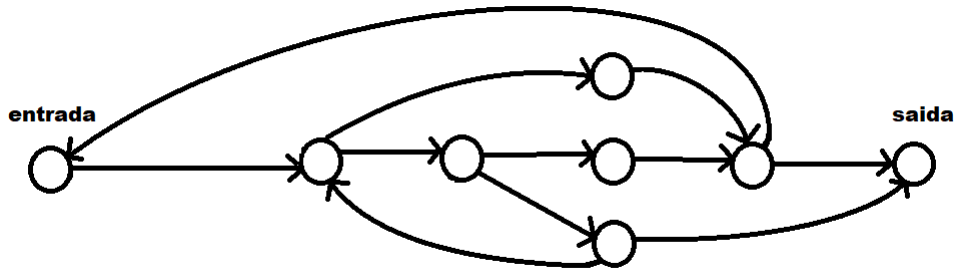

Figura 6 – Exemplo de um Grafo Cíclico Direcionado.

2.3.2 Ferramentas de gerenciamento de fluxo de trabalho

2.3.2.1 Apache Airflow

Dentre várias ferramentas de gerenciamento de fluxo de trabalho, foi selecionado o *Apache Airflow*, uma ferramenta *Open Source* destinada à criação, monitoramento e agendamento de fluxo de trabalho de forma programática. O *Apache Airflow* é um recurso altamente integrável e foi desenvolvido seguindo alguns princípios importantes para gerenciamento de *workflows* e atribuindo características úteis.

O primeiro princípio é ser uma ferramenta escalável, para isso, conta com uma arquitetura modular, em que apresenta uma fila de mensagens para administrar uma quantidade arbitrária de trabalhadores. O segundo é o dinamismo. O *Apache Airflow* apresenta *pipelines* definidos em *Python*, isso possibilita a geração dinâmica de pipelines, por meio de códigos que os instanciam dinamicamente. Por fim, o terceiro princípio se trata de ser uma ferramenta extensível, permitindo estabelecer facilmente os operadores desejados e estender as bibliotecas a fim de definir e controlar o nível de abstração conforme o ambiente esperado [52].

Em relação às características, o *Apache Airflow* apresenta diversos aspectos importantes, focados em torná-lo um recurso simples e completo. A criação e gerenciamento de *workflows*, é realizada utilizando inteiramente a linguagem *Python*, possibilitando a aplicação de diversas funcionalidades, como, por exemplo, agendamento de pipelines utilizando diferentes formatos de data e hora e aplicação de loops aos fluxos de trabalho visando gerar tarefas dinamicamente. Com isso, o *Python* ajuda a garantir total flexibilidade durante a criação e gerenciamento dos *pipelines* [53][54].

O *Airflow* também conta com uma robusta aplicação *web* que auxilia e facilita no monitoramento, agendamento e gerenciamento das tarefas criadas, proporcionando uma visão completa da situação de cada *pipeline* em execução e dos registros de tarefas concluídas. Além destas características, é importante destacar que o *Apache Airflow* não limita o escopo dos *pipelines*, pode ser utilizado para diversos fins e disponibiliza diversos operadores *plug-and-play* para executar tarefas em vários serviços, como, por exemplo, o

Google Cloud Platform e o *Amazon Web Services*. Essas características permitem que o *Airflow* seja extensível e fácil de se adaptar às novas infraestruturas [55][56].

2.3.2.2 Kedro

Kedro é uma estrutura *Python* de código aberto hospedada pela *Linux Foundation*, foi projetada para facilitar a construção de pipelines de aprendizado de máquina e dados complexos. Ele adota melhores práticas para desenvolver código limpo em ciência de dados, reduzindo o tempo gasto em tarefas de "encanamento". Com recursos como a criação de versões de dados, cálculos incrementais e automação da ordem de execução do *pipeline*, o *Kedro* simplifica o desenvolvimento. Ao integrar o *Kedro-Viz*, ele oferece um plano de desenvolvimento de dados, linhagem de dados, acompanhamento de experimentos de aprendizado de máquina e facilita a comunicação com partes interessadas do negócio [57].

Alguns orquestradores de fluxo de trabalho como o *Airflow*, *Luigi* e *Prefect* se dedicam à execução, agendamento e monitoramento de *pipelines*. Em contrapartida, o *Kedro* tem como foco o processo de criação de *pipelines*. Portanto, se os objetivos do gerenciamento de fluxo de trabalho incluem solucionar problemas de como a tarefa será executada ou como a computação será gerenciada, o *Kedro* pode não ser a solução mais adequada [58][59].

O *kedro* possui um catálogo de dados com uma série de conectores de dados leves usados para salvar e carregar dados em diversos formatos e sistemas de arquivos [60].

2.3.2.3 Luigi

Luigi é um pacote *Python* projetado para facilitar a construção de *pipelines* complexos de trabalhos em lote. Ele lida com resolução de dependências, gerenciamento de fluxo de trabalho, visualização, tratamento de falhas, integração de linha de comando e diversos outros recursos. Os usuários têm a capacidade de especificar as dependências entre tarefas utilizando grafos acíclicos direcionados (DAGs), assegurando a execução e repetição adequadas das tarefas na ordem correta [61].

Ele proporciona a flexibilidade de criar virtualmente qualquer tarefa e ainda oferece uma caixa de ferramentas contendo modelos de tarefas comuns para simplificar o processo. Essa ferramenta abrange suporte para a execução de *jobs mapreduce* em *Python* no *Hadoop*, assim como *jobs Hive* e *Pig*. Além disso, *Luigi* fornece abstrações de sistema de arquivos para *HDFS* e arquivos locais, garantindo operações atômicas no sistema de arquivos. Essa característica é crucial, pois assegura que o *pipeline* de dados não fique preso em um estado que contenha dados parciais [62].

O servidor *Luigi* também apresenta uma interface *web*, permitindo que o usuário

pesquise e filtre todas as tarefas de maneira conveniente. Tudo no *Luigi* é implementado em *Python*. Ao contrário de configurações em XML ou arquivos de dados externos similares, o gráfico de dependência é especificado diretamente em *Python*. Essa abordagem simplifica a construção de gráficos de dependência complexos de tarefas, nos quais as dependências podem envolver álgebra de datas ou referências recursivas a outras versões da mesma tarefa [63][64].

2.4 Tarefas de preparação de dados para aprendizado de máquina ponta a ponta

O aprendizado de máquina é uma ramificação da inteligência artificial e da ciência da computação, e tem como foco, o uso de dados, algoritmos e modelos estatísticos para simular o modo como os humanos aprendem. Para isso, essa ramificação aprimora gradualmente sua precisão de forma autônoma, utilizando redes neurais e aprendizado profundo sem a necessidade de ser programado explicitamente, confiando em padrões e inferências. Com isso, quanto maior for a alimentação de dados no sistema, mais precisos serão os resultados [65] [66].

O conceito de aprendizado de máquina é muito importante para se ter ideia de como funciona este subconjunto da inteligência artificial, mas para compreender todo o processo e funcionamento de um projeto que abrange a concepção deste subconjunto é essencial estar inteirado sobre as etapas do aprendizado de máquina ponta a ponta. Este é um processo, em que uma máquina aprende um mapeamento entre uma série de entradas (X) para algumas saídas conhecidas (y), sem ser explicitamente programada [67]. Pode ser definido em três etapas. A primeira, foca em possuir ampla compreensão dos dados, meio de coleta e métodos para limpeza de dados. Em seguida, a segunda etapa, destina-se em selecionar e implementar o modelo. Por fim, a terceira etapa, é a definição dos parâmetros do modelo e ajuste de dados [68][69].

No aprendizado de máquina ponta a ponta, como ilustrado na figura 7, os cientistas de dados gastam 19% de seu tempo na coleta de conjuntos de dados [70], uma das tarefas mais longas na construção de um modelo. Isso acontece, pois na execução deste processo há dois fatores fundamentais a serem considerados, a qualidade e quantidade. Contando com um conjunto de dados grande o suficiente é possível ter mais análises significativas e dispor de dados de alta qualidade garante que as informações sejam relevantes o bastante para o caso de uso em questão e proporcionam modelos resultantes com altas taxas de precisão [71].

Já a limpeza e organização dos dados é responsável por cerca de 60% do trabalho dos cientistas de dados [70] e com isso a etapa mais demorada e importante na construção de um modelo. Neste processo, são realizadas diversas verificações, como tratamento de

valores ausentes com dados inferidos a partir dos dados existentes, modificação de variáveis categóricas, identificação de valores discrepantes, expandir *features* existentes, limpeza e várias outras transformações dos dados de treinamento para que estejam prontos para análise de dados e otimização de modelo [71] [72].

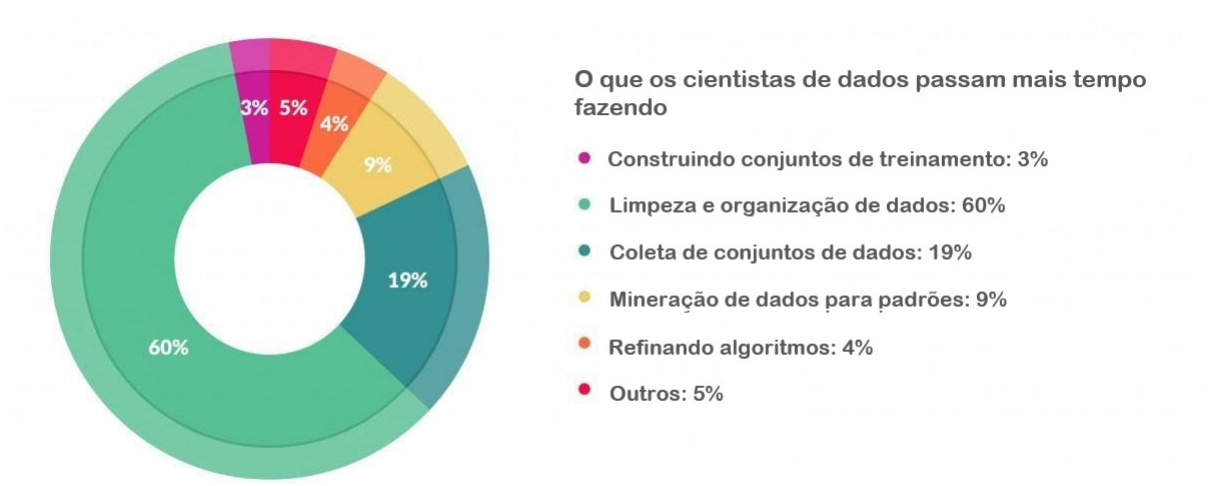

Figura 7 – Gráfico percentual do tempo gasto por cientistas de dados. (Adaptado: 70)

Após o longo processo de filtragem e limpeza dos dados, é efetuada a etapa de engenharia de *features* sobre os dados, caso os usuários desejem realizar a criação de modelos de predição e/ou classificação baseada nestes dados previamente tratados. Neste contexto, a noção de *features* pode ser descrita como a capacidade de denotar a estrutura funcional e as propriedades visíveis de um sistema, ou seja, são entradas que os modelos de aprendizado de máquina utilizam durante o treinamento e a inferência para fazer previsões [73][74][75].

É possível utilizar a engenharia de *features* em diversas áreas de aplicação, como exemplo, têm-se o projeto em que a engenharia de features foi aplicada em conjunto com a avaliação de *features*, para realizar extração de informações em Notas Fiscais. Neste último, a engenharia de *features* foi utilizada para determinar conjuntos de *features* que possuíam maior relevância para identificação de elementos de notas fiscais eletrônicas [76].

Atualmente, existem inúmeros processos disponíveis para serem aplicados em tarefas de preparação de dados na inteligência artificial e aprendizado de máquina, mas este projeto visa focar em processos de ingestão de dados em lote e em tempo real, orquestração de fluxo de trabalho, pré-processamentos e geoprocessamentos.

3 TRABALHOS RELACIONADOS

A construção de uma arquitetura de preparação de dados, de forma geral, envolve estabelecer um fluxo de dados, e, com isso, é preciso definir quais técnicas e ferramentas de gerenciamento serão aplicados ao conjunto de dados. Este processo é de suma importância para ser possível organizar e administrar as demais etapas existentes na arquitetura, desde a coleta e ingestão de dados até a transformação, a distribuição e o consumo[77]. Desta forma, este capitulo focaliza demonstrar trabalhos relacionados que construíram arquiteturas para ingerir e tratar dados de diversos tipos, aplicando as etapas citadas acima, visando solucionar seus problemas específicos em diferentes linhas de aplicação, como o ISOBlue HD, Billion Object Platform (BOP), Polly e a Estrutura de processamento de dados do Sentinel-2, detalhados nas sessões a seguir.

3.1 ISOBlue HD

O trabalho [78] apresenta o *ISOBlue HD*, uma plataforma de código aberto para aquisição de dados em máquinas agrícolas. Essa plataforma, composta por um computador de placa única, modem celular, armazenamento local e *switch Power-over-Ethernet* para sensores, possibilita a obtenção de dados ricos em contexto, permitindo a identificação do *status* da máquina e da logística agrícola. O sistema oferece recursos como diagnóstico e acesso remoto, inicialização e desligamento automático por meio das operações do veículo e utiliza o *Apache Kafka* para possibilitar a troca robusta de dados. O *ISOBlue HD* foi testado em uma colheitadeira durante uma colheita de trigo em 2019, capturando dados significativos com foco na plataforma e nas ações do operador. As análises destacam a capacidade de inferir conhecimento contextual sobre a semântica dos dados dos sensores e a logística da colheita, como, por exemplo, caminhos, velocidades, *status* da plataforma, transferência de material, entre outros fatores.

O software desenvolvido, utilizou as customizações de um *Board Support Package (BSP)* de código aberto. A figura 8 apresenta a estrutura definida do *BSP*, o qual contém o *Ångström*, um sistema operacional leve, aplicativos que encapsulam programas, como o de gerenciamento de sistema, programas de gerenciamento de energia, um *cluster Kafka* e registradores de dados, além de *drivers* de dispositivo e *middlewares* adicionais.

Neste trabalho, o *Apache Kafka* atuou com um *cluster* responsável por gerenciar quatro tópicos: *imp* (dados de barramento Implement), *tra* (dados de barramento de trator), *gps* (dados do GPS) e *remote* (GPS e dados de diagnóstico). Os registradores de dados conectam-se ao *cluster* como produtores Kafka para publicar dados de sensores para esses tópicos. O software *MirrorMaker* do kafka sincroniza os dados que fornecem

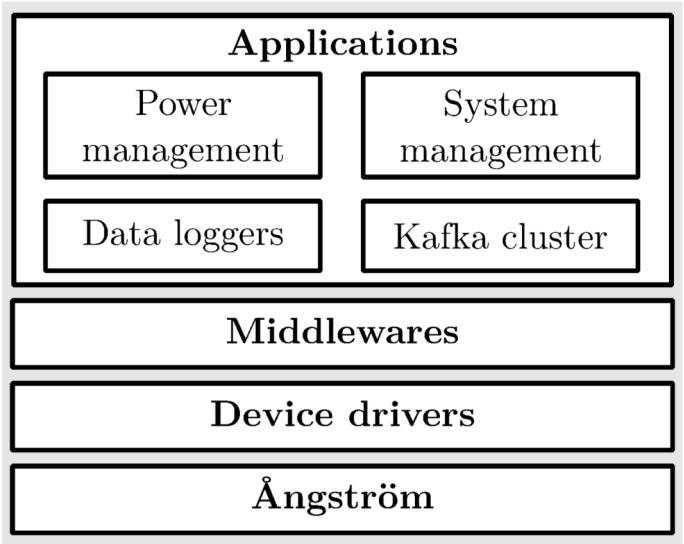

Figura 8 – Estrutura Board Support Package (BSP), desenvolvida com quatro áreas de aplicativos executados em um sistema operacional Ångström: gerenciamento de sistema, gerenciamento de energia, cluster de dados e registro de dados. (Fonte: 78)

informações de diagnóstico e geoespaciais da plataforma, armazenados no tópico remoto para um *cluster* Kafka remoto. Três produtores Kafka personalizados foram implementados para registrar *CAN*, *GNSS* e dados de diagnóstico: *kafka-can-log*, *kafka-gps-log* e *heartbeat*. Dois desses produtores, *kafka-can-log* e *heartbeat*, foram escritos em C e *kafka-gps-log* foi escrito em Python.

Cada aplicativo inicia sua operação conectando-se ao *cluster Kafka*, e, em seguida, tenta estabelecer conexão com uma fonte de dados específica para, posteriormente, entrar em um ciclo de leitura, serialização e, por fim, publicação de dados no *cluster Kafka*, por meio de *APIs* de cliente *Kafka*. As variações surgem das distintas fontes de dados utilizadas. Para ilustrar, o *kafka-can-log* cria um soquete de rede *SocketCAN* para escutar em um barramento *CAN*. Já o *kafka-gps-log* conecta-se ao *gpsd* via *gps3* para obter dados de *GPS*. Enquanto isso, o *heartbeat* verifica diretamente o *status* da conexão com a Internet e a intensidade do sinal celular a cada segundo por meio de um comando do sistema. Após a publicação dos dados de diferentes registradores no *cluster Kafka*, esses dados são armazenados no *SSD*.

3.2 Billion Object Platform (BOP)

Neste trabalho [79], com o apoio financeiro da *Sloan Foundation* e *Harvard Dataverse*, o *Harvard Center for Geographic Analysis (CGA)*, foi desenvolvida a *Billion Object Platform*, conhecida como *BOP*, uma robusta plataforma de visualização para dados

espaço-temporais. O principal propósito desse projeto é eliminar as barreiras enfrentadas por acadêmicos que buscam acessar extensos conjuntos de dados espaço-temporais em tempo real. Dada a rápida expansão dos dados de *streaming* após sua arquivagem e a limitação da maioria dos sistemas *GIS* em lidar com a visualização interativa de milhões de objetos, a criação de uma nova plataforma tornou-se substancial.

Na arquitetura principal da *Billion Object Platform (BOP)*, ilustrada na figura 9, o *BOP* é alimentado pelos mais recentes bilhões de *tweets* geográficos e recebe um fluxo constante de aproximadamente 1 milhão de *tweets* por dia em tempo real. Desde 2012, a *CGA* tem coletado e arquivado *geo-tweets*, enriquecendo-os com informações de sentimentos e dados censitários para facilitar análises mais profundas. A transmissão e armazenamento dos dados de entrada e intermediários são realizados através do *Apache Kafka*, arquivando *geo-tweets* usando um tópico de longo prazo. O núcleo do *BOP* é o *Apache Solr*, que oferece suporte a buscas rápidas. Significativas melhorias foram implementadas no *Solr*, incluindo a contribuição notável do facetamento de mapa de calor 2D para aprimorar a visualização espacial.

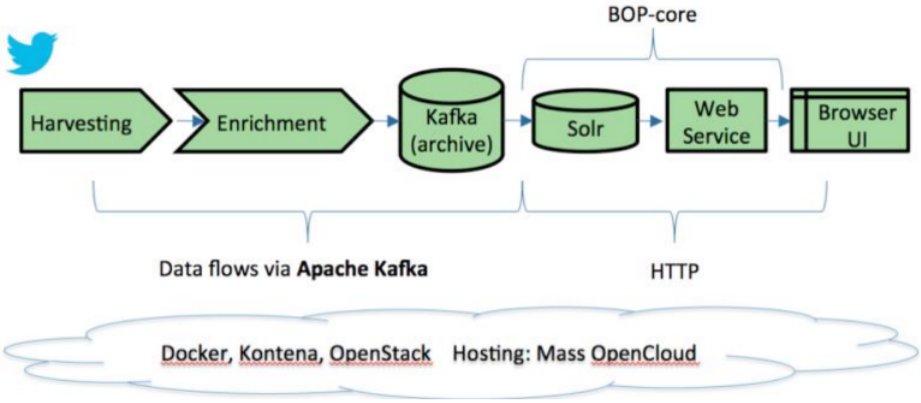

Figura 9 – Arquitetura lógica de alto nível BOP. (Fonte: 79)

O *BOP* disponibiliza ao *Solr* um serviço *web RESTful*, proporcionando uma *API* segura de pesquisa e extração, acessível a partir de um cliente baseado em navegador. O cliente desenvolvido dinamicamente exibe distribuições temporais e espaciais de resultados para conjuntos que contêm centenas de milhões de recursos. O sistema é de código aberto, compatível com *hardware* convencional e hospedado no *Massachusetts Open Cloud (MOC)*, uma plataforma *OpenStack*. Todos os componentes são implantados em contêineres *Docker* e orquestrados pela *Kontena*. O termo *BOP-core* é utilizado para referenciar tanto o índice *Solr* quanto o serviço da *web* associado.

Neste projeto, a *API Kafka Streams* é empregada para consumir de um *stream*, transferir conteúdo para outro *stream* e mover dados do *Kafka* para sistemas como o *Solr*. O *Kafka* é utilizado de maneira não convencional como um armazenamento persistente

de dados a longo prazo. Os *tweets* de arquivos são armazenados em partições mensais em um tópico *Kafka* de longo prazo, simplificando o processo, mas apresentando desafios como a impossibilidade de classificar ou desduplicar dados no *Kafka*. Foi configurado um *cluster Kafka* com 3 *zookeepers* e 3 *brokers*. Criaram 2 consumidores para enriquecimento, um ingeridor e 2 tópicos para entrada e saída. No processo de enriquecimento do *BOP*, ilustrado na figura 10, o tópico de entrada contém tweets em tempo real, alimentado pela colheitadeira. Este tópico tem os *tweets* consumidos pelo processo de enriquecimento e encaminhados para o tópico de saída. Posteriormente, os *tweets* enriquecidos são consumidos deste ultimo tópico pelo ingeridor *BOP*, que os transforma em documentos *Solr* antes de enviá-los para o *Solr*. A *API Kafka Manager* do *Yahoo* é utilizado para administrar o *Kafka*.

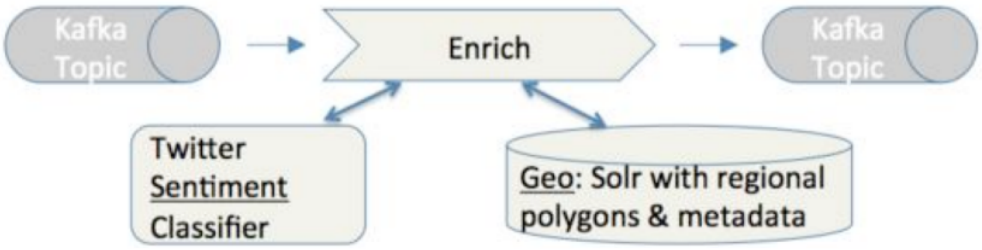

Figura 10 – Processo de enriquecimento no BOP. (Fonte: 79)

3.3 Polly

Neste trabalho [80], a arquitetura é composta por uma interface que possibilita aos usuários carregar seus dados em formatos como *Excel* ou *CSV*. Essa interface permite a aplicação de algoritmos de aprendizado de máquina, a escolha de diferentes parâmetros como variáveis de treinamento e a especificação de uma variável alvo para previsão. Após a seleção de atributos, os usuários podem realizar diversas análises estatísticas e operações de limpeza nos dados. Uma vez concluída a fase de pré-processamento, os usuários escolhem entre diversos algoritmos disponíveis de aprendizado de máquina. Após a seleção do algoritmo, ajustes nos hiperparâmetros podem ser feitos para atender aos requisitos específicos do aplicativo.

A interface *front-end* encaminha cada operação para o serviço web, implementado em *Python* usando *Flask*, um *framework web* leve. A arquitetura foi projetada com foco na escalabilidade, executando cada operação do *Polly* em contêineres Docker separados. Isso proporciona flexibilidade e capacidade de resposta, permite modelagem, limpeza e análise de dados de forma eficiente.

O fluxo de trabalho da arquitetura foi separado em quatro etapas. A primeira é alimentação de dados, permite o usuário realizar *upload* de seus dados em formato

Excel ou conectar-se ao banco de dados *SQL* remoto. A segunda é de análise estatística e processamento de dados, em que o usuário pode visualizar estatísticas básicas de seu conjunto de dados, realizadas automaticamente pela arquitetura, incluindo cálculo de média, moda, mediana, desvio padrão, valores percentuais e quantidade de valores nulos de cada coluna de dados e preenchimento ou exclusão de linhas e colunas que possuem valores ausentes. A terceira etapa é de seleção de algoritmo. Para a análise de regressão, têm-se os seguintes: *Linear, Lasso, Lars (LLR)*, *Elastic Net Regression (ENR)* e *Support Vector Regression (SVR)*. Para análise de classificação, incorporaram seis algoritmos: *Gaussian Naive Bayes (GNB)*, *Bernoulli Naive Bayes*, *Ada Booster*, *Decision Tree*, *Random Forest Tree* e *Support Vector Classifier*. Nesta etapa, também é possível o usuário ajustar o espaço de hiperparâmetros dos algoritmos selecionados. Por fim, na quarta etapa é realizado a visualização de resultados, o usuário pode avaliar os resultados do algoritmo selecionado com o auxílio de métricas definidas por cada tipo de análise, bem como comparar o desempenho de cada algoritmo selecionado.

O caso de uso utilizado, foi um conjunto de dados biofísicos e bioquímicos de plantas de soja coletados em um campo agrícola aberto no Missouri, EUA. O trabalho demonstra as capacidades de regressão e aprendizado de máquina de *Polly* na estimativa da clorofila foliar e concentração de nitrogênio, índice de área foliar, biomassa fresca acima do solo e biomassa seca da soja.

3.4 Estrutura de processamento de dados do Sentinel-2

O estudo de [81] apresentou uma estrutura de *Big Data* voltada para o processamento de dados, no âmbito das salvaguardas nucleares, provenientes da estação meteorológica *Sentinel-2*, vinculada ao programa *Copernicus*. Dada a diversidade potencial dos dados, uma preparação abrangente foi essencial para adequar os dados a formatos utilizáveis no processo, permitindo assim a aplicação de algoritmos de análise apropriados.

A fonte de dados é integrada ao sistema de gerenciamento de fluxo de trabalho *Apache Airflow*, que tem a capacidade de baixar, validar, pré-processar e armazenar os dados em um banco de dados *Rasdaman*. Para finalizar, a eficiência do *Google Earth Engine* foi explorada para executar de maneira efetiva fluxos de trabalho de *Big Data*, utilizando técnicas de aprendizado de máquina fornecidas pelo *Google*. O potencial da estrutura desenvolvida foi avaliado por meio de estudos de caso relacionados a locais vinculados ao ciclo do combustível. O objetivo principal consistiu em classificar o uso da cobertura do solo, uma vez que essas características oferecem informações cruciais, para identificar e monitorar mudanças no estado operacional, construção de novos edifícios, expansões de fábricas, entre outros.

Neste trabalho, a implementação do gerenciamento de fluxo de trabalho foi definida

em *DAGs*, com base em dois arquivos de configuração. O primeiro contém uma visão geral das áreas de interesse, definidas por um nome, o polígono correspondente e o satélite ao qual os dados devem ser solicitados. A configuração do satélite correspondente é armazenada em outro arquivo, onde seu nome, provedor e tipo de produto definem cada satélite. O *Open Access Hub* disponibilizou os dados no Nível 2A, corrigidos e ortorretificados.

Para cada provedor de satélite, foi criado um DAG, como ilustrado na figura 11. No entanto, como o *Copernicus Hub* só pode processar duas solicitações de servidor por vez, o nó *CopernicusHub_start* possui apenas dois nós filhos.

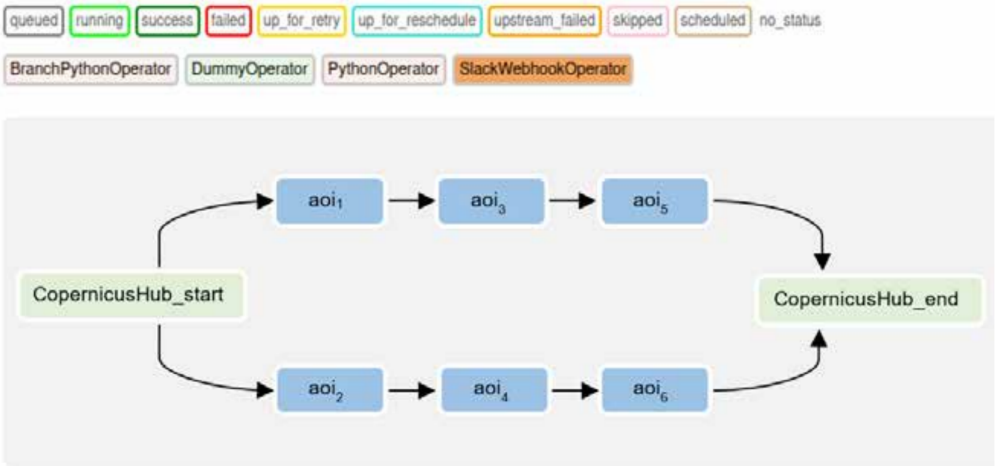

Figura 11 – DAG criado dinamicamente com base nos dois arquivos de configuração. (Fonte: 81)

No user interface (UI), os nós azuis representam os grupos de tarefas, que são um conceito de agrupamento de UI e úteis para criar repetições de padrões. Em cada grupo de tarefas, como ilustrado na figura 12, a mesma sequência de tarefas é executada, alterando apenas os parâmetros de entrada.

Figura 12 – Tarefas reais a serem executadas por área organizadas em Grupos de Tarefas. (Fonte: 81)

Diferente do *ISOBlue HD*, que utiliza o *Apache Kafka* para a transferência robusta de dados de uma colheitadeira, capturando dados significativos com foco na plataforma

e nas ações do operador para análise posterior e do *BOP*, que realiza a transmissão e armazenamento dos dados *geo-tweets* usando um tópico de longo prazo do *Apache Kafka* para enriquecimento e disponibilização dos dados para pesquisa e extração, este projeto tem como foco a coleta de dados agrícolas de diversas fontes com o *Apache NiFi*, como dados provenientes de bancos de dados, arquivos estruturados e não estruturados e dados de diferentes estações meteorológicas, além de realizar a transmissão desses dados para diversas tarefas escritas em *Python*, focadas em preparação de dados, geoprocessamento, visualização e monitoramento para análise de resultados.

Assim como a ferramenta *Polly*, este trabalho também busca aplicar contêineres *Docker*. Porém, diferente da ferramenta *Polly* que implantou contêineres *Docker* separados para integrar as execuções de cada operação, este projeto visa a aplicação dos contêineres *Docker* para a integração de diferentes ferramentas, bibliotecas e sistemas, como o *Apache Airflow*, *Apache Nifi* e bibliotecas *Python* de geoprocessamento, facilitando a criação do ambiente de execução dos processos, estabelecendo conexões eficientes entre as ferramentas integradas e permitindo escalar a arquitetura posteriormente.

Da mesma forma que o trabalho de Estrutura de processamento de dados do Sentinel-2 orquestra seus fluxos de trabalho com grafos acíclicos direcionados, a arquitetura de preparação de dados agrícolas que será desenvolvida também implantará o gerenciamento dos fluxos com a ferramenta *Apache Airflow*. Porém, diferente do projeto citado, em que apenas um pipeline de execução foi desenvolvido para diversos sistemas, alterando apenas as variáveis de entrada, este projeto visa o desenvolvimento de diversos pipelines de execução, para diferentes tipos de dados, atendendo mais de um tipo de processamento de dados e criação de modelos.

4 ARQUITETURA

Para a construção e desenvolvimento da arquitetura de gerência de dados para tarefas de aprendizado de máquina e análise de dados voltadas a agronomia, foi preciso definir quais ferramentas e bibliotecas seriam utilizadas em cada etapa do processo, desde a ingestão de dados até a apresentação dos dados, levando em consideração tanto suas funcionalidades quanto seu desempenho, sendo estes parâmetros indispensáveis durante a escolha das ferramentas que iriam suprir a necessidade de cada etapa da arquitetura.

Além disso, foi necessário destacar quais os benefícios que estes poderiam fornecer para o projeto, e as vantagens que teriam em relação as demais ferramentas, a curva de aprendizado para a sua utilização, observar a performance de cada ferramenta com a arquitetura e a compatibilidade da mesma com as demais ferramentas utilizadas em etapas diferentes.

Com isso, a arquitetura foi criada para atender 3 etapa importantes, a ingestão de dados em lote e em tempo real com o *Apache NiFi*, a orquestração de tarefas com o *Apache Airflow* e geoprocessamentos com bibliotecas *Python*, como o *GeoPandas*, *PyKrige* e *Rasterio*. Todas as ferramentas sendo executadas em contêineres *Docker*, possibilitando escalar ao longo do desenvolvimento da arquitetura.

Ao final da construção do *Docker compose*, incluindo todas as ferramentas necessárias de cada etapa do processo, desde a ingestão de dados até o geoprocessamento, a arquitetura de gerência de dados para tarefas de aprendizado de máquina e análise de dados na agricultura apresentou a estruturação ilustrada na figura 13.

Todas as ferramentas da arquitetura se comunicam fortemente, realizando cada etapa do processo como deveria. Iniciando com o *Apache NiFi*, ingerindo dados em lote de mapas e imagens para o sistema de arquivos local ou ingerindo dados de estações meteorológicas em tempo real e armazenando em uma área de *staging* no banco de dados. Programado ou quando solicitado pelo usuário, é executado tarefas de processamento no escopo do fluxo de trabalho do *Apache Airflow*, abordando tarefas de geoprocessamento, de ações com o banco de dados, de visualização, de filtragem de dados e outros processos. As tarefas de geoprocessamento são gerenciadas pelo orquestrador de tarefas *Apache Airflow*, elas podem incluir recorte de áreas de interesse, interpolação espacial, filtragem de áreas definidas com geometrias, entre outros tratamentos. Nas sessões adiante, são detalhados quais fatores colaboraram para que essas ferramentas fossem escolhidas e como foi configurado cada ferramenta na arquitetura.

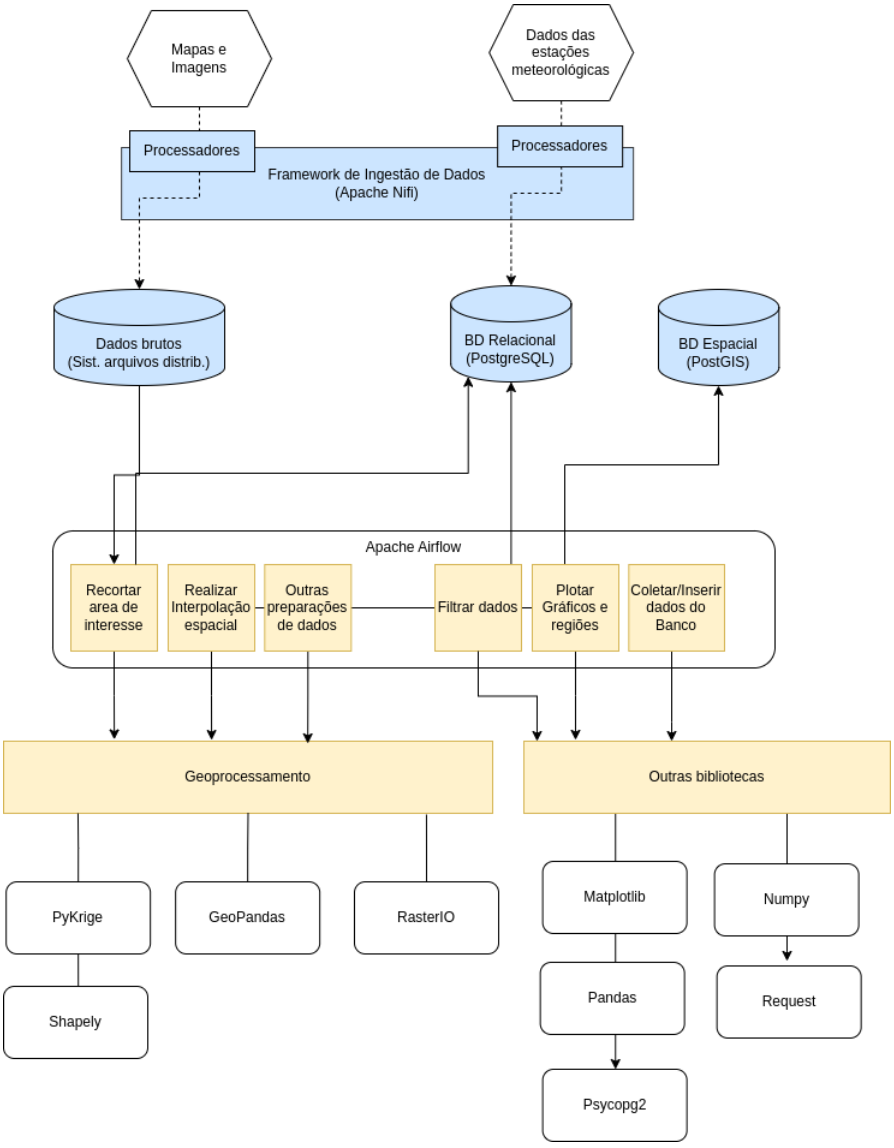

Figura 13 – Arquitetura de gerência de dados para tarefas de aprendizado de máquina e análise de dados na agricultura.

4.1 Integração

A ferramenta escolhida para ser possível integrar fortemente todas as aplicações e bibliotecas da arquitetura, além de disponibilizar futuras extensões com outras ferramentas, foi a plataforma de software livre *Docker*, por permitir empacotar, distribuir e executar as ferramentas em *contêineres*. Um *contêiner* é uma unidade leve e portável que contém tudo que uma ferramenta precisa para ser executada de forma isolada, incluindo código, *runtime*, bibliotecas e dependências.

O *Docker* é amplamente utilizado para construir arquiteturas baseadas em integrações entre diversas ferramentas, já que possui grande portabilidade e consistência, permitindo que o usuário empacote suas aplicações e suas dependências em contêineres,

garantindo que elas se comportem da mesma maneira em qualquer ambiente onde forem executadas, seja no desenvolvimento, teste ou produção [82].

Além disso, os *contêineres* do *Docker* também oferecem isolamento de recursos, em que cada ferramenta em execução em um *contêiner* possui seus próprios recursos, como CPU, memória e espaço em disco. Isso evita conflitos entre aplicativos e melhora a segurança, pois os *contêineres* não compartilham o mesmo espaço de sistema operacional [83].

Em conjunto, também é possível observar facilidade de implantação e escalabilidade, já que permite que o usuário implante facilmente vários aplicativos em *contêineres* em qualquer ambiente que tenha o *Docker* instalado, seja localmente em uma máquina de desenvolvimento ou em um ambiente de nuvem. Além disso, os *contêineres* são consideravelmente leves e inicializam rapidamente, facilitando a escalabilidade horizontal, onde novas instâncias podem ser adicionadas conforme necessário para lidar com a carga de trabalho [84].

Por fim, o *Docker* se mostrou uma alternativa com ótimo desempenho por também apresentar gerenciamento simplificado de dependências, em que o usuário pode especificar todas as dependências da ferramenta que deseja adicionar e personalizar em um arquivo de manifesto *Dockerfile*. Isso ajuda a facilitar a reprodução do ambiente de desenvolvimento em qualquer máquina e garante que todas as dependências necessárias estejam presentes [85][86].

4.2 Ingestão de dados

Para definir a ferramenta correta de coleta e alimentação de dados em diferentes etapas da arquitetura, foi necessário, como primeiro passo, se atentar aos tipos de dados que cada ferramenta é capaz de ingerir. Este primeiro passo, se mostrou de extrema importância para estabelecer o ponto inicial da definição de fatores que cada ferramenta deveria apresentar para ser utilizada na arquitetura.

Neste passo, levando em consideração que a arquitetura poderá ser alimentada com diversos tipos de dados, tanto em lote quanto em tempo real, provenientes de diversos sistemas, foi possível pontuar as ferramentas que mais se destacavam neste quesito. O *Apache NiFi* se mostrou uma ferramenta notável neste fator, por ser capaz de gerenciar fluxos de dados em lote e em tempo real, entre diferentes sistemas. O *Apache Kafka* não ficou muito atrás, apesar de não ingerir dados em lote, se mostra uma grande ferramenta de *streaming* de mensagens de alto desempenho, superando as demais ferramentas em quesito escalabilidade. Já o *Apache Flume*, assim como o *Apache Kafka*, também se trata de uma ferramenta de streaming, porém de dados fontes gerados continuamente. O *Apache Sqoop*, assim como o *Apache NiFi* ingere dados em lote, mas não ingere dados em tempo

real, além de se tratar de uma ponte de transferência de dados apenas entre o Hadoop e bancos de dados relacionais ou mainframes. Para o fator tipo de dados, o *Apache NiFi* e o *Apache Kafka* apresentaram melhores resultados [87][88][89][90][91].

O segundo passo, contou com a análise de pré-processamentos de dados disponíveis em cada ferramenta, desfavorecendo o *Apache Sqoop*, o qual não possui funcionalidades para tratamento de dados. Já o *Apache Kafka* e o *Apache Flume* possuem apenas processamentos limitados de streaming, como transformações, agregações e junções, diferente do *Apache NiFi* que oferece uma ampla gama de funcionalidades para pré-processamento de dados, como transformação, validação, filtragem, agregação, enriquecimento e movimentação de dados em diversos ambientes distribuídos, por meio de processadores integrados para manipulação de dados [92][93][94][95].

No terceiro passo, foi observado a curva de aprendizagem e a interação usuário-ferramenta. Pelo fato do *Apache NiFi* ser a única ferramenta, dentre as citadas acima, que possui uma interface de usuário gráfica (GUI) nativa e intuitiva, que possibilita aos usuários construir, monitorar e gerenciar fluxos de dados de maneira visual e interativa, o torna mais acessível para uma variedade de usuários, incluindo os que não são familiarizados com codificação manual, oferecendo uma curva de aprendizado mais suave, diferente das demais ferramentas, que são configuradas e gerenciadas por meio de linha de comando ou por meio de APIs, embora essas ferramentas possam ter interfaces de monitoramento [96][97][98][99].

Visando atender totalmente, cada um desses fatores citados acima, o *Apache NiFi* foi selecionado para ser responsável pela ingestão de dados na arquitetura, mesmo que seu desempenho não supere o do *Apache Kafka*, o *Apache NiFi* se mostrou a ferramenta perfeita para suprir as necessidades da arquitetura em questão, tendo em vista que ele é uma escolha favorável devido à sua facilidade de uso e flexibilidade, vasto conjunto de funcionalidades e processamentos, carregamento orientado e não orientado a eventos e ingestão de dados em lote e em tempo real.

O *Docker compose* da arquitetura, utiliza a imagem oficial do *Apache NiFi* e define variáveis de ambiente para configurar credenciais de usuário único e fuso horário. Além disso, são montados volumes para persistir os dados do *Apache NiFi*, incluindo *logs*, configurações e repositórios de dados. O *Apache NiFi* é configurado para utilizar o protocolo *HTTPS* na porta 8443 e é iniciado com um *script* que instala dependências adicionais necessárias. Essa configuração do *Apache NiFi* proporciona um ambiente pronto para uso, permitindo a configuração e execução de fluxos de dados de forma segura e confiável, além de ser totalmente extensível para futuras integrações.

4.3 Gestão de fluxo de trabalho

Nesta etapa de gerenciamento de fluxo de trabalho, foi preciso analisar nas ferramentas quais as funcionalidades cada uma oferecia, desempenho, interface de usuário, logs e outras informações de execução e quais linguagens de programação cada ferramenta reconhecia para o processo de execução das tarefas.

No primeiro passo, foi analisado se as ferramentas selecionadas possuíam agendamento de tarefas, já que este fator seria muito importante para a arquitetura, visando a automatização de processos, em que suas execuções se dariam diariamente ou até mesmo em qualquer outro tempo definido pelo usuário. Após as pesquisas realizadas, foi observado que apenas o *Apache Airflow* apresentava esta funcionalidade.

Em seguida, o próximo fator analisado foi a interface de usuário e a curva de aprendizado. O *Kedro* e o *Luigi* possuem uma interface de linha de comando (CLI), em que o usuário pode utilizar para interagir com os pipelines e gerenciar projetos, mas não possuem uma interface de usuário como o *Apache Airflow*, que facilita questões de monitoramento e gerenciamento de pipelines com auxílio de logs, gráficos e pipelines mais detalhados, além de também já possuir uma CLI. Por contar com uma interface pensada no usuário, o *Apache Airflow* se torna mais fácil de trabalhar, ajuda na compreensão do processo de execução, em *debugs* de erros, gerenciar as tarefas, atribuir valores a variáveis globais do *Apache Airflow*, que podem ser utilizadas nas tarefas, sem utilização de linha de código e iniciar/pausar execução de pipelines com apenas um *click* [100][52].

O terceiro detalhe a ser analisado foi a questão de monitoramento e alerta durante a execução dos pipelines. Neste quesito, foi observado que o *Kedro* e o *Luigi* não apresentam funções de monitoramento e alerta para execuções de tarefas, mas é possível integrá-los com ferramentas de monitoramento externas para alertas personalizados. Já o *Apache Airflow*, oferece recursos integrados para monitorar o *status* das tarefas e enviar alertas em casos de falhas [101][102].

O último fator e o mais importante, foi verificar a extensibilidade das ferramentas, já que esta deve ser capaz de se comunicar fortemente com as demais ferramentas da arquitetura, localizadas em outras etapas do processo. Com isso, após pesquisas realizadas, foi possível observar que o *Kedro* mostrou ser uma ferramenta altamente extensível, já que permite a integração com diversas bibliotecas e ferramentas *Python*. Com relação ao *Apache Airflow*, foi observado que esta ferramenta possuía uma característica extremamente extensível, por suportar *plugins* para estender suas funcionalidades e integração com bibliotecas e ferramentas *Python*. Já o *Luigi* por se tratar de uma biblioteca *Python*, não apresenta um nível alto de extensibilidade, já que apenas permite a criação de módulos personalizados para adaptar o seu comportamento [103][104][105].

Após a análise desses fatores citados acima, foi determinado que a ferramenta que

melhor se encaixaria na arquitetura para o gerenciamento de fluxos de tarefas seria o *Apache Airflow*, considerando que esta se destacou em todos os fatores avaliados, apresentando agendamento de tarefas, uma interface de usuário interativa e monitoramento e alertas durante a execução de *pipelines*, além de ser uma ferramenta extremamente extensível, facilitando sua integração com as outras ferramentas da arquitetura.

Para a instalação do *Apache Airflow* na arquitetura, foi utilizado uma imagem oficial da ferramenta definida no *Docker Compose* responsável por integrar todos os recursos e aplicações. Esta imagem do *Apache Airflow* foi personalizada em um arquivo de configuração *Dockerfile*, para ser possível instalar bibliotecas *Python* nos *contêineres* que executariam os serviços do *Apache Airflow*, visando o uso dessas bibliotecas nos *scripts/pipelines* que seriam executados dentro do *Apache Airflow*.

O *Apache Airflow* utiliza o executor *CeleryExecutor* com *Redis* e *PostgreSQL* como *backend*. Os serviços do *Apache Airflow* são definidos com várias variáveis de ambiente que configuram a conexão com o banco de dados *PostgreSQL*, o uso do *Celery* como *broker* e *result backend*, e outras configurações importantes, como a chave *Fernet* para criptografia. Além disso, o *Docker Compose* também monta volumes para persistir os dados do *Apache Airflow* dentro da arquitetura, como as *DAGs* desenvolvidas pelos usuários, logs e configurações. Com isso, os serviços do *Apache Airflow* são configurados para funcionar em um ambiente local de desenvolvimento, além de também fornecer instruções sobre como personalizar a configuração para outros ambientes, como o ambiente de produção.

4.4 Ferramentas de Geoprocessamento

Nesta etapa foram utilizadas diversas bibliotecas *Python* para atender as necessidades de tratamento de dados agrícolas. Uma delas foi a biblioteca *GeoPandas*, que é capaz de estender os tipos de dados da biblioteca pandas para ser possível realizar operações espaciais em dados geoespaciais. O *GeoPandas* simplifica o trabalho com dados geoespaciais ao permitir a realização de leitura, escrita e manipulação desses dados em formatos como *GeoJSON* e *Shapefiles*, além de também oferecer funcionalidades para análise e visualização de dados [106][107][108].

Em seguida, foi adicionado a arquitetura a biblioteca *Rasterio* do *Python*, responsável por realizar a leitura e escrita de dados *raster* (imagens) em formatos como *GeoTIFF*. Esta ferramenta fornece comandos simples de se utilizar para trabalhar com dados *raster*, possibilitando a leitura de metadados, acesso a *pixels* individuais e manipulação de imagens *raster* [109][110].

Já a biblioteca *Shapely*, também do *Python*, foi incluída na arquitetura para disponibilizar funções de manipulação e análise de dados geométricos, como pontos, linhas e polígonos. Ao utilizar o *Shapely*, o usuário poderá realizar operações geométricas indispen-

sáveis em tratamentos de dados geoespaciais, por meio das funcionalidades de união, interseção, diferença e *buffer* em objetos geométricos que a biblioteca disponibiliza [111][112].

Outra ferramenta adicionada foi a *PyKrige*, uma biblioteca *Python* focada em interpolação espacial, sendo este o processo de estimar valores em locais desconhecidos com base em valores conhecidos em locais próximos. O *PyKrige* foi escolhido principalmente por implementar vários métodos de interpolação, incluindo o método de *krigagem*, em que o usuário pode selecionar um modelo dentre vários outros disponibilizados pela biblioteca para a interpolação de dados espaciais [113][114][115].

Além dessas bibliotecas de geoprocessamento que foram incluídas a arquitetura, outras bibliotecas *Python* também foram adicionadas para realizar tratamento de dados não espaciais, dados estes que também seriam úteis para a geração de mapas diários de precipitação, temperatura, umidade e outros fatores, recortes de área de interesse, construção de gráficos de ocorrências médias de determinado elemento climático e várias outras saídas possíveis que a arquitetura seria capaz de gerar.

5 ESTUDO DE CASO

Neste capitulo, são apresentados os estudos de caso desenvolvidos, para ser possível observar os resultados e desempenho que a arquitetura obteve em cada etapa do processo, ingerindo dados tanto em lote quanto em tempo real, gerenciando e processando os fluxos de dados e gerenciando os fluxos de trabalho, para a execução de tarefas responsáveis pela realização de diversos geoprocessamentos e armazenamento de dados.

5.1 Área de interesse no MapBiomas

O primeiro caso desenvolvido foi o recorte da área de interesse dos mapas anuais de cobertura e uso da terra do Brasil, localizados na coleção 8 no site do *MapBiomas*. Essa coleção cobre o período de 1985 a 2022, em resolução espacial de 30 m.

As classificações que resultam nos mapas de cobertura e uso da terra para cada ano, são realizadas a partir dos mosaicos *Landsat*. Conforme a estrutura proposta pelo *MapBiomas*, esses mapas são atualizados toda vez que ocorrer um aperfeiçoamento nos algoritmos de classificação.

Devido a isso, para trabalhar com os mapas de cobertura e uso da terra para cada ano, foi necessário construir um fluxo de dados bem estruturado, de modo a realizar a ingestão de dados em lote sempre que o usuário solicitar dados atualizados. Como os mapas são produzidos anualmente, ou somente quando há melhorias nos algoritmos de classificação, não houve a necessidade de configurar uma ingestão de dados em tempo real, mas sim em lote, por ingerir números grandes de arquivos produzidos em um longo período.

Para iniciar este processo de ingestão de dados em lote dos mapas de cobertura, foi desenvolvida a tarefa *downloader_mp*, ilustrada na figura 14, pertencente a *DAG downloader_map_biomas*, que foi criada no escopo do *Apache Airflow*. Ela foi desenvolvida com o operador *PythonOperator* para ser possível escrever a mesma em formato de *script Python*, facilitando a construção do *JSON* de *URLs*.

Esta tarefa é responsável por enviar ao processador *GenerateFlowFile* do *Apache NiFi*, o *JSON* contendo todas as URLs definidas pelo usuário para *download* dos mapas de cobertura, por meio de requisições HTTP utilizando o método *GET* em cada *URL* do *JSON*, e com isso, cada *URL* é acessada para realizar o *download* de um arquivo diferente do *MapBiomas*. Esta tarefa, também é responsável por iniciar o processo de ingestão dos arquivos do *MapBiomas*, no fluxo de dados desenvolvido na interface do *Apache NiFi*, ao mudar o valor da propriedade *state* do grupo de processadores *DownloadLote_MapBiomas*, ilustrado na figura 15, de *STOPPED* para *RUNNING*.

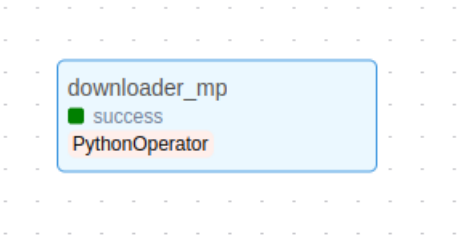

Figura 14 – Dag para iniciar o download em lote dos arquivos do MapBiomas.

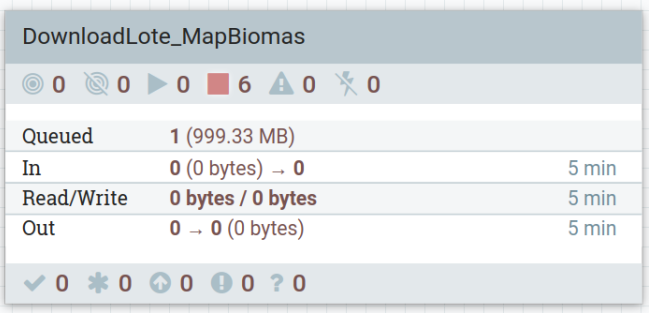

Figura 15 – Grupo de processadores para ingerir mapas de cobertura MapBiomas.

No *Apache NiFi*, o fluxo de dados ilustrado na figura 16, que foi construído dentro do grupo de processadores da figura 15, é iniciado com o processador *GenerateFlowFile*, configurado para receber uma lista de *URLs* em formato *JSON*. Este *JSON* é introduzido no processador por meio da *API do Apache NiFi*, em que é invocado por um *PythonOperator* no *Apache Airflow*, e é copiado para a propriedade *Custom Text* do *GenerateFlowFile*.

O próximo passo envolve o processador *SplitJson*, responsável por dividir o *JSON*, separando o objeto da chave *urls* do *JSON* em varios textos de uma linha, em que cada linha possui uma *URL* da lista para processamento individual.

Após essa divisão, o fluxo prossegue para o *ExtractText*, onde cada *URL* é extraída do fluxo de dados e armazenada em um novo atributo chamado *valorurl*, com cada *Flowfile* do *Apache NiFi* armazenando neste atributo uma *URL* específica. Este processador também usa uma expressão regular para extrair o nome do arquivo de cada *URL* e passa essa informação para o atributo *valorFileName*.

Com esses atributos definidos, o fluxo segue para o processador *UpdateAttribute*, onde o atributo *url* é criado/atualizado com o valor do atributo *valorurl* para cada *FlowFile*, e o atributo *filename*, padrão para nomear os arquivos dentro do *Apache NiFi*, é atualizado com o valor no atributo *valorFileName*.

Esta atribuição de valores é essencial para a execução do próximo processador, o *InvokeHTTP*, que utiliza o valor do atributo *url* para realizar o *download* de cada arquivo.

A propriedade *Remote URL* do *InvokeHTTP* é configurada para utilizar a expressão regular *${url}*, garantindo que o *download* seja realizado conforme a *URL* especificada em cada *FlowFile*.

Por fim, o fluxo de dados é concluído com a execução do processador *PutFile*, que recebe os arquivos baixados pelo *InvokeHTTP* e os grava no sistema de arquivos. O nome de cada arquivo gravado é determinado pelo valor contido no atributo *filename*. Dessa forma, o fluxo de dados desenvolvido, automatiza o processo de ingestão de arquivos em lote do *MapBiomas*, tratando cada *URL* de forma individual e eficiente, do recebimento da lista de *URLs* por meio da tarefa *downloader_mp* executada no *Apache Airflow* até o armazenamento dos arquivos baixados com seus respectivos nomes.

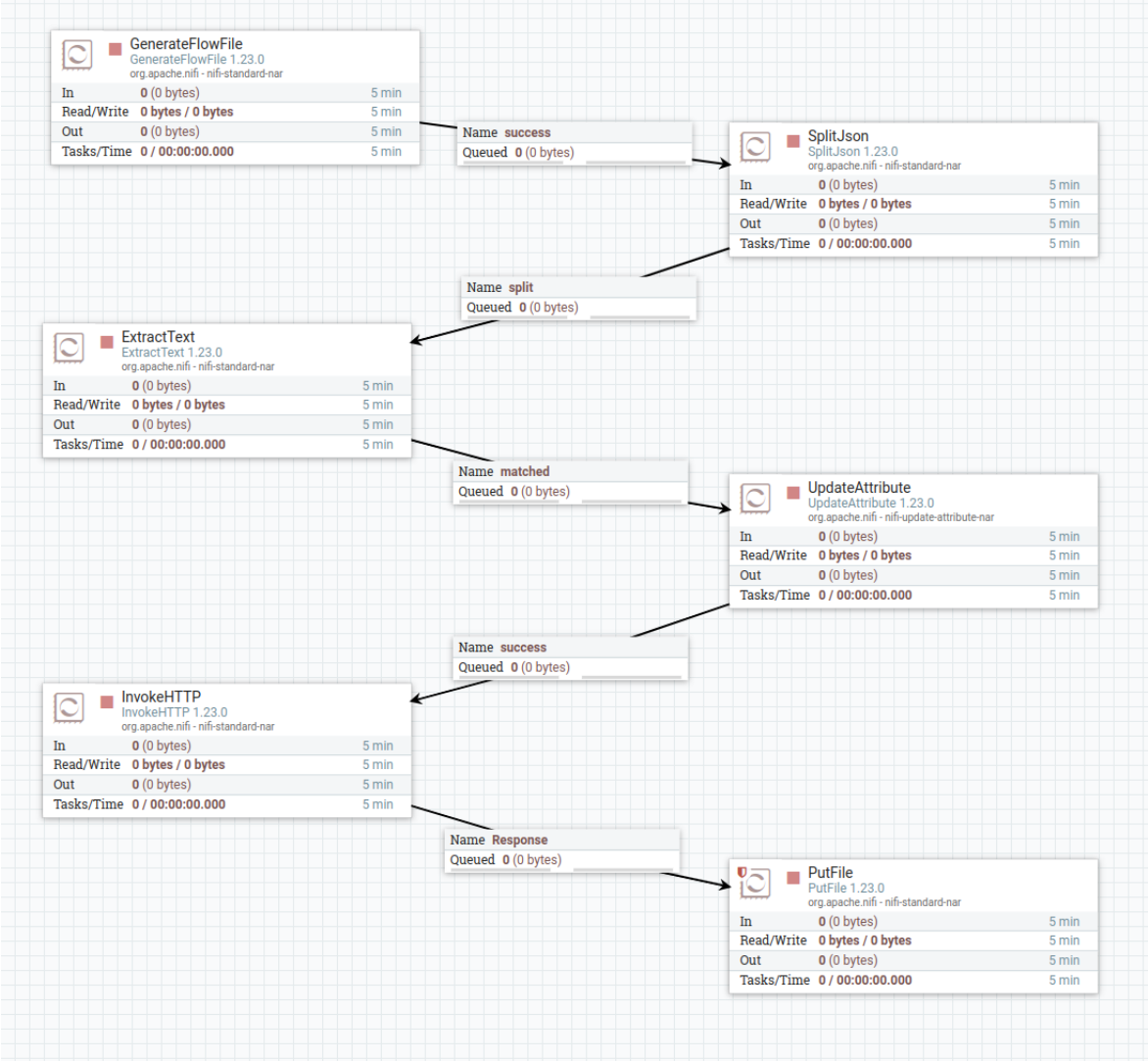

Figura 16 – Fluxo dos arquivos do MapBiomas ingeridos em lote no Apache NiFi.

Dando continuidade ao processamento, o fluxo de trabalho do *Apache Airflow* representado na figura 17, configura o *DAG pipeline_map_biomas* para realizar tarefas de

geoprocessamento nos mapas de cobertura do *MapBiomas*, utilizando bibliotecas especializadas em filtragem e segmentação de regiões em dados geográficos.

A tarefa *start* utiliza um *EmptyOperator*, é executada apenas para iniciar a *DAG* com uma operação vazia, servindo como um ponto de início lógico.

A tarefa *cria_variaveis_airflow*, definida com o operador *PythonOperator*, inicia um processamento de fato, onde as variáveis globais do *Apache Airflow* são definidas. Estas variáveis incluem o diretório em que o mapa de contorno do *IBGE* está localizado, o diretório em que os arquivos do *MapBiomas* foram salvos após a ingestão de dados realizada no fluxo de dados criado no *Apache NiFi*, o diretório que o usuário deseja armazenar os mapas de cobertura limitados a área de interesse e o estado do Brasil que o usuário deseja filtrar como área de interesse.

Em seguida, o fluxo continua com a execução da tarefa *coleta_dados_subdag_1*, que utiliza um *TriggerDagRunOperator*, para iniciar a execução de outra *DAG*, a *coleta_dados*, responsável por coletar os caminhos dos arquivos do *IBGE*, ou seja, os mapas de contorno dos municípios do Brasil e armazená-los como uma lista de diretórios na variável arquivos. O código para esta *sub-DAG* também filtra os arquivos no diretório para incluir apenas aqueles com extensão *.shp* ou *.json*, que são arquivos no formato *Shapefile* e *GeoJSON* respectivamente, utilizados em geoprocessamento.

Após a execução da tarefa acima, é executada a tarefa *filter_region* que utiliza o operador *PythonOperator*. Neste passo, é realizado uma operação de filtragem, lendo as variáveis 'arquivos' e 'UF', utilizando a primeira para carregar os arquivos dos mapas de contorno do *IBGE*, usando a biblioteca *GeoPandas*, e a segunda para filtrar os mapas para a unidade federativa especificada. Por fim, converte a geometria obtida para *WKT (Well-Known Text)*, para ser possível armazenar a área de interesse na variável global do *Apache Airflow* chamada 'regiao'.

Com isso feito, é executada a tarefa *recorta_regiao_subdag_1*, escrita com o operador *TriggerDagRunOperator*, acionando a execução da *DAG recorta_regiao*. Esta *DAG* é responsável por executar um geoprocessamento, que recorta a região de interesse, definida anteriormente, no arquivo *raster* dos mapas de cobertura do MapBiomas. Isso é feito a partir do carregamento da geometria armazenada na variável 'regiao', convertendo-a de volta para o formato de geometria do *GeoDataFrame*, da biblioteca *GeoPandas* e utilizando a função *mask* da biblioteca *Rasterio*, para recortar o *raster* baseado nessa geometria, salvando o *raster* resultante no diretório de saída especificado pelo usuário.

Finalizando o fluxo de trabalho, tem-se a tarefa end, que assim como a tarefa start, utiliza um EmptyOperator, concluindo as execuções com uma operação vazia, indicando o fim do processo.

Resumindo, o fluxo de trabalho *pipeline_map_biomas* desenvolvido no *Apache*

Airflow, gerencia um processo de geoprocessamento que inclui a coleta de arquivos geográficos, os mapas de contorno do Brasil, construídos pelo *IBGE* e os mapas de cobertura do Brasil, construídos pelo *MapBiomas*, filtragem de dados baseada em uma unidade federativa determinada pelo usuário, e o recorte dos arquivos *raster* do *MapBiomas* para obter a região de interesse. As operações são realizadas em diferentes *sub-DAGs* e utilizam as variáveis definidas no *Apache Airflow* para passar dados entre as tarefas. As ferramentas de geoprocessamento utilizadas, como *GeoPandas* e *Rasterio*, foram essenciais para manipular os mapas inseridos na arquitetura e realizar as operações de filtragem e corte de *raster*.

Figura 17 – Fluxo de trabalho para extrair área de interesse do mapa anual de cobertura no Apache Airflow.

A figura 18 apresenta um dos mapas anuais de cobertura e uso da terra, que foi extraído do site oficial do *MapBiomas*, após o processo de ingestão de dados em lote utilizando o *JSON* de *URLs*, criadas pela concatenação da *URL* base do *MapBiomas* com a iteração da lista de anos informada pelo usuário.

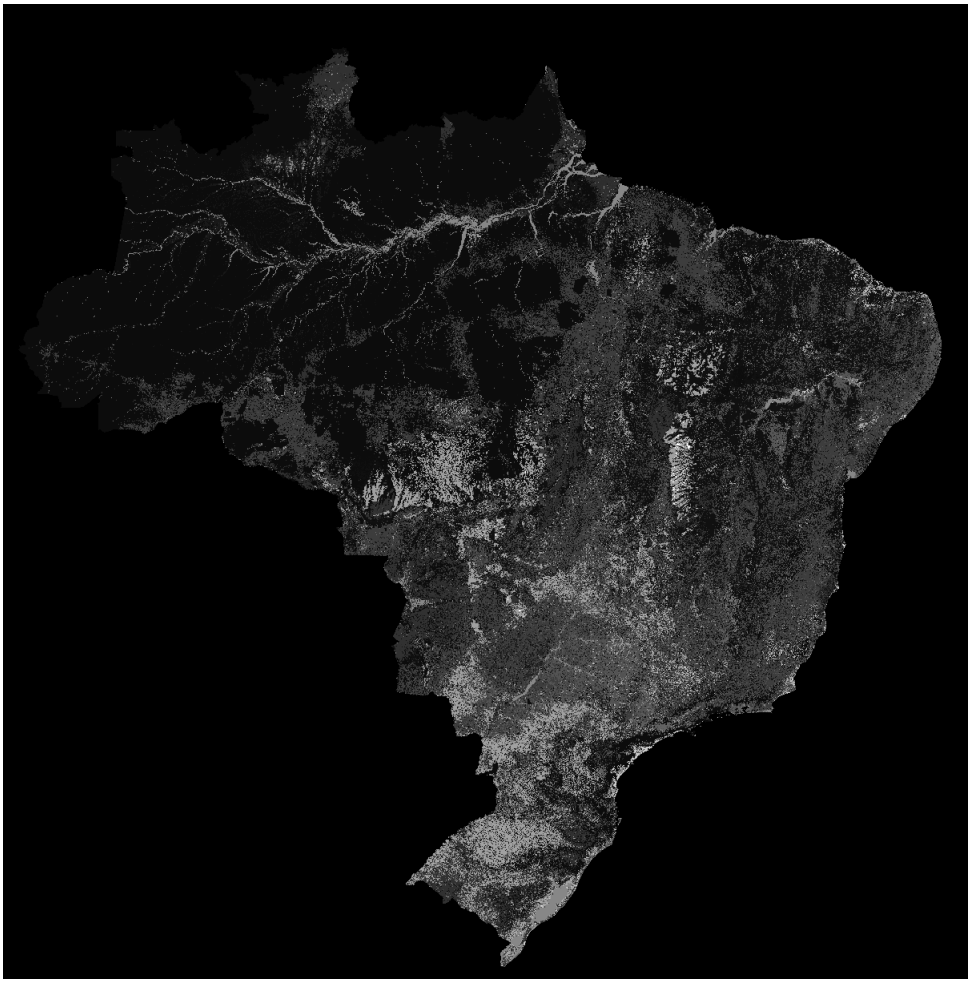

Figura 18 – Mapa anual de cobertura e uso da terra do Brasil produzido pelo MapBiomas.

Na figura 19, tem-se o mapa de contorno utilizado para realizar a delimitação da área de interesse, informada previamente pelo usuário nos parâmetros do fluxo de trabalho. Este arquivo foi obtido por meio de um *download* direto do usuário no site oficial do *IBGE*. Visto que este arquivo é único para todas as execuções de recorte da área de interesse no mapa de cobertura, não houve a necessidade de desenvolver um *pipeline* para ingerir este mapa de contorno automaticamente.

Figura 19 – Camadas dos municípios no mapa do Brasil produzido pelo IBGE.

Por meio da filtragem realizada no mapa de contorno do *IBGE*, definindo a área de interesse, foi gerado como resultado o mapa de contorno ilustrado na figura 20, que possui as camadas de interesse definidas pelo usuário para a realização do recorte dessa área no mapa de cobertura.

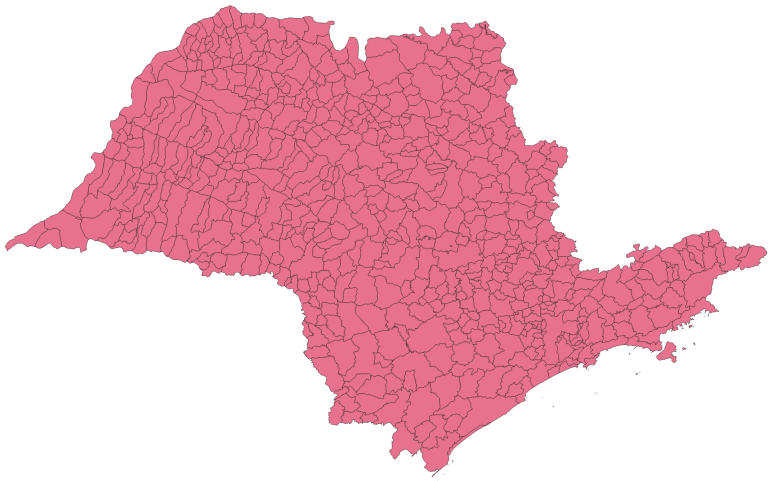

Figura 20 – Camadas das áreas de interesse extraídas do mapa do Brasil.

Por fim, como resultado da execução do fluxo de dados, geoprocessamento e fluxo de trabalho, foi construído o mapa de cobertura, apresentado na figura 21, possuindo apenas a área de interesse definida pelo usuário, por meio da definição da lista de anos de interesse e da área de interesse somente.

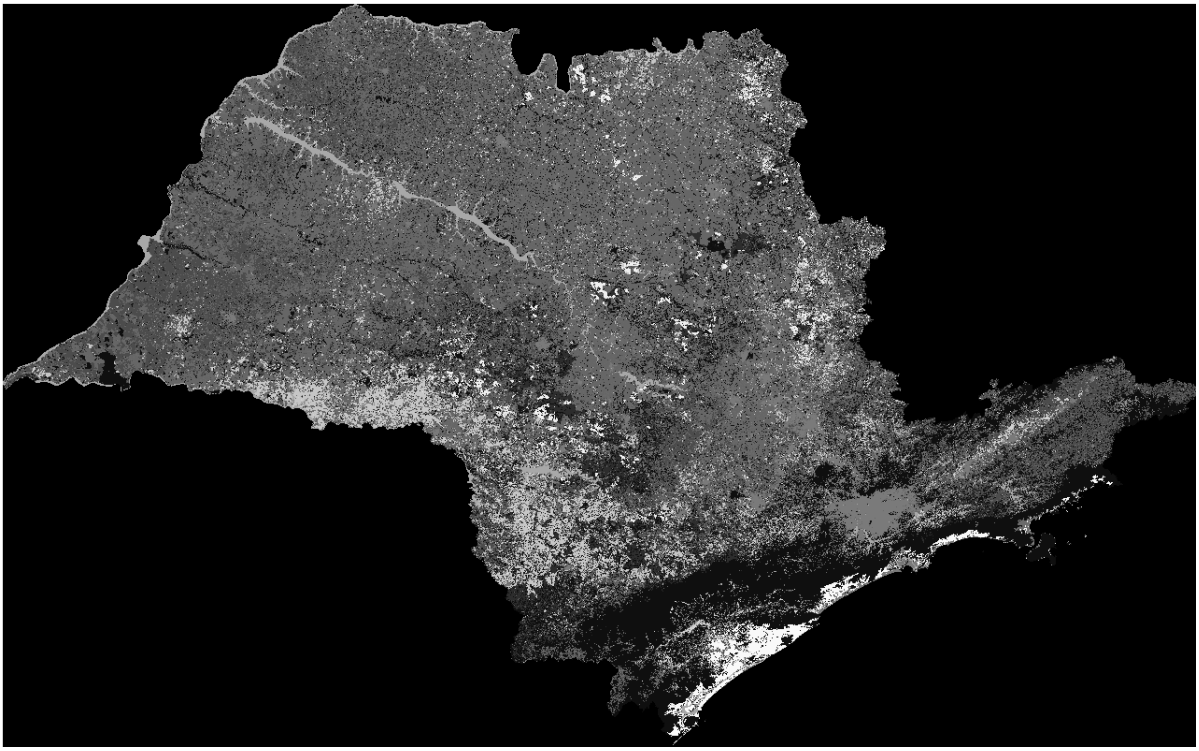

Figura 21 – Área de interesse do mapa anual de cobertura e uso da terra do Brasil.

5.2 Mapa diário de dados meteorológicos

Diferente do caso anterior que realizou ingestão de dados em lote, este estudo de caso foi desenvolvido para ingerir dados em tempo real, provenientes de diversas estações meteorológicas do estado do Paraná, por meio de um fluxo de dados no *Apache NiFi*, e processar esses dados com bibliotecas de geoprocessamento do *Python*, como o *GeoPandas* e o *PyKrige*, em um fluxo de tarefas desenvolvido no *Apache Airflow*. Ao final dessas execuções, tem-se como resultado, os mapas diários de diferentes dados meteorológicos, a partir da aplicação da interpolação espacial nesses dados.

Os dados utilizados neste estudo de caso, foram cedidos pelo professor orientador deste trabalho de conclusão de curso, que solicitou o conjunto de dados de um instituto especializado em dados meteorológicos, com diversas estações meteorológicas localizadas no estado do Paraná, coletando diariamente dados de diferentes variáveis atmosféricas, como de temperatura, umidade, vento, chuva, radiação solar e outras variáveis.

Para desenvolver o fluxo de dados que seria responsável por coletar os dados meteorológicos em tempo real, foi preciso desenvolver uma aplicação *HTTP* com o *framework FastAPI*, que ficou encarregada de realizar a leitura dos dados no arquivo *export_automaticas.csv* e criar um sistema de *streaming*, para enviar esses dados no formato *JSON Schema* aos clientes que iriam requisitá-los pela *API*.

A *API* desenvolvida, contou apenas com um método *HTTP*, nomeado como *stream*, que ficou responsável por permitir requisições do tipo *GET*, para que o cliente pudesse receber os dados a cada solicitação. Para simular precisamente como é realizado o sistema de *streaming* de dados em estações meteorológicas, o método *stream* desenvolvido, cria uma resposta de *streaming* usando a função *StreamingResponse*, da biblioteca *FastAPI*, que chama a função *generate_messages* desenvolvida para enviar continuamente dados extraídos do *CSV* ao cliente, linha por linha, com o tipo de mídia da resposta definido como *text/event-stream*, que é um formato padrão para *streaming* de eventos, e assim, um novo item é enviado a cada chamada subsequente.

O *endpoint* para solicitar esses dados na aplicação *FastAPI*, foi definido como */stream*, e como a mesma é executada localmente neste caso de uso, para acessá-la é preciso se conectar a *URL http://localhost:8800/stream*, com a porta sendo definida no momento da execução da *API* desenvolvida.

Seguindo para a ingestão de dados na interface do *Apache NiFi*, foi configurado um fluxo de dados, que seria responsável por fazer requisições do tipo *GET*, a cada segundo, no *endpoint* da *API* desenvolvida, para coletar cada linha produzida de dados meteorológicos, processá-los no fluxo e inseri-los no banco de dados.

Antes de configurar os processadores que iriam realizar os processamentos descritos anteriormente, foi preciso configurar o banco de dados, para que o fluxo que se-

ria desenvolvido pudesse estabelecer uma conexão com o banco de dados, que fica em execução localmente, para realizar as inserções. Para isso, foi utilizado o serviço de controlador *DBCPConnectionPoll*, configurado com a *URL* de conexão do banco de dados *jdbc:postgresql://192.168.0.106:5433/postgres*, com o usuário sendo o padrão *postgres*, e a senha, ocultada nas configurações após ser inserida nas propriedades, como ilustrado na figura 22. Para que a conexão fosse possível, também foi preciso mapear o diretório em que o *Driver* do banco de dados *PostgreSQL* estava localizado no computador.

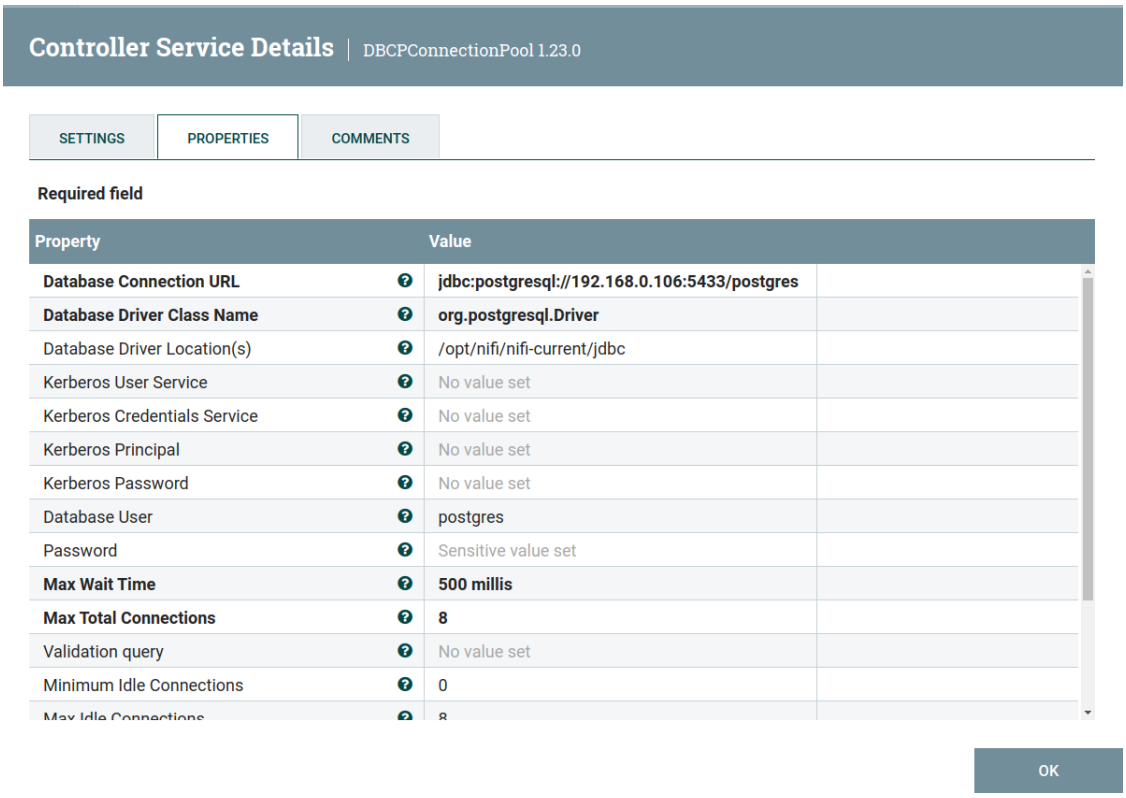

Figura 22 – Configurações do serviço de controlador DBCPConnectionPoll no Apache NiFi.

Após concluída e validada a conexão com o banco de dados, foi preciso realizar a conexão com a tabela neste banco de dados configurado anteriormente, que seria responsável por armazenar os dados meteorológicos que viriam do fluxo, após ingeri-los da *API* desenvolvida. Para isso, foi utilizado o serviço de controlador *DatabaseRecordSink*, apresentado na figura 23, que foi configurado para se conectar com o banco definido acima, no esquema *public* e tabela *api_stream*. Esta tabela, é utilizada como uma área de *staging* para os dados meteorológicos, já que ainda não passaram pela etapa de tratamento, que precisa ser executada antes dos dados serem utilizados na construção dos mapas diários.

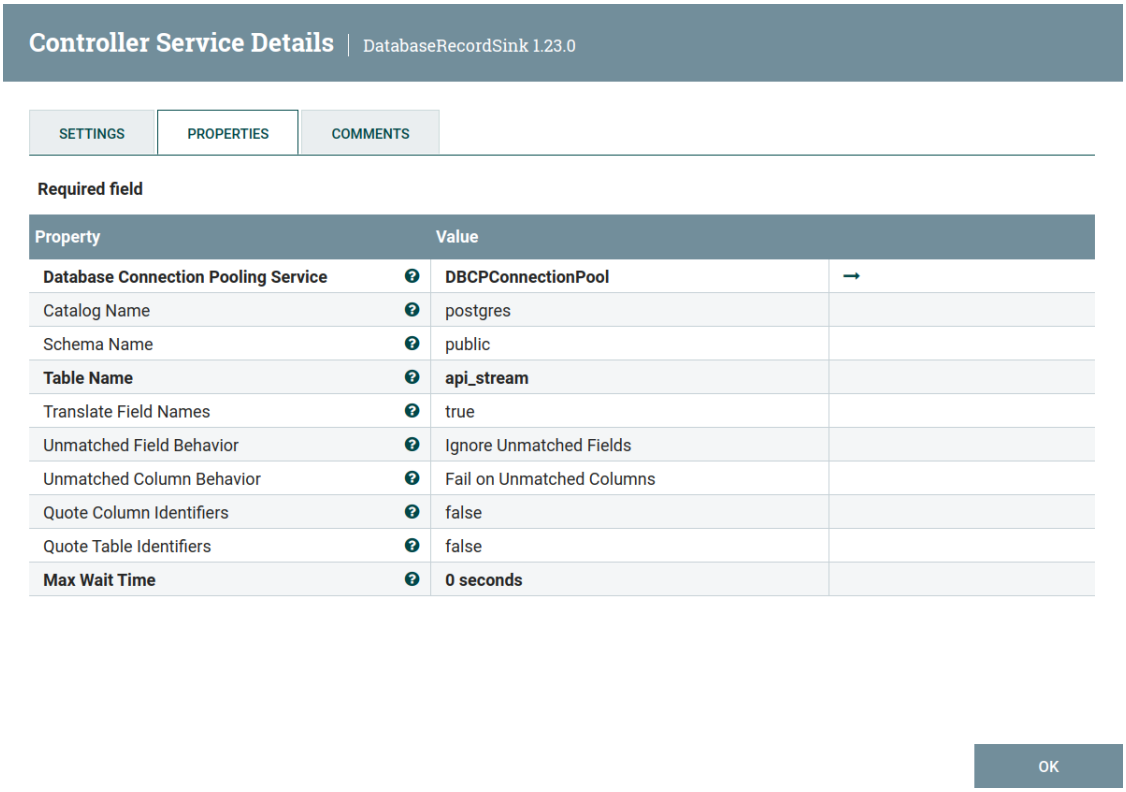

Figura 23 – Configurações do serviço de controlador DatabaseRecordSink no Apache NiFi.

Concluída as configurações necessárias com o banco de dados e a tabela de *staging*, foi construído o fluxo de dados por meio de uma sequência de processadores do *Apache NiFi*, apresentado na figura 25, dentro do grupo de processadores ilustrado na figura 24, em que cada um é responsável por executar uma função específica nos dados meteorológicos capturados da *API* simulada de estação meteorológica.

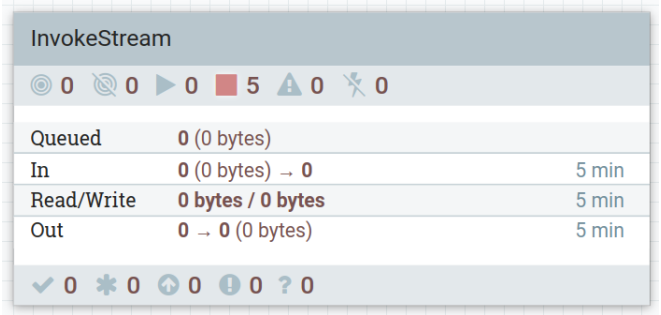

Figura 24 – Grupo de processadores para streaming de dados meteorológicos.

O primeiro processador do fluxo é o *InvokeHTTP*, configurado para realizar requisições *HTTP GET* à *URL http://localhost:8800/stream*. O objetivo é capturar os dados

meteorológicos transmitidos em fluxo contínuo pela *API*, sendo que cada requisição obtém uma linha distinta de dados. Este mecanismo de *streaming* assegura uma captura de dados em tempo real, facilitando o enriquecimento constante do banco de dados, com as novas informações meteorológicas inseridas.

O segundo processador executado, é o *ReplaceText*, que após a captura dos dados, este processador realiza a filtragem do texto recebido, removendo caracteres de aspas simples (') e aspas duplas ("), que podem interferir na interpretação correta dos dados subsequentemente. A utilização da expressão regular [\"'] no valor da propriedade *Search Value* deste processador, possibilita identificar e substituir ambos os tipos de aspas de forma eficiente.

Em seguida, tem-se o *ExtractText* como o terceiro processador do fluxo, que atua extraindo o texto já tratado vindo do fluxo e armazenando-o em um novo atributo chamado *dadoStream*. A expressão regular (?s)(.*) é empregada para capturar qualquer conjunto de caracteres, incluindo possíveis quebras de linha, assegurando que o conteúdo completo dos dados meteorológicos sejam preservados.

Após o armazenamento dos dados no atributo *dadoStream* na etapa anterior, é executado o quarto processador, o *UpdateAttribute*. Ele é empregado para realizar a transformação final dos dados antes da inserção no banco de dados. Ele coleta o conteúdo do atributo *dadoStream* e, por meio de funções do *Apache NiFi*, realiza a substituição de ponto e vírgula (;) por vírgula (,), além de adicionar aspas duplas no início e no final do texto, resultando na criação de um novo atributo denominado *message*. A expressão regular *${dadoStream:replace(';',','):prepend('"'):append('"')}* é responsável por essa transformação.

Por fim, o quinto e último processador, o *PutSQL*, é responsável por inserir os dados tratados no banco de dados. Este processador utiliza a conexão previamente estabelecida com o banco de dados através do *DBCPConnectionPool*. A instrução *SQL INSERT INTO api_stream (message) VALUES (${message});* é construída com a utilização do atributo do *Apache NiFi* definido no processador anterior, onde *${message}* é um *placeholder* que será substituído pelo valor contido no atributo *message*. Isso resulta em uma inserção segura e parametrizada dos dados meteorológicos na tabela de *staging* criada no banco de dados.

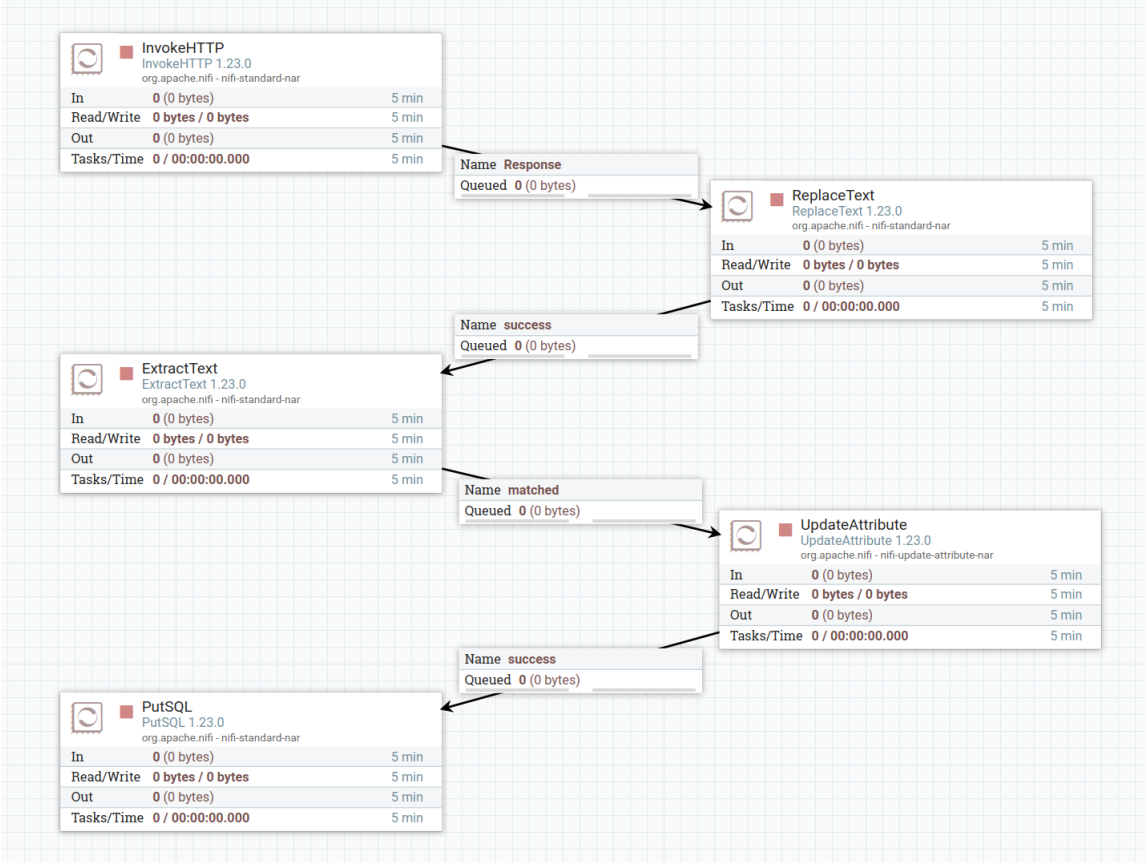

Figura 25 – Fluxo dos dados meteorológicos ingeridos em tempo real no Apache NiFi.

Após este processo, foi construído a tabela *dados_meteorologicos* no mesmo banco de dados *PostgresSQL*, para que ela armazenasse os dados resultantes da execução da *DAG insere_dados_meteorologicos*, desenvolvida no *Apache Airflow*, que será explicada mais adiante. A tabela *dados_meteorologicos*, foi criada utilizando as colunas e tipos de dados especificados na figura 26.

NOME DA COLUNA	TIPO DE DADOS
id	INT
estacao_id	INT
data	DATE
hora	TIME
ventovel10m	NUMERIC
ventodir10m	NUMERIC
raj10m	NUMERIC
rajdir10m	NUMERIC
ventovel2m	NUMERIC
prec	NUMERIC
precacum	NUMERIC
radsolglb	NUMERIC
radsolglbd	NUMERIC
etsz	TEXT
rso	TEXT
radparmed	NUMERIC
pressao	NUMERIC
tempmedar2m	NUMERIC
tempmaxar2m	NUMERIC
tempminar2m	NUMERIC
umidrelmed2m	NUMERIC
umidrelmax2m	NUMERIC
umidrelmin2m	NUMERIC
tempmedar1m	NUMERIC
tempmaxar1m	NUMERIC
tempminar1m	NUMERIC
umidrelmed1m	NUMERIC
umidrelmax1m	NUMERIC
umidrelmin1m	NUMERIC
tempmedrelva	NUMERIC
tempminrelva	NUMERIC
molfoliar1m_sec	NUMERIC
molfoliar1m_con	NUMERIC
molfoliar1m_mol	NUMERIC
molfoliar2m_sec	NUMERIC
molfoliar2m_con	NUMERIC
molfoliar2m_mol	NUMERIC
umidsolocm20cm	NUMERIC
umidsolonu20cm	NUMERIC
tempsologra2cm	NUMERIC
tempsologra5cm	NUMERIC
tempsologra10cm	NUMERIC
tempsologra20cm	NUMERIC
tempsologra40cm	NUMERIC
tempsologra100cm	NUMERIC
tempsolonu2cm	NUMERIC
tempsolonu5cm	NUMERIC
tempsolonu10cm	NUMERIC
tempsolonu20cm	NUMERIC
tempsolonu40cm	NUMERIC
tempsolonu100cm	NUMERIC
tempsolocm2cm	NUMERIC
tempsolocm5cm	NUMERIC
tempsolocm10cm	NUMERIC
tempsolocm20cm	NUMERIC
tempsolocm40cm	NUMERIC
tempsolocm100cm	NUMERIC
arquivo	TEXT
batmin	TEXT

Figura 26 – Nome e tipo de dados das colunas na tabela dados_meteorologicos.

A *DAG* do *Apache Airflow* em questão é destinada à automatização da etapa subsequente à ingestão de dados meteorológicos em tempo real. Ela é responsável por selecionar cada linha de dados da coluna *message* na tabela *api_stream*, processá-la e dividi-la em exatamente 59 valores em um dicionário *Python*, para inserir cada valor em sua respectiva coluna na tabela *dados_meteorologicos*. Esta *DAG* é configurada para ser executada a cada 15 minutos, já que os dados nas estações meteorológicas são gerados a cada 15 minutos, a partir de uma data de início especificada.

Para isso, a *DAG* executa a tarefa *atualiza_tabela_dados_meteorologicos*, ilustrada na figura 27, que inicia o primeiro passo estabelecendo uma conexão com o mesmo banco de dados *PostgreSQL*, utilizado nas etapas anteriores, aplicando a biblioteca *Psycopg2* do *Python* para realizar a conexão, passando as credenciais *dbname*, *user*, *password*, *host* e *port* para os parâmetros da função *connect* desta biblioteca.

Após a conexão bem-sucedida, a tarefa *atualiza_tabela_dados_meteorologicos* executa uma consulta *SQL GET* para buscar registros na tabela *api_stream* em que a coluna *em_dados_meteorologicos* é igual a *false*. Isso indica que os dados ainda não foram processados e inseridos na tabela de destino *dados_meteorologicos*.

Com os registros já selecionados, são realizados dois processamentos em cada um deles. O primeiro, separa cada registro em uma lista de valores usando a vírgula como delimitador, fazendo com que ao final deste processamento a lista armazene 59 itens. O segundo processamento, verifica se algum valor na lista é uma *string* vazia (”) e, se for, substitui pela *string* ’0’. Isso é relevante para os campos numéricos para garantir que não haverá valores nulos ou *strings* vazias, que poderiam causar erros ao inserir os dados no banco.

Por fim, com os dados devidamente processados e ajustados, a tarefa utiliza um comando *SQL INSERT* para adicionar cada valor da lista em sua respectiva coluna na tabela *dados_meteorologicos*. Após a execução da inserção, é executado o comando *SQL UPDATE* para atualizar a coluna *em_dados_meteorologicos* de cada registro selecionado da tabela *api_stream* para *true*, indicando que os dados já foram processados e inseridos na tabela final.

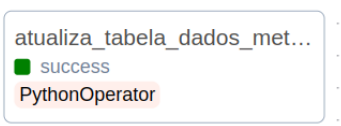

Figura 27 – Tarefa programada no Apache Airflow para inserir dados processados na tabela dados_meteorologicos.

A *DAG* do *Apache Airflow*, denominada *gera_mapa_diario*, que tem seu fluxo de

trabalho ilustrado na figura 28, desempenha um papel crítico na etapa de pós-processamento de dados meteorológicos. Esta *DAG* é projetada para automatizar e escalonar a geração de mapas diários de diversas variáveis atmosféricas, a partir de dados meteorológicos adquiridos em tempo real.

Inicialmente, a tarefa *cria_variaveis_airflow* é executada, ela configura as variáveis globais no contexto do *Apache Airflow*, especificamente o fator meteorológico, ou seja, a variável atmosférica, a data do mapa diário que o usuário deseja gerar, o modelo de variograma utilizado e o número de defasagens do algoritmo de *Krigagem*. Estas variáveis são armazenadas para uso subsequente durante a execução da *DAG*.

A tarefa subsequente, *cria_mapa_diario_meteorologico*, é responsável por realizar várias operações essenciais. Utilizando a biblioteca *Psycopg2*, ela estabelece uma conexão com o banco de dados *PostgreSQL*, permitindo a execução de consultas *SQL* para recuperar dados meteorológicos da tabela *dados_meteorologicos* e dados de localização da tabela *dados_estacoes*, que já possui dados pré-carregados no banco de dados referente as localizações das estações meteorológicas, com suas colunas ilustradas na tabela 2. Estes dados são posteriormente convertidos em *DataFrames* do *Pandas*, para facilitar a manipulação e análise mais adiante.

Tabela 2 – Nome e tipo de dados das colunas na tabela dados_estacoes

NOME DA COLUNA	**TIPO DE DADOS**
id	INT4
nome	VARCHAR(100)
latitude	FLOAT8
longitude	FLOAT8
altitude	FLOAT8

Com os dados recuperados, o procedimento continua no carregamento da geometria da região do Paraná por meio da biblioteca *GeoPandas* do *Python*. Esta geometria serve como base espacial para a interpolação e visualização dos resultados.

A próxima etapa envolve a preparação dos dados para interpolação espacial utilizando o método *OrdinaryKriging*, disponibilizado pela biblioteca *PyKrige*. As coordenadas de longitude e latitude são convertidas em *arrays* do *NumPy*, e os valores de temperatura são transformados em um formato numérico flutuante, adequado para processamento matemático.

Em seguida, é definido os limites da área de interesse, ou seja, a área do estado do Paraná e construído uma grade de interpolação, onde a função *execute* da classe *OrdinaryKriging* é invocada para gerar uma superfície de interpolação. O resultado é uma matriz de valores interpolados correspondentes aos pontos da grade espacial.

Para visualização dos resultados, emprega-se o *matplotlib*, onde uma figura é criada e um mapa de calor é gerado utilizando a função *contourf*. Uma barra de cores é adicionada

para representar a escala da variável atmosférica, e a geometria do Paraná é sobreposta para contextualização geográfica.

Por fim, a figura é salva em um diretório do sistema de arquivos local, proporcionando uma representação visual dos dados meteorológicos interpolados para a região especificada.

Esses processos possibilitam a geração de diversos mapas diários, provenientes de combinações de diversos dados disponíveis no banco de dados, oferecendo para o usuário a opção de definir a data, hora, variável atmosférica, modelo de variograma e o número de defasagens do algoritmo para gerar mapas meteorológicos diários conforme os parâmetros escolhidos.

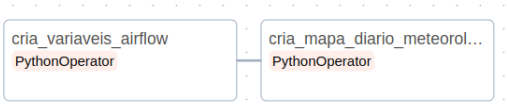

Figura 28 – Fluxo de trabalho da DAG gera_mapa_diario.

Após a execução de todos os processos descritos nesta sessão, passando como parâmetros para a execução da *DAG gera_mapa_diario*, a data *"2021-10-29"* para a variável *data_mapa_diario*, o modelo de variograma *"power"* para a variável modelo, a variável atmosférica *"tempmedar2m"* para a variável *fator_meteorologico* e o número 8 de defasagens do algoritmo na variável *nlags*, foi gerado como resultado o mapa diário ilustrado na figura 29.

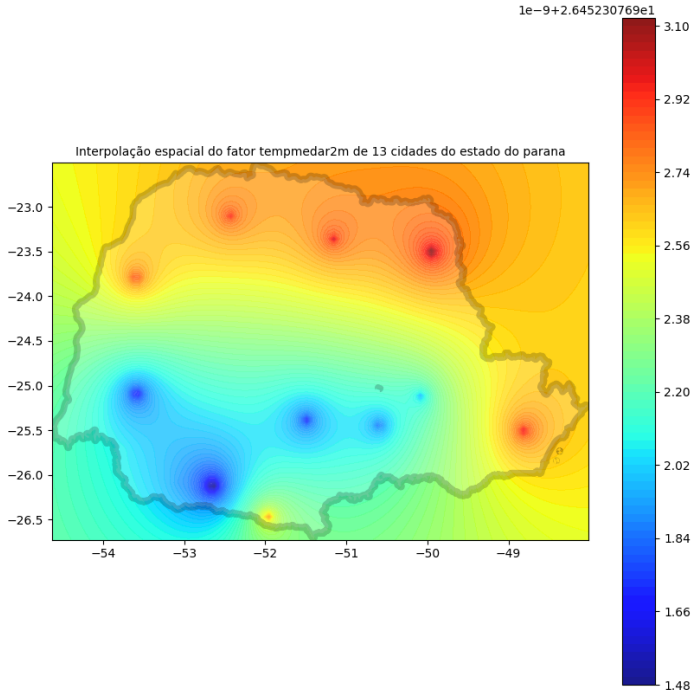

Figura 29 – Mapa diário do fator meteorológico de temperatura média tempmedar2m.

Para fim de demostração, foi construído também outro mapa diário, com a mesma data para a variável *data_mapa_diario*, o mesmo modelo de variograma *"power"* para a variável modelo, a variável atmosférica *"umidrelmed2m"* para a variável *fator_meteorologico* e o número 6 de defasagens do algoritmo na variável *nlags*, e com isso, foi obtido o mapa diário ilustrado na figura 30.

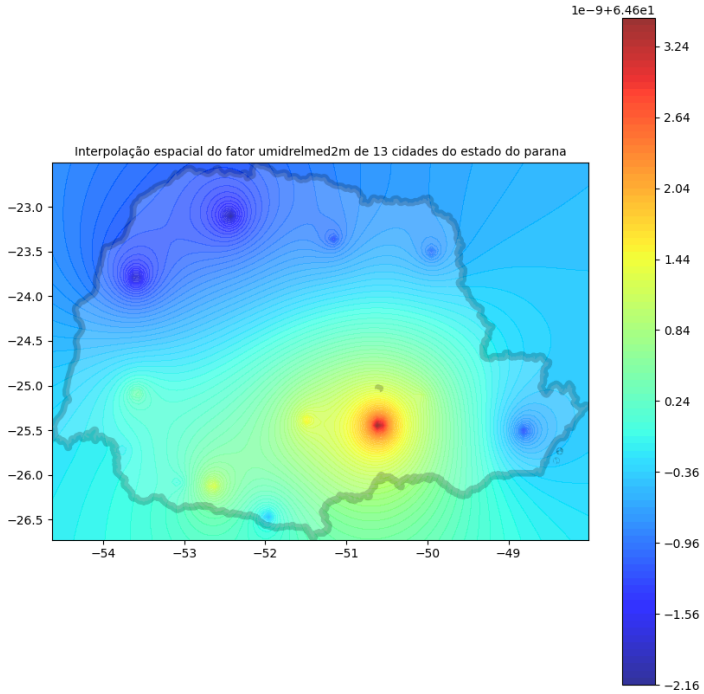

Figura 30 – Mapa diário do fator meteorológico de umidade média umidrelmed2m.

6 CONCLUSÃO

Arquitetura de gerência de dados para tarefas de aprendizado de máquina e análise de dados na agricultura, envolveu uma gama de situações, como as pesquisas, reflexões, escolhas, instalações, aprendizados, aplicações e validações sobre as opções tecnológicas e processos desenvolvidos ao longo do projeto. A arquitetura proposta se mostrou eficaz em lidar com a complexidade e a variedade de dados necessários para análises avançadas na área da agricultura, combinando ferramentas de ponta em um fluxo de trabalho coeso e robusto.

O projeto envolveu a seleção criteriosa de ferramentas e tecnologias adaptadas para a ingestão, processamento e análise de dados agrícolas. A integração do *Apache NiFi*, *Docker*, *Apache Airflow*, e ferramentas de geoprocessamento como *GeoPandas* e *PyKrige*, proporcionou uma solução robusta e escalável para o manejo de dados tanto em tempo real quanto em lote. Este sistema permitiu a automação de processos e a análise precisa de variáveis atmosféricas, essenciais para a tomada de decisão na agricultura, como, por exemplo, possibilitar a avaliação das condições atmosféricas na lavoura.

A escolha do *Apache NiFi* como ponto central para a ingestão de dados provou ser eficaz devido à sua capacidade de gerenciar dados em tempo real e em lote, fornecendo uma interface gráfica que simplifica a configuração de fluxos de dados, permitindo a aplicação de diversos pré-processamentos nos dados ingeridos, sem nenhuma linha de código. Isso facilitou a incorporação de diversas fontes de dados, garantindo a flexibilidade e a escalabilidade necessárias para o projeto.

O *Apache Airflow* desempenhou um papel crucial na orquestração dos processos de dados, com sua interface intuitiva e capacidade de agendar e monitorar pipelines complexos de processamento de dados. A funcionalidade de ajustes minuciosos das operações, juntamente com a visualização de dependências e progresso das tarefas, otimizou o fluxo de trabalho e aumentou a eficiência operacional nas tarefas de geoprocessamento realizadas no escopo do *Apache Airflow*.

A integração de bibliotecas de geoprocessamento como *GeoPandas* e *PyKrige* permitiu a manipulação e análise avançada de dados geoespaciais. A capacidade de realizar interpolações espaciais e criar visualizações geográficas detalhadas facilitou a compreensão das variáveis atmosféricas e seu impacto no ambiente agrícola.

A utilização do *Docker* para encapsular e gerenciar as dependências do projeto, assegurou a portabilidade e a consistência do ambiente de desenvolvimento e produção. Isso simplificou a implementação e a escalabilidade da arquitetura, permitindo adaptações rápidas e eficientes à medida que novas necessidades surgiram. Além de ser uma ferramenta

de integração extremamente escalável, o *Docker* demonstrou sua maior vantagem em ambientes de integração, sendo a redução drástica do consumo de memória e *CPU*, que a arquitetura obteve ao instalar e executar as ferramentas nos contêineres *Docker* em relação à instalação das mesmas diretamente no sistema operacional.

Com isso, foi observado que o sistema desenvolvido teve um impacto significativo no campo da agricultura, permitindo aos agricultores e pesquisadores, acessar análises detalhadas e previsões baseadas em dados confiáveis. Isso não apenas melhora a precisão das decisões agrícolas, mas também contribui para uma gestão mais responsável dos recursos naturais.

Para futuras iterações do projeto, considera-se a integração de novas fontes de dados e tecnologias emergentes em inteligência artificial e aprendizado de máquina, para enriquecer as análises. Além disso, a expansão da arquitetura para incluir ferramentas especializadas em engenharia de *features* e outros geoprocessamentos, que trabalham de modo mais aprofundado e com mais funcionalidades, permitiria a criação de *insights* ainda mais detalhados, potencializando a pesquisa e as práticas agrícolas.

Em suma, o desenvolvimento desta arquitetura de dados representa um avanço significativo na aplicação de novas tecnologias de dados para a agronomia, demonstrando o potencial de ferramentas de *big data* e geoprocessamento na transformação das práticas agrícolas modernas.

REFERÊNCIAS

[1] ECONOMIST, T. *The world's most valuable resource is no longer oil, but data*. 2017. Disponível em: https://www.economist.com/leaders/2017/05/06/the-worlds-most-valuable-resource-is-no-longer-oil-but-data. Acesso em: 21 jun 2023.

[2] RYO, M. Explainable artificial intelligence and interpretable machine learning for agricultural data analysis. *Artificial Intelligence in Agriculture*, v. 6, p. 257–265, 2022. ISSN 2589-7217. Disponível em: <https://www.sciencedirect.com/science/article/pii/S2589721722000216>.

[3] BENOS, L. et al. Machine learning in agriculture: A comprehensive updated review. *Sensors*, v. 21, n. 11, 2021. ISSN 1424-8220. Disponível em: <https://www.mdpi.com/1424-8220/21/11/3758>.

[4] LIAKOS, K. G. et al. Machine learning in agriculture: A review. *Sensors*, v. 18, n. 8, 2018. ISSN 1424-8220. Disponível em: <https://www.mdpi.com/1424-8220/18/8/2674>.

[5] SCIFORCE. *Machine Learning in Agriculture: Applications and Techniques*. 2019. Disponível em: https://medium.com/sciforce/machine-learning-in-agriculture-applications-and-techniques-6ab501f4d1b5. Acesso em: 05 jul 2023.

[6] YUAN, Y. et al. Advanced agricultural disease image recognition technologies: A review. *Information Processing in Agriculture*, v. 9, n. 1, p. 48–59, 2022. ISSN 2214-3173. Disponível em: <https://www.sciencedirect.com/science/article/pii/S2214317321000032>.

[7] KÜHN, D. *Pesquisa e Análise de Dados: problematizando o rural e a agricultura numa perspectiva científica (DERAD604)*. PLAGEDER, 2017. (Série Ensino, Aprendizagem e Tecnologias - UFRGS). ISBN 9788538603788. Disponível em: <https://books.google.com.br/books?id=YZU6DwAAQBAJ>.

[8] BREIMAN, L. Statistical Modeling: The Two Cultures (with comments and a rejoinder by the author). *Statistical Science*, Institute of Mathematical Statistics, v. 16, n. 3, p. 199 – 231, 2001. Disponível em: <https://doi.org/10.1214/ss/1009213726>.

[9] EMBRAPA. *Avanço da ciência de dados e big data, inteligência artificial, aprendizado de máquina e cooperativas de dados*. Disponível em: https://www.embrapa.br/visao-de-futuro/agrodigital/sinal-e-tendencia/avanco-da-ciencia-de-dados-e-big-data-inteligencia-artificial-aprendizado-de-maquina-e-cooperativas-de-dados. Acesso em: 05 jul 2023.

[10] MICROSOFT. *What is Azure Databricks?* 2023. Disponível em: https://learn.microsoft.com/pt-br/azure/databricks/introduction/. Acesso em: 09 ago 2023.

[11] ORACLE. *Build a secure OCI Data Integration environment with pre-built tasks from templates*. 2023. Disponível em: https://docs.oracle.com/en/solutions/oci-data-integration/. Acesso em: 09 ago 2023.

[12] AMAZON. *Amazon SageMaker Data Wrangler*. 2023. Disponível em: https://aws.amazon.com/pt/sagemaker/data-wrangler/. Acesso em: 09 ago 2023.

[13] HAT, R. *What is integration?* 2017. Disponível em: https://www.redhat.com/en/topics/integration/what-is-integration. Acesso em: 12 set 2023.

[14] BEURDEN, I. van. *The Meaning of Tool Integration*. 2016. Disponível em: https://www.exida.com/blog/the-meaning-of-tool-integration. Acesso em: 12 set 2023.

[15] SYDLE. *Systems Integration: Learn the Kinds, Challenges, and Importance*. 2022. Disponível em: https://www.sydle.com/blog/systems-integration-6140d39a84679b13bf127a93. Acesso em: 12 set 2023.

[16] SAP. *What is application integration?* 2023. Disponível em: https://www.sap.com/brazil/products/technology-platform/what-is-enterprise-integration/application-integration.html. Acesso em: 13 set 2023.

[17] EDUCATION, I. C. *What Is an Integration Platform? Do I Need One?* 2021. Disponível em: https://www.ibm.com/blog/integration-platform/. Acesso em: 13 set 2023.

[18] TENSORFLOW. *TFX is an end-to-end platform for deploying production ML pipelines*. 2023. Disponível em: https://www.tensorflow.org/tfx. Acesso em: 13 set 2023.

[19] MODI, A. N. et al. Tfx: A tensorflow-based production-scale machine learning platform. In: *KDD 2017*. [S.l.: s.n.], 2017.

[20] SAWANT, N.; SHAH, H. Big data ingestion and streaming patterns. In: _____. *Big Data Application Architecture Q & A: A Problem-Solution Approach*. Berkeley, CA: Apress, 2013. p. 29–42. ISBN 978-1-4302-6293-0. Disponível em: <https://doi.org/10.1007/978-1-4302-6293-0_3>.

[21] INTEGRAçãO de big data. In: 2013 IEEE 29ª Conferência Internacional sobre Engenharia de Dados (ICDE). [S.l.: s.n.].

[22] SHARMA, G.; TRIPATHI, V.; SRIVASTAVA, A. Recent trends in big data ingestion tools: A study. In: KUMAR, R. et al. (Ed.). *Research in Intelligent and Computing in Engineering*. Singapore: Springer Singapore, 2021. p. 873–881. ISBN 978-981-15-7527-3.

[23] MEEHAN, J. et al. Data ingestion for the connected world. In: *CIDR*. [S.l.: s.n.], 2017. v. 17, p. 8–11.

[24] CHAWLA, H.; KHATTAR, P. Data ingestion. In: _____. *Data Lake Analytics on Microsoft Azure: A Practitioner's Guide to Big Data Engineering*. Berkeley, CA: Apress, 2020. p. 43–85. ISBN 978-1-4842-6252-8. Disponível em: <https://doi.org/10.1007/978-1-4842-6252-8_4>.

[25] STITCH. *Data ingestion: the first step to a sound data strategy*. 2023. Disponível em: <https://www.stitchdata.com/resources/data-ingestion/>. Acesso em: 05.12.2023.

[26] MAVROGIORGOU, A. et al. Batch and streaming data ingestion towards creating holistic health records. *Emerging Science Journal*, v. 7, n. 2, p. 339–353, 2023.

[27] HOW, M. The ingestion architecture. In: _____. *The Modern Data Warehouse in Azure: Building with Speed and Agility on Microsoft's Cloud Platform*. Berkeley, CA: Apress, 2020. p. 105–132. ISBN 978-1-4842-5823-1. Disponível em: <https://doi.org/10.1007/978-1-4842-5823-1_4>.

[28] INGESTãO de Big Data e Arquitetura de Aprendizagem ao Longo da Vida. In: 2018 Conferência Internacional IEEE sobre Big Data (Big Data). [S.l.: s.n.].

[29] INGESTãO em tempo real de Big Data e aprendizado de máquina. In: 2018 IEEE Segunda Conferência Internacional sobre Mineração e Processamento de Fluxo de Dados (DSMP). [S.l.: s.n.].

[30] PICCARDI, M. L. S.; PALOMO, L. E. Del big data al fast data: enfoques modernos de streaming de datos para el procesamiento de datos masivos en tiempo real. *Difusiones*, v. 21, n. 21, p. 38–58, 2021.

[31] APACHE. *Apache Sqoop*. 2019. Disponível em: <https://sqoop.apache.org/>. Acesso em: 07.12.2023.

[32] TING, K.; CECHO, J. J. *Apache sqoop cookbook: Unlocking hadoop for your relational database*. [S.l.]: "O'Reilly Media, Inc.", 2013.

[33] VOHRA, D. Using apache sqoop. In: _____. *Pro Docker*. Berkeley, CA: Apress, 2016. p. 151–183. ISBN 978-1-4842-1830-3. Disponível em: <https://doi.org/10.1007/978-1-4842-1830-3_11>.

[34] ALWIDIAN, J. et al. Big data ingestion and preparation tools. *Modern Applied Science*, Canadian Center of Science and Education, v. 14, n. 9, p. 12–27, 2020.

[35] VOHRA, D. Apache flume. In: _____. *Practical Hadoop Ecosystem: A Definitive Guide to Hadoop-Related Frameworks and Tools*. Berkeley, CA: Apress, 2016. p. 287–300. ISBN 978-1-4842-2199-0. Disponível em: <https://doi.org/10.1007/978-1-4842-2199-0_6>.

[36] SRINIVASA, K. G.; M., S. G.; H., S. Apache flume. In: _____. *Network Data Analytics: A Hands-On Approach for Application Development*. Cham: Springer International Publishing, 2018. p. 95–107. ISBN 978-3-319-77800-6. Disponível em: <https://doi.org/10.1007/978-3-319-77800-6_6>.

[37] BIRJALI, M.; BENI-HSSANE, A.; ERRITALI, M. Analyzing social media through big data using infosphere biginsights and apache flume. *Procedia Computer Science*, v. 113, p. 280–285, 2017. ISSN 1877-0509. The 8th

International Conference on Emerging Ubiquitous Systems and Pervasive Networks (EUSPN 2017) / The 7th International Conference on Current and Future Trends of Information and Communication Technologies in Healthcare (ICTH-2017) / Affiliated Workshops. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1877050917317088>.

[38] APACHE. *Apache Kafka*. 2023. Disponível em: <https://kafka.apache.org/>. Acesso em: 07.12.2023.

[39] MĂTĂCUŢĂ, A.; POPA, C. Big data analytics: Analysis of features and performance of big data ingestion tools. *Informatica Economica*, v. 22, n. 2, 2018.

[40] VOHRA, D. Apache kafka. In: _____. *Practical Hadoop Ecosystem: A Definitive Guide to Hadoop-Related Frameworks and Tools*. Berkeley, CA: Apress, 2016. p. 339–347. ISBN 978-1-4842-2199-0. Disponível em: <https://doi.org/10.1007/978-1-4842-2199-0_9>.

[41] GARG, N. *Apache kafka*. [S.l.]: Packt Publishing Birmingham, UK, 2013.

[42] RAPTIS, T. P.; PASSARELLA, A. A survey on networked data streaming with apache kafka. *IEEE Access*, v. 11, p. 85333–85350, 2023.

[43] RAPTIS, T. P.; PASSARELLA, A. A survey on networked data streaming with apache kafka. *IEEE access*, IEEE, Piscataway, v. 11, p. 1–1, 2023. ISSN 2169-3536.

[44] DATAFLAIR. *Kafka Architecture and Its Fundamental Concepts*. 2023. Disponível em: <https://data-flair.training/blogs/kafka-architecture/>. Acesso em: 07.12.2023.

[45] ANKAM, V. *Big data analytics*. [S.l.]: Packt Publishing Ltd, 2016.

[46] RACKA, K. Apache nifi as a tool for stream processing of measurement data. *Nauki Ekonomiczne*, 2022.

[47] APACHE. *Apache NiFi Overview*. 2023. Disponível em: <https://nifi.apache.org/docs/nifi-docs/html/overview.html>. Acesso em: 09.12.2023.

[48] ALONSO, G.; MOHAN, C. Workflow management: the next generation of distributed processing tools. In: *Advanced Transaction Models and Architectures*. [S.l.]: Springer, 1997. p. 35–59.

[49] TALIA, D. Workflow systems for science: Concepts and tools. *International Scholarly Research Notices*, Hindawi, v. 2013, 2013.

[50] IBM. *What is a workflow?* 2023. Disponível em: <https://www.ibm.com/br-pt/topics/workflow>. Acesso em: 09.12.2023.

[51] HE, J. et al. Consensus mechanism design based on structured directed acyclic graphs. *Blockchain: Research and Applications*, v. 2, n. 1, p. 100011, 2021. ISSN 2096-7209. Disponível em: <https://www.sciencedirect.com/science/article/pii/S2096720921000063>.

[52] HARENSLAK, B. P.; RUITER, J. de. *Data Pipelines with Apache Airflow*. [S.l.]: Simon and Schuster, 2021.

[53] SINGH, P.; SINGH, P. Airflow. *Learn PySpark: Build Python-based Machine Learning and Deep Learning Models*, Springer, p. 67–84, 2019.

[54] FINNIGAN, L.; TONER, E. Building and maintaining metadata aggregation workflows using apache airflow. *Temple University Libraries*, 2021.

[55] SAINI, S.; BHATT, R. Airflow automator-streamlining workflows with dags. Jaypee University of Information Technology, Solan, HP, 2023.

[56] SHUBHA, B.; PRASAD, A. Airflow directed acyclic graph. *J Signal Process*, v. 5, n. 2.

[57] KEDRO. *Introduction to Kedro*. Disponível em: <https://docs.kedro.org/en/stable/introduction/index.html>.

[58] AMAZON. *Using Kedro pipelines to train Amazon SageMaker models*. 2023. Disponível em: <https://aws.amazon.com/pt/blogs/opensource/using-kedro-pipelines-to-train-amazon-sagemaker-models/>. Acesso em: 09.12.2023.

[59] WANG, J. *Encyclopedia of Data Science and Machine Learning*. IGI Global, 2023. (Advances in Data Mining and Database Management Series). ISBN 9781799892212. Disponível em: <https://books.google.com.br/books?id=I2-pEAAAQBAJ>.

[60] KEDRO. *Why Kedro?* 2023. Disponível em: <https://kedro.org/>. Acesso em: 09.12.2023.

[61] PERCHAIS, G. *Why We Switched Our Data Orchestration Service*. 2022. Disponível em: <https://engineering.atspotify.com/2022/03/why-we-switched-our-data-orchestration-service/>. Acesso em: 09.12.2023.

[62] LUIGI. *Introduction to Luigi*. 2023. Disponível em: <https://luigi.readthedocs.io/en/stable/>. Acesso em: 09.12.2023.

[63] DATAREVENUE. *What is Luigi?* 2023. Disponível em: <https://www.datarevenue.com/ml-tools/luigi>. Acesso em: 09.12.2023.

[64] LUIGI. *The Enterprise-Ready Micro Frontend Framework*. 2023. Disponível em: <https://luigi-project.io/>. Acesso em: 09.12.2023.

[65] IBM. *O que é machine learning?* 2023. Disponível em: https://www.ibm.com/br-pt/topics/machine-learning. Acesso em: 11 set 2023.

[66] GOOGLE. *What is Machine Learning?* 2023. Disponível em: https://cloud.google.com/learn/what-is-machine-learning. Acesso em: 11 set 2023.

[67] VICKERY, R. *A Beginner's Guide to End to End Machine Learning*. 2021. Disponível em: https://towardsdatascience.com/a-beginners-guide-to-end-to-end-machine-learning-a42949e15a47. Acesso em: 10 set 2023.

[68] MALIK, F. *End To End Guide For Machine Learning Project*. 2018. Disponível em: https://medium.com/fintechexplained/end-to-end-guide-for-machine-learning-project-146c288186dc. Acesso em: 10 set 2023.

[69] KAZIL, J.; JARMUL, K. *Data Wrangling with Python: Tips and Tools to Make Your Life Easier*. O'Reilly Media, 2016. ISBN 9781491956809. Disponível em: <https://books.google.com.br/books?id=XmeDCwAAQBAJ>.

[70] PRESS, G. *Cleaning Big Data: Most Time-Consuming, Least Enjoyable Data Science Task, Survey Says*. 2016. Disponível em: https://www.forbes.com/sites/gilpress/2016/03/23/data-preparation-most-time-consuming-least-enjoyable-data-science-task-survey-says/?sh=28d139436f63. Acesso em: 12 set 2023.

[71] REILLY, J. *End to End Machine Learning Workflow*. 2022. Disponível em: https://www.akkio.com/post/end-to-end-machine-learning-workflow. Acesso em: 12 set 2023.

[72] AMAZON. *What is Data Preparation?* 2023. Disponível em: https://aws.amazon.com/pt/what-is/data-preparation/. Acesso em: 12 set 2023.

[73] AMAZON. *What Is Feature Engineering?* 2023. Disponível em: <https://aws.amazon.com/what-is/feature-engineering/>. Acesso em: 15.11.2023.

[74] ENGENHARIA de recursos [desenvolvimento de software]. In: PROCEEDINGS Nono Workshop Internacional sobre Especificação e Design de Software. [S.l.: s.n.].

[75] Reid Turner, C. et al. A conceptual basis for feature engineering. *Journal of Systems and Software*, v. 49, n. 1, p. 3–15, 1999. ISSN 0164-1212. Disponível em: <https://www.sciencedirect.com/science/article/pii/S016412129900062X>.

[76] DARRAZãO, E. et al. Engenharia e avaliação de features para extração de informação em notas fiscais. In: *Anais da XVIII Escola Regional de Banco de Dados*. Porto Alegre, RS, Brasil: SBC, 2023. p. 80–89. ISSN 2595-413X. Disponível em: <https://sol.sbc.org.br/index.php/erbd/article/view/24349>.

[77] IBM. *What is a data architecture?* 2023. Disponível em: <https://www.ibm.com/br-pt/topics/data-architecture>. Acesso em: 03.12.2023.

[78] WANG, Y. et al. Isoblue hd: An open-source platform for collecting context-rich agricultural machinery datasets. *Sensors*, v. 20, n. 20, 2020. ISSN 1424-8220. Disponível em: <https://www.mdpi.com/1424-8220/20/20/5768>.

[79] KAKKAR, D. et al. The billion object platform (bop): um sistema para reduzir barreiras para suportar grandes fontes de dados espaço-temporais de streaming. *Software Livre e de Código Aberto para Geoespaciais (FOSS4G) Conference Proceedings*, v. 17, 2017. Disponível em: <https://scholarworks.umass.edu/foss4g/vol17/iss1/15>.

[80] MUHAMMAD, W. et al. Polly: A tool for rapid data integration and analysis in support of agricultural research and education. *Internet of Things*, v. 9, p. 100141, 2020. ISSN 2542-6605. Disponível em: <https://www.sciencedirect.com/science/article/pii/S254266051930246X>.

[81] BEUMER, L.; NIEMEYER, I. Developing a big data framework for processing sentinel-2 data in the context of nuclear safeguards evaluation of apache airflow, rasdaman and google earth engine. *ESARDA BULLETIN*, p. 75, 2022.

[82] DOCKER, I. Docker. *lınea].[Junio de 2017]. Disponible en: https://www. docker. com/what-docker*, 2020.

[83] ANDERSON, C. Docker [software engineering]. *Ieee Software*, IEEE, v. 32, n. 3, p. 102–c3, 2015.

[84] COMBE, T.; MARTIN, A.; PIETRO, R. D. To docker or not to docker: A security perspective. *IEEE Cloud Computing*, IEEE, v. 3, n. 5, p. 54–62, 2016.

[85] MIELL, I.; SAYERS, A. *Docker in practice*. [S.l.]: Simon and Schuster, 2019.

[86] NICKOLOFF, J.; KUENZLI, S. *Docker in action*. [S.l.]: Simon and Schuster, 2019.

[87] WNĘK, K.; BORYŁO, P. A data processing and distribution system based on apache nifi. In: MDPI. *Photonics*. [S.l.], 2023. v. 10, n. 2, p. 210.

[88] PANDYA, A. et al. Privacy preserving sentiment analysis on multiple edge data streams with apache nifi. In: IEEE. *2019 European Intelligence and Security Informatics Conference (EISIC)*. [S.l.], 2019. p. 130–133.

[89] VOHRA, D.; VOHRA, D. Apache flume. *Practical Hadoop Ecosystem: A Definitive Guide to Hadoop-Related Frameworks and Tools*, Springer, p. 287–300, 2016.

[90] SPARK, A. Apache spark. *Retrieved January*, v. 17, n. 1, p. 2018, 2018.

[91] WU, H.; SHANG, Z.; WOLTER, K. Performance prediction for the apache kafka messaging system. In: *2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*. [S.l.: s.n.], 2019. p. 154–161.

[92] KAMMACHI, H. J. et al. Accelerating data migration: A comparative study of apache spark and other leading tools. In: IEEE. *2023 IEEE 11th Region 10 Humanitarian Technology Conference (R10-HTC)*. [S.l.], 2023. p. 790–794.

[93] HOFFMAN, S. *Apache Flume: distributed log collection for Hadoop*. [S.l.]: Packt Publishing Ltd, 2013.

[94] SALLOUM, S. et al. Big data analytics on apache spark. *International Journal of Data Science and Analytics*, Springer, v. 1, p. 145–164, 2016.

[95] HIRAMAN, B. R.; M., C. V.; C., K. A. A study of apache kafka in big data stream processing. In: *2018 International Conference on Information , Communication, Engineering and Technology (ICICET)*. [S.l.: s.n.], 2018. p. 1–3.

[96] ISAH, H.; ZULKERNINE, F. A scalable and robust framework for data stream ingestion. In: IEEE. *2018 IEEE International Conference on Big Data (Big Data)*. [S.l.], 2018. p. 2900–2905.

[97] CHERRADI, M.; HADDADI, A. E. A scalable framework for data lakes ingestion. *Procedia Computer Science*, Elsevier, v. 215, p. 809–814, 2022.

[98] SRINIVASA, K. et al. Apache flume. *Network Data Analytics: A Hands-On Approach for Application Development*, Springer, p. 95–107, 2018.

[99] VYAS, S. et al. Literature review : A comparative study of real time streaming technologies and apache kafka. In: *2021 Fourth International Conference on Computational Intelligence and Communication Technologies (CCICT)*. [S.l.: s.n.], 2021. p. 146–153.

[100]ERDMANN, M. et al. Design and execution of make-like, distributed analyses based on spotify's pipelining package luigi. In: IOP PUBLISHING. *Journal of Physics: Conference Series*. [S.l.], 2017. v. 898, n. 7, p. 072047.

[101]HAINES, S. Workflow orchestration with apache airflow. In: *Modern Data Engineering with Apache Spark: A Hands-On Guide for Building Mission-Critical Streaming Applications*. [S.l.]: Springer, 2022. p. 255–295.

[102]PANHALKAR, S. Libflow: A platform to schedule and manage workflows using dags. 2019.

[103]PETRAITYTE, E. Exploring efficient workflow frameworks for data management. 2024.

[104]ELKINA, H.; ZAKI, T. Data ingestion frameworks for data lakes: An overview. *Journal for ReAttach Therapy and Developmental Diversities*, v. 6, n. 10s (2), p. 1700–1708, 2023.

[105]DEEPA, B.; RAMESH, K. Production level data pipeline environment for machine learning models. In: IEEE. *2021 7th International Conference on Advanced Computing and Communication Systems (ICACCS)*. [S.l.], 2021. v. 1, p. 404–407.

[106]JORDAHL, K. et al. geopandas/geopandas: v0. 5.0. *Zenodo*, 2021.

[107]JORDAHL, K. Geopandas documentation. *URL: http://sethc23. github. io/wiki/Python/geopandas. pdf-Download vom*, v. 26, p. 2022, 2016.

[108]LOPES, G. R.; DELBEM, A. C.; SOUSA, J. B. de. Introdução à análise de dados geoespaciais com python. *Sociedade Brasileira de Computação*, 2021.

[109]GILLIES, S. rasterio documentation. *MapBox, July*, v. 23, 2019.

[110]WASSER, L.; PALOMINO, J.; MCGLINCHY, J. Get started with gis in open source python workshop. 2019.

[111]GILLIES, S. The shapely user manual. *URL https://pypi. org/project/Shapely*, 2013.

[112]WESTRA, E. *Python Geospatial Analysis Essentials*. [S.l.]: Packt Publishing Ltd, 2015.

[113]MURPHY, B. S. Pykrige: development of a kriging toolkit for python. In: *AGU fall meeting abstracts*. [S.l.: s.n.], 2014. v. 2014, p. H51K–0753.

[114]MOLIŃSKI, S. Pyinterpolate: Spatial interpolation in python for point measurements and aggregated datasets. *Journal of Open Source Software*, v. 7, n. 70, p. 2869, 2022.

[115]MÄLICKE, M. Scikit-gstat: A scipy flavored geostatistical analysis toolbox written in python. In: *EGU General Assembly Conference Abstracts*. [S.l.: s.n.], 2020. p. 6678.