UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

GUILHERME VERAS CASTAGNARO CORREIA

COLETA E ANÁLISE DE DADOS EM UMA ARQUITETURA DE BIG DATA PARA CONTROLE DE ENDEMIAS EM CIDADES

SANTA HELENA

2023

GUILHERME VERAS CASTAGNARO CORREIA

COLETA E ANÁLISE DE DADOS EM UMA ARQUITETURA DE BIG DATA PARA CONTROLE DE ENDEMIAS EM CIDADES

Data Collection and Analysis in a Big Data Architecture for Controlling Endemic Diseases in Cities

Trabalho de Conclusão de Curso apresentado como requisito para obtenção do título de Bacharel em Ciência da Computação da Universidade Tecnológica Federal do Paraná (UTFPR).

Orientador: Prof. Dr. Thiago França Naves

SANTA HELENA

2023

GUILHERME VERAS CASTAGNARO CORREIA

COLETA E ANÁLISE DE DADOS EM UMA ARQUITETURA DE BIG DATA PARA CONTROLE DE ENDEMIAS EM CIDADES

Trabalho de Conclusão de Curso apresentado como requisito para obtenção do título de Bacharel em Ciência da Computação da Universidade Tecnológica Federal do Paraná (UTFPR).

Data de aprovação: 01 de Dezembro de 2023

Thiago França Naves
Doutorado em Ciência da Computação
Universidade Federal de Uberlândia

Giuvane Conti
Doutorado em Engenharia Agrícola
Universidade do Oeste do Paraná

Franck Carlos Velez Benito
Doutorado em Ciência da Computação
Universidade Federal do Paraná

SANTA HELENA

2023

AGRADECIMENTOS

Principalmente aos meus pais, Priscila e Carlos, por sempre genuinamente me apoiarem, aos meus amigos, fundamentais por elevarem a qualidade dos momentos de descanso.

Ao meu orientador, Thiago França Naves, por sempre me auxiliar durante todo o desenvolvimento do projeto.

Por fim, mas não menos importante, meus professores que sempre me ajudaram na jornada de aprendizado.

RESUMO

Com a grande necessidade da análise de dados, o uso de arquiteturas de *Big Data* cresceu exponencialmente. Uma abordagem eficaz, simples e barata de se aplicar esse tipo de arquitetura é utilizando técnicas *serverless*. Neste trabalho foi detalhada uma arquitetura de coleta e análise de dados utilizando as tecnologias e serviços da Amazon Web Services (AWS). Para apresentar a arquitetura foi feita a coleta de dados de endemias de Dengue em cidades. A arquitetura proposta foi implementada e testada, e os resultados obtidos mostram que a arquitetura proposta é eficaz e pode ser utilizada em diversos casos de uso. Foi comparado os resultados, tempo de execução e custo da arquitetura proposta com uma arquitetura tradicional, utilizando Hadoop, e feita uma comparação entre seus pontos positivos e negativos.

Palavras-chave: Big data; Dengue; Serverless; AWS.

ABSTRACT

With the great need for data analysis, the use of Big Data architectures has grown exponentially. An effective, simple and cheap approach to applying this type of architecture is using serverless techniques. This work detailed an architecture for data collection and analysis using Amazon Web Services (AWS) technologies and services. To present the architecture, data was collected on Dengue endemics in cities. The proposed architecture was implemented and tested, and the results obtained show that the proposed architecture is effective and can be used in different use cases. The results, execution time and cost of the proposed architecture were compared with a traditional architecture, using Hadoop, and a comparison was made between its positive and negative points.

Keywords: Big data; Dengue; Serverless; AWS.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Quando uma infecção afeta uma população de uma região geográfica ela é considerada uma endemia, porém quando essa mesma infecção possui uma alta incidência e for duradoura ela é chamada de hiperendêmica (DRUMOND *et al.*, 2020). A dengue, por exemplo, por ter uma grande velocidade de propagação entre a população é considerada uma das maiores e mais perigosas endemias do mundo. Junto da *zika* e *chikungunya*, a dengue forma uma trinca de infecções com uma das maiores taxas de mortalidade no Brasil, e com suas recorrências de surtos frequentes estas são classificadas como hiperendêmicas. Um dos principais desafios no combate dessas endemias é o difícil controle e prevenção das doenças, por não haver vacinas ou remédios para as mesmas atualmente (CARLOS; NOGUEIRA; MACHADO, 2017).

De acordo com Santos, Costa, Feitosa *et al.* (2022) entre os anos de 2018 e 2020 foram notificados 2.788.522 casos de dengue no Brasil, onde 76,6% foram curados e 1.620, ou 0,05%, levaram a morte do portador do vírus. Entres esses casos 23,4% dos indivíduos não tiveram os dados de evolução da infecção preenchidos e outras informações complementares em relação ao local de transmissão também não estavam atualizadas (SANTOS; COSTA; FEITOSA *et al.*, 2022). Essa ausência na coleta e gestão de dados é um fator alarmante e que dificulta mecanismos de prevenção e combate aos surtos dessas endemias. Os métodos de monitoramento e armazenamento de dados relativos às endemias são arcaicos e precários, tanto em instâncias federais como estaduais e municipais, fazendo com que análises e previsões com os dados sejam difíceis de se fazer (MAGALHÃES *et al.*, 2016).

Nos anos recentes com o avanço nas tecnologias de monitoramento e de *Big data* foram surgindo estudos com melhores resultados sobre a previsão e análise de dados sobre as endemias da dengue, que utilizam tecnologias de gestão dos dados e análises com inteligência artificial (SUNDRAM *et al.*, 2019). Tecnologias de *Big Data* são uma alternativa para a melhoria no monitoramento e previsão de endemias e outras doenças, segundo Sarma *et al.* (2020) é possível fazer a predição de quando e onde os próximos surtos de uma endemia ocorrerão. Para realizar essas tarefas é necessário o uso massivo de diversos tipos de dados, sendo desde o histórico de casos, dados climáticos, localização dos pacientes, entre diversos outros, que podem ser específicos de uma região ou mais genéricos abrangendo diversas áreas.

Aprimorar a coleta e gestão das informações é importante para que as análises através de inteligência artificial fiquem cada vez mais precisas em relação a previsões e padrões para enfrentamento dos agentes epidemiológicos (OUSSOUS *et al.*, 2018).

No Brasil existem algumas dificuldades na obtenção dos dados relativos a endemias da dengue, primeiro que a maioria estão disponíveis apenas em Interfaces de Programação de Aplicação (APIs) fechadas, ou seja, é preciso pagar para obter os dados. Segundo, os dados são fornecidos por entidades governamentais, mas estão em formatos de difícil leitura ou estão com dados ausentes e faltantes, o que compromete a capacidade de análise dos mesmos. Por fim, um dos maiores problemas é a falta de especificidade dos dados, onde muitas vezes não são informados aspectos como localização geográfica da infecção, análise laboratorial do tipo de ovo ou mosquito, ou mesmo dados coletados por agentes de zoonoses e patrulhas. Para resultados de *Big Data* e análises mais confiáveis os dados precisam estar disponíveis, com um bom volume e variedade e em formatos que facilitem o processo de coleta contínuo.

Além de obter e coletar os dados é necessário armazená-los e processá-los em um ambiente que garanta segurança e agilidade nessas tarefas. Para isso, são utilizados armazéns de dados chamados de *data lake* e *data warehouse*, que são construídos com tecnologias que fornecem ferramentas para o processo de coleta, transformação e organização dos dados. Essas tarefas impactam diretamente na qualidade e correlação dos dados, deixando-os no melhor arranjo possível, para seu uso em algoritmos de inteligência artificial com o objetivo de fazer a previsão e descoberta de padrões para enfrentamento das endemias.

Assim, o objetivo deste trabalho é propor um novo método de coleta de dados relativos a endemia da dengue diretamente com os órgãos municipais, para obtenção de maior volume e variedade de informações com gestão e processamento das mesmas em uma arquitetura de *Big Data*, construída através da conexão de diferentes tecnologias, fornecendo armazém de dados. Com a metodologia e arquitetura espera-se que seja possível obter novos dados e organizá-los de forma a aumentar sua qualidade para uso com algoritmos de inteligência artificial no monitoramento e combate às endemias da dengue.

1.1 Objetivo

Expõem-se a seguir os objetivos geral e específicos que se pretende atingir com o trabalho.

1.1.1 Geral

O objetivo deste trabalho é desenvolver uma metodologia de coleta de dados de endemias como a dengue junto a cidades, armazenar e processar estes dados utilizando uma arquitetura de *Big Data* a partir da junção de tecnologias, para que os mesmos possam ser utilizados por algoritmos de inteligência artificial na análise e previsões para enfrentamento de surtos endêmicos.

1.1.2 Específicos

Dentre os objetivos específicos, estão:

1. Fazer uma revisão bibliográfica dos trabalhos na área de *Big Data* aplicado à análise de endemias;
2. Analisar APIs e fontes de dados sobre dengue e dados meteorológicos usados para previsão de surtos da dengue;
3. Criar uma metodologia para coleta de dados junto a cidades de modo que a mesma consiga volume e variedade de informações;
4. Desenvolver uma arquitetura de *Big Data* com tecnologias de *data lake* e *data warehouse* para armazenamentos e processamento dos dados coletados;
5. Executar testes de monitoramento da endemia da dengue com previsões de surtos utilizando técnicas de inteligência artificial;
6. Analisar os resultados dos testes e das etapas de *Big Data* e qualidade da metodologia e dos dados obtidos.

1.2 Contribuições do Trabalho

A principal contribuição deste trabalho é a proposta de uma metodologia de coleta de dados relativos a endemia da dengue diretamente com cidades, bem como as

possibilidades de integrar novos dados obtidos junto a secretarias e agentes de saúde em uma arquitetura de *Big Data* para análise e processamento no enfrentamento de surtos de endemias. As demais contribuições estão listadas a seguir:

1. Criação de armazéns de dados para limpeza, armazenamento e processamento de dados vindo de fontes e formatos heterogêneos;
2. Junção e conexão de diferentes tecnologias em uma arquitetura de *Big Data*;
3. Validação e testes com novos dados de endemias da dengue e sua qualidade para melhoria no processo de monitoramento epidemiológico.

1.3 Justificativa

Com a atual forma de monitoramento de surtos da endemia da dengue não é possível alcançar resultados satisfatórios diante da realidade de cada cidade brasileira, pois a quantidade de dados coletados e disponibilizados não são suficientes para fazer análises precisas e relevantes para os incidentes. Porém, propor novas e mais eficientes formas de coleta e novos métodos de processamento de dados é caminho para uma melhoria na efetividade dos sistemas de monitoramentos (SOUZA SILVA *et al.*, 2018). Com os dados da dengue atualmente fornecidos pelos governos locais é de extrema dificuldade fazer um algoritmo que consiga fazer a previsão de surtos de dengue, como é possível ver pelos resultados em outros países (LEE; YANG; LIN, 2015). Para isso é necessário implementar novos fluxos de coleta e processamento dos dados da dengue junto aos responsáveis locais, como agentes epidemiológicos e governos municipais, para ter acesso às informações necessárias para gerar as previsões de surtos de dengue e outras endemias.

Segundo Roh, Heo e Whang (2019) e Nguyen *et al.* (2019) é necessário fazer o uso de tecnologias modernas e eficientes de *Big Data* para gerar uma arquitetura capaz de produzir bons resultados com uma grande quantidade de dados. Com isso, faz-se necessário aprimorar também o uso de tecnologias que lidam com a gestão de armazéns de dados de forma eficiente, provendo também formas de executar os diversos processamentos para aumento da qualidade dos dados armazenados.

A metodologia de coleta e arquitetura proposta neste trabalho vai viabilizar que outros pesquisadores e entidades consigam utilizar e aprimorar o formato de obtenção

de dados, conseguindo maior volume e variedade e um ambiente de *Big Data* para realizar a gestão e processamento destes. Com isso, espera-se que avanços sejam feitos nos sistemas de monitoramento e que novas endemias e doenças de outros locais ou países possam ser testadas também, ampliando de forma geral o uso e os conceitos de *Big Data* e suas tecnologias na saúde.

1.4 Delimitações do trabalho

Por ser uma área de amplas possibilidades e muitas tecnologias diferentes disponíveis, além de ser um trabalho em conjunto com os governos locais e limitados pelo acesso às informações internas da dengue, este trabalho se limita aos seguintes pontos:

1. Não serão explicados os passos de instalações de *softwares* utilizados;
2. Os dados coletados junto a cidades serão exibidos de acordo com as permissões concedidas pelas secretarias de saúde;

2 REVISÃO DA LITERATURA

A revisão da literatura foi construída para revisar conceitos fundamentais das áreas correlatas necessárias para a integração de técnicas relacionadas à coleta e análise de dados de endemias. Todos os conceitos comumente encontrados na literatura são revisados com detalhes e exemplos didáticos, e indo além disso, boa parte do capítulo possui sínteses destes conteúdos e conhecimentos elaborados pelos autores. O propósito é apresentar conhecimentos necessários para o uso das tecnologias necessárias, bem como os específicos que serão utilizados na integração destas e controle das etapas necessárias do *Big Data*.

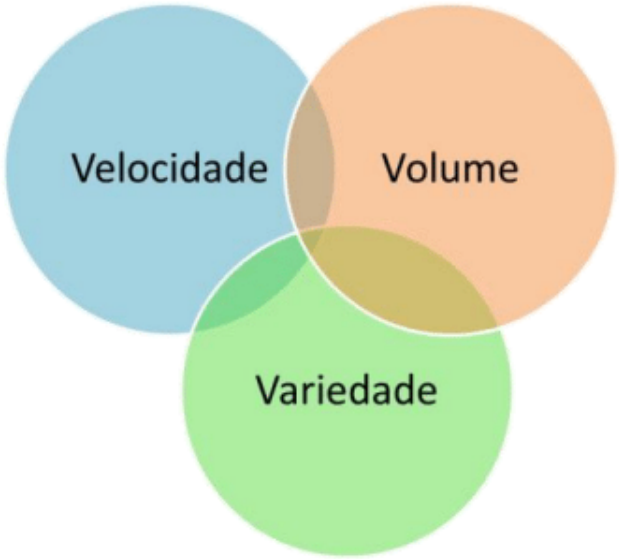
Portanto, este capítulo é parte fundamental no processo de construção do conhecimento gerado neste trabalho.

2.1 Big Data

Em um mundo onde a quantidade de informação disponível não para de aumentar e a necessidade do processamento desses dados acompanha o crescimento, foi necessário o desenvolvimento de ferramentas capazes de executar tais tarefas. Onde o processamento desses dados pode influenciar grandes escolhas dentro de grandes empresas, fazendo que uma grande quantidade monetária esteja em jogo, é necessário que os sistemas de análise de dados possuam resiliência e capacidade de alto processamento. A área onde essas ferramentas atuam é chamada de *Big Data* e, segundo Oussous *et al.* (2018), pode ser definida como a grande quantidade de conjunto de dados que incluem formatos heterogêneos como estruturados, não estruturados e semiestruturados.

- Volume: O volume pode ser definido como a grande quantidade de dados gerados por diferentes dispositivos usados pela população, como celulares, computadores, câmeras, veículos e muitos outros. Para ser considerado um sistema de *Big Data* é necessário que seja utilizado grandes quantidades de dados de fontes diferentes;
- Velocidade: Com a grande quantidade de pessoas utilizando tecnologias a velocidade que esses dados são gerados é imensa onde só no brasil ocorreram mais de 1 milhão e 500 mil transações financeiras via PIX no mês

Figura 1 – As 3 principais características do Big Data.



Fonte: Garcia (2022).

de março de 2022 (BRASIL, 2022), essa grande quantidade de dados que é gerada a cada dia também é uma das condições para um sistema ser considerado de *Big Data*;

- Variedade: Em um sistema de previsão de dados é necessário a utilização de diversos tipos de dados e fontes diferentes para conseguir uma acurácia aceitável. Então em um sistema de *Big Data* é necessário que diversos tipos de informações sejam utilizadas.

Durante o processo de Big Data diversos passos são executados, sendo os principais a Coleta de Dados, o Processamento dos Dados, a Mineração dos Dados, a Visualização dos Resultados e a integração dos resultados (GARCÍA *et al.*, 2016). A execução dessas etapas são contínuas e cíclicas, onde sempre há novos dados e esses passam por todo o processo.

- Coleta de Dados: A coleta de dados é o primeiro passo no processo de *Big Data* e consiste no processo de adquirir as informações de diversas fontes como por meio de APIs, *Web Scraping*, leitura de PDFs e outros. A principal ideia dessa etapa é coletar o máximo de dados possível;
- Processamento dos Dados: Com os dados coletados é necessário fazer a limpeza dos mesmos para que as informações inúteis e invalidadas seja removidas para não causarem problemas nos algoritmos a serem utilizados,

nessa etapa também é feita a separação e o relacionamento entre os dados para que seja mais fácil a leitura durante o próximos processos. Em alguns processos essa etapa também é utilizada para normalizar os dados e formatá-los nos formatos necessários para os algoritmos que serão usados;

- Mineração de Dados: Nessa etapa é onde são rodados os algoritmos de inteligência artificial e são obtidos os resultados brutos dos algoritmos, os dados utilizados nesse processo precisam estar completamente limpos e consistentes, sem nenhum erro para não influenciar o resultado. Com os dados prontos eles são armazenados na arquitetura para a análise e leitura. Os algoritmos a serem utilizados nessa etapa são diversos e devem ser escolhidos de acordo com a necessidade e problema a ser resolvido, também é recomendado utilizar diversos algoritmos diferentes para comparar o resultado entre eles;
- Visualização dos Resultados: A partir dos resultados encontrados pela mineração dos dados é possível gerar gráficos e demonstrar os dados de formas úteis para os cientistas de dados, para isso são utilizadas ferramentas de *Business Intelligence*;
- Integração dos Resultados: A etapa final do processo de *Big Data* é aplicar os resultados e análises encontradas no processo a realidade, então com os resultados gerados serão tomadas as decisões necessárias para afetar o mundo real e fazer a diferença nas áreas em que o processo for aplicado.

Para fazer uso do Big Data é necessário utilizar 4 tipos de software em sua arquitetura, sendo eles: Um sistema de arquivo, um sistema de banco de dados, um sistema de execução de *scripts* e um visualizador de dados (OUSSOUS *et al.*, 2018). Os softwares utilizados para *Big Data* podem fazer 1 ou mais dessas funcionalidades e algumas arquiteturas precisam de outras funções como um software de processamento distribuído. Alguns softwares que podem ser utilizados são o Apache *Hadoop*[1] junto ao Apache HDFS(HADOOP, 2022b), *PySpark*[2], *Amazon Athena*[3], *Amazon Redshift*[4] e diversos outros, sendo que cada um tem sua função e pode ser utilizado junto a outros

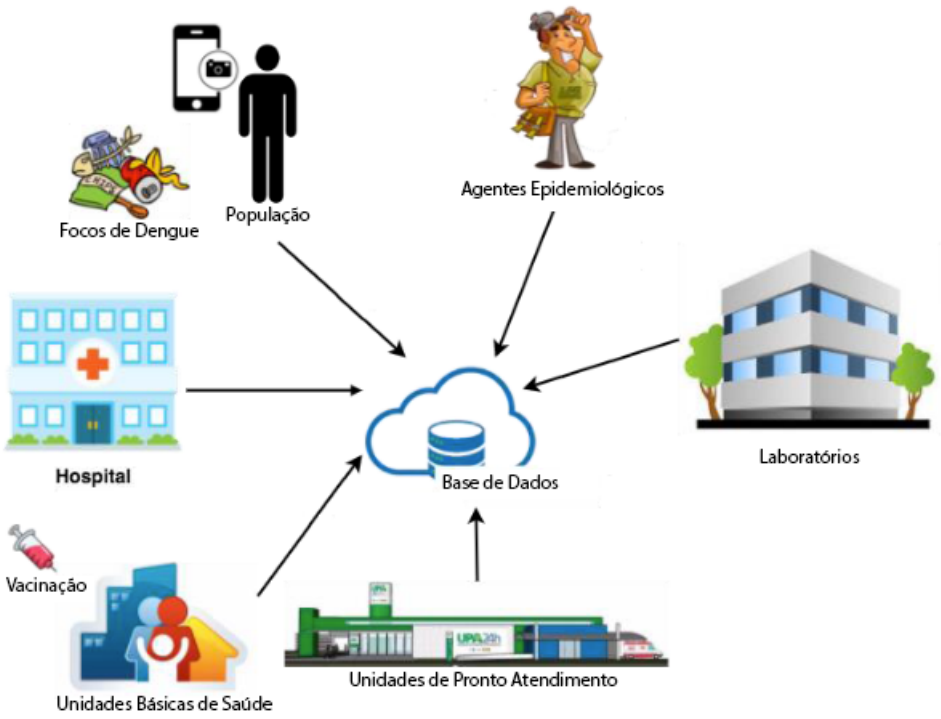
[1] https://hadoop.apache.org/.
[2] https://spark.apache.org/docs/latest/api/python/.
[3] https://aws.amazon.com/pt/athena/.
[4] https://aws.amazon.com/pt/redshift/.

softwares para fazer um arquitetura robusta e tolerante a falhas.

2.2 Coleta e Aquisição de Dados

Entre muitos dos desafios da área do aprendizado de máquina, um dos gargalos mais críticos é a coleta de dados. A maior parte do tempo gasto nos processos de *Big Data* está na etapa de preparação dos dados, que inclui a coleta, a limpeza, a análise e a visualização. Porém entre eles a tarefa que mais está se tornando a mais desafiadora é a falta de dados para treino dos algoritmos de aprendizado de máquina e com o aumento nas demandas a necessidade desses dados é cada vez maior. Com essas dificuldades é necessário a utilização de técnicas de coleta de dados escaláveis e precisas (GARCÍA *et al.*, 2016).

Figura 2 – Fontes de informações para previsão de Dengue.



Fonte: Tradução de Souza Silva *et al.* (2018).

Alguns tipos de dados que podem ser utilizados para melhorar a previsão de endemias são os dados meteorológicos, que são os dados relacionados ao histórico do clima da localização, como a temperatura média e a quantidade de chuva (SOUZA SILVA *et al.*, 2018). Porém para a previsão de endemias os dados mais importantes são os relacionados à infecção, como mostra a Figura 2 alguns exemplos são:

- Fotos de focos de dengue providos pela população;
- Dados hospitalares;
- Dados de vacinação e casos confirmados de hospitais;
- Dados de unidade das UPAs(Unidades de Pronto Atendimento);
- Informações coletadas pelos agentes epidemiológicos;
- Resultados de testes laboratoriais.

Ao utilizar conjuntos de dados já existentes é necessário fazer um trabalho de tratamento dos dados para melhorar os dados, fazer a limpeza e remoção de dados inúteis e re-rotulação dos dados (ROH; HEO; WHANG, 2019). Esses trabalhos são feitos para que o conjunto de dados se adequa melhor às necessidades dos algoritmos de aprendizado de máquina a serem utilizados.

A limpeza dos dados é feita para remover erros e valores que não fazem sentido, o principal objetivo dessas tarefas é reduzir os erros e variações que podem afetar nos resultados dos algoritmos. Para isso podem ser utilizados programas já existentes ou criar seus próprios algoritmos com foco no conjunto de dados que foi adquirido (ROH; HEO; WHANG, 2019).

A re-rotulação dos dados serve para que os dados vindo de fontes externas fiquem com os mesmos padrões entre si, tornando os conjuntos de dados mais consistentes e menos confusos para o processamento e análise dos dados.

O objetivo da aquisição de dados é encontrar dados que possam ser usados para o treinamento de algoritmos de aprendizado de máquina, para essa tarefa existem 3 principais caminhos que podem ser seguidos para adquirir os dados, sendo eles a descoberta de dados, o aumento de dados e a geração de dados. A descoberta de dados ocorre quando é necessário encontrar novos conjuntos de dados e esses estão disponíveis para serem coletados na internet ou em *data lakes*. O aumento de dados é uma extensão da descoberta de dados, onde se utiliza de conjuntos de dados já coletados e se adiciona novos dados nesses conjuntos com o objetivo de aumentá-lo. A geração de dados ocorre quando não é possível adquirir dados de fontes externas, sendo necessário que os dados sejam criados para serem utilizados nos algoritmos. Cada um desses caminhos possuem suas principais tarefas, que serão explicados a seguir. (TERRIZZANO *et al.*, 2015).

2.2.1 *Descoberta de Dados*

A descoberta de dados pode ser vista em 2 processos, primeiro os dados gerados devem ser indexados e publicados, porém esse passo nem sempre é feito com o objetivo de compartilhar o resultado, fazendo com que os próximos processos sejam mais trabalhosos. O segundo processo é a busca pelos conjuntos de dados, onde a busca será feita nesses dados publicados, nesse passo é necessário que seja verificado se os dados se enquadram nas necessidades da tarefa a ser cumprida.

2.2.2 *Aumento de Dados*

Quando a descoberta de dados não retorna a quantidade necessária de dados é possível fazer o aumento de dados para conseguir atingir uma quantidade adequada de dados. Esse processo pode ser feito de diversas formas, como por exemplo o aumento de entidades ou a integração de dados.

O aumento de entidades pode ser utilizado quando o conjunto de dados está incompleto e é necessário ser preenchido por meio da coleta de mais informações. Para cumprir essa função é ideal que se usem de processos automáticos, já que a quantidade de dados pode ser de um tamanho impossível de fazer o processo manualmente. Ferramentas como *Octopus* (CAFARELLA; HALEVY; KHOUSSAINOVA, 2009) e *InfoGather* (YAKOUT *et al.*, 2012) utilizam resultados de consultas na internet para preencher os dados faltantes.

Já a integração de dados acontece quando o conjunto de dados é pequeno e é necessário adicionar novos registros no conjunto. Stonebraker, Ilyas *et al.* (2018) e Doan, Halevy e Ives (2012) mostram as formas de se fazer a integração de dados de forma adequada e aceitas pelo mercado, como o uso de algoritmos especiais de aprendizado de máquina e o uso de humanos para fazer o preenchimento dos dados.

2.2.3 *Geração de Dados*

Quando não existem conjuntos de dados acessíveis para serem utilizados é necessário que os mesmos sejam criados para a execução dos processos de *Big Data*. Essa tarefa pode ser feita de forma manual ou automática, onde a forma manual

normalmente é feita por *Crowdsourcing* e as para gerar os dados de forma automática são utilizadas técnicas de geração de dados sintéticos (GARCÍA *et al.*, 2016).

Crowdsourcing é o método de obter informações utilizando um grupo de pessoas, normalmente um grupo grande e variado. Onde normalmente as pessoas respondem a pesquisas padronizadas e os dados são coletados para o uso dos algoritmos de aprendizado de máquina, mas por ser difícil o controle da origem e consistência dos dados é necessário fazer um controle de qualidade com os dados recebidos (ALLAHBAKHSH *et al.*, 2013). Os principais métodos de aquisição de dados via *crowdsourcing* são demonstrados por Li *et al.* (2016). Esse método pode ser utilizado para gerar conjuntos de dados inteiros ou só para adicionar a conjuntos já existentes ou fazer sua classificação.

A geração de dados por meios sintéticos é muito utilizada por ser de baixo custo e flexível às necessidades Patki, Wedge e Veeramachaneni (2016). Um dos métodos mais simples é utilizar uma distribuição probabilística e gerar dados de amostra a partir dessa distribuição utilizando ferramentas como *Scikit Learn* (PEDREGOSA *et al.*, 2011). Porém é recomendado utilizar técnicas mais avançadas ou mais específicas com a aplicação.

2.3 Data Lake

Um *Data Lake* é um repositório massivo de dados baseados em tecnologias de baixo custo com o objetivo de capturar, refinar, arquivar e explorar dados crus (FANG, 2015). Dentro de um *Data Lake* os dados podem estar em diversas formas e formatos, e não necessariamente possuem um uso ou serão utilizados. A principal ideia é fazer o armazenamento de todas as informações que possivelmente possam ser utilizadas no futuro.

Os dados armazenados dentro do *Data Lake* são armazenados em seu formato original sem nenhum processamento. Segundo Campbell (2015) um *Data Lake* possui três itens principais sendo eles:

- Todos os dados são carregados da fonte;
- Nenhum dado é descartado;
- O dado é armazenado em uma forma não transformada.

Dentro das diferentes formas de implementar um *Data Lake* o uso de tecnologias na nuvem de acordo com Lydia e Swarup (2015) o ecossistema Apache são os mais usados. Dentro de cada uma dessas tecnologias diversas ferramentas são utilizadas com diversos funções diferentes, porém as tecnologias possuem sempre uma base parecida, sendo ela: O uso de um sistema de arquivo para fazer o armazenamento dos dados, um motor para fazer a leitura e download e inserção dos dados no *Data Lake* e uma plataforma de gerenciamento que é responsável por controlar o sistema de arquivo e as tarefas do sistemas, também pode haver um gerenciador de *cluster* para fazer o *Data Lake* mais redundante e performático.

O uso de sistemas de *Data Lake* na nuvem possui grandes vantagens que estão fazendo com que ela esteja crescendo em uso exponencialmente, sendo as principais delas: A facilidade do uso, onde o gerenciamento é feito em grande parte automaticamente e sem necessidade de aprender uma grande quantidade de tecnologias; O custo relativo ao uso, já que as grandes plataformas cobram apenas pelo que for usado, então o custo do uso do sistema está ligado a quantidade de informação armazenado e ao processamento. Escalabilidade quase infinita, por rodar em grandes servidores a capacidade de expandir o sistema é imensa e fácil de se fazer (KHINE; WANG, 2018).

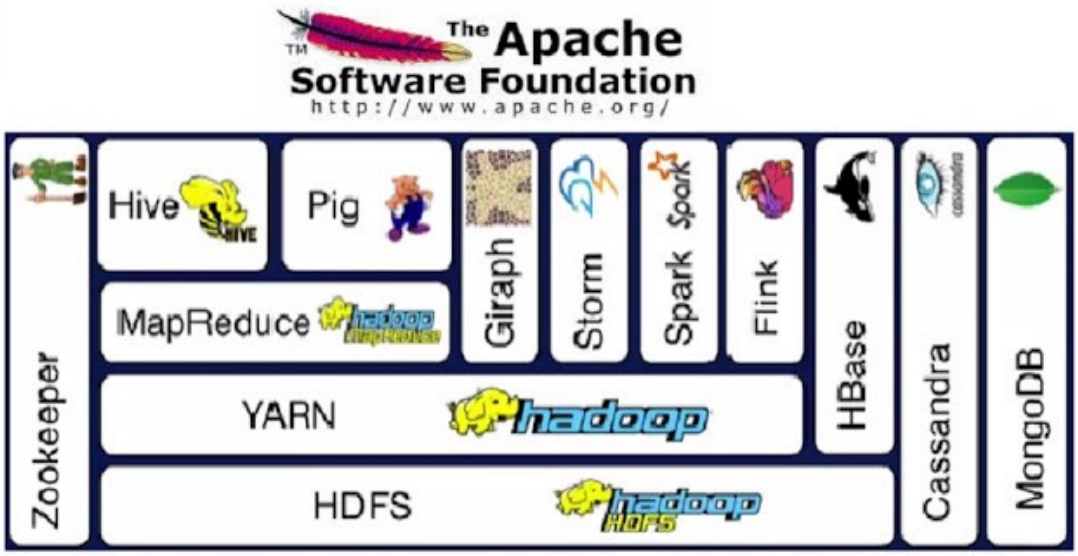
Porém as vezes é necessário um sistema mais customizado ou máquinas e servidores estão disponíveis para uso sem custo extra, para essas a melhor forma é utilizar tecnologias como o ecossistema Apache para *Data Lakes*, em que as principais ferramentas são: *Hadoop*, HDFS, *MapReduce*, *Yarn*, *Flume* e outros.

2.3.1 *Hadoop*

O *Hadoop* é uma das tecnologias mais usadas na área da Big Data. Ele possui a capacidade de processar grandes quantidades de dados rapidamente (LYDIA; SWARUP, 2015). Diferente de outras tecnologias, o *Hadoop* executa suas tarefas direto onde o dado está localizado. O *Hadoop* é feito de dois principais componentes, sendo eles o HDFS e o *MapReduce*.

Um dos pontos fortes do *Hadoop* é seu ecossistema, que como é possível ver na Figura 3 possui diversos softwares, cada um cumprindo uma função diferente e com integração entre eles.

Figura 3 – Ecossistema *Hadoop*.



Fonte: Apache (2022).

2.3.2 *HDFS*

O HDFS, ou *Hadoop Distributed File System*(Sistema de Arquivos Distribuído do *Hadoop*), é um dos dois pilares do *Hadoop*. Ele é um sistema de armazenamento de dados com suporte a clusterização, armazenamento confiável e usado em grandes escalas, podendo ser aplicado a sistemas com milhares de nós (IBM, 2022).

Segundo Hadoop (2022b) os principais objetivos do HDFS são:

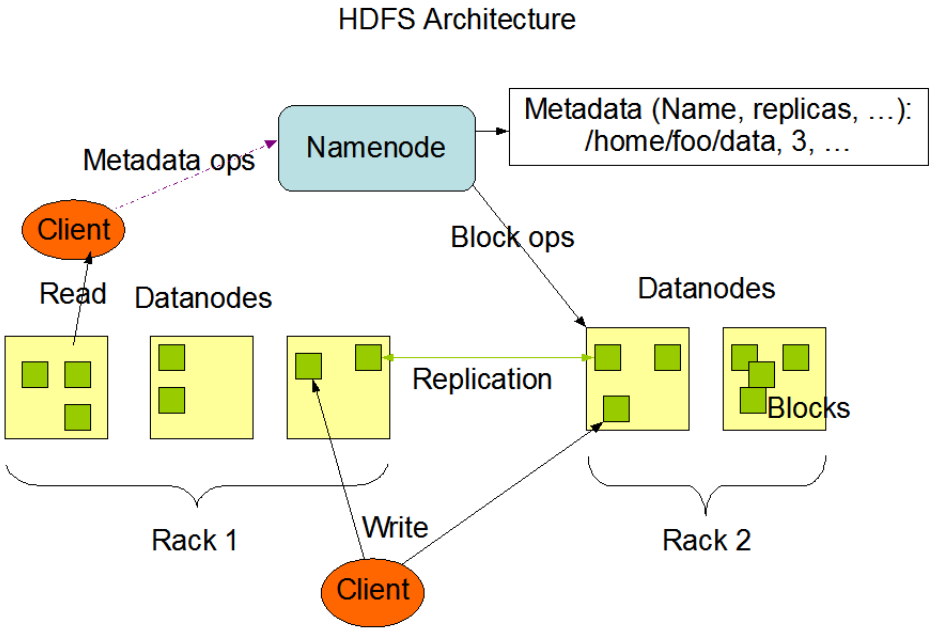
- **Falha de *Hardware*:** Por poder rodar em diversas máquinas, falhas de *Hardware* são esperadas e para contrapor este problema o HDFS consegue detectar falhas e fazer a recuperação dessas rapidamente de forma automatizada;
- **Acesso a Transmissão de Dados:** O HDFS não tem foco em baixa latência, mas sim em uma alta taxa de transferência de dados, para que as aplicações consigam acessar um número grande de dados simultaneamente;
- **Grandes Conjuntos de Dados:** O uso do HDFS é na área de *Big Data*, então se faz necessário ser capaz de conseguir lidar com grande número e tamanho de arquivos, onde tipicamente um arquivo pode ser de *gigabytes* a *terabytes de tamanho*;
- **Modelo de Coerência Simples:** Um arquivo no HDFS só precisa ser escrito uma vez, mas pode ser lido diversas vezes, dessa forma se simplifica o fluxo dos dados e possibilita uma alta taxa de transferência de dados;

- **"*Moving Computation is Cheaper than Moving Data*":** Ou mover computação é mais barato que mover dados, dessa forma é mais eficiente rodar as aplicações onde os dados estão armazenados do que levar os dados para onde a aplicação será rodada, então o HDFS fornece uma arquitetura onde as aplicações se movem para perto dos dados que serão acessados;
- **Portabilidade entre Diferentes *Hardware* e plataformas:** O HDFS tem como seu *design* ser capaz de ser portador de uma plataforma para outra de forma fácil, permitindo uma grande adoção do sistema;

O HDFS é focado para ser utilizado de forma paralelizada, onde é dividida em 2 partes: o *NameNode* e os *DataNodes*. Em uma arquitetura do HDFS há apenas um *NameNodes*, que roda no mestre, e diversos *DataNodes*, onde cada nó possui o seu. O *NameNode* é responsável por gerenciar o sistema de arquivos e regular o acesso a eles pelos clientes, já os *NameNodes* são responsáveis por executar as tarefas de armazenamento em cada nó do *cluster*, como ler, escrever e renomear arquivos (HADOOP, 2022b).

A Figura 4 demonstra um exemplo de uma arquitetura HDFS, onde cada bloco amarelo é um nó e cada bloco verde é um arquivo, como é possível ver os clientes conseguem acessar os nós diretamente para conseguir ler os dados, mas o *NameNode* que armazena os metadados de onde cada dado está armazenado.

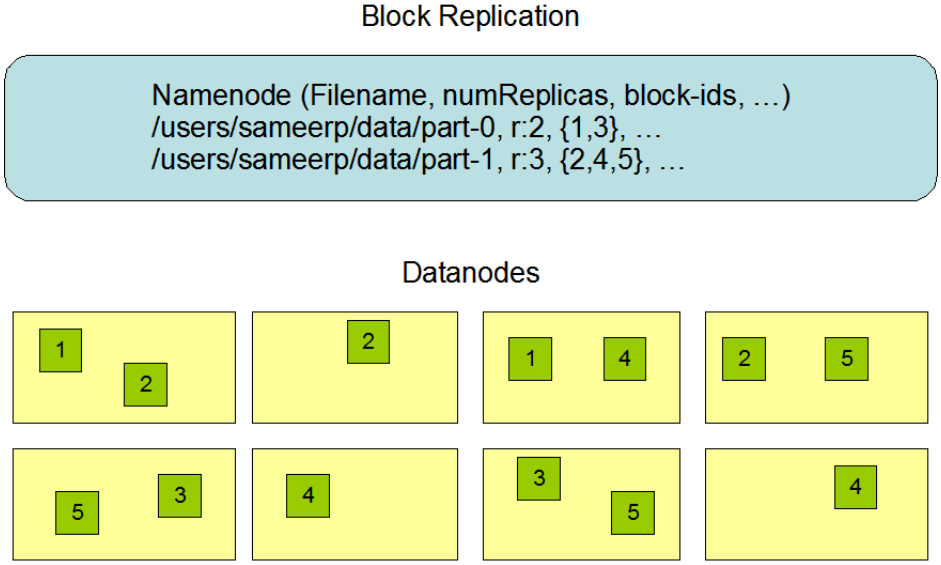
Figura 4 – Arquitetura do HDFS.



Fonte: Hadoop (2022b).

Para garantir a resiliência da arquitetura, o HDFS utiliza da replicação de dados. A replicação de dados é o método de duplicar as informações em diferentes nó, para caso um dos nós falhe o dado ainda estará disponível para acesso, a quantidade de réplicas é definida pela configuração do HDFS e pode ser alterada facilmente. A Figura 5 demonstra como os dados ficam armazenados com uma replicação de 3 onde cada arquivo é armazenado 3 vezes no *cluster*.

Figura 5 – Exemplo de replicação tripla de dados no HDFS.



Fonte: Hadoop (2022b).

2.3.3 *MapReduce*

O *MapReduce* é uma ferramenta de programação para Big Data que utiliza a arquitetura do HDFS, seu objetivo é simplificar o processamento de grandes quantidades de dados de forma eficiente e baixo custo, com suporte a paralelismo e uso em sistemas de *clusters* (LYDIA; SWARUP, 2015). Durante a execução do *MapReduce* os dados são divididos em pedaços que são processador paralelamente e esses 2 pedaços passam por duas funções sendo elas a parte do mapeamento(*Map*) e a parte da redução(*Reduce*).

1. A primeira parte, o mapeamento, divide os dados de entrada em partes consistentes de pares chave-valor (KHINE; WANG, 2018);
2. Já na segunda parte, a função de reduzir, utilizada de listas de pares para filtrar e organizar os dados de acordo com suas chaves (KHINE; WANG, 2018).

O *MapReduce* utiliza a arquitetura do HDFS para fazer o gerenciamento dos seus processos e distribuição dos seus processos de acordo com a localização dos dados entre os *DataNodes*. Para funcionar com um sistema de *clusters* o *MapReduce* é dividido em dois *deamons*, o primeiro é o *ResourceManager* que possui apenas uma instância no nó mestre do sistema distribuído, o segundo é o *NodeManager* que roda em cada nó do *cluster*. Com o *NodeManager* em cada nó é possível fazer processamento dos dados na localização dos mesmos, fazendo a performance e a largura de banda dos dados seja muito alta (HADOOP, 2022c).

2.3.4 *Yarn*

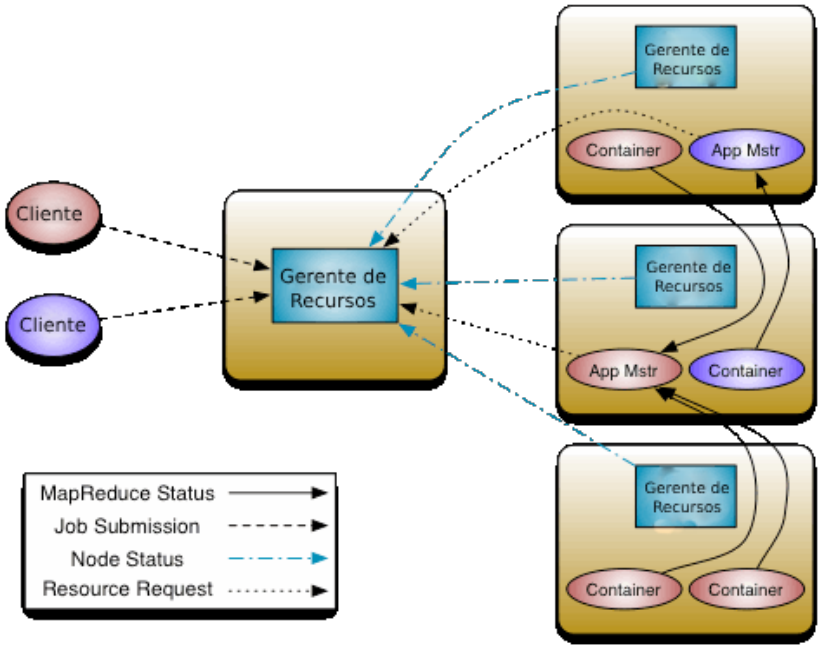
Yarn é uma ferramenta que provê formas de processamento com escalabilidade, paralelismo e gerenciamento de recursos avançados. Sua ideia fundamental é a divisão do processo de Gestor de Recurso que aloca os recursos através de nós e o Agendador de tarefas que faz o agendamento das tarefas a serem realizadas (WHITE, 2012). Com isso o *Yarn* consegue garantir um ciclo de vida das aplicações que estão sendo executadas nos *clusters*.

A ideia dessa divisão é ter um gerenciador de recursos, *ResourceManager* (RM), global e um mestre, *ApplicationMaster* (AM), por trabalho a ser executado. Com essa arquitetura o *Yarn* consegue fazer a execução de tarefas de modo distribuído, garantindo o ciclo de vida de tais tarefas, onde todas devem iniciar, fazer a execução e por fim finalizar com ou sem erro.

Para trabalhar em modo de *clusters* o *Yarn* executa o *ResourceManager* no nó mestre do *Hadoop* e *NodeManagers* nos nós que deverão executar os processamentos. Onde o *ResourceManager* é a autoridade que controla todos os trabalhos executados pelo *Yarn* e é responsável pelo monitoramento e pela exportação das informações do nó, como os dados de cpu, memória, disco e rede (HADOOP, 2022a).

A cada trabalho a ser executado o *Yarn* inicia um *ApplicationMaster*, que é um *framework* que faz a negociação entre o *ResourceManager* e os *NodeManager*, para conseguir acessar os dados necessário para a execução do trabalho e monitorar seu status, como é possível ver na Figura 6 (HADOOP, 2022a).

Figura 6 – Arquitetura do *Yarn*.



Fonte: Hadoop (2022a).

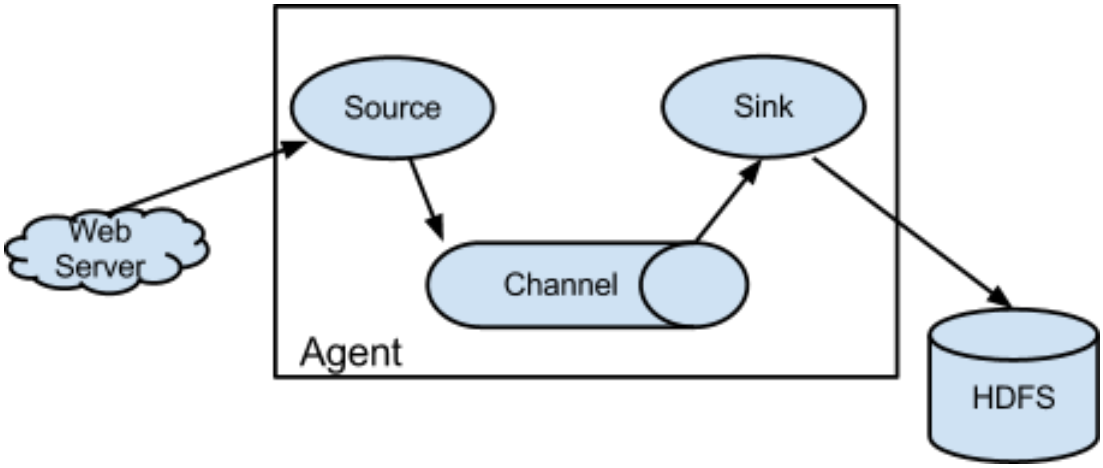
2.3.5 *Flume*

Para otimizar a transferência de dados de dentro para fora e de fora para dentro do HDFS é possível fazer o uso da tecnologia *Flume*, que é um serviço de coleta, agregação e transferência de grandes quantidades de dados (FLUME, 2022). Seu objetivo é cumprir suas tarefas de forma simples, confiável e eficientemente. A principal característica do *Flume* é sua tolerância a erros e mecanismos de *failover* e recuperação (LYDIA; SWARUP, 2015).

A arquitetura de funcionamento do *Flume* consistem em 3 partes, a fonte (*Source*), o canal(*Channel*) e o destino(*sink*) como mostra a Figura 7, onde os dados iniciam em um *web server* e entram no processo de transferência do *Flume* e ao finalizar o processo de transferência eles estarão dentro do HDFS, por exemplo. O *Flume* pode ser configurado para fazer a transferência dos dados sempre que novos dados surgem na fonte ou quando uma quantidade definida de dados estão prontos para serem encaminhados para o destino.

O principal objetivo de se utilizar o *Flume* é garantir a transferência dos dados para dentro do HDFS e fazer essa tarefa automaticamente sempre que necessário, garantindo que os objetos sejam criados dentro do HDFS sem erro.

Figura 7 – Arquitetura do *Flume*.



Fonte: Flume (2022).

2.4 *Data Warehouse* e Processamento de Dados

O *Data Warehouse* é um repositório de dados que é utilizado para gerenciar as informações a serem aplicadas nos algoritmos de aprendizado de máquina (AWS, 2023a). As principais características de um *Data Warehouse* são três de acordo com Miloslavskaya e Tolstoy (2016):

- Entrada rápida de dados não estruturados;
- Uso de aplicações de análise dinâmicas para consultas;
- Os dados ficam disponíveis assim que criados.

Tabela 1 – Comparação entre *Data Warehouse* e *Data Lake*.

Comparação	**Data Warehouse**	**Data Lake**
Dados	Estruturados e processados	Estruturados, semi estruturados e não estruturados
Processamento	Antes da escrita	Depois da escrita
Armazenamento	Caro e confiável	Baixo custo e alto volume
Performance	Otimizado para consultas	Otimizado para escrita e quantidade
Relevância dos dados	Alta importancia	Baixa importância

Fonte: Própria, (2022)

Como a tabela 1 mostra as principais diferenças entre um *Data Warehouse* e um *Data Lake* são o tipo de informação, seu uso e a forma em que eles são acessados e gravados. Onde no *Data Warehouse* os dados tem grande relevância aos problemas a serem resolvidos, os dados já estão com as classificações e ordens que serão utilizadas

nos algoritmos e a performance do *Data Warehouse* é focada na leitura e consulta de dados (OUSSOUS *et al.*, 2018).

Uma das principais características do *Data Warehouse* é que as informações armazenadas estão em seu estado final, reduzidas a apenas os dados necessários e organizadas para aumentar a performance das análises a serem cumpridas.

Os dados dentro de um *Data Warehouse* são agregados por seu tipo de informação, normalmente eles são armazenados em Banco de Dados relacionais onde é possível fazer relação de *features* entre diferentes conjuntos de dados (FERREIRA *et al.*, 2010).

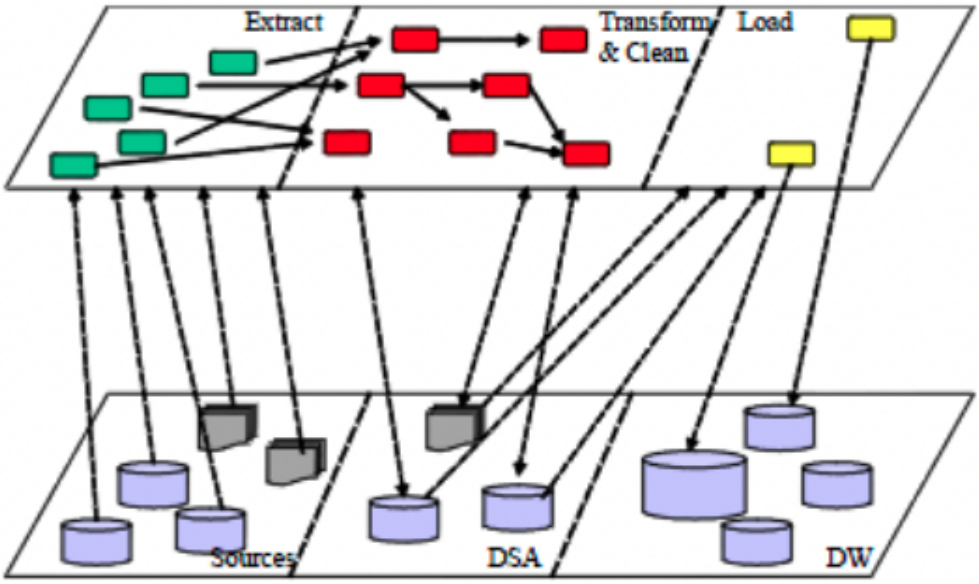
Com os dados dentro de um *Data Lake*(*Source*) é necessário fazer o processamento dos mesmo para que eles possam ser incluídos dentro do *Data Warehouse*(DW), este processamento tem como função transformar os dados que estão de forma bruta em dados preparados para o uso dos algoritmos. Esse processo é chamado de ETL (*Extract Transform Load*), ou em português extrair, transformar e carregar. Como o nome diz esse processo é quebrado em 3 partes, sendo a primeira a extração dos dados que estão dentro do *Data Lake*, o segundo passo o processamento dos dados, que pode ser desde a alteração de chave e colunas como cálculos e filtros, e por último o processo de carregar os dados dentro do *Data Warehouse*. Durante o processo de transformação, os dados podem ser armazenados em um área chamada DSA, *Data Staging Area*, para facilitar o processamento (FERREIRA *et al.*, 2010).

O destino final dos dados dentro do *Data Warehouse* é comumente um banco de dados relacional, onde será mais fácil dos algoritmos de aprendizado de máquina fazerem suas consultas. Para o processo de ETL diversos softwares podem ser utilizados, mas os mais utilizados são o *Spark*[5] e suas implementações, e no *Data Warehouse* é utilizado o Banco de Dados *Hive*[6] junto a arquitetura *Hadoop* (KHINE; WANG, 2018).

O *Spark* é uma tecnologia *open source* da Apache que faz o processamento de dados de forma distribuída, seu processamento é baseado em memória para aumentar a performance e é utilizado para os processos de Big Data e mais comumente para o processo de ETL. O *Spark* é usado em Java, porém existem desenvolvimentos para outras linguagens como o *PySpark* que aplica as funções do *Spark* para a linguagem

[5] https://spark.apache.org/.
[6] https://hive.apache.org/.

Figura 8 – Fluxo do processo de ETL.



Fonte: Ferreira *et al.* (2010).

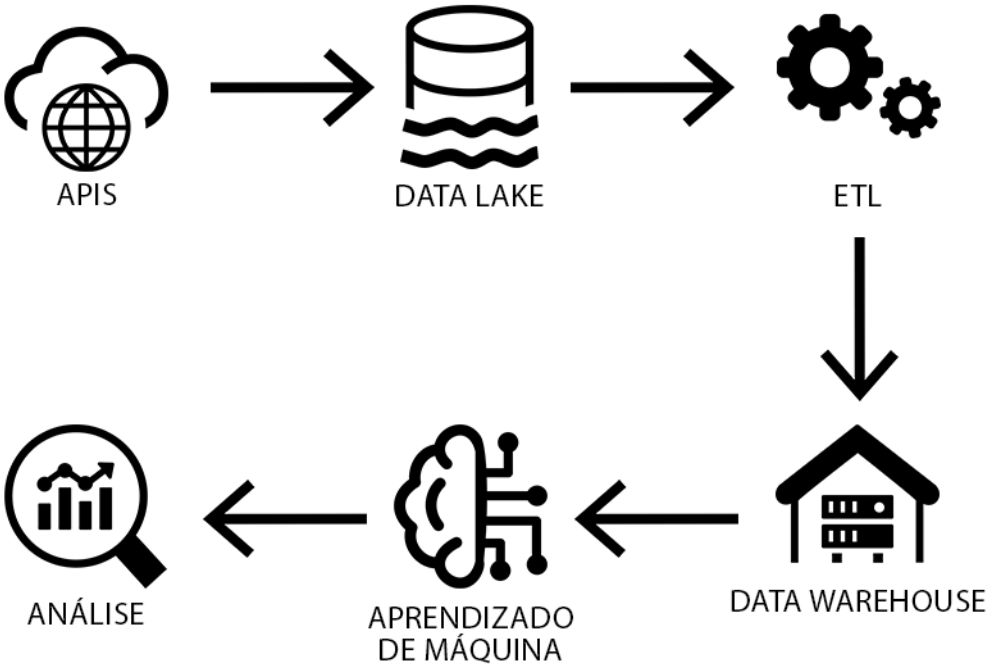
Python[7].

Para o uso de banco de dados em um ecossistema *Hadoop* é recomendado a utilização do *Hive*, que é um sistema de banco de dados que utiliza o HDFS para armazenar suas informações. Com ele é possível utilizar uma linguagem parecida com o SQL para fazer as consultas e inserções nas tabelas. Uma atenção que deve se tomar ao utilizar o *Hive* é que ele não deve ser utilizado como um banco de dados em tempo real, ou seja ele não possui performance com baixa latência, por precissar traduzir cada requisição em múltiplos trabalhos do *MapReduce*(TIAN *et al.*, 2015).

Como é possível ver na Figura 9 o fluxo do processo inteiro de Big Data começa na entrada das informações no *Data Lake*, depois o processo de ETL para salvar os dados no *Data Warehouse*. Com os dados no *Data warehouse* é utilizado algoritmos de aprendizado de máquina para gerar previsões e por último é feita a análise dos resultados. Porém o fluxo pode voltar para qualquer passo anterior conforme for necessário para melhorar os resultados das previsões e análises.

[7] https://www.python.org/.

Figura 9 – Fluxo do processo de Big Data.



Fonte: Própria, (2022)

2.5 AWS

A Amazon Web Services (AWS) é uma plataforma de serviços de computação em nuvem que oferece poder computacional, armazenamento de banco de dados, entrega de conteúdo e outras funcionalidades para ajudar as empresas a expandir e crescer (SBARSKI; KROONENBURG, 2017). A AWS é uma das maiores plataformas de computação em nuvem do mundo, com mais de 200 serviços completos de datacenters em todo o mundo (PISONI; CÂMARA, 2021).

A AWS fornece uma grande quantidade de serviços para o uso de *Big Data*, sendo os principais deles: *Amazon Athena*, *Amazon EMR*, *Amazon Kinesis*, *Amazon Redshift*, *Amazon QuickSight*, *Amazon S3*, *Amazon SageMaker*, *AWS Glue* e *AWS Lambda* (PEGADO, 2021). Cada um desses serviços possui uma função diferente e pode ser utilizado para diferentes tarefas, porém todos eles são integrados entre si e podem ser utilizados em conjunto para fazer uma arquitetura de *Big Data* completa.

Seu foco é fornecer serviços de computação em nuvem de baixo custo e alta performance, com isso a AWS consegue fornecer serviços de *Big Data* com alta escalabilidade e tolerância a falhas. O baixo custo de seus serviços é alcançado por

meio de um modelo de negócios de pagamento conforme o uso, onde o cliente paga apenas pelo que utilizar, sem a necessidade de pagar por um serviço que não será utilizado. A alta performance é alcançada por meio de uma grande quantidade de servidores e *datacenters* espalhados pelo mundo, onde o cliente pode escolher onde seus dados serão armazenados e processados, podendo escolher o local mais próximo de seus clientes para diminuir a latência e aumentar a velocidade de acesso aos dados (PELLE *et al.*, 2019).

Para fazer uma arquitetura mais customizada e de baixo custo de *Big Data* na AWS, pode ser utilizados os serviços *AWS Lambda*, *Amazon S3* e *AWS Step Functions*.

O *AWS Lambda* é um serviço de computação *serverless* que executa o código do cliente em resposta a eventos, gerenciando automaticamente os recursos de computação necessários para a execução do código. O *AWS Lambda* pode ser usado para executar código para processar dados, fazer atualizações de banco de dados, fazer transformações de dados, gerar relatórios, fazer análises em tempo real e outros (SBARSKI; KROONENBURG, 2017).

O *AWS Lambda* pode ser utilizado para fazer a execução de tarefas de forma automática e sem a necessidade de um servidor para o processamento, fazendo com que o custo de manutenção e processamento seja muito menor do que em outros serviços. Porém o *AWS Lambda* possui um limite de tempo de execução de 15 minutos, então ele não pode ser utilizado para processos que demoram mais do que isso. O controle de performance da função é feito pela quantidade de memória que é alocada para a execução da função, quanto mais memória alocada mais rápido será a execução da função (PELLE *et al.*, 2019).

A execução da função *lambda* pode ser feita de forma manual ou automática, onde a execução manual é feita por meio de um gatilho, que pode ser um clique em um botão ou uma chamada de API, e a execução automática é feita por meio de um evento, que pode ser um arquivo novo em um *bucket*, um acionamento causado por um outro serviço na AWS, um evento agendado ou por um serviço de orquestração como o *AWS Step Functions*.

O custo de execução do *AWS Lambda* é calculado por meio da quantidade de memória alocada para a execução da função e o tempo de execução da função. Outro fator que afeta o custo é a região de execução da função, ou seja, em qual servidor da AWS é executado o código. Na tabela 2 é possível ver os custos de execução do *AWS*

Tabela 2 – Custo de execução de funções *lambda* na região *us-east-1*.

Custo	Valor
Custo por GB por segundos	USD 0,0000166667
Custo por 1 milhão de execuções	USD 0,20

Fonte: (AWS, 2023c)

Lambda na região *us-east-1*, que é a região mais barata da AWS e fica localizada na Virgínia do Norte, Estados Unidos.

O *Amazon S3* é um serviço de armazenamento de objetos que oferece escalabilidade, disponibilidade de dados, segurança e performance. Ele pode ser utilizado para armazenar e proteger qualquer quantidade de dados para diversas aplicações (PISONI; CÂMARA, 2021).

O *Amazon S3* pode ser utilizado para armazenar os dados que serão utilizados no *Data Lake* e no *Data Warehouse*. O *Amazon S3* possui um custo de armazenamento de USD 0,023 por GB por mês, porém esse custo pode variar de acordo com a região de armazenamento e a quantidade de dados armazenados. Existe também um custo de leitura e escrita baseado na quantidade de dados trafegados nas requisições (AWS, 2023b).

No *Amazon S3* os arquivos são armazenados em *buckets*, que são como pastas, onde cada *bucket* pode ter um nome único e pode ser configurado para ser acessado de forma pública ou privada. Dentro de cada *bucket* os arquivos são armazenados com *keys*, que são como nomes e caminhos dos arquivos, onde cada *key* é única dentro do *bucket*.

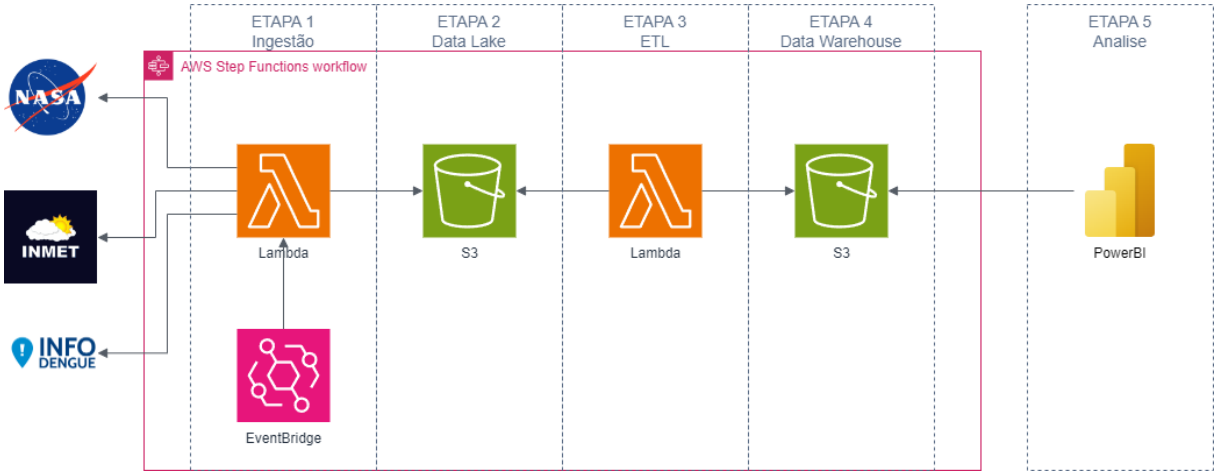
O *AWS Step Functions* é um serviço de orquestração de *workflows* que permite coordenar as funções de aplicativos como um fluxo de trabalho. O *AWS Step Functions* pode ser utilizado para coordenar a execução de diversas funções *lambda* e outros serviços da AWS (PISONI; CÂMARA, 2021).

A configuração do *Step Functions* é feita de forma visual, onde um diagrama é montado com os serviços e instruções de execuções. O custo do *AWS Step Functions* é baseado na quantidade de transições entre estados que são feitas, onde mil transições de estados custa USD 0,025. Porém deve se atentar que esse valor é apenas para a orquestração, cada serviço executado pelo *AWS Step Functions* terá seu próprio custo (PEGADO, 2021).

3 METODOLOGIA

Para o desenvolvimento deste trabalho, que possui o objetivo de criar e demonstrar uma nova metodologia de coleta e análise de dados de endemias em cidades, são necessárias a execução de etapas que são apresentadas na Figura 10.

Figura 10 – Representação gráfica da metodologia deste trabalho.



Fonte: Própria, (2022).

Como é possível ver na Figura 10 o fluxo de desenvolvimento é dividido em 5 etapas. A primeira etapa é o processo de coleta de dados, ou ingestão de dados, a partir do uso de APIs de fontes públicas, a segunda etapa é o armazenamento de dados em um *Data Lake*, a terceira etapa é o processamento dos dados salvos no *Data Lake* utilizando o ETL, a quarta etapa é o armazenamento dos dados processados em um *Data Warehouse* e por último a quinta etapa é a análise dos dados com vista a executar previsões de surtos. Cada etapa possui tecnologias e processos que serão explicados neste capítulo, que está dividido em 3 seções relativas às fases apresentadas na Figura 10. Foram utilizadas tecnologias da AWS para fazer a arquitetura desse fluxo.

3.1 Extração e Aquisição de Dados

A primeira etapa é composta pela coleta de dados que serão armazenados no *Data Lake* e pode ser considerada a mais importante do trabalho, já que erros nessa parte irão influenciar todo o resto do processo de *Big Data* e fazer as previsões com baixa ou nenhuma acurácia.

A coleta de dados tem como objetivo conseguir uma grande quantidade e variedade de dados. A princípio os tipos de dados a serem coletados são dados

climáticos, como por exemplo médias de temperaturas e de precipitações, dado do número de casos de dengue em uma determinada região específica e outros dados que os agentes epidemiológicos e governos locais consigam fornecer.

Os dados meteorológicos podem ser coletados por meio de APIs, como a API do INMET, o Instituto Nacional de Meteorologia, que fornece os dados históricos de estações meteorológicas espalhadas por todo Brasil gratuitamente ou a API da NASA onde é possível adquirir dados climáticos de satélites. É possível também utilizar outras APIs e fontes de informações para adquirir os dados climáticos do Brasil, porém a maiorias ou é paga ou não fornece dados completos ou do passado, como por exemplo a API *OpenWeather* que cobra para o acesso de dados históricos (DE VOS *et al.*, 2020), que são necessários para fazer as previsões de endemias.

Neste trabalho serão utilizados os dados climáticos das estações meteorológicas do INMET e dados originários de satélites fornecidos pela NASA, ambas essas fontes fornecem uma grande variedade de dados históricos e de diversas localidades de forma gratuita. Para acessar esses dados serão utilizadas as APIs do INMET (INMET, 2022) e da NASA (SCHULZE *et al.*, 2013) conforme suas documentações.

A API do INMET fornece dados de dois tipos de estações meteorológicas, as estações automáticas e as estações convencionais. A API fornece 2 formas de consultar seus dados, a primeira e mais complexa e o download de dados com a possibilidade de filtro e seleção de tipo de dados e periodicidade, sendo possível conseguir maior especificidade dos dados de acordo com o necessário. A segunda forma é fazer o download imediato dos dados históricos pelo link `https://portal.inmet.gov.br/dadoshistoricos`, onde é possível fazer o download dos dados completos por ano, nessa opção os dados estão com a periodicidade por hora, porém estão limitados aos dados apenas das estações automáticas.

Na tabela 3 é possível ver os tipos de dados que cada estações fornece, porém é importante se atentar que nem todas as estações fornecem todos dados mostrado na tabela 3, alguns dos dados podem estar vazios. Outro ponto de importante atenção é a falta de dados em alguns horários em algumas estações, fazendo que em algumas delas os dados não sejam contínuos.

É possível ver a distribuição dessas estações na região dos estados de São Paulo e do Paraná na Figura 11, como é possível ver há um grande quantidade de pluviômetros automáticos, que não estão disponíveis para consulta na API, já as

Tabela 3 – Comparação entre estações meteorológicas automáticas e convencionais.

Estações Convencionais	Nuvens altas
	Nuvens baixas
	Nuvens médias
	Nebulosidade
	Precipitação total
	Pressão atmosférica ao nível da estação
	Pressão atmosférica ao nível do mar
	Temperatura do ar - bulbo seco
	Temperatura do ar - bulbo úmido
	Temperatura do ponto de orvalho
	Umidade relativa do ar
	Vento
	Visibilidade
Estações Automáticas	Precipitação total
	Pressão atmosférica ao nível da estação
	Pressão atmosférica max.na hora ant.
	Pressão atmosférica min. na hora ant.
	Radiação global
	Temperatura do ar - bulbo seco
	Temperatura do ponto de orvalho
	Temperatura máxima na hora ant.
	Temperatura mínima na hora ant.
	Temperatura orvalho max. na hora ant.
	Temperatura orvalho min. na hora ant.
	Umidade rel. max. na hora ant.
	Umidade rel. min. na hora ant.
	Umidade relativa do ar
	Vento

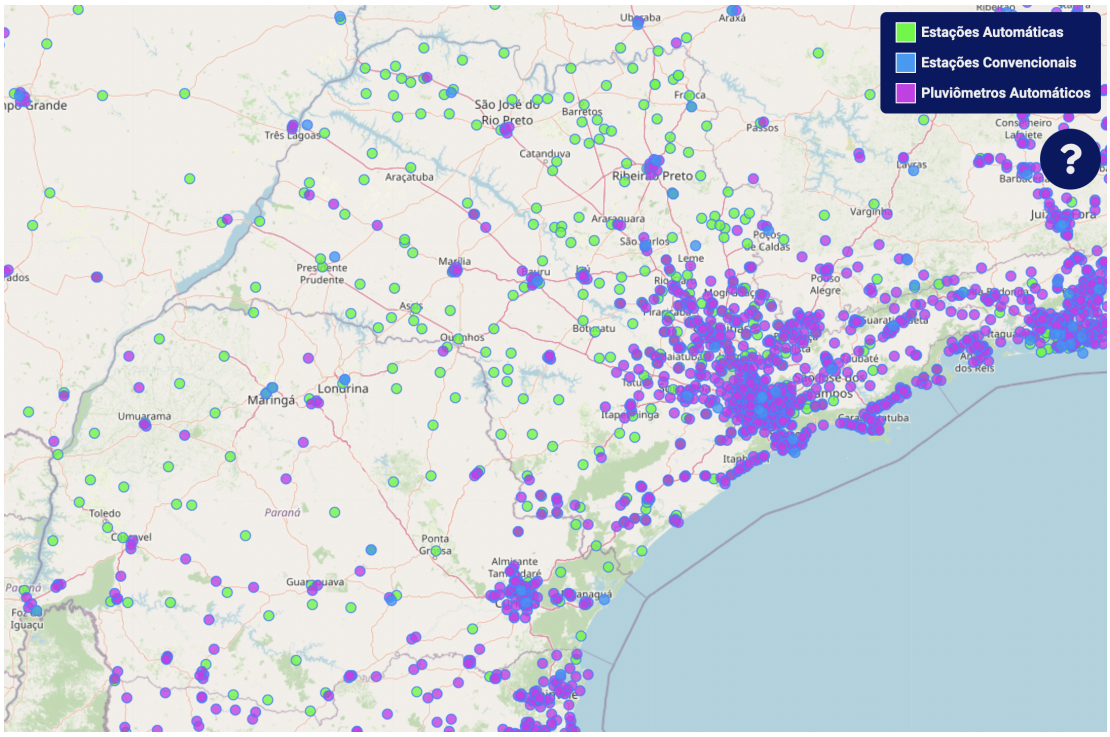
Fonte: Própria, (2022)

estações convencionais são escassas e estão restritas a grandes centros urbanos como a cidade de São Paulo e Curitiba. Já as estações automáticas estão espalhadas pelos estados assim cobrindo uma maior região, porém ainda sim não há dados específicos para cada cidade, fazendo com que seja necessário extrapolar os dados de uma cidade utilizando a estação mais próxima. Essas que normalmente trazem informações acuradas, mas em alguns casos a diferença de elevação e vegetação e rios podem alterar os valores em grandes quantidades.

A API da NASA fornece diversos tipos de dados a partir da coleta via satélites, nela é possível fazer a consulta via coordenadas geográficas e recuperar diversos parâmetros em formato mensal, semanal ou por hora. Para saber quais parâmetros estão disponíveis é necessário a utilização de uma outra API de suporte[1], onde é

[1] `https://power.larc.nasa.gov/api/pages/?urls.primaryName=Manager`.

Figura 11 – Mapa de estações meteorológicas nos estados de São Paulo e Paraná.



Fonte: INMET (2022).

possível consultar os parâmetros e ver suas definições.

Na tabela 4 é possível ver alguns exemplos de dados fornecidos pela API no formato de hora em hora, porém por ter uma imensa quantidade de parâmetros disponíveis para este trabalho serão utilizados apenas os dados da tabela 4 e como esses dados podem ser consultados a partir de coordenadas geográficas é possível ter uma maior precisão em relação a dados de lugares específicos, como cidades pequenas e locais isolados, mas por ser dados de satélites nem sempre os dados serão acurados quanto a uma estação meteorológica em terra (SANTOS; COSTA; ARAUJO *et al.*, 2020).

Tabela 4 – Exemplo de dados fornecidos pela API da NASA.

Código	Descrição
PRECTOTCORR	Precipitação total.
PS	Média de pressão superficial.
PW	Vapor de água na atmosfera.
QV10M	Proporção de vapor de água e massa de ar a 10 metros.
QV2M	Proporção de vapor de água e massa de ar a 2 metros.
WD2M	Direção do vento a 2 metros.
WS2M	Velocidade média do vento a 2 metros.

Fonte: Nasa (2023)

Os dados de caso de Dengue são mais complexos e mais difíceis de acessar, onde idealmente eles seriam coletados diretamente dos governos locais a partir de parcerias com os mesmos. Dessa forma os dados poderiam ser adquiridos com uma menor latência e maior confiança em sua acurácia, além de ser possível combinar um formato de troca de dados para facilitar na inserção dos dados no *Data Lake*.

Como forma alternativa de adquirir os dados de casos da dengue é possível utilizar os boletins da dengue fornecidos pelo governo do paraná, que estão disponíveis no link[2]. Estes boletins são arquivos no formato de PDF divididos em semanas epidemiológicas e estão disponíveis dos anos de 2008 a 2022. Esses arquivos são chamados de informes pela plataforma e cada relatório traz informações dos casos por estado, além de algumas análises.

Figura 12 – Boletim da dengue, Período Epidemiológico 2021/2022, Informe Anexo 40 - 31/05/2022, página 1

RS	MUNICÍPIOS	População	NOTIF	CASOS CONFIRMADOS							CASOS DESCAR.	CASOS INVES.	INCIDÊNCIA ACUMULADA	Sorotipo			
				CLASSIFICAÇÃO FINAL					LPI								
				DENGUE	DSA	DG	TOTAL	ÓBITOS	AUTOC.	IMPOR. OUTRAS UFs				DENV1	DENV2	DENV3	DENV4
1	Antonina	19011	9	0	0	0	0	0	0	0	7	2	0,00	X	-		-
1	Guaraqueçaba	7679	4	0	0	0	0	0	0	0	2	2	0,00	-	-		-
1	Guaratuba	36595	40	1	0	0	1	0	0	0	37	2	0,00	-	X		-
1	Matinhos	34207	12	1	0	0	1	0	0	0	8	3	0,00	X	X		-
1	Morretes	16366	6	0	0	0	0	0	0	0	4	2	0,00	-	X		-
1	Paranaguá	153666	1.345	160	1	0	161	0	155	0	1.177	7	100,87	-	X		X
1	Pontal do Paraná	26636	16	0	0	0	0	0	0	0	0	16	0,00	X	X		-
2	Adrianópolis	5983	0	0	0	0	0	0	0	0	0	0	0,00	X	-		-

Fonte: Paraná (2022).

Figura 13 – Boletim da dengue, Período Epidemiológico 2020/2019, Informe Técnico 03 - 24/08/2019, página 6

TABELA 3 – Número de casos confirmados autóctones, importados, total de confirmados e notificados de Dengue, Dengue Grave (DG), Dengue com Sinais de Alarme (DSA), óbitos e incidência (de autóctones) por 100.000 habitantes por município – Paraná – Semana Epidemiológica 31/2019 a 34/2019 *

RS	MUNICÍPIOS	POP	NOTIF	CLASSIFICAÇÃO FINAL				ÓBITOS	LPI		CASOS DESCAR.	CASOS INVES.	INC
				Dengue	DSA	DG	TOTAL		AUTOC.	IMPOR. OUTRAS UFs			
1	Paranaguá	153.666	70	0	0	0	0	0	0	0	0	70	0,00
2	Curitiba	1.917.185	38	0	0	0	0	0	0	0	33	5	0,00
2	Piraquara	111.052	2	0	0	0	0	0	0	0	2	0	0,00
2	São José dos Pinhais	317.476	1	1	0	0	1	0	0	1	0	0	0,00
3	Ponta Grossa	348.043	7	0	0	0	0	0	0	0	0	7	0,00
3	Sengés	19.267	1	0	0	0	0	0	0	0	1	0	0,00
4	Imbituva	32.179	1	0	0	0	0	0	0	0	0	1	0,00
4	Irati	60.357	1	0	0	0	0	0	0	0	0	1	0,00
7	Cheniezinho	19.343	2	0	0	0	0	0	0	0	1	1	0,00

Fonte: Paraná (2022).

Um dos principais problemas dessa forma de ingerir os dados é a dificuldade

[2] https://www.dengue.pr.gov.br/Pagina/Boletins-da-Dengue.

de fazer a leitura do PDF, onde é necessário fazer um *script* personalizado para fazer sua leitura. Outro problema específico da plataforma é que os informes não estão padronizados entre os anos, e em cada ano existe um padrão diferente como mostra a Figura 12 do ano de 2021/2022 e Figura 13 do anos de 2020/2019. Como é possível ver na Figura 12 os dados estão disponíveis a partir da página 1 e na Figura 13 os dados começam na página 6. Entre os dados do mesmo ano os informes possuem diferenças entre si, que podem ter sido causadas por uso de diferentes softwares para gerá-los, fazendo com que seja necessário o desenvolvimento de vários algoritmos para a leitura dos boletins.

Outra forma de acessar dados de casos de Dengue é pela API do infodengue `info.dengue.mat.br`. Os dados desta API também são fornecidos em semanas epidemiológicas e podem ser acessados de forma gratuita através de sua API. Nesta API diversos dados são fornecidos, como número de casos, população da localidade, dados climáticos e outros, porém por ser uma API que adquire seus dados por meios públicos e formulários de notificações, pode ser que seus dados não tenham uma acurácia muito grande. a tabela 5 mostra o dicionário de dados fornecidos pela API do infodengue.

Para acessar as APIs e fazer o *download* dos dados foi utilizado da arquitetura de nuvem da AWS, como mostra na figura 10, foi utilizado de funções *Lambda* para rodar códigos de extração e processamento de dados em *Python 3.9* e salvar os dados no *Data Lake*, para acionar as funções lambdas foi utilizado do serviço *CloudWatch* para execução recorrente do processo e *Step Functions* para a orquestração. Utilizando as funções lambdas os dados serão baixados das APIs e salvos em um *bucket* no S3, que será o *Data Lake* deste projeto.

O serviço *Step Functions* foi configurado para que primeiramente seja executada a função de ingestão de dados para o *Data Lake* e subsequentemente seja executada a função de ETL da etapa 3 do fluxo de desenvolvimento como é possível ver na figura 14. Para iniciar o processo deste serviço foi utilizado de regras do *CloudWatch* onde todos dia às 04:00 é processo é iniciado, fazendo com que todos os dias os dados sejam baixados e processados, mantendo toda a arquitetura com dados atualizados e recentes.

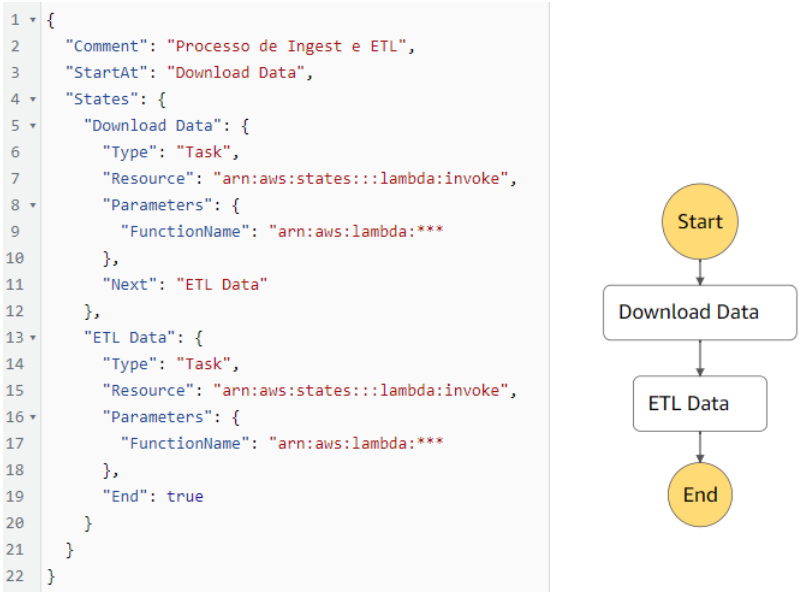
Utilizando como base as configurações de funções *Lambda* conforme a da figura 15, onde a arquitetura é arm, por ser mais eficiente e mais barata de se rodar

Tabela 5 – Dicionário de dados da API do infodengue.

Nome	Valor
data_ini_SE	Primeiro dia da semana epidemiológica
SE	Número estimado de casos por semana
cases_est	Intervalo de credibilidade de 95% do número de casos
casos	Número de casos notificados por semana
p_rt1	Probabilidade de (Rt> 1)
p_inc100k	Taxa de incidência estimada por 100.000
Localidade_id	Divisão sub municipal
nivel	Nível de alerta
id	Índice numérico
versao_modelo	Versão do modelo
Rt	Estimativa pontual do número reprodutivo de casos
pop	População estimada
tempmin	Média das temperaturas mínimas diárias ao longo da semana
tempmed	Média das temperaturas diárias ao longo da semana
tempmax	Média das temperaturas máximas diárias ao longo da semana
umidmin	Média da umidade relativa mínima diária do ar ao longo da semana
umidmed	Média da umidade relativa diária do ar ao longo da semana
umidmax	Média da umidade relativa máxima diária do ar ao longo da semana
receptivo	Receptividade climática
transmissao	Evidência de transmissão sustentada
nivel_inc	Incidência estimada abaixo do limiar pré-epidemia
notif_accum_year	Número acumulado de casos no ano

Fonte: InfodengueApi:online

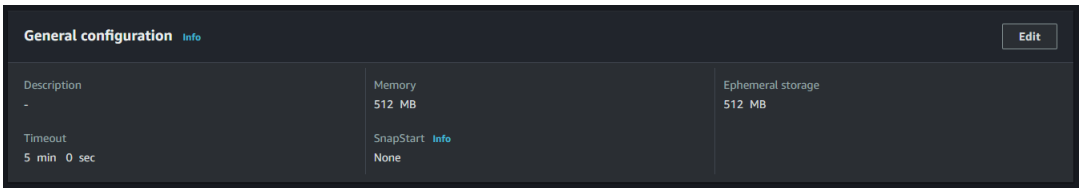
Figura 14 – Configuração da *Step Functions*



```
{
  "Comment": "Processo de Ingest e ETL",
  "StartAt": "Download Data",
  "States": {
    "Download Data": {
      "Type": "Task",
      "Resource": "arn:aws:states:::lambda:invoke",
      "Parameters": {
        "FunctionName": "arn:aws:lambda:***
      },
      "Next": "ETL Data"
    },
    "ETL Data": {
      "Type": "Task",
      "Resource": "arn:aws:states:::lambda:invoke",
      "Parameters": {
        "FunctionName": "arn:aws:lambda:***
      },
      "End": true
    }
  }
}
```


Fonte: Própria, (2023).

Figura 15 – Configuração das funções *Lambda* de ingestão de dados



Fonte: Própria, (2023).

e o *runtime* em *Python 3.9*, que é a versão mais recente do *Python* disponível atualmente na AWS. Foram criadas diversas funções, uma para cada origem de dados, sendo elas: Inmet, NASA e infodengue. Cada função também foi configurada com 512MB de memória e *timeout* de 300 segundos, conforme mostra a figura 15. Por haver dependência da resposta dos servidores da API e grande quantidade de dados requisitados por execução é necessário a configuração de *timeout* em 300 segundos para evitar o interrompimento da função e por lidar com uma grande quantidade de dados e o número de núcleos de processadores estar ligados a quantidade de memória. De acordo com nossos testes, 512MB de memória atingiu um balanço entre custo e performance para a execução de cada função.

Em cada função foram definidas algumas variáveis de ambiente para auxiliar na execução dos códigos, sendo elas *baseUrl*, *bucket*, *keyPrefix*, *startDate* como mostra a tabela 6, onde o *baseUrl* é a url base de acesso da API, o *bucket* é o nome do local onde os dados serão salvo no S3, o *keyPrefix* é a pasta dentro do *bucket* onde serão armazenados as informações e o *startDate* e a data inicio de busca de dados dentro

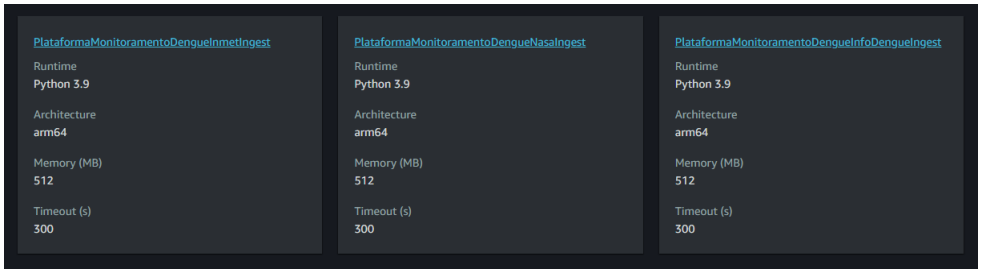
Tabela 6 – Variáveis de ambientes utilizadas para funções de ingestão de dados.

Nome	Valor
baseUrl	https://tempo.inmet.gov.br/estacao
bucket	plataforma-monitoramento-dengue
keyPrefix	dl/inmet/
startDate	2021-01-01

Fonte: Própria, (2023)

das APIs.

Figura 16 – Funções *Lambda* de ingestão de dados



Fonte: Própria, (2023).

Para cada uma das 3 fontes de dados utilizadas, foi criada uma função *Lambda* como mostra a figura 16. Cada função possui o mesmo código base, mas com alterações para se adequar às necessidades de cada API. O código possui a capacidade de baixar todos os dados a partir da data informada nas variáveis de ambiente, caso necessário, para manter a completude.

Como é possível ver na Figura 17 o código é dividido em 2 funções, a principal sendo a função *"lambda_handler"*, na linha 24, que é chamada quando a função lambda for executada e a função *"saveDataFromMonth"*, na linha 9, que faz o download dos dados de cada mês. O acesso aos dados é feito usando o pacote *urllib3* por ser um pacote já incluso dentro do *runtime Python* da função lambda, este pacote faz a interface com api de forma fácil e simples (CHANDRA; VARANASI, 2015). Com os dados baixados é feito os ajustes necessários para transformá-los em formato CSV e o mesmo é salvo dentro do *bucket* do S3 conforme configurado nas variáveis de ambiente. Já a função *"lambda_handler"* verificar de quais meses os dados devem ser coletados para ter a completude dos dados a partir da data definida na variável de ambiente, para isso é feita a consulta no *bucket* para ver quais meses já estão salvos, chamando a função *"saveDataFromMonth"* para cada mês que está pendente e o mês atual.

Figura 17 – Código da função *Lambda* de ingestão de dados

```python
import boto3
import os
import urllib3
import json
from datetime import date, timedelta

s3 = boto3.client('s3')

def saveDataFromMonth(date, station):
    end = date + timedelta(days=31)
    http = urllib3.PoolManager()
    print(os.environ['baseUrl']+'/'+str(date.year)+'-'+str(date.month).zfill(2)+'-01/'+str(end.year)+'-'+str(end.month).zfill(2)+'-01/'+station)
    resp = http.request("GET", os.environ['baseUrl']+'/'+str(date.year)+'-'+str(date.month).zfill(2)+'-01/'+str(end.year)+'-'+str(end.month).zfill(2)+'-01/'+station)
    print(resp.data.decode('utf8'))
    data = json.loads(resp.data.decode('utf8'))
    txt = [",".join(data[0].keys())]
    for d in data:
        dDate = d['DT_MEDICAO'].split('-')
        if str(date.month).zfill(2)==dDate[1]:
            txt.append(",".join(str(x) for x in d.values()))
    if len(txt)<2:
        return
    s3.put_object(Bucket=os.environ['bucket'], Key=os.environ['keyPrefix']+station+'/'+str(date.year)+str(date.month).zfill(2)+'.csv', Body="\n".join(txt))


def lambda_handler(event, context):
    startDate = date.fromisoformat(os.environ['startDate'])
    today = date.today()
    yesterday = today - timedelta(days=1)

    stationsList = s3.list_objects_v2(Bucket=os.environ['bucket'], Prefix=os.environ['keyPrefix'])['Contents']
    stations = list(filter(lambda a: a!='', list(dict.fromkeys([s['Key'].split('/')[2] for s in stationsList]))))
    for station in stations:
        months = list(filter(lambda a: a!='', [s['Key'].split('/')[3].split('.')[0] if len(s['Key'].split('/'))==4 and s['Key'].split('/')[2]==station else '' for s in stationsList]))
        for year in range(startDate.year, today.year+1):
            for month in range(startDate.month if year==startDate.year else 1, today.month if year==today.year else 13):
                if str(year)+str(month).zfill(2) not in months:
                    saveDataFromMonth(date(year, month, 1), station)
        saveDataFromMonth(yesterday, station)
```

Fonte: Própria, (2023).

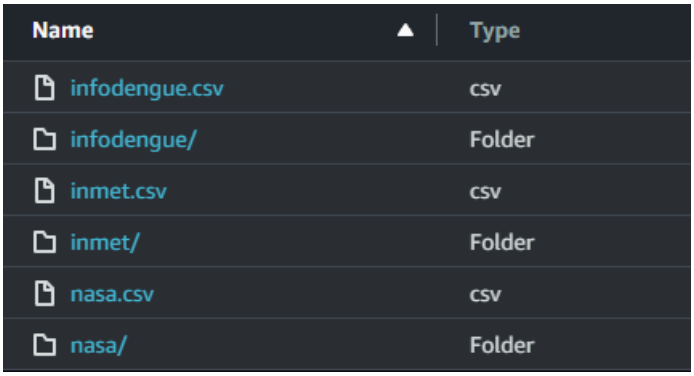
3.2 *Data Lake*

Com os dados ingeridos pelas funções *Lambda*, é necessário fazer o armazenamento dos mesmos no *Data Lake*, para isso foi utilizado o serviço de armazenamento de arquivos da AWS, o S3. Esses arquivos foram colocados em um pasta configurada dentro do *bucket* no S3, dentro dessa pasta os tipos de dados devem ser organizados por suas fontes e seus tipos. Essa organização é importante para que nos próximos processos seja possível fazer a identificação dos tipos de dados e origens.

Para armazenamento dos dados no S3 foi criado um *bucket* privado chamado “plataforma-monitoramento-dengue”, nele foi criada uma pasta para o armazenamento dos dados do *Data Lake* chamada "dl", nesta pasta foram armazenados os dados de cada origem de dado, como é possível ver na Figura 18 cada origem possui sua pasta onde dentro delas é armazenado os dados de cada localidade coletada e um arquivo csv com informações referentes os dados armazenados em sua pasta.

A figura 19 mostra um exemplo de arquivo csv dos dados do INMET, onde cada linha representa uma localidade, nela há a informação do nome da pasta onde os

Figura 18 – Pasta "dl"no *bucket* do S3



Fonte: Própria, (2023).

dados dessa localidade serão armazenados e o nome do local. Por exemplo: a pasta "A846"representa os dados da cidade de Foz do Iguaçu.

Figura 19 – Exemplo de arquivo CSV de fonte de dados

```
1  folder,name
2  A846,FOZ DO IGUAÇU
3  S832,SANTA HELENA
```

Fonte: Própria, (2023).

Dentro da pasta de cada localidade é armazenado um arquivo para cada mês, esse arquivo no formato csv com a nomenclatura de «ano><mes>.csv", na Figura 20 podemos ver os dados do INMET de foz do iguaçu. Cada um desses arquivos possuem dados referente a um mês específico, como é possível ver na Figura 20 o arquivo "202101.csv"possui dados do mês de janeiro de 2021.

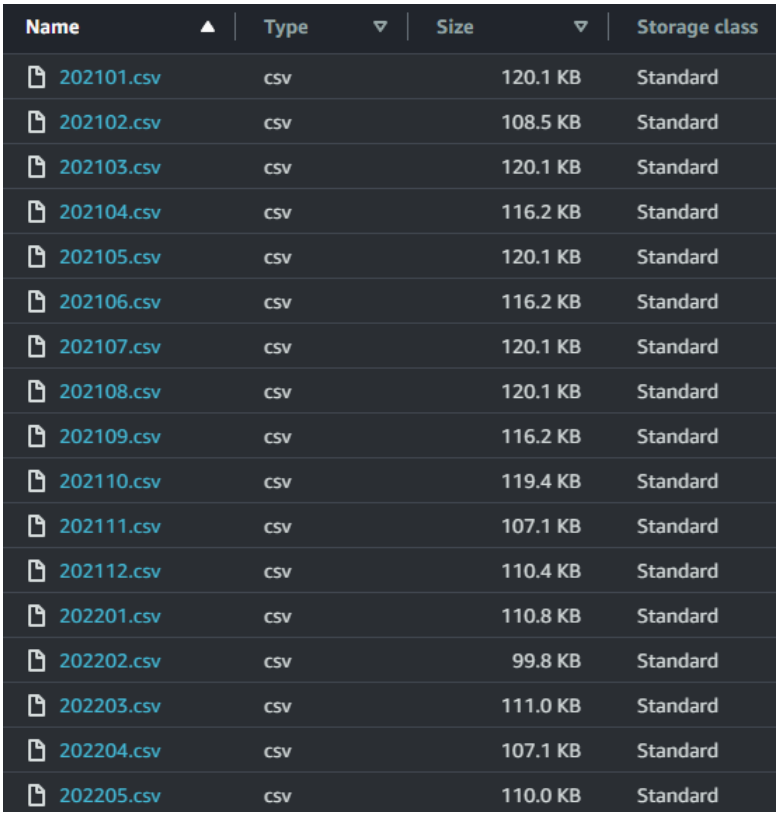
Dentro desses arquivos estão os dados coletados da fonte, e cada fonte possui suas próprias colunas, no exemplo da Figura 21 é possível ver os dados do INMET, onde cada uma das linhas representa um dado e as colunas são descritos na Tabela 7.

3.3 ETL

A terceira etapa é composta pelo processo de ETL, que é o processo de ler os dados que estão no *Data Lake* e fazer o processamento deles para que seja possível fazer a utilização dos algoritmos de aprendizado de máquina. Com os dados processados é feita a inserção no *Data Warehouse* para que os mesmos fiquem disponíveis para consulta.

Para fazer o processamento dos dados foi utilizado de outras funções *lambda* que são chamadas após as de ingestão de dados pelo processo do *Step Functions*,

Figura 20 – Pasta de foz do iguaçu do INMET



Name	Type	Size	Storage class
202101.csv	csv	120.1 KB	Standard
202102.csv	csv	108.5 KB	Standard
202103.csv	csv	120.1 KB	Standard
202104.csv	csv	116.2 KB	Standard
202105.csv	csv	120.1 KB	Standard
202106.csv	csv	116.2 KB	Standard
202107.csv	csv	120.1 KB	Standard
202108.csv	csv	120.1 KB	Standard
202109.csv	csv	116.2 KB	Standard
202110.csv	csv	119.4 KB	Standard
202111.csv	csv	107.1 KB	Standard
202112.csv	csv	110.4 KB	Standard
202201.csv	csv	110.8 KB	Standard
202202.csv	csv	99.8 KB	Standard
202203.csv	csv	111.0 KB	Standard
202204.csv	csv	107.1 KB	Standard
202205.csv	csv	110.0 KB	Standard

Fonte: Própria, (2023).

Figura 21 – Arquivo CSV do INMET no *Data Lake*

```
1 DC_NOME,PRE_INS,TEM_SEN,VL_LATITUDE,PRE_MAX,UF,RAD_GLO,PTO_INS,TEM_MIN,VL_LONGITUDE,UMD_MIN,PTO_MAX,VEN_DIR,D
  T_MEDICAO,CHUVA,PRE_MIN,UMD_MAX,VEN_VEL,PTO_MIN,TEM_MAX,TEN_BAT,VEN_RAJ,TEM_CPU,TEM_INS,UMD_INS,CD_ESTACAO,HR
  _MEDICAO
2 FOZ DO IGUAÇU,None,None,-25.60194444,None,PR,None,None,None,-54.48305554,None,None,None,2021-05-01,None,None,
  None,None,None,None,None,None,None,None,A846,0000
3 FOZ DO IGUAÇU,None,None,-25.60194444,None,PR,None,None,None,-54.48305554,None,None,None,2021-05-01,None,None,
  None,None,None,None,None,None,None,None,A846,0100
4 FOZ DO IGUAÇU,None,None,-25.60194444,None,PR,None,None,None,-54.48305554,None,None,None,2021-05-01,None,None,
  None,None,None,None,None,None,None,None,A846,0200
5 FOZ DO IGUAÇU,None,None,-25.60194444,None,PR,None,None,None,-54.48305554,None,None,None,2021-05-01,None,None,
  None,None,None,None,None,None,None,None,A846,0300
6 FOZ DO IGUAÇU,None,None,-25.60194444,None,PR,None,None,None,-54.48305554,None,None,None,2021-05-01,None,None,
  None,None,None,None,None,None,None,None,A846,0400
7 FOZ DO IGUAÇU,None,None,-25.60194444,None,PR,None,None,None,-54.48305554,None,None,None,2021-05-01,None,None,
  None,None,None,None,None,None,None,None,A846,0500
8 FOZ DO IGUAÇU,None,None,-25.60194444,None,PR,None,None,None,-54.48305554,None,None,None,2021-05-01,None,None,
  None,None,None,None,None,None,None,None,A846,0600
9 FOZ DO IGUAÇU,None,None,-25.60194444,None,PR,None,None,None,-54.48305554,None,None,None,2021-05-01,None,None,
  None,None,None,None,None,None,None,None,A846,0700
```

Fonte: Própria, (2023).

nestas funções foram utilizada o mesmo *runtime* das funções anterior, o *Python 3.9*. Estas funções foram configuradas com 128MB de memória e 300 segundos de *timeout*, como mostra a figura 22.

Também foram criadas variáveis de ambiente parecidas, como mostra a figura 8, sendo elas *bucket*, *keyPrefixDL*, *keyPrefixDW*, a variável *bucket* representa o local onde os dados são armazenados e as variáveis *keyPrefixDL* e *keyPrefixDW* o caminho dentro do *bucket* onde os dados serão lidos no *Data Lake* e salvos no *Data Warehouse* respectivamente.

Tabela 7 – Dados fornecidos pela API do INMET.

Dados	Descrição
DC_NOME	Nome da localização do dado.
PRE_INS	Pressão atmosférica ao nível da estação.
TEM_SEN	Temperatura do sensor.
VL_LATITUDE	Latitude da localização do dado.
PRE_MAX	Pressão atmosférica máxima.
UF	Unidade federativa da localização do dado.
RAD_GLO	Radiação.
PTO_INS	Ponto de orvalho.
TEM_MIN	Temperatura mínima.
VL_LONGITUDE	Longitude da localização do dado.
UMD_MIN	Umidade relativa mínima.
PTO_MAX	Ponto de orvalho máximo.
VEN_DIR	Direção do vento.
DT_MEDICAO	Data da medição.
CHUVA	Precipitação total.
PRE_MIN	Pressão atmosférica mínima.
UMD_MAX	Umidade relativa máxima.
VEN_VEL	Velocidade do vento.
PTO_MIN	Ponto de orvalho mínimo.
TEM_MAX	Temperatura máxima.
TEN_BAT	Tensão da bateria.
VEN_RAJ	Rajada de vento.
TEM_CPU	Temperatura do processador.
TEM_INS	Temperatura do instrumento.
UMD_INS	Umidade relativa do instrumento.
CD_ESTACAO	Código da estação.
HR_MEDICAO	Hora da medição.

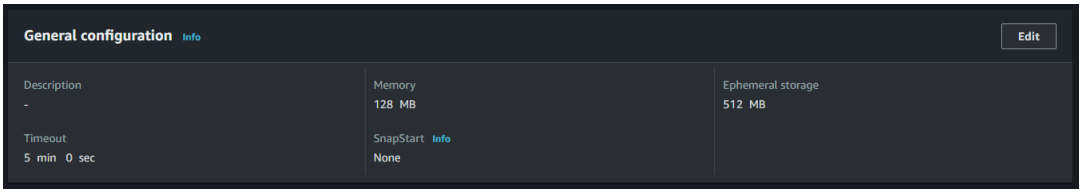
Fonte: Própria, (2023)

Tabela 8 – Variáveis de ambientes utilizadas para funções de ETL de dados.

Nome	Valor
bucket	plataforma-monitoramento-dengue
keyPrefixDL	dl/inmet/
keyPrefixDW	dw/inmet/v1/

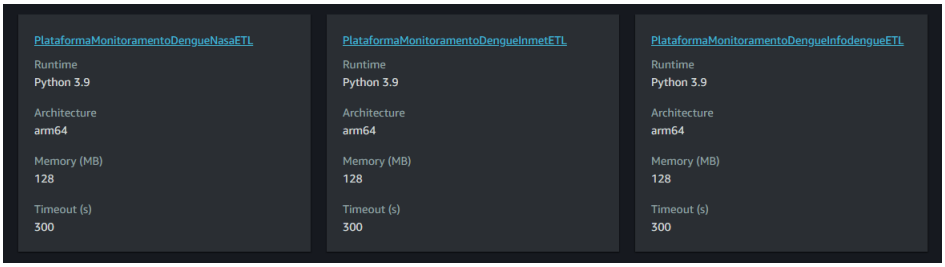
Fonte: Própria, (2023)

Figura 22 – Configuração das funções *Lambda* de ETL de dados



Fonte: Própria, (2023).

Figura 23 – Funções *Lambda* de ETL de dados



Fonte: Própria, (2023).

Para cada uma das fontes foi criada uma função *Lambda* como mostra a Figura 23, Essas funções tem como objetivo fazer a limpeza e pré-processamento dos dados, para deixá-los prontos para as próximas etapas dentro do *Data Warehouse*.

Durante o processo de ETL os arquivos serão processados para fazer 5 tarefas, sendo elas, remover linhas vazias, preencher dados de colunas vazias, descartar as colunas desnecessárias, filtrar as estações e as datas e por último agrupar os dados na periodicidade necessária, sendo cada uma delas como explicado abaixo:

- A remoção das linhas vazias será um filtro para remover os dados faltantes de cada arquivo, pois em alguns arquivos há linhas sem dados, que pode ter sido causada por falta da coleta dos dados pela estação meteorológica.
- Em alguns registros algumas das colunas estão sem dados, nesse casos esses campos deverão ser tratados, esse tratamento podendo ser com a inserção de um valor ou com a remoção da coluna.
- Como mostrado na tabela 3 o INMET fornece diversos dados a partir das estações meteorológicas, porém nem todos serão utilizados, então durante o processo de ETL essas colunas desnecessárias serão removidas.
- O INMET faz o fornecimento dos dados de todas as estações do Brasil, mas como este trabalho foca na coleta e análise de cidades é necessário apenas o uso das estações mais próximas do local a ser analisado.
- Por último será feito o agrupamento dos dados baseados nas datas, já que uma periodicidade de hora em hora é muito específica e pode causar o

sobre-ajuste dos algoritmos de aprendizado de máquina. Esse agrupamento pode ser feito de diversas maneiras. Como por exemplo, agrupar os dados por dia, por semana ou por mês.

Para realizar essas 5 tarefas foi necessário apenas uma única função por fonte, a *"lambda_handler"*, que é chamada pela execução da função *Lambda*, nela é feita todo o processo, como é possível ver na figura 24.

Figura 24 – Código da função *Lambda* de ETL de dados

```
1  import boto3
2  import os
3  from datetime import date, timedelta
4
5  s3 = boto3.client('s3')
6
7  def lambda_handler(event, context):
8      keys = ["DATETIME", "TEMP_MAX", "TEMP_MIN", "UMID_MAX", "UMID_MIN", "WIND_DIR", "WIND_VEL", "RAD_GLO", "CHUVA"]
9
10     stationsList = s3.list_objects_v2(Bucket=os.environ['bucket'], Prefix=os.environ['keyPrefixDL'])['Contents']
11     stations = list(filter(lambda a: a!='', list(dict.fromkeys([s['Key'].split('/')[2] for s in stationsList]))))
12     for station in stations:
13         files = list(filter(lambda a: a!='', [s['Key'].split('/')[3].split('.')[0] if len(s['Key'].split('/'))==4 and s['Key'].
               split('/')[2]==station else '' for s in stationsList]))
14         years = list(dict.fromkeys([f[0:4] for f in files]))
15
16         for y in years:
17             data = [",".join(keys)]
18             for f in list(dict.fromkeys(filter(lambda a: a!='', [f if y==f[0:4] else '' for f in files]))):
19                 fData = s3.get_object(Bucket=os.environ['bucket'], Key=os.environ['keyPrefixDL']+station+'/'+f+'.csv')['Body'].
                       read().decode("utf-8").split("\n")[1:]
20                 for fd in fData:
21                     c = fd.split(",")
22
23                     if(c[19] == "None"):
24                         continue
25                     str = []
26                     str.append(c[13]+" "+c[26][0:2]+":"+c[26][2:]+":00") # DATE TIME
27                     str.append(c[19] if c[19]!="None" else "") # TEMP MAX
28                     str.append(c[8]  if c[8] !="None" else "") # TEMP MIN
29                     str.append(c[16] if c[16]!="None" else "") # UMID MAX
30                     str.append(c[10] if c[10]!="None" else "") # UMID MIN
31                     str.append(c[12] if c[12]!="None" else "") # WIND DIR
32                     str.append(c[17] if c[17]!="None" else "") # WIND VEL
33                     str.append(c[6] if c[6]!="None" else "") # RAD GLO
34                     str.append(c[14] if c[14]!="None" else "") # CHUVA
35                     data.append(",".join(str))
36             if len(data)>1:
37                 s3.put_object(Bucket=os.environ['bucket'], Key=os.environ['keyPrefixDW']+y+'/'+station+'.csv', Body="\n".join(
                       data))
```

Fonte: Própria, (2023).

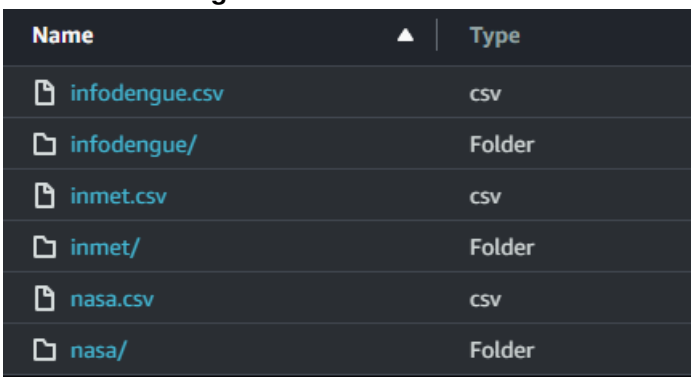
Como é possível ver na Figura 24 primeiro é feita a verificação dos arquivos pendentes de serem feitos o processo de ETL, nas linhas 10 à 14, para cada um deles é feito a leitura do mesmo, como mostra a linha 16. Para cada registro dentro do arquivo CSV lido do S3 e verificado se existem dados para remover as linhas vazias, é feita a seleção dos dados necessários e salvá-los em formato csv, nas linhas 17 em diante.

Com o arquivo do *Data Lake* processado, é feita a criação de um novo arquivo dentro do *Data Warehouse* no local configurado pela variável de ambiente.

3.4 Data Warehouse

Com os dados processados na etapa de ETL, os dados são armazenados no *Data Warehouse*, que é o local onde os dados são organizados para que seja possível a utilização dos mesmos para a geração de visualizações e análises. O armazenado é feito de forma parecida ao do *Data Lake*, em uma pasta no S3. Como é possível ver na Figura 25 os dados são armazenados em uma pasta chamada "dw", dentro dessa pasta os dados são organizados por fonte e versão.

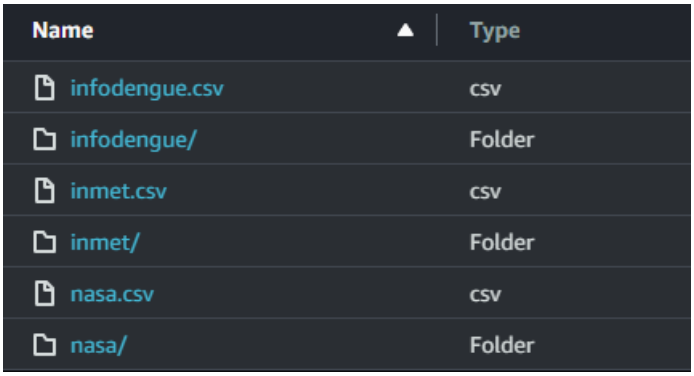
Figura 25 – Pasta "dw"no *bucket* do S3



Fonte: Própria, (2023).

Dentro da pasta de cada fonte há pastas com a versão dos dados, como é possível ver na Figura 26 há a pasta "v1"que representa a versão 1 dos dados, essas versões são usadas para caso seja necessário ter dados em formatos ou com dados diferentes que foram gerados pelo processo de ETL, como por exemplo, dados agrupados por dia ou por semana. Esses diferentes tipos de dados são criados de acordo com a necessidade de cada análise.

Figura 26 – Pasta "dw"no *bucket* do S3



Fonte: Própria, (2023).

Figura 27 – Arquivo CSV do INMET no *Data Warehouse*

```
DATETIME,TEMP_MAX,TEMP_MIN,UMID_MAX,UMID_MIN,WIND_DIR,WIND_VEL,RAD_GLO,CHUVA
2022-01-01 00:00:00,28.9,27.7,50,44,351,0,-3.5,0
2022-01-01 01:00:00,28.5,25.2,59,45,253,0.6,-3.5,0
2022-01-01 02:00:00,25.4,23.9,72,59,256,1.8,-3.5,0
2022-01-01 03:00:00,24.8,23.1,77,65,255,1.7,-3.5,0
2022-01-01 04:00:00,23.3,21.5,83,76,254,2.3,-3.5,0
2022-01-01 05:00:00,22.8,21.2,86,83,239,3.1,-3.5,0
2022-01-01 06:00:00,23.1,22.3,85,82,229,3.3,-3.5,0
2022-01-01 07:00:00,22.6,21.2,86,82,274,0.1,-3.5,0
2022-01-01 08:00:00,21.3,20.8,87,85,255,0.9,-3.5,0
2022-01-01 09:00:00,21,20.5,87,86,240,2.2,0.5,0
2022-01-01 10:00:00,23.4,20.9,87,77,229,4.2,304.1,0
2022-01-01 11:00:00,25.5,23.4,77,67,233,4.5,1138,0
2022-01-01 12:00:00,27.8,25.4,67,58,214,3.1,2016.5,0
2022-01-01 13:00:00,30.7,27.6,60,47,154,5.3,2831,0
2022-01-01 14:00:00,32.2,30.1,49,41,153,4.4,3475,0
2022-01-01 15:00:00,33.6,31.5,44,38,131,3.5,3915.6,0
2022-01-01 16:00:00,34.1,32.9,40,35,83,2.2,3445.4,0
2022-01-01 17:00:00,35.4,33,38,32,155,2.9,2575.9,0
2022-01-01 18:00:00,35.8,33.4,37,33,250,0.7,2295.1,0
```

Fonte: Própria, (2023).

De forma parecida ao *Data Lake* os dados são agrupados em arquivos por data, mas na maioria por ano ao invés de mensal, porém podendo ser em outros agrupamentos de acordo com a necessidade. A figura 27 mostra um exemplo de arquivo CSV no *Data Warehouse*, nesses arquivos as colunas são apenas as necessárias e preparadas para a fase de análise de dados. No caso da Figura 27 de exemplo essas colunas são descritas na Tabela 9.

Tabela 9 – Colunas do arquivo CSV do *Data Warehouse*.

Dados	**Descrição**
DATETIME	Data e hora da medição.
TEMP_MAX	Temperatura máxima.
TEMP_MIN	Temperatura mínima.
UMID_MAX	Umidade relativa máxima.
UMID_MIN	Umidade relativa mínima.
WIND_DIR	Direção do vento.
WIND_VEL	Velocidade do vento.
RAD_GLO	Radiação.
CHUVA	Precipitação total.

Fonte: Própria, (2023)

Outras possibilidade de arquiteturas na AWS de *Data Warehouse* podem utilizar diferentes serviços para o armazenamento dos dados, cada opção traz seus beneficios e maleficios, como por exemplo, o uso do Redshift, que é um serviço específico para fazer *Data Warehouse* porém tem um grande custo de execução, outra possibilidade é o uso de banco de dados relacionais, como o RDS, para esse caso o custo não é muito grande, mas a complexidade da arquitetura é maior, tendo que lidar com conexões e arquitetura de banco de dados.

Neste projeto foi escolhido o uso do S3 para o armazenamento dos dados por

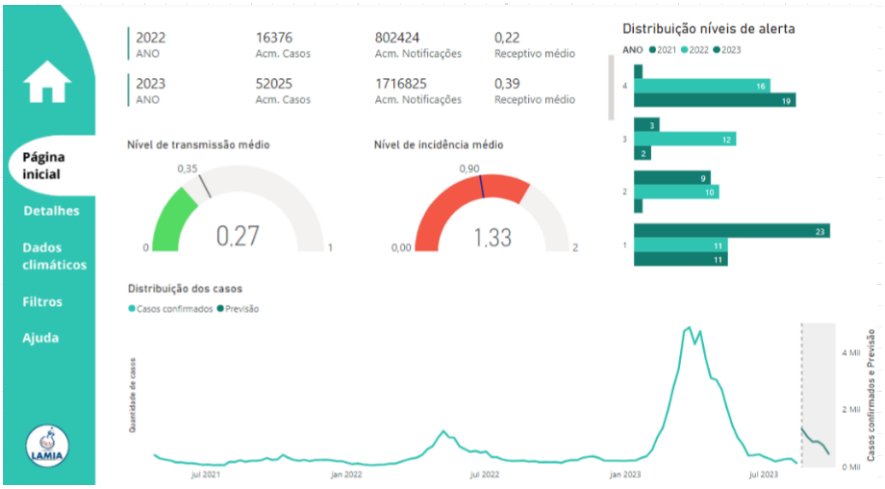
ser um serviço de baixo custo e de fácil utilização, porém com a desvantagem de não ser um banco de dados relacional, fazendo com que seja necessário tratar todos os dados no *Data Warehouse* como arquivos.

3.5 Análise de Dados

Para finalizar o processo de *Big Data* de dados é necessário executar a etapa 5 da Figura 10, que é o processo de rodar os algoritmos e analisar os resultados. Esse trabalho não irá focar nos algoritmos de aprendizado de máquina, então trataremos essa etapa apenas como a parte de análise dos resultados e consequências dos usos dos tipos de dados.

Com o resultados dos algoritmos de aprendizado máquina serão gerados gráficos para a análise dos resultados, esses gráficos devem ser comparados com os dados da realidade para descobrir a acurácia dos algoritmos e do processo de aprendizado de máquina.

Figura 28 – Dashboard no *PowerBI*



Fonte: Própria, (2023).

Para a análise foi criado um *dashboard* utilizando o *PowerBI*[3] para a visualização dos dados, como é possível ver na Figura 28. Esse dashboard tem como objetivo analisar os dados e tentar tirar conclusões sobre os dados da dengue. Alguns dos indicadores que foram criados são:

- **Nível de transmissão médio**: Este gráfico mostra o nível de transmissão médio de dengue atual.

[3] https://powerbi.microsoft.com/.

- **Nível de incidência médio**: Este gráfico mostra o nível de incidência médio de dengue atual.
- **Distribuição níveis de alerta**: Neste gráfico é possível ver a distribuição dos níveis de alerta de dengue por ano.
- **Distribuição de casos**: Neste gráfico é possível ver a quantidade de casos de dengue por semana epidemiológica.

4 RESULTADOS E VALIDAÇÃO

Neste capítulo são apresentados os resultados dos testes e as técnicas utilizadas. Os resultados são apresentados em formato de tabelas, para que uma melhor visualização em relação ao todo seja feito.

Foram feitos testes com a arquitetura com diversas configurações para validar e comparar a arquitetura, também foram feitos testes utilizando uma arquitetura tradicional utilizando o Hadoop para colocar de contraponto a arquitetura apresentada.

Para apresentar os resultados deste trabalho, os mesmos serão divididos em três partes, sendo a primeira referente os conhecimentos necessários para configurar e aplicar a arquitetura, a segunda parte será referente a performance da arquitetura e a terceira parte será referente aos custos.

4.1 Configuração do Ambiente

Para analisar a configuração do ambiente, foi levado em conta os conhecimentos necessários, a dificuldade de configuração e manutenção e por último o tempo gasto para fazer a arquitetura rodar de forma automática.

Para montar a arquitetura é necessário alguns conhecimentos básicos independente da arquitetura utilizada, alguns desses conhecimentos são explicados a seguir.

- **Conhecimento de coletores de dados:** Para montar a arquitetura é necessário conhecer onde os dados estão e as permissões necessárias para acessar os dados.
- **Conhecimento de APIs e leitura de dados:** Para montar a arquitetura é necessário conhecer como acessar os dados e como ler os dados. Uma das principais formas de acessar os dados é através de APIs, por isso é necessário conhecer como utilizar APIs e seus métodos de autenticação. Pode ser necessário também conhecer como ler os dados de um arquivo, como *web scraping* ou leitura de arquivos como PDF ou XLSX.
- **Conhecimento de armazenamento de dados:** Para montar a arquitetura é necessário conhecer sobre armazenamento de dados e estruturas de arquivos. Sabendo ler formatos como CSV e JSON, e conhecendo as estru-

turas de armazenamento de dados como bancos de dados relacionais e não relacionais.

- **Conhecimento de programação:** Para montar a arquitetura é necessário conhecer alguma linguagem de programação, para que seja possível criar os coletores de dados e os processadores de dados. Normalmente é utilizado Python para essas funções, mas de acordo com a arquitetura e tecnologias utilizadas outras linguagens podem ser utilizadas.
- **Conhecimento de processamento de dados:** Para montar a arquitetura é necessário conhecer como processar os dados, como por exemplo, como fazer a limpeza dos dados, como fazer a transformação dos dados e como fazer a organização dos dados.
- **Conhecimento de orquestração de tarefas:** Para montar a arquitetura é necessário conhecer como orquestrar as tarefas, como por exemplo fazer um *workflow* de tarefas, para que seja possível fazer a orquestração das tarefas.
- **Conhecimento de visualização de dados:** Para montar a arquitetura é necessário conhecer como visualizar os dados, como por exemplo, como visualizar os dados utilizando o PowerBI ou outra tecnologia similar.

Porém além dos conhecimentos básicos, são necessários conhecimentos específicos de acordo com a arquitetura utilizada. Para a arquitetura apresentada é necessário ter o conhecimento do uso das ferramentas e serviços da AWS. A princípio pode parecer que é necessário muito tempo para aprender a utilizar a AWS, mas por ser um serviço fragmentado em serviços menores, é possível aprender a utilizar os serviços de forma separada, e como esta arquitetura utiliza apenas alguns serviços da AWS, é possível aprender a utilizar os serviços necessários em pouco tempo. Entretanto, os conhecimentos adquiridos não terão muito uso fora da AWS, fazendo com que o conhecimento adquirido seja muito específico.

Como mostrado na metodologia, esta arquitetura utiliza dos serviços *AWS Lambda*, *Amazon S3* e *AWS Step Functions*. Além deles também são utilizados alguns outros serviços base, como o *AWS Identity and Access Management*, *Amazon CloudWatch* e *AWS Billing Console*. Esses três serviços servem para monitorar e gerenciar os serviços utilizados, mas são utilizados poucas vezes, apenas para configuração inicial do projeto e monitoramento.

O uso do *Amazon S3* é simples, sendo necessário apenas criar um *bucket*

com configurações padrões, o mesmo acontece para o serviço *AWS Step Functions*, onde devesse apenas selecionar as funções que serão executadas em ordem. Já para o serviço *AWS Lambda* é necessário mais configurações, como a escolha de uma linguagem de programação, o tamanho da memória e o tempo máximo de execução. Essas configurações são simples e não necessitam de muito conhecimento para serem feitas, para a linguagem de programação o usuário é livre para escolher a que preferir, fazendo com que a programação seja mais fácil, já para memória e tempo de execução máximo, é necessário fazer alguns testes para encontrar o melhor valor. O que traz mais dificuldade na configuração da função lambda são configurações de permissões utilizando o *AWS Identity and Access Management* que pode ser um pouco confusa, mas essas configurações são feitas apenas uma vez, e podem ser copiadas para outras funções, fazendo com que o tempo gasto para configurar as permissões seja reduzido.

Em comparação, para utilizar uma arquitetura tradicional com o *Hadoop*, diversos conhecimentos adicionais são necessários, e esses sendo mais complexos e tendo uma necessidade de estudo mais profundo, mas muitos desses conhecimentos são utilizados em diversas áreas da tecnologia da informação. Alguns desses conhecimentos são explicados na lista a seguir.

- **Conhecimento de linux:** Para montar a arquitetura é necessário conhecer como utilizar o linux, como por exemplo, como instalar programas, como configurar o sistema operacional, como utilizar o terminal e como utilizar o sistema de arquivos.
- **Conhecimento de redes:** Para montar a arquitetura é necessário conhecer como configurar uma rede, como por exemplo, como configurar IPs e roteamento de máquinas virtuais ou contêineres.
- **Conhecimento de virtualização:** Para montar a arquitetura é necessário conhecer como utilizar máquinas virtuais e contêineres, como por exemplo, como instalar e configurar máquinas virtuais ou o uso de programas como o Docker.
- **Conhecimento de banco de dados:** Para montar a arquitetura é necessário conhecer como utilizar bancos de dados, como por exemplo, como instalar e configurar bancos de dados relacionais e não relacionais.
- **Conhecimento de Java:** É necessário conhecer Java para conseguir configurar e utilizar o Hadoop. Além de ser necessário conseguir gerenciar os

pacotes e dependências de versões diferentes no Java.

- **Conhecimento de Hadoop:** Para montar a arquitetura é necessário conhecer como utilizar o Hadoop, como por exemplo, como instalar e configurar o Hadoop, como editar os arquivos de configuração do Hadoop e como utilizar os comandos do Hadoop.
- **Conhecimento de Spark:** Para montar a arquitetura é necessário conhecer como utilizar o Spark, como por exemplo, como instalar e configurar o Spark, como configurar os arquivos de configuração do Spark e como utilizar os comandos do Spark.
- **Conhecimento de HDFS:** Para montar a arquitetura é necessário conhecer como utilizar o HDFS, como por exemplo, como instalar e configurar o HDFS, como configurar os arquivos de configuração do HDFS e como utilizar os comandos do HDFS.
- **Conhecimento de YARN:** Para montar a arquitetura é necessário conhecer como utilizar o YARN, como por exemplo, como instalar e configurar o YARN, como configurar os arquivos de configuração do YARN e como utilizar os comandos do YARN.
- **Conhecimento de outras tecnologias:** Para montar a arquitetura é necessário conhecer como utilizar outras tecnologias, como por exemplo, como Hive, HBase, Kafka e outros.

Por serem conhecimentos mais complexos, é necessário mais tempo para aprender a utilizar essas tecnologias. Além de que a configuração geralmente é mais difícil de ser feita e os conteúdos encontrados na internet são de versões mais antigas ou incompatíveis, fazendo com que seja necessário um trabalho manual de testes e configurações para conseguir êxito na execução da arquitetura.

Em casos específicos foi necessário alterar o código fonte de algumas tecnologias para execução da arquitetura, ou recomeçar a configuração do zero na tentativa de uso de novos métodos. Algumas alternativas para reduzir o tempo gasto com estes processos é a utilização de ferramentas de automatização da instalação, como Ansible[1] ou Docker[2]. Assim, é possível reproduzir a arquitetura em diversas máquinas, mas

[1] https://www.ansible.com/.
[2] https://www.docker.com/.

para isso é necessário conhecer essas ferramentas e como utilizá-las, trazendo maior complexidade ao sistema.

Utilizando a arquitetura apresentada neste trabalho, são necessárias algumas poucas horas para um usuário de nível iniciante conseguir configurá-la de forma automática, a partir do momento que este já tenha aprendido os conhecimentos básicos de AWS. Já na arquitetura tradicional, são necessários alguns dias para um usuário de nível iniciante conseguir o mesmo resultado de antes. Pelo maior grau de complexidade e tempo gasto, esta opção é recomendada apenas a usuários com maior experiência.

Um profissional com conhecimento avançado da arquitetura tradicional, poderá executar uma arquitetura complexa, com maiores funcionalidades e mais adaptável. Porém, dentro das necessidades de *Big Data* estes prováveis benefícios podem ser desnecessários em muitos casos de problemas que já são bem cobertos com arquiteturas tradicionais.

4.2 Performance

A análise de performance acontece nos momentos em que códigos são rodados, já que o resto apenas é o armazenamento de dados. Na arquitetura apresentada neste trabalho isso ocorre em dois momentos, na coleta de dados e no processamento dos dados. Onde ambos rodam em funções *lambda* na AWS.

Para medir a performance do código executado na função *lambda* foram feitos testes com diversas quantidades de memória e situações diferentes. Como ponto de comparação, também foram executados testes utilizando uma arquitetura tradicional, utilizando o Hadoop, onde o código foi executado em Python com o Flume fazendo a transferência dos arquivos para dentro da arquitetura.

Para os testes foram utilizados os dados das fontes INMET, NASA e infodengue, nas cidades de Santa Helena e Foz do Iguaçu no Paraná. Os dados utilizados são de 2022 até 2023 e foram feitos testes com 128, 256, 512, 1024 e 2048 MB de memória alocada para a função *lambda*. Para cada teste foi medido o tempo de execução da função *lambda* e o tempo de execução total do código, que inclui o tempo de execução da função *lambda* e o tempo de transferência dos dados para a arquitetura e a quantidade de memória utilizada. Vale ressaltar que dentro do tempo medido está incluso o tempo de resposta das APIs, que pode variar de acordo com a quantidade

de requisições feitas e a quantidade de dados retornados, onde esse tempo foge do controle de qualquer arquitetura que utilize esses dados.

Para os testes com a arquitetura tradicional, foram utilizados os mesmos dados e as mesmas cidades. Para cada teste foi medido o tempo de execução, mas por ter que ser feito em máquinas reais não foram feitos testes com diferentes quantidades de memória ou processadores. Sendo que o propósito deste teste é apenas dar uma base de comparação a uma arquitetura tradicional.

4.2.1 Coleta de Dados

Durante os testes foram executados os códigos três vezes para cada quantidade de memória, foram descartados o maior e menor valor, para que os valores mostrados não sejam *outliers*.

Foram testadas duas situações de coleta de dados, a primeira sendo o caso completo, onde é necessário fazer o download de todos os dados, este caso ocorre durante a primeira vez que é feita a coleta de um fonte, onde a arquitetura está em um estado inicial, sem dados salvos tanto no *Data Lake* quanto no *Data Warehouse*. Porém, a partir do momento que já possua os dados, não é necessário baixar todos, apenas os dados novos, fazendo com que o tempo de execução seja menor, este caso é chamado de caso simples. O caso simples é o que acontece na maior parte da execução da arquitetura, já que a maior parte do tempo os dados já estão salvos no *Data Lake* e no *Data Warehouse*, pois estes são ingeridos de forma automática.

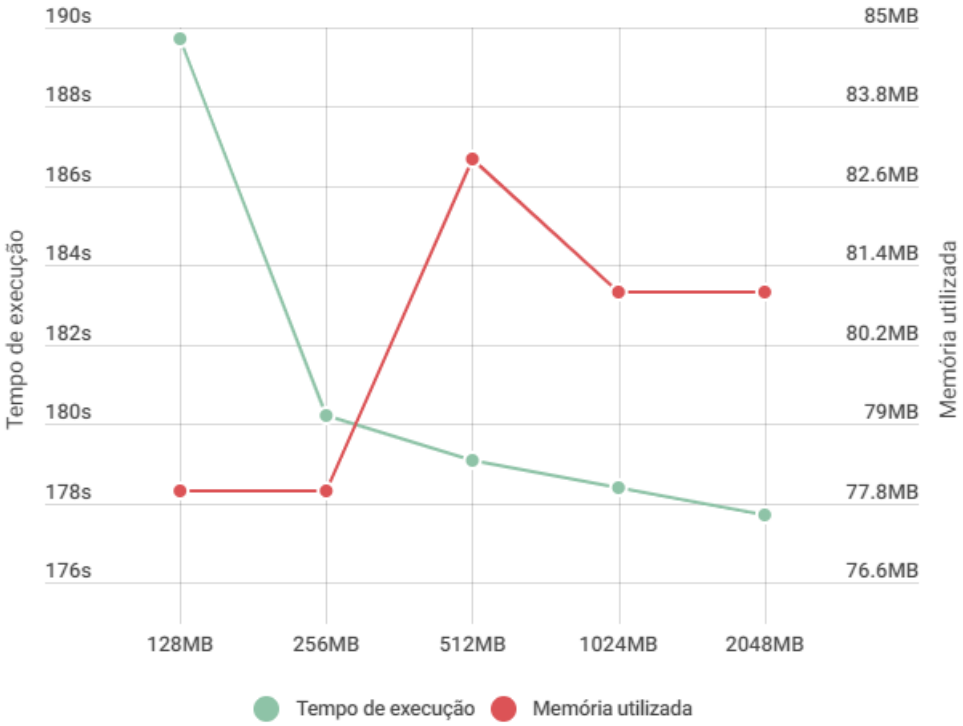
Tabela 10 – Comparação de performance do caso simples de coleta de dados

Quantidade de memória	Tempo de execução	Memória utilizada
128MB	189,7s	78MB
256MB	180,2s	78MB
512MB	179,1s	83MB
1024MB	178,4s	81MB
2048MB	177,7s	81MB

Fonte: Própria, (2023)

Como é possível ver na Figura 29, o tempo de execução, linha em verde, começa em 189,7s para 128MB e vai diminuindo conforme a quantidade de memória aumenta, chegando a 177,7s para 2048MB. Já a quantidade de memória utilizada, linha em vermelho, começa em 78MB para 128MB e pouco se altera ao aumentar a

Figura 29 – Comparação de performance do caso simples de coleta de dados.



Fonte: Própria, (2023).

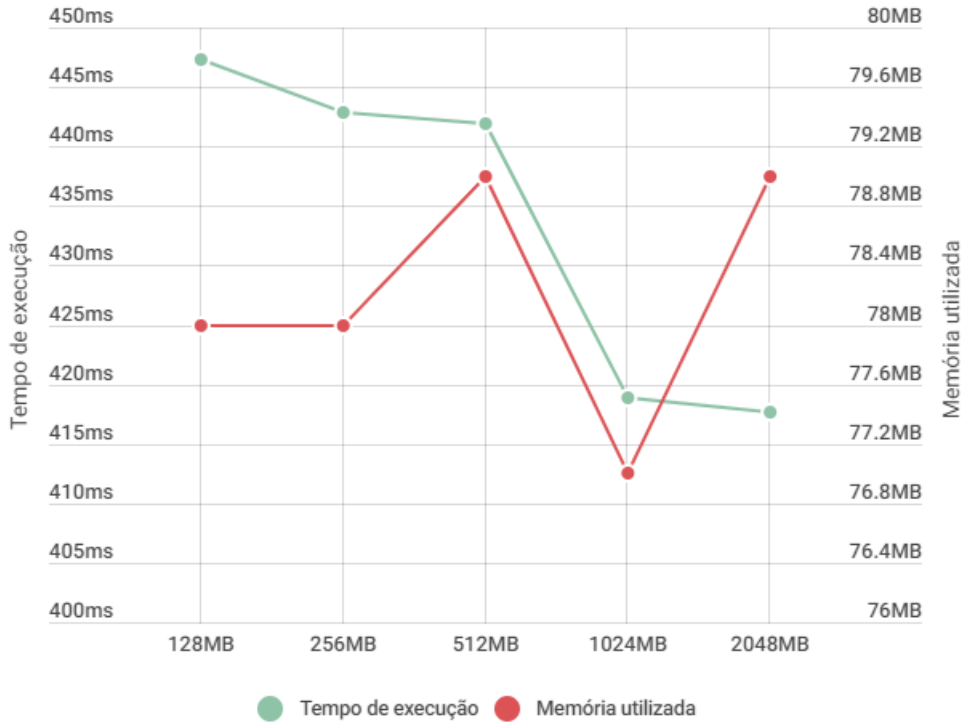
quantidade de memória alocada, chegando a 81MB com 2048MB de memória alocada, neste caso podemos dizer que a memória utilizada continua a mesma, já que a diferença é muito pequena, com diferentes quantidades de memória alocada.

Em relação ao tempo de execução houve uma redução de 0.06% ao aumentar de 128MB para 256MB, porém ao aumentar de 256MB para 512MB a redução de tempo foi de apenas 0.007% e valores semelhantes para cada aumento subsequente, mostrando que quanto mais memória adicionada, menor é a redução de tempo de execução.

A pequena alteração de tempo de execução se dá porque a maior influência no tempo de execução é causada pelas chamadas de API. Outro ponto a se atentar é que ao aumentar a memoria alocada, a quantidade de *vcpus* também aumenta de forma proporcional, fazendo com que mesmo sem a necessidade de mais memória pelo programa, possa haver uma diferença de tempo de execução, já que há mais processadores executando a função.

No caso simples, como é possível ver na tabela 11, a comportamento é parecido, porém o tempo de execução possui uma diferença maior, já que são feitas poucas chamadas para API, mas ainda ela é o principal influenciador do tempo, mas o compor-

Figura 30 – Comparação de performance do caso completo de coleta de dados.



Fonte: Própria, (2023).

Tabela 11 – Comparação de performance do caso completo de coleta de dados

Quantidade de memória	Tempo de execução	Memória utilizada
128MB	447,3ms	78MB
256MB	442,8ms	78MB
512MB	441,9ms	79MB
1024MB	418,9ms	77MB
2048MB	417,7ms	79MB

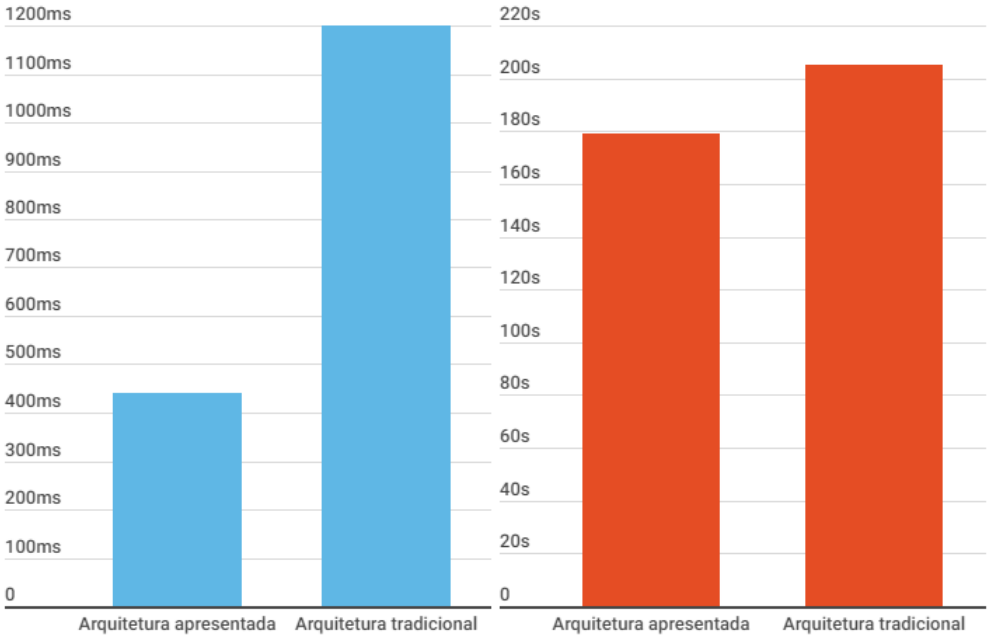
Fonte: Própria, (2023)

tamento ainda é similar ao caso completo, apenas em uma escala de tempo menor, em milissegundos invés de minutos. Esses comportamentos dos dois casos fazem que a quantidade ideal de memória seja definida pelo custo, que será apresentado na seção 4.3.

Neste caso acontece um grande redução na quantidade de tempo de execução na alteração de 512MB para 1024MB, onde essa redução é de 0.06%, onde nas outras alterações as reduções são mínimas, em realização ao uso de memória, mais uma vez é possível ver que a quantidade de memória alocada, pouco influencia a memória utilizada, mantém se em um patamar similar de por volta de 78MB, o que mostra que grande parte da memória nessas utilizações são dados pelos requisitos do *runtime* do

Python, e não pelo código em si.

Figura 31 – Comparação de performance da arquitetura apresentada e da arquitetura tradicional no caso simples, a esquerda, e no caso completo, a direita.



Fonte: Própria, (2023).

Tabela 12 – Comparação de performance da arquitetura apresentada e da arquitetura tradicional

Arquitetura	Quantidade de dados	Tempo de execução
Arquitetura apresentada	Caso simples	441ms
Arquitetura tradicional	Caso simples	1,2s
Arquitetura apresentada	Caso completa	179,1s
Arquitetura tradicional	Caso completa	205,3s

Fonte: Própria, (2023)

Na comparação entre arquiteturas, mostrada na tabela 12, é possível ver que existe uma diferença entre as arquiteturas, porém a comparação não pode ser levada a fundo, já que os valores da arquitetura tradicional podem ser influenciados por diversos motivos, como rodar o código em uma máquina mais ou menos potente, a quantidade de softwares rodando em *background*. Com isso essa comparação deve ser utilizada apenas como ponto de referência e não como uma comparação definitiva.

4.2.2 Processamento de Dados

Para testar o processamento de dados foram feitos testes iguais aos do teste de coleta de dados, porém com a diferença de que os dados utilizados são os dados já

coletados e armazenados no *Data Lake*. Na tabela 13 é possível ver os resultados dos testes.

Tabela 13 – Tempo de execução da função de processamento de dados

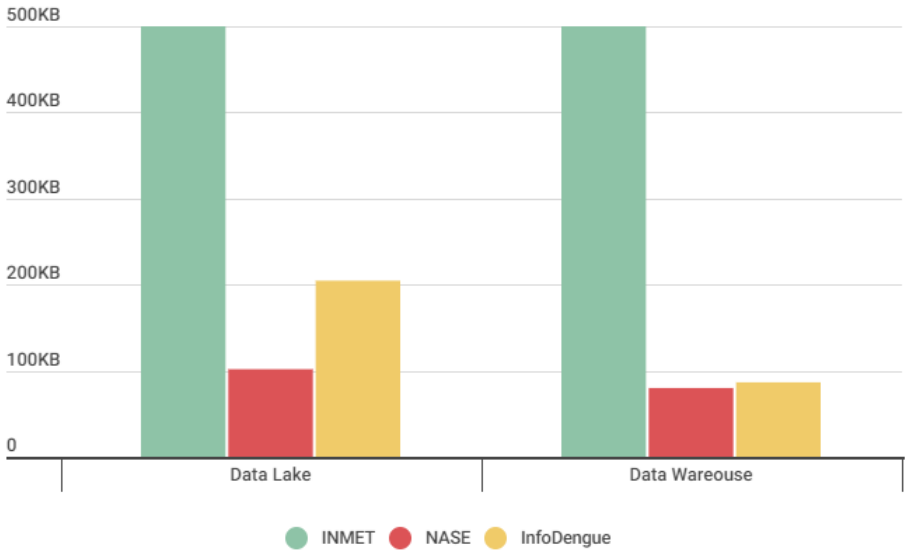
Quantidade de memória	Tempo de execução	Memória utilizada
128MB	2685ms	77MB
256MB	1611ms	76MB
512MB	1370ms	78MB
1024MB	1311ms	78MB
2048MB	1301ms	100MB

Fonte: Própria, (2023)

Como é possível ver, o tempo do processamento de dados é mais afetado pela quantidade de memória alocada, já que nele não há requisições a API, apenas a leitura, transformação e escrita dos dados. Fazendo que um poder maior de processamento afete diretamente no tempo de execução, onde rodar com 2GB reduz o tempo de execução em mais de 50% em relação a rodar com 128MB.

4.2.3 Armazenamento

Figura 32 – Tamanho dos dados armazenamentos no *Data Lake* e no *Data Warehouse*.



Fonte: Própria, (2023).

O armazenamento de dados tem performance parecida em ambas as arquiteturas, na arquitetura apresentada neste trabalho e na tradicional utilizando o Hadoop, já que os dados são os mesmos, na tabela 14 pode ser ver quanto que cada fonte

Tabela 14 – Tamanho dos dados armazenamentos no *Data Lake* e no *Data Warehouse*

Local	Fonte	Tamanho
Data Lake	INMET	5.1MB
Data Lake	NASA	101.7KB
Data Lake	InfoDengue	204.6KB
Data Warehouse	INMET	1.1MB
Data Warehouse	NASA	79.9KB
Data Warehouse	InfoDengue	86.5KB

Fonte: Própria, (2023)

de dados ocupa de no *Data Lake* e no *Data Warehouse*. Quanto maior o período de dados maior será a quantidade de armazenamento utilizado, sendo que a quantidade aumentará de maneira proporcional ao período de dados.

É possível ver também que no *Data Warehouse* a quantidade de dados é menor, já que durante o processamento de dados é feito o processo de limpeza e salvo apenas os dados essenciais para a análise, fazendo com que a quantidade de dados seja menor. Porém caso mais cenários de análise sejam criados serão apenas adicionados dados no *Data Warehouse*, já que os dados das fontes continuam os mesmo.

4.3 Custo

Para a análise de custo foi utilizado os valores de preço da AWS, que podem ser vistos nas documentações de cada serviço. Nesta arquitetura existem apenas dois custos principais, o custo de armazenamento e o custo de processamento, caso seja utilizado uma quantidade exponencial de dados outros custos devem ser incluídos, como de transferência de dados.

O custo de armazenamento é simples de se calcular, deve-se apenas multiplicar o número de GB por 0,0405 USD, que é o preço por GB armazenado na AWS, porém este valor é referente ao armazenamento no servidor sa-east-1, em São Paulo, caso se utilize outros servidores os valores podem variar. Nesta arquitetura utilizando as três fontes de dados é possível ver os custos por fonte na tabela 15

Como é possível ver na tabela 15, o custo de armazenamento é muito baixo, podendo ser considerado insignificante na arquitetura, já que os dados são pequenos e o custo é baixo. Porém, caso seja utilizado uma quantidade muito maior de dados,

Tabela 15 – Custo de armazenamento por fonte de dados

Fonte	GB armazenados	Valor em USD
INMET	6,2MB	U$0,00024
NASA	181,6KB	U$0,000007
InfoDengue	291,1KB	U$0,000011

Fonte: Própria, (2023)

o custo de armazenamento pode ser um problema, já que o custo aumenta de forma proporcional a quantidade de dados.

O custo de processamento é um pouco mais complexo, nele é necessário calcular o tempo de execução e a quantidade de memória utilizada, além da quantidade de execuções que foram feitas na função *lambda*. O custo de execução é de U$0,20 por milhão de execuções, ou seja, pode ser desconsiderado já que são feitas 2 execuções por fonte por dia, totalizando menos de 2 mil execuções no ano. Já o custo de processamento é 0,0000166667 USD para cada GB-segundo, então devemos multiplicar o tempo pela quantidade de GB utilizados.

Utilizando os dados medidos na seção 4.2, podemos calcular o custo de processamento, como mostrado na tabela 16.

Tabela 16 – Comparação de custo por execução da função de coleta de dados completa

Quantidade de memória	Tempo de execução	Custo em USD
128MB	189,7s	U$0,0003716674
256MB	180,2s	U$0,0007508348
512MB	179,1s	U$0,0014926697
1024MB	178,4s	U$0,0029733393
2048MB	177,7s	U$0,0059233452

Fonte: Própria, (2023)

Neste caso é financeiramente vantajoso rodar com 128MB, já que o mesmo gera o menor custo, porém por ser um custo baixo, pode ser feita a execução com maiores quantidades de memória caso queira diminuir o tempo de execução e não se importe com o pequeno aumento de custo.

Na tabela 17 é apresentado o custo na coleta de dados no caso simples, nela podemos ver que por rodar em menos de segundos, o valor começa a ficar insignificante, podendo-se utilizar qualquer quantidade de memória sem se preocupar com o custo.

Somando os valores apresentados o custo de execução da arquitetura apre-

Tabela 17 – Comparação de custo por execução da função de coleta de dados simples

Quantidade de memória	Tempo de execução	Custo em USD
128MB	447,3ms	U$0,0000009319
256MB	442,8ms	U$0,0000018450
512MB	441,9ms	U$0,0000036825
1024MB	418,9ms	U$0,0000069817
2048MB	417,7ms	U$0,0000139777

Fonte: Própria, (2023)

sentada neste trabalho é de menos de U$1,00 por mês, fazendo com que a arquitetura seja muito barata de se executar, deve-se atentar que conforme mais dados forem ingeridos e mais processamento for feito, o custo aumentará, mas ainda assim será baixo comparado com outras arquiteturas.

Na arquitetura tradicional o custo são calculados de forma diferentes, já que não é necessário pagar por execução ou quantidade de dados armazenamentos, o custo se dá de acordo com dois fatores, o custo de energia da máquina, que pode ser desconsiderado caso a arquitetura seja rodada em um servidor na nuvem ou em um datacenter alugado e o custo da compra ou aluguel da máquina. Podemos dizer então que para a arquitetura tradicional o custo é apenas o de uso da máquina.

O custo de uso de uma máquina pode variar de acordo com sua configuração e método de compra/contratação. Normalmente para alugar uma máquina básica o custo será de pelo menos U$10,00, mas essa máquina será bem limitada e lenta. Nos testes feitos neste trabalho foram utilizados a máquina do autor, porém para o uso do mesmo é necessário mantê-la ligado 24 horas por dia e a deixa inutilizável para outras tarefas.

Para entender se vale a pena utilizar uma arquitetura tradicional, deve-se levar em conta como que a mesma vai ser executada, já que isso afetará o custo e performance e pode fazer que seus benefícios não sejam suficientes para que valha a pena utilizá-la. Por isso é importante analisar a arquitetura e informações antes de começar.

5 CONCLUSÃO

Este trabalho apresentou uma arquitetura de *Big Data* que tem como objetivo ser fácil e barata de se executar, utilizando as ferramentas e serviços da AWS e demonstrando métodos de coleta e análise de dados.

Os resultados obtidos neste trabalho apontam que utilizando de fontes de dados públicas é possível fazer a coleta e análise de dados de forma adequada, sendo que para uma arquitetura simples o custo de execução é baixo, sendo que o custo total de execução foi menor que R$1,00 ao mês e sua performance foi satisfatória, juntamente com a facilidade de implementação e manutenção.

O uso da AWS para montar a arquitetura se mostrou fácil e rápido comparado a arquitetura tradicional. A arquitetura apresentada neste trabalho apresenta uma grande redução de tempo de implementação, sendo que a arquitetura não trouxe nenhum malefício para a aplicação desse projeto, mas deve-se atentar que para outros casos os resultados podem variar e deve ser feito uma análise caso a caso antes de iniciar um projeto de *Big Data* para escolher a arquitetura ideal.

Para trabalhos futuros é possível expandir nos métodos de coletas de dados e uso de outras tecnologias para comparação, além de utilizar métodos de predição que possam alterar as necessidades da arquitetura, fazendo uma análise de como a arquitetura apresentada se comporta em casos diferentes e mais complexos. O avanço deste trabalho pode trazer benefícios em diversas áreas que utilizam de *Big Data* para predição e análise de dados.

REFERÊNCIAS

ALLAHBAKHSH, Mohammad *et al.* Quality control in crowdsourcing systems: Issues and directions. **IEEE Internet Computing**, IEEE, v. 17, n. 2, p. 76–81, 2013.

APACHE. **Apache Ecosystem**. Disponível em: https://www.apache.org/. Acesso em: 11 jun. 2022.

AWS. **Conceitos de Data Warehouse**. Disponível em: https://aws.amazon.com/pt/data-warehouse/. Acesso em: 9 jun. 2023.

AWS. **Definição de preço do Amazon S3**. Disponível em: https://aws.amazon.com/pt/s3/pricing/. Acesso em: 20 jun. 2023.

AWS. **Definição de preço do AWS Lambda**. Disponível em: https://aws.amazon.com/pt/lambda/pricing/. Acesso em: 16 jun. 2023.

BRASIL, Banco Central do. **Estatísticas de Meios de Pagamentos**. Disponível em: https://www.bcb.gov.br/estatisticas/spbadendos. Acesso em: 6 jun. 2022.

CAFARELLA, Michael J; HALEVY, Alon; KHOUSSAINOVA, Nodira. Data integration for the relational web. **Proceedings of the VLDB Endowment**, VLDB Endowment, v. 2, n. 1, p. 1090–1101, 2009.

CAMPBELL, Chris. Top five differences between data lakes and data warehouses. **White paper: BLUE GRANITE**, 2015.

CARLOS, Marcelo Aparecido; NOGUEIRA, Marcelo; MACHADO, Ricardo J. Analysis of dengue outbreaks using big data analytics and social networks. *In*: IEEE. 2017 4th International Conference on Systems and Informatics (ICSAI). [*S. l.: s. n.*], 2017. P. 1592–1597.

CHANDRA, Rakesh Vidya; VARANASI, Bala Subrahmanyam. **Python requests essentials**. [*S. l.*]: Packt Publishing Birmingham, UK, 2015.

DE VOS, Martine G *et al.* Open weather and climate science in the digital era. **Geoscience Communication**, Copernicus Publications Göttingen, Germany, v. 3, n. 2, p. 191–201, 2020.

DOAN, AnHai; HALEVY, Alon; IVES, Zachary. **Principles of data integration**. [*S. l.*]: Elsevier, 2012.

DRUMOND, Bruna *et al.* Dinâmica espaço-temporal da dengue no Distrito Federal, Brasil: ocorrência e permanência de epidemias. **Ciência & Saúde Coletiva**, SciELO Brasil, v. 25, p. 1641–1652, 2020.

FANG, Huang. Managing data lakes in big data era: What's a data lake and why has it became popular in data management ecosystem. *In*: IEEE. 2015 IEEE International

Conference on Cyber Technology in Automation, Control, and Intelligent Systems (CYBER). [*S. l.: s. n.*], 2015. P. 820–824.

FERREIRA, João *et al.* O processo etl em sistemas data warehouse*. In*: INFORUM. [*S. l.: s. n.*], 2010. P. 757–765.

FLUME. **Welcome to Apache Flume**. Disponível em: https://flume.apache.org/. Acesso em: 11 jun. 2022.

GARCIA, Marco. **Big Data: O que é, conceito e definição**. Disponível em: https://www.cetax.com.br/blog/big-data/. Acesso em: 11 jun. 2022.

GARCÍA, Salvador *et al.* Big data preprocessing: methods and prospects. **Big Data Analytics**, BioMed Central, v. 1, n. 1, p. 1–22, 2016.

HADOOP, Apache. **Apache Hadoop YARN**. Disponível em: https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html. Acesso em: 11 jun. 2022.

HADOOP, Apache. **HDFS Architecture Guide**. Disponível em: https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html. Acesso em: 11 jun. 2022.

HADOOP, Apache. **MapReduce Tutorial**. Disponível em: https://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html. Acesso em: 11 jun. 2022.

IBM. **What is HDFS?** Disponível em: https://www.ibm.com/topics/hdfs. Acesso em: 11 jun. 2022.

INMET. **Mapa de Estações**. Disponível em: https://mapas.inmet.gov.br/. Acesso em: 11 jun. 2022.

KHINE, Pwint Phyu; WANG, Zhao Shun. Data lake: a new ideology in big data era*. In*: EDP SCIENCES. ITM web of conferences. [*S. l.: s. n.*], 2018. v. 17, p. 03025.

LEE, Chung-Hong; YANG, Hsin-Chang; LIN, Shih-Jan. Incorporating big data and social sensors in a novel early warning system of dengue outbreaks*. In*: IEEE. 2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM). [*S. l.: s. n.*], 2015. P. 1428–1433.

LI, Guoliang *et al.* Crowdsourced data management: A survey. **IEEE Transactions on Knowledge and Data Engineering**, IEEE, v. 28, n. 9, p. 2296–2319, 2016.

LYDIA, E Laxmi; SWARUP, M Ben. Big data analysis using hadoop components like flume, mapreduce, pig and hive. **International Journal of Science, Engineering and Computer Technology**, Indian Association of Health, Research e Welfare, v. 5, n. 11, p. 390, 2015.

MAGALHÃES, Jorge Lima de *et al.* Big Data e a saúde negligenciada em dengue, zika e chicungunha: uma análise translacional da tríplice ameaça no século 21. **Ciência da Informação**, v. 45, n. 3, 2016.

MILOSLAVSKAYA, Natalia; TOLSTOY, Alexander. Application of big data, fast data, and data lake concepts to information security issues. *In*: IEEE. 2016 IEEE 4th International Conference on Future Internet of Things and Cloud Workshops (FiCloudW). [*S. l.: s. n.*], 2016. P. 148–153.

NASA. **POWER Manager API**. Disponível em: https://power.larc.nasa.gov/api/pages/?urls.primaryName=Manager. Acesso em: 10 mar. 2023.

NGUYEN, Giang *et al.* Machine learning and deep learning frameworks and libraries for large-scale data mining: a survey. **Artificial Intelligence Review**, Springer, v. 52, n. 1, p. 77–124, 2019.

OUSSOUS, Ahmed *et al.* Big Data technologies: A survey. **Journal of King Saud University-Computer and Information Sciences**, Elsevier, v. 30, n. 4, p. 431–448, 2018.

PARANÁ, Governo de. **Boletins da Dengue**. Disponível em: https://www.dengue.pr.gov.br/Pagina/Boletins-da-Dengue/. Acesso em: 11 jun. 2022.

PATKI, Neha; WEDGE, Roy; VEERAMACHANENI, Kalyan. The synthetic data vault. *In*: IEEE. 2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA). [*S. l.: s. n.*], 2016. P. 399–410.

PEDREGOSA, Fabian *et al.* Scikit-learn: Machine learning in Python. **the Journal of machine Learning research**, JMLR. org, v. 12, p. 2825–2830, 2011.

PEGADO, Marco Antonio Silva. Aplicação da abordagem Serverless no desenvolvimento de aplicações na nuvem: um estudo de caso com a solução AWS Lambda. Universidade Federal do Maranhão, 2021.

PELLE, István *et al.* Towards latency sensitive cloud native applications: A performance study on AWS. *In*: IEEE. 2019 IEEE 12th International Conference on Cloud Computing (CLOUD). [*S. l.: s. n.*], 2019. P. 272–280.

PISONI, Yasmin Oliveira; CÂMARA, Carlos Eduardo. Análise da Plataforma AWS Lambda em Ambientes de Nuvens Computacionais. **Revista de Ubiquidade**, v. 4, n. 2, p. 34–57, 2021.

ROH, Yuji; HEO, Geon; WHANG, Steven Euijong. A survey on data collection for machine learning: a big data-ai integration perspective. **IEEE Transactions on Knowledge and Data Engineering**, IEEE, v. 33, n. 4, p. 1328–1347, 2019.

SANTOS, Hugo Francisco Lisboa; COSTA, Pedro Vassalo Maia da; ARAUJO, Murilo Lamas Leandro *et al.* Que fonte de dados meteorolÓgicos utilizar no brasil? que incerteza esperar? uma comparaÇÃo entre diferentes abordagens e variadas fontes de dados. *In*: CONGRESSO Brasileiro de Energia Solar-CBENS. [*S. l.: s. n.*], 2020.

SANTOS, Nayara Rocha dos; COSTA, Adolpho Ramsés Maia; FEITOSA, Carlene Alves *et al.* A EVOLUÇÃO DE CASOS DE ARBOVIROSES DENGUE, CHIKUNGUNYA E ZIKA VÍRUS NO BRASIL ENTRE 2018 E 2020. **The Brazilian Journal of Infectious Diseases**, Elsevier, v. 26, p. 101956, 2022.

SARMA, Dhiman *et al.* Dengue Prediction using Machine Learning Algorithms. *In*: IEEE. 2020 IEEE 8th R10 Humanitarian Technology Conference (R10-HTC). [*S. l.: s. n.*], 2020. P. 1–6.

SBARSKI, Peter; KROONENBURG, Sam. **Serverless architectures on AWS: with examples using Aws Lambda**. [*S. l.*]: Simon e Schuster, 2017.

SCHULZE, Christoph *et al.* Model-based testing of NASA's OSAL API—An experience report. *In*: IEEE. 2013 IEEE 24th International Symposium on Software Reliability Engineering (ISSRE). [*S. l.: s. n.*], 2013. P. 300–309.

SOUZA SILVA, Geovanna Cristine de *et al.* Technologies to combat Aedes mosquitoes: a model based on Smart City. *In*: NURSING Informatics 2018. [*S. l.*]: IOS Press, 2018. P. 129–133.

STONEBRAKER, Michael; ILYAS, Ihab F *et al.* Data Integration: The Current Status and the Way Forward. **IEEE Data Eng. Bull.**, v. 41, n. 2, p. 3–9, 2018.

SUNDRAM, BM *et al.* Utilizing Artificial Intelligence as a Dengue Surveillance and Prediction Tool. **J Appl Bioinforma Comput Biol 8**, v. 1, p. 2, 2019.

TERRIZZANO, Ignacio G *et al.* Data Wrangling: The Challenging Yourney from the Wild to the Lake. *In*: CIDR. [*S. l.: s. n.*], 2015.

TIAN, Xinhui *et al.* Latency critical big data computing in finance. **The Journal of Finance and Data Science**, v. 1, n. 1, p. 33–41, 2015.

WHITE, Tom. **Hadoop: The definitive guide**. [*S. l.*]: "O'Reilly Media, Inc.", 2012.

YAKOUT, Mohamed *et al.* Infogather: entity augmentation and attribute discovery by holistic matching with web tables. *In*: PROCEEDINGS of the 2012 ACM SIGMOD International Conference on Management of Data. [*S. l.: s. n.*], 2012. P. 97–108.