Uma plataforma para integração de dados para cidades inteligentes

Murilo Borges Ribeiro

Dissertação apresentada ao
Instituto de Matemática e Estatística
da Universidade de São Paulo
para obtenção do título de
Mestre em Ciências

Programa: Ciência da Computação
Orientadora: Profª. Drª. Kelly Rosa Braghetto

São Paulo
Novembro de 2023

Uma plataforma para integração de dados para cidades inteligentes

Murilo Borges Ribeiro

Esta é a versão original da
dissertação elaborada pelo candidato
Murilo Borges Ribeiro, tal como
submetida à Comissão Julgadora.

Resumo

Murilo Borges Ribeiro. **Uma plataforma para integração de dados para cidades inteligentes**. Dissertação (Mestrado). Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2023.

A coleta, processamento e análise de dados provenientes de diversas fontes podem oferecer uma compreensão mais aprofundada das operações e necessidades urbanas. Contudo, criar soluções eficazes para explorar dados urbanos é desafiador devido ao grande volume, heterogeneidade e falta de acessibilidade e integração desses dados. Embora tenham sido abordados em diversas pesquisas, os sistemas de integração de dados para cidades inteligentes ainda enfrentam questões não resolvidas e apresentam oportunidades de aprimoramento. Entre as deficiências identificadas, destacam-se a falta de suporte adequado para o gerenciamento de metadados e a ausência de facilidades para a consulta de dados direcionada a usuários não especializados, dificultando a descoberta e reutilização de dados urbanos. Neste trabalho, fizemos uma revisão da literatura sobre sistemas de integração de dados para cidades inteligentes e, a partir dela, identificamos os principais requisitos funcionais e não funcionais desse tipo de sistema de software. Com base na revisão da literatura, propomos uma arquitetura de microsserviços para ingestão e integração física dos dados, para facilitar o desenvolvimento de plataformas de software que integram dados heterogêneos em ambientes urbanos inteligentes. A arquitetura proposta é uma extensão das arquiteturas encontradas na literatura e suporta a ingestão de dados, processamento, gerenciamento de metadados, processamento, análise e visualização de dados, ao mesmo tempo que fornece escalabilidade, disponibilidade, segurança e privacidade. A arquitetura apresenta ainda recursos exclusivos como: um único ponto de acesso a microsserviços por aplicativos externos; um serviço de autenticação e autorização de acesso; uma interface centralizadora dos serviços; a criação de novas coleções de dados com base nas existentes; e compatibilização de dados em coleções que sofreram alterações estruturais ou semânticas ao longo do tempo, utilizando o histórico de modificação de metadados e regras de mapeamento. O trabalho apresenta também diretrizes com base na *Cloud Evaluation Experiment Methodology* para avaliar o consumo de CPU, memória, latência, tempo de resposta e escalabilidade dos serviços que compõem a arquitetura proposta. Implementamos uma prova de conceito para os principais serviços da arquitetura (ingestão de dados, gerenciamento de metadados, consulta de dados e visualização de dados) e avaliamos o desempenho com base na metodologia proposta. Os experimentos mostraram que os serviços podem ser escalados horizontalmente para lidar com a demanda de uma cidade inteligente, além de manter tempos de resposta abaixo de dois segundos.

Palavras-chave: Cidades inteligentes. Integração de dados. Gerenciamento de dados. Microsserviços.

Abstract

Murilo Borges Ribeiro. **A platform for data integration for smart cities**. Thesis (Master's). Institute of Mathematics and Statistics, University of São Paulo, São Paulo, 2023.

The collection, processing, and analysis of data generated by varied sources can help us better understand the functioning and demands of the cities. However, developing efficient solutions to explore urban data is challenging due to the large volume, heterogeneity, and lack of accessibility and integration of this data. While data integration systems for smart cities have been the subject of various research efforts, there are still open issues and opportunities for improvement. Identified deficiencies include insufficient support for metadata management and a lack of facilities for user-friendly data queries, hindering the discovery and reuse of urban data. In this work, we conducted a literature review on data integration systems for smart cities to identify the main functional and non-functional requirements this kind of software system has. Building upon the literature review, we propose a microservices architecture for data ingestion and physical integration to facilitate the development of software platforms that integrate heterogeneous data in intelligent urban environments. The proposed architecture extends existing literature architectures, supporting data ingestion, processing, metadata management, analysis, and data visualization while providing scalability, availability, security, and privacy. It also features unique capabilities such as a single access point to microservices for external applications, an authentication and access authorization service, a service centralization interface, the creation of new data collections based on existing ones, and data compatibilization in collections that have undergone structural or semantic changes over time, using metadata modification history and mapping rules. We also present guidelines based on the Cloud Evaluation Experiment Methodology to assess CPU consumption, memory usage, latency, response time, and scalability of the proposed architecture's services. We have implemented a proof of concept for the main services of the architecture (data ingestion, metadata management, data querying, and data visualization) and evaluated their performance based on the proposed methodology. The experiments have shown that the services horizontally scale to handle the demands of a smart city while maintaining response times below two seconds.

Keywords: Smart Cities. Data Integration. Data Management. Microservices.

Lista de Abreviaturas

AL	Arquitetura Lambda
API	Interface de Programação de Aplicação (*Application Programming Interface*)
CSV	*Comma Separated Values*
DL	*Data Lake*
DW	*Data Warehouse*
HDFS	Sistema de Arquivos Distribuídos Hadoop (*Hadoop Distributed File System*)
JSON	Notação de Objetos JavaScript (*JavaScript Object Notation*)
OBDA	Acesso a Dados Baseado em Ontologias (*Ontology-Based Data Access*)
SAD	Sistema de Arquivos Distribuído
SGBDD	Sistema de Gerenciamento de Bancos de Dados Distribuído
SGBDF	Sistema de Gerenciamento de Bancos de Dados Federado
SQL	Linguagem de Consulta Estruturada (*Structured Query Language*)
XML	Linguagem de Marcação Extensível (*Extensible Markup Language*)

Lista de Figuras

Lista de Tabelas

Sumário

Capítulo 1

Introdução

O conceito de cidades inteligentes surgiu da ideia de aplicar a tecnologia da informação e comunicação para tornar as cidades mais eficientes. Não existe uma definição consensual para cidades inteligentes, mas uma de suas principais características é o uso da tecnologia para promover a sustentabilidade ambiental e melhorar a qualidade de vida dos cidadãos (Albino *et al.*, 2015; Al Nuaimi *et al.*, 2015).

Os dados urbanos, ou seja, os dados coletados nas cidades, têm uma importância primordial na construção de cidades inteligentes. A crescente disponibilidade de dispositivos eletrônicos com capacidade de sensoriamento e poder computacional, capazes de receber e enviar informações, faz com que uma grande quantidade de dados de diferentes fontes e em diferentes estruturas sejam continuamente produzidos nas cidades. As cidades também acumulam dados gerados por entidades governamentais, cidadãos e sistemas.

Zheng *et al.* (2014) classificam os dados urbanos em cinco categorias: (i) dados de mobilidade urbana, como dados de tráfego, deslocamento e telefonia móvel; (ii) dados geográficos, como informações sobre a rede de transporte e malha rodoviária; (iii) dados de mídia social, como texto, fotos e vídeos; (iv) dados ambientais, como clima e consumo de energia; e (v) dados de outras fontes não necessariamente relacionadas ao contexto urbano, como serviço público, saúde e economia.

A coleta, limpeza, integração, transformação e análise de dados de fontes distintas permitem um melhor entendimento das deficiências das cidades, auxiliando a tomada de decisão baseada em evidências e o desenvolvimento de políticas públicas visando ao melhor aproveitamento dos recursos disponíveis e a melhoria na qualidade de vida da população (Khan *et al.*, 2013; Raghavan *et al.*, 2020). Por exemplo, o cruzamento de registros da secretaria municipal de saúde com dados sociodemográficos, meteorológicos e de redes sociais pode ser utilizado para monitorar e prevenir a evolução de doenças endêmicas como a Dengue. Mas o desenvolvimento de soluções para explorar dados urbanos encontra dificuldades na falta de acessibilidade e integração dos dados. Isso porque muitos sistemas são construídos em silos, ou seja, são fechados e desenvolvidos para uma necessidade específica.

O grande volume de dados é um desafio adicional, pois aumenta a complexidade no armazenamento, processamento e consulta dos dados. Existem diversas ferramentas, tanto

de software proprietário quanto de software livre, que podem ser combinadas para permitir o armazenamento, descoberta, processamento, consulta e análise de grandes volumes de dados de forma eficiente, mas que exigem conhecimentos específicos de seus usuários e sozinhas não resolvem o problema da integração de dados em cidades inteligentes.

Sistemas de integração de dados para cidades inteligentes já foram objeto de vários trabalhos de pesquisa (Psyllidis *et al.*, 2015; Consoli *et al.*, 2015; Cheng *et al.*, 2015; Rathore *et al.*, 2016; Hashem *et al.*, 2016; Costa e Santos, 2017; Mehmood *et al.*, 2019). Apesar desses trabalhos apresentarem algumas abordagens para as diferentes etapas da integração de dados (como ingestão, processamento, armazenamento, análise e visualização de dados), ainda há questões em aberto e espaço para melhorias em tais sistemas. Exemplos de deficiências encontradas nesses trabalhos são o suporte insuficiente para gerenciamento de metadados e a falta de facilidades para a consulta a dados voltadas a usuários não especialistas, o que dificulta a descoberta e a reutilização de dados urbanos.

Objetivos

Este trabalho tem como objetivo propor uma arquitetura para orientar o desenvolvimento de plataformas de software de integração de dados heterogêneos em cidades inteligentes que facilite a descoberta, utilização, análise, consulta e visualização de dados. Para atingir este objetivo este projeto de mestrado se propôs a responder às seguintes questões de pesquisa:

1. Quais são os principais desafios e problemas identificados pelos pesquisadores para integração de dados em cidades inteligentes?
2. Quais são os requisitos funcionais e não funcionais de uma plataforma de software para integração de dados em cidades inteligentes?
3. Quais métodos para integração de dados melhor se aplicam a cidades inteligentes?

Para responder as questões de pesquisa fizemos uma revisão da literatura a partir de 2015. Analisamos os artigos para identificar os requisitos funcionais, não funcionais e os desafios da integração de dados em cidades inteligentes. Com base nas informações encontradas, propomos uma arquitetura de microsserviços para orientar o desenvolvimento de plataformas de software para integrar dados heterogêneos de cidades inteligentes.

A arquitetura de microsserviços apresentada neste trabalho apoia a ingestão de dados, processamento, gerenciamento de metadados, processamento, análise e visualização de dados, ao mesmo tempo que fornece escalabilidade, disponibilidade, segurança e privacidade.

Apresentamos também diretrizes para avaliar o desempenho de sistemas que implementam a arquitetura proposta. As diretrizes seguem a *Cloud Evaluation Experiment Methodology* (CEEM) (Li, OBrien e Zhang, 2013), usada para avaliar sistematicamente o desempenho de serviços em nuvem por meio de experimentos.

Para avaliar e validar a arquitetura proposta, foi implementada uma prova de conceito de código aberto e foram realizados experimentos baseados na metodologia CEEM para avaliar a latência, tempo de resposta e escalabilidade dos serviços implementados. Os

resultados mostram que os microsserviços escalam automaticamente, ou seja, aumentam ou diminuem o número de instâncias conforme a variação da carga de entrada. Além disso, a latência e tempo de respostas medidos durante os experimentos se mantiveram dentro dos patamares esperados.

Contribuições

As principais contribuições deste trabalho são:

- Uma arquitetura de microsserviços para o desenvolvimento de plataformas de software de integração de dados heterogêneos em cidades inteligentes.
- Diretrizes com base na metodologia CEEM para avaliar o desempenho dos sistemas que implementam a arquitetura proposta.
- Implementação de uma prova de conceito de código aberto para os microsserviços de ingestão de dados, gerenciamento de metadados e consulta de dados. O código desenvolvido pode ser encontrado no repositório *online* Gitlab https://gitlab.com/interscity/data-integration.
- Desenho e execução de experimentos baseados nas diretrizes propostas para avaliar o consumo de CPU, memória, latência, tempo de resposta e escalabilidade dos serviços implementados.
- Publicação em 2021 no Anais do XXXVI Simpósio Brasileiro de Bancos de Dados, intitulada: *A Data Integration Architecture for Smart Cities* (Ribeiro e Braghetto, 2021).
- Publicação em 2022 no *Journal of Information and Data Management*, intitulada: *A Scalable Data Integration Architecture for Smart Cities: Implementation and Evaluation* (Ribeiro e Braghetto, 2022).

Organização da Dissertação

Esta dissertação está estruturada em seis capítulos além da Introdução. O Capítulo 2 introduz o referencial teórico necessário para o entendimento deste trabalho. O Capítulo 3 discute os trabalhos relacionados. O Capítulo 5 apresenta a arquitetura de integração de dados proposta neste trabalho. O Capítulo 6 apresenta o protótipo desenvolvido. O Capítulo 7 apresenta a avaliação de desempenho do propótipo desenvolvido, enquanto o Capítulo 8 apresenta as considerações finais da dissertação e propostas para trabalhos futuros.

Capítulo 2

Conceitos

Conforme mencionado no Capítulo 1, este trabalho tem como objetivo facilitar a integração e uso de dados de cidades inteligentes. Portanto, conceitos relacionados a cidades inteligentes; formatos de dados; *Big Data*; e transporte, armazenamento e integração de dados são essenciais para o entendimento deste trabalho. As seções a seguir introduzem esses conceitos.

2.1 Cidades Inteligentes

O conceito de cidades inteligentes originou-se da aplicação das tecnologias da informação e comunicação (TIC) para tornar as cidades mais eficientes. Não existe uma definição clara e consensual. No início, o termo se referia principalmente ao seu contexto tecnológico mas passou a centrar-se principalmente nas pessoas, com as TIC a funcionarem como ferramentas para impulsionar o envolvimento dos cidadãos. Podemos dizer que uma das principais características de cidades inteligentes é a utilização da tecnologia para coletar e analisar dados de sensores, dispositivos e sistemas de informação com objetivo de promover a sustentabilidade ambiental, a melhora nos serviços prestados e a melhora na qualidade de vida dos cidadãos (Albino *et al.*, 2015; Al Nuaimi *et al.*, 2015).

A arquitetura de uma cidade inteligente pode ser dividida em quatro camadas que interagem entre si (B. N. Silva *et al.*, 2018; Santana *et al.*, 2017). Conforme exibido na Figura 2.1, a camada mais baixa, a de sensoreamento, se refere às fontes de dados disponíveis, detecção e coleta dos dados. A segunda camada, de transmissão, é responsável pela transmissão dos dados utilizando tecnologia sem fio (3G, 4G, 5G, *Bluetooth*), com fio ou via satélite. A terceira camada, de gerenciamento de dados, é considerada o cérebro de qualquer cidade inteligente, pois é responsável pelo armazenamento, organização, manipulação, fusão e análise de dados, gerenciamento de eventos e tomada de decisão. A quarta camada, de aplicações, é responsável por disponibilizar os serviços aos cidadãos com base na camada de gerenciamento dos dados.

Para atingir o patamar de eficiência desejado, é necessário monitorar a infraestrutura e os recursos disponíveis e compreender a dinâmica da cidade, que envolve entender a gestão dos recursos públicos, o comportamento dos cidadãos e suas interações com os

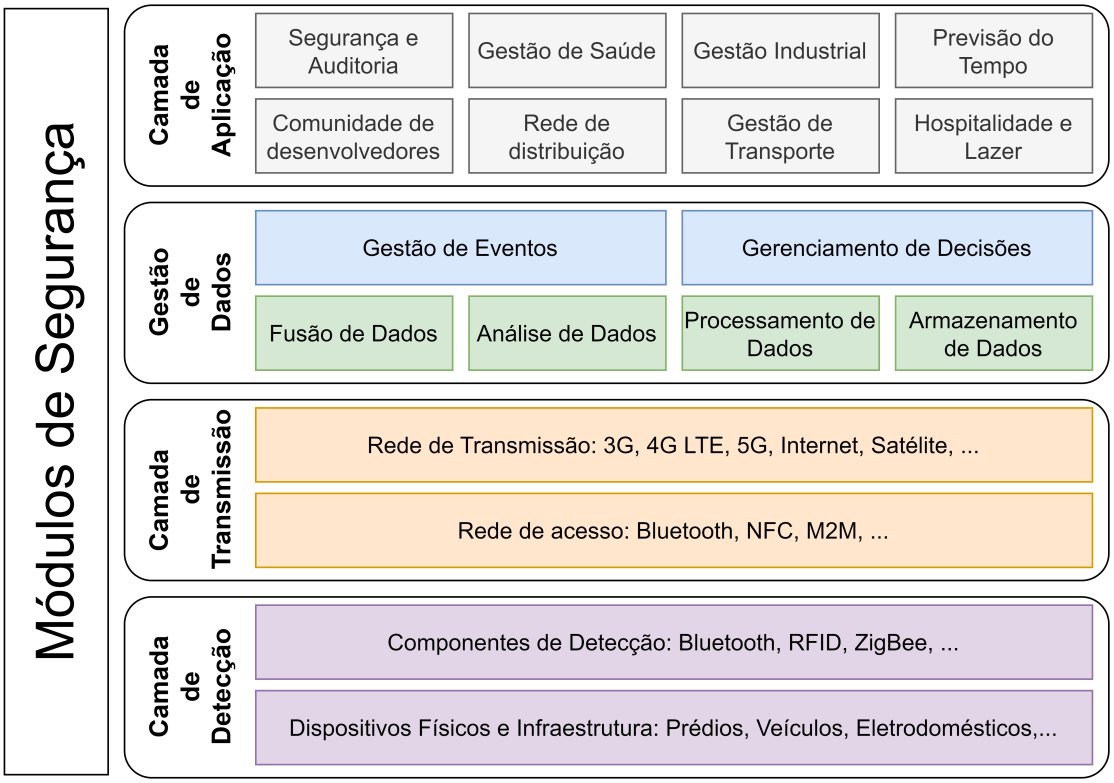

Figura 2.1: *Arquitetura de camadas para cidades inteligentes. Adaptada de (B. N. Silva et al., 2018)*

serviços disponíveis (T. H. Silva *et al.*, 2014).

Sistemas governamentais de diversas áreas como saúde, educação, transporte, segurança e economia geram grande quantidade de dados. Redes de sensores sem fio são usadas em coleta de dados ambientais, rastreamento, monitoramento de segurança, dentre outros, mas possuem limitações na cobertura de grandes áreas, como as grandes metrópoles. Uma abordagem complementar é o sensoriamento participativo, em que pessoas compartilham voluntariamente informação contextual e/ou tornam seus dados sensoreados disponíveis (T. H. Silva *et al.*, 2014). Tais dados podem ser analisados a fim de compreender a utilização e aplicação dos recursos públicos e as necessidades da população, possibilitando o desenvolvimento de políticas públicas e melhora na gestão dos recursos disponíveis, com o objetivo de tornar o ambiente urbano mais eficiente, sustentável e com mais qualidade de vida, assim reduzindo os impactos do crescimento populacional sobre a infraestrutura e serviços públicos.

2.2 Formato dos Dados

Os dados de cidades inteligentes podem estar em diversos formatos, tais como CSV (*Comma-Separated Values*), XML (*Extensible Markup Language*), JSON (*JavaScript Object Notation*), GeoJSON (*Geographic JSON*), *shapefile*, arquivos de texto, imagens e áudios. Eles podem ser classificados como dados estruturados, semiestruturados e não estruturados (Al Nuaimi *et al.*, 2015; Gandomi e Haider, 2015).

Dados estruturados apresentam um esquema pré-definido com estrutura homogênea em termos de atributos e tipos. Tais características não permitem mudanças dinâmicas na estrutura dos dados. Bancos de dados relacionais são um exemplo de dados estruturados. A Listagem 2.1 apresenta um exemplo de comando em SQL (*Structured Query Language*) para definição do esquema da tabela AIH (autorização de internação hospitalar) para um banco de dados relacional. No exemplo, a tabela AIH armazena dados de internações hospitalares e é composta por um identificador sequencial único (id), a latitude e longitude do estabelecimento hospitalar, o número do cartão nacional de saúde (cns) do paciente e o código internacional de doença (cid) que representa a causa primária da internação.

```sql
CREATE TABLE AIH (
   id SERIAL PRIMARY KEY,
   latitude DECIMAL(10, 8),
   longitude DECIMAL(10, 8),
   cns VARCHAR(15),
   cid VARCHAR(4)
);
```

Listagem 2.1: *Exemplo de criação de um esquema de tabela em um banco de dados relacional utilizando SQL*

Dados semiestruturados possuem estrutura irregular, ou seja, cada ocorrência pode ser heterogênea em termos de estrutura – tanto nos atributos quanto nos seus tipos –, portanto, sua estrutura é evolutiva. Além disso, a estrutura de dados semiestruturados fica embutida na própria descrição dos dados, sendo necessário identificá-la e extraí-la a partir dos dados. JSON, CSV e XML são exemplos de arquivos de textos planos bastante usados para o compartilhamento ou troca de dados semiestruturados na Web. As listagens 2.2 e 2.3 mostram exemplos de dados nos formatos CSV e JSON, respectivamente.

```csv
id,latitude,longitude,cns,cid
78,-23.5587614,-46.733619,119919919919191,B34.2
79,-23.5587614,-46.733619,119919919919192,J15.7
101,-22.818439,-47.0669093,119919919919192,J15.7
```

Listagem 2.2: *Exemplo de dados no formato CSV*

```json
{
  "AIH": [
    {
      "id": 78,
      "latitude": -23.5587614,
      "longitude": -46.733619,
      "cns": "119919919919191",
      "cid": "B34.2"
    },
    {
      "id": 79,
      "latitude": -23.5587614,
      "longitude": -46.733619,
      "cns": "119919919919192",
      "cid": "J15.7"
    },
    {
      "id": 101,
```

```
19         "latitude": -22.818439,
20         "longitude": -47.0669093,
21         "cns": "119919919919192",
22         "cid": "O80.1",
23         "n_nasc_vivos" : "1"
24       }
25     ]
26   }
```

Listagem 2.3: *Exemplo de dados no formato JSON*

Dados não estruturados não apresentam marcações ou metadados para organizá-los, portanto, geralmente é necessário conhecer o contexto para conseguir analisá-los. Um exemplo de contexto seria a origem dos dados. Em geral, para analisar esse tipo de dado, há um custo adicional devido à necessidade de utilizar técnicas de extração e transformação. Arquivos de texto, imagens e dados de mídias sociais são exemplos de dados não estruturados. A Listagem 2.4 apresenta um exemplo de dados não estruturados coletado do Twitter.

```
1   Na Rádio USP, a pesquisadora Natália Pasternak comenta o relaxamento do
        isolamento social e alerta para o aumento de casos de covid-19 na Europa
```

Listagem 2.4: *Exemplo de dados não estruturado:* tweet *publicado por* @usponline *(2020)*

2.3 *Big Data* e Cidades Inteligentes

O crescente volume de dados gerado nas cidades por diferentes fontes, em diferentes formatos (dados estruturados, semiestruturados e não estruturados) e seus valores econômicos e sociais permitem que estes sejam classificados como *Big Data*. Portanto, um sistema que lida com dados urbanos deve considerar os Vs do *Big Data*. São eles (Fan e Bifet, 2013):

1. Volume: refere-se ao tamanho dos dados produzidos por todas as fontes;
2. Velocidade: refere-se à velocidade em que os dados são produzidos, armazenados, processados e analisados;
3. Variedade: refere-se à diversidade das fontes e aos tipos, que podem ser estruturados, semiestruturados ou não estruturados;
4. Veracidade: refere-se à relevância, precisão e qualidade dos dados capturados;
5. Valor: refere-se à possibilidade de o *Big Data* oferecer vantagem ao negócio por meio da coleta, gestão e análise dos dados.

Aplicações que lidam com *Big Data* em cidades inteligentes podem ser classificadas em dois tipos: em tempo real e *offline*. Aplicações em tempo real se baseiam em entrada instantânea e análise rápida de dados para chegar a uma decisão ou ação dentro de um curto e específico espaço de tempo. Já as aplicações *offline* oferecem recursos para análises retrospectivas que podem afetar a maioria dos ou todos os dados (Mohamed e Al-Jaroodi, 2014; Al Nuaimi *et al.*, 2015).

Al Nuaimi *et al.* (2015) identificaram na literatura seis principais desafios para implementação de sistemas no contexto de cidades inteligentes:

- Fonte e formato dos dados: os dados são gerados por diferentes fontes e em diferentes formatos. Muitos formatos não são estruturados, como imagens, áudio, vídeos e textos;
- Compartilhamento de dados e informações: cada esfera do governo possui seu próprio conjunto de dados, seja ele confidencial ou público. Alguns dados podem conter condições de privacidade específicas, dificultando o compartilhamento entre diferentes entidades. O desafio é garantir os direitos de privacidade dos dados;
- Qualidade dos dados: como dito anteriormente, os dados podem ser gerados por diferentes fontes, em diferentes estruturas e por diferentes pessoas e sistemas. Consequentemente, existe maior probabilidade de ocorrer problemas de consistência, heterogeneidade e disparidade, gerando incerteza e falta de confiabilidade. Por exemplo, dados de sensoriamento coletados por terceiros podem ter sido produzidos por sensores com defeito, calibrados incorretamente ou além da vida útil;
- Segurança e privacidade: muitos dados podem conter informações confidenciais relacionadas ao governo e às pessoas, como registros médicos e financeiros. Portanto, é necessário garantir que existam diferentes níveis de acesso e mecanismos de segurança para inibir o uso não autorizado e ataques maliciosos;
- Custo: a forma como as autoridades públicas implementam novas soluções pode afetar negativamente as pessoas. Por exemplo, um sistema de redução de perda de água potável utilizando novos sistemas e sensores para monitorar a rede de distribuição leva à criação de um sistema inteligente de água potável. No entanto, os custos de implantação e manutenção podem ser altos, impactando negativamente a população;
- População da cidade inteligente: as pessoas afetam e são afetadas pelas aplicações inteligentes. O tamanho da população da cidade tem um grande efeito no tamanho do *Big Data*. À medida que a população cresce, o volume de dados também cresce. Portanto, é necessário desenvolver e implantar aplicações e sistemas para tratar de forma inteligente o rápido crescimento do *Big Data* para gerar melhores resultados.

De acordo com Marz e Warren (2015), um sistema que lida com *Big Data* precisa ter baixa latência, ser tolerante a falhas, ser escalável para manter o desempenho mesmo com o aumento da carga de processamento, flexível para que novas funcionalidades possam ser adicionadas no futuro e genérico para possibilitar que aplicações possam utilizar o sistema.

2.3.1 Arquiteturas para Processamento de *Big Data*

Na literatura, duas arquiteturas de propósito geral para processamento de dados em lote e em tempo real simultaneamente se destacam: a Lambda (Marz, 2011) e a Kappa (Kreps, 2014).

O termo Arquitetura Lambda (AL) foi criado por Marz (2011) para um modelo genérico,

escalável e com processamento de dados tolerante a falhas de hardware e erros humanos, capaz de atender a uma ampla gama de cargas de trabalho e casos de uso para os quais são necessárias leituras e atualizações de dados de baixa latência.

Como pode ser visto na Figura 2.2, a AL é composta por três camadas: a camada de processamento de dados em lote (*batch layer*), a camada de processamento de dados em tempo real (*speed layer*) e a camada de serviço (*serving layer*). Todos os dados que entram no sistema são imediatamente enviados para as camadas de lote e tempo real. A camada de lote é responsável por armazenar o dado bruto, utilizado para processamento posterior de uma grande quantidade de dados. Este processamento ocorre com alta latência, já que o volume de dados nesta camada é muito grande. Após o processamento em lote, os resultados são condensados em visualizações indexadas na camada de serviço, para que o resultado do processamento dos dados em lote possa ser consultado com baixa latência.

A camada de tempo real processa os dados assim que estão disponíveis, para permitir a geração de visualizações próximas do tempo real para os dados mais recentes. Portanto, ela compensa a alta latência na camada de lote. Os resultados do processamento são condensados em visualizações de tempo real. As consultas realizadas em sistemas de *Big Data* com AL são respondidas pela junção dos resultados da consulta realizada nas camadas de tempo real e lote (Marz, 2011; Marz e Warren, 2015).

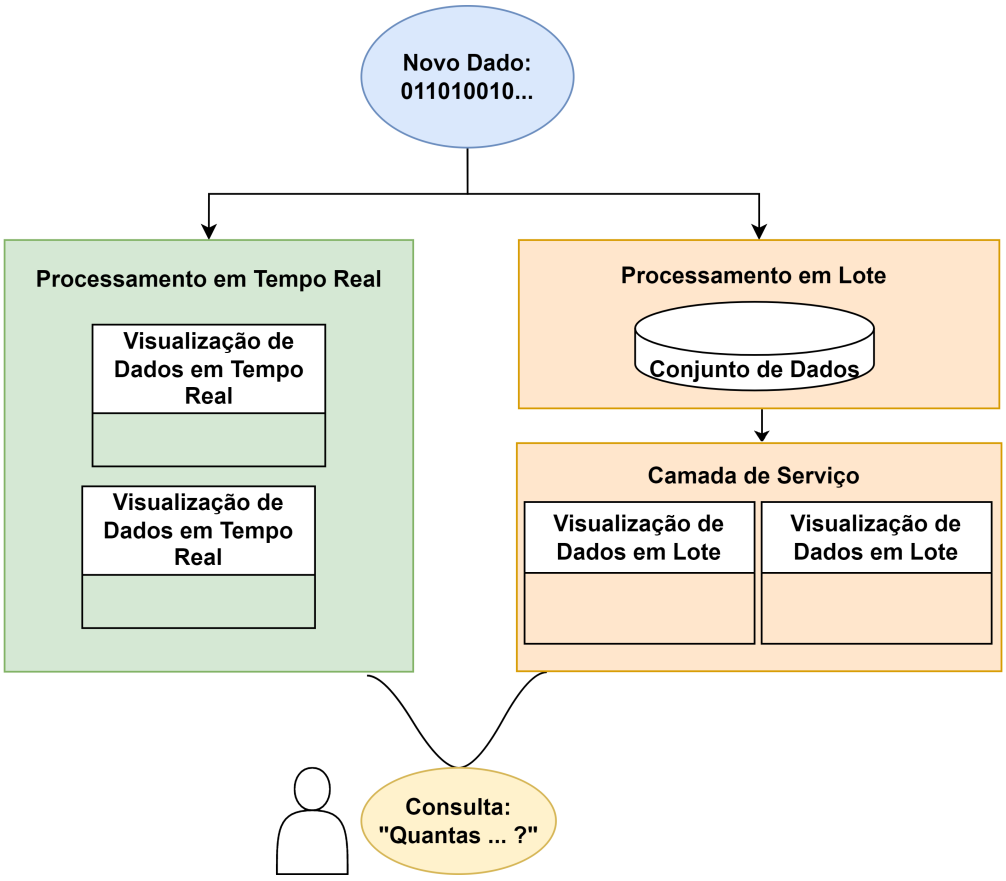

Figura 2.2: *Arquitetura Lambda. Adaptada de (Marz e Warren, 2015)*

A arquitetura Kappa surgiu em 2014 como uma simplificação da AL e se baseia em

quatro princípios: (i) todo o processamento deve ser realizado em tempo real; (ii) os dados devem ser imutáveis; (iii) deve ser utilizado somente um arcabouço para processamento; e (iv) deve existir a possibilidade de reprocessamento dos dados históricos (Kreps, 2014).

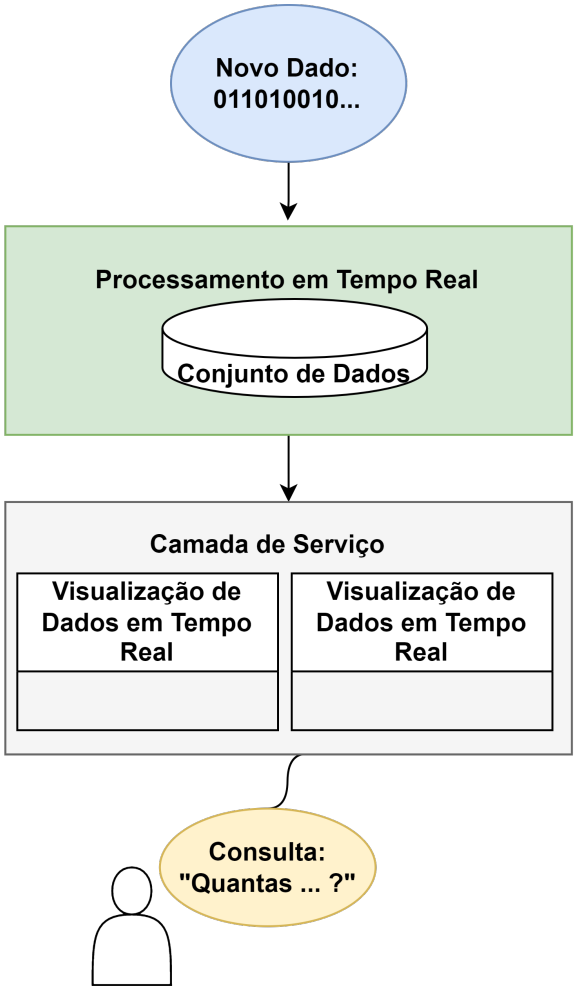

Figura 2.3: *Arquitetura Kappa. Adaptada de (Kreps, 2014)*

Como pode ser visto na Figura 2.3, a arquitetura Kappa apresenta uma camada única de processamento de dados. Isso porque a camada de processamento em tempo real também consegue processar lotes. Os dados originais são armazenados sem sofrer modificações, ou seja, apesar de poderem ser processados e transformados, os originais são imutáveis. A camada de serviço é responsável por possibilitar a consulta com baixa latência (Kreps, 2014).

Em arquiteturas de análise e processamento de mensagens em tempo real, as camadas de transporte de dados e armazenamento de dados são fundamentais, pois os dados devem estar disponíveis para consulta e processamento em tempo próximo do real. Na sequência, são apresentados alguns conceitos relacionados ao transporte de mensagens e ao armazenamento de dados.

2.3.2 Transporte de Mensagens

A camada de transporte de mensagens é responsável por transmitir mensagens entre produtores e consumidores. Dois modelos são bastante utilizados: filas de mensagens e *brokers* de mensagens. No primeiro modelo, a fila recebe as mensagens e garante que elas serão consumidas por apenas um consumidor. A ordem de processamento da fila é definida de acordo com a ordem de armazenamento ou de acordo com a prioridade definida na criação da fila. O algoritmo para escolha do consumidor é definido durante a implementação da fila. Após ser recebida e processada pelo consumidor, a mensagem é removida da fila. Alguns sistemas de enfileiramento de mensagens oferecem recursos para entrega assíncrona de mensagens, porém, a solução não é escalável quando a fila possui muitos consumidores devido às operações adicionais necessárias para garantir a transação e a ordem de processamento (Eugster *et al.*, 2003).

O segundo modelo, *brokers* de mensagem, é uma extensão da fila de mensagens. *Brokers* conhecem as mensagens que passam por eles e permitem a tradução do protocolo do remetente para o seu respectivo receptor, desacoplando o produtor do consumidor. Podem utilizar o padrão publicar-assinar (*publish-subscribe*) para realizar o roteamento das mensagens e podem armazenar mensagens por um determinado intervalo de tempo, portanto, o consumidor não precisa ler a mensagem imediatamente (Dobbelaere e Esmaili, 2017).

O padrão de troca de mensagens publicar-assinar tem como objetivo principal desassociar o produtor do assinante para permitir a criação de sistemas escaláveis e com baixo acoplamento. Eugster *et al.* (2003) dividiu o desacoplamento fornecido por este padrão em três dimensões:

- Desacoplamento de entidades: publicador e assinante não precisam estar cientes um do outro;
- Desacoplamento de tempo: o assinante não precisa ler a mensagem imediatamente, ou seja, o publicador pode publicar eventos enquanto o assinante está desconectado e o assinante pode ser notificado sobre a ocorrência de algum evento enquanto o publicador do evento está desconectado;
- Desacoplamento de sincronização: os publicadores não são bloqueados durante a produção de eventos e os assinantes podem ser notificados de forma assíncrona sobre a ocorrência de um evento durante a execução de alguma atividade simultânea.

Outra funcionalidade importante do padrão publicar-assinar é a lógica de roteamento (também conhecida como modelo de assinatura), que decide se e onde uma mensagem publicada vai para o consumidor. As duas principais lógicas de roteamento são (Eugster *et al.*, 2003):

- Assinatura baseada em tópicos: o publicador adiciona tópicos (etiquetas) de forma estática às mensagens publicadas. O assinante utiliza tais etiquetas para filtrar as mensagens que deseja receber;
- Assinatura baseada em conteúdo: o produtor não precisa marcar explicitamente a mensagem com o contexto de roteamento. Todos os dados e metadados da mensagem

podem ser usados na condição de filtragem. Os consumidores se inscrevem especificando filtros. A utilização de filtros complexos aumenta o custo de processamento.

Além das funcionalidades mencionadas, o padrão publicar-assinar garante a entrega da mensagem. As três possibilidades de garantia de entrega mais comuns são: no máximo, uma vez, ou seja, garante a não duplicação, mas pode ocorrer perda em caso de falha; pelo menos uma vez, ou seja, garante que não haverá perdas, mas pode causar o envio de pacotes duplicados e possivelmente fora de ordem; exatamente uma vez, ou seja, garante que não haverá perda e nem duplicação, o que requer uma confirmação nas duas pontas, aumentando o custo computacional. Também é possível garantir a ordem da entrega. As três possibilidades mais comuns ordenadas do menor custo computacional para o maior são: sem ordem, ordenação por partição (unidade de paralelismo definida na criação do tópico) e ordenação global. O padrão também garante a escalabilidade em várias dimensões, por exemplo: consumidores, produtores, tópicos e mensagens (Eugster *et al.*, 2003).

2.3.3 Armazenamento de Dados

De acordo com Mazumdar *et al.* (2019), os sistemas gerenciadores de dados da atualidade, como os sistemas de arquivos distribuídos (SADs), sistemas de armazenamento de objetos e sistemas gerenciadores de bancos de dados distribuídos (SGBDDs), precisam lidar com dados heterogêneos e oferecer recursos para atender, no mínimo, aos seguintes requisitos não funcionais de armazenamento, gerenciamento e acesso de *Big Data*:

- Desempenho: ter uma alta taxa de transferência e baixa latência;
- Escalabilidade: aumentar ou diminuir os recursos computacionais de acordo com a necessidade do sistema. A escalabilidade pode ser vertical, ou seja, adicionando mais recursos a uma máquina, ou horizontal, quando adiciona mais máquinas ao conjunto de máquinas do sistema;
- Elasticidade: ajustar os recursos computacionais à carga de trabalho sem a necessidade de interromper o serviço;
- Disponibilidade: manter o sistema operacional e acessível quando necessário. A sobrecarga no sistema pode aumentar a latência ou até mesmo fazer com que o sistema não trate as solicitações, caso em que é necessário utilizar da escalabilidade e elasticidade. Outro problema que pode afetar a disponibilidade é a falha em recursos (por exemplo, a falha de hardware) e, neste caso, é utilizada a replicação dos dados para evitar indisponibilidade;
- Consistência: prover a todos os usuários uma mesma visão sobre um mesmo item de dados;
- Processamento de *Big Data*: fornecer *drivers* nativos para o arcabouço de processamento de *Big Data*, permitindo automatizar o processamento e transformação dos dados.

Sistemas de arquivos distribuídos (SADs) são sistemas de arquivos nos quais os arquivos armazenados estão espalhados em diferentes hardwares interconectados por uma rede. SADs podem ser classificados em três modelos: cliente-servidor, distribuído e simétrico. O

modelo distribuído é o mais utilizado em *Big Data*, pois permite armazenar uma grande quantidade de dados particionados em vários servidores para acesso paralelo. Além disso, é tolerante a falhas, pois permite a hospedagem de réplicas. O sistema de arquivos distribuídos Hadoop (HDFS, de *Hadoop Distributed File System*) é o SAD de software livre mais conhecido (Mazumdar *et al.*, 2019).

Sistemas de armazenamento de objetos são projetados para lidar com volumes massivos de dados não estruturados, como imagens, vídeos e documentos. Os dados são organizados como objetos, cada um com seu identificador único (URL ou URI), ou seja, não existe uma hierarquia de pastas ou subpastas. Cada objeto possui o seu metadado que descreve informações como o tipo de conteúdo, autor, data de criação, tamanho e permissões de acesso. Esses metadados facilitam a busca e organização dos dados. *MinIo* (2023), *Amazon S3* (2023), *Google Cloud Storage* (2023) e *Microsoft Azure Blob Storage* (2023) são exemplos de sistemas de armazenamento de objetos.

Sistemas de gerenciamento de bancos de dados distribuídos (SGBDDs) são sistemas que gerenciam e armazenam dados em múltiplos servidores ou dispositivos que podem estar geograficamente separados e interconectados por redes. SGBDDs NoSQL são uma categoria de sistemas de gerenciamento de bancos de dados projetados para armazenar e gerenciar dados não relacionais ou semiestruturados. Os sistemas NoSQL tem como prioridade a flexibilidade na modelagem e consultas dos dados, a alta disponibilidade, escalabilidade e o alto de desempenho. Os modelos de dados implementados pelos sistemas NoSQL podem ser divididos em quatro categorias principais (Elmasri e Navathe, 2010):

1. Documentos: Armazenam dados em documentos semiestruturados, como JSON ou XML. Exemplos incluem *MongoDB* (2023) e *Couchbase* (2023).
2. Chave-Valor: Armazenam pares chave-valor, sendo eficazes para armazenamento em *cache* e recuperação rápida. Exemplos incluem *Redis* (2023) e *Amazon DynamoDB* (2023).
3. Colunas ou Colunas Largas: Armazenam dados em famílias de colunas, em vez de linhas, o que é eficaz para análises agregadas e consultas complexas. Exemplos incluem *Cassandra* (2023) e *Apache HBase* (2023).
4. Grafo: Modelam dados como nós e arestas em um grafo, sendo eficazes para representar dados com muitos relacionamentos e apoiar consultas que percorrem o grafo de diferentes formas. Exemplos incluem *Neo4j* (2023) e *Amazon Neptune* (2023).

Os SGBDDs NoSQL devem ser escolhidos com base nas necessidades específicas do projeto, considerando os requisitos de desempenho, escalabilidade e flexibilidade de esquema. Cada categoria de SGBDD NoSQL tem seus próprios pontos fortes e casos de uso ideais.

2.4 Integração de Dados

A integração de dados é o problema de combinar diferentes coleções de dados para fornecer uma visão unificada deles (Jarke *et al.*, 2013). Este problema não é novo, mas a integração de dados em *Big Data* apresenta novos desafios pois os dados são de fontes

distintas com heterogeneidade de formato e estrutura e o volume de dados é muito grande. Além disso, os dados coletados raramente atendem a todos os padrões de qualidade exigidos em um sistema ideal: precisão, consistência, integridade e atualidade (Kadadi *et al.*, 2014; Karkouch *et al.*, 2016).

Existem diferentes estratégias de integração de dados, que enfocam diferentes aspectos do processo. As seções seguintes apresentarão algumas estratégias de integração de dados usadas no contexto de cidades inteligentes.

2.4.1 Sistema de *Middleware*

Um sistema de *middleware* para integração de dados é um software que tem como objetivo facilitar a comunicação e a troca de dados entre diferentes bancos de dados, sistemas ou aplicações. O *middleware* atua como uma camada intermediária que permite a interoperabilidade entre as diferentes fontes de dados. Assim, o usuário utiliza apenas o *middleware* para acesso aos dados de diferentes origens sem precisar conhecer detalhes das fontes de dados (Ritter, 1998).

De acordo com Haas *et al.* (1997), na construção de um sistema de *middleware* para integração de fontes de dados heterogêneas, deve ser especificado um esquema integrador, uma linguagem de consulta e mecanismos para gerência, otimização e execução de consultas. Existem duas abordagens para integração de dados utilizando sistemas de *middleware*: mediadores e sistemas de gerenciamento de bancos de dados federados (SGBDFs).

Mediadores têm os mecanismos e modelos necessários para integrar as bases de dados de forma virtual com o objetivo de fornecer acesso e recuperação dos dados de fontes heterogêneas, além de permitir a transformação e processamento dos dados recuperados. Mediadores são construídos para domínios específicos e a complexidade está diretamente ligada às necessidades do cliente e às fontes de dados (Wiederhold, 1999).

Um SGBDF é uma coleção de sistemas de bancos de dados autônomos e um componente de administração que tem o controle sobre os sistemas individuais. As consultas realizadas são interpretadas pela federação, que cria os esquemas de importação e, por meio de um protocolo definido, faz a intermediação entre os diversos bancos de dados autônomos integrantes. SGBDFs podem ser classificados em sistemas de acoplamento forte ou fraco. Nos SGBDFs com acoplamento fraco, a integração dos dados é mantida pelo usuário, ou seja, o usuário tem que compreender a semântica de cada base e resolver as heterogeneidades. Nos SGBDFs com acoplamento forte, cada fonte de dados é traduzida para um modelo de dados comum e estes são integrados em um esquema global. Assim, o mapeamento das consultas e operações de modificação são transparentes para o usuário (Sheth e Larson, 1990).

2.4.2 Ontologias

De acordo com Gruber (2009), no contexto de Ciência da Computação e Sistemas de Informação, uma ontologia pode ser definida como um conjunto de primitivas que permitem modelar um domínio do conhecimento. As primitivas são classes (conceitos), atributos (propriedades) e relacionamentos ou restrições entre classes.

No contexto de sistemas de bancos de dados, a ontologia pode ser vista como uma abstração do modelo de dados destinada a modelar o conhecimento independentemente da estrutura dos dados. Enquanto o esquema do banco de dados está no nível físico ou lógico, as ontologias estão no nível semântico. Devido à independência entre o modelo de dados físico e o modelo semântico, as ontologias são usadas para integrar bancos de dados heterogêneos, pois possibilitam acessar os dados de diferentes sistemas, cada um com suas características particulares, utilizando uma única semântica (Gruber, 2009).

O acesso a bancos de dados utilizando ontologias (OBDA, *Ontology-Based Data Access*) tem como objetivo facilitar o acesso do usuário, disponibilizando a ele o vocabulário do domínio com o qual está familiarizado e abstraindo detalhes sobre o armazenamento dos dados. A estrutura OBDA é composta pela ontologia (camada conceitual), a(s) base(s) de dados e o mapeamento responsável pela comunicação entre a camada conceitual e a(s) base(s) de dados (Calvanese *et al.*, 2013).

O OBDA foi inicialmente proposto para bancos de dados relacionais. Mas com o *Big Data*, novas propostas sugerem o uso de ontologias no acesso a bancos de dados NoSQL, com o objetivo de tratar problemas nesse contexto. Tais estudos ainda são recentes e as ferramentas para uso de ontologias com banco de dados NoSQL não são consolidadas (Agena, 2017).

2.4.3 *Data Warehouse*

O termo Data Warehouse (DW) foi criado por Inmon (1996). Inmon definiu um DW como um sistema que coleta dados orientado a assunto, integrado, variável com o tempo e não volátil, para apoiar o processo de tomada de decisão gerencial.

De acordo com Inmon, um DW deve ter somente uma base de dados onde os dados são armazenados com um esquema pré-definido e de forma normalizada, ou seja, evitando ao máximo a redundância de dados. Esse banco de dados normalizado é chamado de *Normalized Data Store* (NDS). No NDS, os dados podem ser extraídos, desnormalizados e indexados, dando origem aos bancos analíticos (*Data Marts*).

Para Kimball (1997), um DW deve ser montado com um conjunto de *Data Marts*. Nessa abordagem, os dados são armazenados utilizando o modelo de dados dimensional para obter eficiência na recuperação dos dados. O modelo de dados dimensional possui dois conceitos principais: fatos e dimensões. Fato é uma medida importante para análise, e dimensão é um contexto relacionado ao fato. Por exemplo, para formulação, monitoramento e avaliação de políticas públicas na área da saúde, é importante conhecer a taxa de mortalidade por faixa etária ou sexo, pois ela possui correlação com as condições de vida da população. Nesse contexto, a taxa de mortalidade é um fato (medida) e faixa etária e sexo são dimensões.

Tão importante quanto o modelo de dados do DW são os componentes que manipulam os dados para serem inseridos na base de dados do DW. As ferramentas ETL (*Extract, Transform, Load*) têm como objetivo extrair dados de diversas fontes e transformar, validar, corrigir e adaptar os dados extraídos conforme a necessidade do DW (Jarke *et al.*, 2013).

A ferramenta de visualização dos dados também é de extrema importância no DW. No

contexto de DW, destacam-se as ferramentas de visualização para manipulação de grandes volumes de dados multidimensionais chamadas de OLAP (*On-Line Analytical Processing*), que permitem a análise de dados realizando a combinação de dimensões com métricas pré-calculadas ou métricas calculadas rapidamente (Jarke *et al.*, 2013).

Conforme mencionado anteriormente, um *Data Warehouse* utiliza um esquema pré-definido, que exclui intencionalmente dados considerados sem importância durante a definição do esquema. Dessa forma, no DW, dados considerados sem importância hoje não poderão ser utilizados no futuro. Portanto, de forma isolada, não é a melhor abordagem no contexto de *Big Data*, em que qualquer dado pode ser valioso para alguém ou algum algoritmo em algum momento do tempo (Rudnicki *et al.*, 2018).

2.4.4 *Data Lakes*

Data Lakes (DLs) são repositórios de dados que armazenam diferentes tipos de dados (estruturados, semiestruturados e/ou não estruturados) no seu formato original. DLs fornecem as funcionalidades de ingestão (carga) de dados, exploração e monitoramento (Fang, 2015). As informações do esquema, mapeamentos e outras restrições não são definidas explicitamente, mas é importante extrair o máximo de metadados possível das fontes de dados durante a fase de ingestão. Sem metadados, um DL dificilmente é utilizável, pois a estrutura e a semântica dos dados não são conhecidas (Hai *et al.*, 2016).

Os principais recursos de um DL são:

- Ingestão e armazenamento de dados heterogêneos em grande escala e com baixo custo computacional e financeiro;
- Ingestão de dados em tempo real e em lote;
- Transformações nos dados;
- Definição da estrutura dos dados no momento em que são usados, evitando a complexa modelagem dos dados;
- Dimensionamento para o crescimento contínuo dos dados;
- Acesso para vários usuários, de cientistas de dados a analistas de negócio, ao mesmo tempo, para monitoramento, exploração e análise. No entanto, ao considerar um público com necessidades e habilidades divergentes, é necessário ter cuidado para disponibilizar os dados certos para cada tipo de usuário.

De acordo com Gorelik (2014), a maioria dos DLs são organizados em quatro áreas: zona bruta, onde os dados são inseridos e mantidos o mais próximo possível de seu estado original; zona de ouro, onde são processados, limpos e armazenados; zona de trabalho, ambiente para os usuários mais técnicos, como cientistas e engenheiros de dados (o resultado de um trabalho realizado na zona de trabalho é movido para zona de ouro após a conclusão); e a zona sensível, onde dados importantes são protegidos de usuários não autorizados, seja por requisitos regulatórios ou necessidades do negócio. A zona de ouro pode conter uma cópia dos dados da zona sensível, mas com os campos sensíveis removidos ou anonimizados pelo processo de desidentificação.

Diferentes tipos de usuários utilizam diferentes zonas do DL. Por exemplo, engenheiros de dados utilizam principalmente a zona de trabalho com dados da zona bruta, já analistas de negócio utilizam principalmente dados da zona de ouro.

Metadados

O gerenciamento de metadados é essencial nos DLs, já que os dados armazenados não apresentam um esquema explícito (Khine e Wang, 2018). Sawadogo e Darmont (2020) identificaram na literatura três categorias principais de metadados para DLs:

- Metadados técnicos – descrevem o formato, estrutura ou esquema dos dados. A estrutura de dados consiste em características como atributos, tipos e tamanho;
- Metadados de negócios – têm as informações necessárias para o entendimento do contexto e significado dos dados. Descrevem as regras de negócio e estrutura dos dados, ou seja, seus atributos e relacionamentos, para facilitar a compreensão dos dados;
- Metadados operacionais – possuem informações geradas automaticamente durante o processamento dos dados, por exemplo, o tamanho do arquivo e o número de registros.

2.5 Arquitetura de Microsserviços

Na criação de sistemas projetados para ambientes de computação em nuvem, é comum adotar a arquitetura de microsserviços. Embora não exista uma definição formal para microsserviço, Fowler e Lewis (2014) definem a arquitetura de microsserviços como uma abordagem para desenvolver um único aplicativo como um conjunto de pequenos serviços, cada um executando em seu próprio processo e se comunicando com mecanismos leves, geralmente uma API de recurso HTTP ou chamada de procedimento remoto (RPC). Algumas das características importantes da arquitetura de microsserviços incluem:

- Decomposição em pequenos serviços: A aplicação é decomposta em pequenos serviços, cada um responsável por uma tarefa específica do negócio. Isso torna mais fácil gerenciar e manter a aplicação.
- Independência dos serviços: Cada serviço é projetado para ser independente e autocontido, o que significa que pode ser desenvolvido, testado e implementado de forma isolada.
- Comunicação entre serviços através de APIs: Os serviços se comunicam entre si através de APIs, o que torna possível que eles sejam desenvolvidos usando diferentes tecnologias e linguagens de programação.
- Escalabilidade horizontal: Cada microsserviço pode ser escalado individualmente, sem afetar os outros serviços da aplicação. Portanto, é favorável para lidar com grandes volumes de requisições e dados.
- Flexibilidade: A arquitetura de microsserviços permite que novas funcionalidades sejam adicionadas ou removidas sem afetar o funcionamento da aplicação como um

todo.

- Facilidade de manutenção: Como cada microsserviço é independente, é possível corrigir *bugs* ou implementar melhorias em um serviço específico sem afetar os outros serviços da aplicação.
- Falhas isoladas: Se um microsserviço falhar, apenas ele será afetado e não toda a aplicação. Isso aumenta a disponibilidade e confiabilidade da aplicação.
- Gerenciamento de Dados: cada microsserviço pode ter seu próprio armazenamento de dados, escolhendo a tecnologia mais adequada às suas necessidades.

2.6 Avaliação de Microsserviços

A avaliação de microsserviços refere-se ao processo de avaliar e analisar diferentes aspectos relacionados a eficácia, desempenho e qualidade dos microsserviços para garantir que eles atendam os requisitos funcionais e não funcionais. Dentre os aspectos que podem ser analisados, destacam-se:

- Desempenho: avaliar o tempo de resposta, latência e taxa de transferência de cada microsserviço, bem como a eficiência do sistema como um todo sob cargas de trabalho baixas, moderadas e altas.
- Escalabilidade: verificar como os microsserviços escalam horizontal ou verticalmente conforme a demanda, para garantir que o sistema se adapte à demanda.
- Disponibilidade: analisar a disponibilidade de cada microsserviço e do sistema como um todo, considerando estratégias de balanceamento de carga, tolerância a falhas e recuperação.
- Confiabilidade: verificar a capacidade dos microsserviços de manter a integridade dos dados, responder corretamente a solicitações e lidar com erros de forma adequada.
- Segurança: avaliar a segurança dos microsserviços, incluindo autenticação, autorização e proteção contra ameaças.

Li, OBrien e Zhang (2013) propuseram a *Cloud Evaluation Experiment Methodology* (CEEM) para avaliar sistematicamente serviços em nuvem por meio de experimentos que podem ser facilmente replicados ou estendidos para qualquer ambiente.

A metodologia propõe dez etapas para avaliar os serviços de uma aplicação:

1. Definição dos Objetivos – consiste em identificar o problema e os objetivos da avaliação. Para ajudar a reconhecer um requisito, foi sugerido preparar um conjunto de perguntas específicas a serem abordadas por potenciais experimentos de avaliação. Dentre os objetivos, pode-se ter avaliar o desempenho, a escalabilidade, a disponibilidade, a segurança ou outros aspectos específicos da solução;
2. Identificação dos Serviços – após a definição dos objetivos e identificação dos requisito de avaliação, é necessário identificar os serviços de nuvem relevantes e suas características a serem avaliadas. Para serviços em nuvem, existem três características gerais que são mais analisadas: desempenho, segurança e custo. Quando se tratar da

avaliação de desempenho de serviços em nuvem, o trabalho de Li, OBrien, Cai *et al.* (2012) pode ser usado para facilitar a exploração das propriedades de desempenho disponíveis.

3. Listagem de Métricas e *Benchmarks* – consiste em listar as métricas e *benchmarks* que podem ser utilizados. Exemplos de métricas e *benchmarks* podem ser encontrados no trabalho de Li, O'Brien *et al.* (2012b);
4. Seleção de Métricas e *Benchmarks* – selecionar as métricas apropriadas e *benchmarks* para avaliação;
5. Listagem de Fatores Experimentais – identificar os fatores (parâmetros ou variáveis) que podem impactar na avaliação é de extrema importância. Listar todos os fatores experimentais não uma tarefa fácil, mas os avaliadores devem sempre manter a lista de fatores o mais abrangente possível, para análise posterior e tomada de decisão;
6. Seleção dos Fatores Experimentais – a seleção dos fatores a serem estudados e os critérios de aceite deve ser feita por especialistas da área. Para apoiar as decisões dos especialistas, pode ser utilizado o arcabouço apresentado no trabalho de Li, O'Brien *et al.* (2012a);
7. Desenho Experimental – identificados os parâmetros, variáveis e as métricas a serem analisados, pode-se projetar os experimentos de avaliação de serviços seguindo o DOE (*Design of Experiments*), que é uma abordagem estatística para planejar, conduzir e analisar experimentos. O desenho dos experimentos deve considerar principalmente a reprodutibilidade, aleatorização e replicação dos experimentos;
8. Implementação Experimental – nesta etapa é necessário preparar o ambiente de teste e executar os experimentos. Qualquer erro na execução pode prejudicar a validade dos resultados experimentais, portanto o processo deve ser monitorado para garantir que cada etapa do experimento siga o projeto;
9. Análise Experimental – os métodos estatísticos são fortemente sugeridos na análise e interpretação dos resultados, por prover objetividade na obtenção de conclusões de avaliação e nos potenciais processos de tomada de decisão;
10. Conclusão e Relatórios – após concluir o trabalho de avaliação, os detalhes do procedimento e os resultados devem ser documentados para o propósito de verificação e compartilhamento de experiências. O trabalho de Runeson e Höst (2009) apresenta uma metodologia e estrutura de documentação para reportar as implementações de avaliação de serviços em nuvem e pode ser utilizado como referência para confecção do relatório.

Capítulo 3

Trabalhos Relacionados

Neste capítulo, são abordados os trabalhos mais relevantes para o tema deste projeto, ressaltando os aspectos em que se assemelham a este último. Buscamos no *Google Acadêmico* (2023) trabalhos relacionados utilizando as *strings* de busca *semantic ontology integration smart cities*, *data warehouse smart cities*, *data lake smart cities* e *big data smart cities*. Selecionamos artigos publicados a partir de 2015. Alguns artigos inicialmente selecionados foram excluídos da análise com base no título, resumo e/ou questões de pesquisa, por não estarem relacionados diretamente a este trabalho.

Os trabalhos encontrados podem ser divididos em duas categorias de acordo com a estratégia de integração de dados na qual se baseiam: os de abordagem semântica e os baseados em *Data Warehouses* e *Data Lakes*.

3.1 Abordagem Semântica

Psyllidis *et al.* (2015) desenvolveram o SocialGlass, uma plataforma Web que oferece recursos para análise, integração e visualização de dados urbanos heterogêneos com o objetivo de auxiliar no planejamento urbano e a tomada de decisões. O SocialGlass combina dados municipais com dados de mídia social (Twitter, Instagram e Foursquare) e permite o mapeamento de informações demográficas, padrões de movimento humano, popularidade de locais, condições de tráfego, opiniões e preferências de cidadãos e visitantes em relação a locais específicos da cidade. A arquitetura do SocialGlass está dividida em três módulos principais: ingestão, integração e exploração. O módulo de ingestão refere-se à aquisição, limpeza e processamento de dados sociais e de sensores. O módulo de integração é responsável por possibilitar a interoperabilidade entre as diversas fontes de dados. Para isso, foi desenvolvido um modelo de representação do conhecimento, baseado em ontologia, que representa os sistemas urbanos, as relações entre eles e as fontes de dados correspondentes. O módulo exploração oferece uma interface Web baseada em mapas para visualização e exploração dos dados, possibilitando obter *insights* sobre parâmetros espaciais e temporais do contexto urbano. Os autores avaliaram que o SocialGlass foi utilizado com sucesso em 2014 na *Milano Design Week*, *Amsterdam Light Festival* e *Como Summer Holiday Season*, mas não foram apresentados detalhes das avaliações e tecnologias utilizadas para armazenamento, processamento, integração e exploração dos dados.

Consoli *et al.* (2015) apresentaram uma abordagem de integração com ontologia utilizando *Linked Data* (um conjunto de práticas para publicar e conectar dados na Web). No trabalho apresentado, cada conjunto de dados foi convertido para um modelo de dados RDF (*Resource Description Framework*) utilizando processos customizados. As ontologias foram geradas para atingir a interoperabilidade conceitual. Portanto, foi necessária a ajuda de especialistas do domínio de cada conjunto de dados para representar cada entidade por uma classe, cada atributo de entidade em uma propriedade de dados e para unificar entidades e propriedades que se referiam aos mesmos conceitos. Os dados e a ontologia são acessíveis por meio de consulta a API SPARQL.

J. Silva *et al.* (2021) apresentaram o Aqüeducte, um serviço para integração de dados no contexto de cidades inteligentes baseado em ontologias e *Linked Data* que possibilita a modelagem estruturada de informações e busca de dados com a realização de inferências. A arquitetura do Aqüeducte está dividida em quatro serviços: Aqüeducte UI, uma interface web implementada em *VueJs* (2023) que permite ao usuário carregar e extrair dados; Aqüeducte service, serviço implementado em Java que realiza os processos de importação, extração, filtragem e persistência dos dados no *MongoDB* (2023); Aqüeconnect, serviço de gerenciamento de arquivos implementado em Java utilizando o *Apache Hadoop* (2023), e Aqüegeo, uma API REST implementada em *Python* (2023) para gerenciamento de arquivos com vetores geográficos, permitindo o *upload* e a conversão deles em arquivos CSV, que são armazenados no HDFS do Aqüeconnect.

3.2 *Data Warehouse* e *Data Lake*

Cheng *et al.* (2015) propuseram uma arquitetura de *Big Data* integrada ao ambiente de teste experimental de IoT (*Internet of Things*) SmartSantander. Essa arquitetura está dividida em quatro módulos principais: (1) coleta de dados, (2) armazenamento de dados, (3) processamento e análise de dados e (4) API para comunicação com aplicativos externos. O módulo de coleta de dados é representado por um *broker* que é responsável por receber dados de diversas fontes. Para o armazenamento de dados em tempo real e histórico, foi proposta a utilização de um banco de dados NoSQL. O processamento e análise de dados podem ser feitos em lote ou em fluxo, utilizando uma ferramenta de computação distribuída. O módulo para comunicação com aplicativos externos tem uma API RESTful para permitir que os aplicativos externos façam consultas simples, consultas complexas e assinaturas. Uma consulta simples solicita resultados agregados sobre o status mais recente de todos os sensores, enquanto uma consulta complexa pode solicitar resultados agregados sobre dados históricos dentro de um intervalo de tempo especificado. A assinatura é o mecanismo usado para que os aplicativos recebam notificações com os resultados mais recentes, evitando que o aplicativo consulte os dados o tempo todo. Essa arquitetura não apresenta um módulo de visualização dos dados e metadados armazenados, dificultando a utilização por gestores públicos.

Rathore *et al.* (2016) propuseram um sistema para coleta, agregação, filtragem, classificação, pré-processamento, computação e tomada de decisão utilizando a abordagem de *Data Lake* combinada com *Data Warehouse*. O sistema proposto está divido em quatro camadas: (1) geração e coleta de dados, (2) transmissão de dados, (3) gerenciamento e processamento de dados e (4) análise de dados. As camadas 1 e 2 são responsáveis pela

coleta de dados utilizando sensores e pela transferência dos dados até a plataforma de armazenamento, portanto, não estão relacionadas ao trabalho aqui proposto. A camada 3, gerenciamento e processamento de dados, utiliza um sistema de arquivos distribuídos (HDFS) e ferramentas de computação distribuída para o processamento dos dados em tempo real, e para dados históricos sugere a utilização do *Apache Hive* (2023) como ferramenta para *Data Warehouse* e do *Apache HBase* (2023) para banco de dados distribuído.

De forma semelhante, Hashem *et al.* (2016) propuseram uma arquitetura de *Big Data* para cidades inteligentes dividida em quatro camadas. A primeira é composta pelas fontes e a transferência dos dados, enquanto a segunda é responsável pelo armazenamento dos dados em um banco de dados distribuído e tolerante a falhas. Na mesma camada, os dados armazenados são processados de acordo com as consultas recebidas utilizando um modelo de programação de processamento paralelo e distribuído. A terceira camada, análise inteligente, foi projetada para possibilitar o aprendizado de máquina e mineração de dados para a extração de padrões e conhecimento a partir de grandes quantidades de dados. A última camada é composta por aplicativos que utilizam os dados armazenados e os processamentos realizados para diferentes propósitos, como gerenciamento inteligente de recursos públicos.

Costa e Santos (2017) apresentaram uma abordagem para projetar e implementar um *Big Data Warehouse* no contexto de cidades inteligentes, com um repositório que armazena dados em formato bruto. A arquitetura proposta está dividida em quatro grandes módulos responsáveis pela coleta, preparação e enriquecimento de dados, armazenamento e acesso, análise e visualização. Os dados podem ser coletados em tempo real, utilizando um *broker Kafka* (2023), ou em lote, utilizando uma ferramenta ETL, por exemplo, *Talend* (2023) e HDFS Upload. Os dados coletados em lote são armazenados diretamente em arquivos de um sistema distribuído e, posteriormente, são preparados e enriquecidos utilizando a ferramenta de computação distribuída *Apache Spark* (2023) e, então, são armazenados em um *Data Warehouse Apache Hive* (2023). Os dados coletados em tempo real também são armazenados em um sistema de arquivos distribuídos, preparados e enriquecidos utilizando uma ferramenta de computação distribuída, e posteriormente armazenados em um banco de dados distribuído *Cassandra* (2023). Os dados armazenados podem ser acessados por uma ferramenta de consulta SQL distribuída (a *Presto* (2023)) e pela ferramenta de visualização de dados. O modelo proposto foi implementado no projeto de pesquisa SusCity e possibilitou realizar análises baseadas em dados coletados na cidade de Lisboa.

De forma semelhante, Mehmood *et al.* (2019) propuseram uma arquitetura dividida em cinco módulos responsáveis pela coleta, ingestão, armazenamento, exploração e análise de dados e visualização. Para ingestão de dados, foi proposta a utilização de uma ferramenta de processamento de *streams* (a *Flume* (2023)) com armazenamento em um sistema de arquivos distribuídos (HDFS). A análise e exploração dos dados foram realizadas com ferramentas de indexação distribuída (*Solr* (2023)) e computação distribuída (*Apache Spark* (2023)). Para a visualização dos dados, foi proposta a utilização de uma ferramenta Web de consulta SQL (*Hue* (2023)) e uma biblioteca baseada em *Python* (2023), chamada *Matplotlib* (2023).

Banni *et al.* (2022) apresentaram o HURRICANE, um serviço configurável e extensível para gerência de dados para aplicações de cidades inteligentes que utiliza o *Apache Airflow*

(2023) para gerência e processamento dos dados. Os dados no HURRICANE têm um fluxo bem definido, que inicia com a ingestão a partir de fontes externas. A aplicação é capaz de importar dados de diferentes fontes, desde que uma API disponibilize os dados para consulta, exceto os dados espaciais e as topologias das cidades, que são importadas do *Open Street Map* (OSM). Os dados brutos coletados no processo de ingestão são armazenados em um *Data Lake Apache Hadoop* (2023). Concluída a etapa de ingestão dos dados, é iniciado o processamento dos dados, que está organizado em quatro etapas: (i) *Raw* (identifica registros duplicados), (ii) Bronze (sumarização e agregação), (iii) Prata (associa os dados agregados a informação georreferenciadas), e (iv) Ouro (cria um DW modelado no *PostgreSQL* (2023) para cada tipo de dado de cidades inteligentes). Durante o processo ETL, os dados podem sofrer um processo de anonimização. Todo o histórico de transformação dos dados é armazenado como metadado de proveniência em um banco de dados específico. Os dados integrados e agregados podem ser consumidos por outras aplicações ou usuários por meio de uma API da aplicação.

3.3 Considerações sobre os Trabalhos Relacionados

A Tabela 3.1 apresenta uma classificação dos trabalhos analisados em relação ao tipo (aplicado ou teórico), o escopo e as principais funcionalidades propostas. Este trabalho se assemelha aos analisados na funcionalidade de ingestão dos dados e na proposta de integração física dos dados, mas propõe novas funcionalidades, como a definição de novas coleções com base nas já existentes e a compatibilização dos dados que possuem alteração estrutural ou semântica ao longo do tempo, utilizando o histórico de modificações dos metadados.

Os trabalhos que apresentam uma abordagem aplicada reutilizaram e sugeriram a reutilização de ferramentas para a implementação das arquiteturas propostas. A Tabela 3.2 apresenta as ferramentas sugeridas em cada um dos trabalhos por funcionalidade. A tabela destaca em negrito as ferramentas utilizadas pelos autores para a implementação dos sistemas propostos. Como pode ser visto, a escolha de um sistema de arquivos distribuído para armazenar os dados predomina. Da mesma forma, o *Apache Spark* (2023) é a principal ferramenta aplicada para processamento dos dados.

Autores	Tipo	Escopo	Funcionalidades Propostas
Psyllidis *et al.* (2015)	Aplicado	Plataforma web para integração, análise e exploração de dados em cidades inteligentes, utilizando um modelo de representação do conhecimento baseado em uma ontologia que representa os sistemas urbanos, as relações entre eles e as fontes de dados correspondentes. Não sugere ferramentas para a implementação da arquitetura proposta.	Ingestão de dados Integração semântica Análise e exploração
Consoli *et al.* (2015)	Teórico	Proposta de uma abordagem de integração de dados em cidades inteligentes utilizando ontologia.	Integração semântica Consulta dos dados
Cheng *et al.* (2015)	Aplicado	Implementação de uma plataforma para *Big Data* capaz de coletar e processar dados históricos e em tempo real, ao mesmo tempo expondo os resultados a vários aplicativos.	Ingestão de dados Processamento Consulta de dados
Rathore *et al.* (2016)	Aplicado	Proposta e implementação de um sistema para planejamento urbano. O sistema é dividido em três níveis: físico (armazenamento dos dados), intermediário (processamento dos dados) e superior (tomada de decisão). O nível superior é composto de várias aplicações de tomada de decisão com objetivos específicos, por exemplo, planejamento de estradas e planejamento de construção.	Ingestão de dados Processamento Tomada de decisão
Hashem *et al.* (2016)	Teórico	Apresenta um modelo conceitual de sistema de *Big Data* para cidades inteligentes.	Ingestão de dados Processamento Automação de decisões Envio de alertas e relatórios
Costa e Santos (2017)	Aplicado	Proposta de um sistema para *Big Data Warehouse* com recursos para visualização de dados e tomada de decisões em cidades inteligentes.	Ingestão de dados Preparação e enriquecimento dos dados Análise e visualização
Mehmood *et al.* (2019)	Aplicado	Proposta de um sistema para coleta e gerenciamento de dados de fontes heterogêneas.	Ingestão de dados Processamento Integração Exploração e análise Visualização
J. Silva *et al.* (2021)	Aplicado	Proposta de um serviço configurável e extensível para integração e agregação de dados utilizando dataflow.	Ingestão de dados Processamento Integração Agregação Consulta
Banni *et al.* (2022)	Aplicado	Proposta de um sistema para integração de dados baseado em ontologias e Linked Data.	Ingestão de dados Processamento Integração Consulta

Tabela 3.1: *Comparação dos trabalhos relacionados*

	Transporte	Armazenamento	Metadados	Processamento	Consulta	Visualização
Cheng *et al.* (2015)		**CouchDB** CouchBase MongoDB Hbase **HDFS**		**Spark**	**CouchDB REST API**	
Rathore *et al.* (2016)		**HDFS** **Hbase** **Hive**		**Spark** Storm	**Hive**	
Costa e Santos (2017)	**Kafka** **HDFS Upload** **Talend**	**HDFS** **Hive** **Cassandra**		**Spark** **Talend**	**Presto**	**Maps API** **Chart.js**
Mehmood *et al.* (2019)	**Flume** Kafka NiFi	**HDFS** HBASE	Atlas Kite SDK	**Solr** Spark	**Hue**	Matplotlib Kibana Grafana
J. Silva *et al.* (2021)		**HDFSPostgreSQL**		**Apache Airflow**	**Python**	
Banni *et al.* (2022)		**HDFSMongoDB**		**Java**		

Tabela 3.2: *Ferramentas mencionadas para reutilização por autor e funcionalidade. Em destaque as ferramentas utilizadas na implementação*

Capítulo 4

Requisitos para Plataformas de Software de Integração de Dados em Cidades Inteligentes

Este capítulo apresenta requisitos funcionais e não funcionais de um sistema de integração de dados para cidades inteligentes, identificados a partir da revisão de literatura apresentada no Capítulo 3.

4.1 Requisitos Funcionais

O principal objetivo de uma plataforma de integração de dados em cidades inteligentes é facilitar o desenvolvimento de aplicações que utilizam dados combinados de diferentes fontes. Para tanto, a maioria das plataformas analisadas implementa funcionalidades relacionadas aos requisitos de ingestão, processamento, análise, visualização e compartilhamento de dados. A Tabela 4.1 fornece uma visão geral de como os trabalhos relacionados cobrem esses requisitos funcionais. Cada requisito é descrito na sequência.

	Ingestão	Metadados	Processamento	Aprendizado de Máquina	Análise e Visualização	Acesso externo
Psyllidis *et al.* (2015)	X	X			X	
Consoli *et al.* (2015)		X				X
Cheng *et al.* (2015)	X		X			X
Rathore *et al.* (2016)	X		X	X		
Hashem *et al.* (2016)	X		X	X		X
Costa e Santos (2017)	X		X		X	
Mehmood *et al.* (2019)	X	X	X		X	
J. Silva *et al.* (2021)	X		X			X
Banni *et al.* (2022)	X	X	X			X

Tabela 4.1: *Requisitos funcionais*

A **ingestão de dados** é o processo de importação de dados em tempo real ou em

lote para a plataforma de armazenamento. Os dados podem vir de diferentes fontes, em diferentes formatos, como CSV, TXT, JSON e outros.

O **gerenciamento de metadados** é o processo de coleta e gerenciamento de informações sobre os dados armazenados na plataforma. Os metadados devem conter informações sobre a semântica e a estrutura das coletas de dados. Os metadados também devem manter informações sobre os mapeamentos necessários para padronizar os dados e garantir a compatibilidade com versões anteriores, a fim de permitir a integração dos dados e a facilidade de uso.

Os dados que chegam à plataforma podem ser imprecisos, incompletos, inconsistentes ou redundantes. Além disso, esses dados podem precisar de agregação, filtragem, análise ou anonimização antes de permitir a descoberta de conhecimento. Assim, as plataformas devem oferecer recursos para a criação e execução de procedimentos de **processamento de dados**.

Extrair conhecimento e *insights* de dados é de suma importância para possibilitar melhores tomadas de decisão nas cidades e possibilitar a implementação de políticas públicas mais eficientes. Portanto, as plataformas de integração de dados devem permitir a criação, manutenção e execução de modelos personalizados de **aprendizado de máquina**.

A **análise e visualização de dados** referem-se à apresentação de dados em formatos gráficos amigáveis, para ajudar os usuários a entender o comportamento das cidades e o uso de recursos. Uma plataforma de integração de dados deve oferecer recursos para apoiar a criação de relatórios e painéis personalizados para os gerentes converterem dados em conhecimento.

O **acesso externo** refere-se à possibilidade de sistemas externos consumirem os dados armazenados na plataforma, permitindo o desenvolvimento de novos aplicativos para melhorar os serviços prestados à população ou otimizar o uso dos recursos disponíveis. Somente usuários ou sistemas autorizados devem ter acesso aos dados.

4.2 Requisitos Não Funcionais

A Tabela 4.2 apresenta os requisitos não funcionais mencionados nos trabalhos relacionados do Capítulo 3. A seguir, cada um dos requisitos não funcionais é descrito.

	Escalabilidade	Disponibilidade	Segurança e Privacidade
Psyllidis *et al.* (2015)	X		
Consoli *et al.* (2015)			
Cheng *et al.* (2015)	X		
Rathore *et al.* (2016)	X		X
Hashem *et al.* (2016)		X	
Costa e Santos (2017)			X
Mehmood *et al.* (2019)	X	X	

Tabela 4.2: *Requisitos não funcionais*

Escalabilidade refere-se à capacidade de aumentar ou diminuir recursos computacionais de acordo com a necessidade do sistema. A escalabilidade pode ser vertical, significando adicionar (remover) recursos para (de) um único nó; ou horizontal, ao adicionar (remover) nós para (de) um sistema distribuído.

Disponibilidade refere-se à capacidade da plataforma de ser resistente a falhas de hardware, software e energia para manter os serviços disponíveis pelo maior tempo possível.

Segurança e privacidade referem-se a restringir o acesso aos dados armazenados a usuários e sistemas autorizados, evitando vazamento e uso indevido de dados. Também inclui a capacidade da plataforma de cumprir as políticas de proteção de dados para que os dados confidenciais sejam devidamente anonimizados e protegidos.

4.3 Desafios e Soluções

A implementação desses requisitos impõe desafios devido ao grande volume de dados e à heterogeneidade de fontes e formatos. Segundo (Cheng *et al.*, 2015), a escalabilidade é uma questão importante, pois a quantidade de dados aumenta muito com o tempo, com a disponibilidade de novos serviços e tecnologias, e também com o crescimento populacional.

O armazenamento eficiente de dados não estruturados ou semiestruturados e o processamento dos dados históricos e em tempo real de forma escalável é um desafio. Para lidar com esses desafios, Cheng *et al.* (2015) propuseram um repositório de *big data* onde os dados podem ser transformados em novos formatos, e novas tabelas com índices podem ser criadas para melhorar as consultas. Processamento de dados mais complexos podem ser executados de forma assíncrona ao aprovisionamento dos dados.

Cheng *et al.* (2015) e Hashem *et al.* (2016) apontam que os dados recebidos na plataforma podem ter anomalias ou estarem incompletos ou incorretos. Portanto, é necessário implementar algoritmos de detecção de anomalias para não afetar o resultado do processamento e análise dos dados. A segurança e privacidade dos dados também são questões importantes, pois há muitos dados confidenciais sendo coletados em cidades inteligentes e as tentativas de ataques cibernéticos se tornam cada vez mais frequentes, embora ainda haja poucas abordagens para mitigar este desafio. Os autores relataram essa dificuldade, mas não apresentaram abordagens para garantir a qualidade dos dados. Mehmood *et al.* (2019) sugerem a implementação de um módulo para detecção de anomalias, dados duplicados, violação de regras ou violação de padrões, de forma que tais dados sejam removidos ou corrigidos para manter a base com dados de boa qualidade.

Mehmood *et al.* (2019) e J. Silva *et al.* (2021) apresentam como desafio a interoperabilidade dos dados e citam como possível abordagem para vencer este desafio o uso de ontologias. A proposta de J. Silva *et al.* (2021) sugere utilizar o padrão *Next Generation Service Interfaces - Linked Data* (NGSI-LD) para padronizar os dados e facilitar a interoperabilidade.

Capítulo 5

Arquitetura de Microsserviços para Integração de Dados em Cidades Inteligentes

Neste capítulo, apresentamos uma nova arquitetura para orientar o desenvolvimento de plataformas de software de integração de dados para cidades inteligentes. A Figura 1 mostra o diagrama da arquitetura. Essa arquitetura foi inicialmente derivada das arquiteturas de trabalhos relacionados e, em seguida, foi significativamente melhorada para atender aos requisitos e problemas discutidos no Capítulo 4.

Uma arquitetura de microsserviços é uma aplicação distribuída em que todos os seus módulos são pequenos serviços, cada um deles sendo um processo coeso e independente que se comunica com os outros por meio de mecanismos leves (Dragoni *et al.*, 2017). Adotamos a arquitetura de microsserviços neste trabalho a fim de alcançar escalabilidade e resiliência individualizada dos serviços para obter um melhor aproveitamento dos recursos computacionais, já que determinadas partes da aplicação exigem mais recursos que outras. Nas próximas seções, apresentamos os padrões de microsserviços adotados e os componentes da arquitetura proposta.

5.1 Princípios

A seguir, descrevemos brevemente os padrões de microsserviços, ou seja, as diretrizes reutilizáveis para resolver problemas comuns no projeto e implementação, que nortearam a arquitetura proposta. São eles:

- Modularidade: consiste em dividir o sistema em unidades funcionais menores (microsserviços) que se comunicam utilizando APIs leves;
- Modelos e dados distribuídos: cada microsserviço tem seu próprio banco de dados e modelos, que podem evoluir de forma independente, facilitando a evolução e permitindo utilizar a tecnologia que melhor se adapta a cada contexto;
- Evolução descentralizada: os microsserviços devem ser autônomos, com limites bem

definidos e APIs de comunicação para que possam evoluir e serem mantidos de forma independente. Os microsserviços devem poder escalar de forma independente e utilizando estratégias diferentes;

- Reutilização de projetos de código aberto: reutilizar componentes de software permite aumentar a produtividade, economia e confiabilidade do software. Os critérios de seleção utilizados neste trabalho são: comunidade de desenvolvedores ativa, suporte de versão estável e documentação apropriada para orientar o uso, desenvolvimento e implantação;
- Adoção de padrões abertos: os padrões abertos têm como objetivo fornecer interoperabilidade em diferentes níveis, evitando o aprisionamento de tecnologia e fornecedor;
- Interações assíncronas: sempre que possível, deve-se implementar serviços assíncronos utilizando notificações, o padrão de projeto publicar-assinar e estratégias baseadas em eventos, para prover baixa latência e escalabilidade, evitando o bloqueio em interações de solicitação-resposta síncronas;
- Serviço sem estado: esse princípio de projeto defende que os serviços devem ser sem estado para permitir que qualquer instância de serviço responda a qualquer solicitação, facilitando a distribuição de carga, a elasticidade e a escalabilidade. Portanto, o design de microsserviços deve separar dados de estado, como dados de contexto e de sessão, para serem gerenciados por um componente externo sempre que possível.

5.2 Componentes da Arquitetura

A Figura 5.1 apresenta a arquitetura proposta. O componente de nível mais baixo na arquitetura proposta é o **Data Lake Storage**, um sistema de armazenamento de objetos responsável por armazenar dados o mais próximo possível de seu formato original, de modo que os usuários finais não precisem entender detalhes de como os dados são armazenados para poder usá-los. Para evitar que o sistema de armazenamento seja um gargalo para os mecanismos de consulta, os dados devem ser armazenados em um formato padronizado e compactado que facilite consultas analíticas e reduza o tamanho e o custo do armazenamento, mas as informações não podem ser descartadas ou perdidas. O software livre de armazenamento de objetos mais utilizado é o *MinIo* (2023) e os dados podem ser padronizados no formato *Apache Avro* (2023) ou *Apache Parquet* (2023) e compactados usando compressão *Google Snappy* (2023) ou *Gzip* (2023).

O microsserviço **Data Ingestion** é responsável por consumir dados assincronamente das filas de mensagens, convertendo os dados originais em um formato de arquivo padronizado e compactado e, em seguida, enviando-os para o **Data Lake Storage** usando um canal de comunicação pré-estabelecido. Assim como na arquitetura Kappa, os dados são tratados como fluxos de eventos. Mas para permitir um melhor escalonamento da aplicação, sugerimos a criação de uma fila para cada coleção de dados, reduzindo assim a concorrência entre diferentes coleções de dados. A separação de filas é importante para permitir a ingestão de dados próxima ao tempo real e melhorar a escalabilidade do sistema.

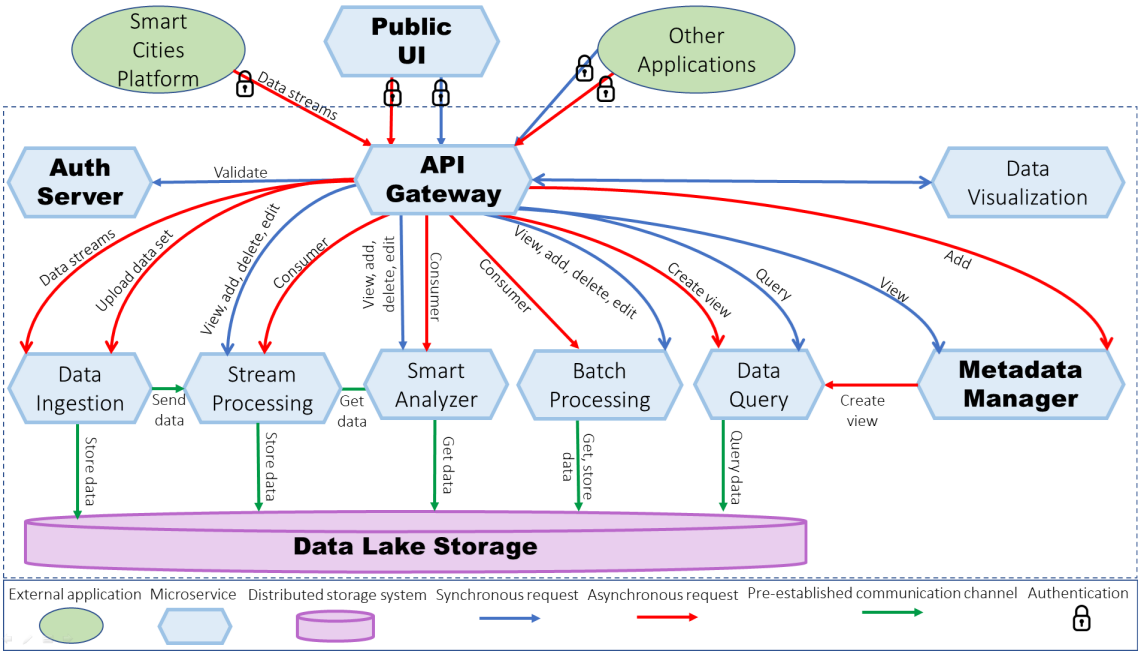

Figura 5.1: *Arquitetura proposta*

Este microsserviço pode ser implementado utilizando os softwares *Kafka* (2023), *RabbitMQ* (2023) ou *Apache Spark* (2023).

O **Metadata Manager** é responsável pelo cadastro e consulta de metadados para os dados armazenados no **Data Lake Storage**. É um catálogo de metadados com recursos para registrar fontes de dados e informações sobre esquema de dados, origem de dados, políticas de privacidade, versão de esquema e regras de mapeamento de dados. Também é responsável por solicitar a criação ou atualização da visualização de dados para o microsserviço **Data Query** sempre que uma nova versão dos metadados for criada. Os metadados precisam ter minimamente informações que identifiquem a origem dos dados, a política de privacidade, o contexto ou esquema dos dados e o modelo de compatibilização da versão atual com as versões anteriores para que uma coleção possa ser encontrada e utilizada facilmente por pessoas autorizadas. O modelo de metadados deve seguir os padrões estabelecidos pela *World Wide Web Consortium (W3C)* (2023). Sugerimos a adoção do *Data Catalog Vocabulary (DCAT) - Version 3* (2023) por apresentar constante evolução, facilitar o consumo e a agregação de metadados de vários catálogos e estar integrado com outros padrões como o *Schema.org* (2023) e *PROV Ontology (PROV-O)* (2023). O catálogo é mantido no próprio banco de dados do microsserviço (preferencialmente NoSQL, para facilitar o armazenamento dos metadados em JSON).

O **Data Query** é o microsserviço responsável por processar as solicitações de criação de visualização dos dados armazenados no **Metadata Manager** e executar consultas SQL de forma síncrona nos dados do **Data Lake Storage**. Estatísticas de consultas, como frequência de execução e tempo de resposta, devem ser armazenadas para permitir o uso de indexação automática e técnicas de *cache* para acelerar o acesso a dados usados com frequência. Para implementar este microsserviço, podem ser utilizadas ferramentas de análise de dados em tempo real, como *Presto* (2023), *Apache Druid* (2023) ou *Apache Drill*

(2023).

Para possibilitar a análise e processamento de dados em tempo real, temos o microsserviço **Stream Processing**. Este microsserviço permite a execução de tarefas nos dados assim que eles entram na plataforma. Ele fornece uma interface para os desenvolvedores criarem e gerenciarem seus trabalhos. Os resultados do processamento dos dados devem ser armazenados, permitindo seu uso por outros aplicativos. Este microsserviço pode ser implementado usando ferramentas como *Kafka* (2023) ou *Apache Spark Streaming* (2023).

O microsserviço **Smart Analyzer** fornece ferramentas para mineração de dados e criação, gerenciamento e execução de modelos de aprendizado de máquina nos conjuntos de dados armazenados no **Data Lake Storage** e *streams* de dados fornecidos pelo **Stream Processing**, podendo gerar eventos de notificação em uma fila de mensagens para consumo por outros aplicativos. Para implementar este microsserviço, pode-se usar o *Apache Spark Machine Learning* (2023) e *Scikit-learn* (2023), por exemplo.

O **Batch Processing** tem como responsabilidade possibilitar o processamento de grandes conjuntos de dados armazenados no **Data Lake Storage**, fornecendo uma interface para criação, gerenciamento e execução de tarefas. Os resultados do processamento podem ser salvos no **Data Lake Storage** ou publicados em uma fila de mensagens para que possam ser consultados por APIs e ferramentas de visualização ou consumidos por outros aplicativos. Este microsserviço pode ser implementado, por exemplo, utilizando MapReduce do *Apache Hadoop* (2023).

O **Data Visualization** oferece interfaces gráficas para apresentar e analisar dados armazenados no **Data Lake Storage** e informações geradas pelo **Stream Processing** e **Smart Analyzer**. Ele permite ao usuário criar relatórios e *dashboards*, possibilitando uma melhor interpretação dos dados. Este serviço pode ser disponibilizado utilizando ferramentas como o *Apache Superset* (2023) ou o *kibana* (2023).

O **API Gateway** fornece para aplicações externas um único ponto de acesso (com balanceamento de carga) aos demais microsserviços. A comunicação entre o **API Gateway** e as aplicações externas deve utilizar um canal de comunicação criptografado. Além disso, toda requisição deve consultar o microsserviço **Auth Server**, que autentica o usuário e gera um *token* para ser utilizado em requisições futuras. Após a autenticação bem sucedida, a solicitação é enriquecida com informações do usuário e encaminhada para o microsserviço de destino. O *gateway* pode ser implementado utilizado o *Apache knox* (2023) ou *Kong Gateway* (2023). Já o **Auth Server** pode ser implementado utilizando o *Apache Syncope* (2023) ou *Auth0* (2023).

O microsserviço **Public UI** tem como responsabilidade prover uma interface Web para as funcionalidades dos demais microsserviços descritos, isto é, carga de dados, gerenciamento de metadados, consulta de dados, processamento de dados em lote e em tempo real, aprendizado de máquina e visualização e análise de dados. Este microsserviço é público e deve utilizar um único ponto de acesso, o **API Gateway**, com canal de comunicação criptografado e usuário identificado. Dentre as tecnologias que podem ser utilizadas para implementar a interface Web e integrar com as funcionalidades disponibilizadas pelos demais microsserviços estão o *NodeJs* (2023), *React* (2023) e *HTML5* (2023).

Há várias preocupações que a implementação desses microsserviços precisa abordar. Por exemplo, a ingestão de dados deve ser capaz de lidar com grandes volumes e heterogeneidade de dados. O gerenciamento de metadados deve mapear mudanças de dados estruturais e semânticas, para fornecer compatibilidade de dados. A consulta de dados deve fornecer bom desempenho (com indexação automática e *cache*). O **API Gateway** deve equilibrar a carga para lidar com solicitações da maneira mais eficiente. Todos os microsserviços devem ser escaláveis e tolerantes a falhas, garantindo a segurança e privacidade dos dados.

Para aumentar a produtividade e confiabilidade do desenvolvimento da plataforma, sugerimos a reutilização de software livre, principalmente aqueles que possuem uma comunidade ativa de desenvolvedores; suporte a versões estáveis; versões evolutivas; uma rica documentação; bem como integração com ferramentas de segurança e privacidade de dados. Para facilitar a operação e manutenção da plataforma, sugerimos a adoção das melhores práticas de DevOps, como integração contínua, entrega contínua, implantação contínua e monitoramento.

5.3 Comparação com Arquiteturas Relacionadas

A arquitetura proposta oferece recursos para ingestão de dados e integração física usando abordagens semelhantes aos trabalhos analisados no Capítulo 3. No entanto, ela estende os trabalhos relacionados com recursos exclusivos, como: um único ponto de acesso a microsserviços por aplicativos externos; um serviço de autenticação e autorização de acesso; uma interface centralizadora dos serviços; a criação de novas coleções de dados com base nas existentes; e compatibilização de dados em coleções que sofreram alterações estruturais ou semânticas ao longo do tempo, utilizando o histórico de modificação de metadados e regras de mapeamento.

Mehmood *et al.* (2019) também propuseram o uso de metadados para apoiar a integração de dados, mas sua arquitetura trabalha com modelo de dados para fornecer vocabulário unificado entre as fontes de dados e alinhar diferenças sintáticas e semânticas, exigindo a definição de modelos de dados para cada setor da cidade (por exemplo, dados ambientais, sociais e econômicos). Em nosso trabalho, fizemos uma proposta mais abrangente de gerenciamento de metadados, usando *Data Catalog Vocabulary (DCAT) - Version 3* (2023) para descrever as fontes de dados. Além disso, não aplicamos uniformização na ingestão de dados. Os dados são armazenados conforme vêm da fonte e podem ser compatibilizados quando forem consultados.

Outra característica que distingue a plataforma de integração de dados apresentada neste trabalho dos trabalhos apresentados no Capítulo 3 é a utilização da arquitetura de microsserviços. Este tipo de arquitetura minimiza a dependência entre os componentes, permite o desenvolvimento independente e a escalabilidade do serviços, além de facilitar a implementação, teste e manutenção da plataforma.

Capítulo 6

Protótipo

Este capítulo apresenta a implementação de uma prova de conceito para a arquitetura proposta no Capítulo 5. Essa prova de conceito considera os principais serviços para a integração de dados em cidades inteligentes.

6.1 Detalhes da Implementação

A Figura 6.1 apresenta a arquitetura de microsserviço implementada na prova de conceito, as ferramentas utilizadas e o fluxo dos dados na plataforma. Os microsserviços foram implementados utilizando a linguagem de programação *Go* (2023) e reutilizando ferramentas de software livre. Para realizar comunicação assíncrona entre os microsserviços, utilizamos o *Kafka* (2023), um o sistema de mensagens distribuído altamente escalável e tolerante a falhas que segue o segue o padrão publicar-assinar (pub-sub), onde os produtores publicam mensagens em tópicos (uma categoria ou canal de mensagens) e os consumidores se inscrevem nos tópicos de seu interesse para receber as mensagens. Cada tópico no Kafka é dividido em partições, que são unidades de paralelismo e escalabilidade. Portanto, ao configurar mais de uma partição, é possível ter mais de um consumidor, sendo que cada consumidor recebe mensagem de um grupo específico de partição.

O serviço **Metadata Management** é responsável pelo gerenciamento de coleções de dados e metadados. O modelo de metadados criado foi baseado no *Data Catalog Vocabulary (DCAT) - Version 3* (2023) e possui atributos para identificar as organizações que geraram os dados e para descrever as coleções de dados. O modelo também possui atributos para dar suporte ao gerenciamento de versões de metadados, como ID da versão e data de criação da versão, a fim de registrar como os esquemas evoluem ao longo do tempo e possibilitar a compatibilização das versões utilizando consulta SQL. A Figura 6.2 apresenta o modelo de metadados criado, representado utilizando o modelo de dados relacional para facilitar o entendimento. O esquema JSON utilizado para representar o metadado está no Apêndice A.1.

Para realizar o cadastro, atualização e consulta dos metadados de uma coleção de dados, foi implementado o *endpoint* `/v1/metadata/dataset-name` com o método POST para criação e atualização, e o *endpoint* `/v1/metadata/dataset-name` com o método GET

para consulta. Na requisição do método POST, deve ser enviado no corpo da solicitação a especificação do metadado no formato JSON. Ao receber a requisição, é validado o formato do corpo da mensagem recebido e, em caso de sucesso, a informação é inserida no *MongoDB* (2023), um sistema de gerenciamento de banco de dados NoSQL de código aberto, escalável e flexível. Após inserir no MongoDB, é publicado no tópico **metadata-version** do Kafka uma mensagem para notificar o microsserviço **Data Ingestion Management**.

Para consultar as informações de uma coleção de dados, deve ser enviado como parâmetro de consulta o nome da coleção. Ao receber a requisição, será realizada a busca no MongoDB e o retorno das informações encontradas.

O microsserviço **Data Ingestion Management** é responsável por criar ou atualizar o processo de ingestão de dados, portanto ele está inscrito no tópico **metadata-version** para ser informado sobre a adição ou atualização de uma coleção de dados. Para evitar a concorrência na ingestão de dados entre as coleções de dados, implementamos a criação de um tópico por coleção de dados. Portanto, ao receber uma notificação, o serviço cria o tópico no Kafka e envia uma requisição para **Data Lake Storage** no *endpoint* `/druid/indexer/v1/supervisor` para criar ou atualizar a tarefa de ingestão de dados. Em seguida, é publicada no tópico **compatibility-model** do Kafka uma mensagem para notificar o microsserviço **Data Query**.

O **Data Lake Storage** é responsável por amazenar os dados das coleções de dados. Este serviço foi disponibilizado utilizando o *Apache Druid* (2023) integrado ao *MinIo* (2023). O MinIo é um sistema de armazenamento de objetos de código aberto projetado para armazenar grandes volumes de dados. É altamente escalável, com replicação de dados para garantir a resiliência contra falhas de hardware ou de servidores, compatível com o *Amazon S3* (2023) (*Simple Storage Service*) e oferece recursos de segurança, incluindo autenticação, autorização, criptografia de dados em repouso e em trânsito, para proteger os dados armazenados. O Apache Druid é um banco de dados analítico com processamento de dados em tempo real, escalável, com suporte a consultas complexas (e.g. agregações, filtros e janelas de tempo) em grandes volumes de dados e com integração com ferramentas para análise de dados, como o *Apache Superset* (2023). Assim, configuramos o Druid para armazenar os dados no MinIo.

No **Data Lake Storage**, o Druid é responsável por disponibilizar o *endpoint* `/druid/indexer/v1/supervisor` para criar ou atualizar a tarefa de ingestão de dados e, após criar a tarefa de ingestão, o Druid consome os dados do Kafka e insere no MinIo.

O microsserviço **Data Query** é composto pelo **Data Query Management** e pelo *Presto* (2023). O **Data Query Management** fica inscrito no tópico **compatibility-model** do Kafka e, ao receber uma mensagem, extrai a consulta SQL e cria uma visualização SQL no Presto para a coleção de dados que está armazenada no **Data Lake Storage**, permitindo efetuar consultas SQL nos dados armazenados.

O **Data Visualization** tem como objetivo permitir ao usuário criar e visualizar painéis de dados sobre os dados armazenados. Para isso, utilizamos o *Apache Superset* (2023), uma plataforma de código aberto de visualização de dados e *business intelligence* (BI) que permite aos usuários criar painéis interativos, gráficos e relatórios. Configuramos o Superset conectado ao Presto para que os usuários possam acessar a interface web do

Superset e escrever consultas SQL personalizadas utilizando diferentes coleções de dados em um mesmo painel, gráfico ou relatório.

Após efetuar o cadastro da coleção de dados, é possível começar a enviar dados para plataforma utilizando o *endpoint* `/v1/dataset/dataset-name` do microsserviço **Data Ingestion Producer**. Este *endpoint* aceita o método POST, o corpo da solicitação pode ser no formato JSON ou CSV e deve ser passado no parâmetro de consulta o nome da coleção. Ao receber a solicitação, é executado o *endpoint* `/v1/metadata/dataset-name` do **Metadata Management** para verificar se a coleção existe e se o formato do dado está correto. Existindo a coleção e o dado estando no formato esperado, ele é publicado no tópico **dataset-<name>** do Kafka para que tal dado seja ingerido pelo **Data Lake Storage**.

Todos os microsserviços implementados estão disponíveis como contêineres *Docker* (2023). Usamos o *Google Kubernetes Engine* (2023) para automatizar a implantação dos contêineres e fornecer dimensionamento individual para os serviços, para suportar adequadamente as variações de carga de trabalho. O código fonte da implementação está em https://gitlab.com/interscity/data-integration.

Os demais microsserviços da arquitetura proposta não foram incluídos na prova de conceito. Entendemos que tais serviços podem ser implementados utilizando ferramentas já existentes e, portanto, não necessitam de prova de conceito.

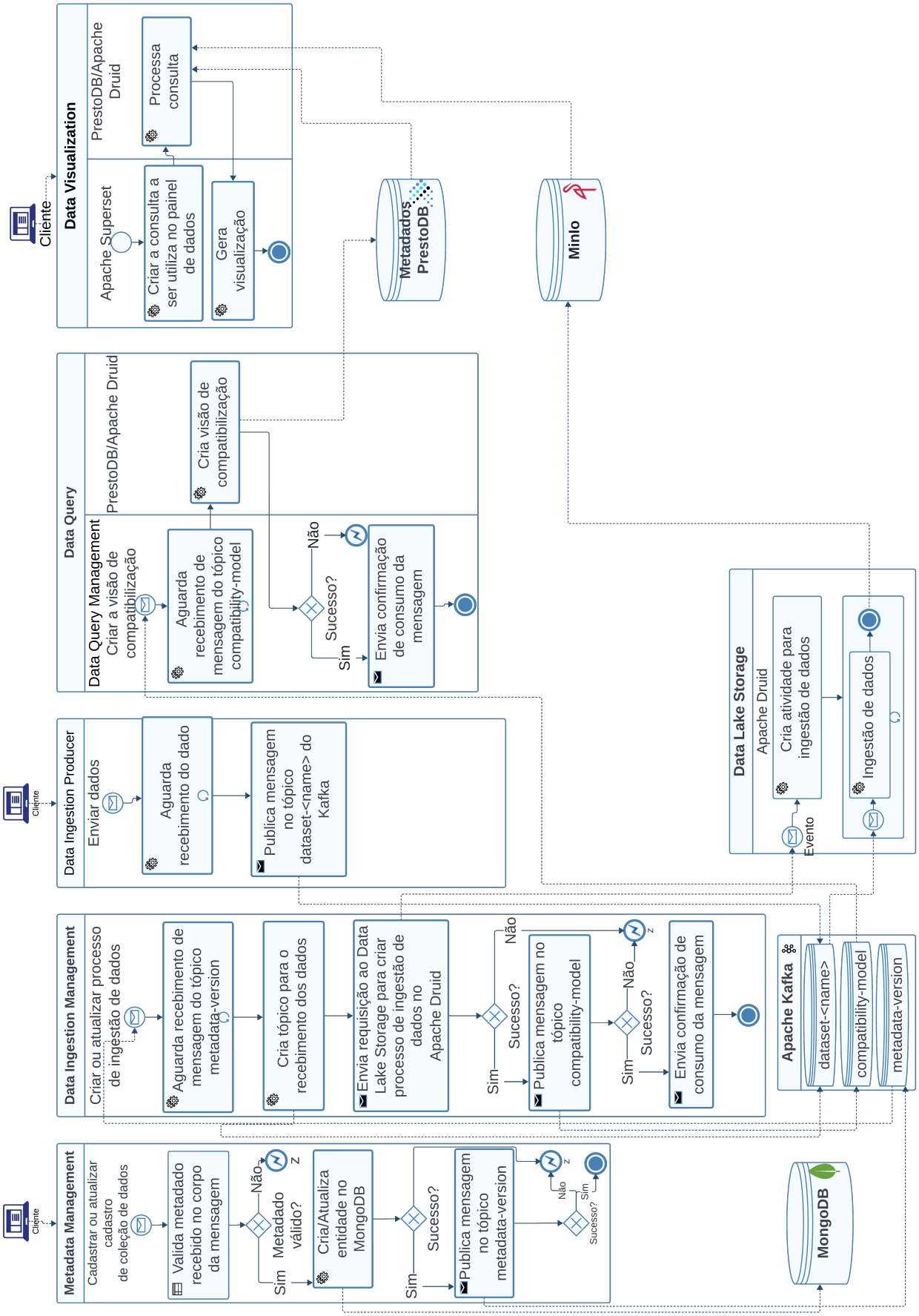

Figura 6.1: *Arquitetura do protótipo*

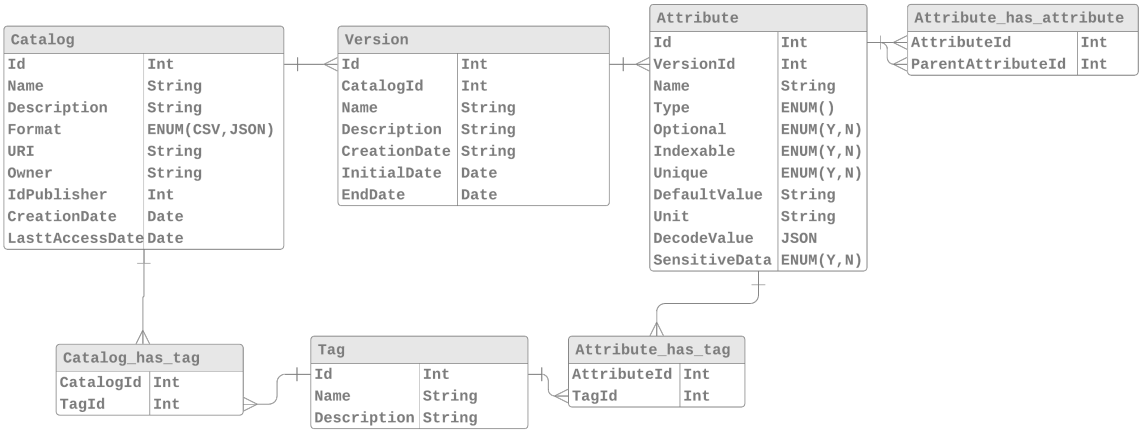

Figura 6.2: *Representação do modelo de metadados utilizando o modelo de dados relacional*

Capítulo 7

Experimentos de Avaliação de Desempenho

Para avaliar o desempenho de um sistema que implementa a arquitetura de software de integração de dados proposta neste trabalho, pode-se usar a metodologia *Cloud Evaluation Experiment Methodology* (CEEM), proposta por Li, OBrien e Zhang (2013) e brevemente apresentada na Seção 2.6. Este capítulo primeiro apresenta diretrizes para a aplicação da CEEM na avaliação de microsserviços da arquitetura. Depois, apresenta os resultados de desempenho obtidos a partir da aplicação das mesmas na avaliação da implementação descrita no Capítulo 6.

7.1 Diretrizes para o Uso da CEEM na Avaliação de Microsserviços da Arquitetura

A seguir, apresentamos diretrizes para a aplicação de cada um dos passos da CEEM para avaliar o desempenho de microsserviços da arquitetura.

7.1.1 Reconhecimento de Requisitos e Identificação de Recursos de Serviço

Deseja-se avaliar a capacidade individual de cada microsserviço para funcionar em condições de carga de trabalho normais e acima do normal. Em particular, pretende-se avaliar a eficácia da autoescalabilidade para suportar um aumento do número de utilizadores ou pedidos simultâneos das principais funcionalidades do microsserviço, mantendo uma qualidade de serviço aceitável. Dessa forma, as principais funcionalidades a serem avaliadas por meio dos experimentos são:

- cadastro de coleção de dados no microsserviço **Metadata Management**;
- ingestão de dados no microsserviço **Data Ingestion Producer**;
- consulta de dados do **Data Lake Storage** pelo microsserviço **Data Query**.

O número de usuários e requisições por intervalo de tempo a serem atendidos por uma instância de microsserviço deve ser definido de acordo com o cenário de uso da aplicação. Esses valores serão utilizados como parâmetros para a execução e análise dos experimentos. Portanto, eles devem ser definidos para cada microsserviço a ser analisado.

Estas diretrizes não incluem cenários experimentais para os microsserviços **API Gateway**, **Auth Server**, **Stream Processing**, **Batch Processing**, **Smart Analyzer** e **Data Visualization**, porque assumimos que as ferramentas de código aberto utilizadas na sua implementação têm bom desempenho. O **Public UI** também não será considerado por ser uma aplicação *front-end*, portanto impacta menos no desempenho geral do sistema que os outros componentes.

7.1.2 Listagem e Seleção de Métricas e *Benchmarks*

Nesta análise, consideraremos o catálogo de métricas apresentado por Li, O'Brien *et al.* (2012c). Quatro métricas do catálogo são particularmente apropriadas para avaliar o desempenho dos microsserviços: utilização de CPU, utilização de memória RAM, latência e o número de solicitações processadas por intervalo de tempo. Essas métricas permitem verificar se o serviço desenvolvido é capaz de atender o número esperado de usuários e requisições com baixa latência.

7.1.3 Listagem e Seleção de Fatores Experimentais

De acordo com Li, O'Brien *et al.* (2012c), as versões do sistema operacional e do gerenciador de contêineres são fatores experimentais a serem considerados. Velocidade do *clock* da CPU e número de núcleos, tipo e capacidade da memória RAM e capacidade de armazenamento também são fatores importantes, pois alterações de software e hardware podem afetar os resultados de desempenho. Para evitar concorrência por recursos computacionais entre as instâncias dos microsserviços, utilizamos contêineres com recursos limitados.

7.1.4 Desenho Experimental e Implementação

Esta seção descreve três experimentos projetados para avaliar o desempenho dos microsserviços. Cada um dos experimentos deve ser executado com as três seguintes configurações diferentes:

- **Configuração 1**: execução com uma única instância do microsserviço com autoescalonamento desabilitado, e uma carga de trabalho que aumenta gradativamente ao longo do tempo até atingir o número máximo esperado para o sistema. O objetivo nesta configuração é medir o desempenho do microsserviço sob cargas de trabalho normais.
- **Configuração 2**: execução do microsserviço com autoescalonamento habilitado e uma carga de trabalho variando entre os valores do limite inferior e superior suportados por um determinado número fixo de instâncias. Nessa configuração, o objetivo é avaliar a capacidade do microsserviço de se autoajustar à demanda atual,

aumentando ou diminuindo o número de instâncias de acordo com as variações da carga de trabalho.

- **Configuração 3**: execução com uma única instância do microsserviço com auto-escalonamento desabilitado, e uma carga de trabalho que aumenta gradualmente ao longo do tempo (tanto no número de usuários quanto no número de solicitações simultâneas por unidade de tempo), até que o uso de CPU ou RAM seja perto de 100%. Nessa configuração, o objetivo é identificar o número máximo de usuários e solicitações simultâneas que uma única instância pode atender.

Os experimentos que devem ser executados em cada uma das configurações acima são os seguintes:

- **Experimento 1: avaliar tempo de resposta para cadastro de coleção de dados**
  Inicie uma única instância do microsserviço **Metadata Management** e simule a execução simultânea de solicitações de criação de metadados para diferentes coleções de dados, variando o número de usuários simulados e solicitações ao longo do tempo de acordo com a configuração do experimento que está sendo executado.

- **Experimento 2: avaliar tempo de resposta na ingestão de dados**
  Inicie uma única instância do microsserviço de **Data Ingestion Producer** e gere requisições variando o número de solicitações ao longo do tempo de acordo com a configuração que está sendo executada.

- **Experimento 3: avaliar tempo de resposta de consultas de cados**
  Inicie uma instância do **DataQuery**. Use um simulador para gerar e executar consultas com filtros e agregações aleatórias sobre as visualizações pré-existentes, variando o número de usuários simulados e solicitações ao longo do tempo de acordo com a configuração do experimento que está sendo executado.

Durante a execução dos experimentos, informações sobre o uso de CPU e RAM, tempo de processamento da solicitação e número de mensagens processadas por unidade de tempo (*throughput*) devem ser coletadas e registradas. O número de repetições de cada experimento deve ser definido levando em consideração os recursos disponíveis, a sensibilidade e a confiança desejadas dos índices de desempenho a serem obtidos nas medições. Geralmente, a sensibilidade aumenta com o número de repetições do experimento.

7.1.5 Análise Experimental

Para analisar os dados coletados nos experimentos e extrair conclusões, é altamente recomendável o uso de métodos estatísticos para garantir a robustez (Li, OBrien e Zhang, 2013). No entanto, mesmo ferramentas gráficas simples (por exemplo, gráficos de pontos, histogramas e gráficos de caixa) mostrando o tempo de resposta, taxa de transferência de dados, uso de CPU e uso de RAM ao longo do tempo podem ajudar a visualizar o quão bem um microsserviço se ajusta às variações de carga de trabalho.

7.2 Análise de Desempenho do Protótipo

Os experimentos de avaliação de desempenho do protótipo descrito no Capítulo 6 seguindo as diretrizes apresentadas na Seção 7.1 foram executados no *Google Kubernetes Engine* (2023), um serviço de gerenciamento de contêineres oferecido pelo *Google Cloud Platform* (2023) que gerencia automaticamente os recursos de computação, como CPU e memória, em resposta ao tráfego e à carga de trabalho. O consumo de recursos foi limitado a 512Mb de memória RAM e duas CPUs por instância do serviço, também chamada de *pod* no *Google Kubernetes Engine* (2023).

Para medir o tempo de resposta e degradação dos microsserviços **Metadata Management** e **Data Ingestion Producer**, realizamos experimentos isolados para cada microsserviço com uma única instância ativa e com o dimensionamento automático desabilitado. Rodamos cada experimento por cinco minutos e o repetimos 15 vezes, a fim de minimizar os efeitos de variáveis incontroláveis inerentes ao ambiente em que os experimentos foram realizados. O experimento foi realizado usando dados reais do Sistema de Informações Hospitalares do SUS (SIHSUS) e do Sistema de Informação sobre Mortalidade Declaração de Óbitos Fetais (SIM-DOFET) disponibilizados pelo Instituto de Comunicação e Informação Científica e Tecnológica em Saúde (ICICT/FIOCRUZ) e obtidos por meio do site https://pcdas.icict.fiocruz.br/conjunto-de-dados/.

Executamos o teste de carga simulando requisições concorrentes com o *Apache Jmeter* (2023), para cargas de trabalho variando de 200 a 2000 requisições por segundo, e coletamos o tempo de resposta. O tempo de resposta, neste caso, é a diferença entre a data e hora de envio da requisição e a data e hora de chegada da resposta. Cada gráfico de caixa nas Figuras 7.1 e 7.2 resume os resultados das 15 rodadas dos experimentos usando um número fixo de solicitações por segundo. A partir de cada rodada, um tempo médio de resposta foi calculado. O gráfico de caixa mostra o mínimo, o primeiro quartil, a mediana (linha verde), o terceiro quartil e o máximo dos tempos médios de resposta das rodadas.

O **Metadata Management** apresentou uma mediana de 369 milissegundos para o cenário de duzentas requisições simultâneas por segundo e de 1215 milissegundos para o cenário de duas mil requisições por segundo. O tempo máximo de resposta foi de 2 segundos. A melhor mediana para o **Metadata Management** foi de 368 milissegundos e ocorreu para 200 solicitações simultâneas; no cenário de maior carga, a mediana foi de 1998 milissegundos. O menor tempo foi de 180 milissegundos e ocorreu no cenário de menor carga. O **Data Ingestion Producer** apresentou a mediana de 240 milissegundos e mínima de 180 milissegundos no cenário de menor carga. No cenário de maior carga, a mediana foi de 1039 milissegundos, mínimo de 253 e máximo de 1829 milissegundos. Para os dois microsserviços, o tempo médio de resposta desconsiderando valores discrepantes permaneceu abaixo de 2 segundos para todos os cenários experimentais, o que é um excelente resultado em aplicações para cidades inteligentes, de acordo com de M. Del Esposte *et al.* (2019).

Para medir o tempo de processamento das mensagens consumidas pelos microsserviços **Data Ingestion Management** e **Data Query Management**, realizamos experimentos isolados para cada microsserviço usando uma única instância ativa e com o dimensionamento automático desabilitado. Inserimos nos tópicos `metadata-version` e `compatibility-`

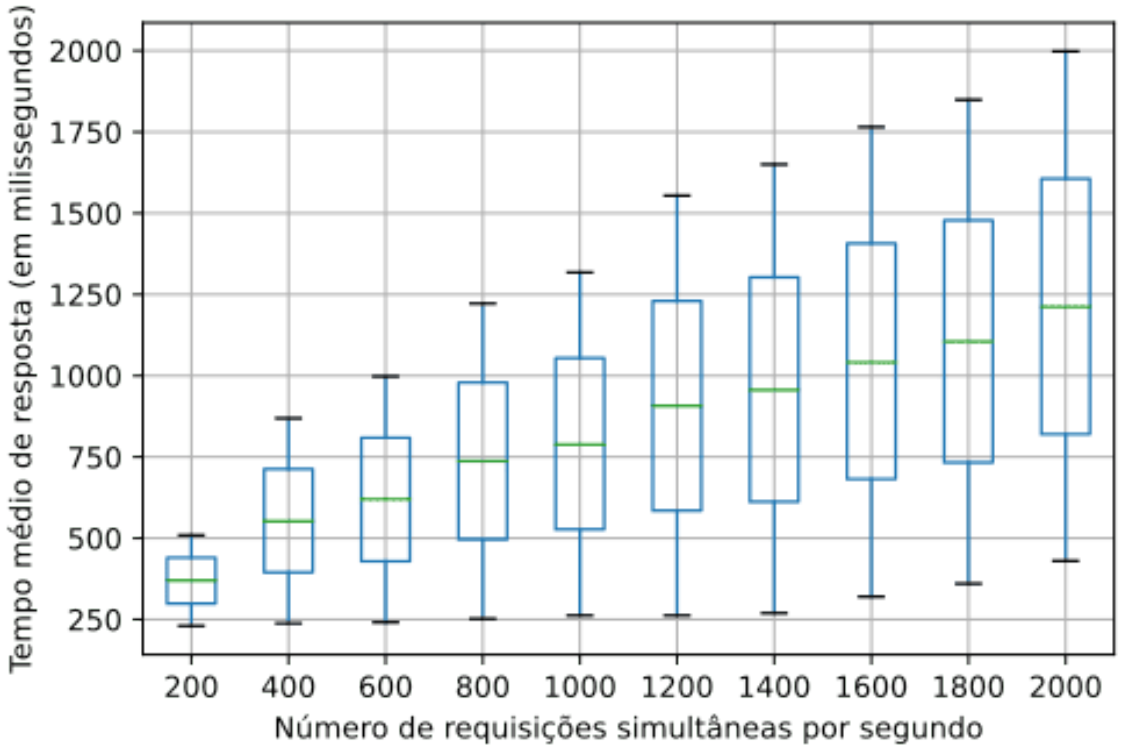

Figura 7.1: *Tempo de resposta para o microsserviço Metadata Management*

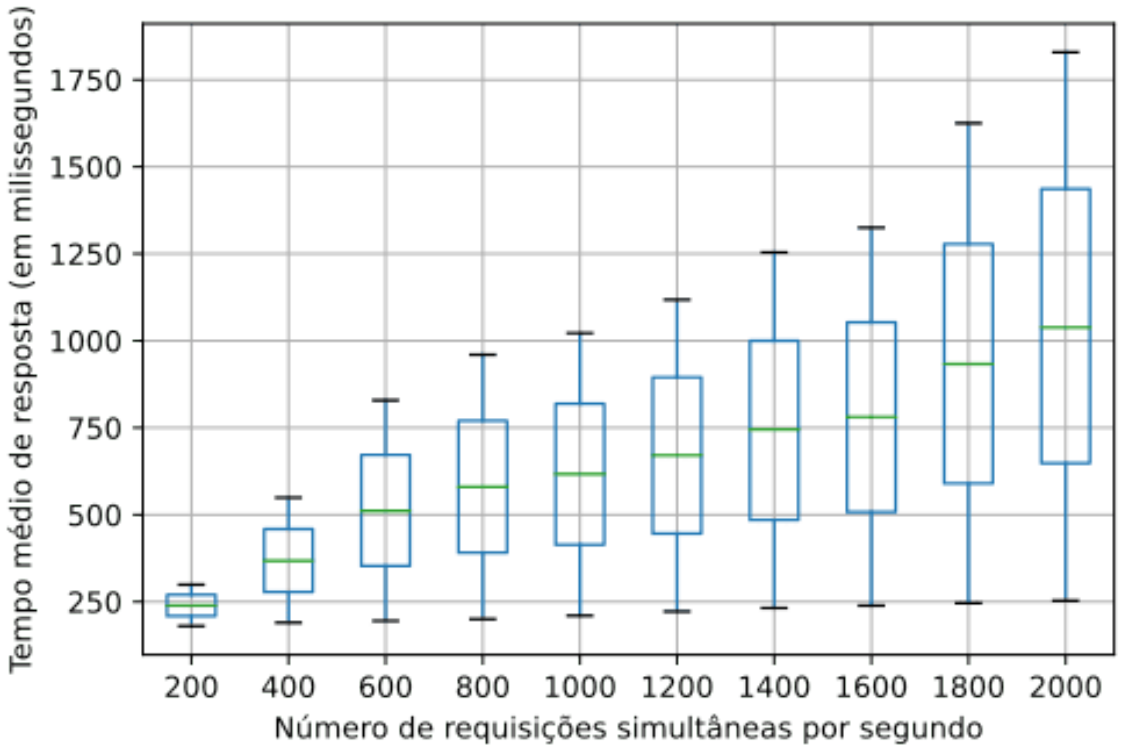

Figura 7.2: *Tempo de resposta para o microsserviço Data Ingestion Producer*

`model` 10 mil mensagens. Iniciamos os serviços para realizar o consumo dos tópicos em momentos distintos. Rodamos cada experimento por 1 minuto e o repetimos 15 vezes. Os gráficos de caixa nas Figuras 7.3 e 7.4 resumem os resultados das 15 rodadas dos experimentos. O **Data Ingestion Management** levou em média 554 milissegundos para processar uma mensagem, sendo seu maior e menor tempo 994 e 239 milissegundos, respectivamente. Já o **Data Query Management** levou em média 453 milissegundos para processar uma mensagem, sendo o maior e menor tempo 794 e 182 milissegundos, respectivamente.

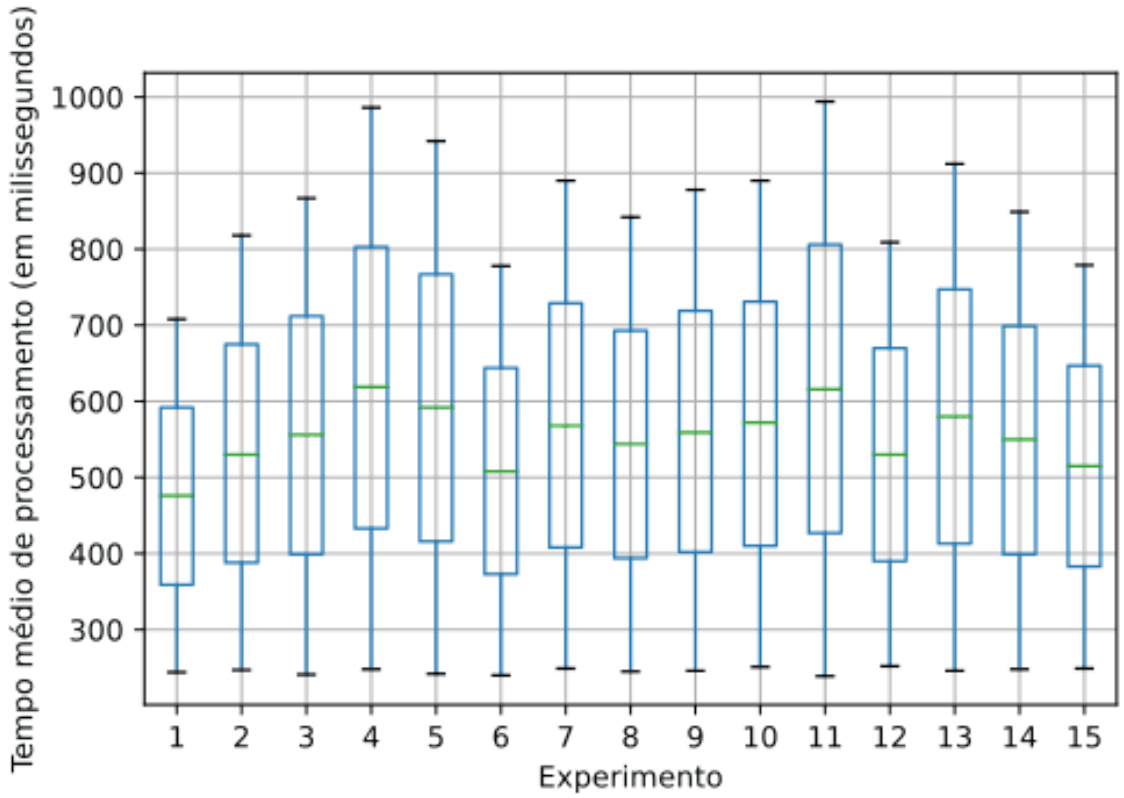

Figura 7.3: *Tempo de processamento de mensagens pelo microserviço Data Ingestion Management*

Para avaliar a escalabilidade horizontal do **Metadata Management**, configuramos o serviço com no mínimo uma réplica e no máximo três e definimos a métrica de escalabilidade com base na utilização de CPU, de forma que o valor desejado de utilização de CPU fosse 40%. Disparamos requisições simultâneas, variando de 1000 a 3000 requisições por segundo durante 350 segundos. Cada vez que a utilização de CPU atingia os 40%, um novo *pod* (instância do serviço) foi instanciado. Além disso, inversamente, toda vez que o uso médio da CPU de todos os contêineres de um microsserviço tornou-se inferior a 40%, um dos *pods* foi destruído e sua carga de trabalho foi transferida para os contêineres restantes. Assim, o sistema aumentou ou diminuiu o número de instâncias por serviço equilibrando a carga de trabalho para corresponder ao uso médio de CPU desejado. A Figura 7.5 mostra a criação e a destruição dos contêineres do microsserviço **Metadata Management** em resposta às flutuações de demanda.

Para avaliar a escalabilidade horizontal do **Data Ingestion Producer**, configuramos o

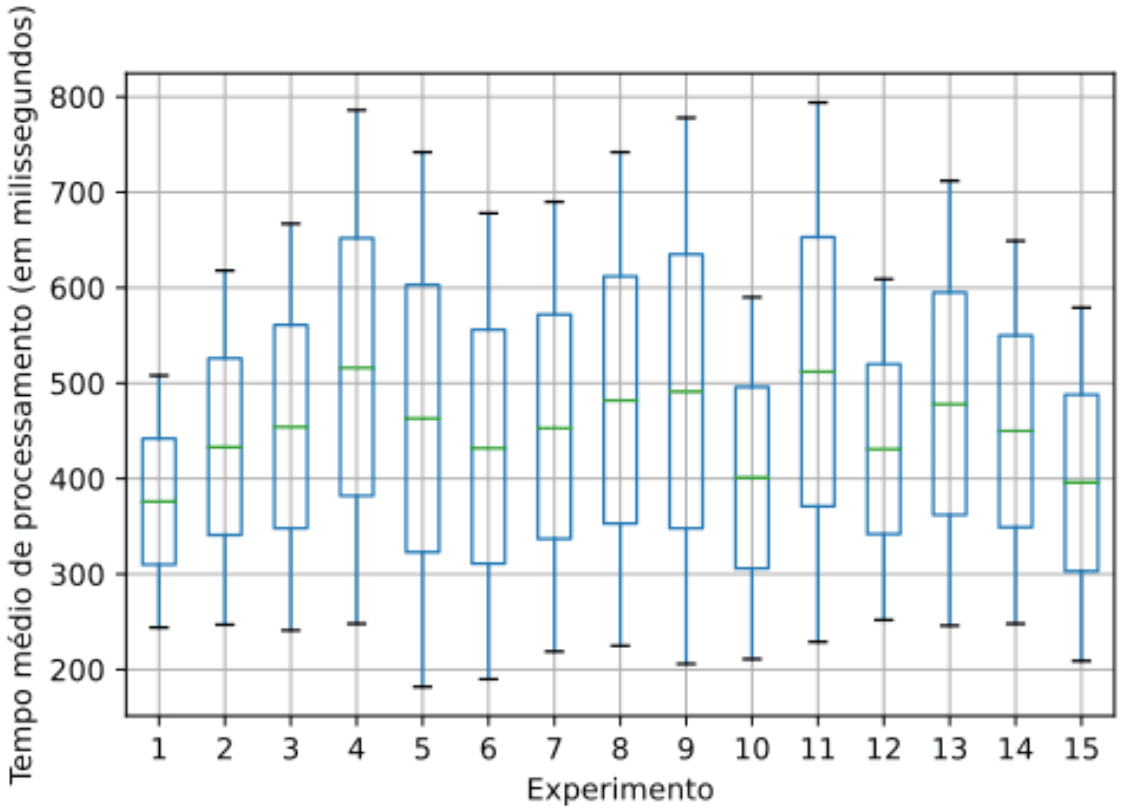

Figura 7.4: *Tempo de processamento de mensagens pelo microsserviço Data Query Management*

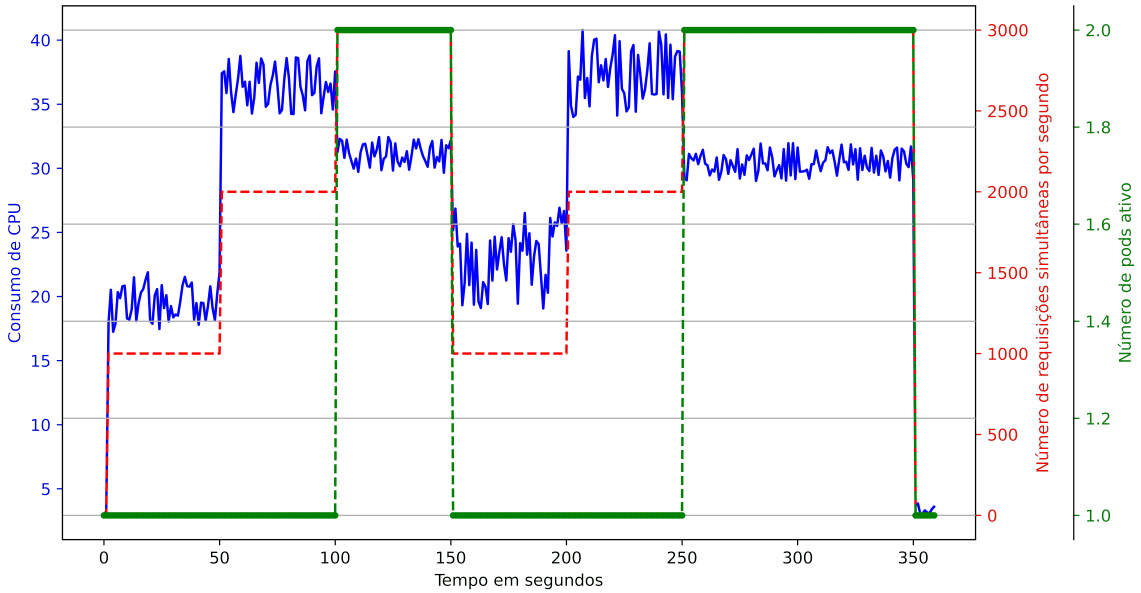

Figura 7.5: *Autoescalonamento do microsserviço Metadata Management*

serviço com no mínimo uma réplica e no máximo três e definimos a métrica de escalabilidade com base na utilização de CPU de forma que o valor desejado de utilização de CPU fosse 50%. Disparamos requisições simultâneas, variando de 1000 a 3000 requisições por

segundo durante 350 segundos. Toda vez que o uso médio da CPU de todos os contêineres de um microsserviço ultrapassou 50%, um novo *pod* foi instanciado e a carga de trabalho foi balanceada entre os *pod* ativos. Além disso, inversamente, toda vez que o uso médio da CPU de todos os contêineres de um microsserviço tornou-se inferior a 50%, um dos *pods* foi destruído e sua carga de trabalho foi transferida para os contêineres restantes. Assim, o sistema aumentou ou diminuiu o número de instâncias por serviço equilibrando a carga de trabalho para corresponder ao uso médio de CPU desejado. A Figura 7.6 mostra a criação e a destruição dos contêineres do microsserviço **Data Ingestion Producer** em resposta às flutuações de demanda.

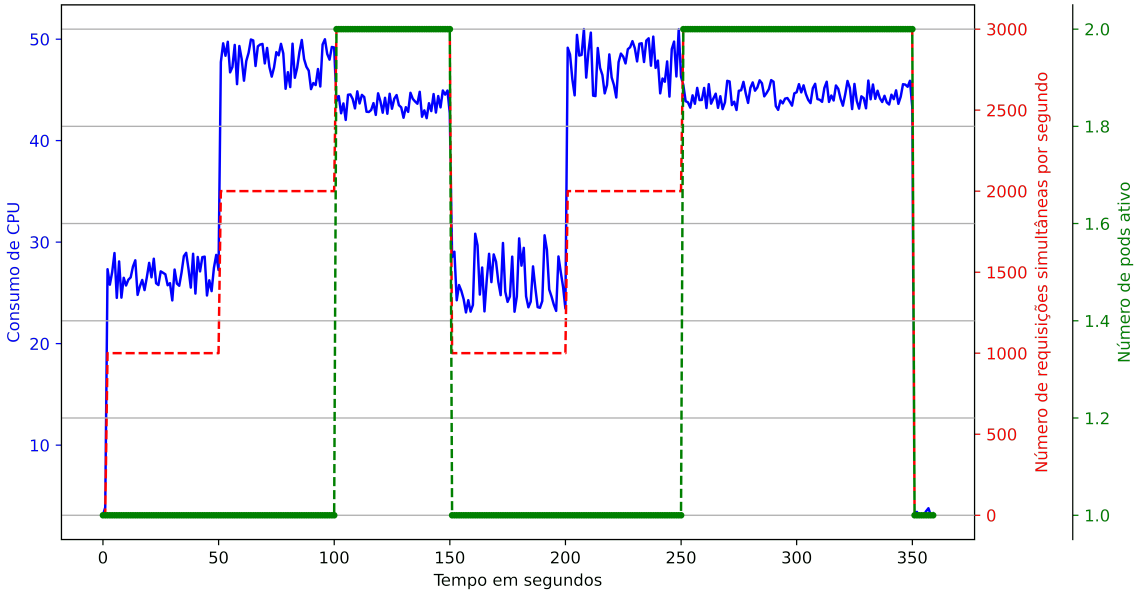

Figura 7.6: *Autoescalonamento do microserviço Data Ingestion Producer*

Para avaliar a escalabilidade horizontal dos microsserviços **Data Ingestion Management** e **Data Query Management**, deixamos a atribuição de partições para os consumidores dos tópicos Kafka no modo automático e configuramos o serviço com escalonamento automático para manter uma latência média no consumo das mensagens de 3 segundos, ou seja, o tempo médio entre a data final de processamento da mensagem e a data de criação deveria ser de 3 segundos. Iniciamos o teste com o tópico **metadata-version** do Kafka zerado e uma instância do consumidor (**Data Ingestion Management**), fomos aumentando o número de mensagens inseridas por segundo até atingir mil mensagens por segundo e, depois, fomos reduzindo o número de mensagens. Rodamos o experimento por 360 segundos. A Figura 7.7 mostra a relação entre o número de instâncias ativas do serviço e o tempo médio para o processamento de uma mensagem. É possível ver que o número de instâncias aumenta e diminui conforme a variação do tempo médio para o processamento da mensagem.

7.3 Considerações Finais

Desenhamos e executamos um conjunto de experimentos seguindo a metodologia CEEM para avaliar o desempenho dos microsserviços implementados para a arquitetura

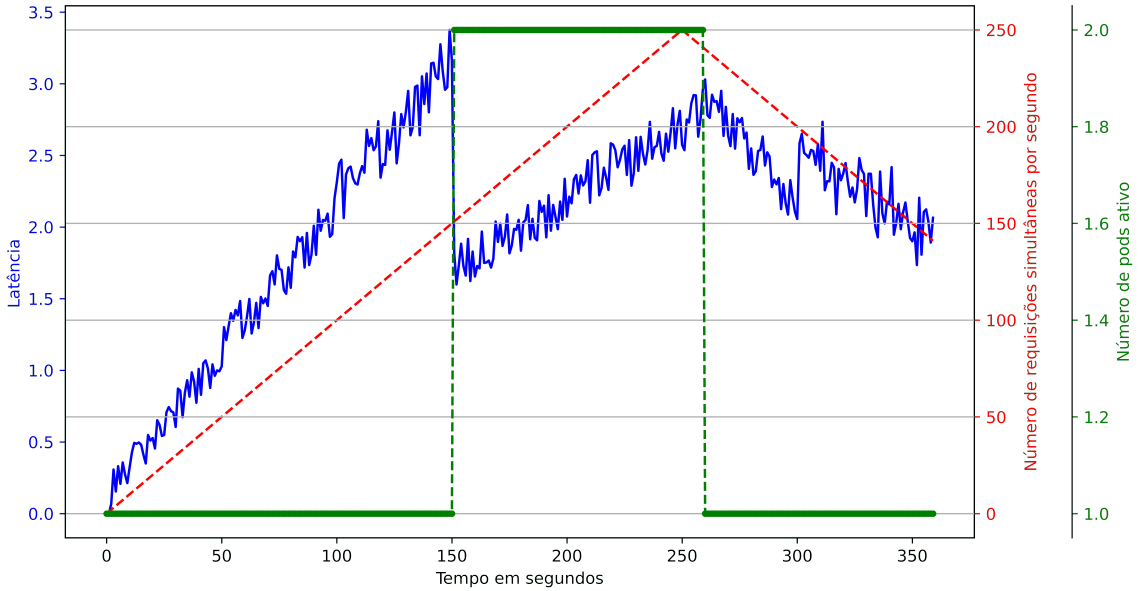

Figura 7.7: *Autoescalonamento do microsserviço Data Ingestion Management*

proposta sob condições de carga de trabalho normais e acima do normal. Na avaliação experimental, os microsserviços implementados apresentaram tempos de resposta dentro do esperado e foram escalonados horizontalmente de acordo com as oscilações da carga de trabalho. Estes resultados mostram que a arquitetura pode atender às demandas de desempenho de uma cidade inteligente.

Capítulo 8

Conclusões

A coleta, limpeza, integração, transformação e análise de dados de fontes distintas permitem um melhor entendimento das deficiências das cidades, auxiliando a tomada de decisão baseada em evidências e o desenvolvimento de políticas públicas visando ao melhor aproveitamento dos recursos disponíveis e a melhoria na qualidade de vida da população.

Sistemas de integração de dados para cidades inteligentes já foram objeto de vários trabalhos de pesquisa, mas ainda há questões em aberto e espaço para melhorias. Exemplos de deficiências encontradas são o suporte insuficiente para gerenciamento de metadados e a falta de facilidades para a consulta a dados voltadas a usuários não especialistas, o que dificulta a descoberta e a reutilização de dados urbanos.

Neste trabalho, fizemos uma revisão da literatura sobre sistemas de integração de dados para cidades inteligentes e, a partir dela, identificamos como principais requisitos funcionais a ingestão, gerenciamento de metadados, processamento, aprendizado de máquina, análise, visualização e acesso externo aos dados. Já a escalabilidade, disponibilidade, segurança e privacidade são os principais requisitos não funcionais para uma plataforma de integração e gerenciamento de dados heterogêneos de cidades inteligentes.

Propomos uma nova arquitetura baseada em microsserviços com base nos requisitos identificados na literatura para orientar o desenvolvimento de plataformas de software de integração de dados para cidades inteligentes.

8.1 Contribuições

A arquitetura proposta neste trabalho estende os trabalhos relacionados e oferece recursos para ingestão de dados e integração física. Ela tem recursos exclusivos como: um único ponto de acesso a microsserviços por aplicativos externos; um serviço de autenticação e autorização de acesso; uma interface centralizadora dos serviços; a criação de novas coleções de dados com base nas existentes; e compatibilização de dados em coleções que sofreram alterações estruturais ou semânticas ao longo do tempo, utilizando o histórico de modificação de metadados e regras de mapeamento.

Apresentamos também diretrizes para avaliar o desempenho (em termos do tempo de resposta/processamento dos serviços, número de requisições processadas por uma única instância do serviço e autoescalonamento dos serviços) de sistemas que implementam a arquitetura proposta. As diretrizes seguem a metodologia CEEM, usada para avaliar sistematicamente o desempenho de serviços em nuvem por meio de experimentos.

Para avaliar e validar a arquitetura proposta, implementamos uma prova de conceito de código aberto e foram realizados experimentos para avaliar a latência, tempo de resposta e o autoescalonamento dos serviços implementados. Os resultados mostram que os microsserviços escalam automaticamente, ou seja, aumentam e diminuem o número de instâncias conforme a variação da carga de entrada. Nos experimentos realizados, a mediana da latência e tempo de respostas medidos durante os experimentos ficaram abaixo de dois segundos para o cenário de maior carga dos microsserviços **Metadata Management**, **Data Ingestion Producer** e **Data Query Management**, mostrando que a arquitetura proposta pode atender às demandas de desempenho de uma cidade inteligente.

8.2 Sugestões para Pesquisas Futuras

Como sugestão para trabalhos futuros em Integração de Dados para Cidades Inteligentes, identificamos algumas possíveis extensões:

- Desenvolvimento dos demais microsserviços apresentados na arquitetura proposta e a integração com uma aplicação de cidade inteligente;
- Realização de estudos de viabilidade em diferentes domínios de aplicação, avaliando o comportamento segundo a visão de especialistas do domínio para identificar a necessidade de novas funcionalidades ou ajustes nas funcionalidades existentes;
- Aplicação de ontologias para facilitar a busca e utilização dos dados;
- Automatização da compatibilização de versões dos esquemas de dados, removendo a necessidade do usuário definir o modelo de compatibilização.

Apêndice A

Esquema dos Metadados

Conforme descrito no Capítulo 6, o serviço **Metadata Management** é responsável pelo gerenciamento de coleções de dados e metadados. O modelo de metadados criado para ele foi baseado no *Data Catalog Vocabulary (DCAT) - Version 3* (2023) e possui atributos para identificar as organizações que geraram os dados, descrever as coleções de dados e dar suporte ao gerenciamento de versões de metadados, permitindo registrar como os esquemas evoluem ao longo do tempo. O esquema JSON utilizado para representar os metadados é mostrado a seguir.

```
{
  "$schema": "http://json-schema.org/draft-07/schema",
  "definitions": {
    "tags": {
      "type": "array",
      "properties": {
        "tag": {
          "type": "string",
          "minLength": 1
        },
        "description": {
          "type": "string",
          "minLength": 1
        }
      },
      "required": [
        "tag"
      ],
      "additionalProperties": false
    },
    "field": {
      "type": "object",
      "title": "Atributo da colecao de dados",
      "description": "Informacoes detalhadas sobre o atributo da colecao de
          dados",
      "required": [
        "name",
        "description",
```

```
28          "type"
29        ],
30        "properties": {
31          "name": {
32            "$id": "#/properties/fields/properties/name",
33            "type": "string",
34            "minLength": 1,
35            "title": "Atributo",
36            "description": "Nome do atributo",
37            "examples": [
38              "P_RACA"
39            ]
40          },
41          "description": {
42            "$id": "#/properties/fields/properties/description",
43            "type": "string",
44            "minLength": 1,
45            "title": "Descricao",
46            "description": "Descricao do atributo",
47            "examples": [
48              "Raca do paciente"
49            ]
50          },
51          "type": {
52            "$id": "#/properties/fields/properties/type",
53            "type": "string",
54            "enum": [
55              "STRING",
56              "NUMBER",
57              "BOOLEAN",
58              "NULL",
59              "OBJECT",
60              "ARRAY"
61            ],
62            "title": "Tipo",
63            "description": "Tipo do atributo",
64            "examples": [
65              "STRING"
66            ]
67          },
68          "optional": {
69            "$id": "#/properties/fields/properties/optional",
70            "type": "boolean",
71            "title": "Atributo Opcional",
72            "default": true,
73            "description": "O atributo pode ser nulo?"
74          },
75          "indexable": {
76            "$id": "#/properties/fields/properties/indexable",
77            "type": "boolean",
78            "title": "Indexar?",
79            "default": false,
80            "description": "Criar index pelo atributo?"
81          },
82          "unique": {
83            "$id": "#/properties/fields/properties/unique",
```

```
84            "type": "boolean",
85            "title": "Valor Unico?",
86            "default": false,
87            "description": "Poder existir duplicidade?"
88          },
89          "sensitive": {
90            "$id": "#/properties/fields/properties/sensitive",
91            "type": "boolean",
92            "title": "Dado sensivel?",
93            "default": false,
94            "description": "Dado sensivel?"
95          },
96          "default_value": {
97            "$id": "#/properties/fields/properties/default_value",
98            "type": "string",
99            "minLength": 1,
100           "title": "Valor default",
101           "description": "Valor default para o atributo",
102           "default": "",
103           "examples": [
104             "Nao preenchido",
105             "0",
106             "false"
107           ]
108         },
109         "unit": {
110           "$id": "#/properties/fields/properties/unit",
111           "type": "string",
112           "minLength": 1,
113           "title": "Unidade de medida",
114           "description": "Unidade de medida dos valores do atributo",
115           "default": "",
116           "examples": [
117             "m/s",
118             "km/h"
119           ]
120         },
121         "decode_value": {
122           "$id": "#/properties/decode_value",
123           "type": "object",
124           "title": "Mapeamento de valores",
125           "description": "",
126           "default": {},
127           "examples": [
128             {
129               "1": "0 a 12 meses",
130               "2": "1 a 7 anos"
131             }
132           ]
133         },
134         "tags": {
135           "$ref": "#/definitions/tags"
136         },
137         "subschema": {
138           "$id": "#/properties/subschema",
139           "type": "array",
```

```
140           "title": "Atributo complexo",
141           "description": "Informacoes detalhadas sobre os atributos de um
                  atributo",
142           "required": [
143             "items"
144           ],
145           "minItems": 1,
146           "minContains": 1,
147           "items": {
148             "$ref": "#/definitions/field"
149           }
150         }
151       },
152       "additionalProperties": false
153     }
154   },
155   "$id": "http://example.com/root.json",
156   "type": "object",
157   "title": "Metadado",
158   "description": "Informacoes sobre a colecao de dados",
159   "required": [
160     "name",
161     "description",
162     "license",
163     "description_of_changes",
164     "format",
165     "fields",
166     "tags"
167   ],
168   "properties": {
169     "name": {
170       "$id": "#/properties/name",
171       "type": "string",
172       "minLength": 1,
173       "title": "Nome da colecao de dados",
174       "description": "",
175       "examples": [
176         "SIH"
177       ]
178     },
179     "description": {
180       "$id": "#/properties/description",
181       "type": "string",
182       "minLength": 1,
183       "title": "Descricao da colecao de dados",
184       "description": "",
185       "default": {},
186       "examples": [
187         "Sistema de Informacoes Hospitalares do SUS"
188       ]
189     },
190     "description_of_changes": {
191       "$id": "#/properties/description_of_changes",
192       "type": "string",
193       "title": "Descricao das alteracoes realizadas no metadado",
194       "description": "",
```

```
195         "minLength": 1,
196         "examples": [
197           "Alteracao do metadado apos a release 04.09.2020. Alteracao no tipo do
                  atributo P_RACA para string"
198         ]
199       },
200       "format": {
201         "$id": "#/properties/format",
202         "type": "string",
203         "enum": [
204           "CSV",
205           "JSON"
206         ],
207         "title": "Formato do dataset",
208         "description": "",
209         "examples": [
210           "CSV"
211         ]
212       },
213       "license": {
214         "$id": "#/properties/license",
215         "type": "string",
216         "enum": [
217           "PUBLIC",
218           "PRIVATE"
219         ],
220         "title": "licenca dos dados",
221         "description": "",
222         "examples": [
223           "PUBLIC"
224         ]
225       },
226       "fields": {
227         "$id": "#/properties/fields",
228         "type": "array",
229         "title": "Atributos da colecao de dados",
230         "description": "Informacoes detalhadas sobre os atributos da colecao de
                dados",
231         "minItems": 1,
232         "minContains": 1,
233         "items": {
234           "$ref": "#/definitions/field"
235         }
236       },
237       "tags": {
238         "$ref": "#/definitions/tags"
239       }
240     }
241   }
```

Listagem A.1: *Schema JSON para Metadados*

Referências

[*@usponline* 2020] *@usponline*. https://twitter.com/usponline. Acesso em: 31/03/2023. Set. de 2020 (citado na pg. 8).

[Agena 2017] Barbara Tieko Agena. *Acesso a dados baseado em ontologias com NoSQL*. Dissertação de Mestrado. Universidade de São Paulo, Instituto de Matemática e Estatística. 2017 (citado na pg. 16).

[Al Nuaimi *et al.* 2015] Eiman Al Nuaimi, Hind Al Neyadi, Nader Mohamed e Jameela Al-Jaroodi. "Applications of Big Data to smart cities". Em: *Journal of Internet Services and Applications* 6.1 (2015), pg. 25 (citado nas pgs. 1, 5, 6, 8, 9).

[Albino *et al.* 2015] Vito Albino, Umberto Berardi e Rosa Maria Dangelico. "Smart cities: definitions, dimensions, performance, and initiatives". Em: *Journal of Urban Technology* 22.1 (2015), pgs. 3–21 (citado nas pgs. 1, 5).

[*Amazon DynamoDB* 2023] *Amazon DynamoDB*. https://aws.amazon.com/pt/dynamodb/. Acesso em: 31/03/2023. 2023 (citado na pg. 14).

[*Amazon Neptune* 2023] *Amazon Neptune*. https://aws.amazon.com/pt/neptune/. Acesso em: 31/03/2023. 2023 (citado na pg. 14).

[*Amazon S3* 2023] *Amazon S3*. https://aws.amazon.com/pt/s3/. Acesso em: 31/03/2023. 2023 (citado nas pgs. 14, 38).

[*Apache Airflow* 2023] *Apache Airflow*. https://airflow.apache.org/. Último acesso em: 13/10/2023. 2023 (citado na pg. 23).

[*Apache Avro* 2023] *Apache Avro*. https://avro.apache.org. Último acesso em: 13/10/2023. 2023 (citado na pg. 32).

[*Apache Drill* 2023] *Apache Drill*. https://drill.apache.org/. Acesso em: 31/03/2023. 2023 (citado na pg. 33).

[*Apache Druid* 2023] *Apache Druid*. https://druid.apache.org/. Último acesso em: 13/10/2023. 2023 (citado nas pgs. 33, 38).

[*Apache Hadoop* 2023] *Apache Hadoop*. https://hadoop.apache.org/. Acesso em: 31/03/2023. 2023 (citado nas pgs. 22, 24, 34).

[*Apache HBase* 2023] *Apache HBase*. https://hbase.apache.org/. Acesso em: 31/03/2023. 2023 (citado nas pgs. 14, 23).

[*Apache Hive* 2023] *Apache Hive*. https://hive.apache.org/. Acesso em: 31/03/2023. 2023 (citado na pg. 23).

[*Apache Jmeter* 2023] *Apache Jmeter*. https://jmeter.apache.org/. Último acesso em: 13/10/2023. 2023 (citado na pg. 46).

[*Apache knox* 2023] *Apache knox*. https://knox.apache.org/. Último acesso em: 13/10/2023. 2023 (citado na pg. 34).

[*Apache Parquet* 2023] *Apache Parquet*. https://parquet.apache.org/. Último acesso em: 13/10/2023. 2023 (citado na pg. 32).

[*Apache Spark* 2023] *Apache Spark*. https://spark.apache.org/. Acesso em: 31/03/2023. 2023 (citado nas pgs. 23, 24, 33).

[*Apache Spark Machine Learning* 2023] *Apache Spark Machine Learning*. https://spark.apache.org/docs/latest/ml-guide.html. Último acesso em: 13/10/2023. 2023 (citado na pg. 34).

[*Apache Spark Streaming* 2023] *Apache Spark Streaming*. https://spark.apache.org/docs/latest/streaming-programming-guide.html. Último acesso em: 13/10/2023. 2023 (citado na pg. 34).

[*Apache Superset* 2023] *Apache Superset*. https://superset.apache.org/. Acesso em: 31/03/2023. 2023 (citado nas pgs. 34, 38).

[*Apache Syncope* 2023] *Apache Syncope*. https://syncope.apache.org/. Último acesso em: 13/10/2023. 2023 (citado na pg. 34).

[*Auth0* 2023] *Auth0*. https://auth0.com/opensource. Último acesso em: 13/10/2023. 2023 (citado na pg. 34).

[Banni *et al.* 2022] Maicon Banni, Isabel Rosseti e Daniel de Oliveira. "Hurricane: um serviço para gerência de dados de aplicações de cidades inteligentes". Em: *Anais do XXXVII Simpósio Brasileiro de Bancos de Dados*. Sociedade Brasileira de Computação, 2022, pgs. 151–163 (citado nas pgs. 23, 25, 27).

[Calvanese *et al.* 2013] D. Calvanese *et al.* "Optique: OBDA solution for Big Data". Em: *The Semantic Web: ESWC 2013 Satellite Events*. Ed. por Philipp Cimiano, Miriam Fernández, Vanessa Lopez, Stefan Schlobach e Johanna Völker. Springer Berlin Heidelberg, 2013, pgs. 293–295 (citado na pg. 16).

[*Cassandra* 2023] *Cassandra*. https://cassandra.apache.org/. Acesso em: 31/03/2023. 2023 (citado nas pgs. 14, 23).

[Cheng *et al.* 2015] B. Cheng, S. Longo, F. Cirillo, M. Bauer e E. Kovacs. "Building a Big Data platform for smart cities: experience and lessons from Santander". Em: *2015 IEEE International Congress on Big Data*. 2015, pgs. 592–599 (citado nas pgs. 2, 22, 25, 27–29).

[Consoli *et al.* 2015] Sergio Consoli *et al.* "A smart city data model based on semantics best practice and principles". Em: *2015WWW '15 Companion: Proceedings of the 24th International Conference on World Wide Web*. Association for Computing Machinery, 2015 (citado nas pgs. 2, 22, 25, 27, 28).

[Costa e Santos 2017] Carlos Costa e Maribel Yasmina Santos. "The SusCity Big Data Warehousing approach for smart cities". Em: *Proceedings of the 21st International Database Engineering & Applications Symposium*. IDEAS 2017. Association for Computing Machinery, 2017, pgs. 264–273 (citado nas pgs. 2, 23, 25, 27, 28).

[*Couchbase* 2023] *Couchbase*. https://www.couchbase.com/. Acesso em: 31/03/2023. 2023 (citado na pg. 14).

[*Data Catalog Vocabulary (DCAT) - Version 3* 2023] *Data Catalog Vocabulary (DCAT) - Version 3*. https://www.w3.org/TR/vocab-dcat-3/. Último acesso em: 13/10/2023. 2023 (citado nas pgs. 33, 35, 37, 55).

[de M. Del Esposte *et al.* 2019] Arthur de M. Del Esposte *et al.* "Design and evaluation of a scalable smart city software platform with large-scale simulations". Em: *Future Generation Computer Systems* 93 (2019), pgs. 427–441 (citado na pg. 46).

[Dobbelaere e Esmaili 2017] Philippe Dobbelaere e Kyumars Sheykh Esmaili. "Kafka versus RabbitMQ: a comparative study of two industry reference publish/subscribe implementations: industry paper". Em: *Proceedings of the 11th ACM International Conference on Distributed and Event-Based Systems*. DEBS '17. Association for Computing Machinery, 2017, pgs. 227–238 (citado na pg. 12).

[*Docker* 2023] *Docker*. https://www.docker.com/. Acesso em: 31/03/2023. 2023 (citado na pg. 39).

[Dragoni *et al.* 2017] Nicola Dragoni *et al.* "Microservices: yesterday, today, and tomorrow". Em: *Present and Ulterior Software Engineering*. Ed. por Manuel Mazzara e Bertrand Meyer. Cham: Springer International Publishing, 2017, pgs. 195–216. isbn: 978-3-319-67425-4. doi: 10.1007/978-3-319-67425-4_12. url: https://doi.org/10.1007/978-3-319-67425-4_12 (citado na pg. 31).

[Elmasri e Navathe 2010] Ramez Elmasri e Shamkant Navathe. *Fundamentals of Database Systems*. 6th. USA: Addison-Wesley Publishing Company, 2010. isbn: 0136086209 (citado na pg. 14).

[Eugster *et al.* 2003] Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui e Anne-Marie Kermarrec. "The many faces of publish/subscribe". Em: *ACM Computing Surveys* 35.2 (jun. de 2003), pgs. 114–131 (citado nas pgs. 12, 13).

[Fan e Bifet 2013] Wei Fan e Albert Bifet. "Mining Big Data: current status, and forecast to the future". Em: *The Association for Computing Machinery's Special Interest Group on Knowledge Discovery and Data Mining* 14.2 (abr. de 2013), pgs. 1–5 (citado na pg. 8).

[Fang 2015] H. Fang. "Managing Data Lakes in Big Data era: what's a Data Lake and why has it became popular in data management ecosystem". Em: *2015 IEEE International Conference on Cyber Technology in Automation, Control, and Intelligent Systems (CYBER)*. 2015, pgs. 820–824 (citado na pg. 17).

[*Flume* 2023] *Flume*. https://flume.apache.org/. Acesso em: 31/03/2023. 2023 (citado na pg. 23).

[Fowler e Lewis 2014] Martin Fowler e James Lewis. *Microservices a definition of this new architectural term*. https://martinfowler.com/articles/microservices.html. Último acesso em 08/02/2023. Mai. de 2014 (citado na pg. 18).

[Gandomi e Haider 2015] Amir Gandomi e Murtaza Haider. "Beyond the hype: Big Data concepts, methods, and analytics". Em: *International Journal of Information Management* 35.2 (2015), pgs. 137–144 (citado na pg. 6).

[*Go* 2023] *Go*. https://go.dev/. Último acesso em: 13/10/2023. 2023 (citado na pg. 37).

[*Google Acadêmico* 2023] *Google Acadêmico*. https://scholar.google.com/. Último acesso em: 13/10/2023. 2023 (citado na pg. 21).

[*Google Cloud Platform* 2023] *Google Cloud Platform*. https://cloud.google.com/. Acesso em: 31/03/2023. 2023 (citado na pg. 46).

[*Google Cloud Storage* 2023] *Google Cloud Storage*. https://cloud.google.com/storage. Acesso em: 31/03/2023. 2023 (citado na pg. 14).

[*Google Kubernetes Engine* 2023] *Google Kubernetes Engine*. https://cloud.google.com/kubernetes-engine. Acesso em: 31/03/2023. 2023 (citado nas pgs. 39, 46).

[*Google Snappy* 2023] *Google Snappy*. https://github.com/google/snappy. Último acesso em: 13/10/2023. 2023 (citado na pg. 32).

[Gorelik 2014] Alex Gorelik. *The Enterprise Big Data Lake*. O'Reilly Media, Inc., 2014 (citado na pg. 17).

[Gruber 2009] Tom Gruber. "Ontology". Em: *Encyclopedia of Database Systems*. Ed. por LING LIU e M. TAMER "OZSU. Boston, MA: Springer US, 2009, pgs. 1963–1965 (citado nas pgs. 15, 16).

[*Gzip* 2023] *Gzip*. https://www.gzip.org/. Último acesso em: 13/10/2023. 2023 (citado na pg. 32).

[Haas *et al.* 1997] L. M. Haas *et al.* "Transforming heterogeneous data with database Middleware: beyond integration". Em: *IEEE Data Engineering Bulletin* 22 (1997), pgs. 31–36 (citado na pg. 15).

[Hai *et al.* 2016] Rihan Hai, Sandra Geisler e Christoph Quix. "Constance: an intelligent Data Lake system". Em: *Proceedings of the 2016 International Conference on Management of Data*. SIGMOD '16. Association for Computing Machinery, 2016, pgs. 2097–2100 (citado na pg. 17).

[Hashem *et al.* 2016] Ibrahim Abaker Targio Hashem *et al.* "The role of big data in smart city". Em: *International Journal of Information Management* 36.5 (2016), pgs. 748–758 (citado nas pgs. 2, 23, 25, 27–29).

[*HTML5* 2023] *HTML5*. https://www.w3.org/TR/2014/REC-html5-20141028/. Último acesso em: 13/10/2023. 2023 (citado na pg. 34).

[*Hue* 2023] *Hue*. https://gethue.com/. Acesso em: 31/03/2023. 2023 (citado na pg. 23).

[Inmon 1996] W.H. Inmon. *Building the Data Warehouse*. Wiley, 1996. isbn: 9780471141617. url: https://books.google.com.br/books?id=9eJQAAAAMAAJ (citado na pg. 16).

[Jarke *et al.* 2013] Matthias Jarke, Maurizio Lenzerini, Yannis Vassiliou e Panos Vassiliadis. *Fundamentals of Data Warehouses*. Springer Science & Business Media, 2013 (citado nas pgs. 14, 16, 17).

[Kadadi *et al.* 2014] A. Kadadi, R. Agrawal, C. Nyamful e R. Atiq. "Challenges of data integration and interoperability in Big Data". Em: *2014 IEEE International Conference on Big Data*. 2014, pgs. 38–40 (citado na pg. 15).

[*Kafka* 2023] *Kafka*. https://kafka.apache.org/. Acesso em: 31/03/2023. 2023 (citado nas pgs. 23, 33, 34, 37).

[Karkouch *et al.* 2016] Aimad Karkouch, Hajar Mousannif, Hassan Al Moatassime e Thomas Noel. "Data quality in internet of things: a state-of-the-art survey". Em: *Journal of Network and Computer Applications* 73 (2016), pgs. 57–81 (citado na pg. 15).

[Khan *et al.* 2013] Z. Khan, A. Anjum e S. L. Kiani. "Cloud based Big Data analytics for smart future cities". Em: *2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing*. 2013, pgs. 381–386 (citado na pg. 1).

[Khine e Wang 2018] Pwint Khine e Zhao Wang. "Data Lake: a new ideology in Big Data era". Em: *Information Technology, Computer Science and Mathematics Web of Conferences* 17 (jan. de 2018), pg. 03025 (citado na pg. 18).

[*kibana* 2023] *kibana*. https://www.elastic.co/pt/kibana. Último acesso em: 13/10/2023. 2023 (citado na pg. 34).

[Kimball 1997] Ralph Kimball. "A dimensional modeling manifesto". Em: *DBMS* 10.9 (ago. de 1997), pgs. 58–70. ISSN: 1041-5173 (citado na pg. 16).

[*Kong Gateway* 2023] *Kong Gateway*. https://konghq.com/products/kong-gateway. Último acesso em: 13/10/2023. 2023 (citado na pg. 34).

[Kreps 2014] Jay Kreps. *Questioning the Lambda architecture. 2014*. https://www.oreilly.com/radar/questioning-the-lambda-architecture/. Acesso em: 31/03/2023. 2014 (citado nas pgs. vii, 9, 11).

[Li, O'Brien *et al.* 2012a] Zheng Li, Liam O'Brien, He Zhang e Rainbow Cai. "A factor framework for experimental design for performance evaluation of commercial cloud services". Em: *4th IEEE International Conference on Cloud Computing Technology and Science Proceedings*. 2012, pgs. 169–176 (citado na pg. 20).

[Li, O'Brien *et al.* 2012b] Zheng Li, Liam O'Brien, He Zhang e Rainbow Cai. "On a catalogue of metrics for evaluating commercial cloud services". Em: *2012 ACM/IEEE 13th International Conference on Grid Computing*. 2012, pgs. 164–173 (citado na pg. 20).

[Li, O'Brien *et al.* 2012c] Zheng Li, Liam O'Brien, He Zhang e Rainbow Cai. "On a catalogue of metrics for evaluating commercial cloud services". Em: *2012 ACM/IEEE 13th International Conference on Grid Computing*. IEEE, 2012, pgs. 164–173 (citado na pg. 44).

[Li, OBrien, Cai *et al.* 2012] Zheng Li, Liam OBrien, Rainbow Cai e He Zhang. "Towards a taxonomy of performance evaluation of commercial cloud services". Em: *2012 IEEE Fifth International Conference on Cloud Computing*. 2012, pgs. 344–351 (citado na pg. 20).

[Li, OBrien e Zhang 2013] Zheng Li, Liam OBrien e He Zhang. "Ceem: a practical methodology for cloud services evaluation". Em: *2013 IEEE Ninth World Congress on Services*. IEEE, 2013, pgs. 44–51 (citado nas pgs. 2, 19, 43, 45).

[Marz 2011] Nathan Marz. *How to beat the CAP theorem*. http://nathanmarz.com/blog/how-to-beat-the-cap-theorem.html. Acesso em: 03/05/2020. 2011 (citado nas pgs. 9, 10).

[Marz e Warren 2015] Nathan Marz e James Warren. *Big Data: Principles and Best Practices of Scalable Realtime Data Systems*. 1st. USA: Manning Publications Co., 2015 (citado nas pgs. vii, 9, 10).

[*Matplotlib* 2023] *Matplotlib*. https://matplotlib.org/. Acesso em: 31/03/2023. 2023 (citado na pg. 23).

[Mazumdar *et al.* 2019] Somnath Mazumdar, Daniel Seybold, Kyriakos Kritikos e Yiannis Verginadis. "A survey on data storage and placement methodologies for Cloud-Big Data ecosystem". Em: *Journal of Big Data* 6.1 (fev. de 2019), pg. 15 (citado nas pgs. 13, 14).

[Mehmood *et al.* 2019] H. Mehmood *et al.* "Implementing Big Data Lake for heterogeneous data sources". Em: *2019 IEEE 35th International Conference on Data Engineering Workshops (ICDEW)*. 2019, pgs. 37–44 (citado nas pgs. 2, 23, 25, 27–29, 35).

[*Microsoft Azure Blob Storage* 2023] *Microsoft Azure Blob Storage*. https://azure.microsoft.com/en-us/products/storage/blobs. Acesso em: 31/03/2023. 2023 (citado na pg. 14).

[*MinIo* 2023] *MinIo*. https://min.io/. Acesso em: 31/03/2023. 2023 (citado nas pgs. 14, 32, 38).

[Mohamed e Al-Jaroodi 2014] N. Mohamed e J. Al-Jaroodi. "Real-time Big Data analytics: applications and challenges". Em: *2014 International Conference on High Performance Computing Simulation (HPCS)*. 2014, pgs. 305–310 (citado na pg. 8).

[*MongoDB* 2023] *MongoDB*. https://www.mongodb.com/. Acesso em: 31/03/2023. 2023 (citado nas pgs. 14, 22, 38).

[*Neo4j* 2023] *Neo4j*. https://neo4j.com/. Acesso em: 31/03/2023. 2023 (citado na pg. 14).

[*NodeJs* 2023] *NodeJs*. https://nodejs.org/en. Último acesso em: 13/10/2023. 2023 (citado na pg. 34).

[*PostgreSQL* 2023] *PostgreSQL*. https://www.postgresql.org/. Último acesso em: 13/10/2023. 2023 (citado na pg. 24).

[*Presto* 2023] *Presto*. https://prestodb.io/. Acesso em: 31/03/2023. 2023 (citado nas pgs. 23, 33, 38).

[*PROV Ontology (PROV-O)* 2023] *PROV Ontology (PROV-O)*. https://www.w3.org/TR/prov-o/. Último acesso em: 13/10/2023. 2023 (citado na pg. 33).

[Psyllidis *et al.* 2015] Achilleas Psyllidis, Alessandro Bozzon, Stefano Bocconi e Christiaan Titos Bolivar. "A platform for urban analytics and semantic data integration in city planning". Em: *Computer-Aided Architectural Design Futures. The Next City - New Technologies and the Future of the Built Environment*. Ed. por Gabriela Celani, David Moreno Sperling e Juarez Moara Santos Franco. Springer Berlin Heidelberg, 2015, pgs. 21–36 (citado nas pgs. 2, 21, 25, 27, 28).

[*Python* 2023] *Python*. https://www.python.org/. Acesso em: 31/03/2023. 2023 (citado nas pgs. 22, 23).

[*RabbitMQ* 2023] *RabbitMQ*. https://www.rabbitmq.com/. Acesso em: 31/03/2023. 2023 (citado na pg. 33).

[Raghavan *et al.* 2020] Subashini Raghavan, Boung Yew Lau Simon, Ying Loong Lee, Wei Lun Tan e Keh Kim Kee. "Data integration for smart cities: opportunities and challenges". Em: *Computational Science and Technology*. Ed. por Rayner Alfred, Yuto Lim, Haviluddin Haviluddin e Chin Kim On. Springer Singapore, 2020, pgs. 393–403 (citado na pg. 1).

[Rathore *et al.* 2016] M. Mazhar Rathore, Awais Ahmad, Anand Paul e Seungmin Rho. "Urban planning and building smart cities based on the Internet of Things using Big Data analytics". Em: *Computer Networks* 101 (2016). Industrial Technologies and Applications for the Internet of Things, pgs. 63–80 (citado nas pgs. 2, 22, 25, 27, 28).

[*React* 2023] *React*. https://react.dev/. Último acesso em: 13/10/2023. 2023 (citado na pg. 34).

[*Redis* 2023] *Redis*. https://redis.io/. Acesso em: 31/03/2023. 2023 (citado na pg. 14).

[Ribeiro e Braghetto 2021] Murilo Borges Ribeiro e Kelly Rosa Braghetto. "A data integration architecture for smart cities". Em: *Anais do XXXVI Simpósio Brasileiro de Bancos de Dados*. Sociedade Brasileira de Computação, 2021, pgs. 205–216 (citado na pg. 3).

[Ribeiro e Braghetto 2022] Murilo Borges Ribeiro e Kelly Rosa Braghetto. "A scalable data integration architecture for smart cities: implementation and evaluation". Em: *Journal of Information and Data Management* 13.2 (set. de 2022). doi: 10.5753/jidm.2022.2485. url: https://sol.sbc.org.br/journals/index.php/jidm/article/view/2485 (citado na pg. 3).

[Ritter 1998] David Ritter. "The Middleware muddle." Em: *SIGMOD Record* 27 (dez. de 1998), pgs. 86–93. doi: 10.1145/306101.306141 (citado na pg. 15).

[Rudnicki *et al.* 2018] Ronald Rudnicki, Alexander P. Cox, Brian Donohue e Mark Jensen. "Towards a methodology for lossless data exchange between NoSQL data structures". Em: *Ground/Air Multisensor Interoperability, Integration, and Networking for Persistent ISR IX*. Ed. por Michael A. Kolodny, Dietrich M. Wiegmann e Tien Pham. Vol. 10635. International Society for Optics e Photonics. SPIE, 2018, pgs. 183–191 (citado na pg. 17).

[Runeson e Höst 2009] Per Runeson e Martin Höst. "Guidelines for conducting and reporting case study research in software engineering". Em: *Empirical Software Engineering* 14.2 (abr. de 2009), pgs. 131–164 (citado na pg. 20).

[Santana *et al.* 2017] Eduardo Felipe Zambom Santana, Ana Paula Chaves, Marco Aurelio Gerosa, Fabio Kon e Dejan S. Milojicic. "Software platforms for smart cities: concepts, requirements, challenges, and a unified reference architecture". Em: *ACM Computing Surveys* 50.6 (nov. de 2017), 78:1–78:37 (citado na pg. 5).

[Sawadogo e Darmont 2020] Pegdwendé Sawadogo e Jérôme Darmont. "On data lake architectures and metadata management". Em: *Journal of Intelligent Information Systems* (jun. de 2020) (citado na pg. 18).

[*Schema.org* 2023] *Schema.org*. https://schema.org/. Último acesso em: 13/10/2023. 2023 (citado na pg. 33).

[*Scikit-learn* 2023] *Scikit-learn*. https://scikit-learn.org/stable/. Último acesso em: 13/10/2023. 2023 (citado na pg. 34).

[Sheth e Larson 1990] Amit P. Sheth e James A. Larson. "Federated Database Systems for managing distributed, heterogeneous, and autonomous databases". Em: *ACM Computing Surveys* 22.3 (set. de 1990), pgs. 183–236 (citado na pg. 15).

[B. N. Silva *et al.* 2018] Bhagya Nathali Silva, Murad Khan e Kijun Han. "Towards sustainable smart cities: a review of trends, architectures, components, and open challenges in smart cities". Em: *Sustainable Cities and Society* 38 (2018), pgs. 697–713 (citado nas pgs. vii, 5, 6).

[J. Silva *et al.* 2021] Jorge Silva, João Gabriel Almeida, Thais Batista e Everton Cavalcante. "Aquedücte: a data integration service for smart cities". Em: *Proceedings of the Brazilian Symposium on Multimedia and the Web*. WebMedia '21. Association for Computing Machinery, 2021, pgs. 177–180 (citado nas pgs. 22, 25, 27, 29).

[T. H. Silva *et al.* 2014] T. H. Silva, P. O. S. Vaz De Melo, J. M. Almeida e A. A. F. Loureiro. "Large-scale study of city dynamics and urban social behavior using participatory sensing". Em: *IEEE Wireless Communications* 21.1 (2014), pgs. 42–51 (citado na pg. 6).

[*Solr* 2023] *Solr*. https://lucene.apache.org/solr/. Acesso em: 31/03/2023. 2023 (citado na pg. 23).

[*Talend* 2023] *Talend*. https://www.talend.com/. Acesso em: 31/03/2023. 2023 (citado na pg. 23).

[*VueJs* 2023] *VueJs*. https://vuejs.org/. Último acesso em: 13/10/2023. 2023 (citado na pg. 22).

[Wiederhold 1999] Gio Wiederhold. "Mediation to deal with heterogeneous data sources". Em: *Interoperating Geographic Information Systems*. Ed. por Andrej Včkovski, Kurt E. Brassel e Hans-Jörg Schek. Springer Berlin Heidelberg, 1999, pgs. 1–16 (citado na pg. 15).

[*World Wide Web Consortium (W3C)* 2023] *World Wide Web Consortium (W3C)*. https://w3c.br/. Último acesso em: 13/10/2023. 2023 (citado na pg. 33).

[Zheng *et al.* 2014] Yu Zheng, Licia Capra, Ouri Wolfson e Hai Yang. "Urban computing: concepts, methodologies, and applications". Em: *ACM Transactions on Intelligent Systems and Technology* 5.3 (set. de 2014), pgs. 1–55. issn: 2157-6904. doi: 10.1145/2629592. url: https://doi.org/10.1145/2629592 (citado na pg. 1).