UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS DE RUSSAS

CURSO DE GRADUAÇÃO EM ENGENHARIA DE SOFTWARE

CLEITON MONTEIRO DA SILVA

PIPELINE DE PROCESSAMENTO DE DADOS DE DISPOSITIVOS MÓVEIS EM TRANSPORTE COLETIVO URBANO

RUSSAS

2022

CLEITON MONTEIRO DA SILVA

PIPELINE DE PROCESSAMENTO DE DADOS DE DISPOSITIVOS MÓVEIS EM TRANSPORTE COLETIVO URBANO

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Software do Campus de Russas da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia de Software.

Orientador: Prof. Ms. Filipe Maciel Roberto

RUSSAS

2022

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

S579p Silva, Cleiton Monteiro da Silva.
Pipeline de processamento de dados de dispositivos móveis em transporte coletivo urbano / Cleiton Monteiro da Silva Silva. – 2022.
47 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Russas, Curso de Engenharia de Software, Russas, 2022.
Orientação: Prof. Me. Filipe Maciel Roberto.

1. Pipeline. 2. Processamento de dados. 3. Dispositivos móveis. 4. Big Data. I. Título.

CDD 005.1

CLEITON MONTEIRO DA SILVA

PIPELINE DE PROCESSAMENTO DE DADOS DE DISPOSITIVOS MÓVEIS EM TRANSPORTE COLETIVO URBANO

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Software do Campus de Russas da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia de Software.

Aprovada em:

BANCA EXAMINADORA

Prof. Ms. Filipe Maciel Roberto (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. Alisson Barbosa de Souza
Universidade Federal do Ceará (UFC)

Prof. Ms. Filipe Fernandes dos Santos Brasil de Matos
Universidade Federal do Ceará (UFC)

AGRADECIMENTOS

A mim por acreditar em mim, por todo o esforço empregado e por conseguir realizar este trabalho.

A toda minha família, em especial a minha mãe Antônia Clenilda, por todo o amor e apoio durante minha trajetória.

A meu orientador Prof. Ms. Filipe Maciel Roberto por conseguir me orientar neste trabalho e por todo o incentivo e ajuda durante os momentos difíceis.

Aos amigos, Alan Sousa, Artur de Castro, Cibele Rodrigues, Herverson de Sousa, Marcos Rogério, Marlo Oliveira, Mateus Franco, Pedro Honorato, Samuel Carvalho e Susana Moreira que fizeram parte de toda minha caminhada na faculdade e na vida.

A todos os professores, servidores e colaboradores da Universidade Federal do Ceará - Campus Russas, por nesse período de curso, proporcionar todo o conhecimento técnico e ajudar na minha evolução de caráter pessoal.

Ao Doutorando em Engenharia Elétrica, Ednardo Moreira Rodrigues, e seu assistente, Alan Batista de Oliveira, aluno de graduação em Engenharia Elétrica, pela adequação do *template* utilizado neste trabalho para que o mesmo ficasse de acordo com as normas da biblioteca da Universidade Federal do Ceará (UFC).

Por fim, a todos que contribuíram de forma direta e indiretamente para a minha graduação. Muito obrigado.

“Nunca deixe que lhe digam que não vale a pena acreditar no sonho que se tem ou que os seus planos nunca vão dar certo ou que você nunca vai ser alguém...”

(Renato Russo)

RESUMO

A cada dia, novos produtos ou serviços são criados buscando suprir uma necessidade existente, ou simplesmente melhorar a forma atual da solução oferecida. Com isso tem-se uma crescente nos volumes dos dados gerados por esses sistemas. Dispositivos móveis com os *smartphones* se tornaram onipresentes na vida cotidiana. Porém, esses dispositivos móveis têm limitações computacionais, com isso, é comum que sistemas modernos façam o uso de computação em nuvem para as tarefas de análise de dados gerados pelos dispositivos. O fluxo completo de análise de dados passa por diversas etapas como: coleta, formatação, transmissão de forma segura, armazenamento, processamento, análise, disponibilização dos resultados. Com isso tem-se o *pipeline* que é um padrão de *design* de *software* que fornece a capacidade de construir e executar uma sequência de operações. Assim, é possível estabelecer ferramentas e soluções para todas as fases do fluxo. Este trabalho apresenta uma proposta de solução para o processamento desses dados por meio da criação de um *pipeline* para processamento de fluxo de dados para aplicações móveis. A solução foi feita usando ferramentas como *Apache Storm*(*Framework* de processamento em fluxo), *Apache Kafka*(*Framework* de processamento em fluxo), *MongoDB*(Banco de dados orientado a documentos), *Express.js*(Framework para Node.js) para a parte de processamento e Flutter(Framework para construir aplicativos multiplataforma) para construção de aplicações móveis. Com o *pipeline* proposto, foi criado um sistema de teste que habilita a notificação de passageiros sobre a proximidade de um veículo de transporte urbano de interesse. Foi demonstrada a competência do *pipeline* para a construção desse padrão de operação. Foi realizado uma série de testes manuais para o cenário proposto de utilização, porém, testes mais robustos precisam ser realizados de modo a comparar desempenho com outras soluções possíveis e escalabilidade.

Palavras-chave: *Pipeline*. Processamento de dados. Dispositivos móveis. *Big Data*.

ABSTRACT

Every day, new products or services are created seeking to meet an existing need, or simply improve the current form of the solution offered. As a result, there is an increase in the volumes of data generated by these systems. Mobile devices like smartphones have become ubiquitous in everyday life. However, these mobile devices have computational limitations, so it is common for modern systems to use cloud computing for data analysis tasks generated by the devices. The complete flow of data analysis goes through several steps such as: collection, formatting, secure transmission, storage, processing, analysis, and availability of results. The pipeline is a software design pattern that provides the ability to build and execute a sequence of operations. Thus, it is possible to establish tools and solutions for all phases of the flow. This work presents a proposal for a solution for processing this data through the creation of a pipeline for processing data streams for mobile applications. The solution was made using tools like Apache Storm(Stream Processing Framework), Apache Kafka(Stream Processing Framework), MongoDB(Document Oriented Database), Express.js(Framework for Node.js) for the part and Flutter(Framework for building cross-platform applications) for building mobile applications. With the proposed pipeline, a test system was created that enables the notification of passengers about the proximity of an urban transport vehicle of interest. The competence of the pipeline for the construction of this operating pattern was demonstrated. A series of manual tests were carried out for the proposed scenario of use, however, more robust tests need to be performed in order to compare performance with other possible solutions and scalability.

Keywords: Pipeline. Data processing. Mobile devices. Big Data.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

API	Interface de programação de aplicações
APK	*Android application package*
BSS	*Basic Service Set*
CSV	*Comma-separated values*
ETL	*Extract, Transform, and Load*
IaaS	*Infraestrutura como Serviço*
ITU	*International Telecommunication Union*
JSON	*JavaScript Object Notation*
PaaS	*Plataforma como Serviço*
SaaS	*Software como Serviço*
SO	Sistema Operacional
TI	Tecnologia da Informação
VMs	Máquinas Virtuais

SUMÁRIO

1 INTRODUÇÃO

Os volumes de dados gerados por sistemas modernos de TI, estão crescendo exponencialmente (BAHGA; MADISETTI, 2019). Assim tem-se a demanda para a criação de mecanismos em larga escala para a coleta, processamento e análise para esse grande volume de dados, com propósito de extrair informações úteis para tomadas de decisões.

A cada dia, novos produtos ou serviços são criados buscando suprir uma necessidade existente, ou , simplesmente, melhorar a forma atual da solução oferecida. Nesse contexto do crescente volume de dados, pode-se citar como o exemplo o Uber[1] que acaba por ultilizar diversos tipos de análise da dados no ramo de transporte. O ponto principal de coleta de dados é o aplicativo que se encontra nos dispositivos de milhares de usuários finais, onde são coletadas diversas informações como localização dos motoristas, percurso, condições de tráfego, etc. Para que, a partir disso, possam ser calculados valores como tarifa, rota, tempo a ser gasto, etc. A equipe de engenharia do Uber postou um artigo (SHIFTEHFAR, 2018) que deixa claro que o volume de dados gerados chega a mais de 100 petabytes por minuto.

Recentemente, os *smartphones* se tornaram onipresentes na vida cotidiana. De acordo com a *International Telecommunication Union* (ITU) (2019), no ano de 2019 existiam cerca de 109 assinaturas de celular para cada 100 pessoas no mundo. *Smartphones* possuem poderosos sensores integrados, como acelerômetro, bússola digital, giroscópio, GPS, microfone e câmera. Todas essas características e fatores como baixo custo, fácil acesso, comodidade, mecanismos de transferências e compartilhamento de dados tornam os dispositivos móveis uma fonte geradora de dados.

Dispositivos móveis têm limitações computacionais, com isso, é comum que sistemas modernos façam o uso de Computação em Nuvem para as tarefas de análise de dados coletados pelos dispositivos. Os sistemas em Nuvem são acessíveis e facilmente modificados para se adaptarem a novas demandas.

1.1 Problemas específicos no tema

O fluxo completo de análise de dados passa por diversas etapas como: coleta, formatação, transmissão de forma segura, armazenamento, processamento, análise e disponibilização dos resultados. Cada serviço traz consigo um contexto diferente, portanto, é necessário se pensar

[1] https://www.uber.com/

como cada solução deve ser projetada para cada fluxo e quais tecnologias devem ser empregadas para cada uma dessas etapas.

1.2 Abordando uma solução

Pipeline é um padrão de projeto de *software* que fornece a capacidade de construir e executar uma sequência de operações. Com esse padrão, é possível estabelecer ferramentas e soluções para todas as fases do fluxo. Nesse sentido, a escolha de tecnologias para as etapas do processo de análise de dados está diretamente relacionada com o contexto apresentado, categoria de entrada de dados e o mecanismo de análise utilizado. Com o entendimento de quais tecnologias podem ser usadas para a análise de diferentes formatos de dados, é possível mapear as ferramentas necessárias para cada etapa de um *pipeline* de coleta de dados de dispositivos móveis em diferentes contextos (BAHGA; MADISETTI, 2019).

Uma possível abordagem seria elaborar um *pipeline* para o processo de coleta de dados partindo de dispositivos móveis com as etapas mais comuns e o uso de tecnologias atuais permite unificar o uso das tecnologias envolvidas de modo a melhorar todo o gerenciamento.

1.3 Objetivo geral

O objetivo deste trabalho é criar, com o uso de tecnologias recentes, um *pipeline* de coleta, armazenamento e processamento de fluxo de dados para aplicações móveis.

1.4 Objetivos específicos

Os objetivos específicos deste trabalho são:

- Realizar um levantamento de tecnologias atuais para o processamento de dados em fluxo;
- Criar um cenário exemplo para uma aplicação móvel que disponibiliza dados para processamento em tempo real no *pipeline*.

1.5 Justificativa

É comum que aplicações móveis façam coleta de dados com o propósito de aplicar simples modelos matemáticos ou, até mesmo, aprendizado de máquina. Inúmeros exemplos

estão no dia a dia, por exemplo, uma aplicação de criação de rotas onde são coletados dados da localização do usuário, assim como informações de trânsito e temporal. Com isso é possível estabelecer uma melhor rota para o cenário atual. A interação de usuário com uma rede social é outro cenário onde a coleta de dados é ponto-chave na criação de sugestões futuras baseados nos dados coletados e processados dessas interações. Logo, essas e outras diversas aplicações, que necessitam da coleta e processamento de dados, podem utilizar o *pipeline* desenvolvido neste trabalho como modelo a ser implantado. Além disso, esse trabalho permitirá o conhecimento e o uso de tecnologias usadas no cenário da problemática.

1.6 Estrutura do trabalho

Este trabalho está estruturado da seguinte forma: O Capítulo 1 apresenta uma introdução ao problema relacionado à pesquisa. O Capítulo 2 apresenta a fundamentação teórica e assuntos necessários para o entendimento desta pesquisa. O Capítulo 3 define alguns trabalhos relacionados. No Capítulo 4 é apresentado a proposta de solução para o problema. O Capítulo 5 apresenta o cenário exemplo e sua execução. Por fim, no Capítulo 6 são apresentadas algumas considerações finais e sugestões para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Para o entendimento do problema abordado na pesquisa e como se pretende modelar uma solução, serão apresentados conceitos utilizados nas áreas de computação móvel, Computação em Nuvem, *pipeline* de dados e *Big Data*.

2.1 Computação móvel

No mundo moderno existem muitas facilidades que foram inseridas pela crescente evolução da tecnologia móvel, diversas atividades podem ser feitas usando um simples dispositivo como *smartphone* ou *tablet*, por exemplo. A principal característica relacionada a essa evolução é a mobilidade proporcionada por esses pequenos dispositivos, onde, em qualquer lugar, é possível ter acesso a uma rede de comunicação. Essa crescente evolução nos dispositivos móveis fez com que a aplicação de computação móvel fosse bem mais simples de implementar pela facilidade de acesso a diversos dispositivos e pelo avanço em tecnologias de redes locais sem fio, redes celulares (3G, 4G, etc.), que servem para a intercomunicação.

Embora existam vantagens neste tipo de computação, também têm-se alguns fortes obstáculos envolvidos. Segundo (SATYANARAYANAN, 1996) , a computação móvel é caracterizada por quatro obstáculos:

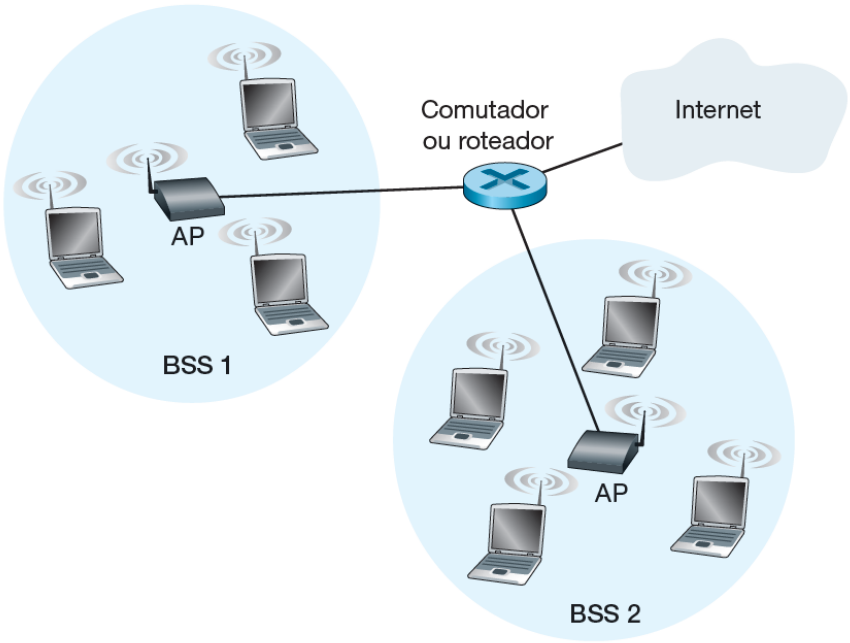
- Os elementos móveis têm poucos recursos em relação aos componentes estáticos: por simples questões de construção como custo, preocupações com o peso final, tamanho e ergonomia.
- Mobilidade é inerentemente mais perigosa: pelo fato de ser um dispositivo móvel há uma infinidade de ações que podem acontecer e causar a perda do próprio dispositivo. A utilização de comunicação através de redes sem fio faz com que a segurança sobre os dados desses dispositivos móveis possa de fato vir a ser um grande problema.
- A conectividade móvel é altamente variável em desempenho e confiabilidade: devido a mobilidade envolvida, a qualidade de acesso à rede é totalmente dependente do local onde o dispositivo se encontra. Com isso, existem locais que podem oferecer conectividade sem fio confiável de alta largura de banda, enquanto outros podem oferecer apenas de baixa largura de banda, o que acaba impactando no desempenho.

- Dispositivos móveis têm energia finita: É comum que existam diversas melhorias tecnológicas relacionadas à bateria de um dispositivo. No entanto, mesmo com todo o avanço tecnológico para esses dispositivos sempre vão existir momentos onde será necessário recarregá-los para continuar o seu uso.

2.1.1 Redes WI-FI

Está tornando-se cada vez mais comum o uso de redes sem fio em diversos tipos de lugares como hotéis, instituições de ensino, lanchonetes, etc. O tipo mais comum de rede sem fio é a LAN sem fio 802.11, que muitos conhecem, como Wi-Fi. Existem diversos padrões para essa tecnologia. Kurose e Ross (2014) listam alguns exemplos de modelos que são 802.11b, 802.11a e 802.11g. Na Figura 1 é possível ver a arquitetura padrão desses três modelos.

Figura 1 – A arquitetura de LAN IEEE 802.11



Fonte: (KUROSE; ROSS, 2014).

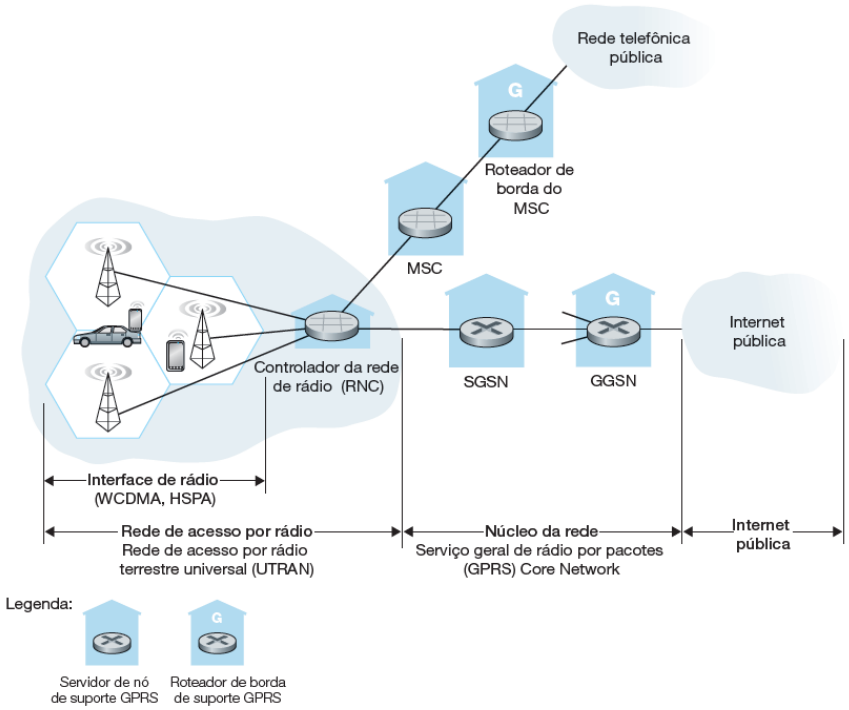
Alguns dos principais componentes da arquitetura de LAN sem fio 802.11 são mostrados na Figura 1, e algumas definições dessa arquitetura são listadas a seguir:

- Estação sem fio: qualquer dispositivo que possa se conectar à rede.
- *Basic Service Set* (BSS) (conjunto básico de serviço): O bloco de construção fundamental da arquitetura 802.11, formado por uma ou mais estações sem fio e que pode ou não incluir um ponto de acesso (AP) para uma rede Ethernet.

2.1.2 Redes celulares

Embora existam diferentes locais equipados com Wi-Fi onde seja possível simplesmente conectar seus dispositivos móveis, é fato que também existirão pontos onde haverá a falta de cobertura e acesso. Logo, isso será problemático e com isso será necessária alguma alternativa tecnológica. Para esse tipo de situação, pode-se usar as redes celulares disponibilizadas pelas diversas operadoras de telefonia celular que fornecem esse acesso à Internet junto a sua principal demanda que é a telefonia de voz. O transporte de dados através de redes celulares foi implantado a partir da geração 2,5G (KUROSE; ROSS, 2014). Com isso gerações posteriores como o 3G e 4G também podem fornecer esse acesso. Esse tpo de rede trabalha através de faixas de radiofrequência licenciadas, com isso, tem-se um alto preço financeiro a ser pago pelas operadoras e clientes finais para ser possível fornecer o acesso à Internet em ambientes externos e em movimento. Um exemplo de arquitetura de uma rede celular 3G pode ser visto na Figura 2.

Figura 2 – Arquitetura do sistema 3G



Fonte: (KUROSE; ROSS, 2014).

2.2 Computação em Nuvem

Embora se tenha uma grande comodidade e mobilidade inserida por dispositivos móveis, esses dispositivos também têm, principalmente, limitações de poder de processamento em

grande escala. Nesse contexto de análise de grandes volumes, são necessários recursos externos, muitas vezes, posicionados em dispositivos estáticos como grandes centros de processamentos de dados com infraestrutura própria. A criação desses grandes centros estão cada vez mais migrando para serviços de Computação em Nuvem principalmente pela redução de custos e facilidade de uso.

De acordo com Mell *et al.* (2011) a Computação em Nuvem *é um modelo para permitir o acesso onipresente, conveniente e sob demanda à rede a um grupo compartilhado de recursos de computação configuráveis (por exemplo, redes, servidores, armazenamento, aplicativos e serviços) que pode ser provisionado e liberado rapidamente com o mínimo de esforço de gerenciamento ou interação do provedor de serviços*. Este modelo de nuvem promove disponibilidade sendo composto de características essenciais, modelos de serviço e modelos de implantação.

2.2.1 Características

A adoção de Computação em Nuvem vem sendo uma grande promoção para o futuro em empresas que precisam de recursos tecnológicos, a seguir são listadas algumas características que justificam essa vasta adoção:

- Custo: A Computação em Nuvem elimina o gasto de capital com a compra de hardware e software, configuração e execução de *data centers* locais, incluindo *racks* de servidores, disponibilidade constante de eletricidade para energia e refrigeração, além de especialistas de Tecnologia da Informação (TI) para o gerenciamento da infraestrutura;
- Escala global: Os benefícios dos serviços de Computação em Nuvem incluem a capacidade de dimensionamento elástico. Em termos de Nuvem, isso significa fornecer a quantidade adequada de recursos de TI sempre que necessário e na localização geográfica correta.
- Desempenho: Os maiores serviços de Computação em Nuvem são executados em uma rede mundial de *data centers*, que são atualizados regularmente com a mais recente geração de *hardware* de computação rápida e eficiente. Isso oferece diversos benefícios em um único *data center* corporativo, incluindo latência de rede reduzida para aplicativos e mais economia de escalonamento.
- Segurança: Muitos provedores em Nuvem oferecem um amplo conjunto de

políticas, tecnologias e controles que fortalecem sua postura geral de segurança, ajudando a proteger os dados, os aplicativos e a infraestrutura contra possíveis ameaças.

- Velocidade: A maior parte dos serviços de Computação em Nuvem é fornecida por autosserviço e sob demanda, para que até grandes quantidades de recursos de computação possam ser provisionadas em minutos, normalmente com apenas alguns cliques, fornecendo às empresas muita flexibilidade e aliviando a pressão do planejamento de capacidade.
- Produtividade: *Data centers* locais normalmente exigem pilhas de equipamentos e implementações, tais como configuração de hardware, correção de software e outras tarefas demoradas de gerenciamento da TI. A Computação em Nuvem remove a necessidade de muitas destas tarefas, para que as equipes de TI possam investir seu tempo na obtenção de suas metas comerciais mais importantes.
- Confiabilidade: A Computação em Nuvem facilita e reduz os custos de *backup* de dados, recuperação de desastres e continuidade dos negócios, já que os dados podem ser espelhados em diversos *sites* redundantes na rede do provedor em Nuvem.

2.2.2 Modelos de Serviço

Para Mell *et al.* (2011), existem formas de disponibilizar oferta de serviços de computação em nuvem, dentre elas temos três modelos principais:

- *Software como Serviço* (SaaS): Pode-se usar os aplicativos do provedor em execução em uma infraestrutura em Nuvem sem a necessidade de se preocupar com a manutenção do serviço ou o gerenciamento da infraestrutura subjacente. Os aplicativos são acessíveis a partir de vários dispositivos clientes por meio de uma interface de cliente final.
- *Plataforma como Serviço* (PaaS): Esse modelo fornece um ambiente sob demanda para desenvolvimento, teste, fornecimento e gerenciamento de aplicativos de software com o intuito de facilitar a configuração ou o gerenciamento de infraestrutura subjacente. Dessa forma é necessário focar somente no desenvolvimento da solução.
- *Infraestrutura como Serviço* (IaaS): Modelo onde o consumidor pode montar

uma infraestrutura completa para execução proprietária do seu produto. São disponibilizados recursos como servidores, banco de dados, redes, firewall.

2.2.3 Modelos de implantação

Segundo (MELL *et al.*, 2011) os modelos de implantação são divididos em:

- Nuvem privada: A infraestrutura em Nuvem é fornecida para uso exclusivo por uma única organização composta. Tal organização pode ser responsável por gerenciar ou, simplesmente, delegar essa responsabilidade para um terceiro.
- Nuvem comunitária: A infraestrutura em Nuvem é fornecida para uso exclusivo por um comunidade de consumidores de organizações com objetivos comuns.
- Nuvem pública: A infraestrutura em Nuvem é fornecida para uso aberto pelo público em geral, deixando o gerenciamento para uma ou mais entidades empresariais, acadêmicas ou governamentais.
- Nuvem híbrida: A infraestrutura de Nuvem é uma composição de duas ou mais infraestruturas de Nuvem distintas que permanecem entidades únicas, mas são unidas por tecnologia padronizada ou proprietária que permite a portabilidade de dados e aplicativos.

2.3 *Big Data*

Com o atual avanço do acesso global à Internet, a sociedade vem produzindo cada vez mais dados por meio de sistemas computacionais. A coleta e armazenamento de enormes quantidades de registros são resultados de um fluxo que provém em 90% dos casos de ferramentas de tecnologias de informação e meios de comunicação (BUGNION *et al.*, 2017).

Partindo da necessidade de consumo e fornecimento de dados de forma abundante surgiu o conceito de *Big Data*, onde o tamanho e a quantidade dos dados torna-se um fator impeditivo em etapas como a captura, armazenamento e análise através de sistemas computacionais simplificados (MANYIKA *et al.*, 2011).

Logo, *Big Data* é um termo inerente aos recentes avanços em manuseios de grandes massas de dados de diferentes tipos na qual são inviáveis de serem processados por soluções computacionais tradicionais do dia a dia. Laney (2001) define 5 características essenciais do *Big Data* no manuseio de dados:

- Volume: Como já mencionado anteriormente, o volume de dados gerados atualmente é imenso, sendo necessário criar estruturas tecnológicas específicas para manejar esses dados.
- Velocidade: A alta velocidade de geração e mudança de dados faz com que o volume de dados acumulados se torne muito grande, em um curto espaço de tempo. Logo para muitos sistemas tem-se a necessidade de criar uma solucao para tratar à análise em tempo real com base na rápida transformação.
- Variedade: Basicamente, a forma como os dados são disponibilizados. Os dados podem ter diferentes formatos como texto, números, imagens, vídeo, áudio, dentre outros disponíveis atualmente.
- Veracidade: Essa grande quantidade de dados pode conter muitas partes descartáveis ou que precisam ser devidamente estruturadas e limpas para que o processo de análise não seja prejudicado.
- Valor: Dado que qualquer serviço de análise busca um resultado final, o valor refere-se à esse resultado e qual sua utilidade para um determinado produto ou serviço.

2.3.1 Big Data Analytics

O termo *Big Data Analytics* se refere ao processo de extração de informações relevantes partindo de *Big Data*. Dessa forma, os dados coletados são contextualizados, limpos e processados com o objetivo de gerar algum valor para o sistema ou usuário final. Para Rajaraman (2016) existem quatro tipos de análises que são consideradas em *Big Data Analytics*:

- Análise descritiva: Propõe uma análise de dados coletados aplicando técnicas estatísticas e apresentando esses resultados de forma enxuta usando gráficos, diagramas, etc. Segundo Bahga e Madisetti (2019) esse tipo de análise tenta responder a pergunta: O que aconteceu?
- Análise preditiva: A análise preditiva usa os dados do passado e presentes para tentar responder o que provavelmente acontecerá em um futuro próximo usando métodos estatísticos, redes neurais e algoritmos de aprendizado de máquina.
- Análise exploratória: A análise exploratória tenta criar padrões de comportamento para descobrir por que um determinado evento aconteceu. Um exemplo seria descobrir padrões no comportamento dos usuários por meio da interação

em redes sociais e, com isso, uma empresa pode, então, apresentar uma oferta atraente para tentar mudar a ação previamente definida por esse cliente.

- Análise prescritiva: O tipo de análise preditiva usa modelos para tentar prever um futuro resultado. Porém a análise prescritiva tenta fazer com que um determinado objetivo escolhido seja alcançado. Assim ela busca responder o que pode ser feito para que seja possível chegar nesse resultado final com base nas ações atuais.

2.4 *Pipeline* de dados

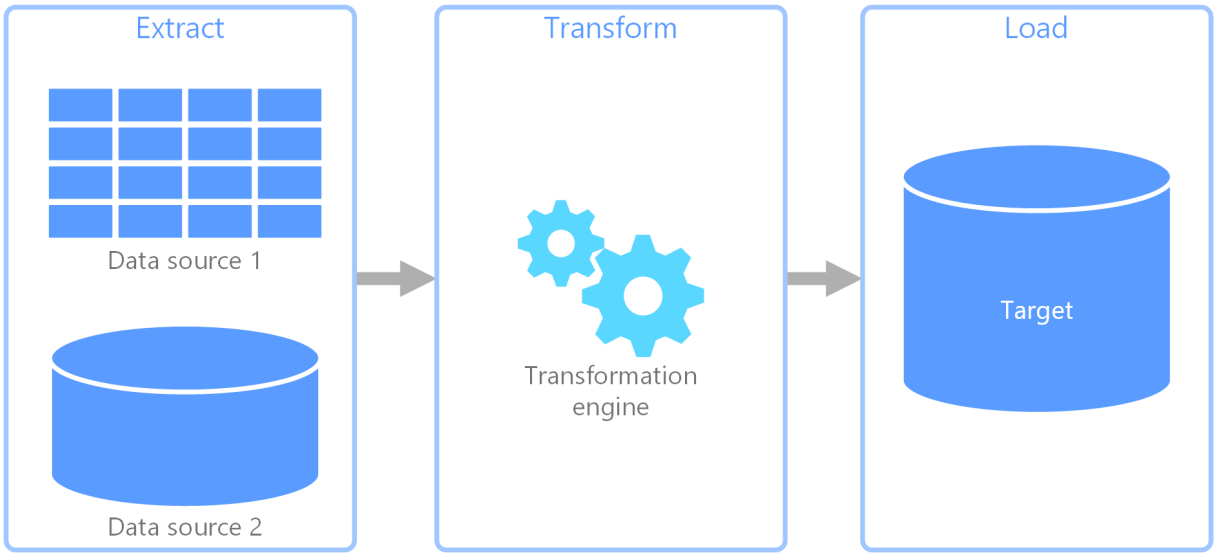
Embora atualmente exista um grande volume de dados sendo gerados, esses dados, em estado brutos, raramente têm alguma utilidade direta. Mecanismos são criados justamente para consumir e transformar esses dados passando por um sucessão de operações, também conhecidas como *pipeline* de dados (QUEMY, 2019). Essas etapas são distribuídas na movimentação de dados entre os sistemas origem e destino garantindo que ocorram de maneira consistente em todos os dados. Embora o conceito de *pipeline* de dados seja algo bastante amplo, alguns exemplos são citados nas proximas subseções.

2.4.1 Pipeline ETL

O *pilepine Extract, Transform, and Load* (ETL), mostrado na Figura 3, é um subconjunto dos *pilepines* de dados que tem, como objetivo, mover os dados que extrai de uma fonte para o destino onde os usuários finais podem acessar e usar para resolver problemas de negócios. Como o próprio nome sugere, esse tem o foco bem definido para três fases:

- Extrair (*Extract*): A etapa inicial com o objetivo de coletar os dados de uma determinada origem com sistemas, aplicações, sensores remotos etc.
- Transformar (*Transform*): Essa etapa permite aplicar transformações aos dados brutos que foram coletados pela etapa de extração. Nessa etapa, os dados são limpos, mapeados e transformados, geralmente em um esquema específico, de modo que atenda às necessidades operacionais, garantindo a qualidade e integridade dos dados.
- Carregar (*Load*): Por fim, o carregamento consiste em manter esses dados armazenados em algum destino como um sistema de banco de dados ou aplicações

Figura 3 – Processo de extração, transformação e carregamento (ETL)



Fonte: Microsoft (2019).

similares.

2.4.2 *Pipeline de dados baseado em lote*

O processamento em lote envolve a manipulação de blocos de dados que já foram previamente armazenados durante um determinado período de tempo. Esse tipo de processamento é feito com um massivo volume de dados e que não exigem processamento e análises em tempo real. O processamento em lote se baseia no paradigma *MapReduce* (DEAN; GHEMAWAT, 2008), em que os dados são armazenados primeiro e depois processados. Uma implementação de código aberto e amplamente usado é o *Hadoop* (WHITE, 2012).

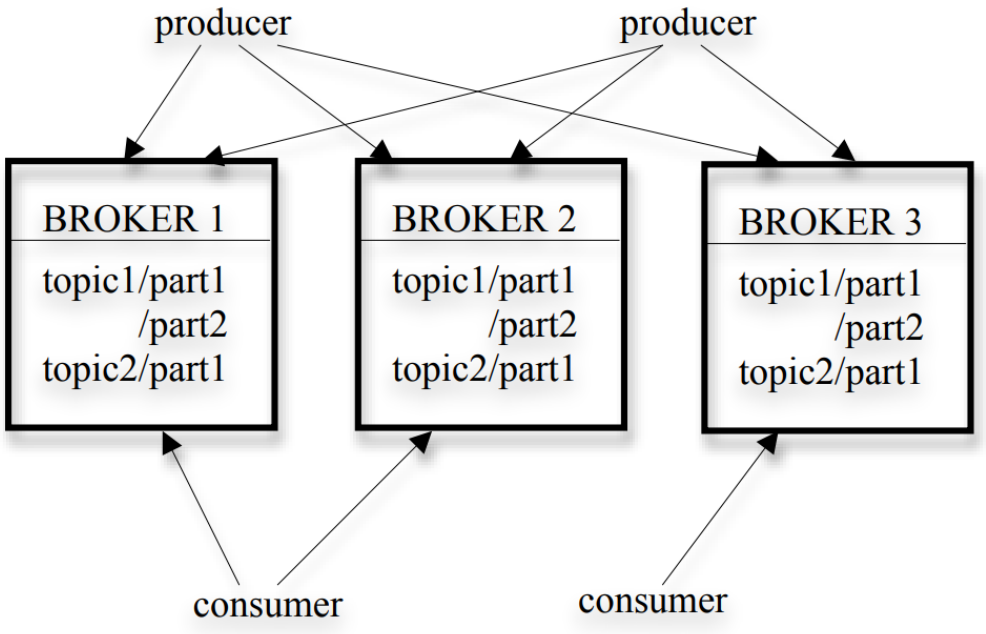
2.4.3 *Pipeline de Dados de Streaming*

Quando é necessário tratar as mudanças em tempo real, o *pipeline* de dados de *streaming* é o ideal, por conta que ele executa operações em dados em tempo real de coleta (VERA-BAQUERO *et al.*, 2016). Assim, é possível gerar rapidamente tomadas de decisões e *insights* imediatos sobre os dados coletados. Para esse paradigma, existem diversas tecnologias que podem ser empregadas, uma delas é *Kafka* [1] que é um sistema de mensagens distribuído desenvolvido para coletar e entregar grandes volumes de dados de registro com baixa latência (KREPS *et al.*, 2011). O *Kafka* tem seu funcionamento baseado no esquema *Publish–subscribe*. Dessa forma são definidos tópicos onde um produtor (*Publish*) pode publicar mensagens. As

[1] https://kafka.apache.org/

mensagens publicadas são então armazenadas em um conjunto de servidores chamado *brokers*. Um consumidor (*Subscribe*) pode assinar um ou mais tópicos dos *brokers* e consumir as mensagens assinadas. A Figura 4 mostra essa arquitetura.

Figura 4 – Arquitetura do Kafka



Fonte: (KREPS *et al.*, 2011).

2.5 *Containers e Docker*

Containers são componentes executáveis padronizados que combinam o código-fonte de um aplicativo com as bibliotecas do Sistema Operacional (SO) e as dependências necessárias para executar esse código em qualquer ambiente. *Containers* são comparados com Máquinas Virtuais (VMs). Alguns benefícios da utilização de *containers* são (EDUCATION, 2021):

- Mais leves: ao contrário das VMs, os *containers* não carregam uma instância inteira do SO, eles apenas incluem os processos do sistema operacional e dependências necessárias para executar aquele determinado código.
- Maior eficiência de recursos: você pode executar várias vezes mais cópias de um aplicativo no mesmo *hardware* do que usando VMs.
- Aumento na produtividade de desenvolvimento: Quando comparados às VMs, *containers* são mais rápidos e fáceis de implantar, provisionar e reiniciar. Isso os torna ideais para uso em *pipelines* de integração contínua.

Para manusear *containers* tem-se o *Docker* [2], uma plataforma de conteinerização de código aberto que permite aos desenvolvedores empacotar aplicativos em *containers*. *Docker* é muito usado por tornar fácil, simples e seguro construir, implantar e gerenciar contêineres (EDUCATION, 2021).

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

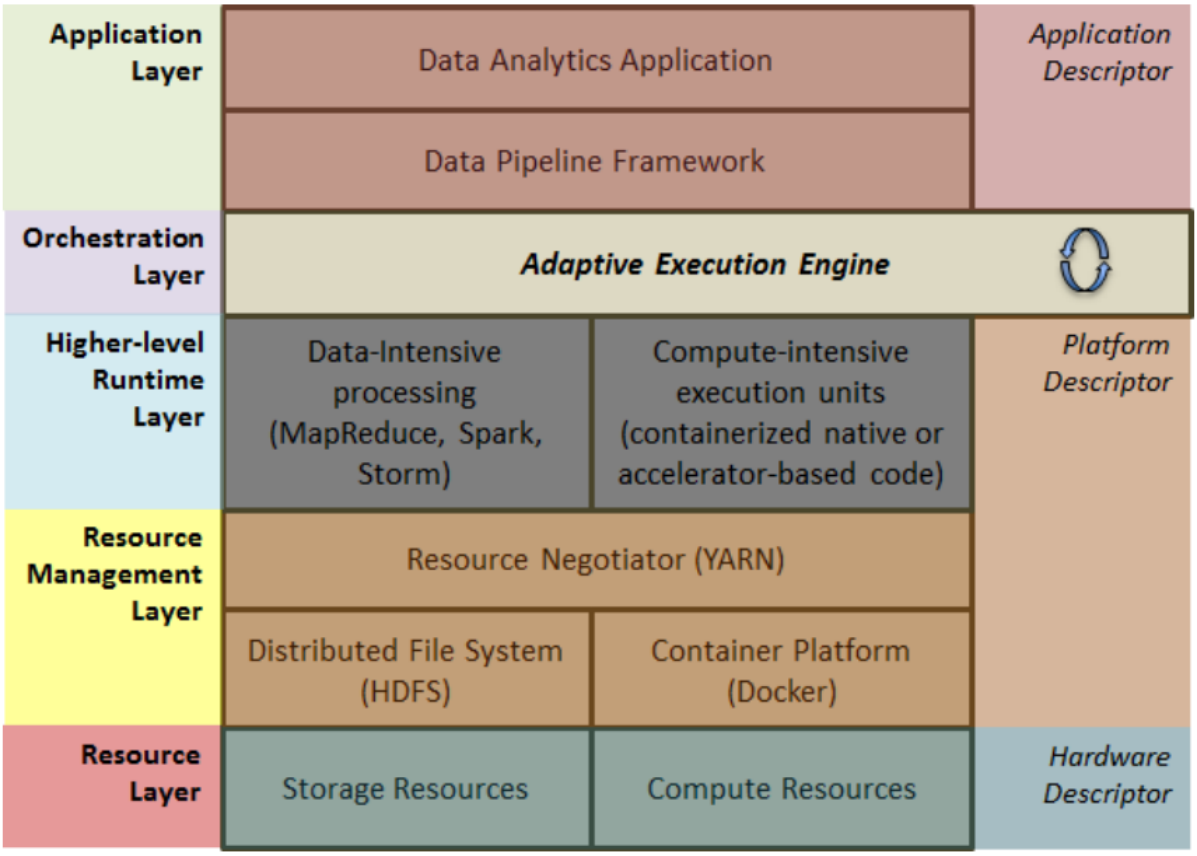
3 TRABALHOS RELACIONADOS

Neste capítulo são apresentados trabalhos relacionados a esta pesquisa focando em temas como *pipeline* de dados e aplicações móveis.

Goodhope *et al.* (2012) apresentaram os problemas de *design* e engenharia encontrados ao mover o *pipeline* de dados do *LinkedIn* de um mecanismo de processamento em *batch* para um sistema de *publish-subscribe* em tempo real usando o *Kafka*. Para o contexto do *LinkedIn* os fatores que levaram a essa mudança, foram: necessidade de monitoramento em tempo real para os dados de atividade dos usuários; eficiência da coleta de dados necessários para funcionalidades, tornando o processo uma dependência para outra parte do próprio produto. Com isso seria necessário montar uma infraestrutura mais robusta. A solução desenvolvida pelos autores foi uma aplicação de monitoramento autônomo que consome tópicos fornecidos pelo sistema do *Kafka* e executa uma reconciliação exata entre todas as camadas para calcular as taxas de perda ou duplicação, bem como para realizar gráficos e alerta sobre os dados coletados.

Kaniovskyi *et al.* (2017) propõe uma abordagem de execução e gerenciamento em *containers* para os diferentes estágios de um *pipeline* de análise de dados, de forma que fosse possível utilizar técnicas e paradigmas de computação intensiva para execução eficaz do *pipeline* implantado sem necessidade de reconstrução do zero com base no modelo de aplicação do *framework* escolhido. A Figura 5, mostra a solução construída pelos autores deste trabalho, sendo estruturada da seguinte forma: camada de aplicação que utiliza descritores de metadados no modelo YAML para as variantes de implementação de cada estágio. Camada de orquestração que utiliza um mecanismo adaptável para execução dos estágios do *pipeline* . Camada de tempo de execução superior que realiza a implementação e programação para técnicas de processamento intensivo de dados. Camada de recursos que fornece descritores e recursos de armazenamento. O resultado apresentado pelos autores demonstrou que a utilização de *containers* em arquiteturas heterogêneas para integrar diferentes plataformas fornece a possibilidade de utilização de diferentes variantes de implementação com base em uma infinidade de linguagens de programação e paradigmas, com recursos e gerenciamento eficiente dos mesmos.

Figura 5 – *Framework architecture for data and compute-intensive data analytics*



Fonte: (KANIOVSKYI *et al.*, 2017).

Conceição *et al.* (2014) propuseram uma solução utilizando Computação em Nuvem para armazenamento de dados móveis. A plataforma permite ainda a criação de aplicações para desenvolvedores sem habilidades de programação por meio de interfaces simples e intuitivas. A solução apresenta uma arquitetura de *pipeline* em nuvem que permite coletar e analisar usando dispositivos *Android* com aplicações desenvolvidas utilizando o projeto Maritaca. Entre os dados passíveis de análise, se destacam os dados convencionais como textos e números e não convencionais como multimídia e dados de localização. A solução desenvolvida pelos autores foi estruturada da seguinte forma: camada de integração entre dispositivo e aplicação utilizando RESTFul para comunicação, camada de funcionalidades dos componentes móveis, responsável por implementar dados e renderizar interfaces da aplicação, camada de dados incomuns com responsabilidade de coleta de dados usuais como texto, números e dados incomuns como localização e arquivos de mídia. E a camada de geração do aplicativo, que permite a geração de um novo arquivo *Android application package* (APK) cada vez que um formulário é salvo no editor de formulários. A solução apresentada pelos autores permite a criação de aplicações por indivíduos sem o conhecimento de linguagens de programação ou infraestruturas de TI, bem

como ferramentas para coleta, compartilhamento e análise de dados que permitem a modificação dos requisitos do software por meio de interfaces simples.

Vouros *et al.* (2020) sugeriram uma implementação da arquitetura chamada δ para a criação de um *pipeline* satisfazendo os requisitos de análise de mobilidade de *Big Data*, explorando dados de mobilidade real para realizar tarefas de análise em tempo real e em *batch*. A arquitetura desenvolvida para solucionar esse problema tem uma série de componentes, dentre eles: o detector de eventos de baixo nível que funciona para procurar os erros e fazer e criar mensagens, o integrador de semântica que permite transformar os dados de entrada no formato esperado pelos outros componentes, o analisador de tarefas e o módulo de reconhecimento de eventos. Os componentes da arquitetura são conectados de forma fracamente acoplada, agem como consumidores, produtores ou ambos, assinando tópicos ou publicando mensagens em sistema de *mensageiria* como o *Kafka*.

Bashaireh *et al.* (2020) propuseram uma técnica de coleta de dados em tempo real da plataforma *Twitter* utilizando ferramentas como *Python*[1], *Pandas*[2] por meio da Interface de programação de aplicações (API) de *streaming* do Twitter[3]. Tal coleta utilizava um conjunto de critérios previamente definidos, como palavra-chave, localização e outros 25 possíveis parâmetros, onde, segundo os autores, o volume de coleta de informações estava diretamente relacionado a quantidade de parâmetros definidos o tráfego atual da API utilizada. A solução proposta pelos autores se divide em três fases: criação de uma aplicação como interface para a API do *Twitter* onde se define parâmetros e credenciais de acesso à plataforma, formatação de dados extraídos com a conversão dos *JavaScript Object Notation* (JSON) obtidos para *Comma-separated values* (CSV) por meio do módulo CSV *Python*, e a criação de *Dataframes Pandas* para manipulação e análise de dados futuras. O experimento foi conduzido analisando as publicações extraídas durante alguns meses com palavras chaves relacionadas a universidades do estado do Michigan onde foi observado padrões e comportamentos relativos ao estilo de publicações relacionado aos estudantes.

Na Tabela 1 são apresentadas algumas das principais atividades e aspectos quando comparados com este trabalho, onde é possível observar e diferenciar este trabalho para os demais apresentados neste capítulo.

[1] https://www.python.org/
[2] https://pandas.pydata.org/
[3] https://twitter.com/

Tabela 1 – Trabalhos relacionados

Atividades	GOODHOPE et al. (2012)	KANIOVSKYI et al. (2017)	CONCEIÇÃO et al. (2014)	VOUROS et al. (2020)	BASHAIREH et al. (2020)	Este trabalho
Criação de aplicações móveis para coleta de dados			X			
Mapeamentos de ferramentas para *pipeline* de dados	X					
Criação de uma abordagem de gerenciamento usando *containers*		X				
Extração de dados de aplicações web					X	
Fluxo de dados diretamente de dispositivos moveis			X			X
Estruturar um pipeline de coleta em *batch*				X		
Estruturar um *pipeline* de coleta em tempo real	X			X		X
Executar um cenário exemplo						X

Fonte: Elaborado pelo autor (2021).

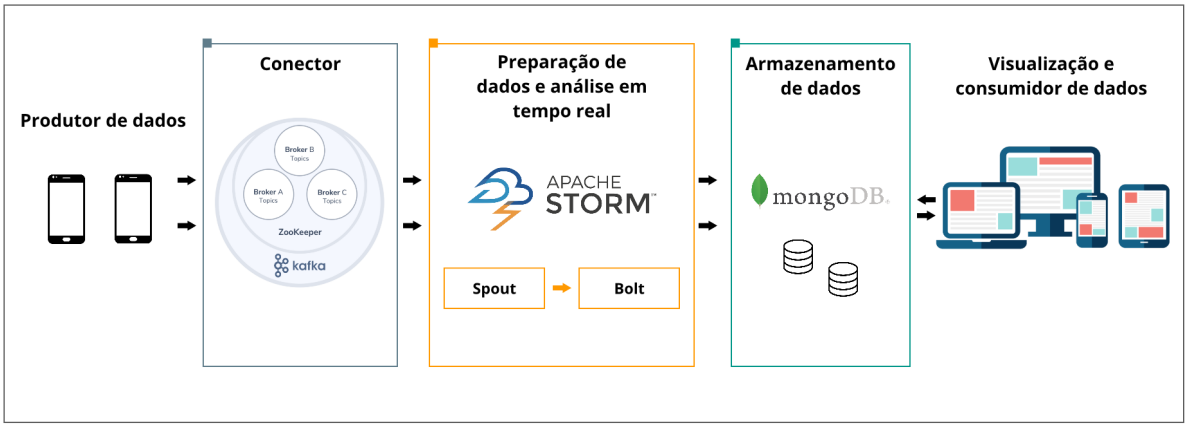
4 PROPOSTA

Este capítulo apresenta a arquitetura definida para a solução proposta, apresentando todas as partes envolvidas e quais tecnologias são usadas para ser possível alcançar os objetivos desta pesquisa.

4.1 Arquitetura proposta

A proposta deste trabalho consiste na implementação de um *pipeline* para coleta e armazenamento de dados de dispositivos móveis com processamento de dados em tempo real. Assim como a criação e execução de um cenário exemplo para esse *pipeline*. A Figura 6 apresenta o modelo de arquitetura proposto que foi desenvolvido usando como base o padrão analítico *Beta* que é um modelo usado para análise em tempo real (BAHGA; MADISETTI, 2019). As partes e componentes dessa arquitetura são descritos mais profundamente nas próximas seções deste capítulo.

Figura 6 – Arquitetura proposta



Fonte: Elaborado pelo autor (2022).

4.2 Produtor de dados

Como já abordado anteriormente, os dispositivos móveis são responsáveis por grande parte da geração de dados atualmente. Para a arquitetura proposta nesse trabalho, serão utilizados dispositivos (*smartphones*, *tablets*, etc.), como os produtores de dados para esse *pipeline*. Esses dispositivos usaram alguns protocolos de redes móveis, como os discutidos no Capítulo 2, para enviar mensagens à tópicos do sistema de *mensageria* implementado em Nuvem usando *Kafka*.

Para a serialização de dados usados nas mensagens enviadas por esses dispositi-

vos, é usado o JSON, por ser relativamente rápido o processo de serialização e favorecer a compatibilidade para a evolução na estrutura dos dados ao longo do tempo.

4.3 Conector

Algumas das alternativas que podem ser usadas para comunicação entre os produtores de dados e sistema de análise em tempo real são ferramentas de filas e sistemas de *mensageria* (BAHGA; MADISETTI, 2019). Para essa arquitetura proposta, foi adotado o sistema de *mensageria Apache Kafka* para gerenciar os dados recebidos dos dispositivos móveis. Esse sistema usa o modelo de *Publish–subscribe*, onde são criados tópicos para onde os dispositivos direcionam suas mensagens com os dados.

4.4 Preparação de dados e análise em tempo real

Uma das etapas presentes em um *pipeline* de dados é a preparação de dados, onde pode ser necessário realizar a limpeza e a manipulação dos dados coletados (BAHGA; MADISETTI, 2019).

Para essa proposta de análise em tempo real, foi usado um *framework* de processamento em *stream* chamado *Apache Storm* [1]. Tem-se algumas alternativas como o *Apache Spark Streaming*, no entanto, esse é focado para o processamento de dados em memória, o que não acontece com o *Storm*. O *Apache Storm* é um sistema livre e *open-source* utilizado na computação distribuída em tempo real que possibilita realizar o processamento do fluxo de dados que chegam através das mensagens do *Kafka*. O uso do *Apache Storm* se divide em basicamente duas fases que são:

- O *Storm Spout* que é uma fonte de fluxos, responsável por pegar os dados publicados através de mensagens nos tópicos do *Apache Kafka* que serão processados a seguir pelo *Storm Bolt*.
- O *Storm Bolt* recebe os dados emitidos pelo *Spout* e realiza o processamento desses dados. Nessa fase são feitas diversas tarefas como filtragem, mesclagem, etc. Pode-se aplicar modelos de análises de dados que ficam dependentes das regras de negócio de cada aplicação. Por exemplo, um modelo de aprendizagem de máquina treinado. Por fim, também é responsabilidade do *Bolt* repassar essas

[1] https://storm.apache.org/

informações para a fase de armazenamento de dados.

4.5 Armazenamento de dados

Depois de realizar o devido processamento e análise, é preciso uma forma de manter salvo todo o resultado da seção anterior. Dentre os diversos sistemas de banco de dados, para essa proposta foi usado o *MongoDB* [2], que é um é um banco de dados *NoSQL* de propósito geral, baseado em documentos. Essa escolha tem grande relação com a facilidade para a reestruturação dos dados em bancos *NoSQL*. Essa parte da arquitetura é responsável por implementar toda a parte de armazenamento e disponibilização para que seja possível consultar e visualizar o resultado final dos dados.

4.6 Visualização e consumidor de dados

Essa parte da arquitetura serve simplesmente para ilustrar as mais diversas aplicações que necessitam consumir e usar diferentes estratégias de visualização para os resultados finais do *pipeline* de dados.

[2] https://www.mongodb.com/

5 EXECUÇÃO DO CENÁRIO EXEMPLO

Este capítulo apresenta um cenário exemplo para a utilização do *pipeline* proposto, destacando sua arquitetura e apresentação geral dos seus componentes.

5.1 Cenário exemplo

O cenário exemplo escolhido para realizar a execução do *pipeline*, trata-se do consumo dos dados da atualização da localização de um determinado veículo. Assim é possivel simular um trasnporte coletivo urbano e processar esses dados. Com isso, qualquer usuário que tenha interesse em receber atualização da posição desse veículo em um determinado raio de atuação pode ser notificado e tenha acesso a última posição conhecida do mesmo. Para isso, é necessário processar todas as atualizações de posição de todos os transportes cadastrados no sistema e verificar se é necessário notificar um determinado usuário. A Figura 7 ilustra essa situação exemplo, os passos do fluxo do exemplo são.

- 1) Casa + Parada de ônibus: parte na qual pode-se ter a necessidade de receber uma notificação para um veiculo nessa parada mais próxima. .
- 2) Usando a aplicação: usar a aplicação para criar uma solicitação de notificação para um determinado veículo num raio também informado.
- 3) Recebendo notificação: usuário final recebendo notificação para a solicitação criado no passo 2.
- 4) Ir até a parada: por fim depois de receber a notificação o usuário pode ser deslocar para a parada a fim de embarcar.

Figura 7 – Situação exemplo

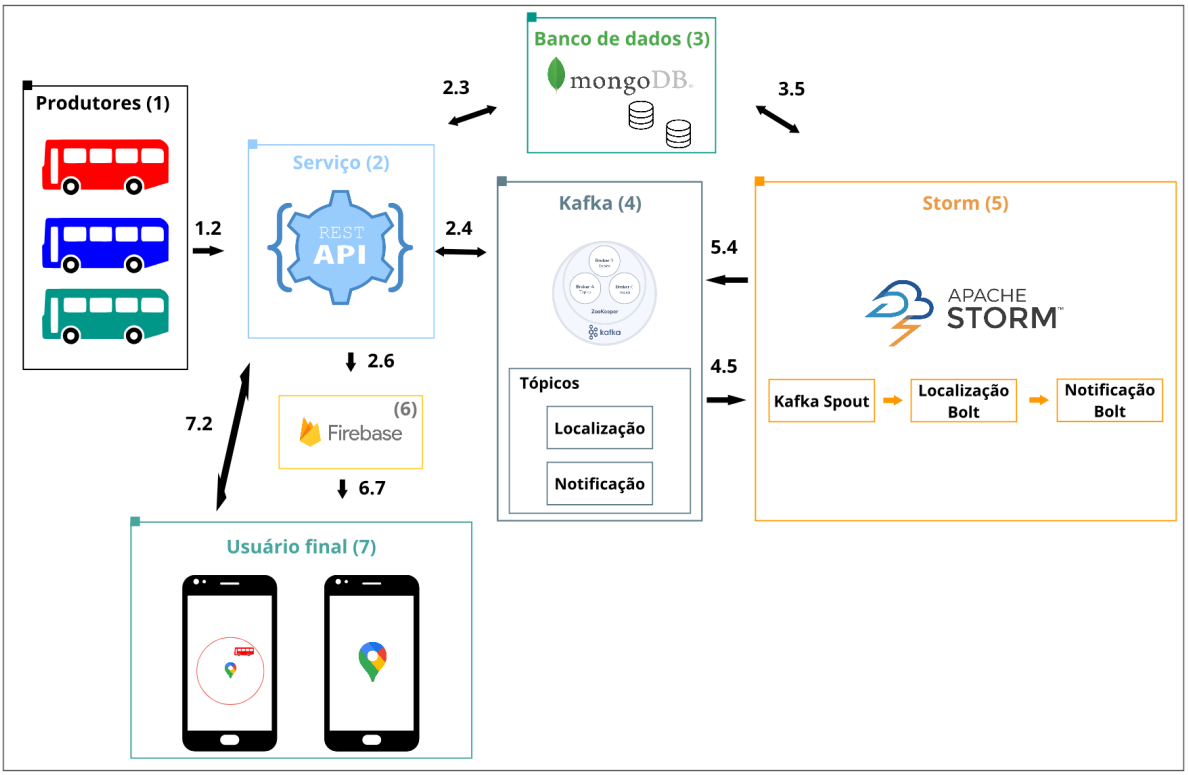
Fonte: Elaborado pelo autor (2022).

5.2 Arquitetura do cenário exemplo

A Figura 8 apresenta o modelo de arquitetura desenvolvido para atuar com o cenário exemplo descrito na Seção 5.1. Nas próximas seções são listados os componentes dessa arquitetura assim como seu funcionamento. Para facilitar todo o processo de configurar o ambiente e executá-lo, os serviços e aplicações foram empacotados em *containers* usando o *Docker*. Toda a implementação pode ser acessada pelo *Github* [1] nos repositórios *mobile-pipeline* e *docker-kafka-storm*, que contem as duas aplicações em *Flutter*[2] com o serviço e o restante do *core* de processamento, respectivamente.

[1] https://github.com/
[2] https://flutter.dev/

Figura 8 – Arquitetura do cenário exemplo



Fonte: Elaborado pelo autor (2022).

5.3 Produtores

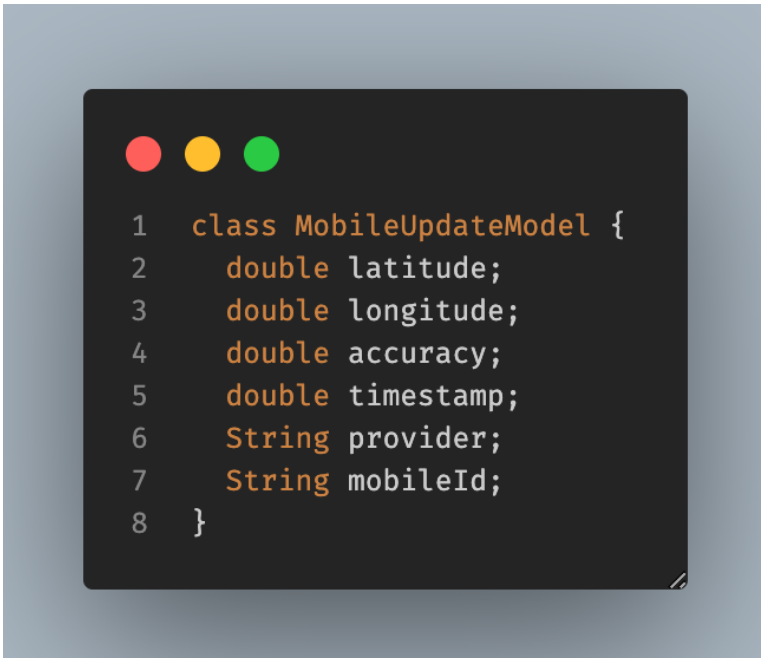
Para esse cenário exemplo, são usados os dados de dispositivos móveis acoplados em veículos para esse *pipeline*. Esses dispositivos usam alguns protocolos de redes móveis, como os discutidos no Capítulo 2 para enviar mensagens com a atualização de sua posição para um serviço definido na seção seguinte.

Para simular as mudanças de posição, foi desenvolvida uma aplicação móvel que coleta e envia esses dados. Essa aplicação foi feita usando um *framework* chamado *Flutter* que foi criado pelo Google com o intuito de facilitar o desenvolvimento multiplataforma de aplicações móveis. A escolha dessa tecnologia foi pelo simples fato do seu tempo de desenvolvimento curto e pelo prévio conhecimento do autor. Na Figura 9, é exibida a estrutura dos dados que são serializados no formato JSON e enviados pelos produtores para o serviço. Vale notar que temos os seguintes atributos:

- **latitude** e **longitude**: referente a posição para a localização;
- **accuracy**: indicando o nível de precisão para esses dados de localização coletados;
- **timestamp**: simplesmente a marca temporal na qual esses dados foram coletados;

- **provider**: representando qual o provedor dos dados da localização;
- **mobileId**: indica qual veículo esta enviando os dados.

Figura 9 – Estrutura dos dados enviados pelos produtores.

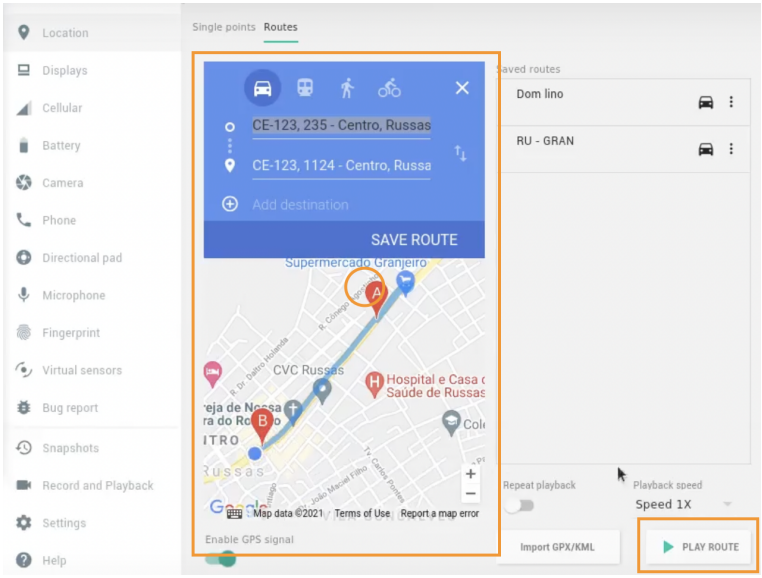


Fonte: Elaborado pelo autor (2022).

As rotas para a simulação foram criadas usando as ferramentas de desenvolvimento e simulação de localização do *Android Studio* [3]. Assim é possível criar diversas rotas com diferentes percursos no próprio mapa da ferramenta, não se limitando à percursos retilíneos. Com isso foi possível simular o deslocamento do veículo e enviar suas atualizações para o serviço descrito na Seção 5.4. Na Figura 10, pode-se ver um pouco da ferramenta onde, ao centro, é possível notar o ambiente de configuração de ponto de início e final da rota e também um botão com o nome *PLAY ROUTE* que serve para dar início a simulação da rota.

[3] https://developer.android.com/studio

Figura 10 – *Android Studio*: Ferramentas de localização



Fonte: Elaborado pelo autor (2022).

5.4 Serviço

Esse é um serviço REST feito com *Node.js*[4] usando o *framework Express*[5]. A Figura 11 apresenta os principais *endpoints* do serviço e exemplos dos dados que são enviados e retornados. Para este serviço tem-se as seguintes finalidades:

- Intermediar o contato dos produtores com o serviço (índice 1.2 da arquitetura exposta na Figura 8) de envio de mensagens a tópicos do *Kafka*. Como isso, os produtores encaminham os dados para o *endpoint* **/mobileLocationUpdate** que simplesmente adiciona ao tópico do *Kafka* (índice 2.4 da arquitetura exposta na Figura 8).
- Consumir o tópico de notificações e encaminhar ao *Firebase*[6] as mensagens que são disparadas depois do processamento feito pelo *Storm* (índice 2.6 da arquitetura exposta na Figura 8).
- Gerenciar a criação de dados necessários para o funcionamento completo do sistema, seja a criação de entidades como veículos (**/mobiles**), usuários (**/login**), assim como solicitações de notificação para determinados veículos pelos usuários finais (**/subscription**) (índice 7.2 da arquitetura exposta na Figura 8).

[4] https://nodejs.org/
[5] https://expressjs.com/
[6] https://firebase.google.com/

Figura 11 – Principais *endpoints* do serviço

```
1) POST /login
{
  "username": "jim",
  "token": "token-firebase"
}

2) POST /mobiles
{
  "description": "BUS XP09"
}

3) GET /mobiles
[Mobile]

4) POST /mobileLocationUpdate
{
  "latitude": -3.93954,
  "longitude": -27.972338,
  "provider": "GPS",
  "accuracy": 10.0,
  "mobileId": "61c5c3e14e9b2ae8b48bb4d5",
  "timestamp": 1639618170713
}

5) POST /subscription
{
  "mobileId": "61c123dfdbc9ba465f1c1c07",
  "userId": "61c125f7dbc9ba465f1c1c0c",
  "track": false,
  "distanceToNotifier": 1000,
  "latitude": -3.939885,
  "longitude":  -27.972604
}
```

Fonte: Elaborado pelo autor (2022).

5.5 Banco de dados

O banco de dados para este exemplo foi criado usando o *MongoDB* e modelado de forma que foram criados simples *Schemas* usando o *Mongoose*[7] que é uma biblioteca para modelagem de objetos do MongoDB, dentre eles:

- *User Schema*: mostrado na Figura 12, responsável por manter o básico de informação sobre o usuário e o *token* do *Firebase* para o envio de notificações;
- *Mobile Schema*: mostrado na Figura 13, responsável por manter o básico de informação sobre o veículo simulado;
- *Subscription Schema*: mostrado na Figura 14, responsável por representar a intenção de um determinado usuário em receber notificação dado um raio informado em metros.

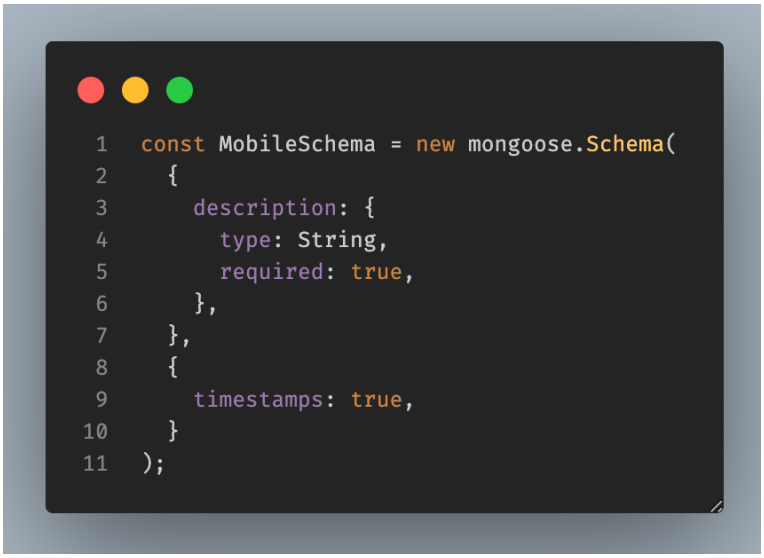
Figura 12 – *User Schema*

```
const UserSchema = new mongoose.Schema(
  {
    username: {
      type: String,
      required: true,
    },
    token: {
      type: String,
      required: true,
    },
  },
  {
    timestamps: true,
  }
);
```

Fonte: Elaborado pelo autor (2022).

[7] https://mongoosejs.com/

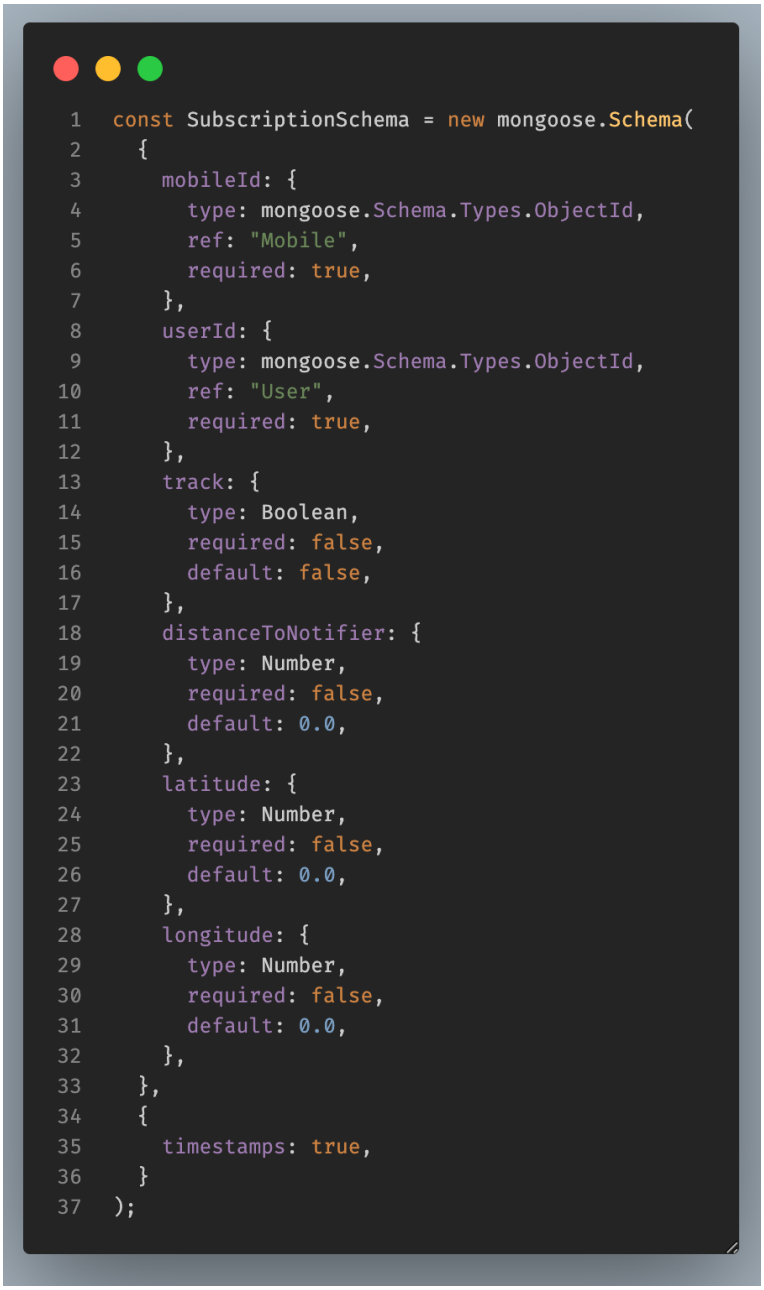
Figura 13 – *Mobile Schema*



```
const MobileSchema = new mongoose.Schema(
  {
    description: {
      type: String,
      required: true,
    },
  },
  {
    timestamps: true,
  }
);
```

Fonte: Elaborado pelo autor (2022).

Figura 14 – *Subscription Schema*



```
const SubscriptionSchema = new mongoose.Schema(
  {
    mobileId: {
      type: mongoose.Schema.Types.ObjectId,
      ref: "Mobile",
      required: true,
    },
    userId: {
      type: mongoose.Schema.Types.ObjectId,
      ref: "User",
      required: true,
    },
    track: {
      type: Boolean,
      required: false,
      default: false,
    },
    distanceToNotifier: {
      type: Number,
      required: false,
      default: 0.0,
    },
    latitude: {
      type: Number,
      required: false,
      default: 0.0,
    },
    longitude: {
      type: Number,
      required: false,
      default: 0.0,
    },
  },
  {
    timestamps: true,
  }
);
```

Fonte: Elaborado pelo autor (2022).

5.6 Kafka

Para esse exemplo, a modelagem dos tópicos do *Kafka* foi realizada de modo que foram criados dois tópicos para conter os dados, sendo eles:

- Tópico de localização: esse tópico foi usado com o intuito de agrupar as mensagens com as atualizações das posições enviadas pelos produtores definidos na Seção 5.3;
- Tópico de notificação: esse tópico será usado para agrupar as mensagens das

notificações que precisam ser encaminhadas para os usuários finais por meio do *Firebase*.

5.7 Storm

Nessa implementação, há um *Spout* e dois *Bolts* que foram responsáveis por realizar todo o processamento dos dados produzidos e submetidos no *pipeline*.

- *Kafka Spout*: esse é o *Spout* responsável por capturar as mensagens do tópico de localização do *Kafka* e disponibilizar para o processamento nos *Bolts*. Vale salientar que essas mensagens seguem o mesmo formato mostrado na Figura 9, pois elas são enviadas pelo serviço descrito na Seção 5.4.
- Localização *Bolt*: depois do *Storm* receber, por meio do *Kafka Spout*, os dados da atualização da posição de um veículo, é necessário verificar se tem algum usuário final que precisa ser notificado com base nessa última atualização. Caso seja necessário realizar a notificação, ele encaminha os dados para o Notificação *Bolt*.
- Notificação *Bolt*: a funcionalidade deste *Bolt* é receber os dados de Localização *Bolt* com a última atualização para o usuário final que precisa ser notificado e encaminhar para o serviço de notificação.

5.8 Firebase

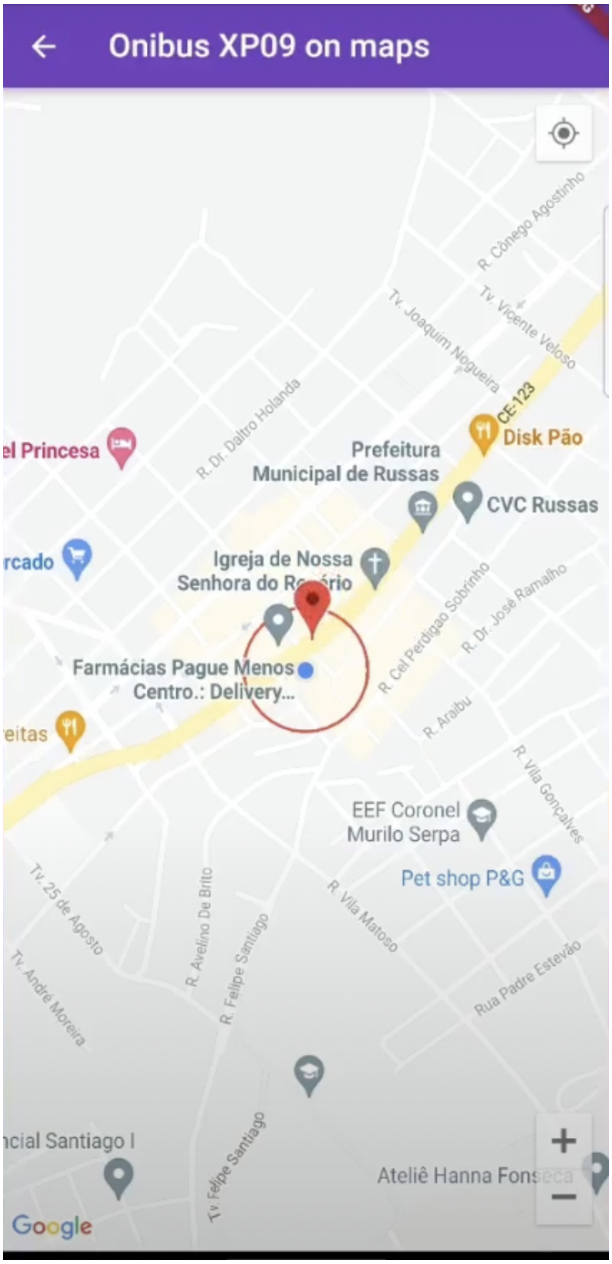
Para enviar as mensagens aos dispositivos dos usuários finais foi usado um serviço chamado *Firebase Cloud Messaging* que foi projetado para fornecer conexão aos seus dispositivos por meio de mensagens e notificações (MORONEY, 2017). Essas mensagens são enviadas por meio do *token* salvo no banco de dados por meio do *User Schema*.

5.9 Usuário Final

Essa parte da arquitetura serve para ilustrar como os usuários finais podem usufruir dos serviços prestados pelo *pipeline*. Trata-se de uma aplicação móvel desenvolvida para permitir criar solicitações (Figura 14) e receber atualizações de posição dentro de um raio informado para um determinado veículo. Para o desenvolvimento desta aplicação também foi utilizado o *framework Flutter*. Assim, quando o veículo mandar uma atualização para o *pipeline* e for

processado pelo *Storm*, sua última posição será exibida em um mapa na aplicação para que o usuário possa ter ciência de onde o veículo se encontra. Na Figura 15, é possível ver parte dessa aplicação onde é notório visualizar um círculo vermelho que representa a área de notificação pelo qual o usuário receberá atualizações. A marcação em vermelho representa o veículo dado a sua última posição conhecida.

Figura 15 – Aplicação para usuário final: área de notificação



Fonte: Elaborado pelo autor (2022).

5.10 Testes manuais com cenário exemplo em execução

Foram realizados alguns testes manuais com a criação de dados de testes tanto do simulador de posição quando do aplicativo para o usuário final e acompanhamento dos *logs* do sistema para verificar a eficácia do mesmo. Vale notar que esse tipo de teste exige um poder de processamento grande e recursos como dispositivos móveis o que o torna complexo. Nesses testes, foi percebido que o objetivo de notificar o usuário final no momento em que o veículo entra no raio solicitado foi alcançado, inclusive a Figura 15 foi extraída de uma dessas execuções. A título de curiosidade, o vídeo de uma dessas execuções pode ser acessado pelo seguinte *link*: https://drive.google.com/file/d/1sDWxIobRJUmC5s2pHvLPEJk_$pvA3vii/view?usp = sharing$.

6 CONCLUSÕES E TRABALHOS FUTUROS

A proposta deste trabalho foi a criação de um *pipeline* para processamento de dados gerados por dispositivos móveis e sua avaliação por meio da aplicação em um cenário exemplo. Com isso foi traçada uma discussão sobre o processamento de dados, tecnologias que podiam ser usadas para apoiar uma solução e modelagem da arquitetura proposta. Conclui-se que o campo de pesquisa e problemática do tema é complexo, porém, quando aplicado ao cenário exemplo foi possível obter o resultado esperado para o mesmo. No entanto, ao fim, fica uma pendência de mais testes de modo a avaliar o comportamento da solução em larga escala de utilização.

Existiram algumas limitações relacionadas ao poder computacional da máquina utilizada. Devido à limitação de tempo para a pesquisa e complexidade da solução, apesar de toda a solução ter sido criada visando a implantação em *Cloud* com a utilização de *containers* e usando o *Docker*, não foi possível alcançar esse objetivo, ficando pendente para trabalhos futuros uma possível implantação.

Durante o processo de desenvolvimento do trabalho foram encontradas dificuldades principalmente na utilização das ferramentas usadas. Assim, foi necessário estudar por meio da própria documentação e também recursos externos. O processo de configuração e junção da solução como um todo tende a ser um pouco complexo, tendo a necessidade de criação de muitos dados para conseguir uma execução. Sabendo disso, foi feita a utilização de *Mocks* dos dados, que são objetos que simulam o comportamento de objetos reais de forma controlada.

6.1 Trabalhos Futuros

Dado o entendimento deste trabalho tem-se como sugestões para trabalhos futuros:

- Usar alguma plataforma de *Cloud* para hospedar o projeto e tentar escalar. Como o desenvolvimento foi feito visando essa parte, a utilização do *Docker* no projeto na totalidade traz uma maior facilidade.
- Aplicar cenários de testes mais robustos. Como explicado no Capítulo 5, foi realizado apenas testes manuais e relativamente simples.
- Fazer comparações com outras ferramentas que poderiam gerar soluções de *pipeline* para o mesmo contexto.

REFERÊNCIAS

BAHGA, A.; MADISETTI, V. **Big Data Science & Analytics: A Hands-on Approach**. [S.l.]: Arshdeep Bahga, Vijay Madisetti, 2019. ISBN 9780996025539.

BASHAIREH, R. A.; ZOHDY, M.; SABEEH, V. Twitter data collection and extraction: A method and a new dataset, the utd-mi. In: **Proceedings of the 2020 the 4th International Conference on Information System and Data Mining**. [S.l.: s.n.], 2020. p. 71–76.

BUGNION, P.; MANIVANNAN, A.; NICOLAS, P. R. **Scala: Guide for Data Science Professionals**. [S.l.]: Packt Publishing Ltd, 2017.

CONCEIçãO, A. F.; SáNCHEZ, J. V.; SANTOS, B. G. dos; VIEIRA, D.; ROCHA, V. Pipeline architecture for mobile data analysis. In: **The International Conference on Information Networking 2014 (ICOIN2014)**. [S.l.: s.n.], 2014. p. 492–496.

DEAN, J.; GHEMAWAT, S. Mapreduce: simplified data processing on large clusters. **Communications of the ACM**, ACM New York, NY, USA, v. 51, n. 1, p. 107–113, 2008.

EDUCATION, I. C. **What is Docker?** 2021. IBM Cloud Education: What is Docker? Disponível em: <https://www.ibm.com/cloud/learn/docker>. Acesso em: 7 jan. 2022.

ITU. **Mobile cellular subscriptions (per 100 people)**. 2019. Disponível em: <https://data.worldbank.org/indicator/IT.CEL.SETS.P2>. Acesso em: 7 fev. 2022.

GOODHOPE, K.; KOSHY, J.; KREPS, J.; NARKHEDE, N.; PARK, R.; RAO, J.; YE, V. Y. Building linkedin's real-time activity data pipeline. **IEEE Data Eng. Bull.**, Citeseer, v. 35, n. 2, p. 33–45, 2012.

KANIOVSKYI, Y.; KOEHLER, M.; BENKNER, S. A containerized analytics framework for data and compute-intensive pipeline applications. In: **Proceedings of the 4th ACM SIGMOD Workshop on Algorithms and Systems for MapReduce and Beyond**. [S.l.: s.n.], 2017. p. 1–10.

KREPS, J.; NARKHEDE, N.; RAO, J. *et al.* Kafka: A distributed messaging system for log processing. In: **Proceedings of the NetDB**. [S.l.: s.n.], 2011. v. 11, p. 1–7.

KUROSE, J. F.; ROSS, K. W. **Redes de Computadores e a Internet: Uma abordagem top-down**. [S.l.]: Pearson, 2014.

LANEY, D. **3D Data Management: Controlling Data Volume, Velocity, and Variety**. [S.l.], 2001. Disponível em: <http://blogs.gartner.com/doug-laney/files/2012/01/ad949-3D-Data-Management-Controlling-Data-Volume-Velocity-and-Variety.pdf>.

MANYIKA, J.; CHUI, M.; BROWN, B.; BUGHIN, J.; DOBBS, R.; ROXBURGH, C.; BYERS, A. H. **Big data: The next frontier for innovation, competition**. [S.l.], 2011.

MELL, P.; GRANCE, T. *et al.* The nist definition of cloud computing. Computer Security Division, Information Technology Laboratory, National . . . , 2011.

MICROSOFT. **ETL (extrair, transformar e carregar) - Azure Architecture Center**. 2019. Disponível em: <https://docs.microsoft.com/pt-br/azure/architecture/data-guide/relational-data/etl>.

MORONEY, L. Firebase cloud messaging. In: **The Definitive Guide to Firebase**. [S.l.]: Springer, 2017. p. 163–188.

QUEMY, A. Data pipeline selection and optimization. In: **DOLAP**. [S.l.: s.n.], 2019.

RAJARAMAN, V. Big data analytics. **Resonance**, Springer, v. 21, n. 8, p. 695–716, 2016.

SATYANARAYANAN, M. Fundamental challenges in mobile computing. In: **Proceedings of the fifteenth annual ACM symposium on Principles of distributed computing**. [S.l.: s.n.], 1996. p. 1–7.

SHIFTEHFAR, R. **Uber's Big Data Platform: 100+ Petabytes with Minute Latency**. 2018. Disponível em: <https://eng.uber.com/uber-big-data-platform/>.

VERA-BAQUERO, A.; COLOMO-PALACIOS, R.; MOLLOY, O. Real-time business activity monitoring and analysis of process performance on big-data domains. **Telematics and Informatics**, Elsevier, v. 33, n. 3, p. 793–807, 2016.

VOUROS, G.; GLENIS, A.; DOULKERIDIS, C. The delta big data architecture for mobility analytics. In: **2020 IEEE Sixth International Conference on Big Data Computing Service and Applications (BigDataService)**. [S.l.: s.n.], 2020. p. 25–32.

WHITE, T. **Hadoop: The definitive guide**. [S.l.]: "O'Reilly Media, Inc.", 2012.