UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

ARQUITETURA MULTISSENSORIAL EM *FOG COMPUTING* PARA DISPOSITIVOS IOT COM FOCO EM AGRICULTURA DE PRECISÃO

JOÃO PAULO DOS SANTOS MORIJO

ORIENTADORA: PROF. DRA. KELEN CRISTIANE TEIXEIRA VIVALDINI

São Carlos – SP

Fevereiro/2019

UNIVERSIDADE FEDERAL DE SÃO CARLOS
CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

ARQUITETURA MULTISSENSORIAL EM *FOG COMPUTING* PARA DISPOSITIVOS IOT COM FOCO EM AGRICULTURA DE PRECISÃO

JOÃO PAULO DOS SANTOS MORIJO

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação, área de concentração: Redes e Sistemas Distribuídos
Orientadora: Prof. Dra. Kelen Cristiane Teixeira Vivaldini

São Carlos – SP
Fevereiro/2019

UNIVERSIDADE FEDERAL DE SÃO CARLOS

Centro de Ciências Exatas e de Tecnologia
Programa de Pós-Graduação em Ciência da Computação

Folha de Aprovação

Assinaturas dos membros da comissão examinadora que avaliou e aprovou a Defesa de Dissertação de Mestrado do candidato João Paulo dos Santos Morijo, realizada em 14/02/2019:

Profa. Dra. Kelen Cristiane Teixeira Vivaldini
UFSCar

Prof. Dr. Luis Carlos Trevelin
UFSCar

Profa. Dra. Kalinka Regina Lucas Jaquie Castelo Branco
USP

Que os vossos esforços desafiem as impossibilidades, lembrai-vos que as grandes coisas do homem foram conquistadas do que parecia impossível.

Charles Chaplin

Dedico este trabalho a todos que me incentivaram e ajudaram na realização deste sonho, sendo em destaques meus pais, Edson e Cilene, minha esposa Lívia, Meu filho Henrique e meu fiel companheiro Thor.

AGRADECIMENTOS

Agradeço primeiramente a Deus, por ter me dado a força necessária em todos os momentos de minha vida e a sabedoria para a conclusão deste trabalho. Agradeço a minha família, minha esposa Lívia pela paciência, amor e companheirismo em toda essa minha jornada; ao meu filho Henrique que está a caminho; aos meus Pais Edson e Cilene que sempre estiveram ao meu lado, me apoiando em minhas decisões e sempre torceram por mim; aos meus irmãos Daniel, Karen e Letícia por sempre estarem por perto e poder contar com eles.

Agradeço aos professores e orientadores Prof. Dr. Luis Carlos Trevelin e Prof. Dr. Fredy João Valente os quais sempre me encaminharam e direcionaram desde o início, tornando esse meu sonho possível. Agradeço também a Prof.ª Dr.ª Kelen Cristiane Teixeira Vivaldini pela orientação e auxílio nesta reta final do projeto.

Aos amigos e companheiros que estiveram comigo durante toda essa caminhada, destes posso destacar Ricardo Sabatine, Éttore Tognoli e Leonardo Lima; aos professores da universidade pelo compartilhamento do conhecimento, sendo fundamental para o crescimento acadêmico e pessoal.

Por fim, agradeço a Universidade Federal de São Carlos - UFSCAR por tornar meu sonho realidade.

RESUMO

A Internet das Coisas (IoT) é caracterizada por um ambiente computacional que pode ser dividido em três grandes camadas: rede de sensores, comunicação e inteligência, esta última localizada em um ambiente *fog* (nevoeiro) ou *cloud* (nuvem) formando um ambiente IoT. A primeira camada pode ser composta por uma ampla variedade de objetos, geralmente chamados de terminais, com várias capacidades computacionais, recursos arquitetônicos, uma variedade de sensores, atuadores e diferentes interfaces de comunicação e padrões para interconexão. A segunda camada pode fazer uso de muitas comunicações sem fio diferentes tecnologias, incluindo Wi-Fi, ZigBee, Bluetooth ou emergentes tecnologias de comunicação 6LoWPAN como Lora, e protocolos de transporte que incorporam estratégias como *publish/subscribe* para enviar mensagens que contenham dados dos sensores/*endpoints* para a camada de inteligência. Os resultados podem ser usados para monitoramento, inferência de problemas, tomada de decisões em nível de negócios, bem como ação em nível de sensor, enviando uma mensagem de atuação para um terminal. À medida que a rede de sensores IoT cresce, uma enorme quantidade de dados de várias fontes flui da camada do sensor para a camada de inteligência no ambiente da IoT. O problema é que, para tomar decisões baseadas em análises sobre esses dados, ele precisa ser concretos e precisos. A fusão de dados é uma maneira eficaz de melhorar a qualidade dos dados. No entanto, os ambientes de IoT ainda estão evoluindo e a melhor maneira ou local em que a fusão de dados deve acontecer é um problema em aberto. Este projeto apresenta uma arquitetura para a fusão de dados de sensores IoT implementando a fusão de dados com microserviços usando uma plataforma de contêiner embutida em um *middleware opensource* IoT baseado em uma infraestrutura *fog* que é capaz de escalar automaticamente conforme o fluxo de dados da camada de sensor cresce. Vários testes de desempenho de fusão de dados foram realizados para diferentes quantidades de pontos e leituras dos sensores sobre as tecnologias ZigBee e LoRa usando o algoritmo de fusão de dados Chauvenet em um ambiente agrícola o qual pode-se aplicar os conceitos da agricultura de precisão.

Palavras-chave: Agricultura de Precisão, Sensores, Middleware, Plataforma Multissensorial, fusão de sensores, IoT, Computação em Névoa.

ABSTRACT

The Internet of Things (IoT) is characterized by a computational environment that can be divided into three main layers: sensor network, communication and intelligence, the latter located in a fog or cloud environment forming an IoT environment. The first layer can be composed of a wide variety of objects, usually called terminals, with various computational capacities, architectural features, a variety of sensors, actuators and different communication interfaces and standards for interconnection. The second layer can make use of many different wireless communications technologies, including Wi-Fi, ZigBee, Bluetooth or emerging 6LoWPAN communication technologies such as Lora, and transport protocols that incorporate strategies like publish / subscribe to send messages containing sensor data / endpoints for the intelligence layer. The results can be used for monitoring, problem inference, business-level decision making, as well as sensor-level action by sending an actuation message to a terminal. As the IoT sensor network grows, a huge amount of data from various sources flows from the sensor layer to the intelligence layer in the IoT environment. The problem is that to make informed decisions about these data, it needs to be concrete and accurate. Data fusion is an effective way to improve data quality. However, IoT environments are still evolving and the best way or place where data fusion should happen is an open problem. This project presents an architecture for merging IoT sensor data by implementing data fusion with microservices using a container platform embedded in an IoT opensource middleware based on a fog infrastructure that is capable of automatically scaling according to the layer data flow sensor grows. Several data fusion performance tests were performed for different amounts of sensor points and readings on ZigBee and LoRa technologies using the Chauvenet data fusion algorithm in an agricultural environment which can be applied to the concepts of precision agriculture.

Keywords: Precision Farming, Sensors, middleware, Multisensory Platform, Sensor Fusion, IoT, Fog Computing.

LISTA DE FIGURAS

2.1 Camadas virtuais Cloud, Fog e Edge computing. 21

2.2 Custo x flexibilidade no ambiente Cloud (SaaS, PaaS e IaaS) 22

2.3 Esquema generalizado de fusão de dados 27

2.4 Principais componentes do nó sensor sem fio. 30

2.5 Funcionamento base sensor Ópticos. 32

2.6 Detecção de Objetos sensor Ultrassônico. 32

2.7 Exemplo de curva do termistor e sua Simbologia. 33

2.8 Arquitetura da pilha do protocolo IEEE 820.15.4/ZigBee 36

2.9 Exemplo comunicação Publish/Subscribe 39

3.1 Arquitetura do conceito do sensorHUB. 41

3.2 Estrutura BDAaaS implementada na arquitetura lambda 43

3.3 Arquitetura do EcoCIT 44

4.1 Arquitetura IoTUfscar 46

4.2 Arquitetura Máquinas virtuais / Contêineres. 49

4.3 Raspberry 51

4.4 ESP32 Lora / Arduino nano 52

4.5 Sensor de Temperatura e umidade DHT 22 53

4.6 Sensor de Pressão Atmosférica BMP180 53

4.7 Sensores anemômetro, biruta eletrônico e pluviômetro 53

4.8 Comando para alocação de novos Contêineres 54

4.9 Arquitetura Kaa 57

4.10 Ilustração arquitetura implementada 59

5.1 Arquitetura implementada - *Server/Endpoint* 60

5.2 Trecho do Código elaborado para estação meteorológica 61

5.3 Trecho do Código elaborado para transmissão *publisher* MQTT 62

5.4 Fonte de dados para aplicação - Classe Subscriber.java 63

5.5 Model da aplicação - Classe StationDataModel.java 64

5.6 Algoritmo de Chauvenet - Classe Chauvenet.java 64

5.7 Trecho do código - Classe SimilarFusionFunctions.java 65

5.8 Trecho do código - Classe SimilarFusionFunctions.java 65

5.9 Trecho do código - Classe SimilarFusionFunctions.java 66

5.10 Inicialização Kaa - Classe WeatherStation.java 67

5.11 Sincronização dos dados - Classe WeatherStation.java 68

5.12 Definição período de coleta Kaa 69

5.13 Tipagem dos dados coletados pelo *endpoint* 69

5.14 Query MongoDB - Seleção de itens 71

5.15 *Dashboard* para análise dos dados 71

6.1 Visão vegetativa do talhão. 72

6.2 Estação Meteorológica em Campo 73

6.3 Cenário do Proposto. 74

6.4 Comparativo de dados fusionados - Pressão atmosférica 74

6.5 Comparativo de dados fusionados - Umidade relativa do ar 75

6.6 Consumo de *hardware* teste em campo 76

6.7 CPU *Utilization* 77

6.8 CPU Load 77

6.9 Memória utilizada 77

6.10 Mensagem estruturada JSON para dados da estação meteorológica 78

6.11 Comparativo Servidor X Raspberry PI B+ 78

6.12 Teste de carga fusão de dados 79

6.13 Teste de Stress - CPU *Utilization* 80

6.14 Teste de Stress - CPU *Load* 80

6.15 Teste de Stress - Memória utilizada 81

Lista de Tabelas

2.1 Tabela adaptada características dos níveis de fusão(VILLANUEVA, 2009) 27
2.2 Tabela simplificada dos métodos para medição de umidade 34
2.3 Classificação Wi-Fi 802.11 35
2.4 Classificação Bluetooth 35
2.5 Faixa de frequência protocolo ZigBee 36
3.1 Dispositivos de comunicação sensorHub 42
4.1 Propriedades de configuração do interpretador MongoDB 57

SUMÁRIO

CAPÍTULO 1 – INTRODUÇÃO **16**

1.1 Contexto 16

1.2 Motivação 17

1.3 Objetivos 18

1.4 Visão Geral 18

CAPÍTULO 2 – FUNDAMENTAÇÃO TEÓRICA **19**

2.1 Conceitos de Agricultura de Precisão 19

2.2 Tecnologia da Informação na Agricultura de Precisão 20

2.3 IoT - Internet of Things 20

2.3.1 *Cloud Computing* 21

2.3.2 *Fog Computing* 23

2.3.3 *Edge Computing* 24

2.4 *Middleware* IoT 24

2.5 Fusão de dados 25

2.5.1 Níveis de Fusão 26

2.5.1.1 Fusão de Baixo Nível 28

2.5.1.2 Fusão de Nível Médio 29

2.5.1.3 Fusão de Alto Nível 29

2.6 Conceito de Sensores 30

2.6.1 Sensoriamento . . . 30
2.6.2 Comunicação . . . 31
2.6.3 Processamento . . . 31
2.7 Categoria de sensores . . . 31
2.7.1 Sensores Ópticos . . . 31
2.7.2 Sensores ultrassônicos . . . 32
2.7.3 Sensores Térmicos . . . 32
2.7.4 Sensores de umidade . . . 33
2.8 Tecnologias de comunicação . . . 34
2.8.1 Wi-Fi (IEEE 802.11) . . . 34
2.8.2 Bluetooth (IEEE 802.15.1) . . . 35
2.8.3 ZigBee (IEEE 802.15.4) . . . 35
2.8.4 LoRa . . . 36
2.9 Protocolos de Transporte . . . 37
2.9.1 HTTP . . . 38
2.9.2 MQTT . . . 38
2.9.3 Comparativo HTTP e MQTT . . . 39
2.10 Conclusão do Capítulo . . . 40

CAPÍTULO 3 – TRABALHOS RELACIONADOS **41**

3.1 Coleta de dados para ampla massa distribuída de sensores . . . 41
3.2 Uma estrutura de análise de Big Data para aplicativos IoT na nuvem . . . 42
3.3 EcoCIT: Uma Plataforma Escalável para Desenvolvimento de Aplicações de IoT 44
3.4 Conclusão do Capítulo . . . 45

CAPÍTULO 4 – COMPONENTES DA ARQUITETURA **46**

4.1 Conceito da Arquitetura . . . 46

4.1.1 Arquitetura de Microserviços 47
4.1.1.1 Resiliência 47
4.1.1.2 Escalabilidade 48
4.1.2 Arquiteturas monolíticas 48
4.1.3 Virtualização 48
4.1.3.1 Máquina Virtual 49
4.1.3.2 Contêiner 49
4.2 Elementos da arquitetura 50
4.2.1 Sistema embarcado 51
4.2.1.1 Raspberry PI 3 B+ 51
4.2.1.2 Arduino nano 51
4.2.1.3 ESP32 52
4.2.1.4 Sensores 52
4.2.1.5 Banco de dados 54
4.2.2 Container Docker JAVA 54
4.2.3 Kaa Iot 55
4.2.4 Zeppelin 56
4.2.5 Zabbix 58
4.3 Conclusão do Capítulo 58

CAPÍTULO 5 – IMPLEMENTAÇÃO 60

5.1 Estação Meteorológica 61
5.2 Fusão de dados 62
5.2.0.1 Fonte de dados 62
5.2.0.2 Definição dos dados 63
5.2.0.3 Método de Fusão Chauvenet 64
5.2.0.4 Inserção Kaa 66

5.3 Kaa IoT platform 68
5.4 Análise dos dados 70
5.5 Conclusão do Capítulo 71

CAPÍTULO 6 – AVALIAÇÃO **72**

6.1 Avaliação de Desempenho 76
6.2 Teste de Carga 79
6.3 Teste de Stress 79
6.4 Conclusão do Capítulo 81

CAPÍTULO 7 – CONSIDERAÇÕES FINAIS **82**

7.1 Conclusões 82
7.2 Contribuições 83
7.3 Trabalhos Futuros 83

REFERÊNCIAS **84**

LISTA DE ABREVIATURA E SIGLAS **88**

Capítulo 1

INTRODUÇÃO

1.1 Contexto

O termo IoT (*Internet of Things*), refere-se a um número crescente de objetos inteligentes heterogêneos totalmente interconectados, capazes de se comunicar através da internet utilizando uma gama de protocolos de transporte. Ambiente esse compreendido em três grandes camadas: rede de sensores e atuadores, camada de comunicação e camada de aplicação. Sua primeira camada pode ser composta por uma grande variedade de objetos com as mais diversas capacidades computacionais, com uma ampla gama de sensores e atuadores interconectados, incluindo dispositivos móveis como *smartphones*, incorporando a computação ubíqua e pervasiva no contexto da gama que é o IoT. A segunda camada foi construída por técnicas diversas implementadas para conectar objetos inteligentes, desempenha papel primordial perante ao consumo de energia entre dispositivos, tornando-se assim um fator crítico, nesta camada apresentam-se tecnologias como WiFi, Bluetooth, Zigbee, LoRa e RFID. Em último, a camada de aplicação está localizada em ambiente de computação em nuvem *(cloud)* ou neblina *(fog)* responsável por todo o tratamento e análise dos dados coletados.

A adoção das tecnologias IoT em diversas áreas industriais incluindo manufatura, alimentícia, agricultura, cidades inteligentes, saúde e entre outras, é premente em função dos benefícios que podem ser auferidos em qualidade, segurança alimentar, saúde e bem estar, além de inovação em modelos de negócios e melhorar os resultados financeiros.

O conceito abordado neste trabalho foi a agricultura de precisão, essa que tornou-se um fator importante no desenvolvimento tecnológico apresentado no campo ao longo das últimas décadas. Soluções como sistemas de geoprocessamento, automação, sensoriamento e gestão ajudaram o crescimento da produção na agricultura e uma redução de custo nunca visto antes

(NOWAK; PIERCE, 1999). Embora existam benefícios, há muitos desafios a serem superados. A carência de infraestrutura tecnológica no campo e a falta de padronização dos dispositivos para comunicação, dificultaram a aceitação dessa prática agrícola.

No conceito da agricultura de precisão, todas as fases devem ter seus dados coletados e analisados, tornando essa uma tarefa crucial (FOUNTAS; PEDERSEN; BLACKMORE, 2005). No entanto, a fabricação de equipamentos de TIC (Tecnologia da Informação e Comunicação) ainda é muito cara e em alguns casos não atingem os resultados esperados. Cenário este que demanda a substituição gradativa por novos dispositivos IoT, ou a sua adequação à padrões de interoperabilidade ou ainda, a sua incorporação por uma plataforma IoT que seja capaz de captar uma diversidade de dispositivos e padronizar as informações coletadas.

1.2 Motivação

O monitoramento em tempo real e interativo com sistemas embarcados são frequentemente barrados pela latência e recepção dos dados no campo, suas principais soluções de comunicação de dados entre equipamentos no meio agrário são soluções M2M (*Machine-to-Machine*) baseadas em tecnologias de rede móvel convencional (GSM – *Global System for mobile Communication*/ GPRS – *General Packet Radio Service*) enfrentando assim grandes problemas técnicos, devido a precariedade ou a falta de recursos que supram essas redes no meio agrário.

O principal problema apresentado na dificuldade da coleta dos dados, resulta em perdas significativas de informações, essas necessárias para um melhor aproveitamento das técnicas da agricultura de precisão (FOUNTAS; PEDERSEN; BLACKMORE, 2005).

A carência no desenvolvimento de sistemas para recolher, processar os dados dos sensores e gerar relatórios, sistemas esses denominados FMIS (*Farm Management Information Systems*) são citados em pesquisas elaboradas por diversas universidades em conjunto a centros de pesquisa dos países nórdicos. Como resultado (PESONEN; KOSKINEN; RYDBERG, 2008), demonstra como as tecnologias e compreensões agrícolas podem trabalhar em conjunto, formando um consenso comum para o gerenciamento da informação e recomenda:

- Formatos de dados e modelos para criação de um padrão aberto para a interface do sistema;
- Capacidade do sistema de comunicar-se de forma transparente do domínio agrícola com outros domínios;
- Implantação de infraestruturas de redes sem fio confiáveis e eficientes no ambiente rural;

- Estudos que auxiliam no processo para tomada de decisão dos agricultores, como base para o desenvolvimento de sistemas de informação.

1.3 Objetivos

Diante esse cenário, o objetivo do trabalho é apresentar uma abordagem para minimizar a complexidade na coleta e análise dos dados entre sensores para obtenção de uma infraestrutura que possam prover esse tipo de informação. Este trabalho foca no desenvolvimento de uma plataforma IoT em um ambiente *Fog Computing* (Computação em Névoa), que inclui uma rede de dispositivos IoT interconectados por rede sem fio com tecnologias diversas como WiFi, LoRa e ZigBee, um *middleware* IoT *open-source* ao qual foi atribuído a capacidade do tratamento individual de cada objeto sensor através da técnica de fusão de dados.

Como ambiente de testes foi desenvolvida uma estação meteorológica composta de diferentes sensores, como *endpoint*, a partir da qual se pretende validar a proposta. Entende-se ser suficiente para comprovar os conceitos e modelos adotados no projeto.

1.4 Visão Geral

Este trabalho está dividido em sete capítulos. Este capítulo introdutório apresentou o contexto, a motivação e os objetivos.

O Capítulo 2 apresenta a fundamentação teórica que embasa o trabalho. Neste capítulo são abordados os conceitos da Agricultura de precisão, IoT, *Middleware*, fusão de dados, Sensores e protocolos de transporte, teorias e técnicas que são utilizadas na implementação.

O Capítulo 3 apresenta o estudo de caso com os trabalhos relacionados a essa área de pesquisa.

O Capítulo 4 apresenta a arquitetura proposta e a utilizada no desenvolvimento, assim como as funcionalidades das ferramentas.

O Capítulo 5 apresenta as etapas realizadas no desenvolvimento da arquitetura bem como o desenvolvimento da estação meteorológica utilizada para homologação da arquitetura.

O Capítulo 6 apresenta os resultados gerados nos teste de campo, comparativo entre tempos de respostas e desempenho da arquitetura.

Por fim, o Capítulo 7 apresenta a conclusão do trabalho e a sugestão de trabalhos futuros.

Capítulo 2

FUNDAMENTAÇÃO TEÓRICA

2.1 Conceitos de Agricultura de Precisão

A função econômica atual trabalha por constantes melhorias, sempre em busca de aumentar sua produtividade e maximizar seus lucros, esses possuem intensivo uso de tecnologias. No setor agrícola essa realidade não é incomum, essa procura é apresentada pela utilização do conceito da agricultura de precisão, no caso, a utilização de tecnologias e técnicas que visam um melhor aproveitamento dos recursos.

Na literatura são apresentadas algumas definições para essa prática agrícola, definições como as apresentadas por Pierce e Nowak (1999) autores citados como base em diversos artigos analisados. Segundo os autores, o conceito se refere as tecnologias aplicada e os princípios para a gestão associada com todos os aspectos de espaço de produção agrícola e variabilidade do tempo para melhorar o desempenho e qualidade ambiental da cultura.

O autor Molin (2004) apresenta em sua visão, a agricultura de precisão como uma forma de gerenciamento da produção agrícola, sendo um conjunto de técnicas e procedimentos para otimização de sistemas de produção de culturas (MOLIN, 2004).

A agricultura de precisão tem como base a oscilação entre os fatores com influência em sua produção agrícola. Fatores esses relacionados ao solo, água, umidade, temperatura entre outros. Com essa análise, torna-se possível a orientação ao manejo da cultura e a aplicação de insumos de forma fixa e precisa. Possibilidade alcançada com a utilização de tecnologias como o GPS (*Global Positioning System*), GIS (*Geographic Information System*), sensores, entre diversas outras (NAIME; NETO; VAZ, 2011).

2.2 Tecnologia da Informação na Agricultura de Precisão

No conceito de agricultura de precisão, as principais atividades são o recolhimento e tratamento de dados. Esses processos incluem ferramentas que consistem em uma extensa gama de técnicas, tecnologias de informação, comunicação, tecnologia de sensores e o sistema de gestão de economia (FOUNTAS; PEDERSEN; BLACKMORE, 2005).

Existe um agrupamento de tecnologias que possibilitam a realização da agricultura de precisão, esse que pode ser dividido em cinco conjuntos: computadores, GPS, sistemas de georreferenciamento, sensores e controladores na aplicação de insumos. Acredita-se que a integração dessas tecnologias permita aos agricultores realizem façanhas com resultados de qualidade acima dos já alcançados (NOWAK; PIERCE, 1999).

Com o constante crescimento das tecnologias e técnicas que visam uma melhor interação e comunicação agregando-se o conceito de IoT (*Internet of Things* / Internet das Coisas) a esse processo da agricultura.

2.3 IoT - Internet of Things

O termo IoT foi introduzido em meados de 1999, tendo sua origem atribuída a Kevin Ashton (WHITMORE; AGARWAL; XU, 2015). Pode-se definir esse conceito como conjunto de *hardwares*, sensores, sistemas de armazenamento, protocolos de comunicação entre outras técnicas e tecnologias que ao se unirem representam um novo sistema de computação e comunicações.

A Internet das Coisas pode ser considerada como uma expansão dos aplicativos da internet em nosso meio físico, possibilitando não apenas a interação entre pessoas, mas sim tecnologias que possam se comunicar e possibilitem a inovação entre diversos dispositivos.

O conceito IoT pode ser baseado em três camadas, entretanto alguns pesquisadores incluem uma quarta camada o *middleware*, separando este procedimento da camada de rede, a seguir são apresentadas essas camadas que possibilitam diferentes abordagens e utilizações.

- Camada de sensores: Camada responsável pela coleta dos dados de diferente dispositivos (sensores) a fim de prover informações relevantes ao meio.
- Camada de Rede: Sua função é prover o serviço de roteamento entre dispositivos e aplicações, com o processo pelo qual a rede consegue identificar o destinatário dos pacotes e encontrar o melhor caminho.

- Camada de *Middleware*: Responsável em integrar diferentes tipos de protocolos e meios de comunicação para com isso remover a complexidade e curva de aprendizado que os desenvolvedores precisam ter sobre determinados equipamentos e se preocuparem mais com as funcionalidades a serem desenvolvidas.

- Camada de Aplicação: Sua função é a representação dos dados, sendo responsável em permitir ao usuário final o acesso aos recursos.

No contexto de IoT apresenta-se três estruturas que compõem essa ampla arquitetura, conforme apresentado na Figura 2.1, serão detalhadas nas subseções a seguir.

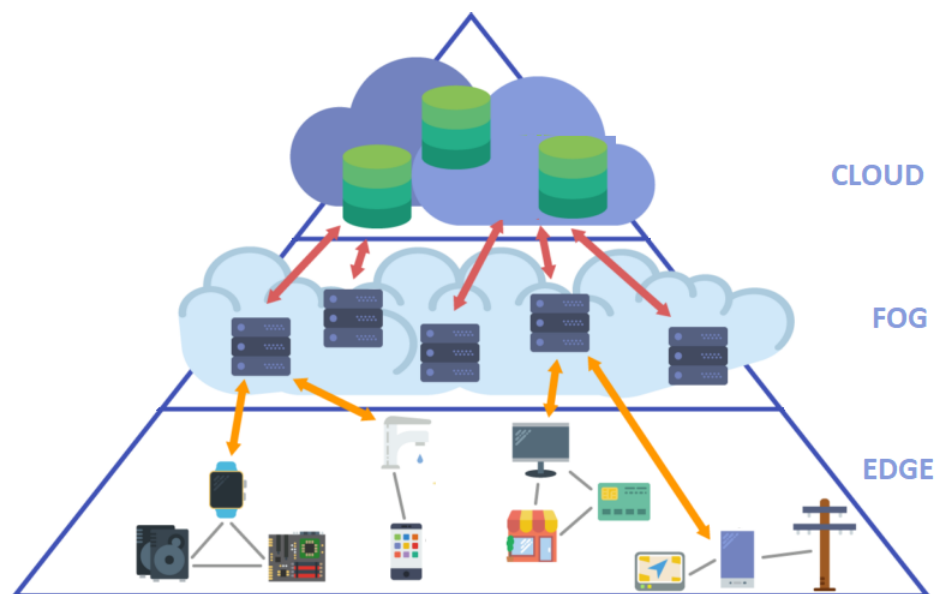

Figura 2.1: Camadas virtuais Cloud, Fog e Edge computing.
Fonte: Adaptado de (ERKLART, 2018)

2.3.1 *Cloud Computing*

O conceito de *Cloud Computing* (Computação em Nuvem) vem do avanço e a junção de técnicas e tecnologias a fim de simplificar o fornecimento de serviços computacionais como por exemplo: Servidores, armazenamento, bancos de dados, rede entre outros recursos, sendo sua principal característica a virtualização destes dispositivos (RAY, 2017).

Os principais modelos de serviço oferecidos no ambiente *cloud computing* podem ser identificados como ilustrado na Figura 2.2, essa ilustração representa os níveis de complexidade para administração dos serviços oferecidos comparado com os custos pela abstração dos mesmos.

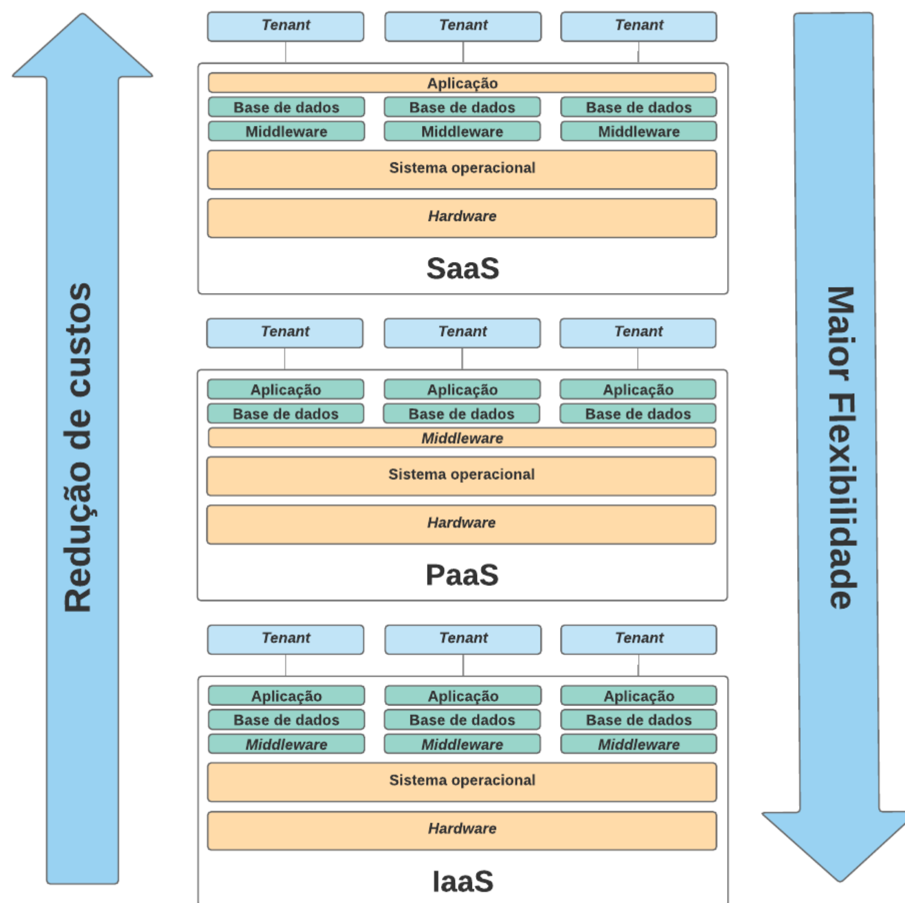

Figura 2.2: Custo x flexibilidade no ambiente Cloud (SaaS, PaaS e IaaS)
Fonte: Adaptado de (IBM, 2018)

- SaaS (*Software as a Service*) - O conceito de *Software* como serviço tem por cultura o fornecimento de aplicações online, como por exemplo servidores de e-mail.
- PaaS (*Platform as a Service*) - O conceito de Plataforma como serviço é uma camada acima da IaaS fornecendo componentes como bancos de dados, armazenamento, monitoramento, balanceamento de carga e roteamento.
- IaaS (*Infrastructure as a Service*) - O conceito de Infraestrutura como serviço, é realizar o compartilhamento dos recursos de infraestrutura tais esses como processadores, memória, rede e armazenamento, essa camada tem como responsável por toda a configuração e instalação dos *softwares* necessários o proprietário.

Uma das principais vantagens na utilização do conceito de *cloud computing*, vem a ser a escalabilidade de recursos, podendo diminuir recursos de ambientes ociosos e acrescentar em ambientes que necessitam, tudo isso de forma simples e rápida. Sua principal limitação está relacionada a problemas como a largura de banda e custos, problemas que podem ser sentido

em aplicações sensíveis a latência de comunicação (RAY, 2017). Para amenizar esse problema surge então o conceito de *fog computing*.

2.3.2 *Fog Computing*

A arquitetura atualmente conhecida como *Fog Computing* consiste na alocação do poder de processamento mais próximo ao limite da rede, conceito esse adotado pela *Cisco Systems* como um novo paradigma em meados de 2012 (HAJIBABA; GORGIN, 2014). Este modelo de arquitetura computacional descentralizada representa o processamento e armazenamento dos dados entre a fonte de dados e a nuvem conforme apresentado na Figura 2.1.

A arquitetura *Fog* força a inteligência para o nível de rede da área local, processando dados em um nó ou *gateway*. Seu intuito é a redução de latência na rede e largura de banda, solucionando totalmente ou parcialmente os problemas encontrados no conceito da *cloud computing* (RAY, 2017).

Suas principais características são:

- Heterogeneidade: O ambiente *Fog* torna-se uma plataforma virtualizada que oferece processamentos computacionais entre o ambiente *cloud* e *edge computing*.
- Distribuição geográfica: Sua utilização oferece serviços de alta qualidade em diferentes pontos de acessos, exemplos disso são os tipos de sensores podendo esses serem fixos ou móveis.
- Baixa latência: Conceito implementado para suprir a qualidade de comunicação entre dispositivos em uma rede.
- Processamento em tempo de execução: Aplicações em ambiente *fog computing* tem como principal característica o processamento em tempo real, este tipo de processamento é utilizado em ambientes com criticidade ou necessidades de monitoramento específicos.
- Comunicação sem fio: Por se tratar de ambientes fora da internet, grande parte dos dispositivos se comunicam com tecnologias sem fio, proporcionando flexibilidade, robustez e redução de custos.
- Interoperabilidade: É a capacidade do sistema comunicar-se de forma transparente com os demais dispositivos presentes, trocando informações de maneira eficaz e eficiente.

2.3.3 *Edge Computing*

A arquitetura *Edge computing* surgiu como uma metodologia para processamento local, fora de um ponto central como exemplo *Middleware*. O processamento nos *endpoints*, reduzem os custo de comunicação entre as arquiteturas, pois com o processamento diretamente nos dispositivos reduz o tráfego na rede, os custos decorrentes disso, latência, além de melhorar a qualidade do serviço (QoS) (RAY, 2017).

A camada *edge* reduz significativamente o processamento do *middleware*, por se tratar de dados já processados, reduz a carga presente na rede, possibilitando assim, aumentar a segurança dos dados, criptografando-os e o *middleware* por sua vez descriptografa os dados. Entre outras vantagens vem a escalabilidade do sistema, por se tratar de um sistema distribuído.

2.4 *Middleware* IoT

O *middleware* é a camada responsável em abstrair toda a complexidade existente entre o *hardware* e *software* para permitir um melhor aproveitamento das tarefas a serem desenvolvidas.

Da perspectiva da computação, um *middleware* fornece uma camada entre o *software* e o sistema. No contexto do IoT, é provável que haja heterogeneidade considerável nas tecnologias de comunicação em uso e também nas tecnologias de nível de sistema, e um *middleware* deve suportar ambas as perspectivas, conforme necessário(RAZZAQUE et al., 2014).

Essa camada pode-se dividir em dois tipos de características sendo essas funcionais e não funcionais. Os requisitos funcionais coletam serviços ou funções que um *middleware* fornece e os requisitos não funcionais (por exemplo, confiabilidade, segurança e disponibilidade) capturam suporte a QoS ou problemas de desempenho(RAZZAQUE et al., 2014).

Alguns dos principais requisitos funcionais do *middleware* IoT podem ser destacados como os seguintes:

- Gerenciamento de recursos: Taxa de QoS aceitável e esperada para aplicativos em um ambiente onde os recursos que afetam a QoS são restritos, como a IoT, é importante que os aplicativos tenham um serviço que gerencie esses recursos.
- Gerenciamento de dados: O gerenciamento dos dados pelo *middleware* é sua principal função. O *middleware* deve ser responsável em fornecer um serviços de gerenciamento de dados a aplicativos, incluindo aquisição de dados,processamento de dados e seu arma-

zenamento. O pré-processamento pode incluir filtragem de dados, compactação de dados e agregação de dados.

- Gerenciamento de eventos: Os eventos registrados no *middleware* devem ser registrados a fim de observações. Esta função é responsável em analisar em tempo real os dados com alto tráfego e encaminhá-los em tempo de execução para cada aplicação correspondente.

Alguns dos principais requisitos não funcionais do *middleware* IoT podem ser destacados como os seguintes:

- Escalabilidade: Um *middleware* IoT precisa ser dimensionável para acomodar o crescimento da rede e suas aplicações.
- Confiabilidade: Um *middleware* deve permanecer operante durante toda o seu funcionamento, mesmo na presença de falhas. A confiabilidade do *middleware* ajuda a alcançar a confiabilidade no nível do sistema.
- Disponibilidade: Um *middleware* que suporte aplicativos da IoT, especialmente aqueles de missão crítica, deve estar disponível ou aparecer sempre disponível. Mesmo se houver uma falha em algum lugar do sistema, o tempo de recuperação e a frequência de falhas devem ser pequenos o suficiente para atingir a disponibilidade desejada.
- Tempo real: Um *middleware* deve fornecer serviços em tempo real, entregas atrasada de dados podem tornar sistema ou aplicações inúteis ou mesmo perigosa de acordo com a criticidade do sistema.

2.5 Fusão de dados

A Internet das Coisas (IoT) vai se tornar uma tecnologia chave, dado que já em 2020 teremos algo em torno de 50 bilhões de objetos interconectados(ALAM et al., 2016).

Um grande desafio é prover a fusão e análise de *Big Data* de IoT, bem como o compartilhamento dos resultados em tempos adequados, para habilitar a tomada de decisão de forma precisa e eficiente em ambientes ubíquos sustentáveis, tais como cidades inteligentes e agriculturas de precisão. Além destas áreas, as estratégias de fusão de dados gerados por IoT beneficiarão áreas de aplicação tais como veículos autônomos e aprendizado profundo.

As terminologias técnicas encontradas na literatura como, integração de sensores, fusão da informação e fusão de dados, são a utilização da combinação de dados coletados de diversas

origens (SALUSTIANO, 2006). Esse conceito é muito utilizado para combinação de sensores com baixa qualidade, técnica essa para extrair informações mais precisas (KHALEGHI et al., 2013).

A fusão de sensores faz com que exista a possibilita de obtenção de novas informações a partir dos sensores em utilização, dependendo dos sensores utilizados a fusão pode obter dados que apenas seriam coletados com sensores específicos (SALUSTIANO, 2006). Sua utilização agrega grandes vantagens estas como:

- Tolerância a falha - É a capacidade do sistema a continuar suas operações normalmente mesmo após o acontecimento de falha em alguns dos seus componentes. Sua capacidade pode diminuir mas deve permanecer em funcionamento.
- Confiabilidade - É a capacidade das informações coletadas pelos sensores serem afirmadas pela verificação de diferentes sensores.
- Sinergia - Informações de um sensor solitário podem ser deficientes ou inadequadas, no entanto, sensores correlativos podem ser utilizados para produzir suposições que seriam difíceis de fazer com a utilização de apenas um único componente sensor.
- Redução na ambiguidade - O arranjo de informações diminui a ambiguidade na elucidação da estima deliberada, sendo que qualidades dispersas no conjunto das medidas podem ser descartadas por algum modelo adotado.
- Custo - A utilização de alguns sensores pode ter seu custo menos elevado com relação à preparação de tempo, ativos e materiais computacionais, e pode haver uma substituição de um sensor de despesas de alta precisão e escalonamento por alguns sensores de menor custo em situações onde a infelicidade na exatidão é suportada.

2.5.1 Níveis de Fusão

No contexto de fusão de dados esse que pode ser classificado em três níveis distintos de acordo com a proximidade da informação a ser consolidada. Na Figura 2.3 é ilustrada a classificação dos níveis de fusão de dados.

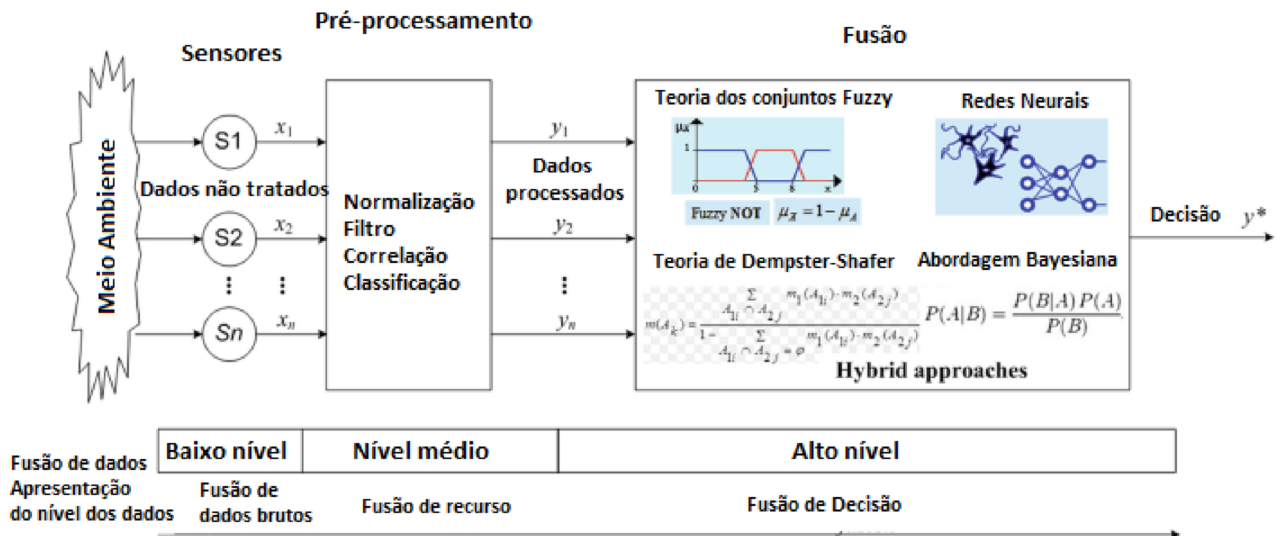

Figura 2.3: Esquema generalizado de fusão de dados
Fonte: Adaptado de (DOLGIY; KOVALEV; KOLODENKOVA, 2018)

A Tabela 2.1 adaptada (VILLANUEVA, 2009) apresenta algumas das características e algoritmos presentes em cada um dos níveis a serem descritos.

Tabela 2.1: Tabela adaptada características dos níveis de fusão(VILLANUEVA, 2009)

Nível da fusão	**Características dos dados**	**Algoritmos de fusão de dados**
Baixo Nível	Dados Brutos	Métodos estatísticos para combinação de informações redundantes - Média Ponderada; Filtro de Kalman; Princípio da Máxima Probabilidade; Conjunto Fuzzy
Nível Médio	Características extraídas dos dados e imagens	Métodos estatísticos para combinação de informações complementares - Teorema de Bayes; Princípio do Máximo a Posteriori; Métodos inteligentes e Híbridos baseados na extração e ponderação do conhecimento quantitativo
Alto Nível	Combinação de informações com múltiplas fontes de dados	Métodos Inteligentes e Híbridos baseados na extração e ponderação do conhecimento qualitativo - Regras de Dempster-Shafer; Funções de Crédito; Variáveis Nebulosas; Lógica Neural; Heurísticas Especializadas Híbridas Inteligentes

2.5.1.1 Fusão de Baixo Nível

Este nível também conhecido como nível de sensores, nível de dados brutos ou nível de sinais é a porta de entrada dos dados ainda sem nenhum tipo de tratamento são de entrada e depois são unidos. Como resultado desta associação, espera-se obter novos dados mais exatos e informativos.

A combinação neste nível resulta em informações que são de natureza indistinguível de sensores, no entanto, com um retrato mais preciso ao seu meio físico. O aprimoramento na qualidade da informação será identificado com o cálculo usado para executar a fusão (SALUSTIANO, 2006). Neste nível de combinação, a informação adquirida deve ser sincronizada para que o procedimento de combinação aconteça de forma eficaz. No caso de não haver sincronia de informações, é concebível fazer a utilização de registros da estação de obtenção das informações juntamente com as informações adquiridas com o objetivo final de reunir as informações de diferentes sensores nesse meio tempo (DOLGIY; KOVALEV; KOLODENKOVA, 2018). Pode utilizar de estratégias em estruturas apropriadas, por exemplo, *Timestamp* globais, podem ser utilizadas para sincronizar estimativas de sensores.

Um dos procedimentos usados para combinar sensores de um tipo similar é o modo de aplicar médias aritméticas à informação. Uma abordagem para refinar este procedimento é aplicar uma média ponderada, e o inverso da estimativa de exatidão de cada dispositivo é utilizado como parâmetro para a ponderação. Dentro deste contexto pode-se apresentar técnicas como o filtro de Kalman e o critério de Chauvenet este utilizando na implementação deste projeto.

O filtro de Kalman intitulado uma abordagem de filtragem linear e problemas de predição, é utilizado no formato *predictor-corrector* onde a ideia geral é projetar o estado para frente, usando uma função de transição de estado e subsequente este estado é corrigido (BISHOP; WELCH, 2001). O algoritmo do filtro de kalman pode ser dividido em duas fases distintas: uma fase de atualização de tempo e uma fase de atualização de medição:

- Atualização de tempo - (*Time Update*): Responsável por estruturar a condição atual dos fatores à frente do seu tempo.
- Atualização de Medição - (*Measurement Update*): Procedimento de correção altera essa medida antecipada por uma estimativa atual.

O método implementado para esse projeto é critério de Chauvenet, utiliza um conjunto de medidas de um objeto, para poder eliminar os valores discrepantes deste conjunto ou como também conhecidos os *outliers* (TAYLOR, 1997).

O calculo do critério de Chauvenet inicia-se com o calculo da média e os desvios padrão pelos quais x_{sus} difere de x (TAYLOR, 1997).

$$t_{sus} = \frac{|x_{sus} - \overline{x}|}{\sigma_x} \tag{2.1}$$

Em seguida, calculamos a probabilidade Prob(*outside* $t_{sus}\sigma$) subtraindo 100% menos a probabilidade Prob(*within* $t_{sus}\sigma$). As probabilidades são calculadas usando a distribuição de Gauss por um intervalo específico. O Prob(*outside* $t_{sus}\sigma$) multiplicado por N, o número de medições resulta em n, o número esperado como desviante como x_{sus} (TAYLOR, 1997).

$$n = N * Prob(outside\ t_{sus}\sigma). \tag{2.2}$$

Por fim se n for menor que 1/2, então, de acordo com o critério de chauvenet, o valor x_{sus} pode ser rejeitado (TAYLOR, 1997).

Desta forma torna-se possível a identificação e eliminação de *outliers* entre as aferições realizadas. Isso faz com que as informações sejam descartadas dos sensores que podem ter erros em contraste com os sensores alternativos.

2.5.1.2 Fusão de Nível Médio

Este nível é intitulado de nível de médio ou nível de recurso. Executa-se uma fusão de formas como resultado de que novos objetos, ou cartões de objetos que podem ser usados para outros problemas, por exemplo, de segmentação e reconhecimentos. Também neste nível há um processamento de dados, ou seja, uma filtragem para eliminação de ruídos, normalização dos dados, correlação, classificação de dados e uso de métodos de mineração de dados (DOLGIY; KOVALEV; KOLODENKOVA, 2018).

A motivação por trás desse nível é aumentar os dados de um sensor a partir do exame das informações dos sensores próximos. A partir disso, é concebível perceber os projetos nas informações e, além disso, dispor de dados questionáveis concebíveis (DOLGIY; KOVALEV; KOLODENKOVA, 2018).

2.5.1.3 Fusão de Alto Nível

Este nível é nomeado como nível de decisão ou nível de símbolos. As estratégias mais usuais e entendidas para a combinação de informações são técnicas probabilísticas (redes bayesianas, a teoria das provas); métodos inteligentes computacionais (Dempster-Shafer, teoria de

conjuntos indistintos e redes neurais). Estes métodos permitem apresentar a opinião coordenada e uniforme no processo de diagnóstico à pessoa que toma a decisão (TORRES et al., 2017).

2.6 Conceito de Sensores

Os sensores são dispositivos com a capacidade de captar ações ou estímulos externos e converter esses em frequências que possam ser lidas e interpretadas por sistemas computacionais.

Um nó sensor tem como característica alguns componentes principais, como: unidade de comunicação sem fio, fonte de alimentação, unidade de sensoriamento e unidade de processamento (NOGUEIRA et al., 2004), componentes esses estão ilustrados na Figura 2.4.

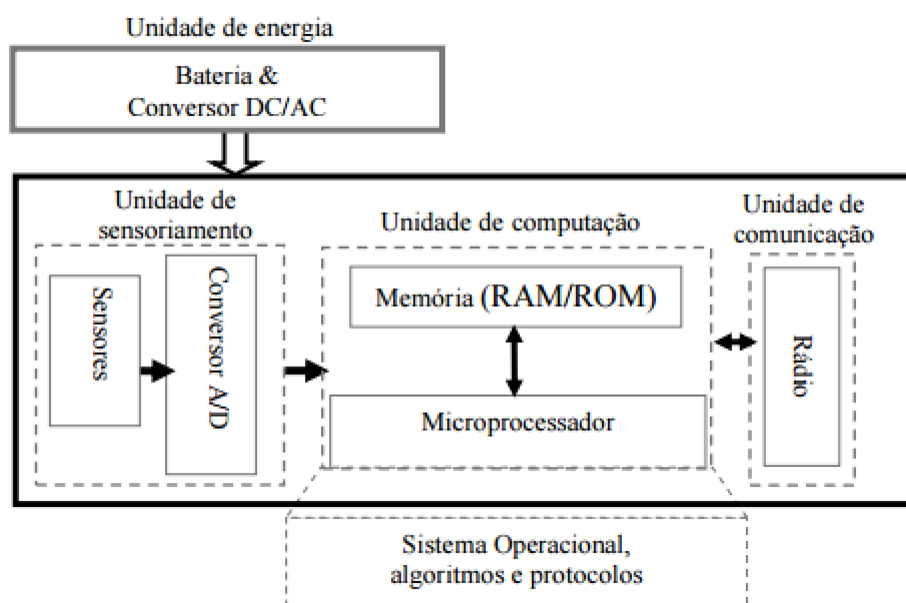

Figura 2.4: Principais componentes do nó sensor sem fio.
Fonte: (NOGUEIRA et al., 2004).

2.6.1 Sensoriamento

Um dispositivo sensor é aquele que produz uma resposta mensurável para uma mudança na condição física (medidas de temperatura, distância, posicionamento, pressão, etc). Os dispositivos de sensores são construídos especificamente para uma determinada função, podendo assim haver uma grande diversidade entre modelos e aplicações (NOGUEIRA et al., 2004). A grande maioria dos sensores no mercado já possuem capacidade de processar esses sinais e transformá-los em dados inteligíveis, esses dispositivos que estão inseridos no meio físico, estão sujeitos a erros de suas medições provocadas, por exemplo, por ruídos eletromagnéticos, falhas

de componentes e interferências, falhas essas que podem ser tratadas e corrigidas pela fusão dos dados.

2.6.2 Comunicação

Radiofrequência: Baseado em ondas eletromagnéticas com frequência variando de dezenas de KHz a centenas de GHz. Seu consumo irá depender das operações realizadas pelo transmissor, que torna-se maior com a transmissão do que a recepção dos dados (NOGUEIRA et al., 2004).

2.6.3 Processamento

A memória e o processador estão envolvidos nas atividades de computação realizada pelo *endpoint*. Quanto maior a frequência do processador, maior será o seu consumo de energia. Algumas das características dos processadores utilizados em nós sensores são: operar em baixa frequência e possuir um baixo custo com energia (NOGUEIRA et al., 2004).

2.7 Categoria de sensores

Existe uma diversidade grande sobre os modelos de sensores presentes atualmente, em sua grande maioria são modelos específicos criados para uma única finalidade. Pode-se apresentar duas categorias de classificação dos sensores essas são os tipos de sensores analógicos e digitais.

Os sensores analógicos tem como finalidade atribuir qualquer valor ao seu sinal de saída ao longo de um período, desde que o mesmo esteja presente em sua faixa de operações, valores esses que podem ser mensurados através de elementos com circuitos eletrônicos não digitais (WENDLING, 2010).

Os sensores digitais são dispositivos capazes de detectar diferentes variedades de impulsos e convertê-los em frequências binárias. Os sinais transmitidos representam quando unicamente um bit, ou sequências de bytes quando transmitidos em paralelo (WENDLING, 2010).

2.7.1 Sensores Ópticos

Os sensores ópticos são componentes eletrônicos capazes de detectar qualquer alteração que possa existir em determinado ambiente sem a interação mecânica com o mesmo. Seu princípio

de funcionamento é baseado entre um emissor e um receptor, onde os sinais de luz são transmitidos do emissor e devem alcançar o receptor, como destino (THOMAZINI; ALBUQUERQUE, 2011). A Figura 2.5 ilustra o princípio do funcionamento deste sensor.

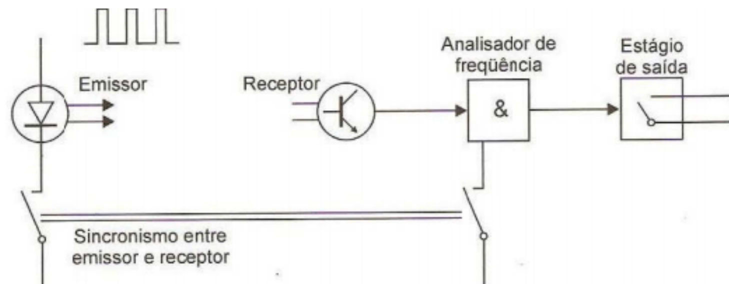

Figura 2.5: Funcionamento base sensor Ópticos.
Fonte: (THOMAZINI; ALBUQUERQUE, 2011).

2.7.2 Sensores ultrassônicos

Esta categoria de sensores tem por finalidade a detecção de objetos de acordo com sua distância, esses objetos devem ser capazes de refletir os sinais enviados. Seu funcionamento tem como base ondas ultrassônicas de curto alcance (THOMAZINI; ALBUQUERQUE, 2011). A Figura 2.6 ilustra o funcionamento básico desse modelo.

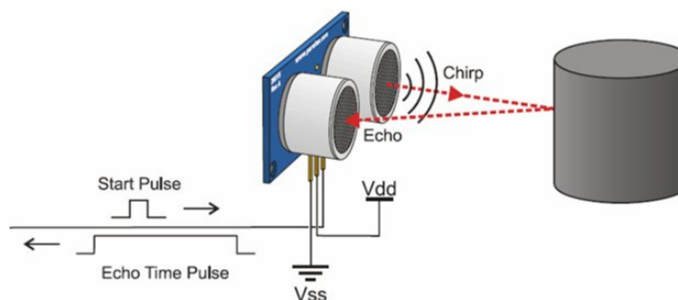

Figura 2.6: Detecção de Objetos sensor Ultrassônico.
Fonte: (THOMAZINI; ALBUQUERQUE, 2011).

O modelo apresentado é baseado no sensor hc-sr04. O ilustrado na Figura 2.6 possui o receptor e transmissor de frequências embutidos em um único sensor.

2.7.3 Sensores Térmicos

Os sensores térmicos como os demais, possuem uma variedade grande de modelos, esses realizam as medições mas utilizando diferentes meios. Serão apresentadas duas variações para esse modelo de sensor, os termistores e os de termorresistência.

Os termistores (*Thermally Sensitive Resistor*), são semicondutores eletrônicos que provocam oscilação de acordo com sua temperatura. Essa categoria se divide em duas variações básicas, os PTC (*Positive temperature coefficient*) resistores que elevam sua resistência de acordo com o aumento da temperatura e os NTC (*Negative temperature coefficient*) elementos cuja resistência tende a diminuir de acordo com a elevação da temperatura (THOMAZINI; ALBUQUERQUE, 2011).

A Figura 2.7 ilustra um exemplo da curva dos termistores e sua simbologia, ao lado direito da imagem encontra-se representado o coeficiente de temperatura positiva e ao lado esquerdo encontra-se representado o coeficiente de temperatura negativa.

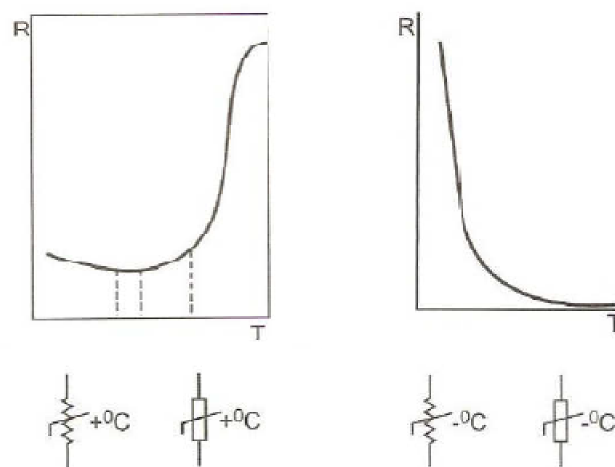

Figura 2.7: Exemplo de curva do termistor e sua Simbologia.
Fonte: (THOMAZINI; ALBUQUERQUE, 2011).

As termoresistências são sensores de temperatura que baseando-se em filamentos de metal onde sua resistência irá variar de acordo com a temperatura. Esses sensores são mais utilizados principalmente nas indústrias, onde sua variação de temperatura pode chegar a níveis muito baixos ou muito elevados, como por exemplo a termorresistência de platina modelo pt-100 onde sua faixa varia de -200 a 650°C (THOMAZINI; ALBUQUERQUE, 2011).

2.7.4 Sensores de umidade

Os sensores de umidade são os dispositivos que captam e quantificam o vapor de água presente em diversos ambientes, como o ar ou solo (THOMAZINI; ALBUQUERQUE, 2011).

A medição da umidade pode se basear em diferentes parâmetros para diferentes tipos de aplicações, a Tabela 2.2 apresenta alguns dos métodos utilizados para esse tipo de medição.

Tabela 2.2: Tabela simplificada dos métodos para medição de umidade

Parâmetro	Descrição	Medição	Aplicações
Umidade relativa (UR)	Temperatura mínima alcançada por um termômetro umedecido em um fluxo de ar.	ºC	Meteorologia
Ponto de Orvalho	Temperatura à qual o ar deve ser resfriado para alcançar a saturação.	ºC	Medições ambientais
Gramas por quilograma	Peso de água em gramas por quilograma de ar.	g /kg	Câmaras de vapor.

2.8 Tecnologias de comunicação

A consequência da tecnologia embarcada e seu uso em ambientes agrícolas possibilitou o desenvolvimento e crescimento da agricultura de precisão. Entretanto, em conjunto com o crescimento da produtividade, surge a carência de estabelecer um padrão de comunicação de forma transparente entre diversos equipamentos de diferentes fabricantes.

Com a utilização de dispositivos sem fio destacam-se três das suas principais vantagens:

- Segurança - Por se tratar de dispositivos sem fio, os mesmo podem ser instalados em ambientes de difícil acesso e com condições extremas, com isso permitindo aos controladores, ou sistemas ficarem analisando e tomando medidas preventivas tudo isso a uma distância segura.
- Comodidade - A utilização destes módulos sensores possibilita o monitoramento em tempo real de diversos cenários, possibilitando ao usuário final uma visão atual do seu ambiente.
- Custos - Redução do custo de materiais como fios de extensão, conduítes, dentre outros acessórios onerosos, em um ambiente pequeno pode não impactar tanto financeiramente, mas levando em consideração grandes empresas e ambientes como o meio agrário torna-se inviável a utilização de fiações.

Nessa seção serão apresentados os protocolos mais utilizados em comunicação sem fio de curto alcance, são esses Wi-Fi (IEEE 802.11), Bluetooth (IEEE 802.15.1), ZigBee (IEEE 802.15.4) e LoRa (IEEE 802.15.4g) protocolo para longas distâncias.

2.8.1 Wi-Fi (IEEE 802.11)

A tecnologia de comunicação Wi-fi (*Wireless Fidelity*) destina-se a comunicação entre dispositivos de curtas distâncias, sua maior utilização é com acesso de computadores ou dispositivos móveis a uma rede local. Também conhecida como WLAN (*Wireless Local Area Network*) (NASCIMENTO, 2007). A Tabela apresenta as classes e distâncias que esta tecnologia pode apresentar.

Tabela 2.3: Classificação Wi-Fi 802.11

Classes	**Classificação Wi-Fi 802.11**
IEEE 802.11b	Frequência 2,4 GHz com capacidade teórica de 11 Mbps
IEEE 802.11g	Frequência 2,4 GHz com capacidade teórica de 54 Mbps
IEEE 802.11n	Frequências 2,4 GHz e/ou 5 GHz com capacidade de 150 a 600 Mbps
IEEE 802.11ac	frequência 5 GHz com capacidade acima de 1 Gbps

Essa tecnologia pode ter sua distância de comunicação limitada de até 100 metros dependendo do ambiente.

2.8.2 Bluetooth (IEEE 802.15.1)

A tecnologia de comunicação Bluetooth é destinada a comunicações entre dispositivos de curto alcance, conhecida como WPAN (*Wireless Personal Area Network*) é utilizada para substituir as conexões via cabos. Seu alcance pode variar de acordo com sua classificação, essa sendo dividida em três níveis conforme apresentado na Tabela 2.4.

Tabela 2.4: Classificação Bluetooth

Classes	**Classificação Bluetooth**
Classe 1	Alcance Máximo 100 metros
Classe 2	Alcance Máximo 10 metros
Classe 3	Alcance Máximo 1 metros

Essa tecnologia opera na banda ISM de 2,4 GHz, na faixa de frequência de 2.400 MHz a 2.483,5 MHz (NASCIMENTO, 2007).

2.8.3 ZigBee (IEEE 802.15.4)

A tecnologia de comunicação ZigBee é baseada em radiofrequência, conhecida como LR-WPAN (*Low-Rate Wireless Personal Area Networks*) destina-se a aplicações embarcadas que exigem baixas taxas de dados e baixo consumo de energia para a comunicação (ZIGBEE, 2017). Essa tecnologia opera nas faixas de frequência que não precisam de licença ISM (*Industrial Sientific and Medical*) conforme apresentado na Tabela 2.5.

Tabela 2.5: Faixa de frequência protocolo ZigBee

Faixa	Localidade
2,4 GHz	Global
915 MHz	América do Norte
868 MHz	Europa
920 MHz	Japão

As taxas para transferência entre dados são de 250Kbs e podem ser alcançadas em 2.4GHz, de 40Kbs em 915MHz e de 20Kbs em 868MHz, sua distância de transmissão pode variar de acordo com a potência de seus rádios que variam de 50 a 200 metros em perímetros urbanos(ZIGBEE, 2017).

O padrão IEEE 802.15.4 define as duas camadas inferiores do protocolo de comunicação: a camada física e a subcamada MAC (*Medium Access Control*). Construído sobre esse alicerce, o ZigBee fornece a camada de rede e o quadro para a camada de aplicação (ZIGBEE, 2017). A Figura 2.8 ilustra a definição das camadas.

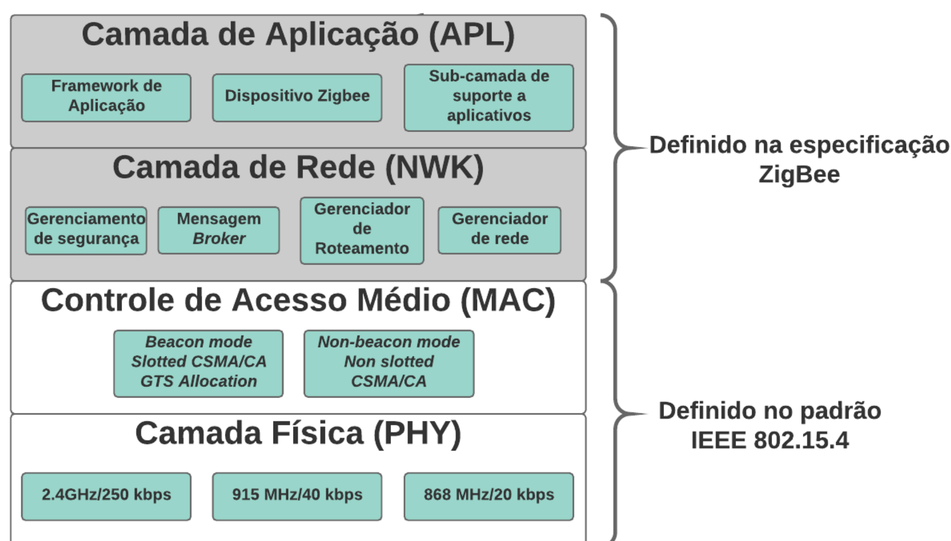

Figura 2.8: Arquitetura da pilha do protocolo IEEE 820.15.4/ZigBee
Fonte: Adaptado de (KOUBÂA; ALVES; TOVAR, 2006)

2.8.4 LoRa

Lora é a camada física de uma tecnologia de comunicação sem fio que pode atuar como parte de uma rede LPWAN, seu esquema proprietário de modulação por espalhamento espectral é derivado da modulação CSS (*Chirp Spread Spectrum*)(SEMTECH, 2015). LoRa é uma implementação da camada PHY (*Physical Layer*) e é agnóstico em relação às implementações

de camadas superiores.

A Tecnologia de comunicação LoraWAN, nome dado ao protocolo que define a arquitetura do sistema bem como os parâmetros de comunicação usando a tecnologia LoRa (*Long Range Radio*)(LAVRIC; POPA, 2017).

Seu funcionamento tem como base a rede de topologia estrela. Suas principais aplicações são sistemas de IoT, sobretudo aqueles operados a bateria, mensagens curtas de status que sejam fácil ou difícil acesso (LAVRIC; POPA, 2017).

No contexto com baixo uso de banda e longo alcance de cobertura é que se encontram as *Low Power Wide Area Networks* (LPWANs). Sua implementação varia de acordo com sua tecnologia dentre elas, LoRa (*Long Range Radio*), Sigfox e UNB (*Ultra Narrow Band*). Suas características principais são o longo alcance, baterias com alta durabilidade como fonte de energia para dispositivos e sensores de baixo custo. Aplicativos que se utilizam de LPWANs não possuem baixa latência de transmissão de dados como um de seus requisitos (LORA, 2017).

Como tecnologia destinada a aplicações IoT sua taxa de comunicação alcança valores entre 300 bps a 50 kbps. Seu consumo de energia é consideravelmente pequeno, permitindo dispositivos se manterem em conectado por um período maior de tempo.

A tecnologia LoRa® utiliza da frequência ISM sub-GHz, onde ondas eletromagnéticas transponham em grandes estruturas estas com distâncias de 3 até 12 km em meio urbano e 45 km no meio rural, estes valores variam de acordo com as interferências apresentadas em cada ambiente. Atualmente no Brasil é utilizado a frequência ISM de 915 MHz (LORA, 2017).

2.9 Protocolos de Transporte

Os Protocolos de transporte são uma combinação de informações, decisões, normas e regras entre processos com o objetivo de executar uma tarefa. Segundo Coulouris (COULOURIS, 2013) sua definição divide-se em duas partes: Formato dos dados transmitidos e Sequência da transmissão.

Com a transmissão dos dados entre os nós e sua utilização na aplicação para análise, propõem a utilização do protocolo HTTP e MQTT.

2.9.1 HTTP

O HTTP (*Hypertext Transfer Protocol*) é um protocolo do nível de camada de aplicação para sistemas de informação distribuídos, colaborativos e hipermídia[2].

O HTTP é do tipo de protocolo cliente-servidor, suas mensagens geralmente são trocadas em pares, tendo o cliente enviando uma requisição contendo operações codificadas à serem analisadas e processadas pelo servidor e um vetor de bytes contendo argumentos associados. A resposta do servidor contém os resultados obtidos (COULOURIS, 2013).

Os principais métodos suportados por esse protocolo são:

- GET: Solicita uma representação do recurso especificado, seus pedidos usando este método só devem apresentar a visualização dos dados.
- POST: Solicita que o servidor aceite a entidade fechada no pedido como um novo subordinado do recurso web identificado pelo URL.
- PUT: Atualiza um recurso em uma URL específica, caso não exista, o mesmo poderá criá-la.
- DELETE: Exclui o recurso especificado.

2.9.2 MQTT

O MQTT (*Message Queuing Telemetry Transport*) ISO/IEC PRF 20922, utiliza do paradigma *publish/subscribe* (Publicação/Assinatura), desenvolvido para transmissões simples e pequenas, tem como foco a utilização em dispositivos com capacidade limitada e em redes com restrições de banda e alta latência (MQTT, 2017), sua arquitetura pode ser definida em três níveis de qualidade de serviço (QoS) para transmissão dos dados (MQTT, 2017).

- O primeiro nível, considerado o mais inferior não disponibiliza de nenhuma garantia na entrega das mensagens;
- O segundo nível, o intermediário, garante a entrega da mensagem ao menos uma vez, podendo haver repetições dos mesmos;
- O terceiro, e mais alto nível garante suas transmissões, mas seu consumo de banda e sua latência sofrem aumentos.

[2]https://tools.ietf.org/html/rfc2616

Na Figura 2.9 é ilustrado o modelo de transmissão utilizado pelo paradigma *publish/subscribe*.

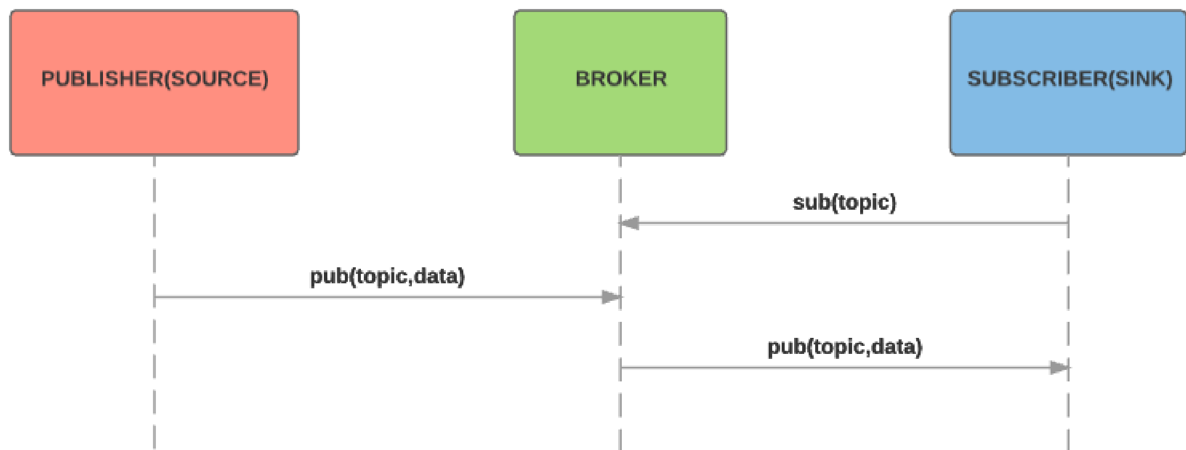

Figura 2.9: Exemplo comunicação Publish/Subscribe
Fonte: Elaborado pelo Autor.

No modelo de transmissão utilizado pelo MQTT o cliente (*subscriber*) envia uma mensagem ao servidor (*broker*) para assinar sua participação no tópico.

O cliente (*publisher*) transmite informações contendo os dados e a qual tópico ele pertence, o servidor recebe essas informações e as transmite à todos os clientes que tenham assinado aquele tópico (HUNKELER, 2008).

Quando um cliente estabelece uma conexão MQTT com o servidor, ele pode definir uma *flag* denominada *clean session*, quando esta *flag* está acionada, todas as assinaturas do cliente ao serem removidas ao se desconectar do servidor, ao definir essa *flag* como false a conexão é tratada como durável, e as assinaturas do cliente permanecerão em vigor após qualquer desconexão. Nesse caso, as mensagens subsequentes que chegam carregando uma designação QoS alta, são armazenadas para a entrega depois que a conexão for restabelecida (MQTT, 2017).

2.9.3 Comparativo HTTP e MQTT

Atualmente o HTTP é o protocolo de comunicação mais utilizado, e serve como referência para comparativo entre demais protocolos como por exemplo o MQTT.

Padrão de mensagens:

- MQTT - Padrão *publish/subscribe* com baixo acoplamento. Os clientes não necessitam ter o conhecimento da rede, apenas indicar ao servidor, o conteúdo a ser entregue ou recebido;

- HTTP - Padrão *request/response*. Neste padrão torna-se necessário o conhecimento do endereço dos dispositivos a se conectar.

Orientação de design:

- MQTT - Concentrada em dados. Estes são transferidos como matrizes de *bytes*;
- HTTP - Concentrada em documentos. Suporte ao padrão MIME (*Multipurpose Internet Mail Extensions*)para definição de conteúdos.

Complexidade de implementação:

- MQTT - Complexidade simples, possuindo comandos simples como : CONNECT, PUBLISH, SUBSCRIBE, UNSUBSCRIBE e DISCONNECT são importantes para os desenvolvedores;
- HTTP - Complexidade mais alta, ele possui um número maior de métodos e seus retornos precisam ser tratados pelos desenvolvedores.

Comprimento das mensagens:

- MQTT - Cabeçalhos curtos e de menores tamanho, um pacote pode ter tamanho de 2 bytes;
- HTTP - Cabeçalhos e mensagens mais extensas.

2.10 Conclusão do Capítulo

Neste capítulo foi apresentado uma fundamentação teórica que faz parte da base da proposta de trabalho da dissertação. Nele são apresentadas as características gerais da agricultura de precisão, meio onde será aplicado este trabalho, as tecnologias que englobam esse conceito e o ambiente IoT e suas camadas, essas para entender e oferecer uma solução que entre no contexto do *Fog Computing* atendendo aos requisitos locais do ambiente sem a necessidade da internet. Neste ainda são apresentados os conceitos dos sensores/*endpoints* que constituem uma das camada fundamentais no desenvolvimento deste projeto. No capítulo seguinte são apresentados alguns dos trabalhos e referências para a elaboração deste projeto.

Capítulo 3

TRABALHOS RELACIONADOS

3.1 Coleta de dados para ampla massa distribuída de sensores

O projeto apresentado por Hideg, Blázovics, Csorba e Gotzy (2016) propõem um método teórico de utilização de nós móveis para implantação e gerenciamento dos nós estáticos que realizam as medições. Os autores utilizam do framework sensorHUB, o qual é responsável por toda a complexidade em extrair e analisar as informações coletadas pelos nós e disponibilizá-las. A Figura 3.1 ilustra a arquitetura proposta com o *framework* sensorHUB descrita.

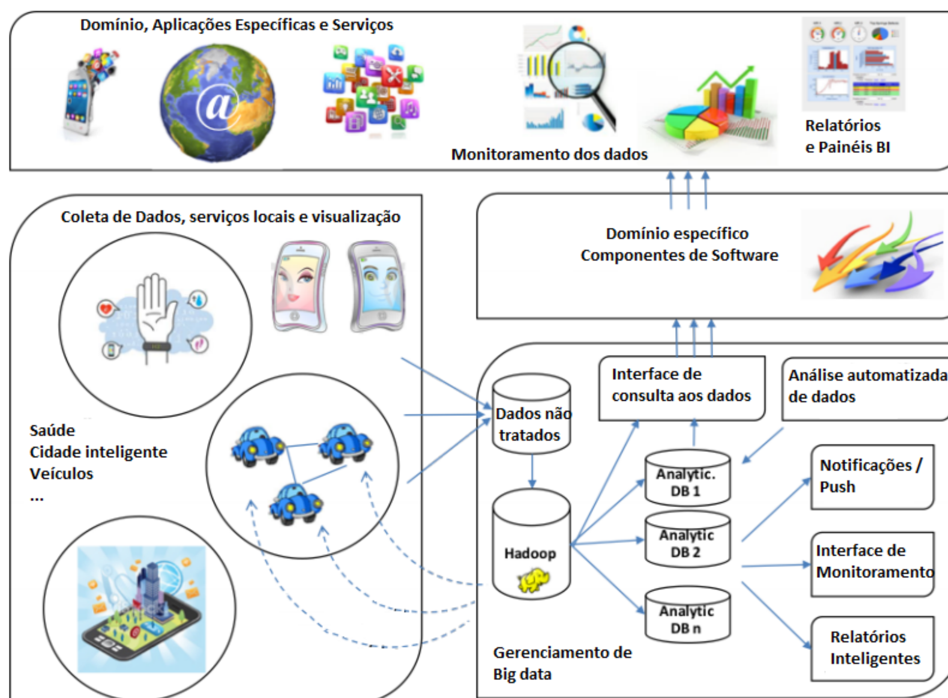

Figura 3.1: Arquitetura do conceito do sensorHUB.
Fonte: (HIDEG et al., 2016).

Segundo os autores a proposta apresentada tem como foco minimizar o consumo dos sensores com a utilização de hibernação dos mesmo, acordando apenas em dois períodos para realização da medição e para transmissão dos dados coletados ao nó móvel, que transmitirá um sinal assim que houver aproximação.

O *framework* sensorhub é concebido como um método e uma estrutura de apoio ao desenvolvimento de aplicativos e serviços relacionados com IoT, com seu princípio *open-source* e baseado em padrões abertos para todas as trocas de dados e fornecendo recursos de processamento avançados.

Sua implementação em JAVA permite uma grande variedade de conexões com diferentes tipos de sensores, por meio de uma API (*Application Programming Interface*) simples, mas genérica. Utilizando de padrões abertos como OGC - (*Open Geospatial Consortium*) e SWE - (*Sensor Web Enablement*), fundamentais para projetar e escalar redes de sensores (SENSORHUB, 2017). Os sensores podem ser conectados de diversas conexões de *hardware* disponíveis atualmente estas conforme apresentado na Tabela 3.1:

Tabela 3.1: Dispositivos de comunicação sensorHub

Dispositivos de Comunicação
Serial / RS232
USB Serial
Bluetooth Serial
Bluetooth LE + GATT
Wireless

Entretanto tecnologias de longo alcance como Zigbee e LoRaWAN ainda não estão disponíveis para sua utilização (SENSORHUB, 2017), contudo apresentou-se uma ótima opção para ambientes conectados a rede e que não dependam de otimização ou análise para exclusão de dados atípicos o que pode ocasionar um alto consumo e armazenamento de dados.

3.2 Uma estrutura de análise de Big Data para aplicativos IoT na nuvem

O projeto apresentado pelo autor Pham (2015) foi o desenvolvimento de um *framework* denominado BDAaaS, como base para BDA (*Big Data Analytics*), estrutura essa projetada para facilitar o provisionamento das soluções BDA em tempo real, mas também para a coleta e análise dos dados dos *gateways* do IoT para a *Cloud*. Seu sistema foi criado no conceito da arquitetura lambda, conceito este para que os dados sejam processados e entregues dentro do

tempo e modo esperados, a arquitetura Lambda segue dividida em três camadas: *batch layer, speed layer e serving layer* (camada de lote, camada de velocidade e camada de resposta).

- Processamento em Lote - Minimização dos erros, aumento na precisão dos dados.
- Velocidade - Processamento em tempo real
- Operações de respostas - Respostas às consultas realizadas ao sistema.

A arquitetura implementada pelos autores segue detalhada conforme a Figura 3.2:

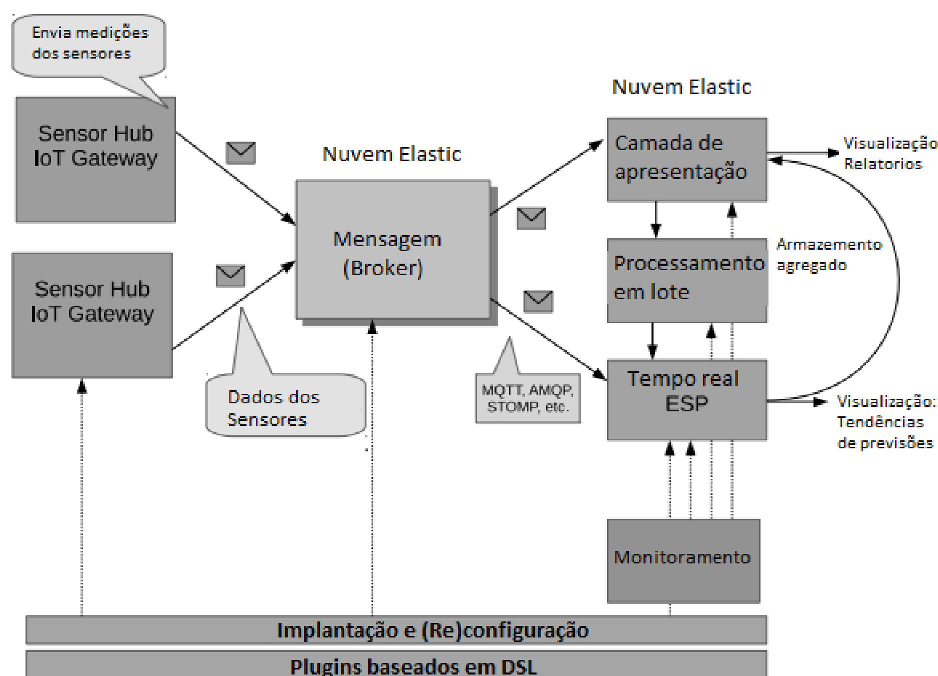

Figura 3.2: Estrutura BDAaaS implementada na arquitetura lambda
Fonte: (PHAM, 2015).

Na Figura 3.2 destacam-se alguns dos principais componentes em uma arquitetura IoT esses sendo:

- Gateway - Camada de comunicação entre os dispositivos e a aplicação.
- Protocolo de Transmissão - Message Broker ou Message-as-a-Service, componente que implementa o padrão de transmissão *publish/subscribe*.
- Armazenamento NoSQL - Ambiente de armazenamento dos dados.
- Processador em lote - Componente para operações *offline* do processamento dos dados armazenados.

- Processador de fluxo de eventos em tempo real - Componente para análise dos dados em tempo constante assim que são recebidos pelo *gateway*.

Um grande diferencial nesta arquitetura apresentada, é a integração do protocolo de transporte MQTT com o *broker* a sua arquitetura, aceitando tecnologias de radiofrequência, entretanto transmissões em lote para dispositivos com baixo recurso computacional pode agravar em uma grande desvantagem, aumentando assim o consumo de banda, energia e ou armazenamento temporário de informações.

3.3 EcoCIT: Uma Plataforma Escalável para Desenvolvimento de Aplicações de IoT

O EcoCIT plataforma *middleware* desenvolvida a fim de suprir as necessidades de sua base a plataforma EcoDIF, tem como foco principal a escalabilidade de seus recursos.

Sua base o EcoDIF faz uso de um único servidor para processamento de requisições, como consequência, incrementar a capacidade computacional implica em transferí-la para uma máquina mais robusta ou adicionar manualmente novos recursos.

Abordagem essa suprida pelo desenvolvimento da plataforma EcoCIT, utilizando o conceito de máquinas virtuais, o ambiente foi projetado para alocação de novos ambientes conforme a demanda de consumo. A Figura 3.3 ilustra a arquitetura e os componentes utilizados na plataforma.

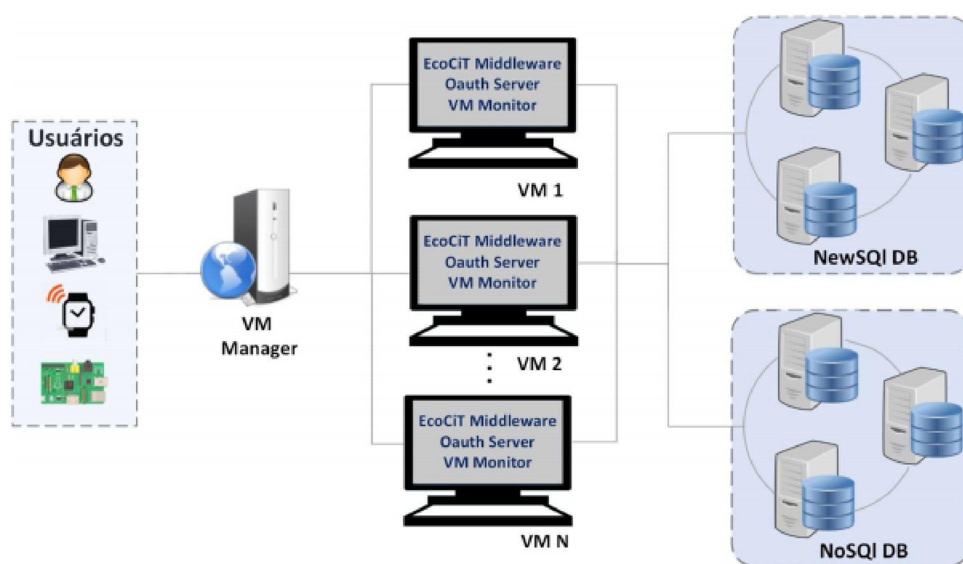

Figura 3.3: Arquitetura do EcoCIT
Fonte: (SILVA, 2017)

Seus principais componentes apresentado no trabalho são:

- *Virtual Machines* (VMs): Responsáveis por alocar toda a configuração da plataforma EcoCIT *Middleware*. Cada VM aloja uma instância de cada um desses componentes;
- VM *Manager*: Responsável por gerenciar a criação e remoção das VMs conforme sua necessidade, nesse serviço é utilizado um balanceador de carga para realizar a distribuição das cargas.
- *NewSQL DB*: Banco de dados relacional responsável por armazenar os dados da estruturados do EcoCIT;
- *NoSQL DB*: Banco de dados não relacional responsável por armazenar os dados encaminhados para o *middleware*.

A plataforma se mostrou escalável conforme a necessidade, entretanto a aplicação tem um foco voltado a escalabilidade de máquinas virtuais, essas que podem ser facilmente manuseadas e adicionadas em um ambiente de *cloud computing*.

3.4 Conclusão do Capítulo

Neste capítulo são apresentados os artigos que refletem o estado da arte e subsidiaram os estudos da proposta. Os trabalhos apresentados propõem arquitetura em ambientes controlados, todos os ambientes são apresentados em nuvem com isso remove-se a complexidade de implantação e manutenção de certos componentes. Ambos os trabalhos apresentam tecnologias semelhantes a serem trabalhadas, essas são tecnologias de curtas distâncias e processamento em nuvem, assim o escalonamento de recursos computacionais impacta diretamente no desempenho das aplicações.

Ambos os projetos apresentados não focaram seus trabalhos para o tratamento da informação e sim em sua apresentação, a aplicação de técnicas de fusão de dados poderia minimizar erros e diminuir o tráfego de informações entre os dispositivos e seu armazenamento.

No capítulo seguinte apresentam-se as tecnologias que fazem parte da arquitetura implementada para esta dissertação.

Capítulo 4

COMPONENTES DA ARQUITETURA

4.1 Conceito da Arquitetura

A arquitetura elaborada para este trabalho se baseia no conceito de microserviços, onde o desenvolvimento do sistema é constituído como um conjunto de pequenos serviços/aplicações, sendo estes trabalho implementados por ferramentas e tecnologias que visam a simplificação e aumento na produtividade, de forma que esta arquitetura possa atender as necessidades e sendo em destaque escalável, estável e confiante para seus usuários. A Figura 4.1 ilustra os microserviços desenvolvidos por este projeto.

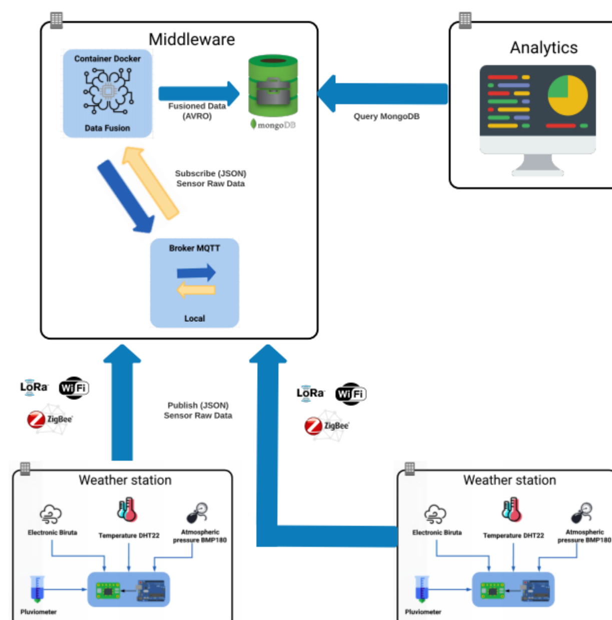

Figura 4.1: Arquitetura IoTUfscar
Fonte: Elaborado pelo Autor.

4.1.1 Arquitetura de Microserviços

De acordo com o autor Newman (2015) com as atuais necessidades de entregas rápidas e contínuas das aplicações, os mecanismos para automatização da infraestrutura e testes tiveram de ser novamente repensados para os conceitos de design de *software*. Quando um sistema possui uma arquitetura engessada, não possibilita melhorias rápidas, então existe limites em suas entregas. As Arquiteturas com granularidade mais fina tende a demonstrar melhor escalabilidade e autonomia às equipes de desenvolvimento, com isso é possível a implantação de novas tecnologias (NEWMAN, 2015).

Outro ponto de vista analisado foi do autor Fowler (2014), a arquitetura de microserviços tem como definição o desenvolvimento de uma aplicação a qual componha de pequenos serviços autossuficiente, onde cada serviço utiliza-se de comunicações HTTP através de APIs. Os serviços que constituem essa arquitetura visam a capacidade do negócio, e seu *deploy* é realizado de maneira automática e independente. O gerenciamento central destes serviços é mínimo, o que torna facilmente as migrações de tecnologias e linguagens utilizados nos projetos (FOWLER, 2014).

O conceito de microserviços pode ser considerado uma evolução, refinando e simplificando o conceito conhecido como SOA (*Service Oriented Architecture*) (AMARAL, 2015).

Suas tarefas são únicas e exclusivas, o que torna cada serviço limitado apenas ao escopo do negócio a qual foi implementado. Com foco centralizado, torna-se fácil a análise do crescimentos desnecessário em suas implementações (NEWMAN, 2015). Uma grande vantagem neste tipo de arquitetura é a diversidade tecnológica presente, por se tratar de uma arquitetura descentralizada, aplicações e ferramentas de diferentes tipos podem se comunicar e colaborar para diversas abordagens (AMARAL, 2015).

4.1.1.1 Resiliência

Arquitetura baseadas em microserviços possuem uma maior resiliência, o padrão de *bulkhead* é um conceito chave para este. Se algum componente presente na arquitetura apresentar falhas ou interrupções este problema não se propaga e é isolado dos demais, com isso liberando a arquitetura para voltar ao seu funcionamento. Aplicações com conceito monolítico não possuem este tipo de vantagem, sendo possível apenas a utilização de diversas máquinas para tentar reduzir a chance de falhas (NEWMAN, 2015).

4.1.1.2 Escalabilidade

O quesito de escalabilidade é outra vantagem na utilização de microserviços. Na utilização de aplicações monolíticas quando existe esta necessidade, todo o conjunto tem de ser escalado.

Com a utilização de serviços isolados, torna-se possível escalar apenas os serviços que realmente necessitem, com isso evitando gargalos nas aplicações e reduções em custos (NAMIOT; SNEPS-SNEPPE, 2014).

4.1.2 Arquiteturas monolíticas

Ao contrário das arquiteturas baseadas em microserviços, as arquiteturas monolíticas são aplicações construídas com apenas um núcleo central, todos os seus métodos e funções estão alocados em uma única aplicação.

De acordo com Fowler (2014), para uma melhor compreender da arquitetura baseada em microserviços, é aconselhável entender e comparar a arquitetura monolítica tradicional. As alterações em aplicações baseadas na arquitetura monolítica requerem a compilação de uma nova versão da aplicação, abordagem essa tradicional na construção de sistemas.

Escalar o monólito significa escalar toda aplicação como uma só, ao invés de escalar apenas as partes que necessitam de mais recursos (FOWLER, 2014).

4.1.3 Virtualização

A virtualização é um conceito que permite a administração de ambientes de computacionais de forma lucrativa, utilizando recursos que geralmente estão ajustados a equipamentos específicos. Com a virtualização, você pode utilizar o limite total de uma máquina física, distribuindo os recursos entre usuários ou ambientes (HUNG et al., 2016).

A Figura 4.2 ilustra as camadas utilizadas nos conceitos abordados, a virtualização por máquinas virtuais e a utilização de contêineres.

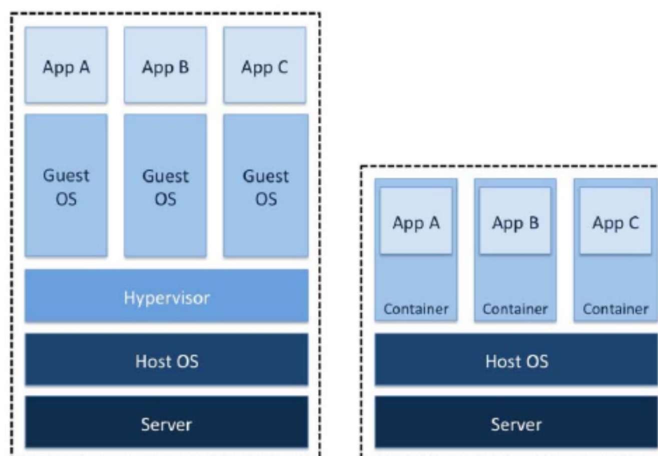

Figura 4.2: Arquitetura Máquinas virtuais / Contêineres.
Fonte: Imagem adaptada (KOVÁCS, 2017)

4.1.3.1 Máquina Virtual

O conceito das VMs (*Virtual machines*) é um sistema operacional ou ambiente de aplicativo que imita a utilização de um *hardware* dedicado. De acordo Popek e Golbderg (1974) uma máquina virtual é considerada uma duplicata eficiente e isolada da máquina real, desde aquele tempo houveram mudanças e grandes avanços tecnológicos mas seu conceito ainda permanece (POPEK; GOLBDERG, 1974).

Uma VM possui como principal elemento um orquestrador, denominado *hipervisor*, o qual é responsável por emular os componentes do meio físico do servidor, permitindo que as máquinas virtuais compartilhem seus recursos. O *hipervisor* pode emular diversas plataformas de *hardware* isoladas umas das outras, permitindo que máquinas virtuais executem sistemas operacionais diferentes no mesmo ambiente (NETO, 2015).

Com a utilização das virtualização limita-se os custos reduzindo a necessidade de sistemas de *hardware* físico. As máquinas virtuais usam com mais eficiência o hardware, o que reduz a quantidade de *hardware* e os custos de manutenção associados, reduzindo a demanda de energia e resfriamento (NETO, 2015).

4.1.3.2 Contêiner

O conceito dos contêiner teve seu surgimento com o LXC (linux *containers*), onde esta técnica utiliza *namespaces* do *kernel* para o isolamento dos recursos. Abordagem essa que limita a utilização de recursos computacionais, entretanto compartilham o *kernel* com o sistema operacional, para que seus sistemas de arquivos e processos em execução sejam visíveis e ge-

renciáveis a partir do sistema operacional (IVANOV, 2017). Esses processos são executados a partir de uma imagem distinta que fornece todos os arquivos necessários. Por fornecer uma imagem que contém todas as dependências de um aplicativo, o contêiner é leve e portátil.

Os contêineres representam uma abstração na camada de aplicativo que agrupa códigos e dependências(KOVÁCS, 2017). Vários contêineres podem ser executados em uma mesma máquina e compartilhar o kernel do sistema operacional com outros contêineres, cada um sendo executado como processos isolados no espaço do usuário(DOCKER, 2017). Uma das grandes vantagens na utilização dos contêineres são sua velocidade de resposta, sua inicialização e remoção, além disso seu tamanho comparado com as máquinas virtuais é insignificante são *Megabytes* MBs contra *Gigabytes* GBs de uma máquinas virtual.

Atualmente, a implantação de aplicativos dentro de algum tipo de contêiner linux é uma prática amplamente adotada, isso ocorre devido à evolução da tecnologia e à facilidade de uso que ela apresenta. Mesmo que os contêineres, ou a virtualização em nível de sistema operacional, sejam tecnologias e conceitos antigos, levou-se algum tempo para sua evolução. Uma das razões para isso é o fato de que as tecnologias baseadas em hipervisor, como KVM (*Kernel Virtual Machine*) e *Xen Hypervisor*, foram capazes de resolver a maioria das limitações do kernel do Linux durante esse período e a sobrecarga apresentada não foi considerada um problema (IVANOV, 2017).

4.2 Elementos da arquitetura

Conforme apresentado a arquitetura desenvolvida se baseia nos conceitos de microserviços com isso é possível ressaltar suas principais ferramentas e tecnologias, sendo cada uma separada por suas respectivas funcionalidades.

- Sistema embarcado - Estação meteorológica
- Contêiner JAVA - Aplicação de Fusionamento dos dados coletados.
- Kaa Iot - Plataforma de *middleware* para desenvolvimentos IoT.
- Zeppelin - Análise gráfica dos dados coletados.
- Zabbix - Monitoramento das aplicações e *hardware*.

4.2.1 Sistema embarcado

Um sistema embarcado é um sistema microprocessado no qual o computador é completamente encapsulado ou dedicado ao dispositivo ou sistema que ele controla. Diferentemente de computadores de propósito geral, como o computador pessoal, um sistema embarcado realiza um conjunto de tarefas predefinidas, geralmente com requisitos específicos. Já que o sistema é dedicado a tarefas específicas, através de engenharia pode-se otimizar o projeto reduzindo tamanho, recursos computacionais e custo do produto.

4.2.1.1 Raspberry PI 3 B+

Raspberry PI é um pequeno computador potente baseado em microcontrolador ARM, ele funciona em distribuições Linux com o sistema operacional Raspbian distribuição Debian. Uma placa Raspberry PI suporta cartão de memória SD, teclado e mouse USB, monitor HDMI e fonte de alimentação. É possível fazer o Raspberry Pi funcionar como um computador normal de propósito geral (PRINCY; NIGEL, 2015). A Figura 4.3 ilustra este pequeno componente.

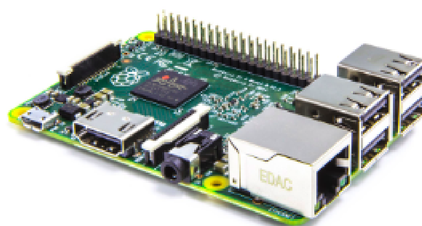

Figura 4.3: Raspberry
Fonte: Elaborado pelo Autor

Apesar do tamanho reduzido, medindo aproximadamente 9 x 6 x 2 cm ele vem equipado com 1GB de memória RAM, processador ARMv8 quadcore de 1.4 Ghz, armazenamento de 32GB (cartão micro SD) podendo ser expandido, suas expansões se dão através das quatro portas USB, uma porta Ethernet e uma HDMI (ANTONI; VIVIAN; PREUSS, 2015).

4.2.1.2 Arduino nano

O Arduino nano é uma pequena placa projetada usando uma variedade de microprocessadores e controladores. Essas placas são equipadas com um conjunto de pinos de entrada/saída (E/S) digitais e analógicos que podem ser conectados a uma variedade de placas de expansão ou *breadboards* (blindagens) e outros circuitos(NAYYAR; PURI, 2016). Essas placas têm uma interface de comunicação serial, incluindo o *Universal Serial Bus* (USB) em alguns modelos.

Microcontroladores normalmente usam a linguagem de programação C / C ++. Além de usar o *toolchain* de construção tradicional, o projeto Arduino também fornece um ambiente de desenvolvimento integrado (IDE) baseado no projeto de linguagem *Processing*.

4.2.1.3 ESP32

O ESP32 é uma placa de desenvolvimento de baixo custo que consolida GPIOs, I2C, UART, ADC, PWM e WiFi (KODALI; SORATKAL, 2016) para prototipagem rápida. Alimentado pela fonte 3.3V, o ESP32 junto com o regulador de tensão e USB para serial é empacotado como módulo ESP-32. Como o arduino suas aplicações podem ser desenvolvidas no Arduino IDE ou do ESPlorer baseado em Lua, esse equipamento com tecnologia LoRa^{TM}, frequência 433 mhz, sobre-148 dBm alta sensibilidade, + 20 dBm de potência de saída.

A Figura 4.6 ilustra as duas placas utilizadas no desenvolvimento do projeto.

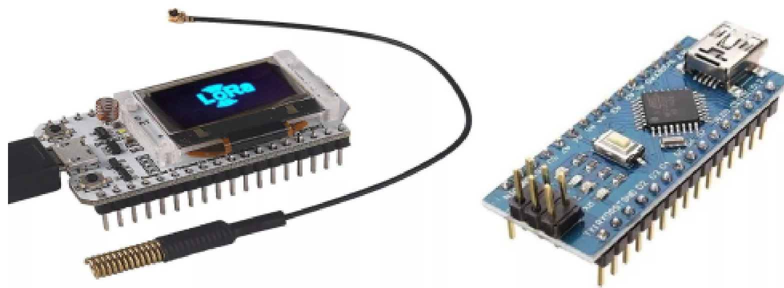

Figura 4.4: ESP32 Lora / Arduino nano
Fonte: Elaborado pelo Autor

4.2.1.4 Sensores

- Anemômetro - Sensor de monitoramento meteorológico usado para medir a velocidade do vento;
- Biruta Eletrônico - Sensor com a capacidade de medir e informar a direção e a velocidade do vento;
- Pluviômetro - Sensor usado para recolher e medir, em milímetros lineares, a quantidade de líquidos ou sólidos precipitados durante um determinado tempo e local;
- DHT 22 - Sensor constituído de um termistor e um sensor capacitivo para medir a temperatura e a umidade do ar ambiente.

Figura 4.5: Sensor de Temperatura e umidade DHT 22
Fonte: Elaborado pelo Autor

- BMP180 - Sensor de monitoramento meteorológico utilizado para medir a pressão atmosférica.

Figura 4.6: Sensor de Pressão Atmosférica BMP180
Fonte: Elaborado pelo Autor

A Figura 4.7 apresenta os sensores anemômetro, biruta eletrônico e pluviômetro fixados no mastro da estação meteorológica desenvolvida.

Figura 4.7: Sensores anemômetro, biruta eletrônico e pluviômetro
Fonte: Elaborado pelo Autor

Um protótipo de uma estacão meteorológica foi construído, utilizando estes sensores, com o proposito de medir e validar os conceitos e resultados da arquitetura proposta este sendo ilustrado na Figura 4.7, item de fundamental importância no ambiente agrário, informações

essas que possibilitam auxiliar na análise dos dados para planejamentos como plantio, colheita e pulverização.

4.2.1.5 Banco de dados

Para realização do armazenamento temporários dos dados coletados pela estação meteorológica foi selecionado o SQLite um mecanismo de banco de dados relacional leve e eficiente para a persistência de dados isolados ao servidor. O SQLite permite a inserção e manipulação de dados com instruções SQL (SQLITE, 2018).

De forma prática e objetiva, o SQLite funciona como um pequeno SGBD capaz de criar, inserir e buscar um arquivo no disco do pequenos servidores ou dispositivos móveis, este arquivo torna-se capaz de armazenar uma grande quantidade de tabelas.

A manipulação de tabelas no SQLite são realizadas através dos comandos padrões da linguagem SQL *CREATE TABLE* e *DROP TABLET*. Os dados das tabelas são manipulados através dos conhecidos comandos DML (*INSERT, UPDATE e DELETE*) e são consultados através do uso do comando *SELECT* (SQLITE, 2018).

4.2.2 Container Docker JAVA

Conforme apresentado na sessão 4.1.3.2 foram utilizados contêineres docker para realizar o procedimento de fusão e armazenamento das informações. O contêiner utilizado para esse procedimento corresponde a versão openjdk 8[3]. Para sua execução o algoritmo de fusão juntamente com o SDK do Kaa gerado foram exportados em um JAR executável o qual é adicionado ao contêiner perante a criação de um *Dockerfile* (Arquivo de instruções e manipulação de contêineres docker). A Figura 4.8 ilustra os comandos utilizados na criação do contêiner.

```
1 FROM openjdk:8
2
3 MAINTAINER jpmorijo@gmail.com
4 WORKDIR /tmp
5
6 COPY FusionWeatherStation.jar ./FusionWeatherStation.jar
7 CMD ["java", "-jar", "/tmp/FusionWeatherStation.jar"]
```

Figura 4.8: Comando para alocação de novos Contêineres
Fonte: *Dockerfile* imagem base

A definição de cada comando executado pelo *Dockerfile* Segue abaixo.

[3]https://github.com/docker-library/openjdk/blob/9a0822673dffd3e5ba66f18a8547aa60faed6d08/8-jdk/alpine/Dockerfile

- *FROM*: Informa a imagem base para a geração do novo contêiner, essa se torna a base para a imagem que será criada.
- *MAINTAINER*: Campo opcional, que informa o nome do mantenedor da nova imagem;
- *WORKDIR*: Define qual será o diretório de trabalho ao instanciar o contêiner;
- *COPY*: Realiza a copia de arquivos ou diretórios locais para dentro da imagem.
- *CMD*: Define quais comandos devem ser executados ao iniciar o contêiner;

4.2.3 Kaa Iot

O Kaa Iot é uma plataforma de *middleware* de múltiplos propósitos para a Internet das Coisas, que permite a criação de soluções IoT completas ponta-a-ponta, aplicativos conectados e produtos inteligentes. A plataforma Kaa oferece um kit de ferramentas aberto e rico em recursos para o desenvolvimento de produtos da IoT e, portanto, reduz drasticamente os custos, riscos e tempo de colocação no mercado associados (KAAIOT, 2018).

Há várias especificações arquitetônicas que tornam o desenvolvimento da IoT com a Kaa tão rápido e fácil. Primeiro, o Kaa é independente de *hardware* e, portanto, compatível com praticamente qualquer tipo de dispositivos conectados, sensores e *gateways*. Sua estrutura fornece uma gama de recursos e extensões de IoT para diferentes tipos de aplicativos IoT. Esses podem ser usados quase como módulos *plug-and-play* com o mínimo de código adicional por parte do desenvolvedor. Combinados com opções ilimitadas para protocolos de conectividade e integração com analítica, esses recursos tornam a Kaa uma metáfora apta para o desenvolvimento criativo de IoT (KAAIOT, 2018). Suas principais características estão descritas como:

- Abstração da camada de transporte - A arquitetura do Kaa possibilita a abstração dos mecanismos de transporte atualmente utilizados para comunicação entre os dispositivos. Seus *endpoints* por sua vez, podem fazer uso de diversos protocolos, sendo os mais conhecidos HTTP, MQTT, CoAP, TCP, etc. Com isso aplicações desenvolvidas não precisam necessariamente conhecer os protocolos utilizados para se comunicar com a plataforma.
- Comunicação baseada em *publish-subscribe* - Utilizando do conceito publicar/assinar todos os registros podem ser inseridos e atualizados e seus utilizadores serão notificados com as modificações.

- Gestão dos logs - A plataforma permite o armazenamento dos temporário dos registro em sua infraestrutura. Seu armazenamento pode ser utilizado para acompanhar o estado das aplicações, analisar anomalias, etc.

Para facilitar o desenvolvimento das aplicações a plataforma Kaa oferece SDKs para as aplicações com isso fornecendo tornando mais acessível a comunicação com a plataforma, encriptação de dados, *marshalling* de dados, autenticação, etc (KAAIOT, 2018). A plataforma Kaa pode ser representada por basicamente por esses elementos:

- Kaa *Server* - Instância responsável pelo processamento e operações realizadas pela plataforma.
- Kaa *Cluster* - Conjunto de instâncias Kaa *server* o qual trabalham paralelamente e realizam o balanceamento da carga aos processos realizados.
- *Endpoint* - São aplicações que fazem uso dos SDKs originalizados do Kaa *server* a fim de coletar e transmitir dados ao servidor Kaa.
- Aplicações - Camada de apresentação das informações coletadas e processadas pelo Kaa.

A arquitetura do Kaa e ilustrada na Figura 4.9, onde o Kaa armazena os dados de configurações em banco de dados SQL e seus registros e eventos em bases NoSQL e seus *nodes* podem se comunicar com os devidos *EndPoints*.

Kaa trabalha de forma com alta disponibilidade e escalabilidade. Seu SDK escolhe as instâncias do serviço *Bootstrap* e das operações quase aleatórias durante o início da sessão. Entretanto, se o *cluster* estiver sobrecarregado, a distribuição aleatória dos *EndPoints* não é eficiente. Além disso, quando um novo node e associado ao *cluster*, é necessário a redistribuição carga na topologia utilizada para obter um melhor desempenho.

O servidor Kaa utiliza o modo balanceador de carga para lidar com o fluxo dos terminais para conectar a um benefício de atividades alternativas, ao longo destas linhas, distribuindo a carga entre os nodes. O cálculo utilizado considera as informações do balanceador do servidor distribuídas pelo nos Kaa como uma informação e ocasionalmente recalcula as estimativas de carga de cada node.

4.2.4 Zeppelin

O Apache Zeppelin é uma ferramenta baseada na Web que pode criar documentos interativos e orientados a dados. *Framework* de código aberto que permite a adoção de diferentes

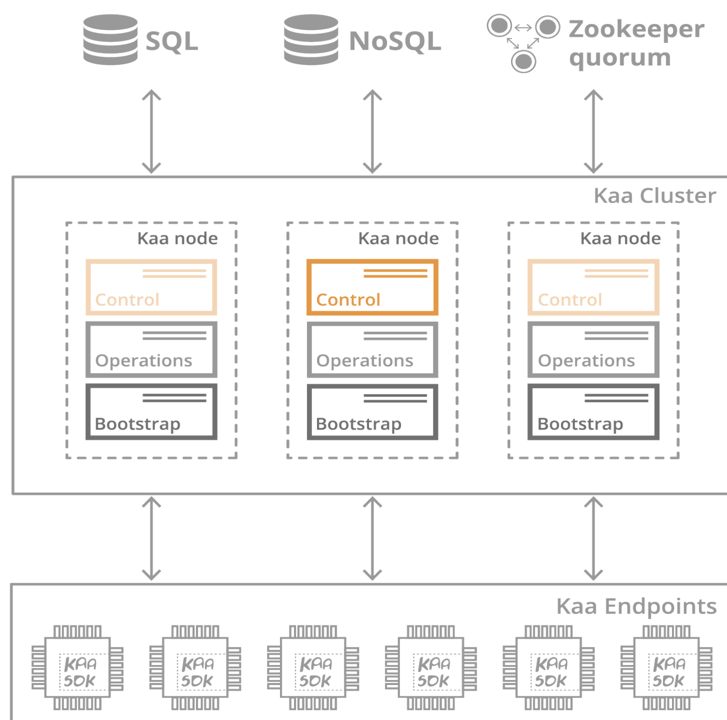

Figura 4.9: Arquitetura Kaa
Fonte: (KAAIOT, 2018).

intérpretes, como Apache Spark, Python, JDBC, Cassandra, MongoDB etc (ZEPPELIN, 2018).

Em sua utilização foi necessário adicionar o intérprete de conexão MongoDB, por padrão não vem integrado na ferramenta sendo necessário realizar um /textitfork[1] do interpretador presente no github. A Tabela 4.1 apresenta a configuração dos valores das propriedades do mongoDB Intérprete.

Tabela 4.1: Propriedades de configuração do interpretador MongoDB

Propriedade	Valor
mongo.server.database	kaa
mongo.server.host	127.0.0.1
mongo.server.port	27017
mongo.server.username	iotufscar
mongo.server.password	********
mongo.shell.command.table.limit	1000
mongo.shell.command.timeout	60000
mongo.shell.path	mongo

No Zeppelin, qualquer outro processo de *backend* pode ser implementado com suporte

[1]https://github.com/bbonnin/zeppelin-mongodb-interpreter

uma vez adicionado o interpretador apropriado. As consultas suportadas pelo interpretador mongoDB são executadas precedendo da tag% mongo.

Uma coisa importante a ser incluída na ferramenta é que ele possui algumas percepções que podem ser criadas sem esforço por meio de consultas SQL e compilação dos gráficos. Esse tipo de percepção é importante para mudanças e adição de novos sensores para ampliar a arquitetura.

4.2.5 Zabbix

O monitoramento da infraestrutura e aplicações vem se tornando vital para gestão da tecnologia da informação. Este monitoramento permite obter as informações necessárias sobre a infraestrutura e aplicações com isso facilitando as tomadas de decisões para suas melhorias.

Atualmente existem várias ferramentas que monitoram e auxiliam essas tomadas de decisões, entre elas destaca-se o Zabbix uma ferramenta de monitoramento de redes, servidores e serviços, ele possibilita a utilização de *templates*, para monitoramento de diversos parâmetros, sendo por estes disponibilizando relatórios bem detalhados e gráficos para melhor visualização (ZABBIX, 2018). O mesmo possui mecanismos de notificações, envios de e-mails e ou integrações com canais de comunicação como Slack, Hipchat, Telegram entre outros para alertas em relação aos eventos ocorridos ao servidores ou aplicações.

4.3 Conclusão do Capítulo

Neste capítulo foi apresentada uma visão dos serviços implementados e utilizados no desenvolvimento desta arquitetura. Nele descreve-se as tecnologias que abrangem desde a coleta das informações pelos *endpoints* até o monitoramento e análise dos dados pelo servidor.

Afim de testar e validar a arquitetura proposta, no capítulo 5, este trabalho usa como base para seus experimentos um protótipo de uma estacão meteorológica desenvolvido com sensores comerciais, com intuito de medir e validar os conceitos e serviços que foram empregados no desenvolvimento desta arquitetura. Esta arquitetura engloba tanto os componentes de *hardware* o protótipo desenvolvido, bem como seus protocolos para comunicação e aplicação web para análise dos dados coletados. Essas três camadas apresentadas podem ser ilustradas conforme a Figura 4.10.

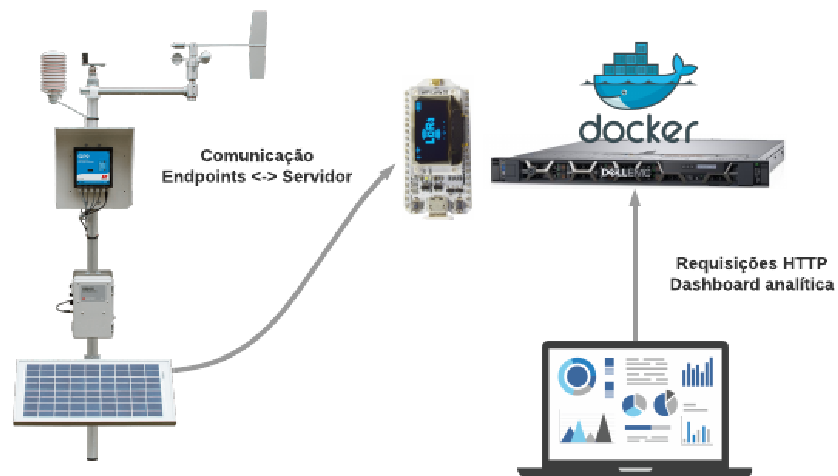

Figura 4.10: Ilustração arquitetura implementada
Fonte: Elaborado pelo Autor

No capítulo seguinte apresenta-se a implementação com parte dos códigos desenvolvidos para cada componente pertencente a esta arquitetura.

Capítulo 5

IMPLEMENTAÇÃO

A arquitetura implementada pode ser dividida em duas etapas, sendo estas a parte cliente, onde são coletados todas as informações dos sensores e realizado a transmissão via MQTT, e a parte servidor onde os dados são processados e analisados. A Figura 5.1 ilustra os pontos de responsabilidade do desenvolvimento pelos *endpoint* e a camada do servidor *middleware*.

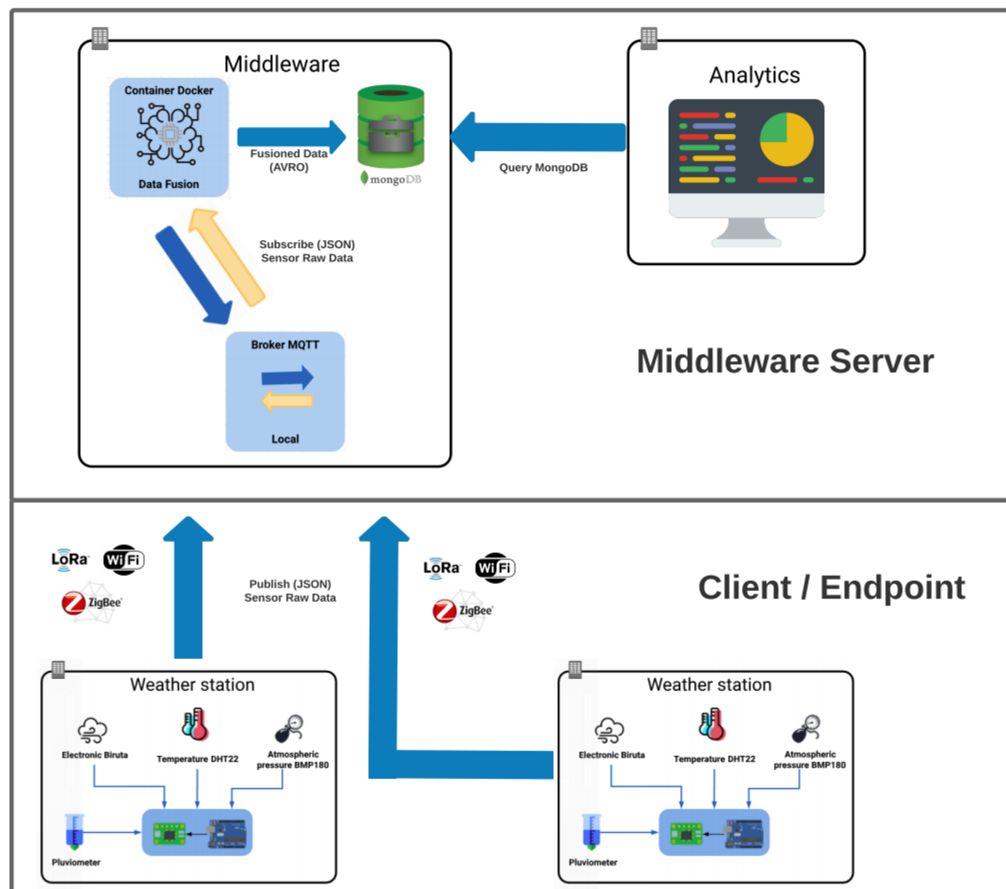

Figura 5.1: Arquitetura implementada - *Server/Endpoint*
Fonte: Elaborado pelo Autor.

5.1 Estação Meteorológica

O desenvolvimento da estação meteorológica foi dividido em duas etapas, a primeira foi realizada a programação em C, onde o arduíno nano torna-se o responsável em gerenciar todo o processo das coletas dos sensores e transmitir essas informações por sua porta serial.

Em sua implementação foi utilizado a biblioteca C *Adafruit-Sensor*, sua função é de abstrair a complexidade dos dados coletados pelos sensores, tornando-se útil para o registro de dados e transmissão, pois trabalha-se com apenas um tipo de dado conhecido para registrar ou transmitir. O desenvolvimento foi realizado em quatro classes distintas, cada uma relacionada a um sensor a fim de testar e validar seu funcionamento. A classe EstacaoSerial.ino é a responsável em unificar as demais classes desenvolvidas e realizar o transporte a porta serial.

A Figura 5.2 ilustra um pequeno trecho do código desenvolvido com a utilização da biblioteca *Adafruit-Sensor*, este coletando as informações dos sensores de umidade e temperatura, sua taxa de transmissão foi desenvolvida na faixa 9600.

```
#include <Wire.h>
#include <Adafruit_BMP085.h>

Adafruit_BMP085 bmp;

void setup() {
  Serial.begin(9600);
  if (!bmp.begin()) {
  Serial.println("Could not find a valid BMP085 sensor, check wiring!");
  while (1) {}
  }
}

void loop() {
    Serial.print("Temperature = ");
    Serial.print(bmp.readTemperature());
    Serial.println(" *C");
    Serial.print("Pressure = ");
    Serial.print(bmp.readPressure());
    Serial.println(" Pa");
    Serial.print("Altitude = ");
    Serial.print(bmp.readAltitude());
    Serial.println(" meters");
    Serial.print("Pressure at sealevel (calculated) = ");
    Serial.print(bmp.readSealevelPressure());
    Serial.println(" Pa");
    Serial.print("Real altitude = ");
    Serial.print(bmp.readAltitude(101500));
    Serial.println(" meters");
    Serial.println();
    delay(500);
}
```

Figura 5.2: Trecho do Código elaborado para estação meteorológica
Fonte: Elaborado pelo Autor.

A etapa seguinte o ponto de transmissão para o MQTT foi desenvolvido em java, este rodando na raspberry pi 3 B+ ligada diretamente ao Arduíno.

Esta etapa consistem no desenvolvimento de um cliente MQTT(*publisher*), o qual realiza as publicações a cada iteração recebida em sua porta serial (USB). A Figura 5.3 ilustra trechos do código desenvolvido, este contendo as informações necessárias para a transmissão ao broker MQTT e seu método *publisher*.

```java
public class Publisher extends AbstractMqttClient {
    public static final String BROKER_URL = "tcp://192.168.0.17:1883";
    public static final String CLIENT_ID = "weather_station_client_publish";
    public static final String TOPIC = "weather_station_client_climate";
    private static final Logger logger = Logger.getLogger(Publisher.class.getName());

    public Publisher(String brokerUrl, String clientId) throws MqttException {
        super(brokerUrl, clientId, new PublisherCallback());
    }
    public void publish(String topic, String msg) {
        try {
            getClient().publish(topic, msg.getBytes(), 1, true);
        } catch (MqttException e) {
            logger.log(Level.SEVERE, "Error publishing to broker.", e);
        }
    }
```

Figura 5.3: Trecho do Código elaborado para transmissão *publisher* MQTT
Fonte: Elaborado pelo Autor.

5.2 Fusão de dados

No desenvolvimento do algoritmo de fusão foi se utilizado uma sequência de etapas essas que devem ser seguidas para sua utilização.

5.2.0.1 Fonte de dados

A captura dos dados vem diretamente do *broker* MQTT local, portando deve-se definir alguns dos requisitos básicos para esse tipo de conexão, esses como QoS estipulado, quais os tópicos a serem consultados, o endereço do *broker* com sua respectiva porta e o identificador do cliente a qual irá realizar as consultas.

Essas informações são necessárias para a conexão com o servidor *broker mosquitto*, instanciado localmente para não haver a necessidade de internet e atender aos requisitos do projeto. Essas informações são adicionadas na classe *Subscriber.java* conforme ilustrado na Figura 5.4.

```
public class Subscriber implements MqttCallback {

    private final int qos = 1;
    private MqttClient client;
    String topic = "weather_station_client_climate";
    String broker = "tcp://192.168.0.17:1883";
    String clientId = "weather_station_client";
    IMessageMttq iMessageMttq;

    public Subscriber(IMessageMttq iMessageMttq) throws MqttException {
        String host = String.format("tcp://%s:%d", "192.168.0.17", 1883);
          MqttConnectOptions conOpt = new MqttConnectOptions();
        conOpt.setCleanSession(true);
        this.client = new MqttClient(host, clientId, new MemoryPersistence());
        this.client.setCallback(this);
        this.client.connect(conOpt);
        this.client.subscribe(this.topic, qos);
        this.iMessageMttq = iMessageMttq;
    }
```

Figura 5.4: Fonte de dados para aplicação - Classe Subscriber.java
Fonte: Elaborado pelo Autor.

Para o desenvolvimento desta classe e a conexão do MQTT foi utilizado a biblioteca java ”org.eclipse.paho.client.mqttv”versão 3-1.2.0.

5.2.0.2 Definição dos dados

No modelo atual da arquitetura desenvolvida, a fusão dos dados está relacionada ao tipo de dado a ser tratado, definimos então esses conforme sua utilização.

Para este projeto foi desenvolvido uma estação meteorológica, em seu desenvolvimento foi constatado que o tipo de dados *BigDecimal* apresentou uma maior precisão quando comparado ao tipo *double*, esse por sua vez apresentou diferenças significativas quanto ao arredondamento das casas decimais, não apresentando uma maior precisão em suas operações aritméticas, sendo está informação facilmente testada com uma soma simples 0.1 + 0.2 = 0.30000000000000004, este problema está relacionado ao tipo de representação binária utilizada no padrão IEEE 754.

As definições dos tipos e atributos dos dados atualmente encontra-se definida na classe *StationDataModel.java*, esta que pode ser facilmente alterado conforme a necessidade do desenvolvedor a utilizar. A Figura 5.5 ilustra a tipagem e os métodos *Gets* e *Sets* utilizados pelo algoritmo.

```java
import java.math.BigDecimal;
/**
 *
 * @author jpmorijo
 */
public class StationDataModel implements DataModel {
    private BigDecimal humidity;
    private BigDecimal temperature;
    private BigDecimal pressure;
    private BigDecimal precipitation;
    private BigDecimal direction;
    private BigDecimal speed;
    public BigDecimal getHumidity() {
    public BigDecimal getTemperature() {
    public BigDecimal getPressure() {
    public BigDecimal getPrecipitation() {
    public BigDecimal getDirection() {
    public BigDecimal getSpeed() {
    public void setHumidity(BigDecimal humidity) {
    public void setTemperature(BigDecimal temperature) {
    public void setPressure(BigDecimal pressure) {
    public void setPrecipitation(BigDecimal precipitation) {
    public void setDirection(BigDecimal direction) {
    public void setSpeed(BigDecimal speed) {
}
```

Figura 5.5: Model da aplicação - Classe StationDataModel.java
Fonte: Elaborado pelo Autor.

5.2.0.3 Método de Fusão Chauvenet

Para o desenvolvimento do algoritmo Chauvenet, utilizamos a classe *Chauvenet.java*, contendo as chamadas de função para a classe BigDecimalUtils.java a qual é responsável por efetuar todos os cálculos com sua conversão em BigDecimal (*package* java.math) conforme ilustrado na Figura 5.6.

```java
public strictfp class Chauvenet {
    public HashMap<Integer,BigDecimal> criterio;
        public BigDecimal newMean;
        public BigDecimal newDesvPadrao;
        public BigDecimal[] newDados;
        protected int n;
        private final BigDecimalUtils stats = new BigDecimalUtils();
        public Chauvenet(){
            criterio = new HashMap<Integer,BigDecimal>(){
                {
                    put(new Integer(2),    new BigDecimal("1.15"));
                    put(new Integer(3),    new BigDecimal("1.38"));
                    put(new Integer(4),    new BigDecimal("1.54"));
                    put(new Integer(5),    new BigDecimal("1.65"));
                    put(new Integer(6),    new BigDecimal("1.73"));
                    put(new Integer(7),    new BigDecimal("1.80"));
                    put(new Integer(10),   new BigDecimal("1.96"));
                    put(new Integer(15),   new BigDecimal("2.13"));
                    put(new Integer(25),   new BigDecimal("2.33"));
                    put(new Integer(50),   new BigDecimal("2.57"));
                    put(new Integer(100),  new BigDecimal("2.81"));
                    put(new Integer(300),  new BigDecimal("3.14"));
                    put(new Integer(500),  new BigDecimal("3.29"));
                    put(new Integer(1000), new BigDecimal("3.48"));
                }
            };
        }
```

```java
 public List run(BigDecimal[] data) throws Exception{
     if(criterio.containsKey(data.length)){
         n = data.length;
         BigDecimal[] dr = stats.dr(data);
         List removedIndex = filter(data, dr);
         return removedIndex;
     }else{
         throw new Exception("Verifique os sensores");
     }
 }
private List filter(BigDecimal[] data, BigDecimal[] dr){
    LinkedList<BigDecimal> dt = new LinkedList();
    dt.addAll(Arrays.asList(data));
    BigDecimal criterioVal = criterio.get(n);
    List<Integer> removedIndex = new LinkedList();
    for(int i = 0; i < dr.length; i++){
        if(dr[i].compareTo(criterioVal) == 1){
            removedIndex.add(new Integer(i));
        }
     }
    BigDecimal[] ret = new BigDecimal[dt.size()];
    for(int i = 0; i< dt.size(); i++){
    ret[i] = dt.get(i);           }
     newMean = stats.mean(ret);
     newDesvPadrao = stats.stdDev(ret);
    return removedIndex;
```

Figura 5.6: Algoritmo de Chauvenet - Classe Chauvenet.java
Fonte: Elaborado pelo Autor.

O trecho do código ilustrado na Figura 5.7 realiza a verificação dos dados que não estão dentro do desvio padrão, os dados a serem excluídos e adiciona os índices excluídos no *HashSet*,

em seguida verifica-se a lista retornada está vazia e se não estiver adiciona o dado coletado.

```
if(hum.length > 1 && temp.length > 1 && pres.length > 1
        && preci.length > 1 && dir.length > 1 && speed.length > 1){
    try {
        Chauvenet chHum = new Chauvenet();
        Chauvenet chTemp = new Chauvenet();
        Chauvenet chPres = new Chauvenet();
        Chauvenet chPreci = new Chauvenet();
        Chauvenet chDir = new Chauvenet();
        Chauvenet chSpeed = new Chauvenet();

        List idxAuxList = chHum.run(hum);
        if(!idxAuxList.isEmpty()){
            removedIndex.addAll(idxAuxList);
        }

        idxAuxList = chTemp.run(temp);
        if(!idxAuxList.isEmpty()){
            removedIndex.addAll(idxAuxList);
        }
```

Figura 5.7: Trecho do código - Classe SimilarFusionFunctions.java
Fonte: Elaborado pelo Autor.

Dando continuação ao código, a Figura 5.8 ilustra a remoção da listagem dos elementos, e percorre a lista para adicionar os dados.

```
if(!removedIndex.isEmpty()){
    Iterator setIterator = removedIndex.iterator();
    while(setIterator.hasNext()){
        Integer idx = (Integer) setIterator.next();
        dataModelList.remove(idx.intValue());
    }
}
hum = new BigDecimal[dataModelList.size()];
temp = new BigDecimal[dataModelList.size()];
pres = new BigDecimal[dataModelList.size()];
preci = new BigDecimal[dataModelList.size()];
dir = new BigDecimal[dataModelList.size()];
speed = new BigDecimal[dataModelList.size()];
for(int j = 0; j < dataModelList.size(); j++){
    hum[j] = dataModelList.get(j).getHumidity();
    temp[j] = dataModelList.get(j).getTemperature();
    pres[j] = dataModelList.get(j).getPressure();
    preci[j] = dataModelList.get(j).getPrecipitation();
    dir[j] = dataModelList.get(j).getDirection();
    speed[j] = dataModelList.get(j).getSpeed();
}
```

Figura 5.8: Trecho do código - Classe SimilarFusionFunctions.java
Fonte: Elaborado pelo Autor.

Finalizando esse trecho da classe *SimilarFusionFunctions.java* a Figura 5.9 ilustra a etapa calculando as médias dos itens novamente.

```
                BigDecimal newMeanHum = stats.mean(hum);
                BigDecimal newMeanTemp = stats.mean(temp);
                BigDecimal newMeanPres = stats.mean(pres);
                BigDecimal newMeanpreci = stats.mean(preci);
                BigDecimal newMeanDir = stats.mean(dir);
                BigDecimal newMeanSpeed = stats.mean(speed);
                StationDataModel dadosNew = new StationDataModel();
                dadosNew.setHumidity(newMeanHum);
                dadosNew.setTemperature(newMeanTemp);
                dadosNew.setPressure(newMeanPres);
                dadosNew.setPrecipitation(newMeanpreci);
                dadosNew.setDirection(newMeanDir);
                dadosNew.setSpeed(newMeanSpeed);
                mid2.getDataModel().add(dadosNew);
                dataModelList.clear();
                removedIndex.clear();
            } catch (Exception ex) {
                System.out.println("Error -> " + ex.getMessage());
            }
        }else{
            StationDataModel aux = new StationDataModel();
            aux.setHumidity(hum[0]);
            aux.setTemperature(temp[0]);
            aux.setPressure(pres[0]);
            aux.setPrecipitation(preci[0]);
            aux.setDirection(dir[0]);
            aux.setSpeed(speed[0]);
            mid2.getDataModel().add(aux);
        }

    }
}
```

Figura 5.9: Trecho do código - Classe SimilarFusionFunctions.java
Fonte: Elaborado pelo Autor.

Esses foram alguns dos trechos do código utilizado para a implementação do critério de Chauvenet, a subseção seguinte descreve a integração do modelo de fusão escolhido com o *middleware open-source* Kaa.

5.2.0.4 Inserção Kaa

Os dados úteis, informações já processados pelo algoritmo de fusão, são então transmitidos para o Kaa. Esta etapa utiliza o SDK gerado no *middleware* com informações como tipo dos dados a serem transmitidos, protocolo de segurança e autenticação, endereço do *host* (IP), tempo de transmissão entre outras informações que serão apresentadas na seção seguinte.

O algoritmo em java implementado para a transmissão ao Kaa pode ser dividido em duas seções distintas, a primeira consiste em analisar e sincronizar as informações implementada pelo SDK. A Figura 5.10 representa esse trecho do código.

```
public class WeatherStation {
    private static final long DEFAULT_START_DELAY = 1000L;
    private static final Logger LOG = LoggerFactory.getLogger(WeatherStation.class);
    private static KaaClient kaaClient;
    private static ScheduledFuture<?> scheduledFuture;
    private static ScheduledExecutorService scheduledExecutorService;
    public static void main(String[] args) {
        LOG.info(WeatherStation.class.getSimpleName() + " app starting!");
        scheduledExecutorService = Executors.newScheduledThreadPool(1);
        DesktopKaaPlatformContext desktopKaaPlatformContext = new DesktopKaaPlatformContext();
        kaaClient = Kaa.newClient(desktopKaaPlatformContext, new FirstKaaClientStateListener(), true);
        RecordCountLogUploadStrategy strategy = new RecordCountLogUploadStrategy(1);
        strategy.setMaxParallelUploads(1);
        kaaClient.setLogUploadStrategy(strategy);
        kaaClient
  .setConfigurationStorage(new SimpleConfigurationStorage(desktopKaaPlatformContext, "saved_config.cfg"));
        kaaClient.addConfigurationListener(new ConfigurationListener() {
            @Override
            public void onConfigurationUpdate(Configuration configuration) {
    LOG.info("Received configuration data. New sample period: {}", configuration.getSamplePeriod());
                onChangedConfiguration(TimeUnit.SECONDS.toMillis(configuration.getSamplePeriod()));
            }
        });
        kaaClient.start();
        try {
            System.in.read();
        } catch (IOException e) {
            LOG.error("IOException has occurred: {}", e.getMessage());
        }
        LOG.info("Stopping...");
        scheduledExecutorService.shutdown();
        kaaClient.stop();
    }
```

Figura 5.10: Inicialização Kaa - Classe WeatherStation.java
Fonte: Elaborado pelo Autor.

Esta parte do algoritmo cria um cliente Kaa e adiciona-se um ouvinte que mostre as configurações ao servidor assim que o cliente é instanciado.

As configurações realizadas neste trecho são persistidas em memória para evitar a reconexão no servidor a toda nova transmissão. A Figura 5.11 representa a última etapa do código, a transmissão dos dados ao *middleware*. Para depuração o código em desenvolvimento foram acrescentados *logs* com informações dos dados a serem transportados.

```java
    private static void onChangedConfiguration(long time) {
        if (time == 5) {
            time = DEFAULT_START_DELAY;
        }
        scheduledFuture.cancel(false);

        scheduledFuture = scheduledExecutorService.scheduleAtFixedRate(
                new Runnable() {
                    @Override
                    public void run() {
                      double temperature = getTemperature();
                      double humidity = getHumidity();
                      double direction = getDirection();
                      double precipitation = getPrecipitation();
                      double pressure = getPressure();
                      double speed = getSpeed();
                        kaaClient.addLogRecord(new WeatherStation(temperature,humidity,
                                direction,precipitation,pressure,speed));
  //LOGs debug
                        LOG.info("Sampled Temperature: {}", temperature);
                        LOG.info("Sampled Humidity: {}", humidity);
                        LOG.info("Sampled Direction: {}", direction);
                        LOG.info("Sampled Precipitation: {}", precipitation);
                        LOG.info("Sampled Pressure: {}", pressure);
                        LOG.info("Sampled Speed: {}", speed);
                    }
                }, 0, time, TimeUnit.MILLISECONDS);
    }
```

Figura 5.11: Sincronização dos dados - Classe WeatherStation.java
Fonte: Elaborado pelo Autor.

5.3 Kaa IoT platform

No desenvolvimento da aplicação utilizando o *middleware open-source* Kaa Iot, para seu funcionamento precisamos cadastrar algumas informações essenciais essas que são utilizadas na criação do SDK para desenvolvimento.

A versão Kaa IoT platform 0.9.0 utilizado no projeto não possui integração direta com o desenvolvimento sem o SDK ou comunicação com MQTT, esses passos foram supridos nas etapas do cliente fazendo a transmissão ao *broker* MQTT *mosquitto* e ao lado do servidor na aplicação de fusão dos dados onde é utilizado a aplicação para realizar a coleta dos dados do MQTT (*subscribe*) e inseridos aos *middleware* com as informações disponíveis no SDK.

O recurso de coleta dos dados (*Configuration schema*) permite o envio dos dados de clientes (*endpoints*) para o servidor Kaa, nosso período de coleta foi definido com intervalo de 5 segundo por coleta, este que pode variar de acordo com o tipo de informação a ser processada pelo *middleware* e podendo ser facilmente alterado na aplicação cliente passando um novo intervalo de tempo quando necessário. A Figura 5.12 apresenta a configuração JSON utilizada na criação desta configuração.

Para o armazenamento das informações coletadas pelos cliente foi utilizado o esquema com

```
{
    "type": "record",
    "name": "GenealConfiguration",
    "namespace": "org.kaaproject.kaa.schema.weatherstation",
    "fields": [
        {
            "name": "samplePeriod",
            "type": "int",
            "by_default": 5
        }
    ]
}
```

Figura 5.12: Definição período de coleta Kaa
Fonte: Elaborado pelo Autor.

os atributos e informações que serão recebidas no servidor. O recurso desta configuração foi elaborado com base nos itens transportados pela estação meteorológica. A Figura 5.13 apresenta a configuração do JSON utilizado na criação dos tipos a serem recebidos pelo servidor.

```
{
    "type": "record",
    "name": "WeatherStation",
    "namespace": "org.kaaproject.kaa.schema.weatherstation",
    "fields": [
        {
            "name": "temperature",
            "type": "double",
            "by_default" : 0
        },
        {
            "name": "humidity",
            "type": "double",
            "by_default" : 0
        },
        {
            "name": "direction",
            "type": "double",
            "by_default" : 0
        },
        {
            "name": "precipitation",
            "type": "double",
            "by_default" : 0
        },
        {
            "name": "speed",
            "type": "double",
            "by_default" : 0
        },

        {
            "name": "pressure",
            "type": "double",
            "by_default" : 0
        }
    ]
}
```

Figura 5.13: Tipagem dos dados coletados pelo *endpoint*
Fonte: Elaborado pelo Autor.

O formato de configuração do kaa é baseado no esquema Apache Avro, uma estrutura de serialização de *big data* que produz dados em um formato JSON binário compacto - BSON (AVRO, 2018). A estrutura do AVRO oferece suporte apenas aos tipos primitivos de dados esse são:

- *null* - informações nulas

- *boolean* - *true* ou *false*
- *int* - Valor numérico de (-2 ^ 31 + 1) a (2 ^ 31-1)
- *long* - Valor numérico de (-2 ^ 63 + 1) a (2 ^ 63-1)
- *float* - Valor de ponto flutuante
- *double* - Valor de ponto flutuante
- *bytes* - JSON matriz de valores de *bytes*
- *string* - Formato simples de *string*

Com a utilização da fusão de sensores na etapa anterior, as informações armazenadas no formato *double* não sofrem alterações em seu arredondamento.

5.4 Análise dos dados

Para visualização dos dados com a utilização do apache Zeppelin, foi necessário apenas a criação das *querys* para visualização dos dados armazenados no mongoDB.

Para não haver poluição visual nos relatórios, as *querys* foram realizadas com a coleta em um intervalo dos 60 últimos resultados coletados, sendo esse facilmente alterado modificando apenas o intervalo de amostra na query.

Um dos grande benefício na utilização do mongoDB para a geração dos relatórios dos dados processados pelo *middleware*, esta na utilização das consultas genéricas, não sendo necessário o cadastro dos campos a serem processados pelo apache Zeppelin, comparado com o banco cassandra suportado pela ferramenta mas necessita do cadastro prévio de suas tabelas em seu interpretador. A Figura 5.14 ilustra a criação do gráfico para análise da temperatura, nela pode-se observar a consulta ao banco e logo abaixo os itens a serem selecionados para criação do gráfico, e a Figura 5.15 apresenta os gráficos elaborados através das informações coletados pela estação meteorológica.

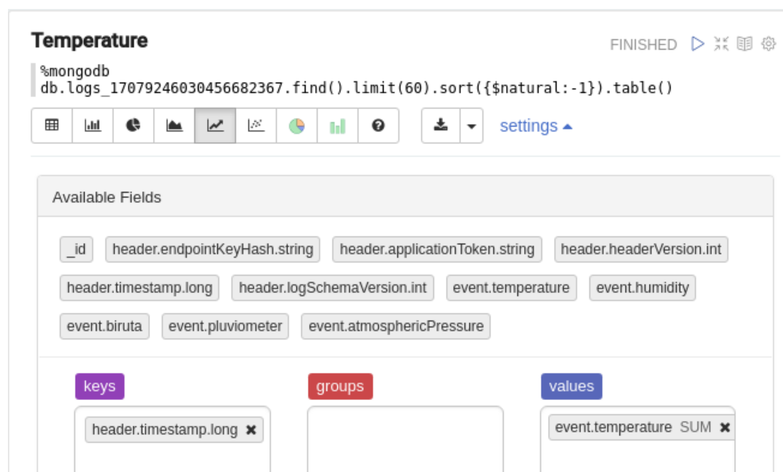

Figura 5.14: Query MongoDB - Seleção de itens
Fonte: Elaborado pelo Autor.

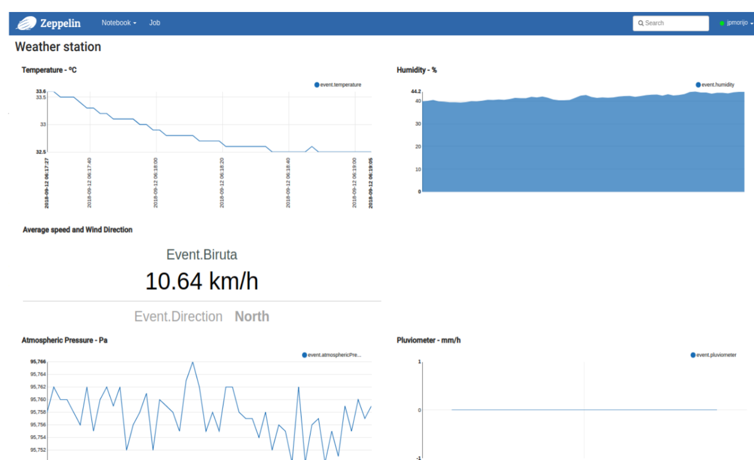

Figura 5.15: *Dashboard* para análise dos dados
Fonte: Elaborado pelo Autor.

5.5 Conclusão do Capítulo

Neste capítulo são apresentados partes das implementações realizadas, essa que se dividem entre o *endpoint* e o servidor. Conforme já apresentado cada funcionalidade é tratada como serviços, com isso cada item torna-se isolado podendo haver substituições ou correções sem afetar a arquitetura como um todo.

No capítulo seguinte são apresentados os testes realizados em campo e simulados para obtenção da capacidade e avaliação da arquitetura.

Capítulo 6

AVALIAÇÃO

A Fazenda Ribeirão dos Índios situada na cidade de Marília - SP foi utilizada como teste no experimento para coleta e transmissões ao *Middleware* implementado. A propriedade com aproximadamente 500 ha, tem como sua principal atividade econômica a plantação de soja, milho e trigo. A Figura 6.1 via satélite gerada pelo aplicativo *Fieldview*, apresenta o principal talhão utilizado para a realização do experimento, o mesmo possuindo uma área de aproximada 208.910,24 m (2.248.691 ft) com perímetro de 2.069,71 m (6.790 pés).

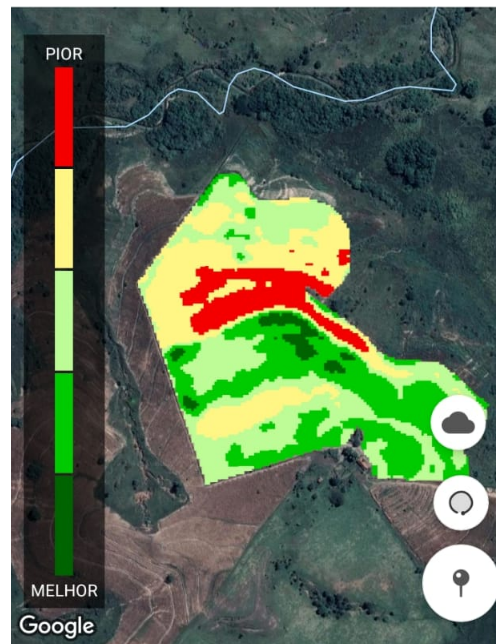

Figura 6.1: Visão vegetativa do talhão.
Fonte: Elaborado pelo autor.

A Figura 6.1 ilustra a análise de vegetação em imagem NDVI (*Normalized Difference Vegetation*), esta interpretada da seguinte maneira. A mancha amarela no meio do talhão é onde ocorreu uma doença em reboleira já controlada, os contornos em vermelho onde o solo está

preparado sem plantio, e no restante do amarelo próximo do vermelho é o solo com palhada pronto para plantio, as partes em verde representam o plantio. Nesta região o índice de doença é alto devido a alta umidade e fortes ventos uma ótima junção para disseminação de doenças essas como a Ferrugem asiática que pode devastar lavouras em aproximadamente 5 dias.

Para a avaliação da solução, a estação meteorológica desenvolvida foi levada a campo, esse *endpoint* fixado a um mastro possuindo 1,8 metros altura, distância suficiente para as transmissões dos dados coletados sem interferência do meio ambiente, a plantação de soja. A Figura 6.2 apresenta a estação meteorológica fixada ao meio da plantação.

Figura 6.2: Estação Meteorológica em Campo
Fonte: Elaborado pelo autor.

A transmissão dos dados utilizando o módulo Xbee S2C 802.15.4 obteve uma distância de aproximadamente 363,55 metros com perda mínima de dados. Foram realizados testes com transmissões com intervalos de 1, 2 e 5 segundos do *endpoint* ao *middleware*. A Figura 6.3 apresenta a distância entre a estação meteorológica localizada no meio da vegetação, e o *Middleware*, base montada para monitorar localizada na porteira de entrada do talhão.

Figura 6.3: Cenário do Proposto.
Fonte: Elaborado pelo autor.

Os dados coletados pela aplicação foram armazenados na raspberry no banco sqlite e transmitidos para o servidor de homologação onde os dados foram fusionados e armazenados em sua respectiva base NoSQL (MongoDB) e apresentados pelo camada web Apache Zeppelin.

As Figuras 6.4 e 6.5 ilustram a coleta e eliminação de dados atípicos, esses coletados através da estação meteorológica com a utilização de dupla leitura do sensor BMP180 para análise da pressão atmosférica e do sensor DHT22 para análise da temperatura e umidade.

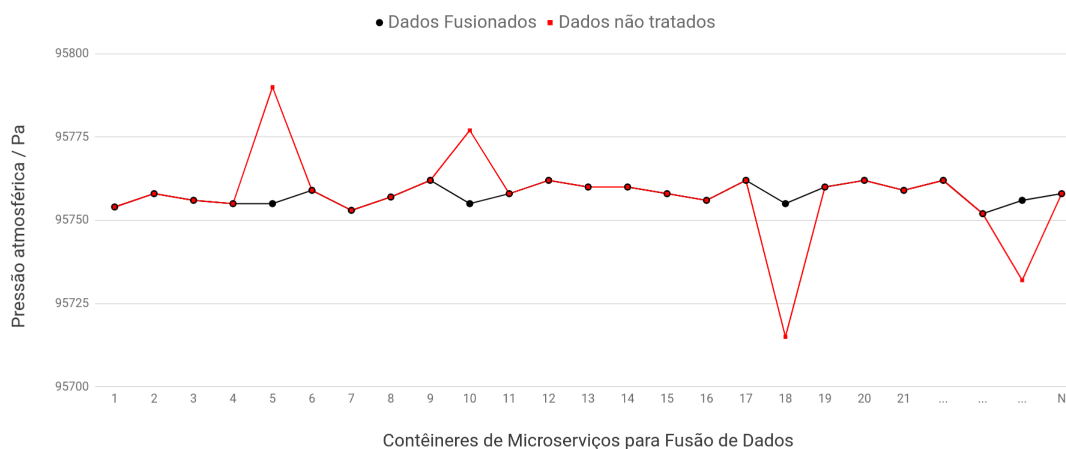

Figura 6.4: Comparativo de dados fusionados - Pressão atmosférica
Fonte: Elaborado pelo autor.

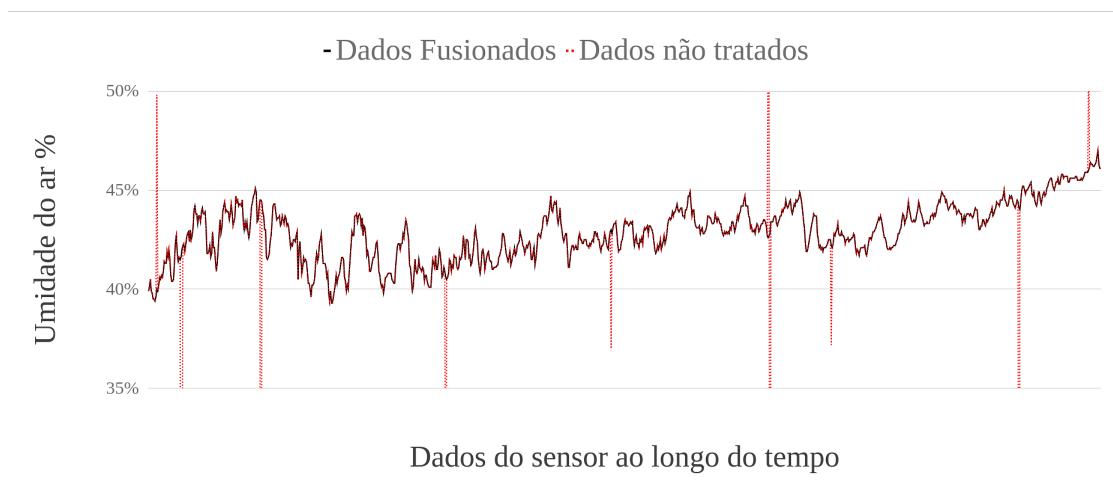

Figura 6.5: Comparativo de dados fusionados - Umidade relativa do ar
Fonte: Elaborado pelo autor.

As Figuras ilustram o descarte de informações que poderiam ser armazenadas de forma erroneamente ocupando espaço em disco desnecessário. Na ilustração a quantidade de itens discrepantes são mínimos mas esses dados sendo armazenados diariamente podem apresentar um grande montante ao final.

As informações geradas com intervalo de 1 segundo ocasionaram em um alto tráfego de informações, ocasionando o reencaminhamento de mensagens não recebidas ao *broker* MQTT. O intervalo de 5 segundos para transmissão apresenta grande perda de dados valiosos a análise e a ocorrência de múltiplos casos de dados atípicos, com isso ocorrendo a exclusão de informações válidas ao sistema.

Entretanto dados com intervalo de 2 segundos se mostraram mais preciso, pois suas variações em relação aos valores continuam dentro da média e desvio padrão calculados pelo algoritmo, não ocasionando a exclusão de informações reais.

O consumo de *hardware* no servidor apresentou-se estável, não havendo oscilação em seus componentes como CPU, memória e disco. A transmissão de um *endpoint* (estação meteorológica) para cinco contêineres java permaneceu-se constante havendo apenas o aumento no período de início das aplicações e logo estabilizando seu consumo.

As Figura 6.6 ilustra o consumo de CPU e seu *load* no dia do teste 09 de dezembro de 2018.

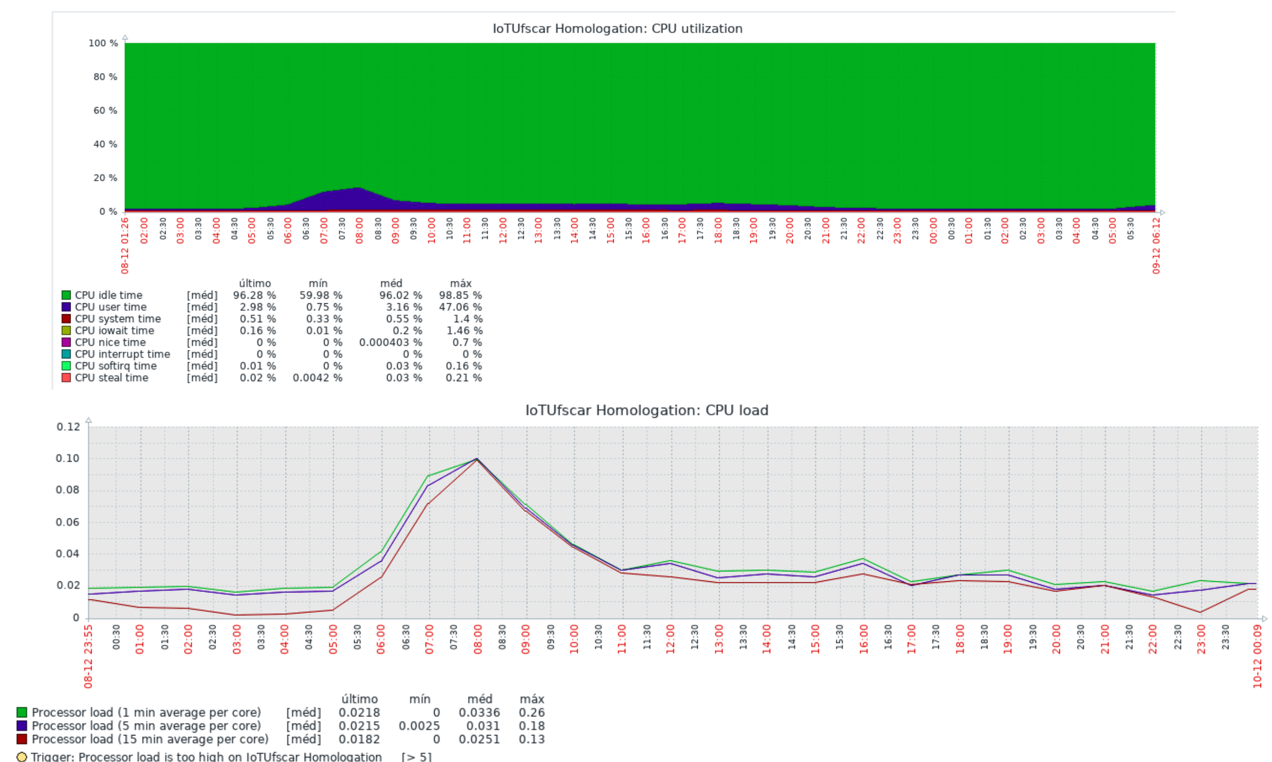

Figura 6.6: Consumo de *hardware* teste em campo

Fonte: Elaborado pelo autor.

Para análise foi selecionado o intervalo de 1 hora de transmissão com maiores oscilações de dados, isso constatando pela pressão atmosférica e umidade presente no ambiente. Esse intervalo corresponde a 1.800 mensagens transmitidas, com aproveitamento de 1.747 uma perda de 2,94%.

6.1 Avaliação de Desempenho

Nas simulações realizadas para análise do desempenho dos microserviços desenvolvidos, o ambiente se mostrou estável com uma carga de até 77 contêineres java rodando simultaneamente em conjunto com toda a arquitetura. Cada contêiner em execução recebe a leitura de 4 *endpoints* distintos a cada 2 segundos, aplicando o critério de Chauvenet seu N corresponde a 4 leituras.

Em sua análise foram levados em conta o consumo de CPU, memória e *load*. O consumo de memória utilizado pelo java se mostrou com crescimento exponencial, sendo necessário a realização de limpezas constantes para sua redução, atualização essa implementada em rotinas de limpeza de intervalo fixo de 2 minutos.

Nas Figuras 6.7, 6.8 e 6.9 são apresentados o consumo de recursos simulando os processos

no período do dia 17 de dezembro de 2018 das 15:10hs às 17:06hs.

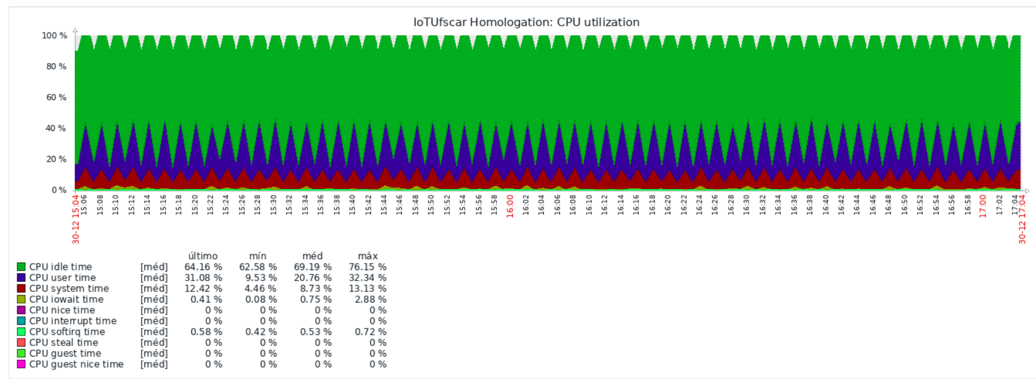

Figura 6.7: CPU *Utilization*
Fonte: Elaborado pelo autor.

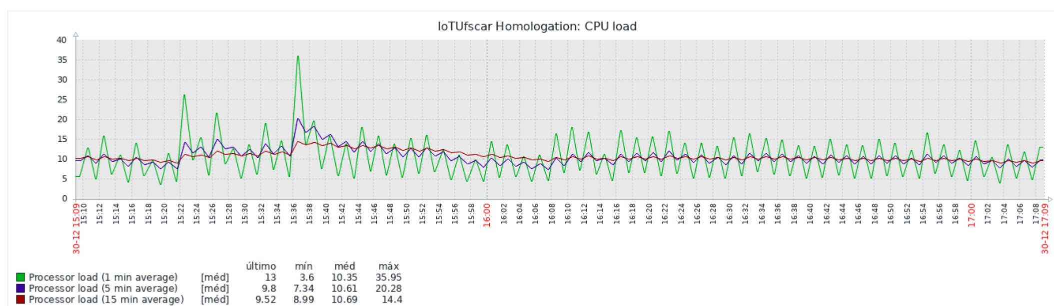

Figura 6.8: CPU Load
Fonte: Elaborado pelo autor.

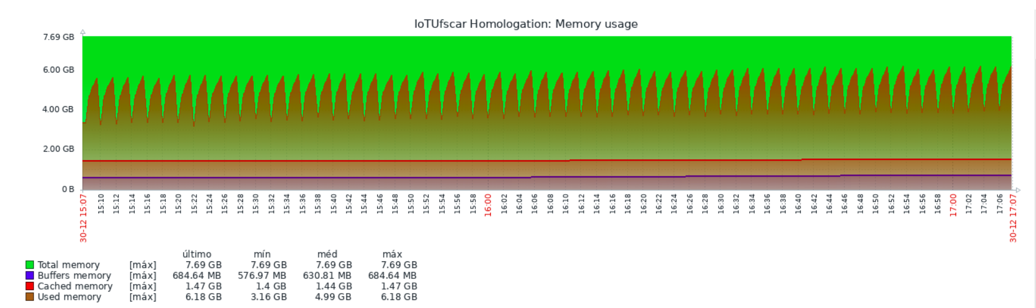

Figura 6.9: Memória utilizada
Fonte: Elaborado pelo autor.

Os testes avaliando as tecnologias de transmissões com os protocolos Zigbee e LoRaWAN, foi realizado a transmissão de mensagens com tamanho fixo de 90 caracteres, totalizando 90 bytes, mensagens estruturadas em formato JSON padronizando assim as transmissões para o *broker* MQTT. A Figura 6.10 exemplifica a mensagem e seu formato enviado.

```
{
    "hum":"39.50",
    "temp":"33.90",
    "pres":"95759.00",
    "preci":"0.00",
    "dir":"90",
    "speed":"11.97"
}
```


Figura 6.10: Mensagem estruturada JSON para dados da estação meteorológica
Fonte: Elaborado pelo Autor.

Na Figura 6.11 é possível observar que há uma tendência no aumento da latência em relação ao aumento da distância entre os rádios. É possível notar que a variância é maior quando maior for a distância. O módulo de rádio Xbee apresenta limitações inferiores ao esp32 Lora, a maior distância alcançada pelo rádio Lora foi 2 Km. Não foi possível localizar um ponto mais distante com visada limpa para testar o limite técnico do rádio, que pode variar até 12 Km.

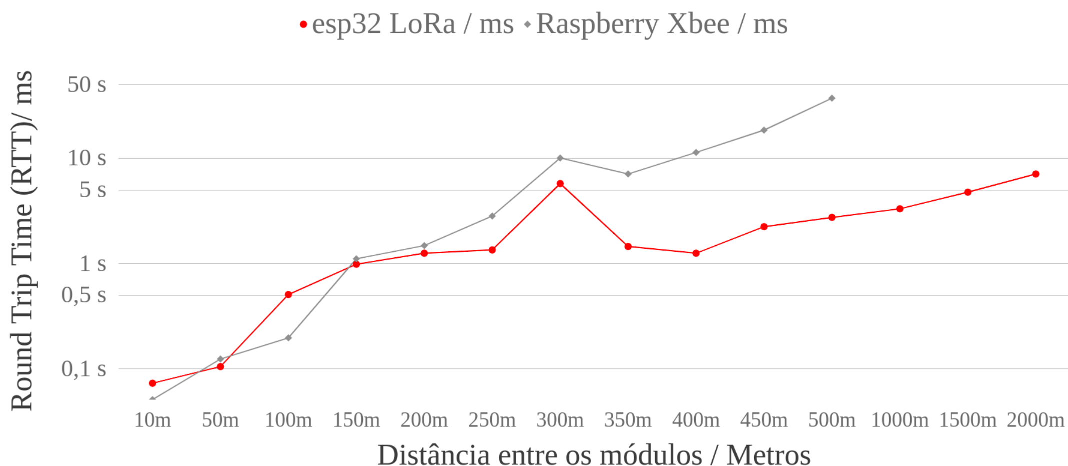

Figura 6.11: Comparativo Servidor X Raspberry PI B+
Fonte: Elaborado pelo Autor.

Avaliando os resultados, com distâncias de 10 até 200 metros, o RTT (*Round Trip Time*) é semelhante para ambas as tecnologias, elevando essas distâncias, o módulo esp32 LoRa mostrou-se com desempenho superior ao xbee, ganho esse que equivale de 4 a 10 vezes mais rápido quando comparado as distâncias de 350 a 500 metros, mostrando que o LoRa é mais eficiente em distâncias maiores para coleta de dados de sensores.

O melhor desempenho do LoRa pode ser explicado pela ausência de tempo de processamento do cabeçalho TCP/IP em comparação com o ZigBee, o que causa um efeito na latência. Além disso, as inovações na modulação do rádio do LoRa, que usa o espectro de dispersão de *chilrear* CSS, contribuem para minimizar a necessidade de retransmissão de quadros.

O pico visto em torno de 300 metros pode ser explicado por obstáculos, arbustos, neste caso, entre os pontos finais e os *gateways* que causaram difração e retransmissão.

6.2 Teste de Carga

Considerando que a fusão de dados pode ser realizada no DFMC (*Data Fusion Microservice Containers*), foram executados testes de desempenho com as fusões de dados sendo executadas no *middleware* IoT com um único DFMC contra a mesma carga sendo processada em uma raspberry pi 3 b+, ambos implementados com mesmo algoritmo de Chauvenet em java. A Figura 6.11 mostra que quando o número de fusão de dados aumenta além de 250 o tempo de fusão tende a crescer exponencialmente em contraste com o crescimento quase linear no tempo de processamento no DFMC implementado em nossa plataforma.

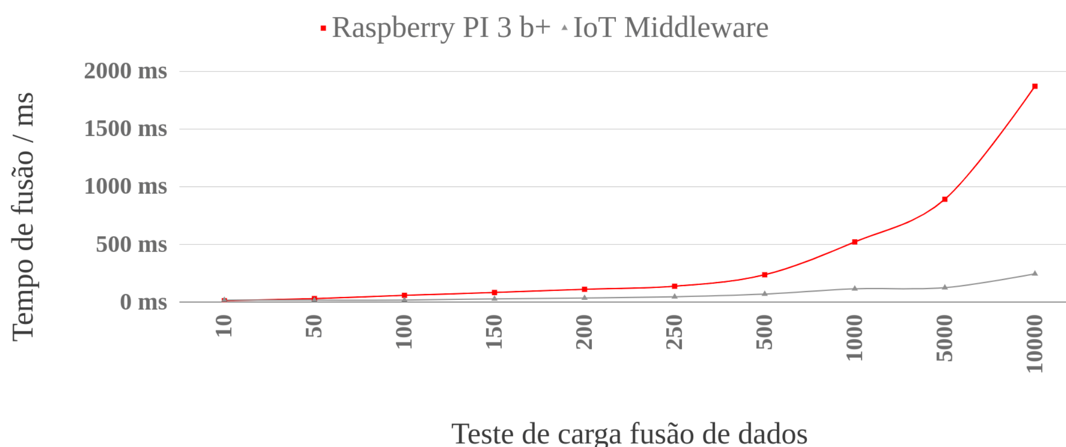

Figura 6.12: Teste de carga fusão de dados
Fonte: Elaborado pelo Autor.

6.3 Teste de Stress

Conforme apresentado na avaliação de desempenho, o ambiente se demonstra estável e escalável com a utilização de até 77 contêineres java rodando simultaneamente. Para estressar a plataforma foram instanciados 100 contêineres java aplicando o critério Chauvenet com N = 3 ou seja , 3 leituras por fusão, sendo removido o intervalo de espera de 5 segundos para cada execução. Cada base com os respectivos dados estava populada com 10.000 leituras, essas que eram reinicializadas assim que havia falhas ou em seu término.

As Figuras 6.13, 6.14 e 6.15 ilustram o momento da execução dos testes.

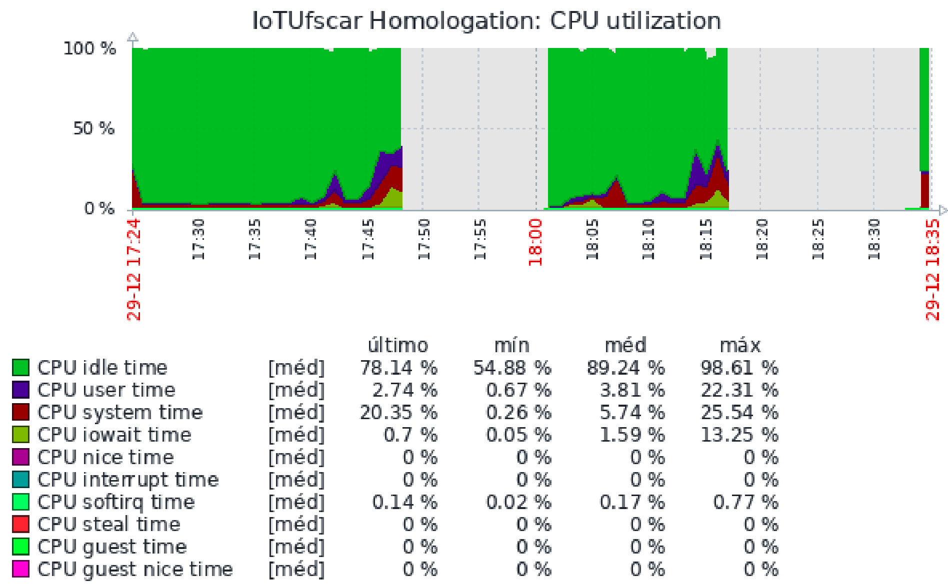

Figura 6.13: Teste de Stress - CPU *Utilization*
Fonte: Elaborado pelo autor.

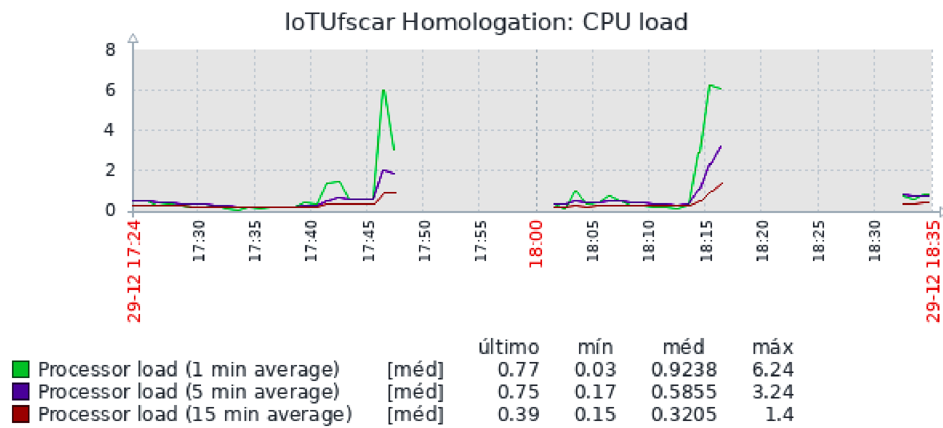

Figura 6.14: Teste de Stress - CPU *Load*
Fonte: Elaborado pelo autor.

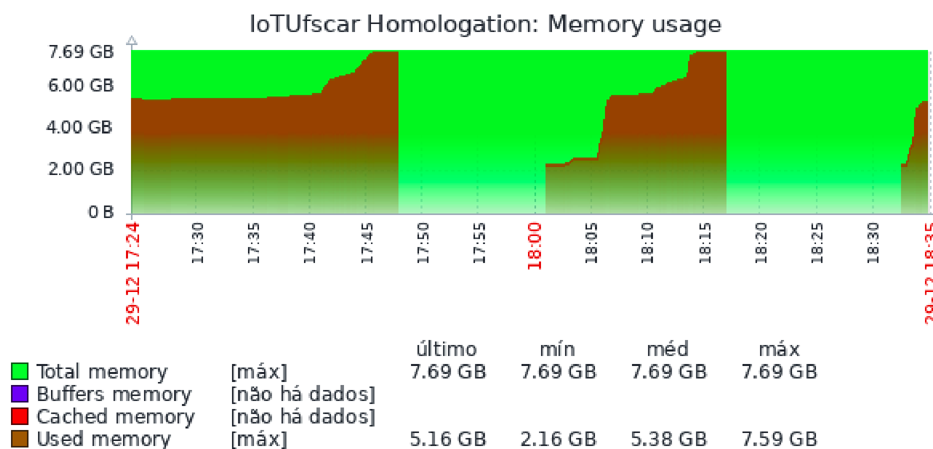

Figura 6.15: Teste de Stress - Memória utilizada
Fonte: Elaborado pelo autor.

Em análise aos gráficos, é possível constatar interrupções de coletas, sendo essas geradas pelo grande aumento de *load* e ou estouro de memória, com isso matando todos os processos existentes no servidor.

6.4 Conclusão do Capítulo

Neste capítulo são apresentados os testes realizados em campo e simulados para obtenção da capacidade e avaliação da arquitetura. Em análise aos testes a arquitetura se mostrou estável compondo até 77 contêineres realizando fusões simultâneas. Entretanto temos de levar em conta que os testes e avaliações foram colocadas em práticas em um ambiente *Fog*, não sendo possível a escalabilidade vertical com facilidade, estes que poderia ser facilmente suprido em um ambiente de *Cloud*.

No capítulo seguinte são apresentadas as considerações finais do desenvolvimento desta arquitetura e possíveis trabalhos futuros.

Capítulo 7

Considerações finais

7.1 Conclusões

O objetivo deste trabalho foi o desenvolvimento de uma arquitetura que possa atender as necessidades e requisitos para uma plataforma IoT em um ambiente *Fog Computing*, sendo esta escalável para suprir o constante aumento de dispositivos que possam ser integrados.

O desenvolvimento foi composto por uma estação meteorológica, representando os *endpoints* o qual ficou responsável pela coleta dos dados e realizar suas transmissões via Zigbee ou LoRaWan de acordo com o microcontrolador escolhido.

Em relação às transmissões do *gateway* para a arquitetura foi utilizado o protocolo de transporte MQTT o qual transportava as mensagens padronizadas e estruturadas em formato JSON.

Em relação a plataforma foram implementados contêineres java os quais funcionam como clientes *subscribe* coletando as informações do *broker* MQTT *mosquitto* local, fusionando as informações e eliminando dados discrepantes, assim inserindo os dados no *middleware* e banco MongoDB através de requisições HTTP padronizadas no formato AVRO. Dados esses que podem ser vistos e analisados em gráficos elaborados pelo apache Zeppelin.

Ao final deste projeto apresenta-se uma arquitetura multissensorial, com a capacidade de abstrair a complexidade de inserções de novos dispositivos(*endpoints*) e a possibilidade de integração de novos algoritmos de fusões ao projeto.

7.2 Contribuições

Com a utilização da arquitetura implementa por este projeto, acredita-se que seja possível beneficiar áreas tecnológicas como a DL (*Deep Learning*) ou ML (*Machine Learning*) em sua análise para os dados processados e armazenado por esta arquitetura.

Acredita-se que este trabalho possa servir como uma ferramenta de busca para possíveis trabalhos que possam surgir nesta linha de pesquisa, esses que podem vir não apenas beneficiar o lado acadêmico mas também empresas e indústrias que estejam atrás de técnicas e tecnologias acessíveis que possam agregar conhecimento e lucros aos seus negócios.

7.3 Trabalhos Futuros

A seguir são apresentadas algumas das possibilidades de melhoria deste trabalho e também ideias para possíveis trabalhos:

- Incluir e validar algoritmos de fusão de dados em dispositivos finais (*endpoints*).
- Aprofundar os estudos nos algoritmos de fusão de dados para contribuir na análise do impacto que os mesmo podem exercer sobre a plataforma.
- Atualizar a plataforma para a nova versão do *middleware* Kaa IoT que ainda está em desenvolvimento, ou realizar um *fork* na versão atual e implementar novos tipos de conectividade.
- Avaliar o comportamento da plataforma em um ambiente *cloud computing*, com isso tirando-se as limitações de consumo de *hardware*.

REFERÊNCIAS

ALAM, F. et al. Analysis of eight data mining algorithms for smarter internet of things (iot). *Procedia Computer Science*, v. 98, p. 437–442, 2016.

AMARAL, U. e. a. Performance evaluation of microservices architectures using containers. *International Symposium on Network Computing and Applications*, IEEE, 2015.

ANTONI, M.; VIVIAN, G. R.; PREUSS, E. Implementação de uma nuvem de armazenamento privada usando owncloud e raspberry pi. *Anais do EATI - Encontro Anual de Tecnologia da Informação e Semana Acadêmica de Tecnologia da Informação*, v. 1, p. 55–62, 2015.

AVRO, A. *Apache Avro*. [S.l.], 2018. Disponível em: <http://avro.apache.org/docs/current/spec.html>. Acesso em: 6 jul. 2017.

BISHOP, G.; WELCH, G. An introduction to the kalman filter. 2001. Disponível em: <http://www.cs.unc.edu/ tracker/media/pdf/SIGGRAPH2001$_{C}$*oursePack*$_{0}$8.*pdf*¿.*Acessoem* : 07*fev*.2018.

COULOURIS, G. Sistemas distribuídos: Conceitos e projeto. Bookman Editora, 2013.

DOCKER, I. *What is a Container*. [S.l.], 2017. Disponível em: <https://www.docker.com/resources/what-container>. Acesso em: 14 nov. 2017.

DOLGIY, A. I.; KOVALEV, S. M.; KOLODENKOVA, A. E. Processing heterogeneous diagnostic information on the basis of a hybrid neural model of dempster-shafer. *16th Russian Conference, RCAI 2018*, 2018.

ERKLART, U. k. Cloud computing, edge computing und fog computing. 2018. Disponível em: <https://innovative-trends.de/2018/01/02/cloud-computing-edge-computing-und-fog-computing-unterschiede-kurz-erklart/>. Acesso em: 05 abr. 2018.

FOUNTAS, S.; PEDERSEN, S.; BLACKMORE, S. Ict in precision agriculture–diffusion of technology. 2005.

FOWLER, M. Microservices. 2014. Disponível em: <http://martinfowler.com/articles/microservices.html>. Acesso em: 16 ago. 2018.

HAJIBABA, M.; GORGIN, S. A review on modern distributed computing paradigms: Cloud computing, jungle computing and fog computing. *Journal of computing and information technology*, v. 22, n. 2, p. 69–84, 2014.

HIDEG, A. et al. Data collection for widely distributed mass of sensors. In: . [S.l.]: IEEE, 2016. p. 000193–000198.

HUNG, L.-H. et al. Guidock: Using docker containers with a common graphics user interface to address the reproducibility of research. 2016. Disponível em: <https://bit.ly/2sTb6vt>. Acesso em: 15 mar. 2018.

HUNKELER, U. Mqtt-s - a publish/subscribe protocol for wireless sensor networks. *Industrial Electronics Society, 2008. Comsware 2008*, IEEE, p. 791–798, 2008.

IBM. Iaas, paas and saas – ibm cloud service models. 2018. Disponível em: <https://www.ibm.com/cloud/learn/iaas-paas-saas>. Acesso em: 22 nov. 2018.

IVANOV, K. *Containerization with LXC*. [S.l.]: Packt Publishing - ebooks Account, 2017.

KAAIOT. *Architecture overview KaaIoT*. 10. ed. [S.l.], 2018. Disponível em: <https://kaaproject.github.io/kaa/docs/v0.10.0/Architecture-overview/>. Acesso em: 26 dez. 2017.

KHALEGHI, B. et al. Multisensor data fusion: A review of the state-of-the-art. *Elsevier Journal*, ScienceDirect, p. 28—-44, 2013.

KODALI, R. K.; SORATKAL, S. Mqtt based home automation system using esp8266. *IEEE Region 10 Humanitarian Technology Conference (R10-HTC)*, IEEE, 2016.

KOUBÂA, A.; ALVES, M.; TOVAR, E. Ieee 802.15.4: a federating communication protocol for time-sensitive wireless sensor networks. *URL: http://www.hurray.isep.ipp.pt*, 2006.

KOVÁCS, Comparison of different linux containers. *International Conference on Telecommunications and Signal Processing (TSP)*, IEEE, 2017.

LAVRIC, A.; POPA, V. Internet of things and loratm low-power widearea networks: A survey. *Signals, Circuits and Systems (ISSCS)*, IEEE, 2017.

LORA. Lora anatel. regulamentação. 2017. Disponível em: <http://abinc.org.br/www/2017/06/26/anatel-muda-lei-para-aprovacao-de-redes-lorawan/>. Acesso em: 14 Nov. 2017.

MOLIN, J. P. Tendências da agricultura de precisão no brasil. *Congresso Brasileiro de Agricultura de Precisão*, 2004. Disponível em: <http://www.agriculturadeprecisao.org.br/upimg/publicacoes/pub$_{t}endencias - da - agricultura - de - precisao - no - brasil - 27 - 08 - 2014.pdf$¿$.Acessoem : 21 julh.$2017.

MQTT. *MQTT v3.1.1*. [S.l.], 2017. Disponível em: <http://mqtt.org/>. Acesso em: 03 mai. 2017.

NAIME, J.; NETO, J.; VAZ, C. Avaliação geral, resultados, perspectivas e uso de ferramentas de agricultura de precisão. *Agricultura de precisão: um novo olhar*, p. 69–72, 2011.

NAMIOT, D.; SNEPS-SNEPPE, M. On micro-services architecture. *International Journal of Open Information Technologies*, v. 2, 2014.

NASCIMENTO, J. A. do. *IEEE 802.15.4 – Redes de sensores sem fio como infraestrutura para comunicação entre veículos e sistemas de controle*. Dissertao (Mestrado) — UNICAMP, 2007.

NAYYAR, A.; PURI, V. A review of arduino board's, lilypad's arduino shields. *3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, 2016.

NETO, M. V. d. S. *Virtualização - 2ª Edição: Tecnologia Central do Datacenter*. [S.l.]: Brasport, 2015. 52–60 p.

NEWMAN, S. *Building Microservices*. [S.l.]: O'Reilly Media, Inc., 2015.

NOGUEIRA, J. et al. Tecnologia de nós sensores sem fio. 2004.

NOWAK, P.; PIERCE, F. Aspects of precision agriculture. v. 67, p. 1–85, 1999.

PESONEN, L.; KOSKINEN, H.; RYDBERG, A. *InfoXT-User-centric mobile information management in automated plant production*. [S.l.], 2008.

PHAM, L. M. A big data analytics framework for iot applications in the cloud. In: . [S.l.]: science computer science and communication engineering, 2015. v. 31, n. 2, p. 44–55.

POPEK, G. J.; GOLBDERG, R. P. Formal requirements for virtualizable third generation architectures. v. 17, p. 412–421, 1974.

PRINCY, S. E.; NIGEL, K. G. J. Implementation of cloud server for real time data storage using raspberry pi. *Online International Conference on Green Engineering and Technologies (IC-GET)*, IEEE, p. 1–4, 2015.

RAY, P. P. An introduction to dew computing: Definition, concept and implications. In: . [S.l.]: IEEE, 2017.

RAZZAQUE, M. A. et al. Middleware for internet of things: A survey. *Journal of computing and information technology*, IEEE INTERNET OF THINGS JOURNAL, v. 3, n. 1, 2014.

SALUSTIANO, R. E. *Aplicação de Técnicas de Fusão de Sensores no Monitoramento de Ambientes*. Dissertao (Mestrado) — UNICAMP, 2006.

SEMTECH. *LoRa™ Modulation Basics*. 2. ed. [S.l.], 2015. Disponível em: <https://www.semtech.com/uploads/documents/an1200.22.pdf>. Acesso em: 07 jan. 2018.

SENSORHUB. 2017. Disponível em: <https://opensensorhub.org>. Acesso em: 13 jan. 2017.

SILVA, J. p. d. Ecocit: Uma plataforma escalável para desenvolvimento de aplicações de iot. *Dissertação (Mestrado) — Universidade federal do Rio Grande do Norte*, 2017.

SQLITE. *SQLite*. [S.l.], 2018. Disponível em: <https://www.sqlite.org/cli.html>. Acesso em: 25 set. 2018.

TAYLOR, J. R. *An Introduction to Error Analysis: The Study of Uncertainties in Physical Measurements*. [S.l.]: University Science Books, 1997.

THOMAZINI, D.; ALBUQUERQUE, P. *Sensores Industriais - Fundamentos e Aplicações*. [S.l.]: eRÍCA, 2011.

TORRES, A. et al. Outlier detection methods and sensor data fusion for precision agriculture. *9º SBCUP – Brazilian Symposium of Ubiquitous and Pervasive Computing, São Paulo, SP, Brazil*, 2017.

VILLANUEVA, J. M. M. *Fusão de Dados das Técnicas de Tempo de Trânsito Utilizando Transdutores Ultra-Sônicos para Medição da Velocidade do Vento*. Dissertao (Mestrado) — PUC-Pontifícia Universidade Católica do Rio de Janeiro, 2009.

WENDLING, M. *Sensores*. 2. ed. [S.l.], 2010. Disponível em: <http://www2.feg.unesp.br/Home/PaginasPessoais/ProfMarceloWendling/4—sensores-v2.0.pdf>. Acesso em: 21 ago. 2017.

WHITMORE, A.; AGARWAL, A.; XU, L. D. The internet of things — a survey of topics and trends. information systems frontiers. *A Journal of Research and Innovation*, v. 17, p. 261–274, 2015.

ZABBIX, L. *Zabbix 4.0*. [S.l.], 2018. Disponível em: <https://www.zabbix.com/documentation/4.0/pt/manual/introduction/features>. Acesso em: 05 jun. 2018.

ZEPPELIN, A. *Apache Zeppelin 0.8.0*. [S.l.], 2018. Disponível em: <https://zeppelin.apache.org/docs/0.8.0/>. Acesso em: 17 abr. 2018.

ZIGBEE. Zigbee alliance. specification. 2017. Disponível em: <http://www.zigbee.org/non-menu-pages/zigbee-pro-download>. Acesso em: 13 ago. 2017.

LISTA DE ABREVIATURA E SIGLAS

AON – *Active Optical Networking*

API – *Application Programming Interface*

APL – *Application Layer*

BDA – *Big Data Analytics*

CSS – *Chirp Spread Spectrum*

DFMC – *Data Fusion Microservice Containers*

DL – *Deep Learning*

FMIS – *Farm Management Information Systems*

GB – *Gigabytes*

GIS – *Geographic Information System*

GPRS – *General Packet Radio Service*

GPS – *Global Positioning System*

GSM – *Global System for Mobile Communications*

HTTP – *HyperText Transfer Protocol*

IDE – *Integrated Development Environment*

IEEE – *Institute of Electrical and Electronic Engineers*

IOT – *Internet of Things*

ISM – *Industrial Sientific and Medical*

IaaS – *Infrastructure as a Service*

KVM – *Kernel Virtual Machine*

LORA – *Long Range Radio*

LPWANs – *Low Power Wide Area Networks*

LRWPAN – *Low-Rate Wireless Personal Area Networks*

M2M – *Machine-to-Machine*

MAC – *Medium Access Control*

MB – *Megabyte*

MIME – *Multipurpose Internet Mail Extensions*

MQTT – *Message Queuing Telemetry Transport*

NDVI – *(Normalized Difference Vegetation*

NTC – *Negative temperature coefficient*

NWK – *Network Layer*

OGC – *Open Geospatial Consortium*

PHY – *Physical Layer*

PON – *Passive Optical Network*

PTC – *Positive temperature coefficient*

PaaS – *Platform as a Service*

QoS – *Quality of service*

RFID – *Radio-Frequency IDentification*

RTT – *Round Trip Time*

SDK – *Software Development Kit*

SOA – *Service Oriented Architecture*

SWE – *Sensor Web Enablement*

SaaS – *Software as a Service*

TIC – *Tecnologias da Informação e Comunicacão*

UNB – *Ultra Narrow Band*

VMs – *Virtual Machines*

WLAN – *Wireless Local Area Network*

WPAN – *Wireless Personal Area Network*

WSN – *Wireless Sensor Networks*

ha – *hectare*