André Filipe Gomes dos Santos

Licenciado em Ciências da Engenharia

Eletrotécnica e de Computadores

Smartphone como *gateway* móvel para a *Internet* das Coisas em ambientes sujeitos a atrasos e a interrupções

Dissertação para obtenção do Grau de Mestre em

Engenharia Eletrotécnica e de Computadores

Orientador: Prof. Doutor Pedro Miguel Negrão Maló, Professor Auxiliar, FCT-UNL

Júri:

Presidente: Prof. Doutor Paulo da Costa Luís da Fonseca Pinto, Professor Catedrático, FCT-UNL

Arguentes: Prof. Doutor Tiago Oliveira Machado de Figueiredo Cardoso, Professor Auxiliar, FCT-UNL

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

março de 2018

***Smartphone* como *gateway* móvel para a *Internet* das Coisas em ambientes sujeitos a atrasos e a interrupções**

Dedicado aos meus pais, irmão, namorada e amigos

Agradecimentos

Em primeiro lugar, gostaria de agradecer ao Prof. Pedro Maló, o meu orientador, pela oportunidade de poder trabalhar numa área que, para mim, tem tanto de potencial futuro como de interessante.

Ao Eng. Edgar Silva, apesar do pouco tempo de trabalho em conjunto, a sua disponibilidade e as discussões que, certamente, me ajudarão a ser um melhor profissional.

Ao Tomás, meu amigo de infância, por estar sempre presente ao longo dos anos, apesar da minha maior ausência no decorrer desta dissertação.

Ao Flávio e ao Ricardo, meus grandes amigos e companheiros de curso, o companheirismo, a ajuda e a presença nos bons e maus momentos ao longo deste curso.

Aos pais da minha namorada, Carlos e Maria José, por sempre me terem recebido em sua casa como seu filho.

À minha tia e meus primos, Cláudia, Inês e Francisco, a alegria e a paciência que sempre me transmitiram no decorrer desta dissertação.

Ao meu irmão, Tomás, a paciência e a compreensão que sempre demonstrou pela minha indisponibilidade ao longo deste percurso.

À minha mãe e ao meu pai, Anabela e João, a educação, o amor e os valores que sempre me transmitiram, bem como a possibilidade que me deram de me formar e ser a pessoa que sou hoje.

À Andreia, minha namorada e minha companheira de dissertação, o apoio nos momentos de maior desmotivação e, acima de tudo, o amor e o carinho que fazem de mim uma pessoa melhor e mais feliz.

Resumo

A *Internet* das Coisas (IdC) possui algumas limitações em termos de conectividade, nomeadamente em meios que possuam dificuldades de acesso à rede de *Internet*, como os que estão sujeitos a atrasos e a interrupções. Neste contexto, o *smartphone* é usado como elemento estrutural ao nível da comunicação na IdC, sendo que vários autores propõem a sua utilização como *gateway* e como elemento central de Redes Tolerantes aos Atrasos e às Interrupções.

O problema identificado pode ser dividido em três desafios: funcionamento do *smartphone* como *gateway* para mediar a passagem dos dados; armazenamento local dos dados, em casos de inexistência de conectividade; e sincronização dos dados para coincidirem com os da *Cloud*. Cada desafio corresponde a um requisito: armazenamento de dados na *Cloud*; ligação entre a camada sensorial e a *Cloud*; e utilização de dispositivos de prototipagem para a IdC.

Para validar a abordagem, foi desenvolvido um protótipo cuja implementação se baseia na recolha local e no envio dos dados para a *Cloud*, aquando da existência de ligação à *Internet*, funcionando o *smartphone* como *gateway* para IdC. O protótipo assenta nas temáticas das Plataformas de *Cloud* para a IdC, dos *Middlewares* Orientados a Mensagens e dos Dispositivos para a IdC.

Palavras-chave: *Internet* das Coisas, *Smartphone*, *Gateway*, Plataforma de *Cloud*, *Middleware* Orientado a Mensagens, Dispositivo, Armazenamento de Dados

Abstract

The Internet of Things (IoT) has some limitations in terms of connectivity, especially in environments that have difficulties accessing the Internet, such as those that are subject to delays and interruptions. In this context, the smartphone is utilised as a structural element at communication level in the IoT and several authors propose its use as a central element of Delay/Disruption-Tolerant Networks.

The identified problem can be divided into three challenges: functioning of the smartphone as gateway to mediate the passage of data; local storage of data, in cases of lack of connectivity; and synchronize the data to match those located at Cloud. Each challenge corresponds to one requirement: data storage in the Cloud; connection between the sensing layer and the Cloud; and use of prototyping devices for the IoT.

To validate the approach, a prototype was developed and its implementation is based on the local collection and the sending of data to the Cloud, when there is an Internet connection, acting the smartphone as an IoT gateway. The prototype is based on the Cloud Platforms for IoT, Message Oriented Middleware, and IoT Devices.

Keywords: Internet of Things, Smartphone, Gateway, Cloud Platform, Message-Oriented Middleware, Device, Data Storage

Índice Geral

Índice de Tabelas

Índice de Figuras

Lista de Acrónimos

6LoWPAN – *Internet Protocol v6 over Low power Wireless Personal Area Networks*

AES – *Advanced Encryption Standard*

AMQP – *Advanced Message Queuing Protocol*

API – *Application Programming Interface*

BLE – *Bluetooth Low Energy*

CoAP – *Constrained Application Protocol*

CSV – *Comma-Separated Values*

DTN – *Delay/Disruption-Tolerant Network*

DTNRG – DTN *Research Group*

ETSI – *European Telecommunication Standards Institute*

FCM – *Firebase Cloud Messaging*

FIFO – *First-In First-Out*

FM – Fila de Mensagens

GCM – *Google Cloud Messaging*

GFS – *Google File System*

GPIO – *General Purpose Input/Ouput*

H2H – *Human-to-Human*

HDFS – *Hadoop Distributed File System*

HDMI – *High-Definition Multimedia Interface*

HH – Humano-Humano

HTML – *HyperText Markup Language*

HTTP – *HyperText Transfer Protocol*

HTTPS – HTTP *Secure*

I2C – *Inter-Integrated Circuit*

IaaS – *Infrastructure as a Service*

IdC – *Internet* das Coisas

IDE – *Integrated Development Environment*

IEEE – *Institute of Electrical and Electronics Engineers*

IM – Intermediário de Mensagens

IoT – *Internet of Things*

IP – *Internet Protocol*

IPN – *Interplanetary Internet*

IPNRG – IPN *Research Group*

IRF – Identificação por Radiofrequência

ITU – *International Telecommunication Union*

JDBC – *Java Database Connectivity*

JS – JavaScript

JSON – JS *Object Notation*

LDR – *Light Dependent Resistor*

LED – *Light Emitting Diode*

LWM2M – *Lightweight Machine-to-Machine*

M2M – *Machine-to-Machine*

MM – Máquina-Máquina

MOM – *Middleware* Orientado a Mensagens

MQTT – *Message Queuing Telemetry Transport*

MR – *Map/Reduce*

MSSQL – *Microsoft Structured Query Language*

NFC – *Near Field Communication*

NoSQL – *Not only Structured Query Language*

NPM – *Node Package Manager*

ODBC – *Open Database Connectivity*

P2P – *Peer-to-Peer*

PaaS – *Platform as a Service*

PIR – *Passive InfraRed*

PM – Passagem de Mensagens

PWM – *Pulse Width Modulation*

RAM – *Random Access Memory*

REST – *Representational state transfer*

RFID – *Radio-Frequency Identification*

ROM – *Read-only Memory*

RPL – *Routing Protocol for Low Power and Lossy Networks*

RSA – *Rivest Shamir Adleman*

RSSF – Rede de Sensores Sem Fios

RTAI – Redes Tolerantes aos Atrasos e às Interrupções

SaaS – *Software as a Service*

SASL – *Simple Authentication and Secure Layer*

SD – *Secure Digital*

SDK – *Software Development Kit*

SMS – *Short Message Service*

SO – Sistema Operativo

SPI – *Serial Peripheral Interface*

SQL – *Structured Query Language*

SSL – *Secure Sockets Layer*

SSO – *Single Sign-On*

STOMP – *Simple Text Oriented Messaging Protocol*

TCP – *Transmission Control Protocol*

TTCN-3 – *Testing and Test Control Notation Version* 3

UART – *Universal Asynchronous Receiver/Transmitter*

UDP – *User Datagram Protocol*

uIP – *micro* IP

USB – *Universal Serial Bus*

uXMPP – *micro eXtensible Messaging and Presence Protocol*

WSN – *Wireless Sensor Networks*

XML – *eXtensible Markup Language*

XMPP – e*Xtensible Messaging and Presence Protocol*

YAML – *YAML Ain't Markup Language*

YARN – *Yet Another Resource Negotiator*

1 Introdução

O termo *Internet* das Coisas (IdC) – em inglês, *Internet of Things* (IoT) – foi abordado, pela primeira vez, em 1999, por Ashton (2009) como sendo um sistema que liga o mundo real e a *Internet* por meio de uma rede de sensores, de forma a minimizar a complexidade na troca de informação. Por sua vez, Gubbi, Buyya, Marusic e Palaniswami (2013) abordam a IdC como sendo a terceira revolução das tecnologias de informação, depois do aparecimento das páginas *web* estáticas (www) e interativas (*web* 2.0). O conceito de IdC, assenta na comunicação no interior de um contexto que conecta a sociedade, o ambiente e o próprio utilizador e possui três importantes características: a identificação única e global das Coisas, a sua acessibilidade e a ligação à *Internet* (Ashton, 2009; Corcoran, 2016; Dupont, Sheikhalishahi, Biswas & Bures, 2017; Gubbi et al., 2013; Minerva, Biru & Rotondi, 2015; Santucci & Lange, 2008; Wu, Lu, Ling, Sun & Du, 2010).

A comunicação é efetuada por interfaces inteligentes que se encontram em Coisas com identidade própria, i.e., que possuem um endereço único, e formam uma rede, de cariz global, que funciona de acordo com protocolos de comunicação padronizados. Por outro lado, o conceito de IdC abrange um modelo de comunicação constituído por Redes Móveis, por Atuadores e por Redes de Sensores Sem Fios (RSSFs) – em inglês, *Wireless Sensor Networks* (WSN) –, em que a comunicação entre as Coisas é efetuada por meio da Identificação por Radiofrequência (IRF) – em inglês, *Radio-Frequency Identification* (RFID) –, *chips*, códigos de barras ou *Near Field Communication* (NFC) (Atzori, Iera & Morabito, 2010; Dores, Reis & Lopes, 2014; Khan, Khan, Zaheer & Khan, 2012; S. Li, Xu & Zhao, 2015; Ma, 2011; Misra, Kumar, Agarwal & Agarwal, 2016; Said & Masud, 2013; Santucci & Lange, 2008; K. J. Singh & Kapoor, 2017)

Para além disso, de acordo com Geng (2017), Pereira e Aguiar (2014) e K. J. Singh e Kapoor (2017), os *smartphones* podem ser vistos como elementos fundamentais para o futuro da IdC e das comunicações móveis Máquina-Máquina (MM) – em inglês, *Machine-to-Machine* (M2M). Isto deve-se, essencialmente, ao facto de os *smartphones* possuírem bastantes sensores e opções de conectividade e de serem sobejamente utilizados (Geng, 2017; Pereira & Aguiar, 2014; K. J. Singh & Kapoor, 2017).

Desta forma, Khalid et al. (2013) e Luzuriaga et al. (2015) abordam a utilização de um *Middleware* Orientado a Mensagens (MOM) – em inglês, *Message-Oriented Middleware* – no paradigma de comunicação MM e Meng, Wu, Muvianto e Gray (2017) indicam que a comunicação entre nós de uma rede para a IdC deve ser gerida por um mecanismo de mensagens. Assim, a utilização de um MOM ou de um Intermediário de Mensagens (IM) torna-se fundamental (Khalid et al., 2013; Luzuriaga et al., 2015; Meng et al., 2017).

Por sua vez, as RSSFs possuem, na sua maioria, um *gateway* capaz de traduzir eficazmente protocolos de *Internet* baseados no *Transmission Control Protocol* (TCP)/*Internet Protocol* (IP) para protocolos específicos usados no âmbito deste tipo de redes. Alguns desses *gateways* permitem armazenar, temporariamente, dados provenientes dos sensores e, devido à falta de padronização na sua elaboração, revelam uma interoperabilidade reduzida. Desta forma, a comunicação entre as Coisas pode ser vista como uma comunicação MM ou como uma comunicação Humano-Humano (HH) – em inglês, *Human-to-Human* (H2H) (Aggarwal, Ashish & Sheth, 2014; Fall & Farrell, 2008; Minoli, 2013).

Em termos de aplicações, Vermesan et al. (2014) referem que a IdC está presente nos domínios das cidades inteligentes (*Smart Cities*), das redes inteligentes de energia (*Smart Energy and Smart Grids*), da mobilidade e dos transportes inteligentes (*Smart Mobility and Smart Transport*), da domótica (*Smart Home, Smart Buildings and Infrastructures*), das fábricas inteligentes (*Smart Factory and Smart Manufacturing*), da saúde inteligente (*Smart Health*) e da logística inteligente e do retalho (*Smart Logistics and Retail*).

1.1 Motivação

Hoje em dia, a falta de acesso à Internet em diversos locais é um problema devido à quantidade de dados que potencialmente podem ser recolhidos para análise. Assim, vários estudos abordam a importância da monitorização de parâmetros em diferentes contextos:

qualidade da água em florestas, densidade populacional em locais fechados, estado de saúde de um paciente e estado das condições das vias rodoviárias (De La Piedra, Benitez-Capistros, Dominguez & Touhafi, 2013; K. Li et al., 2018; Madukwe, Ezika & Iloanuse, 2017; Mamum, Puspo & Das, 2017).

Neste âmbito, são vários os estudos que apontam que a recolha de dados, é, na sua maioria, efetuada através de nós sensores que operam dentro de uma rede fechada no local de operação, sendo necessário que todos os sensores possuam capacidades de conexão por meio de protocolos que, usualmente, não são encontrados nos *smartphones* comuns. Como resultado, torna-se imprescindível a existência de um *gateway* que estabeleça a ponte com outro tipo de rede ou com outro ponto que trabalhe com o mesmo tipo de protocolo, o que, num local de inexistência de rede, pode ser de difícil execução. Por conseguinte, vários autores abordam a utilização do *smartphone* como *gateway* para estabelecer uma ligação à *Internet* (De La Piedra et al., 2013; Pereira, Pinto et al., 2016; Pereira, Rodrigues et al., 2016, Vidanagama, Arai & Ogishi, 2015; J. Zhang, Shan, Hu & Yang, 2012).

No entanto, apesar de o *smartphone* possuir várias tecnologias de interface de redes, este está sujeito a intermitências na sua conectividade, pelo que se torna essencial o uso de Redes Tolerantes aos Atrasos e às Interrupções (RTAI) – em inglês, *Delay/Disruption-Tolerant Network* (DTN) –, uma vez que as características que estas apresentam encaixam, perfeitamente, nos problemas de comunicação que os *smartphones* revelam (Caini, Cornice, Firrincieli, Livini & Lacamera, 2010; Fall & Farrell, 2008; Talipov, Chon & Cha, 2013).

As RTAI surgiram, em 2002, como resultado do insucesso do desenvolvimento do projeto da *Interplanetary Internet* (IPN), o que levou a que se testasse a arquitetura desenvolvida até ao momento em outras áreas que possam estar sujeitas a atrasos e interrupções. Desta forma, através de uma publicação efetuada pelo IPN *Research Group* (IPNRG), que mais tarde veio a tornar-se no DTN *Research Group* (DTNRG), Kevin Fall tornou a arquitetura desenvolvida para a IPN a arquitetura de referência para as RTAI. Assim, para além da conectividade intermitente presente nos *smartphones* e nas RTAI, estas operam em contextos em que a conexão ponta-a-ponta nem sempre é exequível e visam solucionar problemas relacionados com longos atrasos, perda de pacotes e erros (Fall & Farrell, 2008; McMahon & Farrell, 2009).

Tendo em conta os problemas que este tipo de redes se propõe a solucionar, Fall e Farrell (2008) afirmam que as áreas da gestão do congestionamento da rede, da capacidade de multidifusão pelos canais e do encaminhamento dos dados devem ser tidas como as mais importantes para o desenvolvimento desta tecnologia. Seguindo a mesma

linha de pensamento, McMahon e Farrell (2009) indicam que o foco passa, igualmente, pela definição dos princípios relativos à arquitetura em si e aos seus protocolos, de forma a que a rede seja capaz de ser interoperável e que funcione em ambientes extremos onde a conectividade intermitente pode ser uma realidade.

Segundo Fall e Farrell (2008), as RTAI revelam uma estreita relação com as RSSFs, uma vez que a sua arquitetura disponibiliza a capacidade de lidar com a reduzida interoperabilidade apresentada pelos nós que constituem as RSSFs, por meio de técnicas como a disposição em camadas, o encapsulamento e o armazenamento de dados. Dado que a duração necessária para o armazenamento de dados é incerta e que estes podem ser transmitidos para vários destinatários, a capacidade de armazenamento da arquitetura em nós intermédios apresenta-se como essencial (Caini et al., 2010; Fall & Farrell, 2008; McMahon & Farrell, 2009).

Para além disso, a arquitetura exibe a capacidade de lidar com casos extremos de descontinuidade da conectividade da rede, nomeadamente, aplicações baseadas na técnica de *data mule*, que consiste na recolha de dados dos vários nós sensores presentes num determinado local por determinadas pessoas ou veículos que, posteriormente, estabelecem uma conexão à rede (Anastasi, Conti & Francesco, 2008; Caini et al., 2010; Wong, Wan & Ang, 2016).

Desta forma, Distl e Legendre (2015) encaram as RTAI como redes oportunistas que podem ser formadas através da interação que ocorre entre dois ou mais dispositivos móveis que se encontram num determinado espaço, limitado por um alcance estipulado, e que permutam dados entre si. Adicionalmente, Guo et al. (2007) apresentam uma solução para que as áreas rurais ou pouco habitadas tenham acesso à *Internet* por intermédio deste tipo de redes; Hossmann et al. (2011) utilizam redes oportunistas formadas por *smartphones* em caso da inexistência de uma infraestrutura devido a desastres naturais; Park e Heidemann (2011) sugerem a utilização do *smartphone* como *data mule*; e Wong et al. (2016) abordam, no contexto da IdC, a utilização desta técnica com vista a solucionar as dificuldades existentes em locais de difícil acesso à rede, tais como em ambientes de exploração mineira e agrícola.

No âmbito da IdC, apesar de não existir entendimento sobre a arquitetura adotada, a que reúne maior consenso é a de três camadas. A referida arquitetura é constituída pelas camadas de perceção, de rede e de aplicação, tal como se pode verificar por meio da figura 1.1 (Al-Qaseemi, Almulhim, Almulhim & Chaudhry, 2016; N., Kumar & Banu, 2017; Zhong, Zhu & Huang, 2017).

A camada de perceção – também conhecida como camada de perceção e controlo, de dispositivo, física ou sensorial – possui a função de recolher dados de dispositivos e

sensores, de efetuar o seu pré-processamento e de os enviar para a camada de rede. Esta camada é ainda responsável por receber ações de controlo da camada de rede a fim de as executar sobre os dispositivos que controla. O *hardware* utilizado nesta etapa comunica com base em diferentes protocolos como o 802.15.4 definido pelo *Institute of Electrical and Electronics Engineers* (IEEE), o Bluetooth e o RFID (Al-Qaseemi et al., 2016; Datta & Sharma, 2017; N. et al., 2017; Suganuma, Oide, Kitagami, Sugawara & Shiratori, 2018; Tyagi & Kumar, 2017; Zhong et al., 2017).

Por sua vez, a camada de rede – igualmente nomeada de camada de transmissão ou de *gateway* – tem o objetivo de encaminhar os dados provenientes da camada física para a de aplicação, de forma segura, confiável e eficiente por meio de redes com ou sem fios. As redes podem ser remotas – móveis ou de satélite – ou de curto-alcance – Bluetooth, Wi-Fi, Ad-hoc, Mesh, Industrial Bus ou ZigBee (Al-Qaseemi et al., 2016; N. et al., 2017; Tyagi & Kumar, 2017; Zhong et al., 2017).

Por último, a camada de aplicação disponibiliza vários serviços – como o de armazenamento de dados –, ferramentas de desenvolvimento e aplicações direcionadas para as casas, a agricultura e a gestão de trânsito inteligentes e os serviços de saúde e que dependem de tecnologias como a *Cloud*, as bases de dados e os *middlewares* (N. et al., 2017; Suganuma et al., 2018; Tyagi & Kumar, 2017; Zhong et al., 2017).

Figura 1.1: Arquitetura de três camadas para a IdC – adaptado de Zhong et al. (2017).

1.2 Identificação do Problema

De acordo com Zhu, Wang, Chen, Liu e Qin (2010), o *gateway* desempenha uma ação de fundamental relevo no âmbito das aplicações para a IdC, uma vez que permite realizar a ponte entre as RSSFs e as redes de comunicação móvel. Por sua vez, Li et al. (2013) e Vidanagama et al. (2015) referem a existência de aplicações de comunicação

MM que são usadas como *gateways* por *smartphones* para estabelecer uma ligação à *Internet*. Para além disso, Pereira, Pinto et al. (2016) e Pereira, Rodrigues et al. (2016) propõem a implementação de um *gateway* MM num *smartphone* que utiliza o Sistema Operativo (SO) Android e J. Zhang et al. (2012) revelam o uso do *smartphone* como nó sensor e *gateway*. Desta forma, torna-se interessante a utilização do *smartphone* para a recolha de dados de vários sensores localizados em florestas isoladas e que não possuam qualquer acesso à rede. O *smartphone*, por conseguinte, possui capacidade de armazenamento próprio e, aquando da sua ligação à rede, o papel de proceder ao envio da totalidade dos dados recolhidos para a *Cloud*.

A figura 1.2 representa o conceito genérico da utilização do *smartphone* como *gateway* sugerido por Li et al. (2013), Pereira, Pinto, et al. (2016), Pereira, Rodrigues, et al. (2016) e Vidanagama et al. (2015). De uma forma geral, cada dispositivo comunica com o *smartphone*, a fim de lhe transmitir os dados das grandezas medidas. O *smartphone*, por sua vez, funciona como *gateway* e disponibiliza uma aplicação. Ao desempenhar o papel de *gateway*, estabelece uma conexão com a *Cloud*, na qual são colocados os dados recolhidos. Por outro lado, quando em modo de aplicação, o *smartphone* disponibiliza o acesso e a visualização dos dados dos sensores públicos, bem como os do utilizador.

Figura 1.2: Transmissão dos dados dos sensores para a *Cloud* através do *smartphone*.

Deste modo, o problema identificado pode ser dividido em três importantes desafios: *gateway*, armazenamento local e sincronização. O primeiro desafio está relacionado com a receção no *smartphone* e respetivo envio dos seus dados e de outros utilizadores para a *Cloud*; o segundo com o armazenamento dos dados em situações em que a ligação à rede não seja possível; e o último com a sincronização entre a aplicação presente no *smartphone* e a *Cloud*.

1.2.1 Desafio 1: *Gateway*

De acordo com Hao, Agrawal, Aranya e Ungureanu (2013), um aspeto crucial no desenvolvimento de aplicações móveis e serviços é a capacidade de estes disponibilizarem a sincronização dos dados do utilizador com a *Cloud* e/ou com dispositivos de outros utilizadores. Na mesma linha de pensamento, Cui, Lai, Wang e Dai (2017) apontam que modificações nos dados dos dispositivos devem ser, numa primeira fase, sincronizados com a *Cloud* e, posteriormente, retransmitidos para outros dispositivos. Conforme Jemel, Msahli e Serhrouchni (2015), os serviços de armazenamento da *Cloud* são capazes de sincronizar os dados do utilizador com outros dispositivos, sejam estes do mesmo utilizador ou de outros.

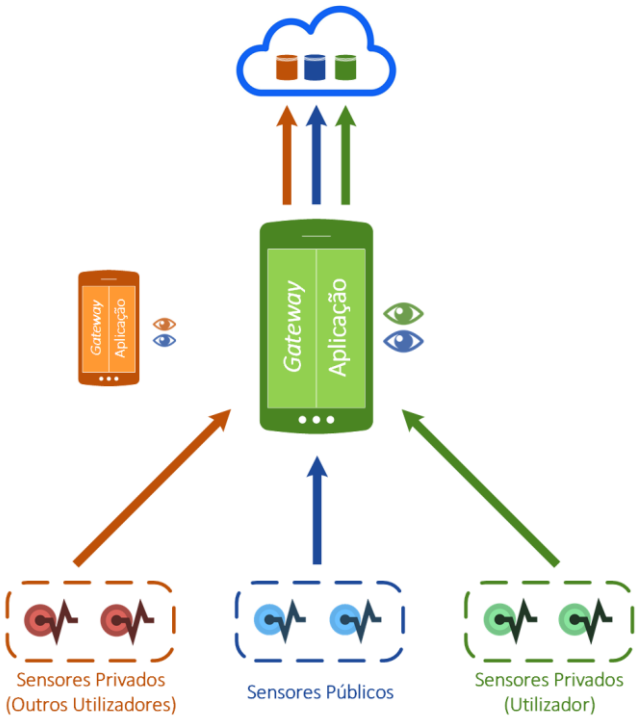

Figura 1.3: Envio dos dados do próprio e de outros utilizadores para a *Cloud*.

Deste modo, o primeiro desafio proposto para o desenvolvimento desta dissertação passa pela receção de dados emitidos pelos dispositivos de outros utilizadores no *smartphone*, sendo estes retransmitidos, posteriormente, para a *Cloud*, não sendo possível aceder-lhes. Assim, este desafio está relacionado com o conceito de *gateway*, uma vez que o *smartphone* estabelece a ponte com outro tipo de rede, ao funcionar como mediador para a passagem dos dados dos dispositivos para a *Cloud* (De La Piedra et al., 2013). Esta situação está representada na figura 1.3, onde as ligações de cores diferentes representam os vários utilizadores, sendo os dados enviados para a *Cloud* por meio do *smartphone* do utilizador representado pela cor verde.

1.2.2 Desafio 2: Armazenamento Local

As ligações à rede que o *smartphone* é capaz de disponibilizar nem sempre são possíveis de estabelecer, seja por falta de acesso ou por opção do utilizador. Assim, caso não seja possível enviar os dados no momento, estes necessitam de ser armazenados no *smartphone* – tal como se pode aferir através da situação representada na figura 1.4 –, com vista a posterior reencaminhamento para a *Cloud*. Desta forma, Park e Heidemann (2011) sugerem a utilização do *smartphone* como *data mule* e Caini et al. (2010), Fall e Farrell (2008) e Talipov et al. (2013) afirmam que a utilização de RTAI se torna fundamental, uma vez que as suas características cobrem os problemas de comunicação que os *smartphones* revelam. Para além disso, o utilizador garante, deste modo, uma cópia de segurança dos dados recolhidos.

Figura 1.4: Falha no acesso à rede e capacidade de armazenamento do *smartphone*.

1.2.3 Desafio 3: Sincronização

Devido ao aumento do número das aplicações sensitivas, de partilha e de colaboração, a quantidade de dados gerada pelos dispositivos móveis tem, de igual forma, vindo a aumentar. Para além disso, a existência de um grande número de dispositivos leva a que seja necessário sincronizar os dados dos utilizadores, independentemente do local, do tempo e da qualidade de conetividade que estes possuam. Desta forma, um aspeto importante do desenvolvimento deste tipo de aplicações é a semântica do armazenamento de dados e da sua sincronização. Mais concretamente, o processo de sincronização permite que o utilizador atualize, de forma automática, os seus dados na *Cloud*. No entanto, caso a sincronização não possa ser efetuada no momento, o utilizador pode gerir os seus dados de forma *offline* e, posteriormente, quando esta for possível, possuir os dados em todos os seus dispositivos (Cui et al., 2017; Hao et al., 2013; Jemel et al., 2015).

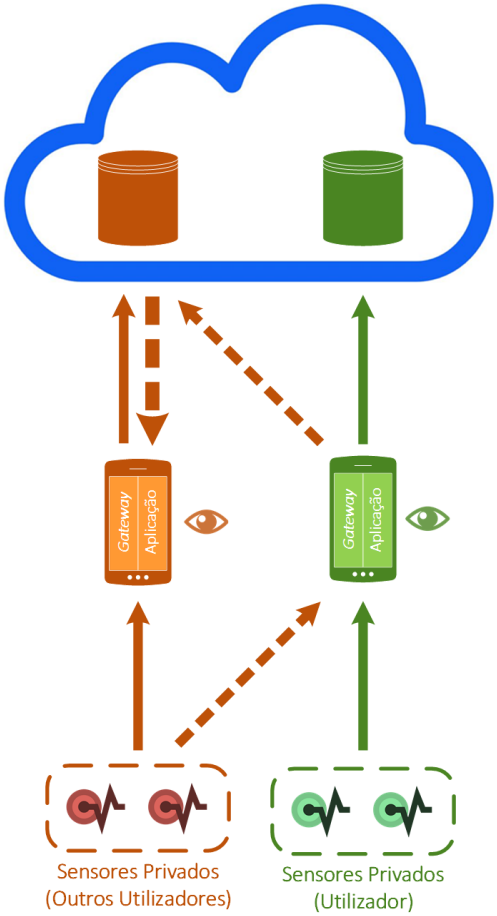

Figura 1.5: Sincronização dos dados entre a aplicação e a *Cloud*.

Consequentemente, um dos aspetos importantes para o desenvolvimento desta dissertação passa pela sincronização entre a aplicação disponibilizada pelo *smartphone* e a *Cloud*. Esta situação está representada na figura 1.5, onde o *smartphone* verde recebe os dados de vários utilizadores e os envia para a *Cloud*, sendo estes, mais tarde, reencaminhados para os respetivos *smartphones*.

1.2.4 Síntese

Em suma, a figura 1.6 ilustra o panorama global do problema identificado, onde se pode verificar que o *smartphone* é o elemento central, que funciona como intermediário no envio dos dados, desde a sua recolha – ao nível dos sensores – até ao seu armazenamento na *Cloud*. Para além disso, este possui capacidade de armazenamento dos dados, estando as suas permissões de acesso bem definidas, e respetiva visualização em tempo real.

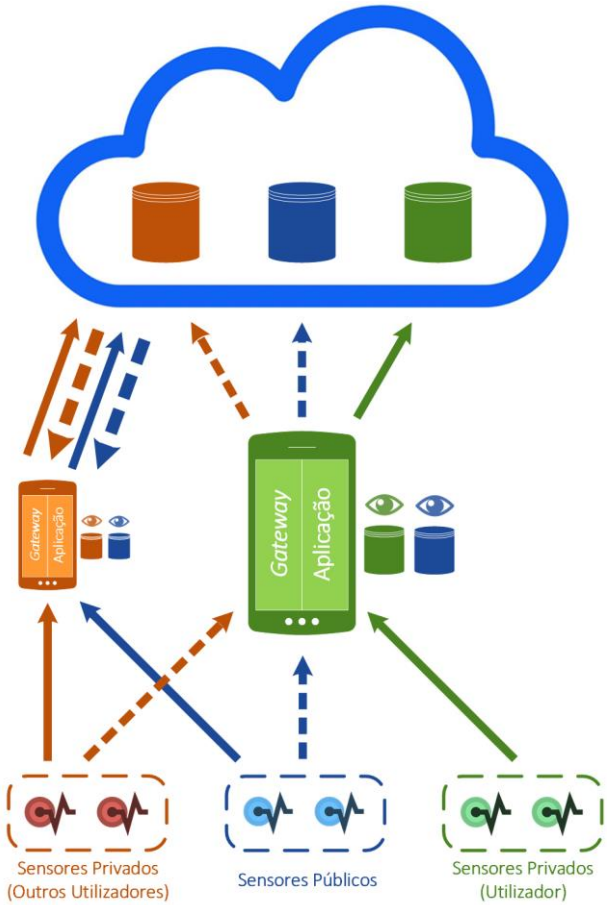

Figura 1.6: Panorama global do problema identificado.

1.3 Requisitos do Sistema

Depois do problema ter sido decomposto em três desafios, foram identificados três requisitos para o sistema a desenvolver: armazenamento de dados na *Cloud*; ligação entre a camada sensorial e a *Cloud*; e utilização de dispositivos de prototipagem de livre-acesso para a IdC.

O contexto em que o desafio 1 está inserido relaciona-se com os três requisitos assinalados. Assim, o armazenamento de dados na *Cloud* advém da necessidade em colocar os dados dos utilizadores produzidos pelos sensores num local de armazenamento externo. Já a ligação entre a camada sensorial e a *Cloud* funciona como um mediador para a passagem dos dados dos dispositivos para o ambiente de *Cloud*. Por último, a utilização de dispositivos de prototipagem de livre-acesso para a IdC surge devido às características que estes oferecem em termos de desenvolvimento rápido, fácil e modular.

O desafio 2 encontra-se apenas associado à ligação entre a camada sensorial e a *Cloud*, uma vez que é neste âmbito que os dados são armazenados localmente.

No que diz respeito ao desafio 3, este apresenta-se dependente do armazenamento de dados na *Cloud* e da ligação entre a camada sensorial e a *Cloud*. Deste modo, para efetuar a sincronização com os dados armazenados localmente, este desafio necessita de um mecanismo de armazenamento externo, que, neste caso, possui o papel do envio dos dados. Por sua vez, o referido desafio encontra-se igualmente relacionado com a ligação entre a camada sensorial e a *Cloud*, uma vez que necessita de um mediador que seja capaz de estabelecer esta conexão.

A figura 1.7 apresenta, graficamente, as associações efetuadas entre os desafios do problema identificado e os requisitos do sistema a implementar.

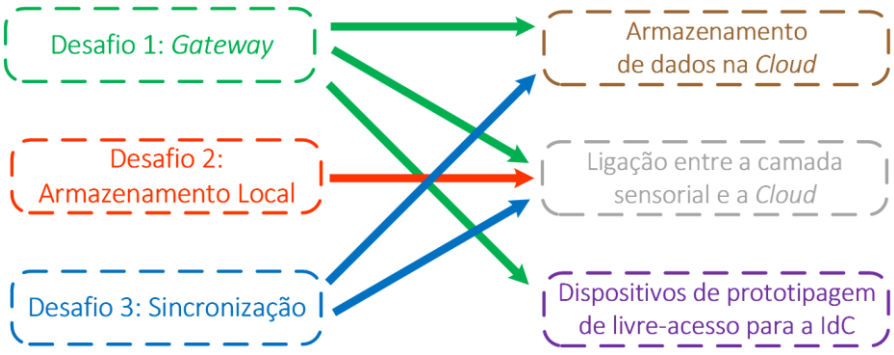

Figura 1.7: Associação dos desafios do problema aos requisitos do sistema.

1.4 Metodologia de Trabalho

A metodologia de trabalho adotada para a elaboração do problema identificado para esta dissertação baseia-se nas etapas do método de engenharia, esquematizado na figura 1.8.

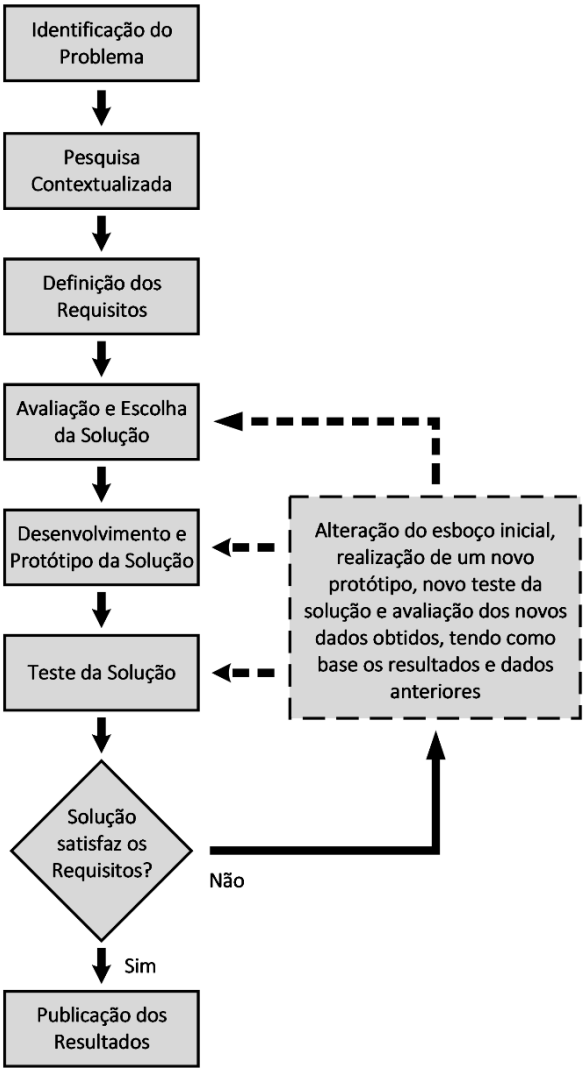

Figura 1.8: Método de Engenharia – adaptado de "Comparing the Engineering Design Process and the Scientific Method" (2017).

Inicialmente, procede-se à identificação do problema e contextualiza-se a sua área de estudo. Posteriormente, elabora-se uma revisão de literatura relativamente à tecnologia existente sobre o foco do problema. De seguida, definem-se os requisitos pretendidos para o sistema a realizar. Finalizada a parte inicial descrita, procede-se à avaliação e à escolha

da solução, sendo o passo seguinte o seu desenvolvimento e consequente realização de um protótipo. Uma vez concluída a etapa anterior, procede-se à fase de testes. Caso a solução satisfaça totalmente os requisitos, o problema pode ser considerado resolvido e os seus resultados são publicados. Caso contrário, ajusta-se o esboço inicial, elabora-se um novo protótipo da solução e respetivo teste e volta-se a verificar se a solução cumpre os requisitos identificados.

1.5 Organização do Documento

O presente documento está estruturado em seis capítulos. O primeiro é o atual e disponibiliza uma breve introdução e contextualização do domínio do problema, para além da sua identificação, dos requisitos propostos para o sistema a desenvolver e da metodologia de trabalho selecionada.

O capítulo 2 revela o Estado da Arte, i.e., um estudo de investigação relativo aos tópicos fundamentais para análise no âmbito do problema identificado. Com base nas tecnologias existentes utilizadas na comunidade científica, conhecidas por meio da análise de artigos de jornais científicos, de livros e dos *websites* das respetivas tecnologias, é efetuada uma visão geral relativa aos conceitos específicos do domínio do problema.

O terceiro capítulo expõe, de forma detalhada, o desenvolvimento e o protótipo efetuados ao longo da dissertação, estando presentes a arquitetura e a especificação da solução, bem como o ambiente de trabalho, a configuração e a implementação efetuadas para cada um dos elementos de estudo.

O capítulo 4 aborda a metodologia de testes utilizada e contém, para cada funcionalidade do sistema, a proposta de um teste e respetiva validação.

O quinto capítulo, referente à Conclusão, revela, sumariamente, o que foi efetuado em cada capítulo, sendo enfatizados os estudos encontrados em torno da identificação do problema, os desafios propostos para o desenvolvimento da presente dissertação, o desenvolvimento e o protótipo e o teste e a validação da solução implementada, bem como o trabalho futuro que se pretende efetuar como seguimento do projeto realizado.

Para finalizar, existe um capítulo relativo às Referências Bibliográficas consultadas no desenrolar do desenvolvimento desta dissertação.

2 Estado da Arte

Este capítulo revela a revisão de literatura efetuada no que respeita à tecnologia relevante para a presente dissertação e encontra-se dividido em três secções, que correspondem, como se pode verificar por meio da figura 2.1, aos requisitos propostos para o sistema a desenvolver: Dispositivos para a IdC, *Middlewares* Orientados a Mensagens e Plataformas de *Cloud* para a IdC. Cada secção é estruturada pela contextualização do seu tema, pelos elementos de estudo selecionados e por uma síntese comparativa desses mesmos elementos, sendo enfatizados os aspetos pertinentes para o desenvolvimento desta dissertação.

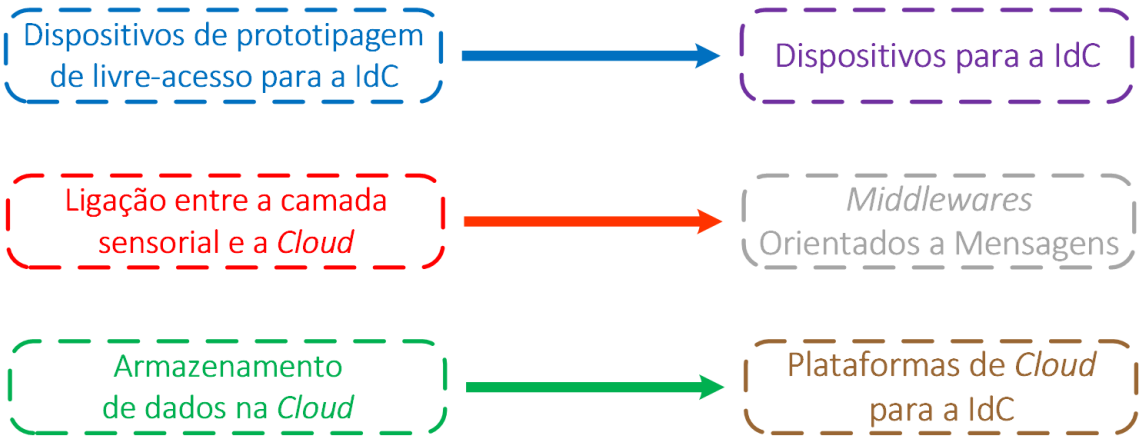

Figura 2.1: Associação dos requisitos do sistema às tecnologias estudadas.

2.1 Dispositivos para a IdC

Com o advento da IdC, a ligação entre objetos comuns e a *Internet* torna-se possível por meio da utilização de dispositivos – como microssensores e plataformas embebidas – geradores de dados e com limitações temporais ao nível da bateria (Polianytsia, Starkova & Herasymenko, 2016; Premsankar, Di Francesco, & Taleb, 2018; Ray, 2016; Zachariah et al., 2015).

Um dispositivo para a IdC, também referido como objeto inteligente, é definido como sendo um elemento de um sistema que lhe confere capacidades sensitivas, de atuação, de controlo e de monitorização. Para além disso, este deve apresentar dimensões reduzidas e ser energeticamente independente. Os dispositivos deste tipo possuem a aptidão para recolher dados de outros e para os trocar entre si, sendo estes tratados localmente, na *Cloud* ou em ambos. Para efetuar estas operações, o dispositivo deve ser capaz de interagir com sensores, com a *Internet* e com mecanismos de armazenamento de dados, através de comunicação com e sem fios (Alioto, 2017; Alioto & Shahghasemi, 2018; Nuratch, 2017; Ray, 2016; K. J. Singh & Kapoor, 2017).

Uma das características mais importantes deste tipo de dispositivos é a possibilidade de serem utilizados como *gateways*. Assim, no âmbito da IdC, um *gateway* pode ser encarado como um elemento que estabelece a ponte entre as camadas de rede e de aplicação, permitindo que sensores e dispositivos comuniquem com as aplicações por meio de redes móveis, Wi-Fi e *Bluetooth Low Energy* (BLE) (Grygoruk & Legierski, 2016; Zachariah et al., 2015).

Neste âmbito, os dispositivos utilizados podem ser computadores comuns ou embebidos, nós sensores ou sensores inteligentes e circuitos integrados com unidades de microcontrolador (Al-Fuqaha, Guizani, Mohammadi, Aledhari & Ayyash, 2015; Alioto & Shahghasemi, 2018).

2.1.1 Caracterização do Sistema

Para além de sensores e atuadores, um dispositivo para a IdC é, geralmente, composto por interfaces *General Purpose Input/Ouput* (GPIO) e de comunicação série – como *Inter-Integrated Circuit* (I2C), *Serial Peripheral Interface* (SPI) e *Universal Asynchronous Receiver/Transmitter* (UART) –, por conversores digital-analógico e analógico-digital, por um microcontrolador e por elementos de comunicação com e sem fios. Apesar de a comunicação de baixo consumo energético ser a mais usual – por meio do protocolo BLE –, também são utilizados protocolos como o Wi-Fi, o IEEE 802.15.4 e o Z-Wave. Estes dispositivos possuem ainda uma bateria e, caso sejam autónomos, um

elemento de gestão de potência, um armazenador de energia – na forma de super-condensador ou bateria recarregável – e um coletor que recolhe energia fotovoltaica ou de vibração (Al-Fuqaha et al., 2015; Nuratch, 2017; Polianytsia et al., 2016; K. J. Singh & Kapoor, 2017; Urard & Vucinic, 2017; Zachariah et al., 2015).

2.1.2 Elementos de Estudo

Tendo como base a análise efetuada sobre o funcionamento geral dos Dispositivos para a IdC, nesta secção são apresentados, de forma sucinta, exemplos como os Computadores Embebidos, os Nós Sensores e as Placas com Microcontrolador.

2.1.2.1 Computadores Embebidos

Os computadores embebidos são dispositivos de reduzidas dimensões, que apresentam um baixo custo e proporcionam um ambiente de desenvolvimento de *software* rápido e eficiente, possuindo uma grande comunidade de utilizadores. Um dispositivo deste tipo está, habitualmente, presente em campos como a educação e disponibiliza aplicações existentes num computador comum como a navegação na *Internet* e o desenvolvimento de *software*, de documentos e de folhas de cálculo (Chaczko & Braun, 2017; Saari, Bin Baharudin & Hyrynsalmi, 2017; K. J. Singh & Kapoor, 2017)

Este tipo de computador é capaz de interagir, por meio dos vários GPIOs disponibilizados, com componentes lógicos e eletrónicos, sensores e atuadores e oferece vários protocolos de transmissão de dados e interfaces de comunicação, sendo possível a utilização de um monitor, um teclado e um rato. Adicionalmente, é composto por um processador e/ou um microcontrolador e por quantidades de *Random Access Memory* (RAM) entre os 256 MB e os 2 GB e de *Read-only Memory* (ROM) entre os 4 e os 16 GB, sendo alimentado por tensões na ordem dos 5 a 15 V (Arduino, 2018b; Chaczko & Braun, 2017; De Luca, Carnuccio, Garcia & Barillaro, 2016; Intel, 2017; Jie, Ghayvat & Mukhopadhyay, 2015; Khedkar & Malwatkar, 2016; Polianytsia et al., 2016; Raza, Ikram, Amin & Ikram, 2017; K. J. Singh & Kapoor, 2017; Wei-Lin, Lin, Yang & Lin, 2017).

Em termos tecnológicos, este tipo de computador utiliza não só SOs como o Android, o Ubuntu, o ArchLinux, o Debian, o Fedora e algumas versões do Windows, mas também linguagens de programação como Python, C, C++ e Java. Para além disso, dispõe de interfaces de comunicação com – Ethernet – e sem fios – Wi-Fi e Bluetooth –, de armazenamento – micro *Secure Digital* (SD) –, de multimédia – *High-Definition Multimedia Interface* (HDMI) – e de comunicação série – *Universal Serial Bus* (USB),

I2C, UART e SPI (Arduino, 2018b; Intel, 2017; Khedkar & Malwatkar, 2016; Polianytsia et al., 2016; Raza et al., 2017; K. J. Singh & Kapoor, 2017; Wei-Lin et al., 2017).

2.1.2.2 Nós Sensores

Um nó sensor, também conhecido como *mote*, apresenta dimensões reduzidas e é formado por sensores, um microcontrolador, uma unidade de comunicação sem fios com uma antena, uma fonte de energia – que pode ser uma bateria ou um coletor de energia – e memória e armazenamento de reduzidas dimensões para dados. Assim, o microcontrolador fica encarregue do processamento e do armazenamento dos dados recolhidos pelos sensores e a unidade de comunicação sem fios trata do envio dos dados e da recolha de comandos de controlo. Por outro lado, devido à reduzida capacidade energética, a configuração da rede, a técnica de processamento de dados e o desenho do SO necessitam de ser pensados, de forma atenta, para pouparem energia. Para além dos cuidados referidos, caso o nó sensor esteja colocado numa zona de difícil acesso, outra das razões para a poupança dos recursos energéticos passa pela dificuldade na substituição da sua fonte de energia (Botta, de Donato, Persico & Pescapé, 2016; Dash, Mohapatra & Pattnaik, 2010; Dores et al., 2014; Fong, 2017; Hwang, Fox & Dongarra, 2012; Kopetz, 2011; Lee, Murray, Hughes & Joosen, 2010; D. Singh, Tripathi & Jara, 2014; Zhou, 2013).

Apesar deste tipo de nós possuir limitações computacionais, de comunicação e de armazenamento, estes são capazes de detetar eventos, recolher, reencaminhar, processar e garantir a integridade dos dados dos sensores, recuperar e lidar com falhas do sistema e operar em conjunto, formando uma RSSF (Botta et al., 2016; Dash et al., 2010; Dores et al., 2014; Fong, 2017; Hwang et al., 2012; Kopetz, 2011; Lee et al., 2010; D. Singh et al., 2014; Zhou, 2013).

Em termos tecnológicos, esta classe de dispositivos utiliza padrões protocolares para fins de comunicação como o Bluetooth, o ZigBee, o Z-Wave, o Wavenis e o Insteon. Assim, para potenciar todas as suas funcionalidades, um nó sensor pode utilizar SOs como o Contiki e o TinyOS (Gaur & Tahiliani, 2015; Hahm, Baccelli, Petersen & Tsiftes, 2016; Lee et al., 2010; Mainetti, Patrono & Vilei, 2011; N. et al., 2017).

O Contiki é um SO flexível, leve e de livre acesso, implementado em linguagem C e desenvolvido para operar em dispositivos com baixos recursos de memória e energia, podendo funcionar apenas com duas pilhas AA durante um ano. Este suporta programação multiprocesso e baseada em eventos em microprocessadores de 8, 16 e 32 bits, necessitando apenas de 10 e 30 kB de RAM e ROM, respetivamente. Em relação à comunicação, este SO utiliza os protocolos *micro eXtensible Messaging and Presence Protocol* (uXMPP) e *micro* IP (uIP), que disponibiliza o IPv6, o *IPv6 over Low power*

Wireless Personal Area Networks (6LoWPAN), o *Routing Protocol for Low Power and Lossy Networks* (RPL) e o *Constrained Application Protocol* (CoAP) (Gaur & Tahiliani, 2015; Hahm et al., 2016; N. et al., 2017).

Tal como o Contiki, o TinyOS funciona em dispositivos com limitações de energia e de memória, cujos processadores possuem arquiteturas de 8 ou 16 bits e, por ser implementado em nesC, possui um cariz modular e orientado a eventos. Por último, este SO suporta os protocolos de comunicação TCP, *User Datagram Protocol* (UDP), IPv6, RPL, CoAP e 6LoWPAN (Gaur & Tahiliani, 2015; Hahm et al., 2016; N. et al., 2017).

2.1.2.3 Placas com Microcontrolador

As placas com microcontrolador são elementos que possuem restrições a nível energético, de memória e processamento e que necessitam de pouca manutenção. Estes componentes funcionam como plataformas de prototipagem de desenvolvimento acessível e de rápida execução, disponibilizam acesso a bibliotecas para interação com o *hardware* e com os dispositivos que a este são conectados, apresentando uma vasta comunidade de suporte. Ainda possibilitam o contacto com atuadores e sensores e utilizam os vários GPIO, alguns com *Pulse Width Modulation* (PWM), e o conversor analógico-digital para lidar com as leituras analógicas. Em termos aplicacionais, as placas com microcontrolador surgem em contextos como a IdC e as impressoras 3-D (Adafruit, n.d.; Arduino, 2018a; Arduino, 2018c; Hahm et al., 2016; Indiegogo, 2018; Jie et al., 2015; Raza et al., 2017; K. J. Singh & Kapoor, 2017; Wilderness-Labs, 2018).

Relativamente aos aspetos mais técnicos, estes componentes apresentam reduzidas capacidades de armazenamento e memória – 32 a 256 kB e 2 a 32 kB, respetivamente –, são alimentados entre os 5 e os 12 V e detêm interfaces de comunicação série – como o I2C, o SPI e o UART – e, em alguns casos, Bluetooth, Ethernet e Wi-Fi. Relativamente à sua programação, estes dispositivos oferecem ambientes para a elaboração do *software* que corre dentro do microcontrolador, utilizando linguagens de programação como Wiring e C#, e *Light Emitting Diodes* (LEDs) para teste e *debugging* (Adafruit, n.d.; Arduino, 2018c; Indiegogo, 2018; Raza et al., 2017; K. J. Singh & Kapoor, 2017; Wilderness-Labs, 2018).

2.1.3 Síntese

Os Dispositivos para a IdC relacionam-se com o primeiro desafio proposto na identificação do problema, sendo que, neste contexto, os elementos necessários para a sua concretização passam pela utilização de mecanismos de comunicação sem fios. Deste

modo, a tabela 2.1 apresenta-se como uma tabela-resumo que sintetiza a informação pertinente para a satisfação do primeiro desafio proposto na identificação do problema.

Tabela 2.1: Tabela síntese dos Dispositivos para a IdC estudados.

	Computadores Embebidos	**Nós Sensores**	**Placas com Microcontrolador**
Desafio 1	Bluetooth, Wi-Fi	Bluetooth, ZigBee, Z-Wave, Wavenis, Insteon	Bluetooth, Wi-Fi

2.2 *Middlewares* Orientados a Mensagens

Com o advento da computação na *Cloud*, os engenheiros de *software* tiveram a necessidade de desenvolver sistemas facilmente escaláveis e que possuem técnicas para que a informação gerada em tempo real, por aplicações sociais ou de negócios, seja tratada com confiabilidade e distribuída por vários tipos de recetores. Assim, surgiu a necessidade de interligar vários sistemas, passando a ser utilizado o mecanismo de troca de mensagens. Para esse fim, têm vindo a ser desenvolvidos IMs (Dossot, 2014; Garg, 2013).

2.2.1 Caracterização do Sistema

Segundo Curry (2004), um MOM é um mecanismo que fornece métodos de comunicação entre dois componentes de *software* distintos e que permite a construção de sistemas coesivos flexíveis, i.e., sistemas modulares. Desta forma, qualquer *middleware* pode ser considerado um MOM quando possui capacidades para receber e enviar mensagens e, geralmente, opera na camada de aplicação de uma arquitetura de rede baseada no protocolo TCP/IP (Chen, Du, Qin & Zhang, 2013; Curry, 2004).

De acordo com Curry (2004), cada subsistema constituinte do MOM pode ser escalado independentemente, praticamente sem ser necessário a intervenção sobre outros subsistemas. O MOM garante que o funcionamento do sistema global não é afetado quando ocorrem picos de atividade imprevisíveis num dos seus subsistemas e proporciona uma distribuição de carga simples e eficaz, permitindo que um subsistema seja capaz de decidir se deve aceitar receber uma mensagem no imediato ou quando estiver pronto para a processar. Em adição à troca de mensagens, o MOM inclui tarefas de tradução e transmissão de dados, recuperação de erros, segurança, localização de recursos ao longo

da rede, cálculo do menor caminho a seguir, gestão de prioridades de pedidos de mensagens e *debugging* (Curry, 2004; Talarian, 2000).

2.2.1.1 Comunicação

O MOM comunica com base num modelo de interação assíncrona, garantindo que o sistema é não-bloqueável, por meio de Filas de Mensagens (FMs). Adicionalmente, devido ao cariz assíncrono do MOM, este é bastante eficiente em aspetos como o processamento paralelo e independente do tempo. No entanto, segundo Pinus (2004), o MOM também é capaz de comunicar de forma síncrona, apesar de ser necessário implementar, no Cliente e de forma manual, este tipo de comunicação. Na comunicação síncrona, o MOM utiliza o método de Passagem de Mensagens (PM), que se baseia no modelo *Publish-Subscribe* (Chen et al., 2013; Curry, 2004; Pinus, 2004).

Devido à sua característica não-bloqueável, o MOM, enquanto aguarda pelo envio de uma mensagem, pode processar outras que foram recebidas anteriormente. O uso de um MOM não garante que uma mensagem enviada por determinada aplicação seja lida por parte de outra. Para além disso, não possui a capacidade de percecionar o tempo que esta possa demorar a ser lida, sendo isso função da aplicação que recebe a mensagem. Geralmente, um Cliente de um MOM pode trocar mensagens com outros Clientes do sistema de mensagens por meio de uma relação *Peer-to-Peer* (P2P), agindo, cada um deles, como um intermediário na receção e no envio de mensagens, através da conexão estabelecida com um ou mais servidores. O MOM garante que uma mensagem é entregue a todos os destinatários pretendidos apenas uma vez. Para assegurar esta característica de entrega, este possui um mecanismo de distribuição de alta confiabilidade, onde a perda de mensagens devida à falha da rede ou do sistema é solucionada pela utilização de uma técnica de armazenamento e reencaminhamento de mensagens. Adicionalmente, o MOM possui a capacidade de armazenar mensagens quando a rede se encontra indisponível, reencaminhando-as quando esta se revela novamente acessível (Chen et al., 2013; Curry, 2004; Morais & Elias, 2010; Sukarsa, Mahajaya, Made & Sasmita, 2012).

2.2.1.2 Modelos de Mensagens

Um MOM opera de acordo com um modelo de mensagens que se baseia na troca de mensagens por meio de um canal que, tipicamente, é uma FM. Os modelos mais comuns são o Ponto a Ponto e o *Publish/Subscribe* e podem ser conjugados de modo a satisfazer diferentes aspetos do sistema de mensagens (Curry, 2004).

2.2.1.2.1 Ponto a Ponto

O modelo Ponto a Ponto disponibiliza uma troca de mensagens assíncrona entre diferentes elementos de *software*, através de um canal virtual representado por uma FM. Neste modelo, enquanto as mensagens não podem ser entregues ao seu destinatário, existe a garantia de que estas são armazenadas numa FM de forma segura. Estas, representadas, normalmente, por ficheiros, são o elemento-chave para a implementação do modelo de interação assíncrona que o MOM utiliza para comunicar e permitem que os Clientes deste tipo de *middleware* a utilizem para enviar ou receber mensagens. Por outro lado, uma FM pode ser encarada como um elemento que recebe mensagens de aplicações distintas, cuja função é de as reencaminhar para todos os destinatários que tenham sobrescrito a referida FM. Em contrapartida, estas necessitam de ser configuradas para se comportarem como é pretendido, pois o seu desempenho pode ser fraco e, caso seja perdida a ligação com uma FM, todo o sistema pode ser prejudicado (Albano, Ferreira, Pinho & Alkhawaja, 2015; Curry, 2004; Kalra, 2014; Talarian, 2000).

Como se verifica por meio da figura 2.2, uma FM de um MOM funciona, geralmente, segundo a técnica *First-In First-Out* (FIFO), i.e., a primeira mensagem que entra na fila, é a primeira a sair. No entanto, o conteúdo presente numa FM pode ser ordenado de qualquer outra forma (Albano et al., 2015; Curry, 2004).

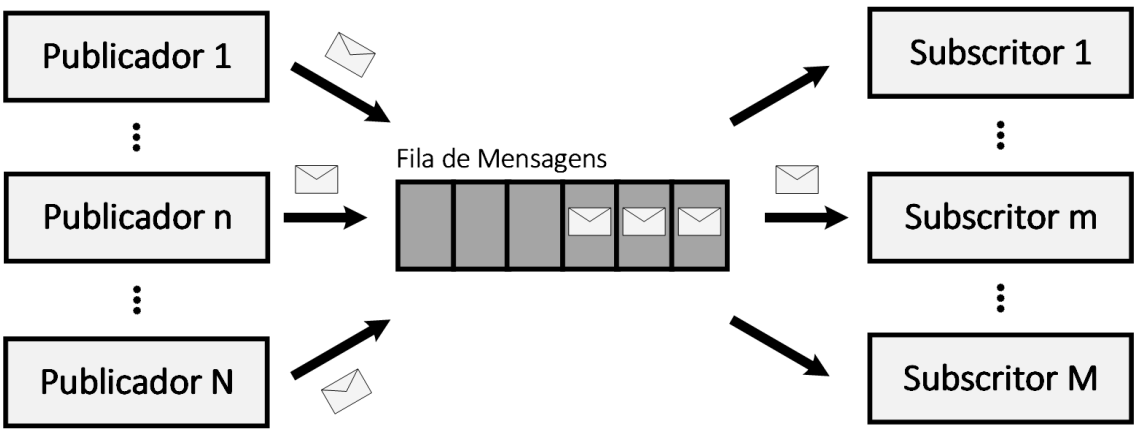

Figura 2.2: Modelo Ponto a Ponto – adaptado de Curry (2004).

O comportamento deste modelo de mensagens pode ser averiguado por intermédio da figura 2.2: um produtor encaminha uma mensagem para uma FM que, por sua vez, a envia para um consumidor. Apesar de poderem ser conectados vários produtores e consumidores a uma FM, apenas um dos consumidores recebe a mensagem porque, neste modelo, uma mensagem apenas é entregue uma vez. No entanto, a possibilidade de conexão de vários consumidores serve para gerir a carga de mensagens a ser entregue.

Adicionalmente, o consumidor possui a capacidade de enviar uma mensagem de confirmação para o produtor (Curry, 2004; Kalra, 2014).

2.2.1.2.2 *Publish/Subscribe*

Por outro lado, o modelo *Publish/Subscribe* disponibiliza uma troca de mensagens anónima entre produtores e consumidores que pode ser utilizada num contexto de um para muitos ou de muitos para muitos, i.e., um produtor pode enviar uma mensagem para um ou centenas de milhares de consumidores. O modelo, também conhecido como *pub/sub*, utiliza um canal virtual denominado tópico e é utilizado pelo método de PM. Neste modelo, tal como se pode verificar por meio da figura 2.3, o produtor publica uma mensagem num determinado tópico e este, por sua vez, é subscrito pelos consumidores que pretendam receber este tipo de mensagem (Curry, 2004; Indrasiri, 2016; Kalra, 2014; Kreps, Narkhede & Rao, 2011; Talarian, 2000).

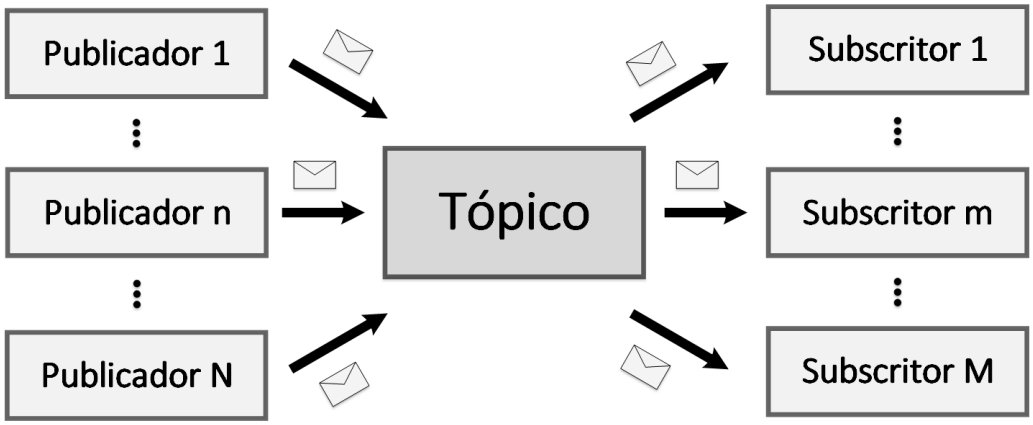

Figura 2.3: Modelo *Publish/Subscribe* – adaptado de Curry (2004).

2.2.2 Elementos de Estudo

Tendo em conta o funcionamento geral dos *Middlewares* Orientados a Mensagens, neste subcapítulo são apresentados, de forma sumária, exemplos como o Apache Kafka, o Node-RED e o RabbitMQ.

2.2.2.1 Apache Kafka

O Apache Kafka é um sistema de mensagens distribuído de livre-acesso e de funcionamento em tempo real que fornece uma troca de mensagens rápida e altamente escalável e utiliza tópicos, i.e., fluxos de mensagens, por meio do modelo *Publish-Subscribe*. A razão pela qual o Apache Kafka é considerado um sistema de mensagens distribuído reside no facto de os seus tópicos serem organizados em partições e o seu

conteúdo ser duplicado para vários servidores do sistema, tornando-o tolerante a falhas. Uma das suas características fundamentais centra-se nas suas estruturas de armazenamento que fornecem um desempenho constante e independente do volume de mensagens a armazenar ("Apache Kafka", 2016; Garg, 2013; Indrasiri, 2016; Kreps et al., 2011; Magnoni, 2015; Z. Wang et al., 2015).

As mensagens são armazenadas no sistema de arquivos e escritas, de imediato, no núcleo do SO. Para além disso, como os produtores enviam mensagens em *batches* e estas podem ser comprimidas e, posteriormente, descomprimidas, o Apache Kafka é capaz de processar mensagens a uma enorme velocidade, atingindo os milhões de mensagens por segundo. Ao contrário do que sucede com a maioria dos sistemas de mensagens, no Apache Kafka, o estado de cada uma das mensagens que foi consumida não fica guardado ao nível do IM, mas ao nível do consumidor, mais concretamente no Apache ZooKeeper. Desta forma, caso a qualidade do consumidor desenvolvido seja reduzida, este pode não chegar a ler a mensagem ou lê-la repetidas vezes (Garg, 2013; Z. Wang et al., 2015).

O Apache Kafka possui dois elementos fundamentais na sua estrutura: os produtores e os consumidores. Um produtor é uma aplicação que possui o intuito de construir uma mensagem e de a publicar, de forma síncrona ou assíncrona, num tópico presente no IM. Por sua vez, um consumidor é uma aplicação que subscreve as mensagens publicadas pelo IM referentes a um dado tópico e que processa os dados nelas contidos. A estrutura de funcionamento dos produtores e dos consumidores está representada na figura 2.4 e, através desta, é possível averiguar que os consumidores – que, neste contexto, são encarados como processos – são organizados em conjuntos denominados Grupo de Consumidores (Garg, 2013).

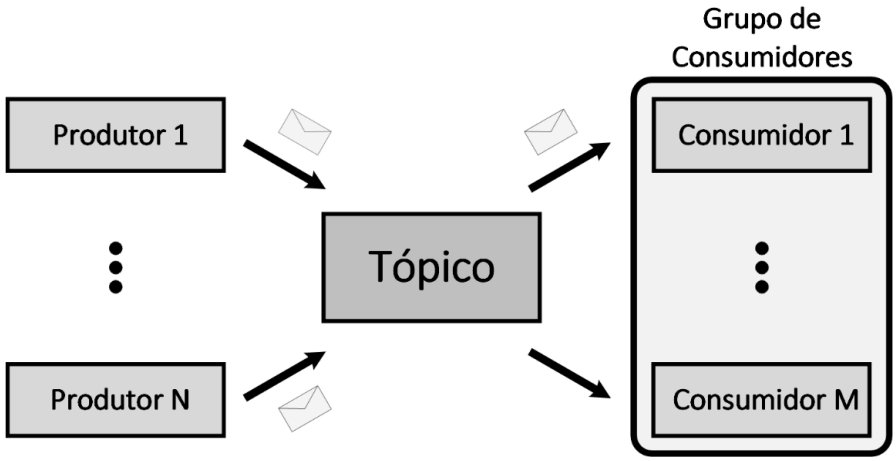

Figura 2.4: Funcionamento geral do Apache Kafka – adaptado de Garg (2013).

O Apache Kafka é integrável com Clientes desenvolvidos em Java, .NET, PHP, Ruby e Python, mas apenas opera de acordo com o seu formato binário através do protocolo TCP. No que diz respeito à segurança dos dados transferidos, estes são encriptados através do protocolo *Secure Sockets Layer* (SSL) e a autenticação entre dispositivos é efetuada por meio do *Simple Authentication and Secure Layer* (SASL) ("Apache Kafka Documentation", 2016; Garg, 2013; Magnoni, 2015).

2.2.2.2 Node-RED

O Node-RED surgiu, em 2013, como resultado do trabalho de Nick O'Leary e Dave Conway-Jones do *Emerging Technology Services* da IBM, sendo, desde 2016, parte constituinte do JavaScript (JS) *Foundation* (Node-RED, n.d.-a).

Este projeto de livre-acesso trata-se de uma plataforma de programação visual e orientada por fluxos de mensagens, implementada em JS por meio de Node.js, que pretende conectar dispositivos de *hardware*, *Application Programming Interfaces* (APIs) e serviços baseados na *web*, de forma a serem desenvolvidas aplicações para a *Cloud* no âmbito da IdC. Este paradigma de desenvolvimento vai ao encontro do conceito de *fog computing*, cujo intuito passa por abranger a computação e os serviços presentes na *Cloud* para uma das pontas da rede, o que permite aos programadores dividir a quantidade de computação que ocorre no dispositivo e na *Cloud*, facilitando a transferência de dados (Blackstock & Lea, 2014; Chaczko & Braun, 2017; Node-RED, n.d.-a; Node-RED, n.d.-e; Palaiokrassas, Karlis, Litke, Charlaftis & Varvarigou, 2017; Pan & McElhannon, 2018; Shih, Chuang & Yeh, 2017; Szydlo, Brzoza-Woch, Sendorek, Windak & Gniady, 2017).

O Node-RED tem como objetivo a construção de um fluxo de dados formado por diversos nós – em que cada um representa um determinado serviço – que estabelecem ligações entre si e proporciona um editor de programação gráfico que facilita o trabalho dos programadores e a integração de um maior número de utilizadores comuns, uma vez que não são necessários conhecimentos profundos de programação. Neste contexto, surge, como consequência do aumento de facilidade referido, uma comunidade de programadores que desenvolvem ativamente novos serviços, i.e., novos nós e fluxos, sendo a lógica estruturada em JS e a interface do utilizador gerada em *HyperText Markup Language* (HTML) (Blackstock & Lea, 2014; Chaczko & Braun, 2017; Node-RED, n.d.-a; Shih et al., 2017).

No Node-RED, cada fluxo de dados possui uma representação em JS *Object Notation* (JSON) e o código gerado nesse formato pode ser exportado facilmente. Por sua vez, cada nó possui a função de receber dados, de os processar e de os enviar para o próximo nó. As mensagens trocadas pelos nós são instâncias de objetos JS e, para além de outros atributos que podem ser adicionados, possuem o *payload* – a parte que possui

os dados – e o *_msgid* – o identificador da mensagem –, que podem ser representados por qualquer tipo de dados (Blackstock & Lea, 2014; Chaczko & Braun, 2017; Node-RED, n.d.-a; Node-RED, n.d.-e).

De acordo com Shih et al. (2017), a arquitetura do Node-RED está alicerçada no ciclo de eventos – em inglês, *event loop* – do Node.js, que, como é possível verificar através da figura 2.5, é caracterizado por cinco etapas de funcionamento: supervisão dos fluxos e descoberta do próximo nó a ser executado; envio dos eventos despoletados pelo nó para o mecanismo do ciclo de eventos; execução do conteúdo do nó; devolução do trabalho resultante da intervenção do ciclo de eventos; e entrega dos resultados ao nó gerador dos eventos. Paralelamente, cada nó representa uma sub-classe do *EventEmitter* e herda o comportamento da classe *Node* presentes no API do Node.js. Resultante da utilização deste tipo de arquitetura, o Node-RED adquire a vantagem de executar JS nos serviços do Cliente e do Servidor. No entanto, por correr num processo *single-thread*, i.e., num processo único, o ciclo de eventos apenas permite a execução de um fluxo de dados de cada vez, sendo necessário recorrer a protocolos externos para comunicar com outros fluxos, o que pode levar a algum congestionamento da rede (Blackstock & Lea, 2014; Shih et al., 2017).

Este *software* possui a capacidade de comunicar com sensores – por meio de interfaces GPIO e USB – e serviços do SO, podendo ser utilizado localmente em SOs baseados em Linux ou Windows ou na *Cloud* em plataformas como o IBM Bluemix, o SenseTecnic FRED, o Amazon Web Services ou o Microsoft Azure e em ambientes como o Docker. Para além dos elementos referidos, devido à estrutura leve do modelo de programação orientado a eventos do Node.js e à simplicidade com que foi desenvolvido o Node-RED, este último é capaz de correr em dispositivos como o Raspberry Pi, o BeagleBone Black e que possuam o SO Android e de interagir com dispositivos Arduino (Blackstock & Lea, 2014; Node-RED, n.d.-c).

O Node-RED suporta os protocolos de comunicação *Advanced Message Queuing Protocol* (AMQP), CoAP, *HyperText Transfer Protocol* (HTTP), *Lightweight* M2M (LWM2M), *Message Queuing Telemetry Transport* (MQTT), *Simple Text Oriented Messaging Protocol* (STOMP), TCP, UDP, *WebSockets* e e*Xtensible Messaging and Presence Protocol* (XMPP); as estruturas de dados *Comma-Separated Values* (CSV), JSON, *eXtensible Markup Language* (XML) e *YAML Ain't Markup Language* (YAML); e interage com bases de dados relacionais – *Microsoft Structured Query Language* (MSSQL), MySQL, PostgreSQL e SQLite – e não-relacionais – Apache Cassandra, CouchDB e MongoDB. Apesar de esta plataforma não possuir nenhum mecanismo de

segurança de origem, esta suporta os serviços de autenticação OAuth e OpenID e o protocolo de encriptação SSL (Node-RED, n.d.-b; Node-RED, n.d.-d).

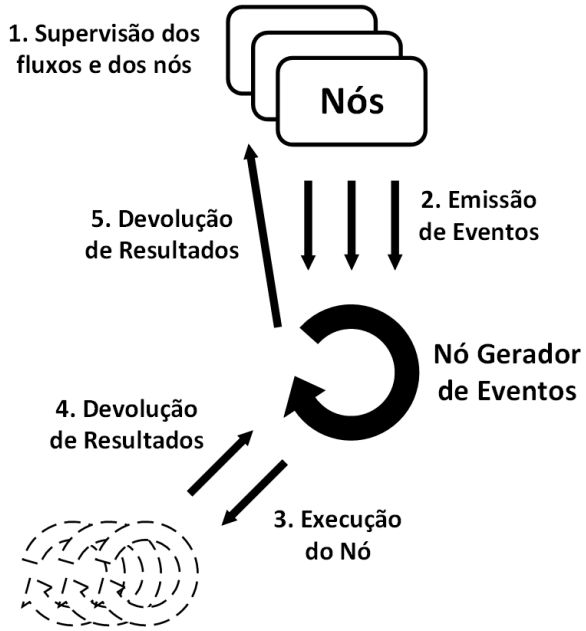

Figura 2.5: Ciclo de Eventos do Node.js – traduzido de Shih et al. (2017).

2.2.2.3 RabbitMQ

O RabbitMQ implementa, através da linguagem de programação Erlang, um IM de livre-acesso baseado no protocolo de comunicação AMQP, tendo sido desenvolvido para resolver os problemas relativos à integração de aplicações e à transmissão de grandes quantidades de dados. A implementação em Erlang reside no facto de esta linguagem possuir suporte próprio para o desenvolvimento de aplicações distribuídas e confiáveis e ser compatível com qualquer SO. Segundo Ayanoglu, Aytas e Nahum (2015), o RabbitMQ é igualmente referido como sendo um MOM. Assim, o RabbitMQ pode ser visto como uma plataforma de receção e envio de mensagens que assegura a integridade destas até serem recebidas, atuando como um mediador entre os consumidores e os produtores de mensagens (Ayanoglu et al., 2015; Dossot, 2014; Ionescu, 2015; Magnoni, 2015; Videla & Williams, 2012).

De acordo com Videla e Williams (2012), o RabbitMQ funciona segundo um paradigma composto por produtores e consumidores. Um produtor possui a função de gerar as mensagens e de as marcar com a etiqueta de encaminhamento correta, enviando-as para o IM por meio de uma comunicação unidirecional. Por sua vez, o IM estabelece uma ligação com um consumidor, subscrevendo a informação fornecida pelas FMs. Quando a mensagem se encontra no produtor, é constituída por duas partes: o *payload* e

a etiqueta de encaminhamento, que serve para o IM saber para que consumidores deve encaminhar os dados. Ao chegar ao consumidor, a mensagem apenas é formada pelo *payload*, uma vez que, nesta etapa, a etiqueta de encaminhamento costuma ser dispensável. No entanto, caso seja necessário saber o remetente da mensagem, cabe ao produtor introduzir essa informação no *payload*. Em suma, o funcionamento genérico do RabbitMQ pode ser representado por intermédio do esquema da figura 2.6 (Videla & Williams, 2012).

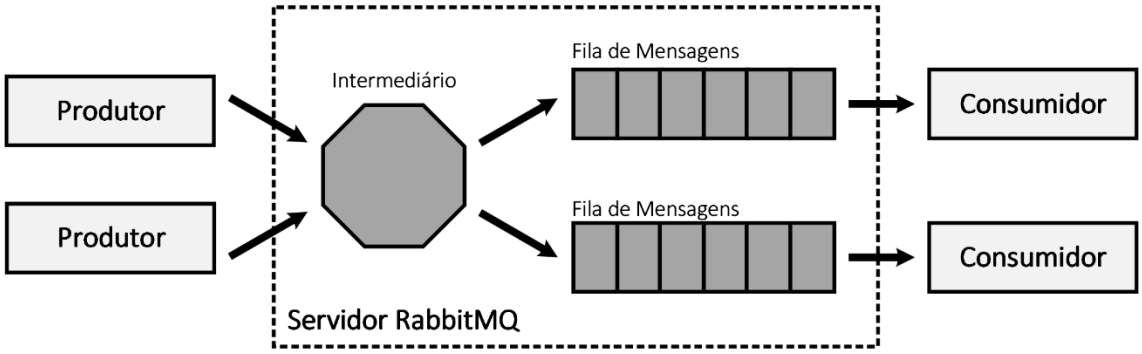

Figura 2.6: Funcionamento geral do RabbitMQ – adaptado de Subramanian (2015).

O RabbitMQ é extensível através da integração de *plugins* e é responsável pelo armazenamento de dados e pelo agrupamento de vários servidores, formando um único IM, cujo objetivo é tornar o sistema tolerante a falhas. O armazenamento de dados pode ser efetuado por intermédio da base de dados própria do Erlang – a Mnesia – ou por meio de um armazenamento de mensagens específico, sendo que não existe uma estrutura pré-definida para estas, o que leva a que possam ser representadas inclusive através de dados binários. Por sua vez, o agrupamento de vários IMs é efetuado de acordo com as capacidades de agrupamento intrínsecas do Erlang. Para além disso, é capaz de encriptar os dados recebidos através do protocolo SSL, efetua a autenticação entre dispositivos por meio do SASL, suporta o envio de mensagens através de vários protocolos de comunicação – STOMP, MQTT, HTTP e as versões 0-9-1 e 1.0 do AMQP – e possui Clientes desenvolvidos para a maioria das linguagens utilizadas atualmente – como Java, .NET, Ruby, Python, PHP, Objective-C, Node.js, C++ e JS (Ayanoglu et al., 2015; Dossot, 2014; Ionescu, 2015; Magnoni, 2015; "RabbitMQ - Clients & Developer Tools", 2017; "RabbitMQ - Protocols", 2017; Videla & Williams, 2012).

2.2.3 Síntese

Os *Middlewares* Orientados a Mensagens relacionam-se com todos os desafios propostos na identificação do problema, sendo que, neste contexto, os elementos

necessários para a sua concretização passam, para os desafios 1 e 3, por mecanismos de troca de mensagens e, para o desafio 2, de armazenamento de dados. Assim, a tabela 2.2 revela-se como uma tabela-resumo que sintetiza a informação relevante para o cumprimento dos referidos desafios.

Tabela 2.2: Tabela síntese dos *Middlewares* Orientados a Mensagens estudados.

	Apache Kafka	**Node-RED**	**RabbitMQ**
Desafio 1 **Desafio 3**	- *Publisher/Subscriber*	- Programação orientada por fluxos de mensagens	- Subscrição de Tópicos - Filas de Mensagens
Desafio 2	- Sistema de Arquivos - Núcleo do SO	- SQLite	- Mnesia - Armazenamento de mensagens específico

2.3 Plataformas de *Cloud* para a IdC

Em 2006, Eric Schmidt, da Google, popularizou o conceito de *Cloud Computing*, i.e., de computação na *Cloud*, ao afirmar que este contém serviços sobre os dados que podem estar em qualquer local e ser acedidos por qualquer dispositivo que disponibilize ligação à *Internet*. O conceito de computação na *Cloud* abrange um ambiente para o desenvolvimento e a integração de aplicações e de sistemas. Este conceito apresenta um modelo que compartilha recursos computacionais configuráveis como aplicações, redes, servidores e serviços de armazenamento a qualquer tipo de utilizador. Para além disso, apoia-se num ambiente flexível, acessível e escalável, com um plano de preços adaptável e baseado nas necessidades do utilizador (Dash et al., 2010; Dores et al., 2014; Mell & Grance, 2011; Premsankar et al., 2018; Q. Zhang, Cheng & Boutaba, 2010; Zhou, 2013).

Dentro do âmbito deste modelo de computação, os tipos mais comuns de *Cloud* são: a pública, a privada, a híbrida, a privada virtual e a comunitária. Apesar de não proporcionar uma confiabilidade elevada ao nível do controlo dos dados, da rede e da segurança, a *Cloud* pública apresenta a vantagem de disponibilizar serviços ao público em geral. Por outro lado, a *Cloud* privada apresenta a maior fiabilidade em termos de segurança e desempenho e foi projetada para utilização restrita a apenas um organismo, podendo ser gerida pelo próprio ou por um outro externo. Já o modelo híbrido consiste na agregação dos dois tipos de *Cloud* anteriores e o modelo comunitário foi desenhado para suprir as necessidades de um grupo específico de utilizadores, podendo ser controlado por uma ou mais entidades. Por último, o tipo privado virtual tira partido de

redes do mesmo tipo para permitir aos seus proprietários implementarem os seus próprios mecanismos de segurança e utiliza a sua capacidade de virtualização para proporcionar uma transição mais eficiente dos serviços para a *Cloud* (Dash et al., 2010; Mell & Grance, 2011; Q. Zhang et al., 2010).

A *Cloud* dispõe de serviços de *software* – em inglês, *Software as a Service* (SaaS) –, de plataforma – em inglês, *Platform as a Service* (PaaS) – e de infraestrutura – em inglês, *Infrastructure as a Service* (IaaS). O primeiro tipo de serviço tem a função de disponibilizar aplicações que correm na *Cloud*. Os serviços de plataforma permitem a implementação de aplicações por meio de ferramentas, bibliotecas e linguagens de programação disponibilizadas pela *Cloud* e fornecem apoio relativamente aos sistemas operativos utilizados. Por fim, os serviços de infraestrutura asseguram ao consumidor, geralmente através de máquinas virtuais, a possibilidade de correr as suas aplicações e o seu SO, garantindo-lhe acesso a recursos de rede, processamento e armazenamento de dados (Botta et al., 2016; Dash et al., 2010; Mell & Grance, 2011; Q. Zhang et al., 2010).

Segundo Q. Zhang et al. (2010), a computação na *Cloud* assenta numa arquitetura composta pelas camadas de *hardware*, de infraestrutura, de plataforma e de aplicação: de um modo geral, a camada de *hardware* está presente nos centros de armazenamento de dados e tem o intuito de dirigir os meios físicos – como os servidores; o segundo elemento referido é responsável por gerir os recursos materiais por meio de mecanismos de virtualização; a camada de plataforma é composta por aplicações e SOs e pelo apoio ao desenvolvimento de serviços de armazenamento; por fim, o elemento de mais alto nível é formado pelas aplicações de *Cloud*.

Zhou (2013) indica que a computação na *Cloud* permite à IdC potenciar todas as suas capacidades e Botta et al. (2016) defendem que as suas características são complementares com a IdC, afirmando que, segundo muitos autores, a *Cloud* é o elemento-chave para a integração dos dois contextos. Deste modo, verifica-se que a *Cloud* é capaz de proporcionar à IdC armazenamento, processamento e recursos de comunicação virtualmente infinitos e uma camada intermédia de abstração entre as Coisas e as aplicações. Por outro lado, a IdC garante ao ambiente da *Cloud* a capacidade de lidar com problemas de forma dinâmica e distribuída. Para terminar, vários autores apontam que a *Cloud* reúne características como a escalabilidade, a interoperabilidade, a confiabilidade, a flexibilidade, a acessibilidade e a segurança, sendo que Botta et al. (2016) sugerem que a IdC necessita de um sistema que apresente as referidas características (B., Saluja, Sharma, Mittal & Sharma, 2012; Botta et al., 2016; Dash et al., 2010; Dores et al., 2014; Fox, Kamburugamuve & Hartman, 2012; Premsankar et al., 2018; Q. Zhang et al., 2010; Zhou, 2013).

2.3.1 Caracterização do Sistema

Ganguly (2016) apresenta a plataforma de *Cloud* para a IdC como um serviço destinado a operar especificamente sobre um ou mais domínios, pago ou de livre-acesso, que possua comunidades de programadores e que permita a gestão das Coisas e dos eventos a elas associado através de, por exemplo, o envio de notificações para o utilizador. A plataforma deve ser capaz de integrar distintos protocolos de *hardware* e *software*, possuir um vasto armazenamento e processamento de dados e permitir a análise e supervisão em tempo real dos dados e dos dispositivos, em modo *offline* ou em ambas as situações (Ganguly, 2016; IoT-Analytics, 2015; Rimal & Lumb, 2017; K. J. Singh & Kapoor, 2017).

Segundo Ganguly (2016), este serviço deve ser integrável com as redes sociais mais comuns e altamente escalável, preferencialmente de um modo automático. Para além da escalabilidade, IoT-Analytics (2015) e Rimal e Lumb (2017) referem a interoperabilidade e Ganguly (2016) aborda a disponibilidade, o desempenho, a confiabilidade e a possibilidade de efetuar cópias de segurança dos dados como outros requisitos não-funcionais a ter em conta para este tipo de plataforma.

Relativamente aos requisitos técnicos, Ganguly (2016) indica que a plataforma deve dispor de suporte relativo aos protocolos mais utilizados atualmente, como *WebSockets*, MQTT, CoAP, XMPP ou outros baseados no *standard* de operações para o desenvolvimento de serviços para a *web* 2.0 *Representational state transfer* (REST), permitindo a tradução entre protocolos distintos. O serviço deve oferecer comunicação seguindo o paradigma MM e ferramentas de desenvolvimento, como APIs e *Software Development Kits* (SDKs), para implementação de *software* em linguagens de programação como Java, Python, Embedded C, Node.js, C# ou JS e construção de aplicações móveis para os SOs Android e iOS. Uma ferramenta deste género, de acordo com Suciu, Halunga, Vulpe e Suciu (2013), deve ser capaz de estabelecer uma ligação física por intermédio de uma das seguintes tecnologias: NFC, RFID e ZigBee (Battle & Benson, 2008; Ganguly, 2016; IoT-Analytics, 2015; K. J. Singh & Kapoor, 2017; Suciu et al., 2013).

Em termos de segurança, esta deve possuir uma camada própria para o efeito, apresentando, pelo menos, encriptação de dados ao nível das mensagens. Para além disso, os serviços de autenticação mais recorrentes são o *Single Sign-On* (SSO), o OAuth e o OpenID (Ganguly, 2016).

Relativamente ao formato de mensagens, os mais comummente utilizados são o JSON e o XML, sendo possível efetuar a conversão entre os diferentes formatos. Adicionalmente, a plataforma deve possuir a capacidade de utilizar bases de dados que

lidam com dados estruturados e não-estruturados, i.e., do tipo SQL e *Not only* SQL (NoSQL) (Ganguly, 2016; IoT-Analytics, 2015).

2.3.2 Elementos de Estudo

Tendo sido descrito com maior detalhe o funcionamento geral das Plataformas de *Cloud* para a IdC, nesta secção são apresentados, de forma breve, exemplos como o Apache Hadoop, o Firebase e o Kaa.

2.3.2.1 Apache Hadoop

Em 2005, como resultado do trabalho realizado por Doug Cutting e Mike Cafarella, surge o Apache Hadoop: uma plataforma de dados integrável e de livre acesso implementada em Java e baseada no projeto *Map/Reduce* (MR) desenvolvido pela Google. O seu foco principal passa pela gestão e pelo processamento de grandes blocos de dados por meio de aplicações que operam de forma paralela e distribuída e utilizam *hardware* de baixo-custo para o seu funcionamento. Desenvolvido para ser uma ferramenta escalável e com grande capacidade de armazenamento e desempenho, o Apache Hadoop tem caminhado no sentido de se tornar a plataforma de referência no âmbito do *Big Data* e marca presença em áreas como o agendamento de processos e a otimização de algoritmos de ordenação e de aplicações de técnicas baseadas em MR. O seu modo de operação consiste na divisão de um grande bloco de dados em pequenas frações, sendo cada uma destas trabalhada em processos separados que funcionam em paralelo (A. & K., 2013; Gazal & Kaur, 2015; Hazarika, Ram & Jain, 2017; Jeong, Hyun, Lim & You, 2012; Johri, Das, Kumar & Arora, 2017; Manwal & Gupta, n.d.; Melorose, Perroy & Careas, 2010; Reddy et al., 2013; Shaw, 2017; Zheng & Jiang, 2014).

O Apache Hadoop é composto pelo MR, pelo *Hadoop Distributed File System* (HDFS) e pelo *Yet Another Resource Negotiator* (YARN) (The Apache Software Foundation, 2017a; Jeong et al., 2012; Johri et al., 2017; Shaw, 2017).

O HDFS é um sistema de ficheiros distribuído, baseado no *Google File System* (GFS), que foi projetado para proporcionar um acesso eficaz aos dados, independentemente das suas dimensões e formato, possuir uma elevada taxa de transferência de dados e um alto nível de tolerância a falhas, utilizando *hardware* de custo reduzido. Para além disso, este sistema de ficheiros é caracterizado pela sua fiabilidade – devido à geração de cópias de segurança, o que leva a que o sistema nunca pare –, por não possuir nenhum sistema de segurança de origem e por seguir um paradigma em que é possível escrever apenas uma vez num ficheiro, mas lê-lo diversas vezes – em inglês, *write-once-read-many* (A. & K., 2013; Alam & Ahmed, 2014; Azzedin, 2013; Elomari,

Maizate & Hassouni, 2016; Gazal & Kaur, 2015; Jeong et al., 2012; Johri et al., 2017; Pinto, Jain & Kumar, 2016; Shaw, 2017; Zheng & Jiang, 2014).

O HDFS, tal como se pode verificar por meio da figura 2.7, trabalha segundo uma arquitetura *master/slave*, em que o *Name Node* é o mestre e os vários *Data Node* são os escravos. O *Name Node* é responsável pela configuração do *cluster* e pela gestão do sistema de ficheiros, da comunicação com dispositivos externos, do mapeamento, da informação dos metadados dos *Data Nodes* e dos nós ativos e passivos. Por sua vez, cada *Data Node* encontra-se presente em cada nó do *cluster*, gerindo o armazenamento de dados do nó a si atribuído em conjunto com os processos relativos ao *Task Tracker* e ao *Job Tracker*, cuja função deste último passa pelo agendamento de eventos. Para além dos referidos nós, Alam e Ahmed (2014) apresentam o *Secondary Name Node*, cuja finalidade é armazenar uma cópia de segurança do conteúdo do *Name Node* cada vez que este é modificado (A. & K., 2013; Alam & Ahmed, 2014; Azzedin, 2013; Elomari et al., 2016; Hazarika et al., 2017; Jeong et al., 2012; Manwal & Gupta, n.d.; Pinto et al., 2016; Reddy et al., 2013; C. Wang, Zheng & Yang, 2014).

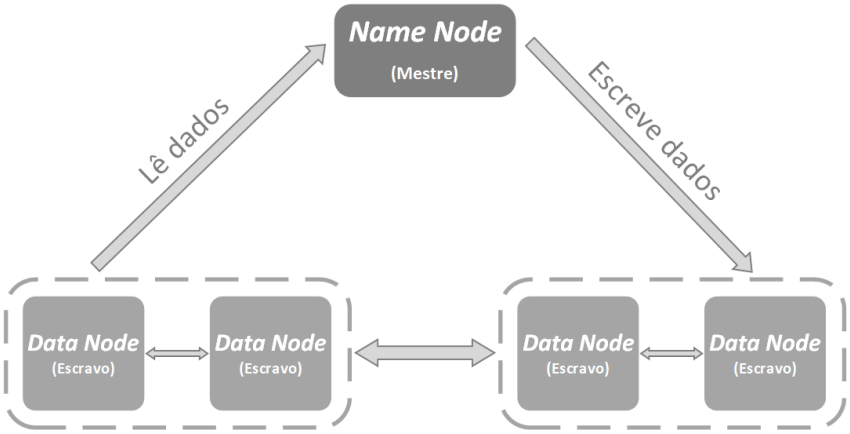

Figura 2.7: Funcionamento geral do HDFS – adaptado de C. Wang et al. (2014).

O MR é uma ferramenta computacional cujo objetivo passa, como se pode ver por intermédio da figura 2.8, pelo tratamento de grandes blocos de dados do grupo de servidores de forma distribuída e paralela, cujo padrão de desenvolvimento assenta em duas operações: o mapeamento – em inglês, *map* – e a redução – em inglês, *reduce*. A função de mapeamento possui o intuito de produzir um conjunto intermédio de dados, resultante da atribuição de elementos chave/valor, proveniente dos dados do utilizador, tendo em conta a sua chave como atributo de ordenação. Já a ação de redução agrupa os elementos chave/valor que possuem a mesma chave para os transformar num conjunto de dimensões mais reduzidas, que é o *output* do MR (A. & K., 2013; Alam & Ahmed, 2014; Gazal & Kaur, 2015; Hazarika et al., 2017; Manwal & Gupta, n.d.; Reddy et al., 2013).

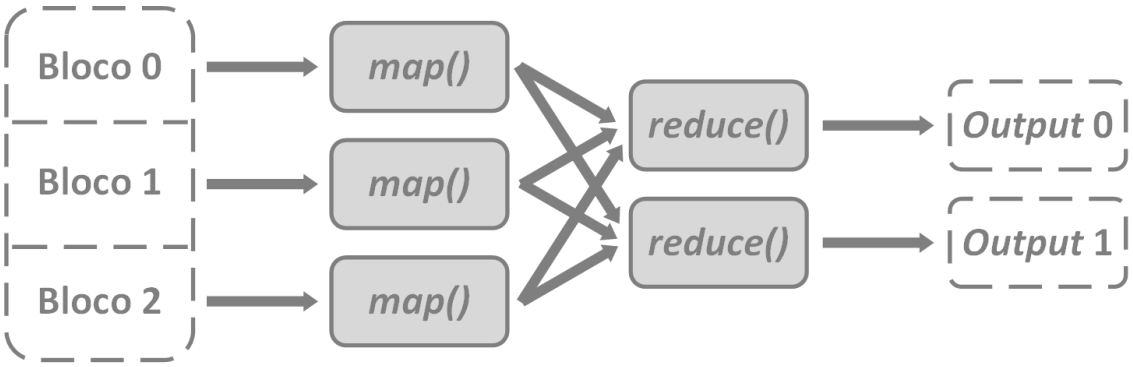

Figura 2.8: Funcionamento geral do MR – adaptado de Hazarika et al. (2017).

O YARN é encarado como o SO do Apache Hadoop e possui o intuito de separar em diferentes processos a gestão de recursos e o agendamento e monitorização de tarefas. Desta forma, este componente do Apache Hadoop divide-se em três elementos fundamentais: o *ResourceManager*, o *NodeManager* e o *ApplicationMaster*. O *ResourceManager* é responsável por gerir os recursos das várias aplicações e tarefas operantes, o *NodeManager* controla os recursos relacionados com o *hardware* e o *ApplicationMaster* executa e monitoriza as tarefas do sistema (The Apache Software Foundation, 2017b; Shaw, 2017; Zheng & Jiang, 2014).

O Apache Hadoop utiliza uma camada baseada no protocolo TCP/IP para comunicação entre os diversos nós e é compatível com algumas ferramentas de segurança como *firewalls* e o Kerberos. Ao ser uma plataforma integrável, esta suporta APIs como a *Open Database Connectivity* (ODBC) e a *Java Database Connectivity* (JDBC) e trabalha com operações baseadas em SQL e com linguagens como Java, Python e Scala (Azzedin, 2013; Johri et al., 2017; Shaw, 2017).

Ao instanciar o Apache Hadoop na *Cloud*, este passa a usufruir de elevado poder de computação, bem como de capacidade de armazenamento escalável, através de um plano de custos igualmente adaptável. Desta forma, é possível que esta plataforma de dados opere como base para a IdC em infraestruturas de *Cloud* públicas como o Amazon Web Services, o Rackspace, o Microsoft Azure, o Google Compute Engine ou o OpenStack e na qual utiliza serviços como o Elastic MapReduce da Amazon ou o MapR (Al-Qaseemi et al., 2016; Melorose et al., 2010; S, Alam & Srinivasan, 2017).

Relativamente ao armazenamento de dados, este pode ser efetuado em bases de dados relacionais como o MySQL e não-relacionais como o MongoDB, o Apache Cassandra e o Apache HBase. No que concerne à sincronização entre o Cliente e o Servidor, esta pode ser efetuada, mais concretamente nos casos do MySQL e do MongoDB, diretamente a partir das bases de dados (Jangra, Bhatia, Lakhinaza & Singh, 2015; Kacfah Emani, Cullot & Nicolle, 2015; Krishnan, 2013; Malhotra, Doja, Alam &

Alam, 2016; MongoDB, 2018; Oracle, 2018; Oussous, Benjelloun, Ait Lahcen & Belfkih, 2017; R. & A., 2016).

2.3.2.2 Firebase

O Firebase é uma plataforma móvel sem custos para programadores que fornece várias ferramentas ajustáveis e APIs contidos num único SDK, com vista à resolução de problemas e à elaboração de aplicações de elevada qualidade. Os APIs são desenvolvidos em Swift, Objective-C, Java, JS e C++ e são compatíveis com os SOs Android e iOS, sendo facilmente expansíveis sem a necessidade de alterar a infraestrutura utilizada. O Firebase disponibiliza ainda uma solução de troca de mensagens e notificações entre plataformas distintas – Android, iOS e *web* – que se denomina *Firebase Cloud Messaging* (FCM). A referida solução gere a entrega de mensagens e possibilita o seu envio para dispositivos isolados, para um determinado público-alvo ou para dispositivos com tópicos subscritos ("Firebase," n.d.; "Firebase Documentation - Firebase Cloud Messaging," 2016; "Firebase - Features," n.d.; "Firebase: Pricing," 2016).

A infraestrutura relativa ao servidor do FCM é constituída por um servidor de aplicações e por servidores de conexão com o *Google Cloud Messaging* (GCM). Um servidor de aplicações usa os protocolos HTTP ou XMPP para encaminhar dados para um Cliente através de um dos servidores de conexão do FCM. Para além disso, armazena, de forma segura, a chave do servidor e os *tokens* de registo do Cliente. Por sua vez, os servidores de conexão do FCM recebem mensagens provenientes de um servidor de aplicações e reencaminham-nas para um dispositivo Cliente ("Firebase Documentation - Firebase Cloud Messaging," 2016).

O FCM possibilita a escolha entre os protocolos HTTP e XMPP para aceder a um servidor de conexão. O protocolo HTTP permite, no sentido da *Cloud* para o dispositivo, enviar mensagens síncronas, em formato JSON ou de texto simples, como solicitações HTTP POST e o seu sincronismo impossibilita o Cliente de enviar uma nova mensagem enquanto não for recebida uma resposta. Por outro lado, o protocolo XMPP possibilita o envio de mensagens de cariz assíncrono, no formato JSON, cuja transmissão pode ser efetuada da *Cloud* para o dispositivo, ou no sentido contrário, à taxa de transferência máxima permitida pela linha. Num servidor de conexão do FCM, são emitidas, assincronamente, notificações de falha ou de confirmação, cuja codificação é efetuada em JSON ("Firebase Documentation - Firebase Cloud Messaging," 2016).

A autenticação de uma aplicação baseada no Firebase é efetuada por meio do serviço *Firebase Authentication*. Este serviço disponibiliza vários SDKs versáteis e usa protocolos como o OAuth 2.0 e o OpenID Connect para proceder à autenticação de utilizadores, o que permite uma integração simples com *backends* existentes. Para além

disso, a autenticação pode ser realizada por intermédio da utilização de várias redes sociais ou *e-mail*, existindo a possibilidade de o utilizador permanecer anónimo (“Firebase Documentation - Firebase Authentication,” 2016).

O Firebase armazena os dados na *Cloud* em bases de dados NoSQL sob o formato JSON, obtendo-os do SDK do Cliente – tal como pode ser verificado através da figura 2.9. O facto de a aplicação permanecer reativa independentemente da qualidade e da rapidez da rede leva a que os dados estejam acessíveis em modo *offline*. Porém, assim que a ligação à rede esteja novamente disponível, os dados são sincronizados entre dispositivos ligados entre si em tempo real (“Firebase - Features,” n.d.).

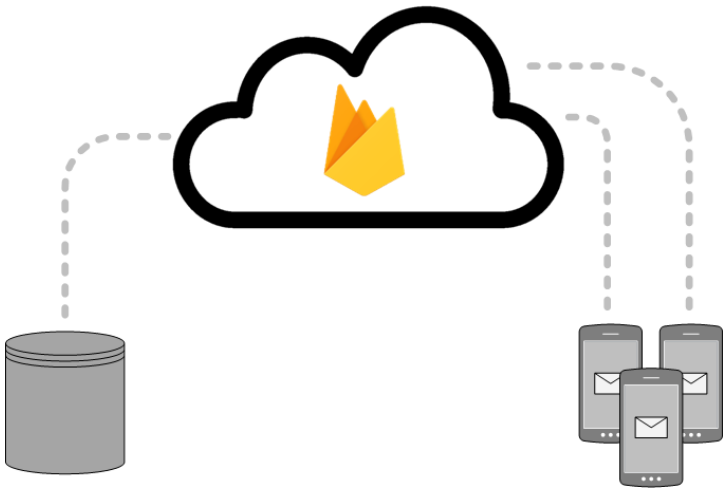

Figura 2.9: Conceito geral do Firebase – adaptado de Firebase (2016).

2.3.2.3 Kaa

O Kaa é uma plataforma de *middleware* de livre-acesso que disponibiliza várias ferramentas para o desenvolvimento de produtos e soluções relativas à IdC. O Kaa é integrável com a maioria dos dispositivos, sensores e *gateways* existentes no mercado e compatível com SOs como Linux – através de C, C++ e Java –, Windows – por meio de C++ e Java –, Android – por intermédio de Java – e iOS – ao utilizar Objective-C (“Kaa IoT Overview,” 2017; T, 2016; “Using Kaa endpoint SDKs,” 2016).

A plataforma providencia uma estrutura de funcionalidades e extensões para a construção de aplicações no âmbito da IdC e vários SDKs integráveis com a aplicação do Cliente. Os SDKs disponibilizados possuem a capacidade de lidar com a comunicação entre Cliente e Servidor, com a autenticação de dispositivos, com o agrupamento de dados e com a sua encriptação. A predisposição do Kaa para utilizar dados estruturados e não-estruturados facilita na abstração de detalhes pouco importantes ao nível das implementações de baixo-nível e do *hardware*. Por sua vez, esta ferramenta permite que

dispositivos baseados em diferentes tecnologias sejam geridos de forma idêntica por possuírem os mesmos conjuntos de esquemas de dados (“Kaa IoT Overview,” 2017; “Kaa open-source IoT Platform - SDK,” 2017; Malche & Maheshwary, 2015).

Outra funcionalidade do Kaa consiste no envio de notificações, através do UDP e do *Short Message Service* (SMS), e na sincronização de dados de perfil e de configuração, por meio de um protocolo binário baseado no TCP, cuja encriptação se apoia no *Advanced Encryption Standard* (AES) (“Kaa open-source IoT Platform - Connectivity,” 2017; Malche & Maheshwary, 2015; T, 2016).

O Servidor pertencente ao Kaa apresenta interfaces que utilizam serviços baseados em REST para a integração com serviços específicos do produto, a gestão de dados e a incorporação de protocolos de comunicação personalizados. Em termos organizacionais, este está estruturado em grupos de nós expansíveis, tornando-o altamente escalável, uma vez que, quanto maior for o número de nós presente num grupo, maior será a sua performance. Para além disso, o Servidor do Kaa não possui períodos de inatividade nem pontos de interrupção ao longo do grupo de nós, o que leva a que seja tolerante a falhas. Um elemento fundamental para a organização dos servidores, bem como para a seleção do servidor a utilizar e para a tolerância a falhas é o Apache ZooKeeper, um serviço centralizado de livre-acesso construído para preservar informações de configuração e fornecer sincronização distribuída e serviços de agrupamento (“Apache ZooKeeper,” 2016; “Kaa open-source IoT Platform - FAQ,” 2017; “Kaa open-source IoT Platform - Kaa Server,” 2016).

Em termos de conectividade, o Kaa permite o desenvolvimento de aplicações cuja operação é exequível com qualquer tipo de ligação à rede, seja por meio de um protocolo personalizado pelo utilizador ou através de um dos protocolos de comunicação suportado pela plataforma – protocolos baseados em HTTP *Secure* (HTTPS) ou TCP. Ao utilizar os protocolos de comunicação referidos, a segurança dos dados é garantida por intermédio de uma encriptação híbrida que utiliza uma chave do tipo *Rivest Shamir Adleman* (RSA) de 2048 bits juntamente com uma chave do tipo AES de 256 bits (“Kaa open-source IoT Platform - Connectivity,” 2017; “Kaa open-source IoT Platform - FAQ,” 2017).

Relativamente ao armazenamento de dados, este pode ser efetuado em bases de dados do tipo SQL e NoSQL, sendo disponibilizada uma camada de abstração para o armazenamento de dados e para o desenvolvimento de *plugins*. Atualmente, as bases de dados SQL suportadas são o PostgresSQL e o MariaDB e as bases de dados NoSQL disponíveis são o Apache Cassandra e o MongoDB. Na figura 2.10, pode ser vista a interação entre o grupo de nós relativo aos Servidores do Kaa e os elementos de armazenamento, bem como com as respetivas aplicações. No que diz respeito à sincronização entre o Cliente e o Servidor, esta pode ser realizada através do API

desenvolvido em Java que se encontra na página do GitHub pertencente ao Kaa ("Kaa open-source IoT Platform - FAQ," 2017; "Kaa Project - GitHub," 2016; "Kaa System Installation," 2016).

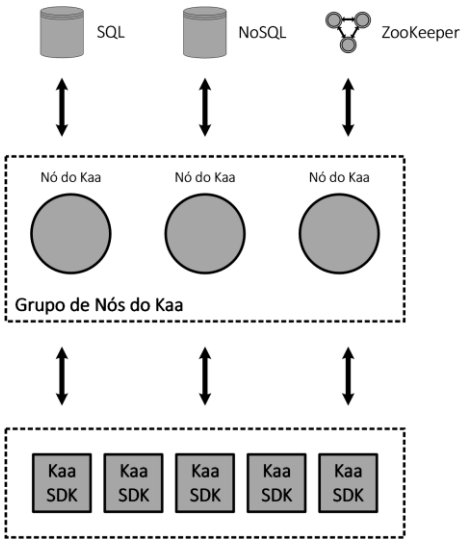

Figura 2.10: Interação entre os nós do Kaa e os elementos de armazenamento – adaptado de "Kaa open-source IoT Platform - Architecture Overview" (2016).

2.3.3 Síntese

As Plataformas de *Cloud* para a IdC relacionam-se com os desafios 1 e 3, sendo que, neste contexto, os elementos necessários para a sua concretização passam, respetivamente, por mecanismos de armazenamento de dados e de sincronização. Desta forma, a tabela 2.3 apresenta-se como uma tabela-resumo que sintetiza a informação relevante para o cumprimento dos desafios referidos e propostos na identificação do problema.

Tabela 2.3: Tabela síntese das Plataformas de *Cloud* para a IdC estudadas.

	Apache Hadoop	**Firebase**	**Kaa**
Desafio 1	- Bases de dados relacionais e não-relacionais - Sistema de ficheiros	- Bases de dados relacionais e não-relacionais	- Bases de dados relacionais e não-relacionais
Desafio 3	- Sincronização efetuada através das bases de dados MySQL e MongoDB	- Sincronização em Tempo Real	- Sincronização realizada por meio de um API desenvolvido em Java

3 Desenvolvimento e Protótipo

Este capítulo está divido em três secções: a Arquitetura, a Especificação e o Protótipo. Na primeira, é revelado o diagrama de alto nível da solução desenvolvida. Por sua vez, na segunda parte, é abordada a especificação do projeto, sendo realçadas as interações entre cada um dos constituintes da arquitetura apresentada. Por último, no subcapítulo do Protótipo, são exibidos o ambiente de trabalho, a configuração e a implementação de cada um dos elementos presentes na arquitetura.

3.1 Arquitetura

Tendo como base a arquitetura para a IdC abordada no subcapítulo 1.1 e o estudo efetuado no capítulo 2, tal como se pode verificar por meio da figura 3.1, a arquitetura da solução implementada possui três camadas: a física, a de rede e a de aplicação.

A camada física recolhe, pré-processa e envia para a camada de rede os dados dos dispositivos. A recolha passa pela leitura periódica dos sensores, sendo os dados depois agrupados, formatados e enviados por Bluetooth para a próxima camada.

A camada de rede, representada pelo *middleware*, funciona num *smartphone* e possui o papel de *gateway*, tendo o objetivo de encaminhar os dados, que podem ser armazenados localmente, para a camada de aplicação. A comunicação entre esta camada e a de aplicação é efetuada por meio de uma ligação à *Internet* – nomeadamente, Wi-Fi ou 3G – que utiliza, para esse fim, o protocolo de comunicação MQTT.

Para terminar, a camada de aplicação, representada pela aplicação móvel, possui o objetivo de permitir ao utilizador visualizar os seus dados.

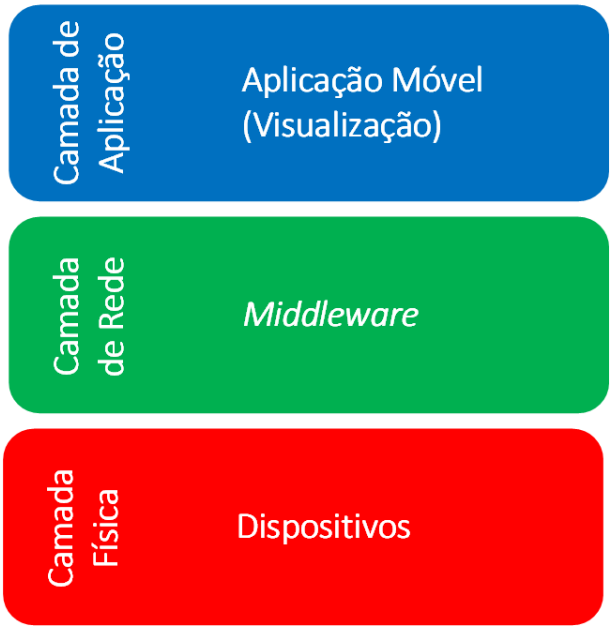

Figura 3.1: Arquitetura da solução implementada.

3.2 Especificação

Na figura 3.2, estão representadas, por meio de um diagrama de sequência de mensagens, de um ponto de vista de alto nível, as interações entre cada um dos constituintes da arquitetura, sendo possível identificar, de forma respetiva, três funcionalidades: a recolha, a sincronização e o envio para a *Cloud* dos dados. Desta forma, a camada física é representada pelos dispositivos, a de rede pelo *middleware* e a de aplicação pela aplicação móvel.

O primeiro conjunto de interações presente na figura representa a funcionalidade de recolha dos dados e o seu objetivo passa pelo envio de um pedido de receção dos dados armazenados nos dispositivos por parte do *smartphone*, cuja resposta contém os dados solicitados. Depois de recebidos, a aplicação indica aos dispositivos que estes devem apagar os dados que já foram recolhidos.

O segundo grupo de relações caracteriza o envio dos dados para a *Cloud* e tem em conta que estes se encontram armazenados na base de dados local da aplicação. Deste modo, a aplicação começa por requerer ao *middleware* autorização para enviar os dados, procedendo, em caso de sucesso, ao respetivo processo de envio. Ao transmitir os dados para o *middleware*, estes são diretamente enviados para a base de dados presente na *Cloud* e exportados, de seguida, para o sistema de ficheiros. Quando os dados locais e os colocados na *Cloud* coincidem, o *middleware* informa a aplicação móvel que o processo terminou.

Por fim, a última associação de troca de mensagens especifica a funcionalidade de sincronização dos dados, assumindo que estes foram colocados na *Cloud* pelo próprio, através de outro dispositivo móvel, ou por intermédio do envio de outro utilizador. No início, a aplicação procede ao envio de um pedido de sincronização, sendo requeridos os dados presentes na *Cloud* que não se encontrem na base de dados local. De seguida, caso a solicitação seja aceite, o *middleware* informa a aplicação e requere à base de dados externa os dados em falta. Por sua vez, esta envia-os, sendo reencaminhados, por intermédio do *middleware*, para a aplicação, até que processo termine. Quando este evento ocorre, o elemento mediador envia para a aplicação uma mensagem de finalização do procedimento.

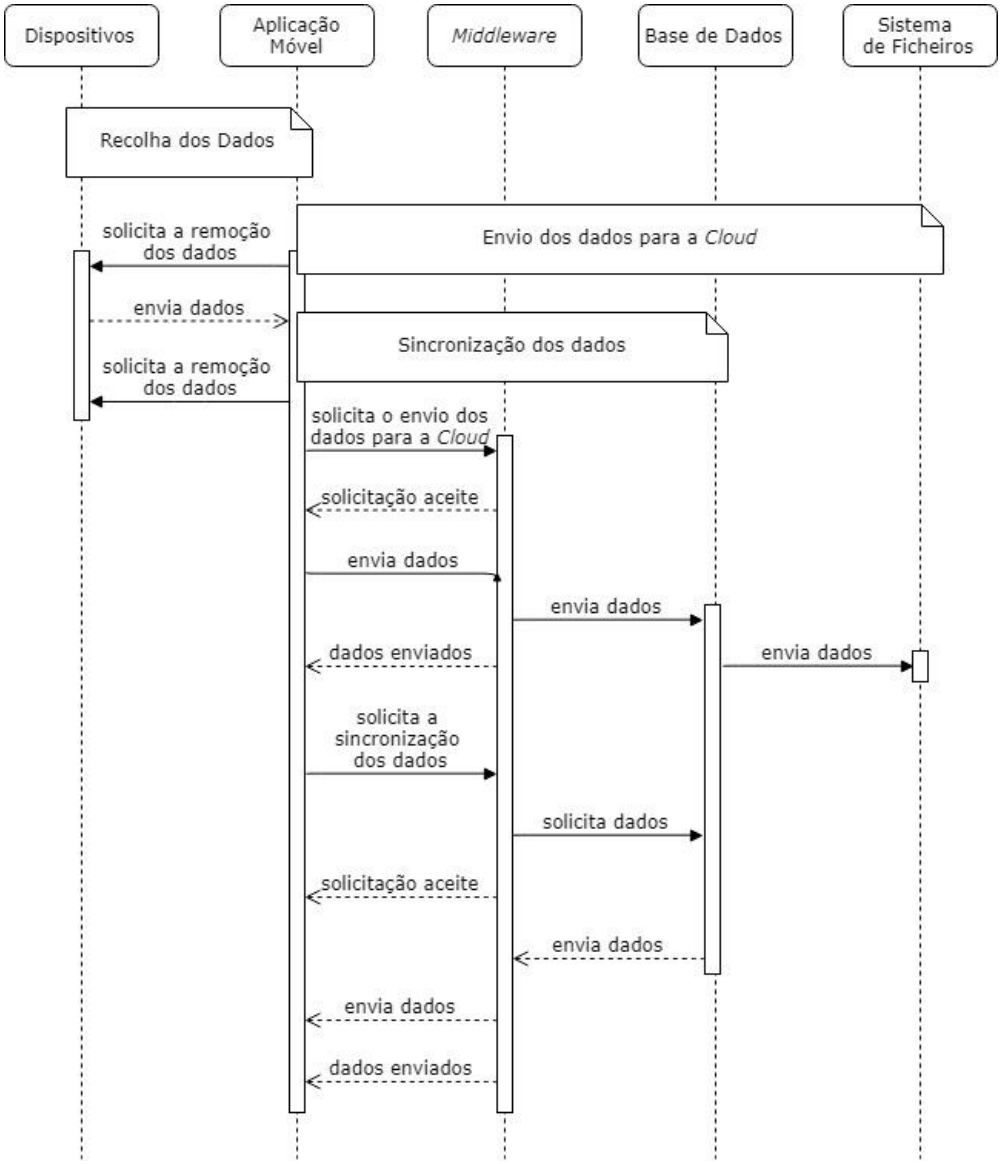

Figura 3.2: Especificação da arquitetura selecionada.

3.3 Protótipo

Tendo em conta a estrutura do capítulo do Estado da Arte, a implementação do projeto e, consequentemente, o presente subcapítulo é composto por três partes: Dispositivos para a IdC, *Middleware* Orientado a Mensagens e Plataforma de *Cloud* para a IdC. Desta forma, em cada uma das divisões, serão abordadas as razões da escolha de um elemento em detrimento de outros, bem como o ambiente de trabalho, a configuração e a implementação de cada componente utilizado.

3.3.1 Dispositivos para a IdC

Ao efetuar a comparação dos elementos de estudo, foi possível verificar que os computadores embebidos possuem uma capacidade de processamento acima da média quando comparados com os nós sensores e com as placas com microcontrolador. Por sua vez, os nós sensores apresentam capacidades de armazenamento limitadas e utilizam protocolos de comunicação que não são comuns em *smartphones*. Desta forma, o tipo de dispositivo selecionado foi a placa com microcontrolador, uma vez que a maior parte do processamento dos dados foi efetuada na camada de rede, sendo, por isso necessária uma capacidade de processamento reduzida. Para além disso, este tipo de placa integra facilmente módulos de comunicação que interagem com o *smartphone*.

3.3.1.1 Ambiente de Trabalho

Esta etapa, correspondente à implementação da camada física enunciada, é representada por três placas com microcontrolador, sendo cada uma composta por um Arduino, um leitor de cartões microSD, um módulo Bluetooth e sensores, como se pode verificar por intermédio da tabela 3.1. Como o dispositivo 2 possui um maior número de sensores, a escolha da placa com microcontrolador recaiu sobre um Arduino Mega, uma vez que este possui maior quantidade de memórias RAM e ROM quando comparado com um Arduino Uno. Para além dos elementos físicos, o código para o funcionamento dos dispositivos foi desenvolvido no Arduino *Integrated Development Environment* (IDE) 1.8.5 na linguagem de programação Wiring.

Tabela 3.1: Caracterização dos Dispositivos da Camada Física.

	Placa com Microcontrolador	**Comunicação (Bluetooth)**	**Armazenamento**	**Sensores**
Dispositivo 1	Arduino Uno	Funduino HC-05	Cartão microSD	DHT-11, *Light Dependent Resistor* (LDR)
Dispositivo 2	Arduino Mega			DHT-11, LDR, SEN-0121, Itead PIR (*Passive InfraRed)*
Dispositivo 3	Arduino Uno			DHT-11, LDR, SEN-0121

3.3.1.2 Configuração

Neste contexto, apenas foi necessário configurar os dispositivos Bluetooth, pelo que, através da figura 3.3, é revelado um exemplo ilustrativo. Deste modo, o primeiro comando enviado para o dispositivo funciona apenas para saber se a ligação foi estabelecida. Por sua vez, o comando "AT+ADDR?" possui o objetivo de obter o endereço físico do dispositivo e o "AT+UART?" a velocidade de transmissão e os bits de paragem e paridade.

```
> AT
OK

> AT+ADDR?
+ADDR:XX:XX:XX:XX:XX:XX
OK

> AT+UART?
+UART:9600,0,0
OK
```

Figura 3.3: Exemplo de configuração de um dispositivo Bluetooth.

3.3.1.3 Implementação

Tendo em conta a constituição dos dispositivos presentes na tabela 3.1, as figuras 3.4, 3.5 e 3.6 revelam os circuitos implementados. Assim, através das figuras, é possível verificar a necessidade de colocar uma resistência de 10 kΩ em série com o LDR, de modo a precaver picos de corrente, e de efetuar um divisor de tensão, com resistências de

1 kΩ e 2 kΩ, no canal de receção do Bluetooth para a tensão descer de 5 para cerca de 3.3 V. Para além disso, os *outputs* do LDR e do DHT-11 são, respetivamente, analógicos e digitais e o leitor de cartões microSD funciona de acordo com o protocolo SPI.

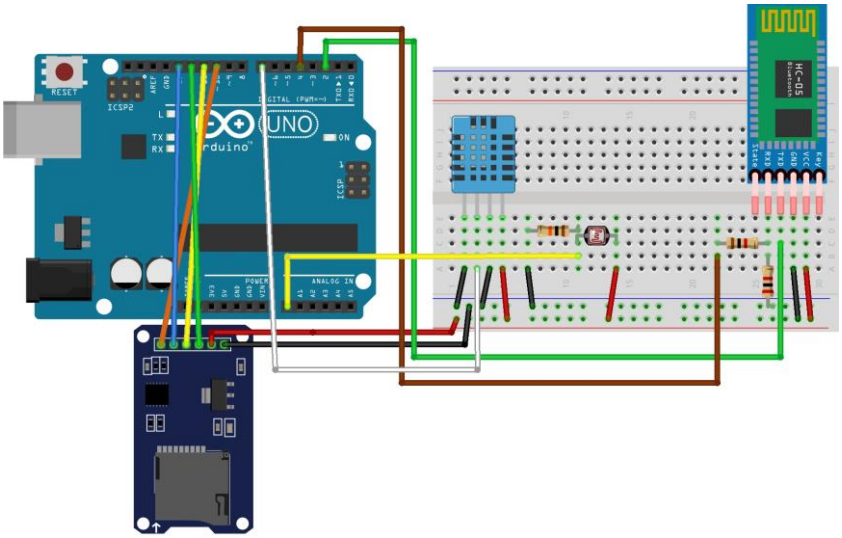

Figura 3.4: Constituição do Dispositivo 1.

Entre as implementações dos dispositivos 1 e 3, a única diferença presente passa pela introdução do sensor de vapor na montagem do terceiro dispositivo. Através da inspeção da figura 3.5, verifica-se que o referido sensor possui *output* analógico.

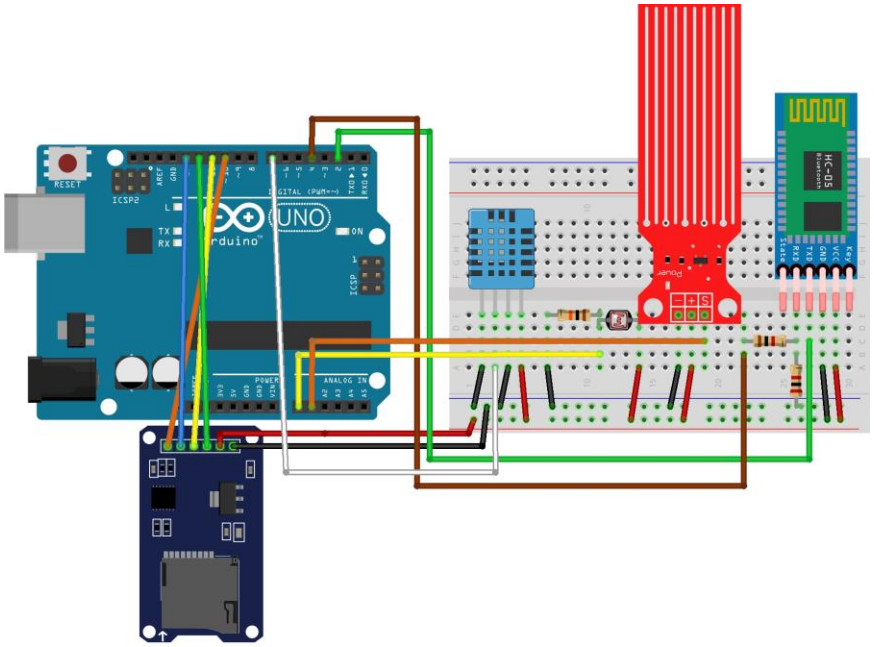

Figura 3.5: Constituição do Dispositivo 3.

Por fim, a montagem do dispositivo 2, ilustrada pela figura 3.6, possui, ao contrário das outras duas, um Arduino Mega e, em adição aos sensores do dispositivo 3, um PIR, cuja leitura se caracteriza pelo seu cariz digital, uma vez que apenas deteta, ou não, a presença de movimento.

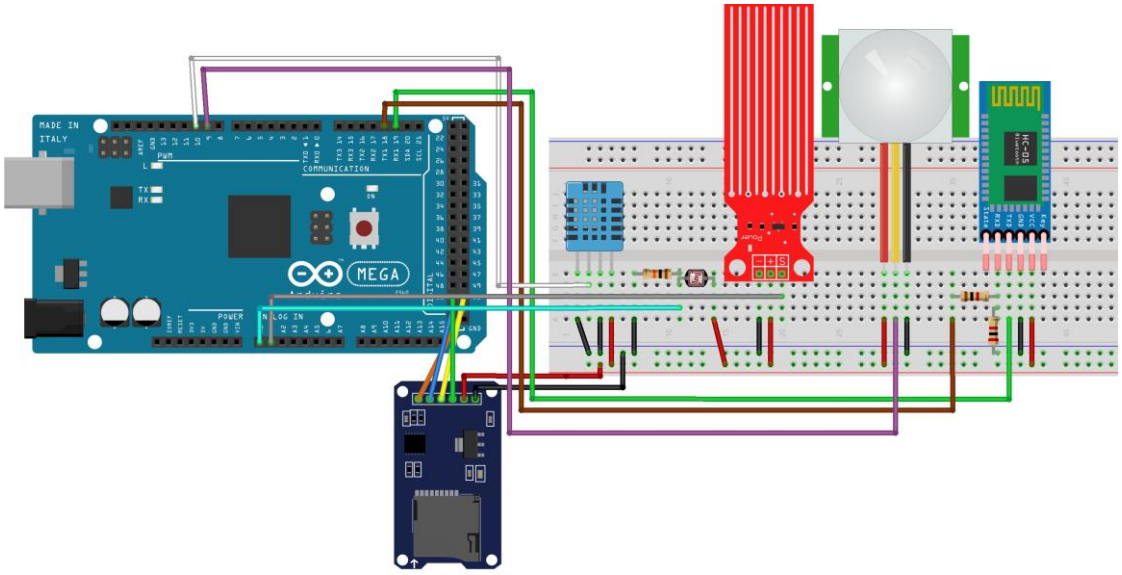

Figura 3.6: Constituição do Dispositivo 2.

3.3.2 *Middleware* Orientado a Mensagens

Tendo como base de comparação o Node-RED, o Apache Kafka e o RabbitMQ, foi possível concluir que o primeiro é integrável com vários protocolos de comunicação e apresenta uma plataforma de programação visual por blocos, o que o torna acessível e editável por um maior número de utilizadores. Pelo contrário, o Apache Kafka apenas opera segundo o protocolo de comunicação TCP e, tal como o RabbitMQ, necessita de de maiores conhecimentos de programação, sendo apenas destinada a utilizadores mais técnicos. Deste modo, a escolha para *Middleware* Orientado a Mensagens recaiu sobre o Node-RED.

3.3.2.1 Ambiente de Trabalho

Esta etapa, que corresponde à camada de rede, corre num *smartphone* que possui a versão 7.0 do SO Android e é composto pelo *middleware* e por uma aplicação, cuja lógica e parte visual são efetuadas, respetivamente, através da linguagem de programação Java e da linguagem de marcação XML. A aplicação funciona como *gateway* e permite a exibição e o armazenamento de dados, bem como o envio e a receção de comandos MQTT. O *middleware* funciona com a versão 0.18.4 do Node-RED – que utiliza as

versões 8.9.4 do Node.js e 5.7.1 do *Node Package Manager* (NPM) –, através da linha de comandos disponibilizada pela versão 0.60 do Termux, e o ambiente de programação selecionado para a codificação da aplicação móvel foi o Android Studio v3.0.1 para Windows 10. A escolha das versões recaiu sobre as últimas que apresentam maior grau de estabilidade.

3.3.2.2 Configuração

Nesta etapa, apenas foi necessário proceder à configuração do ambiente envolvente do Node-RED, uma vez que o Android Studio apenas necessita de uma instalação prévia do Java para pleno funcionamento.

Desta forma, a instalação do Node-RED possui os passos presentes na figura 3.7: atualização do repositório de aplicações do Linux com vista a proporcionar a possibilidade de instalar as versões de *software* mais recentes e respetiva instalação (1); instalação dos componentes de interação com as diretorias, de edição de ficheiros de texto e do Node.js, respetivamente (2); instalação da última versão do NPM (3); instalação do Node-RED (4); e início do serviço Node-RED (5).

```
1.  apt update && apt upgrade -y
2.  apt install coreutils nano nodejs
3.  npm install -g npm@latest
4.  npm i -g --unsafe-perm node-red
5.  node-red
```

Figura 3.7: Instruções para a instalação do Node-RED e componentes necessários.

3.3.2.3 Implementação

Uma vez concluída a configuração dos elementos escolhidos como ambiente de trabalho, a próxima etapa passa pela implementação da aplicação Android e pelo desenvolvimento do *middleware*, utilizando o Node-RED, que é uma plataforma de programação visual e orientada por fluxos de mensagens.

3.3.2.3.1 Ambiente Android

Quando a aplicação é inicializada, esta disponibiliza as opções de *log-in* e de registo – presentes na figura 3.8 –, sendo que a primeira apenas pode ser usada por utilizadores que já se encontrem registados. O registo na aplicação é necessário com vista à identificação dos dispositivos de cada utilizador e, depois de ser efetuado, a aplicação procede, de forma automática, ao seu *log-in*.

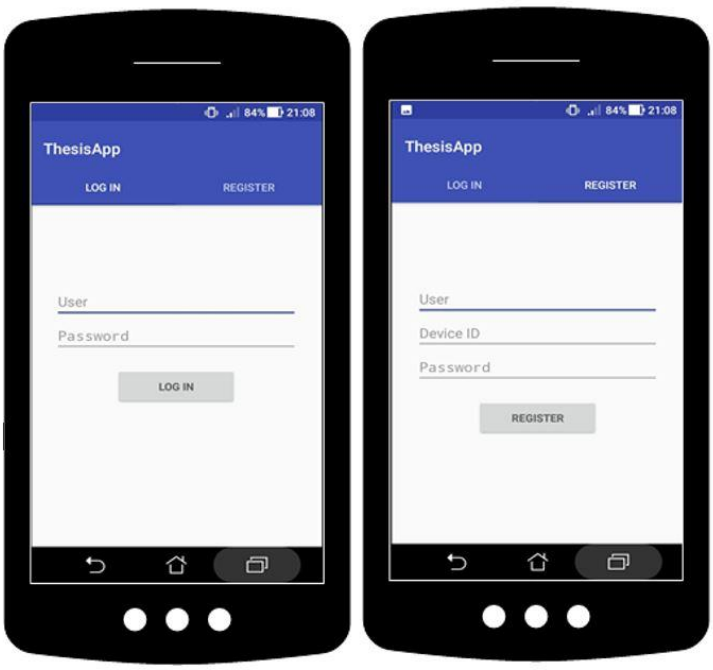

Figura 3.8: Android – Funcionalidades de *Log-In* e de Registo.

Depois de o utilizador ter efetuado o *log-in* na aplicação, este passa a ter acesso aos dados e às opções presentes no menu, tal como se pode verificar, respetivamente, através das figuras 3.9 e 3.10.

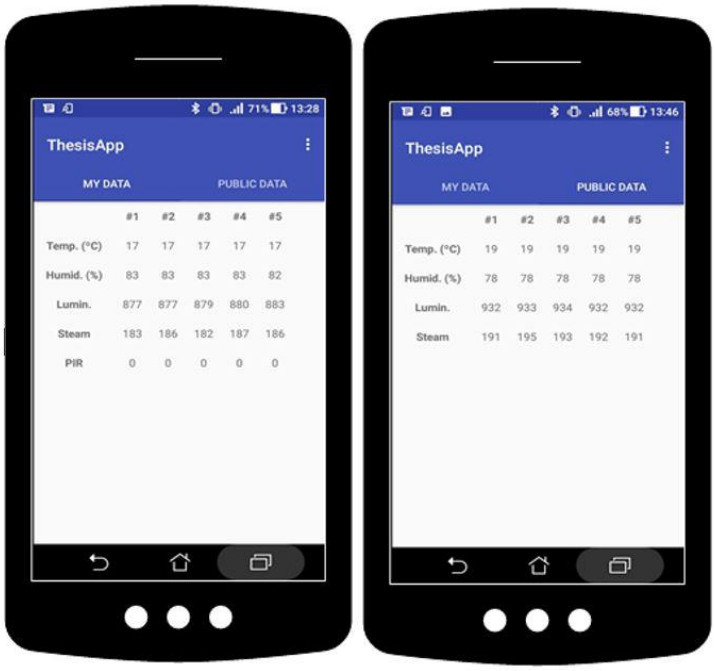

Figura 3.9: Visualização dos dados do utilizador e dos dados públicos.

As opções presentes no menu representado na figura 3.10 contemplam a ativação do Bluetooth, a obtenção de dados provenientes do Arduino, a sincronização dos dados com os presentes na *Cloud*, o envio de novos dados para a *Cloud* e o *log-out*. A primeira opção referida possui o objetivo de ligar, com a devida autorização do *smartphone*, o Bluetooth, de forma a possibilitar a receção de dados. Por conseguinte, a segunda opção presente no menu permite estabelecer uma conexão Bluetooth entre os dispositivos e o *smartphone*, com vista à obtenção dos referidos dados. As opções 3 e 4 permitem que a aplicação se ligue ao *middleware* por meio do protocolo de comunicação MQTT, que utiliza um servidor de livre-acesso do projeto IoT Eclipse, com a finalidade de receber e enviar, respetivamente, os dados de e para a *Cloud*. Por fim, a última opção autoriza o utilizador a efetuar o *log-out*.

Figura 3.10: Android – Opções presentes no menu.

De forma a correr o Node-RED no *smartphone*, foi utilizada a aplicação Termux, que disponibiliza uma linha de comandos baseada na dos SOs Unix. Assim, como se pode averiguar mediante a análise da figura 3.11, apenas é necessário escrever no terminal "node-red" para que o *middleware* fique operacional.

Figura 3.11: Termux – Node-RED em funcionamento.

3.3.2.3.2 Node-RED

A figura 3.12 expõe um exemplo representativo do fluxo de dados desenvolvido para um utilizador, sendo, no âmbito desta implementação, este estendido para permitir o funcionamento com três utilizadores. Assim, os fluxos de dados caracterizados pela figura foram desenhados para implementarem as funcionalidades de sincronização, i.e., da receção dos dados da *Cloud*, e de envio dos dados para a *Cloud*. Adicionalmente, o processo de envio de dados é constituído pelos três últimos ramos do fluxo, sendo apenas analisado um deles, uma vez que o seu funcionamento é igual, com a exceção do utilizador para o qual a funcionalidade é executada. Nos sub-fluxos que trocam comandos e/ou dados com a aplicação Android, foi utilizado, tal como referido na secção anterior, o protocolo de comunicação MQTT, por meio de um servidor de livre-acesso do projeto IoT Eclipse.

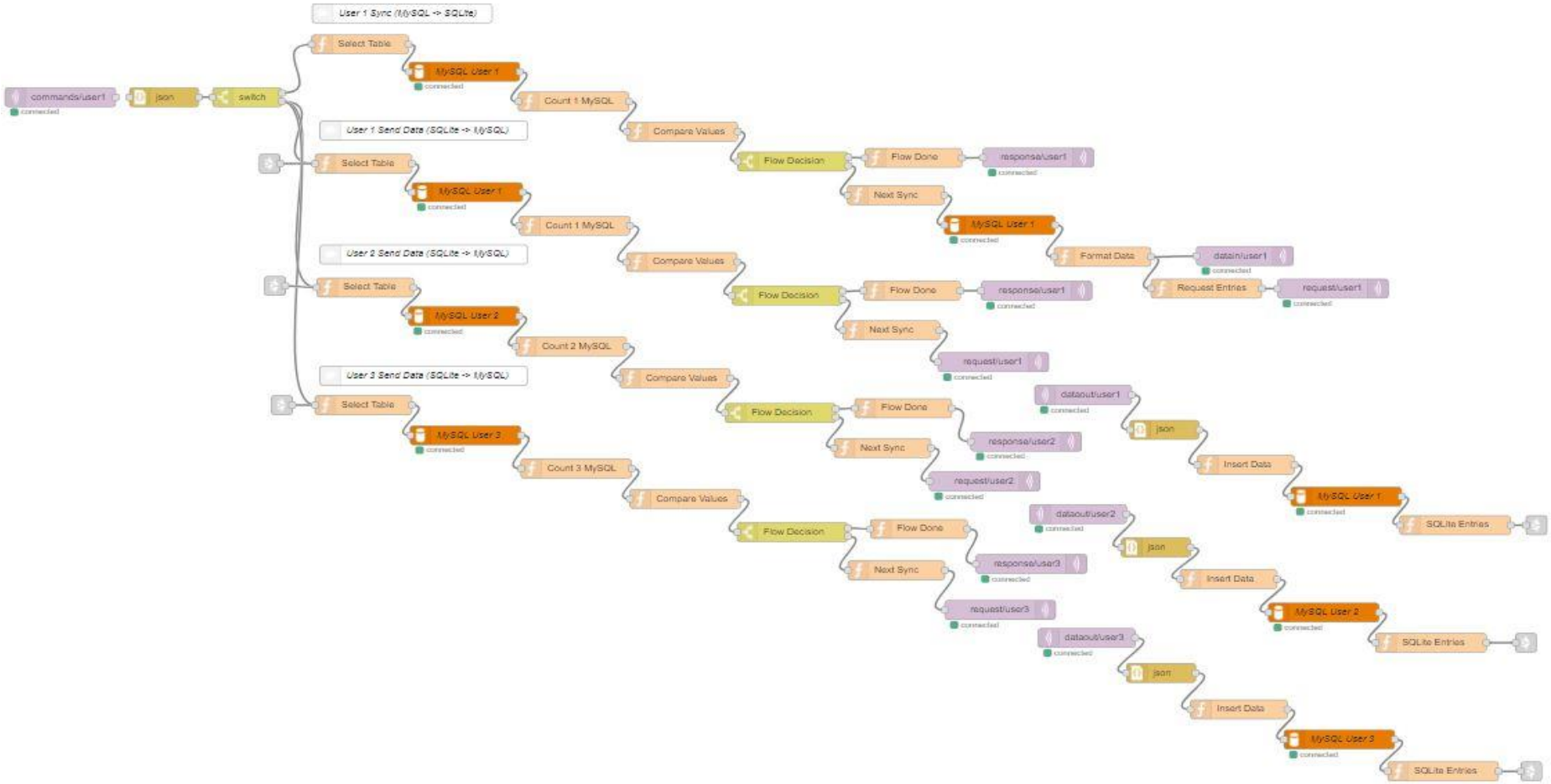

Figura 3.12: Funcionalidades de sincronização e envio de dados para a Cloud.

Os fluxos podem ser, respetivamente, decompostos em cinco e seis sub-fluxos, sendo o primeiro comum a ambos. Desta forma, o primeiro sub-fluxo, ilustrado pela figura 3.13, é constituído por três nós: o primeiro possui o objetivo de receber mensagens provenientes da aplicação Android, cujo conteúdo é composto pelo ID do dispositivo que enviou a mensagem, o número de entradas na tabela da base de dados local SQLite e o comando da ação que se pretende efetuar; o segundo converte a mensagem recebida numa *String* com formato JSON para um objeto JS; e o último, tendo em conta o comando recebido, decide se o fluxo de dados segue, respetivamente, o caminho da sincronização ou do envio dos dados para a *Cloud*.

Figura 3.13: Receção dos comandos enviados pela aplicação Android.

Por sua vez, o segundo sub-fluxo, representado pela figura 3.14, é bastante semelhante para ambos os fluxos, tendo apenas uma diferença no penúltimo nó. O primeiro nó apenas está presente no fluxo de envio de dados para a *Cloud* e representa o seguimento de um ciclo, verificável por meio da figura 3.20, começado aquando da realização do referido envio. O segundo, o terceiro e o quarto nós possuem o intuito de obter o número de entradas na tabela da base de dados MySQL e são responsáveis, respetivamente, por estruturar e realizar a *query* à base de dados e por intrepertar a resposta resultante. Os nós seguintes têm, de forma respetiva, a seu cargo a tarefa de comparar o número de entradas presente em cada uma das tabelas das bases de dados local e da *Cloud* e de verificar se o processo de receção ou envio de dados para a *Cloud* solicitado pela aplicação Android foi concluído.

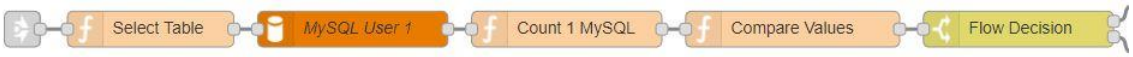

Figura 3.14: Comparação das entradas das tabelas MySQL e SQLite.

O terceiro sub-fluxo, visível na figura 3.15, surge como resultado da conclusão do processo de receção ou envio de dados para a *Cloud* solicitado pela aplicação Android e é idêntico em ambos os fluxos. A única diferença está relacionada com a mensagem enviada para a aplicação, uma vez que um deles informa que a sincronização foi finalizada, enquanto que o outro comunica o fim do envio dos dados.

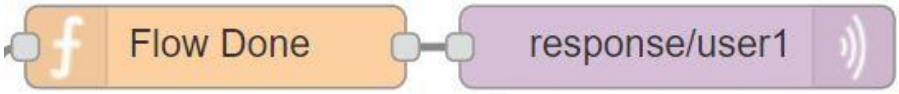

Figura 3.15: Indicação à aplicação Android da conclusão do processo solicitado.

As semelhanças entre os sub-fluxos terminam com o outro caminho que pode ser tomado pelo sub-fluxo 2. Desta forma, a etapa seguinte do fluxo de sincronização, caracterizada na figura 3.16, passa pela estruturação (nó 1) e realização (nó 2) da *query* a efetuar à base de dados MySQL para a da próxima entrada presente na tabela e pela formatação dos dados recolhidos, de modo a serem interpretados pela aplicação móvel (nó 3).

Figura 3.16: Recolha e formatação da próxima entrada da tabela MySQL.

Uma vez concluída a formatação dos dados, estes são enviados para a aplicação Android, que se encarrega de os armazenar localmente. Adicionalmente, é efetuada uma solicitação à referida aplicação do número de entradas existentes na tabela da base de dados local, sendo este fluxo realizado de novo caso ainda existam dados a sincronizar. As duas tarefas descritas podem ser verificadas por intermédio da figura 3.17.

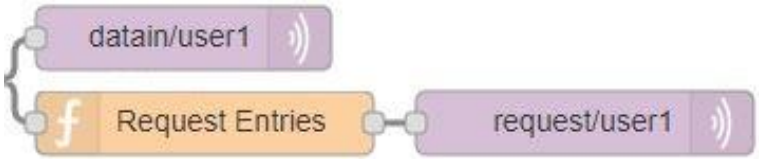

Figura 3.17: Envio dos dados e solicitação do número de entradas da tabela SQLite.

No que concerne ao fluxo do envio dos dados para a *Cloud*, este, caso a sua ação não tenha sido finalizada, prepara-se, como se pode inspecionar mediante a análise da figura 3.18, para a próxima iteração do ciclo de envio dos dados, requerendo ao ambiente Android os dados contidos na próxima entrada da tabela a transmitir para a *Cloud*.

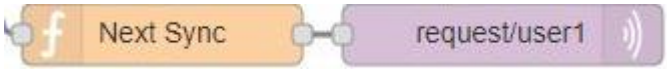

Figura 3.18: Solicitação da próxima entrada na tabela da base de dados SQLite.

Depois de enviado o pedido dos próximos dados contidos na tabela da base de dados SQLite, estes, como se pode averiguar por meio da figura 3.19, são recebidos pelo primeiro nó deste sub-fluxo, sendo, posteriormente, interpretados e convertidos num objeto JS. Quando este processo de conversão é finalizado, estes dados são introduzidos, de forma adequada, na base de dados MySQL.

Figura 3.19: Receção dos dados e respetiva inserção na base de dados MySQL.

Por último, o sub-fluxo ilustrado pela figura 3.20 revela-se como a continuação do anterior e apenas tem o intuito de adicionar o número de elementos presentes na tabela da base de dados SQLite ao objeto da mensagem a enviar para o próximo nó. Depois de a mensagem ser transmitida pelo último nó presente neste sub-fluxo, o fluxo de dados continua no que está representado na figura 3.14.

Figura 3.20: Envio do número de entradas na tabela da base de dados SQLite.

3.3.3 Plataforma de *Cloud* para a IdC

Ao comparar os elementos de estudo relativos a esta temática, a conclusão tirada foi que a versão gratuita do Firebase possui limitações ao nível dos serviços e da infraestrutura e o Kaa, apesar de poder ser corrido localmente, disponibiliza recursos não necessários. Desta forma, o Apache Hadoop foi a plataforma de *Cloud* para a IdC selecionada, uma vez que este pode ser implementado numa instância local, apenas com as funcionalidades desejadas e sem barreiras em termos de preço.

3.3.3.1 Ambiente de Trabalho

A plataforma é composta por uma máquina virtual que contém o Apache Hadoop, uma base de dados do tipo MySQL e uma ferramenta de exportação de dados – o Apache Sqoop.

A máquina virtual corre na versão 5.2.6 r120293 do *software* Oracle VM VirtualBox, utiliza a versão de 32 bits do SO Ubuntu 14.04 LTS baseado em Linux e possui 10 GB para armazenamento de dados, 1.5 GB de memória RAM e uma configuração de rede do tipo *Bridged Adapter*, pois esta permite que o ambiente da

máquina virtual possua um endereço IP próprio. O ambiente Linux foi escolhido por ser mais leve e por possuir uma linha de comandos mais intuitiva, a opção sobre o Ubuntu 14.04 LTS baseia-se na comunidade de suporte, na sua estabilidade e na familiarização do autor da presente dissertação com este tipo de SO e a versão de 32 bits foi a selecionada por serem necessários menores recursos de memória RAM.

Por sua vez, por possuírem versões estáveis, foram utilizados o Apache Hadoop 2.7.4, o MySQL 5.5.58 e o Apache Sqoop 1.4.6. A escolha da base de dados recaiu sobre uma do tipo relacional, uma vez que os dados armazenados são caracterizados pela sua estrutura fixa. Como consequência, dado que a sua função passa pela exportação de dados de bases de dados relacionais para o Apache Hadoop, o Apache Sqoop foi a ferramenta escolhida.

3.3.3.2 Configuração

Nesta secção, são analisados e replicados todos os passos necessários para a instalação e configuração do Apache Hadoop, da base dados MySQL e do Apache Sqoop.

3.3.3.2.1 Apache Hadoop

A primeira etapa da configuração deste elemento passa pela instalação do *Secure Shell* e da versão 8 do Java. A habilitação do *Secure Shell* é necessária para operações de inicialização, paragem e outras operações efetuadas na linha de comandos do *cluster*. Por sua vez, o Java é indispensável para correr o Apache Hadoop, uma vez que este é desenvolvido tendo esta linguagem como base.

Desta forma, para se proceder à instalação do *Secure Shell*, é necessário abrir uma instância da linha de comandos. Uma vez aberta, digitam-se, respetivamente, as instruções para a instalação (1) e o reinício (2) do serviço, presentes na figura 3.21.

```
1. sudo apt-get install openssh-server
2. sudo service ssh restart
```

Figura 3.21: Instruções para a instalação do *Secure Shell*.

Antes do fim desta etapa, é ainda relevante efetuar os passos contidos na figura 3.22: colocar a linha de comandos a apontar para a diretoria onde se vão realizar as operações pretendidas (1); gerar um par chave-valor para a utilização do *Secure Shell* (2); colocar as chaves geradas no ficheiro pretendido (3); e atribuir as permissões de leitura e escrita ao ficheiro que contém as chaves geradas (4).

```
1.  cd /home/thesis/.ssh
2.  ssh-keygen -t rsa
3.  cat /home/thesis/.ssh/authorized_keys
4.  chmod 0600 ~/.ssh/authorized_keys
```

Figura 3.22: Geração de um par chave-valor para a utilização do *Secure Shell*.

Como se pode verificar pela figura 3.23, a configuração do Java passa pela obtenção (1) e disponibilização (2) do respetivo ficheiro, seguida da instalação (3) e da colocação correta e automática das variáveis de ambiente convenientes (4).

```
1.  sudo add-apt-repository ppa:webupd8team/java -y
2.  sudo apt-get update
3.  sudo apt-get install oracle-java8-installer
4.  sudo apt-get install oracle-java8-set-default
```

Figura 3.23: Instruções para a instalação do Java.

Para fins demonstrativos, a instalação do Apache Hadoop efetuada segue o modo pseudo-distribuído, que consiste numa simulação realizada numa única máquina de forma distribuída. Deste modo, a instalação possui os passos verificáveis por intermédio da figura 3.24: entrar (1) e sair (7) do modo de super-utilizador, que é equivalente ao modo de administrador; colocar a linha de comandos a apontar para a diretoria onde se vão realizar as operações pretendidas (2); realizar o *download* do ficheiro de instalação (3); descompactar o ficheiro de instalação (4); criar uma nova diretoria com uma designação mais adequada (5); e mover o conteúdo descompactado para a diretoria gerada (6).

```
1.  sudo su
2.  cd /usr/local
3.  wget http://mirrors.fe.up.pt/pub/apache/hadoop/common/hadoop-2.7.4/hadoop-2.7.4.tar.gz
4.  tar -xzf hadoop-2.7.4.tar.gz
5.  mkdir hadoop/
6.  mv hadoop-2.7.4/* hadoop/
7.  exit
```

Figura 3.24: Obtenção do Apache Hadoop.

A próxima etapa passa pela colocação correta das variáveis de ambiente do Java e do Apache Hadoop, de modo a indicar os locais dos ficheiros de instalação. Para executar a referida tarefa, é necessário abrir o ficheiro conveniente (sudo gedit ~/.bashrc) e, depois de adicionadas as linhas de código presentes na figura 3.25, efetivar as modificações (source ~/.bashrc).

```
# Java Environment Variables
export JAVA_HOME=/usr/lib/jvm/java-8-oracle
export PATH=$PATH:$JAVA_HOME/bin

# Hadoop Environment Variables
export HADOOP_HOME=/usr/local/hadoop
export HADOOP_MAPRED_HOME=$HADOOP_HOME
export HADOOP_COMMON_HOME=$HADOOP_HOME
export HADOOP_HDFS_HOME=$HADOOP_HOME
export YARN_HOME=$HADOOP_HOME
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native
export PATH=$PATH:$HADOOP_HOME/sbin:$HADOOP_HOME/bin
export HADOOP_INSTALL=$HADOOP_HOME
```

Figura 3.25: Configuração das variáveis de ambiente do Java e do Apache Hadoop.

Tendo o Apache Hadoop sido instalado, os quatro passos seguintes estão relacionados com a sua configuração. Desta forma, é pertinente alterar a diretoria de trabalho utilizando a instrução "cd $HADOOP_HOME/etc/hadoop". O primeiro ficheiro de configuração que necessita de ser editado é o core-site.xml, sendo, para isso, essencial digitar o seguinte conjunto de comandos no *shell*: "sudo gedit core-site.xml". Assim, ao analisar o código evidenciado pela figura 3.26, é possível verificar que o endereço definido para o HDFS está contido entre os campos *value*.

```
<configuration>
    <property>
        <name>fs.default.name</name>
        <value>hdfs://localhost:9000</value>
    </property>
</configuration>
```

Figura 3.26: Edição do ficheiro core-site.xml.

Para configurar o HDFS, é necessário acrescentar as linhas de código destacadas na figura 3.27 ao ficheiro hdfs-site.xml (sudo gedit hdfs-site.xml). A primeira propriedade verificável define o fator de replicação do sistema de ficheiros como 1. Por sua vez, as duas últimas propriedades indicam, respetivamente, os endereços para o *Name* e o *Data Nodes*.

```
<configuration>
    <property>
        <name>dfs.replication</name>
        <value>1</value>
    </property>

    <property>
        <name>dfs.name.dir</name>
        <value>file:///home/thesis/hadoopinfra/hdfs/namenode</value>
    </property>

    <property>
        <name>dfs.data.dir</name>
        <value>file:///home/thesis/hadoopinfra/hdfs/datanode</value>
    </property>
</configuration>
```

Figura 3.27: Edição do ficheiro hdfs-site.xml.

A próxima fase passa pela adição do conjunto de comandos presente na figura 3.28 ao ficheiro yarn-site.xml (sudo gedit yarn-site.xml). O primeiro atributo passa por definir o MR como serviço do YARN e o segundo a respetiva classe implementada em Java.

```
<configuration>
    <property>
        <name>yarn.nodemanager.aux-services</name>
        <value>mapreduce_shuffle</value>
    </property>

    <property>
        <name>yarn.nodemanager.auxservices.mapreduce.shuffle.class</name>
        <value>org.apache.hadoop.mapred.ShuffleHandler</value>
    </property>
</configuration>
```

Figura 3.28: Edição do ficheiro yarn-site.xml.

Para finalizar a configuração, é necessário digitar, na linha de comandos, as instruções salientadas na figura 3.29. Assim, é copiado o modelo do ficheiro de configuração do MR para o mapred-site.xml (1) e, de seguida, este é aberto para edição (2).

```
1.  sudo cp mapred-site.xml.template mapred-site.xml
2.  sudo gedit mapred-site.xml
```

Figura 3.29: Geração do ficheiro mapred-site.xml.

Uma vez aberto, é adicionada a propriedade explicitada na figura 3.30 para que seja definido o YARN como ferramenta para a execução do MR.

```
<configuration>
    <property>
        <name>mapreduce.framework.name</name>
        <value>yarn</value>
    </property>
</configuration>
```

Figura 3.30: Edição do ficheiro mapred-site.xml.

A próxima etapa tem como objetivo indicar ao Apache Hadoop onde se encontra o Java, sendo, para isso, necessário substituir, no *script* hadoop-env.sh implementado em Bash, o comando ${JAVA_HOME} pelo caminho do Java: "/usr/lib/jvm/java-8-oracle". De seguida, ativam-se as permissões de leitura e escrita na diretoria onde está localizado o Apache Hadoop (sudo chmod 777 -R /usr/local/hadoop) e formata-se o *Name Node* do sistema de ficheiros, por intermédio da introdução da instrução "hdfs namenode -format" na linha de comandos. Por fim, inicializam-se o sistema de ficheiros e o YARN através da execução dos seguintes *scripts* Bash pelo *shell*: start-dfs.sh e start-yarn.sh.

3.3.3.2.2 MySQL

Como se pode averiguar através da inspeção da figura 3.31, a configuração da base de dados MySQL é um processo que exige poucas etapas: atualização do repositório de aplicações do Ubuntu com vista a proporcionar a possibilidade de instalar a versão mais recente do MySQL (1); instalação do servidor para a base de dados (2); seleção da instalação segura (3); instalação da base de dados (4); e início do serviço MySQL (5).

```
1.  sudo apt-get update
2.  sudo apt-get install mysql-server
3.  sudo mysql_secure_installation
4.  sudo mysql_install_db
5.  sudo service mysql start
```

Figura 3.31: Instruções para a instalação do MySQL.

Com vista a tornar a base de dados acessível ao exterior, é necessário editar o ficheiro "/etc/mysql/my.cnf", atribuindo o valor 0.0.0.0 à propriedade *bind-address*. Para finalizar, reinicia-se o serviço por meio da instrução "sudo service mysql restart".

3.3.3.2.3 Apache Sqoop

Tal como para o Apache Hadoop, para esta ferramenta funcionar em pleno, é fundamental possuir uma instalação do Java. No entanto, este passo já foi efetuado na configuração do Apache Hadoop. Desta forma, a configuração necessária para o Apache Sqoop compreende a execução das instruções exibidas na figura 3.32: entrar no modo de

super-utilizador (1); colocar a linha de comandos a apontar para a diretoria onde se vão realizar as operações pretendidas (2); proceder ao *download* do ficheiro de instalação (3); descompactar o ficheiro de instalação (4); criar uma nova diretoria com uma designação mais adequada (5); mover o conteúdo descompactado para a diretoria gerada (6); e sair do modo de super-utilizador (7).

```
1.  sudo su
2.  cd /usr/lib
3.  wget http://mirrors.up.pt/pub/apache/sqoop/1.4.6/sqoop-1.4.6.bin__hadoop-2.0.4-alpha.tar.gz
4.  tar -xvf sqoop-1.4.6.bin__hadoop-2.0.4-alpha.tar.gz
5.  mkdir sqoop/
6.  mv sqoop-1.4.6.bin__hadoop-2.0.4-alpha/* sqoop/
7.  exit
```

Figura 3.32: Instruções para a instalação do Apache Sqoop.

A próxima etapa passa pela colocação correta das variáveis de ambiente do Apache Sqoop, de modo a indicar os locais dos ficheiros de instalação. Para executar a referida tarefa, é necessário abrir o ficheiro conveniente (sudo gedit ~/.bashrc) e, depois de adicionadas as linhas de código presentes na figura 3.33, tornar as alterações efetivas (source ~/.bashrc).

```
# Sqoop Environment Variables
export SQOOP_HOME=/usr/lib/sqoop
export PATH=$PATH:$SQOOP_HOME/bin
```

Figura 3.33: Configuração das variáveis de ambiente do Apache Sqoop.

Por fim, coloca-se o conector que liga o MySQL e o Apache Sqoop na localização correta, podendo esta tarefa ser realizada por intermédio do conjunto de comandos detalhados na figura 3.34: entrar no modo de super-utilizador (1); colocar a linha de comandos a apontar para a diretoria onde se vão realizar as operações pretendidas (2); descompactar a diretoria que contém o conector (3); colocar a linha de comandos a apontar para a diretoria onde se encontra o conector (4); copiar o ficheiro executável Java para a devida diretoria (5); e sair do modo de super-utilizador (6).

```
1.  sudo su
2.  cd /usr/lib/sqoop/lib
3.  tar -zxf mysql-connector-java-5.1.45.tar.gz
4.  cd mysql-connector-java-5.1.45
5.  cp mysql-connector-java-5.1.45-bin.jar /usr/lib/sqoop/lib
6.  exit
```

Figura 3.34: Configuração do conector entre o MySQL e o Apache Sqoop.

3.3.3.3 Implementação

Uma vez concluída a fase de configuração de todos os elementos, as próximas etapas passam por colocar o ambiente em pleno funcionamento. Desta forma, a primeira tarefa consiste na realização dos passos presentes na figura 3.35: formatar o nó que contém o sistema de ficheiros, i.e., o *Name Node* do HDFS, onde serão armazenados os dados (1); ligar o serviço que gere o HDFS (2); e inicializar o serviço relativo ao YARN (3).

```
1.  hadoop namenode -format
2.  start-dfs.sh
3.  start-yarn.sh
```

Figura 3.35: Passos para o funcionamento do ambiente Apache Hadoop.

Finalizada a primeira etapa, a próxima tem como intuito gerar os utilizadores e respetivas bases de dados e tabelas MySQL. Assim, as instruções representadas pela figura 3.36 encontram-se generalizadas para um utilizador, sendo exibido o "userX" como exemplo representativo, em que X={1, 2, 3}. Deste modo, para atingir o propósito referido, é necessário proceder à introdução, na linha de comandos, das instruções ilustradas na figura: entrar no serviço MySQL (1); criar o utilizador "userX", de forma a que este possa ser utilizado em qualquer IP a que a base de dados esteja ligada ('%'), com a palavra-passe "userX" (2); atribuir ao mesmo utilizador todos os privilégios necessários para efetuar as ações pretendidas sobre a base de dados (3); ativar os privilégios recebidos (4); criar, se ainda não existir, a base de dados pertencente ao utilizador "userX" (5); sair do ambiente MySQL (6); voltar a entrar no serviço MySQL, efetuando o *log-in* como "userX" no IP local da *Cloud* – em que o valor de "YY" é dinâmico –, de modo a este estar acessível ao exterior (7); aceder à base de dados do utilizador (8); e criar a tabela para armazenamento de dados dos dispositivos, utilizando a estrutura ilustrada (9).

```
1.  mysql
2.  create user 'userX'@'%' identified by 'userX';
3.  grant all privileges on * . * to 'userX'@'%';
4.  flush privileges;
5.  create database if not exists userX;
6.  exit;
7.  mysql -u userX -p -h 192.168.1.YY
8.  use userX;
9.  create table readings(_id INTEGER UNIQUE NOT NULL PRIMARY KEY
    AUTO_INCREMENT, timestamp TEXT, type TEXT, value TEXT);
```

Figura 3.36: Geração de utilizadores e respetivas bases de dados e tabelas MySQL.

Uma vez criados os utilizadores e as respetivas bases de dados e tabelas MySQL, passa a ser possível armazenar os dados que os dispositivos vão gerando. Assim, tal como mostra a figura 3.37, estando no ambiente da base de dados, é possível verificar os dados lá contidos por meio da instrução "SELECT * FROM readings", em que "readings" é o nome atribuído à tabela criada.

```
mysql> SELECT * FROM readings;
+-----+---------------------+-------------+-------+
| _id | timestamp           | type        | value |
+-----+---------------------+-------------+-------+
|   1 | 2395-01-01 10:00:20 | temperature | 19    |
|   2 | 2039-01-01 10:00:20 | luminosity  | 14    |
|   3 | 2469-01-01 10:00:20 | humidity    | 20    |
|   4 | 2019-01-01 10:00:20 | humidity    | 14    |
|   5 | 2039-01-01 10:00:20 | luminosity  | 534   |
|   6 | 2569-01-01 10:00:20 | humidity    | 34    |
|   7 | 2149-01-01 10:00:20 | luminosity  | 939   |
|   8 | 2075-01-01 10:00:20 | temperature | 34    |
+-----+---------------------+-------------+-------+
```

Figura 3.37: Exemplo de dados armazenados na base de dados MySQL.

Tendo em conta esta possibilidade, o passo seguinte está relacionado com a exportação de dados efetuada da base de dados para o sistema de ficheiros do ambiente Apache Hadoop. Assim, de forma a permitir a exportação de dados, procedeu-se ao desenvolvimento de um *script* baseado na ferramenta Apache Sqoop e que se encontra caracterizado na figura 3.38. Os passos presentes na figura possuem, respetivamente, o intuito de construir a tarefa de exportação e de a executar. Ao observar a figura, constata-se que a tarefa de exportação, denominada "SQOOP_IMPORT", atua sobre a base de dados MySQL "userX" que se encontra no servidor local, utilizando o *log-in* do mesmo utilizador. Para além disso, o *script* corre sobre a tabela definida para conter os dados dos dispositivos e adiciona os dados ao HDFS de forma a que estes sejam introduzidos após os anteriores, evitando a repetição dos dados, ao verificar o número da última entrada armazenada na tabela. Dado que este processo não revela um nível de computação muito elevado, este utiliza apenas uma tarefa de MR. Por fim, depois de definido o *script*, este é executado por meio da segunda instrução presente na mesma figura.

```
1.  sqoop job --create SQOOP_IMPORT -- import
    --connect jdbc:mysql://localhost/userX
    --username userX
    --password userX
    --table readings
    --incremental append
    --check-column _id
    --m 1
2.  sqoop job --exec SQOOP_IMPORT
```

Figura 3.38: Passos para construir (1) e executar (2) o *script* Apache Sqoop.

Por uma questão de simplicidade e automação do processo de exportação de dados, foi desenvolvido um *script* em linguagem Bash, cujo exemplo representativo para um utilizador está descrito na figura 3.39. Assim, as primeiras duas instruções iniciam, respetivamente, os serviços referentes ao HDFS e ao YARN. De seguida, por meio do Apache Sqoop, procede-se à exportação dos dados da base de dados para o ambiente Apache Hadoop. Para finalizar, são desligados, de forma respetiva, os serviços relativos ao HDFS e ao YARN.

```
#!/bin/bash

# This bash script exports data from
# MySQL (with Apache Sqoop) to Hadoop
# Distributed File System.

# Start HDFS
("/usr/local/hadoop/sbin/start-dfs.sh")

# Start YARN
("/usr/local/hadoop/sbin/start-yarn.sh")

# Apache Sqoop Job
sqoop job --exec SQOOP_IMPORT

# Stop HDFS
("/usr/local/hadoop/sbin/stop-dfs.sh")

# Stop YARN
("/usr/local/hadoop/sbin/stop-yarn.sh")
```

Figura 3.39: *Script* Bash para automação do processo de exportação de dados.

Por fim, uma vez tendo sido corrido o *script* de exportação de dados da base de dados MySQL para o sistema de ficheiros do Apache Hadoop, é possível verificar que estes se encontram presentes na respetiva plataforma. Desta forma, tal como se verifica por meio da figura 3.40, ao efetuar a instrução ilustrada, os dados exibidos pela figura 3.37 aparecem no HDFS.

```
thesis@thesis-vBox:~$ hdfs dfs -cat /user/thesis/readings/part-m-000**

1,2395-01-01 10:00:20,temperature,19
2,2039-01-01 10:00:20,luminosity,14
3,2469-01-01 10:00:20,humidity,20
4,2019-01-01 10:00:20,humidity,14
5,2039-01-01 10:00:20,luminosity,534
6,2569-01-01 10:00:20,humidity,34
7,2149-01-01 10:00:20,luminosity,939
8,2075-01-01 10:00:20,temperature,34
```

Figura 3.40: Exemplo de dados armazenados no HDFS.

4 Teste e Validação

Este capítulo serve como validação da solução proposta e está dividido em três partes: a metodologia utilizada para a realização dos testes, a sua definição e validação. Na primeira parte, é apresentada a metodologia adotada para se proceder à realização dos testes. De seguida, estes são definidos, organizados em grupos, especificados e descritos. Por fim, de acordo com a estrutura definida, estes são executados e validados.

4.1 Metodologia de Testes

A metodologia de testes selecionada baseia-se na norma para fins de teste e certificação desenvolvido pelo *European Telecommunication Standards Institute* (ETSI) e adotado pelo *International Telecommunication Union* (ITU): o *Testing and Test Control Notation Version* 3 (TTCN-3). O TTCN-3 apresenta-se, desde 2000, como uma linguagem de especificação de testes assente no modelo de caixa negra, sendo encarado como padrão em contextos de investigação, industrial e de ensino (ETSI, 2013).

Desta forma, a estrutura de especificação de testes escolhida baseia-se na versão adaptada por Agostinho (2012) do modelo proposto pelo ITU (2001). Agostinho (2012) organiza cada teste através de uma tabela que, para além de conter uma descrição, apresenta colunas relativas ao seu comportamento, às restrições necessárias para a sua execução e ao seu resultado. O comportamento de cada teste é determinado por meio de uma sucessão de instruções, caracterizada através de uma árvore cujos ramos representam eventos. Cada evento determina uma ação realizada sobre o sistema ou uma tomada de decisão relativamente ao seu *output*. Assim, para representar ações e tomadas de decisão, colocam-se, respetivamente, pontos de exclamação (“!”) e de interrogação (“?”) antes da descrição de cada instrução a efetuar. Para finalizar, depois de cada evento ser realizado,

o seu *output* deve ser avaliado a fim de devolver um resultado, que pode ser de três tipos: "Sucesso", "Falha" ou "Inconclusivo".

4.2 Definição de Testes

Tendo em conta os elementos estudados no capítulo 2 e a interação existente entre eles, os testes efetuados sobre o protótipo foram organizados de acordo com os cinco grupos representados na tabela 4.1.

Tabela 4.1: Agrupamento de testes de acordo com os elementos do capítulo 2.

Grupo	**Teste**
Dispositivos para a IdC	1.1: Leitura dos dados dos sensores pelo Arduino
	1.2: Armazenamento dos dados dos sensores no Arduino
Middlewares Orientados a Mensagens, Dispositivos para a IdC	2.1: Envio dos dados do Arduino para a aplicação Android
Middlewares Orientados a Mensagens	3.1: Armazenamento dos dados na base de dados SQLite presente no *smartphone*
	3.2: Visualização dos dados públicos e dos pertencentes ao utilizador no *smartphone*
Plataformas de *Cloud* para a IdC, *Middlewares* Orientados a Mensagens	4.1: Envio dos dados da base de dados SQLite para a base de dados MySQL presente na Plataforma de *Cloud*
	4.2: Envio dos dados da base de dados MySQL para a base de dados SQLite (processo de sincronização)
Plataformas de *Cloud* para a IdC	5.1: Exportação dos dados presentes na base de dados MySQL para o sistema de ficheiros do Apache Hadoop

4.2.1 Dispositivos para a IdC

O teste 1.1 foi desenvolvido com o intuito de validar a leitura dos dados produzidos pelos sensores por parte do Arduino. Desta forma, este começa por verificar se os sensores se encontram ligados e configurados corretamente e, em caso afirmativo, os dados são lidos com sucesso. Por outro lado, caso alguma das etapas falhe, a leitura dos dados também falha. A especificação TTCN-3 para este teste está representada na tabela 4.2.

Tabela 4.2: Teste 1.1 – Leitura dos dados dos sensores pelo Arduino.

Teste:	Leitura dos dados dos sensores pelo Arduino
Grupo:	Dispositivos para a IdC
Objetivo:	Verificar se a leitura dos dados dos sensores foi bem-sucedida
Comentários:	A leitura dos dados dos sensores é efetuada de forma periódica

Comportamento	**Restrições**
? Sensores ligados e configurados corretamente ! Lê os dados dos sensores Caso Contrário	

Por sua vez, o teste 1.2, cuja especificação TTCN-3 está caracterizada na tabela 4.3, foi criado com o objetivo de confirmar o correto armazenamento dos dados recolhidos pelos sensores conectados ao Arduino no cartão microSD com o qual também possui interação.

Tabela 4.3: Teste 1.2 – Armazenamento dos dados dos sensores no Arduino.

Teste:	Armazenamento dos dados dos sensores no Arduino
Grupo:	Dispositivos para a IdC (Desafio 1)
Objetivo:	Verificar se o armazenamento dos dados dos sensores foi bem-sucedido
Comentários:	O armazenamento local dos dados recolhidos é temporário, sendo necessário uma vez que não estão sempre a ser retirados pelo *smartphone* (Desafio 1)

Comportamento	**Restrições**
? Leitor de cartões microSD ativo ? Ficheiro de armazenamento de dados existe ? Abre ficheiro de armazenamento de dados em modo de escrita ! Formata os dados recolhidos numa única *String* ! Grava dados no ficheiro de armazenamento de dados Caso Contrário Caso Contrário Caso Contrário	

Inicialmente, o Arduino averigua se o leitor de cartões microSD se encontra ativo, se o ficheiro de armazenamento dos dados existe e se o consegue abrir em modo de

escrita. Caso alguma destas etapas não seja bem-sucedida, o teste é concluído com falha. No entanto, caso contrário, a placa com microcontrolador formata as amostras recolhidas no mesmo instante numa única *String* e armazena os dados com sucesso.

4.2.2 Dipositivos para a IdC e MOM

O teste 2.1, especificado de acordo com o método TTCN-3, encontra-se na tabela 4.4 e foi gerado a fim de se apurar se os dados são enviados do Arduino para a aplicação Android de forma acertada. Assim, a aplicação móvel começa por solicitar o acesso ao Bluetooth do *smartphone* e, caso esse acesso seja concedido, o Android verifica se consegue estabelecer a conexão e o emparelhamento com a placa com microcontrolador. Caso algum destes passos não obtenha êxito, o teste falha, sendo finalizado. De outro modo, a aplicação pede ao Arduino que este lhe envie os dados e, em caso de sucesso, este verifica se o leitor de cartões microSD se encontra disponível, se o ficheiro de armazenamento de dados existe e se é possível abri-lo em modo de leitura. Se todas as etapas forem bem-sucedidas, o Android recebe os dados até o Arduino lhe indicar o fim do processo. No entanto, mais uma vez, caso alguma das fases aborte, o teste falha e é concluído sem sucesso.

Tabela 4.4: Teste 2.1 – Envio dos dados do Arduino para a aplicação Android.

Teste:	Envio dos dados do Arduino para a aplicação Android
Grupo:	*Middlewares* Orientados a Mensagens, Dispositivos para a IdC (Desafio 1)
Objetivo:	Verificar se os dados são enviados corretamente do Arduino para o Android
Comentários:	O envio é efetuado para os dados dos sensores de todos os utilizadores que se encontrem ao alcance do *smartphone* (Desafio 1)

Comportamento	**Restrições**
! Aplicação móvel solicita o acesso ao Bluetooth do *smartphone*	
? Acesso ao Bluetooth concedido	
? Bluetooth do Arduino ligado e acessível	
? Emparelhamento do Bluetooth do Arduino com o do Android	
! Aplicação móvel solicita dados ao Arduino	Bluetooth
? Solicitação recebida	
? Leitor de cartões microSD ativo	
? Ficheiro de dados existe	
? Abre ficheiro de dados em modo de leitura	
! Envia os dados	Bluetooth
? Processo de envio concluído	
! Arduino indica o fim do processo	
Caso Contrário	
Caso Contrário	
Caso Contrário	
Caso Contrário	
Caso Contrário	
Caso Contrário	
Caso Contrário	
Caso Contrário	

4.2.3 MOM

O teste 3.1 surge como seguimento do teste 2.1, utilizando o seu *output*, e foi desenvolvido a fim de se confirmar o armazenamento dos dados na base de bados SQLite com sucesso. Desta forma, quando os dados forem recebidos pela aplicação Android, esta verifica se a base de dados existe e envereda pelo processo de abertura em modo de escrita. Quando falha alguma das etapas referidas, o teste é concluído sem sucesso. Por outro lado, caso os passos tenham sido bem-sucedidos, o Android interpreta o formato

que os dados possuem, extrai-os e tenta armazená-los na base de dados. Caso os dados sejam guardados com êxito, o teste finaliza da mesma forma, de outro modo, o teste falha, concluindo. A especificação TTCN-3 para este teste está representada na tabela 4.5.

Tabela 4.5: Teste 3.1 – Armazenamento dos dados na base de dados SQLite.

Teste:	Armazenamento dos dados na base de dados SQLite presente no *smartphone*
Grupo:	*Middlewares* Orientados a Mensagens (Desafio 2)
Objetivo:	Verificar se os dados são armazenados corretamente
Comentários:	O armazenamento local dos dados é efetuado devido a eventuais falhas ou inexistência de ligação à *Internet* (Desafio 2)

Comportamento	**Restrições**
! Dados recebidos pela aplicação móvel ? Base de dados SQLite existe ? Abre base de dados em modo de escrita ! Interpretação do formato de dados e respetiva extração ? Armazena os dados na base de dados SQLite Caso Contrário Caso Contrário Caso Contrário	

O desenvolvimento do teste 3.2 advém da intenção de confirmar se os dados públicos e os dados pertencentes ao utilizar são carregados pela aplicação Android para visualização, destacando a sua especificação TTCN-3 na tabela 4.6. O teste tem início com a averiguação da existência da base de dados local, sendo que, em caso afirmativo, a aplicação tenta abri-la em modo de leitura. Em caso de sucesso, verifica-se se o ID do utilizador apresenta correspondência com o dos dados da tabela da base de dados e procede-se à sua requisição. Caso algum dos passos anterimente referidos falhe, o teste acaba com a indicação de falha. Caso contrário, a visualização que possui os dados públicos e os do utilizador é atualizada, sendo o teste finalizado com sucesso.

Tabela 4.6: Teste 3.2 – Visualização dos dados públicos e dos do utilizador.

Teste:	Visualização dos dados públicos e dos do utilizador do *smartphone*
Grupo:	*Middlewares* Orientados a Mensagens
Objetivo:	Verificar se os dados são demonstrados corretamente
Comentários:	A visualização dos dados apenas é possível se estes foram públicos ou dos sensores pertencentes ao utilizador

Comportamento	**Restrições**
? Base de dados SQLite existe ? Abre base de dados em modo de leitura ? ID do utilizador corresponde aos dos dados dos sensores ? Solicita os dados ! Atualiza os valores dos dados disponibilizados Caso Contrário Caso Contrário Caso Contrário Caso Contrário	

4.2.4 MOM e Plataformas de *Cloud* para a IdC

A realização do teste 4.1 deveu-se, essencialmente, à necessidade de se averiguar se os dados da base de dados SQLite são enviados corretamente para a base de dados MySQL, estando a sua especificação TTCN-3 caracterizada na tabela 4.7. Para começar, a aplicação móvel solicita ao Node-RED permissão para enviar os dados para a *Cloud*. Caso seja concedida, o Android verifica se a base de dados SQLite existe e tenta abri-la em modo de leitura. De seguida, os dados da próxima entrada na tabela são recolhidos, formatados numa *String* JSON e enviados, por meio do protocolo de comunicação MQTT, para o *middleware*, sendo necessária uma ligação à *Internet* para este passo. Uma vez recebidos os dados, o Node-RED interpreta-os e verifica, novamente por meio de uma conexão à *Internet*, se consegue estabelecer uma ligação com a base de dados presente na *Cloud*. Se a ligação for alcançada, os dados são enviados até o processo se encontrar finalizado e, nesse momento, o *middleware* informa a aplicação da conclusão do procedimento.

Tabela 4.7: Teste 4.1 – Envio dos dados da base de dados SQLite para a MySQL.

Teste:	Envio dos dados da base de dados SQLite para a base de dados MySQL presente na Plataforma de *Cloud*
Grupo:	Plataformas de *Cloud* para a IdC, *Middlewares* Orientados a Mensagens (Desafio 1)
Objetivo:	Verificar se o envio dos dados para a *Cloud* é bem-sucedido
Comentários:	O envio é efetuado para os dados dos sensores de todos os utilizadores que se encontrem ao alcance do *smartphone* (Desafio 1)

Comportamento	**Restrições**
! Android solicita ao Node-RED a permissão para o envio dos dados	Wi-Fi/3G
? Permissão concedida ao Android	
? Base de dados SQLite existe	
? Abre base de dados em modo de leitura	
? Gera um JSON com os dados da próxima entrada da tabela	
? Node-RED recebe os dados por MQTT	Wi-Fi/3G
! Node-RED interpreta os dados recebidos	
? Ligação à base de dados MySQL	Wi-Fi/3G
! Envio dos dados para a base de dados MySQL	Wi-Fi/3G
? Processo de envio concluído	
! Node-RED indica o fim do processo	
Caso Contrário	
Caso Contrário	
Caso Contrário	
Caso Contrário	
Caso Contrário	
Caso Contrário	
Caso Contrário	

Por sua vez, o teste 4.2 foi criado com o objetivo de confirmar o sucesso da funcionalidade de sincronização, i.e., do envio dados da base de dados MySQL para a base de dados SQLite e encontra-se especificado de acordo com a norma TTCN-3 na tabela 4.8.

Tabela 4.8: Teste 4.2 – Envio dos dados da base de dados MySQL para a SQLite.

Teste:	Envio dos dados da base de dados MySQL para a base de dados SQLite (processo de sincronização)
Grupo:	Plataformas de *Cloud* para a IdC, *Middlewares* Orientados a Mensagens (Desafio 3)
Objetivo:	Verificar se o envio dos dados para a base de dados local é bem-sucedido
Comentários:	Os dados recolhidos pelo utilizador foram introduzidos na *Cloud* por outro dispositivo (Desafio 3)

Comportamento	**Restrições**
! Android solicita ao Node-RED a recolha dos dados	Wi-Fi/3G
? Solicitação aceite	
? Ligação à base de dados MySQL	Wi-Fi/3G
? Recolha dos dados da próxima entrada da tabela	
? Node-RED recebe os dados solicitados	Wi-Fi/3G
! Node-RED agrupa os dados numa única *String*	
? Android recebe os dados por MQTT	Wi-Fi/3G
? Base de dados SQLite existe	
? Abre base de dados em modo de escrita	
! Armazena os dados na base de dados	
? Processo de envio concluído	
! Node-RED indica o fim do processo	
Caso Contrário	
Caso Contrário	
Caso Contrário	
Caso Contrário	
Caso Contrário	
Caso Contrário	
Caso Contrário	

Inicialmente, a aplicação móvel solicita ao *middleware* a recolha dos dados presentes na *Cloud* em falta na base de dados local. Caso a solicitação seja aceite, o Node-RED interpreta-os e verifica, através de uma ligação à *Internet*, se consegue conectar-se à base de dados MySQL. Caso a conexão seja estabelecida, os dados da próxima entrada contida na tabela são enviados para o *middleware*, agrupados numa única *String* e reencaminhados, por intermédio do protocolo MQTT, para a aplicação móvel. No ambiente Android, procede-se à verificação da existência da base de dados SQLite, à sua

abertura em modo de escrita e ao armazenamento dos dados recebidos. Os processos de envio e de armazenamento são repetidos até que o Node-RED informe a aplicação da sua conclusão, originando, desta forma, o fim bem-sucedido do teste. Por outro lado, caso ocorra alguma falha durante o decorrer das etapas, o teste é finalizado com a indicação de falha.

4.2.5 Plataformas de *Cloud* para a IdC

Por fim, o teste 5.1 foi criado com a finalidade de garantir a eficácia da exportação dos dados presentes na base de dados MySQL para o sistema de ficheiros do Apache Hadoop – o HDFS. Deste modo, o teste começa ao ser corrido o *script* que utiliza comandos próprios do Apache Sqoop, sendo efetuada a verificação da existência de novos dados na base de dados presente na *Cloud*. Em caso afirmativo, o Apache Sqoop tenta proceder à exportação dos dados para o HDFS. Caso algum dos passos indicados aborte, o teste conclui com informação de falha. No entanto, se passar todas as etapas com êxito, termina com sucesso. A especificação TTCN-3 deste teste está representada na tabela 4.9.

Tabela 4.9: Teste 5.1 – Exportação dos dados de MySQL para o Apache Hadoop.

Teste:	Exportação dos dados presentes na base de dados MySQL para o sistema de ficheiros do Apache Hadoop
Grupo:	Plataformas de *Cloud* para a IdC
Objetivo:	Verificar se os dados presentes na base de dados são exportados com sucesso
Comentários:	Os dados são exportados da base de dados MySQL para o HDFS

Comportamento	**Restrições**
? Existência de novos dados na base de dados MySQL ? Apache Sqoop exporta os dados para o HDFS Caso Contrário Caso Contrário	

4.3 Validação

Neste subcapítulo, os testes estão organizados tendo em conta as funcionalidades e interações dos elementos de estudo que se pretendem validar. Assim, são apresentados os

testes de acordo com a especificação TTCN-3, bem como uma análise detalhada do fluxo do seu comportamento.

4.3.1 Dispositivos para a IdC

Por meio da imagem 4.1, verifica-se que a ligação e a configuração correta dos sensores levam a que, como se pode verificar ao observar a figura 4.2, os dados dos sensores sejam lidos e, consequentemente, o teste 1.1 seja validado com sucesso.

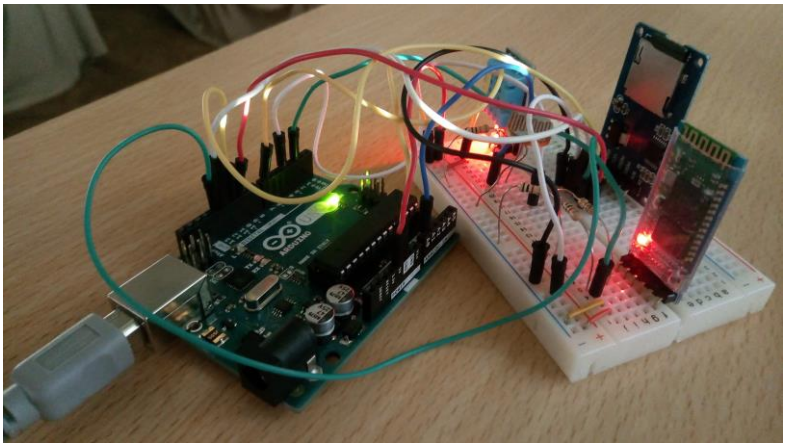

Figura 4.1: Ligação e configuração correta dos sensores.

Para além disso, a imagem 4.2 confirma também o sucesso da validação do teste 1.2 após a conclusão das seguintes etapas: ativação do leitor de cartões microSD; existência do ficheiro de armazenamento dos dados; abertura do referido ficheiro em modo de escrita; formatação das amostras recolhidas no mesmo instante numa única *String*; e armazenamento dos dados com sucesso.

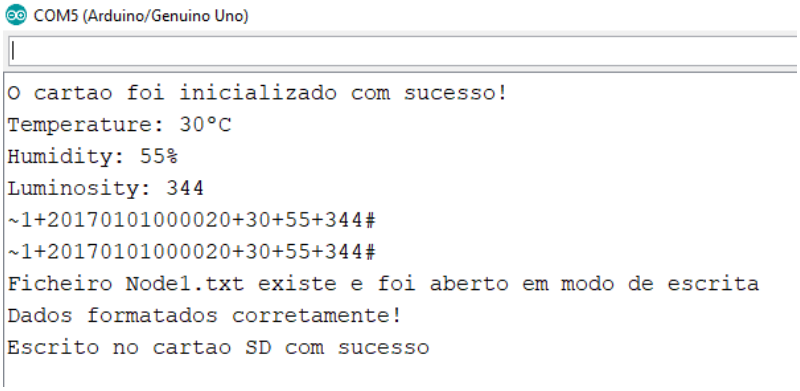

Figura 4.2: Leitura dos dados dos sensores.

4.3.2 Dipositivos para a IdC e MOM

O sucesso da validação do teste 2.1 confirma-se por intermédio das seguintes etapas: solicitação do acesso ao Bluetooth do *smartphone* e acesso ao Bluetooth concedido (figura 4.3); estabelecimento da conexão e do emparelhamento do Bluetooth com o Arduino (figura 4.4); solicitação dos dados ao Arduino (figura 4.5); ativação do leitor de cartões microSD, existência do ficheiro de armazenamento dos dados, abertura do referido ficheiro em modo de leitura, envio dos dados para o Arduino e conclusão do processo de envio (figura 4.4).

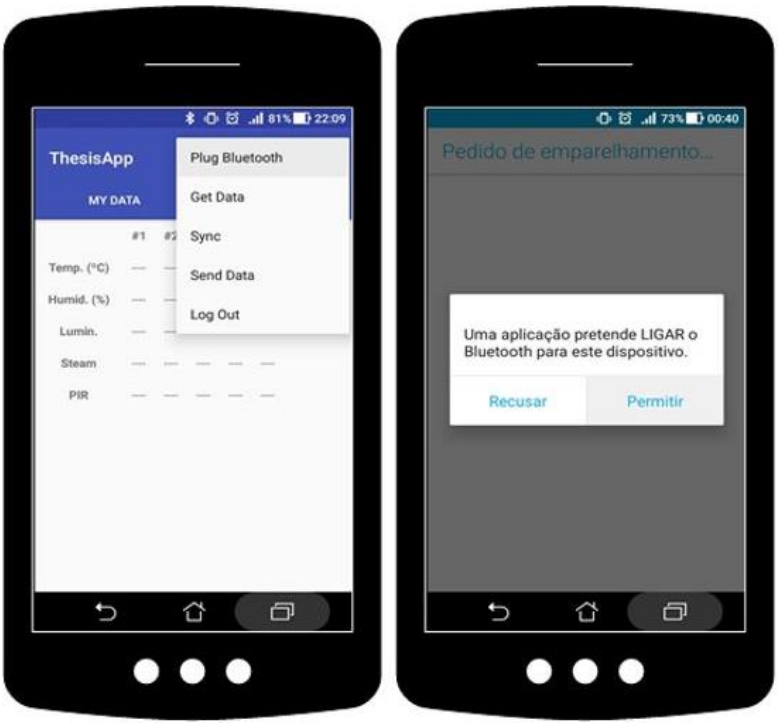

Figura 4.3: Solicitação do acesso ao Bluetooth do *smartphone* e acesso concedido.

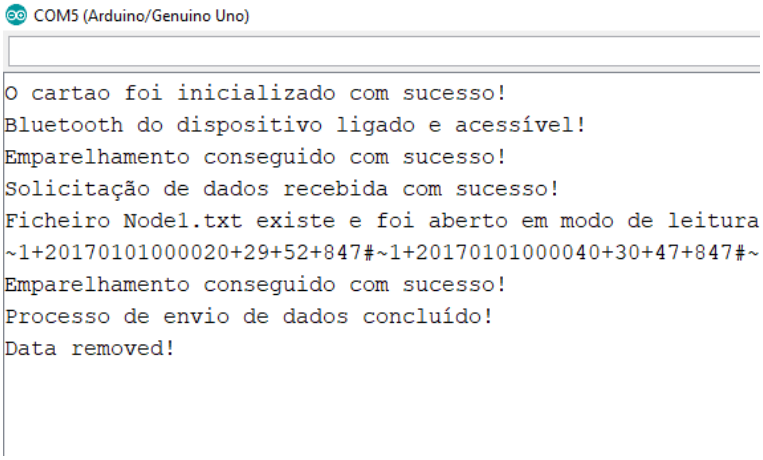

Figura 4.4: Operações relativas ao Bluetooth e ao leitor de cartões microSD.

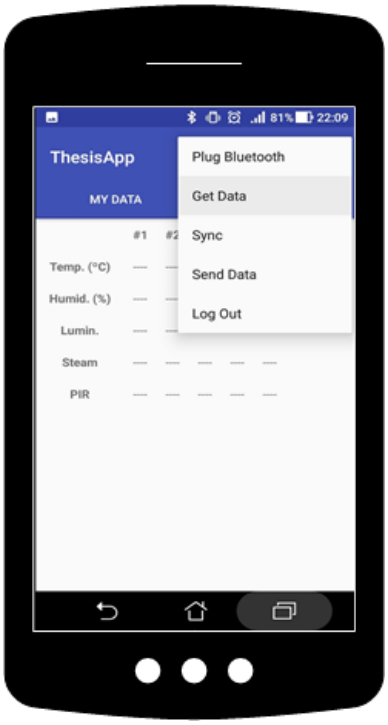

Figura 4.5: Solicitação dos dados ao Arduino.

4.3.3 MOM

O teste 3.1 é validado com a realização das etapas que se seguem: receção dos dados pela aplicação Android, existência da base de dados SQLite, abertura da base de dados em modo de escrita, interpretação do formato dos dados e respetiva extração e armazenamento dos dados na base de dados (Figura 4.6).

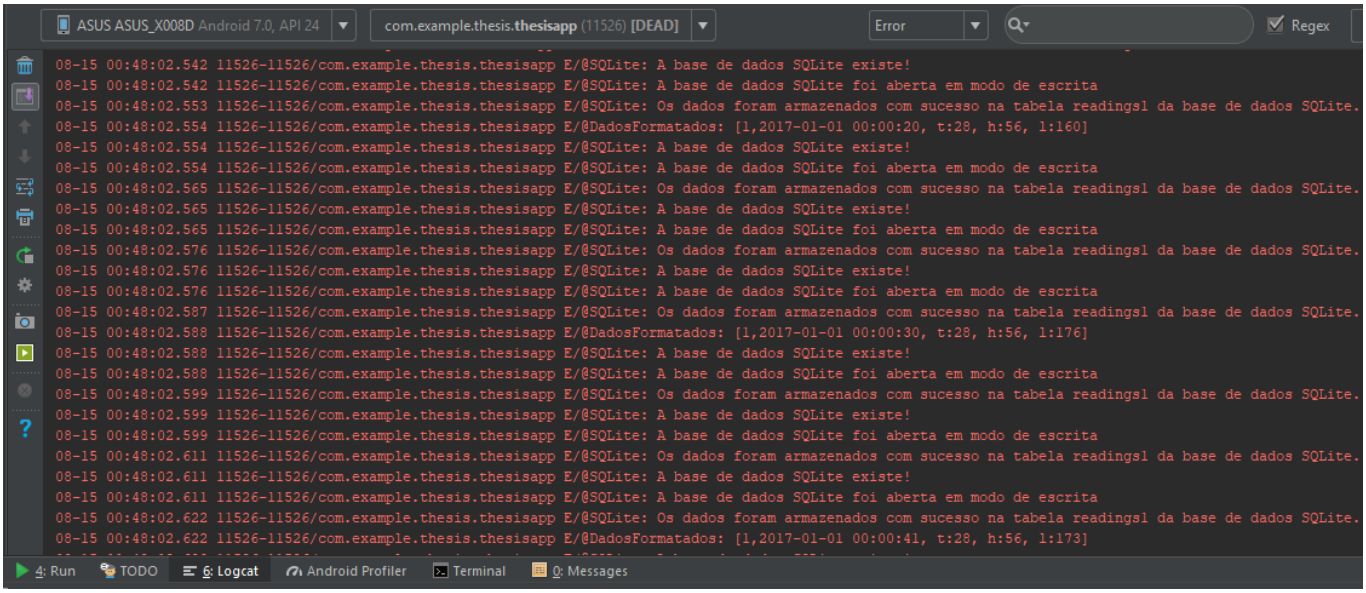

Figura 4.6: Operações relativas aos dados e à base de dados SQLite.

O teste 3.2 é validado atavés das seguintes etapas: existência da base de dados SQLite, abertura da base de dados em modo de leitura, verificação da correspondência entre o ID do utilizador e o dos dados da tabela da base de dados, requisição dos dados e atualização da visualização que possui os dados públicos e os do utilizador (Figura 4.7).

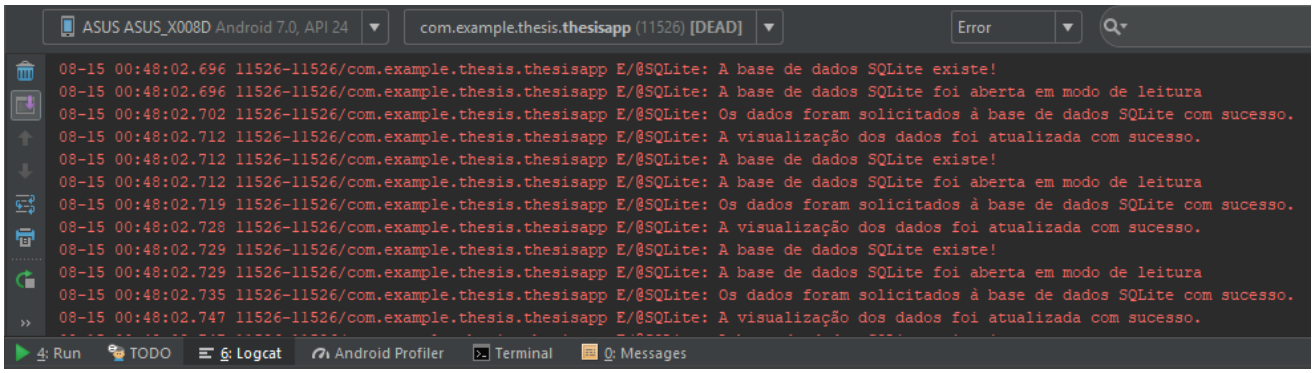

Figura 4.7: Operações relativas à base de dados SQLite e à visualização dos dados.

4.3.4 MOM e Plataformas de *Cloud* para a IdC

As imagens 4.8 e 4.9 possuem o objetivo de validar o teste 4.1 com a realização das etapas que se seguem: solicitação de permissão ao Node-RED para o envio dos dados para a *Cloud*, existência da base de dados SQLite, abertura da base de dados em modo de leitura e construção de um JSON com os dados da próxima entrada da tabela (Figura 4.8); receção dos dados pelo Node-RED através do protocolo de comunicação MQTT, interpretação dos dados recebidos pelo Node-RED, ligação à base de dados MySQL, envio dos dados para a base de dados MySQL e conclusão do processo de envio (Figura 4.9).

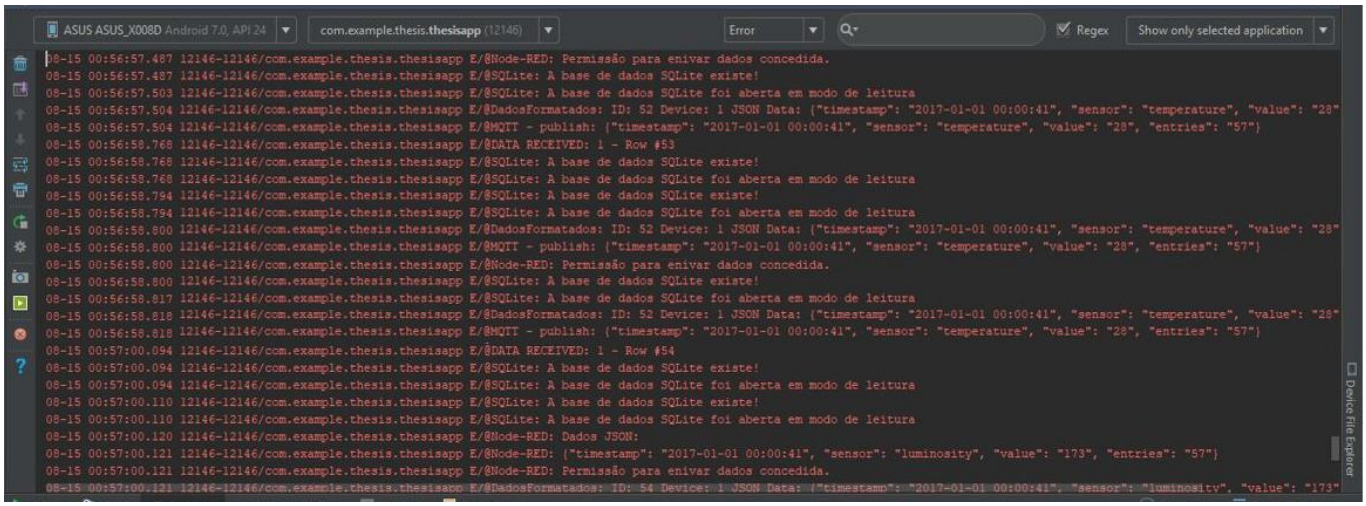

Figura 4.8: Operações de formatação de dados e da base de dados SQLite.

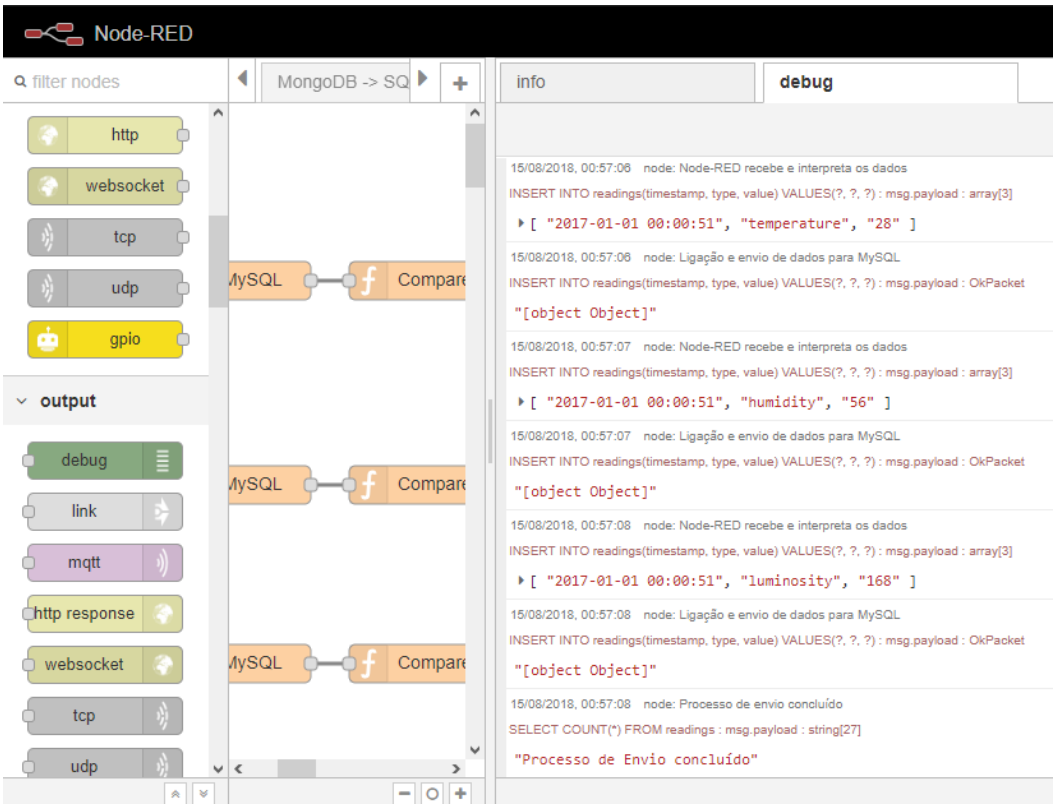

Figura 4.9: Interpretação de dados e respetivo armazenamento em MySQL.

O sucesso da validação do teste 4.2 confirma-se por intermédio das imagens 4.10, 4.11 e 4.12, que contêm as seguintes etapas: solicitação da recolha de dados pelo Android (Figura 4.10); ligação à base de dados MySQL, recolha dos dados com a próxima entrada da tabela, receção dos dados solicitados pelo Node-RED e formatação dos dados numa única *String* (Figura 4.11); receção dos dados pelo Android através de MQTT, existência da base de dados SQLite, abertura da base de dados SQLite em modo de escrita, armazenamento dos dados e conclusão do processo de envio (Figura 4.12).

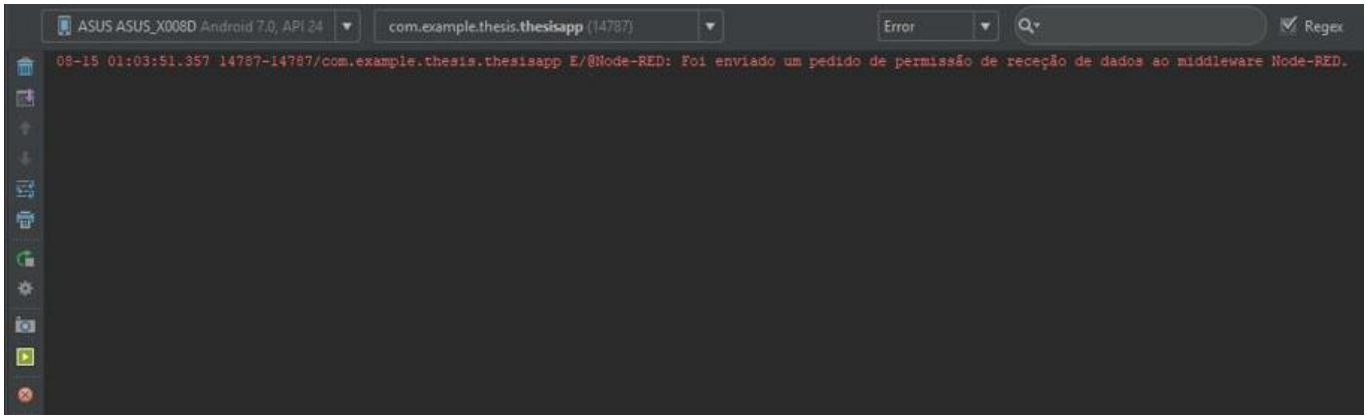

Figura 4.10: Solicitação da recolha de dados pelo Android.

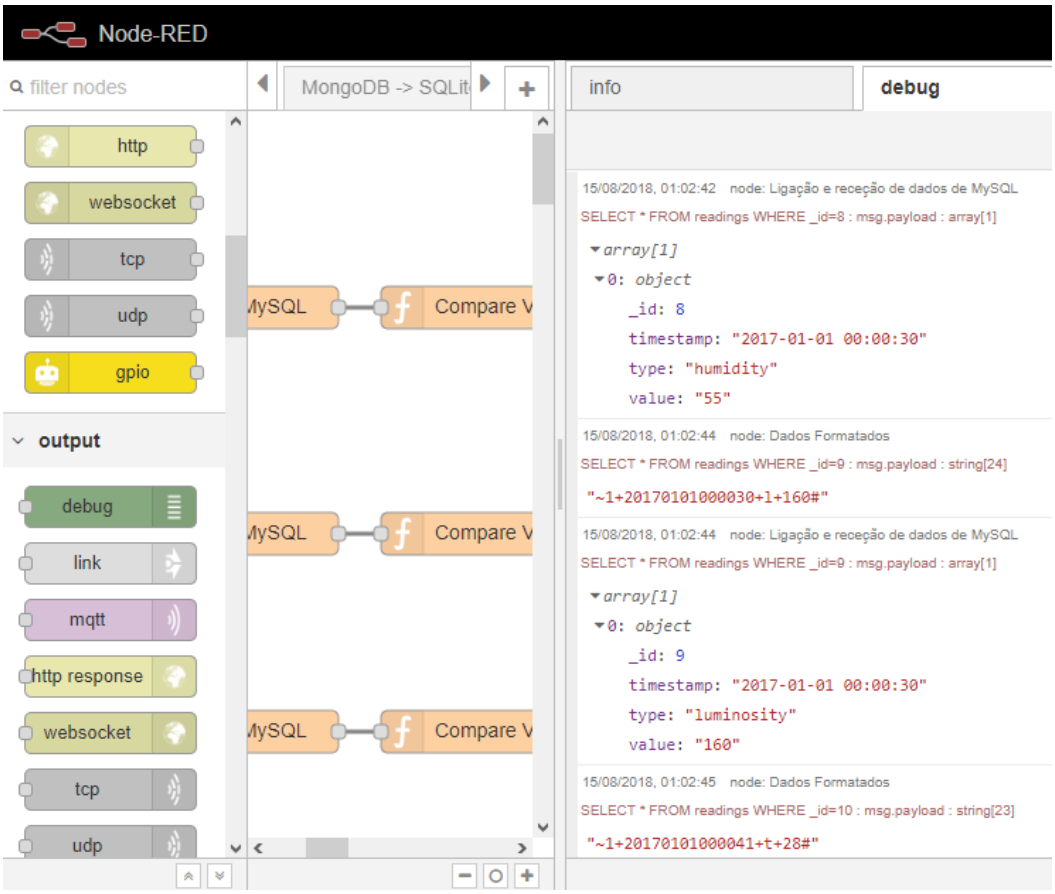

Figura 4.11: Recolha e formatação dos dados da base de dados MySQL.

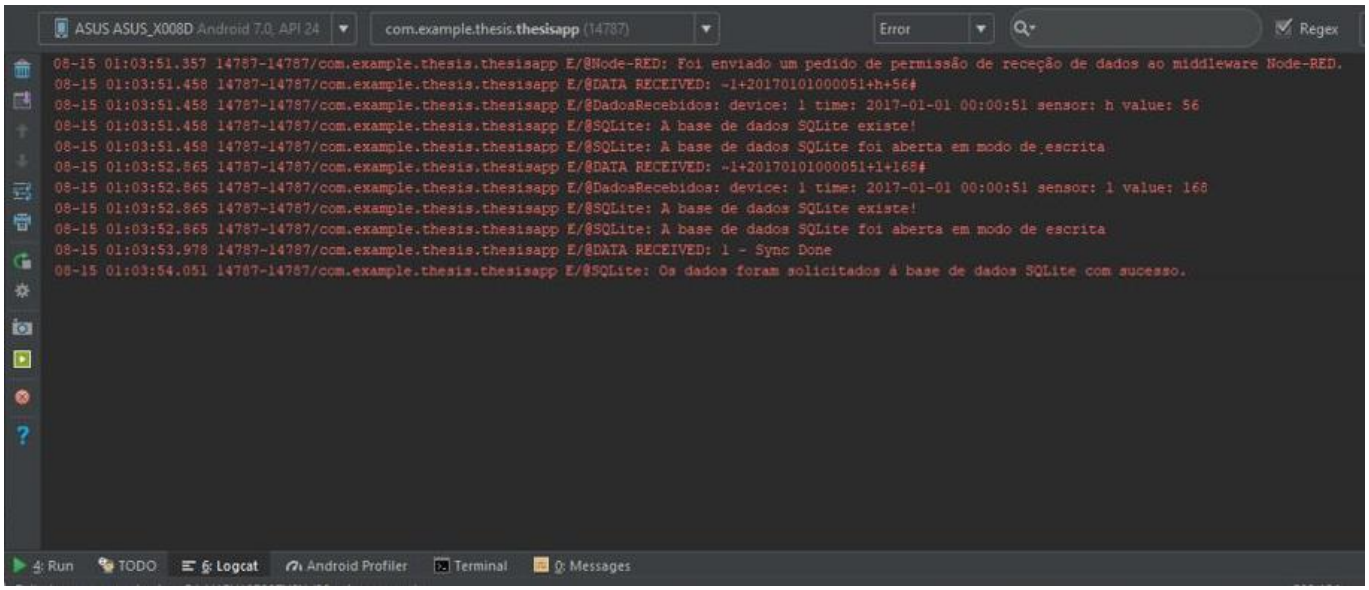

Figura 4.12: Recolha dos dados e respetivo armazenamento em SQLite.

4.3.5 Plataformas de *Cloud* para a IdC

As seguintes figuras validam com sucesso o teste 5.1 com a execução das seguintes etapas: existência de novos dados na base de dados MySQL (Figura 4.13) e exportação dos dados para o HDFS por meio do Apache Sqoop (Figuras 4.14 a 4.17).

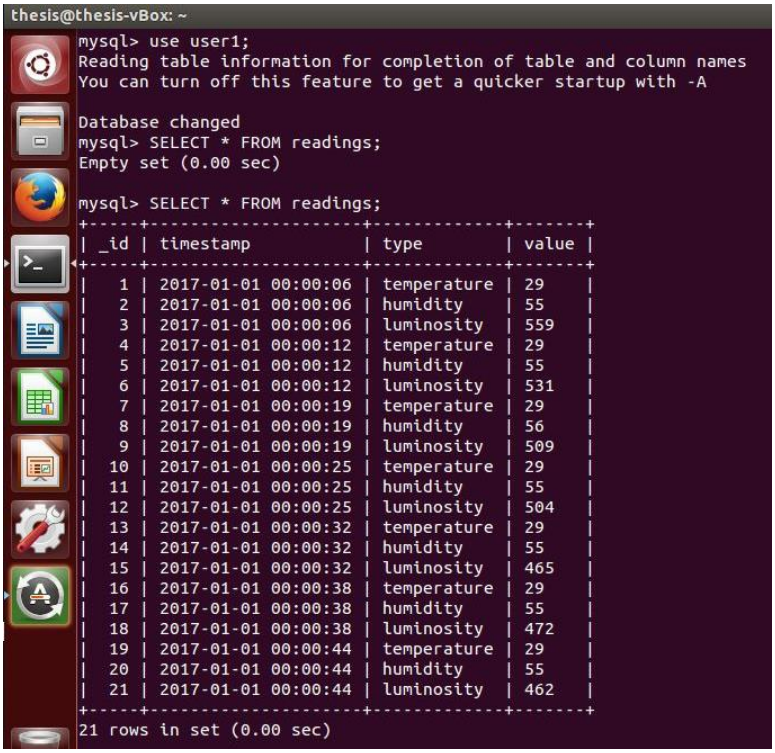

Figura 4.13: Existência de novos dados na base de dados MySQL.

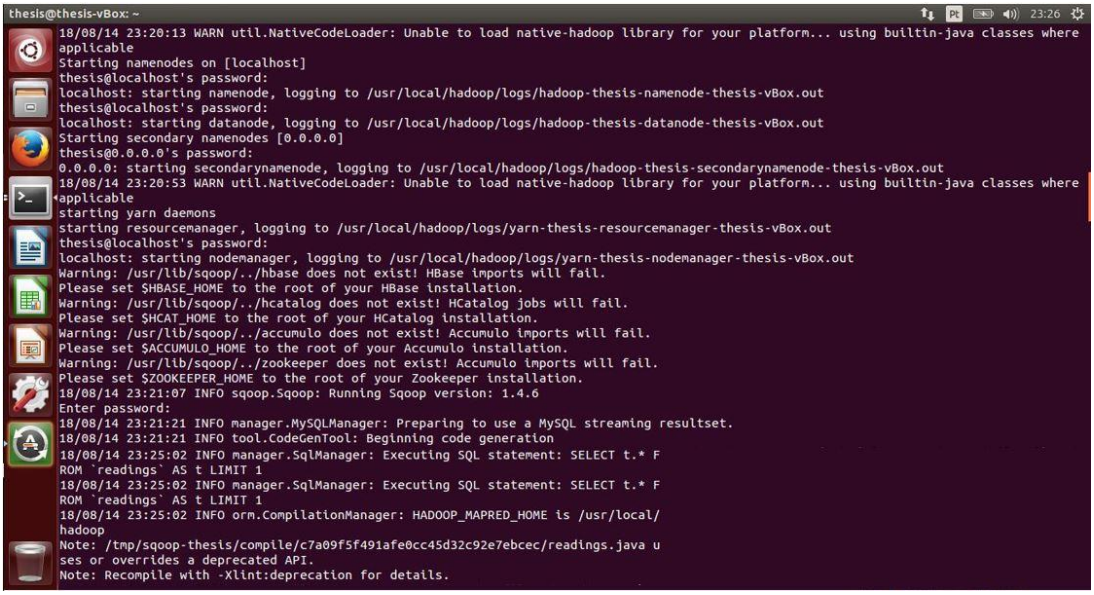

Figura 4.14: Apache Sqoop exporta os dados para o HDFS (Parte 1/4).

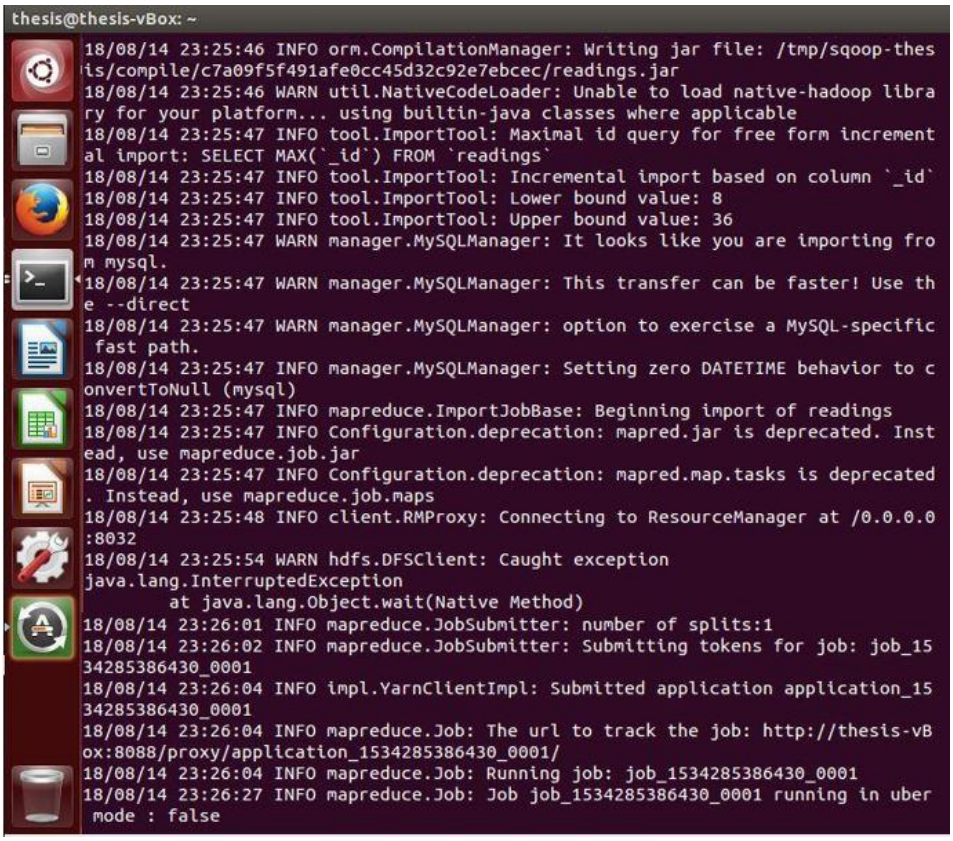

Figura 4.15: Apache Sqoop exporta os dados para o HDFS (Parte 2/4).

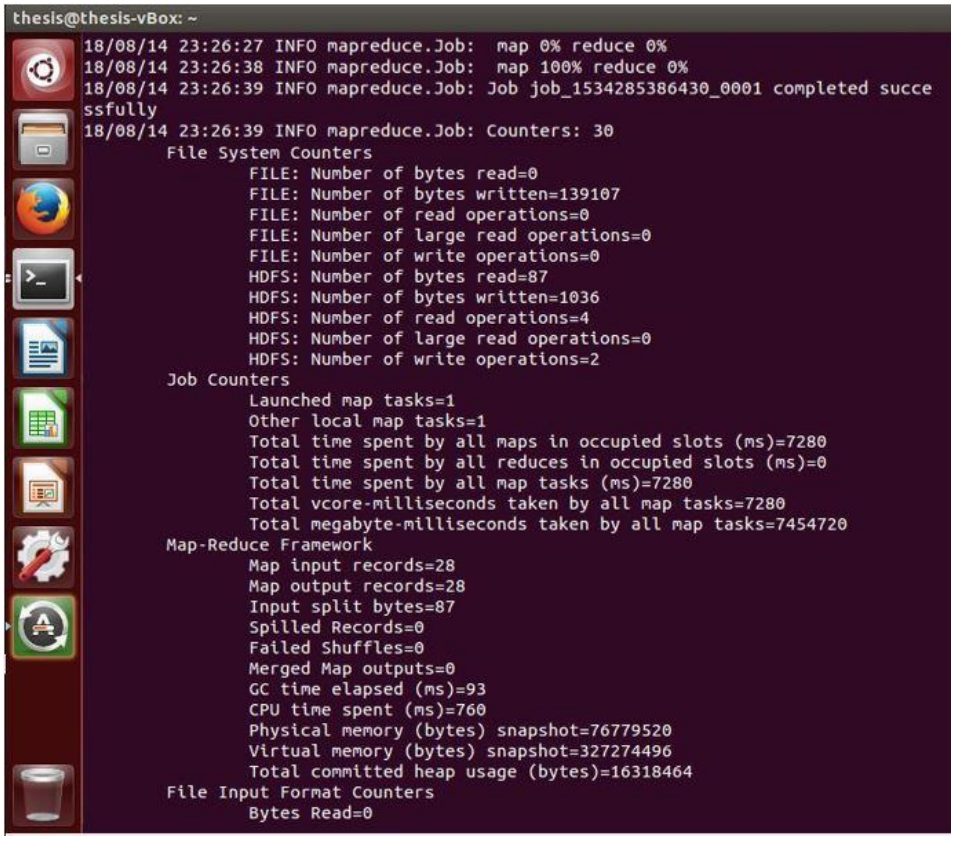

Figura 4.16: Apache Sqoop exporta os dados para o HDFS (Parte 3/4).

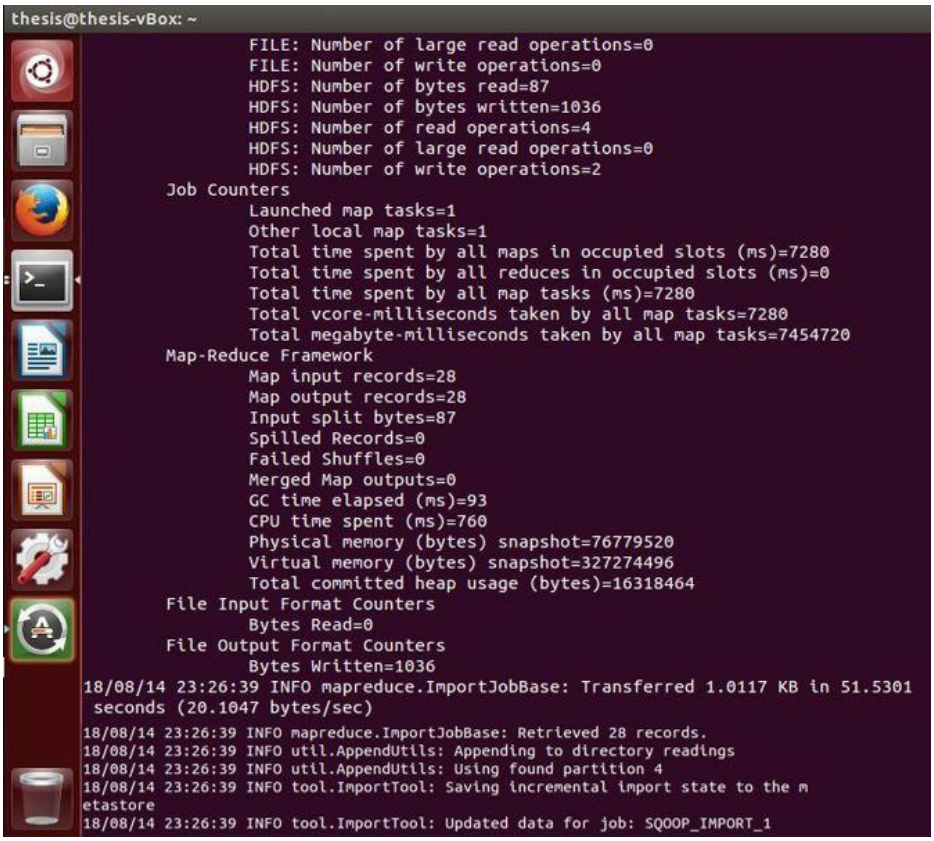

Figura 4.17: Apache Sqoop exporta os dados para o HDFS (Parte 4/4).

5 Conclusões e Trabalho Futuro

Neste capítulo, são redigidas conclusões retiradas ao longo do desenvolvimento desta dissertação, bem como sobre o trabalho realizado. Para complementar, existe também uma secção relativa ao trabalho futuro que se pretende elaborar.

5.1 Conclusões

A recolha de dados é, na sua maioria, efetuada através de dispositivos que formam uma rede que abrange apenas o local de operação, sendo necessário que estes possuam capacidades de conectividade que, habitualmente, não são compatíveis com o *smartphone*. Apesar disso, vários autores sugerem que este seja usado como *gateway*, permitindo o estabelecimento de uma ligação com outros tipos de rede. A utilização do *smartphone* nestes ambientes está, contudo, sujeita a atrasos e a interrupções, pelo que é comum a aplicação da técnica de *data mule*, que consiste na recolha de dados dos vários nós sensores presentes num determinado local por determinadas pessoas ou veículos que, posteriormente, estabelecem uma conexão à rede. A sincronização desses dados, que podem pertencer aos dispositivos de qualquer utilizador que se encontre no raio de alcance do *smartphone*, é efetuada entre uma aplicação móvel e a *Cloud*. No entanto, caso a sincronização não possa ser efetuada no momento, o utilizador pode gerir os seus dados de forma *offline* e, quando for possível, proceder à operação de sincronismo.

Deste modo, durante o desenrolar desta dissertação estiveram sempre presentes os três desafios provenientes da identificação do problema: funcionamento como *gateway*, armazenamento local e sincronização de dados. O primeiro desafio proposto passa pela

receção de dados emitidos pelos dispositivos de outros utilizadores por parte do *smartphone*, sendo estes retransmitidos, posteriormente, para a *Cloud*. Por sua vez, caso não seja possível enviar os dados no momento, estes necessitam de ser armazenados no *smartphone*, com vista a posterior reencaminhamento para a *Cloud*. Por último, o desafio 3 está relacionado com o processo de sincronização, i.e., com a receção, através da aplicação móvel, dos dados enviados para a *Cloud* por parte de outros utilizadores ou dispositivos do utilizador.

A revisão de literatura efetuada relativamente à tecnologia relevante para a presente dissertação foi dividida em três secções que estão relacionadas com os requisitos propostos para o sistema desenvolvido: Dispositivos para a IdC, *Middlewares* Orientados a Mensagens e Plataformas de *Cloud* para a IdC.

Os dispositivos para a IdC apresentam dimensões reduzidas e são compostos por interfaces GPIO e de comunicação série, por conversores digital-analógico e analógico-digital, por um microcontrolador, por elementos de comunicação com e sem fios e por uma bateria ou um armazenador de energia. Os dispositivos deste tipo possuem a aptidão para recolher informação de outros dispositivos e para a trocar entre si, sendo esta tratada localmente, na *Cloud* ou em ambos. Assim, o dispositivo deve ser capaz de interagir com sensores, com a *Internet* e com mecanismos de armazenamento de dados, permitindo que comuniquem com as aplicações por meio de redes móveis, Wi-Fi e BLE. Neste contexto, os dispositivos utilizados podem ser computadores embebidos, circuitos integrados com unidades de microcontrolador ou nós sensores.

Um *Middleware* Orientado a Mensagens permite a construção de sistemas modulares e altamente escaláveis e apresenta-se como um mediador de mensagens que disponibiliza mecanismos de comunicação entre dois componentes de *software* distintos. As principais tarefas deste elemento passam pela tradução e transmissão de dados, recuperação de erros, cálculo do trajeto mais eficiente a seguir e envio, receção e gestão de mensagens, sendo capaz de decidir se deve receber uma mensagem no imediato ou quando estiver pronto para a processar. Como elementos de estudo, foram selecionados o Apache Kafka, o Node-RED e o RabbitMQ.

Uma plataforma de *Cloud* para a IdC apresenta-se como um serviço pago ou de livre-acesso que permite a gestão das Coisas e dos eventos a elas associados e a integração de elementos distintos de *hardware* e *software*. Para além disso, deve ser capaz de processar e armazenar dados, permitindo a sua análise e supervisão em tempo real, em modo *offline* ou em ambas as situações. Este tipo de plataforma deve ainda disponibilizar ferramentas e bibliotecas de apoio ao desenvolvimento e possuir a capacidade de usufruir dos protocolos de comunicação mais utilizados atualmente, permitindo a tradução entre

protocolos distintos. Assim, o Apache Hadoop, o Firebase e o Kaa apresentam-se como elementos que encaixam neste tipo de plataformas.

Assim, o desenvolvimento e o protótipo da solução estão organizados de acordo com os elementos de estudo referidos anteriormente: Dispositivos para a IdC, *Middlewares* Orientados a Mensagens e Plataformas de *Cloud* para a IdC. Desta forma, ao realizar a comparação dos elementos de estudo, foram selecionados, respetivamente, o Apache Hadoop, o Node-RED e as Placas com Microcontrolador, mais concretamente, as versões Uno e Mega do Arduino.

A escolha da plataforma de *Cloud* baseou-se na possibilidade de o Apache Hadoop poder ser implementado numa instância local, apenas com as funcionalidades desejadas e sem barreiras em termos de preço. Por sua vez, o Node-RED foi selecionado por ser integrável com vários protocolos de comunicação e por se basear num ambiente de programação visual por blocos, o que o torna acessível e editável por um maior número de utilizadores. Por último, a placa com microcontrolador foi a preferida como dispositivo no âmbito da IdC, pois integra facilmente módulos de comunicação que interagem com o *smartphone* e apresenta uma reduzida capacidade de processamento.

Resumidamente, no contexto dos dispositivos, foram utilizadas, como já referido, as versões Uno e Mega do Arduino com vista à interação e recolha de dados produzidos por sensores de temperatura, humidade, luminosidade, vapor e de presença. Uma vez realizado este passo, quando solicitado pela aplicação Android, os dados são enviados por meio de Bluetooth. Estando os dados armazenados no *smartphone*, estes, caso sejam públicos ou do utilizador em questão, podem ser visualizados. No ambiente móvel, a aplicação comunica com o *middleware*, através de MQTT, com vista à transmissão de dados para e da *Cloud*. Por fim, a plataforma de *Cloud* é constituída por uma instância da base de dados MySQL, do sistema de ficheiros do Apache Hadoop e pelo Apache Sqoop, uma ferramenta de exportação de dados.

Para a validação de testes foi utilizado o TTCN-3, que se revela, desde 2000, como a metodologia padrão em contextos de investigação, industrial e de ensino. Desta forma, foram definidos testes com vista à validação das seguintes funcionalidades: leitura dos dados dos sensores pelo Arduino; armazenamento dos dados dos sensores no Arduino; envio dos dados do Arduino para a aplicação Android; armazenamento dos dados na base de dados SQLite presente no *smartphone*; visualização dos dados públicos e dos pertencentes ao utilizador no *smartphone*; envio dos dados da base de dados SQLite para a base de dados MySQL presente na Plataforma de *Cloud*; envio dos dados da base de dados MySQL para a base de dados SQLite (Processo de sincronização); e exportação dos dados presentes na base de dados MySQL para o sistema de ficheiros do Apache

Hadoop. Assim, depois de aplicados os testes definidos, as funcionalidades foram validadas com sucesso.

5.2 Trabalho Futuro

Uma vez que o protótipo construído com o decorrer da presente dissertação foi concebido de um ponto de vista genérico, um dos objetivos futuros passa pela sua validação em ambientes e contextos de difícil acesso à rede, como em casos de exploração mineira, agrícola e florestal.

Para além disso, é necessário desenvolver a capacidade de escalabilidade do sistema, dado que, visto tratar-se de um protótipo, este apenas foi elaborado para dispositivos específicos.

De modo a poupar a energia consumida pelos dispositivos, proponho, com base no estudo de Maló, Almeida, Melo, Kalaboukas & Cousin (2013), a implementação de um mecanismo de subscrição com restrições para a obtenção dos dados pretendidos a partir de valores pré-definidos pelo utilizador.

Outro aspeto importante que não foi trabalhado na solução apresentada, seguindo a mesma linha de pensamento de Jemel et al. (2015), prende-se com a segurança e a encriptação dos dados, um fator a ter em conta como futura implementação.

A automatização do processo de recolha de dados por Bluetooth e o seu envio quando existe uma ligação à rede Internet é outro fator a ter em conta para desenvolvimento como trabalho futuro.

Por fim, apesar de não ser o foco da presente dissertação, o desenvolvimento de mecanismos de análise e processamento de dados na *Cloud* torna-se essencial para serem obtidas informações e conclusões relativamente aos dados recolhidos.

Referências Bibliográficas

A., K. K., & K., C. (2013). A Review on Hadoop — HDFS Infrastructure Extensions. In *Proceedings of 2013 IEEE Conference on Information and Communication Technologies (ICT 2013)* (pp. 132–137). Retrieved from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6558077

Adafruit. (n.d.). Adafruit Feather 328P. Retrieved March 16, 2018, from https://www.adafruit.com/product/3458

Aggarwal, C. C., Ashish, N., & Sheth, A. (2014). The internet of things: A survey from the data-centric perspective. In C. C. Aggarwal (Ed.), *Managing and Mining Sensor Data* (pp. 383–428). Boston/Dordrecht/London: Kluwer Academic Publishers. https://doi.org/10.1007/978-1-4614-6309-2_12

Agostinho, C. M. M. (2012). *Sustainability of Systems Interoperability in Dynamic Business Networks*. Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa.

Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M., & Ayyash, M. (2015). Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. *IEEE Communications Surveys & Tutorials*, *17*(4), 2347–2376. https://doi.org/10.1109/COMST.2015.2444095

Al-Qaseemi, S. A., Almulhim, H. A., Almulhim, M. F., & Chaudhry, S. R. (2016). IoT Architecture Challenges and Issues: Lack of Standardization. In *FTC 2016 - Proceedings of Future Technologies Conference* (pp. 731–738). https://doi.org/10.1109/FTC.2016.7821686

Alam, A., & Ahmed, J. (2014). Hadoop Architecture and Its Issues. In *Proceedings - 2014 International Conference on Computational Science and Computational Intelligence, CSCI 2014* (Vol. 2, pp. 288–291). https://doi.org/10.1109/CSCI.2014.140

Albano, M., Ferreira, L. L., Pinho, L. M., & Alkhawaja, A. R. (2015). Computer Standards & Interfaces Message-oriented middleware for smart grids. *Computer Standards & Interfaces*, *38*, 133–143. https://doi.org/10.1016/j.csi.2014.08.002

Alioto, M. (2017). IoT: Bird's Eye View, Megatrends and Perspectives. In M. Alioto (Ed.), *Enabling the Internet of Things* (1st Editio, pp. 1–46). Gewerbestrasse: Springer International Publishing. https://doi.org/10.1007/978-3-319-51482-6

Alioto, M., & Shahghasemi, M. (2018). The Internet of Things on Its Edge: Trends toward its tipping point. *IEEE Consumer Electronics Magazine*, *7*(1), 77–87. https://doi.org/10.1109/MCE.2017.2755218

Anastasi, G., Conti, M., & Francesco, M. Di. (2008). Data Collection in Sensor Networks with Data Mules: an Integrated Simulation Analysis, 1096–1102.

Apache Kafka. (2016). Retrieved February 5, 2017, from http://kafka.apache.org/intro.html

Apache Kafka Documentation. (2016). Retrieved February 13, 2017, from http://kafka.apache.org/documentation/

Apache ZooKeeper. (2016). Retrieved February 12, 2017, from https://zookeeper.apache.org/

Arduino. (2018a). Arduino Uno Rev3. Retrieved March 16, 2018, from https://store.arduino.cc/arduino-uno-rev3

Arduino. (2018b). Arduino Yún. Retrieved March 16, 2018, from https://store.arduino.cc/arduino-yun

Arduino. (2018c). What is Arduino? Retrieved March 16, 2018, from https://www.arduino.cc/en/Guide/Introduction#

Ashton, K. (2009, June 22). That "Internet of Things" Thing. *RFID Journal*, p. 1. Retrieved from http://www.rfidjournal.com/articles/view?4986

Atzori, L., Iera, A., & Morabito, G. (2010). The Internet of Things: A survey. *Computer Networks*, *54*(15), 2787–2805. https://doi.org/10.1016/j.comnet.2010.05.010

Ayanoglu, E., Aytas, Y., & Nahum, D. (2015). *Mastering RabbitMQ*. Birmingham: PACKT Publishing.

Azzedin, F. (2013). Towards a Scalable HDFS Architecture. *Proceedings of the 2013 International Conference on Collaboration Technologies and Systems, CTS 2013*, 155–161. https://doi.org/10.1109/CTS.2013.6567222

B., P. R. B., Saluja, P., Sharma, N., Mittal, A., & Sharma, S. V. (2012). Cloud Computing for Internet of Things & Sensing Based Applications. In *6th International Conference on Sensing Technology* (pp. 374–380).

Battle, R., & Benson, E. (2008). Bridging the semantic Web and Web 2.0 with Representational State Transfer (REST). *Web Semantics*, *6*(1), 61–69. https://doi.org/10.1016/j.websem.2007.11.002

Blackstock, M., & Lea, R. (2014). Toward a Distributed Data Flow Platform for the Web of Things (Distributed Node-RED). In *Proceedings of the 5th International Workshop on Web of Things - WoT '14* (pp. 34–39). https://doi.org/10.1145/2684432.2684439

Botta, A., de Donato, W., Persico, V., & Pescapé, A. (2016). Integration of Cloud

Computing and Internet of Things: A survey. *Future Generation Computer Systems*, *56*, 684–700. https://doi.org/10.1016/j.future.2015.09.021

Caini, C., Cornice, P., Firrincieli, R., Livini, M., & Lacamera, D. (2010). DTN meets smartphones: Future prospects and tests. In *ISWPC 2010 - IEEE 5th International Symposium on Wireless Pervasive Computing 2010* (pp. 355–360). https://doi.org/10.1109/ISWPC.2010.5483772

Chaczko, Z., & Braun, R. (2017). Learning Data Engineering: Creating IoT Apps using the Node-RED and the RPI Technologies. In *2017 16th International Conference on Information Technology Based Higher Education and Training, ITHET 2017*. https://doi.org/10.1109/ITHET.2017.8067827

Chen, G., Du, Y., Qin, P., & Zhang, L. (2013). Research of JMS Based Message Oriented Middleware for Cluster. In *2013 International Conference on Computational and Information Sciences* (pp. 1628–1631). https://doi.org/10.1109/ICCIS.2013.426

Comparing the Engineering Design Process and the Scientific Method. (2017). Retrieved February 8, 2017, from http://www.sciencebuddies.org/engineering-design-process/engineering-design-compare-scientific-method.shtml

Corcoran, P. (2016). The Internet of Things: Why now, and what's next? *IEEE Consumer Electronics Magazine*, *5*(1), 63–68. https://doi.org/10.1109/MCE.2015.2484659

Cui, Y., Lai, Z., Wang, X., & Dai, N. (2017). QuickSync: Improving Synchronization Efficiency for Mobile Cloud Storage Services. *IEEE Transactions on Mobile Computing*, *1233*, 1–14. https://doi.org/10.1109/TMC.2017.2693370

Curry, E. (2004). Message-Oriented Middleware. In *Middleware for Communications* (pp. 1–28). West Sussex: John Wiley & Sons, Ltd. https://doi.org/10.1002/0470862084

Dash, S. K., Mohapatra, S., & Pattnaik, P. K. (2010). A Survey on Applications of Wireless Sensor Network Using Cloud Computing. *International Journal of Computer Science & Emerging Technologies*, *1*(4), 50–55.

Datta, P., & Sharma, B. (2017). A Survey on IoT Architectures, Protocols, Security and Smart City based Applications. In *8th ICCCNT 2017*.

De La Piedra, A., Benitez-Capistros, F., Dominguez, F., & Touhafi, A. (2013). Wireless Sensor Networks for Environmental Research : A Survey on Limitations and Challenges. *EuroCon 2013*, (July), 267–274. https://doi.org/10.1109/EUROCON.2013.6624996

De Luca, G. E., Carnuccio, E. A., Garcia, G. G., & Barillaro, S. (2016). IoT fall detection system for the elderly using Intel Galileo development boards generation I. In *CACIDI 2016 - Congreso Aergentino de Ciencias de la Informatica y Desarrollos de Investigacion*. https://doi.org/10.1109/CACIDI.2016.7785997

Distl, B., & Legendre, F. (2015). Are smartphones suited for DTN networking? A methodological teardown of smartphones' WiFi performance. In *2015 International Workshop on Wireless Network Measurements and Experimentation* (pp. 90–95). https://doi.org/10.1109/WIOPT.2015.7151058

Dores, C., Reis, L. P., & Lopes, N. V. (2014). Internet of Things and Cloud Computing.

2014 9th Iberian Conference on Information Systems and Technologies (CISTI), *6*(2), 1–4. https://doi.org/10.1109/CISTI.2014.6877071

Dossot, D. (2014). *RabbitMQ Essentials*. Birmingham: PACKT Publishing. Retrieved from https://www.packtpub.com/application-development/rabbitmq-essentials

Dupont, C., Sheikhalishahi, M., Biswas, A. R., & Bures, T. (2017). IoT, Big Data, and Cloud Platform for Rural African Needs. In *IST-Africa 2017 Conference Proceedings* (pp. 1–7). https://doi.org/10.23919/ISTAFRICA.2017.8102386

Elomari, A., Maizate, A., & Hassouni, L. (2016). Data storage in big data context: A survey. In *2016 Third International Conference on Systems of Collaboration (SysCo)* (pp. 1–4). https://doi.org/10.1109/SYSCO.2016.7831344

ETSI. (2013). TTCN-3 Introduction. Retrieved March 13, 2018, from http://www.ttcn-3.org/index.php/about/introduction

Fall, K., & Farrell, S. (2008). DTN : An Architectural Retrospective, *26*(5), 828–836. https://doi.org/10.1109/JSAC.2008.080609

Firebase. (n.d.). Retrieved February 14, 2017, from https://firebase.google.com/

Firebase: Pricing. (2016). Retrieved February 16, 2017, from https://firebase.google.com/pricing/

Firebase - Features. (n.d.). Retrieved February 14, 2017, from https://firebase.google.com/features/

Firebase Documentation - Firebase Authentication. (2016). Retrieved February 16, 2017, from https://firebase.google.com/docs/auth/

Firebase Documentation - Firebase Cloud Messaging. (2016). Retrieved February 14, 2017, from https://firebase.google.com/docs/cloud-messaging/

Fong, D. Y. (2017). Wireless Sensor Networks. In H. Geng (Ed.), *Internet of Things and Data Analytics Handbook* (1st Ed., pp. 197–214). New Jersey: John Wiley & Sons, Inc.

Foundation, T. A. S. (2017a). Apache Hadoop. Retrieved February 22, 2018, from http://hadoop.apache.org

Foundation, T. A. S. (2017b). Apache Hadoop YARN 2.7.4. Retrieved February 22, 2018, from https://hadoop.apache.org/docs/r2.7.4/hadoop-yarn/hadoop-yarn-site/YARN.html

Fox, G. C., Kamburugamuve, S., & Hartman, R. D. (2012). Architecture and Measured Characteristics of a Cloud Based Internet of Things. In *Proceedings of the 2012 International Conference on Collaboration Technologies and Systems, CTS 2012* (pp. 6–12). https://doi.org/10.1109/CTS.2012.6261020

Ganguly, P. (2016). Selecting the right IoT Cloud Platform. *2016 International Conference on Internet of Things and Applications, IOTA 2016*, 316–320. https://doi.org/10.1109/IOTA.2016.7562744

Garg, N. (2013). *Apache Kafka*. Birmingham: PACKT Publishing. https://doi.org/10.1017/CBO9781107415324.004

Gaur, P., & Tahiliani, M. P. (2015). Operating Systems for IoT devices: A Critical Survey. In *Proceedings - 2015 IEEE Region 10 Symposium, TENSYMP 2015* (pp. 33–36). https://doi.org/10.1109/TENSYMP.2015.17

Gazal, & Kaur, P. D. (2015). A Survey on Big Data Strategies. In *2015 International Conference on Green Computing and Internet af Things (ICGClaT)* (pp. 280–284). https://doi.org/10.9790/0661-1905027578

Geng, H. (2017). Internet of Things and Data Analytics in the Cloud with Innovation and Sustainability. In H. Geng (Ed.), *The Internet of Things & Data Analytics Handbook* (1st Editio, pp. 3–28). New Jersey: John Wiley & Sons, Inc.

Grygoruk, A., & Legierski, J. (2016). IoT gateway – implementation proposal based on Arduino board. In *Proceedings of the Federated Conference on Computer Science and Information Systems* (Vol. 8, pp. 1011–1014). https://doi.org/10.15439/2016F283

Gubbi, J., Buyya, R., Marusic, S., & Palaniswami, M. (2013). Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, *29*(7), 1645–1660. https://doi.org/10.1016/j.future.2013.01.010

Guo, S., Falaki, M. H., Oliver, E. A., Ur Rahman, S., Seth, A., Zaharia, M. A., & Keshav, S. (2007). Very low-cost internet access using KioskNet. *ACM SIGCOMM Computer Communication Review*, *37*(5), 95–100. https://doi.org/10.1145/1290168.1290181

Hahm, O., Baccelli, E., Petersen, H., & Tsiftes, N. (2016). Operating Systems for Low-End Devices in the Internet of Things: A Survey. *IEEE Internet of Things Journal*, *3*(5), 720–734. https://doi.org/10.1109/JIOT.2015.2505901

Hao, S., Agrawal, N., Aranya, A., & Ungureanu, C. (2013). Building a delay-tolerant cloud for mobile data. *IEEE 14th International Conference on Mobile Data Management*, 293–300. https://doi.org/10.1109/MDM.2013.43

Hazarika, A. Vi., Ram, G. J. S. R., & Jain, E. (2017). Performance Comparision of Hadoop and Spark Engine. In *International conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC 2017) Performance* (pp. 671–674).

Hossmann, T., Carta, P., Schatzmann, D., Legendre, F., Gunningberg, P., & Rohner, C. (2011). Twitter in Disaster Mode: Security Architecture. *SWID '11 Proceedings of the Special Workshop on Internet and Disasters*. https://doi.org/10.1145/2079360.2079367

Hwang, K., Fox, G. C., & Dongarra, J. J. (2012). *Distributed and Cloud Computing: From Parallel Processing to the Internet of Things*. (T. Green & R. Day, Eds.) (1st Editio). Waltham: Elsevier.

Indiegogo. (2018). Pinoccio - A Complete Ecosystem for Building the Internet of Things. Retrieved March 16, 2018, from https://www.indiegogo.com/projects/pinoccio-a-complete-ecosystem-for-building-the-internet-of-things#/

Indrasiri, K. (2016). Enterprise Messaging with JMS, AMQP, MQTT, and Kafka. In *Beginning WSO2 ESB* (pp. 133–160). California: Apress. https://doi.org/10.1007/978-1-4842-2343-7

Intel. (2017). Introduction to Intel Galileo Boards. Retrieved March 16, 2018, from https://www.intel.com/content/www/us/en/support/articles/000005912/boards-and-kits/intel-galileo-boards.html

Ionescu, V. M. (2015). The analysis of the performance of RabbitMQ and ActiveMQ. In *2015 14th RoEduNet International Conference - Networking in Education and Research, RoEduNet NER 2015 - Proceedings* (pp. 132–137). https://doi.org/10.1109/RoEduNet.2015.7311982

IoT-Analytics. (2015). *IoT Platforms: The Central Backbone for the Internet of Things*. *IoT Platforms*.

ITU. (2001). *The Tree and Tabular Combined Notation version 3*.

Jangra, A., Bhatia, V., Lakhinaza, U., & Singh, N. (2015). An Efficient Storage Framework design for Cloud Computing: Deploying Compression on De-duplicated No-SQL DB using HDFS. In *2015 1st International Conference on Next Generation Computing Technologies (NGCT-2015)* (pp. 55–60). https://doi.org/10.1109/NGCT.2015.7375082

Jemel, M., Msahli, M., & Serhrouchni, A. (2015). Digital safe: Secure synchronization of shared files. *Proceedings of the 2015 11th International Conference on Information Assurance and Security, IAS 2015*, 67–72. https://doi.org/10.1109/ISIAS.2015.7492747

Jeong, H.-D. J., Hyun, W., Lim, J., & You, I. (2012). Anomaly teletraffic intrusion detection systems on Hadoop-based platforms: A survey of some problems and solutions. In *Proceedings of the 2012 15th International Conference on Network-Based Information Systems, NBIS 2012* (pp. 766–770). https://doi.org/10.1109/NBiS.2012.139

Jie, L., Ghayvat, H., & Mukhopadhyay, S. C. (2015). Introducing Intel Galileo as a development platform of smart sensor: Evolution, opportunities and challenges. In *Proceedings of the 2015 10th IEEE Conference on Industrial Electronics and Applications, ICIEA 2015* (pp. 1797–1802). https://doi.org/10.1109/ICIEA.2015.7334403

Johri, P., Das, S., Kumar, A., & Arora, S. (2017). Security framework using Hadoop for Big Data. In *International Conference on Computing, Communication and Automation (ICCCA2017) Security* (pp. 268–272).

Kaa IoT Overview. (2017). Retrieved February 13, 2017, from https://www.kaaproject.org/overview/

Kaa open-source IoT Platform - Architecture Overview. (2016). Retrieved February 13, 2017, from https://kaaproject.github.io/kaa/docs/v0.10.0/Architecture-overview/

Kaa open-source IoT Platform - Connectivity. (2017). Retrieved February 13, 2017, from https://www.kaaproject.org/platform/#connectivity

Kaa open-source IoT Platform - FAQ. (2017). Retrieved February 13, 2017, from https://www.kaaproject.org/faq/

Kaa open-source IoT Platform - Kaa Server. (2016). Retrieved February 13, 2017, from https://www.kaaproject.org/platform/#kaa-server

Kaa open-source IoT Platform - SDK. (2017). Retrieved February 13, 2017, from https://www.kaaproject.org/platform/#sdk

Kaa Project - GitHub. (2016). Retrieved February 14, 2017, from https://github.com/kaaproject/kaa/tree/master/server/common/server-shared/src/main/java/org/kaaproject/kaa/server/sync

Kaa System Installation. (2016). Retrieved February 14, 2017, from https://kaaproject.github.io/kaa/docs/v0.10.0/Administration-guide/System-installation/

Kacfah Emani, C., Cullot, N., & Nicolle, C. (2015). Understandable Big Data: A survey. *Computer Science Review*, *17*, 70–81. https://doi.org/10.1016/j.cosrev.2015.05.002

Kalra, G. S. (2014). Threat analysis of an enterprise messaging system. *Network Security*, (12), 7–13. https://doi.org/10.1016/S1353-4858(14)70121-7

Khalid, Z., Fisal, N., Ullah, R., Safdar, H., Maqbool, W., Zubair, S., & Khan, A. S. (2013). M2M communication in virtual sensor network for SHAAL. *Jurnal Teknologi (Sciences and Engineering)*, *65*(1), 99–105. https://doi.org/10.11113/jt.v65.1749

Khan, R., Khan, S. U., Zaheer, R., & Khan, S. (2012). Future internet: The internet of things architecture, possible applications and key challenges. In *Proceedings - 10th International Conference on Frontiers of Information Technology, FIT 2012* (pp. 257–260). https://doi.org/10.1109/FIT.2012.53

Khedkar, S., & Malwatkar, G. M. (2016). Using Raspberry Pi and GSM Survey on Home Automation. In *International Conference on Electrical, Electronics, and Optimization Techniques, ICEEOT 2016* (pp. 758–761). https://doi.org/10.1109/ICEEOT.2016.7754787

Kopetz, H. (2011). *Real-Time Systems*. (J. A. Stankovic, Ed.) (2nd Ed.). London: Springer International Publishing. https://doi.org/10.1007/978-1-4419-8237-7

Kreps, J., Narkhede, N., & Rao, J. (2011). Kafka: a Distributed Messaging System for Log Processing. *ACM SIGMOD Workshop on Networking Meets Databases*, 6. Retrieved from http://research.microsoft.com/en-us/um/people/srikanth/netdb11/netdb11papers/netdb11-final12.pdf

Krishnan, K. (2013). *Data Warehousing in the Age of Big Data*. (A. Dierna & H. Scherer, Eds.) (1st ed.). Waltham: Elsevier.

Lee, K., Murray, D., Hughes, D., & Joosen, W. (2010). Extending Sensor Networks into the Cloud using Amazon Web Services. In *2010 IEEE International Conference on Networked Embedded Systems for Enterprise Applications, NESEA 2010*. https://doi.org/10.1109/NESEA.2010.5678063

Li, J., Zhang, Y., Chen, Y. F., Nagaraja, K., Li, S., & Raychaudhuri, D. (2013). A mobile phone based WSN infrastructure for IoT over future internet architecture. In *Proceedings - 2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing, GreenCom-iThings-CPSCom 2013* (pp. 426–433). https://doi.org/10.1109/GreenCom-iThings-CPSCom.2013.89

Li, K., Yuen, C., Kanhere, S., Hu, K., Zhang, W., Jiang, F., & Liu, X. (2018).

Understanding Crowd Density with A Smartphone Sensing System. In 2018 IEEE 4th World Forum on Internet of Things (WF-IoT) (pp. 517-522). https://doi.org/10.1109/WF-IoT.2018.8355126

Li, S., Xu, L. Da, & Zhao, S. (2015). The internet of things: a survey. *Information Systems Frontiers*, *17*(2), 243–259. https://doi.org/10.1007/s10796-014-9492-7

Luzuriaga, J. E., Perez, M., Boronat, P., Cano, J. C., Calafate, C., & Manzoni, P. (2015). Impact of mobility on Message Oriented Middleware (MOM) protocols for collaboration in transportation. In *Proceedings of the 2015 IEEE 19th International Conference on Computer Supported Cooperative Work in Design, CSCWD 2015* (pp. 115–120). https://doi.org/10.1109/CSCWD.2015.7230943

Ma, H. D. (2011). Internet of things: Objectives and scientific challenges. *Journal of Computer Science and Technology*, *26*(6), 919–924. https://doi.org/10.1007/s11390-011-1189-5

Magnoni, L. (2015). Modern Messaging for Distributed Sytems. *Journal of Physics: Conference Series*, *608*, 1–8. https://doi.org/10.1088/1742-6596/608/1/012038

Madukwe, K., Ezika, I., & Iloanuse, O. (2017). Leveraging Edge Analysis for Internet of Things Based Healthcare Solutions. In 2017 IEEE 3rd International Conference on Electro-Technology for National Development (NIGERCON) (pp. 720-725). https://doi.org/10.1109/NIGERCON.2017.8281940

Mainetti, L., Patrono, L., & Vilei, A. (2011). Evolution of Wireless Sensor Networks towards the Internet of Things: a Survey. In *SoftCOM 2011, 19th International Conference on Software, Telecommunications and Computer Networks* (pp. 1–6). Retrieved from http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6064380%5Cnhttp://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6064380

Malche, T., & Maheshwary, P. (2015). Harnessing the Internet of Things (IoT): A Review. *International Journal of Advanced Research in Computer Science and Software Engineering*, *5*(8), 320–323.

Malhotra, S., Doja, M. N., Alam, B., & Alam, M. (2016). Data Integration of Cloud-based and Relational Databases. In *International Conference on Soft Computing Techniques and Implementations, ICSCTI 2015* (pp. 83–86). https://doi.org/10.1109/ICSCTI.2015.7489542

Maló, P., Almeida, B., Melo, R., Kalaboukas, K., & Cousin, P. (2013). Self-Organised Middleware Architecture for the Internet-of-Things. In 2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing (pp. 445-451). https://doi.org/10.1109/GreenCom-iThings-CPSCom.2013.92

Mamum, M., Puspo, J., & Das, A. (2017). An Intelligent Smartphone Based Approach Using IoT for Ensuring Safe Driving. In International Conference on Electrical Engineering and Computer Science (ICECOS) 2017 (pp. 217-223). https://doi.org/10.1109/ICECOS.2017.8167137

Manwal, M., & Gupta, A. (n.d.). Big Data and Hadoop - A Technological Survey.

McMahon, A., & Farrell, S. (2009). Delay-and disruption-tolerant networking. *IEEE Internet Computing*, *13*(6), 82–87. https://doi.org/10.1109/MIC.2009.127

Mell, P., & Grance, T. (2011). *The NIST Definition of Cloud Computing: Recommendations of the National Institute of Standards and Technology*. *NIST Special Publication*. Gaithersburg.

Melorose, J., Perroy, R., & Careas, S. (2010). *Hadoop in Action*. (C. Lam, Ed.), *Statewide Agricultural Land Use Baseline 2015* (Vol. 1). Manning Publications. https://doi.org/10.1017/CBO9781107415324.004

Meng, Z., Wu, Z., Muvianto, C., & Gray, J. (2017). A Data-Oriented M2M Messaging Mechanism for Industrial IoT Applications. *IEEE Internet of Things Journal*, *4*(1), 236–246. https://doi.org/10.1109/JIOT.2016.2646375

Minerva, R., Biru, A., & Rotondi, D. (2015). Towards a definition of the Internet of Things (IoT). *IEEE: Internet of Things*, (1), 1–86. https://doi.org/10.5120/19787-1571

Minoli, D. (2013). What is the Internet of Things?

Misra, G., Kumar, V., Agarwal, A., & Agarwal, K. (2016). Internet of Things (IoT) – A Technological Analysis and Survey on Vision, Concepts, Challenges, Innovation Directions, Technologies, and Applications (An Upcoming or Future Generation Computer Communication System Technology). *American Journal of Electrical and Electronic Engineering*, *4*(1), 23–32. https://doi.org/10.12691/AJEEE-4-1-4

MongoDB. (2018). MongoDB Replica Set Data Synchronization. Retrieved February 23, 2018, from https://docs.mongodb.com/manual/core/replica-set-sync/

Morais, Y., & Elias, G. (2010). Customizing message-oriented mobile middleware. In *Proceedings - 6th International Conference on Wireless and Mobile Communications, ICWMC 2010* (pp. 356–361). https://doi.org/10.1109/ICWMC.2010.27

N., S. N., Kumar, S. M. D., & Banu, R. (2017). Internet of Things for Neophytes: A Survey. In *2017 International Conference on Electrical, Electronics, Communication, Computer and Optimization Techniques (ICEECCOT) Internet* (pp. 234–242).

Node-RED. (n.d.-a). Node-RED About. Retrieved February 22, 2018, from https://nodered.org/about/

Node-RED. (n.d.-b). Node-RED Flows. Retrieved February 22, 2018, from https://flows.nodered.org

Node-RED. (n.d.-c). Node-RED Getting Started. Retrieved February 22, 2018, from https://nodered.org/docs/getting-started/

Node-RED. (n.d.-d). Node-RED Security. Retrieved February 22, 2018, from https://nodered.org/docs/security

Node-RED. (n.d.-e). Node-RED Working with messages. Retrieved February 22, 2018, from https://nodered.org/docs/user-guide/messages

Nuratch, S. (2017). A Universal Microcontroller Circuit and Firmware Design and

Implementation for IoT-based Real-time Measurement and Control Applications. In *5th International Electrical Engineering Congress*. https://doi.org/10.1109/IEECON.2017.8075906

Oracle. (2018). MySQL Database Synchronization. Retrieved February 23, 2018, from https://dev.mysql.com/doc/workbench/en/wb-database-synchronization.html

Oussous, A., Benjelloun, F.-Z., Lahcen, A. A., & Belfkih, S. (2017). Big Data technologies: A survey. *Journal of King Saud University - Computer and Information Sciences*, 1–17. https://doi.org/10.1016/j.jksuci.2017.06.001

Palaiokrassas, G., Karlis, I., Litke, A., Charlaftis, V., & Varvarigou, T. (2017). An IoT Architecture for Personalized Recommendations over Big Data Oriented Applications. In *Proceedings - International Computer Software and Applications Conference* (Vol. 2, pp. 475–480). https://doi.org/10.1109/COMPSAC.2017.59

Pan, J., & McElhannon, J. (2018). Future Edge Cloud and Edge Computing for Internet of Things Applications. *IEEE Internet of Things Journal*, *5*(1), 439–449. https://doi.org/10.1109/JIOT.2017.2767608

Park, U., & Heidemann, J. (2011). Data Muling with Mobile Phones for Sensornets. *Proceedings of the 9th ACM Sensys Conference*, 162–175. https://doi.org/10.1145/2070942.2070960

Pereira, C., & Aguiar, A. (2014). Towards efficient mobile M2M communications: survey and open challenges. *Sensors (Basel, Switzerland)*, *14*(10), 19582–19608. https://doi.org/10.3390/s141019582

Pereira, C., Pinto, A., Aguiar, A., Rocha, P., Santiago, F., & Sousa, J. (2016). IoT Interoperability for Actuating Applications through Standardised M2M Communications.

Pereira, C., Rodrigues, J., Pinto, A., Rocha, P., Santiago, F., Sousa, J., & Aguiar, A. (2016). Smartphones as M2M gateways in smart cities IoT applications. In *2016 23rd International Conference on Telecommunications, ICT 2016*. https://doi.org/10.1109/ICT.2016.7500481

Pinto, J., Jain, P., & Kumar, T. (2016). Hadoop Distributed Computing Clusters for Fault Prediction.

Pinus, H. (2004). Middleware: Past and present a comparison, 1–5. Retrieved from http://userpages.umbc.edu/~dgorin1/451/middleware/middleware.pdf

Polianytsia, A., Starkova, O., & Herasymenko, K. (2016). Survey of Hardware IoT platforms. In *PIC S&T`2016* (pp. 152–153).

Premsankar, G., Di Francesco, M., & Taleb, T. (2018). Edge Computing for the Internet of Things: A Case Study. *IEEE Internet of Things Journal*, 1–10. https://doi.org/10.1109/JIOT.2018.2805263

R., C. G., & A., T. B. (2016). A Generic Tool to Process Mongodb or Cassandra Dataset using Hadoop Streaming. In *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)* (pp. 273–276).

RabbitMQ - Clients & Developer Tools. (2017). Retrieved February 4, 2017, from

https://www.rabbitmq.com/devtools.html

RabbitMQ - Protocols. (2017). Retrieved February 4, 2017, from https://www.rabbitmq.com/protocols.html

Ray, P. P. (2016). A survey on Internet of Things architectures. *Journal of King Saud University - Computer and Information Sciences*, 1–29. https://doi.org/10.1016/j.jksuci.2016.10.003

Raza, A., Ikram, A. A., Amin, A., & Ikram, A. J. (2017). A Review of Low Cost and Power Efficient Development Boards for IoT Applications. In *FTC 2016 - Proceedings of Future Technologies Conference* (pp. 786–790). https://doi.org/10.1109/FTC.2016.7821693

Reddy, G. S., Feng, Y., Liu, Y., Dong, J. S., Jun, S., & Kanagasabai, R. (2013). Towards Formal Modeling and Verification of Cloud Architectures: A Case Study on Hadoop. In *Proceedings - 2013 IEEE 9th World Congress on Services, SERVICES 2013* (Vol. 1, pp. 306–311). https://doi.org/10.1109/SERVICES.2013.47

Rimal, B. P., & Lumb, I. (2017). The Rise of Cloud Computing in the Era of Emerging Networked Society. In N. Antonopoulos & L. Gillam (Eds.), *Cloud Computing: Principles, Systems and Applications* (2nd Editio, pp. 3–23). Gewerbestrasse: Springer International Publishing. https://doi.org/10.1007/978-3-319-54645-2

S, J., Alam, S., & Srinivasan, R. (2017). Approaches to Deployment of Hadoop on Cloud Platforms: Analysis and Research Issues. In *2nd IEEE International Conference On Recent Trends in Electronics Information & Communication Technology (RTEICT)* (pp. 1985–1990).

Saari, M., Bin Baharudin, A. M., & Hyrynsalmi, S. (2017). Survey of Prototyping Solutions Utilizing Raspberry Pi. In *2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics, MIPRO 2017 - Proceedings* (pp. 991–994). https://doi.org/10.23919/MIPRO.2017.7973568

Said, O., & Masud, M. (2013). Towards internet of things: Survey and future vision. *International Journal of Computer Networks*, *5*(1), 1–17. Retrieved from http://www.cscjournals.org/csc/manuscript/Journals/IJCN/volume5/Issue1/IJCN-265.pdf

Santucci, G., & Lange, S. (2008). Internet of Things in 2020: A Roadmap for the Future, 1–32. Retrieved from http://www.smart-systems-integration.org/public/documents/publications/Internet-of-Things_in_2020_EC-EPoSS_Workshop_Report_2008_v3.pdf

Shaw, S. (2017). Hadoop Technology. In H. Geng (Ed.), *The Internet of Things and Data Analytics Handbook* (pp. 384–393). New Jersey: Wiley.

Shih, C.-S., Chuang, C.-C., & Yeh, H.-Y. (2017). Federating Public and Private Intelligent Services for IoT Applications. In *2017 13th International Wireless Communications and Mobile Computing Conference, IWCMC 2017* (pp. 558–563). https://doi.org/10.1109/IWCMC.2017.7986346

Singh, D., Tripathi, G., & Jara, A. J. (2014). A survey of Internet-of-Things: Future

Vision, Architecture, Challenges and Services. In *2014 IEEE World Forum on Internet of Things, WF-IoT 2014* (pp. 287–292). https://doi.org/10.1109/WF-IoT.2014.6803174

Singh, K. J., & Kapoor, D. S. (2017). Create Your Own Internet of Things: A survey of IoT platforms. *IEEE Consumer Electronics Magazine*. https://doi.org/10.1109/MCE.2016.2640718

Subramanian, S. (2015). RabbitMQ : A Cloud based Message Oriented Middleware. Retrieved February 4, 2017, from https://techietweak.wordpress.com/2015/08/14/rabbitmq-a-cloud-based-message-oriented-middleware/

Suciu, G., Halunga, S., Vulpe, A., & Suciu, V. (2013). Generic Platform for IoT and Cloud Computing Interoperability Study. *ISSCS 2013 - International Symposium on Signals, Circuits and Systems*. https://doi.org/10.1109/ISSCS.2013.6651222

Suganuma, T., Oide, T., Kitagami, S., Sugawara, K., & Shiratori, N. (2018). Multiagent-Based Flexible Edge Computing Architecture for IoT. *IEEE Network*, (January/February), 16–23. https://doi.org/10.1109/MNET.2018.1700201

Sukarsa, M., Mahajaya, N. S., Made, G., & Sasmita, A. (2012). Design of Message-Oriented Middleware Based on Social Messenger. *International Journal of Computer Science Engineering and Technology*, *2*(4), 1129–1134. Retrieved from http://www.ijcset.net/docs/Volumes/volume2issue4/ijcset2012020423.pdf

Szydlo, T., Brzoza-Woch, R., Sendorek, J., Windak, M., & Gniady, C. (2017). Flow-based programming for IoT leveraging fog computing. In *Proceedings - 2017 IEEE 26th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises, WETICE 2017* (pp. 74–79). https://doi.org/10.1109/WETICE.2017.17

T, J. (2016). Open Source Middleware for Internet of Things, *4*(11), 71–75. https://doi.org/10.17148/IJIREEICE.2016.41113

Talarian. (2000). *Talarian : Everything You Need To Know About Middleware*. Retrieved from http://media.techtarget.com/searchWebServices/downloads/Talarian.pdf

Talipov, E., Chon, Y., & Cha, H. (2013). Content Sharing Over Smartphone-Based Delay-Tolerant Networks. *IEEE Transactions on Mobile Computing (TMC)*, *12*(3), 581–595. https://doi.org/10.1109/TMC.2012.21

Tyagi, V., & Kumar, A. (2017). Internet of Things and Social Networks: A survey. In *International Conference on Computing, Communication and Automation (ICCCA2017) Internet* (pp. 1268–1270). https://doi.org/10.1109/CCAA.2017.8230013

Urard, P., & Vucinic, M. (2017). IoT Nodes: System-Level View. In M. Alioto (Ed.), *Enabling the Internet of Things* (1st Editio, pp. 47–68). Gewerbestrasse: Springer International Publishing. https://doi.org/10.1007/978-3-319-51482-6

Using Kaa endpoint SDKs. (2016). Retrieved February 13, 2017, from https://kaaproject.github.io/kaa/docs/v0.10.0/Programming-guide/Using-Kaa-endpoint-SDKs/

Vermesan, O., Friess, P., Guillemin, P., Sundmaeker, H., Eisenhauer, M., Moessner, K., … Baldini, G. (2014). Internet of Things Strategic Research and Innovation Agenda. In *Internet of Things – From Research and Innovation to Market Deployment* (p. 143). River Publishers. Retrieved from http://www.internet-of-things-research.eu/pdf/IERC_Cluster_Book_2014_Ch.3_SRIA_WEB.pdf

Vidanagama, V. G. T. N., Arai, D., & Ogishi, T. (2015). Service environment for smart wireless devices: An M2M gateway selection scheme. *IEEE Access*, *3*, 666–677. https://doi.org/10.1109/ACCESS.2015.2436907

Videla, A., & Williams, J. J. W. (2012). *RabbitMQ in action*. Stamford: Manning Publications Co. Retrieved from http://cds.cern.ch/record/1528001%5Cn/Users/jrimland/Library/Application Support/Firefox/Profiles/6b0hv2c5.default/zotero/storage/IX59NZZB/1528001.html

Wang, C., Zheng, Z., & Yang, Z. (2014). The Research of Recommendation System Based on Hadoop Cloud Platform. In *The 9th International Conference on Computer Science & Education (ICCSE 2014)* (pp. 193–196). Vancouver.

Wang, Z., Dai, W., Wang, F., Deng, H., Wei, S., Zhang, X., & Liang, B. (2015). Kafka and its using in high-throughput and reliable message distribution. In *Proceedings - 8th International Conference on Intelligent Networks and Intelligent Systems, ICINIS 2015* (pp. 117–120). https://doi.org/10.1109/ICINIS.2015.53

Wei-Lin, Y., Lin, Y.-B., Yang, M.-T., & Lin, J.-H. (2017). ArduTalk: An Arduino Network Application Development Platform Based on IoTtalk. *IEEE Systems Journal (Volume: PP, Issue: 99)*, 1–9. https://doi.org/10.1109/JSYST.2017.2773077

Wilderness-Labs. (2018). About Netduino. Retrieved March 16, 2018, from http://developer.wildernesslabs.co/Netduino/About/

Wong, K.-S., Wan, T.-C., & Ang, W.-C. (2016). A Survey on Current Status of Disruption Tolerant Network Support for Multicast. In *3rd International Conference on Computer and Information Sciences (ICCOINS)* (pp. 276–281).

Wu, M., Lu, T. J., Ling, F. Y., Sun, J., & Du, H. Y. (2010). Research on the architecture of Internet of Things. In *ICACTE 2010 - 2010 3rd International Conference on Advanced Computer Theory and Engineering, Proceedings* (Vol. 5, pp. 484–487). https://doi.org/10.1109/ICACTE.2010.5579493

Zachariah, T., Klugman, N., Campbell, B., Adkins, J., Jackson, N., & Dutta, P. (2015). The Internet of Things Has a Gateway Problem. In *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications - HotMobile '15* (pp. 27–32). https://doi.org/10.1145/2699343.2699344

Zhang, J., Shan, L., Hu, H., & Yang, Y. (2012). Mobile cellular networks and wireless sensor networks: Toward convergence. *IEEE Communications Magazine*, *50*(3), 164–169. https://doi.org/10.1109/MCOM.2012.6163597

Zhang, Q., Cheng, L., & Boutaba, R. (2010). Cloud computing: State-of-the-art and research challenges. *Journal of Internet Services and Applications*, *1*(1), 7–18. https://doi.org/10.1007/s13174-010-0007-6

Zheng, K., & Jiang, W. (2014). A Token Authentication Solution for Hadoop Based on Kerberos Pre-Authentication. In *2014 International Conference on Data Science and Advanced Analytics (Dsaa)* (pp. 354–360).

Zhong, C., Zhu, Z., & Huang, R. (2017). Study on the IOT Architecture and Access Technology. In *2017 16th International Symposium on Distributed Computing and Applications to Business, Engineering and Science (DCABES)* (pp. 113–116). IEEE. https://doi.org/10.1109/DCABES.2017.32

Zhou, H. (2013). *The Internet of Things in the Cloud: A Middleware Perspective*. (H. Zhou, Ed.) (1st Editio). Boca Raton: CRC Press. https://doi.org/doi:10.1201/b13090-14

Zhu, Q., Wang, R., Chen, Q., Liu, Y., & Qin, W. (2010). IOT Gateway: BridgingWireless Sensor Networks into Internet of Things. *2010 IEEE/IFIP International Conference on Embedded and Ubiquitous Computing*, 347–352. https://doi.org/10.1109/EUC.2010.58