INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Departamento de Engenharia de Electrónica e Telecomunicações e de Computadores

Plataforma descentralizada de monitorização de poluição

Ivo Miguel Almeida Pedroso

Licenciado em Engenharia Informática e de Computadores

Projecto Final para obtenção do Grau de Mestre
em Engenharia Informática e de Computadores

Orientadores : Professor Doutor Nuno Miguel Soares Datia
Professor Doutor Nuno Miguel Machado Cruz

Júri:

Presidente: Professor Doutor Carlos Jorge de Sousa Gonçalves

Vogais: Professor Doutor José Manuel de Campos Lages Garcia Simão
Professor Doutor Nuno Miguel Soares Datia

Fevereiro, 2022

INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Departamento de Engenharia de Electrónica e Telecomunicações e de Computadores

Plataforma descentralizada de monitorização de poluição

Ivo Miguel Almeida Pedroso

Licenciado em Engenharia Informática e de Computadores

Projecto Final para obtenção do Grau de Mestre
em Engenharia Informática e de Computadores

Orientadores : Professor Doutor Nuno Miguel Soares Datia
Professor Doutor Nuno Miguel Machado Cruz

Júri:

Presidente: Professor Doutor Carlos Jorge de Sousa Gonçalves

Vogais: Professor Doutor José Manuel de Campos Lages Garcia Simão
Professor Doutor Nuno Miguel Soares Datia

Fevereiro, 2022

À minha família.

Agradecimentos

Aos meus orientadores, pela sua disponibilidade, orientação, motivação e apoio, que sempre demonstraram ao longo de todo o projeto, os meus profundos agradecimentos.

Agradeço também à minha família e amigos, por todo o apoio e amizade que sempre me transmitiram ao longo do meu percurso académico.

Resumo

A poluição atmosférica é um fator que afeta cada vez mais a saúde e bem-estar da população, com especial incidência nos grandes meios urbanos e industriais. Dados da Organização Mundial de Saúde (OMS) indicam que a poluição atmosférica é um fator crítico de risco para doenças não transmissíveis, causador de 24% das mortes por doença cardiovascular, 25% dos acidentes vascular cerebral, 43% das doenças pulmonares obstrutivas crónicas e 29% dos cancros do pulmão. Atualmente é praticável a recolha de medições da poluição atmosférica através de dispositivos portáteis e sensores de baixo custo, permitindo a observação de dados em tempo real das observações realizadas, juntamente com a localização e hora de observação. O objetivo deste projeto de final de mestrado é desenhar e implementar uma plataforma descentralizada em código aberto baseada na nuvem, que permita acoplar e desacoplar em tempo real dispositivos de medição da poluição atmosférica, armazenar medições observadas, agregar esses dados e disponibilizar um *dashboard* que permita posterior análise dos dados observados. A arquitetura proposta para esta plataforma é composta por elementos implementados em ambiente de contentores de execução virtual, utilizado componentes da *framework* FIWARE e componentes desenvolvidos à medida. Utiliza como intermediário de comunicação com os dispositivos de observação (dispositivos IoT) de poluição atmosférica, a plataforma *The Things Network Stack* que serve de interface com redes LoRaWAN, permitindo comunicação de longo alcance com dispositivos de baixa potência. O resultado final é um protótipo da plataforma proposta, com os seus componentes correndo na nuvem, cuja instalação e instanciação é possível através da execução de um simples comando. Também permite o registo de dispositivos na plataforma através da submissão de um simples formulário com informação básica que identifique o dispositivo.

Palavras-chave: FIWARE, Microserviços, IoT, LoRaWAN, Cidades Inteligentes

Abstract

Air pollution is a factor that increasingly affects the health and well-being of the population, with a particular focus on large urban and industrial environments. World Health Organization (WHO) data indicate that air pollution is a critical risk factor for non-communicable diseases, causing 24% of deaths from cardiovascular disease, 25% of stroke, 43% of chronic obstructive pulmonary diseases and 29% of lung cancers.

Currently, it is feasible to collect air pollution measurements through portable devices and low-cost sensors, allowing the observation of data in real time of the observations made, together with the location and time of observation.

The purpose of this final master's project is to design and implement a decentralized open source cloud-based platform dashboard that allows to attach and detach in real time, air pollution measuring devices, store measured data, aggregate that data and allow subsequent analysis of the observed data.

The proposed architecture for this platform is composed of elements in a virtual execution container environment, using components from the framework FIWARE and custom developed components.

Use as a communication intermediary with air pollution observation devices (IoT devices), The Things Network Stack platform, which serves as an interface with LoRaWAN networks, allowing long-range communication to low power devices.

The end result is a prototype of the proposed platform, with it's components running in the cloud, whose installation and instantiation is possible through the execution of a simple command. It also allows the registration of devices in the platform, by submitting a simple form with basic information that identifies the device.

Keywords: FIWARE, Microservices, IoT, LoRaWAN, Smart Cities

Índice

Lista de Figuras

Lista de Tabelas

1

Introdução

A poluição é uma preocupação constante e consensual entre a população, nomeadamente a poluição atmosférica, pois provoca graves problemas de saúde [4]. A OMS estima que a poluição atmosférica seja responsável por 4.2 milhões de mortes anuais devido acidente vascular cerebral, doença cardíaca, cancro do pulmão, doenças respiratórias crónicas e que 91% da população mundial viva em locais onde o nível da qualidade do ar excede os limites definidos pela OMS [14].

Entidades com responsabilidade na gestão dos grandes centros urbanos, aumentam o investimento em sensores de monitorização [18], com o objetivo de monitorizar a qualidade do ar em tempo real.

Apesar de melhorar a precisão nas medições, o uso de estações de monitorização especializadas traz algumas desvantagens, nomeadamente o elevado custo financeiro na aquisição e manutenção, a pouca ou nenhuma mobilidade e uma reduzida área de cobertura dos centros urbanos. Estas limitações fomentam as entidades a procurarem novas soluções, que permitam poupança na aquisição e gestão de sensores, com maior mobilidade e com melhor capacidade de escala. Uma solução possível é o uso pequenos sensores de monitorização da qualidade do ar, móveis, auto-alimentados, com capacidade de comunicação de longo alcance e de baixo consumo [7, 23].

Para suportar tal rede de dispositivos disseminados pelos centros urbanos, potencialmente geradora de grande volume de informação, é necessária uma plataforma que recolha os dados registados, os armazene e disponibilize uma forma de obtenção ou visualização, que permita posterior análise dos dados registados.

Este documento apresenta uma proposta de solução de uma plataforma que permita comunicar, gerir e armazenar dados de dispositivos IoT com as características acima indicadas. É uma solução descentralizada em código aberto, baseada em componentes disponibilizados em contentores de execução virtual e que disponibiliza uma interface Web com o utilizador, de fácil uso e simplicidade.

A Figura 1.1 ilustra de forma geral o contexto no qual se enquadra a plataforma apresentada. A plataforma serve de intermediário entre os utilizadores e dispositivos IoT dispersos geograficamente, comunicando através de uma rede móvel de baixa potência.

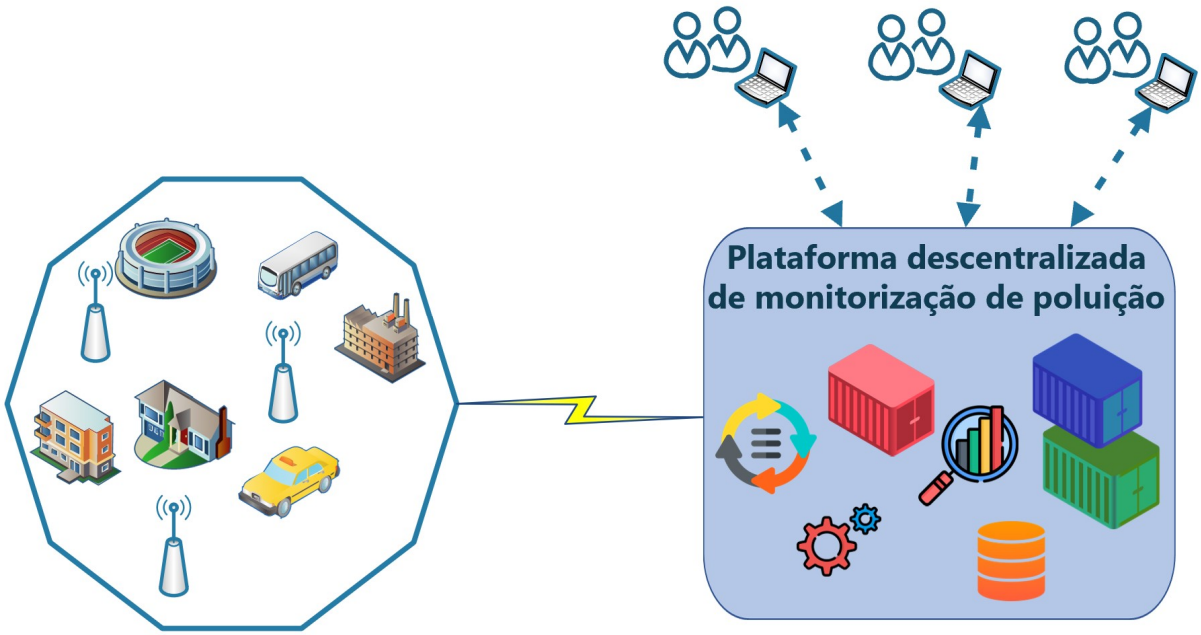

Figura 1.1: Diagrama geral do contexto de utilização da plataforma.

1.1 Enquadramento

Todos nós somos afetados pela poluição atmosférica, mesmo que no presente não sintamos os efeitos negativos sobre a nossa saúde, num futuro próximo, nós, os nossos familiares, amigos e conhecidos, podem vir a ser vítimas desta problemática. Sendo um problema coletivo, cabe a cada um de nós contribuir e ajudar a mitigar este problema.

Este projeto permite combinar o processo de descoberta e aprendizagem inerentes a um projeto final de mestrado, com o contributo no desenvolvimento de soluções que ajudem a sociedade, a monitorizar, a divulgar e a combater a poluição atmosférica.

O sistema proposto permite a recolha de observações transmitidas por diferentes tipos de dispositivos, mantendo um histórico das observações recolhidas, as quais serão agregadas, permitindo posterior visualização e análise da informação gerada e armazenada, através de um *dashboard*.

1.2 Objetivos

A proposta deste documento é desenhar uma arquitetura de solução e implementar protótipo de uma plataforma em código aberto, descentralizada, que permite o registo e consulta de dados observados por pequenos dispositivos móveis de baixa potência, disseminados geograficamente sobre uma área urbana, os quais monitorizam a qualidade do ar. A arquitetura permite registar os dispositivos de forma simples e sem necessidade de posterior acompanhamento. A instalação da plataforma é possível através da execução de um simples comando, o qual instancia todos os componentes da plataforma, a executar sobre um ambiente "contentorizado", na nuvem ou *on-premises*.

Os objetivos deste trabalho são: 1. desenhar uma arquitetura de gestão e armazenamento de sensores, e 2. implementar um protótipo de uma plataforma descentralizada, de código aberto, baseada em contentores. Como domínio do problema, ilustrativo da solução, será focado a monitorização da poluição atmosférica.

Esta plataforma permite a recolha de dados gerados por diversos dispositivos IoT móveis espalhados por uma área urbana. Com a utilização de um elevado número de dispositivos dispersos geograficamente, é desejável que os mesmos necessitem de pouca manutenção, ou seja, que os dispositivos sejam autónomos energeticamente durante largos períodos de tempo (vários meses ou anos) sem trocas constantes de baterias. Tal condiciona ao uso de dispositivos de muito baixo consumo, obrigando também à utilização de um meio de comunicação de muito baixa potência. A plataforma tem que suportar a comunicação com os dispositivos através de tal meio de comunicação.

Para associar os dispositivos à plataforma, esta tem que permitir o registo de dispositivos. O registo deverá ser simples e rápido, necessitando apenas a indicação da identificação do dispositivo e de quais os parâmetros que o dispositivo observa.

A orquestração dos componentes deverá ser automatizada, dando resposta a situações de falha e necessidades de escala.

A segurança também é um fator importante na implementação, nomeadamente na permissão de acessos externos aos componentes da plataforma, bem como na segurança em troca de dados entre componentes e possíveis serviços externos.

1.3 Estrutura do documento

Este documento é composto por 5 capítulos. O primeiro capítulo é uma introdução ao projeto, com o enquadramento e objetivos do mesmo. O segundo capítulo descreve o estado da arte, no qual são analisadas algumas soluções com funcionalidades similares, ou que resolvem alguns dos problemas identificados. De seguida, no terceiro capítulo, descreve-se a implementação do projeto. O quarto capítulo descreve os testes de carga e escala realizados à solução apresentada. O último capítulo faz a conclusão deste documento, bem como algumas propostas de melhoria para trabalhos futuros.

2

Estado de Arte

Com a ascensão dos grandes centros urbanos, com milhares de pessoas a residir e deslocar-se dentro e fora das cidades diariamente, tornou-se essencial às entidades de gestão destas grandes cidades, o recurso à tecnologia, que lhes permita de forma informada, a tomada de decisão. O recurso à utilização de múltiplos e diversos sensores dispersos pelos centros urbanos, é cada vez mais comum, os quais são potencialmente geradores de muita informação.

Mas não basta ter muitos sensores e muitos dados, para se ter uma cidade inteligente. É necessário que os dados gerados sejam acessíveis, sejam tratados e que a informação gerada por múltiplas fontes possa ser relacionada entre si, para que a mesma seja útil, dê uma perspetiva mais realista da situação atual e no momento adequado, permitindo analisar o passado, refletir o presente e perspetivar o futuro.

Com este pressuposto em mente, são apresentados neste capítulo alguns projetos de *frameworks*, plataformas e aplicações, orientados a dar suporte ao desenvolvimento de aplicações e serviços descentralizados, que auxiliam à implementação de soluções inteligentes orientadas para tornar as cidades inteligentes, ou seja, cidades conectadas, sustentáveis e otimizadas, onde a tecnologia é um pilar essencial na gestão de recursos, análise e tomada de decisão.

2.1 FIWARE

A FIWARE [5] é uma iniciativa com o apoio da União Europeia, cujo foco é o desenvolvimento de um ecossistema de componentes de livre acesso, assentes em interfaces comuns e modelos de dados standards, orientados para facilitar o desenvolvimento de serviços e aplicações inteligentes, enquadrados também no contexto das cidades inteligente. Define um conjunto de especificações e disponibiliza uma implementação de referência, materializada em diferentes componentes, formando uma *framework* de acesso livre à comunidade.

Os componentes da *framework* FIWARE são orientados à interoperabilidade e troca de dados de contexto, graças a uma interface programática comum, a NGSI [12], e à utilização de modelos de dados standard e expansíveis. Estas características promovem a simplicidade e facilidade na interligação entre componentes FIWARE e outros componentes terceiros.

A FIWARE oferece um largo catálogo de componentes, ricos em funcionalidade e assentes em 3 camadas principais, nomeadamente a camada de gestão de dados de contexto, a camada de interface com outros sistemas na obtenção de dados de contexto, e a camada de processamento e análise dos dados de contexto. Também existe uma camada transversal de suporte, relacionada com segurança, gestão e monetização. A Figura 2.1 ilustra as camadas FIWARE e enumeração de alguns dos seus componentes.

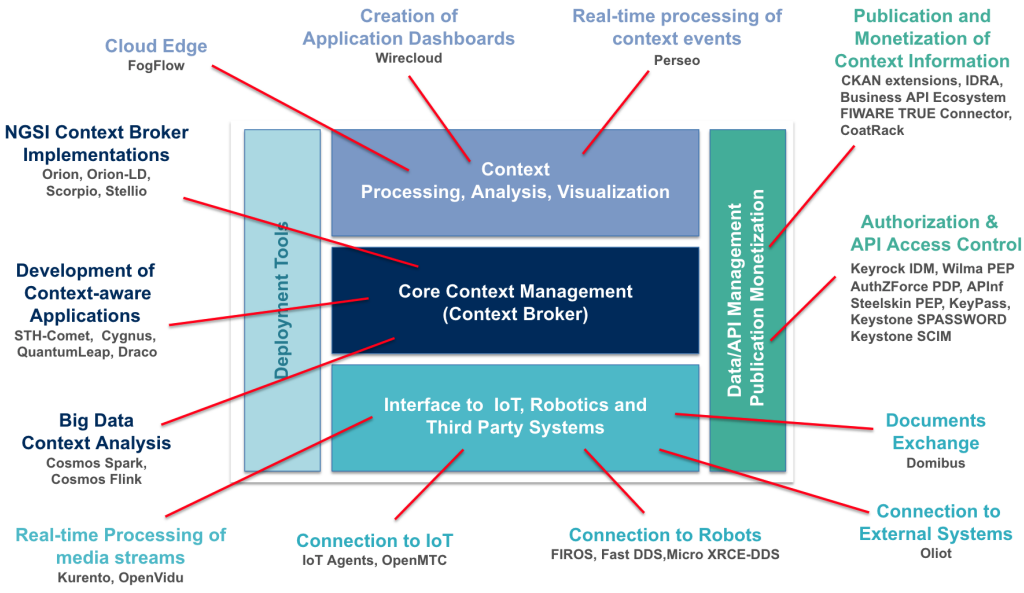

Figura 2.1: Camadas FIWARE e alguns dos seus componentes. [a]

[a] Retirado de: https://github.com/FIWARE/catalogue.

Para que uma aplicação possa ser considerada "Powered by FIWARE", é obrigatório o uso de um componente gestor de contexto na aplicação, o Context Broker. Este é geralmente o componente central das aplicações que recorrem à utilização da *framework* FIWARE, sendo este responsável pela informação de contexto corrente da aplicação. É tipicamente no Context Broker, por onde fluem os dados dos restantes componentes, nomeadamente os que fornecem informação de contexto, ou os que leem informação de contexto.

No âmbito das cidades inteligentes, a obtenção de dados de dispositivos IoT sensorizados, é um fator relevante na aquisição de dados. O FIWARE disponibiliza para esse efeito, um conjunto de componentes para interação com dispositivos IoT em diferentes meios de comunicação, bem como disponibiliza um componente genérico que permite a adaptação e extensibilidade do seu código fonte, para utilização em cenários não abrangidos pela funcionalidade dos componentes já existentes.

A Figura 2.2 ilustra uma arquitetura referência de uma aplicação orientada às cidades inteligentes.

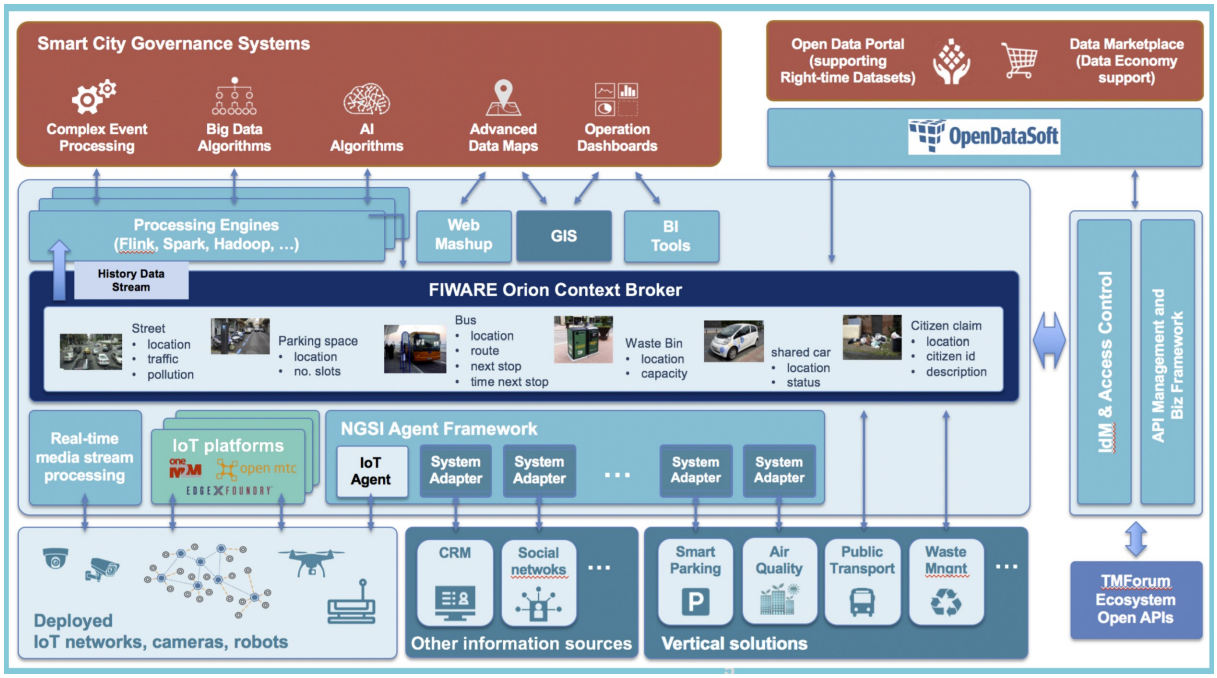

Figura 2.2: Arquitetura de referência de uma aplicação para cidade inteligente[a].

[a]Retirado de: https://oascities.org/wp-content/uploads/2019/01/20190117_OASC_FIWARE_Ahle_v2.pdf

Os componentes da *framework* FIWARE são disponibilizados em repositório público na Internet, nomeadamente o seu código fonte, permitindo o livre acesso, utilização e

adaptação. Também são disponibilizados em repositório público, imagens de contentores de execução virtual dos componentes da *framework*, as quais são parametrizáveis através da definição de variáveis de ambiente na instanciação dos respetivos contentores.

Esta alta disponibilidade no acesso aos componentes, a elevada parametrização e a flexibilidade de interligação entre componentes, são fatores que permitem alavancar o desenvolvimento de aplicações inteligentes, permitindo orientar o esforço de implementação, para o desenvolvimento de componentes com funcionalidades mais específicas e que não estão disponíveis na comunidade de acesso livre.

Um dos aspetos negativos observados nesta *framework*, foi os diferentes níveis de documentação de suporte e nível de acompanhamento por parte da comunidade, entre os diferentes componentes da *framework* FIWARE. Existem componentes com documentação bastante detalhada, em especial os componentes mais utilizados pela comunidade, os quais a comunidade é bastante ativa no seu desenvolvimento e no suporte de problemas. Mas também se verifica, que alguns componentes disponibilizam pouca ou insuficiente documentação, o que dificulta a utilização dos mesmos. Nestes casos é comum encontrar na documentação, simples descrições de parâmetros de configuração do componente, não ficando claro qual o comportamento esperado com a utilização de diferentes valores de dado parâmetro. Por vezes também é comum encontrar exemplos de utilização para certos cenários em concreto, com uma dada parametrização, sendo necessário o recurso a testes de tentativa/erro com diferentes valores de configuração, para compreender o comportamento do componente com uma configuração alternativa.

2.2 TTN —The Things Network

O recurso à utilização de pequenos dispositivos disseminados por grandes áreas urbanas, nomeadamente a utilização de dispositivos IoT sensorizados, que enviam informação de forma regular e autónoma, implica que os mesmos consigam transmitir e receber dados, através de algum meio de comunicação. No entanto, a utilização massiva destes tipos de dispositivos, por vezes limita a possibilidade de recorrer a meios mais tradicionais de comunicação, como a utilização de cabos ethernet ou comunicação sem fios, como o GSM, 3G, 4G, 5G ou WiFi por exemplo. O recurso a cabos de rede nem sempre é fisicamente viável em todos os locais que os dispositivos possam estar localizados, e dependendo do número de dispositivos existente, o custo financeiro com a utilização tecnologias sem fios como as acima indicadas, pode ser demasiado elevado.

O consumo energético inerente ao meio de comunicação usado, também é um fator importante em consideração, pois caso os dispositivos IoT sejam móveis e alimentados através de baterias, importa garantir a longevidade das mesma, diminuindo os custos de manutenção associados.

Com esta limitações em mente, surge a necessidade de utilização de um meio de comunicação sem fios, de longo alcance e de baixa potência, que permita disseminar múltiplos dispositivos IoT de forma conectada sobre grandes áreas, e com baixo custo de manutenção.

Um dos protocolos orientado para transmissão de informação sem fios com muito baixo consumo entre longas distâncias, é o protocolo LoRaWAN [22] (*Long Range Wide Area Network*). A comunicação em LoRaWAN é realizada entre dispositivos com suporte para tal e *gateways* LoRaWAN, que estejam ao alcance desses dispositivos. Estes *gateways* têm a responsabilidade de transmitir os dados recebidos dos dispositivos, através de meios de comunicação mais tradicionais, nomeadamente ethernet, 3G ou Wifi, para servidores de rede, que encaminham a informação transmitida pelos dispostos, para servidores aplicacionais que disponibilizam o acesso a esses dados a aplicações. A LoRaWAN suporta bidirecinalidade, ou seja, a transmissão de dados entre *gateways* e dispositivos, e vice-versa. A Figura 2.3 ilustra a arquitetura geral de uma rede LoRaWAN.

Para tomar partido deste meio de comunicação, existem várias implementações dos servidores de rede e servidores aplicacionais que dão suporte a utilização do LoRaWAN, abstraindo a complexidade associada à instalação, configuração, interligação e acesso entre dispositivos, *gateways*, servidores de rede e servidores aplicacionais. A maioria das implementações existente é proprietária e com custo de acesso associados.

Uma destas implementações é a plataforma TTN (*The Things Network*), que oferece acesso a um conjunto de servidores de rede e servidor aplicacional, via Web ou via API (*Application Programming Interface*), permitindo o registo de dispositivos e *gateways* numa rede LoRAWAN. A plataforma disponibiliza diversos métodos de integração com outras aplicações, no acesso aos dados transmitidos pelos dispositivos registados na plataforma, nomeadamente através do protocolo MQTT [21] (*Message Queuing Telemetry Transport*), que implementa o padrão Publish-Subscribe [17], permitido a notificação de todas as aplicações interessadas, da existência de novas mensagens de um dado dispositivo. A Figura 2.4 identifica as formas de integração disponível, no acesso aos dados transmitidos pelos dispositivos registados na plataforma TTN.

A TTN, disponibiliza em código aberto a implementação de um servidor de rede LoRaWAN, que permite instanciar o próprio servidor de rede LoRaWAN. Mas também

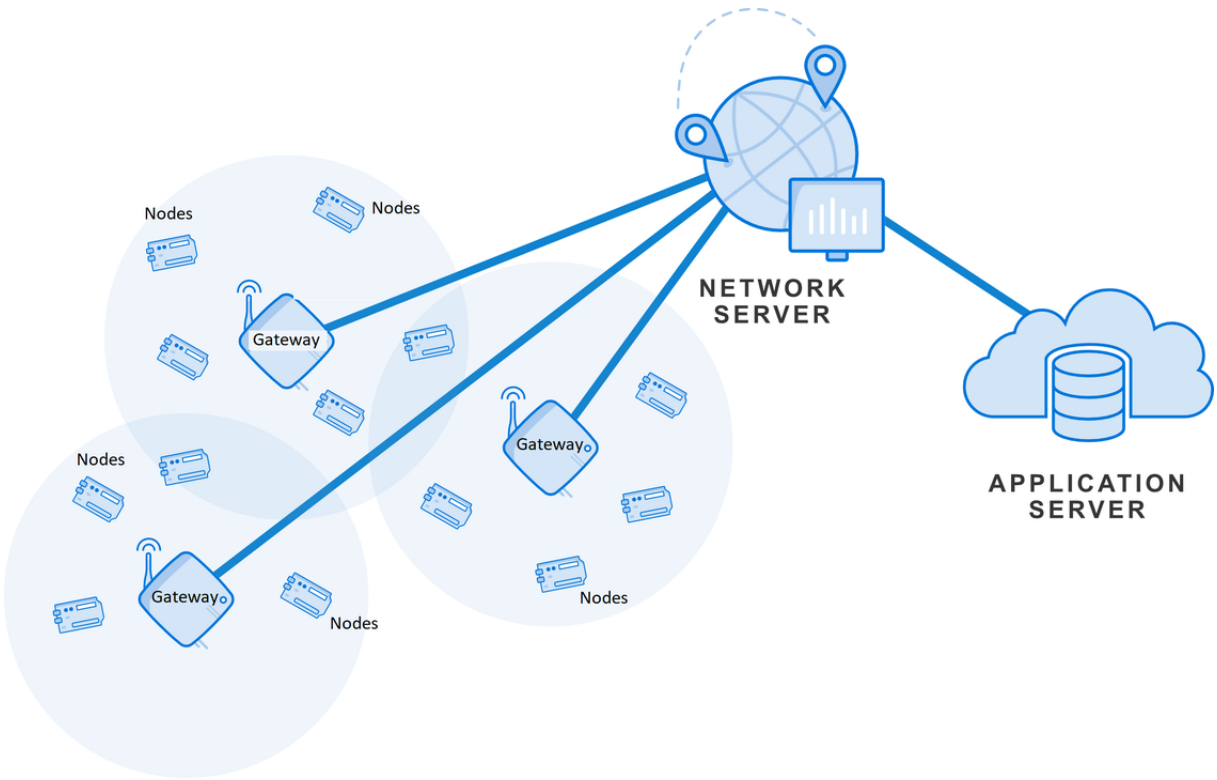

Figura 2.3: Arquitetura da Rede LoRaWAN[a]

.

[a]Retirado de: https://www.thethingsnetwork.org/docs/network/architecture/

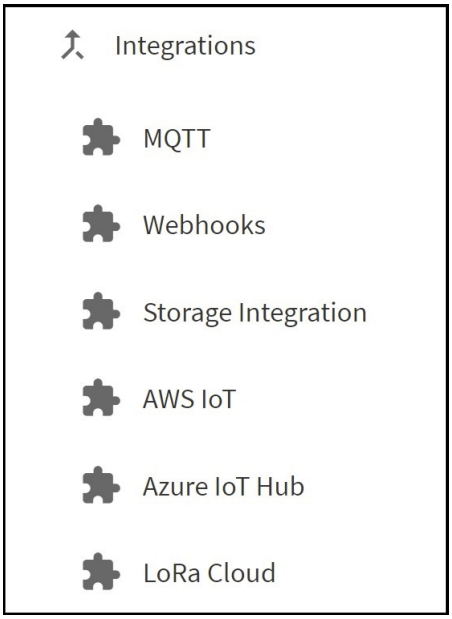

Figura 2.4: Formas de integração disponíveis na TTN para acesso a dados.

disponibiliza um acesso comunitário à sua plataforma aplicacional na nuvem, o qual

permite a utilização de um conjunto de serviços que permitem interagir com dispositivos na rede LoRaWAN da TTN, sem custos financeiros associados.

A utilização desta plataforma, implica a necessidade da criação de uma conta de utilizador. Caso o utilizador possua *gateways* LoRaWAN, também é possível associar as mesmas à plataforma TTN, permitindo que estes façam parte da rede LoRaWAN da TTN.

Após a criação da conta de utilizador, é necessário a definição de uma aplicação na plataforma, na qual serão posteriormente registados os dispositivos IoT que o utilizador pretende adicionar à rede LoRaWAN. O acesso programático à TTN via API, requer a criação de tokens de autenticação com permissões necessárias, nomeadamente permissões que permitam o registo de novos dispositivos, caso assim o pretenda.

Uma desvantagem significativa na utilização deste recurso, é a necessidade de configurar um *script* em Javascript na aplicação criada, que permita descodificação das mensagens recebidas dos dispositivos IoT, e converta esses dados para um formato JSON (*JavaScript Object Notation*), que será entregue aos subscritores de mensagens via MQTT (*Message Queuing Telemetry Transport*).

2.3 KubeEdge

No desenvolvimento de sistemas descentralizados, a orquestração de componentes permite a gestão e coordenação de forma automatizada, dando resposta a mudanças de configuração e reação a falhas. O KubeEdge [24] é um sistema de código aberto que permite estender a capacidade de orquestração de aplicações contentorizadas, a dispositivos que se encontram nos extremos da arquitetura de uma aplicação (Edge) [19], nomeadamente a dispositivos que tipicamente não estão na nuvem, como os dispositivos IoT ou PC. Estes dispositivos são habitualmente chamados de dispositivos Edge.

Ao fornecer esta capacidade de orquestração a aplicações contentorizadas nestes dispositivos, fornecendo suporte para rede, desenvolvimento de aplicações e sincronização de metadata, permite a descentralização das aplicações com a deslocalização de alguma da computação habitualmente realizada na nuvem, para os dispositivos Edge. As vantagens identificadas com esta abordagem são:

- **Menor latência** - Elimina a necessidade de envio de dados para processamento central, para posterior atuação;
- **Redução na quantidade de dados transmitidos** - Capacidade de análise local, com envio apenas dos dados realmente necessários ao nível global;

- **Maior privacidade** - Processamento local, evita envio de dados sensíveis;
- **Maior autonomia** - Capacidade de operar *offline* por períodos de tempo;

A orquestração dos componentes da aplicação é suportada com o recurso ao sistema Kubernetes[1]. Para contentorizar componentes é suportado Docker[2], Containerd[3], Cri-o[4] ou Virtlet[5].

Uma das desvantagens desta solução, é que as funcionalidades oferecidas, são apenas orientadas para dar suporte à arquitetura da aplicação a implementar e não aos objetivos da aplicação, além da descentralização. Também parece que esta solução não é apropriada para dispositivos Edge de baixo recursos computacionais e também de muito baixo consumo, pois obriga a ter a componente aplicacional contentorizada e o respetivo ambiente de execução a correr no dispositivo. Isso provoca um maior consumo e afeta a autonomia energética, ou simplesmente o dispositivo pode não ter suficiente capacidade de processamento para executar tal carga, apesar de o KubeEdge também suportar dispositivos com microprocessador ARM.

A 2.5 ilustra a arquitetura base da *framework* KubeEdge.

2.4 FIWARE FogFlow

O FogFlow [3] é uma *framework* de suporte à computação em dispositivos Edge, nomeadamente dispositivos IoT, os quais fazem parte de um sistema distribuído, gerido centralmente através de serviços computacionais na nuvem.

A motivação do FogFlow é a orquestração de serviços IoT distribuídos geograficamente. A infraestrutura do FogFlow é dividida verticalmente por nó na nuvem, nós Edge e sensores/atuadores. A Figura 2.6 ilustra essa infraestrutura.

O modelo de programação suportado é baseado em intenções, ou seja, o programador indica que operações realizar, a topologia pretendida, e como os dados recebidos são processados, ou seja, que tipos de dados são processados por cada nó da topologia e como os nós estão ligados entre si. Indicadas as intenções, a infraestrutura automaticamente determina que tarefas deve realizar e onde, quando chegam novos inputs à infraestrutura. O modelo gerado com as intenções definidas pelo programador é armazenado num grafo e distribuído por todos os nós.

[1]Disponível em: https://kubernetes.io/docs/home/
[2]Disponível em: https://www.docker.com/
[3]Disponível em:https://containerd.io/
[4]Disponível em:https://cri-o.io/
[5]Disponível em:https://docs.virtlet.cloud/

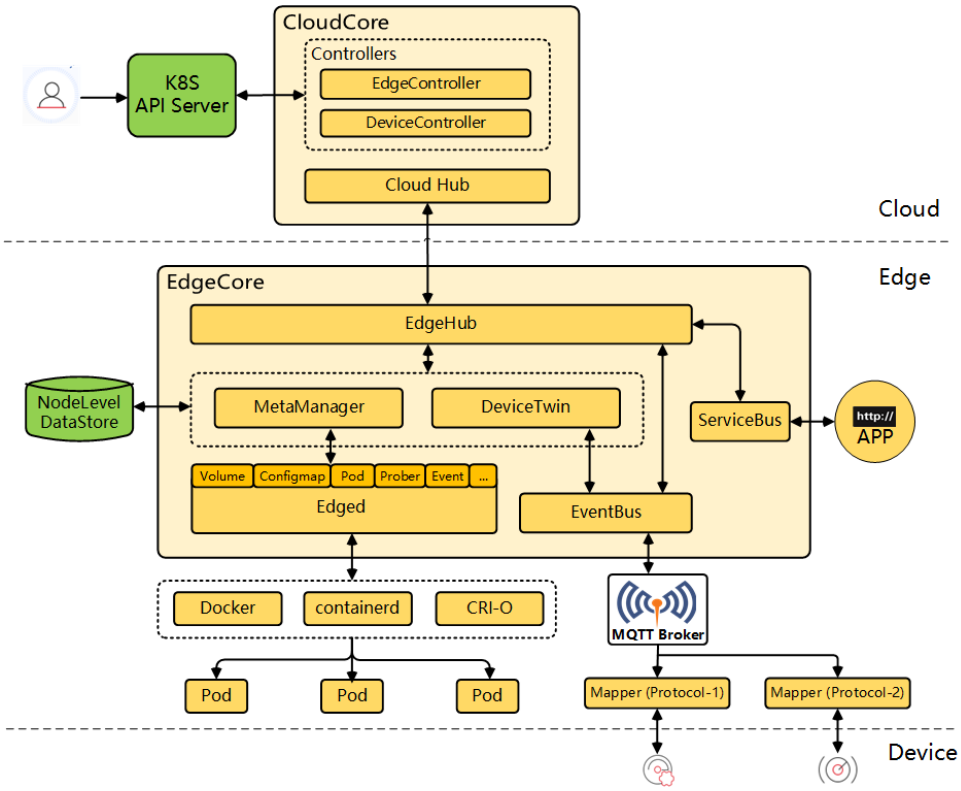

Figura 2.5: Arquitetura base do KubeEdge[a]

[a]Retirado de: https://kubeedge.io/ko/docs/kubeedge/

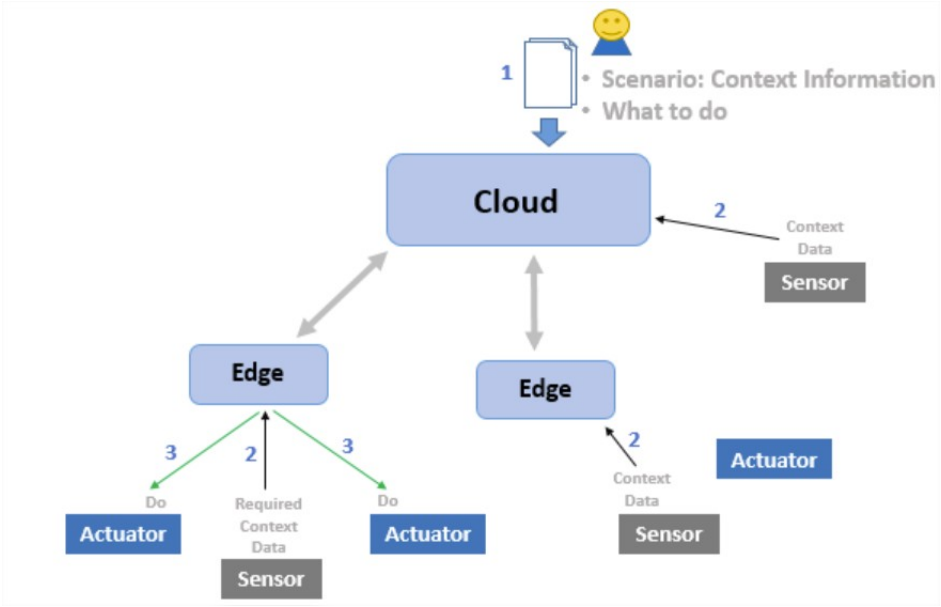

Figura 2.6: Infraestrutura FogFlow[a]

.

[a]Retirado de: https://fogflow.readthedocs.io/en/latest/introduction.html

Este tipo de solução permite uma computação descentralizada, onde o elemento na nuvem (ponto central da infraestrutura) apenas recebe a informação necessária e os nós Edge processam localmente os dados recebidos e encaminham apenas o necessário para o nó na nuvem. A orquestração nesta infraestrutura é conduzida por contexto e não por eventos, como em outras soluções, por exemplo a KEDA [8]. Assim um nó determina que operações realizar sobre os dados recebidos, mediante o seu contexto local, eliminado dependência de uma nuvem global. A definição de operadores e respetivas funções, necessitam ser encapsulados em imagens docker, para que possam ser carregados e posteriormente executados nos nós.

A infraestrutura revela alguma complexidade e a curva de aprendizagem necessária para configurar e programar toda a topologia é elevada, apesar de oferecer uma consola web para gestão e configuração.

Apesar de permitir a orquestração dos componentes de uma aplicação, esta arquitetura não é indicada para utilização no problema que este projeto pretende resolver, pois implica a utilização de contentores nos nós Edge, o que poderá não ser sempre possível em dispositivos de baixa potência.

Resumindo, neste capítulo foram introduzidas algumas plataformas e *frameworks*, orientadas a dar suporte ao desenvolvimento de soluções inteligentes, nomeadamente soluções para desenvolvimentos de cidades inteligentes.

3

Implementação

Este capítulo descreve a implementação proposta para o projeto apresentado. Descreve a arquitetura desenvolvida, os modelos de dados criados, relativos aos dispositivos registados na plataforma e aos dados rececionados dos dispositivos registados. Também descreve quais os componentes utilizados e desenvolvidos para a plataforma, qual a interação entre estes componentes e por fim como a plataforma proposta é instalada e instanciada.

3.1 Arquitetura da solução

A arquitetura proposta para esta solução, tem como um dos requisitos ser uma plataforma descentralizada. Para tal, a plataforma desenvolvida é composta por múltiplos componentes, os quais executam de forma isolada entre si, utilizado a rede para interação entre componentes e entidades externas. Tal configuração permite por exemplo, executar cada componente em máquinas diferentes em caso de necessidade. Este tipo de configuração, permite a falha de componentes individuais sem afetar a plataforma integramente, podendo afetar a disponibilidade de serviço prestado de forma parcial, ou no pior caso na sua totalidade, dependendo do número de componentes em falha ou da criticidade dos componentes afetados.

Para garantir o isolamento dos componentes durante a execução, cada componente é executado num contentor de execução virtual. Isto exige a escolha de um motor de

execução que execute serviços ou aplicações contentorizadas. Foi escolhido usar a tecnologia Docker Engine como motor de execução, tendo como critérios ponderados na sua escolha, ser uma tecnologia em código aberto, tem extensa documentação disponível, é de relativa fácil utilização, e tem elevada aceitação de utilização e suporte, por parte da comunidade em geral.

A contentorização dos componentes permite melhor portabilidade, pois é possível executar as imagens dos contentores de cada componente, em qualquer máquina física ou virtual, independentemente do sistema operativo da máquina, desde que a mesma tenha o motor de execução de contentores instalado, neste caso o Docker Engine instalado.

A contentorização também oferece melhor eficiência, pois permite o isolamento de componentes, sem ser necessária uma máquina individual por componente, como ilustra a Figura 3.1. É possível executar várias aplicações isoladamente em contentores, partilhando a mesma máquina hospedeira e respetivo sistema operativo.

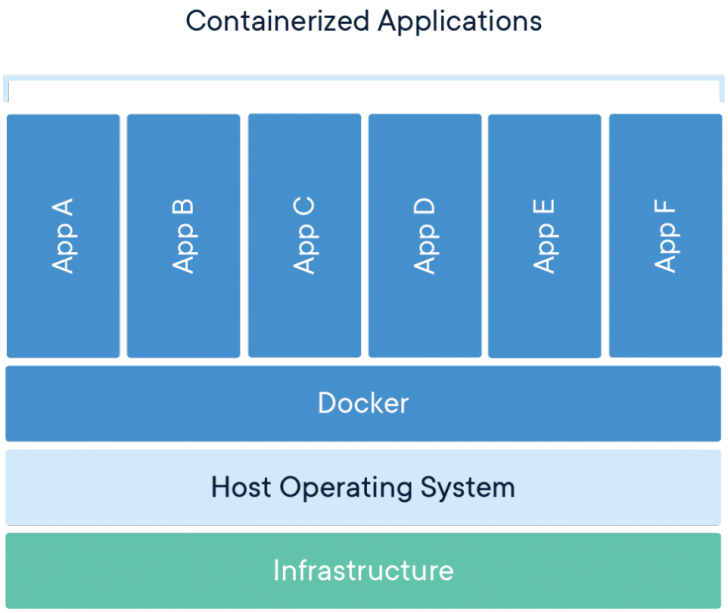

Figura 3.1: Aplicações em execução isoladas em contentores[a]

[a]Retirado de: https://www.docker.com/resources/what-container

Dada a natureza da plataforma proposta e os objetivos da mesma se enquadrarem no âmbito do desenvolvimento de software para cidades inteligente, foi tomada a decisão de implementação de utilizar a *framework* FIWARE apresentada no capítulo 2 deste

documento, no auxílio ao desenvolvimento da plataforma apresentada, através da utilização de alguns dos componentes que são disponibilizados no catálogo da *framework* FIWARE.

A utilização da *framework* FIWARE, permite agilizar a implementação da solução apresentada, beneficiando da utilização de software em código aberto disponibilizado pela comunidade, e assim direcionar boa parte do esforço de desenvolvimento na implementação de componentes de funcionalidade mais específica para a plataforma em desenvolvimento e que não foi possível encontrar em bibliotecas de código aberto.

A plataforma que apresentamos servirá, como já indicado anteriormente, para permitir aos utilizadores registarem dispositivos IoT sensorizados, numa rede de baixo potência, e observar informação enviada por estes.

A rede de comunicação de baixa potência sobre a qual os dispositivos IoT comunicam é a LoRaWAN, pois é um protocolo de comunicação que garante comunicações móveis de grande alcance, com consumos de muito baixa potência, que apresentam bons resultados na comunicação com dispositivos IoT em ambientes urbanos[10].Estas característica tornam este protocolo apropriado no uso dos dispositivos com os requisitos acima indicados. Assim é necessária uma infraestrutura que permita à plataforma desenvolvida, interagir com os dispositivos presentes numa rede LoRaWAN. Foi escolhido utilizar a plataforma TTN para esse efeito, a qual foi introduzida no capítulo 2 deste documento.

A TTN oferece um acesso comunitário sem custos, a um servidor aplicacional, onde é possível via API, registar dispositivos na rede LoRaWAN, e também possibilita a integração via protocolo MQTT, no acesso a notificações com mensagens transmitidas pelos dispositivos registados.

A Figura 3.2 ilustra a arquitetura proposta. Os utilizadores da plataforma acedem à mesma via browser. Os diversos componentes são executados em contentores de execução virtual. A interação com os dispositivos IoT é realizada através da TTN, acedida via Internet.

3.1.1 Componentes da solução

Como já referido, a solução implementada utiliza vários componentes do catálogo[6] disponibilizado pela *framework* FIWARE, também utiliza componentes SGBD (Sistemas de Gestão de Bases de Dados) para o armazenamento de dados, bem como componentes implementados à medida para esta solução.

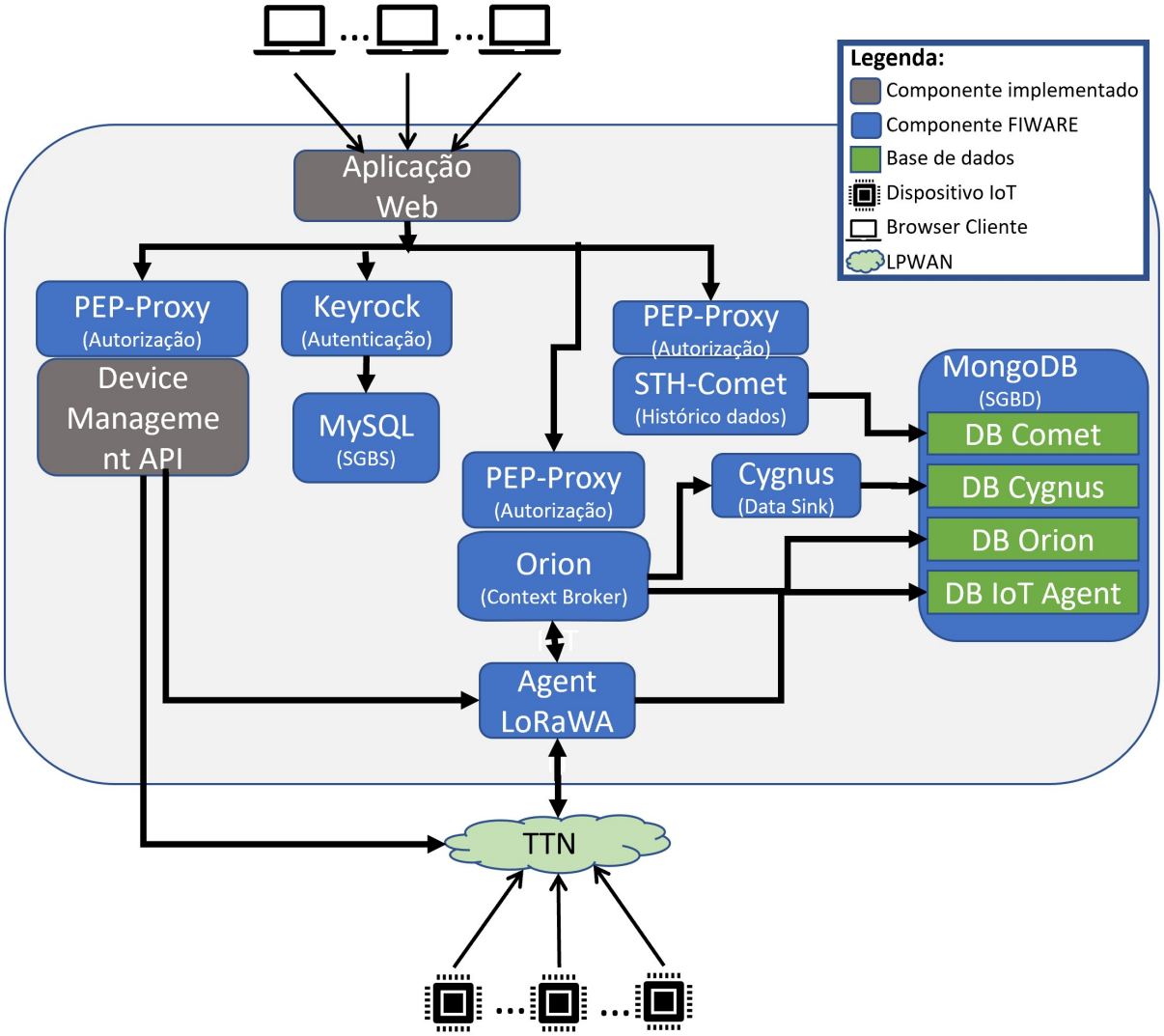

Figura 3.2: Arquitetura proposta para a plataforma.

Segue uma descrição geral dos componentes utilizados na solução apresentada e quais as principais funcionalidades utilizadas em cada um deles.

Orion (Context Broker) [1] Este é o componente central da solução apresentada, e é responsável pela gestão dos dados de contexto, nomeadamente os dados que caraterizam os dispositivos IoT registados em cada momento na solução, bem como são registados os dados mais atuais das métricas enviadas por estes dispositivos. Disponibiliza funcionalidades que permitem, interrogar os dados de contexto, atualizar os dados de contexto, registar subscritores acerca de alterações nos dados de contexto e notificar subscritores da existência de alterações nos dados de contexto. Para garantir a persistência dos seus dados, necessita de estar ligado a uma instância do SGBD MongoDB,

[1] Disponível em: https://fiware-orion.readthedocs.io/en/master/

no qual são criadas duas coleções, uma para os dados dos dispositivos e suas métricas, e outra coleção para registar os subscritores de alterações de contexto. Estas coleções são criadas, à medida que são rececionados os respetivos primeiros dados. Este componente faz parte do catálogo da *framework* FIWARE e a sua existência é obrigatória para a aplicação possa ser considerada uma aplicação "Powered by FIWARE", pois em geral os restantes componentes pertencentes à *framework* FIWARE, são projetados para executar em conjunto com uma instância do Orion.

IoT Agent LoRaWAN[2] Este componente é responsável pela comunicação com exterior na obtenção de dados oriundos de dispositivos IoT numa rede LoRaWAN [9] e registar esses dados no Context Broker Orion. Para rececionar as mensagens dos dispositivos, este componente comunica via protocolo MQTT com um servidor aplicacional de uma rede LoRaWAN, no qual se regista como subscritor em tópicos de mensagens. Os tópicos aos quais se regista, são tópicos associados a filas de mensagens, nas quais são registadas as mensagens enviadas pelos dispositivos IoT existente na rede LoRaWAN usada. No servidor aplicacional existe um tópico por dispositivo existente na rede LoRaWAN. Deste modo o servidor aplicacional notifica via MQTT, todos os subscritores da existência de novas mensagens por parte do dispositivo IoT em causa. Ao receber novas mensagens via notificação MQTT por parte do servidor aplicacional, com os dados enviados por um dispositivo, o IoT Agent converte automaticamente esses dados num formato compatível com interface programática NGSIv2, e regista essa informação no Context Broker Orion. Para que este componente se registe como subscritor num servidor aplicacional de uma rede LoRaWAN e assim possa ser notificado com as mensagens enviadas por um dado dispositivo IoT, necessita receber um pedido NGSIv2 via a sua interface REST, com informação de identidade do dispositivo na rede LoRaWAN, qual o endpoint do servidor aplicacional acessível via MQTT ao qual se deve registar e quais as credenciais de acesso ao mesmo. Para persistir o seu estado interno, necessita de uma ligação a uma instância do SGBD MongoDB. Disponibiliza uma interface NGSIv2. Também faz parte do catálogo da *framework* FIWARE.

Cygnus[3] Este componente é responsável pela persistência do histórico dos dados de contexto. Tem a função de coletor de dados (Data Sink), registando todas as alterações de contexto ocorridas no Orion, além de também realizar agregação dos dados persistidos. A agregação é realizada com base numa série temporal, nomeadamente são agregados em períodos com a resolução ao mês, dia, hora, minuto ou segundo. As

[2]Disponível em: https://fiware-lorawan.readthedocs.io/en/latest/
[3]Disponível em: https://fiware-cygnus.readthedocs.io/en/latest/

agregações realizadas para estes períodos são os valores mínimos, máximos, somatórios, somatórios elevados à potência de dois e o número de ocorrências para dados não quantitativos. Para que registe os dados de contexto, necessita de ser registado como subscritor das alterações de contexto no Orion. Este registo deverá ser feito logo após a instanciação deste componente. Permite a persistência dos seus dados em diferentes tipos de SGBD, mas dada a existência de uma instância do MongoDB por exigência de outros componentes, foi aproveitado para também persistir a informação deste componente em MongoDB e assim evitar a utilização de mais um outro componente na plataforma, só para este fim. Uma alteração relevante realizada à sua configuração por omissão, foi alterar o comportamento de criar duas coleções de dados no SGBD por cada dispositivo registado, nomeadamente uma coleção para registo dos dados históricos e outra coleção para o registo dos dados agregados. Foi observado durante os testes realizados, que essa configuração por omissão pode levar à falha da instância MongoDB usada, pois caso existam registos de dados de múltiplos dispositivos em simultâneo, o mongoDB usa um descritor de ficheiros no acesso a cada coleção e caso o número de descritores de ficheiros em uso exceder o limite (ulimits) permitido por processo no contentor onde SGBD é executado, existe uma exceção e o processo onde o MongoDB corre é terminado. Para evitar esta possível falha, foram alterados os parâmetros de configuração do Cygnus, de forma que existam apenas duas coleções para o conjunto de todos dos dispositivos registados. Uma coleção para registar os dados históricos de todos os dispositivos e outra coleção para registar os dados agregados de todos os dispositivos também. Também faz parte do catálogo de componentes da *framework* FIWARE.

STH-Comet[4] Este componente permite a consulta do histórico de contexto e respetivos dados agregados, que foram persistidos pelo componente Cygnus. Acede à mesma instância MongoDb usada pelo Cygnus, no resgate da informação persistida e dessa forma responder às interrogações recebidas.
Disponibiliza uma interface NGSIv2, através da qual permite realização de interrogações aos dados históricos registados e também aos dados agregados, com base nas séries temporais com que este foram persistidos. Pode ser parametrizado para executar em um de dois modos distintos. No modo minimalista, fica responsável pela persistência do histórico de dados de contexto do Orion e respetivas agregações, evitando o uso do Componente Cygnus, e também de responder a pedidos de consulta dados persistidos. No modo formal, é delegada a coleta e persistência de dados ao Cygnus e

[4]Disponível em: https://fiware-sth-comet.readthedocs.io/en/latest/

apenas realiza a funcionalidade de responder a pedidos de consulta de dados. Foi escolhido executar este componente no modo formal, pois na sua documentação indica que apesar do modo formal ter a uma parametrização entre os componentes intervenientes mais complexa, permite melhor rendimento que o modo minimalista. Também faz parte do catálogo de componentes da *framework* FIWARE.

MongoDB[5] Este componente é um SGBD NoSQL[6]. Faz parte da dependência direta de alguns componentes usados na solução, nomeadamente do Orion, do IoTAgent LoRaWAN, do Cygnus, do STH-Comet e do *Device Management API*. Permite a estes componentes persistirem o seu estado interno e assim suportar a recuperação dos componentes e seu estado interno em caso de falha e/ou reinício do contentor onde está em execução, ou seja, permite a estes componentes executar em contentores sem persistência de estado associada ao contentor, recorrendo ao MondoDB para persistir informação de forma estruturada.

KeyRock[7] Componente responsável por autenticar e autorizar os acessos à solução apresentada. Permite o registo de utilizadores e de papéis para grupos de utilizadores, aos quais é possível definir autorizações de acesso a determinados recursos. Estes recursos são identificados por URLs, sendo possível definir expressões regulares para abranger conjuntos de recursos a uma dada autorização. Só suporta definições de autorização simples baseadas em URLs, pois para definição de regras de autorização mais complexas, seria necessário introduzir mais um componente da *framework* FIWARE, chamado AuthzForce. O protocolo de segurança utilizado por este componente na transmissão de dados de autenticação e autorização com os restantes componentes, é baseado no protocolo OAuth2[8]. Tem como dependência uma instância do SGBD MySQL, necessária para persistência do seu estado interno.
Também faz parte do catálogo de componentes da *framework* FIWARE.

MySQL[9] SGBD relacional necessário, pois é dependência direta do componente Keyrock, para a persistência dos seus dados.

[5]Disponível em: https://www.mongodb.com/
[6]https://www.mongodb.com/nosql-explained
[7]Disponível em: https://fiware-idm.readthedocs.io/en/latest/
[8]Disponível em: https://oauth.net/2/
[9]Disponível em: https://www.mysql.com/
[10]Disponível em: https://fiware-pep-proxy.readthedocs.io/en/latest/

PEP-Proxy[10] Este componente é um componente de segurança. É um proxy de autorização no acesso por parte de entidades externas a alguns componentes da solução, daí existir o termo PEP (Policy Enforcement Point) no nome do componente. Oferece um acesso transparente aos componentes que protege, mas força que os pedidos realizados a estes estejam autenticados, ou seja, quem tentar aceder ao componente protegido, acede da mesma forma como se acedesse diretamente ao componente, mas necessita de fornecer um token de autenticação gerado pelo componente Keyrock (Gestor de identidade e autorizações). O PEP-Proxy valida o token apresentado com o Keyrock e verifica também se a entidade em causa, pode aceder ao recurso que protege. Para que esteja autorizado a validar tokens e verificar acessos a recursos com o Keyrock, também ele necessita de um token de autenticação, específico para PEP-Proxy's. Este token é necessário ser fornecido como parâmetro no momento de instanciação deste componente. Esta dependência de um token vindo do Keyrock, força que as instâncias do PEP-Proxy só possam ser instanciadas após a instância do Keyrock esteja em execução, e ter sido gerado e disponibilizado o token necessário ao PEP-Proxy. Nesta solução existem três instâncias deste componente, uma instância para proteger os acessos ao componente Orion, outra instância para proteger os acessos ao componente *Device Management API* e finalmente uma instância para proteger os acessos ao componente STH-Comet. Também faz parte do catálogo de componentes da *framework* FIWARE.

Device Management API Este componente tem a responsabilidade de concretizar na plataforma, os pedidos para aprovisionamento de novos dispositivos, os pedidos de remoção de dispositivos, os pedidos de interrogação dos dispositivos registados, os pedidos para definição de novos tipos de dispositivos e os pedidos de definição de novas métricas que possam ser utilizadas no aprovisionamento dispositivos. Para aprovisionamento de dispositivos, é necessário que este componente interaja com o IoTAgent LoRaWAN e a TTN, notificando ambos com a identificação do dispositivo a registar e respetivos parâmetros necessários. Necessita de acesso a uma instância do SGBD MongoDB, para persistir os dados das novas métricas e novos tipos dispositivos criados. É um componente desenvolvido especificamente para a solução apresentada.

Aplicação Web Componente que realiza a interface entre o utilizador e a plataforma. Disponibiliza funcionalidades que permitem ao utilizador aprovisionar e remover dispositivos, visualizar um *dashboard* com um mapa geográfico no qual é apresentada a localização dos dispositivos que já enviaram dados com informação de localização, bem como permitir analisar o estado do dispositivo e o histórico de métricas enviadas por este. É um componente desenvolvido especificamente para a solução apresentada.

3.2 Modelo de Dados

Os dispositivos de monitorização da qualidade do ar, produzem informação sobre as observações realizadas. Essa informação necessita ser armazenada e lida. Esses dados dos diferentes dispositivos, irão fazer parte dos dados de contexto e histórico da aplicação.

Para representar os dados das entidades de contexto de forma uniforme e compatível com todos os componentes da aplicação, especialmente com os componentes FIWARE usados, é necessário definir um modelo de dados comum e em conformidade com a especificação NGSI.

O modelo de dados definido segue a especificação definida pela Smart Data Model [20], a qual permite conformidade com o NGSIv2 [13]. A NGSIv2 é uma evolução da NGSI, que será descontinuada no futuro.

As entidades cujos dados é necessário modelar são:

- Dispositivos IoT de monitorização da qualidade do ar;
- Observações realizadas por dispositivos IoT de monitorização da qualidade do ar;

3.2.1 Dispositivos IoT

O modelo de dados associado aos dispositivos IoT de monitorização da qualidade do ar tem os seguintes atributos:

- **id:** Identificador único do dispositivo dentro da aplicação (obrigatório);
- **devEUI:** Identificador único do dispositivo dentro no meio de comunicação usado pelo dispositivo (opcional);
- **fiware-service:** Classificador que segrega informação existente entre diferentes classificadores (obrigatório);
- **fiware-path:** Sub-Classificador que segrega informação existente entre diferentes sub-classificadores, no contexto de um dado classificador(obrigatório);
- **entity_type:** Identifica o tipo de dispositivo (obrigatório);

- **data_model:** Identifica o formato dos dados transmitidos pelo meio de comunicação usado pelo dispositivo (obrigatório);
- **dateCreated:** Data de criação do dispositivo (obrigatório);
- **dateModified:** Data da última modificação do dispositivo (obrigatório);

O valor dos atributos **fiware-service**, **fiware-path**, **entity_type** e **data_model** têm que ser definidos no momento da instanciação da plataforma, pois são valores comuns entre todos os dispositivos. O valor dos atributos **dataCreated** e **dateModified** apesar de obrigatórios, são gerados automaticamente pela plataforma no momento do registo dos dispositivos aprovisionados na plataforma. O valor dos atributos **id** e **devEUI** são definidos no momento do registo de cada dispositivo na plataforma. O atributo **devEUI** é opcional, pois caso não seja indicado no momento do registo do dispositivo, este é gerado automaticamente pela plataforma.

3.2.2 Observações qualidade do ar

Para representar os dados obtidos dos dispositivos IoT de monitorização da qualidade do ar, foi definido o seguinte modelo de dados, constituído pelos seguintes atributos:

- **id:** Identificador a observação realizada (obrigatório);
- **mandatory:** Atributos gerados pelos dispositivos e obrigatórios;
  - **location:** Localização da observação registada.
- **optional:** Atributos geradados pelos dispositivos e opcionais;
  - **pm2.5:** Quantidade de partículas com dimensão inferior a 2.5 μm, em partes por milhão;
  - **pm10:** Quantidade de partículas com dimensão inferior a 10 μm, em partes por milhão;
- **dateCreated:** Data de criação da observação na plataforma (obrigatório);
- **dateModified:** Data da última modificação da observação na plataforma (obrigatório);

O atributo **mandatory** identifica e lista as métricas comuns, geradas por todos os dispositivos e obrigatórias. O atributo **optional** identifica e lista as métricas que alguns dispositivos podem gerar e são opcionais. Caso um dispositivo suporte uma métrica opcional, tal informação têm que ser indicada no aprovisionamento do dispositivo. Este modelo de dados é extensível, pois novas métricas **optional**, podem ser definidas em tempo de execução, através do componente Device Managemente API.

3.3 Lógica de Implementação

Os componentes da plataforma proposta são disponibilizados em imagens de contentores de execução virtual. Isto permite entrega e execução em ambientes virtuais, nomeadamente na nuvem. Esta abordagem e o uso de imagens de contentores disponibilizados por outras entidades que implementem funcionalidades pretendidas e úteis na resolução de problemas comuns, permite a reutilização de componentes já desenvolvidos e testados pela comunidade, minimizando esforço no desenvolvimento de aplicações.

Os componentes de terceiros usados estão todos disponíveis em código aberto no GitHub[11] e as imagens dos seus contentores de execução virtual também estão disponíveis publicamente no repositório DockerHub[12] através da internet, nomeadamente os componentes da *framework* FIWARE e os SGBD MongoDB e MySQL.

O código dos componentes implementados especificamente para esta plataforma, também são disponibilizados à comunidade para reutilização e possível melhoramento. Foram criadas imagens públicas dos contentores destes componentes, através da aplicação Docker, e disponibilizadas em repositório público no DockerHub também, para download e instanciação.

3.3.1 *Device Management API*

Para permitir o aprovisionamento, remoção e listagem dos dispositivos IoT na solução apresentada, foi implementado o componente *Device Management API*. A implementação deste componente foi realizada na linguagem Javascript e executada sobre um serviço Node.js[13], disponibilizando as suas funcionalidades através de uma interface programática RESTful acessível via HTTP.

[11] https://github.com/
[12] Disponível em:https://hub.docker.com/
[13] Disponível em:https://nodejs.org/en

A função principal deste componente é, registar e atualizar de forma consistente e em simultâneo os dados dos dispositivos aprovisionados, tanto na plataforma TTN, como no componente IoT Agent LoRaWAN, para que esta duas entidades coordenem entre si, a receção de mensagens oriundas dos dispositivos.

A TTN tem a capacidade de rececionar mensagens numa rede LoRa através do seu servidor de rede LoRaWAN e disponibilizar as mesmas, a subscritores no seu servidor aplicacional via protocolo MQTT.

O componente IoT Agent LoRaWAN, consegue ser subscritor de mensagens via protocolo MQTT no servidor aplicacional da TTN, e ao ser notificado de uma nova mensagem enviada por um dispositivo, converte a mensagem conforme enviada na TTN, num formato compatível com a interface NGSIv2 e entrega a mensagens recebida ao Context Broker Orion, alimentado assim a plataforma apresentada com os dados dos dispositivos aprovisionados.

Sem este componente implementado, o processo de aprovisionamento de um dispositivo seria menos automatizado, pois seria necessário aceder manualmente via browser à interface Web da TTN, navegar através de uma série de menus da interface, preencher no formulário de submissão os vários parâmetros relativos ao dispositivo a ser aprovisionado e submeter o formulário.

Do lado do IoT Agent LoRaWAN, também seria necessário realizar um pedido HTTP POST para a interface RESTful do componente, por exemplo via aplicação Curl, ou aplicação Postman, ou outra aplicação similar, enviado os dados do dispositivo a aprovisionar, no formato compatível à interface NGSIv2 do componente e seguindo o modelo de dados definido para o efeito.

A Figura 3.3 ilustra o processo de aprovisionamento coordenado por este componente implementado e as interações existentes durante o processo de aprovisionamento de um dispositivo. O processo é despoletado por um pedido de registo de um dispositivo e rececionado por este componente.

Segue uma descrição da iteração necessária entre os componentes da plataforma e a TTN, no processo de aprovisionamento de um dispositivo na plataforma apresentada.

1. Um pedido de registo de dispositivo chega ao componente *Device Management API* via HTTP, contendo os dados necessários para registar o dispositivo IoT a aprovisionar.

2. O *Device Management API*, faz um pedido à TTN via HTTP à API de gestão que a TTN disponibiliza, usando tokens de acesso que permitem validar a autorização

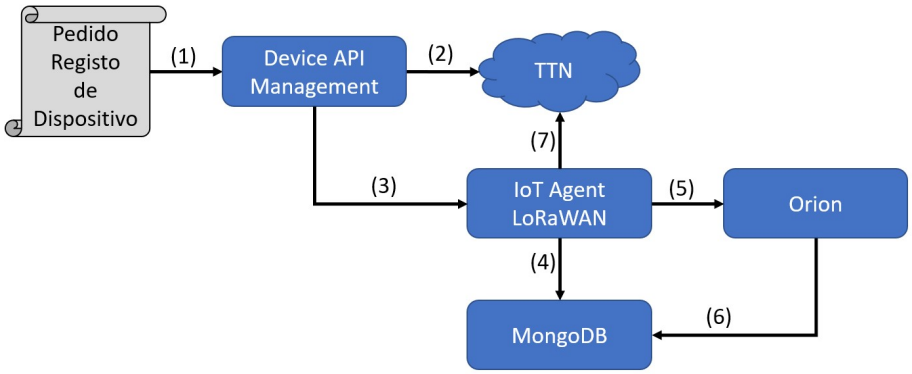

Figura 3.3: Processo de registo de um dispositivo na plataforma.

para realizar esta operação e assim regista o dispositivo na mesma. Este pedido contém os dados do dispositivo e as chaves que permitem a TTN contactar com dispositivo na rede LoRaWAN a que ambos pertencem. Estas chaves são chaves de negociação, necessárias para acordar chaves únicas de comunicação entre o dispositivo e a TTN, que permitem a comunicação segura sobre uma rede LoRaWAN.

3. O *Device Management API*, faz um pedido via HTTP à interface NGSIv2 do IoT Agent LoRaWAN, com os dados do dispositivo e dados que identificam um endpoint e tokens de acesso, de um serviço MQTT Broker, ao qual se deve subscrever via protocolo MQTT. Tal subscrição irá permitir que possa receber as mensagens que irão ser transmitidas pelo dispositivo em causa.
   Nos dados do dispositivo, também são indicados quais as métricas que o dispositivo poderá transmitir nas suas mensagens.

4. O IoT Agent LoRaWAN atualiza a coleção que contém na instância MongoDB, com os dados do novo dispositivo aprovisionado. Isto permite que o componente recupere o seu estado em caso de reinício do componente, ou recuperação após falha. Este componente necessita de manter estado acerca dos dispositivos registados, pois este componente filtra as mensagens futuras dos dispositivos, transmitindo para plataforma apresentada, apenas os atributos indicados no processo de aprovisionamento do dispositivo.

5. O IoT Agent notifica o Orion, da existência de um novo dispositivo aprovisionado na plataforma.
   Esta notificação destina-se a atualizar os dados de contexto da plataforma, sendo possível ao Orion responder a interrogações acerca dos dispositivos registados na plataforma.

6. De igual forma o Orion garante a persistência dos dados relativos ao dispositivo na coleção existente para efeito no MongoDB. Igualmente permite reinício do componente e recuperação após falha, retornando ao estado corrente.

7. Ao receber um registo de um dispositivo, o IoT Agent LoRaWAN regista-se no endpoint transmitido no ponto anterior, neste caso regista-se no servidor aplicacional da TTN como subscritor de todas as mensagens provenientes do dispositivo em causa. A TTN disponibiliza no seu servidor aplicacional um serviço MQTT Broker, no qual o IoT Agent se inscreve via protocolo MQTT, para receber novas mensagem transmitidas pelo dispositivo em causa. O serviço MQTT Broker organiza as mensagens dos dispositivos por tópicos, que na prática são filas de mensagens, onde são registadas as mensagens associadas ao dispositivo a que pertence tópico. As mensagens ao chegarem à uma fila de mensagens são encaminhadas para os subscritores do tópico em questão.

A remoção de dispositivos processa-se de forma similar ao processo de registo de um dispositivo, mas neste caso apenas é transmitida a identificação do dispositivo a remover. O pedido de remoção chega ao componente *Device Management API* via pedido HTTP. Este componente faz um pedido tanto à TTN como ao IoT Agent LoRaWAN para remoção do dispositivo dos seus registos. Também notifica o Orion, que dado dispositivo, não se encontra aprovisionado na plataforma. Por fim o IoT Agent LoRAWAN faz pedido à TTN para remover a sua subscrição existente, no tópico associado à de fila de mensagens do dispositivo em causa.

3.3.2 Registo de observações dos dispositivos

Quando um dispositivo IoT envia uma mensagem na rede LoRaWAN à qual a TTN faz parte e o dispositivo já se encontra aprovisionado na plataforma apresentada, a TTN notifica todos os subscritores registados no tópico de mensagens associado ao dispositivo em causa, da existência de uma nova mensagem. O componente IoT Agent LoRaWAN ao ser notificado pela TTN com nova mensagem, converte a mensagem num formato compatível com a interface NGSI e regista os dados relevantes da mensagem transmitida, no Orion (Context Broker), assim atualizando a informação de contexto da plataforma.

A Figura 3.4 ilustra o processo de receção de mensagens oriundas da TTN. Segue uma descrição mais detalhada do processo.

1. Quando a TTN tem nova mensagem com origem em um dispositivo ao qual o IoT Agent LoRaWAN se registou como subscritor no seu serviço de MQTT Broker,

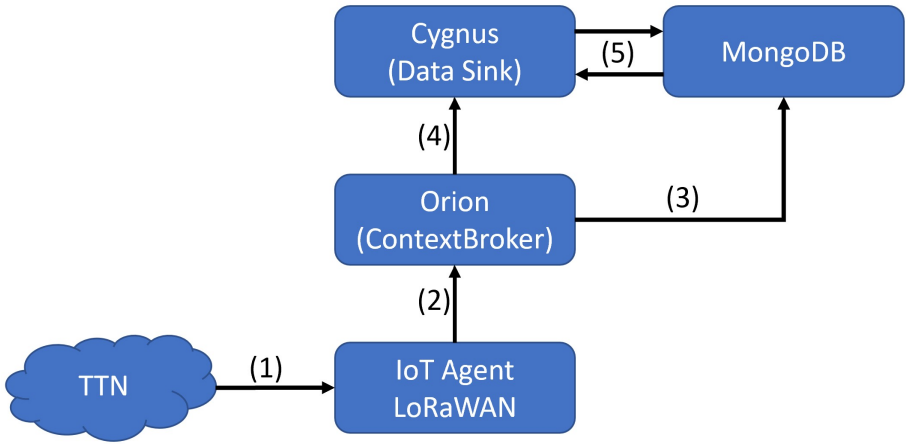

Figura 3.4: Receção de nova mensagem de um dispositivo

envia mensagem via protocolo MQTT ao IoT Agent LoRaWAN com os dados transmitidos pelo dispositivo em causa.

2. O IoT Agent LoRaWAN, ao ser notificado com os dados transmitidos por um dado dispositivo, converte os dados das mensagens do dispositivo num formato compatível com a interface NGSIv2, para que seja possível enviar essa informação ao gestor de contexto da plataforma, ou seja o Orion.
   Antes de enviar os dados, o IoT Agent filtra os atributos enviados na mensagem do dispositivo, enviado apenas ao Orion, os atributos indicados no momento em que o dispositivo foi registado na plataforma.

3. A mensagem ao ser rececionada pelo Orion, significa que a informação de contexto da plataforma está a ser atualizada. O Orion para persistir esta informação, atualiza os dados na coleção de entidades de contexto que tem instanciada no componente MongoDB. Esta coleção de dados, não mantém histórico de versões dos valores de contexto, ou seja, se por exemplo na mensagem enviada pelo dispositivo, houver dados de uma métrica já observada e se já existir um valor em contexto para essa mesma métrica, o valor anterior é substituído pelo novo valor transmitido pelo dispositivo.

4. O Orion notifica todos os subscritores de alteração de contexto, que os dados de contexto sofreram alterações, enviado nas notificações os novos dados de contexto. Como na arquitetura da plataforma, o componente Cygnus é subscritor de alterações de contexto no componente Orion, neste cenário recebe notificação com os dados enviados pelo dispositivo.

5. O Cygnus persiste os dados recebidos do Orion, registando todas as alterações de contexto, recebidas via notificação. A persistência dos dados é realizada na

instância MongoDB existente na plataforma. Além de registar o histórico das alterações de valores de contexto, o Cygnus também realiza agregações a esses dados, com base em períodos temporais. Para isso, obtém o histórico armazenado até então no MongoDB, processar as operações de agregação sobre os dados já atualizados e persiste novamente os dados de agregação.

Para possibilitar o registo de dispositivos com novos atributos (métricas) em tempo de execução, ou seja, novos atributos até então desconhecidos da solução, foi implementado no *Device Management API* a funcionalidade que permite a definição de novos atributos para os dispositivos, via API que disponibiliza. Após definição de um novo atributo, este passa a estar disponível no aprovisionamento de novos dispositivos. A informação relativa às novas métricas definidas, são persistidas na instância MongoDB existente na plataforma.

Outra funcionalidade implementada, foi a possibilidade de criar tipos de dispositivos e quais os atributos que os definem. Esta funcionalidade é útil para a interface Web com o utilizador, permitindo a este escolher um tipo de dispositivo no momento do registo de um dispositivo, sem ter que identificar individualmente cada um dos atributos (métricas) que este dispositivo transmite, e sim apenas ter que escolher um tipo que associa ao dispositivo um conjunto de métricas predefinidas. A informação de tipos de dispositivos também é persistida em MongoDB.

A informação acerca das novas métricas e dos novos tipos de dispositivos, fica persistida numa coleção criada para o efeito, na instância MongoDB existente na arquitetura apresentada.

Por fim segue um resumo das funcionalidades implementadas por este componente e disponibilizadas na sua API:

- Registo de novos dispositivos IoT na solução.
- Remoção de dispositivos IoT na solução.
- Listagem dos dispositivos registados na IoT na solução.
- Definição de novos atributos (métricas) para dispositivos.
- Definição de novos tipos de dispositivos.

3.3.3 Aplicação Web

Para permitir a interação do utilizador com a solução apresentada, foi implementada uma aplicação Web para o efeito. Foi implementada na linguagem Javascript, com o recurso à biblioteca React[14] e executada sobre um serviço NodeJs. A interface Web, além de permitir ao utilizador visualizar informação relativa aos dispositivos e suas métricas transmitidas, permite também aprovisionar e remover dispositivos, definir novas métricas (atributos) que novos dispositivos possam transmitir e definir novos tipos de dispositivos.

Para que disponibilize informação e efetue os pedidos realizados pelo utilizador, este componente necessita de interagir com diversos outros componentes da plataforma, nomeadamente com o componente *Device Management API* para satisfazer os pedidos do utilizador, no aprovisionamento e remoção de dispositivos na plataforma, na definição de novas métricas, na parametrização de escalas de cor usadas na demonstração de resultados associados a métricas transmitidas pelos dispositivos, e na definição de novos tipos de dispositivos. Também interage com o componente Keyrock na autenticação de utilizadores, interage com o componente Orion na consulta dos dados correntes enviados pelos dispositivos e interage com o componente STH-Comet na obtenção de dados históricos enviados pelos dispositivos.

Para visualizar os dispositivos, a interface Web apresenta no browser do utilizador um mapa geográfico iterativo, que permite a navegação e ampliação, no qual são projetados através de marcadores, os dispositivos posicionados no mapa, na última localização registada e enviada em mensagem pelo dispositivo.

A Figura 3.5 ilustra um dispositivo representado no mapa através de um marcador, na última posição transmitida por este. A cor verde do marcador, indica que este dispositivo transmitiu mensagens últimas 24 horas, caso contrário o marcador seria representado a vermelho, sinalizado a existência de algum possível problema com o dispositivo. Ao pairar o rato sobre o marcador, é apresentado um texto sobre o mesmo, indicado a identificação do dispositivo em causa.

Ao clicar sobre o marcador de um dispositivo, é mostrada informação relativa ao dispositivo e métricas enviadas por este. É apresentada a data e hora da última comunicação do dispositivo, e quais os valores das métricas transmitidas na última comunicação. Também é apresentada a média diária para os últimos 30 dias das métricas transmitidas pelo dispositivo, sendo o valor representado por uma cor pertencente a uma escala de estado da métrica. Estas escalas de estado de cada métrica, podem ser

[14]https://reactjs.org/

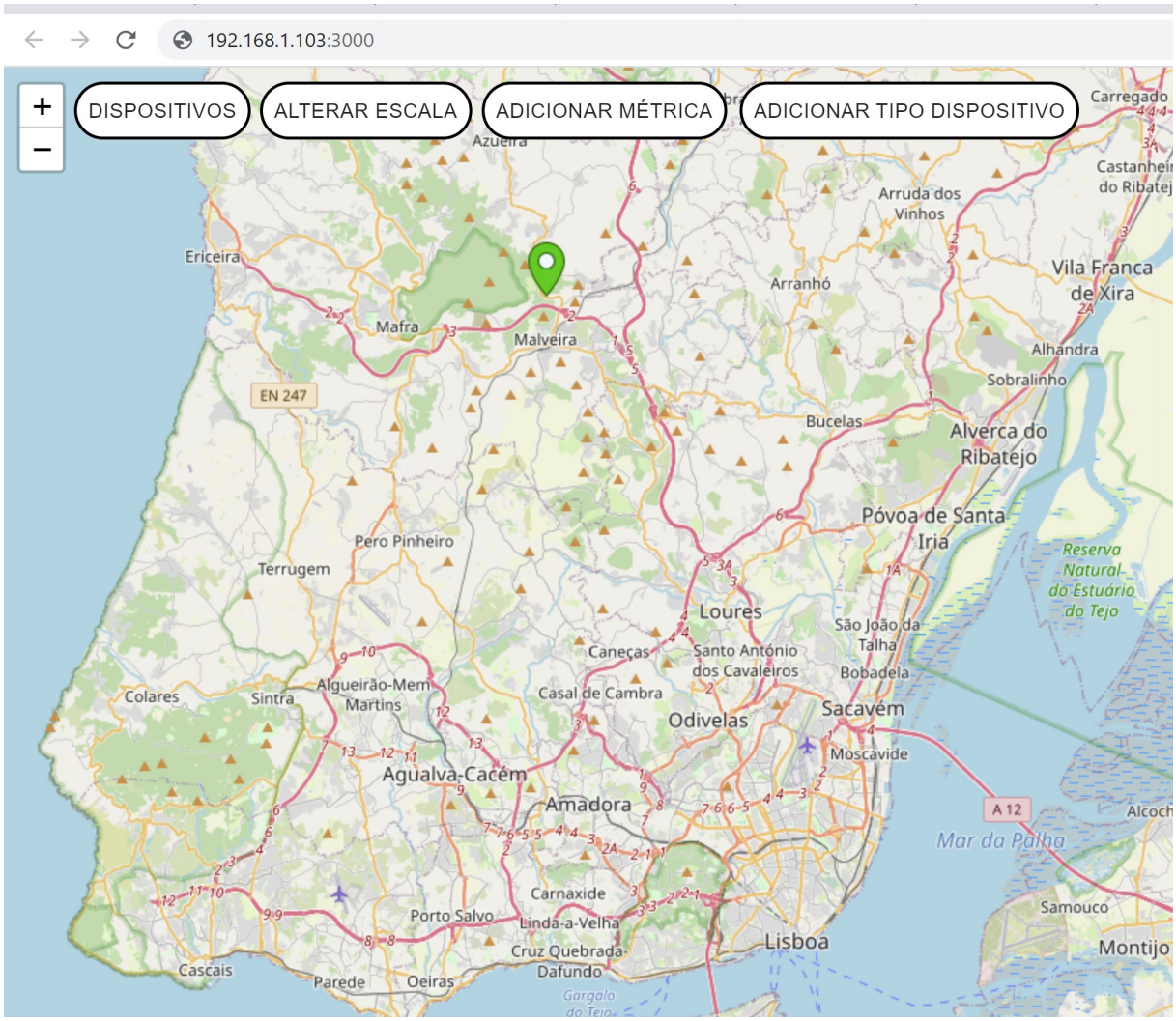

Figura 3.5: Mapa com marcador que representa um dispositivo numa dada localização, apresentado no browser do utilizador

parametrizadas individualmente. A métrica em análise pode ser alternada através de um tabulador superior, que permite mudar a métrica representada. A Figura 3.6 ilustra a informação apresentada relativa a um dispositivo, no qual é possível alternar a métrica em análise.

No topo da interface gráfica apresentada, estão disponíveis quatro botões. O botão Dispositivos, disponibiliza um submenu, com três opções. Uma opção para aprovisionar um dispositivo, uma opção para remover um dispositivo e outra opção para listar os dispositivos registados.

Para aprovisionar um dispositivo na plataforma, é apresentado um formulário no qual é necessário preencher com os dados do dispositivo a aprovisionar, nomeadamente o **DevId** do dispositivo, com o qual o utilizador identifica o dispositivo, o **DevEUI** que identifica o dispositivo numa rede LoRa, o **Tipo de Dispositivo** e os **Atributos**

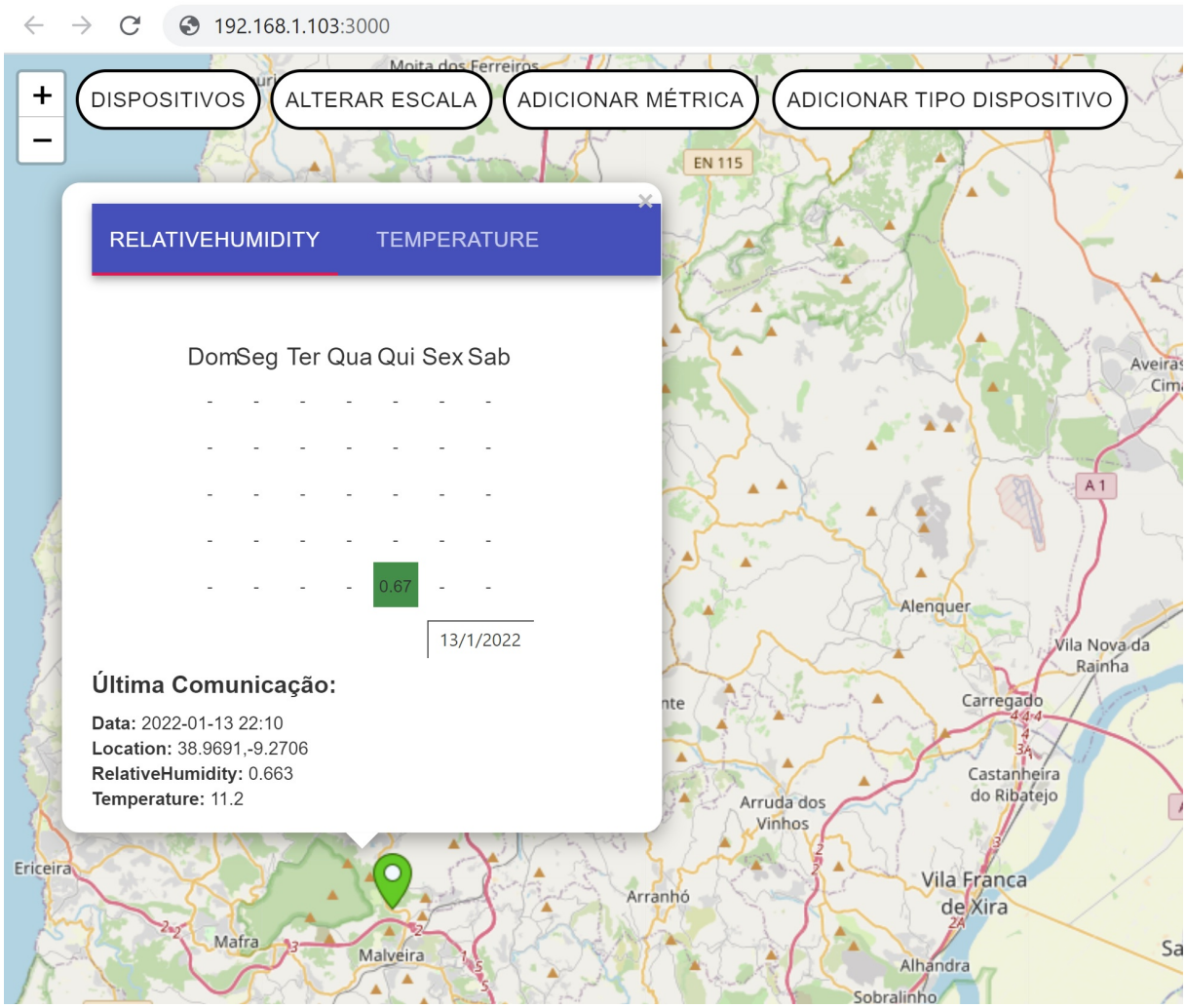

Figura 3.6: Mapa com informação relativa a um dispositivo e suas métricas.

Opcionais. Destes campos, apenas é de preenchimento obrigatório o **DevId**. O campo **DevEUI** é opcional, pois é gerado automaticamente na plataforma se o campo ficar em branco. Este campo só deve ficar em branco nos casos que o dispositivo em causa permita a parametrização deste valor e posterior programação no próprio dispositivo. O campo **Tipo Dispositivo** é opcional, pois a sua escolha, preenche o campo Atributos Opcionais com todos os atributos definidos para o tipo de dispositivo selecionado. O campo **Atributos Opcionais**, também pode ficar em branco, caso o dispositivo não transmita nenhum atributo, além dos atributos obrigatórios definidos no modelo de dados.

Para remoção de dispositivos, também é necessário preencher um formulário com o **DevId** do dispositivo a remover.

A Figura 3.7 e a Figura 3.8 ilustram os formulários que permitem o registo e remoção de dispositivos na solução.

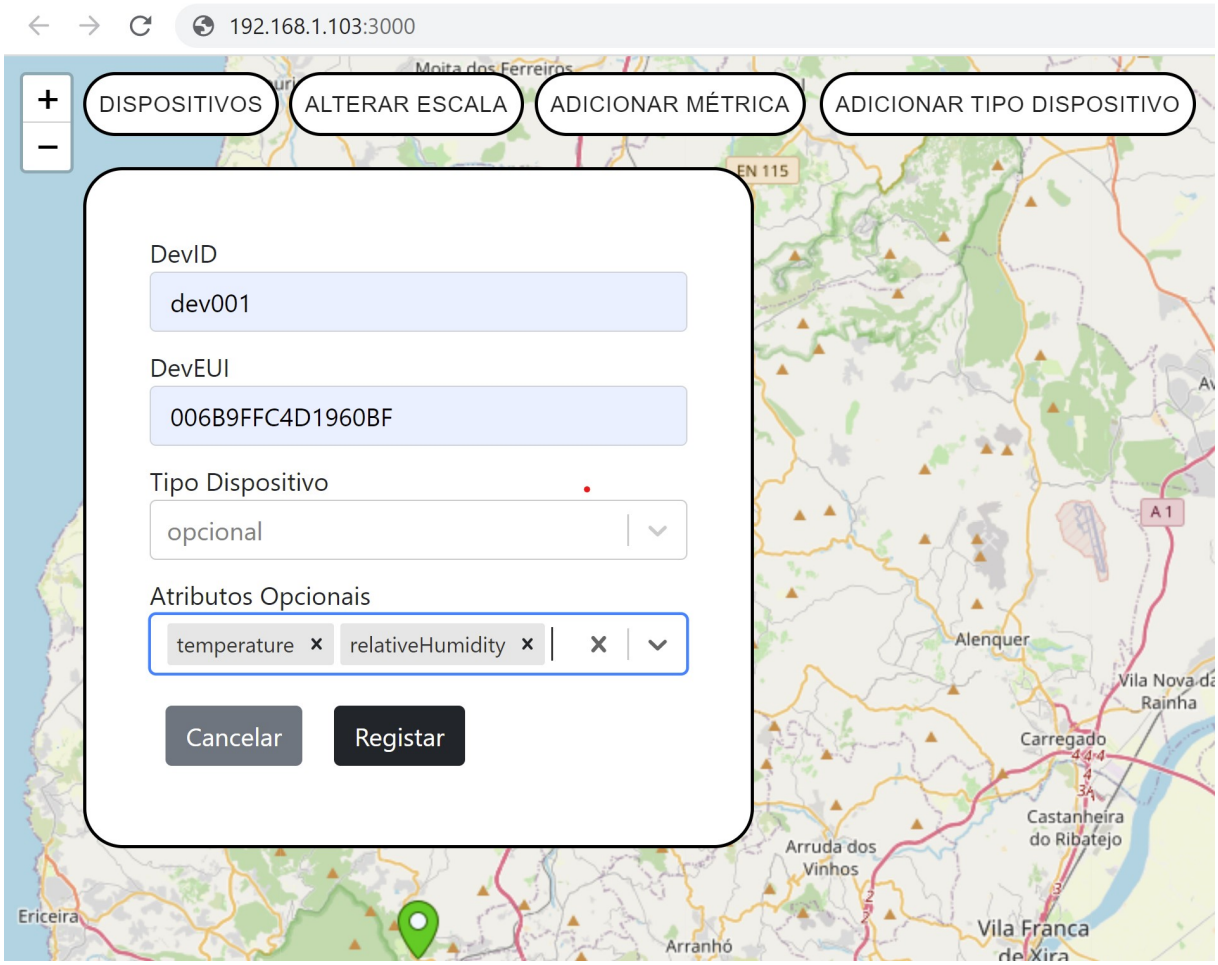

Figura 3.7: Formulário para o aprovisionamento de novo dispositivo.

As escalas de cor apresentadas para cada métricas podem ser parametrizadas pelo utilizador. Na interface gráfica é disponibilizado um botão "Alterar Escala"que abre um formulário de parametrização. Este formulário permite ao utilizador ajustar os valores limite dos intervalos e qual a cor representada para cada intervalo, com que os valores de uma dada métrica são representados.

A escolha da métrica a parametrizar é possível através de uma caixa de seleção, e cada cor para cada intervalo da escala pode ser alterada. Ao clicar na cor do intervalo a alterar, é mostrada uma palete de cores, para selecionar uma cor pretendida. A Figura 3.9 ilustra o formulário para parametrização das métricas.

A definição de novas métricas é disponibilizada através do botão "Adicionar Métrica", que adiciona à interface um formulário, que permite definir o nome da nova métrica e parametrizar os intervalos de cores para escala, que será usada na interface gráfica para representar os valores da nova métrica. Também é possível definir ou remover intervalos para a escala da nova métrica a ser criada. Ao submeter uma nova métrica,

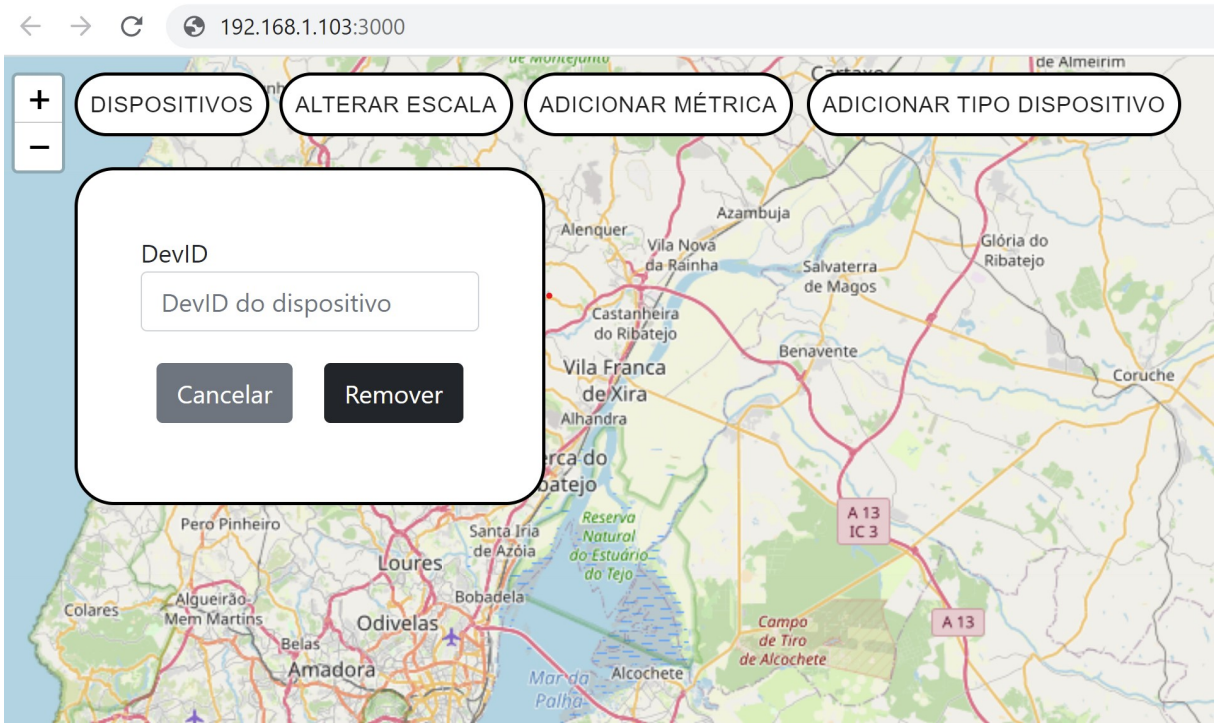

Figura 3.8: Formulário para remoção de dispositivo.

esta passa a estar disponível no aprovisionamento de novos dispositivos. A Figura 3.10 ilustra o formulário para definição de nova métrica.

Por fim existe um botão "Adicionar Tipo Dispositivo"que permite definir novos tipos de dispositivos. Este botão disponibiliza um formulário onde se define o nome para o novo tipo de dispositivo e quais os atributos (métricas) existente que caracterizam este tipo de dispositivo. Permite selecionar um subconjunto das métricas já existente na plataforma.

Ao submeter um novo tipo de dispositivo, este passa a estar disponível no formulário de aprovisionamento de dispositivos. Caso seja selecionado nesse formulário, associa automaticamente as métricas deste tipo de dispositivo ao novo dispositivo que se pretende aprovisionar, evitando selecionar métrica a métrica no aprovisionamento de dispositivos comuns. A Figura 3.11 ilustra o formulário para definição de novos tipos de dispositivos na plataforma.

3.3.4 Adaptação da plataforma à versão 3 da TTN

Em meados de 2021, a TTN lançou uma nova versão da sua plataforma, chamada de *The Things Stack V3* e anunciou que a versão corrente seria descontinuada até ao fim de 2021. Apesar de a nova versão manter as funcionalidades oferecidas até então,

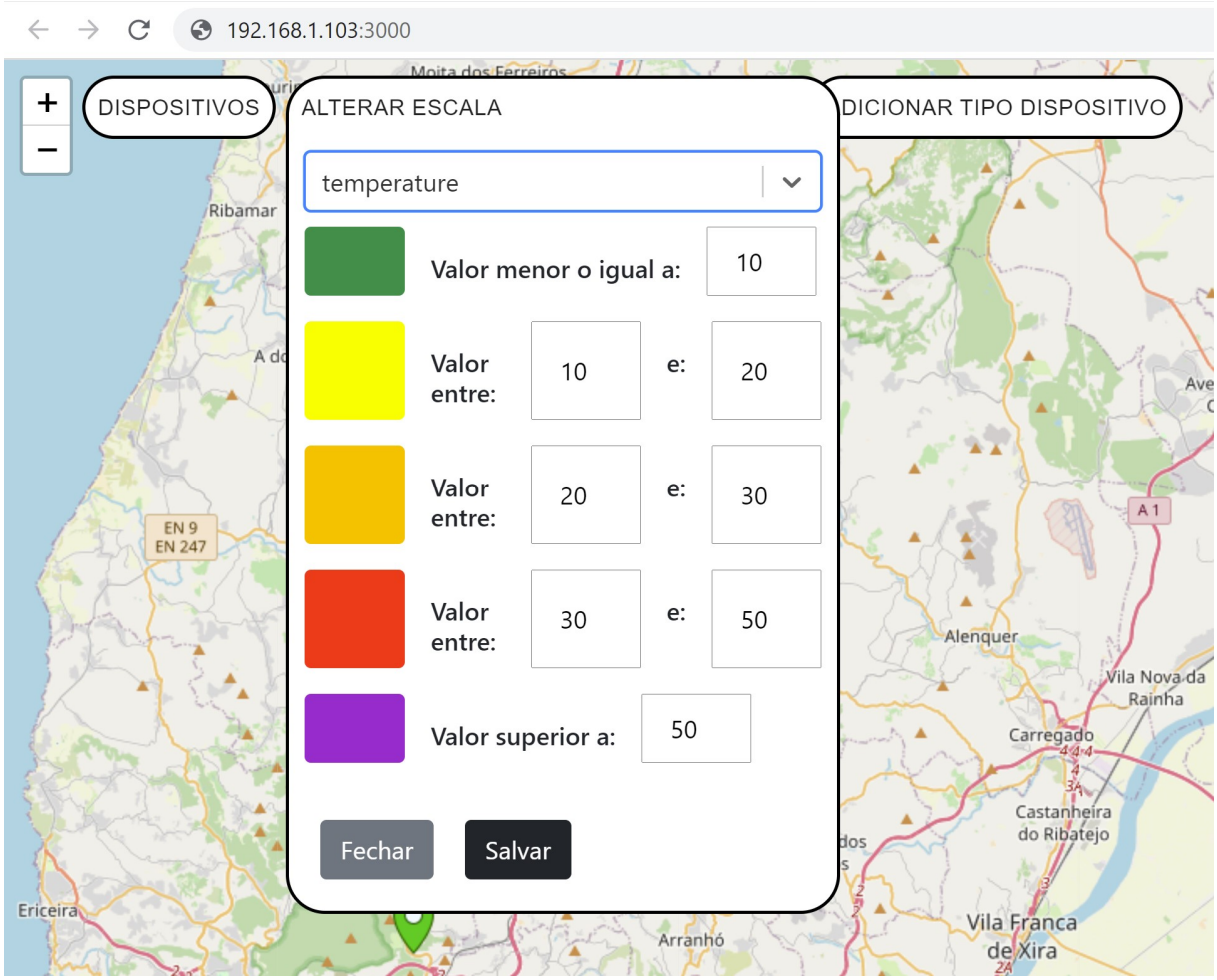

Figura 3.9: Formulário para alterar escala de cores.

inclusive com incremento de funcionalidade, a API de acesso sofreu alterações, nomeadamente a API de gestão, acessível através do protocolo HTTP, bem como a API de subscrição de mensagens enviadas pelos dispositivos, acessível via protocolo MQTT.

No momento em que a nova versão da TTN foi lançada, este projeto já tinha a sua arquitetura completamente definida e muita da sua implementação concluída. Este acontecimento pôs em risco a viabilidade do projeto tal como estava planeado, pois findado o prazo de vida dado à versão corrente da TTN, deixaria de ser possível a esta plataforma desenvolvida interagir com a TTN, nomeadamente o componente *Device Management API*, no aprovisionamento e remoção de dispositivos na TTN, bem como o componente IoT Agent LoRaWAN, seria incapaz de subscrever a mensagens de dispositivos na TTN, e assim rececionar de novas mensagens. Foi necessário abordar o problema emergente, com a identificação dos pontos de quebra nos componentes afetados e aplicação de correções nos mesmos.

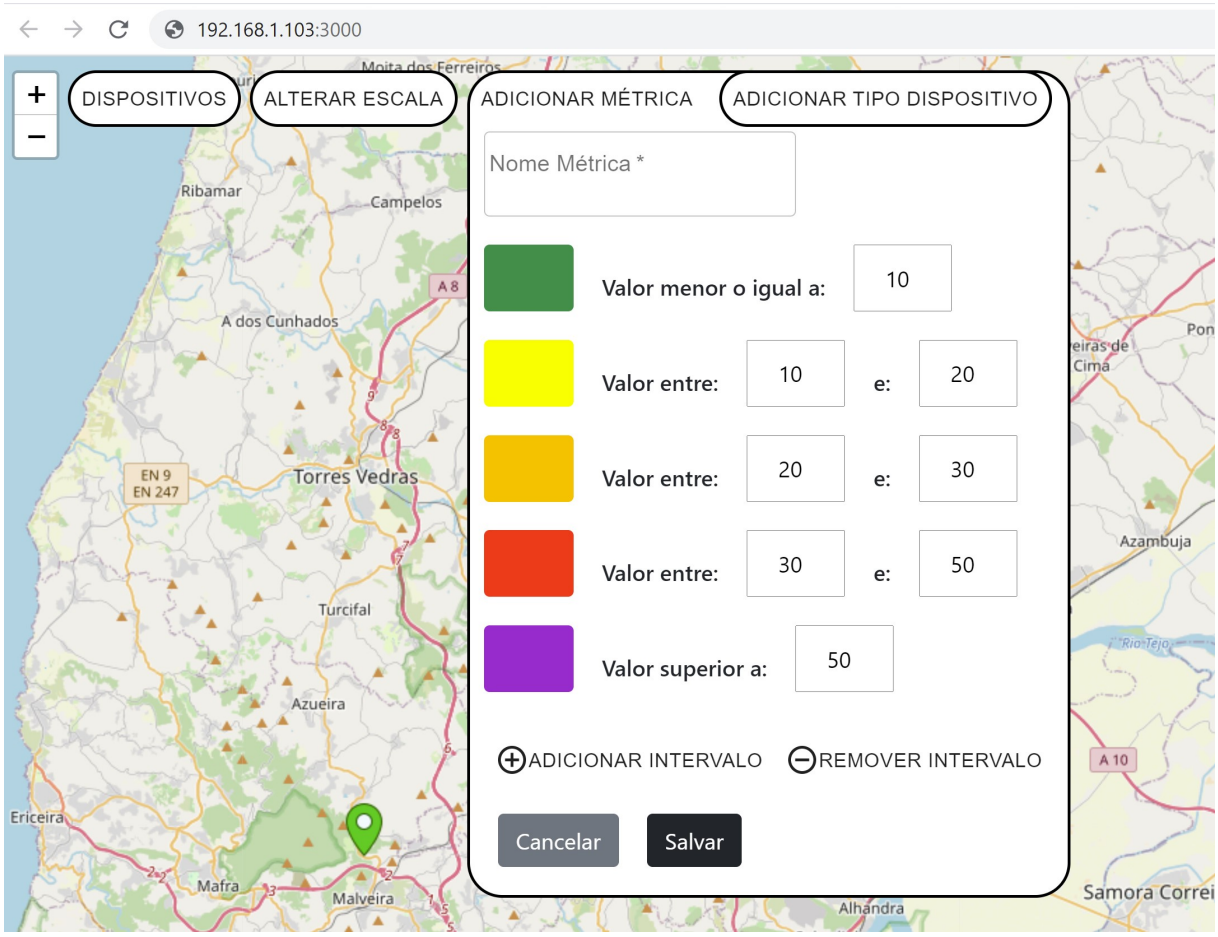

Figura 3.10: Formulário para definir nova métrica.

3.3.4.1 Modificar o componente IoT Agent LoRaWAN

Foi analisado o código fonte deste componente pertencente à *framework* FIWARE, para identificar as bibliotecas de acesso à TTN e como o acesso era realizado. Também foi analisada a interface MQTT da nova versão da TTN. Foi identificado que os URLs que identificam os tópicos de mensagens associados aos dispositivos, sofreram alterações na nova versão da TTN. Os URLs passaram a conter mais subdiretórios na sua composição.

As diferenças nos URLs MQTT entre versões é:

URL na versão TTN V2
<ttn app id>/devices/<ttn device id>/

URL na versão TTN V3
v3/<ttn app id>@ttn/devices/<ttn device id>/up

Foram aplicadas as correções necessárias ao componente e disponibilizado o código do

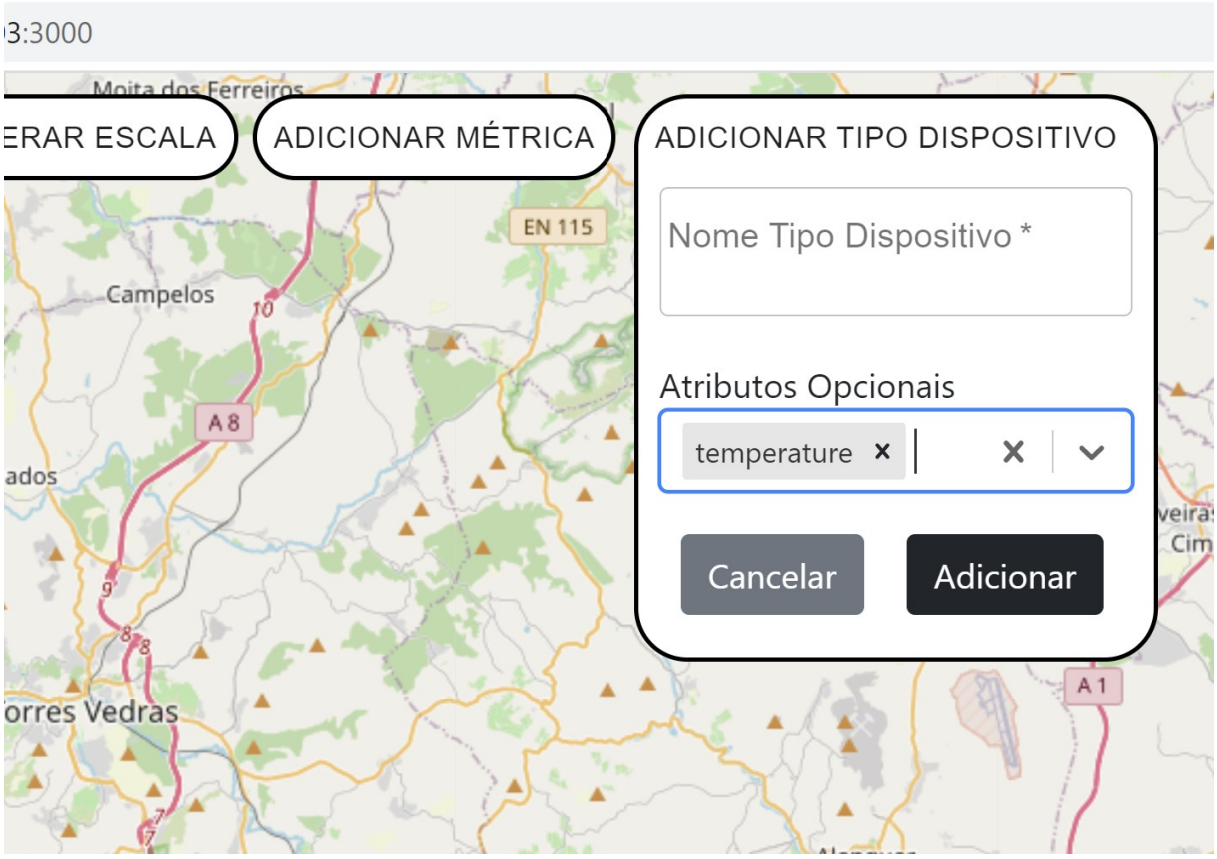

Figura 3.11: Formulário para definir um novo tipo de dispositivo.

mesmo em repositório público no GitHub[15]. Também foi construída a respetiva imagem do contentor do componente e disponibilizado em repositório publico no DockerHub[16].

3.3.4.2 Atualizar o componente *Device Management API*

O componente *Device Management API* foi implementado em Javascript para ser executado num serviço NodeJs. A versão inicialmente implementada deste componente, realizava o acesso à TTN V2 utilizado um pacote NodeJs desenvolvido por johanstokking[17]. Este pacote disponibilizava uma série de bibliotecas de acesso à API de gestão da TTN via protocolo gRPC[18]. O projeto de desenvolvimento do pacote utilizado, segundo o autor foi descontinuado.

Como passos de resolução desta questão, sugiram duas alternativas. Ou modificar o

[15]Disponível em: https://github.com/IvoPedroso/IoTagent-LoRaWAN
[16]Disponível em: https://hub.docker.com/repository/docker/ivomiguel/iotagent-lorawan
[17]Disponível em: https://github.com/TheThingsArchive/node-app-sdk
[18]https://grpc.io/

pacote usado até então, para que seja conforme com a nova versão da TTN, ou desenvolver uma biblioteca completamente nova para aceder à TTN. A escolha foi desenvolver uma nova biblioteca, pois era esta opção que minimizava o esforço necessário para obter a funcionalidade pretendida, pois a biblioteca usada anteriormente era extensa, com diversas dependências, e múltiplas funcionalidades que não eram necessárias a este projeto.

Foi analisa a API de gestão da nova versão da TTN e foi implementada uma biblioteca Javascript que acede à API de gestão da TTN via protocolo HTTP, implementando as funções estritamente necessárias para este projeto, ou seja, aprovisionar dispositivos, remover dispositivos e listar dispositivos registados. O código fonte desta nova biblioteca implementada, acompanha o código fonte do componente *Device Management API*.

3.4 Instanciação da plataforma

Nesta secção são descritos os processos desenvolvidos e disponibilizados para instanciação da plataforma apresentada. Procurou-se disponibilizar processos que permitam instanciar a plataforma da forma mais simples possível, minimizando a interação necessária. Para tal, as formas de instanciação disponibilizadas são através da execução de um simples comando, e caso seja necessária parametrização, basta a edição de um único ficheiro, que contém a definição das várias variáveis que parametrizam os vários componentes.

Como todos os componentes que constituem a plataforma, são disponibilizados em contentores de execução virtual e os mesmos são disponibilizados em repositório público na internet, não é necessário realizar manualmente o download dos componentes para a instanciação, basta apenas ter os *scripts* de instanciação e o respetivo ficheiro de parâmetros, na infraestrutura onde será realizada a instanciação.

São disponibilizadas duas formas de instanciação. Uma forma para instanciação sobre um ambiente de execução de contentores virtuais Docker, sem o suporte de qualquer ferramenta de orquestração de contentores. Outra forma disponibilizada, é para instanciação sobre um ambiente de execução de contentores suportada pela ferramenta de orquestração Kubernetes.

3.4.1 Instanciação em plataforma Docker

A forma inicial desenvolvida para instanciar a plataforma e seus componentes no decorrer do projeto apresentado, foi para o ambiente Docker sem qualquer ferramenta de suporte para orquestração de contentores.

A escolha do Docker foi devido a este demonstrar ser uma ferramenta que permite a introdução à tecnologia de contentores virtuais com um esforço relativamente menor que outras ferramentas similares, pois antes de participar neste projeto, não tinha tido qualquer experiência prévia com este tipo de tecnologia, apesar de saber de forma geral em que é que consiste. Outro fator que pesou na escolha foi a imensa popularidade e extensa comunidade que utiliza e produz documentação, o qual facilita a aprendizagem e resolução de problemas. Também é de opinião comum na comunidade que é uma ferramenta relativamente menos complexa que outras ferramentas semelhantes.

Para automatizar a instanciação dos componentes sobre o motor de execução de contentores Docker, foi utilizada a ferramenta Docker Compose[19]. O Docker Compose define uma especificação que permite a definição de objetos que caracterizam contentores e a sua parametrização, permitido criação de manifestos (*scripts*) yaml nos quais são definidos múltiplos contentores, que são interpretados pelo Docker Compose e instanciados automaticamente no motor de execução de contentores Docker.

Para instanciar os múltiplos componentes da plataforma apresentada, foram definidos dois *scripts* Docker Compose, que permitem instanciar a plataforma em duas fases distintas, na qual os componentes da segunda fase só são instanciados após a instanciação dos componentes da primeira fase ter terminado. Este faseamento na instanciação de componentes, deve-se à dependência que alguns componentes têm no momento de instanciação, de informação que é gerada durante a execução de outros componentes, nomeadamente dependência de tokens de autenticação gerados pelo componente Keyrock.

Na primeira fase são instanciados os componentes Orion, IoT Agent LoRaWAN, Cygnus, STH-Comet, KeyRock, MongoDB, MySQLDB e *Device Management API*, pois estes não têm dependência sobre dados gerados por outros componentes no momento da sua instanciação.

Na segunda fase são instanciados os componentes PEP-Proxy e o componente Aplicação Web, pois estes dependem de credenciais de autenticação geradas no componente KeyRock, necessárias para parametrização dos contentores destes componentes antes da instanciação. Estas credenciais são geradas a pedido no KeyRock, devendo estes

[19]Disponível em:https://docs.docker.com/compose

pedidos ser executados após conclusão da primeira fase da instanciação, quando o Keyrock já se encontra em execução e a atender pedidos através do protocolo HTTP.

Entre as duas fases, como já indicado é necessário realizar um pedido ao Keyrock para gerar tokens de autenticação, mas também é necessário realizar um pedido ao componente Orion, para realizar o registo neste, do componente Cygnus como subscritor de alterações de contexto. Para realizar estes pedidos entre fases, foi desenvolvido um *shell script* , que é ser executado neste instante indicado.

Para que seja possível instanciar a plataforma implementada através de um único comando, é necessário conjugar a instanciação das duas fases, juntamente com o *script* a executar entre fases acima indicadas. Foi desenvolvido um *shell script* principal[20] que executará as duas fases e o *script* intermédio acima indicados na sequência correta. Este *script* principal ao instanciar a primeira fase, aguarda que o que todos os componentes desta primeira fase terminem, depois executa o *script* intermédio, e de seguida instância a segunda fase de instanciação.

As variáveis que parametrizam a instanciação dos contentores, são definidas num ficheiro distinto dedicado à parametrização de variáveis de ambiente que são associadas aos respetivos contentores, permitindo configurar como os componentes interagem entre si. Também foram criados segredos Docker, para armazenar credenciais necessárias a alguns componentes de forma mais segura.

A instanciação em motor de execução Docker sem orquestração, instancia e executa todos os contentores na mesma máquina anfitriã, ou seja, na máquina onde o motor Docker foi instalado. Esta é uma desvantagem, pois se a máquina anfitriã falha, todos os componentes da plataforma também falham. Também faz com que a plataforma esteja a executar de forma não descentralizada, do ponto de vista físico, pois apesar de os componentes estarem isolados em contentores, estão fisicamente a executar na mesma máquina e a disputar os recursos disponíveis da máquina.

3.4.2 Instanciação em plataforma Kubernetes

Para oferecer uma forma de instanciação da plataforma apresentada mais orientada para execução em serviços computacionais na nuvem, foi definido um manifesto no formato yaml, que permite a instanciação da plataforma num cluster de nós computacionais geridos por Kubernetes[11].

O Kubernetes é um sistema de orquestração de contentores de execução virtual, que

[20]Disponível em:https://github.com/IvoPedroso/PlataformaFIWAREInstancia

gere automaticamente um conjunto de um ou mais nós computacionais, os quais executam um motor de execução de contentores virtuais. Trata-se de uma aplicação em código aberto, que permite aos utilizadores a criação do seu próprio cluster de nós computacionais, ou alternativa utilizar os serviços Kubernetes disponibilizados pelos principais operadores de serviços de computação na nuvem.

O Kubernetes realiza a gestão automática dos contentores no cluster, procurando manter o estado de execução dos contentores, tal qual como definido no manifesto do utilizador.

O sistema de orquestração oferecido pelo Kubernetes, realizada a distribuição dos contentores de forma automática pelos diferentes nós do cluster, tentado maximizar a utilização dos recursos computacionais disponíveis, sendo indiferente do ponto de vista do contentor em execução, em qual nó se encontra. Oferece alguns serviços que automatizam a gestão e facilitam a instanciação de sistemas compostos por múltiplos componentes instanciados em contentores virtuais, nomeadamente realiza a recuperação automática de contentores após falha, também disponibiliza um serviço de descoberta de nomes de rede do tipo DNS para os contentores em execução, gere e atribui automaticamente volumes de armazenamento para os contentores, e também oferece a possibilidade de escalar automaticamente na horizontal, através de replicação de contentores em execução, entre outros serviços oferecidos pelo Kubernetes.

Em contraste, a utilização do kubernetes também tem alguns pontos negativos, pois a configuração do sistema de orquestração em si, não é trivial, pois trata-se de um sistema altamente parametrizável e complexo. A definição de manifestos para instanciação de elementos no cluster também pode ser complexa e de difícil introdução aos diferentes conceitos existentes. Tal como na utilização da tecnologia Docker, foi necessário realizar um estudo introdutório a esta ferramenta, pois trata-se de ferramenta na qual não tinha experiência prévia e com uma curva de aprendizagem à tecnologia, relativamente exigente.

Os manifestos definidos pelo utilizador, permitem caracterizar diversos objetos Kubernetes, que identificam de forma declarativa, entre outros conceitos, quais os contentores a instanciar e como devem ser parametrizados.

Os contentores em Kubernetes são executados em Pods. O Pod é um conceito em Kubernetes que define uma entidade instanciável, na qual podem existir diversos contentores em execução, sendo estes geridos como uma única entidade, na qual partilham os recursos disponível no Pod. Caso existam vários contentores num Pod, estes são chamados de Sidecar-Container[15]. No Pod também podem existir Init-Container,

que são contentores que são executados no momento de instanciação do Pod para realização de tarefas de iniciação e preparação, sendo encerrados após terminarem a sua tarefa. Podem existir vários Init-Container, que são instanciados sequencialmente, após o Init-Container antecessor terminar. Após todos os Init-Containers terminarem, são então instanciados em simultâneo todos os contentores convencionais (não Init-Container) do Pod.

A parametrização dos contentores é realizada através de ConfigMaps e Secrets. Os ConfigMaps são objetos que permitem a definição de variáveis de ambiente, acessíveis aos componentes em execução nos contentores. Os Secrets são objetos que normalmente permitem definir credenciais de acesso a outros serviços e que ficam disponíveis ao componente no contentor, em sistema de ficheiros.

Para instanciação da plataforma apresentada, foi definido um manifesto yaml[21], que define diferentes objetos da especificação Kubernetes, nomeadamente ConfigMaps e Secrets para definição de valores de parametrização dos componentes, Deployments e StatefulSets para definição de modos de instanciação de contentores, Services para definição de lógica de acesso por rede aos contentores, e por fim ServiceAccounts, Roles e RoleBinding para definições de permissões de acesso ao cluster por parte dos contentores.

Os componentes que não requerem armazenamento próprio ou recorrem a outros componentes para persistir os seus dados, foram definidos no manifesto, como objetos Deployment. Estes objetos definem contentores Stateless, ou seja, contentores sem estado persistente armazenado, que podem ser reiniciados automaticamente e movidos através dos diferentes nós do cluster, sem prejuízo da disponibilidade ou consistência do serviço prestado pelo componente associado ao contentor. Os componentes em contentores definidos com este tipo de objeto foram, o Orion, o IoT Agent LoRaWAN, o Cygnus, o STH-Comet, o *Device Management API*, a Aplicação Web e os PEP-Proxy.

Dado alguns deste componentes dependerem de outros para persistência dos seus dados, foi definido um Init-Container cuja imagem do contentor é um pequeno componente implementado para o efeito, que verifica se o serviço ao qual existe dependência, responde no endpoint pretendido e termina a execução em caso afirmativo. Em caso negativo, vai repetido a verificação de disponibilidade no endpoint. Na prática, o Init-Container apenas permite o contentor principal iniciar, quando o serviço que existe dependência fica disponível.

Os componentes que necessitam de Volumes de armazenamento não efémeros para

[21]Disponível em:https://github.com/IvoPedroso/PlataformaFIWAREInstancia

persistir os seus dados, foram definidos como objetos StatefulSet. O Kubernetes atribui um Volume de armazenamento automaticamente aos contentores definidos através deste tipo de objeto, garantido que é sempre atribuído o mesmo volume ao componente, em caso de reinício de contentor ou se o contentor for movido para outro nó do cluster. Os componentes MongoDB, Keyrock e MySQL foram definidos como StatefulSet.

Apesar do componente Keyrock não precisar de volume para persistência de dados e usar o componente MySQL para persistir os seus dados, foi definido como SideCar-Container juntamente com o componente MySQL num StatefulSet, pois estes dois componentes estão intrinsecamente ligados, permitindo também garantir restrição de acessos ao componente MySQL, existindo apenas comunicações internamente no Pod, do Keyrock com o MySQL e externamente apenas é possível aceder ao Keyrock.

Dada a necessidade dos componentes PEP-Proxy e Aplicação Web, terem tokens de autenticação gerados no componente Keyrock, foi necessário implementar um componente e disponibilizada a respetiva imagem de contentor virtual, que execute pedidos ao Keyrock, para geração dos tokens. Este contentor é definido como Init-Container de um dos PEP-Proxys, para que execute antes que contentor do PEP-Proxy seja instanciado e seja gerado os tokens necessários à instanciação do PEP-Proxy. Este componente implementado para o Init-Container executa um *shell script* que faz os pedidos necessários através da aplicação Curl ao Keyrock.

Após gerados os tokens no Init-Container, é necessário parametrizar a informação dos token nos contentores que necessitam destes dados para instanciação. Para tal após geração dos token no *shell script* do Init-Container, é feita uma chamada à API do Kubernetes a partir deste Init-Contaiter, no qual é definido um novo ConfigMap no cluster Kubernetes, com a informação dos tokens.

Para que o Init-Container tenha permissão para fazer pedidos à API do Kubernetes, foi definido um Role com as permissões necessárias, definido um AccountService ao contentor em causa, e por fim definido um RoleBinding entre o Role e ServiceAccount criados. Ou seja, foram definidas autorizações que permitem a um contentor em execução no cluster, realizar chamadas à API do cluster onde se entra em execução

Com o manifesto definido desta forma é possível instanciar toda a plataforma através da aplicação de um único manifesto, no cluster Kubernetes.

4

Avaliação de desempenho da solução sobre carga

Neste capítulo são descritos os testes efetuados à solução apresentada e resultados obtidos, de modo a avaliar a sua capacidade em suportar carga constante de mensagens oriundas dos dispositivos, ao longo de um determinado período de tempo.

Foram testados diferentes cenários, nomeadamente diferentes configurações e formas de execução dos componentes da solução, com especial destaque para o componente MongoDB, que é utilizado por outros componente para persistir os seus dados, nomeadamente durante o processamento de novas mensagens transmitidas por dispositivos. O foco no componente MongodDB nos testes realizados, foi devido a este componente ter sido identificado como o principal estrangulador do desempenho da solução. Também foram testados diferentes número de réplicas de alguns dos componentes da solução, nomeadamente do componente Orion e do componente MongoDB, para avaliar a sua elasticidade e capacidade de escalar horizontalmente.

As medições realizadas têm com base um número fixo de mensagens enviadas por segundo, durante um período de 15 minutos, no qual é verificado se a plataforma consegue ou são persistir a totalidade das mensagens recebidas. A cada teste realizado são enviados diferentes números de mensagens por segundo e encontrado o ponto de falha no qual existem perdas de mensagens. Também é verificado se com o aumentar do número de mensagens por segundo além do ponto de falha da plataforma, a perda de mensagens ao longo do teste realizado é significativa ou não.

Cada mensagem enviada ao serviço MQTT, inclui um campo com o número de identificação da mensagem, o qual é gerado no momento de envio da mensagem através de um contador. Dependendo do valor incluído no campo, as mensagens enviadas têm uma dimensão que varia entre os 96 e os 101 bytes.

Os testes realizados foram feitos através de simulação de mensagens enviadas por dispositivos, pois não foi possível ter acesso a dispositivos reais para realizar os testes em ambiente real.

4.1 Configuração do ambiente de testes

Dada a solução apresentada utilizar a TTN e esta disponibilizar um acesso partilhado de uso aberto à comunidade, a utilização massiva deste meio para a realização de testes, iria possivelmente prejudicar o bom uso e disponibilidade deste recurso para a restante comunidade, bem como, infringir algumas das suas regras de utilização. Assim foi modelado um ambiente de testes, onde a plataforma TTN é substituída por um serviço que disponibiliza um MQTT Broker, que emula as funções da TTN na publicação e subscrição de mensagens oriundas dos dispositivos. A aplicação usada para disponibilizar o serviço MQTT Broker foi o Mosquitto[1], o qual foi instalado numa máquina virtual e o seu serviço acessível por rede aos componentes da plataforma.

Foi necessário também realizar uma ligeira alteração ao componente IoT Agent LoRaWAN[2], para que este comunique com o serviço MQTT instalado para este ambiente de teste, de modo a subscrever tópicos de mensagens e seja notificado de novas mensagens neste serviço. A alteração deste componente não modifica o comportamento ou a forma de processamento do subconjunto da plataforma sobre o qual recaem os testes realizados, apenas permite alterar a forma como o IoT Agent LoRaWAN subscreve os tópicos de mensagens. Deste modo os resultados obtidos nos testes realizados, podem ser extrapolados para a solução proposta.

A Figura 4.1 ilustra o subconjunto da solução usado para realização dos testes. Os restantes componentes da solução não foram incluídos, pois não participam no processamento de novas mensagens dos dispositivos. Como já descrito no Capítulo 3, o IoT Agent recebe notificações com mensagens dos dispositivos, o Orion regista os dados correntes de contexto, o Cygnus persiste o histórico de contexto, o MongoDB dá suporte à persistência dos dados dos outros componentes e por fim o STH-Comet

[1]Disponível em: https://mosquitto.org/
[2]Disponível em: https://github.com/IvoPedroso/IoTagent-LoRaWAN/tree/TestToMqttMosquito

permite consultar o histórico registado. Para simular o envio de mensagens, foi desenvolvido um pequeno *script* (App.js) em Javascript e executado em NodeJs em cada teste realizado. Este script regista no MQTT Broker, mensagens associadas a um dispositivo. A latência de envio das mensagens é indicada por parâmetro de execução, identificando o número de mensagens por segundo a enviar no teste realizado.

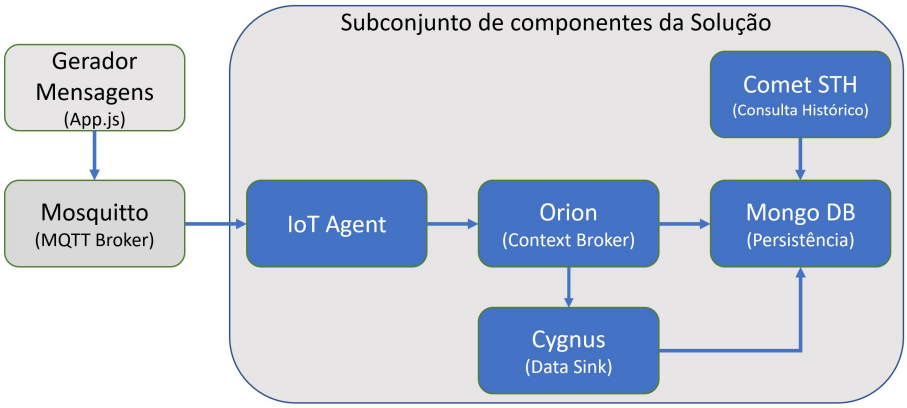

Figura 4.1: Arquitetura do ambiente de testes.

A Figura 4.2 ilustra a interacção entre os diferentes componentes, no processamento das mensagens oriundas dos dispositivos. Nesta Figura podemos constatar a utilização recorrente do componente MongoDB na persistência dos dados enviado pelos dispositivos. Como o armazenamento dos dados nestes cenários de testes é suportado por um serviço NFS (*Network File System*), recorrendo a comunicação rede para transferência de dados e posterior acesso a armazenamento em disco físico, o custo na obtenção e persistência de dados, acumulado por todos os pedidos realizados em cada teste, deverá ser significativo e influenciar negativamente os resultados obtidos nos testes realizados.

Foram criados 12 cenários de teste, designadamente:

Cenário D1 — Execução em Docker, sem orquestração de componentes, em máquina virtual configurada com 2Gb de memória RAM;

Cenário D2 — Execução em Docker, sem orquestração de componentes, em máquina virtual configurada com 4Gb de memória RAM;

Cenário K1a — Usando Kubernetes com os nós do cluster a executar em máquinas virtuais configuradas com 2Gb de memória RAM e com o Componente Mongo DB configurado como StatefullSet;

Cenário K1b — Usando Kubernetes com os nós do cluster a executar em máquinas

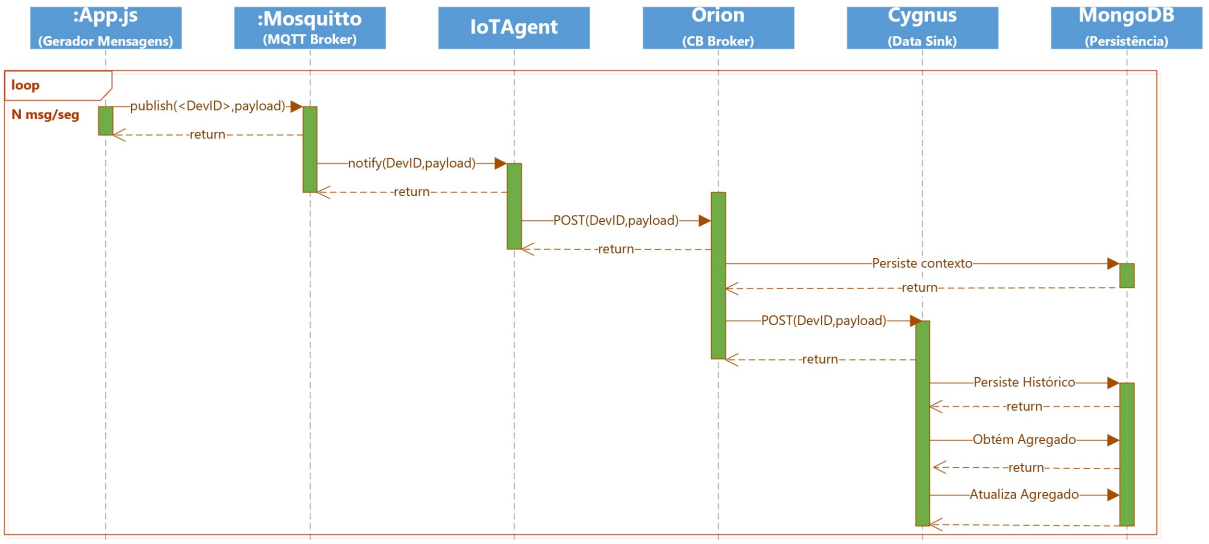

Figura 4.2: Diagrama de sequência do envio de mensagens de teste.

virtuais configuradas com 2Gb de memória RAM, com o Componente Mongo DB configurado como StatefullSet e com o componente Orion com 2 réplicas;

Cenário K1c — Usando Kubernetes com os nós do cluster a executar em máquinas virtuais configuradas com 4Gb de memória RAM e com o Componente Mongo DB configurado como StatefullSet;

Cenário K1d — Usando Kubernetes com os nós do cluster a executar em máquinas virtuais configuradas com 4Gb de memória RAM, com o Componente Mongo DB configurado como StatefullSet e com o componente Orion com 2 réplicas;

Cenário K2a — Usando Kubernetes com os nós do cluster a executar em máquinas virtuais configuradas com 2Gb de memória RAM, com o Componente Mongo DB configurado em modo Shard;

Cenário K2b — Usando Kubernetes com os nós do cluster a executar em máquinas virtuais configuradas com 2Gb de memória RAM, com o Componente Mongo DB configurado em modo Shard e com o componente Orion com 2 réplicas;

Cenário K2c — Usando Kubernetes com os nós do cluster a executar em máquinas virtuais configuradas com 4Gb de memória RAM, com o Componente Mongo DB configurado em modo Shard;

Cenário K2d — Usando Kubernetes com os nós do cluster a executar em máquinas virtuais configuradas com 4Gb de memória RAM, com o Componente Mongo DB configurado em modo Shard e com o componente Orion com 2 réplicas;

Cenário K2c — Usando Kubernetes com os nós do cluster a executar em máquinas virtuais configuradas com 2Gb de memória RAM, com o Componente Mongo DB configurado em modo Shard e com os subcomponentes do MongoDb configurados com duas réplicas cada;

Cenário K2c — Usando Kubernetes com os nós do cluster a executar em máquinas virtuais configuradas com 4Gb de memória RAM, com o Componente Mongo DB configurado em modo Shard e com os subcomponentes do MongoDb configurados com duas réplicas cada;

A subsecções seguintes descrevem com mais detalhe cada cenário e os resultados obtidos.

4.1.1 Execução em Docker, sem orquestração de componentes

Neste cenário, os componentes da solução são configurados e executados em contentores Docker, não existindo uma ferramenta de orquestração de contentores. Toda a gestão dos contentores fica da responsabilidade do motor Docker que executa os contentores e aplica as configurações definidas para cada contentor.

Esta configuração implica que todos os componentes da solução, correm em contentores na mesma máquina anfitriã (*host*) dos contentores, partilhando os mesmos recursos computacionais disponíveis. Foi testado executar os contentores e respetivos conjunto de testes sobre uma máquina virtual anfitriã com Sistema Operativo Ubuntu com 2Gb de memória RAM alocados. Também foi testado executar os contentores e conjunto de testes na mesma máquina, mas configurada com 4Gb de memória RAM alocados.

Após realização dos testes, observou-se que os testes realizados com a máquina anfitriã configurada com 2Gb de RAM, foi possível enviar até 70 mensagens por segundo sem perda de mensagens, ou seja, todas as mensagens enviadas foram recebidas e persistidas na plataforma neste ambiente de teste. A partir deste limiar de 70 mensagens por segundo, a solução não reteve a totalidade das mensagens enviadas e a máquina anfitriã onde os contentores executam, fica com capacidade de processamento (CPU) reduzida. Isso acontece, pois o Sistema Operativo inicia um processo de swap de memória para disco, que esgota a capacidade de processamento disponível, ficando a própria máquina anfitriã com resposta limitada. Além disso, o terminal de linha de comandos da máquina anfitriã fica com um delay de 5 a 10 segundos para dar feedback do input inserido, e uma simples listagem dos componentes docker em execução pode levar 1 minuto mostrar os resultados. Quando ocorrem estes casos de sobrecarga

de processamento devido a processo de swap de memória, e após já ter terminado do envio de mensagens, o processo de swap continua a afetar a máquina anfitriã e a sua capacidade de processamento durante 20 a 30 minutos, exceto se o contentor do componente MongoDB for forçado a encerrar, via comando Docker.

Com a máquina anfitriã configurada com 4GB de RAM, foi possível enviar até 110 mensagens por segundo sem perda de mensagens. A partir deste valor, observa-se também perda de mensagens, e a capacidade de resposta fica limitada na máquina anfitriã devido sobrecarga no processador, também provocado por processamento de swap de memória, como no setup anterior com 2Gb de memória RAM.

A Figura 4.3 ilustra os resultados obtidos neste cenário de testes. Podemos verificar que ao ultrapassar a capacidade da plataforma no processamento de mensagens por segundo, há um decréscimo acentuado da capacidade de retenção das mensagens enviadas para a plataforma. O esgotar de memória RAM durante os testes por parte do componente MongoDB, leva à troca de segmentos de memória para disco por parte do Sistema Operativo, afetando também a capacidade de leitura e escrita em disco para os restantes processos e contentores em execução.

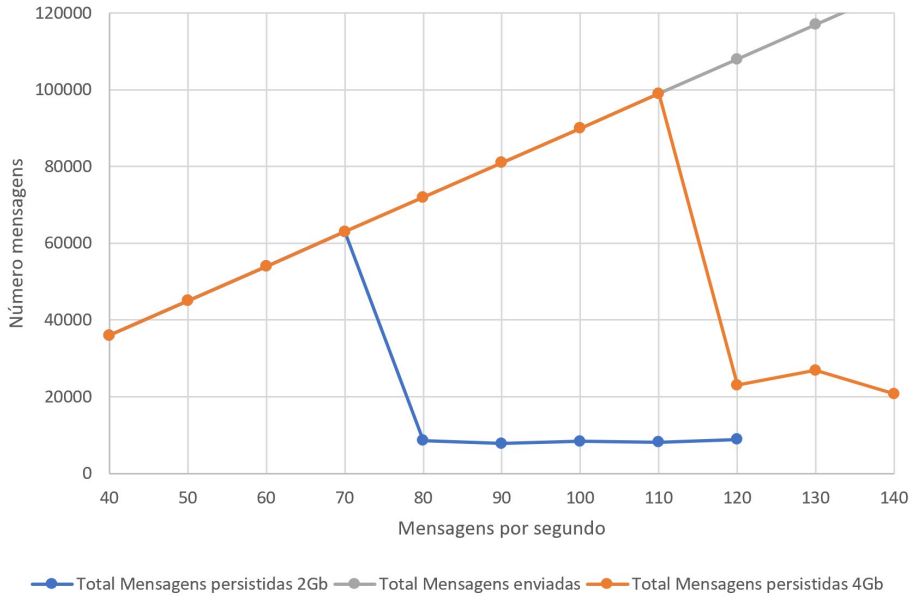

Figura 4.3: Resultados dos testes com os contentores a executar em Docker.

4.1.2 Execução da solução em Docker, usando Kubernetes para orquestração de contentores

Este cenário procurar validar o desempenho e as possíveis vantagens ao executar os componentes da solução apresentada, num cluster de nós computacionais geridos pela

aplicação Kubernetes.

Para este cenário, foi instalado o Kubernetes em três máquinas virtuais disponibilizadas no cluster computacional do ISEL, com as mesmas características da máquina usada no cenário anterior. As máquinas virtuais que foram disponibilizadas, estão localizadas em máquinas físicas diferentes, de forma a minimizar interferências de processos entre máquinas virtuais durante os testes realizados.

Os três nós que compõem o cluster Kubernetes criado, foram configurados para permitir o agendamento de contentores para execução, o que no contexto Kubernetes são normalmente chamados de *Worker Nodes*. Um desses nós também foi configurado como nó mestre, no contexto Kubernetes também designado normalmente de *Master Node*, cuja responsabilidade é controlar e gerir o cluster, e os seus respetivos contentores. O cluster foi configurado para usar o motor de execução de contentores Docker.

Este cenário permite distribuir automaticamente os diferentes componentes da plataforma nas diferentes máquinas que compõem o cluster, aumentando a capacidade computacional disponível para a solução, além de também permitir a replicação de componentes da plataforma, caso necessário.

Para conseguir persistência dos dados associados aos contentores em execução, num cluster gerido através Kubernetes, é necessário a configuração de Volumes de armazenamento que possam ser atribuídos dinamicamente aos contentores. Os sistemas de armazenamento associados aos Volumes necessitam de ser externos ao cluster, ou seja, os Volumes não podem usar sistemas de ficheiros locais a um dos nós do cluster, pois como os contentores podem ser alocados para execução a qualquer um dos nós do cluster, podia ocorrer um contentor ser alocado num nó diferente do qual existe o volume que lhe foi atribuído e não conseguir aceder ao mesmo. Assim, os sistemas de armazenamento externos necessitam de ser configurados no cluster Kubernetes, de modo a serem associados a Volumes de armazenamento criados e geridos pelo cluster.

Como sistema de armazenamento para o cluster, foi configurado um serviço NFS numa das máquinas virtuais, disponibilizando através de comunicação de rede um serviço de acesso a pastas de rede, que irá permitir configurar pontos de persistência para associação a Volumes no cluster, permitido alocar os mesmos a contentores que necessitem, sendo acessíveis independentemente do nó em que os contentores se encontrem. O tráfego de dados entre os contentores e os Volumes associados ao serviço NFS, é gerido automaticamente pelos controladores do cluster Kubernetes. O serviço NFS, apesar de utilizar protocolos pouco eficientes comparativamente a formas de armazenamento orientados para o armazenamento em clusters (e.g. aws-ebs, azure-disk, vsphere-volume, entre outros), era a única forma menos complexa de criar, configurar

e disponibilizar um sistema de armazenamento compatível com o cluster Kubernetes criado.

Para que seja possível o acesso ao serviço NFS para armazenamento dentro do cluster Kubernetes, é necessário definir objetos no cluster Kubernetes que identificam como deve usar o serviço NFS para alocar Volumes dinamicamente. Foi definido um objeto StorageClass no cluster, no qual é necessário definir a propriedade Provider, que identifica o Plugin a ser usado pelo Kubernetes na alocação de Volumes associados a um tipo de armazenamento. Também foram definidos vários objetos PersistenceVolume no cluster, os quais definem os vários Volumes que podem ser criados pelo Kubernetes, juntamente com tipo de armazenamento e suas propriedades. Na definição dos PersistenceVolume foram definidas as seguintes propriedades, o nome do StorageClass a usar (neste caso com nome do StorageClass acima indicado), a capacidade disponível no meio de armazenamento usado para o Volume, o modo de acesso (leitura e/ou escrita) permitido, e qual o tipo de armazenamento usado e suas propriedades, neste caso do tipo NFS, e qual o endereço do servidor usado e o caminho para a pasta partilha no servidor.

4.1.2.1 Componente Mongo DB, configurado como StatefullSet

Esta configuração, replica a configuração descrita na secção "Execução em Docker, sem orquestração de componentes", onde os contentores eram executados pelo motor Docker sem orquestração associada, mas agora com a vantagem da orquestração do Kubernetes a distribuir e gerir os contentores em execução, pelos nós do cluster.

Todos os componentes da solução foram configurados no cluster como objetos Deployment, os quais não necessitam que a persistência de dados seja duradoura no contentor, ou seja, sem Volume de armazenamento associado. Existe uma exceção, que é o componente MongoDB, o qual foi configurado como um objeto StatefullSet. Este objeto StatefullSet tem na própria definição, a definição de um objeto PersistenceVolumeClaim, que indica a necessidade que lhe seja atribuído um Volume para armazenamento, com determinadas características. O Kubernetes ao agendar para execução um StatefulSet, tenta montar um volume dinamicamente em tempo de execução ao contentor criado para o StatefulSet, a partir dos objetos PersistenceVolume previamente definidos e que cumpram os requisitos solicitados no PersistenceVolumeClaim do StatefulSet.

Nos testes realizados para este cenário, usando o cluster Kubernetes com os três nós do cluster configurados com 2GB de memória RAM cada, obteve-se um máximo de 140 mensagens por segundo sem perda de mensagens. A partir deste limiar, o componente IoT Agent reporta que não consegue entregar novas mensagens ao componente Orion,

pois a ligação de rede com este é quebrada. Também se verificaram falhas no componente Cygnus. Ainda assim, tentamos verificar qual o desempenho com a replicação do Orion com dois contentores, não tendo havido uma melhoria de desempenho.

Os resultados obtidos, levam a crer que existe um estrangulamento no acesso aos Volumes de armazenamento, provavelmente pela utilização de um sistema de persistência não ideal para clusters de execução de contentores, como é o caso do NFS. Como o MongoDB não processa de forma célere os pedidos oriundos do Cygnus e do Orion, devido aos constrangimentos no acesso ao sistema de armazenamento, causa também que o Cygnus e o Orion entrem em sobrecarga, não atendendo a pedidos de Readiness por parte do Kubernetes e por vezes são encerrados e reiniciado pelo Kubernetes. Ou seja, os contentores não respondem ao cluster a pedidos de verificação efetuados para confirmar se estes se encontram em boas condições de execução, sendo encerrados por serem considerados pelo cluster, como contentores em estado de falha.

Com os nós do cluster configurados com 4Gb de memória RAM cada, obtivemos um limite de 165 mensagens por segundo, sem perdas. Como espectável, o aumento de recursos de memória RAM, levou a uma melhoria da capacidade da plataforma.

A replicação do contentor do componente Orion durante os testes de carga revelou ser ineficaz, pois este componente está dependente do componente MongoDB na persistência dos seus dados, o qual foi identificado como o ponto de estrangulamento (*bottleneck*) da plataforma neste cenário. Como o componente que entra em sobrecarga é o MongoDB, e este deixa de responder a pedidos de persistência de forma célere, fazendo os outros componentes aguardar e acumular dados para serem persistidos, é irrelevante adicionar mais réplicas do componente Orion, pois se com uma réplica o MongoDB não consegue dar resposta, então com mais réplicas o resultado será semelhante.

Com os resultados obtidos nos testes realizados, observamos que a vantagem obtida neste cenário em relação ao cenário com motor Docker sem orquestração é significativa, pois houve uma melhoria de capacidade de resposta da plataforma de 100% quando ambós cenários usaram máquinas virtuais com 2Gb de memória RAM, e uma melhoria de 54,4% quando ambos cenários usaram máquinas virtuais com 4Gb de memória RAM. Esta melhoria deve-se a ter alguns contentores dos componentes a executar em nós diferentes do cluster, aproveitando os recursos computacional disponíveis, nomeadamente em memória RAM e com isso aumentando a capacidade em reter mensagens para entrega a outros componentes ou enquanto aguarda a persistência de dados.

O custo da sobrecarga dos protocolos de gestão e comunicação inerentes à utilização do

cluster Kubernetes, necessários para a sua gestão e orquestração de contentores, mais a latência existente na transferência de dados entres nós do cluster, não foi suficiente para invalidar as vantagens obtidas com utilização do cluster Kubernetes, comparativamente ao cenário anterior sem orquestração.

A Figura 4.4 ilustra os resultados obtidos. Podemos verificar que atingido o limiar de capacidade da plataforma, por vezes há um decréscimo na capacidade da plataforma em processar mensagens, dado que um componente em falha tem um efeito sistémico nos restantes componentes.

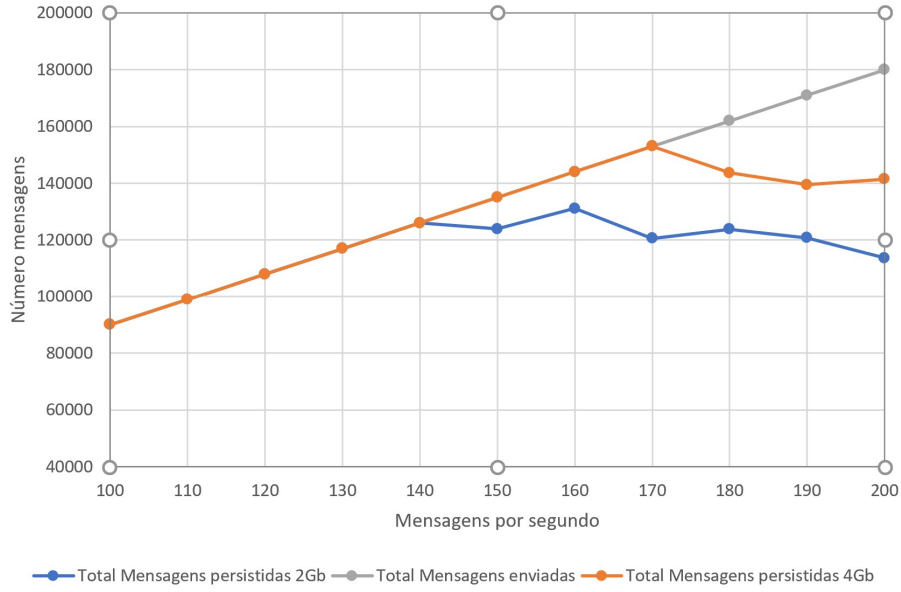

Figura 4.4: Resultados dos testes em ambiente com MongoDB como StatefulSet.

4.1.2.2 Componente MongoDB, configurado em modo Shard

Como possível solução para melhoria da capacidade de resposta do componente MongoDB, foi testado executar este componente em modo Shard[1, 2], ou seja, distribuir a gestão e processamento dos dados persistidos por este componente em diferentes subcomponentes executados em contentores diferentes, tentando tirar partido dos nós do cluster Kubernetes para repartir o esforço do componente MongoDB. Esta configuração do MongoDB em modo Shard é suportada pelo fabricante e ilustrada na Figura 4.5.

O MongoDB em modo Shard é composto pelos seguintes subcomponentes, o Router (mongos) que é responsável pelo encaminhamento dos pedidos dos clientes do MongoDB aos restantes subcomponentes do MondoDB, o ConfigServer que é responsável pela configuração e gestão dos subcomponentes que compõem o MongoDB em modo

Shard, e os Shards que persistem os dados dos clientes, segmentados nos diferentes Shards. O mínimo de Shards necessários são dois.

A configuração usada para o MongoDB em modo Shard ao nível de réplicas de contentores foi, um contentor para Router(mongos), um contentor para ConfigServer e um contentor para cada um dos dois Shards usados.

A instanciação do MongoDb em modo Shard em contentores, é complexa e difícil, dada a extensa parametrização necessária. Para auxiliar nesta tarefa, foi utilizado um *chart Helm* disponibilizado pela fabricante de Software Bitnami[3]. Helm[4] é um gestor de pacote que permite a instalação e configuração de contentores em clusters Kubernetes de forma automatizada, através da definição de *charts* que identificam e definem os componentes a ser instalados no cluster.

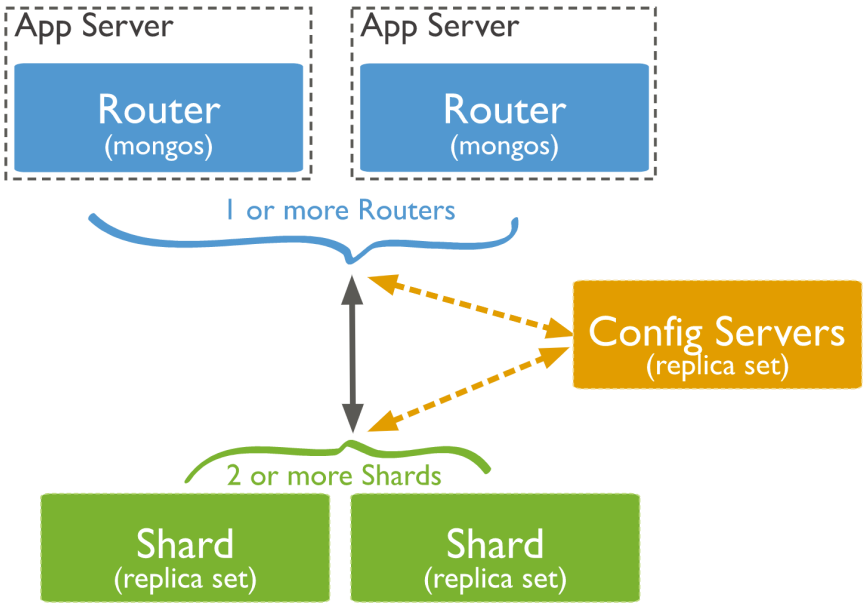

Figura 4.5: MongoDB em modo Shard.[a]

[a]Disponível em: https://docs.mongodb.com/manual/sharding/

Os resultados obtidos nos testes realizados com esta configuração, demonstram que não foi obtida melhoria na capacidade de resposta do MongoDB, pelo contrário a performance piorou. No cluster com os nós configurados com 2GB de memória RAM, a plataforma teve a capacidade máxima de processar 80 mensagens por segundo, e a replicação do componente Orion não trouxe melhorias significativas. No cluster com os nós configurados com 4GB de memória RAM, a plataforma foi capaz de processar

[3]Disponível em:https://github.com/bitnami/charts/tree/master/bitnami/mongodb-sharded/
[4]Disponível em:https://helm.sh/docs/

até 110 mensagens por segundo, e a replicação do Orion nesta configuração, também não revelou melhoria de desempenho.

Neste cenário, a falha observada é geralmente provocada por um dos subcomponentes do MongoDB, tipicamente um dos Shards, entrando em sobrecarga e provocando a falha do componente MongoDB em si. A execução do MongoDB em modo Shard revelou alguns aspetos negativos em caso de falha de um dos subcomponentes, pois a recuperação do componente MongoDB é bastante mais demorada e existe um efeito sistémico com a falha de um subcomponente a provoca falhas de alguns dos outros subcomponentes, pois tipicamente observa-se que falha de um dos Shards, leva à falha do subcomponente MongoS também, e apesar dos restantes componentes manterem-se em execução, o serviço prestado pelo MongoDB ficar indisponível. É observada uma demora de vários minutos na recuperação do componente MongoDB (recuperação de todos os contentores dos seus subcomponentes em falha), tipicamente entre 5 a 10 minutos. Pensamos que a demora seja devido a processos internos de recuperação após os contentores em falha retornarem à execução, mas tal suposição carece de mais análise e estudo, para devida confirmação.

Na Figura 4.6 verificamos que quando ocorre falha, existe um decréscimo de desempenho quanto maior for o número de mensagens enviadas por segundo no teste realizado. Como o número de mensagens enviadas por segundo é superior, a falha ocorre mais cedo, e dada a demora do componente MongoDB a recuperar, tipicamente o período no qual aceita pedidos de persistência também é menor.

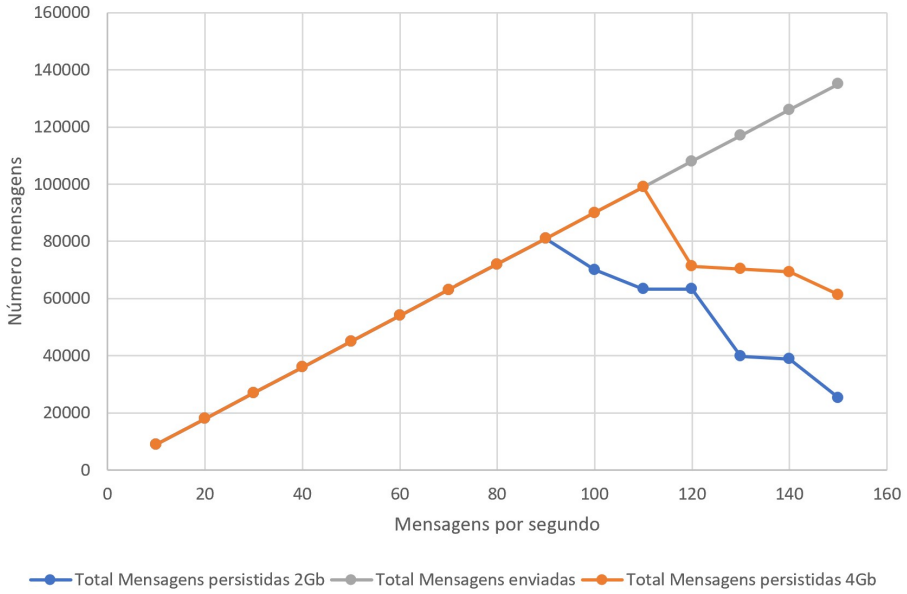

Figura 4.6: Resultados em ambiente com MongoDB em modo Shard.

4.1.2.3 Componente MongoDB, configurado em modo Shard com replicação dos seus subcomponentes

Como cenário final nos testes realizados, foi utilizado novamente o componente MongoDB em modo Shard e a executar no cluster Kubernetes, mas agora com os subcomponentes do MongoDB replicados. O chart Helm usado para instanciar o MongoDB em modo Shard, também suporta parametrização no qual é possível definir o número de replicas para cada um dos subcomponentes. Na parametrização usada foram definidos os seguintes números de replicas para cada subcomponente, duas réplicas do subcomponente Router(mongos), duas réplicas do subcomponente ConfigServer e duas réplicas para cada um dos dois Shards usados.

Esta configuração procura verificar se a replicação de subcomponentes do MongoDB, traz alguma melhoria de disponibilidade em caso de falha de algum subcomponente, ou seja, se caso um subcomponente falhe, a sua réplica assegurara o serviço prestado, evitando ou adiando de forma significativa a falha do componente MongoDB em si.

Após realizados os testes, verificamos que esta configuração não trouxe melhorias, bem pelo contrário, tendo-se registado o pior desempenho. O cluster configurado com 2GB de RAM em cada nó, apenas permitiu que fossem processadas até 10 mensagens por segundo, sem perda de mensagens. Com o cluster configurado com 4GB de RAM em cada nó, a plataforma melhorou um pouco a sua capacidade de processar para até 30 mensagens por segundo, sem perda de mensagens.

A expectativa de melhoria de disponibilidade não se verificou, ocorrendo o oposto, pois a sobrecarga provocada pelos recursos que o MongoDB necessita para gestão das diferentes réplicas e o peso de existirem mais contentores em execução a consumirem recursos no cluster, provoca a rápida exaustão e pressão sobre a memória RAM nos nós do cluster, despoletando processos de swap de memória RAM para discos, o qual pressiona a capacidade de processamento do CPU e capacidade de armazenamento também.

A Figura 4.7 revela os resultados obtidos neste cenário e a quebra muito significativa de performance quando existe falha neste cenário.

4.2 Resumo dos resultados obtidos

Dos resultados obtidos nos diferentes cenários e configurações testadas, é possível verificar que apesar da utilização do componente MongoDB em modo Shard permite a utilização de recursos computacionais de diferentes nós do cluster Kubernetes para

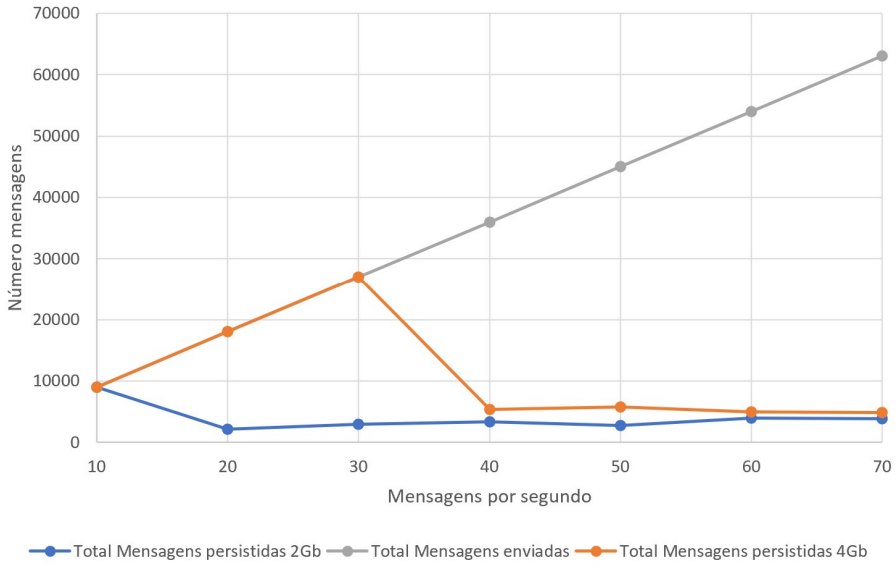

Figura 4.7: Resultados em ambiente com MongoDB em modo Shard, com replicação de componentes que constituem o MongoDB.

execução das tarefas deste componente, não houve melhoria dos resultados obtidos, pelo contrário, pois o peso dos protocolos usados internamente e o tráfego de rede gerado pelo MongoDB para coordenação dos seus componentes, tornaram esta configuração uma má opção, em máquinas com recursos computacionais de dimensão semelhante aos aqui testados. Quando adicionado réplicas aos subcomponentes do MongoDB em modo Shard, o desempenho ainda piora mais, pois os recursos necessários para coordenar e gerir os diferentes subcomponentes do MongoDB em modo Shard com réplicas são superiores, levado à rápida exaustão da capacidade dos nós do cluster Kubernetes.

Era esperado uma melhoria no desempenho da plataforma com o MongoDB em modo Shard, mas a forma como foi usada a persistência de dados nos testes realizados, invalidou e ainda agravou os resultados obtidos.

Como foi usado apenas um serviço NFS, a carga no acesso ao serviço de persistência incidiu apenas numa máquina, basicamente inutilizado as possíveis vantagens do modo Shard do MongoDB.

Na Tabela 4.1 é feito um resumo dos testes realizados, caracterizado os cenários usados e melhores resultados obtidos em cada cenário. Na primeira coluna da tabela é indicado o identificador do cenário testado, na segunda coluna a memória RAM das máquinas usadas, na terceira coluna é indicado se o cenário foi realizado em cluster Kubernetes, na quarta coluna é indicado se o componente MongoDB está configurado

Id Cenário	Host RAM	Kubernetes Cluster	Mongo modo Shard	Número Replicas Mongo Shard	Número Replicas Orion	msg/s
D1	2 GB	F	NA	NA	NA	70
K1a	2 GB	V	F	NA	1	80
K1b	2 GB	V	F	NA	2	80
K2a	2 GB	V	V	1	1	50
K2b	2 GB	V	V	1	2	50
K3a	2 GB	V	V	2	1	10
D2	4 GB	F	NA	NA	NA	110
K1c	**4 GB**	**V**	**F**	**NA**	**1**	**165**
K1d	**4 GB**	**V**	**F**	**NA**	**2**	**165**
K2c	4 GB	V	V	1	1	90
K2d	4 GB	V	V	1	2	100
K3b	4 GB	V	V	2	1	30

Tabela 4.1: Número máximo de mensagens por segundo processadas sem ocorrência de falhas nos diferentes cenários de teste realizados.

em modo Shard, na quinta coluna é indicado quantas replicas foram usadas nos sub-componentes do MongoDB, na sexta coluna é indicado quantas réplicas dos Orion foram usadas e por fim na sétima e última coluna qual o valor máximo de mensagens enviadas por segundo, que a plataforma reteve sem falhas. A negrito é evidenciado os melhores resultados nos diferentes cenários testados.

Dos resultados observados, a configuração que obteve os melhores resultados, foi o cenário de testes com a utilização do cluster Kubernetes, com o componente MongoDB definido como um objeto StatefulSet, a executar em máquinas com 4Gb de memória RAM.

Podemos concluir que a memória RAM disponível nos nós do cluster Kubernetes onde executam os contentores, influência o desempenho da solução. A opção de configurar o MongoDB em modo Shard, não revelou ser eficaz, levando a perda de performance da plataforma.

5

Conclusões

Neste capítulo são apresentadas as conclusões do trabalho realizado, bem como algumas propostas de desenvolvimento futuro.

5.1 Conclusões

O projeto apresentado foi realizado com o objetivo de desenhar e implementar uma arquitetura descentralizada de código aberto, que permita monitorizar dispositivos IoT. Como exemplo motivador usou-se o problema poluição atmosférica e de sensores de observação da qualidade do ar.

A plataforma desenvolvida tem como objetivo permitir o aprovisionamento de dispositivos IoT por parte dos utilizadores e disponibilizar um *dashboard* com um mapa geográfico, que permita visualizar os dispositivos já aprovisionados, consultar o respetivo estado dos dispositivos, bem como permitir analisar indicadores relativos às métricas observadas e enviadas pelos dispositivos.

Sendo um projeto orientado a auxiliar o desenvolvimento de cidades inteligentes, foram utilizadas algumas tecnologias de suporte a projetos com esta temática, nomeadamente foi utilizada a *framework* FIWARE, que facilita o desenvolvimento de aplicações e serviços, disponibilizando diversos componentes que permitem fácil integração de componente, e auxiliam a realização de tarefas orientadas à troca de informação de contexto. Também foi utilizada a plataforma TTN, que permite o acesso e interação

com dispositivos sobre uma rede móvel de grande alcance e de muito baixa potência — uma rede LoRaWAN.

Para a arquitetura definida foram também implementados componentes com funcionalidade mais específica, os quais são disponibilizados para a comunidade em repositório público de acesso livre[1]. Foi implementado um componente que possibilita a interação e coordenação dos componentes da plataforma com entidades externas, nomeadamente a plataforma TTN V3, na coordenação para o aprovisionamento e remoção de dispositivos IoT, de modo que informação relativa aos dispositivos seja consistente entre a plataforma e a rede de comunicação suporte sobre os quais operam. Também foi implementado um serviço Web que disponibiliza uma interface com o utilizador, permitindo a estes interagir com a plataforma apresentada.

A instanciação da plataforma é suporta através da disponibilização dos diferentes componentes, em imagens de contentores de execução virtual. Tal permite a execução de forma descentralizada, tanto em infraestrutura local, como em nuvem. Para facilitar a instanciação da plataforma e os seus diferentes componentes, foram desenvolvidas duas formas de instanciação. Uma forma que permite a instanciação numa máquina com motor de execução de contentores Docker, sendo criado um *script* que realiza o *download* das imagens dos contentores dos componente e instancia os mesmos. Outra forma permite a instanciação num cluster Kubernetes. Tal permite que a plataforma apresentada possa ser instanciada com a execução de um simples comando, muito embora possa ser necessário alteração de alguns parâmetros, nos manifestos definidos para instanciação.

Apesar de esforço realizado para que a instanciação seja o mais simples possível, a utilização da TTN como infraestrutura de suporte acarreta algumas ações necessárias, para que a interação seja possível, nomeadamente é necessária uma conta de utilizador que permita o acesso à TTN. É também necessário a definição de uma aplicação na sua consola e obtenção de tokens, que permitam o acesso programático via API. Também é necessário definir um script de descodificação das mensagens oriundas dos dispositivos, antes de serem encaminhas para integração com outros sistemas. Aliás, esta é uma das desvantagens na utilização da plataforma TTN.

O desempenho da plataforma foi aferido em testes de carga que simulam o envio de mensagens em massa por parte dos dispositivos. Estes cenários foram desenhados como pior caso, garantindo que a utilização da plataforma terá menos carga. No entanto os resultados mostraram que o número de mensagens processadas por segundo não é muito alto. Ficou determinado que a persistência dos dados, nomeadamente o

[1] https://github.com/IvoPedroso?tab=repositories

armazenamento em base de dados MongoDB, é a principal causa que estrangula a capacidade de resposta na atualização da plataforma com os dados transmitidos pelos dispositivos.

Como conclusão final, a plataforma apresentada atingiu todos os objetivos do trabalho, havendo no entanto, margem para melhoramentos e incremento de funcionalidades.

5.2 Publicações

Como resultado do trabalho desenvolvido, foi publicado e apresentado um artigo no 12º simpósio de informática — INForum 2021, intitulado, intitulado "Plataforma FIWARE para monitorização de poluição" [16]. Foi submetido e aceite como *full paper* na *track* CMU – Computação Móvel e Ubíqua.

5.3 Trabalho futuro

No projeto apresentado, foram identificadas algumas possibilidades de trabalho futuro, quer em incremento de funcionalidade, quer em melhoria de automatização de processos e na melhoria de formas de visualização dos dados apresentados ao utilizador.

No campo do incremento de funcionalidade, fica a possibilidade do desenvolvimento de uma interface gráfica que gere códigos QR, os quais possam ser impressos e colados aos dispositivos por aprovisionar, permitindo que no terreno se aprovisione os dispositivos no momento da instalação física, usando um simples smartphone que ao digitalizar o código QR, faça um pedido HTTP a realizar o aprovisionamento.

Na demonstração de resultados, também se propõem uma melhoria da interface Web, nomeadamente a criação de camadas de visualização no mapa geográfico apresentado, que para dado dispositivo selecionado, identifique as localizações nas quais os dispositivos registou medições e quais os valores observados.

No campo da melhoria da automatização de processos, seria possível modificar o componente IoT Agent LoRaWAN ou implementar um componente alternativo, que consiga receber mensagens da TTN com os dados enviados pelos dispositivos IoT, sem necessidade de serem descodificados na TTN. Tal funcionalidade, faria que a configuração inicial necessária para instanciar de raiz a plataforma apresentada, apenas necessitasse da criação de uma conta na TTN e respetivos tokens de acesso.

Como trabalho futuro também e com disponibilidade de recursos, seria um complemento aos testes realizados à plataforma, a realização dos mesmos testes, em máquinas com memória RAM bastante superior (ex:16GB ou 32GB) e com meios de persistência mais eficientes, para verificar se, com mais recursos disponível nas máquinas onde os contentores executam, os benefícios aparentes da utilização do componente MongoDB em modo Shard, emergem e a solução melhora a sua capacidade de resposta com o escalonamento horizontal. Também se propõe a utilização de diversos serviços de persistência em máquinas diferentes, no mínimo em igual número ao número de Shards usado na configuração do componente MongoDB, e validar se existe melhoria de performance em testes com igual cenário.

Referências

[1] Yimeng Liu, Yizhi Wang & Yi Jin, "Research on the improvement of mongodb auto-sharding in cloud environment", em *2012 7th International Conference on Computer Science Education (ICCSE)*, 2012, páginas 851–854. DOI: 10.1109/ICCSE.2012.6295203.

[2] et al Bradshaw Shannon, *MongoDB: The Definitive Guide, 3rd Edition*. O'Reilly, 2019, ISBN: 9781491954461.

[3] Bin Cheng, Gürkan Solmaz, Flavio Cirillo, Ernö Kovacs, Kazuyuki Terasawa & Atsushi Kitazawa, "Fogflow: Easy programming of iot services over cloud and edges for smart cities", *IEEE Internet of Things Journal*, vol. 5, n.º 2, páginas 696–707, 2017.

[4] Sourangsu Chowdhury & Sagnik Dey, "Cause-specific premature death from ambient pm2.5 exposure in India: Estimate adjusted for baseline mortality", *Environment International*, vol. 91, páginas 283–290, 2016.

[5] FIWARE Foundation, *A curated framework of open source platform components to accelerate the development of smart solutions*,
online: https://www.fiware.org, 2021.

[6] *Developers catalogue*, 2019. URL: https://www.fiware.org/developers/catalogue/.

[7] Walter Fuertes, Diego Carrera, César Villacís, Theofilos Toulkeridis, Fernando Galárraga, Edgar Torres & Hernán Aules, "Distributed system as internet of things for a new low-cost, air pollution wireless monitoring on real time", em *2015 IEEE/ACM 19th International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*, IEEE, 2015, páginas 58–67.

[8] (Mar. de 2021). "Keda - kubernetes event-driven autoscaling", URL: https://keda.sh/.

[9] (Mar. de 2021). "Lorawan", URL: https://lora-alliance.org/about-lorawan/.

[10] Nuno Cruz, Nuno Cota & João Tremoceiro, "Lorawan and urban waste management—a trial", *Sensors*, vol. 21, n.º 6, 2021, ISSN: 1424-8220. DOI: 10.3390/s21062142. URL: https://www.mdpi.com/1424-8220/21/6/2142.

[11] M. Luksa, *Kubernetes in Action*. Manning, 2018, ISBN: 9781617293726. URL: https://books.google.pt/books?id=8bE5MQAACAAJ.

[12] Telecom Italia, Telefónica I+D & NEC, *Ngsiv2 api specification (v2.0)*, online: https://fiware.github.io/specifications/ngsiv2/stable/, 2018.

[13] (Mar. de 2021). "Considerations on ngsiv1 and ngsiv2 coexistence", URL: https://fiware-orion.readthedocs.io/en/1.15.0/user/v1_v2_coexistence/index.html.

[14] OMS. (fev. de 2021). "Ambient air pollution - a major threat to health and climate", URL: https://www.who.int/airpollution/ambient/en/.

[15] Brendan Burns, *Designing Distributed Systems: Patterns and Paradigms for Scalable, Reliable Services*, 1st. O'Reilly Media, Inc., 2018, ISBN: 1491983647.

[16] Ivo Pedroso, Nuno Datia & Nuno Cruz, "Plataforma fiware para monitorização de poluição", *Inforum-Simpósio de Informática*, 2021. URL: https://www.researchgate.net/publication/354603541_Plataforma_FIWARE_para_monitorizacao_de_poluicao.

[17] (Mar. de 2021). "Publisher-subscriber pattern", URL: https://docs.microsoft.com/en-us/azure/architecture/patterns/publisher-subscriber.

[18] (Mar. de 2021). "Rede de sensores monitoriza a qualidade do ar, níveis de ruído e trânsito", URL: https://www.lisboa.pt/atualidade/noticias/detalhe/rede-de-sensores-monitoriza-a-qualidade-do-ar-niveis-de-ruido-e-transito.

[19] W. Shi & S. Dustdar, "The promise of edge computing", *Computer*, vol. 49, n.º 5, páginas 78–81, 2016. DOI: 10.1109/MC.2016.145.

[20] (Mar. de 2021). "Smart data model", URL: https://www.fiware.org/developers/smart-data-models/.

[21] Dipa Soni & Ashwin Makwana, "A survey on mqtt: A protocol of internet of things (iot)", em *International Conference On Telecommunication, Power Analysis And Computing Techniques (ICTPACT-2017)*, vol. 20, 2017.

[22] Nicolas Sornin, Miguel Luis, Thomas Eirich, Thorsten Kramp & Olivier Hersent, "Lorawan specification", *LoRa alliance*, 2015.

[23] Ruben Taborda, Nuno Datia, M.P.M. Pato & João Moura Pires, "Exploring air quality using a multiple spatial resolution dashboard — a case study in lisbon", em *2020 24th International Conference Information Visualisation (IV)*, 2020, páginas 140–145. DOI: 10.1109/IV51561.2020.00032.

[24] Ying Xiong, Yulin Sun, Li Xing & Ying Huang, "Extend cloud to edge with kubeedge", em *2018 IEEE/ACM Symposium on Edge Computing (SEC)*, IEEE, 2018, páginas 373–377.