Capítulo 1

Plataformas de *Fog Computing*: da Teoria à Prática

Thiago P. Silva (UFMT), Aluizio Rocha (UFRN), Thais Batista (UFRN), Frederico Lopes (UFRN), Flávia C. Delicato (UFF), Paulo F. Pires (UFF)

Abstract

Nowadays we are witnessing a significant increase in the amount of IoT devices and the volume of data generated by these devices. Associated with this, more IoT applications have ultra low latency and response time requirements, such that cloud-centric IoT platforms are unable to address these requirements and to adequately handle the increasing volume of data. To support these new applications and handle the increasing volume of IoT data, Fog Computing platforms have been proposed. The Fog Computing paradigm is characterized by a horizontal, system-level architecture where devices that are in the path of IoT data, from its origin in IoT devices to the cloud, are used for processing distribution, storage, and data control. Fog Computing platforms aim to abstract the highly dynamic and heterogeneous environment where they are inserted, in order to facilitate the development of applications by users. This chapter has the following objectives: (i) to present the Fog Computing paradigm; (ii) to survey and discuss the main requirements that guide the development of Fog Computing platforms; (iii) to discuss two proposals of reference architecture for Fog Computing; (iv) to describe the main Fog Computing platforms and contrast them with the requirements previously raised; (v) to demonstrate in a practical way the use of a Fog Computing platform for the context of smart cities.

Resumo

Atualmente presenciamos um aumento significativo na quantidade de dispositivos IoT e no volume de dados gerados por estes dispositivos. Associado a isto, cada vez mais aplicações IoT possuem requisitos de latência e tempo de resposta ultra baixo, de tal forma que as plataformas IoT centradas na nuvem não conseguem endereçar estes requisitos e tão pouco tratar de maneira adequada o volume crescente de dados. Para suportar estas novas aplicações e tratar o volume crescente de dados IoT, plataformas que empregam

o paradigma Fog Computing têm sido propostas. O paradigma Fog Computing é caracterizado por uma arquitetura horizontal em nível de sistema onde os dispositivos que estão no caminho percorrido pelos dados IoT, desde sua origem nos dispositivos IoT até a nuvem, são utilizados para distribuição de processamento, armazenamento e controle de dados. As plataformas que incorporam o paradigma Fog Computing visam abstrair o ambiente altamente dinâmico e heterogêneo onde estão inseridas, de forma a facilitar o desenvolvimento de aplicações. Este capítulo tem como objetivos: (i) apresentar o paradigma Fog Computing; (ii) levantar e discutir os principais requisitos que norteiam o desenvolvimento das plataformas Fog Computing; (iii) discutir duas propostas de arquitetura de referência para Fog Computing; (iv) descrever as principais plataformas Fog Computing e contrapô-las aos requisitos levantados; (v) demonstrar, de forma prática, o uso de uma plataforma Fog Computing para o contexto de cidades inteligentes.

1.1. Introdução

A Internet das Coisas (do inglês *Internet of Things – IoT*) [1] é um paradigma de computação que engloba *software*, *hardware* e serviços visando interconectar objetos físicos com a infraestrutura de comunicação da Internet. Tais objetos, comumente chamados de dispositivos IoT, são dotados de sensores e atuadores e possuem, em sua maioria, baixo poder de processamento e armazenamento. Entretanto, quando estes objetos são interconectados, eles moldam uma rede de dispositivos inteligentes capaz de realizar vários tipos de processamentos, capturar variáveis ambientais e reagir a estímulos externos, fornecendo serviços de valor agregado ao usuário. Atualmente existem diferentes fabricantes de dispositivos IoT e uma infinidade de sensores e atuadores, que possuem diferentes capacidades de monitorar e controlar remotamente o ambiente em que estão inseridos. Desta forma, a interconexão dos dispositivos IoT, associada ao barateamento da tecnologia e a sua capacidade de promover inovação, impulsionam a criação de aplicações interessantes em vários domínios.

Dentre os diversos domínios possíveis de uso da tecnologia IoT, destaca-se o domínio das cidades inteligentes, no qual o uso de tecnologias avançadas de comunicação e sensoriamento visa prover serviços de valor agregado para os órgãos administrativos das cidades e para seus cidadãos [2]. Ainda no domínio de cidade inteligente, a tecnologia IoT, conforme aponta [3], forma a infraestrutura de sensores necessária para que as cidades melhorem sua eficiência operacional e forneçam serviços de qualidade, os quais impactam positivamente no bem-estar dos cidadãos. Entretanto, a inerente heterogeneidade dos ambientes de IoT requer soluções que permitam a interoperabilidade e integração dos diversos componentes que fazem parte desse ambiente. Neste sentido, naturalmente plataformas de *middleware* surgiram como solução para prover interoperabilidade e gerenciar a crescente variedade de dispositivos associados a aplicações e o consumo de dados por parte dos usuários finais [4, 5]. A maioria das plataformas para IoT são centradas na nuvem e utilizam o paradigma *Cloud of Things* (CoT), que preconiza a integração sinergética entre IoT e a nuvem. Em sistemas de CoT, tipicamente os dados gerados por dispositivos IoT fluem continuamente através da infraestrutura de comunicação da Internet até chegar à nuvem, onde são processados e armazenados [6].

Atualmente presenciamos um aumento significativo na quantidade de dispositivos IoT, de modo que, em um futuro próximo, o volume de dados gerados por eles não

poderá ser tratado adequadamente por plataformas centradas na nuvem. Além disso, conforme [7, 8] argumentam, soluções IoT centralizadas na nuvem não apresentam modelos econômicos e sustentáveis, uma vez que o custo de largura de banda e armazenamento pode ser substancialmente alto ao enviar todos os dados IoT para a nuvem. Ademais aos problemas supracitados, a tecnologia IoT impulsionou o surgimento de aplicações e serviços que requerem latência e tempo de resposta ultra baixo e, portanto, plataformas centradas na nuvem nem sempre podem suprir as necessidades deste tipo de aplicação. Por exemplo, algumas aplicações em que milissegundos têm um impacto significativo em sua execução, como na comunicação veículo-veículo, para evitar colisões ou acidentes, não são adequadas para uma arquitetura centralizada como a proposta da CoT [9].

Os novos desafios impostos pelas aplicações emergentes mobilizaram a academia e indústria a discutirem soluções viáveis para lidar com tais desafios e atingir o pleno potencial da IoT. Uma nova tendência atualmente é migrar a computação - seja ela na forma de processamento ou armazenamento - da camada de nuvem para as camadas próximas aos dispositivos IoT. Esse novo paradigma é chamado de *fog computing* e é caracterizado por uma arquitetura horizontal em nível de sistema que contrasta com o modelo CoT [10]. No paradigma *fog computing*, os dispositivos que estão no caminho (chamado de *Cloud-to-Thing Continuum*) percorrido pelos dados IoT, desde sua origem nos dispositivos finais IoT até a nuvem, são utilizados para distribuição de processamento, armazenamento e controle de dados [8]. Os dispositivos que estão no *Cloud-to-Thing Continuum* são chamados de nós *fog* (do inglês, *fog node*) e incluem roteadores, pontos de acesso, *gateways*, entre outros dispositivos que, atualmente, possuem maior capacidade de armazenamento e processamento computacional que seus antecessores.

O desenvolvimento de plataformas que incorporam o paradigma de *fog computing* é uma área de pesquisa recente e em evolução. Isto posto, o paradigma apresenta muitos desafios não superados e várias ideias ainda estão em processo de elaboração e desenvolvimento [11, 3, 12]. Como consequência do grau de incipiência da tecnologia, a maioria dos trabalhos publicados na literatura que abordam o tema *fog computing* são de cunho teórico. Porém, o consórcio OpenFog, formado por várias universidades e grandes empresas da área de comunicação e tecnologia, numa tentativa de progresso da adoção e padronização desse paradigma, criou uma Arquitetura de Referência (AR) para *fog computing*. Tal arquitetura é formada por oito pilares que representam as principais características de um sistema que visa fornecer a distribuição das funções de computação, armazenamento, controle e rede mais próximas à fonte de dados ao longo do *Cloud-to-Thing Continuum*. Em julho de 2018, o IEEE adotou a arquitetura de referência como o primeiro padrão para *fog computing* [13].

Apesar dos esforços do consórcio OpenFog e IEEE, observa-se que as plataformas existentes apresentam características distintas e muitas vezes não alinhadas aos pilares estabelecidos pela arquitetura de referência, desta forma não contribuindo para a interoperabilidade e formando silos isolados. Associado a isto, ainda não existe um consenso a respeito da definição do termo *fog computing* e, por vezes, o termo é usado de forma intercambiável com *Edge Computing*. Considerando a relevância do paradigma e o potencial de aplicabilidade no contexto de cidades inteligentes, de forma a criar plataformas disruptivas, este capítulo tem como objetivos: i) apresentar os principais conceitos do paradigma bem como os requisitos de *software* que direcionam o projeto de plataformas

fog computing; ii) apresentar e discutir as principais arquiteturas de referência publicadas na literatura; iii) apresentar quatro plataformas *fog computing* com ênfase em cidades inteligentes; iv) demonstrar um estudo de caso implementado em um plataforma alvo.

Neste capítulo, para facilitar o entendimento a respeito do paradigma, adotaremos a definição proposta pelo modelo conceitual de *fog computing* criado pelo NIST [10]. Tal definição está alinhada ao propósito do OpenFog e considera *edge computing* apenas como a camada que compreende os dispositivos finais e usuários, tipicamente chamada de borda da rede (do inglês, *edge network)*. Já *fog computing* oferece uma arquitetura multi-camadas organizada de forma hierárquica para a execução de aplicações no *Cloud-to-Thing Continuum*. Sendo assim, segundo essa visão *edge computing* limita-se a execução de aplicações específicas em uma rede fixa formada por um conjunto limitado de dispositivos acessíveis de forma direta que estão na mesma rede, desconsiderando a nuvem. Por outro lado, *fog computing* utiliza, inclusive, os dispositivos da borda da rede, mas engloba esse continuum desde os dispositivos produtores de dados até a nuvem.

Outra definição importante adotada neste texto refere-se a distinção entre dispositivos IoT e os dispositivos nós *fog*. Consideramos o dispositivo IoT um *hardware* que tem sensores e atuadores a ele acoplado, tendo as capacidades de sensoriamento e atuação no ambiente em que está inserido. Tipicamente, os dispositivos IoT são os produtores de dados enquanto os nós *fog* são organizados e interconectados formando uma rede de nós *fog* (do inglês *fog network*) para processar e/ou armazenar estes dados. Neste sentido, a Internet das Coisas proporciona a interconexão dos dispositivos IoT entre e (direta ou indiretamente) com a Internet, permitindo a criação de serviços/aplicações de valor agregado [14]. Por outro lado, um nó *fog* é um dispositivo que tem capacidades de computação, armazenamento e rede, essenciais para a execução de aplicações IoT [15]. Apesar do fato de que atualmente dispositivos IoT como, por exemplo, Raspberry e BeagleBone, também podem atuar como nós *fog*, ao longo do texto deixaremos claro o papel que estes dispositivos exercem, de forma a não confundir o leitor.

O restante deste capítulo está estruturado da seguinte forma: a Seção 1.2 apresenta os principais requisitos que direcionam o desenvolvimento de plataformas de *fog computing*; a Seção 1.3 discute as principais propostas de arquitetura de referência para área; a Seção 1.4 apresenta as plataformas *fog computing* de código aberto e uma comparação entre elas é proposta; na Seção 1.5 será apresentado um estudo de caso implementado na plataforma FogFlow para a detecção e identificação facial utilizando-se processamento de vídeo em dispositivos de baixo custo; e por fim, a Seção 1.6 apresenta as conclusões e perspectivas futuras.

1.2. Requisitos de plataformas de *fog computing*

Esta seção apresenta os principais requisitos que direcionam a construção de plataformas de *fog computing*. Tais requisitos visam atender as necessidades das aplicações e dos usuários, e foram levantados a partir da análise dos principais trabalhos publicados na área [16, 17, 18, 19, 20, 21, 22, 9, 12, 18, 23, 24]. Cabe ressaltar que o conjunto de requisitos que serão apresentados determinam o que uma plataforma *fog computing* de propósito geral necessita prover/atender. Entretanto, o que observamos na prática, é que as plataformas atuais são em sua maioria específicas de domínio e, por conseguinte,

possuem diferentes características e atendem também a requisitos inerentes ao domínio onde estão inseridas. Os seguintes requisitos serão aqui apresentados: 1) Distribuição Geográfica; 2) Suporte a aplicações sensíveis a latência; 3) Suporte a Heterogeneidade; 4) Interoperabilidade; 5) Otimização do uso da largura de banda; 6) Ciência de Localização; 7) Ciência de Contexto; 8) Suporte a Mobilidade; 9) Disponibilidade; 10) Eficiência Energética; 11) Segurança e Privacidade; 12) Escalabilidade.

O requisito de **distribuição geográfica** diz respeito à distribuição em diferentes localizações geográficas dos nós *fog* que a plataforma necessita gerenciar para fornecer a distribuição das funções de computação, armazenamento, controle e rede mais próximas à fonte de dados e/ou usuários. No paradigma *fog computing*, em contraste com a computação em nuvem tradicional, os recursos e serviços estão distribuídos em posições estratégicas em uma área geográfica potencialmente ampla, para atender a uma grande variedade de usuários [18]. Desta forma, segundo [25], as plataformas precisam escolher de forma adequada os nós *fog* que executarão as aplicações, levando em consideração não somente as posições geográficas, mas também os critérios de QoS (Qualidade de Serviço, do inglês *Quality of Service*) demandados pela aplicação. Tal escolha precisa ser transparente para as aplicações e usuários, i.e., os usuários não precisam saber da localização onde suas aplicações serão executadas.

Um dos principais objetivos de *fog computing* é reduzir a latência, de forma a **habilitar aplicações ultra sensíveis à latência**. Este tipo de aplicação requer baixa latência fim-a-fim e, portanto, os componentes ou unidades de processamentos que compõem tais aplicações não podem ser completamente executados na nuvem. Por exemplo, jogos de realidade alternativa (do inglês, *reality games*) requerem latência ultra baixa, menores que 20 ms, conforme aponta [18]. Outro exemplo, no contexto de veículos conectados (do inglês *connected vehicles*), são as aplicações de segurança como, por exemplo, aviso de violação de sinal de trânsito, aviso colaborativo de colisão frontal e aviso de mudança de faixa de rolagem. Tais aplicações requerem latência menor que 100 ms, conforme aponta [26]. Desta forma, nas aplicações exemplificadas, os dados oriundos de dispositivos finais (aqueles localizados na borda da rede) precisam alcançar o local onde serão processados e/ou armazenados, e uma resposta deverá ser dada em um intervalo de tempo ultra baixo. Ao considerarmos a computação em nuvem, normalmente a latência entre o usuário final e a nuvem é em torno de 20 a 40 ms em redes cabeadas, atingindo 150 ms em redes 4G. Além da latência na comunicação, deve-se considerar também o problema de congestionamento da rede que é mais comum em soluções centradas na nuvem, pois a nuvem está a vários saltos (*hops*) de distância dos dispositivos produtores de dados e o dados precisam percorrer diferentes enlaces que são compartilhados por diversos usuários e aplicações.

Apenas a distribuição geográfica dos nós *fog* não é suficiente para assegurar baixa latência, pois uma característica importante de *fog computing* é usar nós *fog* heterogêneos, que possuem recursos variados e são interligados por diferentes meios de comunicação (cabo, rede sem fio, etc). Portanto, as plataformas também devem considerar a velocidade dos links que interligam os nós *fog*, de forma a distribuir o processamento e/ou armazenamento nos nós das camadas mais próximas ao usuário final e/ou fonte de dados, considerando o link de comunicação entre eles. Além dos nós *fog* heterogêneos, a plataforma também precisa lidar com diferentes tipos de dispositivos IoT, com seus sensores e atuadores que apresentam capacidades, protocolos de comunicação e representação de

dados distintas. Portanto, o requisito de **suporte a heterogeneidade** diz respeito à capacidade da plataforma de assegurar o devido gerenciamento e coordenação da rede dos dispositivos IoT, bem como os nós *fog* heterogêneos, de modo a oferecer uma abstração apropriada para ocultar os detalhes de heterogeneidade dos recursos. Tal requisito é essencial para evitar que os desenvolvedores de aplicações tenham que lidar diretamente com os detalhes técnicos de cada dispositivo IoT e nós *fog* que executarão as aplicações.

É vital garantir que os componentes de uma plataforma *fog computing* forneçam **interoperabilidade** para liberar o potencial da IoT. Conforme discute [14], as redes IoT são largamente distribuídas, heterogêneas e oferecem vários tipos de serviços (do inglês *multi-services*). Consequentemente, o risco de não-interoperabilidade é alto, levando a indisponibilidade de alguns serviços para os usuários/aplicações finais ou a perda de informações importantes fora da IoT devido a falta de interoperabilidade. Ainda segundo [14], existem alguns desafios para interoperabilidade na IoT: i) a coexistência de muitos dispositivos no ambiente que precisam se comunicar e trocar informações; ii) muitos dispositivos são concebidos por diferentes fabricantes para diferentes propósitos, visando diversos domínios de aplicação; iii) a dinamicidade do ambiente, pois dispositivos que suportam novas capacidades e utilizam protocolos imprevistos, mas que precisam se comunicar e compartilhar dados na IoT, podem entrar e sair do ambiente; iv) a existência de muitos formatos de dados que necessitam seguir os mesmo princípios de modelagem e podem ser inter-relacionados. A interoperabilidade é, muitas vezes, centrada em protocolos de comunicação, tipicamente M2M (do inglês *Machine-to-Machine*), e na infraestrutura necessária para que esses protocolos funcionem. Desta forma, os componentes de *hardware/software* e serviços de uma plataforma *fog computing* precisam implementar os protocolos desenvolvidos por órgãos de padronização como, por exemplo IEEE e ISO, de forma a promover a comunicação entre os dispositivos e uniformizar os procedimentos de conectividade e troca de dados, promovendo a interoperabilidade. Conforme aponta o consórcio OpenFog [13], a interoperabilidade é essencial para o sucesso de um ecossistema formado por plataformas *fog computing* e aplicações IoT, evitando o surgimento de silos isolados.

As aplicações que produzem grande volume de dados como, por exemplo, videomonitoramento (do inglês *video surveillance*), tipicamente necessitam de grande largura de banda, e soluções centradas na nuvem requerem que todo o *stream* de vídeo seja enviado para nuvem para que o processamento adequado possa ser realizado. Entretanto, um cenário onde uma câmera digital captura vídeo em alta definição (HD, do inglês *High Definition*) requer no mínimo 1 Mbps de *bitrate* e no mínimo 2 Mbps de taxa de *upload*. Já uma transmissão em Full-HD precisa de no mínimo 3 Mbps de *bitrate* e 6 Mbps de taxa de *upload*[1]. Uma aplicação para videomonitoramento pode ser executada por uma plataforma *fog computing* utilizando-se os dispositivos computacionais da borda da rede que estão próximos à fonte de *streaming* de vídeo. Desta forma, pode-se **otimizar o uso da largura de banda**, na medida que o dado bruto poderá ser tratado na *fog* e, por exemplo, apenas um subconjunto de informações necessitarão ser transmitidas para a nuvem.

Para endereçar o requisito de **ciência de localização**, a plataforma precisa conhecer a localização física e topológica dos dispositivos IoT e dos nós *fog*. Considerando um

[1] Cálculo de bitrate - https://toolstud.io/video/bitrate.php

cenário de cidade inteligente, onde nós *fog* estão espalhados e podem se tornar indisponíveis a qualquer momento, a plataforma precisa fazer uso de estratégias de descoberta e gerenciamento de dispositivos presentes no ambiente. Ademais, a plataforma precisa saber o conjunto de nós *fog* que estão em uma determinada região geográfica, quais serviços e aplicações estão sendo executados nestes nós, bem como as informações das cargas de trabalho e as condições da rede. Por outro lado, a **ciência de contexto** é a capacidade da plataforma de recuperar informações a respeito das mudanças de estado das diferentes variáveis contextuais de seu interesse no ambiente dinâmico em que os seus componentes estão inseridos. O contexto é qualquer informação que pode ser usada para caracterizar o estado de uma entidade [27]. Entidades podem ser, por exemplo, uma pessoa, uma aplicação, um lugar físico como uma sala, um nó *fog* ou um dispositivo IoT. Desta forma, as informações de estado são produzidas pelo processamento de dados brutos e incluem, mas vão muito além da localização de um dispositivo. Por exemplo, o contexto pode englobar o estado da rede (conexão, banda, congestionamento), a situação de um dispositivo IoT (energia residual, modo de operação, etc), o estado de um nó *fog* (carga de trabalho, uso de CPU e memória, etc), bem como informações que determinam o perfil da aplicação e do usuário. A plataforma necessita coletar, processar e armazenar essas informações com o objetivo de realizar ações ou reagir a estímulos do meio do qual tem interesse.

Uma característica de *fog computing* é que seus nós computacionais não necessariamente precisam ser fixos, ou seja, eles podem ser móveis [28]. Considerando um cenário de cidade inteligente onde nós *fog* são acoplados a veículos que se locomovem pela cidade, a plataforma deve assegurar a devida comunicação com esses nós e gerenciar sua disponibilidade, mantendo a continuidade dos serviços ofertados mesmo quando eles estiverem em trânsito por diferentes redes de acesso [29]. A mobilidade envolve o processo de *handover* (também conhecido como *handoff*). Um nó *fog* móvel realiza um *handover* quando ele muda de canal de conexão para outro canal mantendo-se na mesma rede. Para **suportar a mobilidade**, o mecanismo de *handover* precisa cancelar o registro de um nó de um ponto de acesso de origem e registrá-lo em um novo ponto de acesso [29]. Atualmente muitos trabalhos propõem soluções baseadas em LISP (do inglês, *Locator/ID Separation Protocol*), Mobile IPv6[2] e Mobile IPv4[3] para lidar com a mudança dos endereços IP à medida que os nós *fog* se movimentam, desta forma assegurando o suporte a mobilidade [29, 30]. Estas soluções desacoplam a identificação do dispositivo (a qual rede ele pertence no momento) da identificação da sua localização (endereçamento de localização na rede) usando um sistema de indexação distribuído, mantendo informações de um dispositivo independentemente da sua localização geográfica. A mobilidade do nó *fog* não deve ser confundido com o processo de *offloading* de processamento e de dados que pode ocorrer em *fog computing*, conforme discute [31]. *Offloading* pode envolver a migração de processamento computacional atrelado a uma tarefa quando um nó *fog* não é capaz de realizá-la, seja por falta de recursos ou pelo tempo necessário para execução da tarefa. Também, *offloading* envolve a migração de dados de um nó *fog* para outro nó *fog* ou para nuvem, que pode ser motivado por exemplo, por questões de segurança, disponibilidade dos dados ou restrições de armazenamento no nó *fog*, etc.

No contexto de *fog computing*, algumas aplicações críticas como, por exemplo,

[2]Mobile IPv6 - https://tools.ietf.org/html/rfc6275

[3]Mobile IPv4 - https://tools.ietf.org/html/rfc3344

telemedicina e condução assistida (do inglês *assisted driving*), requerem **disponibilidade** contínua, isto é, as plataformas precisam assegurar a oferta de seus serviços mesmo sob condições adversas de operação, evitando assim a degradação dos serviços ao longo do tempo. Técnicas como redundância e duplicação de nós *fog*, isolamento de falhas e aprendizado de máquina são usadas para ajudar a melhorar o MTTR (do inglês *Mean Time To Repair*), que é a média do tempo necessário para a plataforma *fog* se recuperar de uma falha.

Outro proeminente requisito mencionado na literatura é a **diminuição do consumo de energia elétrica**. Este requisito está alinhado ao conceito de sustentabilidade, que preza pela preservação do planeta e ao atendimento das necessidades humanas [32]. Posto isto, a distribuição do processamento em dispositivos de recursos limitados e, consequentemente com baixo consumo energético, torna toda a solução com menor custo energético quando comparado à computação em nuvem, conforme aponta [33]. Alguns trabalhos nesta linha visam otimizar a comunicação de controle e dados entre os nós *fog*, usando protocolos de baixo *overhead* a fim de aumentar a vazão de dados (do inglês *throughput*) de todo o sistema e, por conseguinte, minimizar o consumo energético. Outra técnica promissora é o uso de MIF (do inglês *Multilevel Information Fusion*) para reduzir o volume de dados que precisam fluir entre as camadas *fog* até atingir a nuvem [34].

Segurança e Privacidade é a capacidade da plataforma de proteger as informações das aplicações contra acessos não autorizados, além de manter a integridade e a privacidade. Segundo [35], devido às características de heterogeneidade, distribuição e mobilidade, o ambiente *fog computing* é mais vulnerável e menos seguro que a computação em nuvem. Ademais, as medidas existentes de segurança e privacidade de computação em nuvem não podem ser diretamente aplicadas em *fog computing*. Logo, muitos estudos enfocam na criptografia e autenticação para melhorar a segurança da rede a fim de proteger o ambiente *fog* contra ciberataques. Além disso, Redes Definidas por Software (do inglês *Software-defined Networking* - SDN) e Blockchain têm sido apresentadas como soluções de segurança promissoras para o ambiente *fog* [36].

Por fim, uma plataforma *fog computing* deve assimilar uma carga crescente de requisições e dispositivos IoT, mantendo seu funcionamento correto mesmo com alto volume de utilização, i.e., a plataforma precisa ser escalável. Atender ao requisito de escalabilidade é primordial, pois cada vez mais surgem dispositivos IoT conectados que geram uma grande quantidade de dados que requerem recursos para processamento e armazenamento. As soluções mais simples para endereçar **escalabilidade** são incorporar novos nós computacionais ou escalar internamente os nós com a adição de *hardware*, ou *software*. Porém, nem sempre essas soluções simplistas são possíveis de serem realizadas e, portanto, a plataforma deve empregar outras soluções como, por exemplo, o provisionamento de recursos com vistas a otimização da carga de trabalho dos nós disponíveis.

1.3. Arquiteturas de Referência

A primeira arquitetura para *fog computing* foi proposta por Bonomi et. al. em 2012 [17]. Na época, os autores vislumbraram *fog computing* como a extensão das capacidades da nuvem para a borda da rede e propuseram uma arquitetura formada por três camadas: i) a camada inferior formada pelos dispositivos IoT que são capazes de sensoriar e atuar no

ambiente em questão; ii) a camada intermediária, que é a camada *fog*, responsável pelo processamento dos dados localmente e quando da indisponibilidade de processá-los, por encaminhá-los para a nuvem; por fim, iii) a camada superior, formada pelos centros de dados na nuvem.

A arquitetura inicial para *fog computing* proposta por [17] impulsionou o surgimento de várias outras propostas nos últimos anos, algumas mais concretas e outras puramente teóricas [37, 38, 39, 6, 40, 19, 20, 41, 42, 43, 44]. Entretanto, conforme discutem [45, 23], ainda não existe uma arquitetura padronizada universalmente reconhecida, pois o que se percebe é que as arquiteturas diferem não somente quanto ao estilo arquitetural adotado, mas principalmente na forma da organização e tipos de dispositivos usados como nós *fog*, além dos objetivos gerais da arquitetura, métricas usadas para o provisionamento de recursos e serviços gerenciados pela plataforma, além do tipo de rede a qual é aplicável.

A diversidade de propostas de arquitetura para *fog computing* revelam a falta de consenso entre pesquisadores, arquitetos de *software* e profissionais de IoT sobre uma arquitetura unificada para *fog computing*, conforme argumentam os autores em [45]. Desta forma, é importante fazer um balanço das arquiteturas a partir de perspectivas de padronização para interoperabilidade entre sistemas *fog computing*, considerando a experiência na área dos principais fabricantes de *hardware* e *software*. Neste sentido, uma Arquitetura de Referência (AR) é o principal artefato para prover tal interoperabilidade, na medida que uma AR específica de maneira unificada e não ambígua, regras de negócio, estilos/padrões arquiteturais, decisões arquiteturais, boas práticas de desenvolvimento e elementos de *hardware* e/ou *software* necessários para a construção de arquiteturas concretas, que dizem respeito aos sistemas propriamente ditos no domínio em questão [46].

Considerando o importante papel desempenhado por uma AR para facilitar o desenvolvimento de sistemas, promovendo a redução de tempo e custo, além de padronizar as arquiteturas de sistemas em um domínio, esta seção apresentará duas ARs para *fog computing*, a saber: i) a AR proposta em [39] e tida como a primeira arquitetura de referência para *fog computing*; e ii) a AR OpenFog, criada pelo consórcio OpenFog e que tornou-se a normativa IEEE 1934-2018 em agosto de 2018.

1.3.1. Arquitetura de Referência (Dastjerdi e Buyya, 2016)

A arquitetura de referência proposta por [39] acrescentou mais duas camadas (*IoT applications* e *Software-defined resource management*) no topo da arquitetura de 3 camadas inicialmente proposta por [17], conforme ilustrado na Figura 1.1. Apesar da simplicidade desta AR, ela tem a sua relevância, pois foi uma das primeiras iniciativas de padronização de arquitetura para *fog computing* que estabeleceu a clara separação entre as camadas *fog*, a camada nuvem, além da camada que compreende os dispositivos IoT da borda da rede. Outra contribuição para a área foi o estabelecimento de um vocabulário adequado.

Conforme a Figura 1.1, a camada inferior é formada pelos dispositivos IoT, *gateways*, sensores e atuadores que são capazes de sensoriar e atuar no ambiente em questão. Estes dispositivos são os principais produtores de dados e utilizam os serviços da camada superior, chamada de *Network*, para comunicação entre eles e também com a nuvem. Nesta AR a camada *Network* se assemelha à camada *fog* da arquitetura proposta

por [17], pois ela é responsável por manter a conectividade dos dispositivos da camada subjacente, realizar o processamento dos dados localmente e quando da indisponibilidade de processá-los, encaminhá-los para a camada superior. A camada *Cloud Services and Resources* é responsável pela gerência dos serviços e recursos disponíveis na nuvem, que serão eventualmente usados para processar os dados IoT que alcançarem esta camada. A camada *Software-Defined Resource Management* gerencia toda a infraestrutura e tenta assegurar a qualidade dos serviços que serão ofertados para as aplicações, endereçando questões relacionadas com o monitoramento, provisionamento, segurança e gestão dos dados IoT. Por fim, a camada IoT *Applications* oferece um conjunto de facilidades - não descritas pelos autores da AR - para os desenvolvedores criarem as aplicações IoT que serão executadas na *fog*.

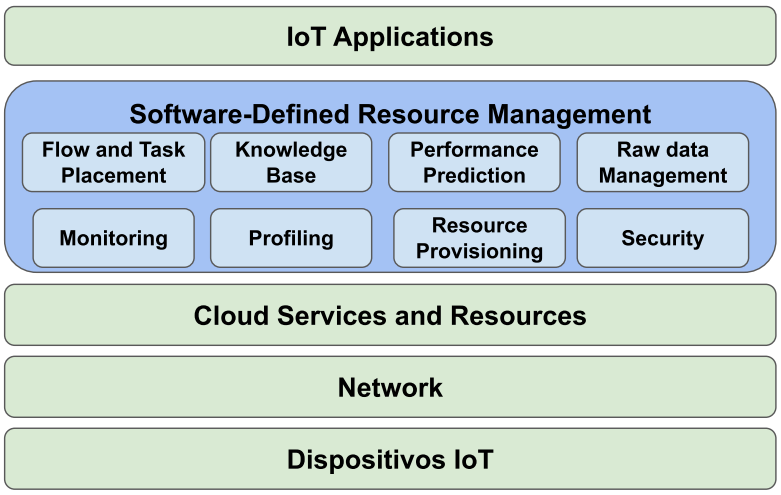

Figura 1.1. Arquitetura de Referência em cinco camadas proposta por [39]) (adaptado de [39])

Segundo os autores, dois principais objetivos foram considerados no projeto da AR: i) diminuir o custo financeiro do uso de recursos da nuvem; e ii) propiciar a execução de aplicações sensíveis a latência. Assim sendo, a camada *Software-Defined Resource Management* dispõe de vários componentes que trabalham cooperativamente para ofertar serviços de gerência, controle e execução das aplicações com o intuito de usar de forma apropriada as camadas *fog* (representada como *Network*) e nuvem para atingir os objetivos almejados. Os seguintes componentes estão presentes nesta camada:

- ***Monitoring*** - este componente é responsável por monitorar a execução das aplicações e a utilização dos recursos computacionais da *fog* e da nuvem. Ele fornece informações de telemetria como, por exemplo, a carga de trabalho, quantidade de memória e CPU em uso por cada nó *fog*, bem como o estado de cada aplicação. Estas informações são primordiais para todo o funcionamento do sistema, uma vez que as tarefas que compõem uma aplicação precisam ser distribuídas adequadamente entre os recursos da *fog* e nuvem de maneira a atender aos requisitos da aplicação e também não saturar as capacidades de cada camada.
- ***Flow and task placement*** - este componente é responsável por identificar os locais

apropriados para executar as tarefas das aplicações e lidar com seus respectivos fluxos de dados. Ele faz uso dos serviços providos pelo componente *Monitoring* para obter informações de telemetria dos recursos disponíveis na *fog* e nuvem. Este componente também determina ao componente *Resource Provisioning* que faça o provisionamento dos recursos necessários para atender as demandas. Não existem informações a respeito das métricas e estratégias usadas para posicionar as tarefas nos nós.

- ***Knowledge Base*** - as informações históricas a respeito do uso de recursos e demandas das aplicações são capturadas ao longo do tempo e organizadas em uma base de conhecimento para servir na tomada de decisões de outros componentes. Por exemplo, o componente *Flow and task placement* pode tomar suas decisões sobre a distribuição de tarefas considerando o comportamento histórico das aplicações.

- ***Performance Prediction*** - este componente faz uso da base de conhecimento mantida pelo componente *Knowledge Base* para realizar estimativas sobre o desempenho dos recursos disponíveis na camada de nuvem. Esta informação é útil para o componente *Resource Provisioning* na tomada de decisão sobre o provisionamento de recursos quando existirem muitas tarefas em execução e os critérios de QoS não forem satisfatórios, por exemplo, quando o desempenho (em termos de latência ou banda) exigido por uma aplicação não é satisfeito.

- ***Raw Data Management*** - este componente gerencia o acesso às fontes de dados, abstraindo o acesso a eles por outros componentes da arquitetura. As abstrações fornecidas podem ser simples como, por exemplo, consultas SQL e APIs REST (do inglês *Representational State Transfer*), ou exigirem acessos mais elaborados que envolvam o uso de técnicas como MapReduce.

- ***Security*** - este componente fornece autenticação, autorização e criptografia, conforme requerido pelos serviços da plataforma e aplicações.

- ***Profiling*** - este componente cria perfis que caracterizam as aplicações e os recursos disponíveis a partir da análise das informações obtidas dos componentes Knowledge Base e Monitoring. Os autores não forneceram detalhes destes perfis.

- ***Resource Provisioning*** - este componente realiza o provisionamento de recursos das camadas para a execução das aplicações. O ambiente altamente dinâmico em que plataformas de *fog computing* estão inseridas requer que as decisões de alocação sejam dinâmicas, pois os requisitos das aplicações, bem como o número de aplicações em execução, mudam ao longo do tempo. Portanto, as decisões de quantos e quais recursos alocar devem ser tomadas levando em consideração várias informações que são fornecidas por outros componentes (*Profiling*, *Performance Prediction* e *Monitoring*) além dos requisitos das aplicações. De posse destas informações, o componente pode tomar a decisão, por exemplo, de alocar uma aplicação para ser executada parcialmente na nuvem, pois os dados históricos mostram que tal aplicação possui necessidade de escalabilidade. Por outro lado, uma aplicação talvez não possa ser executada na borda da rede pois os recursos ali presentes estão saturados.

1.3.2. Arquitetura de Referência OpenFog

A Arquitetura de Referência OpenFog foi criada pelo consórcio de mesmo nome, com o objetivo de criar uma linguagem padronizada e estabelecer um arcabouço (*framework*) universal para a distribuição dos recursos e serviços, de forma a estimular a interoperabilidade entre aplicações de rede complexas que fazem uso intensivo de dados e que são pertencentes a diversos domínios como IoT, 5G, etc. O consórcio OpenFog foi fundado em novembro de 2015 pelas empresas ARM, Microsoft, Intel, Cisco, Dell e a Universidade de Princeton. Desde sua origem, os grupos de trabalhos do consórcio atuaram de forma conjunta com outros grupos de outros consórcios IoT e tecnologias correlatas como, ETSI-MEC[4] (*Mobile Edge Computing*), OPC-UA[5](*OPC Foundation Unified Architecture*), OCF[6] (*Open Connectivity Foundation*) e OPNFV[7] (*Open Platform for NFV*), com a finalidade de evitar duplicação de esforços e padronizar termos comuns.

Em agosto de 2018 o IEEE (*Institute of Electrical and Electronics Engineers*) adotou a OpenFog como AR para *fog computing* sob o padrão IEEE 1934-2018. Algum tempo depois, em janeiro de 2019, os consórcios OpenFog e IIC[8] (*Industrial Internet Consortium*) anunciaram a sua fusão e a incorporação dos grupos de trabalho nos projetos da IIC. Esta decisão foi tomada pois os dois consórcios trabalhavam com objetivos comuns e, dessa forma, passariam a poder impulsionar o desenvolvimento e a implantação de aplicações *fog computing* para a indústria 4.0.

A AR OpenFog é estruturada em oito pilares e cada pilar representa um princípio ou atributo necessário para a distribuição de computação, armazenamento, controle e funções de rede próximo a origem dos dados. Os pilares são os seguintes:

- **Pilar Segurança**- visa garantir a segurança fim-a-fim de aplicações que serão executadas. Para criar uma ambiente seguro para execução das aplicações, é necessário adotar estratégias de segurança nos dispositivos usados como nós *fog* e na rede que interliga os nós.

- **Pilar Escalabilidade** - visa permitir a adaptação a cargas de trabalho crescentes e diversas estratégias podem ser usadas para assegurar a escalabilidade. Por exemplo, o aumento das demandas de desempenho exigida pelas aplicações requerem o uso de protocolos de comunicação com baixo *overhead*. Além disso, a rede *fog* pode variar seu tamanho à medida que mais aplicações, usuários e dispositivos IoT são adicionados ou removidos. Também, as capacidades podem ser incorporadas aos nós *fog* pela adição de *hardware* como, por exemplo, processadores, dispositivos de armazenamento, aceleradores gráficos e interfaces de rede. A capacidade pode também ser aumentada via *software*, com a adição de serviços e componentes de *software*.

- **Pilar Abertura** - tem por objetivo assegurar a abertura das tecnologias usadas para evitar soluções proprietárias ou de um único fornecedor, o que pode ter um impacto

[4]ETSI-MEC - https://www.etsi.org/technologies/multi-access-edge-computing
[5]OPC-UA - https://opcfoundation.org/
[6]OCF - https://openconnectivity.org/
[7]OPNFV - https://www.opnfv.org/
[8]IIC - https://www.iiconsortium.org/

negativo no custo, qualidade e inovação que a plataforma pode ofertar. A utilização de padrões abertos é essencial para a criação e manutenção de um ecossistema de *fog computing*, onde plataformas e aplicações possam coexistir e compartilhar dados e recursos. Este pilar possibilita por exemplo, que nós *fog* possam existir em qualquer lugar, serem descobertos e dinamicamente criados. Ademais, a comunicação e representação de dados baseada em padrões abertos fortalece a interoperabilidade e habilita a portabilidade de serviços e aplicações na rede *fog*.

- **Pilar Autonomia** - possibilita os nós *fog* fornecerem suas funcionalidades mesmo que outros nós falhem. Diferentemente da computação em nuvem onde as decisões são centralizadas na nuvem, a natureza hierárquica de *fog computing* possibilita que as decisões de implantação de aplicações sejam realizadas em todos os níveis.

- **Pilar Programabilidade** - tem como objetivo assegurar a capacidade do nó ser flexível e mudar a sua configuração e forma de operar para resolver um determinado problema. A reconfiguração de um nó *fog* possibilita a criação de um ambiente dinâmico para execução de diversos tipos de aplicações. Obviamente os nós precisam oferecer às interfaces necessárias para sua reprogramação.

- **Pilar RAS** - o acrônimo RAS (do inglês *Reliability, Availability, and Serviceability*) significa confiabilidade, disponibilidade e operacionalidade, três importantes requisitos não funcionais. A *confiabilidade* diz respeito a entrega da funcionalidade esperada em condições normais e adversas de operação. *Disponibilidade* é definida tradicionalmente como a probabilidade de um sistema ou elemento do sistema estar em um estado funcional no momento em que o usuário precisar. No contexto da AR, o requisitos de disponibilidade assegura o gerenciamento e orquestração contínuas dos nós *fog* para salvaguardar a disponibilidade e integridade dos dados e computação realizadas. Por fim, a *operacionalidade* diz respeito a assegurar o funcionamento correto das funcionalidades ofertadas pela plataforma.

- **Pilar Agilidade** - visa a transformação dos grandes volumes de dados IoT em formatos gerenciáveis e que tenham valor agregado. A agilidade também lida com a natureza dinâmica de *fog computing* e com a necessidade de responder rapidamente às mudanças. Por exemplo, muitos dados IoT podem não ter um contexto definido e consequentemente valor para as aplicações se forem utilizados isoladamente. Porém, após serem coletados, agregados e analisados, tais dados podem apresentar um contexto e representar informações importantes para regra de negócio da aplicação. Se as atividades de coleta, agregação e análise forem realizadas na *fog*, a criação de contexto será mais próximo a geração de dados e decisões operacionais podem ser tomadas mais rapidamente.

- **Pilar Hierarquia** - este pilar assegura diferentes hierarquias que a rede *fog* pode ter para atender as necessidades das aplicações. Por exemplo, uma aplicação que gerencia o controle de acesso aos laboratórios em uma universidade será completamente instalada nos nós *fog* locais, que estão situados no prédio, e possivelmente não fará uso da camada nuvem. Por outro lado, uma aplicação de grande porte para cidades inteligentes como, por exemplo, o controle de tráfego, pode ser instalada em diversos níveis da rede *fog* e distribuída geograficamente na cidade.

A AR OpenFog segue a essência da arquitetura proposta por [17], ao dividir a arquitetura em três camadas, sendo que a camada intermediária é subdividida em quatro outras camadas. Esta subdivisão possibilita que implementações concretas da arquitetura possam residir em diferentes camadas da topologia de uma rede e assegurar os benefícios oferecidos por computação em nuvem como, por exemplo, virtualização, orquestração, eficiência e capacidade de gerenciamento.

1.3.2.1. Descrição da Arquitetura de Referência OpenFog

A descrição da AR OpenFog segue o *guideline* proposto pelo padrão ISO/IEC/IEEE 42010:2011 e utiliza os seguintes termos do vocabulário especificado por este padrão:

- **Ponto de Vista** (do inglês *Viewpoint*) - é uma maneira ou forma de se olhar um sistema. O ponto de vista contém um conjunto de interesses (do inglês *concerns*) e os modelos e técnicas usadas para descrever a arquitetura que atenda estes interesses.
- **Visão** (do inglês *View*) - uma visão é uma representação de um ou mais aspectos estruturais da arquitetura e fornece descrições variadas que mostram a arquitetura sob diferentes ângulos. As visões representam os conceitos de interesse de um conjunto de *stakeholders* e pode ser entendida como uma realização de um ponto de vista que permite reduzir a quantidade de informação para um *stakeholder* em um dado momento.
- **Perspectiva** (do inglês *Perspective*) - representam as questões ou capacidades que permeiam toda a arquitetura, i.e., afetam todas as camadas e componentes da arquitetura. As perspectivas definem decisões arquiteturais relacionadas a atributos de qualidade ou requisitos não funcionais.

A descrição da AR OpenFog, ilustrada na Figura 1.2, é a representação composta pelas diferentes visões que atendem aos anseios dos *stakeholders*, além das diferentes perspectivas.

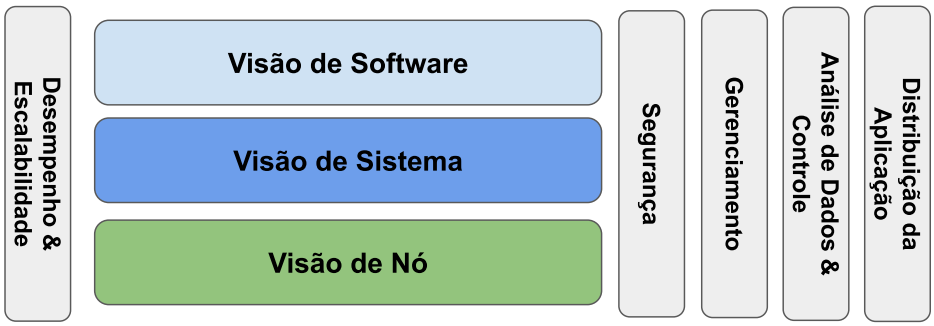

Figura 1.2. Descrição da AR OpenFog (adaptado de IEEE 1934-2018)

Na Figura 1.2 as **perspectivas** são identificadas como barras verticais. São elas: i) Desempenho & Escalabilidade; ii) Segurança; iii) Gerenciamento; iv) Análise de Dados

& Controle; e v) Distribuição da Aplicação. Ainda na Figura 1.2, a AR apresenta três diferentes visões, a saber: i) visão de *software*; ii) visão de sistema; e iii) visão de nó. Cada visão é formada por camadas que serão descritas no decorrer deste texto.

Dois pontos de vista são apresentados pela AR: i) **ponto de vista funcional** que demonstra como aplicar os elementos da arquitetura e as diferentes visões para atender as necessidades dos *stakeholders* em um dado cenário; e ii) **ponto de vista de implantação** que demonstra como sistemas *fog* são implantados/instalados para lidar com um determinado cenário. O documento OpenFog apresenta sucintamente no ponto de vista funcional um cenário de vigilância em aeroportos com o uso de câmeras de vídeo e devido a simplicidade deste ponto de vista, não a discutiremos neste texto. As próximas subseções apresentam o ponto de vista de implantação, as diferentes perspectivas e visões da AR.

1.3.2.2. Ponto de Vista de Implantação

A implantação diz respeito à organização lógica hierárquica dos nós *fog* em um determinado momento. Uma característica da *fog computing* é a sua capacidade de mudar a organização lógica para cada cenário diferente, adequando-se às necessidades das aplicações. As implantações podem ser grandes ou pequenas devido aos requisitos do cenário que será endereçado. Vários fatores contribuem para o dimensionamento da implantação como, por exemplo, quantidade e tipos de processamento exigidos por cada camada, quantidade de sensores/atuadores, capacidades de cada nó *fog*, latência na comunicação entre os nós e também entre os sensores e atuadores, bem como a existência de nós redundantes para atender ao requisitos de confiabilidade. O tipo mais comum de organização dos nós *fog* é em camadas (N-camadas), conforme ilustrado na Figura 1.3.

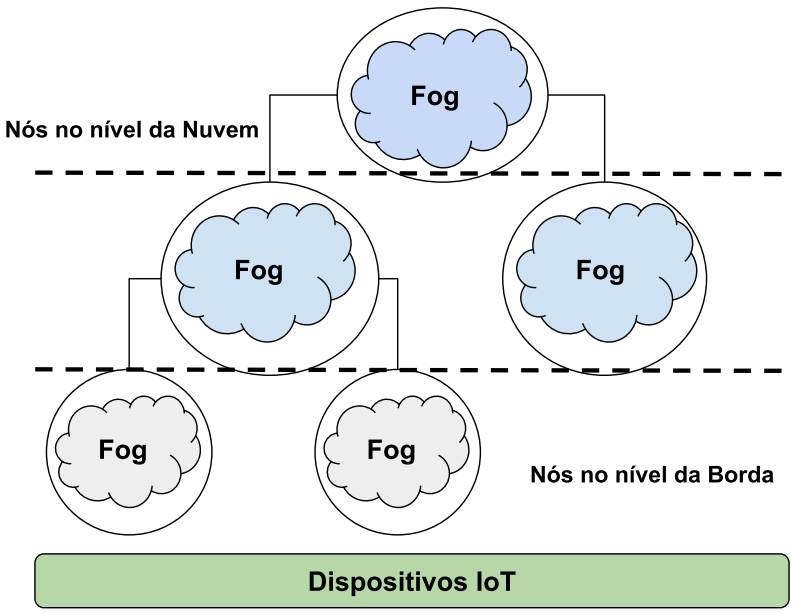

Figura 1.3. Modelo de implantação em N-camadas (adaptado de IEEE 1934-2018)

Neste tipo de organização os nós que estão localizados na camada mais próximas a

borda da rede (do inglês *edge network*), tipicamente possuem menor poder computacional e são responsáveis pela aquisição, coleta e normalização dos dados dos dispositivos IoT, e também o controle dos sensores e atuadores acoplados a estes dispositivos. Os nós que estão na camada sobrejacente, realizam a filtragem, compressão e transformação dos dados advindos da camada subjacente. Também, nesta camada podem ocorrer processos de análise de dados e inferências (do inglês *Data Analytics*) para tomada de decisões mais rápidas, a fim de endereçar requisitos de sistemas de tempo real ou próximo a tempo real. Entretanto, as análises são simples pois neste nível os recursos computacionais e as capacidades de inferência normalmente são limitadas. Por fim, os nós *fog* que estão na camada mais alta ou próximas a nuvem são, normalmente, responsáveis pela agregação de dados de forma a gerar conhecimento.

De maneira geral, no modelo N-camadas cada camada filtra e extrai dados significativos para criar mais inteligência. Desta forma, as camadas inferiores da hierarquia tratam do aquisição dos dados IoT e alguns processamentos simples nestes dados, enquanto que as camadas superiores têm interesse na criação de inteligência. Entretanto, em alguns modelos de implantação como, por exemplo, para análise de vídeo de câmeras de segurança, a análise - que é um processamento de inteligência - deverá ser realizada na borda da rede, pois a largura da banda de rede pode não ser suficiente para enviar todo *stream* de vídeo para as camadas superiores. Desta forma, realizando os processos de análise na borda da rede a quantidade de dados que necessitam ser enviados para as camadas superiores será menor.

1.3.2.3. Perspectivas

A AR OpenFog define cinco perspectivas e cada perspectiva representa as capacidades ou requisitos de qualidade que permeiam todas as camadas da arquitetura *fog computing*, a saber: i) Desempenho & Escalabilidade; ii) Segurança; iii) Gerenciamento; iv) Dados, Análise & Controle; e v) Distribuição da Aplicação. As perspectivas também envolvem decisões de projeto que devem ser tomadas para atender as demandas dos *stakeholders*.

A **Perspectiva de Desempenho & Escalabilidade** possibilita a execução de processos de inteligência e análise - que antes tipicamente eram executados na nuvem - na borda da rede ou perto dela e, consequentemente próximo aos produtores de dados, o desempenho de todo o sistema *fog computing* será melhorado. Outra característica de *fog computing* é possibilitar às aplicações uma rápida adequação às mudanças de tráfego de dados, por exemplo, o surgimento de carga crescente de dados IoT advindos de vários sensores que controlam o tráfego de uma cidade no horário de pico. Desta forma, o sistema *fog computing* pode detectar esta alteração mais rapidamente e escalar todo o sistema ou parte dele. Porém, ao escalar uma parte do sistema *fog computing*, deve ser garantido que não haverá a interrupção ou diminuição na oferta de serviços da plataforma e na execução de outras aplicações.

A **Perspectiva de Segurança** preocupa-se com a segurança fim-a-fim que é um elemento crítico para o sucesso de uma plataforma *fog computing* e envolve todos os dispositivos que estão entre a nuvem e os dispositivos finais. Os dispositivos IoT e nós *fog* necessitam ser seguros, bem como a rede formada por eles também necessita ser segura e

confiável. Deste modo, nos diferentes modelos de implantação (ver seção 1.3.2.2) se uma camada não for segura e as demais camadas forem seguras (ou vice versa), toda a solução não será segura. Sistemas confiáveis dependem da utilização de elementos de confiança que são responsável pela manutenção das políticas de segurança para os seus dispositivos e a realização da inspeção do comportamento dos dispositivos para determinar se o sistema e seus componentes estão operando de forma confiável. As políticas de segurança especificam quem ou o que pode acessar os componentes e dispositivos e sob quais circunstâncias pode ocorrer o acesso. Além do mais, as diferentes implantações de *fog computing* requerem diferentes mecanismos de segurança que são dependentes das ameaças e do valor do ativo que precisa ser protegido em um determinado nó *fog*. As ameaças compreendem os diferentes tipos de ataques que podem prejudicar um ativo ou ocasionar uma falha de segurança na plataforma [47]. São exemplos de ameaças, ataques diretos ao *hardware*, ataques aos componentes de *software* e de rede, e ataques internos que partem de dentro da rede *fog*. As implicações desses ataques, normalmente, dizem respeito à violação da confidencialidade, integridade, autenticidade, disponibilidade e privacidade dos dados. Por sua vez, os ativos podem envolver dados financeiros, informações sensíveis e pessoais, propriedade intelectual, informações da própria infraestrutura da plataforma, etc.

Na **Perspectiva de Gerenciamento** os nós utilizados em uma implantação *fog computing* podem ser instalados em ambientes remotos, fixos e não-fixos, e em ambientes com condições extremas como, por exemplo, no domínio de agronegócio os nós *fog* podem ser instalados em equipamentos agrícolas e estarão sujeitos às intempéries do clima. Ademais, os nós *fog* podem precisar de atualização de *software* e/ou *firmware*. Desta forma, configurar e gerenciar manualmente os nós é praticamente inviável e, portanto toda implantação *fog* precisa fornecer uma entidade de gerenciamento. Muitas plataformas usam *heartbeat* como estratégia para gerenciar a saúde das implantações. Nessa estratégia, os componentes da plataforma precisam enviar periodicamente uma informação de controle atestando sua saúde para a entidade de gerenciamento da plataforma. Ainda na perspectiva de gerenciamento, todos os nós *fog* seguem um ciclo de vida de gerenciamento formado por cinco fases, a saber: i) *Ativação*; ii) *Provisão*; iii) *Operação*; iv) *Recuperação*; e v) *Desativação*. O ciclo de vida deve ser assegurado por agentes de gerenciamento em cada nó. Na fase *Ativação*, são realizadas algumas ações para a correta identificação do nó *fog*, verificação de certificados, calibração de relógios, atestação e autenticidade do nó, etc. A fase *Provisão*, inclui atividades como descoberta, identificação e anúncio de recursos ou capacidades do nó. Na fase de *Operação*, os nós estão em funcionamento normal, mas no momento em que eles começam a funcionar de forma não esperada, então a fase de *Recuperação* se inicia. O processo de recuperação precisa ser autônomo, i.e., o próprio nó precisa realizar operações de recuperação e, eventualmente, outros nós podem ajudá-lo na ação de recuperação. Por fim, na *Desativação*, o nó precisa, muitas vezes, apagar informações atualmente usadas para uma implantação de forma a deixá-lo pronto para utilização em outras implantações.

Na **Perspectiva de Dados, Análise & Controle** a análise (do ingês *analytics*) é o processo de criar conhecimento a partir de dados e informações. O modelo tradicional de análise centrado na nuvem, em muitos casos, não é viável devido ao grande volume de dados que precisa ser plenamente capturado, armazenado e transportado. *Fog com-*

puting permite capturar, armazenar e transportar apenas dados relevantes à medida que tem a capacidade de realizar processos de análise no continuum. Quatro tipos de análises podem ser realizadas em diferentes camadas da *fog* de acordo com as necessidades das aplicações: i) a *análise descritiva* é realizada nas camadas mais baixas e é usada normalmente pelas aplicações para compreender o estado atual de suas operações, i.e., a análise do que está acontecendo ou do que aconteceu; ii) a *análise diagnóstica* tem como objetivo apresentar o que motivou determinado evento; iii) a *análise preditiva* utiliza todo o conhecimento da análise anterior juntamente com outros conhecimentos da regra de negócio da aplicação para entender o que irá acontecer; por fim,a iv) *análise prescritiva* é utilizada para melhorar os processos de análise.

A **Perspectiva Distribuição da Aplicação** tenta assegurar um ecossistema *fog computing* formando por múltiplos fornecedores (do inglês *multi-vendor*), onde as aplicações precisam ser espalhadas e ter a capacidade de migrar e operar/executar corretamente em qualquer nível da hierarquia de uma implantação *fog*. Desta forma, as aplicações precisam interoperar com todos os níveis da implantação. Assim, os dados que são coletados ou gerados por um nó *fog* devem ser compartilhados com outros nós na hierarquia. Por exemplo, uma aplicação pode ser executada em diferentes nós *fog* que requerem a comunicação entre eles. Assim, os nós que vão se comunicar podem estar situados na mesma camada na hierarquia *fog*, configurando uma distribuição leste-oeste. Por outro lado os nós podem estar em camadas diferentes, o que configura uma distribuição norte-sul.

1.3.2.4. Visões

Conforme dito anteriormente as visões representam os conceitos de interesse de um conjunto de *stakeholders* e pode ser entendida como uma realização de um ponto de vista que permite reduzir a quantidade de informação para um *stakeholder* em um dado momento. A AR OpenFog apresenta três visões que serão discutidas nas próximas subseções.

1.3.2.4.1. Visão de Nó

A visão de nó é formada pelas duas camadas (*Abstração dos Protocolos* e *Sensores e Atuadores*) mais baixas da representação da AR, exibidas em cor verde, além de alguns aspectos, em azul, conforme ilustrado na Figura 1.4.

A inserção de um nó na rede *fog computing* requer o endereçamento de várias aspectos, a saber: i) Segurança; ii) Gerenciamento; iii) Rede; iv) Aceleradores de *Hardware*; v) Computação; e vi) Armazenamento. Estes aspectos são discutidos em seguida:

- **Segurança** - a segurança de cada nó *fog* é essencial para a segurança de todo o sistema *fog computing*, conforme dito na descrição da Perspectiva de Segurança (seção 1.3.2.3). Em alguns casos um nó *fog* pode servir também como *gateway* para sensores e atuadores legados que não implementam mecanismos de segurança. Os nós *fog* estarão, em muitos casos, em espaços físicos públicos e estão sujeitos a alterações indesejáveis. Desta forma, o nó precisa empregar mecanismos para a detecção e notificação de adulterações de forma a garantir a confiabilidade do nó e, por conseguintes, de todo sistema *fog*.

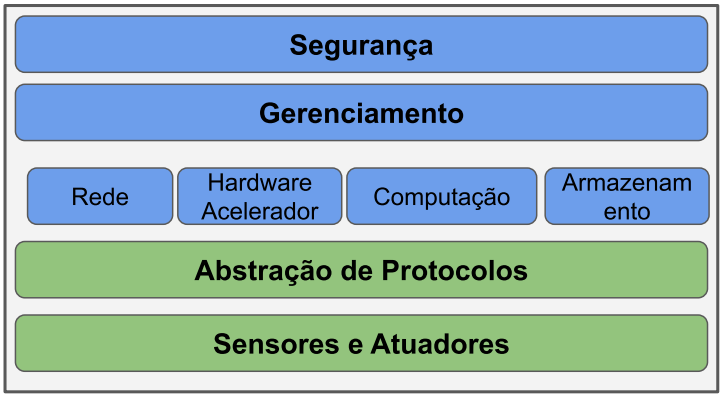

Figura 1.4. Visão de Nó (adaptado de IEEE 1934-2018)

- **Gerenciamento** - o gerenciamento do nó é somente realizado se o nó provê uma interface de gerenciamento. Tal interface precisa implementar um protocolo de gerenciamento para possibilitar a agentes de gerenciamento averiguar a saúde e controlar a configuração interna do nó. As informações de saúde podem envolver informações dos elementos internos do nó como as unidades de armazenamento, CPU, memória e o estado dos *hardwares* acelerados, a temperatura, a voltagem, etc. A configuração interna do nó pode incluir, por exemplo, parâmetros de configuração como endereço IP, e configurações dos *hardwares* acelerados.

- **Rede** - todo nó *fog* precisa estar interconectado por uma rede (*network*), que deverá prover a escalabilidade e conectividade requeridas para as diferentes implantações *fog computing*. O modelo de conectividade da rede dependerá da finalidade e localização do nó. Por exemplo, um nó *fog* que está situado em uma fábrica e é usado para capturar e analisar dados relativos às operações da fábrica, provavelmente será conectado as camadas da rede *fog* por uma rede cabeada. Por outro lado, um nó *fog* usado para capturar e analisar informações de localização de veículos em uma cidade será conectado aos sensores nos veículos por uma rede sem fio. Também, os nós *fog* podem ter conexões diretas entre eles usando tecnologias de interconexão de baixa latência, como RDMA (do inglês *Remote Direct Memory Access*). As redes sem fio desempenham um papel muito importante na tecnologia IoT. Existem três principais tipos de redes sem fio que *fog computing* pode fazer uso dependendo do modelo de implantação, a saber: i) WWAN (*Wireless WAN*); ii) WLAN (*Wireless LAN*); e iii) WPAN (*Wireless Personal Area Networks*). Cada tipo de rede possui seus protocolos e padrões. A WWAN é usado para cobrir uma grande área geográfica. São exemplos de WWAN as tecnologias de comunicação celular como, 3G, 4G LTE e 5G, os padrões NB-IoT (*Narrow Band IoT*) e LPWAN (*Low-Power Wide Area Networks*). A rede WLAN, também conhecida como WiFi, são apropriadas para espaços geográficos menores e velocidades mais altas que os demais tipos de rede, sendo que os padrões mais utilizados pertencem a família IEEE 802.11 (a, b, g, n, ac). A WPAN são redes menores que requerem menor consumo energético. São exemplos deste tipo de rede, Bluetooth, comunicação via Infravermelho (IR), ZigBee, e o padrão IEEE 802.15.4 (*Low Rate WPAN*).

- ***Hardwares* aceleradores** - algumas aplicações necessitam de *hardwares* aceleradores como, por exemplo FPGA (do inglês *Field-Programmable Gate Array*) e GPGPU (do inglês *General Purpose Graphics Processing Unit*) para satisfazer requisitos de latência e poder computacional no processamento de grandes volumes de dados. Tais *hardwares* permitem por exemplo, operações paralelas entre matrizes gigantescas reduzindo consideravelmente o tempo computacional.

- **Computação** - o nó *fog* também necessita ter capacidade para computação de propósito geral. Desta forma, o nó *fog*, em muitos casos, necessita ter um conjunto de bibliotecas e APIs padrão para que as aplicações possam ser executadas adequadamente. Por exemplo, muitas plataformas fazem uso de virtualização leve - como Docker - para execução das aplicações que são encapsuladas como contêineres. Desta forma, o nó *fog* precisa fazer o devido controle da execução dos contêineres sobre o uso dos recursos disponíveis como processadores, memória e dispositivos de entrada e saída.

- **Armazenamento** - O armazenamento pode ser de dados de sensores que estão diretamentes associados ao nó *fog*, dados de contexto, *logs* e dados de aplicações que estão residindo/executando atualmente no nó. Muitas tecnologias de armazenamento podem ser usadas pelos nós *fog*, por exemplo, RAM Arrays para armazenamento em memória de forma a diminuir o tempo de acesso aos dados, uso de dispositivos de estado sólido (SSD do inglês *Solid-state Drive*) e discos rígidos que podem eventualmente ter organizações específicas como, por exemplo, serem organizados em uma única unidade lógica para formar um sistema de armazenamento, como propõe RAID (*Redundant Array of Independent Disks*).

A **camada Sensores e Atuadores** compreende os sensores e atuadores que são os elementos de *hardware* ou *software* (pois podem ser virtualizados) fundamentais na IoT. Vários deles, até mesmo milhares, podem estar associados a um único nó *fog*. Existem uma infinidade de diferentes dispositivos IoT, desde dispositivos do tipo *dumb*, que não têm capacidade de processamento significativa, até dispositivos que apresentam as capacidades necessárias para executar funções da *fog*, como por exemplo, os dispositivos Raspberry da família Pi e NVIDIA Jetson Nano. Os dispositivos possuem capacidades de conectividade via cabo via sem fio, e fazem uso de diferentes meios e protocolos de comunicação como, I2C, GPIO, SPI, BTLE, ZigBee, USB e Ethernet. Devido às restrições de conectividade de alguns sensores e atuadores, eles não são capazes de interagir diretamente com um nó *fog*. Desta maneira, a **camada de Abstração de Protocolos** tem como objetivo tornar possível a conexão deles com um nó*fog*com o intuito que os dados dos sensores possam ser adequadamente capturados, bem com os comandos possam ser enviados aos atuadores. Também, esta camada precisa assegurar o acesso às políticas de segurança, conforme descrito na perspectiva segurança.

1.3.2.4.2. Visão de Sistema

A visão de sistema é formada pela junção de uma ou mais *Visões de Nó* e as camadas *Virtualização de Hardware* e *Plataforma de Hardware*, conforme ilustrado na Figura 1.5. Esta visão proporciona uma representação do sistema *fog* e, apesar de que a Figura 1.5

ilustra apenas um nó *fog* na visão, não se descarta a possibilidade de existirem vários nós para criar um sistema *fog*.

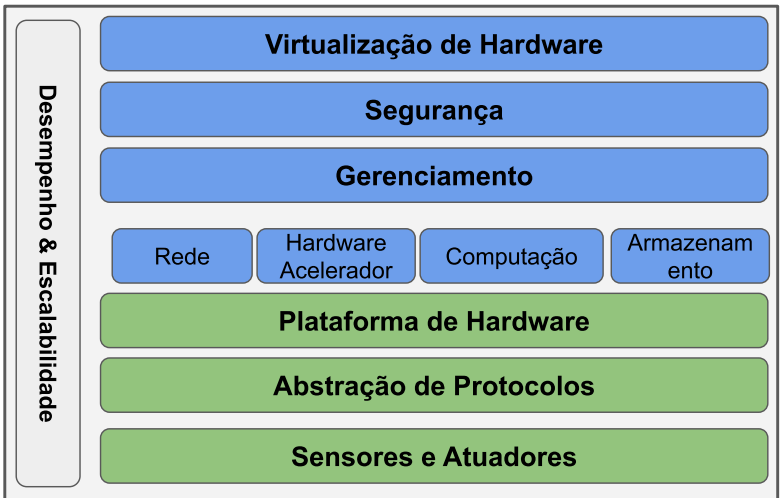

Figura 1.5. Visão de Sistema (adaptado de IEEE 1934-2018)

Na visão de sistema, a **camada Plataforma de *Hardware*** representa os dispositivos físicos da plataforma *fog computing*. Tais componentes podem estar inseridos em ambientes com condições severas e, portanto requerem proteção física. Por exemplo, no contexto de cidades inteligentes, nós *fog* podem ser instalados em postes que são usados para iluminação pública de forma que os nós estarão sujeitos às intempéries do clima, poluição, vandalismo, etc. Desta maneira, várias questões precisam ser atendidas para manter uma boa infraestrutura de *hardware*. São exemplos: a conformidade com regulamentos locais e padrões como emissão de calor e ruído; proteção contra fatores ambientais (temperatura, umidade, radiação solar, vibrações, quedas, etc); resistência a ataques físicos, vandalismo ou roubo; adequação de tamanho e peso do *hardware* e consumo de energia elétrica; suporte mecânico para acesso aos componentes internos mais facilmente; e refrigeração dos componentes internos.

A **camada Virtualização de *Hardware*** representa os mecanismos de virtualização, sejam eles baseados em virtualização completa (todo o sistema) ou leve (apenas a aplicação), que são essenciais para *fog computing*, pois permitem, por exemplo, que diferentes aplicações possam coexistir no mesmo nó *fog* e compartilhar o mesmo *hardware* e/ou sistema operacional, sem interferir nas operações (por exemplo entrada e saída) e processos de regra de negócio das outras aplicações.

1.3.2.4.3. Visão de *Software*

A visão de *software* demonstra a execução de um *software* em uma plataforma formada por uma ou mais visões de nó juntamente com outros componentes para endereçar um determinado cenário *fog computing*. A visão de *software* é representada pelas três camadas superiores (**Serviços de Aplicação**, **Suporte de Aplicação**, e **Gerenciamento do Nó & Execução de *Software***) da descrição AR OpenFog. Essas três camadas estão acima da camada **Plataforma de *Hardware***, conforme ilustrado na Figura 1.6.

Figura 1.6. Visão de *Software* (adaptado de IEEE 1934-2018)

A **camada Gerenciamento do Nó & Execução de *Software*** é responsável pelas operações e serviços de gerenciamento do nó, mantendo os elementos de *hardware* e *software* configurados adequadamente para o modelo de implantação em uso, além de manter o nó em funcionamento em níveis especificados previamente de disponibilidade, resiliência e desempenho. Outra responsabilidade da camada é fornecer suporte a execução de *software* no nó e facilitar a comunicação entre os nó e os demais, de forma a atender a perspectiva de distribuição da aplicação. Nesta camada, o *software* em execução representa uma aplicação *fog computing* ou parte dela, que tipicamente são executadas em ambientes virtualizados ou em contêineres.

A **camada Suporte a Aplicação** é formada por um conjunto de *software* e sistemas que fornecem serviços de infraestrutura que não são relacionados a nenhum domínio de problema ou aplicação, mas que servem para ajudar e apoiar serviços de aplicação (*software* de aplicação). São exemplos de *software* e sistemas pertencentes a esta camada, motores de execução de máquinas virtuais e contêineres (Docker, VMs, etc), API(do inglês *Application Programming Interface*) e bibliotecas de linguagens de programação, *software* e sistemas para comunicação assíncrona e síncrona (ActiveMQ, RabbitMQ, DDS, etc), sistemas para persistência (MySQL, SQLite, Cassandra, MongoDB, Redis, etc), ferramentas e *frameworks* para análises (Spark, Hadoop, etc), etc.

Por fim, a **camada Serviços de Aplicação** representa o conjunto de *software* que são dependentes das camadas subjacentes e são projetados para endereçar requisitos específicos de domínio e de uma aplicação. Isto é, são as aplicações *fog computing*.

1.4. Plataformas open source de *fog computing*

Esta seção discute as principais plataformas de *fog computing* de código aberto (*open source*). Tal discussão será embasada pelos requisitos previamente levantados para plataformas de *fog* (ver seção 1.2) e uma comparação entre as plataformas será apresentada. Para a escolha das plataformas foram utilizados os seguintes critérios: i) quantidade de citações do artigo científico que apresenta a plataforma; ii) a atualidade do repositório de código fonte, i.e., o quão ativo e atual é o projeto da plataforma e se ela ainda está em evolução; e iii) a disponibilidade plena da plataforma, pois algumas plataformas são apresentadas apenas em artigos científicos e não estão disponíveis para uso, enquanto outras não tiveram seus projetos continuados e estão defasadas. De posse destes critérios serão descritas as plataformas FogFlow, Stack4Things, Microsoft Azure IoT Edge e ParaDrop.

1.4.1. FogFlow

A plataforma FogFlow [7] faz parte do projeto FIWARE[9] e serve para orquestrar a carga de trabalho de aplicações em nós *fog* considerando a localização geográfica dos produtores de dados e dos nós *fog*, de forma a tratar dos requisitos de otimização de uso da largura de banda e suportar aplicações sensíveis à latência. FIWARE é uma plataforma ampla formada por vários componentes genéricos, interoperáveis e extensíveis, chamados de GEs (do inglês *Generic Enablers*). Os GEs fazem uso do protocolo NGSI (*Next Generation Service Interfaces*) que especifica a forma de acesso aos serviços ofertados pelos componentes da plataforma FIWARE, bem como define um modelo de representação dos dados de contexto. O protocolo NGSI é mantido pela OMA (*Open Mobile Alliance*) e define a interface de comunicação para troca de informações contextuais entre diferentes GEs, i.e., promove a interoperabilidade no ecossistema FIWARE. O modelo de representação de dados de contexto NGSI utiliza o conceito de entidades de contexto, que são representações virtuais de objetos, sejam eles físicos ou não, e suas respectivas propriedades. Por exemplo, uma sala de aula pode ser representada como uma entidade de contexto com as seguintes propriedades: quantidade de pessoas; temperatura; localização geográfica; e luminosidade.

FogFlow fornece um modelo de programação onde a lógica de processamento de cada aplicação é encapsulada em *tarefas* (do inglês *tasks*) e cada *tarefa* possui entradas e saídas representadas como *entidades de contexto*. Desta forma, as aplicações são definidas como um grafo dirigido de tarefas (DAG, do inglês *Directed Acyclic Graph*) que executam a lógica de processamento dos dados. O resultado do processamento de uma *tarefa* é uma entidade de contexto que servirá como entrada para outra *tarefa*. Cada tarefa é implementada como uma imagem Docker e a plataforma se responsabiliza pela orquestração das tarefas considerando a topologia (organização lógica) e características dos nós *fog*. Todos os componentes desta plataforma comunicam-se através de um *broker* de comunicação compatível com o protocolo NGSI e cabe ressaltar que a plataforma espera que todos os dados IoT sigam o formato NGSI. Entretanto, muitos dispositivos IoT são simples e incapazes de publicar e/ou receber dados no formato NGSI devido, principalmente, a sua limitação de processamento. Os nós *fog* utilizados pela plataforma para a distribuição das múltiplas tarefas são chamados de *workers* e cada *worker* atua como um provedor para publicar o resultado de sua computação no *broker*, e também atua como consumidor no *broker* para receber dados de contexto ou novas tarefas para serem executadas.

A arquitetura da plataforma FogFlow, ilustrada na Figura 1.7, é baseada em três camadas. A **camada Gerenciamento de Serviço** (do inglês *Service Management*) está localizada na nuvem e contém os componentes para o gerenciamento de toda a plataforma e para criação de aplicações. Nesta camada, o componente *Repositório de Tarefas* é responsável por armazenar todas as descrições tarefas que compõem as aplicações, sendo que cada tarefa é implementada e encapsulada como uma imagem Docker. O componente *Designer de Tarefas* oferece uma interface gráfica Web para a especificação das aplicações em termos de tarefas e entidades de contextos. Isto é, o *Designer de Tarefas* ajuda o desenvolvedor a criar visualmente o DAG que representa a aplicação. Por fim, o componente *Orquestrador* é responsável por fazer a distribuição das tarefas que com-

[9]FIWARE - https://www.fiware.org/foundation/

põem uma aplicação nos nós *fog* mantidos pela plataforma. A distribuição das tarefas é guiada pela localização geográfica da fonte de dados de interesse da aplicação. Uma vez que a plataforma conhece previamente a localização geográfica de cada nó *fog* e também da fonte de dados, o componente *Orquestrador* tenta associar o nó *fog* mais próximo à fonte de dados para executar as tarefas da aplicação. Além da informação de localização, o *Orquestrador* também utiliza a informação de capacidade de execução simultânea de tarefas de cada nó, de forma a não sobrecarregá-los, bem como sua arquitetura (X86 ou ARM) para verificar a compatibilidade da imagem Docker correspondente a tarefa.

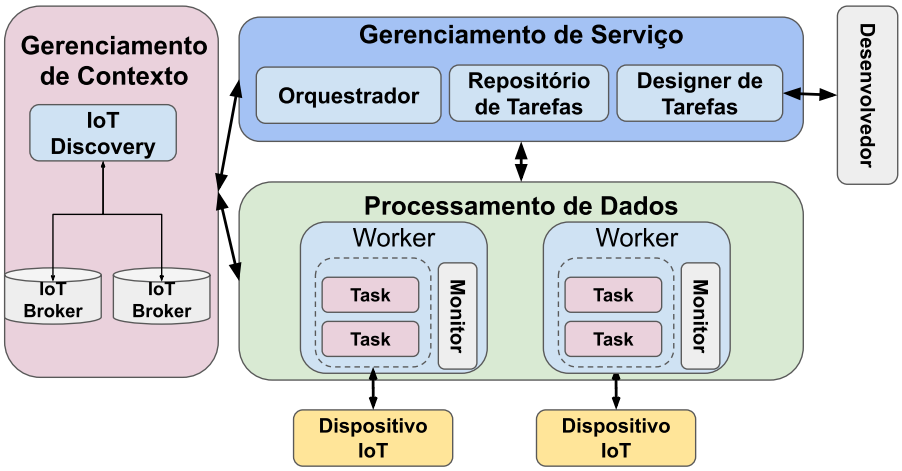

Figura 1.7. Arquitetura da plataforma FogFlow (adaptado de [7])

A **camada de Processamento de Dados** (do inglês *Data Processing*) compreende os nós *fog* que são responsáveis pela execução das tarefas demandadas pelo componente *Orquestrador*. Na plataforma FogFlow cada nó *fog* (*Worker*), ao receber a requisição para a execução de uma dada tarefa, solicita a imagem Docker correspondente da tarefa ao componente Repositório de Tarefas. Logo em seguida, após obter a imagem, o *Worker* instancia a imagem e executa o contêiner Docker (tarefa) no ambiente virtualizado. Os nós *fog* podem estar localizados na borda da rede ou também podem ser completamente virtualizados na nuvem. Cada nó possui uma capacidade diferente de execução de tarefas que é levada em consideração no ato da orquestração das tarefas pelo *Orquestrador*. Também, em cada *Worker* existe um componente chamado *Monitor* que é responsável por inspecionar o dispositivo e recuperar dados de telemetria (quantidade de tarefas em execução, consumo de memória, processador e armazenamento, etc) que são submetidos periodicamente ao *Orquestrador*.

Por fim, a **camada Gerenciamento de Contexto** (do inglês *Context Management*) tem como objetivo gerenciar todas as entidades de contexto e torná-las passíveis de descoberta e acessíveis via consulta e/ou subscrições. O FogFlow mantém o componente centralizado *IoT Discovery* que provê uma visão global das entidades de contexto disponíveis para todos os outros componentes da plataforma e também para as tarefas em execução. O *IoT Discovery* não armazena os valores das entidades de contexto, ele serve apenas como um elemento indexador que sabe onde recuperar tais informações. As entidades de contexto são gerenciadas pelas várias instâncias do componente *IoT Broker*.

Idealmente, devem existir várias instâncias de *IoT Broker* distribuídas geograficamente, de tal forma que um dispositivo IoT possa publicar seus dados no *IoT Broker* mais próximo a ele. Por sua vez, as tarefas também precisam se subscrever em um *IoT Broker* para receber as entidades de contexto de seu interesse, assim como publicar o resultado do seu processamento (uma entidade de contexto). Neste sentido, é comum ter um *IoT Broker* para cada nó *fog* disponível. Cada instância do componente *IoT Broker* armazenam em memória somente o último valor de cada entidade de contexto. Além do mais, devido a dinâmica de orquestração da plataforma, muito provavelmente a tarefa consumidora de dados (subscribe) estará associada no mesmo *IoT Broker* que o dispositivo produtor de dados (*publisher*).

Considerando os requisitos previamente levantados para plataformas *fog computing*, identificamos que a plataforma não apresenta elementos para endereçar os requisitos de suporte a mobilidade e eficiência energética. Enquanto, o requisito de segurança e privacidade é atendido parcialmente, pois a plataforma pode fazer uso do protocolo de comunicação segura HTTPs (*Hyper Text Transfer Protocol Secure*) na comunicação entre os componentes da nuvem e os nós *fog*, mas não especifica a comunicação segura entre os dispositivos IoT e o *broker*. Por sua vez, o requisito de escalabilidade também é atendido parcialmente, uma vez que a plataforma não oferece mecanismos para escalar automaticamente como, por exemplo, manter um conjunto de nós *fog* em estado de espera para serem utilizados apenas quando a demanda crescer ou então implementar estratégias de duplicação de tarefas em nós *fog* diferentes para assegurar também a disponibilidade. Por fim, o protocolo NGSI é baseado em HTTP e, conforme [48], este protocolo não é adequado para aplicações ultra sensíveis a latência devido ao *overhead* exigido pelo protocolo TCP (*Transmission Control Protocol*) no estabelecimento e manutenção da conexão.

1.4.2. Stack4Things

Stack4Things [49, 50] é uma plataforma baseada no OpenStack[10] que tem como objetivo fornecer recursos de sensoriamento e atuação como serviços, implementando a ideia de SAaaS (do inglês *Sensing-and-Actuation-as-a-Service*) [51]. A plataforma realiza o provisionamento sob demanda dos recursos e provê uma representação uniforme dos dispositivos IoT distribuídos geograficamente em uma cidade, abstraindo e gerenciando-os em um ecossistema unificado de objetos a serem configurados de acordo com os requisitos das aplicações. A plataforma mapeia os requisitos das aplicações em níveis mais baixos correspondentes as capacidades dos nós *fog*.

Conforme ilustrado na Figura1.8, a arquitetura da plataforma é baseada em duas camadas. A **camada IaaS** (do inglês *Infrastructure-as-a-Service*) está localizada na nuvem e é responsável por manter os componentes *Virtual Boards*, que são representações virtuais dos nós *fog*. A plataforma faz uso de uma abordagem SDN (do inglês *Software Defined Network*) para associar a virtualização de funções nos nós virtuais (*Virtual Boards*) as operações executadas diretamente nos nós *fog*. As funções virtuais podem ser parcialmente ou completamente executadas nos nós *fog*, dependendo das suas capacidades e da complexidade da função. Também, esta camada gerencia a infraestrutura de comunicação (rede) dos nós virtuais e dos nós *fog* por intermédio do componente *Control*

[10]OpenStack - https://www.openstack.org/

Plane. O mecanismo de comunicação entre eles é baseado na criação de túneis TCP sobre WebSocket[11]. Assim, esta camada fornece serviços para os desenvolvedores gerenciar e controlar a rede de nós *fog*, bem como virtualizar funções e criar camadas adicionais de sobreposição de rede (do inglês *network overlays*) entre as funções. De acordo com os autores, a camada IaaS da plataforma fornece o serviço chamado de *IoTronic*, que possibilita gerenciar os nós *fog* via linha de comando por intermédio do programa *Stack4Things Command Line Client* e também por uma API Web RESTful. Além disso, o serviço *IoTronic* possui os seguintes agentes: *Registration Agent*, que trata do registro dos nós *fog* ao sistema no momento de inicialização; e *Command Agent*, que trata do gerenciamento e distribuição dos comandos para configuração dos nós além da execução das aplicações.

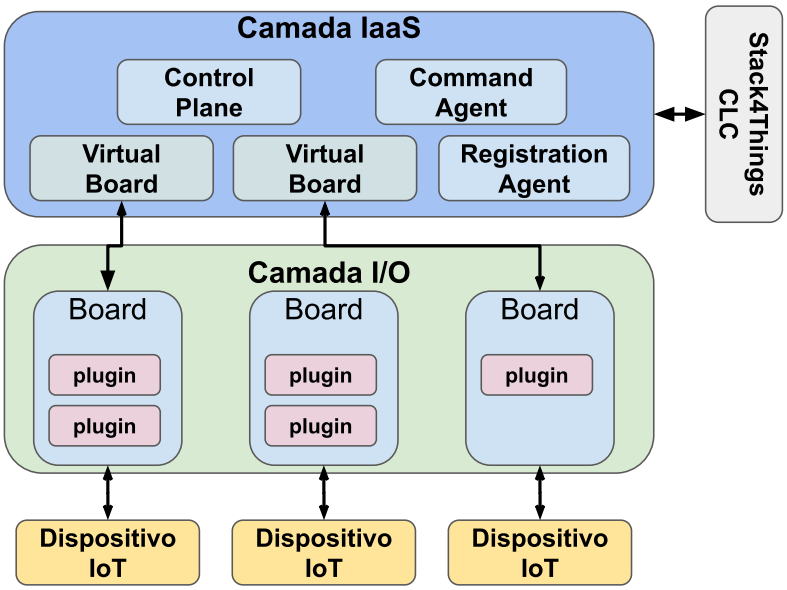

Figura 1.8. Arquitetura da plataforma Stach4Things (adaptado de [49])

A **camada I/O** compreende os nós *fog* (representados como *board* na Figura1.8) que estão conectados diretamente aos dispositivos IoT da camada subjacente. A parte da plataforma que é executada nos nós *fog* é chamada de *lightning-rod* e ela é responsável por manter a comunicação com os componentes que estão na nuvem, assim como implementar o protocolo preestabelecido para o registro do nó no sistema. As aplicações são desenvolvidas como *plugins* que são executados nos ambientes de execução mantidos pelos nós *fog*. Os *plugins* são desenvolvidos em código fonte Node.js. Desta forma, o desenvolvedor de uma aplicação projeta e implementa um conjunto de *plugins* para cada aplicação e a plataforma realiza a distribuição destes *plugins* nos diferentes nós considerando suas funções. Por sua vez, o nó mantém o isolamento da execução dos *plugins* de diferentes aplicações fazendo a devida separação dos processos para execução e manutenção do ciclo de vida do código fonte Node.js.

Considerando os requisitos previamente levantados para plataformas *fog computing*, percebemos que a plataforma não apresenta elementos para endereçar os requisitos de suporte a mobilidade, disponibilidade, segurança e privacidade, e eficiência energética.

[11] Protocolo WebSocket - https://tools.ietf.org/html/rfc6455

Assim como na plataforma FogFlow, o requisito escalabilidade também é atendido parcialmente pois a plataforma não oferece mecanismos para escalar automaticamente todo o sistema. Ademais, o desenvolvedor do *plugin* deverá conhecer os detalhes dos dispositivos IoT com que ele vai lidar, de forma a não atender completamente o requisito de heterogeneidade.

1.4.3. Microsoft Azure IoT Edge

Microsoft Azure IoT Edge[12] é um projeto *open source* que disponibiliza uma extensão da plataforma Microsoft Azure IoT para as camadas mais próximas dos dispositivos IoT. A plataforma Azure IoT provê um conjunto de componentes modulares, com interfaces bem definidas e que fornecem serviços genéricos relacionados ao domínio IoT como, por exemplo, serviço de ingestão de dados IoT, processamento de *streams*, armazenamento não estruturado de dados IoT, armazenamento de dados em série temporal, *dashboards* e painéis de visualização de dados IoT, serviço de gerenciamento, controle e comunicação com dispositivos IoT, etc. Tais componentes e seus respectivos serviços podem ser usados de diferentes formas dependendo das necessidades das aplicações. Entretanto, todos os componentes da plataforma Azure IoT são executados na nuvem, sendo primordial para o funcionamento existir conectividade entre os dispositivos IoT e a plataforma centrada na nuvem.

A extensão Azure IoT Edge possibilita a execução de alguns componentes da plataforma em nós *fog*, de forma a expor parcialmente as funcionalidades da plataforma próximo à fonte de dados. Desta maneira, uma aplicação é criada utilizando-se múltiplos módulos (*IoT Edge modules*) que são configurados para comunicarem entre si e também com os dispositivos IoT. Os módulos são compatíveis com imagens Docker e são executados em um ambiente de virtualização mantido por cada nó *fog*. Desta forma, o resultado da execução de um módulo pode ser a entrada para outro módulo, de forma a criar um fluxo de processamento (do inglês *pipeline*).

Outra característica da extensão Azure IoT Edge é possibilitar que algumas funcionalidades possam ser executadas nos dispositivos *fog* mesmo sem a conectividade plena com os demais componentes da nuvem. Desta forma, considerando um cenário onde os nós *fog* estão instalados em locais remotos, que não são facilmente acessíveis e com conectividade intermitente, alguns componentes podem ser executados localmente, na borda da rede, e o resultado de sua execução pode ser enviado para a nuvem assim que a conexão for restabelecida. Por outro lado, se o nó *fog* estiver sem conectividade, os componentes da nuvem podem enviar os módulos das aplicações e suas respectivas configurações assim que o nó *fog* ganhar conectividade.

A arquitetura da plataforma está ilustrada na Figura1.9. O componente *Dispositivos* representa os dispositivos IoT com seus sensores e atuadores. Diferentemente da proposta inicial da plataforma Azure IoT, os dispositivos IoT não estão diretamente ligados à nuvem. Em vez disso, eles estão ligados a um nó *fog*, que atua como um *gateway* que pode traduzir os protocolos de comunicação e formatos de dados usados pelos dispositivos IoT. Para cada dispositivo IoT um módulo (*Module*) correspondente deve ser registrado no nó *fog*. Porém, podem existir módulos que não estão associados a dispositivos IoT

[12]Microsoft Azure IoT Edge Project - https://github.com/Azure/iotedge

como, por exemplo, um módulo que implementa uma regra de negócio da aplicação que necessita do resultado do processamento de outros nós que estão diretamente ligados ao dispositivos IoT. Os módulos são orquestrados pelo componente *IoT Edge Runtime*, que por sua vez estabelece a interface de comunicação para os nós se comunicarem entre eles além de gerenciar toda comunicação do nó *fog* com os demais componentes na nuvem. Outra responsabilidade deste componente é reportar a saúde do nó *fog* e dos módulos em execução para a nuvem. Este módulo contém dois subcomponentes, *IoT Edge Agent* e *IoT Edge Hub*, que serão descritos a seguir.

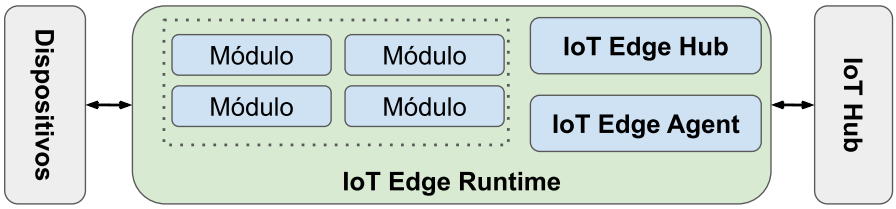

Figura 1.9. Arquitetura da plataforma Microsoft Azure IoT Edge

O componente *IoT Edge Agent* é responsável por baixar da nuvem as informações de implantação dos módulos no nó *fog*. Tais informações são encapsuladas em um manifesto de implantação (do inglês *deployment manifest*) mantido pela plataforma Azure IoT na nuvem e que contém, por exemplo, as configurações necessárias para a execução dos módulos, a imagem Docker de cada módulo, as interfaces de comunicação entre os módulos, etc. Desta forma, o componente *IoT Edge Agent* tem a responsabilidade de garantir que o estado e a configuração dos contêineres estejam de acordo com a definição original associada ao nó *fog*. Para tal finalidade, o componente *IoT Edge Agent* deve constantemente sincronizar as mudanças no manifesto mantido na nuvem pela plataforma, iniciando ou finalizando a execução dos módulos no nó *fog*. Em termos de implementação, este componente também é um módulo, sendo sempre o primeiro a ser executado no ato de inicialização do nó *fog*.

Por fim, o componente *IoT Edge Hub* é um *broker* de comunicação que facilita a comunicação local entre os dispositivos IoT e os módulos. O *IoT Edge Hub* atua também como um *proxy* local para o componente *IoT Hub* que está centrado na nuvem. São suportados diversos protocolos de comunicação IoT, como por exemplo AMQP, MQTT e HTTP. Uma característica desta plataforma é a comunicação segura fim-a-fim.

1.4.4. ParaDrop

ParaDrop [52, 53] é uma plataforma que permite a execução de aplicações em *gateways* sem fio que estão na borda da rede como, por exemplo, Pontos de Acesso Wi-fi (do inglês *Access Point*) e receptores de TV (do inglês *set-top boxes*). Tais *gateways* são os nós *fog* e os dispositivos IoT estão conectados diretamente a eles. A ideia da plataforma é fazer uso dos recursos computacionais dos *gateways* para executar aplicações que são encapsuladas como imagens Docker. Atualmente os *gateways* conectam os dispositivos da rede, estão sempre ligados e possuem poderosos recursos computacionais que, muitas vezes, são subutilizados. Desta forma, na visão dos autores, utilizar os *gateways* como nós *fog* apresenta alguns benefícios como, por exemplo, a privacidade, uma vez que os

dados sensíveis podem ser processados apenas localmente sem a necessidade de ser enviado para outro local. Também, tal solução pode atender o requisito de baixa latência na comunicação com os dispositivos IoT que estão na mesma rede, além de poder operar mesmo na ausência de conectividade com a Internet.

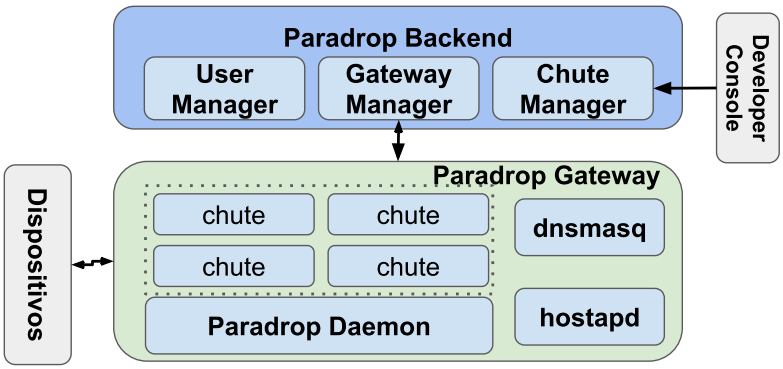

Figura 1.10. Arquitetura da plataforma ParaDrop (adaptado de [52])

Conforme ilustrado na Figura 1.10, a arquitetura da plataforma compreende dois elementos: *Paradrop Backend* e *Paradrop Gateway*. A plataforma centraliza a decisão de distribuição das aplicações em um elemento orquestrador na nuvem chamado de *Paradrop Backend*. Os desenvolvedores criam as aplicações, chamadas de *chute*, por intermédio da ferramenta *Developer Console* ofertada pela plataforma. Tais aplicações são encapsuladas no formato de imagens Docker e em um passo posterior, utilizando a mesma ferramenta, os desenvolvedores submetem as aplicações ao componente *Chute Manager*. O componente *Paradrop Daemon*, que está em cada nó *fog*, recebe o pedido de implantação da aplicação e realiza a configuração necessária para o provisionamento de recursos e instanciação da imagem Docker localmente.

O componente *Paradrop Backend* é o elemento centralizado na nuvem que provê o gerenciamento e comunicação entre os desenvolvedores de aplicações e os nós *fog*. A comunicação entre o *Paradrop Backend* e o nó *fog* é via WAMP (*Web Application Messaging Protocol*) enquanto a comunicação entre o *Developer Console* (Desenvolvedor) e o *Paradrop Backend* é por HTTP. O *Paradrop Backend* se comunica com todos os nós *fog* para encaminhar comandos e receber respostas e relatórios do estado de utilização dos recursos em cada nó. Ele também fornece uma interface Web para a fácil visualização dos nós *fog*, registro de usuários (via componente *User Manager*) e instalação de aplicações.

A plataforma ParaDrop requer que em cada nó *fog* um sistema operacional extremamente leve (Ubuntu Snappy), destinado para dispositivos de recursos limitados, seja utilizado. O Ubuntu Snappy fornece um conjunto de ferramentas e APIs para suportar um ambiente de virtualização leve como o Docker. Desta forma, o componente *Paradrop Daemon*, que está presente em cada nó *fog*, tem a responsabilidade de gerenciar o ambiente de virtualização Docker, controlando o uso dos recursos pelas aplicações em execução, além de controlar a comunicação do nó com a parte da plataforma na nuvem (*Paradrop Backend*), reportando periodicamente o estado de utilização dos recursos e recebendo os pedidos para controle das aplicações como, por exemplo, pedidos para iniciar, parar, ins-

talar e desinstalar as aplicações. Também, este componente é responsável por fazer o registro do nó no sistema no ato da inicialização do nó *fog*. Ainda na Figura 1.10, os componentes *hostapd* (*Host access point daemon*) e *dnsmasq* são responsáveis por assegurar a funcionalidade de ponto de acesso do dispositivo usado como nó *fog* e prover serviços de DNS e DHCP.

Ao considerar os requisitos para plataformas *fog computing* previamente apresentados, percebemos que a plataforma não apresenta elementos para endereçar os requisitos de suporte a mobilidade, disponibilidade, ciência de localização e interoperabilidade. A escalabilidade da plataforma é limitada na medida que o orquestrador não faz distinção entre situações onde o nó *fog* está saturado ou não no ato de implantação de uma aplicação. Outro limitante da plataforma é que ela só pode ser usada em alguns poucos tipos de dispositivos *gateways* que apresentam suporte a programação, desta forma não atendendo por completo o requisito de heterogeneidade.

1.4.5. Comparação entre as plataformas

Esta subseção apresenta uma comparação entre as plataformas discutidas anteriormente usando os requisitos supracitados (veja seção 1.2), a saber: 1) Distribuição Geográfica; 2) Suporte a aplicações sensíveis a latência; 3) Suporte a Heterogeneidade; 4) Interoperabilidade; 5) Otimização do uso da largura de banda; 6) Ciência de Localização; 7) Ciência de Contexto; 8) Suporte a Mobilidade; 9) Disponibilidade; 10) Eficiência Energética; 11) Segurança e Privacidade; 12) Escalabilidade. Na Tabela 1.1 `sim` significa que a plataforma atende completamente o requisito, `não` representa que a plataforma não atende o requisito, enquanto `parcial` significa que o requisito é atendido parcialmente.

Tabela 1.1. Plataformas *fog computing* vs Requisitos

Requisitos/Plataformas	FogFlow	Stack4Things	Azure IoT Edge	ParaDrop
Distribuição Geográfica	sim	sim	sim	sim
Latência	parcial	parcial	sim	sim
Heterogeneidade	sim	parcial	sim	parcial
Interoperabilidade	sim	parcial	parcial	não
Largura de banda	sim	sim	sim	sim
Ciência de Localização	sim	sim	sim	não
Ciência de Contexto	sim	sim	sim	sim
Suporte a Mobilidade	não	não	não	não
Disponibilidade	parcial	parcial	parcial	não
Eficiência Energética	não	não	não	não
Segurança e Privacidade	parcial	não	sim	parcial
Escalabilidade	parcial	parcial	parcial	não

Os dados da Tabela 1.1 demonstram que nenhuma plataforma atende a todos os requisitos que estabelecemos anteriormente (Seção 1.2). Os requisitos Distribuição Geográfica, Largura de Banda e Ciência de Contexto são atendidos por todas as plataformas. Já os requisitos de Suporte a Mobilidade e Eficiência Energética não são atendidos por nenhuma plataforma. Ressalta-se que apenas prover a execução das aplicações em dispositivos de baixo poder computacional não é suficiente para prover uma estratégia de

redução de custo de consumo energético em todo o sistema [33]. Por exemplo, na plataforma ParaDrop os nós *fog* estão sempre ligados, independente de existirem aplicações em execução. Também, nenhuma plataforma considerada neste estudo apresenta estratégias para assegurar a Mobilidade de nós *fog*. Tal característica é essencial no contexto de cidades inteligentes onde diversos dispositivos móveis como, por exemplo, telefones celulares e veículos, podem ser usados para processamento e/ou armazenamento de dados.

O requisito Latência não é atendido completamente pelas plataformas FogFlow e Stack4Things devido ao *overhead* dos protocolos que tais plataformas utilizam para a comunicação. Portanto, essas plataformas não são adequadas para aplicações do tipo tempo real (do inglês *real-time*), mas elas podem ser usadas para aplicações próximas a tempo real (do inglês *near real-time*) quanto existe um relaxamento do tempo de resposta da aplicação. O requisito de Heterogeneidade não é atendido completamente pela plataforma Stack4Things pois ela não abstrai os detalhes dos dispositivos IoT e a forma de acessá-los. Já a plataforma ParaDrop pode ser usada em um conjunto limitado de dispositivos que servirão como nós *fog*. O requisitos de Interoperabilidade envolve necessariamente o uso de padrões amplamente aceitos, o que ocorre apenas na plataforma FogFlow.

O requisito Disponibilidade é endereçado parcialmente por quase todas as plataformas, exceto a plataforma ParaDrop. As estratégias usadas para assegurar o fornecimento dos serviços são baseadas no monitoramento constante da saúde dos nós *fog* a fim de que as plataformas possam considerar o estado do nó no processo de orquestração das aplicações. Não foram mencionadas estratégias tradicionais para assegurar disponibilidade como, por exemplo, a adaptação dinâmica da plataforma e migração das aplicações para outros nós. Por fim, o requisito Segurança e Privacidade é atendido plenamente apenas pela plataforma Microsoft Azure IoT Edge enquanto que o requisito Escalabilidade é atendido parcialmente por todas as plataformas.

1.5. Estudo de Caso

Esta seção apresenta uma experiência de implementação de uma aplicação IoT usando o FogFlow como plataforma de processamento distribuído na *fog*. Em um primeiro momento, na subseção 1.5.1, será apresentado o processo de instalação e configuração da plataforma em um ambiente distribuído. Este processo é relativamente semelhante ao processo de instalação de outras plataformas *fog computing* que têm os principais componentes tomadores de decisões centrados na nuvem como, por exemplo, o componente de orquestração das tarefas. Logo em seguida, na subseção 1.5.2, será descrita uma visão abstrata da aplicação em termos de tarefas e a ordem de execução destas tarefas. Na subseção 1.5.3 será descrito sucintamente os modelos de programação da plataforma para representar a topologia de uma aplicação, i.e., como a plataforma representa as tarefas de uma aplicação. Na subseção 1.5.4, serão discutidos os operadores que implementam as tarefas e o cenário usado para execução do estudo de caso, apresentando os dispositivos utilizados como nós *fog*. Por fim, na Seção 1.5.5 serão apresentados os resultados obtidos nos testes de execução do estudo de caso, considerando a métrica tempo de processamento demandado pelos operadores que implementam as tarefas da aplicação.

1.5.1. Instalação e configuração da plataforma FogFlow

Como já explanado na seção 1.4.1, o FogFlow define a lógica de processamento dos dados através de um DAG de tarefas. Cada vértice deste DAG representa uma tarefa (parte do processamento como um todo) e as arestas definem as interações de entrada e saída de dados entre estas tarefas. Como em um *pipeline* de processamento, o resultado (saída) de uma tarefa é uma entrada para outra tarefa. Cada tarefa é implementada através de um operador na forma de um container Docker. Como um container Docker é dependente da plataforma de *hardware* naqual ele será executado, haverá uma imagem do container do operador para cada tipo de nó *fog* (atualmente são suportadas as arquiteturas ARM e X86). A plataforma faz a orquestração dos operadores nos nós *fog* considerando as capacidades e a localização geográfica do nó em relação a fonte de dados. Os operadores executam o protocolo NGSI e se comunicam por troca de mensagens assíncronas no paradigma *publish/subscribe* assinando e publicando entidades de contexto via *IoT Broker*. O protocolo NGSI determina que as mensagens sejam compatíveis com JSON (*JavaScript Object Notation*) e transmitidas utilizando o protocolo HTTP.

Os componentes da plataforma Fogflow são disponibilizados como imagens Docker e organizados em documento Docker-compose[13], que por sua vez define os parâmetros de configuração para a execução dos contêineres. Esses parâmetros especificam as portas que serão utilizadas por cada componente, bem como as interfaces de rede internas (ou interfaces virtualizadas Docker) de comunicação entre esses componentes. A instalação da plataforma é dividida em duas partes, sendo que cada parte compreende um conjunto de passos que devem ser seguidos em ordem. A Figura 1.11 ilustra as duas partes e segue a nomenclatura proposta pela plataforma. Na parte *cloud* é realizado a instalação dos componentes que tipicamente executam na nuvem. Na parte *Edge* é realizada a instalação dos componentes nos dispositivos que serão usados como nós *fog*. Os componentes da parte *cloud* podem ser instalados em microcomputadores ou microservidores na borda da rede, desde que esses tenham recursos computacionais suficientes e sejam compatíveis com a plataforma X86. Por sua vez, os dispositivos que serão usado como nós *fog* podem ter a arquitetura X86 e ARM e tipicamente possuem recursos computacionais limitados.

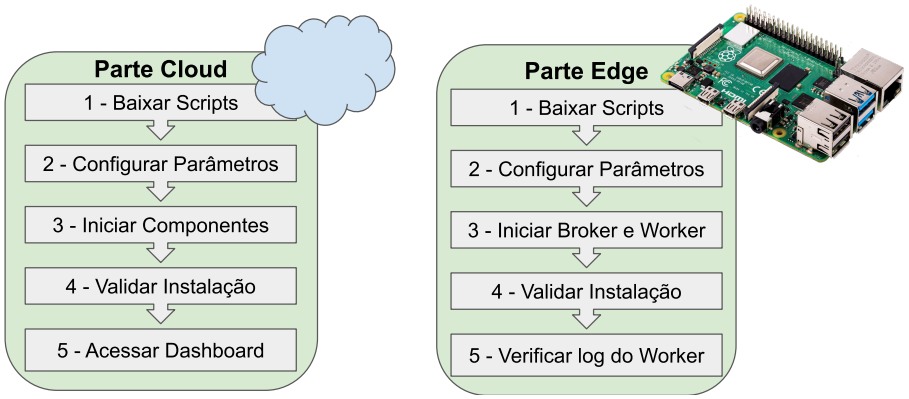

Figura 1.11. Processo de instalação da plataforma FogFlow

O primeiro passo na instalação da parte *cloud* envolve o *download* diretamente da

[13]Docker-compose - https://docs.docker.com/compose/

conta GitHub[14] do FogFlow do arquivo Docker-compose e dos arquivos de configuração da plataforma (*config.json*) e do servidor Web Nginx (*nginx.conf*). No arquivo de configuração *config.json* é necessário inserir os endereços IP da máquina que executará a parte *cloud* e da rede interna Docker formada para a comunicação dos componentes da plataforma, além das portas que serão usadas por cada componente. Já o arquivo *nginx.conf* define os parâmetros para a execução do servidor Web utilizado pela plataforma para expor suas funcionalidades. No passo seguinte, Iniciar Componentes, os componentes (imagens Docker) serão inicializados com o Docker-compose (comando `docker-compose up`). O próximo passo, *Validar Instalação*, visa verificar se os componentes foram devidamente inicializados. Basicamente, deve-se verificar se os contêineres foram iniciados (comando `docker ps`) e se a rede interna Docker foi montada de forma apropriada.

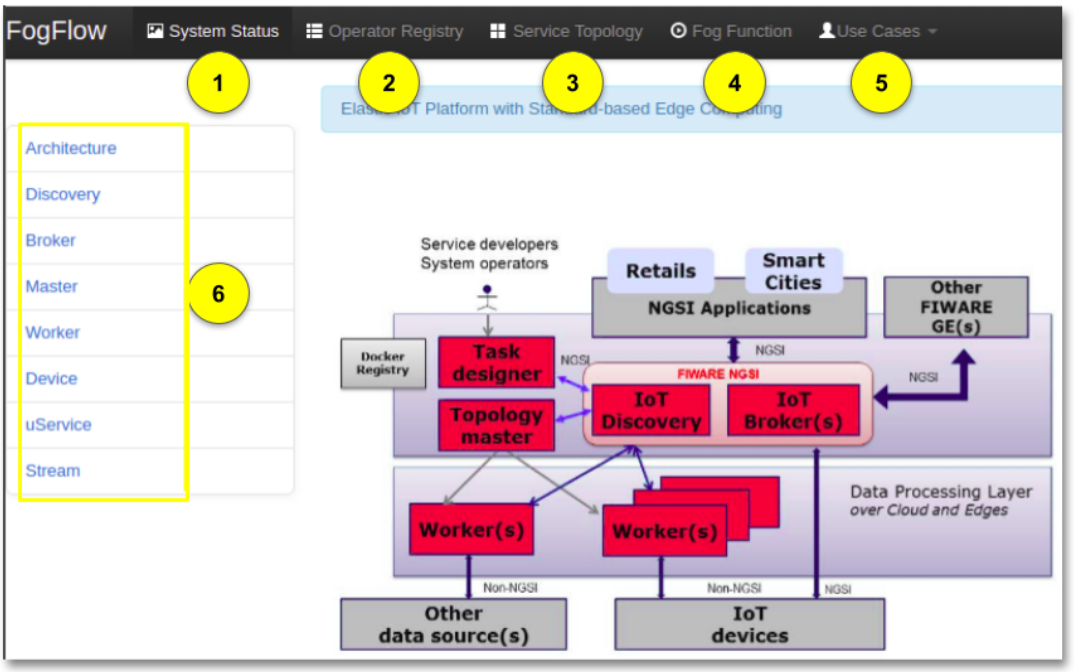

Figura 1.12. Interface gráfica de gerenciamento da plataforma FogFlow

A plataforma oferece uma interface gráfica Web, ilustrada na Figura 1.12, para que os desenvolvedores possam especificar as aplicações e verificar o estado dos elementos da plataforma, inclusive a carga de trabalho de cada nó *fog*. Nas abas do painel superior da interface gráfica o usuário tem acesso, respectivamente, ao estado geral da plataforma (círculo 1), a funcionalidade de cadastro de operadores que definem o comportamento das tarefas (círculo 2), a interface para criação de aplicações utilizando-se o modelo de programação *service topology* (círculo 3), a interface para criação de serviços utilizando-se o modelo de programação *fog function* (círculo 4), e alguns estudos de caso que a plataforma fornece (círculo 5). Por fim, na região definida no círculo 6 o usuário tem acesso a algumas informações dos componentes da plataforma e das entidades de contexto que são mantidas pelo *broker*. Por exemplo, são fornecidas informações a respeito da arquitetura da plataforma (*link Architecture*), o endereço (*endpoint*) do componente *IoT*

[14]Conta GitHub do FogFlow - https://github.com/smartfog/fogflow

Discovery (*link Discovery*), o endereço do *broker* (*link Broker*), informações da parte *cloud* (*link Master*), informações das aplicações em execução nos nós *fog*, bem como a localização geográfica em um mapa dos nós (*link Worker*), informação dos dispositivos produtores de dados que estão publicando entidades de contexto no *broker* (*link Devices*), e informações das entidades de contexto que são geridas pela plataforma. Os modelos de programação providos pela plataforma, *service topology* e *fog function*, serão descritos mais adiante neste texto.

Na instalação da parte *Edge* deve-se, no primeiro passo, fazer o *download* do *script* de inicialização (*start.sh*) e do arquivo de configuração (*config.json*) dos componentes que executarão nos nós *fog*. O passo seguinte compreende a edição do arquivo *config.json* com a configuração adequada do nó *fog*. Conforme ilustrado na Figura 1.13, no arquivo *config.json* devem ser especificados o endereço IP (atributo `coreservice_ip`) do componente *Orquestrador* (instalado anteriormente na parte *cloud*), o identificador (atributo `site_id`) e as coordenadas geográficas do nó *fog* (atributo `physical_location`), a capacidade (atributo `capacity`) que o nó possui para executar contêineres em paralelo, além de outras configurações. O passo seguinte do processo de instalação consiste na execução do *script start.sh*, que por sua vez configura e executa o contêiner Docker que possui os componentes da parte *Edge*. Por fim, os últimos passos constituem-se da checagem dos contêineres para verificar se a comunicação com os demais componentes distribuídos está ocorrendo adequadamente.

```
{
  "coreservice_ip": "10.7.40.146",
  "external_hostip": "10.7.128.15",
  "internal_hostip": "10.7.128.15",
  "physical_location": {
    "latitude": 35.14703,
    "longitude": 136.786218
  },
  "site_id": "003",
  "logging": {
    "info": "stdout",
    "error": "stdout",
    "protocol": "stdout",
    "debug": "stdout"
  },
  "discovery": {
    "http_port": 80,
    "https_port": 443
  },
  "broker": {
    "http_port": 8070,
    "https_port": 443
  },
  "worker": {
    "container_autoremove": false,
    "start_actual_task": true,
    "capacity": 6
  }
```

Figura 1.13. Trecho do arquivo de configuração do nó *fog* da plataforma FogFlow

Os passos executados na parte *edge* devem ser repetidos para cada dispositivo que servirá como nó *fog*.

1.5.2. Planejamento Lógico das Tarefas

O cenário do estudo de caso é de processamento de imagens de câmeras de monitoramento, realizado por nós *fog* próximos destas câmeras, atendendo a requisitos de baixa latência e alta vazão de dados (do inglês *throughput*). Para reduzir a latência e aumentar o *throughput*, alguns nós *fog* são equipados com *hardwares* aceleradores com desempenho significativo no processamento de imagens para a detecção e identificação de objetos de interesse. No contexto de um prédio inteligente (do inglês *smart building*), a implementação focou em uma aplicação que permite abrir automaticamente a porta de uma sala quando a câmera instalada na frente desta porta identifica a face (rosto) de uma pessoa como autorizada para tal. As características da faces das pessoas autorizadas foram previamente guardadas em um banco de dados. Assim, a divisão do processamento foi feito com as seguintes tarefas:

- **Face Detection (FD)** - detecta faces no fluxo de vídeo proveniente da câmera gerando uma imagem de interesse para o fluxo de processamento;
- **Face Recognition (FR)** - reconhece as pessoas na imagem de interesse a partir das faces detectadas gerando eventos de interesse;
- **Business Rule (BR)** - emprega a regra de negócio da aplicação no evento de interesse detectado. Nesse caso, a regra de negócio é abrir ou negar a abertura da porta e registrar o evento (detecção de pessoa).

A Figura 1.14 mostra o planejamento lógico das tarefas de processamento. A tarefa FD tem a função de filtrar do fluxo de vídeo da câmera apenas os quadros (imagens) que possuem uma ou mais faces de pessoas, repassando para a tarefa seguinte esta imagem com as coordenadas do(s) retângulo(s) contendo a(s) face(s) detectada(s), bem como a identificação da câmera que gerou tal imagem. Em seguida, a tarefa FR recorta do quadro capturado cada (retângulo de) face e busca em um banco de pessoas previamente conhecidas as faces com as mesmas características, identificando a pessoa daquela face. Por fim, a tarefa BR aplica a ação apropriada para o evento analisado e interage com a aplicação que é executada na nuvem.

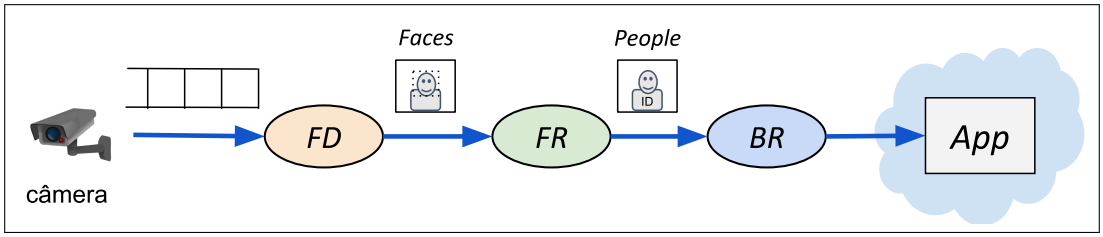

Figura 1.14. Planejamento lógico das tarefas do estudo de caso

A Figura 1.15 ilustra os resultados parciais obtidos do fluxo de processamento da imagem de interesse, desde sua obtenção na tarefa FD a partir do fluxo de vídeo, até a identificação da face na imagem de interesse na tarefa FR.

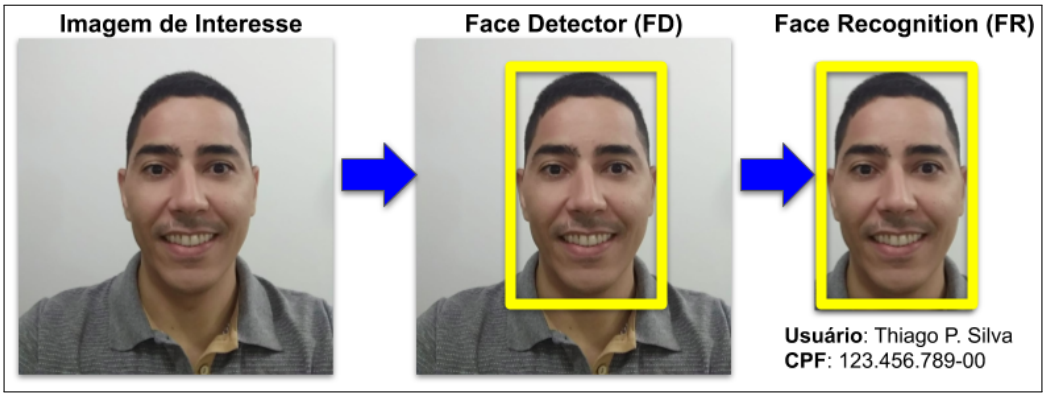

Figura 1.15. Exemplo dos resultados parciais do fluxo de processamento de uma imagem de interesse

1.5.3. Criação da Topologia de Serviço

No FogFlow, o DAG que representa o fluxo de tarefas de processamento de uma aplicação é chamado de topologia de serviço (do inglês *service topology*). Na topologia do serviço cada tarefa de processamento deve ser implementada através de um operador (do inglês *operator*). Na plataforma, o modelo de programação adotado impõe que cada tarefa que compõe uma aplicação é executada apenas quando surgir no sistema uma entidade de contexto associada à tarefa ou ocorrer uma atualização desta entidade de contexto. Desta forma, os operadores que implementam tais tarefas são tidos como dirigidos por contexto (do inglês *context-driven*) e obrigatoriamente cada operador tem uma entidade de contexto que dispara a sua execução. Por vez, o resultado do processamento de um operador pode ser uma entidade de contexto que dispara a execução de outro operador. Assim, a semelhança da topologia de serviço com um DAG é que os vértices representam as tarefas, com seus respectivos operadores, e as arestas representam as entidades de contexto que determinam a ordem de execução das tarefas.

A plataforma define além do modelo de programação *service topology* outro modelo chamado de *fog function*, que é uma versão simplificada de um topologia de serviço com apenas uma tarefa e, por conseguinte, apenas um operador. Uma *fog function* obrigatoriamente tem uma entidade de contexto que dispara a sua execução e opcionalmente pode gerar como saída do processamento outra entidade de contexto. Neste sentido, uma topologia de serviço pode ser enxergada como um conjunto de *fog functions* que são interrelacionadas pelas entidades de contexto. Desta forma, a plataforma fornece ao desenvolvedor duas maneiras para criar aplicações IoT: a primeira usando *fog function* com a definição da aplicação em apenas uma tarefa; ou usando topologia de serviço com a quebra do processamento várias tarefas.

Uma entidade de contexto, que representa uma unidade de dado a ser processada, é chamada na plataforma de *entity stream*. A tarefa FD é quem inicia o fluxo de processamento, publicando a *entity stream* chamada *Faces*. Uma *entity stream* é compatível com o formato JSON (*JavaScript Object Notation*), conforme ilustrado na Figura 1.16, e obrigatoriamente precisa ter um identificador único (`id`) e um tipo (`Type`). O formato JSON originalmente não permite o transporte de conteúdos binários e, portanto, a imagem de interesse capturada pelo operador FD - que é uma arquivo binário - não é embutida na

entity stream. Para contornar esta limitação e não perder desempenho, a estratégia adotada foi manter um servidor HTTP simples e leve que é gerenciado pelo operador FD. Desta forma, as imagens de interesse são obtidas do fluxo de vídeo, armazenadas localmente no dispositivo *fog* que executa o operador FD e compartilhadas pelo servidor HTTP. O atributo `image_url` da *entity stream* contém a URL para que os outros operadores possam recuperar esta imagem. Um outro atributo importante desta entidade de contexto é `faces`, que é um vetor de elementos contendo inicialmente apenas o *bounding box* (coordenadas do retângulo de pixels) envolvendo cada `face` detectada na imagem. Posteriormente serão acrescentados a este vetor faces, pelo operador FR, dados sobre a identificação da face.

```
[
  {
    "entityId": {
      "type": "Devices.Faces.imd-cam-02",
      "isPattern": false,
      "id": "Faces.id"
    },
    "attributes": [
      {
        "name": "image_url",
        "type": "string",
        "value": "http://10.7.16.34.80:88/img/15022272633968.jpg"
      },
      {
        "name": "faces",
        "type": "array",
        "value": [
          {
            "box": [
              358,
              84,
              410,
              160
            ]
          }
        ]
      }
    ]
  }
]
```

Figura 1.16. Exemplo de conteúdo da entidade de contexto `Faces`

A definição do fluxo de processamento das *entity streams* é feito através da ferramenta projeto de topologia de serviço (do inglês *designer service topology*) na interface de gerenciamento do FogFlow (círculo 3 da Figura 1.12). Na Figura 1.17, vemos a definição da topologia de serviço (*service topology*) criada para o estudo de caso. Perceba que na representação gráfica os elementos **Task** (em português Tarefa), **Shuffle** e **EntityStream** são representados como retângulos e o fluxo do processamento é definido com as setas que interligam estes elementos. Ademais, esses elementos possuem atributos essenciais utilizados pelo componente *Orquestrador* na decisão de orquestração dos operadores que implementam as tarefas.

No elemento **EntityStream**, o atributo `SelectedType` representa o tipo da entidade de contexto e o atributo `SelectedAttributes` especifica quais atributos da entidade de contexto deverão ser encaminhados para a Tarefa. O atributo `Groupby` especifica a granularidade da entidade de contexto e é usada para definir a quantidade de instâncias do operador relacionado à Tarefa. Por exemplo, considerando uma entidade de contexto do tipo *Pessoa* com o atributo `Groupby` especificado como *EntityID*, então uma instância do operador será executada para cada *id* diferente da entidade de contexto *Pessoa*. Por outro lado, se o atributo `Groupby` for especificado como *ALL*, então apenas uma instância do operador será mantida para todas as entidades de contexto do tipo *Pessoa*, independente do *id* que as entidades possam ter. Já o atributo `Scoped` representa um valor booleano que indica se a informação de localização geográfica está contida na entidade de contexto. Por exemplo, a entidade de contexto que representa uma imagem de interesse capturada por uma câmera, terá o valor do atribuito `Scoped` será setada como Verdadeiro (do inglês *True*), pois a câmera é um sensor produtor de dados fixo que tem sua localização conhecida.

No elemento **Task** o atributo `Name` especifica o nome da tarefa enquanto o atributo `Operator` indica a imagem Docker que implementa tal tarefa. Também, neste elemento, deve ser especificado a saída (do inglês *Output*) gerada pela tarefa. Isto é, qual é a entidade de contexto que será publicada no *broker* após a execução da tarefa, de modo a continuar o fluxo de processamento. Por fim, o elemento **Shuffle** é usada para interligar dois elementos do tipo *Task* (ou em português Tarefa), quando a primeira *Task* produz uma entidade de contexto que é a entrada para a segunda *Task*.

Desta forma, para o estudo de caso foi definida a tarefa *Face_Detection_Task* (FD), que é implementada pelo operador *face_detection_operator*. Este operador inicia o fluxo de processamento do *stream* proveniente da câmera e gera a entidade de contexto *Faces*. Entretanto, devido ao modelo de programação adotado pela plataforma, o início da execução deste operador está atrelado necessariamente ao surgimento da entidade de contexto do tipo *Start*. Desta forma, definimos a entidade de contexto `Start` e atribuímos a propriedade `Groupby` com o valor *EntityID*, de modo que para cada câmera uma instância do operador *face_detection_operator* seja criado.

A tarefa *Face_Recognition_Task* (FR) é implementada pelo operador *face_recognition_operator* e recebe como entidade de contexto o tipo *Faces* e executa o reconhecimento facial, produzindo como resultado a entidade de contexto do tipo *People*, a qual é similar a *Faces* mas contém o atributo `person_id` para cada elemento do vetor do atributo `faces` original. O atributo `person_id` informa uma chave de identificação única de uma pessoa dentro do banco de dados de pessoas conhecidas, como o CPF, por exemplo, ou a palavra *unknown* para indicar que a pessoa não foi localizada neste banco.

Como vimos anteriormente, a comunicação das *entity streams* entre operadores é representada pelo elemento de interface *Shuffle*. Nesse caso, definimos esse agrupamento como *ALL* de forma que todas as entidades de contexto dos tipos *Faces* e *People* serão passados para o próximo operador no fluxo de processamento. A terceira e última tarefa de processamento *Door_Control_Task* é implementada pelo operador *door_control_operator*. Este operador então, ao receber a entidade de contexto `People`, aplica a regra de negócio da aplicação permitindo ou negando a abertura da porta vinculada à câmera. Em

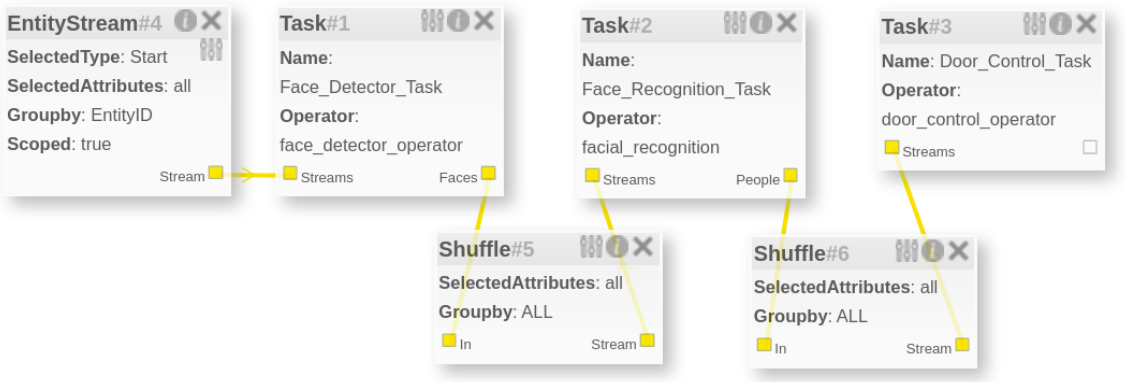

Figura 1.17. Definição da Topologia de Serviço do estudo de caso

linhas gerais, *door_control_operator* precisa ter um banco de dados com a identificação das pessoas que estariam autorizadas a entrarem naquela sala associada com aquela câmera/porta, e procurar a pessoa identificada nesse banco. De qualquer forma, a decisão deste operador é notificada ao usuário através de um pequeno *display* na porta, no qual ele vê sua face sendo capturada e, caso necessário, ele aciona a fechadura eletrônica da porta para abri-la. Este banco de dados é encapsulado junto com o código fonte da tarefa no contêiner do operador.

1.5.4. Cenário e Configuração dos nós fog usados para o processamento distribuído

Para a execução das tarefas de processamento de imagens preparamos dois dispositivos equipados com aceleradores otimizados para a execução de algoritmos de inteligência artificial, conforme ilustrado na Figura 1.18. O primeiro dispositivo é um Raspberry Pi[15] 4 (CPU ARM quad-core, 4GB de memória RAM) equipado com o acelerador Google Coral[16] Edge TPU (*Tensor Processing Unit*) em uma porta USB-3. O segundo dispositivo é um NVIDIA Jetson Nano[17] (CPU ARM quad-core, 4GB de memória RAM), que possui uma GPU (*Graphics Processing Unit*) NVIDIA Pascal 128-cores já integrada em seu *hardware*. Esse TPU é capaz de executar 4 TOPS (trilhões de operações por segundo) consumindo apenas 2 Watts de energia. Já a Jetson, é capaz de atingir 472 GFLOPS (bilhões de operações com ponto flutuante por segundo) e processar a detecção de objetos em até 8 fluxos de vídeo simultâneos com resolução Full-HD (1920 x 1080 pixels). Os resultados obtidos por esses *hardware* mostram a rápida evolução no desenvolvimento dos dispositivos chamados pela indústria como dispositivos *edge* (do inglês *edge devices*), alcançando capacidade computacional expressiva e que permitem uma certa independência de soluções baseadas na nuvem. Esse aspecto é um dos motivos para o desenvolvimento de aplicações na *Fog*.

O dispositivo Raspberry Pi 4 com seu acelerador Coral ficou com a tarefa *Face_Detection_Task* (FD) no fluxo de vídeo proveniente da câmera. Para esse processamento, foi usado o modelo de rede neural conhecido como MobileNet [54], a qual foi desen-

[15]Raspberry - https://www.raspberrypi.org/

[16]Google Coral - https://coral.ai/

[17]NVIDIA Jetson Nano - https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-nano/

Figura 1.18. Dispositivos usados como nó *fog*: (a) Raspberry Pi 4 e Google Coral USB e (b) NVIDIA Jetson Nano

volvida pela Google especialmente para ser usada em dispositivos móveis e embarcados da IoT. Já na Jetson, que ficou com a tarefa *Face_Recognition_Task* (FR), foi usado o modelo de rede neural chamado FaceNet, que aprende um mapeamento de imagens de rosto para um espaço euclidiano compacto, onde as distâncias correspondem diretamente a uma medida de semelhança de rosto [55].

Além destes dois dispositivo, o ambiente ainda contou com um computador laptop onde foram instalados os componentes da parte *cloud* do FogFlow. O operador correspondente a tarefa *Door_Control_Task*, por ter um algoritmo simples e que não faria diferença se fosse executado na *fog* ou na nuvem, foi executado neste computador laptop. A configuração do laptop é um Dell com processador Intel Core i7, 16 GB de memória RAM e placa gráfica dedicada NVIDIA MX150 com 2GB de memória. No cenário montado todos os dispositivos pertencem à mesma rede local.

Não entraremos em detalhes a respeito das técnicas utilizadas na implementação dos operadores que realizam a detecção e identificação das faces, pois estes detalhes estão fora do escopo do minicurso. Entretanto, o código fonte dos operadores está disponível no repositório GitHub[18] e a documentação necessária para criar operadores em linguagem Java, Python e JavaScript pode ser obtida na documentação da plataforma FogFlow[19].

1.5.5. Resultados Obtidos

Depois de tudo configurado e instanciado, tanto na parte *cloud* quanto na parte *fog*, iniciamos o experimento de processar o fluxo de vídeo de uma câmera de monitoramento posicionada em frente a uma porta com abertura automatizada. Esta câmera gerava imagens Full-HD (1920 x 1080 pixels) a 25 quadros por segundo com compressão H.264. Foi definida uma *região de interesse* dentro de cada quadro de 500 x 500 pixels representando a posição ideal de captura da face das pessoas. O operador *face_detection_operator* executando no Raspberry Pi 4 acessava o *stream* desta câmera via protocolo RTSP (*Real Time Streaming Protocol*) e passava cada imagem da região de interesse pela rede neural de detecção de faces rodando na TPU (Coral USB). Ao detectar alguma face nesta imagem, *face_detection_operator* publicava a entidade de contexto do tipo *Faces* (re-

[18]Código fonte dos operadores do estudo de caso- https://github.com/thiagopereirasilva/sbrc2020
[19]Documentação da plataforma FogFlow - https://fogflow.readthedocs.io/en/latest/index.html

presentada na Figura 1.16). Em seguida, o operador *face_recognition_operator* recebe a entidade de contexto *Faces*, obtém a imagem JPEG via requisição HTTP ao operador *face_detection_operator*, passa essa imagem pela rede neural rodando em sua GPU para extrair as características da face ali presente, e busca em um banco de dados de faces de pessoas conhecidas, aquelas que são similares à face da imagem. Ao final deste processo, o operador *face_recognition_operator* publica a entidade de contexto do tipo *People* no *broker*. Por último, o operador *door_control_operator* executando no mesmo computador laptop da parte *cloud* do FogFlow, encerrava o processamento gerando a ação de permitir ou negar a abertura da porta.

A Figura 1.19 mostra os tempos que cada operador tomou no processamento do *stream* e na publicação da entidade de contexto resultante do seu processamento. Para o cálculo destes tempos, desenvolvemos um protocolo próprio de contagem de tempo com relógio global rodando em um servidor HTTP na parte *cloud*. O tempo de processamento do primeiro operador *face_detection_operator* foi calculado a partir do instante em que uma pessoa se posiciona na frente da câmera, de forma que a sua face aparece por inteiro na imagem, até o momento de publicar a entidade de contexto do tipo *Faces* no *broker*. O tempo do operador *face_recognition_operator* é contado desde a publicação de *Faces* até este publicar *People*. E por último, o tempo do operador *door_control_operator* é o intervalo entre a publicação de *People* e a tomada de decisão para controlar a porta. Neste gráfico de tempo acumulativo (Figura 1.19), o eixo vertical representa o tempo em segundos necessário para o processamento de cada tarefa pelo operador, enquanto o eixo horizontal representa as iterações executadas. O gráfico ilustra que em todas as 50 interações o tempo total de processamento foi inferior a 0,16 segundos (160 milissegundos) e que o operador *face_detection_operator* foi aquele que demandou o maior tempo de processamento.

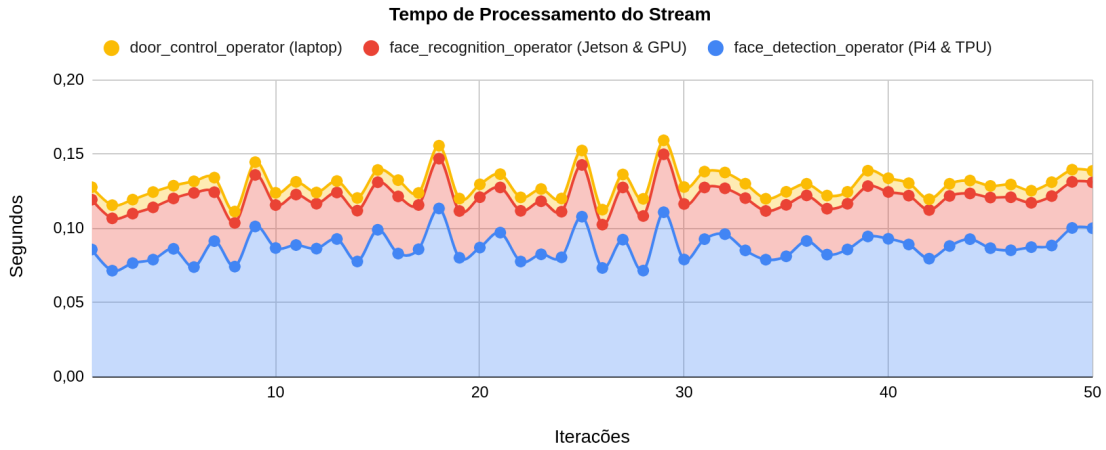

Figura 1.19. Tempo de processamento dos operadores para o processamento do sream de video

A Tabela 1.2 mostra os dados estatísticos sobre estes tempos, revelando um bom tempo máximo de processamento, uma vez que também foram computados também todos os tempos de troca de mensagens entre os nós da plataforma. Isto é, não descontamos o tempo necessário para comunicação no *broker*. Dado o tempo médio de 129 milissegundos, se considerarmos que em torno de 1 segundo seja um tempo máximo razoável para

que a pessoa na frente da porta espere esta abrir, poderíamos ter até 8 conjuntos câmera-porta a serem controlados pelo FogFlow com esta topologia e dispositivos *fog*. Além disso, essa quantidade de câmeras pode ser muito maior, pois nesse cálculo da quantidade de câmeras consideramos que a cada 1 segundo haverá uma nova pessoa na frente de cada porta, e isso é uma situação muito rara de acontecer. Ou seja, a verdadeira capacidade de processamento deste experimento em termos de quantidade de câmeras para processar vai depender do volume de eventos que acontecem em cada câmera.

Tabela 1.2. Estatísticas dos tempos de processamento em segundos por cada operador

	FD	**FR**	**DC**	**Total**
Média	0,087561	0,033456	0,008885	0,129902
Mínimo	0,071595	0,028977	0,007228	0,111474
Máximo	0,113525	0,050001	0,011576	0,159481
Desvio Padrão	0,009567	0,003342	0,001014	0,009792

Uma comparação rápida com uma solução cujo processamento do *stream* é feito todo na nuvem, mostra que somente o tempo de transmissão do vídeo para o servidor na nuvem já supera o tempo de processamento na *fog*. Considerando que uma transmissão de vídeo, comprimido em H.264, Full-HD e 20 frames por segundo, gera um *bitrate* de até 30 Mbps, e que a velocidade final entre a câmera e o servidor na nuvem é de 10 Mbps, temos que:

tamanho de um simples quadro = 30 Mbps / 20 fps = 1,5 Mb
tempo de transmissão de um quadro = 1,5 Mb / 10 Mbps = 0,150 segundos

Ou seja, seriam necessários 150 milissegundos para que cada quadro chegasse até o servidor na nuvem, o que já ultrapassa o tempo total de processamento de nosso experimento na *fog*. Portanto, consideramos que, para este tipo de processamento de vídeo monitoramento com IoT, o FogFlow se mostrou uma plataforma bastante promissora.

1.6. Conclusão

Este capítulo discorreu sobre plataformas computacionais de *Fog Computing*. Inicialmente foram definidos os requisitos essenciais para uma plataforma de *fog computing* com base em vários trabalhos recentes reportados na literatura da área. Logo em seguida, foram apresentadas duas Arquiteturas de Referência (AR) para *fog computing*, enfatizando a necessidade de padronização para área a fim de proporcionar interoperabilidade entre as plataformas existentes. As duas AR apresentadas são baseadas na arquitetura para *fog computing* proposta por [17] que, na época, os autores vislumbraram *fog computing* como a extensão das capacidades da nuvem para a borda da rede e propuseram uma arquitetura formada por três camadas: i) a camada inferior formada pelos dispositivos IoT que são capazes de sensoriar e atuar no ambiente em questão; ii) a camada intermediária, que é a camada *fog computing*, responsável pelo processamento dos dados localmente e quando da indisponibilidade de processá-los, por encaminhá-los para a nuvem; por fim, iii) a camada superior, formada pelos dispositivos de maior poder computacional na nuvem.

Foram apresentadas algumas plataformas de código aberto, detalhando a organização dos seus componentes e apresentando-se uma análise breve de como atendem (ou não) os requisitos previamente levantados. As plataformas atuais são em sua maioria específicas de domínio e, por conseguinte, possuem diferentes características e atendem também a requisitos inerentes ao domínio onde estão inseridas. A comparação entre as plataformas incluídas neste capítulo revelou que os requisitos de suporte a mobilidade e eficiência energética não são atendidos pelas plataformas. Já o requisito segurança e privacidade é atendido plenamente apenas por uma plataforma enquanto o requisito de disponibilidade é atendido parcialmente pelas plataformas. Desta maneira, percebe-se que muitas questões ainda estão em aberto no paradigma *Fog Computing*, o que torna esta área um campo fértil com várias oportunidades de pesquisa.

Por fim, vimos a viabilidade do paradigma em um estudo de caso para processamento de vídeo em um ambiente distribuído utilizando dispositivos que são coordenados por uma plataforma *fog computing*. Aplicações de análise de vídeo geralmente requerem recursos computacionais robustos para sua execução, uma vez que tais aplicações lidam com uma quantidade grande de dados (*stream* de vídeo). Entretanto, no estudo de caso foi demonstrado o uso de dispositivos de baixo custo e poder computacional para executar tarefas que compõem uma aplicação de processamento de vídeo atendendo os requisitos de baixa latência e redução do uso da largura de banda.

Referências

[1] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The internet of things: A survey. *Computer Networks*, 54(15):2787 – 2805, 2010.

[2] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi. Internet of things for smart cities. *IEEE Internet of Things Journal*, 1(1):22–32, 2014.

[3] Charith Perera, Yongrui Qin, Julio C. Estrella, Stephan Reiff-Marganiec, and Athanasios V. Vasilakos. Fog computing for sustainable smart cities: A survey. *ACM Comput. Surv.*, 50(3):32:1–32:43, June 2017.

[4] Thiago Teixeira, Sara Hachem, Valérie Issarny, and Nikolaos Georgantas. Service oriented middleware for the internet of things: A perspective. In Witold Abramowicz, Ignacio M. Llorente, Mike Surridge, Andrea Zisman, and Julien Vayssière, editors, *Towards a Service-Based Internet*, pages 220–229, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[5] Flavia C. Delicato, Paulo F. Pires, and Thais Batista. *Middleware Solutions for the Internet of Things*. Springer Publishing Company, Incorporated, 2013.

[6] M. Aazam, I. Khan, A. A. Alsaffar, and E. Huh. Cloud of things: Integrating internet of things and cloud computing and the issues involved. In *Proceedings of 2014 11th International Bhurban Conference on Applied Sciences Technology (IBCAST) Islamabad, Pakistan, 14th - 18th January, 2014*, pages 414–419, 2014.

[7] Bin Cheng, Guerkan Solmaz, Flavio Cirillo, Ernoe Kovacs, Kazuyuki Terasawa, and Atsushi Kitazawa. FogFlow: Easy Programming of IoT Services Over Cloud and

Edges for Smart Cities. *IEEE INTERNET OF THINGS JOURNAL*, 5(2, SI):696–707, APR 2018.

[8] M. Chiang and T. Zhang. Fog and iot: An overview of research opportunities. *IEEE Internet of Things Journal*, 3(6):854–864, 2016.

[9] Carlo Puliafito, Enzo Mingozzi, Francesco Longo, Antonio Puliafito, and Omer Rana. Fog computing for the internet of things: A survey. *ACM Transactions on Internet Technology*, 19, 04 2019.

[10] Larry Feldman (G2) Robert Barton (Cisco) Michael Martin (IBM Canada) Charif Mahmoudi (NIST) Michaela Iorga (NIST), Nedim Goren (NIST). Fog computing conceptual model. *National Institute of Standards and Technology*, 19, 03 2018.

[11] Pengfei Hu, Sahraoui Dhelim, Huansheng Ning, and Tie Qiu. Survey on fog computing: architecture, key technologies, applications and open issues. *Journal of Network and Computer Applications*, 98:27 – 42, 2017.

[12] Ashkan Yousefpour, Caleb Fung, Tam Nguyen, Krishna Kadiyala, Fatemeh Jalali, Amirreza Niakanlahiji, Jian Kong, and Jason P. Jue. All one needs to know about fog computing and related edge computing paradigms: A complete survey. *Journal of Systems Architecture*, 98:289 – 330, 2019.

[13] IEEE. *IEEE Std 1934-2018: IEEE Standard for Adoption of OpenFog Reference Architecture for Fog Computing*. IEEE, 2018.

[14] O. Vermesan and P. Friess. *Internet of Things: Converging Technologies for Smart Environments and Integrated Ecosystems*. River Publishers Series in Communications Series. River Publishers, 2013.

[15] E. Marin-Tordera, Xavi Masip, Jordi Garcia Almiñana, Admela Jukan, Guang-Jie Ren, Jiafeng Zhu, and Josep Farre. What is a fog node a tutorial on current concepts towards a common definition, 11 2016.

[16] Flavio Bonomi, Rodolfo Milito, Preethi Natarajan, and Jiang Zhu. *Fog Computing: A Platform for Internet of Things and Analytics*, pages 169–186. Springer International Publishing, Cham, 2014.

[17] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, MCC '12, pages 13–16, New York, NY, USA, 2012. ACM. cited By 2351.

[18] Arif Ahmed, HamidReza Arkian, Davaadorj Battulga, Ali J. Fahs, Mozhdeh Farhadi, Dimitrios Giouroukis, Adrien Gougeon, Felipe Oliveira Gutierrez, Guillaume Pierre, Paulo R. R. de Souza, Mulugeta Ayalew Tamiru, and Li Wu. Fog computing applications: Taxonomy and requirements. *ArXiv*, abs/1907.11621, 2019.

[19] R. K. Naha, S. Garg, D. Georgakopoulos, P. P. Jayaraman, L. Gao, Y. Xiang, and R. Ranjan. Fog computing: Survey of trends, architectures, requirements, and research directions. *IEEE Access*, 6:47980–48009, 2018.

[20] Ranesh Kumar Naha, Saurabh Kumar Garg, and Andrew Chan. Fog computing architecture: Survey and challenges. *ArXiv*, abs/1811.09047, 2018.

[21] Sourav Kunal, Arijit Saha, and Ruhul Amin. An overview of cloud-fog computing: Architectures, applications with security challenges. *Security and Privacy*, 05 2019.

[22] Karima Velasquez, David Perez Abreu, Marcio Assis, Carlos Senna, Diego Aranha, Luiz Fernando Bittencourt, Nuno Laranjeiro, Marilia Curado, Marco Vieira, Edmundo Monteiro, and Edmundo Madeira. Fog orchestration for the internet of everything: State-of-the-art and research challenges. *Journal of Internet Services and Applications*, 9, 05 2018.

[23] Redowan Mahmud, Ramamohanarao Kotagiri, and Rajkumar Buyya. *Fog Computing: A Taxonomy, Survey and Future Directions*, pages 103–130. Springer Singapore, Singapore, 2018.

[24] Md Mahmud and Rajkumar Buyya. *Fog Computing: A Taxonomy, Survey and Future Directions*, chapter 1, page 36. 11 2016.

[25] Ali Fahs and Guillaume Pierre. Proximity-Aware Traffic Routing in Distributed Fog Computing Platforms. In *CCGrid 2019 - IEEE/ACM International Symposium in Cluster, Cloud, and Grid Computing*, pages 1–10, Larnaca, Cyprus, May 2019. IEEE.

[26] Lusheng Miao, K. Djouani, B. J. Van Wyk, and Y. Hamam. Performance evaluation of ieee 802.11p mac protocol in vanets safety applications. In *2013 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1663–1668, 2013.

[27] Anind Dey and Gregory Abowd. Towards a better understanding of context and context-awareness. pages 304–307, 01 2000.

[28] C. Puliafito, E. Mingozzi, and G. Anastasi. Fog computing for the internet of mobile things: Issues and challenges. In *2017 IEEE International Conference on Smart Computing (SMARTCOMP)*, pages 1–6, 2017.

[29] T. Nguyen Gia, A. M. Rahmani, T. Westerlund, P. Liljeberg, and H. Tenhunen. Fog computing approach for mobility support in internet-of-things systems. *IEEE Access*, 6:36064–36082, 2018.

[30] Y. Bi, G. Han, C. Lin, Q. Deng, L. Guo, and F. Li. Mobility support for fog computing: An sdn approach. *IEEE Communications Magazine*, 56(5):53–59, 2018.

[31] T. Wang, J. Zhou, A. Liu, M. Z. A. Bhuiyan, G. Wang, and W. Jia. Fog-based computing and storage offloading for data synchronization in iot. *IEEE Internet of Things Journal*, 6(3):4272–4282, 2019.

[32] W. Li, T. Yang, F. C. Delicato, P. F. Pires, Z. Tari, S. U. Khan, and A. Y. Zomaya. On enabling sustainable edge computing with renewable energy resources. *IEEE Communications Magazine*, 56(5):94–101, 2018.

[33] R. Oma, S. Nakamura, T. Enokido, and M. Takizawa. An energy-efficient model of fog and device nodes in iot. In *2018 32nd International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, pages 301–306, 2018.

[34] Gabriel Mujica, Roberto Rodriguez-Zurrunero, Mark Wilby, Jorge Portilla, Ana González, Alvaro Araujo, Teresa Riesgo, and Juan Díaz. Edge and fog computing platform for data fusion of complex heterogeneous sensors. *Sensors*, 18:3630, 10 2018.

[35] Paolo Bellavista, Javier Berrocal, Antonio Corradi, Sajal K. Das, Luca Foschini, and Alessandro Zanni. A survey on fog computing for the internet of things. *Pervasive and Mobile Computing*, 52:71 – 99, 2019.

[36] Upul Jayasinghe, Gyu Myoung Lee, Áine MacDermott, and Woo Seop Rhee. Trustchain: A privacy preserving blockchain with edge computing. *Wireless Communications and Mobile Computing*, 2019:2014697:1–2014697:17, 2019.

[37] M. A. Nadeem and M. A. Saeed. Fog computing: An emerging paradigm. In *2016 Sixth International Conference on Innovative Computing Technology (INTECH)*, pages 83–86, 2016.

[38] Mohit Taneja and Alan Davy. Resource aware placement of data analytics platform in fog computing. *Procedia Computer Science*, 97:153 – 156, 2016. 2nd International Conference on Cloud Forward: From Distributed to Complete Computing.

[39] A.V. Dastjerdi, H. Gupta, R.N. Calheiros, S.K. Ghosh, and R. Buyya. Chapter 4 - fog computing: principles, architectures, and applications. In Rajkumar Buyya and Amir [Vahid Dastjerdi], editors, *Internet of Things*, pages 61 – 75. Morgan Kaufmann, 2016.

[40] M. Aazam and E. Huh. Fog computing micro datacenter based dynamic resource estimation and pricing model for iot. In *2015 IEEE 29th International Conference on Advanced Information Networking and Applications*, pages 687–694, 2015.

[41] X. Masip-Bruin, E. Marín-Tordera, A. Alonso, and J. Garcia. Fog-to-cloud computing (f2c): The key technology enabler for dependable e-health services deployment. In *2016 Mediterranean Ad Hoc Networking Workshop (Med-Hoc-Net)*, pages 1–5, 2016.

[42] Amir Sinaeepourfard, John Krogstie, Sobah Abbas Petersen, and Dirk Ahlers. F2c2c-dm: A fog-to-cloudlet-to-cloud data management architecture in smart city. *2019 IEEE 5th World Forum on Internet of Things (WF-IoT)*, pages 590–595, 2019.

[43] Farhoud Hosseinpour, Yan Meng, Tomi Westerlund, Juha Plosila, Ran Liu, and Hannu Tenhunen. A review on fog computing systems. *International Journal of Advancements in Computing Technology*, 8:48–61, 12 2016.

[44] Wilfried Steiner and Stefan Poledna. Fog computing as enabler for the industrial internet of things. *e & i Elektrotechnik und Informationstechnik*, 133, 10 2016.

[45] Mattia Antonini, Massimo Vecchio, and Fabio Antonelli. Fog computing architectures: A reference for practitioners. *IEEE Internet of Things Magazine*, 2:19–25, 2019.

[46] Elisa Yumi Nakagawa, Flavio Oquendo, and José Carlos Maldonado. *Reference Architectures*, chapter 2, pages 55–82. John Wiley & Sons, Ltd, 2014.

[47] Sourav Kunal, Arijit Saha, and Ruhul Amin. An overview of cloud-fog computing: Architectures, applications with security challenges. *Security and Privacy*, 2(4):e72, 2019.

[48] N. Naik. Choice of effective messaging protocols for iot systems: Mqtt, coap, amqp and http. In *2017 IEEE International Systems Engineering Symposium (ISSE)*, pages 1–7, 2017.

[49] F. Longo, D. Bruneo, S. Distefano, G. Merlino, and A. Puliafito. Stack4things: An openstack-based framework for iot. In *2015 3rd International Conference on Future Internet of Things and Cloud*, pages 204–211, 2015.

[50] Giovanni Merlino, Dario Bruneo, Salvatore Distefano, Francesco Longo, and Antonio Puliafito. Stack4things: Integrating iot with openstack in a smart city context. *Proceedings of 2014 International Conference on Smart Computing Workshops, SMARTCOMP Workshops 2014*, pages 21–28, 02 2015.

[51] S. Distefano, G. Merlino, and A. Puliafito. Sensing and actuation as a service: A new development for clouds. In *2012 IEEE 11th International Symposium on Network Computing and Applications*, pages 272–275, 2012.

[52] P. Liu, D. Willis, and S. Banerjee. Paradrop: Enabling lightweight multi-tenancy at the network's extreme edge. In *2016 IEEE/ACM Symposium on Edge Computing (SEC)*, pages 1–13, 2016.

[53] Suman Banerjee, Peng Liu, Ashish Patro, and Dale Willis. *ParaDrop*, chapter 1, pages 11–23. John Wiley Sons, Ltd, 2017.

[54] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *ArXiv*, abs/1704.04861, 2017.

[55] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun 2015.