Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Implantação de um Ambiente Real de Fog Computing para Monitoramento por meio de Raspberry Pi

William Xavier dos Santos

Monografia apresentada como requisito parcial
para conclusão do Curso de Bacharelado em Ciência da Computação

Orientadora
Prof.a Dr.a Aletéia Patrícia Favacho de Araújo von Paumgartten

Brasília
2023

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Implantação de um Ambiente Real de Fog Computing para Monitoramento por meio de Raspberry Pi

William Xavier dos Santos

Monografia apresentada como requisito parcial
para conclusão do Curso de Bacharelado em Ciência da Computação

Prof.a Dr.a Aletéia Patrícia Favacho de Araújo von Paumgartten (Orientadora)
CIC/UnB

Prof.a Dr.a Edna Dias Canedo
CIC/UnB

Dr. Marcelo Grandi Mandelli
CIC/UnB

Prof. Dr. Marcelo Grandi Mandelli
Coordenador do Curso de Bacharelado em Ciência da Computação

Brasília, 14 de dezembro de 2023

Dedicatória

Com gratidão, primeiramente, dedico este projeto a Deus que me sustentou nos momentos em que eu pensei que estava sozinho. Aos meus filhos Fernando e Pedro que são os motivos de minha existência. À minha esposa Priscilla que, além de cuidar da manutenção do lar enquanto eu permanecia ocupado na confecção deste trabalho, foi capaz de me incentivar e também me orientar no processo inicial de desenvolvimento deste trabalho. Serei eternamente grato por isso. E também aos meus pais, alicerces da minha formação como ser humano, dedico este projeto.

Agradecimentos

Agradeço eternamente à professora Aletéia pela atenção dedicada ao longo de todo o projeto da minha monografia e também por toda a paciência. Professora, o prato não caiu! Muito obrigado por tudo!

Resumo

A *fog Computing* é uma abordagem que traz a computação para mais perto dos dispositivos e usuários. Isso permite que os dados sejam processados e analisados mais rapidamente, o que reduz a latência e melhora a experiência do usuário. A *fog Computing* complementa a Computação em Nuvem, fornecendo recursos e serviços de computação na borda da rede. O monitoramento é uma atividade fundamental para o gerenciamento de recursos, pois garante a disponibilidade e o desempenho dos sistemas. A *fog* é composta por dispositivos pequenos e de baixa potência (chamados de *fog nodes*), o que dificulta a coleta e o armazenamento de dados. Além disso, a *fog* é frequentemente distribuída em ambientes heterogêneos, o que torna a análise dos dados mais complexa. O monitoramento de um ambiente de *fog* é um desafio, pois as soluções existentes são limitadas e podem não atender às necessidades de um serviço de monitoramento completo. Além disso, os provedores de nuvem pública não disponibilizam ainda serviços de acesso para um ambiente de *fog computing*. Isso limita a análise de soluções de monitoramento em um ambiente real. Portanto, este trabalho apresenta a implementação de um ambiente real de *fog computing* por meio de dispositivos do tipo *Raspberry Pi*.

Palavras-chave: Fog Computing, Gerenciamento de Recursos, Monitoramento em Fog, Raspberry Pi

Abstract

Fog Computing is an approach that brings computing closer to devices and users. This allows data to be processed and analyzed further quickly, which reduces latency and improves the user experience. Fog Computing complements Cloud Computing, providing computing resources and services in the edge of the network. Monitoring is a fundamental activity for managing resources, as it guarantees the availability and performance of the systems. The fog is composed by small, low-power devices (called fog nodes), which makes it difficult to data collection and storage. Furthermore, fog is often distributed in heterogeneous environments, which makes data analysis more complex. The monitor Managing a fog environment is a challenge as existing solutions are limited and may not meet the needs of a full monitoring service. Furthermore, public cloud providers do not yet provide access services for a fog computing environment. This limits the analysis of monitoring solutions in a real environment. Therefore, this work presents the implementation of an environment real fog computing through Raspberry Pi-type devices.

Keywords: Fog Computing, Resource Management, Monitoring in Fog, Raspberry Pi

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

A computação em nuvem é uma tecnologia madura, com mais de 15 anos de existência [7]. Ela oferece recursos virtualizados, que são compartilhados entre vários clientes. Os clientes pagam apenas pelo que usam [8], em um modelo de negócios flexível [9]. A computação em nuvem é baseada em grandes centros de dados, distribuídos ao redor do mundo [1]. Esses centros de dados são compostos por milhares de servidores físicos, interconectados por uma rede redundante e estável. A computação em nuvem também é escalável, podendo ser ajustada de acordo com a demanda.

Por outro lado, a Internet das Coisas *(Internet of Things* - IoT) [10] é uma realidade cada vez mais presente em nossas vidas. Com a conectividade cada vez mais acessível, os objetos ao nosso redor estão ganhando inteligência e capacidade de coletar e transmitir dados. Essa nova realidade traz novos desafios, tal como a necessidade de respostas rápidas das aplicações. Isso requer uma comunicação de baixa latência, que não é atendida adequadamente pela computação em nuvem. A computação em nuvem é uma solução centralizada que exige que os dados sejam transmitidos para um servidor remoto. Isso pode gerar latência, que é o tempo decorrido entre o envio de uma mensagem e o recebimento da resposta [11]. Para atender às demandas da IoT, é necessário um paradigma computacional mais próximo do usuário final e dos dispositivos.

Além disso, o cenário da Internet das Coisas é marcado pelo crescimento exponencial do número de dispositivos IoT conectados. Essa tendência traz consigo a necessidade de expandir a capacidade de processamento de rede e de armazenamento para mais perto dos usuários finais. A Computação em Nuvem é uma solução eficaz para essa demanda, mas apresenta a fragilidade de ser centralizada [12], o que pode resultar em latência e baixa disponibilidade.

Nesse cenário, a *fog computing* surgiu como uma alternativa promissora para atender a essa demanda. Essa arquitetura descentralizada permite que o processamento seja realizado na borda da rede, o que reduz a latência e melhora a disponibilidade dos serviços.

Além disso, a *fog computing* é caracterizada por sua grande distribuição geográfica, heterogeneidade, interoperabilidade, interações em tempo real e escalabilidade [11]. Apesar de seu potencial, a *fog computing* ainda é uma tecnologia em desenvolvimento. Logo, existem diversos desafios a serem superados, tais como a falta de padronização e a segurança dos dados.

Nesse contexto, a *fog computing* é uma arquitetura que permite a execução de tarefas computacionais próximas aos dispositivos de borda. Isso é possível graças aos *fog nodes*, que são dispositivos de *hardware* que possuem capacidade computacional, de comunicação e de virtualização [5]. O gerenciamento dos *fog nodes* é um desafio importante na *fog computing*. Isso se deve ao fato de que a *fog computing* utiliza contextos dinâmicos, que podem levar a eventos imprevisíveis, como indisponibilidade de serviço, alto tempo de resposta e confiabilidade mitigada [13].

Assim, o processo de gerenciamento de recursos é amplo e inclui etapas como estimativa, escalonamento, orquestração, monitoração e outras. A etapa de monitoração é responsável por melhorar a utilização dos recursos disponíveis em um ambiente de *fog computing* [14]. Contudo, para um monitoramento adequado há diversas variáveis que influenciam diretamente em uma monitoração mais ou menos eficiente. Assim sendo, este trabalho tem como objetivo geral implantar um ambiente real de *fog computing* para proporcionar um ambiente de testes a ser usado na análise de diferentes mecanismos de monitoração nesse tipo de ambiente.

1.1 Motivação

Construir um ambiente real de *fog computing* a fim de disponibilizar um ambiente com poder de processamento, e armazenamento e baixa latência. Isso é necessário pelo fato de que os grandes provedores ainda não disponibilizam um ambiente real.

1.2 Objetivos deste Trabalho

Desenvolver um ambiente real de *fog computing* para analisar ferramentas de software para monitoramento. Para cumprir o objetivo geral, este trabalho tem os seguintes objetivos específicos:

- Implantar infraestrutura de hardware por meio de diferentes *fog nodes*, tal como *Raspberry*;
- Selecionar um conjunto de software que sejam capazes de auxiliar no monitoramento de um ambiente de *fog computing*.

1.3 Sumário do Trabalho de Conclusão de Curso

O presente trabalho está organizado, além deste capítulo introdutório, em mais cinco capítulos. O Capítulo 2 define *fog computing* e *fog node*. O Capítulo 3 contextualiza o gerenciamento de recursos em *fog computing*. O Capítulo 4 apresenta a infraestrutura de hardware utilizada nesta pesquisa. O Capítulo 5 aborda o ambiente de monitoração escolhido e os resultados. O Capítulo 6 encerra o debate sobre o tema com uma síntese dos pontos principais e indicação dos trabalhos futuros.

Capítulo 2

Fog Computing e Outros Modelos Computacionais

No presente capítulo são apresentadas as principais características da Computação em Nuvem (*Cloud Computing*) e Computação em Névoa (*Fog Computing*),bem como alguns outros paradigmas computacionais relacionados. Além disso, são abordadas as características de um recurso computacional para a *Fog Computing* e o gerenciamento desses recursos, os quais serão utilizados no desenvolvimento deste trabalho

Deste modo, este capítulo começa com uma introdução às definições de Computação em Nuvem e Computação em Névoa (Seções 2.1 e 2.2 respectivamente). Em seguida, a Seção 2.3 apresenta detalhes sobre a perspectiva computacional de um *Fog Node*. A Seção 2.4 apresenta a definição dos demais modelos computacionais, tal como *Edge Computing*. Por fim, a Seção 2.5 faz um comparativo entre todos os modelos abordados neste capítulo.

2.1 Computação em Nuvem

No início da década de 60, John McCarthy [15] introduziu um conceito inicial sobre Computação em Nuvem quando ele disse que, no futuro, a computação poderia ser organizada como um serviço público, como era o sistema telefônico no ano de 1961.Com o desenvolvimento vertiginoso da computação e da comunicação, o paradigma da Computação em Nuvem se expandiu e, atualmente, esta plataforma é extremamente usada pela indústria, academia e usuários em geral.

A Computação em Nuvem é uma infraestrutura computacional que impacta todas as outras tecnologias e paradigmas apresentados neste trabalho. Ela surgiu em um contexto de transição, no qual recursos computacionais, antes caros e escassos, tornaram-se mais acessíveis e abundantes [16]. A Computação em Nuvem possibilitou a democratização da computação, oferecendo aos usuários a ilusão de recursos computacionais infinitos e

com rápida escalabilidade.Assim, segundo NIST (National Institute of Standards and Technology) [17] Computação em Nuvem é um modelo de serviço que permite o acesso sob demanda a um conjunto compartilhado de recursos computacionais configuráveis, como redes, servidores, armazenamento, aplicações e serviços. Esses recursos podem ser rapidamente provisionados e liberados com um esforço de gerenciamento mínimo ou interação com o provedor de serviços.

Diante do exposto, a Computação em Nuvem torna a computação uma utilidade básica, tais como a água ou a eletricidade. Há na literatura, cinco características essenciais para ambientes de Computação em Nuvem[7]:

- Autoatendimento sob demanda: os usuários podem provisionar recursos de computação sem a necessidade de intervenção humana;
- Acesso amplo à rede: os recursos de computação estão disponíveis por meio da Internet, de qualquer lugar do mundo;
- Conjunto de serviços computacionais: os recursos de computação são compartilhados entre vários usuários, o que reduz os custos;
- Elasticidade: os recursos computacionais podem ser aumentados ou reduzidos de forma dinâmica, conforme as necessidades do usuário;
- Medição dos serviços em baixa granularidade: os usuários são cobrados apenas pelos recursos que consomem.

Todavia, a Computação em Nuvem apresenta algumas limitações, sendo a principal delas a conectividade entre a nuvem e os dispositivos finais. Isso ocorre porque os provedores de nuvem pública têm grandes *datacenters* localizados em todo o mundo, mas nem sempre próximos dos usuários finais. Como resultado, a Qualidade de Serviço (*Quality of Service* - QoS) pode ser degradada, tornando a infraestrutura inadequada para solicitações de serviço em tempo real [18]. Dessa forma, como no ambiente de nuvem o processamento e o armazenamento são realizados em servidores remotos, a largura de banda da rede ainda é um obstáculo para a Computação em Nuvem [19]. Exemplos de casos de uso afetados por essa limitação incluem veículos conectados, cidades inteligentes, realidade virtual e saúde [20]. A superação dessa limitação beneficiaria todas as aplicações sensíveis a latência.

2.1.1 Sky Computing

A competição entre os provedores públicos de Computação em Nuvem aumentou ao longo do tempo. Como resultado, há uma grande variedade de serviços disponíveis, com muitos

provedores oferecendo produtos semelhantes, mas usando abordagens diferentes. Como consequência, as organizações estão, cada vez mais, adotando implantações *multicloud*, nas quais os recursos de diferentes provedores são combinados para formar uma infraestrutura de suporte tecnológico. Essa abordagem, conhecida como *Sky computing* [21], pode ser vista como um nível superior à Computação em Nuvem tradicional, pois os recursos são provisionados dinamicamente de acordo com as necessidades. O *Sky computing*, portanto, consiste em uma camada de gerenciamento de ambientes em nuvem que fornece capacidade de armazenamento flexível e suporte dinâmico para demandas em tempo real [22].

O *Sky computing* é um modelo de computação que permite que os usuários acessem recursos a partir de várias nuvens. Existem diferentes maneiras de implementar o *Sky Computing*, que podem ser classificadas de acordo com a forma como lidam com o escalonamento de recursos, a integração entre nuvens e o conhecimento da existência da camada de *Sky Computing* pelos recursos. Há outras denominações encontradas na literatura para esse paradigma, tais como: *multicloud* [21], *cross-cloud* [23], nuvens federadas [24] [21] e *inter-clouds* [25]. A diferença de nomenclatura pode ser resultado da forma como as arquiteturas distribuem o escalonador de recursos. Esse componente pode ser interno ou externo ao provedor. As nuvens participantes da camada de *Sky Computing* podem ser integradas de diferentes maneiras, e os recursos em seus respectivos provedores podem ter (ou não) conhecimento da existência dessa camada.

Em um ambiente de *Sky Computing*, a consolidação de recursos é fundamental para oferecer uma visão unificada de todos os recursos de nuvem disponíveis. As ferramentas conhecidas como orquestradoras de nuvem (do inglês, *Cloud Orchestrator*) são utilizadas para integrar, descobrir e consolidar recursos na nuvem de forma contínua [26]. Alguns exemplos de orquestradores atualmente usados são: *Terraform* [27], *Cloudifym* [28] e *Heat* [29]

Nesse cenário, a Computação em Nuvem e a *Sky Computing* são modelos de entrega de poder computacional que são especialmente adequados para o processamento de grandes volumes de dados. Todavia, apesar de seus benefícios, a conectividade entre a nuvem e os dispositivos finais é a principal limitação dos conceitos de nuvem e dispositivos finais. A distância entre os grandes *datacenters* de Computação em Nuvem e os usuários finais é um desafio para a Qualidade de Serviço (QoS). Essa distância pode resultar em atrasos e latências que prejudicam as aplicações que exigem respostas rápidas [18] [20], tais como jogos *online*, aplicações de realidade virtual, cidades inteligentes, IoT e realidade virtual. Para superar essas limitações alguns paradigmas e tecnologias foram propostos nos últimos anos, e serão apresentados nas próximas seções.

2.2 Fog Computing

O termo Computação em Névoa ou (*Fog Computing*) foi inspirado na névoa da meteorologia, que é uma nuvem baixa e próxima ao solo. Da mesma forma, a Computação em Névoa é um paradigma que move parte do processamento e do armazenamento de dados para dispositivos próximos aos usuários e dispositivos finais [30]. Isso reduz a latência e melhora a performance de aplicações que exigem respostas rápidas. Assim, a ideia é que parte das tarefas que até então eram executadas exclusivamente na Computação em Nuvem, sejam implementadas na Computação em Névoa.

A Computação em Névoa foi originalmente definida por Bonomi et al.[14] como uma plataforma virtualizada que fornece serviços de computação, armazenamento e rede entre dispositivos finais e *datacenters* de nuvem tradicionais. Essa plataforma está tipicamente localizada na borda da rede. A definição original de Computação em Névoa foi expandida e refinada por diversos pesquisadores. Para Vaquero et al. [31] e Yi et al. [32], a Computação em Névoa é uma arquitetura na qual dispositivos descentralizados e heterogêneos se comunicam e cooperam entre si para realizar tarefas, como armazenamento e processamento, sem a necessidade de intervenção de terceiros. Essas tarefas podem incluir funções básicas de rede, serviços e aplicativos executados em um ambiente de área restrita.

Assim sendo, de acordo com Dastjerdi et al. [13], a Computação em Névoa é uma arquitetura de computação distribuída que coloca recursos de computação e armazenamento na borda da rede, perto dos dispositivos que geram dados. Essa arquitetura é ideal para aplicações que precisam de baixa latência e alta disponibilidade, como Internet das Coisas (IoT), veículos autônomos e cirurgias remotas.

Para Naha et al. [20], a Computação em Névoa é uma plataforma que distribui o processamento entre os dispositivos de ponta e a nuvem. Os dispositivos de ponta, que podem ser virtualizados ou não, são responsáveis por um processamento mais simples, enquanto que a nuvem é responsável pelo armazenamento de dados por longos períodos de tempo, e por processamento mais robusto. As empresas de tecnologia Cisco [33] e IBM [34] concordam que a Computação em Névoa é uma extensão da nuvem. No entanto, a Cisco enfatiza que ela fica mais perto dos dispositivos IoT, enquanto que a IBM destaca que ela opera nas extremidades da rede.

De acordo com o NIST [17], a Computação em Névoa é um modelo de computação descentralizada que fornece recursos de computação próximos aos dispositivos finais. Isso permite a implantação de aplicativos e serviços distribuídos e sensíveis à latência. A Computação em Névoa minimiza o tempo de resposta de solicitação, fornecendo recursos de computação locais aos dispositivos finais e, quando necessário, conectividade de rede para serviços centralizados. Assim sendo, a Computação em Névoa é uma infraestrutura de computação descentralizada que distribui dados, processamento e aplicativos entre a

fonte de dados e a nuvem. O Open Fog Consortium [3] afirma que essa abordagem é uma complementação, e também uma extensão dos modelos tradicionais baseados em nuvem, pois permite que as cargas de trabalho sejam executadas no local mais adequado, de acordo com os requisitos de latência, segurança e custo.

Em todas as definições apresentadas é possível perceber que a Computação em Névoa é um complemento à computação em nuvem, não um substituto. A Computação em Névoa é mais adequada para aplicações que exigem baixa latência ou processamento local, enquanto que a Computação em Nuvem é mais adequada para aplicações que exigem armazenamento ou processamento de dados em grande escala com um grau de complexidade superior [35].

2.2.1 Principais Características

Diante das diversas definições de Computação em Névoa apresentadas na seção anterior, é importante ressaltar as características essenciais dessa plataforma. Assim, segundo NIST [17] essas características são:

- Resposta em tempo real: a Computação em Névoa permite que os dados sejam processados e respondidos em tempo real, pois os dispositivos de névoa, geralmente, estão localizados na borda da rede. Isso é uma vantagem significativa para aplicações que requerem respostas rápidas, como jogos, realidade aumentada e realidade virtual;
- Heterogeneidade: capacidade de coletar e processar dados de diferentes origens, adquiridas por meio de diferentes tipos de recursos de comunicação em rede;
- Dispersão geográfica: a Computação em Névoa é uma arquitetura de computação distribuída que coloca recursos computacionais próximos aos usuários. Isso é diferente da Computação em Nuvem, que coloca recursos em um local centralizado. A distribuição geográfica da Computação em Névoa é necessária para melhorar o desempenho e reduzir a latência para os usuários;
- Interoperabilidade e Federação: capacidade de dois ou mais sistemas de operarem de forma conjunta, compartilhando dados e recursos, e a integração de sistemas de diferentes domínios, permitindo que eles compartilhem informações e serviços;
- Interações em tempo real: as aplicações de Computação de Névoa são caracterizadas por interações em tempo real, em vez de processamento em lote. Isso significa que as respostas aos eventos devem ser fornecidas no momento em que ocorrem, ou com uma latência mínima;

- Escalabilidade: capacidade de um sistema de aumentar ou diminuir seu desempenho de forma eficiente e previsível, de acordo com as demandas de carga suportando mudanças nos recursos computacionais, nas cargas de trabalho, e nas condições da rede e do dispositivo.

Tendo em mente essas características, a Computação em Névoa é uma arquitetura de computação descentralizada que apresenta latência e tempo de resposta reduzidos, tornando-a uma solução adequada para aplicações que exigem respostas rápidas no momento em que a Computação em Nuvem não atender aos requisitos de latência ou tempo de execução exigidos por alguns aplicativos.

2.2.2 Arquitetura

A arquitetura de três camadas é a abordagem mais comum para modelar arquiteturas de Computação em Névoa [20], conforme apresentado na Figura 2.1. A camada de *IoT* é responsável pela coleta de dados dos dispositivos de borda. A camada de *Fog* é responsável pelo processamento e armazenamento dos dados coletados. A camada de *Cloud* é responsável pelo armazenamento e o processamento de dados de grande volume.

A camada *IoT* é onde os dispositivos conectados à Internet das Coisas (*IoT*) estão localizados. Os usuários finais fazem solicitações de serviços nesta camada, que são então processadas pelas camadas *Fog* e *Cloud*. A camada *Fog* é um intermediário entre as camadas IoT e *Cloud*. Ela fornece recursos computacionais para aplicativos de processamento de dados, antes de enviar os dados para a *Cloud* [36]. A camada *Fog* é composta por nós, também chamados de *fog nodes*, que são dispositivos inteligentes capazes de processar e armazenar dados. A camada *Cloud* possui recursos computacionais mais robustos que a camada *Fog*. Ela é usada para processar as solicitações que não podem ser totalmente atendidas pela camada *Fog*.

Para Bachiega et al. [5], a visualização horizontal desta arquitetura, conforme mostrado na Figura 2.1, é mais adequada porque tanto a camada IoT quanto a camada Cloud, localizadas na base e no topo da Figura 2.2, apresentam um número infinito de dispositivos e recursos [1]. Já a camada interna de *Fog*, localizada entre as duas extremidades, possui um número limitado de dispositivos e recursos computacionais.

Figura 2.1: Visão horizontal da arquitetura.

2.3 Fog Node

Na computação um recurso é qualquer coisa que pode ser usada para executar um processo ou serviço, seja ela física ou virtual. Em Computação em Névoa, um *fog node* é um dispositivo que fornece recursos computacionais, de armazenamento e de rede para aplicações na borda da rede. No entanto, não existe uma definição única para o termo "*fog node*"[17], pois ele pode variar de acordo com o caso de uso e a área de aplicação. Por exemplo, um *fog node* pode ser um celular, um roteador, um sensor ou mesmo um dispositivo *IoT*.

Assim, como os *fog nodes* têm requisitos semelhantes aos dispositivos *IoT*, Bachiega et al. [5] fez uso das definições e classificações de dispositivos *IoT* como base para definir *fog nodes*. Dessa forma a *International Telecommunication Union* (ITU) e a *Internet Engineering Task Force* (IETF) propuseram modelos de referência para dispositivos *IoT*. O modelo da ITU define um dispositivo *IoT* como um equipamento com capacidades obrigatórias de comunicação e capacidades opcionais de detecção, atuação, captura de dados, armazenamento de dados e processamento de dados [2].

A recomendação da ITU [2] afirma que, como a conectividade depende do processamento, dispositivos com alto processamento e baixa conectividade (área APBC na Figura 2.3) são incomuns. Portanto, esses dispositivos não foram considerados na definição proposta por Bachiega et al. [5]. A seguir, uma breve descrição das classificações restantes:

- Dispositivos de Alto Processamento e Alta Conectividade (APAC): são dispositivos

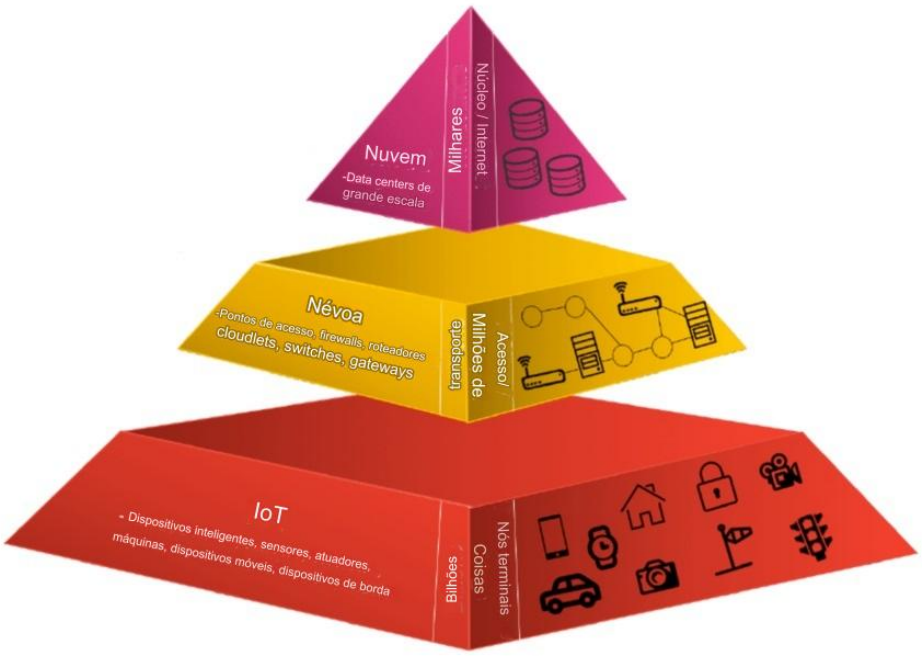

Figura 2.2: Visão piramidal de uma arquitetura de Computação em Névoa (Fonte [1])

que podem executar aplicativos e algoritmos complexos, conectam-se diretamente à Internet e coordenar outros dispositivos. Alguns exemplos incluem os Nós coletores (como o Raspberry Pi), que são usados para coletar dados de sensores e dispositivos *IoT*; dispositivos computacionais de baixo custo (como *smartphones*), que são usados para executar aplicativos e serviços simples e dispositivos computacionais de ponta (como computadores pessoais), que são usados para executar aplicativos e serviços complexos [37];

- Dispositivos de baixo processamento e alta conectividade (BPAC): são dispositivos com pouco poder de processamento, mas podem se comunicar diretamente com aplicativos ou serviços em nuvem pela Internet. Roteadores, *Firewalls*, *Gateways*, set-top boxes e Pontos de Acesso sem fio (*Access Points*) são exemplos desse tipo de dispositivo;

- Dispositivos de baixo processamento e baixa conectividade (BPBC): esses dispositivos têm capacidade de processamento limitada e não se conectam diretamente a uma rede de comunicação. Eles precisam de outro elemento de rede para acessar a Internet ou outra rede. Exemplos incluem nós sensores, como Waspmote e Meshlium [37].

Além de processar e se comunicar, um *fog node* precisa atender a uma série de requisitos funcionais. Existem várias definições desses requisitos, mas a arquitetura de referência

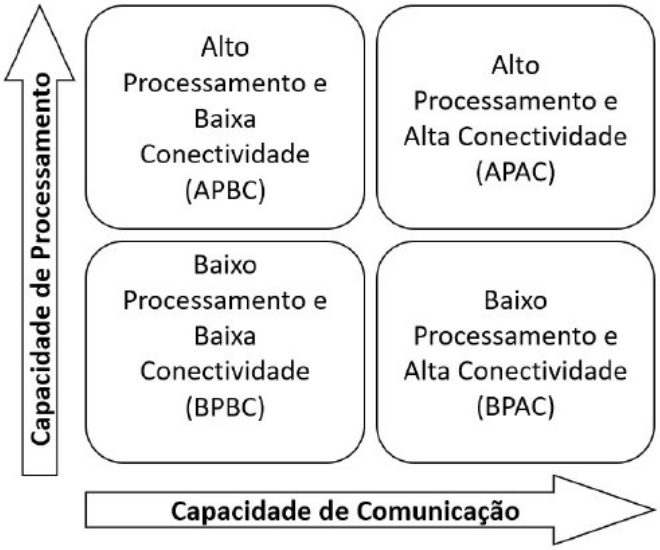

Figura 2.3: Categorias de dispositivos baseadas na relação entre processamento e comunicação[2].

proposta pelo OpenFog Consortium [3] identifica oito aspectos essenciais, como mostrado na Figura 2.4:

Figura 2.4: Estrutura do *fog node* proposta pelo OpenFog Consortium [3].

- **Segurança:** os *fog nodes* são componentes importantes da segurança de um sistema. Eles devem ser protegidos contra ataques, pois podem atuar como um *gateway* para sensores e atuadores legados. A segurança dos *fog nodes* deve ser considerada em todos os níveis, do hardware ao software;
- **Gerenciamento:** *fog nodes* devem ser compatíveis com as interfaces de gerenciamento fornecidas por outros *fog nodes*. Essas interfaces permitem que agentes de sistemas de nível superior visualizem e controlem *fog nodes* de nível inferior. O mesmo protocolo de gerenciamento pode ser usado em diferentes interfaces físicas;

- **Rede:** os *fog nodes* devem ser capazes de atender às demandas de Qualidade de Serviço (QoS) de aplicações críticas, como escalabilidade, disponibilidade e flexibilidade. Eles também devem ser capazes de priorizar dados críticos ou sensíveis à latência. Dependendo do cenário de implantação, os nós de névoa podem ser implementados como elementos de rede, como pontos de acesso, *gateways* ou roteadores;
- **Aceleradores:** alguns *fog nodes* usam módulos aceleradores para realizar análises de alta performance. Esses módulos podem ser *Graphics Processing Unit* (GPUs) ou *Field Programmable Gate Arrays* (FPGAs);
- **Computação:** um *fog node* é um dispositivo de computação que está localizado próximo aos usuários ou dispositivos finais. Ele é usado para processar dados e fornecer serviços em tempo real. Para aumentar a interoperabilidade, um *fog node* deve ter recursos de computação de propósito geral;
- **Armazenamento:***fog nodes* requerem uma variedade de soluções de armazenamento. Isso significa que as camadas de armazenamento usadas em *datacenters* também devem ser consideradas tais como: *RAM-based solid-state drive* (SSD) deve ser suportada em um *fog node*;
- **Sensores e atuadores:** são os dispositivos mais básicos da Internet das Coisas (IoT). Eles podem ser baseados em hardware ou software e são responsáveis por coletar dados do mundo real e controlar dispositivos físicos.Em um único *fog node*, podem ser conectados vários sensores e atuadores. Alguns desses dispositivos são simples, sem nenhuma capacidade de processamento significativa. Outros, no entanto, podem realizar algumas funções básicas de computação. Alguns desses elementos são: I2C, GPIO, SPI, BTLE, ZigBee, USB e Ethernet, etc;
- **Protocolo de abstração:** a camada de protocolo de abstração permite que os dados de sensores e atuadores sejam enviados para um *fog node*, mesmo que esses dispositivos não usem o mesmo protocolo de comunicação. Isso simplifica a integração de dispositivos heterogêneos em um sistema de Internet das Coisas (IoT).

Assim, levando em consideração as definições que foram apresentadas nesta seção, Bachiega et al. [5] definiu que um *fog node* é composto por duas camadas: uma camada de *hardware* e uma camada de sistema operacional. A camada de hardware é responsável pela infraestrutura física do nó (contém CPU, memória, interface de rede), enquanto a camada de sistema operacional abstrai o hardware e fornece um ambiente para executar aplicações [38], conforme ilustrado na Figura 2.4(a)

Contudo, um recurso desejável para o *fog node* é a capacidade de virtualização [39]. A virtualização pode ser definida como "uma abstração de software com a aparência de

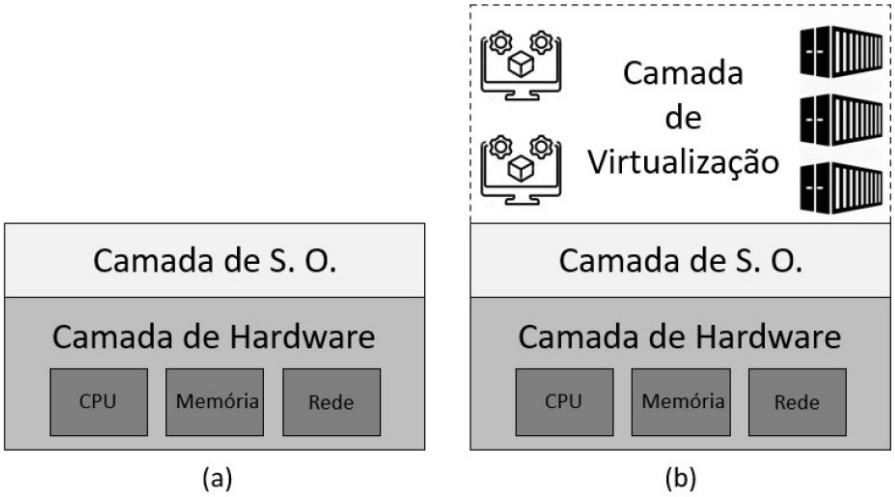

Figura 2.5: *Fog node* básico (a) e um *fog node* com virtualização (b). Adaptada de [4].

um hardware de sistema de computador" [40]. E além disso, a maioria dos processadores modernos, como os da Intel e da AMD, suportam virtualização de hardware. Isso significa que podem ser usados para criar plataformas de Computação em Névoa que executam vários sistemas operacionais e aplicativos em máquinas virtuais [3]. Neste raciocínio, para garantir o uso eficiente dos recursos de hardware, disponível no "*fog node* básico" (Figura 2.4(a)), é mais vantajoso ter uma Camada de Virtualização [41, 42], conforme ilustrado na Figura 2.4(b).

O uso de contêineres para entregar recursos virtuais está crescendo, à medida que as organizações buscam uma alternativa mais eficiente e flexível às máquinas virtuais [43]. Isto porque eles oferecem um mecanismo de isolamento mais aderente a um ambiente de Computação em Névoa [3] justamente por realizarem a virtualização na Camada de Sistema Operacional e não mais na Camada de Hardware, como era utilizado por outros modelos de virtualização [44]. Um diagrama que compara as duas maneiras de virtualização (máquinas virtuais e contêineres) é apresentado na Figura 2.5.

Por fim, considerando a abrangência e a diversidade de dispositivos que podem fazer parte de uma ambiente de Computação em Névoa (conforme apresentado nesta seção), e por ainda não haver uma padronização, a definição adotada neste trabalho foi proposta por Bachiega et al. [5], na qual um de *fog node* é qualquer dispositivo de hardware que possua capacidades computacionais, de comunicação e de virtualização que esteja em um ambiente de *fog computing*". A capacidade de virtualização é fundamental para que aplicações possam ser executadas na Camada de *Fog*, mesmo do ponto de vista computacional.

Além da definição de fog node, Bachiega et al. [5] propuseram uma abordagem mínima do ponto de vista computacional, apresentada na Figura 2.6. Neste diagrama de cebola

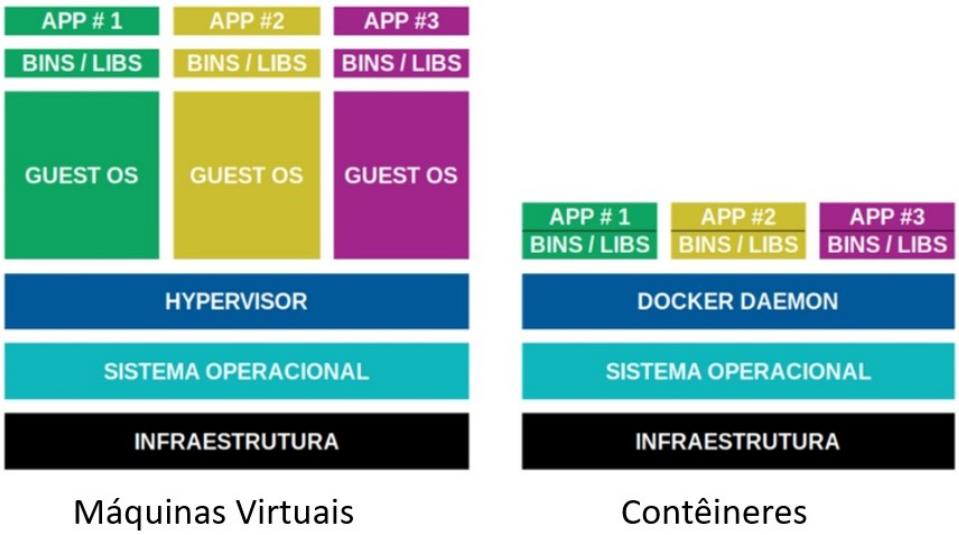

Figura 2.6: Representação da arquitetura de máquina virtual e contêiner

.

é possível perceber que o núcleo é composto por sensores e atuadores que fazem parte da Camada Dispositivos/Usuários Finais. A partir do momento em que as camadas da cebola se expandem, elas agregam capacidades de hardware, capacidades de software e capacidades de virtualização, tais como máquina virtual ou contêiner. Ao abranger mais camadas, o *fog node* torna-se mais favorável para aplicações e serviços de computação, possibilitando atender a diversos outros requisitos tais como segurança, gerenciamento e programabilidade.

Os *fog nodes* possuem atributos objetivos, que podem ser facilmente medidos, como capacidade de processamento, armazenamento e memória. Além disso, eles também possuem atributos subjetivos, que são mais difíceis de medir, como disponibilidade, mobilidade e confiabilidade [45]. Embora os atributos subjetivos não sejam impeditivos para a execução da aplicação em ambientes dinâmicos como o da Computação em Névoa, eles podem impactar no seu desempenho.

2.4 Considerações Finais

Neste capítulo foi apresentado as definições sobre Computação em Nuvem e Computação em Névoa que são duas tecnologias emergentes que têm o potencial de revolucionar toda estrutura de comunicação no mundo.

A computação em nuvem é um modelo de entrega de serviços de computação, armazenamento e rede pela internet. Os recursos computacionais são disponibilizados como

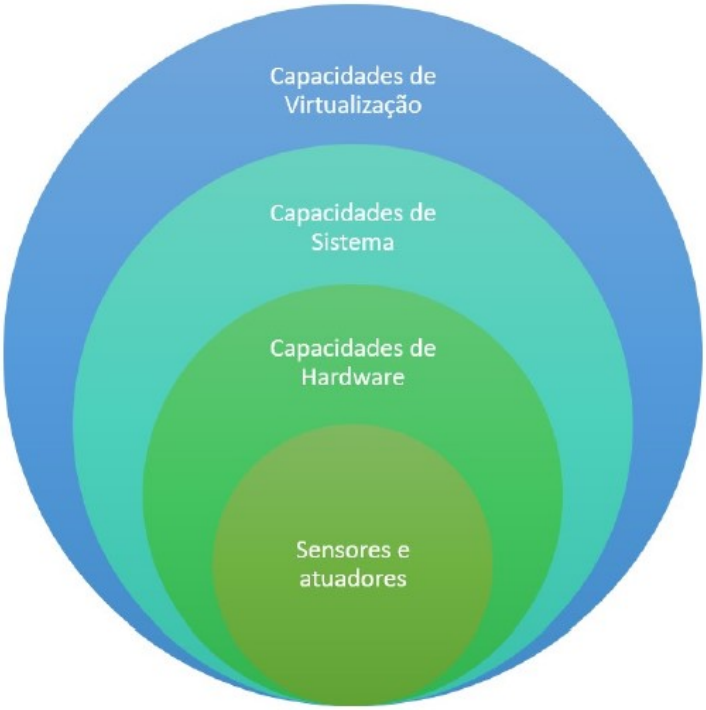

Figura 2.7: Classificação do *fog node* sob a perspectiva computacional [5]

.

um serviço, o que significa que as empresas e os indivíduos não precisam mais investir em infraestrutura de TI própria.

A computação em névoa é uma extensão da computação em nuvem que distribui os recursos computacionais para dispositivos de borda, como roteadores, sensores e dispositivos *IoT*. Isso permite que os dados sejam processados e armazenados mais perto de onde são gerados, o que pode melhorar o desempenho, a segurança e a confiabilidade dos sistemas.

Foi também apresentado as características e a arquitetura da Computação em Nuvem. Foi apresentado também o conceito de *fog node*.

O próximo capítulo irá analisar o gerenciamento de recursos na Computação em Névoa e também apresentar uma visão geral de trabalhos sobre descoberta de recursos.

Capítulo 3

Gerenciamento de Recursos

Este capítulo discute o gerenciamento de recursos em *fog computing*. Ele também apresenta uma visão geral de trabalhos sobre descoberta de recursos para dar sustentação aos meios de software e hardware que serão apresentados nos capítulos seguintes. O capítulo está dividido em três seções. A Seção 3.1 apresenta um breve resumo sobre gerenciamento de recursos e define sua acepção para este estudo. A Seção 3.2 descreve as aplicações utilizadas no monitoramento, que é uma das etapas no gerenciamento de recursos. Por fim, a Seção 3.3 define o escopo da pilha de software usada neste trabalho.

3.1 Gerenciamento de Recursos em Fog Computing

O gerenciamento de recursos é uma disciplina que busca o uso eficaz dos recursos disponíveis [46], o que é fundamental em diversas áreas, incluindo a Computação. Na Computação, os recursos computacionais são muitas vezes limitados, o que torna o gerenciamento de recursos ainda mais importante.

A *fog computing* é um tema que vem sendo pesquisado ativamente pela academia e pela indústria nos últimos anos [20]. Como resultado, existem várias publicações de revisão de literatura sobre o assunto. Outra questão importante a ser ressaltada é que a pesquisa sobre computação em *fog computing* aborda uma ampla gama de tópicos, desde conceitos gerais até questões específicas. Esta seção se concentra no tema de gerenciamento de recursos, que é um dos principais desafios da computação em *fog computing*.

Segundo Bachiega et al. [5], o processo de gestão de recursos pode ser dividido em cinco etapas, a saber: Descoberta, Estimativa, Alocação, Monitoração e Orquestração.

A Descoberta é a etapa inicial do ciclo de vida da carga de trabalho em *fog computing*. Ela visa identificar os recursos disponíveis no ambiente, como servidores, armazenamento e rede. A Estimativa é a etapa seguinte. Ela define o número e o tipo de recursos necessários para executar a carga de trabalho. Essa definição é feita com base nos requisitos da carga

de trabalho, tais como desempenho, disponibilidade e segurança [47]. A Alocação é a etapa em que os recursos são selecionados e reservados para a execução da carga de trabalho. Os recursos selecionados devem atender aos requisitos definidos na etapa de Estimativa e quando esta etapa estiver concluída, o processo relacionado ao Monitoramento é iniciado [48]. A Monitoração é a etapa em que a carga de trabalho é monitorada para garantir que ela esteja sendo executada de acordo com os requisitos. O Monitoramento considera aspectos como elasticidade, balanceamento de carga, tolerância a falhas e integridade [49]. A Orquestração é o processo que coordena as etapas anteriores. Ela é responsável por garantir que a carga de trabalho seja executada de forma eficiente e segura. As próximas seções descrevem detalhadamente todas as etapas do ciclo de gerenciamento de recursos em um ambiente de *fog computing*.

3.1.1 Descoberta de Recursos

Dois pesquisadores, Manvi e Shyam [50], definem a descoberta de recursos como o processo de encontrar e selecionar recursos computacionais para uma determinada tarefa. Singh e Chana [51] definem a descoberta de recursos como o processo de encontrar e listar recursos disponíveis. As duas definições incluem as ações de localizar e divulgar informações sobre os recursos, que são essenciais para usar todos os recursos disponíveis em um ambiente distribuído [52].

A descoberta de recursos é um desafio fundamental na computação em névoa [52]. Isso se deve às características inerentes à névoa, como a mobilidade, a alta distribuição geográfica e a heterogeneidade. O artigo [53] define o problema de descoberta de recursos na computação em névoa como a tarefa de encontrar recursos que atendam aos requisitos de um componente que deseja se juntar a uma rede de névoa. Uma solução para esse problema deve considerar as características da névoa, como a mobilidade e os modelos colaborativos.

Em suma, segundo Bachiega *et al.*[14], a descoberta é a tarefa do serviço de gerenciamento de recursos que visa encontrar os recursos disponíveis no ambiente de *fog computing*, mantendo o catálogo de recursos atualizado.

3.1.2 Estimativa de Recursos

A estimativa da demanda de recursos computacionais é uma tarefa muito importante no gerenciamento de recursos computacionais [11]. Segundo Manvi e Shyam [50], a estimativa de recursos é um processo de cálculo ou estimativa subjetiva dos recursos reais necessários para executar uma aplicação. De acordo com Mahmud *et al.*[54], a estimativa é um

processo que ajuda a garantir que os recursos computacionais sejam alocados de forma eficiente e eficaz, a fim de atender aos requisitos de qualidade de serviço.

O planejamento da capacidade para a *fog computing* é uma tarefa desafiadora que exige uma compreensão profunda dos requisitos de recursos computacionais [1], das opções de precificação [55] e das implicações do consumo de energia [56]. Além disso, a estimativa de recursos deve levar em consideração o tipo de dispositivo, a mobilidade, a energia disponível, os tipos de dados, o método de comunicação, as medidas de segurança e o comportamento do usuário [57].

Assim, devido às limitações computacionais dos *fog nodes*, o processo de estimativa é fundamental para a alocação e uso otimizados dos recursos na *fog computing*.

3.1.3 Monitoração de Recursos

Esta seção descreverá a etapa de monitoração porque este trabalho objetiva apresentar o funcionamento de um ambiente real. Logo, a monitoração será uma etapa do ciclo de gerenciamento de recursos em *fog computing* explorada neste trabalho.

A monitoração de recursos em *fog computing* torna possível distribuir componentes de sistemas computacionais por uma ampla área geográfica. Isso pode trazer benefícios, tais como redução da latência e aumento da confiabilidade. No entanto, também apresenta desafios, como por exemplo a dificuldade de gerenciar e monitorar esses componentes de forma centralizada [58]. As três funções principais que compõem um serviço de monitoração são: Observação dos Recursos e Serviços Monitorados, Processamento de Dados, e Exposição de Dados [59]. A Observação dos Recursos é o processo de obter informações sobre o estado atual do uso de recursos (por exemplo, carga e latência) ou desempenho do serviço (por exemplo, tempo de resposta). O Processamento de Dados é o conjunto de operações que transformam os dados brutos em informações úteis. Essas operações podem incluir filtragem, agregação, criação de eventos e notificações. E por último, a Exposição de Dados está diretamente relacionada ao sítio onde os dados gerados são armazenados (por exemplo, em um banco de dados local), e como são acessados por um sistema de gerenciamento.

Paralelo a isso, a Monitoração abrange três domínios de instrumentação [60]: *traces*, *logs* e métricas. Os domínios são caracterizados por diferentes aspectos, como a natureza dos dados coletados, os objetivos da coleta e os riscos envolvidos. Esses aspectos influenciam os processos de tomada de decisão, que podem ocorrer em diferentes momentos da coleta de dados. Um *trace* é um registro de todas as etapas de execução de uma requisição de um usuário. Ele mostra como a requisição foi processada por um serviço, desde o início até o fim. *Logs* são registros de eventos que podem ser formatados ou não formatados. Eles fornecem informações detalhadas e contexto adicional. E, por último, uma métrica é

uma representação de um valor numérico ou qualitativo em um momento específico. Ela é composta por um nome, um valor, uma data e hora, e informações adicionais opcionais [60].

A monitoração pode ser classificada como caixa preta ou caixa branca, de acordo com a terminologia do teste de software [61]. A monitoração de caixa preta é uma abordagem que avalia o estado de um sistema ou serviço a partir de sua interface pública. O seu objetivo é determinar se o sistema ou serviço está disponível e funcionando corretamente. A caixa branca é uma técnica de teste de software que analisa o funcionamento interno do objeto. Ela permite identificar a causa raiz de problemas de disponibilidade ou funcionalidade, fornecendo informações detalhadas sobre os processos internos [62]. Os *logs* e os *traces* estão mais relacionados à monitoração de caixa branca, ao mesmo tempo que as métricas estão mais próximas à monitoração de caixa preta.

Além disso, há o conceito de observabilidade que é uma abordagem holística para a monitoração de aplicativos baseados em microsserviços. Ela combina *logs*, *traces* e métricas para fornecer uma visão completa do comportamento de uma aplicação. A observabilidade é um conceito mais amplo do que o monitoramento, pois inclui não apenas a coleta e o armazenamento de dados, mas também a análise desses dados para entender o comportamento do sistema. Essa análise pode ser usada para identificar problemas antes que eles ocorram, bem como para entender a causa raiz de problemas existentes [63].

3.1.4 Orquestração

Orquestração, de acordo com o artigo [64], refere-se a "um gráfico descrevendo relacionamentos entre elementos de software ou processos". Por outro lado, tem-se que a orquestração envolve a seleção de nós, que são unidades de processamento, para executar um serviço ou aplicativo. Os nós selecionados se comunicam entre si por meio de um protocolo seguro que garante a privacidade das informações trocadas [65].

Há uma diversidade de conceitos relacionado à orquestração. Como não existe uma definição completa no contexto da *fog computing*, e por não haver uma padronização adotada pela academia, o artigo [63] propõe a seguinte definição: *"a orquestração na fog computing é uma função de gerenciamento responsável pelo ciclo de vida do serviço. Para fornecer os serviços solicitados ao usuário e garantir os SLAs, deve-se monitorar a infraestrutura subjacente, reagir em tempo hábil às suas mudanças e cumprir as regras de privacidade e segurança".*

E por último, para exemplificar, Bonomi *et al.* [66] formularam uma arquitetura de software para execução de serviços de *fog computing*, cuja representação é mostrada na Figura 3.1. Os autores propõem uma camada de orquestração de *fog computing* que utiliza o *loop* de controle Monitorar-Analisar-Planejar-Executar (do inglês, *Monitor-Analise-Plan-*

Execute - MAPE) para gerenciar o ciclo de vida de serviços de *fog* de forma distribuída. Na fase Monitorar, a orquestração coleta dados de monitoramento de todos os recursos gerenciados e serviços em execução. Esses dados são analisados para construir uma visão atualizada e abrangente do ambiente de *fog computing*. Com essa visão, é possível planejar as mudanças necessárias para manter os serviços dentro dos limites de SLAs (*service-level agreement - Garantia de Nível de Serviço*) e QoS (Quality of service - Qualidade do Serviço), além de fornecer novos serviços solicitados. As mudanças planejadas são então executadas, liberando e alocando recursos adequados para fornecer serviços próximos aos usuários finais.

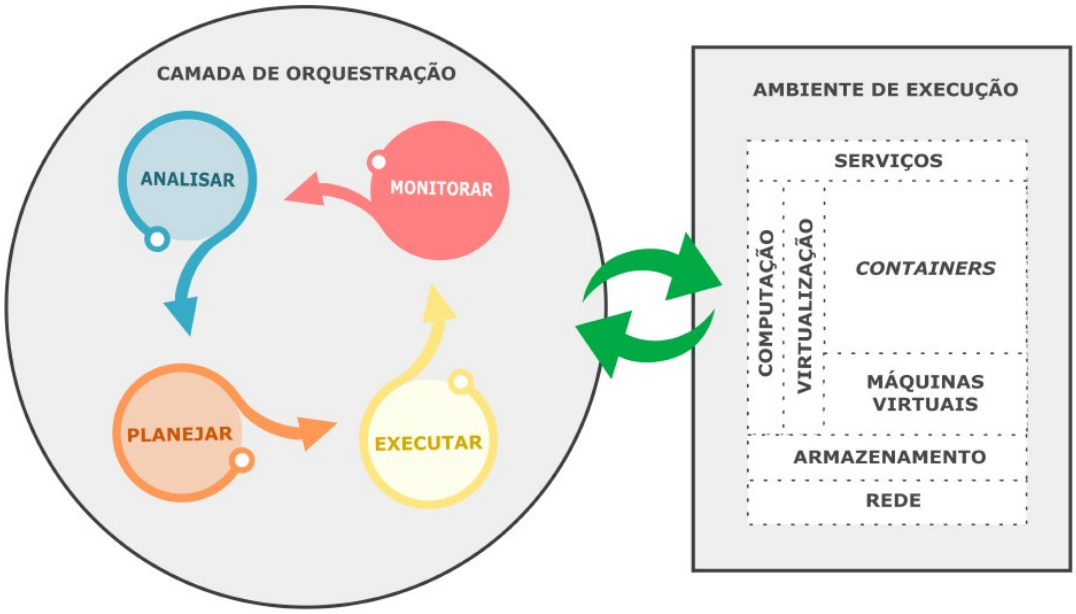

Figura 3.1: Camada de orquestração na *fog computing* [5].

3.1.5 Alocação de Recursos

A alocação de recursos é um componente fundamental da gestão de recursos, pois é responsável por distribuir os recursos disponíveis de forma a atender aos objetivos do projeto. Esta etapa é complexa e desafiadora, pois envolve uma série de fatores a serem considerados, tais como a disponibilidade de recursos, as necessidades do projeto e os custos envolvidos.

Os autores pesquisados concordam que a alocação de recursos é uma etapa crítica do gerenciamento de projetos. Nath *et al.* [67] e Luo *et al.* [68] afirmam que a alocação de recursos deve ser feita de forma a garantir que os recursos sejam usados de maneira

eficiente e eficaz. Mahmud *et al.* [54] e Mijuskovic *et al.* [69] concordam que a alocação de recursos deve ser feita de forma a otimizar o uso dos recursos disponíveis.

Assim, a Alocação de Recursos é a etapa que garante que a tarefa seja executada no ambiente de *fog computing* de acordo com os requisitos de *QoS*. Ela faz isso reservando os recursos computacionais necessários, de acordo com as informações de estimativa da etapa anterior. A técnica de alocação, o método de virtualização e as camadas de arquitetura a serem cobertas são definidos pela saída da alocação de recursos.

3.2 Ferramentas Utilizadas

Este trabalho visa apresentar, de forma clara e objetiva, a implantação e a monitoração de um ambiente real de *fog computing*. Para isso, foram utilizados alguns softwares, que serão descritos e comentados nesta seção.

3.2.1 *Ubuntu Server*

O *Ubuntu Server* é uma distribuição Linux de código aberto desenvolvida pela Canonical. É uma versão do Ubuntu projetada para uso em servidores, com foco na estabilidade, segurança e eficiência. Este Sistema Operacional (SO) é baseado no *kernel* (núcleo do sistema operacional) Linux e usa o gerenciador de pacotes APT (*Advanced Package Tool* - Ferramenta Avançada de Pacotes) para instalar e gerenciar *softwares*. Ele vem com uma ampla gama de software que pode ser pré-instalado, incluindo o servidor *web* Apache, o servidor de *e-mail Postfix*, e o servidor de banco de dados *MySQL*. O Ubuntu Server é uma escolha popular para uma variedade de aplicações de servidor, incluindo: hospedagem web, aplicação web, banco de dados, armazenamento, virtualização, e também, conteinerização.

O sistema operacional em destaque tem as características de oferecer uma série de recursos que o tornam uma escolha atraente para a implementação deste trabalho tais como [70]:

- Estabilidade: projetado para ser estável e confiável. Ele é testado extensivamente antes do lançamento e recebe atualizações regulares de segurança e correções;
- Segurança: idealizado para ser seguro. Ele vem com uma variedade de recursos de segurança integrados, como autenticação de dois fatores e criptografia. Há ainda atualizações regulares com novas versões e *patches* de segurança. As versões principais são lançadas a cada seis meses, e as versões de manutenção (*"Long-Term Support" (LTS)* - indicando que o sistema terá 5 anos de suporte oficial) são lançadas a cada dois anos. Melhorando, cada vez mais, a segurança do SO. Atualmente,

a versão *LTS* mais recente é o *Ubuntu 22.04*. O software foi liberado em 21 de abril de 2022 e será atualizado até 2027;

- Eficiência: desenvolvido para ser eficiente. Ele utiliza recursos do sistema de forma eficiente, o que ajuda a reduzir o custo operacional.
- Instalação: é facilmente instalado a partir de um CD ou DVD, ou de um arquivo ISO (Identical Storage image of Optical media - Imagem de armazenamento idêntica de mídia óptica) que pode ser gravado em uma unidade USB. O processo de instalação é relativamente simples e pode ser concluído em poucos minutos;
- Documentação: outra questão é a excelente gama de informações de uso do sistema operacional. A documentação oficial está disponível no *link https://help.ubuntu.com/* e em formato impresso.
- Gratuito: O SO é distribuído de forma gratuita e não há taxa adicional pela "versão empresarial". Todas as versões são disponibilizadas sem custos, estando disponível para cópia na Internet. Inclui as melhores traduções e infraestrutura de acessibilidade que a comunidade software livre tem a oferecer, para que seja útil para o maior número possível de usuários.

Em suma, o *Ubuntu Server 22.04* foi escolhido para ser o sistema operacional da infraestrutura de *fog computing* implantada neste trabalho por causa de suas características, tais como segurança, desempenho e facilidade de uso.

3.2.2 *Docker* e *Containers*

Na *fog computing* dispositivos com recursos limitados são comuns. Nesses ambientes *containers* são uma opção mais adequada do que máquinas virtuais para executar aplicações. *Containers* são uma forma de virtualização leve e compacta [71], que compartilham o *kernel* do sistema operacional do *host*. Eles são mais eficientes em termos de recursos do que máquinas virtuais, que precisam de um sistema operacional completo para cada instância. O *Docker* [72] é um ambiente de execução de *containers* popular e amplamente utilizado sendo o ambiente de execução de *containers* escolhido neste trabalho.

Assim sendo, *containers* são unidades de software que empacotam código, bibliotecas, configurações e outros recursos necessários para executar um aplicativo [43]. Eles são executados de forma isolada, compartilhando o *kernel* do sistema operacional com outros *containers*. O *Docker* é uma plataforma de código aberto que facilita a criação, o gerenciamento e a execução de *containers*. Ele fornece uma Interface de Linha de Comando (CLI) e uma Interface de Programação de Aplicação (API) para criar e gerenciar *containers* [73].

Para criar um *container* é necessário primeiro criar uma imagem do *Docker*. Uma imagem do *Docker* é um arquivo que contém todas as informações necessárias para executar um *container*. Ela pode ser criada a partir de uma imagem existente ou a partir de um conjunto de instruções. Uma vez criada uma imagem, ela pode ser executada para criar um *container*. Um *container* é uma instância de uma imagem do *Docker*. Os *containers* oferecem uma série de vantagens sobre as máquinas virtuais *(Virtual Machines - VMs)*. Eles são mais leves e eficientes, pois compartilham o mesmo *(kernel)* do sistema operacional com outros *containers*. Isso os torna ideais para aplicações que requerem um alto desempenho ou que precisam ser executadas em ambientes com recursos limitados [43].

Além disso, os *containers* são mais portáteis do que as VMs. Eles podem ser executados em qualquer sistema operacional que suporte o *Docker*. Isso facilita o desenvolvimento e o gerenciamento de aplicações em ambientes heterogêneos.

3.2.3 *Portainer*

Portainer é uma ferramenta de gerenciamento de *containers Docker* de código aberto [74]. Ele é uma interface *web* baseada em navegador que fornece uma maneira fácil de gerenciar *containers Docker*, incluindo criação, inicialização, parada, reinicialização, remoção e monitoramento.

Portainer é uma ferramenta ideal para desenvolvedores, administradores de sistemas e usuários finais que desejam gerenciar *containers Docker* sem precisar se envolver com a CLI do *Docker*. Ela é fácil de usar e oferece uma interface intuitiva que permite que os usuários gerenciem seus *containers Docker* com facilidade. As principais características do Portainer são [74]:

- Oferece uma variedade de recursos que o tornam uma ferramenta poderosa para gerenciar *containers Docker*;
- Oferece uma interface *web* baseada em navegador que permite gerenciar *containers Docker* sem precisar se envolver com a CLI do *Docker*;
- Permite que os usuários criem *containers Docker* usando imagens *Docker* existentes ou criando suas próprias imagens;
- Permite que os usuários iniciem, parem, reiniciem e removam *containers Docker* com facilidade;
- Fornece informações sobre o estado dos *containers Docker*, incluindo CPU, memória, rede e armazenamento.

Por fim, é importante frisar que a instalação pode ser feita em uma variedade de sistemas operacionais. A instalação é simples e pode ser realizada usando um *script* ou um pacote. A configuração precisa ser inicialmente feita em um servidor *Docker*. Uma vez que um servidor *Docker* esteja configurado, os usuários podem configurar o cAdvisor (*Container Advisor*) [75] para se conectar ao servidor *Docker*. E o *Portainer* é fácil de usar, pois uma vez configurado, os usuários podem começar a gerenciar seus próprios *containers Docker*.

3.2.4 *cAdvisor*

cAdvisor (Container Advisor) [**?**] é um analisador de recursos e desempenho de *containers* desenvolvido pelo *Google*. Ele fornece aos usuários de *containers* uma compreensão do uso de recursos e das características de desempenho de seus *containers* em execução.

cAdvisor é um *daemon* que coleta, agrega, processa e exporta informações sobre *containers* em execução. Especificamente, para cada *containers*, ele mantém os seguintes dados: parâmetros de isolamento de recursos, uso de recursos histórico, histogramas de uso de recursos histórico completo, e estatísticas de rede. Esses dados são exportados por *container* e em escala de máquina [**?**].

cAdvisor tem suporte nativo para *containers Docker* e deve suportar praticamente qualquer outro tipo de *container* pronto para uso. *cAdvisor* pode ser usado para uma variedade de propósitos tais como: monitoramento, análise de desempenho e otimização de recursos de *containers*.

Assim sendo, o *cAdvisor* é um recurso valioso para qualquer usuário de *containers*. Ele fornece uma maneira fácil de coletar e analisar dados sobre o uso de recursos e o desempenho de seus *containers*.

3.2.5 *Prometheus*

O *Prometheus* [76] é um sistema de monitoramento e alerta de código aberto, desenvolvido pela *SoundCloud*. É um sistema baseado em *pull*, ou seja, o servidor *Prometheus* solicita as métricas dos clientes. Os clientes podem ser qualquer tipo de serviço, incluindo *Dockers* e *containers*.

Para monitorar *Dockers* e *containers* com o *Prometheus* é necessário configurar o servidor *Prometheus* para coletar as métricas desejadas. As métricas disponíveis variam de acordo com o tipo de *Docker* ou *container*. No caso de *Dockers*, as métricas disponíveis incluem [77]:

- *Docker daemon*: essas métricas fornecem informações sobre o estado do *daemon Docker*, como o número de *containers* em execução, o uso de CPU e memória, e o número de imagens baixadas;
- *Container*: essas métricas fornecem informações sobre o estado de cada *container*, tais como o uso de CPU e memória, o número de processos em execução, e o estado de saúde.
- Aplicativo: essas métricas fornecem informações sobre o aplicativo em execução no contêiner, tais como o número de solicitações recebidas, o tempo de resposta médio, e a porcentagem de erros.

Após configurar o servidor *Prometheus* para coletar as métricas desejadas, é possível visualizar as informações em gráficos ou *dashboards*. O *Prometheus* fornece uma variedade de ferramentas para visualizar as métricas, incluindo a interface *web Grafana* a qual será descrita na Seção 3.2.6.

Assim, como apresentado, o *Prometheus* é uma ferramenta poderosa para o monitoramento de Dockers e contêineres. É um sistema flexível e escalável, que pode ser usado para monitorar ambientes de qualquer tamanho. Por isso, o *Prometheus* foi a ferramenta usada na etapa de monitoração do ambiente real de *fog computinmg* implantado neste trabalho.

3.2.6 *Grafana*

Grafana [78] é uma plataforma de observabilidade *open source* que permite visualizar, analisar e alertar sobre dados de infraestrutura e aplicações. É uma ferramenta eficiente que pode ser usada por equipes de engenharia e operações para monitorar o desempenho de seus sistemas e identificar problemas potenciais.

Ela é baseada em um modelo de dados de painéis, que são compostos de fontes de dados, visualizações e regras de alerta. As fontes de dados podem ser de uma variedade de origens, incluindo o *Prometheus*. As visualizações podem ser personalizadas para atender às necessidades específicas de cada usuário. As regras de alerta permitem que os usuários sejam notificados quando os valores de um ou mais métricas ultrapassarem um limite definido.

Grafana oferece uma ampla gama de recursos que o tornam uma ferramenta versátil para observabilidade. Alguns dos principais recursos são [78]:

- Visualizações poderosas: oferece uma variedade de visualizações prontas para uso, incluindo gráficos, tabelas, mapas e *heatmaps*. Também é possível criar visualizações personalizadas usando o poderoso editor de visualizações do *Grafana*;

- Alertas flexíveis: permite que os usuários criem regras de alerta complexas que podem ser baseadas em uma variedade de condições, incluindo valores de métricas, eventos e *logs*;
- Integrações com outros sistemas: pode ser integrado com uma variedade de outros sistemas, incluindo sistemas de gerenciamento de configuração, sistemas de controle de versão e sistemas de *ticketing*.

Em resumo, o *Grafana* é uma ferramenta robusta e eficiente que pode ser usada para melhorar a observabilidade de sistemas e aplicações. É uma ferramenta gratuita e *open source* que pode ser facilmente implantada em qualquer infraestrutura.

3.2.7 *NodeExporter*

NodeExporter [79] é um *software* que coleta e exporta métricas de sistemas Linux. Ele é usado por ferramentas de monitoramento, como *Prometheus*, para obter informações sobre o desempenho e a saúde dos sistemas. *NodeExporter* coleta mais de 200 métricas, incluindo o uso de CPU, memória, disco e rede, temperatura, voltagem e carga da bateria, estado dos serviços e processos. Exporta as métricas em formato *Prometheus*. Além disso, ele é leve e exige pouco recurso de hardware.

O *NodeExporter* é uma ferramenta valiosa para qualquer administrador de sistemas Linux. Ele pode ser usado para melhorar a visibilidade e o controle de sistemas Linux, e pode ajudar a identificar problemas de desempenho ou saúde antes que eles causem problemas.

3.3 Pilha de Aplicações de Monitoramento

Nesta seção será apresentada a pilha de monitoramento implementada neste trabalho. A pilha fornece uma visão geral de todas as aplicações que foram citadas anteriormente. Uma pilha de monitoramento é um conjunto de ferramentas e processos que ajudam a coletar, analisar e visualizar dados de desempenho de uma infraestrutura de Tecnologia da Informação. O uso de *containers* do *Docker*, para criar uma pilha de monitoramento, pode simplificar o processo de implantação e gerenciamento, além de oferecer uma série de benefícios, tais como [72]:

- Escalabilidade: podem ser facilmente escalonados para atender às necessidades de uma infraestrutura crescente;
- Portabilidade: podem ser facilmente implantados em diferentes ambientes, como nuvens públicas ou privadas;

- Segurança: podem ser isolados uns dos outros, o que ajuda a proteger os dados e os sistemas.

Uma pilha de monitoramento é composta por diversos componentes que coletam, armazenam e analisam dados de desempenho e disponibilidade de aplicações. Os componentes mais comuns de uma pilha de monitoramento são [72, 78]:

- Coletores: são responsáveis por coletar dados de desempenho e disponibilidade de aplicações. Os coletores podem ser executados em agentes instalados nos *hosts* ou em *containers*. *Prometheus* e *cAdvisor* são exemplos de coletores;
- Agentes: são programas instalados nos *hosts* que coletam dados de desempenho e disponibilidade de aplicações. *NodeExporter* é um agente do *Prometheus* que coleta dados de desempenho e disponibilidade de *hosts*;
- Armazenamento: responsável por armazenar os dados coletados pelos coletores. *Prometheus* é um exemplo de componente de armazenamento;
- Analisadores: são responsáveis por analisar os dados coletados pelo armazenamento. *Grafana* é um exemplo de analisador;
- Visualizadores: são responsáveis por apresentar os dados analisados pelos analisadores. O *Grafana* pode ser citado como um exemplo de visualizador.

Assim sendo, a Figura 3.2, apresenta todas as ferramentas usadas no monitoramento do ambiente de fog computing implantado neste trabalho.

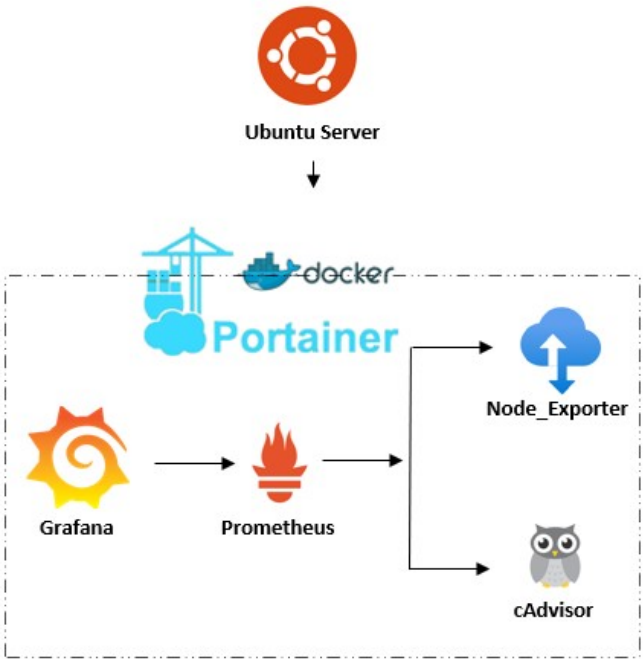

Figura 3.2: Estutura dos softwares utilizados neste trabalho.

Capítulo 4

Infraestura de Hardware

Este capítulo descreve a infraestrutura de hardware usada para implantar o ambiente real de *fog computing* usado neste trabalho.. A Seção 4.1 apresenta os conceitos fundamentais da estrutura de monitoração. Em seguida, a Seção 4.2 descreve toda a infraestrutura de hardware proposta nete trabalho para o monitoramento.

4.1 Considerações Iniciais

O monitoramento no ambiente de *fog computing* está diretamente relacionado ao processo de coleta, análise, e visualização de dados de dispositivos e aplicações em um ambiente *fog computing*. Como apresentado no Capítulo 2, *fog computing* é uma arquitetura de computação distribuída que coloca a computação e o armazenamento de dados mais próximos dos dispositivos IoT [17].

Neste contexto, a *fog computing* permite que dispositivos com poucos recursos de hardware realizem tarefas de forma mais eficiente. Por exemplo, um dispositivo IoT com poucos recursos de hardware pode usar a computação em névoa para processar dados de sensores e enviar alertas para o usuário. Isso evita que um dispositivo IoT tenha que realizar essas tarefas por conta própria, o que pode consumir muita energia e processamento. O objetivo do presente trabalho é utilizar dispositivos com as características supracitadas para implantar um ambiente de monitoramento em dispositivos na *fog* conhecidos por *fog nodes*.

Neste contexto, o *Raspberry Pi* surge para dar suporte a este projeto. O *Raspberry Pi* é um computador de placa única desenvolvido pela *Raspberry Pi Foundation*, uma organização sem fins lucrativos do Reino Unido. O objetivo do *Raspberry Pi* é promover a educação em ciência da computação e tecnologia em escolas e outros ambientes .

A ideia do *Raspberry Pi* surgiu em 2006, quando Eben Upton, então estudante de doutorado no Departamento de Eletrônica e Computação da Universidade de Cambridge,

estava preocupado com o declínio das habilidades de programação no Reino Unido. Ele acreditava que um computador barato e fácil de usar poderia ajudar a reverter essa tendência [6]. Upton começou a trabalhar no projeto *Raspberry Pi* em 2008, com a ajuda de um grupo de engenheiros e designers. O primeiro modelo do *Raspberry Pi*, o *Model B*, foi lançado em 2012 por um preço de US$ 25. O modelo foi um sucesso imediato, com mais de 1 milhão de unidades vendidas em seu primeiro ano de lançamento.

Desde então, a *Raspberry Pi Foundation* lançou uma série de novos modelos do *Raspberry Pi*, cada um com recursos e especificações mais avançados. O modelo mais recente, o *Raspberry Pi 5*, foi lançado em 2023 e é equipado com um processador Quad Arm Cortex-A76 de 2.4 GHz, 8 GB de RAM e suporte a Wi-Fi e Bluetooth.

Em suma, o *Raspberry Pi* é usado em uma ampla variedade de aplicações, incluindo educação, entretenimento, pesquisa e desenvolvimento. É uma ferramenta popular para ensinar programação, eletrônica e robótica. Também é usado para criar projetos de hardware de código aberto, como retroconsoles, sistemas de vigilância e dispositivos IoT.

4.2 Tipos de Raspberry Pi

Atualmente, o *Raspberry Pi* está em sua quinta geração. O primeiro modelo foi lançado em 2012 e o mais recente é outubro de 2023. Mesmo com novidades acrescentadas durante o tempo, como conexão Wi-Fi dual band, a *Raspberry Foundation* conseguiu manter o projeto na faixa dos US$ 35 (cerca de R$ 134) e chegou a lançar versões ainda mais baratas, como o Pi Zero, que custa a partir de US$ 5, ou seja, menos de R$ 20.

Atualmente, existem mais de 20 modelos de Raspberry Pi disponíveis, cada um com suas próprias características e especificações. Os principais tipos de Raspberry Pi são:

- *Raspberry Pi Zero*: modelo mais básico da série. É pequeno, barato e possui um processador ARM1176JZF-S de 1 GHz, 512 MB de memória RAM e suporte para Wi-Fi e *bluetooth*;
- *Raspberry Pi Zero W*: é uma versão do *Raspberry Pi Zero* com suporte para Wi-Fi e *bluetooth* integrados;
- *Raspberry Pi 3*: modelo mais robusto que o *Raspberry Pi Zero*. Possui um processador ARMv8 quad-core de 1,2 GHz, 1 GB de memória RAM e suporte para Wi-Fi, *Bluetooth* e *Ethernet*;
- Raspberry Pi 4: modelo utilizado neste projeto. Possui um processador ARMv8 *quad-core* de até 1,8 GHz, até 8 GB de memória RAM e suporte para wi-Fi, *bluetooth*, *Ethernet* e USB 3.0;

- O Raspberry Pi 5: modelo mais recente da série. Apresenta uma série de melhorias em relação à geração anterior tais como: CPU quad-core ARM Cortex-A76 de 2,4 GHz, GPU *VideoCore* VII de 800 MHz, Portas USB 3.0 e suporte para wi-Fi 6 e *bluetooth* 5.0

Além desses modelos, *a Fundação Raspberry Pi* também oferece uma variedade de modelos especializados, como o *Raspberry Pi Compute Module*, que é um modelo sem caixa projetado para ser integrado em outros produtos, e o *Raspberry Pi Pico*, que é um modelo menor e mais barato que o *Raspberry Pi Zero*.

A escolha do modelo de Raspberry Pi mais adequado depende das necessidades específicas do usuário. Para aplicações básicas, como educação e entretenimento, o Raspberry Pi Zero ou o Raspberry Pi Zero W são opções suficientes. Para aplicações mais exigentes, como automação residencial ou IoT, o *Raspberry Pi 3*, o *Raspberry Pi 4* ou o o *Raspberry Pi 5* são recomendados.

4.3 *Raspberry Pi 4*

O *Raspberry Pi*4 é um computador de placa única (*Single Board Computer - SBC*) que oferece a combinação de um processador, memória, entrada e saída e outros recursos em uma única placa integrada com baixo custo e baixo consumo de energia, tornando-o mais atrativo para projetos de hardware e software de código aberto. Ele possui um processador quad-core de 1,8 GHz, 4 GB de RAM, compartimento para MicroSD com, no mínimo, 16GB de armazenamento, recursos de áudio, portas Ethernet, wi-Fi e *bluetooth*. A Figura 4.1 mostra a estrutura básica do *Raspberry Pi 4* com as seguintes especificações:

- processador Quad core Cortex-A72 (ARM v8) 64-bit 1.8GHz de *clock*;
- memória SDRAM com 4GB LPDDR4-3200;
- conectividade;
  - físico: Gigabit Ethernet;
  - sem fio: 2.4 GHz e 5.0 GHz IEEE 802.11ac *wireless*, *bluetooth*B 5.0, BLE;
- 2 portas USB 3.0 e 2 portas USB 2.0.;
- 2 portas micro-HDMI.

O *Raspberry Pi 4* é uma boa opção para aplicações em *fog computing* por causa de seu baixo custo, baixo consumo de energia (5V/3A - 15W na porta USB-C) e tamanho compacto. Ele pode ser usado para executar tarefas de processamento de dados e análise

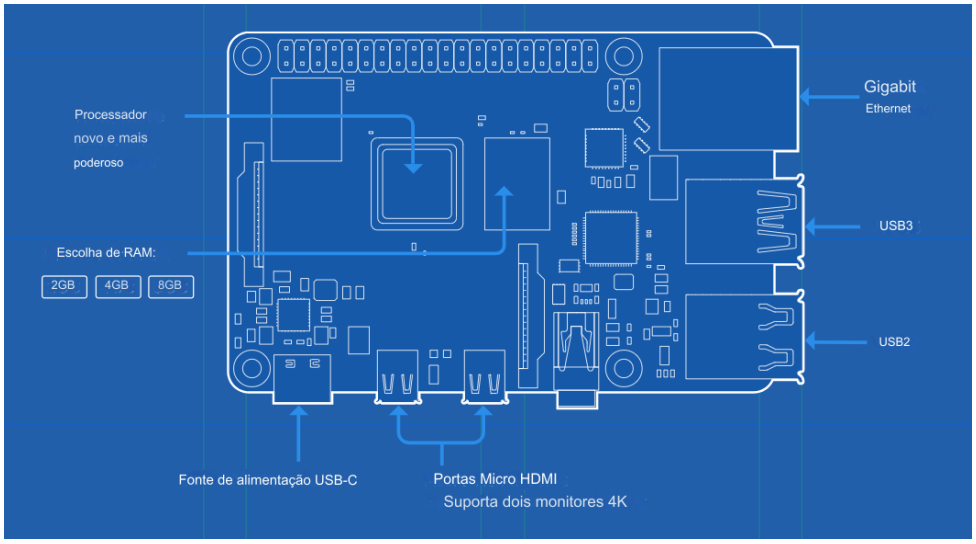

Figura 4.1: Estrutura de hardware do Raspberry Pi 4 [6].

de dados na borda da rede, perto dos dispositivos IoT que estão gerando os dados. Isso pode ajudar a reduzir a latência e melhorar o desempenho das aplicações.

Alguns exemplos de aplicações em *fog computing* que podem ser executadas em um *Raspberry Pi 4* incluem:

- Monitoramento de dispositivos IoT;
- Análise de dados de sensores;
- Processamento de imagens;
- Controle de robótica.

Assim sendo, o *Raspberry Pi 4* é um dispositivo versátil e acessível que pode ser usado para uma variedade de aplicações na *fog computing*.

4.4 Montagem Física

A montagem física de um *Raspberry Pi 4* é um processo relativamente simples que pode ser concluído em minutos. Neste trabalho este processo foi dividido em etapas, as quais serão descritas a seguir:

- Etapa 1: Montagem da caixa

A caixa protege o *Raspberry Pi 4* de danos. A Figura 4.2 mostra a caixa totalmente aberta. Logo em seguida, a Figura 4.3 apresenta a conexão da caixa com a ventoinha, e a Figura 4.4 fornece a imagem tanto da ventoinha quanto da placa SBC *Raspberry Pi 4* já conectados.

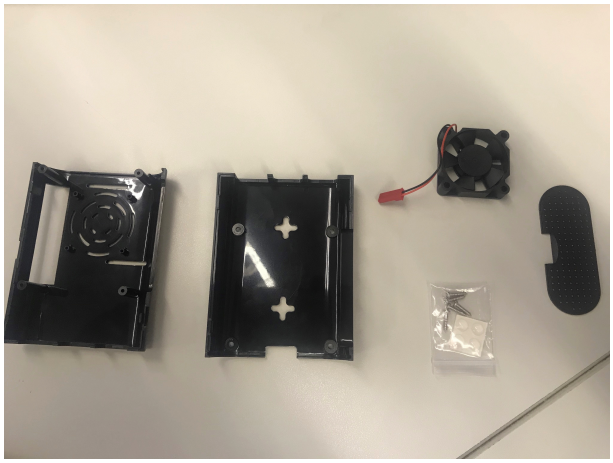

Figura 4.2: Caixa aberta.

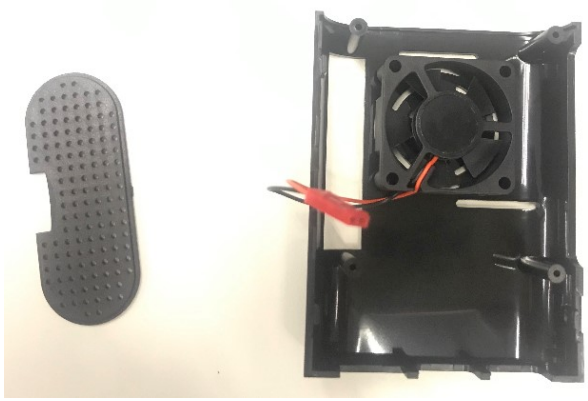

Figura 4.3: Ventoinha conectada.

- Etapa 2: Conexão do gpio

O *Raspberry Pi 4* tem alguns pinos de entrada e saída de propósito geral chamados gpio - *General Purpose Input Output*. A Figura 4.5 descreve toda a pinagem do gpio. Para ligar a ventoinha foram utilizados os pinos 4 e 6 porque são os pinos de 5V e *ground* respectivamente. A ativação da ventoinha é opcional, mas é recomendado para ajudar a manter o *Raspberry Pi 4* numa temperatura adequada e evitar o desgaste. A Figura 4.6 registra a conexão da ventoinha no gpio. Nesse mesmo escopo, é importante a utilização do conjunto de dissipador de calor, conforme mostrado na Figura 4.7.

- Etapa 3: Conexão dos periféricos, fonte e cartão de memória.

A Figura 4.8 mostra a conexão do mouse e do teclado e a Figura 4.9 mostra o *Raspberry Pi 4* completamente montado.

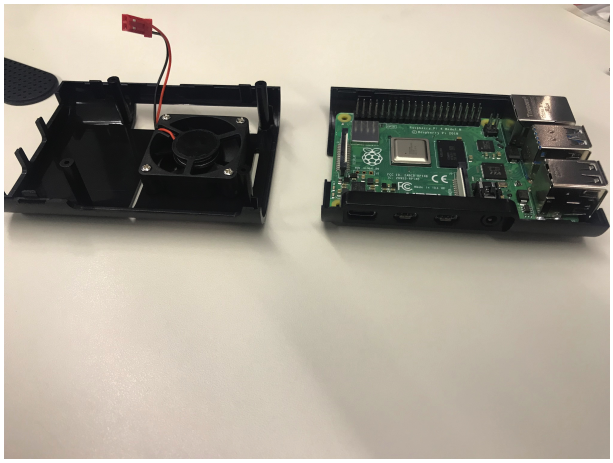

Figura 4.4: Ventoinha e placa conectadas.

Deste modo, a montagem de um *Raspberry Pi 4* é um processo acessível a qualquer pessoa, mesmo que não tenha experiência em eletrônica. Nesta instalação foram utilizados um *Raspberry Pi 4*, cartão microSD , fonte de alimentação, monitor ou TV com conexão HDMI, e teclado e mouse USB. Os passos foram descritos na Seção 4.2. Importante salientar que o cartão microSD esteja formatado em FAT32. O sistema operacional utilizado foi o Ubuntu Server 22.04.

Este processo de instalação foi implementado em 2 *Raspberry Pi 4* e estes foram conectados em uma rede local por meio de um *switch* com velocidade FastEthernet (10/100 Mbps) em cada porta.

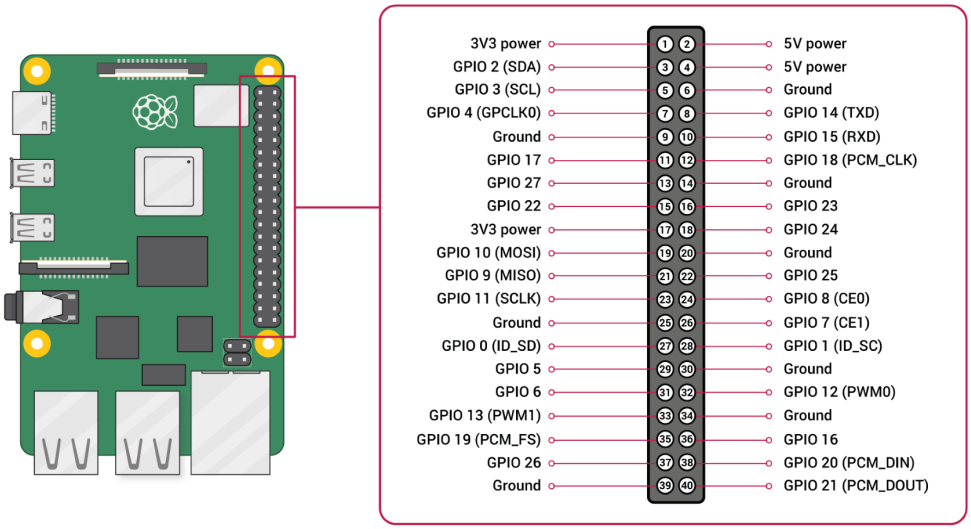

Figura 4.5: Diagrama gpio.

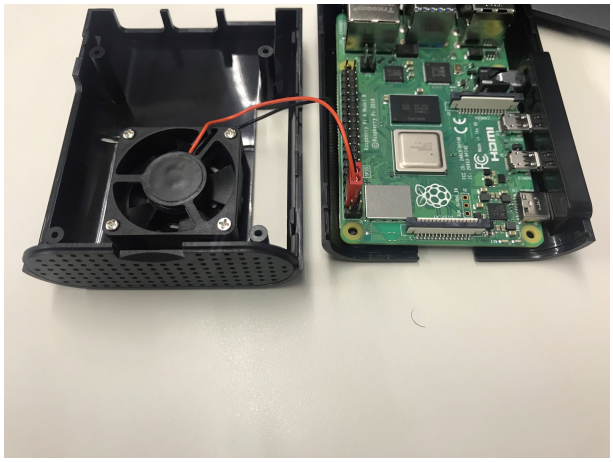

Figura 4.6: Ventoinha conectada no gpio.

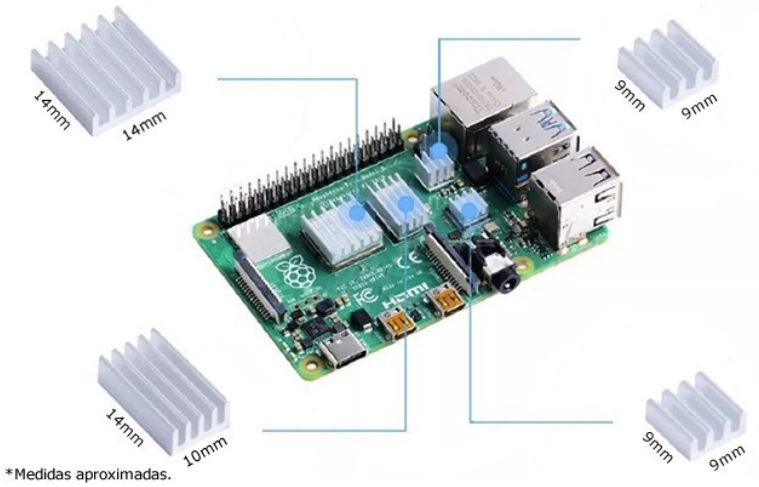

Figura 4.7: Dissipadores de calor.

Figura 4.8: Teclado e mouse.

Figura 4.9: *Raspberry Pi 4*.

Capítulo 5

Ambiente de Monitoração e Resultados

Este capítulo apresenta a pilha de software utilizada para o monitoramento de *fog computing*, e os testes que vão ser feitos utilizando esta pilha. Para isso, o capítulo está dividido em três seções. A Seção 5.1 descreve a instalação da desta pilha e propõe o *Raspberry Pi Docker Monitor*, uma solução de monitoramento para dispositivos *Raspberry Pi* que executam contêineres *Docker*. A Seção 5.2 mostra a instalação do *framework* de monitoramento e também a implementação do *Wordpress*, que foi utilizado como estudo de caso para fazer a avaliação de acesso a recursos web para análise de latência tanto na névoa quanto na nuvem. E a Seção 5.3 apresenta os resultados dos testes realizados.

5.1 *Raspberry Pi Docker Monitor*

O *Raspberry Pi Docker Monitor* é uma solução de monitoramento para dispositivos Raspberry Pi que executam contêineres Docker [72]. Ele consiste em um conjunto de ferramentas que coletam métricas de qualquer *host* e dos contêineres Docker e, em seguida, visualizam essas métricas em um painel baseado na web, conforme apresentado na Figura 5.1.

5.1.1 Componentes do *Monitor Raspberry Pi Docker*

Os principais componentes do Raspberry Pi Docker Monitor foram discutidos no Capítulo 3, e estão listados abaixo [73]:

- *Prometheus*: uma ferramenta de coleta de métricas que coleta dados do *host Raspberry Pi* e dos contêineres *Docker*;

Figura 5.1: Painel principal do *Raspberry Pi Docker Monitor*.

- *Grafana*: uma ferramenta de visualização que cria *dashboards* para exibir as métricas coletadas;
- *cAdvisor*: um coletor de métricas de tempo de execução de contêiner que fornece informações detalhadas sobre contêineres *Docker*;
- *Node-Exporter*: um coletor de métricas do sistema que fornece informações sobre o hardware e o sistema operacional do *Raspberry Pi*.

5.1.2 Benefícios do Monitor Docker Raspberry Pi

*Raspberry Pi Docker Monito*r oferece vários benefícios, tais como:

- Visibilidade aprimorada do desempenho dos contêineres *Raspberry Pi* e *Docker*: o monitor fornece uma visão centralizada de todas as principais métricas dos contêineres *Raspberry Pi* e *Docker*, facilitando a identificação e a solução de problemas de desempenho;
- Tempo de inatividade reduzido: ao monitorar proativamente os contêineres *Raspberry Pi* e *Docker*, é possível identificar e resolver possíveis problemas antes que eles causem tempo de inatividade;
- Utilização aprimorada de recursos: o monitor pode ajudá-lo a otimizar a utilização de recursos, identificando recursos subutilizados ou superutilizados.

5.2 Instalação

Existem algumas maneiras diferentes de instalar e usar o *Raspberry Pi Docker Monitor*. Neste trabalho será usada uma mescla de linha de comandos, junto com o *Portainer* para ser o mais simples possível.

5.2.1 Sistema Operacional

A distribuição utilizada no *Raspiberry Pi 4* foi Ubuntu Server 22.04 minimizado. Essa escolha foi feita por ser uma versão do Ubuntu Server que é instalada com um conjunto mínimo de software. Isso significa que ele é mais leve e eficiente em termos de recursos do que a versão padrão do Ubuntu Server. Assim sendo, utilizou-se a aplicação Raspberry Pi Imager no Wndows 10, a qual foi baixado a partir do sítio https://www.raspberrypi.com/software/, e instalado no cartão MicroSD o Ubuntu Server. As Figuras 5.2 e 5.3 mostram o início e a conclusão da instalação do sistema operacional.

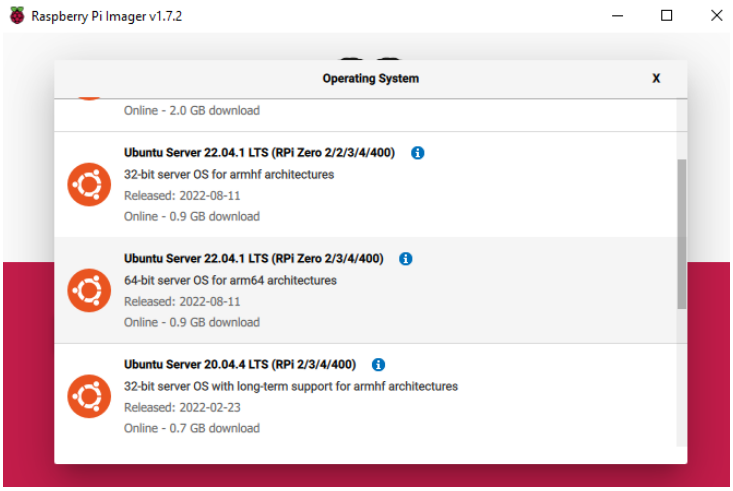

Figura 5.2: Imagem baixada e instalada no cartão MicroSD.

É importante ressaltar que durante a instalação do SO, fez-se a escolha pelo o pacote de recursos de acesso remoto seguro (SSH) como opção de aplicação disponível.

5.2.2 Ferramentas de Rede, *GitHub* e *Docker*

Como o sistema operacional está em sua forma mínima, fez-se necessário instalar duas ferramentas de rede muito importantes, o *Git Hub* e o *Docker* via linha de comando, as quais foram instaladas conforme os passos a seguir:

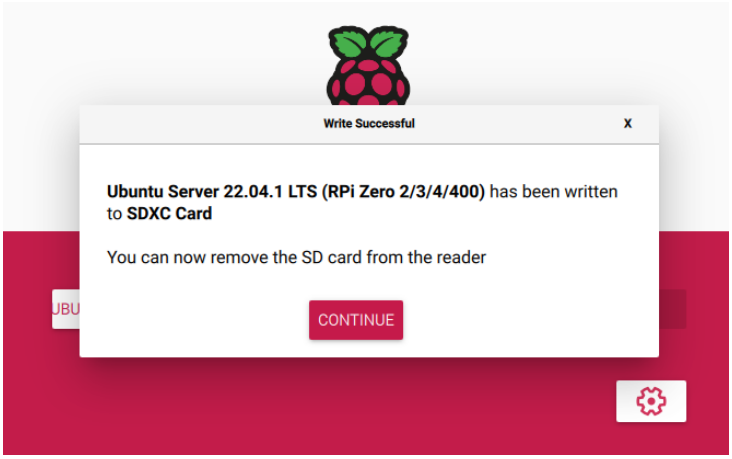

Figura 5.3: Finalização da instalação.

1. Instalação das ferramentas de rede *ping* (verificar conectividade entre dois *hosts*) e *ifconfig* (verificar estrutura da pilha TCP/IP e placa de rede) :

1. sudo apt update
2. sudo apt upgrade
3. sudo apt install net-tools
4. sudo apt install iputils-ping

2. Instalação do *Git Hub* e *download* do pacote de monitoramento do site https://github.com:

1. sudo apt install git
2. git config - -global user.name "William"
3. git config - -global user.email "william.santos@aluno.unb.br"
4. mkdir Downloads
5. cd Downloads
6. git clone https://github.com/wilxavier/pi-hosted

A pasta /pi-hosted, que está disponível no *GitHub* para qualquer usuário baixar, contem os *scripts* e todas as ferramentas de instalação, conforme apresentado na Figura 5.4.

```
slave@raspi02:~$ cd Downloads
slave@raspi02:~/Downloads$ git clone https://github.com/wilxavier/pi-hos
ted
Cloning into 'pi-hosted'...
remote: Enumerating objects: 8007, done.
remote: Counting objects: 100% (669/669), done.
remote: Compressing objects: 100% (169/169), done.
remote: Total 8007 (delta 494), reused 637 (delta 478), pack-reused 733
Receiving objects: 100% (8007/8007), 16.84 MiB | 19.25 MiB/s, done.
Resolving deltas: 100% (4753/4753), done.
slave@raspi02:~/Downloads$ ls
pi-hosted
slave@raspi02:~/Downloads$ cd pi-hosted/
slave@raspi02:~/Downloads/pi-hosted$ ls
apptemplate.png  install_docker.sh       template
build            install_portainer.sh    tools
configs          pi-hosted_template      update_portainer.sh
docs             reinstall_portainer.sh
images           stack
slave@raspi02:~/Downloads/pi-hosted$
```

Figura 5.4: Pasta /pi-hosted com os arquivos de instalação.

3. Instalação do *Docker* e do *Portainer*:

1. cd */Downloads/pi-hosted*
2. sudo ./install_docker.sh
3. sudo ./install_portainer.sh
4. cd Downloads/pi-hosted/tools/
5. sudo ./rpi_docker_monitor.sh
6. sudo apt install vim (editor de arquivos)
7. cd /boot
8. mkdir firmware
9. cd /boot/firmware

5.2.3 *Raspberry Pi Docker Monitor*

O primeiro passo para a instalação é habilitar *c-groups* (grupos de controle que são um recurso do *kernel* Linux que permite alocar recursos, como tempo de CPU, memória e largura de banda de rede, entre grupos de processos ordenados hierarquicamente) para que a pilha de monitoramento funcione imediatamente. Para fazer isso é preciso criar e editar o arquivo de configuração *cmdline.txt* que é armazenado no Ubuntu

22.04 em */boot/firmware/cmdline.txt*, inserindo na primeira linha do arquivo o comando cgroup_enable cgroup_memory=1 descrito na Figura 5.5. Na sequência, é necessário salvar o arquivo e reiniciar o sistema operacional: *sudo reboot*.

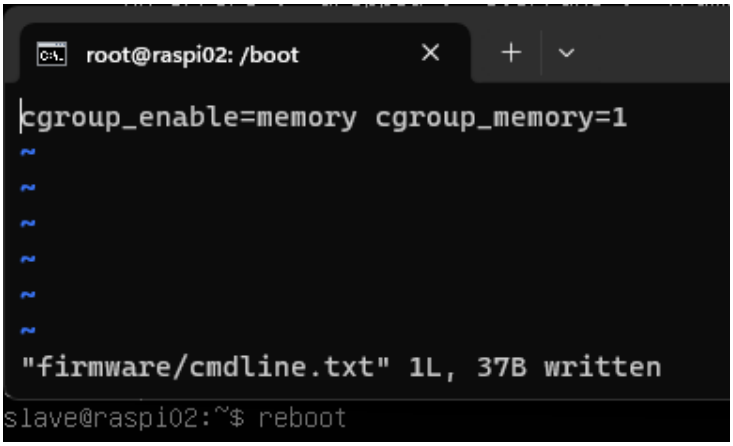

Figura 5.5: Comando para configurar a memória no grupo de controle.

O comando *cat /proc/cgroups* confirma se o *c-groups* está habilitado. A Figura 5.5 mostra o resultado. Assim, nesta Figura 5.6 é importante verificar a memória na lista. Se não estiver, é importante confirmar que o comando está no arquivo correto. Além disso, para continuar é preciso que o *memory* apareça na listagem.

```
#subsys_name    hierarchy       num_cgroups     enabled
cpuset  9       15      1
cpu     7       69      1
cpuacct 7       69      1
blkio   8       69      1
memory  11      158     1
devices 3       69      1
freezer 5       16      1
net_cls 2       15      1
perf_event      6       15      1
net_prio        2       15      1
pids    4       76      1
rdma    10      1       1
```

Figura 5.6: Grupos de controle.

Dentro do Portainer

Depois de instalado o *Docker* e o *Portainer* foi dado o comando *ifconfig* para saber qual foi o IP que o *host* recebeu. A porta padrão configurado no *script install_portainer.sh* é a 9000. Em qualquer navegador *web* é necessário digitar o ipMáquina:9000 para entrar no *Portainer*, conforme Figura 5.7, e instalar a pilha de monitoramento. Depois de criar o usuário é importante reiniciar o servidor novamente.

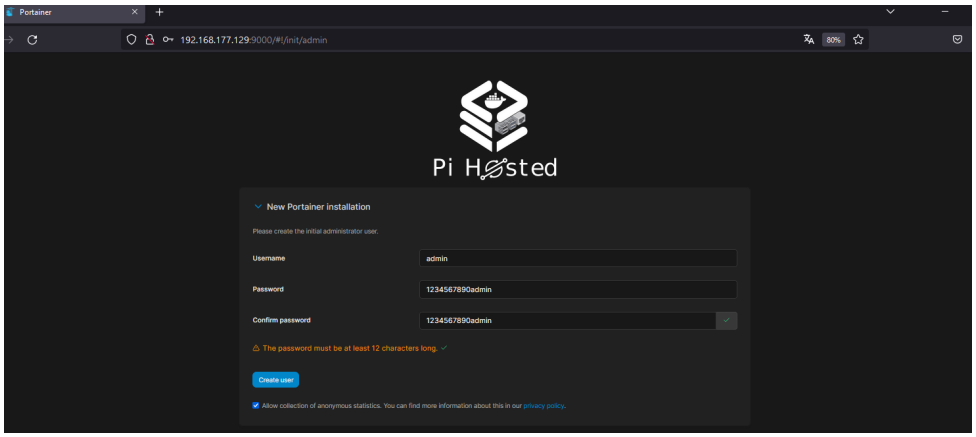

Figura 5.7: Tela de login *PI Hosted* (*Portainer*).

Instalação do *Template Docker Monitor*

Em configurações, conforme Figura 5.8, foi trocada a URL em modelos de aplicativos:

- de *https://raw.githubusercontent.com/portainer/templates/master/templates-2.0.json*
- para *https://raw.githubusercontent.com/pi-hosted/pi-hosted/master/template/portainer-v2-arm32.json*

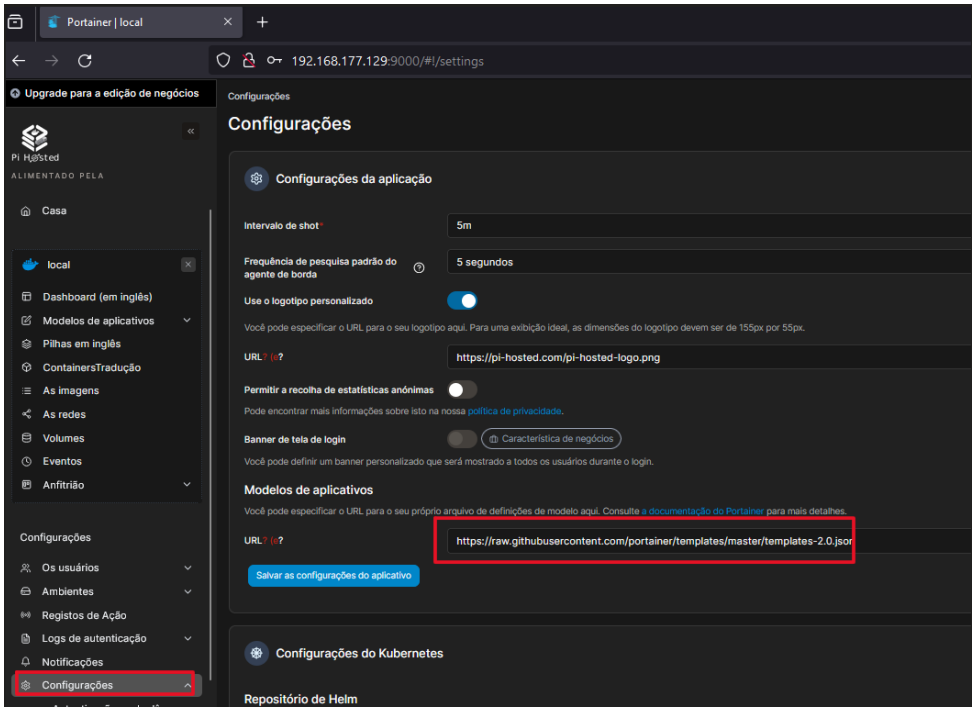

Figura 5.8: Tela de login *Configuração* (*Portainer*).

Em seguida, em modelos de aplicativos, no ícone lupa, foi digitado "Raspberry Pi Docker"e selecionado (conforme Figura 5.9), e implantando a pilha conforme Figura 5.10. E finalmente, conforme indicada na Figura 5.11, a pilha de monitoramento foi instalada. Com *cAdvisor*, *Grafana*, *Node-exporter* e *Prometheus* sendo executados normalmente.

Configuração do Grafana

O próximo e último passo foi configurar o*Grafana* com as informações que serão coletadas dos *hosts*. Para acessar o *Grafana* é necessário obter o IP da máquina e a porta padrão que será a 3000 (exemplo: ipMáquina:3000, com login *admin* e senha *admin*. Em seguida, é necessário configurar a tela inicial para que apareçam, de forma gráfica, as informações coletadas dos *hosts*. Em configuração foi adicionado, em *data source*, o *Prometheus e, em seguida, foi copiado o IP que esse conteiner recebeu na pilha de monitoramento (conforme*

*apresent*ado na Figura 5.11) com a porta 9090. Foi utilizado o seguinte caminho para fazer a configuração do *Prometheus*: Home -> Connections -> Data sources -> *Prometheus*. (indicado na Figura 5.12).

Em seguida, foi configurado painél de visualização. Home -> Dashboards -> Import dashboard. Copiado e colado o seguinte arquivo jason, amd_rpi_dashboard.json, encontrado na url *https://github.com/wilxavier/pi-hosted/tree/master/configs/rpi_dashboard*. Importando, carregando e escolhendo o Promehteus ao final. Finalizando, deste modo, a etapa de instalacção do sistema de monitoramento (conforme Figura 5.1).

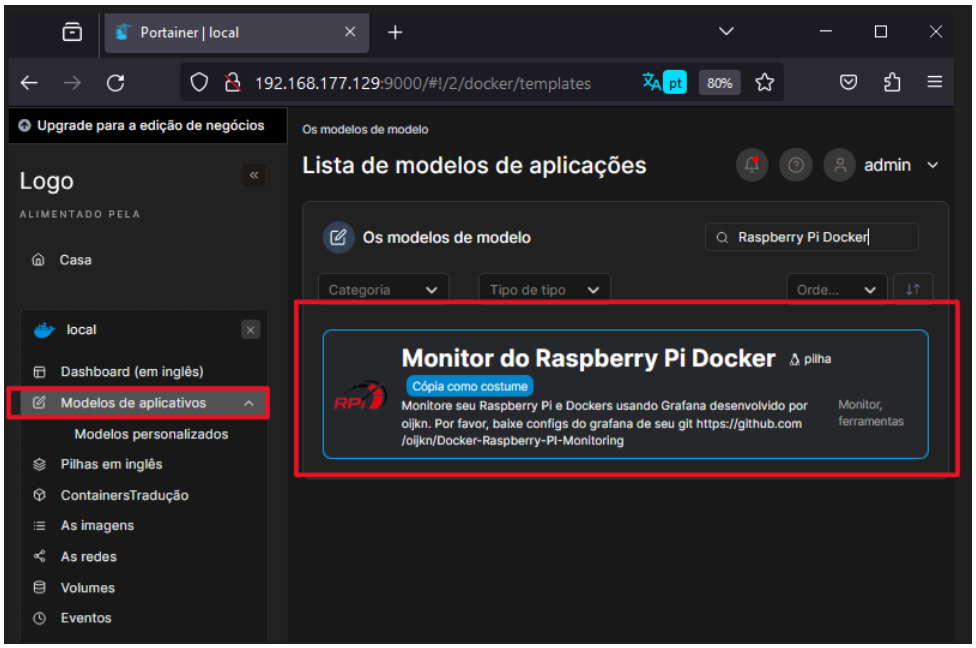

Figura 5.9: Escolha da pilha de monitoramento.

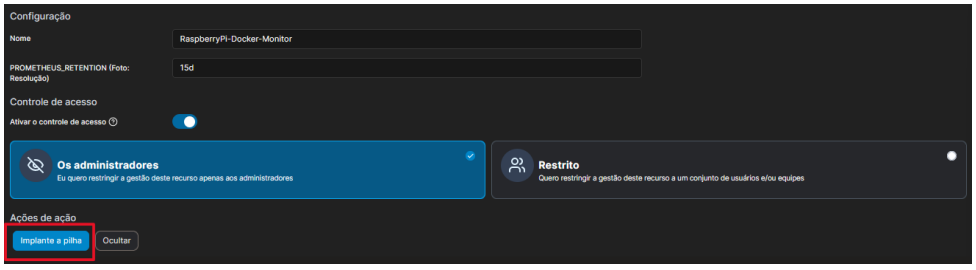

Figura 5.10: Implantação da pilha de monitoramento.

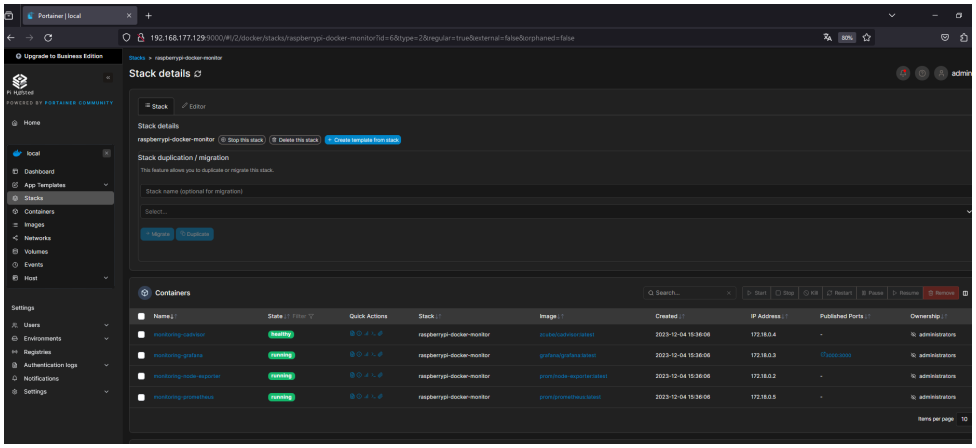

Figura 5.11: A pilha de monitoramento.

Configuração do *Wordpress*

Com o *Portainer* já configurado, instalar o *WordPress* exigirá apenas alguns cliques usando o *template* já disponibilizado. Alguns passos necessários são:

1. No menu à esquerda, clique em *App Templates* ou modelos de aplicativos, como mostrado na Figura 5.9;
2. No ícone lupa procure por *Wordpress*;
3. Clique no botão *Deploy* ou implantar;
4. Na janela de implantação, insira os valores de nome do servidor (que será o nome do site *Wordpress*), nome de usuário do banco de dados, senha do banco de dados e o endereço IP do servidor onde o Wordpress será implantado.

Depois que o *WordPress* for implantado, o acesso estará disponível no navegador usando o endereço IP do servidor e a porta 80.

5.3 *Resultados*

Neste trabalho o principal caso de uso foi monitorar o desempenho de um aplicativo *Web* em execução em um contêiner *Docker*. O monitor pode ser usado para rastrear métricas como uso de *CPU*, uso de memória e tráfego de rede. Essas informações podem ser usadas para identificar e solucionar problemas de desempenho do aplicativo *Web*.

Nesse contexto, foi idealizado um ambiente em *fog* para manipulação de dispositivos que requisitam pouco recurso de hardware e de rede (baixa latência <= 1ms), em comparação com dois servidores *Linux* (com a mesma configuração de *hardware* do *Raspberry Pi*) que estão na nuvem (um no Brasil e o outro nos EUA). O objetivo é mostrar a variação

Host	Camadas	Latência
Notebook – Raspberry Pi 4	Edge – Fog	<1ms
Notebook – VM Nuvem	Edge – Cloud Brasil	23ms
Notebook – VM Nuvem	Edge – Cloud EUA	339ms

Tabela 5.1: Métrica de latência

de latência entre a nuvem e a *fog*. O ambiente de *fog computing* foi estruturado usando dois *Raspberry Pi 4* e um *host* como dispositivo final. O recurso de monitoramento de *software* que foi implementado nos quatro nós foi baseado usando a solução de monitoramento descrita na seção anterior, sendo hospedado em um Servidor Ubuntu Server 22.04. Para o monitoramento desses processos foram utilizados quatro contêineres: *Prometheus*, *Grafana*, *cAdvisor* e *NodeExporter*, e um contêiner *Wordpress* para simular uma estrutura de servidor Web, que neste caso é um processo Web, que será sempre requisitado pelos dispositivos finais. Estes se integram entre si formando um sistema de monitoramento de processos para análise de métricas baseadas nas latências de cada contêiner e recursos de *hardware*.

A Figura 5.14 descreve o *Network Traffic on Node* - (tráfego de rede em cada nó). Recurso de monitoramento escolhido neste trabalho para monitorar a latência em cada dispositivo descrito na Figura 5.15. A linha verde é o servidor hospedado nos EUA, a amarela é o servidor hospedado no Brasil e a azul são os *Raspberry Pi*. O gráfico informa que os atrasos são maiores de acordo com a localização de cada dispositivo.

Assim sendo, foi obtido êxito nessa análise, principalmente com as métricas de monitoramento de recursos na arquitetura *fog computing* relacionados à latência, abrindo um leque de questões tanto a respeito do monitoramento como também provisionamento de recursos e, ao mesmo tempo, contemplando os paradigmas de computação que é o objetivo deste trabalho: *cloud* e *fog computing*, exemplificado na Figura 5.14.

O *Fog node* foi projetado para ser um *sniffer* da comunicação entre IoT e *cloud computing* gerando métricas para que pudessem ser avaliados. Para coletar resultados reais foi medida a latência de rede entre os dispositivos na borda e o nó de névoa. Além disso, foram feitas medições similares para as duas máquinas virtuais disponíveis na camada de nuvem. Assim sendo, foram feitas dez medições em cada cenário e calculada a média aritmética entre os resultados. Os resultados nos mostraram que a latência média dessa comunicação entre borda da rede e nó da névoa foi muitas vezes menor do que a mesma comunicação realizada entre máquinas virtuais na nuvem, usando também centros de dados da nuvem localizados em países diferentes para evidenciar como a localização geográfica pode alterar a latência. Os valores de latência média foram compostos em testes, IoT para *Fog*: <1ms, IoT para *Cloud* Brasil: 23ms, e IoT para Cloud EUA: 339ms, conforme apresentado na Tabela 5.1.

Sendo assim, a computação em névoa é mais adequada para viabilização de serviços em tempo real, no qual o cenário exige baixa latência de comunicação entre borda e os serviços que precisam consumir, sendo todo o tráfego monitorado por um *framework* de monitoramento de recursos no nó da névoa.

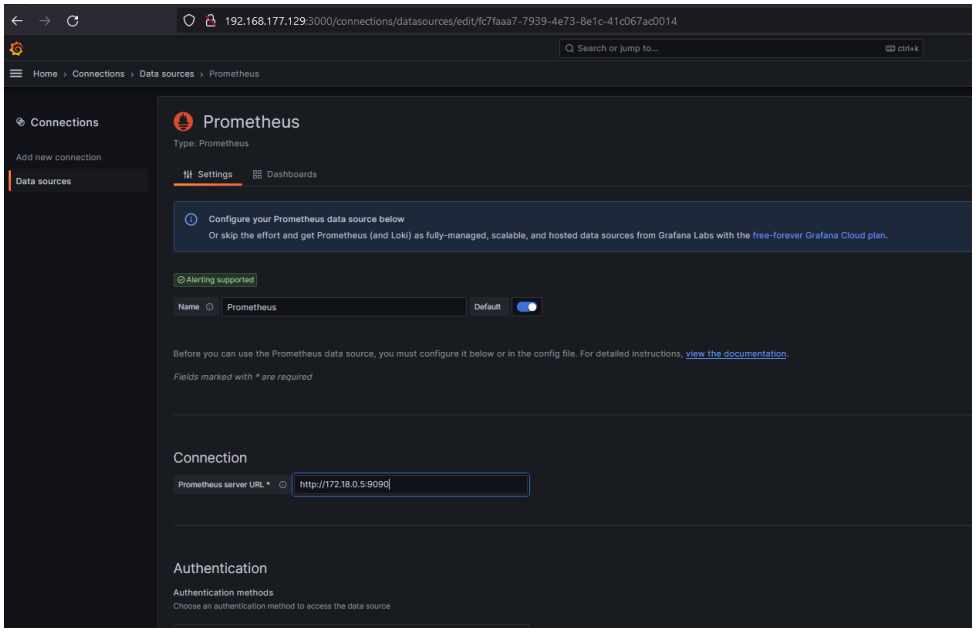

Figura 5.12: Adição do Prometheus.

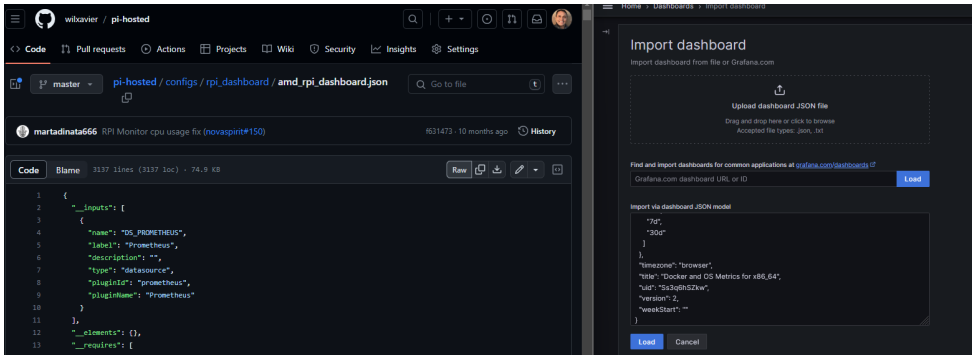

Figura 5.13: Criação do painel de monitoramento.

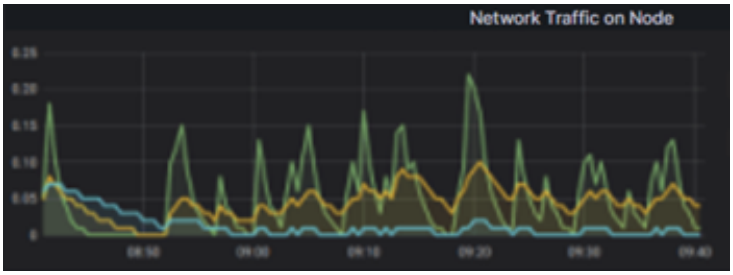

Figura 5.14: Monitoramento de métrica em cada nó.

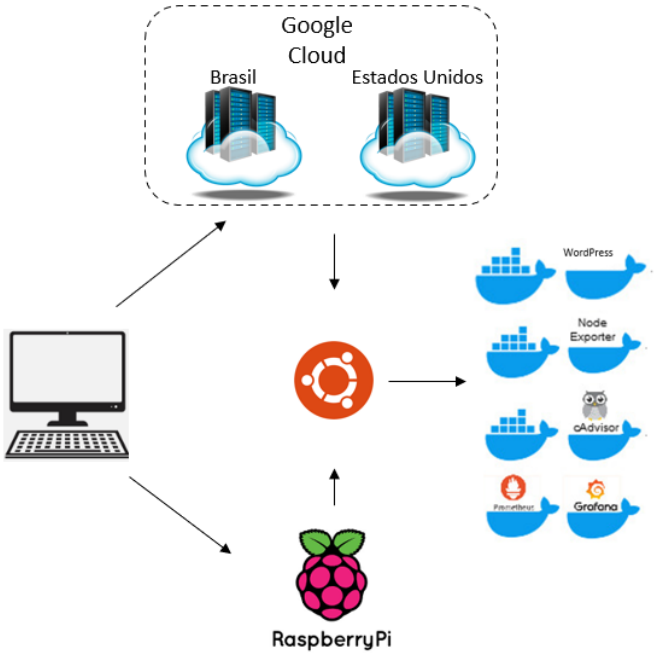

Figura 5.15: Criação do painel de monitoramento.

Capítulo 6

Conclusão

Neste capítulo, são apresentadas as considerações finais deste Trabalho de Conclusão de Curso, o qual teve como objetivo principal desenvolver um ambiente real de *fog computing* por meio de dispositivos do tipo *Raspberry Pi* a fim de garantir um ambiente para implantação e análise de soluções de monitoramento nesta infraestrutura. Inicialmente, buscou-se analisar aplicações de monitoramento *Open Source* disponíveis no mercado para análise na computação em *fog*. Para isso, foram realizados estudos complementares sobre os paradigmas computacionais relacionados à computação em *fog* e sobre os *fog nodes*. Esses estudos permitiram definir, inicialmente, quais ferramentas usar no processo de monitoramento de recursos na computação em *fog* como composto pelos Capítulos 4 e 5.

Outro ponto importante foi definir a descrição de um ambiente de *fog computing* e como ela se diferencia da computação em nuvem, sendo mais adequada para a viabilização de serviços de tempo real, em que se exige baixa latência das comunicações entre os usuários na borda da rede e os serviços que precisam consumir. Para evidenciar as características da *fog*, um ambiente de testes foi criado e foi instalado um *framework* de monitoramento de recursos no nó da *fog*.

Os resultados iniciais mostraram uma latência muitas vezes menor nas comunicações entre o usuário e um nó da *fog*, em comparação com a comunicação realizada com os nós da nuvem. A proposta foi submetida a uma análise experimental na qual avaliou-se a métrica latência, porém há outras métricas tais como: uso de memória, temperatura, tráfego entre os contêineres e etc. Este trabalho indica os passos inicias para construção de um modelo de monitoramento mais robusto com as ferramentas apresentadas nos capítulos anteriores.

Como trabalho futuro, o ambiente de testes será acrescido de mais dispositivos em cada camada, alocará mais contêineres, analisará mais métricas, e deverá lidar com um maior volume de dados. Os dados coletados pelo serviço de monitoramento serão utilizados por processos de gerenciamento de recursos, demonstrando a relevante integração entre os processos.

Referências

[1] Yousefpour, Ashkan, Caleb Fung, Tam Nguyen, Krishna Kadiyala, Fatemeh Jalali, Amirreza Niakanlahiji, Jian Kong e Jason P. Jue: *All one needs to know about fog computing and related edge computing paradigms: A complete survey.* Journal of Systems Architecture, (December 2018), 2019, ISSN 13837621. ix, 1, 9, 11, 19

[2] Union, International Telecommunication: *ITU-T y.4460 Recommendation - Architectural reference models of devices for internet of things applications. released 06/2019*, 2019. ix, 10, 12

[3] OpenFog: *Openfog consortium architecture working group*, February 2017. ix, 8, 12, 14

[4] Marín-Tordera, Eva, Xavi Masip-Bruin, Jordi García-Almiñana, Admela Jukan, Guang Jie Ren e Jiafeng Zhu: *Do we all really know what a fog node is? current trends towards an open definition.* Computer Communications, 109:117–130, 2017. ix, 14

[5] Bachiega, Joao, Breno Gustavo Soares da Costa e Aleteia P. F. Araujo: *Computational perspective of the fog node.* Em *22nd International Conference on Internet Computing IoT*. ACSE, 2021. ix, 2, 9, 10, 13, 14, 16, 17, 21

[6] *Raspiberry pi documentation.* `https://www.raspberrypi.org/`. ix, 31, 33

[7] Mell, Peter, Tim Grance *et al.*: *The nist definition of cloud computing.* 2011. 1, 5

[8] Vaquero, Luis M, Luis Rodero-Merino, Juan Caceres e Maik Lindner: *A break in the clouds: towards a cloud definition.* ACM SIGCOMM Computer Communication Review, 39(1):50–55, 2008. 1

[9] Costa, Breno GS, Marco Antonio Sousa Reis, Aletéia PF Araújo e Priscila Solis: *Performance and cost analysis between on-demand and preemptive virtual machines.* Em *CLOSER*, páginas 169–178, 2018. 1

[10] Taivalsaari, Antero e Tommi Mikkonen: *A roadmap to the programmable world: software challenges in the iot era.* IEEE Software, 34(1):72–80, 2017. 1

[11] Toczé, Klervie e Simin Nadjm-Tehrani: *A Taxonomy for Management and Optimization of Multiple Resources in Edge Computing.* Wireless Communications and Mobile Computing, 2018, 2018. 1, 2, 18

[12] Bonomi, Flavio, Rodolfo Milito, Jiang Zhu e Sateesh Addepalli: *Fog computing and its role in the internet of things*. Em *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, MCC '12, páginas 13–16, New York, NY, USA, 2012. ACM, ISBN 978-1-4503-1519-7. http://doi.acm.org/10.1145/2342509.2342513. 1

[13] Dastjerdi, Amir Vahid, Harshit Gupta, Rodrigo Neves Calheiros, Soumya K. Ghosh e Rajkumar Buyya: *Fog computing: Principles, architectures, and applications*. CoRR, abs/1601.02752, 2016. 2, 7

[14] Bachiega Jr., João, Breno Costa, Leonardo Carvalho, Victor Hugo Oliveira, William Santos, Maria Clícia S. de Castro e A. Araujo: *From the sky to the ground: Comparing fog computing with related distributed paradigms.* Em *Proceedings of the 12th International Conference on Cloud Computing and Services Science (CLOSER 2022)*, páginas 158–169, 2022. 2, 7, 18

[15] Foster, Ian, Yong Zhao, Ioan Raicu e Shiyong Lu: *Cloud computing and grid computing 360-degree compared*. arXiv preprint arXiv:0901.0131, 2008. 4

[16] Kushida, Kenji E, Jonathan Murray e John Zysman: *Cloud computing: From scarcity to abundance*. Journal of Industry, Competition and Trade, 15(1):5–19, 2015. 4

[17] Iorga, Michaela, Larry Feldman, Robert Barton, Michael Martin, Nedim Goren e Charif Mahmoudi: *The nist definition of fog computing*. Relatório Técnico, National Institute of Standards and Technology, 2018. 5, 7, 8, 10, 30

[18] Mukherjee, Mithun, Lei Shu e Di Wang: *Survey of fog computing: Fundamental, network applications, and research challenges*. IEEE Communications Surveys and Tutorials, 20(3):1826–1857, 2018. 5, 6

[19] Hu, Pengfei, Sahraoui Dhelim, Huansheng Ning e Tie Qiu: *Survey on fog computing: architecture, key technologies, applications and open issues*. Journal of Network and Computer Applications, 98(April):27–42, 2017. http://dx.doi.org/10.1016/j.jnca.2017.09.002. 5

[20] Naha, Ranesh Kumar, Saurabh Garg, Dimitrios Georgakopoulos, Prem Prakash Jayaraman, Longxiang Gao, Yong Xiang e Rajiv Ranjan: *Fog computing: Survey of trends, architectures, requirements, and research directions*. IEEE Access, 6:47980–48009, 2018. 5, 6, 7, 9, 17

[21] Kritikos, K. e D. Plexousakis: *Multi-cloud application design through cloud service composition*. Em *2015 IEEE 8th Int. Conf. on Cloud Computing*, páginas 686–693, June 2015. 6

[22] Keahey, Katarzyna, Maurício Tsugawa, Andréa Matsunaga e José A. B. Fortes: *Sky computing*. Em *IEEE Internet Computing*, página 43–51. IEEE Computer Society, 2009. 6

[23] Alhamazani, Khalid, Rajiv Ranjan, Prem Prakash Jayaraman, Karan Mitra, Chang Liu, Fethi Rabhi, Dimitrios Georgakopoulos e Lizhe Wang: *Cross-layer multi-cloud real-time application qos monitoring and benchmarking as-a-service framework*. IEEE Transactions on Cloud Computing, 7(1):48–61, 2015. 6

[24] Paraiso, F., N. Haderer, P. Merle, R. Rouvoy e L. Seinturier: *A federated multi-cloud paas infrastructure*. Em *2012 IEEE Fifth Int. Conf. on Cloud Computing*, páginas 392–399, June 2012. 6

[25] Grozev, Nikolay e Rajkumar Buyya: *Inter-cloud architectures and application brokering: taxonomy and survey*. Software: Practice and Experience, 44(3):369–390, 2014. `https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.2168`, acesso em 05/07/2020. 6

[26] Wamser, Florian, Chiara Lombardo, Constantinos Vassilakis, Lam Dinh-Xuan, Paolo Lago, Roberto Bruschi e Phuoc Tran-Gia: *Orchestration and monitoring in fog computing for personal edge cloud service support*. Em *2018 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*, páginas 91–96. IEEE, 2018. 6

[27] Shirinkin, Kirill: *Getting Started with Terraform*. Packt Publishing Ltd, 2017. 6

[28] Cloudify: *Getting started*, 2021. `https://cloudify.co/getting-started/`, acesso em 2021-11-29. 6

[29] Michelino, D., J. C. Leon e L. F. Alvarez: *Implementation and testing of openstack heat*, setembro 2013. `https://doi.org/10.5281/zenodo.7571`. 6

[30] Fan, Chih Tien, Zong You Wu, Che Pin Chang e Shyan Ming Yuan: *Web resource cacheable edge device in fog computing*. Proceedings - 15th International Symposium on Parallel and Distributed Computing, ISPDC 2016, (November 2010):432–439, 2017. 7

[31] Vaquero, Luis M. e Luis Rodero-Merino: *Finding your way in the fog: Towards a comprehensive definition of fog computing*. SIGCOMM Comput. Commun. Rev., 44(5):27–32, outubro 2014, ISSN 0146-4833. `http://doi.acm.org/10.1145/2677046.2677052`. 7

[32] Yi, Shanhe, Zijiang Hao, Zhengrui Qin e Qun Li: *Fog computing: Platform and applications*. Em *2015 Third IEEE workshop on hot topics in web systems and technologies (HotWeb)*, páginas 73–78. IEEE, 2015. 7

[33] Cisco: *Fog computing and the internet of things: Extend the cloud to where the things are*, 2015. `https://www.cisco.com/c/dam/en-us/solutions/trends/iot/docs/computing-overview.pdf`. 7

[34] IBM: *What is fog computing?*, 2014. `https://www.ibm.com/blogs/cloud-computing/2014/08/25/fog-computing/`. 7

[35] Gill, Sukhpal Singh, Peter Garraghan e Rajkumar Buyya: *Router: Fog enabled cloud based intelligent resource management approach for smart home iot devices*. Journal of Systems and Software, 2019. 8

[36] Al-Doghman, F., Z. Chaczko, A. R. Ajayan e R. Klempous: *A review on fog computing technology*. Em *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, páginas 001525–001530, Oct 2016. 9

[37] Perera, Charith, Yongrui Qin, Julio C Estrella, Stephan Reiff-Marganiec e Athanasios V Vasilakos: *Fog computing for sustainable smart cities: A survey*. ACM Computing Surveys (CSUR), 50(3):1–43, 2017. 11

[38] Taneja, Mohit e Alan Davy: *Resource aware placement of iot application modules in fog-cloud computing paradigm*. Em *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, páginas 1222–1228. IEEE, 2017. 13

[39] Abouaomar, Amine, Soumaya Cherkaoui, Abdellatif Kobbane e Oussama Abderrahmane Dambri: *A resources representation for resource allocation in fog computing networks*. Em *2019 IEEE Global Communications Conference (GLOBECOM)*, páginas 1–6. IEEE, 2019. 13

[40] Goldberg, Robert P: *Survey of virtual machine research*. Computer, 7(6):34–45, 1974. 14

[41] Mann, Zoltán Ádám: *Notions of architecture in fog computing*. Computing, páginas 1–23, 2020. 14

[42] Tiburski, Ramao Tiago, Carlos Roberto Moratelli, Sergio F Johann, Marcelo Veiga Neves, Everton de Matos, Leonardo Albernaz Amaral e Fabiano Hessel: *Lightweight security architecture based on embedded virtualization and trust mechanisms for iot edge devices*. IEEE Communications Magazine, 57(2):67–73, 2019. 14

[43] Merkel, Dirk: *Docker: lightweight linux containers for consistent development and deployment*. Linux journal, 2014(239):2, 2014. 14, 23, 24

[44] Maenhaut, Pieter Jan, Bruno Volckaert Veerle Ongenae e Filip De Turck: *Resource management in a containerized cloud: Status and challenges*. Em *2020 Journal of Network and Systems Management*, página 197–246. JNSM, 2020. 14

[45] FogMon, Adaptive: *Adaptive fogmon online*, 2022. `https://github.com/veracoo/FogMon/tree/adaptive-fogmon`, . Accessed in June, 2022. 15

[46] Katoh, Naoki e Toshihide Ibaraki: *Resource allocation problems*. Em *Handbook of combinatorial optimization*, páginas 905–1006. Springer, 1998. 17

[47] Mahmud, Redowan, Kotagiri Ramamohanarao e Rajkumar Buyya: *Towards observability data management at scale*. 1(1), 2021. 18

[48] Sudhakara, M, K Dinesh Kumar, Ravi Kumar Poluru, R Lokesh Kumar e S Bharath Bhushan: *Towards efficient resource management in fog computing: A survey and future directions*. Em *Architecture and Security Issues in Fog Computing Applications*, páginas 158–182. IGI Global, 2020. 18

[49] Mustafa, Saad, Babar Nazir, Amir Hayat, Atta Ur Rehman Khan e Sajjad A. Madani: *Resource management in cloud computing: Taxonomy, prospects, and challenges.* Computers and Electrical Engineering, 47:186–203, 2015. 18

[50] Manvi, Sunilkumar S. e Gopal Krishna Shyam: *Resource management for Infrastructure as a Service (IaaS) in cloud computing: A survey.* Journal of Network and Computer Applications, 41(1):424–440, 2014. 18

[51] Singh, Sukhpal e Inderveer Chana: *Cloud resource provisioning: survey, status and future research directions.* Knowledge and Information Systems, 49(3):1005–1069, 2016. 18

[52] Datta, Soumya Kanti, Rui Pedro Ferreira Da Costa e Christian Bonnet: *Resource discovery in internet of things: Current trends and future standardization aspects.* Em *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, páginas 542–547. IEEE, 2015. 18

[53] Masip-Bruin, X, E Marín-Tordera, A J Ferrer, A Salis, J Kennedy, J Jensen, A Jukan, A Bartoli, R M Badia, M Cankar e M E Bégin: *mF2C: The Evolution of Cloud Computing Towards an Open and Coordinated Ecosystem of Fogs and Clouds.* Lecture Notes in Computer Science, 11997 LNCS:136–147, 2020, ISSN 03029743. `https://www.scopus.com/inward/record.uri?eid=2-s2.0-85086225692{&}doi=10.1007{%}2F978-3-030-48340-1{_}11{&}partnerID=40{&}md5=8d5cd5d3a7255b089ff7254f4a4375b9`. 18

[54] Mahmud, Md e Rajkumar Buyya: *Fog Computing: A Taxonomy, Survey and Future Directions.* novembro 2016, ISBN 978-981-10-5861-5. 18, 22

[55] Aazam, Mohammad e Eui Nam Huh: *Dynamic resource provisioning through fog micro datacenter.* Em *2015 IEEE international conference on pervasive computing and communication workshops (PerCom workshops)*, páginas 105–110. IEEE, 2015. 19

[56] Li, Qiuping, Junhui Zhao, Yi Gong e Qingmiao Zhang: *Energy-efficient computation offloading and resource allocation in fog computing for internet of everything.* China Communications, 16(3):32–41, 2019. 19

[57] Mostafa, Nour, Ismaeel Al Ridhawi e Moayad Aloqaily: *Fog resource selection using historical executions.* Em *2018 Third International Conference on Fog and Mobile Edge Computing (FMEC)*, páginas 272–276. IEEE, 2018. 19

[58] Arpaci-Dusseau, Remzi H, Andrea Arpaci-Dusseau e Venkat Venkataramani: {*Cloud-Native*} *file systems.* Em *10th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 18)*, 2018. 19

[59] Abderrahim, Mohamed, Meryem Ouzzif, Karine Guillouard, Jerome Francois e Adrien Lebre: *A holistic monitoring service for fog/edge infrastructures: a foresight study.* Em *2017 IEEE 5th International Conference on Future Internet of Things and Cloud (FiCloud)*, páginas 337–344. IEEE, 2017. 19

[60] Karumuri, Suman, Franco Solleza, Stan Zdonik e Nesime Tatbul: *Towards observability data management at scale*. ACM SIGMOD Record, 49(4):18–23, 2021. 19, 20

[61] Brandón, Álvaro, María S Pérez, Jesus Montes e Alberto Sanchez: *Fmone: A flexible monitoring solution at the edge*. Wireless Communications and Mobile Computing, 2018, 2018. 20

[62] Ewaschuk, Rob e Betsy Beyer: *Monitoring distributed systems. site reliability engineering: How google runs production systems, chapter 6*, 2016. 20

[63] Costa, Breno, João Bachiega, Leonardo Rebouças Carvalho, Michel Rosa e Aleteia Araujo: *Monitoring fog computing: A review, taxonomy and open challenges*. Computer Networks, 215:109189, 2022, ISSN 1389-1286. `https://www.sciencedirect.com/science/article/pii/S1389128622002845`. 20

[64] Nguyen, Duong Tung, Long Bao Le e Vijay K Bhargava: *A market-based framework for multi-resource allocation in fog computing*. IEEE/ACM Transactions on Networking, 2019. 20

[65] Viejo, Alexandre e David Sánchez: *Secure and privacy-preserving orchestration and delivery of fog-enabled iot services*. Ad Hoc Networks, 82:113–125, 2019. 20

[66] Bonomi, Flavio, Rodolfo Milito, Preethi Natarajan e Jiang Zhu: *Fog computing: A platform for internet of things and analytics*. Em *Big data and internet of things: A roadmap for smart environments*, páginas 169–186. Springer, 2014. 20

[67] Nath, S B, S Chattopadhyay, R Karmakar, S K Addya, S Chakraborty e S K Ghosh: *PTC: Pick-Test-Choose to Place Containerized Micro-Services in IoT*. Em *2019 IEEE Global Communications Conference (GLOBECOM)*, páginas 1–6, 2019. 21

[68] Luo, Juan, Luxiu Yin, Jinyu Hu, Chun Wang, Xuan Liu, Xin Fan e Haibo Luo: *Container-based fog computing architecture and energy-balancing scheduling algorithm for energy iot*. Future Generation Computer Systems, 97:50–60, 2019. 21

[69] BMijuskovic, Adriana, Alessandro Chiumento Rob Bemthuis Adina Aldea e Paul Havinga: *Resource management techniques for cloud/fog and edge computing: An evaluation framework and classification*. 21(5):30–42, 2021. 22

[70] *Ubuntu server*. `https://ubuntu.com/server`, acesso em 2023-11-11. 22

[71] Hong, Cheol Ho e Blesson Varghese: *Resource management in fog/edge computing: A survey on architectures, infrastructure, and algorithms*. ACM Computing Surveys (CSUR), 52(5):97, 2019. 23

[72] *Docker - empowering App Development for Developers*. `https://www.docker.com/`. 23, 27, 28, 39

[73] *Overview of docker hub*. `https://docs.docker.com/docker-hub/`, acesso em 2023-11-11. 23, 39

[74] *Welcome - portainer documentation.* https://docs.portainer.io/, acesso em 2023-11-11. 24

[75] *Monitoring docker container metrics using cadvisor.* https://prometheus.io/docs/guides/cadvisor/, acesso em 2023-11-11. 25

[76] Prometheus: *Prometheus*, 2022. https://prometheus.io/, . Accessed February 28, 2022. 25

[77] *Welcome - prometheus documentation.* https://prometheus.io/docs/, acesso em 2023-11-11. 25

[78] *Welcome - grafana documentation.* https://grafana.com/docs/, acesso em 2023-11-11. 26, 28

[79] *Welcome - node prpmetheus documentation.* https://prometheus.io/docs/guides/node-exporter/, acesso em 2023-11-11. 27