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

Uma Arquitetura Autoadaptável para a Implantação de Observabilidade em Fog Computing

Breno Gustavo Soares da Costa

Tese apresentada como requisito parcial para
conclusão do Doutorado em Informática

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

Brasília
2025

Ficha Catalográfica de Teses e Dissertações

Está página existe apenas para indicar onde a ficha catalográfica gerada para dissertações de mestrado e teses de doutorado defendidas na UnB. A Biblioteca Central é responsável pela ficha, mais informações nos sítios:

http://www.bce.unb.br
http://www.bce.unb.br/elaboracao-de-fichas-catalograficas-de-teses-e-dissertacoes

Esta página não deve ser inclusa na versão final do texto.

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

Uma Arquitetura Autoadaptável para a Implantação de Observabilidade em Fog Computing

Breno Gustavo Soares da Costa

Tese apresentada como requisito parcial para
conclusão do Doutorado em Informática

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

Prof.a Dr.a Alba Cristina Magalhães Alves de Melo
CIC/UnB

Prof. Dr. Edson Norberto Cáceres
FACOM/UFMS

Prof. Dr. Eugene Francis Vinod Rebello
IC/UFF

Prof. Dr. Rodrigo Bonifácio de Almeida
Coordenador do Programa de Pós-graduação em Informática

Brasília, 28 de Março de 2025

Dedicatória

Dedico esta tese à Tatiana, a Pedro e a Heitor, meus três amores.

Agradecimentos

Agradeço à Tatiana pelo apoio incondicional e pela compreensão. Sem seu apoio, teria sido muito difícil me dedicar o necessário para fazer pesquisa e escrever. Agradeço a meus filhos, Pedro e Heitor, por aceitarem uma menor participação minha nos momentos de lazer, nos períodos em que eu precisei focar no doutorado.

A meu pai e minhas mães, obrigado pelo incentivo (algumas vezes, estímulo!) à leitura e ao pensamento crítico. São as melhores ferramentas que tenho como pesquisador. A Zezinho, agradeço por compartilhar livros e histórias, que consolidaram meu gosto pela literatura.

À Aletéia Araújo, minha querida orientadora, reservo imensa gratidão por ter me guiado de forma tão participativa, tão fraterna e tão sincera por esse caminho tortuoso que é fazer pesquisa acadêmica no Brasil. O seu compromisso com os resultados, com a pesquisa, e com os seres humanos por trás dos alunos é admirável e exemplar.

Aos meus colegas do doutorado, em especial João e Leonardo, agradeço a amizade, a solicitude e o espírito de equipe, que me proporcionaram não apenas bons resultados, mas também boas histórias!

Agradeço aos Professores Prem Prakash Jayaraman e Abhik Banerjee por terem me recebido por seis curtos meses na Swinburne University of Tecnology, Austrália, no âmbito do doutorado sanduíche.

Agradeço também ao projeto BioCloud2, aprovado no Edital CNPq/AWS nº 064/2022, processo CNPq nº 421828/2022-6. À CAPES, pelo acesso ao Portal Periódicos e pela bolsa do doutorado-sanduíche, recursos que me ajudaram a evoluir como pesquisador. Por fim, agradeço ao Estado Brasileiro que, por meio do Departamento de Ciência da Computação (CIC) da UnB e de uma política pública de sucesso, deu-me oportunidade de cursar um doutorado de alto nível gratuitamente.

Resumo

Fog Computing é um paradigma computacional que estende a *Cloud Computing*, fornecendo recursos de computação mais próximos dos usuários na borda da rede. O paradigma *Fog Computing* distingue-se por possuir uma infraestrutura significativamente mais distribuída e heterogênea em comparação com *Cloud Computing*, o que, por sua vez, aumenta a complexidade do gerenciamento em relação à *Cloud Computing*. A orquestração de serviços e recursos é fundamental nesse contexto, lidando com a dinamicidade da infraestrutura e garantindo o cumprimento dos Acordos de Nível de Serviço. A gestão da observabilidade é uma funcionalidade crucial para a orquestração, coletando informações sobre o *status* dos serviços, dos dispositivos e dos *links* de comunicação para permitir uma tomada de decisão rápida e eficaz. No entanto, a literatura sobre orquestração em *Fog Computing*, frequentemente, assume a existência de uma solução de gestão da observabilidade sem apresentar métodos de implementação, ou abordar os desafios. Adicionalmente, soluções de gestão da observabilidade existentes para *Cloud Computing* não são adequadas para ambientes *Fog* devido às suas particularidades. Há uma carência de trabalhos que abordem o aumento da observabilidade em *Fog* e o desafio de gerenciar diversos fluxos de dados heterogêneos em um ambiente com recursos restritos. Para suprir essas lacunas, esta tese propõe FogObserver, uma arquitetura de referência para sistemas de gestão da observabilidade em *Fog*, que lida com a coleta, o processamento e o armazenamento de dados de observabilidade. Ela gerencia fluxos de dados heterogêneos dos domínios de instrumentação (métricas, *logs* e *traces*) e utiliza um *framework* autoadaptável, capaz de identificar de forma dinâmica alterações significativas no ambiente e de selecionar de modo autônomo a resposta mais apropriada para assegurar a continuidade da operação do sistema. A avaliação da proposta foi realizada por meio de um estudo de caso em um cenário real de cidades inteligentes. Os resultados mostraram que é possível aumentar a observabilidade em *Fog Computing* de forma eficaz, adicionando um *overhead* baixo à infraestrutura e aos canais de comunicação. Por meio de estratégias customizadas para o contexto da aplicação, conseguiu-se uma redução de 80% no volume de dados de observabilidade, transmitidos dos dispositivos IoT para *Fog*, e o volume resultante representou menos de 1% do volume de dados transmitidos pela aplicação quando em operação. Além do pequeno impacto sobre o *overhead*, foi observado um aumento significativo no nível de observabilidade, de 1 para 6, quando comparado às soluções existentes na literatura.

Palavras-chave: Cloud Computing, Fog Computing, Observabilidade, Adaptabilidade, Arquitetura Autoadaptável

Abstract

Fog Computing is a computational paradigm that extends *Cloud Computing*, providing computing resources closer to users at the edge of the network. The *Fog Computing* paradigm is characterized by a distributed and heterogeneous infrastructure, which increases management complexity compared to *Cloud Computing*. The orchestration of services and resources is fundamental in this context, dealing with the dynamic nature of the infrastructure and ensuring compliance with Service Level Agreements. Observability management is a crucial functionality for orchestration, collecting information about the *status* of services, devices, and communication channels to allow quick and effective decision making. However, the literature on orchestration in *Fog Computing* often assumes the existence of an observability management solution without presenting implementation methods or addressing the challenges. Additionally, existing observability management solutions for *Cloud Computing* are not suitable for *Fog* environments due to their specificities. There is a lack of work addressing the increase in observability level in *Fog* and the challenge of managing various heterogeneous data flows on an environment with restricted resources. To fill these gaps, this thesis proposes FogObserver, a reference architecture for observability management systems in *Fog*, which deals with the collection, processing, and storage of observability data. It manages heterogeneous data flows from the instrumentation domains (metrics, *logs*, and *traces*) and uses a *framework* that is self-adaptive, capable of dynamically recognizing relevant changes in the environment and autonomously selecting the most appropriate response to ensure the continuity of system operation. The proposal was evaluated through a case study in a real smart city scenario. The results demonstrated that it is possible to effectively increase observability in *Fog Computing* without adding a high *overhead* to the infrastructure and communication channels. Through customized strategies for the application context, a reduction of 80% in the volume of observability data transmitted from IoT devices to *Fog* was achieved, and the resulting volume represented less than 1% of the data volume transmitted by the application while in operation. In addition to the small impact on overhead, a significant increase in the level of observability from 1 to 6 was observed when compared to existing solutions in the literature.

Keywords: Cloud Computing, Fog Computing, Observability, Adaptability, Self-adaptable architecture

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Cloud Computing é um modelo de computação que permite o acesso remoto a um conjunto de recursos computacionais compartilhados, como servidores, armazenamento, aplicativos e serviços, pela internet. Esse modelo facilita o gerenciamento, a escalabilidade e a manutenção de recursos, oferecendo flexibilidade e economia de custo, pois os usuários pagam apenas pelo que utilizam, sem a necessidade de manter infraestrutura local. *Fog Computing* é um paradigma computacional que complementa *Cloud Computing*, fornecendo recursos computacionais na borda da rede, mais próximo dos usuários. Como uma infraestrutura mais distribuída, *Fog Computing* deve lidar com a heterogeneidade de *links* de rede e capacidade de processamento de seus dispositivos [2]. Essas características trazem complexidade adicional ao gerenciamento de *Fog*, quando comparada com *Cloud Computing*, que tem maior homogeneidade de dispositivos e de conexões, além de ter menor distribuição geográfica.

A orquestração de serviços e recursos em *Fog* é uma área recente que estrutura o gerenciamento do ambiente *Fog*. Ela é composta por várias funcionalidades complementares, e é responsável por lidar com a dinamicidade da infraestrutura, por tomar ações oportunas e por garantir que os Acordos de Nível de Serviço (SLA, do inglês *Service Level Agreement*) sejam respeitados [3].

O monitoramento é uma funcionalidade de primordial importância e é crucial para orquestrar adequadamente os serviços de *Fog* [4]. Ele coleta informações de *status* atualizadas sobre os serviços em execução, os *Fog Nodes* e os *links* de comunicação, e as envia ao orquestrador. Com uma visão abrangente e atualizada da infraestrutura de *Fog* e dos serviços em execução, o orquestrador pode tomar as ações adequadas para garantir os SLAs. Por exemplo, descarregar um serviço para um *Fog Node* com maior capacidade, e otimizar o posicionamento do serviço de acordo com dados históricos sobre falhas dos *Fog Nodes* [5].

Além de coletar *status* de disponibilidade e outras métricas objetivas, como percentual de CPU em uso, espaço em disco livre, e tempo de resposta de uma aplicação, outros tipos de informação geradas pelos serviços mostraram-se relevantes, como os *logs* e os *traces*. *Logs* trazem registros de sucesso ou de erro na execução de funções dos sistemas. *Traces* mostram a hierarquia de chamadas de subfunções e os tempos de execução de cada uma. Estas informações aumentam a observabilidade dos sistemas, que é a possibilidade de se conhecer o estado interno do sistema a partir de sua interface externa. Ao ter mais detalhes do que ocorre internamente com os sistemas, é possível agilizar o entendimento de um problema (sistema apresentando lentidão, ou com falha) e resolvê-lo mais rapidamente, diminuindo assim a indisponibilidade e melhorando os indicadores de SLA.

Entretanto, como *Fog* é composta por dispositivos limitados em recursos e conectada por *links* heterogêneos e instáveis, injetar um grande volume de dados adicionais na estrutura pode prejudicar o seu funcionamento. Assim, é fundamental encontrar o equilíbrio entre o aumento da observabilidade, que traz benefícios à operação e à manutenção dos sistemas, e a sobrecarga que esse aumento pode causar em um ambiente restrito.

Este capítulo apresenta a introdução da Tese, que analisa o contexto do aumento da observabilidade de serviços e aplicações que executam em *Fog Computing* e apresenta estratégias e propostas para alcançá-lo. Assim, na Seção 1.1 é detalhada a motivação desta pesquisa. Em seguida, na Seção 1.2 são apresentados os objetivos que conduzem este trabalho e na Seção 1.3 são elencadas as contribuições. Ao final, na Seção 1.4, é detalhada a estrutura dos demais capítulos desta Tese.

1.1 Motivação

Uma revisão sistemática da literatura, publicada pelo autor desta tese e coautores, sobre orquestração de serviços em *Fog* [5], identificou que a maioria dos artigos analisados destacava o monitoramento como uma funcionalidade relevante, mas frequentemente supunha que uma solução de monitoramento de *Fog* estaria disponível para fornecer as informações necessárias, sem apresentar métodos de implementação, discutir desafios ou apresentar informações específicas sobre o assunto.

Todavia, o monitoramento não é apenas sobre relatar a disponibilidade, ou seja, a capacidade de responder à pergunta se um *Fog Node* ou serviço está *online* e funcionando corretamente. O monitoramento trata também de explicar porque um *Fog Node* ou serviço parou de funcionar corretamente. A disponibilidade pode ser inferida a partir da análise de métricas, por exemplo, tempo de resposta do serviço. A causa para um mal-funcionamento pode ser descoberta a partir da análise de *logs* – *strings* de texto não estruturadas – e *traces* – registros de solicitações feitas por um usuário em um serviço. Métricas, *logs* e *traces* formam o que é chamado de domínios de instrumentação [1]. Diferentes domínios de instrumentação podem ser usados simultaneamente para se obter diferentes perspectivas de um serviço. Nesse cenário, há mais capacidade de tomada de decisão do lado do servidor, mas ao custo de aumento da complexidade, pois as características específicas de cada domínio de instrumentação (por exemplo, ciclo de vida, volume de dados) devem ser gerenciadas simultaneamente.

No âmbito dos sistemas distribuídos, recentemente a literatura começou a usar o termo "gestão da observabilidade" para representar esse monitoramento mais abrangente, que lida com vários domínios de instrumentação simultaneamente, que responde não apenas se um serviço ou aplicação está disponível ou não, mas também o porquê de o serviço ou aplicação não estar funcionando como deveria [6]. A gestão da observabilidade, portanto, é definida como um superconjunto de monitoramento que utiliza técnicas de análise nos dados coletados, com o objetivo de diminuir o tempo necessário para restaurar o serviço ou a aplicação a um estado operacional.

Neste contexto, diversos trabalhos analisaram soluções de gestão da observabilidade em *Cloud* e verificaram que nenhuma delas é adequada para uso em ambientes de *Fog* [7–10]. Apenas o trabalho [9] analisou soluções de gestão da observabilidade de *Fog*, e essa análise incluiu somente duas propostas [7, 8]. Além

disso, nenhum desses trabalhos abordou o aumento da observabilidade em *Fog* e, portanto, não lidaram com o desafio de tratar diversos fluxos de dados heterogêneos em um ambiente restrito.

Para lidar com essas limitações, este trabalho mapeia o ciclo de vida dos dados de observabilidade em *Fog* e propõe uma forma de medir o nível de observabilidade de uma aplicação ou serviço a partir destes dados. Nesta tese também é proposta a FogObserver, que é uma arquitetura de sistema de gestão de observabilidade para *Fog* que prevê o gerenciamento de diversos fluxos de dados dos domínios de instrumentação. Além disso, é proposto um *framework* autoadaptável para a arquitetura FogObserver, o qual é capaz de reconhecer dinamicamente cenários e situações que necessitem de atuação rápida, selecionar e implementar a melhor reação ao cenário identificado no ambiente.

1.2 Objetivos

O objetivo geral desta tese de doutorado é investigar o aumento da observabilidade da infraestrutura e dos sistemas que executam em *Fog Computing*. Essa investigação considera os desafios impostos pela heterogeneidade dos dispositivos e a instabilidade das conexões, que precisarão lidar com um maior tráfego de dados devido ao incremento da observabilidade. Para cumprir este objetivo geral, esta tese deve alcançar os seguintes objetivos específicos:

- Compreender o funcionamento de sistemas de gestão da observabilidade: modelos de operação, funcionalidades que proveem, desafios que enfrentam;
- Compreender o contexto em que a gestão da observabilidade atua, identificando os recursos de que necessita e as informações que provê para outros processos importantes de *Fog Computing*;
- Comparar as soluções de gestão de observabilidade disponíveis na literatura para identificar os avanços realizados e os desafios que ainda estão em aberto nesse campo;
- Mapear o ciclo de vida dos dados de observabilidade em Fog;
- Realizar um estudo de caso para verificar a viabilidade de aumentar a observabilidade da infraestrutura e de sistemas que executam em *Fog*;

1.3 Contribuições

Esclarecer as contribuições de uma tese é essencial para destacar o impacto e a originalidade do trabalho no campo de estudo. Assim, esta seção apresenta as principais inovações teóricas e práticas deste trabalho, as quais são consolidadas como:

- Definição de uma taxonomia de soluções de gestão da observabilidade em *Fog Computing*, que pode ser usada para classificar as soluções disponíveis na literatura, reforçando o entendimento do estado da arte nessa área;
- Mapeamento do ciclo de vida dos dados de observabilidade em *Fog* para identificar suas fases, requisitos e desafios, o que permitirá a proposição de estratégias adequadas de gerenciamento dos dados no ambiente *Fog*;

- Proposta de uma arquitetura de sistemas de gestão da observabilidade em *Fog* que supra as lacunas identificadas na literatura e que apoie o ciclo de vida dos dados de observabilidade, respeitando as características e limitações do ambiente;
- Proposta de *framework* para tornar autoadaptáveis os sistemas de gestão da observabilidade de *Fog*. Esse *framework* visa a lidar com a dinamicidade e a heterogeneidade do ambiente, adaptando o comportamento do sistema de acordo com as condições da rede e dos dispositivos;
- Realização de estudo de caso, composto pela execução de uma aplicação real de Internet das Coisas (IoT, do termo em inglês) para cidades inteligentes, juntamente com um sistema de gestão da observabilidade que utiliza a estrutura da arquitetura proposta. O sistema é composto por ferramentas de código aberto e permite demonstrar as características, os benefícios e os desafios de aumentar a observabilidade de sistemas em *Fog*.

1.4 Sumário desta Tese

Para uma melhor organização do conteúdo a ser apresentado, optou-se por estruturar esta tese em três partes. A primeira parte (Referencial Teórico), composta por três capítulos, apresenta a fundamentação teórica e os conceitos abordados ao longo da tese. O Capítulo 2 define *Fog Computing*, *Fog Node*, faz um comparativo de *Fog* com os demais paradigmas relacionados, e contextualiza a orquestração de serviços em *Fog Computing*. O Capítulo 3 apresenta os requisitos e os desafios de um sistema de gestão da observabilidade em *Fog* e descreve um caso de uso de cidades inteligentes, que será utilizado como cenário motivador para avaliação das propostas da tese. O Capítulo 4 apresenta os conceitos relacionados à adaptabilidade de sistemas distribuídos.

A segunda parte da tese (Contribuições), composta por cinco capítulos, descreve as contribuições resultantes desta pesquisa. O Capítulo 5 define e descreve uma nova taxonomia para classificação de sistemas de gestão da observabilidade em *Fog*. O Capítulo 6 apresenta os resultados de uma revisão sistemática da literatura que selecionou as soluções de observabilidade em *Fog* existentes, que são caracterizadas com o uso da nova taxonomia definida. Na sequência, o Capítulo 7 apresenta um mapeamento do ciclo de vida dos dados de observabilidade em *Fog*, e define uma forma de medir o nível de observabilidade de um sistema, assim como o *overhead* que ela adiciona. O Capítulo 8 propõe a FogObserver, uma arquitetura de sistemas de gestão da observabilidade em *Fog* que é autoadaptável. O Capítulo 9 usa um estudo de caso de cidades inteligentes para avaliar os desafios de aumentar a observabilidade em *Fog* e avaliar a arquitetura proposta.

Por fim, as conclusões da pesquisa e os trabalhos futuros são discutidos na terceira parte desta Tese. Além do Capítulo 10, com as conclusões e sugestões de trabalhos futuros, essa última parte contém as referências bibliográficas e cinco anexos relativos à produção científica derivada da pesquisa realizada nesta tese, apresentando a primeira página de cada artigo publicado.

Parte I

Referencial Teórico

Capítulo 2

Fundamentos de *Fog Computing*

Este capítulo tem como objetivo apresentar as características e os desafios de *Fog Computing*, paradigma de computação distribuída que estende a *Cloud Computing*, provendo serviços mais próximos aos usuários finais na borda da rede. Um dos componentes da arquitetura de *Fog Computing* é o *Fog Node*, um componente de hardware ou de software em que os serviços são executados. Além de *Fog Computing*, há outros paradigmas computacionais distribuídos, propostos com o objetivo de estender a *Cloud Computing*, e que se diferenciam de *Fog Computing* em termos de poder de processamento, distribuição, flexibilidade, entre outras variáveis. Uma breve apresentação dos principais paradigmas, assim como um comparativo entre eles e *Fog Computing* é apresentado. Por fim, a orquestração de serviços em *Fog* é apresentada, ressaltando a complexidade de gerenciamento do ambiente, e o monitoramento se consolida como uma das principais funcionalidades neste contexto. O estudo comparativo entre os paradigmas computacionais foi publicado pelo autor desta tese e coautores em [11]. Um estudo acerca da perspectiva computacional do *Fog Node* foi publicado pelos mesmos autores em [12]. A revisão sistemátia da literatura sobre orquestração em *Fog* foi publicada em [5].

Assim, a estrutura deste capítulo é dividida em cinco seções. Inicialmente é apresentada a definição de *Cloud Computing* (Seção 2.1) e de *Fog Computing* (Seção 2.2). Na sequência, a Seção 2.3 apresenta uma breve definição dos demais paradigmas computacionais analisados e faz um comparativo entre eles. A Seção 2.4 define "orquestração" em *Fog*, apresenta os resultados de uma revisão sistemática da literatura sobre o tema, e aponta a relevância do monitoramento neste contexto. Por fim, a Seção 2.5 mostra uma síntese dos principais conceitos apresentados neste capítulo.

2.1 *Cloud Computing*

O conceito subjacente de *Cloud Computing* foi introduzido em 1961 por John McCarthy quando ele disse que, no futuro, a computação poderia ser organizada como um serviço público, como era o sistema telefônico naqueles dias [13]. A evolução da computação e da comunicação permitiu que o paradigma de *Cloud Computing* se expandisse e, hoje em dia, esta plataforma é amplamente utilizada pela academia e indústrias. *Cloud Computing* desempenha um papel importante como infraestrutura computacional e impacta todas as outras tecnologias e paradigmas apresentados neste capítulo.

Assim, tem-se observado, nas últimas décadas, que recursos computacionais, antes caros e escassos, agora se tornaram baratos e abundantes [14]. A *Cloud Computing* surgiu nesse contexto de transição, possibilitando a democratização da computação. Logo, a *Cloud Computing* abrangeu dois pontos importantes, os quais são a disponibilidade de volume muito alto de recursos computacionais e a rápida escalabilidade dos recursos. Assim sendo, a *Cloud Computing* possibilita o consumo de recursos computacionais de forma similar a que usamos para consumir energia, água, etc.

Na literatura, é possível encontrar no artigo [15] que as cinco características essenciais para ambientes de *Cloud Computing* são definidas como: autoatendimento sob demanda, amplo acesso à rede, *pool* de recursos, elasticidade rápida, e medição dos serviços em baixa granularidade. Portanto, as principais vantagens da *Cloud Computing* são a disponibilidade de alto poder computacional e grande armazenamento, pagando apenas pelo recurso consumido. A elasticidade tem uma função fundamental neste contexto, permitindo o crescimento ou a diminuição dos recursos de forma dinâmica. Mas a *Cloud Computing* também tem algumas restrições. A limitação fundamental é a conectividade entre a *Cloud* e os dispositivos finais e usuários.

Cloud Computing se fundamenta na interconexão de um número reduzido de datacenters, caracterizados por um grande volume de recursos computacionais que podem estar geograficamente distantes. Embora a velocidade de processamento de dados tenha aumentado significativamente, a largura de banda da rede, mesmo em conexões de alta velocidade como a fibra ótica, está se tornando um fator limitante em *Cloud Computing* devido à grande quantidade de dados [16]. Como consequência, essa infraestrutura não é adequada para solicitações de serviço de tempo real [17]. A superação dessas limitações beneficiará casos de uso específicos como veículos conectados, cidades inteligentes, realidade virtual, e saúde [18].

No cenário atual de *Cloud Computing*, a abordagem chamada de *Sky Computing* [19] destaca-se pela consolidação de recursos de diversos provedores, oferecendo uma visão unificada dos serviços disponíveis. Isso é crucial para organizações que adotam estratégias *multicloud*, lidando com a complexidade de provedores variados. O uso de orquestradores, como Terraform [20] e Cloudify [21], auxilia nesse processo de integração e gestão dinâmica, conforme as demandas dos usuários.

2.2 *Fog Computing*

O conceito de *Fog Computing* foi apresentado em 2012 pela Cisco [2] para abordar os desafios das aplicações de IoT que operam em ambientes de *Cloud Computing*. Yi *et al.* [22] consideram que a *Fog Computing* é composta de uma grande quantidade de dispositivos descentralizados e heterogêneos, e que esses dispositivos se comunicam e cooperam entre si e com a rede para realizar o processamento e o armazenamento de dados, sem a intervenção de terceiros. A *Fog Computing* é um paradigma de computação distribuída que atua como uma camada intermediária entre os *datacenters* da *Cloud* e os dispositivos IoT, diminuindo a latência característica da *Cloud* ao usar recursos ociosos de vários dispositivos próximos aos usuários finais [18, 23].

As seis características de *Fog Computing*, definidas pelo NIST [24], são cruciais para permitir um ambiente mais eficiente e escalável. Essas características, coletivamente, aumentam a capacidade de *Fog Computing* de processar e analisar dados mais perto da fonte, reduzindo a latência e melhorando a velo-

cidade dos processos de tomada de decisão. Essas seis características essenciais de um ambiente de *Fog Computing* são definidas da seguinte forma [24]:

- Baixa latência: ao conhecer a localização lógica dos dispositivos IoT e estar em sua proximidade;
- Distribuição geográfica: permite que um serviço ou aplicação seja amplamente distribuído fisicamente;
- Heterogeneidade: dispositivos com diferentes capacidades computacionais e com pilhas de software heterogêneas cooperam usando múltiplos tipos de redes;
- Interoperabilidade: um serviço pode abranger a infraestrutura de diferentes provedores;
- Interações em tempo real: as aplicações de *Fog Computing* envolvem interações em tempo real em vez de processamento em lote;
- Escalabilidade: deve oferecer adaptabilidade e escalabilidade, usar agrupamento de recursos e sobreviver a mudanças na carga de dados e variações nas condições da rede.

Fog Computing é especialmente importante para aplicações que requerem respostas em tempo real, como redes de veículos autônomos, casos de uso de cidades inteligentes, e IoT industrial. Além disso, o apoio à mobilidade e à distribuição geográfica garante que os dados possam ser processados e utilizados independentemente da localização. Ao mesmo tempo, a heterogeneidade e a interoperabilidade permitem uma integração contínua entre diversos dispositivos e sistemas. Essas características tornam a *Fog Computing* um complemento versátil e poderoso para a *Cloud Computing*, impulsionando a inovação em múltiplas indústrias.

A arquitetura mais comumente utilizada para representar um ambiente de *Fog Computing* compreende três camadas: Camada de IoT, Camada de *Fog* e Camada de *Cloud*, conforme apresentado na Figura 2.1. A Camada de IoT é composta por dispositivos de IoT conectados na borda da rede, por meio dos quais os usuários finais podem solicitar serviços a serem processados nas camadas superiores. Por exemplo, usuários podem solicitar uma lista de restaurantes cujo nível de ruído esteve, na média, abaixo de 70 dB nos últimos 15 minutos. A Camada de *Fog* é localizada entre as Camadas de IoT e *Cloud*. Essa camada fornece recursos compartilhados que aplicações de IoT podem usar conforme necessário, como recursos de processamento e armazenamento de dados, antes que os dados sejam transferidos para a *Cloud* [25]. Por fim, a Camada de *Cloud* compreende os serviços dos provedores de *Cloud*, com recursos computacionais mais robustos para oferecer processamento de alta ordem e armazenamento a longo prazo. A existência de *Cloud* é fundamental em um ambiente de *Fog* [25], porque *Fog Computing* complementa *Cloud Computing*, mas não a substitui.

Assim, um ambiente de *Fog Computing* que sustenta aplicações de IoT é caracterizado por ter uma organização mais distribuída, pela heterogeneidade de dispositivos físicos e de redes, e pela incerteza de conectividade causada pela mobilidade dos dispositivos, instabilidades da rede e exaustão da bateria [24]. Este cenário difere de um ambiente de *Cloud Computing*, sustentado por servidores homogêneos ricos em recursos, com fornecimento ininterrupto de energia e com conexões de rede redundantes e estáveis.

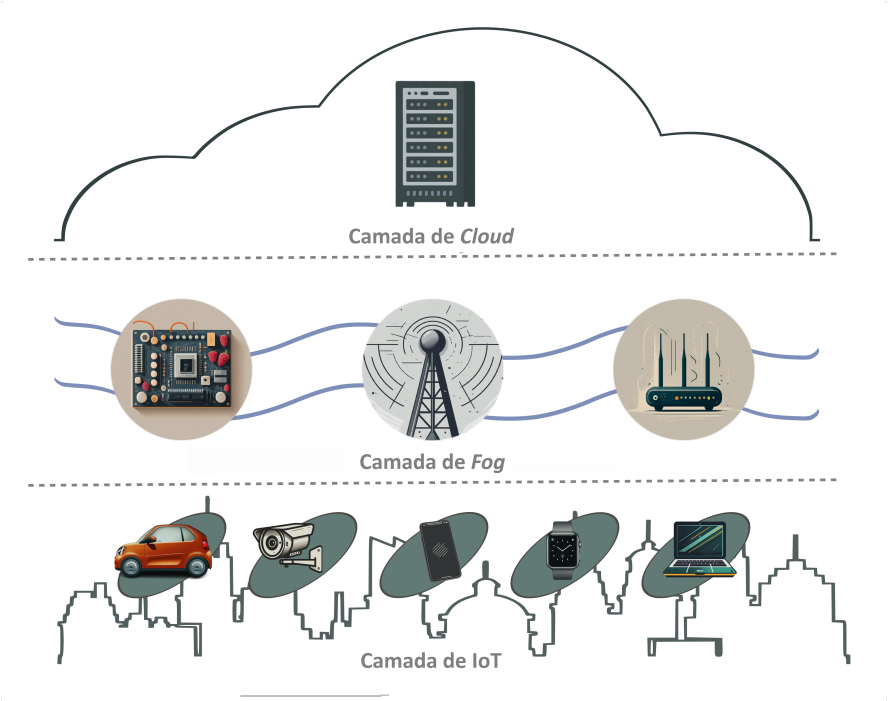


Figura 2.1: Visão geral de uma arquitetura de *Fog Computing*.

2.2.1 *Fog Node*

Pela perspectiva computacional, um recurso é qualquer componente físico ou virtual [26], por exemplo, CPU, memória, dados, dispositivos de rede, sistemas operacionais, sistemas de virtualização, etc. Especificamente em *Fog Computing*, é comum o uso do termo "*Fog Node*" [27] para descrever um componente da arquitetura, localizado na Camada *Fog*. Um *Fog Node* é qualquer dispositivo em um ambiente de *Fog Computing* que pode compartilhar recursos com dispositivos de IoT e outros *Fog Nodes*. Muitos dispositivos podem ser um *Fog Node*, incluindo *smartphones*, roteadores, *notebooks* e servidores especializados.

Ao contrário de outros paradigmas computacionais, como *Cloud Computing* ou *Grid Computing* [28] – em que os recursos computacionais possuem alta capacidade de processamento e de armazenamento, são mais lineares em termos de arquitetura e compatibilidade, e ainda se localizam em *datacenters* instalados em pontos específicos do mundo – os dispositivos que compõem a *Fog Computing*, em geral, possuem menor capacidade computacional, são mais heterogêneos [29], e estão amplamente distribuídos geograficamente [30].

Um *Fog Node* é composto por uma camada de hardware – na qual estão localizados os recursos físicos (por exemplo, CPU, memória, interface de rede, entre outros) – e também uma camada de sistema, que é necessária para a abstração de hardware e execução de aplicativos [31], conforme mostrado na Figura 2.2. Em uma perspectiva computacional, uma característica desejável de um *Fog Node* é a capacidade de virtualização [32], que não pode ser implementada por dispositivos que tenham baixo poder de processamento e baixa capacidade de comunicação. A capacidade de oferecer um ambiente virtualizado de execução de aplicações faz com que nem todos os dispositivos da Camada IoT sejam capazes de atuar como *Fog Nodes*.

A virtualização pode ser definida como "uma abstração de software com a aparência de um hardware de sistema de computador" [33]. Além disso, mecanismos de virtualização baseados em hardware estão disponíveis em quase todos os hardwares de processador que seriam usados para implementar plataformas de

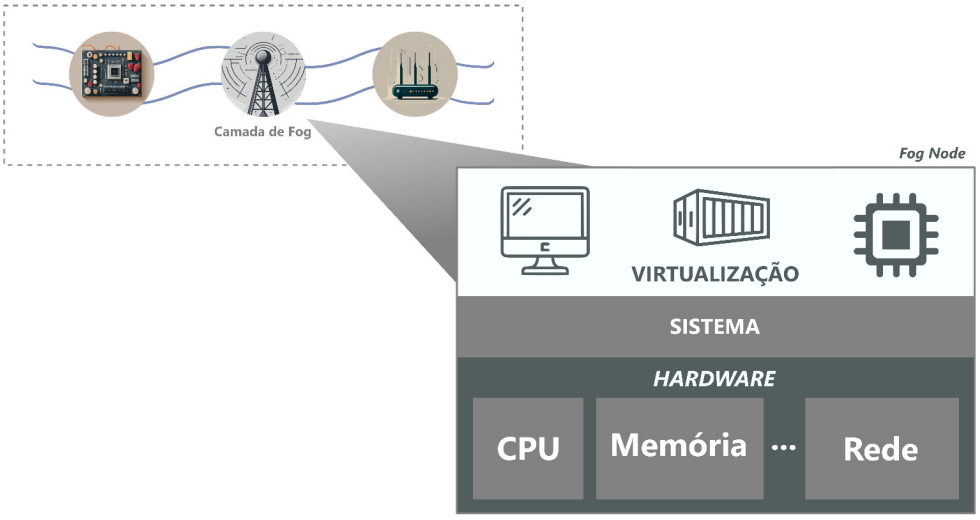


Figura 2.2: Abstração de recursos em um *Fog Node*.

Fog Computing [34]. Uma análise abrangente da perspectiva computacional de um *Fog Node* foi publicada pelo autor desta tese e coautores no artigo [35].

A forma mais comum de entrega de recursos virtuais é por meio de máquinas virtuais [33], frequentemente utilizadas em ambientes de *Cloud Computing*, pois permitem dividir recursos de uma máquina física e realocá-los em uma ou mais máquinas virtuais, que podem ser usadas em um número grande de tarefas. Mais recentemente, o método de virtualização migrou para o uso de *containers* por oferecerem um mecanismo de isolamento mais aderente a um ambiente de *Fog Computing*. Os *containers* realizam a virtualização na camada do sistema operacional e não mais na camada de hardware, como ocorre com as máquinas virtuais [30]. A virtualização baseada em *container* atua como sistemas operacionais *sandbox*, com um sistema operacional *host* sendo executado na camada inferior, e compartilhando seu *kernel* no modo somente leitura. Por fim, um aspecto relevante diz respeito à disponibilidade de recursos do dispositivo. Mesmo que um dispositivo tenha a capacidade de processamento e de comunicação necessárias, os seus recursos podem não estar disponíveis para executar serviços em um determinado momento.

2.3 Paradigmas Computacionais Relacionados

Além de *Fog Computing*, existem outros paradigmas baseados na proximidade de dispositivos do usuário, como *Edge Computing* [36], *Mobile Edge Computing* (MEC) [37], *Mist Computing* [38] e *Cloudlet Computing* [39], que são frequentemente confundidos com a *Fog*. Estes quatro paradigmas computacionais compartilham características distintas que os relacionam a *Fog Computing*. Todos buscam otimizar o processamento de dados e reduzir a latência, aproximando os recursos computacionais dos dispositivos de usuários finais e da origem dos dados. Essa proximidade reduz a necessidade de transferências demoradas para centros de dados centrais, economizando tempo e energia.

Assim como *Fog Computing*, cada um destes paradigmas esboça uma extensão do conceito de *Cloud Computing*, permitindo um processamento mais distribuído e uma resposta mais rápida a solicitações dos usuários. Enquanto *Edge Computing* e MEC lidam com a execução local e a otimização da latência na

borda da rede, *Mist Computing* foca na computação em dispositivos altamente localizados, usando recursos limitados. *Cloudlet Computing* pode ser entendido como uma "*Cloud* em miniatura", que se aproxima do poder de computação de *datacenters* tradicionais. No entanto, *Fog Computing* se destaca por oferecer uma integração em camadas, permitindo uma coordenação mais hierárquica e ampla entre diferentes pontos da rede.

Outros paradigmas computacionais têm características que os distanciam de *Fog*. Os paradigmas *Mobile Cloud Computing* (MCC) [40], *Mobile ad hoc Cloud Computing* (MACC) [41] e *Dew Computing* [42] diferem fundamentalmente em como abordam o uso de dispositivos móveis e a conectividade com a nuvem. MCC combina computação móvel e computação em nuvem para atender às demandas crescentes dos dispositivos conectados, exigindo uma infraestrutura confiável com baixa latência e alta largura de banda. Por outro lado, MACC oferece uma solução descentralizada através de uma rede móvel ad hoc que dispensa a necessidade de uma infraestrutura centralizada de *Cloud*, algo que o difere do MCC principalmente em situações de conectividade instável ou inexistente. Em comparação com *Fog Computing*, MACC se destaca por sua natureza mais descentralizada, permitindo a formação de redes dinâmicas em locais onde a conectividade é esporádica. *Dew Computing*, por sua vez, enfatiza a independência dos recursos computacionais locais, permitindo que funcionem sem conexão com a internet e proporcionando colaboração com serviços em *Cloud* quando a conectividade está presente. Isso difere tanto de MCC, que depende fortemente de uma infraestrutura de *Cloud* centralizada, quanto de MACC, que opera como uma rede *ad hoc* independentemente dos serviços de *Cloud*. Além disso, *Dew Computing* integra os conceitos de armazenamento e rede em sua plataforma, utilizando microsserviços para expandir a hierarquia computacional. Esse enfoque na independência e colaboração persegue o uso otimizado de recursos locais e serviços em *Cloud*, situando-se como um meio-termo entre a centralização do MCC e a descentralização extrema do MACC.

Tabela 2.1: Comparação entre *Fog Computing* e paradigmas relacionados.

Paradigmas	**Baixa Latência**	**Geo-Distribuição**	**Heterogeneidade**	**Alto Poder Computacional**	**Tempo Real**	**Escalabilidade**
Sky	-	-	✓	✓	-	✓
Cloud	-	-	✓	✓	-	✓
Fog	✓	✓	✓	-	✓	✓
Edge	✓	✓	✓	-	✓	✓
MEC	✓	✓	-	-	✓	-
MCC	-	-	✓	-	-	-
MACC	-	✓	-	-	-	-
Mist	-	✓	✓	-	✓	-
Cloudlet	✓	✓	-	✓	✓	✓
Dew	-	-	-	✓	✓	-

Com base nas principais características de *Fog Computing* descritas na Seção 2.2, uma comparação com os demais paradigmas mencionados é apresentada na Tabela 2.1. O objetivo é destacar as diferenças e as semelhanças entre os paradigmas e *Fog Computing*. Analisando a Tabela 2.1, percebe-se que *Edge Computing* é o paradigma mais similar a *Fog Computing*, compartilhando características como baixa latência, distribuição geográfica, heterogeneidade, interoperabilidade, tempo real e escalabilidade. Isso indica que soluções desenvolvidas para *Edge Computing* podem ser mais facilmente adaptadas para *Fog*, e vice-versa. Por outro lado, paradigmas como MCC, MACC e *Dew Computing* apresentam menor similaridade com a

Fog Computing, possuindo poucas características em comum, o que sugere que o uso, na Fog, de soluções projetadas para estes paradigmas demande adaptações complexas ou inviáveis.

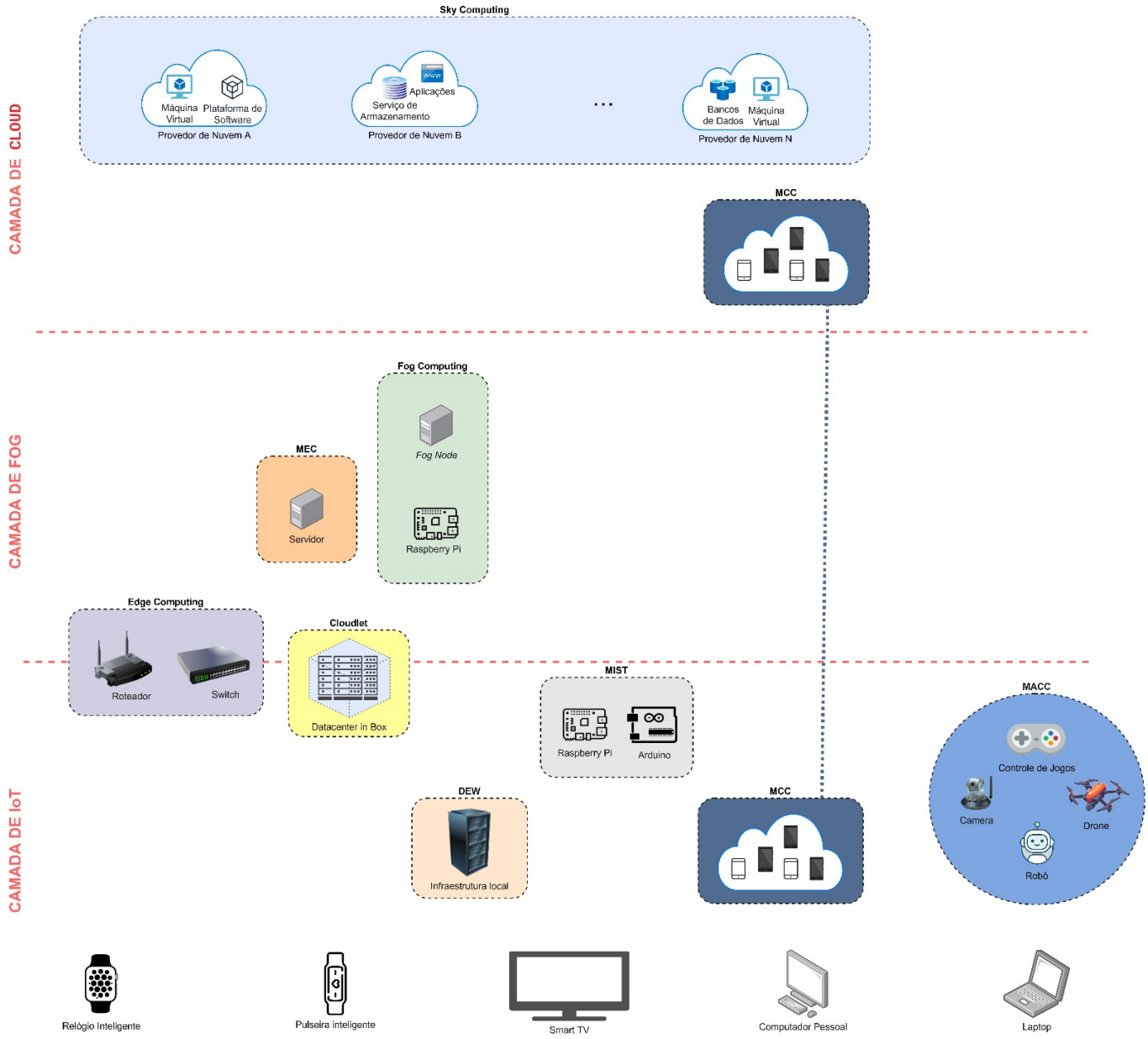


Figura 2.3: Posição relativa dos paradigmas de computação no *continuum* IoT-*Fog*-*Cloud*.

A Figura 2.3 complementa a análise comparativa entre os paradigmas ao apresentar o posicionamento relativo de cada paradigma no *continuum IoT-Fog-Cloud*. A *Fog Computing* se posiciona intermediariamente entre a IoT e a *Cloud*, atuando como uma ponte que estende os recursos da *Cloud* para mais perto dos usuários finais. Isso diferencia *Fog* de outros paradigmas como a *Cloud* que tem um escopo de atuação mais geral e, para alguns casos de uso, tais como soluções de cidades inteligentes e de monitoramento de pacientes, pode não apresentar as características mais adequadas para viabilizá-los, como uma maior latência. Um estudo mediu as latências reais em um cenário de rede veicular, utilizando a rede celular para comunicação com nós em *Fog* e em *Cloud* [43]. Os resultados mostraram que *Cloud* apresentou uma latência 56% maior que *Fog*, considerando um dos cenários avaliados. Nesse cenário, a latência média da comunicação do dispositivo IoT para *Cloud* mediu 125ms, enquanto que a latência para *Fog* mediu 80ms.

Realizar um comparativo entre paradigmas similares a *Fog* é importante, pois permite ampliar o escopo de pesquisa, uma vez que a adaptação de soluções entre paradigmas com características similares, tais como *Edge* e *MEC*, tende a ser mais viável. Por outro lado, reconhecer diferenças relevantes entre os paradigmas destaca as dificuldades em implantar em *Fog* soluções projetadas para paradigmas com características diferentes, tais como *Cloud*, MCC, MACC e *Dew*.

2.4 Orquestração em *Fog Computing*

A orquestração é um processo de gerenciamento e coordenação e é composta por diversas funcionalidades que interagem entre si e se complementam. Assim, a orquestração tem como objetivo gerenciar o ciclo de vida dos serviços que executam em *Fog Computing*, com mecanismos para garantir os SLAs estabelecidos. Um estudo foi realizado por meio de uma revisão sistemática da literatura e seus resultados foram publicados em [5], cuja primeira página está disponível no Anexo I. Além de apresentar a orquestração como área de pesquisa relevante na evolução de *Fog*, esta seção visa a evidenciar o papel fundamental que o monitoramento tem nesse contexto.

Ao longo do tempo, houve várias definições diferentes sobre o que constituía orquestração em diferentes áreas de pesquisa, tais como Sistemas Autônomos [44] em 1983, *Web Services* [45] em 2003, até *Fog Computing* [2] em 2012. Mais recentemente, alguns autores têm relacionado a orquestração não apenas com a gestão do ambiente, mas também com os resultados esperados, representados pelos SLAs [46, 47]. Considerando a diversidade de conceitos apresentada e por ainda não haver uma padronização adotada pela academia, o autor desta tese propôs a seguinte definição: "Orquestração em *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, ela deve monitorar a infraestrutura subjacente, reagir em tempo hábil às suas mudanças, e cumprir as regras de privacidade e de segurança" [5].

O método de Revisão Sistemática da Literatura (RSL) [48,49] sobre orquestração empregou as seguintes etapas: primeiramente, foram definidas as questões de pesquisa e realizada uma busca abrangente nas bases de artigos Scopus[1], Web of Science[2], ACM Digital Library[3] e IEEE XploreLibrary[4]. A *string* de busca utilizada foi "orchestrat* AND (fog OR edge OR mec)", incluindo "*Edge*" e "MEC", devido às similaridades entre os paradigmas, conforme analisado na Seção 2.3. A aplicação de critérios de inclusão e exclusão resultou na seleção inicial de 145 publicações a partir de um total de 1066. Finalmente, a leitura completa desses 145 trabalhos levou à seleção de 50 publicações para uma análise detalhada.

2.4.1 Funcionalidades de uma Arquitetura Genérica de Orquestração de *Fog*

Após analisar os artigos selecionados, foram extraídas a descrição da arquitetura e das funcionalidades contidas nas propostas de orquestração de *Fog*. Algumas funcionalidades são bem conhecidas e facilmente

[1] scopus.com
[2] webofknowledge.com
[3] dl.acm.org
[4] ieeexplore.ieee.org

reconhecíveis como parte de uma estrutura de orquestração (por exemplo, gerenciamento de recursos, que apareceu em 48 dos 50 artigos, e monitoramento, que apareceu em 40 dos 50 artigos) [5].

Para expor as abordagens e as soluções dadas pela literatura, definiu-se uma arquitetura genérica que as consolidou, conforme apresentado na Figura 2.4. Nessa figura, são identificadas as principais funcionalidades da orquestração que foram descritas nos artigos analisados e que são resumidas a seguir:

- **Controle de Admissão de Solicitações de Entrada** - a interface com o usuário final. Recebe as solicitações, avalia as credenciais do solicitante e decide onde será servido (*Fog*/*Cloud*).
- **Gestão de Serviços** - gerencia o ciclo de vida do serviço, ou seja, registro de serviço, imagens de serviço para diferentes plataformas de virtualização, restrições e requisitos de serviço. Utiliza dados de monitoramento para verificar a necessidade de ações visando garantir os SLAs.
- **Gerenciamento de Recursos** - gerencia o ciclo de vida dos recursos, ou seja, descobre novos *Fog Nodes*, aloca recursos para atender a solicitações aceitas; identifica a necessidade de descarregar serviços e desaloca recursos. Dados de monitoramento são responsáveis por manter o inventário de recursos atualizado.
- **Monitoramento** - atualiza o *status* de disponibilidade e uso de recursos e serviços, além de gerenciar *logs* e *traces*.
- **Otimização** - processa dados disponíveis usando algoritmos e técnicas para minimizar algumas métricas ou maximizar outras. Seus benefícios são a potencial redução de custos e de tempo de resposta, aumento no uso da CPU e da elasticidade.
- **Gerenciamento de Comunicação** - usa protocolos e padrões para lidar com a heterogeneidade de nós e *links* de comunicação de um ambiente *Fog*.
- **Agente de Nó** - um agente local que gerencia o ambiente de execução de um *Fog Node*, realiza as ações exigidas pelo gerenciador de serviços, por exemplo, replica um serviço, baixa uma nova imagem de serviço e coleta dados de monitoramento.
- **Segurança** - é responsável pela aplicação de políticas de segurança e de privacidade.

2.4.2 Discussão

A arquitetura genérica de orquestração de *Fog Computing* apresentada na Figura 2.4 consolidou todas as propostas analisadas. Ela apresenta as funcionalidades de orquestração abordadas, e também as escolhas feitas pelos autores para implementá-las, sendo útil para pesquisadores que desejam conhecer melhor o estado da arte sobre o assunto.

Como área de pesquisa recente, as propostas têm buscado primeiramente agregar, de forma estruturada e coordenada, as diversas funcionalidades relacionadas à orquestração em *Fog*, focando nas interações entre elas e considerando comunicações entre as camadas. Assim, o gerenciamento e o monitoramento de recursos foram as funcionalidades mais implementadas pelas propostas analisadas, conforme o estudo publicado em [5]. Essas funcionalidades estão diretamente relacionadas ao ambiente de execução dos serviços, e são

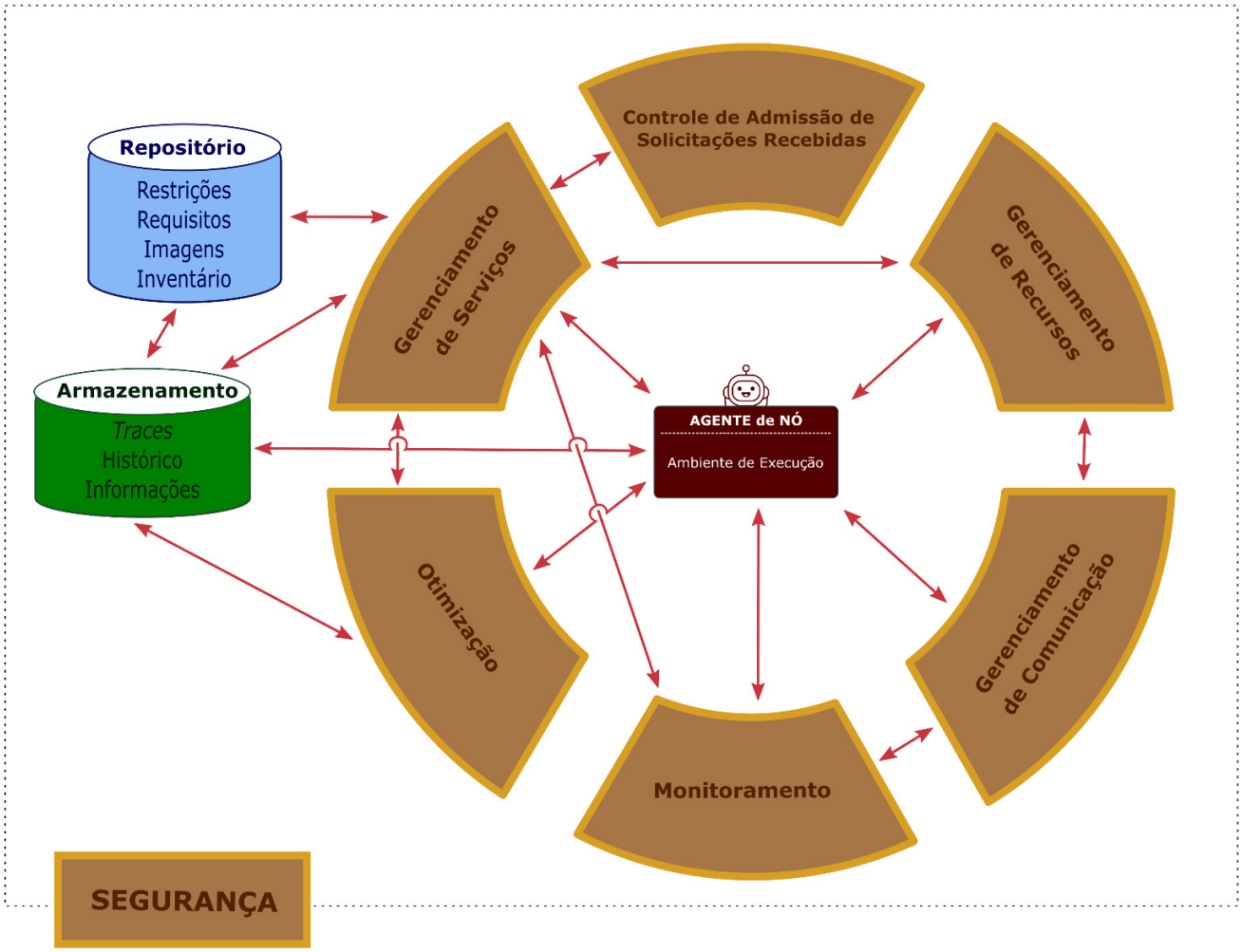


Figura 2.4: Uma arquitetura genérica de orquestração de *Fog Computing*.

fundamentais para a implementação de algumas das características essenciais da *Fog*, mencionadas na Seção 2.2. Algumas dessas características são: baixa latência (obtida por meio do fornecimento de recursos próximos ao usuário, medido e acompanhado por monitoramento), interações em tempo real e escalabilidade, que é alavancada pelo monitoramento de eventos em tempo hábil, indicando a necessidade de alteração na alocação de recursos.

Bonomi *et al.* [50] propuseram uma arquitetura de execução de serviços em *Fog*. Esta arquitetura é apresentada na Figura 2.5. Ela mostra um processo de orquestração de *Fog*, estruturado como um *loop* de controle Monitorar-Analisar-Planejar-Executar (MAPE) [51], que é responsável por fornecer gerenciamento de ciclo de vida de serviços de *Fog* de maneira distribuída. Esse *loop* de controle aplicado à *Fog Computing*, será explicado a partir da fase Monitorar. Nesta fase, a orquestração deve coletar o *status* atualizado sobre cada recurso gerenciado e serviços em execução. A partir da análise dos dados de monitoramento, pode-se construir uma visão atualizada e abrangente do ambiente de *Fog*, e planejar as mudanças necessárias para manter os serviços dentro dos limites de SLAs e QoS, além de fornecer novos serviços solicitados. A execução dessas mudanças planejadas liberará e alocará recursos adequados, fornecendo serviços próximos aos usuários finais.

A orquestração de serviços tem recebido bastante atenção da academia pela capacidade de mitigar os

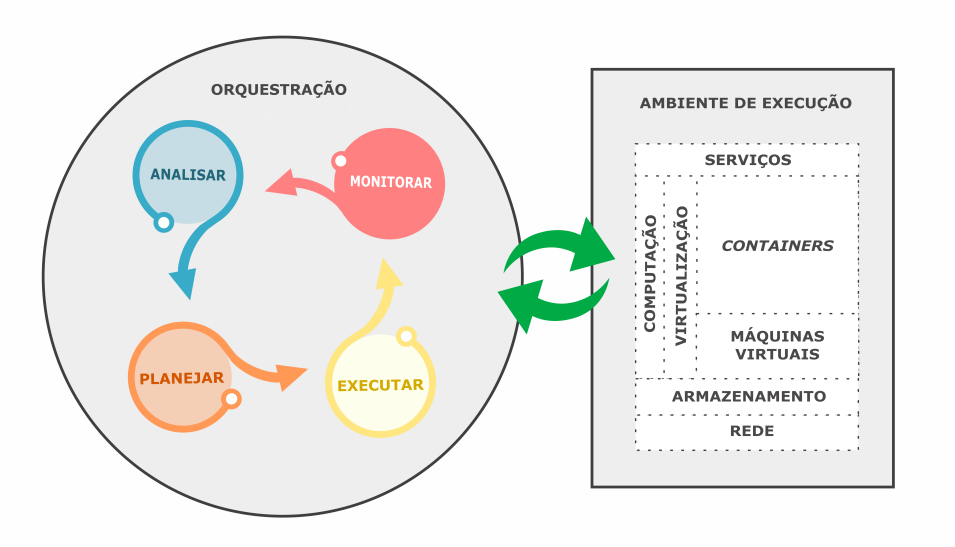


Figura 2.5: Processo de orquestração *Fog*.

riscos decorrentes das características de *Fog Computing*, tais como heterogeneidade dos dispositivos, mobilidade, alta distribuição geográfica, e a necessidade de se comunicar com baixa latência. A orquestração é formada por várias funcionalidades que se complementam. Como a *Fog Computing* pode viabilizar diversos casos de uso diferentes – por exemplo, a coleta de informações ambientais, redes veiculares, cidades inteligentes, entre outros –, é interessante que os algoritmos, técnicas e funcionalidades disponíveis sejam parametrizáveis, de forma a dar flexibilidade na sua aplicação, de acordo com a necessidade específica. Por exemplo, em dispositivos com baixa disponibilidade de espaço de armazenamento local, o envio das informações coletadas deve ser mais frequente, quando comparado a dispositivos com maior disponibilidade de recursos.

Como o ambiente é dinâmico, a disponibilidade de recursos pode se modificar drasticamente em curto espaço de tempo, exigindo ações rápidas do orquestrador de forma a garantir os SLAs. O monitoramento de recursos, nesse contexto, é de grande importância, pois visa a disponibilizar para o orquestrador as informações de *status* dos recursos e dos serviços, mantendo-as atualizadas, permitindo uma análise acurada da situação, e uma tomada de decisão tempestiva. O monitoramento foi a segunda funcionalidade mais citada entre as propostas de orquestração analisadas por este trabalho. Ele parece ser fundamental para que os desafios da orquestração, apresentados nos parágrafos anteriores, sejam superados. Mas a maioria das propostas de orquestração apenas assume que um serviço de monitoramento exista, sem que sejam disponibilizadas informações suficientes acerca da implementação e da avaliação no contexto da orquestração de *Fog Computing*.

Nos últimos anos, a literatura sobre sistemas distribuídos tem feito uso do termo “Observabilidade” para simbolizar um avanço na noção de monitoramento como serviço ou funcionalidade. Nesse contexto, a observabilidade é definida como uma característica de softwares e sistemas relacionada às informações que geram e que permitem monitorá-los e compreendê-los de forma mais abrangente, inclusive em tempo de execução [1]. Portanto, a observabilidade é uma característica inerente aos serviços e sistemas, e não uma ação ou um processo. Assim sendo, a expressão “Gestão da Observabilidade” será adotada nesta tese para definir o que previamente se conhecia como monitoramento, já que a Gestão da Observabilidade

abrange toda a prática de monitoramento, além de administrar categorias adicionais de informação. O uso do termo "monitorar" como ação ou tarefa (como, por exemplo, "monitorar *Fog Nodes*") continua com seu significado claro neste contexto, e por esse motivo, este capítulo mantém a terminologia tradicional. Por outro lado, nos próximos capítulos, os sistemas que implementam as funções necessárias para monitorar recursos serão abordados como "Sistemas de Gestão da Observabilidade".

2.5 Considerações Finais

Este capítulo apresentou o paradigma *Fog Computing* como uma extensão de *Cloud Computing*, que leva os recursos computacionais para mais perto dos usuários na borda da rede. Foi explorada a arquitetura de *Fog Computing*, seus componentes, os *Fog Nodes*, e as diferenças entre *Fog* e outros paradigmas computacionais. Realizar um comparativo entre paradigmas similares a *Fog* é importante, pois permite ampliar o escopo de pesquisa, uma vez que a adaptação de soluções entre paradigmas com características similares, tais como *Edge* e MEC, tende a ser mais viável. Por outro lado, reconhecer diferenças relevantes entre os paradigmas destaca as dificuldades em implantar em *Fog* soluções projetadas para paradigmas com características diferentes, tais como *Cloud*, MCC, MACC e *Dew*.

Neste capítulo, analisou-se também a orquestração de serviços em *Fog Computing*. Para isso, foram abordadas as funcionalidades responsáveis pela integração eficaz de serviços descentralizados que operam em *Fog Nodes*, possibilitando a otimização de recursos e a redução da latência.

No próximo capítulo, será apresentado um caso de uso de cidades inteligentes, que servirá como cenário motivador para a utilização e avaliação das propostas desta tese. Além disso, será dado foco à gestão da observabilidade em *Fog*, uma das funcionalidades da orquestração mais presentes nas propostas de orquestração avaliadas, e a menos estudada neste contexto [5].

Capítulo 3

Gestão da Observabilidade em *Fog Computing*

Este capítulo utiliza o termo "Gestão da Observabilidade" no lugar de "Monitoramento", conforme explicado na Seção 2.4.2. Aqui são apresentadas as principais características e funcionalidades (subprocessos, componentes do sistema, domínios de instrumentação) de um sistema de gestão da observabilidade, reunindo conhecimento do estado da arte na área de *Fog Computing*. Ele também fornece uma análise do papel da gestão da observabilidade no contexto da orquestração de serviços em *Fog*, apresentando os requisitos e desafios. O capítulo apresenta, ainda, um cenário motivador com o intuito de auxiliar o entendimento da aplicação dos conceitos e dos experimentos definidos na tese em exemplos práticos.

Para isso, a estrutura deste capítulo está organizada em oito seções. Na Seção 3.1, a observabilidade é abordada inicialmente como um conceito que amplia o monitoramento, proporcionando uma visão abrangente do desempenho dos serviços. O cenário motivador desta tese é descrito na Seção 3.2. A Seção 3.3 detalha as funções de um sistema de gestão da observabilidade, ou seja, observação, processamento de dados e exposição dos dados coletados. Na sequência, a Seção 3.4 apresenta os três principais domínios de instrumentação, os quais são métricas, *logs* e *traces*, cada um com suas próprias características. Na Seção 3.5, os componentes do sistema de gestão da observabilidade são apresentados. Os requisitos e desafios da gestão da observabilidade em *Fog* são discutidos na seção 3.6, destacando a necessidade de novas soluções adaptadas a este contexto. Na Seção 3.7, a gestão da observabilidade é descrita como uma função essencial na orquestração de serviços em *Fog*, integrando dados de instrumentação para apoio a decisões sobre o gerenciamento do ambiente. Por fim, a Seção 3.8 apresenta um resumo sobre os principais conceitos deste capítulo.

3.1 Observabilidade

Observabilidade é um termo emprestado da teoria de controle [52]. Na Ciência da Computação, é definida como uma característica, de software e de sistemas, relacionada às informações que geram e que permitem monitorá-los e compreendê-los de forma mais abrangente, inclusive em tempo de execução [1]. Quanto mais alto o nível de observabilidade, mais fácil será conhecer os comportamentos atuais e passados do sistema. Esse conhecimento, a partir do momento em que está disponível, permite uma atuação adequada sobre o sistema quando necessário, garantindo maior disponibilidade.

Além de um monitoramento simples, a gestão da observabilidade pode fornecer uma maior compreensão do bom funcionamento e do desempenho dos serviços. Dessa forma, um de seus objetivos é diminuir o tempo necessário para saber por que algo não está funcionando como deveria. É um processo inerentemente intensivo em dados e sensível ao tempo [1]. A gestão da observabilidade em sistemas distribuídos é um conceito emergente que tem sido usado para fazer referência a funções avançadas de monitoramento no contexto de aplicativos baseados em microsserviços. Ela é considerada um superconjunto do monitoramento, pois abrange diferentes visões do objeto monitorado, representadas por diferentes conjuntos de dados, que precisam ser incorporados ao sistema e gerenciados. A gestão da observabilidade faz uso de técnicas de análise de dados para apoio à tomada de decisão [53].

Neste trabalho, será usado o termo "gestão da observabilidade" para denominar o macro processo de coletar informações (métricas, *logs* e *traces*) de objetos monitorados (hardware, ambientes virtualizados e serviços), transmitir as informações pelas camadas da arquitetura *Fog*, tomar decisões tempestivas e oportunas com base nas informações coletadas, e armazená-las para análise posterior ou para retenção de longo prazo.

3.2 Cenário Motivador: Cidades Inteligentes

Uma aplicação de cidade inteligente é um software desenvolvido para ajudar a gerenciar e melhorar a qualidade de vida em uma cidade. Essas aplicações aproveitam tecnologias avançadas, como sensores, dispositivos IoT, redes 5G e Inteligência Artificial, para coletar e analisar dados em tempo real sobre diferentes aspectos da cidade, incluindo segurança pública, tráfego, consumo de energia, qualidade do ar, etc [54]. Mobile IoT-RoadBot [55] é uma aplicação de cidade inteligente que monitora e detecta problemas de manutenção de ativos de trânsito à beira da estrada. Além disso, é usada como um ambiente de teste do mundo real para avaliar o desempenho de rede 5G. Ela foi implantada em caminhões de serviço de coleta de lixo no conselho da cidade de Brimbank, localizado na área metropolitana de Melbourne, Austrália. Este ambiente de teste forneceu um cenário ideal de implantação no mundo real para avaliar o desempenho da rede 5G devido à mobilidade natural dos caminhões. Ele permitiu que o desempenho de uma rede 5G fosse testado usando uma aplicação que transmitia uma grande quantidade de dados de vídeo em tempo real, em uma ampla área geográfica.

A arquitetura do Mobile IoT-RoadBot é apresentada na Figura 3.1, e descrita da seguinte forma. O Mobile IoT-RoadBot foi desenvolvido como um sistema de cinco camadas, onde cada camada é responsável por tarefas específicas de dados, tais como produção, ingestão, análise, armazenamento e visualização. Os 11 caminhões de serviço de coleta de resíduos foram equipados com dispositivos IoT, incluindo câmeras estereoscópicas com sensor de profundidade, roteadores 5G, antenas dome 5G, computadores de borda e Sistemas de Navegação por Satélite Global (GNSS). Após a implantação desses dispositivos IoT, os caminhões são referidos como "*SmartTrucks*". O processo de implantação é apresentado na Figura 3.1. A câmera e a antena foram montadas na barra frontal de cada *SmartTruck*. O roteador e o computador de borda foram colocados na cabine do *SmartTruck*. Os dispositivos são alimentados quando o caminhão é ligado.

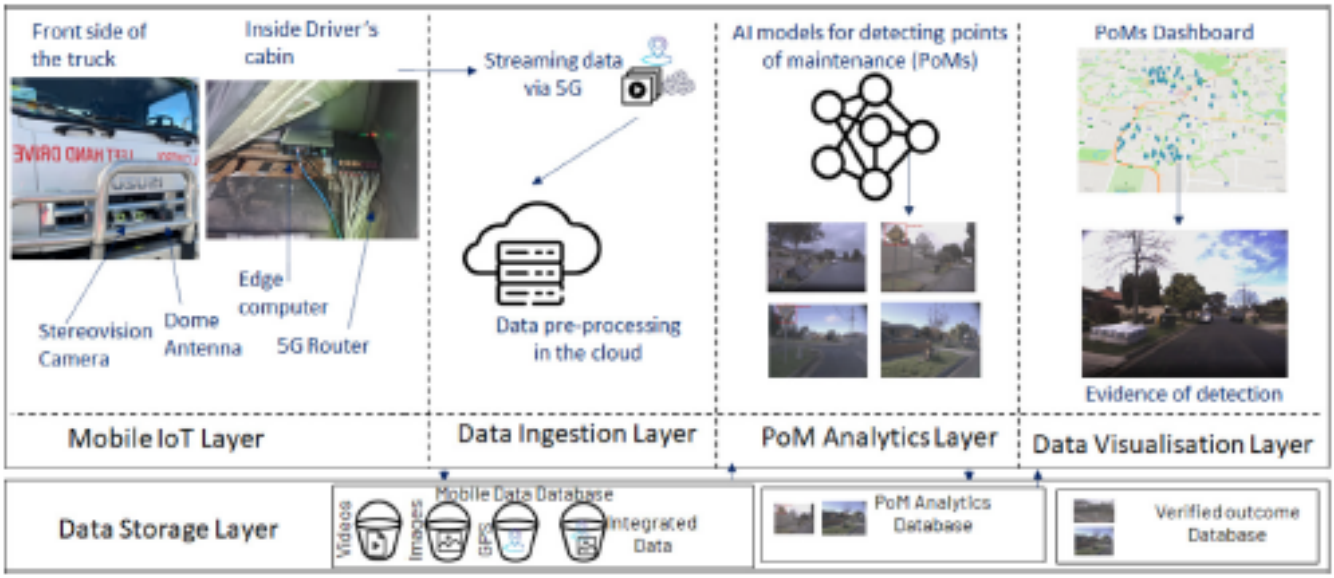


Figura 3.1: A arquitetura do Mobile IoT-RoadBot.

A câmera de estereovisão instalada é capaz de produzir vídeos a 32 quadros por segundo (em formato RGB), bem como RGB-D e nuvens de pontos 3D. RGB-D se refere a *Red Green Blue-Depth* e fornece informações de profundidade associadas aos dados RGB correspondentes. Uma nuvem de pontos é um conjunto de pontos de dados no espaço que representa uma forma ou objeto 3D. O dispositivo GNSS fornece localização em tempo real (ou seja, longitude e latitude), bem como informações de velocidade dos *SmartTrucks*. Quando um *SmartTruck* está ligado e na estrada, ele produz dados em tempo real capturados por sua câmera e GNSS, os quais são referidos como o *Mobile IoT-Layer*. Nesta camada, a câmera e outros dispositivos de IoT estão conectados ao computador de borda. O software instalado no computador de borda é capaz de capturar e gravar vídeo, dados de GNSS e medições de desempenho de rede 5G. Durante as rondas de coleta de lixo, à medida que os *SmartTrucks* se movem pela cidade, o vídeo gravado e as informações de GNSS são consolidados e transmitidos para o *Data Ingestion Layer* executando em um sistema de IA baseado em *Cloud* por meio da rede 5G. O processo de transmissão de dados 5G é configurado por meio de um roteador 5G e antena Dome. Os dados de vídeo são enviados em pequenos pedaços para evitar a perda de dados. Para avaliar o desempenho de *upload* em 5G em diferentes cargas de dados, os dados são enviados em diferentes formatos, incluindo vídeo comprimido e nuvens de pontos 3D. Os dados relacionados à avaliação de desempenho 5G, juntamente com os dados de GNSS, são armazenados em *logs* no dispositivo de borda para análise *offline*.

A *Data Ingestion Layer* é responsável por receber dados de *streaming* dos *SmartTrucks* na *Cloud*, usando AWS como provedor de *Cloud*. O software desenvolvido opera nesta camada para ler e pré-processar o *streaming*, preparando-o em um formato adequado para análise. Tanto os dados brutos quanto os dados pré-processados são armazenados na *Data Storage Layer*, que serve como dados históricos para análise. O *PoMs Analytics Layer* automatiza a análise dos dados pré-processados usando modelos de Deep Learning (DL) baseados em IA. Esses modelos são executados periodicamente para identificar objetos na beira da estrada que requerem manutenção, que são referidos como *Points of Maintenance* (*PoMs*) (por exemplo, lixo de grande volume atrapalhando o caminho dos pedestres, placas de sinalização danificadas

e abrigos de ônibus depredados). Os PoMs identificados também são armazenados no *Data Storage Layer* para fins de consulta e visualização. A *Data Visualisation Layer* apresenta os PoMs para os usuários por meio de uma interface baseada em mapa. A interface do mapa é um painel de aplicação web projetado para ser usado pela Equipe de Gestão de Ativos e pelas Equipes de Manutenção do Conselho Municipal de Brimbank para inspecionar e revisar os ativos na beira da estrada identificados pela *PoMs Analytics Layer*.

Os *SmartTrucks* usam tecnologia 5G, mas a disponibilidade de redes 5G não é ubíqua na região. Portanto, às vezes não haverá conexão disponível para enviar dados ao servidor, e os vídeos devem ser armazenados para transmissão posterior, assim que a rede se tornar disponível. Além disso, existem outros problemas que podem causar interrupção no fluxo de dados entre os *SmartTrucks* e os servidores, como falta de espaço para armazenar vídeos, mau funcionamento da câmera e falha da aplicação. Nesses casos, a demora na identificação de tais problemas pode causar perda de dados, desperdício de dinheiro e serviço de baixa qualidade para o cidadão.

Dessa forma, seria benéfico para uma aplicação de IoT de cidade inteligente se tais questões fossem identificadas rapidamente e automaticamente. Uma solução poderia ser entregue mais rapidamente e o sistema poderia retornar ao seu estado normal, potencialmente atuando autonomamente [56]. Por exemplo, após identificar que a aplicação de coleta de imagens do *SmartTruck* está travada, ou seja, recursos de hardware e rede estão disponíveis e operacionais, mas a aplicação não está coletando imagens da estrada nem as transmitindo, poderia ser enviado um comando remoto para reiniciar a aplicação e verificar, após um curto período de tempo, se alguma atuação adicional ainda é necessária.

Além de agir após a identificação de um problema, é possível agir preventivamente. Se houver mais conhecimento sobre cenários nos quais os problemas são prováveis de ocorrer, a atuação pode ser preventiva, por exemplo, agindo para liberar espaço de armazenamento no computador de bordo do *SmartTruck* quando a ocupação atinge um limite pré-determinado, tal como 90%. Para saber mais sobre os cenários, dados históricos e modelos de Aprendizado de Máquina podem ser usados para entender correlações entre métricas, prevê-las, etc. Nesses casos, o servidor pode regularmente definir novos limites para a aplicação, após executar modelos de Aprendizado de Máquina nos dados históricos.

Assim, para fornecer uma visão abrangente e atualizada de todo o sistema, é necessário aumentar sua observabilidade, coletando prontamente o *status* de cada componente (sensores, hardware, rede, módulos de aplicação). Esses dados podem ser processados para fornecer gatilhos para atuação, de acordo com limites e regras predefinidos. Além disso, os dados estarão disponíveis para análise histórica.

3.3 Funções de um Sistema de Gestão da Observabilidade em *Fog*

Considerando as características de *Fog*, os sistemas em execução neste ambiente terão uma alta distribuição de seus componentes, carga variável e imprevisível, ocasionada pela heterogeneidade de dispositivos, dos *links* de comunicação e das falhas. Esse cenário torna difícil prever como esses sistemas se comportarão ao longo do tempo [57]. Dessa forma, monitorar a infraestrutura é a base para atingir vários objetivos, tais como: fazer uso eficiente de recursos, medir o desempenho de recursos e serviços, gerar faturas precisas [58], e implementar processos de tolerância a falhas.

Logo, um serviço de gestão da observabilidade pode ser estruturado como uma composição de três funções distintas, as quais são: observação dos recursos e serviços monitorados; processamento de dados; e exposição de dados [7, 59]. A observação significa a aquisição de *status* atualizados de uso de recursos (por exemplo, carga da CPU e latência da rede) ou desempenho do serviço (por exemplo, tempo de resposta). O processamento está relacionado aos ajustes necessários e à transformação exigida nos dados, como filtragem e agregação, criação e gerenciamento de eventos e notificações derivadas de regras e limites pré-configurados. A exposição está relacionada ao local onde os dados coletados são armazenados (por exemplo, em um banco de dados local, arquivos JSON) e como podem ser acessados por um sistema de gerenciamento (por exemplo, visualização por meio de painéis de dados, outras funcionalidades consumindo os dados diretamente).

3.4 Domínios de Instrumentação

A observabilidade pode ser instrumentada em um sistema gerando saídas que informam o estado interno do sistema em momentos específicos. Os diferentes tipos de dados que compõem a saída são chamados de Domínios de Instrumentação (ID, do inglês *Instrumentation Domain*). Cada domínio de instrumentação contribui para a observabilidade de um sistema, oferecendo uma perspectiva diferente sobre o sistema. Há um consenso na literatura de que os domínios de instrumentação mais importantes da observabilidade são métricas, *logs* e *traces* [1, 60, 61], que serão detalhados a seguir. Alguns autores também consideram outros domínios de instrumentação, tais como eventos [1], *profiles* [60, 61] e *crash dumps* [60], embora eles não sejam unanimidade entre os pesquisadores. Com o passar do tempo, alguns desses novos domínios podem ser padronizados e incorporados ao contexto da observabilidade.

Assim, define-se $ID = \{ID_1, ID_2, ID_3 ... ID_n\}$ como o conjunto de domínios de instrumentação que são gerados por um sistema. Ter mais domínios de instrumentação disponíveis significa um nível mais alto de observabilidade. Dessa forma, observabilidade está diretamente relacionada à cardinalidade de ID ($|ID|$). Por exemplo, uma solução de gestão da observabilidade de *Fog* que gerencia apenas métricas tem menor nível de observabilidade do que uma que gerencia métricas, *logs* e *traces* simultaneamente.

3.4.1 Métricas

Métricas estão mais relacionadas ao desempenho e capacidade de um componente de hardware ou de software. Elas são valores numéricos coletados em um ponto no tempo, e sua coleta pode ser caracterizada como uma série temporal. No cenário motivador, as seguintes métricas estão disponíveis: porcentagem de uso da CPU, velocidade de um *SmartTruck* em km/h, *throughput* da rede 5G em Mbps, quantidade de dados de vídeo enviados para a *Cloud* em MB, etc. Métricas diferentes podem levar a atuações diferentes. Um *throughput* de rede 5G que ficou abaixo de um determinado limiar, pode sinalizar que transferências de dados diferentes daquelas que enviam o vídeo coletado pelos *SmartTrucks* devem ser adiadas para outro momento em que o *throughput* estiver mais alto.

As métricas podem ser coletadas de diversas fontes, como sensores, dispositivos IoT, sistemas operacionais e aplicações. A escolha das métricas a serem coletadas deve ser baseada nos objetivos de monitoramento, que devem refletir as necessidades de outros sistemas e serviços que utilizarão os dados coletados.

É importante considerar o impacto da coleta de métricas no desempenho do sistema que está sendo monitorado, evitando a coleta de métricas desnecessárias ou redundantes. Além disso, deve-se considerar a possibilidade de agregação e sumarização de métricas, como uma alternativa para facilitar a análise e a visualização dos dados.

3.4.2 *Logs*

Logs são arquivos de texto não estruturados ou semi-estruturados, relatando eventos relevantes e informações contextuais, cuja instrumentação geralmente é feita no momento do desenvolvimento. Por exemplo, optou-se por registrar em *log* as transações bem-sucedidas, além de detalhes referentes a erros. No cenário motivador, encontram-se nos *logs* informações pertinentes à qualidade do serviço na conexão de rede 5G, bem como as coordenadas geográficas do *SmartTruck*, entre outros dados, todos associados ao momento exato em que foram documentados. Ao analisar os *logs* de rede 5G dos *SmartTrucks* ao longo de um período de tempo, é possível determinar os locais onde a largura de banda da rede é baixa em muitos momentos do dia. Esse dado pode levar a uma investigação da cobertura da rede 5G pela operadora da rede.

Os *logs* podem ser gerados por diversos componentes do sistema, como aplicações, sistemas operacionais e dispositivos de rede. A análise de *logs* pode ser usada para identificar problemas de desempenho, erros de aplicação e falhas de segurança. É importante estruturar os *logs* de forma a facilitar a análise e a correlação dos dados. Além disso, a retenção dos *logs* deve ser definida com base nas necessidades de auditoria e conformidade. As informações relevantes extraídas dos *logs* ajudam as equipes de manutenção a melhorar o tratamento de erros e restaurar o sistema a um estado operacional.

3.4.3 *Traces*

Traces são registros das chamadas de serviço feitas pelo sistema. Eles permitem observações da sequência de chamadas de serviço, e o tempo gasto em cada chamada desde o início até o fim de uma solicitação. A análise de *traces* pode mostrar quais chamadas de serviço estão demorando mais na composição do tempo de resposta de uma aplicação. Eles também podem mostrar solicitações que não terminam corretamente. A solução no primeiro caso pode ser uma otimização de código entregue como uma nova versão da aplicação. No último caso, uma melhor gestão de erros pode tornar a aplicação mais resiliente ao processamento incompleto de solicitações. No cenário motivador, as informações de desempenho da aplicação (taxa de *upload* de vídeo) são relatadas de forma agregada, por subúrbio, para a Prefeitura de Brimbank. A agregação por subúrbio levou muito tempo para processar devido ao alto volume (2,5 milhões de medições por semana). Após otimizar o código usando a abordagem de ponto dentro do polígono [62] em vez de força bruta, o tempo gasto nessa agregação de dados foi reduzido a 1% do tempo original.

Os *traces* podem ser usados para monitorar o fluxo de requisições entre os diferentes componentes de um sistema distribuído. A análise de *traces* pode ajudar a identificar gargalos de desempenho, erros de comunicação e dependências entre os serviços. É importante instrumentar as aplicações para gerar *traces* detalhados e relevantes. Além disso, a visualização dos *traces* pode facilitar a compreensão do comportamento do sistema e a identificação de problemas.

Tabela 3.1: Características de dados dos domínios da observabilidade [1].

Domínio	**Tipo**	**Consulta**	**Armazenamento**
Métrica	Numérico	Agregações	BDs de Séries Temporais
Log	*Strings* semi/não estruturadas	Pesquisa aproximada de *strings*	Índice Invertido
Trace	DAGs de duração da execução	Pesquisa em grafo dissociado	Índice Invertido

3.4.4 Análise Comparativa dos Domínios de Instrumentação

Métricas, *logs* e *traces* carregam diferentes tipos de informações, como pode ser visto na Tabela 3.1. A tabela apresenta ainda qual o tipo de consulta de dados mais comum para cada domínio, assim como o tipo de armazenamento apropriado para oferecer uso de espaço otimizado e baixo tempo de resposta para as consultas realizadas nos dados armazenados. Cada um destes domínios de instrumentação contribui independentemente para aumentar a observabilidade de um sistema, permitindo uma atuação complementar.

Métricas fornecem informações objetivas sobre a interface externa de um sistema, por exemplo, a taxa de *upload* de vídeo. Elas permitem uma tomada de decisão rápida em resposta a medições que estão fora de um intervalo regular especificado. A frequência de coleta de métricas tem correlação com a velocidade da tomada de decisão, criando um *trade-off* entre a tempestividade da informação e o *overhead* gerado pelas tarefas de coleta, processamento e transmissão de dados. Como dizem respeito à interface externa do sistema, métricas podem ser coletadas por uma biblioteca ou agente externo ao sistema, implantado no mesmo dispositivo ou acessado pela rede. O volume de dados é geralmente baixo e constante, e diretamente proporcional ao número de métricas coletadas multiplicado pela frequência de coleta. As métricas são comumente utilizadas de duas maneiras, as quais são: (i). gerando alertas; e (ii). permitindo análise de dados e visualização em painéis.

Logs geralmente fornecem informações internas sobre eventos de falha, como mensagens de erro específicas, mensagens de tratamento de exceção, e erros em tempo de execução. Eles podem fornecer as informações necessárias para acelerar uma análise de causa raiz, ajudando a equipe de manutenção a melhorar o tratamento de erros e a retornar o sistema a um estado saudável. Entretanto, as informações relevantes disponíveis nos *logs* precisam ser extraídas para distinguir dados úteis de irrelevantes. Geralmente, uma análise de *log* eficaz pode precisar avaliar *logs* escritos algumas horas, ou até dias, antes do evento anômalo sendo analisado.

Traces fornecem detalhes sobre o fluxo interno de informações e de controle, incluindo a sequência e a demora de cada chamada de serviço necessária para processar uma requisição. Esses dados podem ser visualizados como um grafo, e um caminho crítico pode ser gerado a partir dele, permitindo a análise da dependência entre os componentes de um sistema [63]. O volume de dados por período de tempo depende da quantidade de requisições e pode ser esporádico, alcançando seu pico no mesmo momento em que o sistema está experimentando uma demanda máxima.

Usando terminologia criada no domínio de teste de software, a gestão da observabilidade pode ser dividida em *black-box* e *white-box* [64]. *Black-box* é a coleta feita a partir da interface pública do objeto monitorado, e tem como objetivo responder se o objeto está disponível/funcionando, ou seja, identifica se há algum problema. *White-box* baseia-se na coleta de informação detalhada sobre o funcionamento

dos processos internos do objeto, e tem como objetivo responder porque o objeto não está disponível ou funcionando a contento, isto é, permite uma análise de causa raiz [65]. As métricas estão mais relacionadas à gestão da observabilidade *black-box*, enquanto *logs* e *traces* estão mais relacionados à *white-box*.

Existem dezenas de métricas que podem ser coletadas a partir do objeto monitorado, independente de ele ser um dispositivo físico ou virtual (por exemplo, percentual de uso de CPU, percentual de memória livre, etc.), um sistema de gerenciamento de *containers* como Kubernetes [66] (número de *containers*, número de solicitações por *container* [67]), ou um (micro)serviço executado em um *Fog Node* (por exemplo, tempo de resposta, etc.). A literatura define os sinais de ouro da gestão da observabilidade, que compõem um conjunto mínimo de dados de telemetria (ou seja, dados de instrumentação) que fornecem informações essenciais para monitorar adequadamente a infraestrutura, as plataformas de execução e os serviços. Nesse contexto, esses sinais de ouro da gestão da observabilidade são: latência, tráfego, erros e saturação [65].

A escolha de quais métricas devem ser coletadas para melhor representar o *status* de um ativo é uma questão desafiadora no contexto de gestão da observabilidade em *Fog Computing*. Aumentar o número de métricas pode significar mais *overhead* na coleta e mais dados para transmitir, analisar e armazenar. Por outro lado, informações valiosas devem fluir no processo de gestão da observabilidade, permitindo a tomada de decisão adequada, e auxiliando outras funcionalidades da orquestração a atingirem seus objetivos. Assim, deve haver flexibilidade na seleção do conjunto de métricas que são suficientes para informar com precisão o *status* do objeto monitorado e tomar decisões oportunas, de acordo com os requisitos da orquestração de *Fog Computing*.

3.5 Componentes do Sistema de Gestão da Observabilidade

O objeto monitorado precisa fornecer acesso aos seus valores de métrica. Isso, geralmente, é implementado por meio de chamadas do sistema operacional (*System Calls*) feitas por um agente de gestão da observabilidade local que coleta os dados em um período recorrente predefinido, e os disponibiliza para processamento. O mesmo agente, ou outro processo especializado, pode verificar regularmente se os dados coletados atingem algum limite predefinido. Caso positivo, ele realizará as ações de gestão da observabilidade adequadas, como criar um evento ou notificar a equipe de operação. Por fim, o servidor de gestão da observabilidade é responsável por armazenar dados e disponibilizá-los para outros processos de gerenciamento. Embora descritos como três componentes independentes, eles também podem estar localizados no mesmo ambiente de execução, implementados como funções diferentes de um mesmo serviço.

A Figura 3.2 mostra uma visão detalhada da camada de orquestração em *Fog* já exibida na Figura 2.5. Na Figura 3.2, a Fase Monitorar é detalhada para expor as funções de gestão da observabilidade, os domínios de instrumentação, e os componentes do sistema. Também são mostradas as funcionalidades da orquestração (Seção 2.4) responsáveis pela implementação das Fases Analisar, Planejar e Executar, corroborando o papel fundamental que a gestão da observabilidade tem na orquestração de serviços em *Fog*. A Fase Monitorar é central, coletando dados de observabilidade que alimentam as demais fases do *loop* de controle MAPE [51]. Esta figura demonstra visualmente como a gestão da observabilidade se integra com as outras funcionalidades da orquestração, mostrando a importância dos dados de observabilidade para o suporte à tomada de decisões em *Fog Computing*.

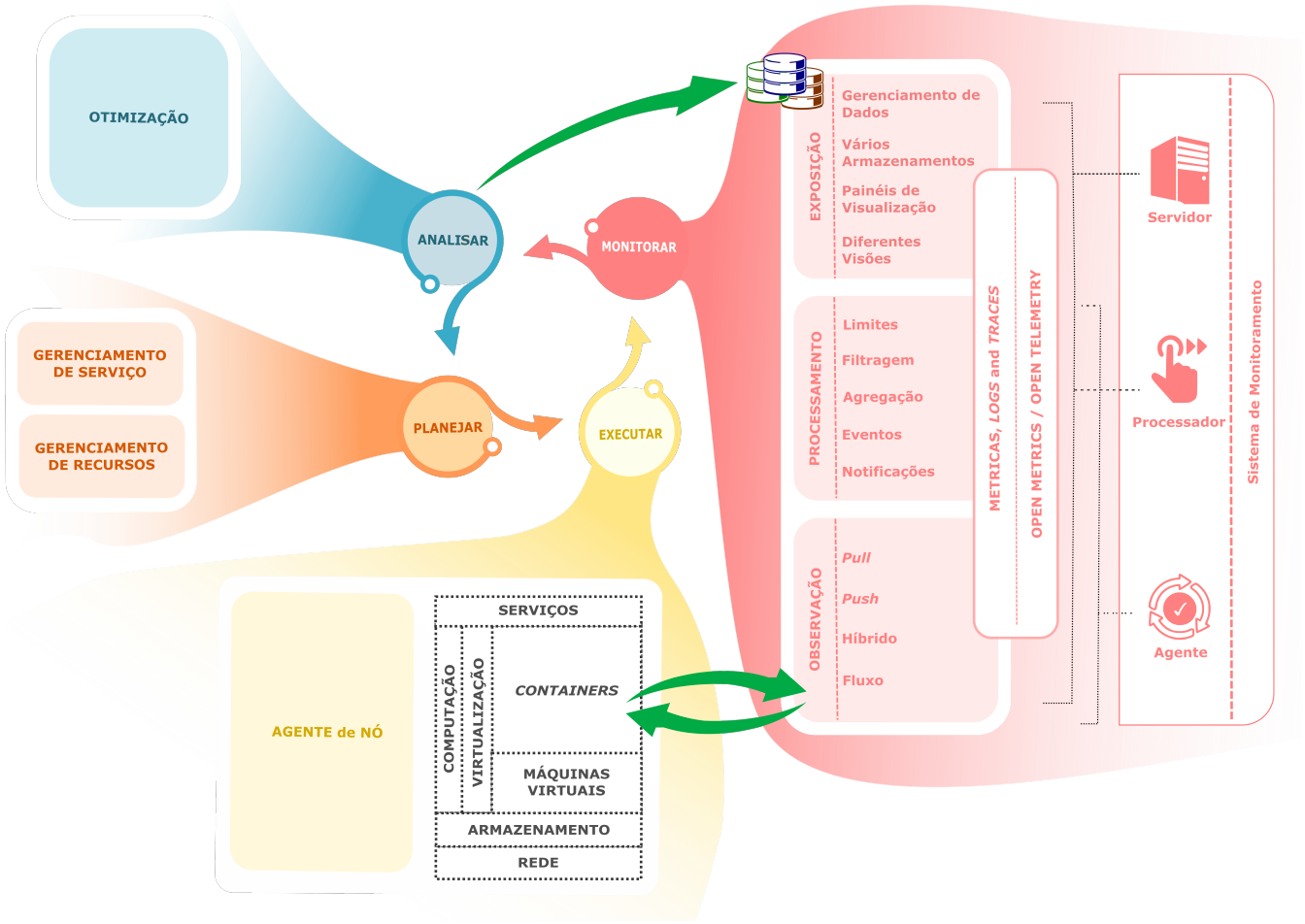


Figura 3.2: Camada de orquestração *Fog* detalhando as funções, os domínios de instrumentação e os componentes do sistema de gestão da observabilidade.

3.6 Requisitos e Desafios da Gestão da Observabilidade em *Fog*

Para lidar adequadamente com características de *Fog* – por exemplo, dispositivos heterogêneos e com restrição de recursos, variedade e instabilidade de conexões –, as soluções de gestão da observabilidade de *Fog* devem atender a novos requisitos que não estão disponíveis em soluções de gestão da observabilidade em *Cloud* [7], demandando abordagens específicas. De acordo com Abderrahim *et al.* [7], as soluções para gestão da observabilidade de infraestruturas de *Fog* devem atender aos seguintes requisitos:

- Escalabilidade - para lidar com o aumento do número de *Fog Nodes*;
- Resiliência a Aparições/Remoções de *Fog Nodes* - mobilidade é uma característica potencial de um *Fog Node*;
- Resiliência a Mudanças/Falhas na Rede - já que esta é uma característica de *Fog*;
- Modularidade - deve haver espaço para adaptação ou parametrização, pois existem diferentes cenários de execução de serviços em ambientes *Fog* de acordo com cada caso de uso específico. Por exemplo, capacidade de recursos e estabilidade de rede podem ser muito diferentes entre os casos de uso de *Industrial* IoT (IIoT) e de gerenciamento de tráfego veicular;

- Localidade - a gestão da observabilidade deve estar o mais próximo possível dos recursos e serviços monitorados.

Segundo Taherizadeh *et al.* [8], as soluções de gestão da observabilidade em *Fog Computing* têm vários desafios, sendo os principais:

1. Gerenciamento de Dados - coleta, processamento e transmissão de dados de instrumentação podem sobrecarregar a rede em um ambiente de grande escala. Há pouca pesquisa sobre o armazenamento e o gerenciamento de *logs* e *traces* [1];

2. Descentralização Coordenada - quando a topologia de controle é hierárquica ou distribuída, existe o risco de dessincronização das ações de gerenciamento, e perda de esforço e tempo para ressincronizar os componentes distribuídos (gerenciando consenso e sincronizando dados replicados) [8]; Topologias centralizadas já são abordadas por diversas pesquisas da academia, e há muitas soluções propostas;

3. Tolerância a Falhas - o serviço pode continuar operando sob um evento falho [68], por exemplo quando o *Fog Node* perde a conexão com o orquestrador, mas o solicitante está consumindo o serviço normalmente. Mecanismos de detecção e recuperação *offline* devem ser necessários para reintegrar o *Fog Node* à infraestrutura, e adquirir os dados gerados no período da desconexão;

4. Gerenciamento de Mobilidade - um dispositivo móvel do usuário final pode ter uma variação nos parâmetros de rede da conexão com o *Fog Node*, alterando rapidamente a Qualidade de Experiência (QoE) do serviço [69]. Assim, dependendo das configurações de coleta de dados, essas informações podem chegar ao módulo de tomada de decisão com um atraso, o que dificultará a realização de ações para preservar o SLA e o QoE dentro da faixa acordada;

5. Escalabilidade e Disponibilidade de Recursos na Borda - embora haja restrição de recursos nos *Fog Nodes*, quando um serviço é executado, ele deve acomodar certo aumento de demanda sob o risco de causar indisponibilidade do serviço por falta de recursos [69];

6. Conhecimento Prévio - é necessário conhecimento prévio sobre infraestrutura subjacente e distribuição de componentes de serviço para garantir que um *Fog Node* atenda aos requisitos de QoS [70];

7. Interoperabilidade e Evitar o Bloqueio do Fornecedor - o bloqueio do fornecedor é uma desvantagem da *Cloud* [71]. *EdgeX Foundry* [72], *Open Edge Computing* [73] e *OpenFog RA* [34] são projetos que visam a padronizar um *framework* para *Fog Computing*;

8. Agendamento Ótimo de Recursos entre *Fog Nodes* - o agendamento de recursos deve estar ciente da dinamicidade do ambiente de execução em relação à mobilidade do usuário, e variação na QoS da conexão. Assim, ele deve garantir respostas rápidas nestas condições [74];

9. Computação Proativa - para antecipar eventos críticos e desencadear ações e decisões para tratá-los de forma proativa, restrições de tempo devem ser consideradas, e isso demanda grandes quantidades de dados históricos [75];

10. Replicação de Serviços - a replicação de serviços é uma estratégia para aumentar o desempenho, a disponibilidade e a tolerância a falhas do serviço. Mas também aumenta a complexidade do gerenciamento, exigindo verificações de sincronização [76];

11. Segurança de *Container* - o uso de *containers* como ambientes de execução, embora sejam considerados adequados para *Fog Computing*, podem representar novas ameaças à segurança [77]. Contudo, além desse risco, existem muitas ferramentas e técnicas adequadas que podem diminuir a superfície de ataque dos *containers*.

Assim sendo, esses requisitos e desafios de gestão da observabilidade apresentam outra perspectiva para destacar as principais diferenças entre gestão da observabilidade de *Cloud* e gestão da observabilidade de *Fog*. A heterogeneidade de *Fog Nodes* e a distribuição maior exigem que a solução de gestão da observabilidade de *Fog* seja independente de plataforma de execução. Além disso, ela deve interoperar com uma variedade de ambientes de virtualização e estar preparada para rajadas de aumento no número de dispositivos e no volume de dados. Uma maneira de conseguir isso é ser modular e escalável, pois a baixa latência esperada de aplicativos de *Fog* exige que as informações coletadas estejam disponíveis em tempo hábil para a tomada de decisões. Todavia, é necessário um equilíbrio para lidar com a restrição de recursos de *Fog Nodes*, conexões instáveis e mobilidade do dispositivo. É fundamental estar mais próximo dos itens que estão sendo monitorados, fornecer diferentes estratégias para coletar os dados (coleta em massa, *push versus pull*, taxa adaptativa, conjunto de métricas adaptativas, etc.), e ser resiliente por meio do gerenciamento de tolerância a falhas.

3.7 Gestão da Observabilidade como uma Função que Compõe a Orquestração de *Fog*

Apesar de ser uma funcionalidade importante, o módulo de gestão da observabilidade de um orquestrador deve focar em seus objetivos específicos: coletar dados sobre uso de recursos e desempenho, gerar eventos quando os limites pré-configurados são atingidos, transmitir os eventos, as notificações e os dados coletados para um módulo decisor. Assim, estar focado significa que as soluções de gestão da observabilidade que atendem às necessidades de um orquestrador de *Fog* não devem implementar funções associadas a outras funcionalidades de orquestração. Não haveria valor se os dados de gestão da observabilidade coletados, e os eventos gerados, não fossem usados para a tomada de decisões sobre o gerenciamento do ciclo de vida de recursos e serviços. Mas os módulos de Gerenciamento de Serviços e de Gerenciamento de Recursos são os responsáveis por tomar essas decisões. E eles fazem isso usando a visão mais ampla que o orquestrador possui da infraestrutura de *Fog* e serviços providos, além do relacionamento do usuário final com *Fog* e com o provedor de *Cloud*.

A orquestração exige que funcionalidades complementares atuem em conjunto, mas cada uma fazendo seu trabalho específico, e isso aponta para implementações leves de cada módulo visando que o *overhead* de orquestração seja tão baixo quanto possível no ambiente de *Fog* de forma a não impactar negativamente o funcionamento das aplicações. Soluções de gestão da observabilidade (e isso vale também para outras funcionalidades de um orquestrador) que incorporam ferramentas e funções de outras funcionalidades, embora

possam ser usadas como um serviço de *Fog* autônomo, não são adequadas para compor um orquestrador devido ao potencial desperdício de recursos, e ao alto *overhead* de ter funcionalidades replicadas.

Na seção anterior, apenas os três primeiros desafios de gestão da observabilidade (gerenciamento de dados, descentralização coordenada e tolerância a falhas) apresentados por Taherizadeh *et al.* [8] são específicos de um módulo de gestão da observabilidade que compõe um orquestrador de *Fog*. Os demais desafios são preocupações primordiais de outras funcionalidades da orquestração (Seção 2.4.1) como Gerenciamento de Serviços (Itens 4, 5, 6 e 10), Gerenciamento de Recursos (Itens 5, 6 e 8), Otimização (Itens 4 e 9), Segurança (Item 11) e Gerenciamento da Comunicação (Item 7).

No desafio de gerenciamento de mobilidade (Item 4), por exemplo, monitorar é coletar e registrar métricas de infraestrutura e serviço da mesma forma que se o usuário final estivesse parado. Mas se uma das métricas coletadas estiver diretamente relacionada à QoE, como é o caso de tempo de resposta do serviço, assim que essa informação estiver disponível para o orquestrador, o gerenciamento de serviços poderá verificar se o tempo de resposta está dentro do intervalo acordado, e agir se não estiver. A ação pode ser, por exemplo, replicar o serviço para mais próximo da localização atual do usuário final. Para isso, ele pode chamar o Gerenciamento de Recursos e passar, como parâmetros, informações de serviço e requisitos. O gerenciamento de recursos alocará um novo *Fog Node* (ou retornará informando que nenhum está disponível, e encaminhará a requisição para a *Cloud*), examinando o inventário de recursos, e implantará a réplica do serviço. Portanto, o gerenciamento de mobilidade não é um desafio específico de gestão da observabilidade de *Fog*, mas sim um desafio de orquestração de *Fog*.

Essa especialização é importante não apenas para economizar recursos e atender às necessidades de baixa latência, mas também porque *Fog* pode agregar valor a vários casos de uso com diferentes requisitos de conectividade, mobilidade, etc. Para ser útil em diferentes cenários, um orquestrador de *Fog* deve ser implementado de forma modular, e este requisito deve ser estendido às suas funcionalidades [7]. Cada funcionalidade de orquestração deve permitir a parametrização e a adaptação, para que possa atender aos requisitos do caso de uso específico em vigor. Quanto mais focada for uma solução de gestão da observabilidade, mais fácil será trocá-la, quando necessário, para outra que possa ser mais apropriada em um cenário diferente (por exemplo, dentro de um dispositivo proprietário).

3.8 Considerações Finais

Este capítulo apresentou o conceito de gestão da observabilidade, ampliando o conceito tradicional de monitoramento para incluir a coleta e análise de métricas, *logs* e *traces*, a fim de obter uma visão mais abrangente do desempenho dos serviços e da infraestrutura. Assim, foram detalhadas as funções de um sistema de gestão da observabilidade, os domínios de instrumentação (métricas, *logs* e *traces*), e os desafios específicos para implementação em ambientes *Fog*. Além disso, ressaltou-se o papel fundamental da gestão da observabilidade no contexto da orquestração de serviços em *Fog*. O cenário motivador desta tese, que é um caso de uso de cidades inteligentes, foi descrito. Este cenário, caracterizado pela necessidade de processamento de dados na borda da rede, servirá como um exemplo prático para avaliar e validar as propostas apresentadas nos capítulos seguintes.

No próximo capítulo, serão apresentados os conceitos relacionados à adaptabilidade de sistemas distribuídos, estabelecendo as bases para a discussão de como tornar os sistemas de gestão da observabilidade em *Fog* mais flexíveis e responsivos às mudanças no ambiente.

Capítulo 4

Adaptabilidade em Sistemas Distribuídos

A adaptabilidade em sistemas distribuídos é um conceito fundamental que permite que esses sistemas se ajustem dinamicamente a mudanças no ambiente, carga de trabalho ou requisitos dos usuários. Um sistema é autoadaptável e configurável se consegue controlar e operar, de forma autônoma, em condições dinâmicas [78]. No contexto de *Fog Computing*, em que recursos computacionais e serviços são distribuídos de maneira descentralizada, mais próximos dos usuários finais e dispositivos de borda, a adaptabilidade se torna ainda mais crítica. Isso se deve à natureza heterogênea e volátil dos ambientes nos quais *Fog Computing* opera, como no caso das cidades inteligentes. Nesses cenários, sistemas de gestão de observabilidade adaptáveis são essenciais para monitorar, analisar e responder em tempo real a eventos, garantindo a eficiência, a resiliência e a qualidade dos serviços prestados. A adaptabilidade nesses sistemas permite auto-organização, auto-otimização e a capacidade de recuperação automática [79], facilitando a gestão de recursos distribuídos e a manutenção de desempenho e disponibilidade diante de flutuações de demanda ou de falhas de componentes.

O capítulo está estruturado em três seções. A Seção 4.1 aborda conceitos relacionados à adaptabilidade de sistemas distribuídos, como os *loops* de controle. Na sequência, a Seção 4.2 apresenta os padrões de projeto que incorporam a lógica de adaptabilidade aos sistemas de gestão de observabilidade em *Fog*. Por fim, a Seção 4.3 faz uma síntese dos principais tópicos abordados neste capítulo.

4.1 Adaptabilidade

A infraestrutura de *Fog* consiste em dispositivos heterogêneos, eventualmente limitados em recursos e conectados por canais de comunicação potencialmente não confiáveis. Além disso, mobilidade é uma característica comum tanto dos *Fog Nodes* quanto dos dispositivos de usuário final. Nesse cenário, um sistema de observabilidade que muda seu modo de operação para se adaptar a situações dinâmicas inesperadas é altamente benéfico. Dependendo da carga total no sistema, seja no dispositivo de usuário final ou no *Fog Node*, algumas decisões podem ser tomadas para reduzir o impacto do fluxo de dados de observabilidade. Esse comportamento adaptativo pode ser aplicado à frequência de coleta de dados, ao volume e ao tipo (domínio de instrumentação) de dados a serem transmitidos, e ao modelo de comunicação. O objetivo é limitar ou atrasar ações específicas quando a carga do sistema é alta, e tentar retomar essas ações o mais rápido possível [80].

Um *loop* de controle é uma estrutura projetada para impor controle automático sobre o comportamento dinâmico de um sistema, e tem sido reconhecido como um componente vital na conquista da autoadaptação em sistemas de software [81, 82]. Existem diferentes *loops* de controle, como OODA (Observar, Orientar, Decidir, Agir) [83], MAPE-K (Monitoramento, Análise, Planejamento, Execução, Conhecimento) [51] e o ciclo cognitivo (Sensorial, Análise, Decisão, Ação) [84]. Todos eles começam por entender o contexto, verificando se uma ação é necessária e tomando a ação, de acordo com sua configuração, que pode ser atualizada dinamicamente, proporcionando ao sistema que os implementa um comportamento autoadaptável ou autônomo.

O *loop* MAPE-K foi introduzido em 2003 pela IBM, e tem sido usado como o padrão para autoadaptação e comportamento autônomo em sistemas [51]. Bonomi *et al.* [50] usaram esse *loop* de controle para estruturar sua proposta de uma camada de orquestração de *Fog* capaz de sustentar serviços de IoT. Esse *loop* de controle também tem sido utilizado em alguns sistemas de IoT recentes como uma referência de modelo de controle [85–87].

Os benefícios de um sistema autoadaptável incluem eficiência aprimorada, resiliência e a capacidade de atender às necessidades dinâmicas dos usuários e às condições ambientais [88, 89]. Esses sistemas podem ajustar automaticamente seu comportamento em resposta a mudanças, otimizando a utilização de recursos e mantendo a qualidade do serviço sem intervenção humana. Essa capacidade é preciosa em *Fog Computing*, em que a distribuição de recursos requer uma gestão sofisticada para lidar com a variabilidade na demanda, nas condições da rede e na integração de vários dispositivos e serviços.

No entanto, fornecer comportamento autoadaptável a um sistema de observabilidade em ambientes de *Fog* apresenta vários desafios. Esses incluem a complexidade de gerenciar e configurar dinamicamente muitos componentes distribuídos, garantindo a consistência e a integridade dos dados ao longo do *continuum* IoT-*Fog*-*Cloud*, e mantendo a segurança e privacidade em um contexto altamente dinâmico e descentralizado. Além disso, equilibrar a granularidade dos dados de observabilidade com as restrições de recursos requer algoritmos e políticas sofisticados para determinar quais dados coletar, analisar e armazenar, e quando adaptar essas decisões com base nas condições e prioridades atuais.

Barba e Giorno [79] definiram a adaptabilidade, no contexto dos serviços de IoT, como a habilidade de reorganizar os componentes dos serviços sem interromper o funcionamento dos outros componentes. Por meio de uma revisão de literatura, eles unificaram os requisitos para uma arquitetura de referência que promove a adaptabilidade dos serviços de IoT. Os autores identificaram que certos requisitos estão diretamente associados a três elementos fundamentais dessa arquitetura, que são: percepção, raciocínio e comportamento. A pesquisa de Barba e Giorno [79] culminou na lista a seguir de requisitos para uma arquitetura de referência que facilita a adaptabilidade dos serviços de IoT:

1. **Existência de componentes de serviço independentes** - para reconfigurar os serviços de IoT durante a execução, componentes de software independentes devem ser substituíveis sem afetar a funcionalidade de outras partes;

2. **Existência de uma interface de configuração de serviço de IoT** - fornece uma interface que permite que os usuários interajam com o sistema ou permite que os administradores de sistema configurem serviços de IoT em tempo real;

3. **Contextualização dos componentes que atendem o serviço de IoT (percepção)** - compreende módulos para coleta de contexto e criação de modelos de situações e cenários;

4. **Capacidade de tomada de decisão do serviço (raciocínio)** - a capacidade do serviço de IoT de tomar decisões com base nas informações reunidas a partir de vários componentes, e agir no ambiente de acordo com as conclusões tiradas dessas informações;

5. **Capacidade de o serviço de IoT agir sobre o ambiente (comportamento)** - ao interagir com o ambiente e com outros componentes, guiar o comportamento dos componentes com base em objetivos definidos e no reconhecimento do contexto em que estão situados. Esses comportamentos estão encapsulados dentro dos componentes;

6. **Existência de uma estrutura de linguagem comum** - um formato de mensagem unificado para todos os componentes garante comunicação eficaz, pois estabelece um protocolo para interação entre os componentes;

7. **Roteamento de mensagens e *middleware*** - permite que os componentes identifiquem partes interessadas para seus dados referindo-se a outro componente centralizado que contém informações abrangentes do sistema;

8. **Mudança dinâmica dos serviços IoT** - o reconhecimento dos contextos ambientais, a tomada de decisão com base nesses contextos e a diferenciação dos comportamentos dos componentes permitem a reconfiguração dos serviços em tempo de execução.

4.2 Lógica e Implementação de Controles de AutoAdaptação

Ramirez & Cheng [89] definiram um projeto para abordar os desafios de construção de sistemas que podem se adaptar a condições e requisitos em mudança. De acordo com eles, "esses padrões orientados à adaptação facilitam o desenvolvimento separado da lógica funcional e da lógica adaptativa", tornando mais fácil projetar, implementar e manter esses sistemas complexos. Ao encapsular soluções comprovadas para problemas comuns de adaptação, esses padrões promovem a reutilização e melhoram a confiabilidade dos sistemas adaptativos. Esses doze padrões de *design* podem servir como a base para incorporar um comportamento autoadaptativo na arquitetura. Eles foram classificados pelos autores em: (i) padrões de gestão da observabilidade; (ii) padrões de decisão; e (iii) padrões de reconfiguração, de acordo com a parte em que atuam no processo de adaptação. Barba e Giorno [79] propuseram uma categorização comparável para os agentes que formam um serviço autoadaptativo, mas utilizaram os termos percepção, raciocínio e comportamento.

Embora todos os 12 padrões definidos por [89] possam agregar valor a uma variedade de arquiteturas e sistemas, alguns padrões foram considerados menos relevantes por focarem em aspectos mais detalhados da implementação ou por lidarem com reconfigurações complexas que não se alinhavam com a abordagem direta adotada. Outros foram deixados de lado por pressuporem a necessidade de inserir ou remover componentes inteiros, em vez de simplesmente ajustar a configuração dos componentes existentes. Assim, foram selecionados aqueles padrões que podem contribuir para um sistema de gestão da observabilidade de *Fog Computing*, considerando os conceitos, as características e as limitações descritas no Capítulo 3.

Os padrões de gestão da observabilidade selecionados se concentram na coleta eficiente de dados de componentes distribuídos e remotos, expondo atributos encapsulados do componente para consulta, e garantindo a distribuição eficiente de dados em todo o sistema adaptativo. Assim, esses padrões são:

- **Fábrica de Sensores**: este padrão foca na necessidade de coletar dados sobre o sistema e seu ambiente de execução em sistemas adaptativos. Ele utiliza uma rede distribuída de sensores. Essa abstração separa clientes e componentes dos sensores, facilitando uma infraestrutura de gestão da observabilidade versátil que pode se adaptar a mudanças;
- **Roteamento Baseado em Conteúdo**: quando múltiplos clientes precisam da mesma informação de gestão da observabilidade, consultar um componente repetidamente pode degradar o desempenho. Este padrão introduz uma arquitetura de publicação/subscrição, de muitos para muitos, para resolver este problema. Clientes se subscrevem em tipos específicos de eventos, e a arquitetura garante que eles sejam informados sempre que os sensores publicarem os dados pertinentes. Essa abordagem alivia a carga sobre o componente monitorado e permite um número dinâmico de clientes.

Padrões de tomada de decisão são cruciais em sistemas adaptativos, pois determinam quando e como o sistema deve ser reconfigurado com base nas informações coletadas por padrões de gestão da observabilidade. Esses padrões selecionados têm a intenção de analisar os dados coletados, identificar situações que requerem adaptação e selecionar as reconfigurações apropriadas para garantir o funcionamento ideal do sistema. São eles:

- **Detecção de Adaptação** - este padrão interpreta os dados brutos dos sensores, associando-os a valores que indicam a diferença entre o comportamento esperado e o observado. Se a divergência exceder um limite definido, um evento é acionado, levando à seleção e aplicação de uma reconfiguração;
- **Raciocínio Baseado em Casos** - ideal para cenários nos quais as situações que requerem reconfiguração são conhecidas e a lógica de decisão é simples. Este padrão utiliza um repositório de conhecimento. Os eventos monitorados são comparados com regras predefinidas e as instruções de reconfiguração correspondentes são aplicadas quando ocorre uma correspondência.

Padrões de reconfiguração são essenciais em sistemas autoadaptativos porque são responsáveis por fazer as mudanças estruturais e comportamentais necessárias para que o sistema se adapte às condições em tempo real. O padrão selecionado garante que essas mudanças ocorram de forma segura e consistente, evitando estados de erro no sistema, como a seguir:

- **Reconfiguração do Servidor**: este padrão descreve a reconfiguração de um servidor em tempo real, sem interrupções de serviço. Ele utiliza *buffers* para armazenar solicitações recebidas durante a reconfiguração, garantindo que todas sejam processadas após a conclusão.

Esses padrões de projeto selecionados facilitam a construção de sistemas de gestão de observabilidade de *Fog* autoadaptáveis devido à sua funcionalidade intrínseca. Eles serão instanciados na arquitetura proposta neste trabalho, descrita detalhadamente no Capítulo 8.

4.3 Considerações Finais

Este capítulo abordou os conceitos relacionados à adaptabilidade de sistemas distribuídos. Dessa forma, foram apresentados os requisitos de uma arquitetura de referência que favorece a adição de adaptabilidade aos sistemas. Ademais, foram selecionados padrões de projeto que podem ser utilizados para incorporar comportamento autoadaptativo em sistemas que executam em *Fog Computing*. De um total de doze padrões de projeto identificados na literatura, cinco foram selecionados por oferecerem uma maior contribuição para um sistema de gestão de observabilidade em *Fog Computing*.

No próximo capítulo, será definida uma taxonomia para sistemas de gestão da observabilidade em *Fog Computing*. A taxonomia vai considerar a adaptabilidade como um domínio relevante nesse contexto, e servirá como base para classificar e comparar as soluções existentes na literatura.

Parte II

Contribuições desta Tese

Capítulo 5

Taxonomia de Sistemas de Gestão da Observabilidade em *Fog Computing*

Este capítulo apresenta uma taxonomia para auxiliar no processo de categorização de soluções de gestão da observabilidade em *Fog Computing*. A taxonomia foi baseada em uma revisão sistemática da literatura, realizada pelo autor desta tese, e está publicada em [80], cuja primeira página pode ser visualizada no Anexo II. Assim sendo, a estrutura deste capítulo está organizada em três seções. A Seção 5.1 apresenta o método utilizado na revisão sistemática da literatura. Na sequência, a Seção 5.2 descreve os domínios e as categorias que compõem a taxonomia proposta de soluções de gestão da observabilidade em *Fog*. Por fim, a Seção 5.3 apresenta uma síntese sobre a taxonomia proposta.

5.1 Método de Seleção de Artigos

Esta seção descreve o método usado para revisar sistematicamente a literatura sobre gestão da observabilidade de *Fog*. A revisão foi inspirada nos trabalhos de [48] e [49], e as etapas realizadas foram: (i) definir as questões de pesquisa (QP); (ii) escolher as bases de dados da pesquisa; (iii) criar uma *string* de busca feita de palavras-chave relevantes; (iv) coletar todos os resultados; (v) aplicar os critérios de inclusão e de exclusão; (vi) filtrar os artigos com base nas palavras-chave, título e resumo; e (vii) ler e analisar os artigos selecionados. As questões de pesquisa definidas para nortear a revisão sistemática foram:

- QP1 – Quais são as características relevantes de uma solução de gestão da observabilidade de *Fog*? A resposta a esta pergunta será consolidada como uma nova taxonomia, e ajudará os pesquisadores a identificar quais características relevantes uma solução de gestão da observabilidade de *Fog* deve ter;
- QP2 - Quais soluções de gestão da observabilidade estão preparadas para compor um orquestrador de serviço de *Fog*? A resposta a esta questão virá utilizando a taxonomia para categorizar as propostas de estado da arte selecionadas por esta revisão sistemática da literatura;

Para esta revisão sistemática, foram usadas, como fonte de pesquisa, as seguintes bases de dados: Scopus[1], Web of Science[2], ACM Digital Library[3] e IEEE Xplore Library[4]. A *string* de pesquisa básica criada foi "(Fog OR Edge) AND (Monitor* OR Observability)". Pesquisas complementares foram feitas usando nomes de outros paradigmas distribuídos (por exemplo, MEC, Cloudlet, etc, conforme listado na Seção 2.2). O conjunto resultante foi composto por 75 artigos. Por fim, após a leitura e a análise do texto completo dos artigos restantes, foram selecionados 10 artigos, os quais foram analisados para responder às questões de pesquisa.

5.2 Taxonomia de Características de Sistemas de Gestão da Observabilidade na *Fog*

Esta seção apresenta uma nova taxonomia de soluções de gestão da observabilidade de *Fog*, que foi criada a partir da revisão sistemática da literatura. A taxonomia consolida domínios e categorias relevantes em uma solução de gestão da observabilidade de *Fog* de última geração. O conteúdo fornecido por esta seção responde à primeira questão de pesquisa: "QP1 - Quais são as características relevantes de uma solução adequada de gestão da observabilidade de *Fog*?".

Vários trabalhos propuseram taxonomias de soluções de gestão da observabilidade de *Cloud Computing* [58, 90–92]. Eles classificaram as ferramentas de gestão da observabilidade de *Cloud* disponíveis por meio das taxonomias propostas em seus trabalhos. Assim, utilizando o processo descrito por Usman *et al.* [93], criou-se uma taxonomia para categorizar soluções de gestão da observabilidade de *Fog Computing*. Apesar das diferenças entre *Cloud* e *Fog*, alguns domínios e categorias de soluções de gestão da observabilidade de *Cloud* são aplicáveis em soluções de gestão da observabilidade de *Fog* – por exemplo, topologia e frequência de transmissão de dados –, enquanto outros não são aplicáveis – por exemplo, tipos de *Cloud*: privada/pública. Mas mesmo os domínios aplicáveis precisam ser revisados e, eventualmente, adaptados para refletir os novos requisitos e cenários de *Fog Computing*.

Dessa forma, analisou-se diferentes taxonomias de *Cloud*, e revisou-se a literatura sobre gestão da observabilidade de ambientes e serviços de *Fog*. Com base nos desafios da orquestração de *Fog Computing* (Seção 2.4), nas características das soluções de gestão da observabilidade de *Fog* (Seção 3.3), e em vários trabalhos selecionados da literatura, identificou-se domínios e categorias relevantes neste contexto. Eles estão resumidos na Figura 5.1, e uma descrição detalhada deles é apresentada nas seções a seguir.

5.2.1 Objetivos

Este domínio define os objetivos de uma solução de gestão da observabilidade de *Fog*. Os mais frequentes encontrados na literatura são o rastreamento de uso de recursos/serviços, e o monitoramento de desempenho. A partir dos dados coletados para atingir esses objetivos, também podem ser apoiados os processos de tolerância a falhas e de faturamento [10]. Um orquestrador de *Fog*, ou outro sistema de gerenciamento,

[1] scopus.com
[2] webofknowledge.com
[3] dl.acm.org
[4] ieeexplore.ieee.org

Figura 5.1: Taxonomia proposta para uma solução de gestão da observabilidade de *Fog*.

deve ter informações atualizadas sobre seus recursos gerenciados para tomar decisões informadas. Cada tipo de recurso (CPU e rede, por exemplo) terá um conjunto específico de métricas que informam seu *status*, tais como porcentagem de CPU livre e porcentagem de perda de pacotes. De acordo com uma configuração estática ou dinâmica, uma solução de gestão da observabilidade deve coletar os dados relevantes para os demais processos de gerenciamento, armazená-los localmente, quando houver espaço suficiente, e transmiti-los em tempo hábil para análise e armazenamento persistente.

Fog Computing é um paradigma mais distribuído que *Cloud Computing*, e seus *Fog Nodes* e canais de comunicação (físicos e virtuais) são potencialmente restritos em recursos e instáveis, devido à heterogeneidade e mobilidade. Em um cenário tão instável, o monitoramento de desempenho pode alavancar o gerenciamento de SLA feito pelo orquestrador de *Fog*. O monitoramento de desempenho pode ser feito em diferentes níveis ou modelos de serviço, e cada um deles pode ter seus próprios indicadores. O desempenho de um canal de comunicação pode ser medido pela taxa de transferência em bytes por segundo. O desempenho de um banco de dados pode ser medido pelo volume de transações por segundo, e o desempenho de um serviço pode ser medido pelo tempo de resposta.

Outra função de uma solução de gestão da observabilidade de *Fog* é o apoio a processos de tolerância a falhas. Para verificar a disponibilidade de um *Fog Node*, uma mensagem de pulsação pode ser enviada

do *Fog Node* para o orquestrador em tempo hábil. Após um determinado período sem tal mensagem, o *Fog Node* é considerado *offline*, e o orquestrador de serviços pode tomar decisões sobre alocação de recursos semelhantes disponíveis, sobre migração de serviços para outros *Fog Nodes* e comunicação com os usuários. Uma solução de gestão da observabilidade de *Fog* pode ser responsável pela implementação desse processo de manutenção, e pela geração dos eventos de indisponibilidade ao sistema de gerenciamento.

Embora o modelo de negócios de *Fog Computing* ainda não esteja definido, os registros de uso e alocação de recursos são a base para descrever o consumo de um usuário, e gerar faturas precisas e verificáveis quando necessário [58]. Os registros de faturamento podem ser relatados em uma granularidade diferente da que é utilizada para registro de uso e de desempenho de recursos. Uma solução de gestão da observabilidade de *Fog*, que tem vários objetivos, pode precisar de diferentes modelos de comunicação (ver Seção 5.2.3) e parametrização para alcançá-los corretamente.

5.2.2 Topologia

A topologia descreve como o sistema de gestão da observabilidade é estruturado em termos de distribuição de seus componentes e fluxo de dados. Masip *et al.* [94] descreveram três topologias de controle que podem ser utilizadas em ambientes *Fog*, e que foram adaptadas para descrever topologias de sistemas de gestão da observabilidade: centralizada, com apenas um servidor; hierárquica, na qual um conjunto de recursos/*Fog Nodes* possui um servidor local e estes servidores atuam de alguma forma sobre os dados (filtrando, armazenando, etc.), colaborando entre eles de forma pré-definida; distribuída, em que um componente de gestão da observabilidade é localizado em cada *Fog Node*, e todos os componentes interagem para compartilhar sua visão dos recursos monitorados, e juntos mantêm a visão de todo o ambiente atualizada. Na taxonomia proposta adotou-se 'hierárquica' em vez de 'descentralizada', como nomeado por Masip *et al.* [94], porque parece mais significativo neste contexto.

Em uma topologia centralizada, há apenas um servidor. Este servidor recebe todos os dados de instrumentação enviados pelos agentes implementados nos *Fog Nodes* monitorados, ou o próprio servidor consulta cada *Fog Node* sobre os dados de interesse. Essa topologia é mais fácil de implementar, tornando os agentes de gestão da observabilidade mais simples, mas apresenta algumas desvantagens. Em primeiro lugar, há a questão do Ponto Único de Falha (SPoF, do inglês *Single Point of Failure*), em que uma falha no servidor pode interromper as atualizações de gestão da observabilidade de todo o ambiente de *Fog*, e prejudicar a tomada de decisão. Em segundo lugar, o servidor deve ser executado em um *Fog Node* rico em recursos para lidar com o fluxo e o armazenamento de dados. Na falta de um *Fog Node* com poder computacional suficiente, uma alternativa possível é colocar o servidor em um *cluster* de *Fog Nodes*, e usar diferentes modelos de comunicação. Por último, os canais de rede do servidor podem ficar sobrecarregados com o fluxo de dados de gestão da observabilidade.

Em uma topologia hierárquica há, pelo menos, mais uma camada adicional de migração de dados entre o componente monitorado e o servidor. Os *Fog Nodes* são categorizados por localidade e designados a um servidor local. Esse componente intermediário pode funcionar para esses *Fog Nodes* como um servidor centralizado, para onde todos os dados de gestão da observabilidade são transferidos e armazenados. Além disso, é possível filtrar e agregar os dados antes de enviá-los para apoiar a tomada de decisão. Mas esse servidor local pode não ser o destino final dos dados coletados. O servidor pode ser um componente

independente no sistema, recebendo e gerenciando uma visão de alto nível dos dados de gestão da observabilidade, e integrando-se ao orquestrador de serviços. Caso contrário, o servidor pode ser implementado como uma rede *peer-to-peer* de servidores locais, e a topologia hierárquica funcionará como uma topologia híbrida entre a centralizada e a distribuída.

Em uma topologia distribuída, o servidor é implementado como uma rede *peer to peer* (P2P) de componentes que são distribuídos em todos os *Fog Nodes*. Os dados de gestão da observabilidade coletados devem ser compartilhados (replicados) com todos os *peers* que precisam deles e, eventualmente, com o orquestrador de serviços para tomada de decisão sobre gerenciamento de recursos e serviços. Essa topologia é referida por Abderrahim *et al.* [7] como sendo a melhor opção para ambientes de *Fog*. Ela tem os benefícios de superar problemas de servidor encontrados na topologia centralizada, como SPoF, execução restrita a *Fog Nodes* ricos em recursos, e congestionamento de rede). No entanto, distribuir essa função relevante por alguns *Fog Nodes* não confiáveis pode levar a dados desatualizados sobre o *status* dos recursos. Além disso, quanto maior a rede P2P de servidores distribuídos, maior o risco de dessincronização dos dados de gestão da observabilidade replicados.

5.2.3 Modelos de Comunicação

Uma solução de gestão da observabilidade de *Fog* deve coletar dados de domínios de instrumentação de um *Fog Node*, recurso ou serviço monitorados, e disponibilizar esses dados em tempo hábil para serem analisados e apoiarem a tomada de decisões. Quem consumirá esses dados pode ser um orquestrador de *Fog*, outro sistema de gerenciamento, ou mesmo uma equipe humana. Os dados podem ser enviados periodicamente por um agente para o servidor, e essa categoria de modelo de comunicação é chamada de *Push*. O servidor pode solicitar dados de maneira orientada a eventos, e isso é chamado de *Pull*. A mistura dos dois modelos cria a categoria Híbrido. Por fim, o modelo em Fluxo descreve um fluxo de dados contínuo entre o agente e o servidor.

No modelo *Push*, o agente é responsável por iniciar a transmissão dos dados coletados para o servidor, por isso o agente precisa saber previamente o endereço do servidor. Essas informações são enviadas ao agente no processo de inicialização e podem ser atualizadas enquanto ele está em execução, com base nos eventos do servidor (por exemplo, o servidor está sobrecarregado), ou com base nos eventos do componente monitorado (por exemplo, iminência de fim de carga da bateria).

No modelo *Pull*, o servidor é responsável por iniciar a transmissão de dados e precisa solicitar os dados para os componentes monitorados. Somente após uma solicitação adequada, os dados são enviados pelo agente. Nesse modelo, o servidor pode selecionar as informações necessárias no momento, enviando parâmetros dentro da requisição. Isso pode ajudar a implementar uma estratégia em que o servidor solicite informações de gestão da observabilidade de alta prioridade quando estiver sobrecarregado, e solicite informações em massa quando puder lidar com a carga.

Um modelo Híbrido é uma forma mais flexível de lidar com situações dinâmicas e com a heterogeneidade da infraestrutura e dos serviços monitorados. Com base na capacidade de recursos dos *Fog Nodes*, por exemplo, *Fog Nodes* que estejam com pouco espaço livre de armazenamento de dados podem usar o modelo *Push* como forma de não perder dados. Os *Fog Nodes* com mais recursos podem usar o modelo *Push* somente quando as métricas selecionadas atingirem um limite predefinido, por exemplo, a cada 10%

de aumento ou diminuição de CPU livre. Em outras situações, esses *Fog Nodes* de alta capacidade podem armazenar dados coletados, e aguardar uma solicitação do servidor para envio de dados em massa. Existem outras combinações de modelos *Pull* e *Push* que podem ser configuradas para atender a requisitos ou cenários específicos, nos quais o uso de apenas um modelo não é suficiente.

Por último, em um modelo de comunicação em Fluxo, o agente cria um fluxo de dados com o servidor e transmite dados coletados continuamente. Este é um modelo apropriado para transmitir pequenos volumes de dados prioritários, como mensagens de pulsação e notificações de alta prioridade.

5.2.4 Frequência

Na orquestração, uma tomada de decisão eficaz é apoiada por informações atualizadas sobre recursos e serviços. Quanto maior a frequência de atualizações, por exemplo, cada alteração em um valor de métrica sendo entregue imediatamente ao servidor, menor o risco de lidar com informações desatualizadas. Mas uma alta frequência de atualizações pode causar sobrecarga computacional e congestionamento de rede, principalmente ao lidar com um grande número de dispositivos monitorados. A frequência de coleta deve ser equilibrada com a capacidade computacional e a sobrecarga da rede. Existem diferentes abordagens de frequência para gestão da observabilidade, as quais são:

- Contínua - uma vez iniciada, os dados de gestão da observabilidade fluem para o servidor continuamente;
- Periódica - em que um período recorrente pode ser configurado e os dados são enviados a cada instância de tempo;
- Baseada em Eventos - em que o envio de dados é iniciado pela detecção de um evento no *Fog Node* ou pela resposta a uma solicitação explícita do servidor.

5.2.5 Camadas Monitoradas

Um ambiente de *Fog* é composto, principalmente, por *Fog Nodes*, dispositivos físicos ou virtuais, que executam serviços de *Fog* [95]. Para fornecer os serviços solicitados aos usuários finais, um orquestrador de *Fog* deve gerenciar os recursos e a distribuição de serviços. Assim, devido à heterogeneidade dos *Fog Nodes*, um serviço pode ser executado diretamente no hardware (o que é referenciado como '*bare metal*'), ou na plataforma de virtualização disponível no *Fog Node* (VM, *container*, *unikernel* [96]), considerando que o gerenciamento de serviços (Seção 2.4.1) possui uma imagem específica para cada um deles.

Dessa forma, um sistema de gestão da observabilidade deve coletar dados de várias camadas dentro dos *Fog Nodes*, cada uma potencialmente exigindo bibliotecas de coleta específicas, e gerando eventos e notificações, pois são independentes umas das outras. Métricas associadas ao hardware, tais como percentuais de uso de CPU e de RAM disponíveis, normalmente, são coletadas a partir do sistema operacional, embora também possam ser fornecidas pelo software de virtualização (Docker [97], por exemplo). Métricas associadas à plataforma de virtualização podem ser de interesse, assim como métricas associadas ao serviço, como tempo de resposta. O agente no *Fog Node* (ou fora do *Fog Node*) deve coletar esses valores de métrica usando bibliotecas específicas, assim como protocolos e portas pré-configurados. Este domínio indica para

quais camadas uma solução de gestão da observabilidade está preparada para coletar valores de métrica, sendo: infraestrutura, plataforma ou serviço.

5.2.6 Domínios de Instrumentação

Conforme visto na Seção 3.4, os três domínios de instrumentação mais comuns em sistemas distribuídos são métricas, *logs* e *traces*. O uso de métricas é bem difundido e a maioria das soluções de gestão da observabilidade o implementa. As métricas permitem uma gestão de observabilidade *black-box*, coletando dados do sistema operacional, solução de gerenciamento de *containers* ou outro ambiente de virtualização, sem a necessidade de modificar o sistema que está sendo monitorado. As métricas são leves, quando comparadas a *logs* e *traces*. Devido a essas características, as métricas são bem ajustadas para ambientes de *Fog*, compostos de dispositivos com restrição de recursos e potencialmente móveis, e conectados por canais de comunicação instáveis.

Por outro lado, *logs* e *traces* permitem a análise do estado interno do sistema, permitindo a descoberta das causas do mau desempenho do serviço, e prevendo problemas futuros que podem ser causados por um mau comportamento atual. *Logs* e *traces* podem fazer uso do gerenciamento de fluxo de dados estabelecido para lidar com dados de métricas. No entanto, devido à necessidade de maior disponibilidade de recursos, o impacto do uso de um fluxo único de dados no SLA do serviço deve ser avaliado antes de sua utilização, e deve ser fornecida uma forma dinâmica de ativá-lo e desativá-lo.

5.2.7 Processamento de Dados

Uma solução de gestão da observabilidade de *Fog* deve processar os dados coletados (por exemplo, filtrar, agregar, transformar) como parte da função de processamento [7] (Seção 3.3). Contudo, há um *trade-off* neste domínio. Caso muitas funções de processamento de dados estejam disponíveis em uma solução de gestão da observabilidade, um volume menor de dados de gestão da observabilidade vai fluir para o servidor, reduzindo a carga da rede. No entanto, para executar essas funções apropriadamente, um *Fog Node* mais rico em recursos deve estar disponível. Na categoria de gerenciamento de eventos, os dados coletados podem acionar um evento e o evento pode gerar ações no próprio *Fog Node* (por exemplo, interromper a coleta ao esgotar o espaço de armazenamento), ou no servidor (por exemplo, enviar uma notificação).

De acordo com a estratégia de gerenciamento de dados em vigor, assim que os dados são coletados da fonte de dados (por exemplo, um dispositivo final, uma plataforma como Kubernetes, ou um serviço implantado), eles podem ser filtrados, agregados ou sofrer outro tipo de modificação antes de serem armazenados e transmitidos para o servidor. Alguns valores de métrica detalhados podem ser armazenados no local por um curto período de tempo. Isso pode permitir a geração de alertas para situações pré-configuradas específicas e a recuperação de dados detalhados quando necessário nessa janela de tempo. Além disso, os limites de recursos do dispositivo podem ser respeitados. Uma visualização consolidada desses dados pode ser gerada periodicamente por agregação e transmitida ao servidor. A filtragem é uma função diferente aplicada aos dados, em que apenas os valores de interesse são selecionados para transmissão. Outras funções diferentes podem ser implementadas nos dados coletados, tais como a amostragem [61], em que apenas um percentual dos *traces* são coletados e enviados, com base em uma heurística pré-determinada.

5.2.8 Intrusividade

A gestão da observabilidade de recursos e serviços pode ser realizada por meio de diferentes níveis de intrusão. As soluções de gestão da observabilidade podem ser classificadas como ativas ou passivas, de acordo com a interferência que injetam nas métricas que estão sendo coletadas [98]. Assim, se o processo de coleta de valores das métricas altera a carga do sistema, esse processo é chamado de ativo. Caso contrário, é um processo passivo.

Diferentes métricas podem ter diferentes níveis de intrusão. A execução de um agente local para coletar valores de métrica de CPU pode interferir no valor coletado, pois o próprio agente local utilizará ciclos de CPU para realizar suas ações. No sentido contrário, o StatsD [99], um protocolo de rede, permite a coleta de métricas de rede passivamente sem interferir na carga de rede do sistema.

5.2.9 Escalabilidade

A escalabilidade é um requisito não funcional [100] que garante que uma solução de gestão da observabilidade de *Fog* possa ser dimensionada para absorver um aumento dos *Fog Nodes* monitorados, e dos serviços, sem degradação relevante no desempenho geral do sistema. A escalabilidade também se refere à capacidade de manter o desempenho da solução de gestão da observabilidade sob condições de carga variável. Isso pode envolver a alocação dinâmica de recursos, a replicação de componentes e o uso de técnicas de balanceamento de carga.

As escolhas de arquitetura, topologia, bibliotecas de coleta de dados, protocolos de comunicação, banco de dados para armazenamento local e de longo prazo são relevantes para determinar a capacidade de escalabilidade de uma solução de gestão da observabilidade de *Fog*. A taxonomia proposta define como "Escalável" cada solução de gestão da observabilidade que foi avaliada quanto à escalabilidade, e apresentou evidências de que o sistema proposto é redimensionável quando necessário. Caso contrário, a proposta foi classificada como "Não escalável".

5.2.10 *Overhead*

Para coletar dados de instrumentação dos componentes monitorados, transmiti-los e armazená-los, espera-se que esse processo consuma parte da capacidade computacional do ambiente e largura de banda da rede. Assim, a própria gestão da observabilidade pode ser uma fonte de contenção de recursos, principalmente, em ambientes virtualizados, nos quais agentes e aplicações, funcionando no mesmo ambiente de execução, competem por recursos compartilhados [101]. Quanto maior a quantidade de dados coletados, maior o *overhead* causada pelo processo de gestão da observabilidade [102].

O overhead de uma solução de gestão da observabilidade pode ser medido em termos de consumo de CPU, memória, largura de banda de rede e espaço de armazenamento. É importante minimizar o *overhead* para evitar o impacto no desempenho das aplicações e serviços monitorados. Isso pode ser feito por meio da otimização da coleta, da agregação e da filtragem de dados, assim como por meio do uso de protocolos de comunicação eficientes. Além disso, é importante monitorar o *overhead* da solução para identificar oportunidades de otimização.

Assim, ao fornecer dados e funcionalidades de gestão da observabilidade para apoiar a tomada de decisão na *Fog*, uma solução adequada de gestão da observabilidade de *Fog* deve manter o *overhead* de comunicação e de processamento o mais baixo possível. A taxonomia proposta define como "*Overhead* Baixo" uma solução de gestão da observabilidade de *Fog* que foi avaliada quanto ao *overhead* que injeta no sistema e apresentou resultados positivos no cenário de avaliação.

5.2.11 Adaptabilidade

A infraestrutura *Fog* é composta por dispositivos heterogêneos com recursos restritos, conectados por canais de comunicação potencialmente instáveis. Além disso, a mobilidade é uma característica esperada tanto dos *Fog Nodes* quanto dos dispositivos do usuário final. Nesse cenário, um processo de gestão da observabilidade adaptativo é de grande valia. De acordo com a carga geral no sistema, no componente monitorado ou no servidor, algumas escolhas podem ser feitas para diminuir o impacto do processo de gestão da observabilidade.

A adaptabilidade pode ser implementada por meio de mecanismos de auto-configuração, auto-otimização e auto-recuperação [79]. Isso pode envolver a alteração dinâmica da frequência de coleta de dados, do volume e tipo de dados a serem transmitidos, e do modelo de comunicação. O objetivo é restringir ou adiar algumas ações enquanto a carga no sistema estiver alta, e tentar retomá-las assim que possível. A adaptabilidade também pode ser usada para lidar com falhas e interrupções na rede. A arquitetura da solução deve ser projetada para permitir a adaptabilidade, com componentes modulares e interfaces bem definidas. Além disso, é importante monitorar o ambiente para detectar mudanças e acionar os mecanismos de adaptação.

5.2.12 Integração

Este domínio verifica se a solução de gestão da observabilidade de *Fog* adere a algum formato de dados padronizado. Algumas soluções usam arquivos XML ou JSON como um formato de dados padronizado básico. Nos últimos anos, o OpenMetrics [103] foi publicado para tentar padronizar a troca de dados de métricas entre soluções de gestão da observabilidade. Da mesma forma, foi proposto o OpenTelemetry [104], baseado em dois padrões anteriores: OpenCensus [105] e OpenTracing [106]. Outros padrões como OpenXTrace [107] foram propostos, embora não tenham recebido muita atenção da academia e da indústria. O benefício de usar esses padrões é facilitar a troca de dados e a integração de diferentes componentes de gestão da observabilidade. Esses componentes podem ser de diferentes fornecedores e estarem em diversas camadas da arquitetura *Fog* (Seção 2.2).

A integração com outras ferramentas e plataformas também pode envolver o uso de interface de programação da aplicação (API, do termo em inglês *Application Programming Interface*) e protocolos de comunicação padronizados. É importante definir interfaces bem documentadas para facilitar a integração com outros sistemas. Além disso, a solução de gestão da observabilidade deve ser compatível com os padrões de mercado e as tecnologias mais utilizadas no ambiente de *Fog Computing*. A integração pode ser usada para compartilhar dados de observabilidade com outras ferramentas de análise e visualização, e para automatizar tarefas de gerenciamento e orquestração.

5.2.13 Atender às Necessidades da Orquestração

A orquestração de serviços em *Fog* precisa de dados de gestão da observabilidade para implementar uma tomada de decisão adequada, e integrar as diversas funcionalidades que a compõem (Seção 2.4.1). Uma solução de gestão da observabilidade de *Fog* que está 'Pronta' para atender às necessidades de orquestração é aquela que é:

1. Leve (por exemplo, implementada com um volume de código pequeno) e multiplataforma (ou seja, fornece diferentes versões de seus agentes para atender à heterogeneidade dos *Fog Nodes*);
2. Focada em gestão da observabilidade - significa que não implementa outras funcionalidades de orquestração, ou pelo menos permite desativá-las, causando um *overhead* e um esforço de gerenciamento pequenos;
3. Adaptativa - ser flexível o suficiente para mudar seu comportamento em tempo real, de acordo com as alterações de configuração fornecidas pelo orquestrador de serviço de *Fog*, ou por outro sistema ou equipe de gerenciamento.

Se uma solução de gestão da observabilidade atender a, pelo menos, dois desses requisitos e puder ser adaptada com um esforço razoável para atender ao terceiro, ela será considerada 'Parcialmente Pronta' para atender às necessidades de orquestração. Caso contrário, é considerada 'Não Pronta'.

5.3 Considerações Finais

Nas seções anteriores, foram detalhados os domínios que compõem a nova taxonomia para a gestão da observabilidade em *Fog Computing*. Essa taxonomia oferece uma estrutura clara e organizada para categorizar e entender as soluções disponíveis, destacando-se como uma ferramenta valiosa para pesquisadores e profissionais da área. Os benefícios de se ter uma taxonomia dedicada são numerosos, pois ela não apenas facilita a identificação e a classificação das características essenciais das soluções, mas também promove uma compreensão mais profunda dos desafios e oportunidades no gerenciamento da observabilidade em ambientes de *Fog*.

Além disso, a taxonomia proposta será fundamental para a categorização das soluções encontradas na literatura, permitindo uma análise mais estruturada e comparativa, conforme será explorado no próximo capítulo. Essa abordagem sistemática não só enriquece o campo de estudo, mas também oferece diretrizes claras para o desenvolvimento de futuras pesquisas e inovações na área.

Capítulo 6

Soluções de Gestão da Observabilidade para *Fog Computing*

Neste capítulo é apresentada uma categorização de soluções de gestão da observabilidade de *Fog*, com base nos domínios e categorias definidos na taxonomia proposta no capítulo anterior. Por meio de uma revisão sistemática da literatura, 10 trabalhos de gestão da observabilidade em *Fog Computing* foram analisados, descritos e categorizados com o uso da taxonomia proposta. O capítulo está estruturado em três seções. A Seção 6.1 apresenta e categoriza cada uma das 10 soluções de gestão da observabilidade disponíveis na literatura. Na sequência, a Seção 6.2 faz uma análise comparativa entre as soluções apresentadas. Por fim, a Seção 6.3 resume as principais características das soluções de gestão da observabilidade para *Fog Computing* apresentadas neste capítulo.

6.1 Soluções de Gestão da Observabilidade em *Fog*

Devido às diferenças relevantes entre *Cloud* e *Fog Computing*, não havia garantia de que uma solução de gestão da observabilidade desenvolvida para a *Cloud* funcionaria corretamente em um ambiente de *Fog Computing* [108]. Para confirmar isso, alguns trabalhos recentes analisaram e testaram soluções de gestão da observabilidade de *Cloud*, tanto comerciais quanto de código aberto, como, por exemplo, Nagios [109], Zabbix [110], DARGOS [111], PCMONS [112] e JCatascopia [113]. Essas soluções foram confrontadas com os requisitos e desafios de *Fog Computing*, e o resultado foi que nenhuma delas é adequada para ambientes de *Fog* [7–10]. Para superar os desafios de gestão da observabilidade, alguns autores propuseram soluções e arquiteturas específicas para ambientes de *Fog Computing* e paradigmas relacionados. As próximas subseções descrevem cada uma delas, abordando suas características de acordo com os domínios e categorias que compõem a taxonomia apresentada no capítulo anterior.

6.1.1 PyMon

O trabalho de Großmann e Klug [114] propõe o PyMon, uma solução de gestão da observabilidade para computadores de placa única (SBC, do inglês *Small Board Computer*) baseados na arquitetura ARM. O objetivo é fornecer dados de utilização do *Fog Node* e de *containers* para viabilizar uma orquestração mais

eficiente de serviços em *containers*. O PyMon é construído como uma extensão do Monit [115], uma ferramenta de gestão da observabilidade capaz de inspecionar *containers* Docker. Monit é uma ferramenta leve de código aberto desenvolvida para monitorar sistemas baseados em Unix. Ela é fornecida por meio de imagens Docker, e é executada em arquiteturas que são compatíveis com o Docker, ou seja, x86_64, ARM e AARCH64.

PyMon tem como objetivo monitorar o desempenho. Para isso, a solução usa uma topologia centralizada para coletar valores de métricas de dispositivos IoT a uma taxa periódica usando o modelo de comunicação *push*. Ele está preparado para monitorar camadas de infraestrutura e plataforma, e implementa a agregação de dados de gestão da observabilidade. Os dados recebidos são armazenados em um banco de dados PostgreSQL e podem ser exibidos por meio de uma interface web.

PyMon é uma solução de gestão da observabilidade de *Fog* simples, leve e multiplataforma, com baixo *overhead* de consumo de recursos e desenvolvida para executar em SBCs. Embora focado em gestão da observabilidade, seu conjunto de recursos não é suficiente para atender aos requisitos listados na Seção 3.6, contemplando apenas Localidade, pois possui um agente local para coleta de métricas. Além disso, ele não é adaptável, pois apresenta baixa flexibilidade em termos de modelos de comunicação disponíveis, frequência de transmissão de dados, e mudança de configuração em tempo real, limitando os cenários de gestão da observabilidade que são atendidos por ele. Outrossim, a escalabilidade de PyMon não foi avaliada no artigo [114]. Devido a essas limitações, o PyMon não atende às necessidades de orquestração. A ferramenta PyMon está disponível no Github [116].

6.1.2 FMonE

Brandon *et al.* [59] propuseram o FMonE como uma solução que atende aos requisitos de gestão da observabilidade de *Fog* descritos em seu trabalho. O FMonE é baseado no Marathon [117], uma conhecida solução de orquestração de *containers*, embora o artigo aponte que outra solução de *container* possa ser usada, desde que atenda aos requisitos definidos.

FMonE tem a finalidade de monitorar o desempenho e dar apoio à tolerância a falhas. A solução usa topologias centralizada e hierárquica para coletar valores de métricas a uma taxa periódica, usando modelos de comunicação *pull* e *push*. A solução está preparada para monitorar as camadas de infraestrutura, plataforma e serviço, e fazer a filtragem dos dados de gestão da observabilidade.

Os autores usaram o ambiente de testes do Grid5000 [118] para simular uma infraestrutura de *Fog*, usando 78 VMs, definindo largura de banda e latência entre elas. Eles avaliaram o desempenho do serviço em operações por segundo nos *Fog Nodes*, comparando arquitetura centralizada *versus* hierárquica com o uso do FMonE. Os resultados mostraram que a solução é escalável e teve um baixo *overhead* de consumo de recursos, executando o mesmo serviço com e sem o agente FMonE instalado nos *Fog Nodes*. Esta solução é oferecida como uma estrutura de gestão da observabilidade independente, na qual o usuário final interage diretamente e cria fluxos de trabalho. Em um cenário de orquestração de serviço de *Fog*, os parâmetros de entrada serão fornecidos pelo módulo de Gerenciamento de Serviço (Seção 2.4.1), de acordo com os requisitos do serviço e as necessidades do usuário no momento da solicitação. Além disso, não é focado em gestão da observabilidade, pois é responsável por detectar novos *Fog Nodes* para monitorar. No entanto,

essas duas questões podem ser adaptadas com um esforço razoável. É leve e multiplataforma, mas como não é adaptável, não atende às necessidades de orquestração de *Fog*. FMonE está disponível no Github [119].

6.1.3 Pilha Prometheus

A pilha Prometheus [120] é um sistema de gestão da observabilidade construído com a integração do servidor Prometheus e outros componentes complementares de código aberto, como exportadores de métricas e ferramentas de visualização de dados. Para a coleta de dados de gestão da observabilidade, os seus desenvolvedores disponibilizam a ferramenta Node Exporter [121], que coleta métricas em sistemas baseados em Unix. Para *containers*, o CAdvisor [122], desenvolvido pelo Google, é a ferramenta escolhida. Ele fornece informações sobre o uso de recursos do *host* e dos *containers* em execução em um mesmo dispositivo.

A pilha Prometheus tem o objetivo de monitorar o desempenho. A solução usa uma topologia centralizada para coletar valores de métricas a uma taxa periódica usando o modelo de comunicação *pull*. Está preparado para monitorar camadas de infraestrutura e plataforma, e implementa a agregação de dados de gestão da observabilidade. De acordo com Großmann e Klug [108], a pilha Prometheus mostrou uma boa adaptabilidade viabilizada por um baixo acoplamento de seus componentes de software. Assim, não é uma tarefa complexa modificar partes do *framework* para melhor adaptá-lo a novos cenários. Além disso, o servidor Prometheus é compatível com diversos exportadores de métricas, e pode ser usado para coletar métricas de bancos de dados, servidores web e outros serviços.

O modelo de comunicação padrão usado no Prometheus é *pull*, mas também implementa *push* por meio de um *gateway* [123]. O formato de troca de métricas definido pelo Prometheus foi a base para criar o OpenMetrics [103]. Portanto, a pilha Prometheus é uma solução focada em gestão da observabilidade, padronizada, adaptável e abrangente. Embora não haja avaliação sobre o *overhead* geral do sistema, sua flexibilidade e modularidade podem viabilizar o seu uso em um cenário de baixo *overhead*. Assim, a pilha Prometheus é classificada como atendendo às necessidades de orquestração. Por outro lado, Prometheus é especializado na coleta e gestão de métricas. Uma solução de gestão da observabilidade que gerencie os três domínios de instrumentação pode se utilizar do que Prometheus oferece e adicionar o tratamento para *logs* e *traces*, avaliando a escalabilidade da solução neste cenário. A pilha Prometheus está disponível no GitHub [124].

6.1.4 Monitoramento Osmótico

Souza *et al.* [125] propuseram uma ferramenta para monitorar microsserviços implantados em um ambiente de computação osmótica, ou seja, um ambiente *Fog-Cloud* que permite um fluxo bidirecional de microsserviços. É uma extensão do CLAMBS [126], que é uma ferramenta *multicloud* para monitorar desempenho de microsserviços.

O Monitoramento Osmótico tem a finalidade de monitorar o desempenho. A solução usa uma topologia centralizada para coletar valores de métrica a uma taxa periódica usando o modelo de comunicação *push*. Está preparado para monitorar apenas a camada de infraestrutura e não implementa nenhum recurso de processamento de dados.

A solução requer um agente em cada dispositivo IoT que envia dados para o *Manager*, componente que executa na *Cloud*. As avaliações foram sobre CPU, latência e uso de memória em seis cenários diferentes: três na *Cloud* e três na *Fog*. Entre os cenários, foram comparadas variações de uso de apenas um *container* executando mais de um microsserviço, e um *container* por microsserviço. Assim sendo, embora a computação osmótica permita uma migração bidirecional de microsserviços entre *Cloud* e *Fog*, os experimentos foram feitos com eles em posições fixas. O artigo não aborda o *overhead* nem a escalabilidade da proposta. É específico para o cenário de computação osmótica e não é adaptativo. O seu servidor está localizado na *Cloud* e o gerenciamento de serviços e recursos não é totalmente separado da funcionalidade de gestão da observabilidade, portanto, não atende às necessidades de um orquestrador de *Fog*.

6.1.5 Gestão da Observabilidade de *Fog* e *Cloud* Móvel (*Mobile*)

Os autores do artigo [127] propuseram uma arquitetura baseada em MCC/Cloudlet (ver Seção 2.3), composta por *Cloudlets* distribuídos em vários locais para apoiar dispositivos móveis usando serviços em *Cloud*. Apesar das diferenças entre *Cloudlets* e *Fog Nodes*, em que o primeiro possui maior capacidade computacional e é denominado *datacenter-in-a-box*, a arquitetura proposta e as ferramentas de gestão da observabilidade escolhidas podem ser utilizadas em um ambiente *Fog* com pequenas adaptações, e sob certas condições (por exemplo, em um caso de uso de IIoT em que os *Fog Nodes* são mais ricos em recursos). Os *Cloudlets* são conectados a um sistema de gestão da observabilidade e sua solução é baseada no IEEE 1451 para comunicar os sensores em uma Rede de Sensores Sem Fio (WSN, do inglês *Wireless Sensor Network*) e os *Cloudlets*. Ele usa *Virtual Device Representation* (VDR), um "gêmeo digital" de um dispositivo que está localizado no *Cloudlet*. Sensu [128], um *framework* flexível, é usado para implementar funções de gestão da observabilidade, e Graphite [129] e Grafana [130], soluções de código aberto, para armazenamento e visualização de dados, respectivamente.

A solução tem como objetivo monitorar o desempenho. Ela usa uma topologia centralizada para coletar valores de métricas e *logs* a uma taxa periódica, usando o modelo de comunicação *push*. A solução está preparada para monitorar as camadas de infraestrutura, plataforma e serviço, e para fazer filtragem e agregação de dados de gestão da observabilidade.

Nenhuma avaliação foi feita sobre o *overhead* que a proposta causa no sistema, mas a sua escalabilidade é uma característica comprovada do Sensu [128]. O Sensu oferece uma estrutura abrangente para processar os dados de gestão da observabilidade, além de gerenciamento de eventos. Por ser padronizada e adaptável, proporcionando alta possibilidade de customização e diversas implementações de plataforma, esta solução atende às necessidades de orquestração de serviços de *Fog*.

6.1.6 Switch

Taherizadeh *et al.* [131] apresentaram uma arquitetura de computação capilar distribuída. Ela implementa o *loop* de controle MAPE-K [132], já apresentado na Seção 4.1. A arquitetura proposta inclui um sistema de gestão da observabilidade, denominado *Switch* [133].

Switch tem a finalidade de monitorar o desempenho. A solução usa uma topologia centralizada para coletar valores de métrica a uma taxa periódica usando o modelo de comunicação *push*. Ela está preparada

para monitorar as camadas de infraestrutura, plataforma e serviço, e para gerenciar os eventos criados a partir dos dados de gestão da observabilidade.

Os agentes coletores são desenvolvidos usando o protocolo não intrusivo StatsD [99]. O servidor armazena os dados recebidos no Cassandra [134], um banco de dados de séries temporais (TSDB, do inglês *Time Series Database*) gratuito e de código aberto. O sistema de gestão da observabilidade utiliza *containers* Docker, sendo uma opção leve e multiplataforma. Agente, servidor e outros componentes como um disparador de alarmes, responsável por analisar os dados de gestão da observabilidade e criar eventos e notificações, foram propostos em trabalho anterior dos autores [135]. Embora seja uma solução que oferece baixa flexibilidade em termos de modelos de comunicação e frequência de coleta de dados, é leve, multiplataforma, focada em gestão da observabilidade e adaptativa. Assim, ela atende às necessidades de orquestração de serviços de *Fog*. Por outro lado, o Switch não implementa o gerenciamento de *logs* e de *traces*, não usa formato padronizado de dados, e não teve a escalabilidade avaliada. O sistema de gestão da observabilidade de Switch está disponível no GitHub [133].

6.1.7 Gestão da Observabilidade Baseado em Apoio e Confiança (SCB)

O trabalho [10] propôs uma técnica baseada em Apoio e Confiança (SCB, do inglês *Support and Confidence Based*), visando otimizar o uso de recursos da gestão da observabilidade. O SCB é baseado na predição de confiabilidade de cada dispositivo de *Fog*, com base em seus dados históricos. Adaptado de soluções voltadas à *Cloud*, o trabalho propôs algoritmos para desenvolver modelos de comunicação *push*, *pull* e híbrido, e avaliou esses modelos em um protótipo construído em Java, e os comparou com a abordagem baseada em SCB.

SCB tem como objetivo monitorar o desempenho. A solução utiliza uma topologia hierárquica para coletar valores de métricas a uma taxa periódica ou acionada por eventos, utilizando modelos de comunicação *push*, *pull* e híbrido. Está preparado para monitorar apenas a camada de infraestrutura e não implementa nenhum recurso de processamento de dados.

O seu desempenho é avaliado analisando um caso de uso de tráfego em tempo real em um emulador de *Fog*, e os resultados são comparados com técnicas tradicionais de computação distribuída. Os resultados mostram que a técnica proposta consome menos recursos quando comparada às abordagens convencionais para monitorar recursos, resultando em um baixo *overhead* do sistema e boa escalabilidade. É adaptável e considera que outros processos de gerenciamento de serviços e gerenciamento de recursos existem, e são dependentes de dados e recursos de gestão da observabilidade. Assim, esta proposta atende às necessidades de orquestração de *Fog*. Por outro lado, o SCB não implementa o gerenciamento de *logs* e de *traces*, e nem usa formato padronizado de dados.

6.1.8 Gestão da Observabilidade Baseado em Regras

Os autores do [136] propuseram um sistema de gestão da observabilidade direcionado a sistemas baseados em *containers*. Ele potencializa o uso de regras para avaliar a importância das métricas. É composto por *Fog Nodes* trabalhadores e mestre. O *Fog Node* trabalhador é composto por três módulos: *Metrics Collector*, *Rules Updater*, e *Analyzer*. O *Analyzer* é responsável por processar os dados coletados pelo

Metrics Collector e avaliá-los de acordo com o conjunto de regras atual. Essa avaliação apoiará a decisão de transmitir ou não esse conjunto de métricas ao *Fog Node* mestre, para processamento e armazenamento adicionais. O *Rules Updater* é responsável por gerenciar o conjunto de regras que são atualizadas de acordo com os cenários observados pelo *Fog Node* mestre, ao analisar as métricas enviadas por vários *Fog Nodes* trabalhadores.

A gestão da observabilidade baseada em regras tem o objetivo de monitorar o desempenho. A solução utiliza uma topologia centralizada para coletar valores de métricas a uma taxa periódica ou acionada por eventos, utilizando modelo de comunicação *push*. Ela está preparada para monitorar as camadas de infraestrutura e de plataforma, e para fazer a filtragem de dados de gestão da observabilidade.

Esta proposta não foi avaliada quanto à escalabilidade ou ao *overhead* que injeta no sistema. Apesar de muito simplista em termos de funcionalidades, é leve, multiplataforma, focada em gestão da observabilidade e adaptável. Assim, ela atende às necessidades de orquestração do serviço de *Fog*.

6.1.9 TEEMon

O *Trusted Execution Environment* (TEE) é uma abordagem promissora para enfrentar os desafios de segurança em ambientes distribuídos como *Fog Computing*. Os TEEs melhoram a confidencialidade e a integridade do código e dos dados do aplicativo, mesmo contra invasores privilegiados com acesso físico e de *root*, fornecendo uma área de memória segura e isolada. TEEMon [137] é uma ferramenta de análise para monitorar o desempenho contínuo de aplicativos baseados em TEE. Ela fornece métricas de desempenho durante o tempo de execução, e auxilia na análise da identificação de causas de problemas de desempenho. Ela se integra ao Prometheus e ao Grafana, conhecidas ferramentas de gestão da observabilidade de código aberto, visando a uma solução holística, otimizada para sistemas implantados por meio de *containers* Docker ou Kubernetes [66]. O TEEMon consiste em quatro módulos principais: exportadores de métricas; agregador de métricas; analisador de métricas, e visualizador de métricas.

O TEEMon tem como objetivo monitorar o desempenho. A solução utiliza uma topologia centralizada para coletar valores de métricas a uma taxa periódica ou acionada por eventos, utilizando um modelo de comunicação *pull*. Está preparado para monitorar apenas a camada de infraestrutura, e para fazer a filtragem e a agregação de dados de gestão da observabilidade.

Essa solução é leve e multiplataforma, pois trabalha com TEEs de vários fornecedores e pode monitorar aplicativos baseados em Docker. É escalável, padronizada, apresenta baixo *overhead* e é focada em gestão da observabilidade. Mas, como não é adaptável, foi classificada como atendendo parcialmente às necessidades de orquestração do serviço de *Fog*. O TEEMon está disponível no GitHub [138].

6.1.10 FogMon

Os trabalhos [4] e [139] propuseram FogMon, uma ferramenta leve de gestão da observabilidade, P2P, hierárquica, baseada em um agente que executa em cada *Fog Node*, medindo e reportando sobre o uso de recursos de hardware e de QoS de rede fim-a-fim entre esses *Fog Nodes*. Ele também detecta automaticamente dispositivos IoT conectados aos *Fog Nodes*. O FogMon modifica de forma adaptativa e automática sua rede P2P com base nas condições atuais da rede para manter os invariantes de precisão e escalabilidade

da gestão da observabilidade. Ele pode lidar com *Fog Nodes* que saem e entram na rede, e conta com atualizações diferenciais de dados de gestão da observabilidade para reduzir a sobrecarga geral da rede.

FogMon tem como objetivo monitorar o desempenho e apoiar a tolerância a falhas. A solução usa uma topologia hierárquica para coletar valores de métricas a uma taxa periódica usando modelos de comunicação *pull* e *push*. Está preparada para monitorar apenas a camada de infraestrutura e fazer a agregação de dados de gestão da observabilidade.

Os autores desenvolveram um protótipo que foi avaliado em um ambiente de testes real [140]. A avaliação mediu o *overhead* de FogMon (uso de CPU, de memória e de rede pelo agente FogMon) verificando que é leve, não intrusivo e escalável. A adaptação do sistema com base na mudança de configuração também foi confirmada. Por não ser focada apenas em gestão da observabilidade, essa solução é classificada como atendendo parcialmente às necessidades de orquestração de serviços de *Fog*. O FogMon está disponível no Github [141]. Um trabalho mais recente, chamado Adaptive FogMon [142], adicionou outra camada de adaptabilidade ao FogMon [141]. Os autores implementaram um sistema especialista leve, baseado em regras, que explora os dados coletados de gestão da observabilidade para ajustar o comportamento do *Fog Node*. Ele visa a reduzir o uso de recursos e o consumo de energia no *Fog Node*. Implementou duas contramedidas que são ativadas com base no sistema de regras quando necessário: i) Seleção de Indicadores, que reduz o número de métricas coletadas; e ii) Taxa, que modifica a frequência de entrega de métricas. Quando comparado ao FogMon, o Adaptive FogMon economizou energia e recursos a um custo de maior uso de memória. Por ser baseado em FogMon, que já possui um recurso de adaptabilidade, apenas FogMon irá compor a Tabela 6.1. Adaptive FogMon também está disponível no Github [143].

6.2 Análise Geral das Soluções

Para permitir que os pesquisadores encontrem facilmente as características de cada proposta de gestão da observabilidade de *Fog* analisada, um resumo da classificação baseada na taxonomia é apresentado na Tabela 6.1. Os trabalhos são apresentados na mesma ordem em que apareceram nas seções anteriores. Assim sendo, ao analisar a Tabela 6.1 é possível perceber que as características mais prevalentes das soluções de gestão da observabilidade de *Fog* disponíveis são: ter o objetivo de monitorar desempenho, usar topologia centralizada, com um agente coletando apenas métricas de infraestrutura, enviar dados periodicamente para o servidor por meio do modelo de comunicação *push*, e atender, ao menos parcialmente, às necessidades de orquestração do serviço de *Fog*.

Existem modelos de cobrança possíveis, como o baseado em consumo, em que os usuários são cobrados por uso, ou com base em assinatura, em que os usuários pagam uma taxa mensal fixa e podem usar *Fog* de forma ampla [144]. Mas o preço e o faturamento continuam sendo um desafio em termos de sustentação de um ecossistema comercial de serviços de valor agregado, pois o modelo de negócios ainda não está definido [22], devido à falta de fornecedores comerciais de *Fog* [23]. Esses argumentos podem explicar por que nenhuma das propostas analisadas teve o objetivo de gerar faturas precisas, embora a gestão da observabilidade tenha um papel importante nessa área.

Apesar de a topologia de controle centralizada ser a mais utilizada entre os trabalhos analisados, alguns deles não verificaram a escalabilidade da solução [114, 125, 131], e isso pode aumentar o risco de falha por

Tabela 6.1: Análise comparativa das soluções de gestão da observabilidade para *Fog*.

Artigo	**PyMon** [114]	**FMonE** [59]	**Prometheus** [108]	**Osmótico** [125]	***Mobile*** [127]	***Switch*** [131]	**SCB** [10]	***Rule Based*** [136]	**TEEMon** [137]	**FogMon** [4]
Objetivos	Desempenho	Desempenho Toler. a Falhas	Desempenho	Desempenho	Desempenho Uso de Recursos	Desempenho	Desempenho Uso de Recursos Toler. a Falhas	Desempenho	Desempenho	Desempenho Toler. a Falhas
Topologia	Centralizado	Centralizado Hierárquico	Centralizado	Centralizado	Centralizado	Centralizado	Hierárquico	Centralizado	Centralizado	Hierárquico
Modelo de Comunicação	*push*	*push*/*pull*	*pull*	*push*	*push*	*push*	*push*/*pull*/Híbrido	*push*	*pull*	*push*/*pull*
Frequência	Periódica	Periódica	Periódica	Periódica	Periódica	Periódica	Eventos Periódica	Eventos Periódica	Eventos Periódica	Periódica
Camadas Monitoradas	Infra Plataforma	Infra Plataforma Serviço	Infra Plataforma	Infra Plataforma	Infra Plataforma Serviço	Infra Plataforma Serviço	Infra	Infra Plataforma	Infra	Infra
Domínios de Instrumentação	Métricas	Métricas	Métricas	Métricas	Métricas *Logs*	Métricas	Métricas	Métricas	Métricas	Métricas
Processamento de Dados	Agregação	Filtragem	Agregação	✗	Filtragem Agregação	Gerenc. de Eventos	✗	Filtragem	Filtragem Agregação	Agregação
Intrusividade	Ativa	Ativa	Ativa	Ativa	Ativa	Ativa Passiva	✗	Ativa	Ativa	Ativa Passiva
Escalabilidade	✗	✓	✓	✗	✓	✗	✓	✗	✗	✓
Overhead	✓	✓	✗	✗	✗	✗	✓	✓	✓	✓
Adaptabilidade	✗	✗	✓	✗	✓	✓	✓	✓	✗	✓
Integração	✗	✗	✓	✗	✓	✗	✗	✗	✓	✗
Necessidades da Orquestração	✗	✗	✓	✗	✓	✓	✓	✓	∂	∂

✓ denota que o item é implementado/atendido;
∂ denota que o item é parcialmente implementado/atendido;
✗ denota que o item não é implementado/atendido;

esgotamento de recursos no servidor, em caso de grande volume de componentes sendo monitorados, ou no caso de um cenário em que haja rajadas de dados. A *Fog* é um paradigma distribuído. É natural pensar que a topologia de controle adequada também seria um esquema distribuído, mas essa suposição não é confirmada pelas propostas de gestão da observabilidade de *Fog* disponíveis. De acordo com Ward & Barker [91], topologias distribuídas possuem melhorias de escalabilidade inerentes em relação às centralizadas. Mas trazem um conjunto de desafios diferentes, incluindo inicialização do sistema, processo de pesquisa de *Fog Node*, e replicação de dados. As soluções de gestão da observabilidade com topologias distribuídas devem primeiro superar esses desafios e riscos, mas podem se tornar mais lentas e complicadas do que as soluções centralizadas.

A maioria dos trabalhos analisados coleta dados de gestão da observabilidade periodicamente, e dentre eles, algumas propostas utilizam apenas o método *push* como modelo de comunicação [125, 127, 131, 136]. Essa combinação pode levar a cenários em que uma grande quantidade de dados seja injetada no sistema, dependendo do número de dispositivos e serviços monitorados. Nesses cenários, os canais de comunicação e o servidor podem ficar sobrecarregados, potencialmente causando ineficácia, perda de dados ou até mesmo indisponibilidade do sistema de gestão da observabilidade [137]. É fundamental verificar se essas propostas são escaláveis e se o *overhead* que injetam no sistema é o menor possível para que possam lidar com os cenários mencionados.

O aumento da observabilidade e o gerenciamento de *logs* e de *traces* (Seção 3.4) são temas recentes no contexto de *Fog*. Apenas uma das dez propostas analisadas coleta *logs* [127], e nenhuma delas coleta *traces*. *Logs* e *traces* estão relacionados à gestão da observabilidade *white-box*, e podem auxiliar na avaliação do estado interno dos serviços, visando antecipar mau funcionamento, depurar problemas já detectados e verificar se tudo está funcionando corretamente em momentos específicos, por exemplo, após atualizar o serviço com uma nova versão [145]. Unificar o gerenciamento do ciclo de vida dos dados dos domínios de instrumentação (métricas, *logs* e *traces*) pode reduzir o esforço de manter vários fluxos de dados [1]. Por outro lado, é fundamental considerar a natureza heterogênea dos dados de instrumentação em termos de frequência de geração, volume de dados e consumo (o que determina o tipo de armazenamento e consultas do usuário), equilibrando os benefícios de usar um mesmo mecanismo de gerenciamento de dados com os riscos de um sistema de gestão da observabilidade mais complexo e frágil [65].

Em relação às necessidades de um orquestrador de serviço de *Fog*, cujos requisitos foram descritos na Seção 3.6, algumas propostas analisadas as atenderam plenamente [10, 120, 127, 131, 136]. Esta é a resposta para a QP2 da revisão da literatura empreendida (Seção 5.1): "Quais soluções de gestão da observabilidade estão preparadas para compor um orquestrador de serviço de *Fog*?". Essas propostas são leves e multiplataforma, focadas na gestão da observabilidade e aceitam mudanças de configuração em tempo de execução, permitindo o gerenciamento a partir do orquestrador de serviços de *Fog*. TEEMon [137] e FogMon [4, 139, 142], embora categorizados como atendendo parcialmente às necessidades de orquestração, devem ser considerados e acompanhados, pois podem evoluir e alterar sua categorização. Para ajudar os pesquisadores a visualizar as vantagens e limitações de cada solução analisada, a Tabela 6.2 resume essas informações.

6.3 Considerações Finais

Neste capítulo, foi realizada uma análise do estado da arte em soluções de gestão da observabilidade para *Fog Computing*, utilizando a taxonomia proposta no capítulo anterior. Dez soluções foram avaliadas e classificadas, com base nos domínios e categorias da taxonomia. A análise revelou que a maioria das soluções possui como objetivo monitorar o desempenho, utiliza topologias centralizadas e coleta dados periodicamente por meio do modelo de comunicação *push*. Também foi constatado que apenas uma das soluções analisadas implementa o gerenciamento de *logs*, e que nenhuma das soluções fazia gestão de *traces*.

Assim, no próximo capítulo será definido o ciclo de vida dos dados de observabilidade em *Fog*, um processo essencial para maximizar a eficiência da gestão de dados neste ambiente.

Tabela 6.2: Vantagens e limitações das soluções de gestão da observabilidade para *Fog*.

Artigo	**Ano**	**Cód. Aberto**	**Vantagens**	**Limitações**
PyMon [114]	2017	Sim	Simples, focado em gestão da observabilidade, leve	Conjunto limitado de funções
FMonE [59]	2018	Sim	Modularidade, arquitetura flexível, ambiente de testes real	Não adaptativo, específico para Marathon [117]
Prometheus [108]	2018	Sim	Extenso conjunto de funções, modularidade, padronizado	Ponto único de falha no lado do servidor.
Osmótico [125]	2018	Não	Simples	Conjunto limitado de funções, pode não escalar
Mobile [127]	2018	Não	Derivado do Sensu [128]	Demanda *Fog Nodes* potentes, pode não escalar
Switch [131]	2018	Sim	focado em gestão da observabilidade, leve	Conjunto limitado de funções, pode não escalar
SCB [10]	2019	Não	Simples, focado em gestão da observabilidade, leve	Infraestrutura apenas, sem processamento de dados
Rule Based [136]	2019	Não	Adaptativo, configuração flexível	Conjunto limitado de recurso, pode não escalar
TEEMon [137]	2020	Sim	O mesmo que o Prometheus	Pode não escalar
Fogmon [139]	2022	Sim	Adaptativo, perto de estar pronto para a orquestração	Apenas infraestrutura

Capítulo 7

Ciclo de Vida dos Dados de Observabilidade em *Fog*

Neste capítulo, será definido o Ciclo de Vida dos Dados de Observabilidade em *Fog Computing*, o qual é um processo essencial para maximizar a eficiência e a eficácia da gestão de dados em ambientes de *Fog*. Na Seção 7.1, são apresentadas as duas fases do ciclo de vida, iniciando pela fase de Coleta de Dados, a qual é crucial para garantir que os dados de observabilidade sejam capturados e armazenados de forma adequada, minimizando o *overhead* em dispositivos com recursos limitados. A fase seguinte é a fase de Análise de Dados, que é fundamental para fornecer uma visão abrangente e atualizada da infraestrutura e dos serviços em execução em *Fog*, permitindo uma tomada de decisão informada e oportuna. Na Seção 7.2, é definido o Indicador de Nível de Observabilidade, uma forma de se medir as variações de observabilidade de um sistema em *Fog*. Na sequência, a Seção 7.3 apresenta uma forma de medir o *overhead* da observabilidade, destacando a importância de equilibrar a coleta e a análise de dados com o consumo de recursos. Por fim, a Seção 7.4 resume os principais pontos apresentados neste capítulo.

7.1 Ciclo de Vida dos Dados de Observabilidade em *Fog*

Para obter informações valiosas de cada domínio de instrumentação e aumentar a observabilidade de uma aplicação em execução em um ambiente de *Fog Computing*, é necessário estar ciente das etapas do ciclo de vida dos dados de observabilidade em *Fog* (ODLC, do inglês Observability Data Life Cycle) [146], conforme ilustrado na Figura 7.1. Nessa figura, é possível notar que o ciclo de vida é dividido em seis etapas, as quais são: 1. Coleta; 2. Armazenamento IoT; 3. dados transmitidos para *Fog*; 4. armazenamento *Fog*; 5. análise de dados e visualização; 6. Armazenamento *Cloud*. As três primeiras etapas compreendem a fase de Coleta de Dados do ciclo de vida. As três últimas etapas formam a fase de Análise de Dados.

7.1.1 Coleta

Os dados são coletados na etapa inicial do ciclo de vida dos dados de observabilidade de *Fog*. Isso pode acontecer de várias maneiras, dependendo do domínio de instrumentação em questão. Métricas podem ser adquiridas a partir do sistema operacional dos computadores de bordo dos *SmartTrucks*, utilizando

Figura 7.1: Ciclo de vida dos dados de observabilidade em *Fog*.

chamadas de sistema que reportam a quantidade de recursos disponíveis (por exemplo, CPU, memória, armazenamento em disco). Plataformas como sistemas de gerenciamento de *containers*, geralmente, também reportam métricas de uso de recursos. Finalmente, a aplicação pode realizar um processamento específico para relatar, por exemplo, o tempo médio de resposta ou o número de requisições entregues em um período. *Logs* são escritos de acordo com o fluxo de eventos específico que foi instrumentado para ser registrado em texto. Por exemplo, eventos bem-sucedidos, como o registro das coordenadas geográficas de *SmartTruck* a cada segundo de operação; e falhas, como pilha de chamadas de função no caso de exceções capturadas por ausência da rede 5G. Quando previamente instrumentadas, chamadas específicas de APIs podem criar *traces* que registram a sequência e a demora de cada chamada de serviço.

Recursos computacionais como ciclos de CPU e espaço de memória devem ser usados para coletar dados de observabilidade. Assim sendo, quanto maior o número de métricas coletadas ou linhas escritas em um arquivo de *log*, por exemplo, maior será o *overhead* de observabilidade adicionada ao ambiente. Um ambiente de *Fog* é composto por dispositivos com recursos restritos. Portanto, o *overhead* causado pela observabilidade dos coletores de dados deve ser baixo.

7.1.2 Armazenamento IoT

Nesta etapa, os dados de observabilidade foram coletados e agora estão armazenados no dispositivo, aguardando transmissão ou remoção. Como esses dados são geralmente imutáveis, mas o conjunto aumenta com o tempo [147], a tendência é que o volume de dados armazenados se torne cada vez maior. Uma política de remoção de dados deve estar em vigor para evitar que os recursos de armazenamento se esgotem. Por exemplo, toda vez que os dados são transmitidos para *Fog*, eles devem ser deletados para liberar espaço para mais dados serem armazenados. Essa remoção pode ser apenas parcial se uma janela mínima de dados for necessária para algum processamento local. O período em que um dispositivo pode lidar com os dados de observabilidade armazenados dependerá de vários fatores, tais como o volume de dados por unidade de coleta, a frequência de geração, e o espaço de armazenamento disponível reservado para o sistema.

Dessa forma, embora as métricas possam ser estáveis em relação ao volume de dados, *logs* e *traces* apresentam uma variabilidade mais significativa [1]. Essa característica torna difícil monitorar e atuar tempestivamente quando o espaço de armazenamento está acabando. Além da exclusão local de dados, uma alternativa de atuação pode ser informar ao servidor de observabilidade que há risco de perda de dados, e verificar se as circunstâncias dinâmicas permitem que esses dados sejam imediatamente transmitidos e apagados.

7.1.3 Transmissão

A observabilidade pode permitir uma tomada de decisão adequada e oportuna. Embora seja possível tomar algumas decisões mais simples localmente por um único dispositivo, decisões críticas devem ser feitas por um processo que possa avaliar um volume maior de dados provenientes de diferentes componentes, proporcionando uma visão mais abrangente do sistema. Portanto, os dados coletados da Camada IoT devem ser transmitidos para a Camada *Fog*, onde um nó mais rico em recursos os armazenará, permitindo uma análise de dados mais abrangente.

Como a incerteza da rede é uma característica de *Fog Computing*, uma solução adequada de observabilidade de *Fog* deve lidar com interrupções temporárias da rede. Idealmente, deve modular de forma adaptativa o fluxo de dados de observabilidade, considerando a quantidade de dados a ser transmitida, seu tipo (métricas, *logs*, *traces*) e a largura de banda disponível. As conexões de rede utilizadas pela aplicação para receber e responder a solicitações de usuários podem ser as mesmas usadas pelo fluxo de dados de observabilidade [148]. Assim, transmitir grandes volumes de dados de observabilidade pode sobrecarregar a rede, particularmente, em ambientes de *Fog* com largura de banda limitada. Isso pode aumentar a latência tanto para os dados de observabilidade quanto para o tráfego da aplicação, impactando negativamente o QoS, e potencialmente levando a violações de SLA. Um processo adaptativo pode estar em vigor para definir a quantidade de dados que devem ser transferidos dos dispositivos, selecionando quais domínios de instrumentação serão incluídos em cada transmissão, e o período ao qual os dados coletados se referirão.

7.1.4 Armazenamento *Fog*

Os *Fog Nodes* têm potencialmente mais recursos do que os dispositivos IoT [35]. Devido a isso, é na Camada de *Fog* que os dados de observabilidade de vários dispositivos IoT são armazenados para uma atuação, e tomada de decisão rápidas. Nesta etapa, os dados recebidos podem ser pré-processados para obter informações contextuais, por exemplo, a adição do identificador do dispositivo. No entanto, os *Fog Nodes* não são tão ricos em recursos quanto os dispositivos de *Cloud* [35]. Portanto, o volume de dados deve ser limitado a um volume que esses dispositivos possam manejar.

Métricas podem ser vistas como séries temporais, e um TSDB deve ser usado para armazená-las de forma ideal. No entanto, *logs* e *traces* são estruturados de forma diferente e se beneficiarão de outras soluções de armazenamento. Karumuri *et al.* [1] analisaram dados de observabilidade em um ambiente de Cloud e consideraram que *logs* e *traces* se beneficiariam de armazenamento baseado em índice invertido devido ao tipo de consultas que, geralmente, são feitas para recuperar informações a partir deles. Portanto, um serviço de ingestão de dados de observabilidade em *Fog* deve considerar os requisitos de dados de cada domínio de instrumentação (Tabela 3.1), ao mesmo tempo em que permite que análises cruzadas sejam realizadas.

7.1.5 Análise de Dados e Visualização

Uma vez que os dados de observabilidade estão disponíveis em *Fog*, é possível consultá-los, tomar decisões e efetuar ações de acordo com elas, entregando o valor que se espera de *Fog Computing*. Localizado na borda da rede, um serviço de *Fog* pode fornecer respostas mais rápidas para dispositivos IoT e usuários

finais. Além disso, estando localizado entre as camadas de IoT e de *Cloud*, previne o congestionamento na rede de *Cloud*.

Dessa forma, os dados de observabilidade tendem a fornecer respostas mais relevantes quando são consultados tão logo sejam disponibilizados, o que significa que a maioria das consultas e análises utiliza dados mais recentes (menos do que 24 horas) [147]. Assim, é essencial garantir acesso rápido a essa janela de tempo. Além disso, para economizar recursos e continuar recebendo dados de IoT, é crucial fornecer mecanismos automatizados para enviar os dados fora desse intervalo para armazenamento a longo prazo em *Cloud*.

7.1.6 Armazenamento *Cloud*

Cloud é o ambiente apropriado para armazenar grandes volumes de dados e executar modelos mais robustos de processamento de dados, como a análise histórica de dados de observabilidade [149]. Os dados podem ser exportados automaticamente do sistema de armazenamento de *Fog* quando eles permanecem fora da janela de tempo pré-configurada, por exemplo, uma semana após o seu tempo de coleta. Esses dados podem ser movidos para *Cloud*, mantendo um volume predeterminado de dados no armazenamento de *Fog*, e ajudando a garantir um baixo tempo de resposta nas consultas.

7.1.7 Considerações Gerais sobre o Ciclo de Vida dos Dados de Observabilidade

Como visto anteriormente, as etapas do ciclo de vida apresentadas são as mesmas para cada domínio de instrumentação. No entanto, os domínios de instrumentação têm características diferentes (veja a Tabela 3.1), e contribuem de maneiras distintas para a observabilidade do sistema, dependendo das circunstâncias. Por exemplo, em casos nos quais as métricas coletadas mostram um desempenho adequado do dispositivo, mas há problemas de tempo de execução identificados, pode-se coletar temporariamente apenas *logs* e *traces*, visando acelerar a descoberta da causa raiz do problema enquanto reduz o *overhead* do sistema relacionado à observabilidade. Um ambiente com recursos limitados como *Fog* pode se beneficiar desse comportamento adaptativo.

O ciclo de vida dos dados de observabilidade pode ser interpretado como a convergência de múltiplos fluxos de dados oriundos dos domínios de instrumentação. Em cada um desses fluxos, desenvolvem-se as etapas pormenorizadas nas seções precedentes. A quantidade de dados em cada fluxo é suscetível a variações, da mesma forma que o número de fluxos de dados ativos em um dado momento é variável. Ademais, observa-se um limite no volume total de dados transmissíveis, o que, de forma dinâmica, pode demandar a imposição de restrições sobre um ou mais fluxos internos.

7.2 Indicador de Nível da Observabilidade

É possível conectar os três domínios de instrumentação considerando o momento em que cada informação é gerada. Quando é viável relacionar dois ou três deles na mesma análise, surgem mais oportunidades de atuação. Os dados de *log*, por exemplo, quando agrupados com as métricas coletadas durante o mesmo período, fornecem uma inspeção mais abrangente dos problemas em tempo de execução, consolidando

simultaneamente visões externas e internas do sistema. Um *trace* pode ser visto como a decomposição de uma métrica de tempo de resposta, permitindo a identificação dos componentes nos quais uma melhoria no tempo de processamento ou de comunicação pode resultar em um tempo de resposta final mais curto.

Até a escrita desta tese, não foi encontrado trabalho semelhante na literatura que tenha definido uma forma objetiva de medir o nível de observabilidade de um sistema, ou de toda a infraestrutura dinamicamente. Assim, inicia-se essa definição a partir da quantidade de domínios de instrumentação disponíveis, e que são alvos da atuação do ODLC. Ter mais domínios de instrumentação disponíveis significa um maior nível de observabilidade. Além do valor independente de cada domínio, existe um valor adicional na análise cruzada entre domínios devido às suas interações sinérgicas [150], ou seja, quando dois ou mais fatores atuam como causas de um resultado particular. Este efeito é popularmente conhecido como "o todo é maior do que a soma das suas partes".

Portanto, em vez de ter uma fórmula em que o nível de observabilidade de um sistema é determinado apenas pelo número de domínios de instrumentação, cujos dados estão disponíveis para análise, como na Equação 7.1:

$$Observabilidade = Metricas + Logs + Traces \tag{7.1}$$

onde:

$$Metricas, Logs, Traces = \begin{cases} 1, & \text{se os dados do domínio de instrumentação estão disponíveis para análise;} \\ 0, & \text{caso contrário.} \end{cases} \tag{7.2}$$

Adicionou-se as interações sinérgicas entre eles também, como apresentado na Equação 7.3:

$$Observabilidade = Metricas + Logs + Traces + (MetricasXLogsXTraces) \tag{7.3}$$

em que X é um operador que filtra os dados para cada par diferente dos domínios de instrumentação (ID) disponíveis, e retorna o subconjunto de cada ID que corresponde a um período específico. A expressão (IDi X IDj X IDk) é a mesma que (IDi X IDj)+(IDi X IDk)+(IDj X IDk). E (IDi X IDj) pode assumir um dos seguintes valores:

$$IDi\ X\ IDj = \begin{cases} 1, & \text{se ambos os domínios de instrumentação são iguais a 1 (Equação 7.2), e} \\ & \text{o filtro de tempo aplicado retorna um conjunto não vazio para ambos.} \\ 0, & \text{caso contrário.} \end{cases} \tag{7.4}$$

Considerando o exemplo do cenário motivador descrito na Seção 3.2, o Mobile IoT-Roadbot gerou apenas métricas e *logs* quando estava em execução nas ruas de Brimbank, Austrália. Assim, usando a Equação 7.3, temos que Observabilidade = Metricas + Logs + (Metricas X Logs), resultando em 1 + 1 + 1 = 3. Esse é o valor máximo do nível de observabilidade para um sistema que consegue gerenciar dois domínios de instrumentação. Mas, de acordo com a Equação 7.4, o valor adicional referente às interações sinérgicas apenas é somado caso ambos os domínios tenham dados para o mesmo período de tempo. Ou seja, nos momentos em que haja falha de coleta, de registro ou de envio de dados de um dos domínios, esse

valor adicional é 0, de acordo com a Equação 7.4. Nos momentos do dia em que apenas um dos domínios esteja disponível para análise (por exemplo, das 14h00 às 15h00 de um dia de execução, o sistema coletou apenas métricas, pois o módulo de coleta de *logs* apresentava falha), o valor do nível de observabilidade para aquela hora do dia vai ser 1, que é o máximo valor que um sistema pode alcançar quando gerencia apenas um domínio de instrumentação. Por outro lado, um sistema que consiga gerenciar os três domínios de instrumentação tem um nível máximo de observabilidade de 6, de acordo com a Equação 7.3.

Essa definição mostra que, para aumentar o nível de observabilidade de um sistema, é importante não apenas coletar as informações dos domínios de instrumentação e analisar cada conjunto de dados isoladamente. Também é relevante estar preparado para aprender com suas interações para gerar mais valor a partir dos mesmos conjuntos de dados disponíveis. Outro ponto importante é que o nível de observabilidade não é uma medida estática, já que é dependente da disponibilidade dos dados e essa disponibilidade pode ser afetada por diversos fatores ao longo do tempo. A utilidade de medir o nível de observabilidade de um sistema é poder selecionar os momentos mais adequados para investigar a causa-raiz de um problema, que são os momentos em que o nível apresenta valor máximo.

7.3 *Overhead* da Observabilidade

A observabilidade não é um fim em si mesma. Aumentar a observabilidade de um sistema é um meio de melhorar o funcionamento e a disponibilidade do sistema, ajudando a garantir que os SLAs sejam atendidos. O valor que um nível aumentado de observabilidade pode oferecer deve ser equilibrado com a quantidade de recurso computacional necessário (o *overhead*) para alcançar esse nível. Embora óbvio em um sentido genérico, essa declaração ganha mais relevância em um ambiente de *Fog Computing* devido às características de restrição de recursos e de incerteza de rede. Esse valor pode ser modelado conforme indicado na Equação 7.5:

$$Valor = \frac{Observabilidade}{Overhead} \tag{7.5}$$

em que *Overhead* é um número diferente de zero e menor que cem, representando a porcentagem de recursos (ou seja, CPU, memória, largura de banda de rede) consumidos ao executar o ciclo de vida descrito na Seção 7.1. Quanto maior o *overhead*, menor poderá ser o valor do aumento da observabilidade para a aplicação, considerando que a restrição de recursos é uma característica de *Fog Computing*.

Assim sendo, a observabilidade de um sistema está diretamente relacionada à cardinalidade de ID ($|ID|$), além das interações sinérgicas $X(ID)$, como visto nas Seções 3.4 e 7.2. Ter $ID = \{ID_{Met}, ID_{Log}, ID_{Tra}\}$ significa que o ODLC é capaz de gerenciar métricas, *logs*, e *traces* em todo o sistema. Devido à incerteza de conectividade e à restrição de recursos dos ambientes de *Fog*, parte do conjunto de dados de observabilidade pode ser perdida quando não há recursos disponíveis, como armazenamento, largura de banda de rede, bateria do dispositivo, etc. Para lidar com isso, é útil definir pesos (W_i) para cada domínio de instrumentação, representando a importância desse domínio em comparação com os outros em um determinado momento. Como uma propriedade de dados, é importante garantir que a soma total dos pesos seja igual a 100%, ou seja, que $\sum W_i = 1$, quando pelo menos um dos pesos for acima de 0%. Os pesos podem ser usados para tomar decisões adaptativas sobre quais dados de domínio serão coletados ou transmitidos, caso os

recursos se esgotem, ou em outras situações específicas. Por exemplo, um peso de 0% significa que dados daquele domínio não devem ser gerenciados naquele momento e sua coleta está suspensa temporariamente. Pesos acima de 0% determinarão a ordem de prioridade, se necessário, podendo significar uma estratégia de amostragem dos dados, ou seja, uma coleta de apenas fração dos dados gerados para aquele ID, usando o peso para definir o percentual de amostragem. Além disso, os pesos podem variar de acordo com as necessidades, ou seja, definir pesos diferentes para os domínios quando o sistema estiver em uma operação regular e após um erro massivo ter ocorrido.

Usando a definição de observabilidade da Equação 7.3 na Equação 7.5, mas substituindo *ID* pelo conjunto de pesos $W = \{W_{Met}, W_{Log}, W_{Tra}\}$, tem-se um cálculo do valor de executar o OLDC que reflete a variação dinâmica dos pesos. Separando a equação resultante para isolar o peso de cada *ID* com seu *overhead* respectivo, pode-se comparar o quanto cada componente agrega no resultado final, gerando a seguinte equação resultante:

$$Valor = \frac{W_{Met}}{Over_{Met}} + \frac{W_{Log}}{Over_{Log}} + \frac{W_{Tra}}{Over_{Tra}} + \frac{X(ID)}{Over_{X(ID)}} \tag{7.6}$$

Ao executar o ciclo de vida dos dados de observabilidade, o *overhead* de cada domínio de instrumentação, a saber, $Over_{Met}$, $Over_{Log}$ e $Over_{Tra}$ para métricas, *logs* e *traces*, respectivamente, pode ser modelado como uma função da quantidade de recursos consumidos para gerar, armazenar e transmitir as informações desse domínio. Um modelo sugerido é atribuir a porcentagem máxima de consumo de recursos entre os principais recursos de hardware, tais como CPU, memória e largura de banda da rede, como em $Over_{Met} = \max\{\%CPU_{Met}, \%Mem_{Met}, \%Net_{Met}\}$ * 100. Por simplicidade, considerou-se que os três primeiros fatores da Equação 7.6, juntos, representem a fase de coleta de dados do OLDC. O *overhead* das interações sinérgicas $Over_{X(ID)}$ pode ser modelado da mesma forma que os anteriores, mas em relação aos recursos consumidos para armazenar e analisar os dados em *Fog Computing* e transmiti-los para *Cloud* (etapas 4 a 6 do ODLC, que representam a fase de análise de dados).

$$Valor = \overbrace{\frac{W_{Met}}{Over_{Met}} + \frac{W_{Log}}{Over_{Log}} + \frac{W_{Tra}}{Over_{Tra}}}^{\text{Fase de Coleta de Dados}} + \overbrace{\frac{X(ID)}{Over_{X(ID)}}}^{\text{Fase de Análise de Dados}} \tag{7.7}$$

Em um ambiente dinâmico de *Fog Computing*, com restrição de recursos e incerteza de rede, a tomada de decisão pode ser mais eficaz utilizando uma fórmula como a Equação 7.6, que equilibra a importância relativa de cada domínio de instrumentação com o esforço necessário para coletar e analisar seus dados.

7.4 Considerações Finais

Este capítulo definiu o Ciclo de Vida dos Dados de Observabilidade (ODLC) na *Fog Computing*, um processo que inicia na coleta dos dados de observabilidade nos dispositivos da Camada IoT, passa pelo processamento, armazenamento e análise dos dados através do *continuum* IoT-*Fog*-*Cloud*, e é fundamental para maximizar a eficiência da gestão de dados e a tomada de decisão rápida e informada. Além disso, definiu-se o Indicador de Nível de Observabilidade, que é útil para apontar os períodos em que há mais informações

de observabilidade disponíveis, tornando mais efetivas as análises de causa raiz dos problemas identificados. Por fim, foi apresentado como medir o *overhead* da observabilidade, destacando a importância de equilibrar a coleta e análise de dados com o consumo de recursos. A definição de pesos para cada domínio de instrumentação (métricas, *logs* e *traces*) foi apresentada para auxiliar na tomada de decisões adaptativas sobre quais dados serão coletados ou transmitidos.

No próximo capítulo, será apresentada a arquitetura FogObserver, uma proposta para sistemas de gestão da observabilidade em *Fog*, que visa a endereçar os desafios encontrados na literatura e atender às necessidades da orquestração de serviços.

Capítulo 8

FogObserver - Arquitetura de Sistemas de Gestão da Observabilidade em *Fog*

Este capítulo descreve FogObserver, que é uma proposta de arquitetura para sistemas de gestão da observabilidade que endereça vários desafios encontrados na literatura (Seção 3.6), e que atende às necessidades da orquestração de serviços em *Fog* (Seção 3.7). Para isso, este capítulo está estruturado em quatro seções. A Seção 8.1 apresenta uma descrição de alto nível de FogObserver, descrevendo os componentes da arquitetura proposta: *Collector*, *Transformer* e *Manager*. Na sequência, a Seção 8.2 propõe um *framework* que adiciona adaptabilidade à arquitetura FogObserver. A Seção 8.3 compara FogObserver e o *framework* de adaptabilidade com a literatura. Por fim, a Seção 8.4 apresenta uma visão geral das principais características da arquitetura proposta neste capítulo.

8.1 FogObserver

FogObserver é uma arquitetura que viabiliza a coleta, o gerenciamento e a entrega de dados de gestão da observabilidade referentes aos três principais domínios de instrumentação: métricas, *logs* e *traces*. A arquitetura é extensível e aceita a adição de outros domínios de instrumentação sem demandar alterações importantes. FogObserver tem o objetivo de facilitar o desenvolvimento de soluções abrangentes, atualizadas e expansíveis [1], e que possam lidar com os desafios característicos da plataforma, apresentados na Seção 3.6.

Assim sendo, FogObserver visa a preencher algumas lacunas encontradas nos sistemas de gestão da observabilidade de *Fog* disponíveis na literatura e apresentados no Capítulo 6. Principalmente, a gestão simultânea de vários domínios de instrumentação dentro de um ciclo de vida de dados de observabilidade, conforme definido no Capítulo 7. Embora usando o mesmo padrão e se utilizando dos mesmos canais para geração e envio de dados, cada domínio de instrumentação tem configurações independentes, permitindo ao sistema de gestão da observabilidade definir quais são os dados de interesse em cada momento, alterando o volume a ser enviado, a frequência de envio, e o modelo de comunicação (por exemplo, *push*, *pull*), de forma a lidar com os picos de processamento temporários, tanto do componente sendo monitorado quanto do sistema de gestão da observabilidade.

A Figura 8.1 mostra um exemplo da organização dos componentes da arquitetura de gestão da observabilidade proposta em um ambiente de *Fog Computing*. Os componentes do sistema são: *Collector*, *Transformer* e *Manager*, os quais são detalhados nas seções a seguir.

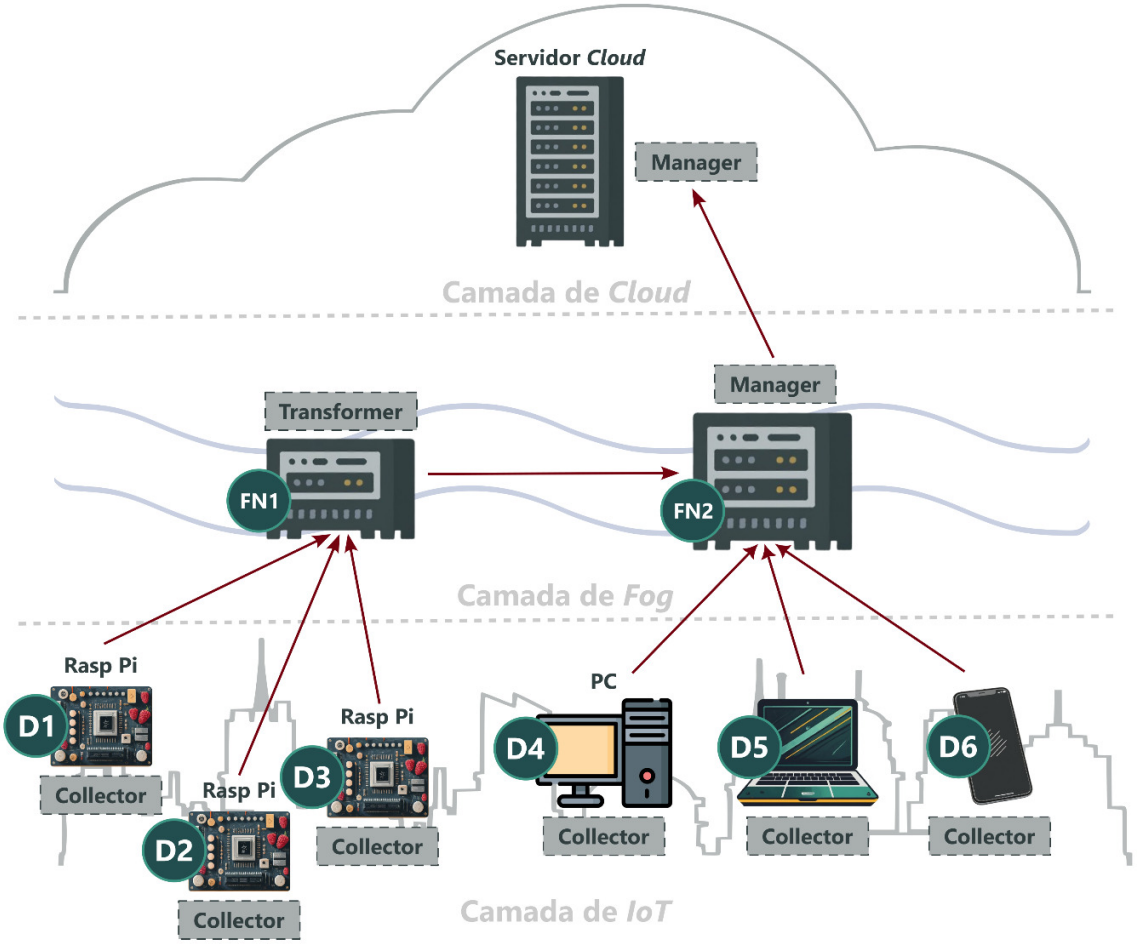


Figura 8.1: Distribuição dos componentes na arquitetura FogObserver.

8.1.1 *Collector*

O *Collector* é o componente da FogObserver que implementa a função de Observação (Seção 3.3), responsável por coletar as informações atualizadas dos serviços sendo monitorados. Essas informações podem ser os valores de métricas de desempenho, *logs* de atividades ou *traces*. As métricas podem estar relacionadas a componentes de infraestrutura (por exemplo, CPU, rede, memória, etc), de plataforma (por exemplo, *containers*, banco de dados, etc) ou de serviço em execução no dispositivo IoT.

O *Collector* armazena localmente as informações. Os *logs* são armazenados como texto puro, sem uma formatação específica. Como os dispositivos IoT podem ser limitados em recursos, o armazenamento utiliza

um tamanho máximo por domínio de instrumentação, definido como um parâmetro do sistema de gestão da observabilidade. Assim, sempre que esse limite for alcançado, os novos dados coletados sobrescreverão os dados mais antigos.

A configuração irá indicar o endereço do componente de destino para o qual o *Collector* irá enviar os dados coletados, assim como o modelo de comunicação padrão (por exemplo, *push* ou *pull*). Além disso, ele indicará também quais serviços devem ser monitorados no dispositivo (por exemplo, o próprio dispositivo, o ambiente de virtualização, serviços específicos), qual o conjunto de métricas a coletar, assim como quais os domínios de instrumentação devem ser enviados. Essa configuração pode ser alterada de forma dinâmica pelo componente *Manager*, quando eventos indicarem essa necessidade (por exemplo, *Fog Node* do *Manager* com pouco espaço em disco ou com percentual de uso da CPU acima de um limite pré-definido), ou quando o próprio dispositivo em que o *Collector* executa estiver com escassez de recursos. A Figura 8.2 mostra uma visão mais detalhada dos módulos que compõem o *Collector*, descritos nos itens a seguir:

- **Observadores** – os observadores são os responsáveis pela coleta de dados de telemetria. Existe apenas um *Collector* por dispositivo, instalado pelo orquestrador quando o dispositivo é aceito na infraestrutura, porém pode haver mais de um observador por *Collector*. O observador pode ser exclusivo de apenas um serviço monitorado, por exemplo, o hardware do *SmartTruck*, ou o próprio aplicativo do Mobile IoT-RoadBot. Ou pode ser um observador compartilhado, que coleta dados de um domínio de instrumentação, independente de qual serviço gere o dado;
- **Parâmetros** – cada *Collector* tem parâmetros que determinam seu funcionamento. Os parâmetros podem ser alterados em tempo de execução por meio de chamadas do Módulo de Comunicação. O modelo de comunicação, os domínios de instrumentação coletados, a frequência de envio dos dados são exemplos de parâmetros que auxiliam o sistema a se adaptar à dinâmica do ambiente de *Fog Computing*;
- **Armazenamento Local** – os observadores coletam as informações de observabilidade e as registram no espaço de armazenamento local, que é gerenciado pelo *Collector*. Essa capacidade é de grande importância quando os dados são gerados por sensores externos, acoplados ou não ao dispositivo. Quando os dados já forem criados inicialmente no dispositivo, como arquivos de *log*, não há necessidade de providenciar forma adicional de armazenamento;
- **Módulo de Comunicação** – cada *Collector* disponibiliza um Módulo de Comunicação para trocar informações de controle com outros componentes autorizados. Por exemplo, informar que houve um evento crítico, ou receber o comando para diminuir a frequência do envio de dados, caso o canal de comunicação esteja congestionado.

8.1.2 *Transformer*

O *Transformer* é o componente do sistema de gestão da observabilidade que implementa a função de processamento de dados (Seção 3.3). Ele atua nos dados coletados e implementa funções de agregação, filtragem e transformação de dados, quando necessário. Assim sendo, este componente é opcional, e pode não existir

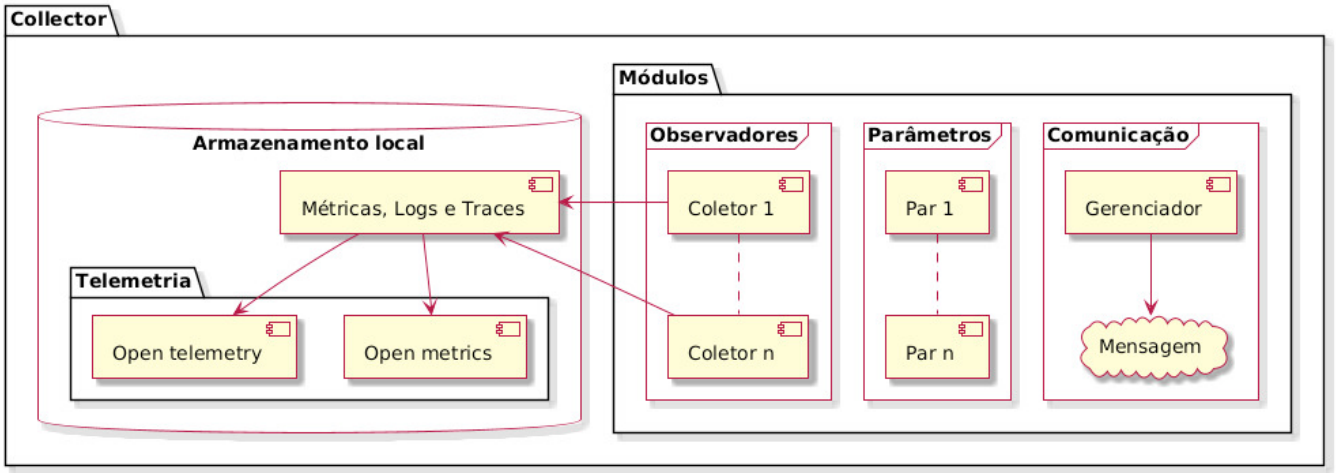


Figura 8.2: Módulos que constituem o componente *Collector*.

quando o caso de uso não necessitar de ações de processamento de dados, antes de os dados serem enviados para o componente *Manager*.

Em casos de uso que demandem o *Transformer* – como o envio de médias de temperatura ambiente, calculadas a partir do envio de medições de dezenas de sensores – e que o dispositivo tenha maior limitação em recursos, executará nele apenas o componente *Collector*, que terá configurado como componente de destino um componente *Transformer*, que estará em um *Fog Node*. Em dispositivos mais robustos e cujo caso de uso demande processamento de dados, o *Transformer* pode executar simultaneamente ao componente *Collector* e esse componente *Collector* terá como destino de envio dos dados o próprio dispositivo.

O componente *Transformer* viabiliza a implementação da topologia de controle hierárquica (Seção 5.2.2). Ele pode ser utilizado mesmo que não haja necessidade real de processamento de dados. Assim, ele pode ser colocado em *Fog Nodes* mais ricos em recursos, e estrategicamente distribuídos na infraestrutura de *Fog*, para servirem como estágio intermediário de armazenamento e, portanto, como uma estratégia de tolerância a falhas. A Figura 8.3 mostra uma visão mais detalhada dos módulos que compõem o *Transformer*, descritos nos itens a seguir:

- **Coletor de Dados** – é o módulo responsável por solicitar e receber os dados dos *Collectors*, enviando-os para armazenamento local;
- **Componentes** – um *Transformer* pode atender a dezenas de *Collectors* simultaneamente. Esse módulo gerencia os identificadores de acesso (por exemplo, endereços de rede, portas de acesso, etc), assim como relaciona as funções de processamento que atuarão nos dados recebidos de cada *Collector*;
- **Funções** – esse módulo contém as funções que serão aplicadas aos dados recebidos. Soma, média, máximo e mínimo são exemplos de funções matemáticas que podem ser aplicadas. Estes resultados podem ser adicionados aos dados originais, criando assim visões consolidadas das medições, além de manter os dados analíticos. Outra possibilidade é descartar os dados originais e manter apenas os valores consolidados. Essa definição acerca do gerenciamento de dados, assim como acerca das funções que serão aplicadas, estão registradas no cadastro dos componentes, que é enviado e atualizado pelo *Manager*, utilizando-se do Módulo de Comunicação;

- **Exportador** – o *Transformer* não é o destino final dos dados. Assim, o módulo exportador simula um *Collector* que irá enviar os dados coletados para um destinatário definido (outro *Transformer* ou o *Manager*), viabilizando o encadeamento de um ou mais *Transformers* em sequência;
- **Armazenamento Local** – o *Transformer* executará em dispositivos mais robustos da *Fog*. Um poder computacional mais elevado viabilizará o recebimento de dados de vários *Collectors*, assim como transformações e consolidações. Para tanto, o *Transformer* usará bases de dados leves, adequadas à *Fog*, e que consigam lidar com volumes moderados de dados, antes de transmiti-los ao *Manager*. No *Transformer*, os dados não poderão ser visualizados pelo usuário final, ou consumidos por outra aplicação ou serviço. Essa função é do *Manager*;
- **Módulo de Comunicação** – esse módulo é responsável pela troca de mensagens de controle com o *Manager* e com os *Collectors* vinculados. Alterações na lista de componentes e nas funções a serem aplicadas aos dados recebidos são efetuadas por esse módulo a partir do recebimento de mensagens do Manager.

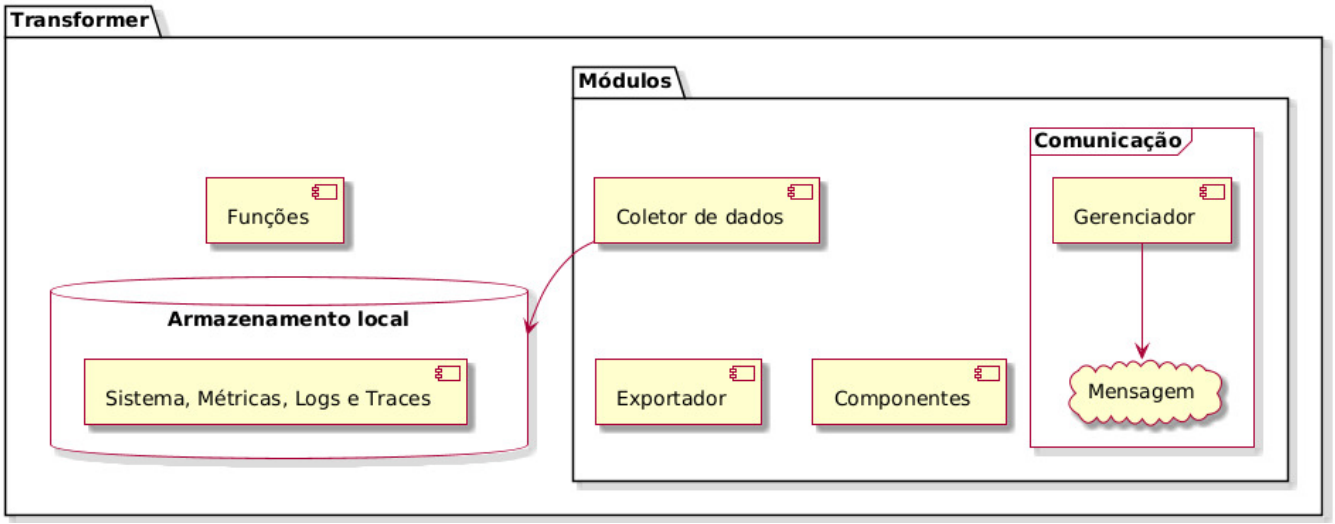


Figura 8.3: Módulos que constituem o componente *Transformer*.

8.1.3 *Manager*

O componente *Manager* representa o destino final dos dados coletados pelos *Collectors*, e processados pelos *Transformers* (quando for o caso), em termos de processo de gestão da observabilidade. Ele implementa a função de exposição (Seção 3.3), mostrada na Figura 3.2, que objetiva a retenção de longo prazo das informações, e a visualização dos dados por meio de painéis e gráficos especializados. Além disso, ele é responsável pela disponibilização dos dados para as outras funcionalidades da orquestração de serviços em *Fog*, como por exemplo o gerenciamento de recursos, o gerenciamento de serviços e a otimização, descritos na Seção 2.4.1.

Ao gerenciar os três domínios de instrumentação, é atribuição do *Manager* oferecer armazenamento e consultas apropriadas para cada um deles. De acordo com Karumuri *et al.* [1], métricas, *logs* e *traces* diferem nas características dos dados, nas necessidades de armazenamento e na forma em que são consumidos.

Isso ocorre porque as métricas têm tipos de dados numéricos, cujo armazenamento e consumo podem ser melhor atendidos com um TSDB. Os *traces*, por sua vez, têm tipos de dados representados por Grafos Diretos Acíclicos (DAG, do inglês *Direct Acyclic Graph*) das durações de execução, e o seu armazenamento e consumo podem ser melhor atendidos com um banco de dados colunar ou de índice invertido. *Logs*, *strings* de texto sem formatação, também se beneficiam de bancos de dados de índice invertido [1]. O volume de recebimento de dados de cada domínio, assim como o tempo de retenção, também são independentes entre si. A Figura 8.4 mostra uma visão mais detalhada dos módulos que compõem o *Manager*, descritos nos itens a seguir:

- **Interface** – o *Manager* terá uma interface gráfica por meio da qual é possível modificar as suas configurações, adicionar serviços a serem monitorados, gerenciar limites para a geração de eventos e notificações, além de acessar o módulo de consultas e visualização dos dados de observabilidade. A interface gráfica resultará em chamadas ao Módulo de Comunicação;
- **Coletor de Dados** – é o módulo responsável por solicitar e receber os dados dos *Collectors* (ou dos *Transformers*, quando estes estiverem em sequência com os *Collectors*), enviando-os para o armazenamento local nos bancos de dados especializados;
- **Componentes** – esse módulo gerencia os identificadores de acesso (por exemplo, endereços de rede, portas de acesso, etc), assim como relaciona as funções de processamento que atuarão nos dados recebidos de cada *Collector* ou *Transformer*;
- **Funções** – contém as funções que serão aplicadas aos dados recebidos, quando necessário. É o mesmo mecanismo já apresentado para o *Transformer*;
- **Visualizador** – o *Manager* é responsável pela exposição de dados, que é uma das três funções primordiais do sistema de gestão da observabilidade, conforme descrito na Seção 3.3 e mostrado na Figura 3.2. Este módulo apresenta os dados de observabilidade coletados por meio de gráficos especializados, que servirão para análise e tomada de decisão pela equipe responsável pela gestão da observabilidade.
- **Bancos de Dados** – os domínios de instrumentação – métricas, *logs* e *traces* – têm características diferentes quanto aos dados que são coletados e quanto aos tipos e volume de consultas que são feitas nestes dados [1]. O *Manager* irá prover bases de dados apropriadas para a gestão do ciclo de vida dos dados e para a otimização das consultas mais frequentes;
- **Módulo de Comunicação** – o Módulo de Comunicação é a forma que o orquestrador de *Fog* (ou outro sistema autorizado) tem para solicitar serviços e requisitar informações do *Manager*. Também é por meio deste módulo que o Manager troca mensagens de controle com os demais componentes da arquitetura. Ele oferece todas as possibilidades que a interface gráfica provê;
- **Notificações** – o módulo de notificações mantém um cadastro dos contatos da equipe responsável pela gestão da observabilidade para viabilizar o envio de notificações (*e-mails*, mensagens de texto de celulares, etc), quando algum dos limites cadastrados for atingido nos dados de observabilidade recém-recebidos.

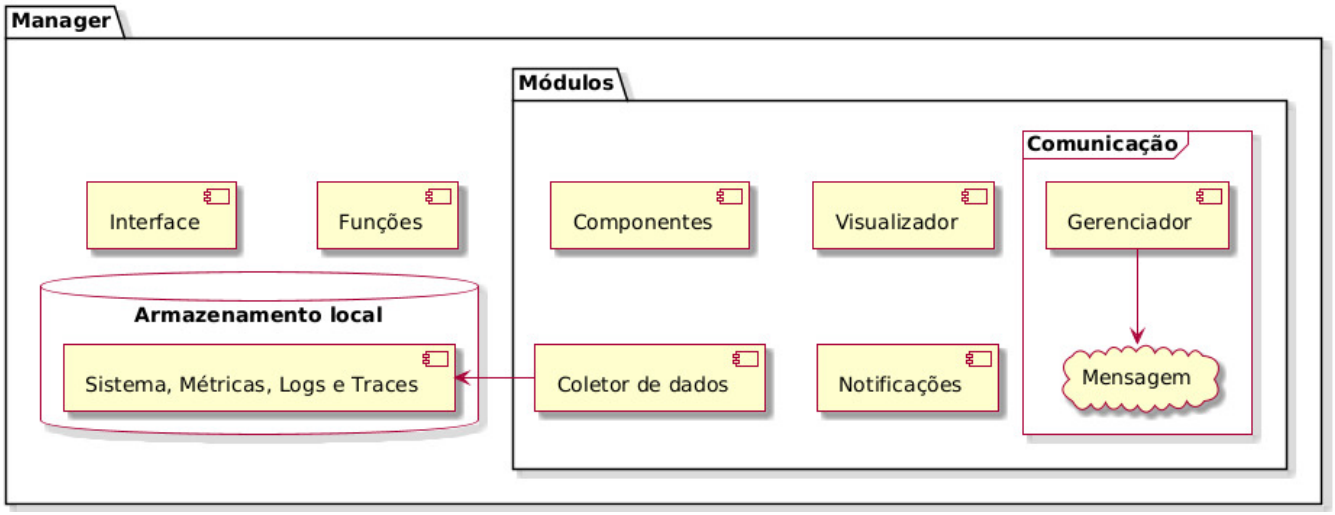


Figura 8.4: Módulos que constituem o componente *Manager*.

8.2 *Framework* de Adaptabilidade para FogObserver

Esta seção apresenta um *framework* de adaptabilidade para FogObserver, auxiliando o desenvolvimento de sistemas autoadaptáveis de gestão da observabilidade em *Fog*. Para lidar com os desafios descritos na Seção 3.6, e para ajudar a garantir os SLAs de um serviço de IoT, um sistema de gestão da observabilidade de *Fog* pode fornecer um comportamento autoadaptável, significando que reagirá ao ambiente dinâmico e se reconfigurará automaticamente, de acordo com os parâmetros especificados. Mas projetar e desenvolver adaptabilidade é uma questão complexa em tal cenário [79]. Para facilitar esse esforço, projetou-se um *framework* de adaptabilidade para um sistema de gestão da observabilidade baseado na arquitetura FogObserver. Assim sendo, o *framework* proposto tem as seguintes características:

- reconhece o ODLC; [146];
- é baseado no ciclo de controle MAPE-K [51];
- atende aos requisitos de adaptabilidade consolidados por Barba e Giorno [79], descritos na Seção 4.1;
- incorpora os padrões de lógica de controle selecionados [82], apresentados na Seção 4.2; e
- é agnóstico quanto à topologia, o que significa que pode ser usado para desenvolver sistemas de gestão da observabilidade autoadaptáveis que são centralizados, hierárquicos ou distribuídos [94].

O *framework* se utiliza de dois componentes principais da arquitetura FogObserver: *Collector* e *Manager*. O *Collector* é o componente implantado nos dispositivos IoT, como o computador de bordo do *SmartTruck* do cenário motivador (Seção 3.2), coletando dados de observabilidade do ambiente (métricas, *logs* e *traces*), e enviando-os para o *Manager* para análise e armazenamento a longo prazo. O *Manager* é o servidor de observabilidade que recebe dados de observabilidade de todos os dispositivos. Ele armazena esses dados em repositórios apropriados de acordo com as necessidades da equipe de manutenção de serviços, gera alertas e notificações, e fornece uma interface de usuário para visualização de dados e atualização de parâmetros. Ambos os componentes recebem do *framework* módulos de tomada de decisão e de reconfiguração do comportamento, que implementam um mecanismo autoadaptável. Além disso, eles se

comunicam entre si com base em um formato de mensagem e protocolo estabelecidos. A Figura 8.5 apresenta internamente os componentes da arquitetura FogObserver, detalhando o fluxo de dados e de controle entre seus módulos e os módulos que compõem o *framework*. O funcionamento da arquitetura com uso do *framework* de adaptabilidade é descrito a seguir:

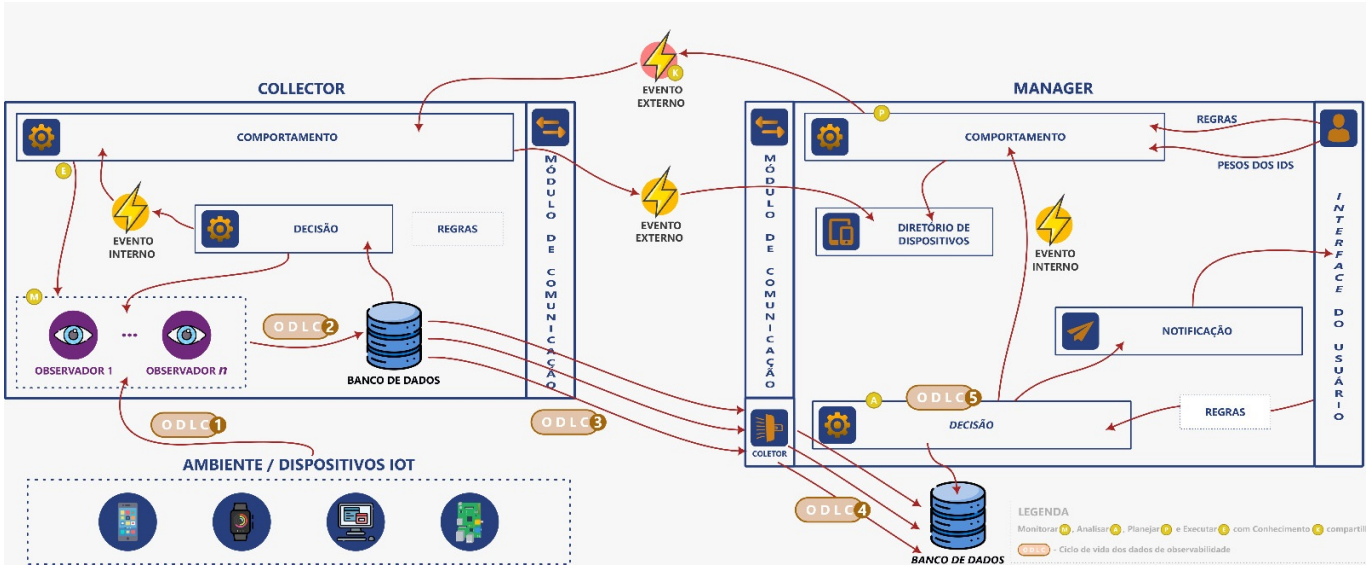


Figura 8.5: Componentes de um sistema de observabilidade autoadaptável.

Collector - os módulos observadores coletam dados de métricas, *logs* e *traces* do ambiente e os registram no dispositivo IoT para transmissão posterior. Essas são as duas primeiras etapas do ODLC. O módulo de decisão verifica continuamente se os dados armazenados correspondem a alguma regra definida. Caso positivo, ele cria eventos internos pré-definidos, por exemplo, disco cheio, largura de banda de rede muito baixa e bateria baixa, especificando a situação que surgiu. Além disso, ele muda o *status* do dispositivo de acordo com as regras, por exemplo, Normal, Crítico e Recuperação. O módulo de comportamento é responsável por reconfigurar o componente *Collector*. Após receber eventos do módulo de decisão ou diretamente do *Manager*, ele pode executar ações como parar um ou mais observadores, atualizar as regras e realizar outras ações necessárias para evitar a exaustão de recursos do componente *Collector* ou do dispositivo IoT. O módulo de comunicação fornece comunicação bidirecional. Recebe eventos externos do *Manager* (mensagens com ações ou dados, como um comando de reinício e um novo conjunto de regras) e sinaliza ao módulo de comportamento para processá-los. Além disso, ele envia as mudanças de *status* e os eventos que causaram as mudanças para o *Manager*.

Manager - o módulo coletor de dados recebe os dados de observabilidade (métricas, *logs* e *traces*) dos módulos observadores nos dispositivos. Ele é responsável por registrá-los no armazenamento adequado de acordo com as necessidades do usuário. Este é o quarto passo do ODLC. A interface do usuário permite que os usuários do sistema de gestão da observabilidade (geralmente a equipe de operações e manutenção) visualizem os dados de observabilidade coletados e atualizem o *status* dos dispositivos. O *framework* adiciona a capacidade de alterar manualmente: 1. as regras utilizadas para caracterizar os modos de operação dos dispositivos; e 2. os pesos dos domínios de instrumentação (Seção 7.3). Essas mudanças podem ser aplicadas na interface para todos os dispositivos ou para dispositivos selecionados. O módulo de comunicação é semelhante ao do componente *Collector* e funciona de forma bidirecional. Ele envia atualizações de

dados (regras, pesos do domínio de instrumentação) para os *Collectors* que implementam comportamento autoadaptável. Além disso, recebe alterações de *status* dos *Collectors* e atualiza essas informações no diretório de dispositivos. O módulo de decisão verifica continuamente se os dados armazenados correspondem a alguma das regras definidas. Caso positivo, ele cria notificações, que são enviadas diretamente para os usuários, e eventos que serão enviados para o módulo de comportamento. Este é o quinto passo do ODLC. O módulo de comportamento no *Manager* cria os eventos externos para mudar o comportamento de um ou mais dispositivos. Ele utiliza dados provenientes da interface do usuário, o *status* dos dispositivos e eventos internos gerados pelo módulo de decisão.

8.2.1 Interação entre os Componentes da Arquitetura

O *framework* de adaptabilidade permite que o componente *Collector* de um sistema de gestão da observabilidade em *Fog* monitore as mudanças no ambiente dos dispositivos em que executa e tome decisões locais autônomas com base nessas mudanças e nas regras definidas. Entre as possíveis decisões estão, por exemplo, comunicar eventos críticos, suspender a coleta de dados de um ou mais domínios de instrumentação e remover dados antigos que já foram enviados ao *Manager*. Todos os eventos, ações e comunicações devem ser registrados localmente para viabilizarem auditorias futuras.

O componente *Manager*, ao detectar que um dispositivo entrou em modo crítico, pode tomar medidas autônomas, como solicitar o envio de todos os dados de observabilidade disponíveis, ou modificar os pesos dos domínios de instrumentação. Além das decisões autônomas, o *Manager* permite intervenções manuais da equipe de manutenção por meio da interface visual. Decisões no *Manager* podem modificar regras de comportamento, atualizando assim os dispositivos selecionados. Devido ao grande volume de dispositivos em ambientes *Fog*, o uso de filas de mensagens e categorias de prioridades de mensagens é importante para garantir que mensagens críticas sejam processadas prioritariamente.

Para entender como esses componentes funcionam juntos, fornecendo o comportamento autoadaptativo no sistema de gestão de observabilidade, foram descritos dois cenários característicos. Eles podem ser visualizados nos diagramas de sequência mostrados nas Figuras 8.6 e 8.7. No primeiro cenário, o *framework* de adaptabilidade mudará o *status* de um dispositivo e comunicará isso ao *Manager*. O componente *Collector* está executando em um dispositivo IoT com três observadores coletando dados de métricas, *logs* e *traces*, e os entregando ao *Manager* seguindo as etapas do OLDC. O conjunto de regras disponíveis estabelece que o espaço disponível em disco deve ser superior a 10%, a bateria dos dispositivos deve estar acima de 20%, e a largura de banda da rede deve ser, pelo menos, de 100 Kbps. Essas regras são verificadas pelo módulo de decisão em um intervalo de tempo pré-determinado. Neste exemplo, foi definido o tempo de cinco segundos. O *status* do dispositivo está definido como MODO_NORMAL. Em um determinado momento, o espaço em disco disponível atinge 9%, o módulo de decisão gera um evento interno DISCO_CHEIO, e o envia ao módulo de comportamento. O módulo de comportamento recebe o evento, muda o *status* do dispositivo de MODO_NORMAL para MODO_CRÍTICO, cria uma mensagem, e a envia ao módulo de comunicação. O módulo de comunicação envia essa mensagem ao módulo análogo no componente *Manager*. No *Manager*, o módulo de comunicação recebe a mensagem e a transmite ao módulo de comportamento, que atualiza o *status* do dispositivo no diretório.

No segundo cenário, um usuário deste sistema autoadaptável de gestão da observabilidade deseja investigar por que o serviço está apresentando erros em dispositivos específicos, enquanto funciona corretamente em dezenas de outros. Mas a *Fog* está sobrecarregada processando todos os fluxos de dados. O usuário decide mudar as prioridades dos domínios de instrumentação, e desativar a coleta e transmissão de métricas, *logs* e *traces* dos dispositivos que não estão apresentando problemas, aliviando a carga em *Fog* para acelerar a investigação. O usuário acessa a interface e modifica os pesos das métricas, *logs* e *traces* para zero, e aplica essa configuração a todos os dispositivos nos quais o serviço está funcionando corretamente. A Interface do Usuário envia esses pesos e a lista de dispositivos para o módulo de comportamento. Na sequência, o módulo de comportamento gera um evento externo DEFINIR_PESOS (0,0,0), registra essa ação no diretório de dispositivos, e envia uma mensagem contendo o evento externo e a lista de dispositivos para o módulo de comunicação. Lá, a mensagem é enviada para cada dispositivo na lista. No *Collector* do dispositivo, o módulo de comunicação recebe a mensagem, valida o remetente e passa a mensagem para o módulo de comportamento local. O módulo de comportamento no dispositivo IoT desliga os observadores, parando o ODLC deste dispositivo.

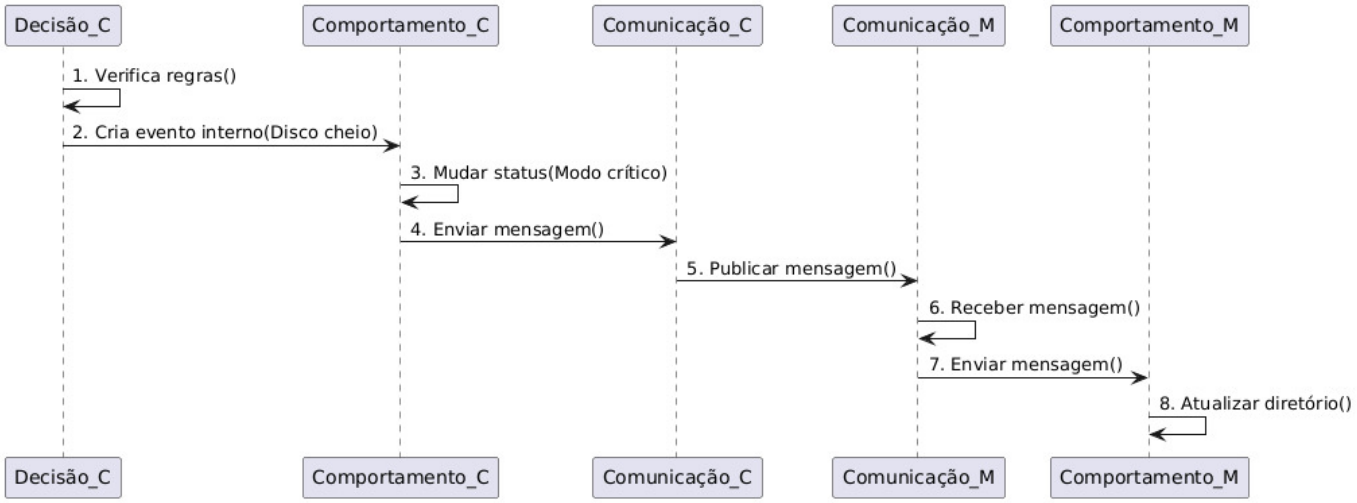


Figura 8.6: Primeiro cenário: o *framework* muda o *status* de um dispositivo e comunica ao *Manager*. A letra após o nome do módulo identifica o componente da arquitetura: C=*Collector* e M=*Manager*.

8.2.2 Fluxo de Execução do *Framework* de Adaptabilidade em Dois Cenários Hipotéticos

Nesta seção, é apresentada uma simulação da execução do *framework* de adaptabilidade proposto. A intenção é demonstrar como o *framework* se comporta em cenários reais de operação. Nesta simulação, serão descritos dois cenários que representam situações comuns em ambientes de *Fog Computing*. Em cada cenário, será simulada a resposta do *framework* e será analisada a adequação de suas ações, demonstrando como o *framework* implementa o *loop* de controle MAPE-K [51], descrito no Capítulo 4.

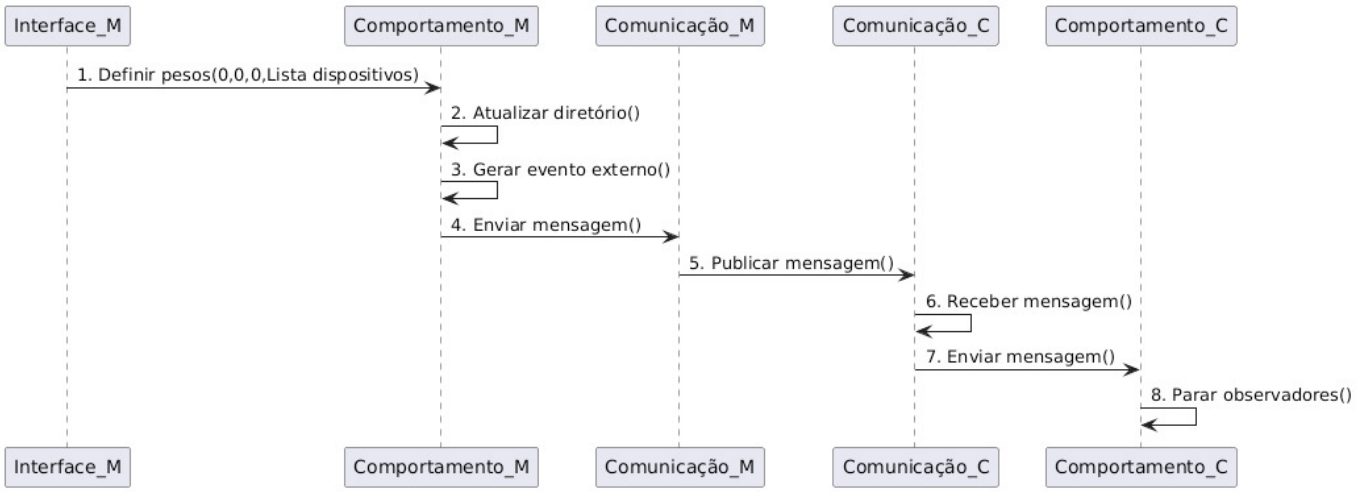


Figura 8.7: Segundo cenário: *Manager* desativa a coleta e transmissão de métricas, *logs* e *traces* dos dispositivos que não estão apresentando problemas. A letra após o nome do módulo identifica o componente da arquitetura: C=*Collector* e M=*Manager*.

Cenário 1: Instabilidade na Conexão de Rede

- **Descrição do Cenário:** um *link* de comunicação entre um conjunto de dispositivos IoT e um *Fog Node* torna-se instável, com alta perda de pacotes e latência variável. Este *link* é usado para transmitir dados de observabilidade para o *Fog Node*.
- **Simulação da Resposta do *Framework***:
  - **Monitoramento:** o *Collector*, nos dispositivos IoT, e o *Manager*, no *Fog Node*, detectam a alta perda de pacotes e latência. O módulo de decisão do *Collector* identifica a instabilidade da rede com base em limites pré-definidos;
  - **Análise:** o módulo de decisão do *Collector* conclui que enviar dados na taxa atual pode levar à perda de informações e à sobrecarga da rede, aumentando a latência;
  - **Planejamento:** o módulo de comportamento do *Collector* muda o modelo de comunicação de *push* para *pull*, e adia a transmissão de dados de *logs* e *traces*. Aumenta a frequência de envio de métricas para monitorar a estabilidade da rede, reduzindo a quantidade total de dados enviados de forma contínua. Os dados serão armazenados localmente e temporariamente, quando possível;
  - **Execução:** o módulo de comportamento do *Collector* reconfigura seu módulo de comunicação para usar o modelo *pull* e adia o envio dos *logs* e *traces*. O módulo de comunicação informa o *Manager* da mudança de estratégia e aguarda instruções, que podem vir do usuário.
- **Análise da Adequação:** a resposta do *framework* é apropriada, pois evita sobrecarregar a rede instável e prioriza o envio de dados essenciais para monitorar o estado da rede. A mudança para o modelo *pull* e o adiamento do envio de *logs* e *traces* garantem que os dados sejam enviados quando a conexão estiver estável, preservando a integridade dos dados de observabilidade. Ao mesmo tempo, métricas são priorizadas para garantir a tomada de decisão em tempo hábil.

Cenário 2: Bateria Baixa em um Dispositivo IoT

- **Descrição do Cenário:** um dispositivo IoT específico está com a bateria se esgotando rapidamente, com um nível de carga abaixo de 20%, o que pode levar à perda de dados e à interrupção da sua operação. Este dispositivo está coletando dados de observabilidade e os enviando para um *Fog Node*.
- **Simulação da Resposta do *Framework***:
  - **Monitoramento:** o *Collector*, no dispositivo IoT, monitora continuamente o nível da bateria. O módulo de decisão detecta que a bateria está abaixo de 20%;
  - **Análise:** o módulo de decisão do *Collector* entende que o envio de dados na taxa atual pode acelerar a descarga da bateria, e causar a perda de dados, indisponibilidade do dispositivo e perda de informações relevantes sobre o estado do sistema;
  - **Planejamento:** o módulo de comportamento do *Collector* decide desativar temporariamente a coleta de *logs* e *traces* e reduzir a frequência da coleta de métricas para economizar energia. O módulo também muda o status do dispositivo de "Normal" para "Crítico";
  - **Execução:** o módulo de comportamento do *Collector* desativa os observadores de *logs* e *traces* e ajusta o observador de métricas para coletar dados com menos frequência, diminuindo o consumo de energia. O módulo de comunicação envia um evento com o novo *status* e a nova configuração para o *Manager*.
- **Análise da Adequação:** a resposta do *framework* é apropriada, pois prolonga a vida útil da bateria do dispositivo, garantindo que ele continue funcionando o máximo de tempo possível, mesmo em condições adversas. Ao desativar a coleta de *logs* e *traces*, e priorizar a coleta de métricas com menor frequência, o *framework* economiza energia e continua coletando informações essenciais sobre o estado do sistema.

Esses dois cenários hipotéticos demonstram a capacidade do *framework* de adaptabilidade de responder a diferentes situações em um ambiente de *Fog Computing*, ajustando a coleta e o envio de dados de observabilidade de forma autônoma. Cada cenário ilustra como os componentes *Collector* e *Manager* interagem para tomar decisões com base em monitoramento, análise e planejamento, demonstrando a implementação do ciclo de controle MAPE-K [51]. O *framework* atende aos requisitos de adaptabilidade, como a percepção do contexto, o raciocínio baseado em regras e a reconfiguração dinâmica dos componentes, descritos na Seção 4.1.

8.2.3 Análise do *Framework*

Esta seção conduz uma análise do *framework*, estruturada em três ações complementares: (i) a revisão de aderência do *framework* aos requisitos de adaptabilidade; (ii) avaliação da modularidade e extensibilidade do *framework* e (iii) uma análise da implementação dos padrões de projetos (apresentados no Capítulo 4), que são descritos a seguir:

1. **Aderência aos Requisitos de Adaptabilidade:** O *framework* foi projetado para atender aos requisitos de adaptabilidade, consolidados por Barba e Giorno [79], que orbitam em torno de três elementos

essenciais para viabilizar um comportamento autoadaptável em *Fog*: percepção, raciocínio e comportamento [79];

- **Percepção:** o componente *Collector* implementa a percepção ao monitorar continuamente o ambiente e os recursos dos dispositivos IoT, coletando dados de observabilidade (métricas, *logs* e *traces*). Ele utiliza observadores para adquirir informações de diferentes fontes, como o uso da CPU, memória, nível de bateria, condições da rede, entre outros;
- **Raciocínio:** o módulo de decisão, tanto no *Collector* quanto no *Manager*, realiza o raciocínio com base em regras predefinidas e dados históricos. Ele interpreta os dados coletados, identificando situações que requerem adaptação, como sobrecarga, instabilidade da rede ou bateria baixa. O raciocínio também envolve a avaliação da importância relativa de cada domínio de instrumentação e o esforço necessário para coletar e analisar os dados;
- **Comportamento:** o módulo de comportamento, também presente no *Collector* e no *Manager*, define e executa as ações de reconfiguração com base no raciocínio do módulo de decisão. Essas ações incluem ajustar a frequência da coleta de dados, mudar o modelo de comunicação (*push/pull*), priorizar domínios de instrumentação específicos, ou mesmo desativar temporariamente a coleta de alguns dados. O módulo de comportamento também é responsável por comunicar as mudanças de *status* e configuração entre os componentes do sistema.

  A arquitetura, ao separar as funções de percepção (coleta de dados), raciocínio (análise e tomada de decisão) e comportamento (execução de ações), promove a adaptabilidade, já que cada parte pode ser alterada ou estendida sem afetar as outras, facilitando a evolução do sistema.

2. **Modularidade e Extensibilidade:** O *framework* foi projetado com uma arquitetura modular, com componentes bem definidos e interfaces claras, facilitando a adição de novas funcionalidades e a adaptação a diferentes cenários.

   - **Componentes Independentes:** os componentes *Collector*, *Transformer* e *Manager* são independentes, e podem ser implementados e modificados separadamente. Essa separação permite que diferentes domínios de instrumentação (métricas, *logs* e *traces*) sejam gerenciados de forma independente e flexível;
   - **Extensibilidade de Domínios:** a arquitetura permite a adição de novos domínios de instrumentação sem demandar alterações importantes nos componentes existentes. Isso é essencial, pois diferentes casos de uso em *Fog* podem exigir diferentes tipos de dados de observabilidade;
   - **Reutilização de Componentes:** os módulos de decisão e de comportamento são genéricos e podem ser reutilizados em diferentes componentes, garantindo a consistência do comportamento adaptativo em todo o sistema;
   - **Agnosticismo de Topologia:** o *framework* é agnóstico quanto à topologia, o que significa que pode ser usado para desenvolver sistemas de gestão da observabilidade autoadaptáveis que são centralizados, hierárquicos ou distribuídos. Isso permite que o sistema seja adaptado às necessidades específicas de cada ambiente, aproveitando as vantagens de cada topologia.

3. **Implementação dos Padrões de Projeto:** a arquitetura do *framework* incorpora os seguintes padrões de projeto, descritos na Seção 4.2:

    - **Fábrica de Sensores:** o padrão Fábrica de Sensores é implementado pelo componente *Collector*, que gerencia uma rede distribuída de observadores (sensores) para coletar dados do sistema e do ambiente. Essa abstração facilita a adaptação da infraestrutura de gestão da observabilidade a mudanças, permitindo que novos tipos de sensores sejam adicionados ou removidos sem afetar os outros componentes;
    - **Roteamento Baseado em Conteúdo:** o *framework* utiliza um modelo de comunicação flexível que permite rotear dados de observabilidade com base em seu tipo e conteúdo. Os clientes (módulos de análise, de visualização, ou outros) se inscrevem em tipos específicos de dados, e o sistema garante que eles recebam apenas as informações relevantes. Este padrão pode ser implementado por meio da combinação de modelos de comunicação *pull* e *push*;
    - **Detecção de Adaptação:** o padrão Detecção de Adaptação é implementado pelo módulo de decisão, que interpreta os dados dos sensores e compara-os com limites pré-definidos ou dados históricos. Quando uma divergência significativa é detectada, o módulo aciona um evento, levando à aplicação de uma reconfiguração;
    - **Raciocínio Baseado em Casos:** o módulo de decisão também pode implementar o padrão Raciocínio Baseado em Casos, utilizando um repositório de conhecimento com regras predefinidas para selecionar as reconfigurações adequadas com base nos eventos detectados;
    - **Reconfiguração do Servidor:** o padrão Reconfiguração do Servidor é implementado pelo módulo de comportamento, que garante que as reconfigurações do sistema ocorram em tempo real e de forma consistente, evitando interrupções nos serviços. O módulo pode utilizar *buffers* para armazenar dados em trânsito durante a reconfiguração, garantindo que eles não sejam perdidos.

A análise teórica do *framework* demonstra que ele foi projetado para ser adaptável, modular e extensível, incorporando os padrões de projeto adequados para lidar com a dinamicidade e as restrições de recursos típicas de ambientes de *Fog Computing*. Ao separar as funções de percepção, raciocínio e comportamento, e ao integrar o ciclo de vida dos dados de observabilidade, o *framework* oferece uma solução abrangente e flexível para a gestão da observabilidade em sistemas distribuídos. A capacidade de adaptação do *framework* foi demonstrada pelos cenários hipotéticos apresentados anteriormente, em que o sistema ajusta seu comportamento de forma autônoma em resposta a diferentes condições do ambiente.

8.3 Trabalhos Relacionados

Com base na literatura, esta seção relaciona a arquitetura FogObserver, e o *framework* de adaptabilidade, a trabalhos existentes, destacando suas contribuições e diferenciais. A arquitetura FogObserver destaca-se por abordar a gestão da observabilidade em ambientes de *Fog Computing* de forma abrangente e adaptável, indo além das soluções existentes que, frequentemente, focam em aspectos isolados ou em outros paradigmas computacionais [114, 120, 127]. A revisão da literatura, apresentada no Capítulo 2, revelou que, embora a

gestão da observabilidade seja reconhecida como crucial para a orquestração em *Fog* [5], muitas propostas negligenciam detalhes de implementação e desafios específicos. A FogObserver surge, portanto, como uma resposta a essa lacuna, propondo uma arquitetura que integra coleta, processamento, e gestão de dados de observabilidade de forma adaptativa, considerando as restrições e a dinamicidade do ambiente de *Fog*. Assim sendo, as principais contribuições da FogObserver em relação à literatura são:

- **Gestão Unificada de Domínios de Instrumentação:** diferentemente de outras soluções que se concentram principalmente em métricas, a FogObserver é projetada para lidar com múltiplos domínios de instrumentação (métricas, *logs* e *traces*), simultaneamente. Essa característica é fundamental para aumentar o nível de observabilidade da infraestrutura e dos sistemas, permitindo a análise da causa raiz de problemas e o monitoramento do comportamento interno dos serviços. Essa abordagem encontra apoio na literatura, que reconhece o valor da análise cruzada entre diferentes domínios de instrumentação [147], mas que ainda não foi totalmente implementada em ambientes de *Fog* [146];
- **Adaptabilidade e Autonomia:** a FogObserver incorpora um *framework* de adaptabilidade que permite que o sistema ajuste seu comportamento em tempo real, com base nas condições do ambiente e nas necessidades dos serviços. Essa adaptabilidade é crucial em ambientes de *Fog*, caracterizados por restrições de recursos, redes instáveis e alta mobilidade [27]. As soluções existentes, como PyMon [114], FMonE [59] e a pilha Prometheus [120], não oferecem essa flexibilidade, limitando sua capacidade de lidar com as variações do ambiente de *Fog*. A literatura reconhece a importância da adaptabilidade em sistemas de observabilidade de *Fog* [79], mas poucas propostas implementam esse conceito de forma efetiva;
- **Arquitetura Modular e Extensível:** a FogObserver é construída como uma arquitetura modular, com componentes bem definidos e interfaces claras. Essa modularidade facilita a adição de novas funcionalidades e a adaptação a diferentes cenários de uso. Além disso, a arquitetura é agnóstica em relação à topologia da rede, permitindo sua implantação em diferentes configurações (centralizada, hierárquica ou distribuída). Essa flexibilidade é uma vantagem em relação a muitas propostas existentes que são fortemente acopladas a uma topologia específica;
- **Integração com o Ciclo de Vida dos Dados de Observabilidade (ODLC):** a arquitetura FogObserver é projetada para gerenciar todo o ciclo de vida dos dados de observabilidade. Isso garante que os dados sejam coletados, processados e disponibilizados de forma eficiente, atendendo às necessidades da orquestração de serviços em *Fog* [5]. Embora algumas ferramentas de código aberto, como Prometheus [120] e ElasticSearch [151], abordem aspectos específicos do ciclo de vida, a FogObserver integra essas funcionalidades. A literatura aponta que a gestão do ciclo de vida dos dados de observabilidade é essencial [146], mas ainda são necessários estudos que a implementem de forma mais abrangente;
- **Implementação de Padrões de Projeto:** a arquitetura FogObserver é baseada em padrões de projeto bem estabelecidos [89], como a Fábrica de Sensores, o Roteamento Baseado em Conteúdo, a Detecção de Adaptação e a Reconfiguração do Servidor, descritos no Capítulo 4. Essa abordagem garante que o sistema seja robusto, flexível e fácil de manter, aproveitando as melhores práticas de projeto

de sistemas distribuídos. A aplicação de padrões de projeto específicos para sistemas autoadaptáveis, assim como do uso do *loop* de controle MAPE-K [51], também são inovações da FogObserver;

- **Apoio à Orquestração:** a análise da literatura revelou que poucas soluções de gestão da observabilidade estão preparadas para serem integradas a orquestradores de serviços de *Fog* [80]. A FogObserver, ao ser modular, leve, multiplataforma, focada na gestão da observabilidade e adaptável, busca preencher essa lacuna, oferecendo uma solução que pode ser integrada e gerenciada por um orquestrador de serviços.

Tabela 8.1: Comparativo entre FogObserver e soluções de gestão da observabilidade.

Solução	**Três Domínios de Instrumentação**	**Implementa ODLC**	**Adaptável**	**Extensível**	**Apoio à Orquestração**
PyMon [114]	✗	✗	✗	✗	✗
FMonE [59]	✗	✗	✗	✗	✗
Prometheus [108]	✗	✓	✓	✓	✓
Osmótico [125]	✗	✗	✗	✗	✗
Mobile [127]	∂	✗	✓	✗	✓
Switch [131]	✗	∂	✓	✓	✓
SCB [10]	✗	✗	✓	✗	✓
Rule Based [136]	✗	✗	✓	✓	✓
TEEMon [137]	✗	✗	✗	✗	∂
FogMon [4]	✗	∂	✓	✓	∂
FogObserver	✓	✓	✓	✓	✓

✓ denota que o item é implementado/atendido;
∂ denota que o item é parcialmente implementado/atendido;
✗ denota que o item não é implementado/atendido.

A tabela 8.1 consolida o comparativo entre FogObserver e as soluções analisadas no Capítulo 6. A arquitetura FogObserver se destaca na literatura por ser uma solução abrangente e adaptável para a gestão da observabilidade em ambientes de *Fog Computing*. Ao integrar múltiplos domínios de instrumentação, um *framework* autoadaptável, uma arquitetura modular e extensível, e a gestão do ciclo de vida dos dados de observabilidade, a FogObserver oferece uma contribuição original para a área. A sua relação com a literatura é marcada pela superação das limitações das soluções existentes e pela incorporação das melhores práticas de projeto de sistemas distribuídos, contribuindo efetivamente para a construção de sistemas de observabilidade mais robustos, flexíveis e eficientes em ambientes de *Fog*. A arquitetura FogObserver é, portanto, uma contribuição relevante que permite avançar a área de pesquisa de gestão da observabilidade em *Fog*, ao endereçar as principais limitações e lacunas existentes.

8.4 Considerações Finais

Este capítulo apresentou a arquitetura FogObserver, uma proposta para sistemas de gestão da observabilidade em *Fog*, que viabiliza a coleta, o gerenciamento e a entrega de dados referentes a métricas, *logs* e *traces*. Para isso, foram detalhados os componentes da arquitetura: *Collector*, *Transformer* e *Manager*, e

proposto um *framework* que torna a arquitetura autoadaptável, baseada no ciclo de controle MAPE-K. A arquitetura foi projetada para ser extensível, aceitando a adição de outros domínios de instrumentação sem necessitar alterações significativas, superando assim os desafios característicos de *Fog*.

No próximo capítulo, será apresentada uma avaliação da proposta desta tese em um contexto real de cidades inteligentes, utilizando um estudo de caso para validar as propostas descritas neste e no capítulo anterior.

Capítulo 9

Avaliação da Proposta em um Contexto Real de Cidades Inteligentes

Este capítulo apresenta uma avaliação das propostas descritas no Capítulo 7 e no Capítulo 8, utilizando um estudo de caso de cidades inteligentes. A pesquisa explora os benefícios e os desafios de aumentar a observabilidade em ambientes de *Fog Computing*, com foco em um estudo de caso real: o Mobile IoT-RoadBot, uma aplicação de cidade inteligente que monitora a infraestrutura urbana, descrita na Seção 3.2.

Para isso, o capítulo está estruturado em cinco seções. A Seção 9.1 apresenta a medição do nível de observabilidade de um *SmartTruck* executando Mobile IoT-RoadBot. Em seguida, a Seção 9.2 descreve a infraestrutura do ambiente de testes, detalhando os dispositivos e as ferramentas de código aberto utilizados para implementar a arquitetura FogObserver. Na sequência, a Seção 9.3 apresenta a avaliação experimental da implementação do ODLC e da arquitetura FogObserver no ambiente de testes. A Seção 9.4 descreve a implementação do *framework* de adaptabilidade da FogObserver em ambiente simulado. Por fim, a Seção 9.5 apresenta uma síntese da avaliação realizada neste capítulo.

9.1 Nível de Observabilidade

Para demonstrar a utilidade do indicador de nível de observabilidade, definido no Capítulo 7, é apresentado um gráfico da medida do nível de observabilidade de um *SmartTruck*, enquanto percorria as ruas de Brimbank, Austrália, em julho de 2022. A Figura 9.1 apresenta o resultado da Equação 7.3, consolidado para períodos de 30 minutos, durante um dia da jornada real de trabalho de um *SmartTruck*.

Na Seção 8.2, a Equação 8.3 define o nível de observabilidade como a soma da disponibilidade individual de cada domínio de instrumentação mais o resultado da interação sinérgica entre eles: Observabilidade = Metricas + Logs + Traces + (Metricas X Logs X Traces). Considerando que o Mobile IoT-Roadbot gerava apenas métricas e *logs*, o termo Traces na equação é sempre 0. Assim, a equação para este caso específico se torna Observabilidade = Metricas + Logs + (Metricas X Logs). O valor máximo de 3 ocorre quando tanto as métricas quanto os *logs* estão disponíveis para análise (cada um contribuindo com 1) e, adicionalmente, há uma interação sinérgica entre eles (contribuindo com mais 1), o que acontece quando os dados de ambos os domínios correspondem a um período de tempo específico. As variações ao longo do dia refletem as mudanças na disponibilidade desses dois domínios de instrumentação. Em momentos em que apenas um

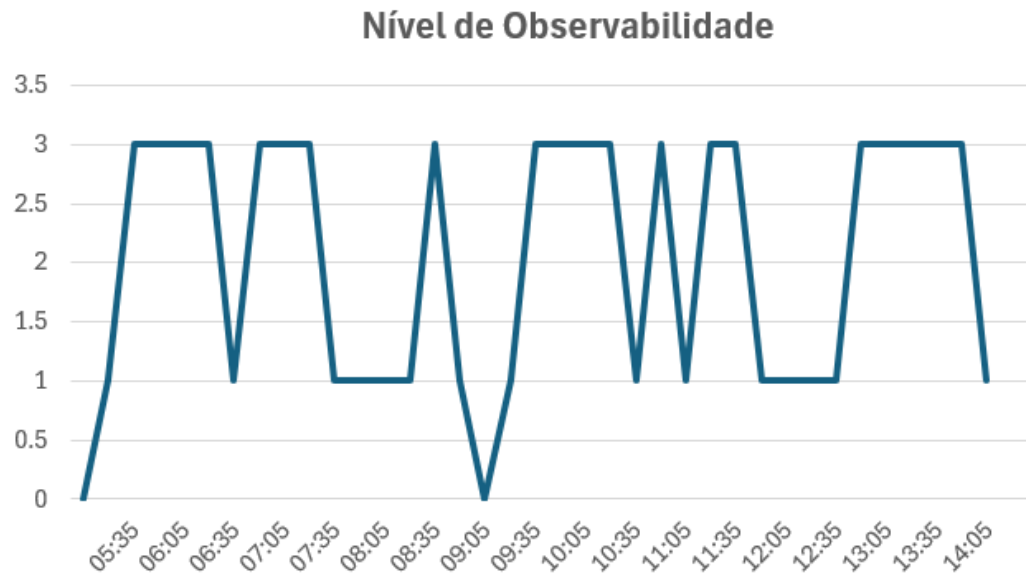


Figura 9.1: Medição do nível de observabilidade de um *SmartTruck* do Mobile IoT-RoadBot.

dos domínios estava disponível (por exemplo, apenas métricas ou apenas *logs*), o nível de observabilidade era 1. Já o valor zero (0) observado às 09:05 indica um momento em que nenhum dos dois domínios de instrumentação (métricas e *logs*) tinha dados disponíveis para análise.

Como visto na Seção 7.2, um maior nível de observabilidade representa um volume maior de informações da infraestrutura e dos sistemas em *Fog*, que vão apoiar a equipe de operação e manutenção na análise da causa-raiz dos problemas. Quanto mais rápida e efetiva for essa análise, mas rapidamente os serviços voltarão ao estado normal, o que contribui para o atingimento dos SLAs. Por exemplo, em um problema que tenha ocorrido durante todo o dia da jornada do mesmo *SmarTruck* representado na Figura 9.1, os momentos mais apropriados para se iniciar a investigação do problema a partir dos dados de observabilidade são os horários em que o nível de observabilidade está no máximo.

9.2 Ambiente de Testes

Esta seção apresenta a infraestrutura real, utilizada nesta tese para construir o ambiente de testes e realizar os experimentos.

9.2.1 Hardware

A infraestrutura de testes foi construída com quatro unidades Raspberry Pi 4 (dispositivos IoT), dois *Fog Nodes* e uma máquina virtual executando em *Cloud*, conforme descrito na Tabela 9.1. O Raspberry Pi 4 tem uma configuração de 4 GB de RAM e executa o Ubuntu 22.04 LTS. Cada unidade Raspberry Pi 4 reproduz os dados gerados por um *SmartTruck* real quando estava em serviço em 2022 nas ruas de Brimbank, na Austrália, executando o aplicativo Mobile-IoT-RoadBot [152]. Esses dispositivos e as conexões que os ligam à Camada de *Fog* vão executar a fase de coleta de dados do ODLC descrita na Seção 7.1.

Os próximos componentes do ambiente de teste são os *Fog Nodes*, dispositivos com 4 CPUs e 16GB de RAM, que hospedam as ferramentas de código aberto utilizadas para armazenar e analisar os dados

Tabela 9.1: Configuração de hardware do ambiente de testes.

Dispositivo	Configuração	Quantidade	Tipo
Raspberry PI 4B model	Quad-core Cortex A72, 4GB RAM	4	Dispositivo IoT
Fog server	Quad-core 16GB RAM	2	*Fog Node*
Máquina Virtual	8-core 32 GB RAM	1	Servidor *Cloud*

de observabilidade. Métricas, *logs* e *traces* vêm de dispositivos IoT, e são armazenados no *Fog Node*. Essa agregação de dados permite uma visão abrangente do *status* de cada componente desta aplicação distribuída de cidade inteligente. A tomada de decisão pode ser rápida após aplicar regras simples nos dados atualizados, criando alertas, atualizando painéis e realizando análises cruzadas entre domínios. É nessa etapa que a maior parte da fase de análise de dados ocorre.

A instância *e2-standard-2* do *Google Cloud* compreende 8 VCPUs, 32 GB de RAM e 200 GB de disco rígido. A máquina virtual no *Google Cloud* hospeda o armazenamento de longo prazo dos dados de observabilidade. Os dados de observabilidade têm a característica de crescer indefinidamente. Para controlar os recursos necessários que viabilizam uma tomada de decisão rápida na Camada de *Fog*, o volume de dados deve ser limitado no tempo. Neste experimento, o período de uma semana foi definido como o limite. No entanto, isso pode ser facilmente adaptado a cada aplicação. Assim, todos os dias, os dados que excedem esse limite são automaticamente movidos para a *Cloud*. Na *Cloud*, uma análise mais complexa e computacionalmente intensiva pode ser realizada sobre os dados históricos. A Figura 9.2 mostra o ambiente de teste como uma arquitetura de três camadas de *Fog Computing*, relacionando cada passo do ODLC aos dispositivos onde usualmente cada etapa acontece.

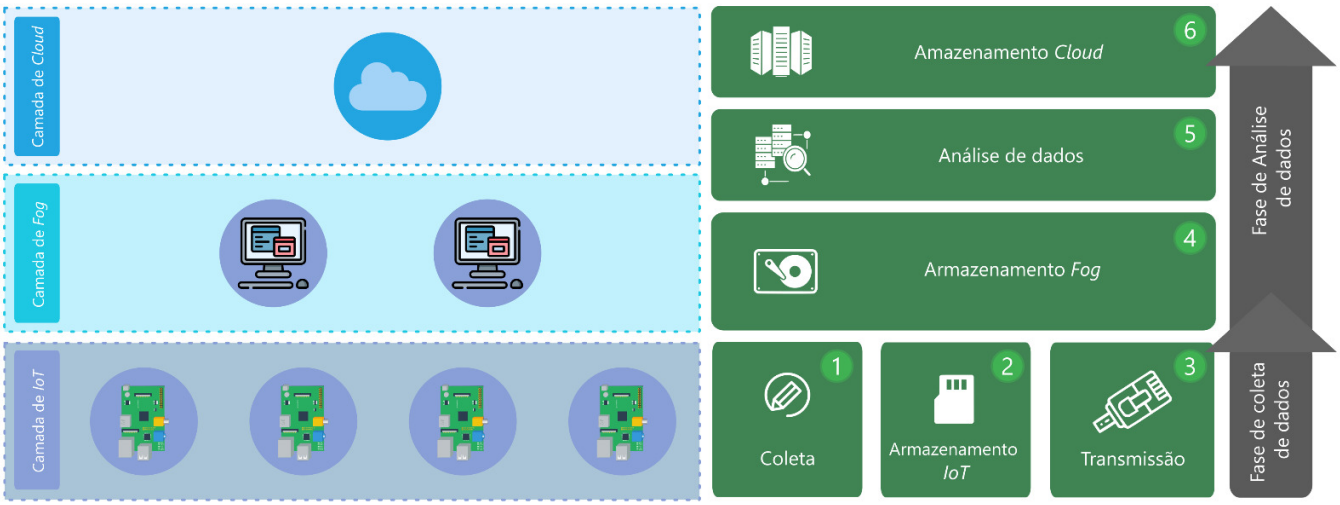


Figura 9.2: Arquitetura do ambiente de testes.

9.2.2 Software

Até a escrita desta tese, não foi encontrada nenhuma solução disponível que permitisse a coleta de métricas, *logs* e *traces*, simultaneamente, em um ambiente de *Fog Computing* [80], conforme demonstrado na avaliação do estado da arte da observabilidade em *Fog* (Capítulo 6). No entanto, algumas propostas analisadas no

Tabela 9.2: Ferramentas de código aberto implantadas no ambiente de testes.

Ferramenta	**Domínio**	**IoT**	***Fog***	***Cloud***	**Observação**
Node Exporter	Métricas	✓			Coleta métricas do dispositivo e as expõe via uma chamada HTTP.
Filebeat	*Logs*	✓			Monitora arquivos específicos e transmite seu conteúdo para o Elastic Search.
OpenTelemetry	*Traces*	✓			Bibliotecas de linguagem de programação para criar e transmitir *traces* para o Jaeger.
Prometheus	Métricas		✓	✓	Coleta métricas do Node Exporter em cada dispositivo IoT.
Elastic Search	*Logs*		✓	✓	Armazena *logs* transmitidos pelo Filebeat e dados de *traces* recebidos pelo Jaeger.
Jaeger	*Traces*		✓	✓	Recebe dados de *traces* transmitidos pelo SDK OpenTelemetry.
Grafana			✓	✓	Apresenta painéis de dados do Prometheus, Elastic Search e Jaeger.

Capítulo 6 utilizaram ferramentas de código aberto que implementavam fragmentos dos componentes propostos na arquitetura FogObserver [121,124,128]. Assim sendo, para investigar a viabilidade do aumento da observabilidade em *Fog*, a estratégia usada foi integrar diversas ferramentas que, em conjunto, oferecessem uma implementação da FogObserver, registrando os desafios encontrados e os resultados alcançados. As ferramentas de código aberto mais referenciadas na literatura foram selecionadas, e um resumo do propósito de uso de cada uma delas é apresentado na Tabela 9.2. As ferramentas selecionadas foram implantadas nos dispositivos utilizando *containers* Docker [97].

Os dispositivos que hospedam cada ferramenta, assim como o fluxo de dados entre elas, podem ser vistos na Figura 9.3. A máquina virtual na *Cloud* implementa o mesmo ambiente de software do *Fog Node* e não é mostrada na Figura 9.3, por simplicidade. O fluxo de dados da Camada de *Fog* para a Camada de *Cloud* é baseado em operações recorrentes de exportação e importação, respectivamente. Assim, as ferramentas de código aberto selecionadas para os experimentos, categorizadas pelo domínio de instrumentação que apoiam, são descritas a seguir.

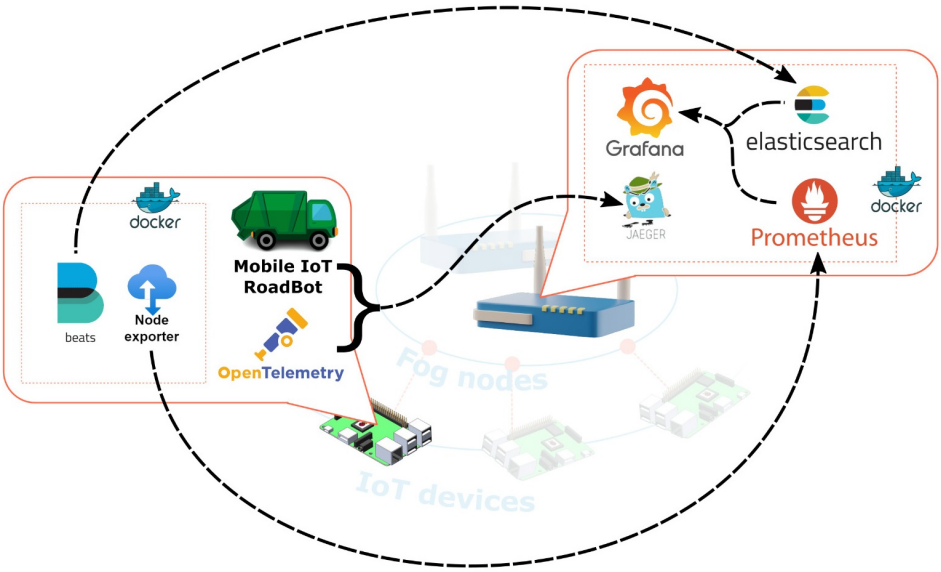


Figura 9.3: Fluxo de dados das ferramentas de observabilidade de código aberto.

- **Gerenciamento de Métricas -** Prometheus [120] é o padrão de código aberto para a coleta e armazenamento de métricas. Ele pode ser configurado para coletar as métricas dos dispositivos IoT em uma frequência definida (a cada 5s por padrão), e todas as métricas expostas são coletadas e armazenadas em um TSDB. O Prometheus foi implantado no *Fog Node*, e ele provê vários módulos definidos

no componente *Manager* da FogObserver, além de suprir parte do armazenamento local. Mas esses módulos funcionam apenas para o fluxo de métricas e não atendem aos fluxos de *logs* e de *traces*. Grafana [130] é um software leve de painel de dados que é usado para visualizar as métricas armazenadas no Prometheus. Neste experimento, ele também se conectará ao ElasticSearch [151] para mostrar informações de *log*, figurando como a interface do componente *Manager* da FogObserver. Prometheus se baseia em agentes externos que executam em dispositivos IoT para gerar dados de métricas e torná-los disponíveis para coleta. Neste trabalho, o Node Exporter [121] é utilizado como um módulo observador do componente *Collector* da FogObserver. Ele expõe métricas de hardware e de rede, como a porcentagem de CPU livre e a taxa de transferência da conexão de rede;

- **Gerenciamento de *Logs* -** ElasticSearch [151] é um banco de dados de índice invertido de código aberto. Ele é utilizado para indexação, armazenamento e recuperação rápida de dados de *logs* e de *traces* enviados para ele. Ele representa parte do armazenamento local do componente *Manager* da FogObserver. Filebeat [153] é um agente que executa localmente no dispositivo IoT, sendo, portanto, um módulo observador do componente *Collector* da FogObserver, enviando dados de *log* para o ElasticSearch;

  Gerenciamento de *Traces* - Jaeger [154] foi selecionado como uma ferramenta de gerenciamento de *traces* de código aberto. Ele pode ser configurado para usar ElasticSearch como armazenamento, fornece uma visualização adequada, e um tempo de resposta pequeno em consultas. Para coletar, armazenar e transmitir *traces* para Jaeger, a aplicação deve ser instrumentada com as chamadas que criam os *traces*. Neste experimento, Open Telemetry SDK [104], uma biblioteca de código aberto, será utilizado para instrumentar a funcionalidade de relatório do Mobile IoT-RoadBot e será o terceiro módulo observador dessa implementação da FogObserver.

9.3 Avaliação

Esta seção apresenta a avaliação experimental de ferramentas de observabilidade de código aberto usando um ambiente de teste de *Fog* do mundo real, executando uma aplicação IoT de cidade inteligente. Assim, serão mensurados o *overhead* dos componentes de software, e o volume de dados coletados e transmitidos. Além disso, são apresentadas e discutidas as descobertas sobre os benefícios e desafios de aumentar o nível de observabilidade em um ambiente de *Fog*. Esta avaliação demonstra que um ODLC pode ser implantado em um ambiente de *Fog* usando ferramentas de código aberto, e mostra os desafios e decisões de projeto que devem ser considerados.

9.3.1 Metodologia

Para avaliar o *overhead* de alcançar um nível mais alto de observabilidade em um ambiente de *Fog*, o ambiente de testes foi utilizado para medir: (i) o *overhead* de cada ferramenta nos dispositivos IoT; (ii) o *overhead* do componente de servidor de cada ferramenta executando em um *Fog Node*; (iii) o volume de dados de observabilidade coletados, armazenados e transmitidos por meio do ciclo de vida dos dados de observabilidade. Enquanto o Mobile IoT-RoadBot estava em execução em um Raspberry Pi, foram

instalados os agentes de coleta no mesmo dispositivo, e medido por quatro horas o *overhead* em termos de uso de CPU e de memória. Utilizou-se o System Activity Reporter (SAR) [155] para obter a média de uso de CPU e de memória a cada 5 minutos. Após registrar essas informações, desinstalou-se a ferramenta e repetiu-se o processo com as outras ferramentas (Node Exporter, Filebeat e OpenTelemetry SDK, descritos na Seção 9.2.2).

Assim sendo, foi realizado um procedimento similar no *Fog Node* para as ferramentas do lado do servidor (Prometheus, ElasticSearch e Jaeger, descritos na Seção 9.2.2). Após medir o *overhead*, o Mobile IoT-RoadBot executou por nove horas (uma jornada regular de um dia inteiro para cada *SmartTruck*) para avaliar o volume de dados coletados, por cada domínio de instrumentação, e transmitidos por meio das camadas da arquitetura do ambiente de testes. Finalmente, para avaliar os benefícios de alcançar um nível mais alto de observabilidade em ambientes de *Fog*, reproduziu-se por nove horas no ambiente de teste os dados reais de produção de quatro *SmartTrucks*. Os dados gerados por cada *SmartTruck* do mundo real são reproduzidos por um Raspberry Pi que também executa agentes de observabilidade e envia os dados por meio de ODLC dos dispositivos IoT para o *Fog Node* e, posteriormente, para a *Cloud*.

9.3.2 *Overhead*

As Figuras 9.4, 9.5, 9.6 e 9.7 mostram o uso de CPU e de memória de cada componente do conjunto de ferramentas de observabilidade de código aberto sendo avaliado neste trabalho. As Figuras 9.4 e 9.5 mostram o *overhead* dos dispositivos IoT. Assim, foi possível observar um *overhead* de CPU e de memória insignificante. A quantidade agregada de CPU quando todos os três componentes (ou seja, NodeExporter, Filebeat e OpenTelemetry SDK) são executados simultaneamente está abaixo de 12% em média. Em relação ao uso de memória, uma quantidade agregada de menos de 150MB de RAM é necessária.

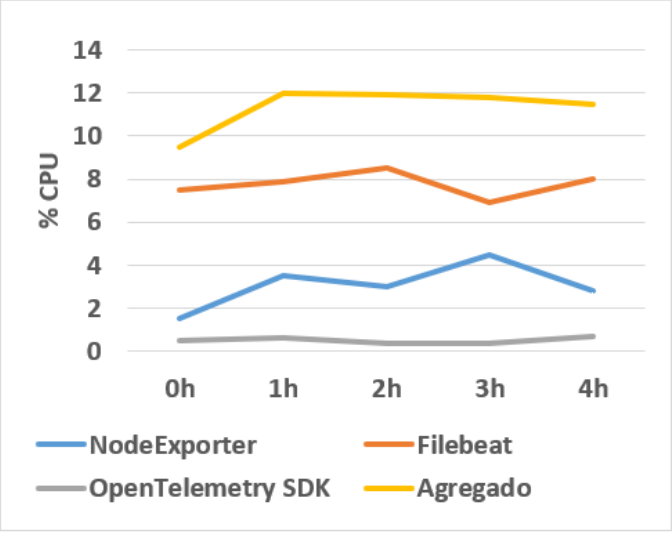


Figura 9.4: *Overhead* de CPU em IoT.

As Figuras 9.6 e 9.7 mostram o *overhead* no *Fog Node*. Ao contrário dos dispositivos IoT, o uso da CPU é maior. Isso é esperado, uma vez que o *Fog Node* lida com quatro vezes o volume de dados (enviados pelos quatro dispositivos IoT), com o propósito de receber, processar e armazenar dados. Uma média de uso da CPU de menos de 25% para esse tipo de processamento parece valer a pena. No entanto, isso impede que o

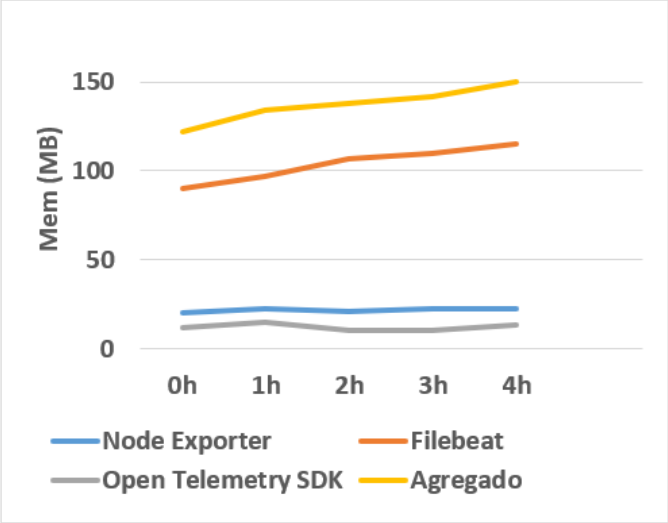


Figura 9.5: *Overhead* de memória em IoT.

lado do servidor da ferramenta de observabilidade esteja na camada IoT, na qual os dispositivos têm menos recursos. Para garantir um desempenho constante, o volume de dados armazenados foi limitado ao total de registros coletados em uma semana. Em termos de uso de memória nos *Fog Nodes*, o Prometheus alocou 400 MB de RAM em média, enquanto o Jaeger alocou cerca de 200 MB, um volume de memória baixo para a carga de dados de observabilidade do Mobile IoT-RoadBot. No entanto, ElasticSearch alocou quase 4,5 GB de RAM. O Mobile IoT-RoadBot possui um fluxo constante de coleta de dados e, geralmente, não gera picos de dados transmitidos. Todavia, ao lidar com uma aplicação mais intensiva em dados ou uma aplicação que apresenta um comportamento variável, o *overhead* do lado do servidor deve ser monitorado para garantir que sustente a carga necessária.

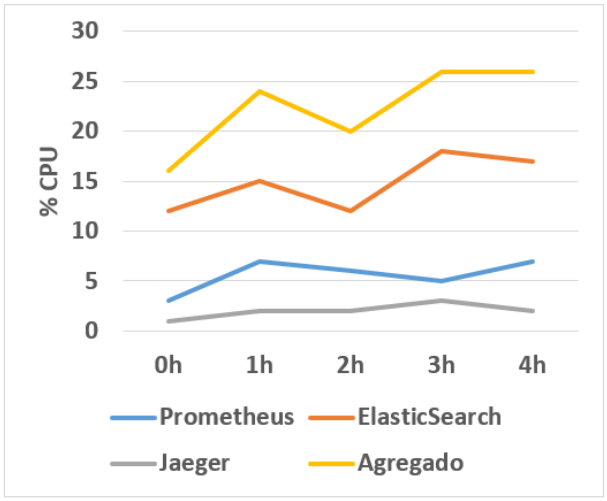


Figura 9.6: *Overhead* de CPU em *Fog*.

9.3.3 Análise Cruzada de Métricas, *Logs* e *Traces* em *Fog*

Na última subseção, o *overhead* adicionado à infraestrutura de *Fog*, após a implementação de ferramentas de observabilidade de código aberto que implementam um ODLC (Seção 7.1), foi detalhado. Esta sub-

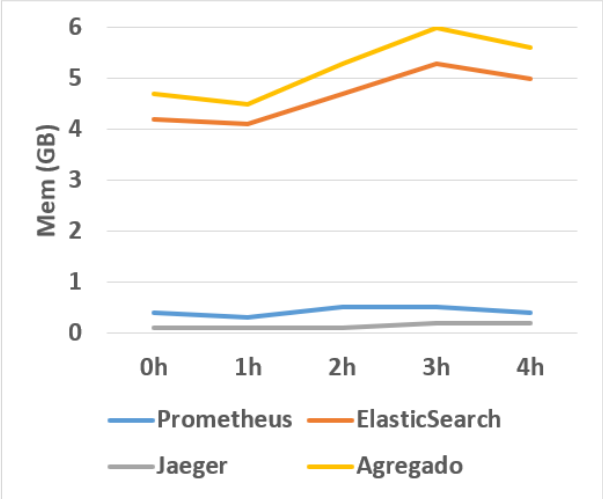


Figura 9.7: *Overhead* de memória em *Fog*.

seção mostrará os benefícios que o Mobile IoT-RoadBot poderia ter se estivesse usando tal conjunto de ferramentas de observabilidade.

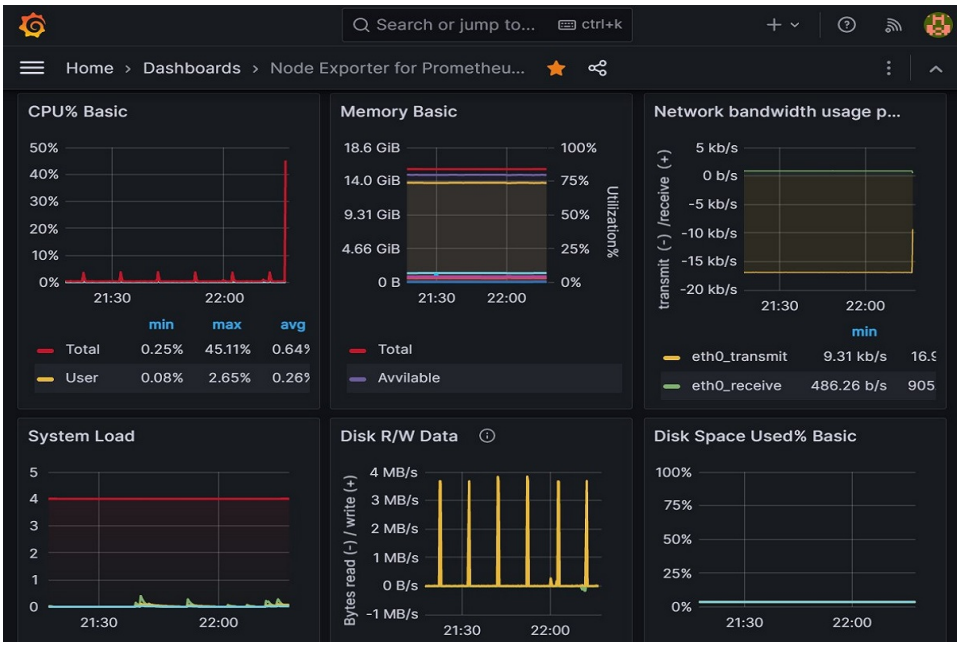


Figura 9.8: Visualização de dados de métricas na Camada de *Fog*.

As Figuras 9.8, 9.9, 9.10 e 9.11 mostram como os dados de observabilidade podem ser analisados em *Fog* usando as ferramentas de código aberto implantadas no ambiente de testes. A Figura 9.8 mostra o valor de algumas métricas (uso de CPU, memória e largura de banda) coletadas pelos experimentos durante a última hora a partir de dispositivos IoT. Caso alguma métrica esteja fora da faixa considerada segura, uma mensagem de alerta pode ser enviada à equipe de manutenção, permitindo uma ação rápida. A Figura 9.9 mostra a visualização dos *logs* coletados de dispositivos IoT a cada um segundo. Os *logs* podem ser facilmente pré-processados enquanto estão sendo ingeridos para identificar diferentes campos de informação, proporcionando consultas rápidas, e permitindo a gestão de alertas. A Figura 9.9 mostra o

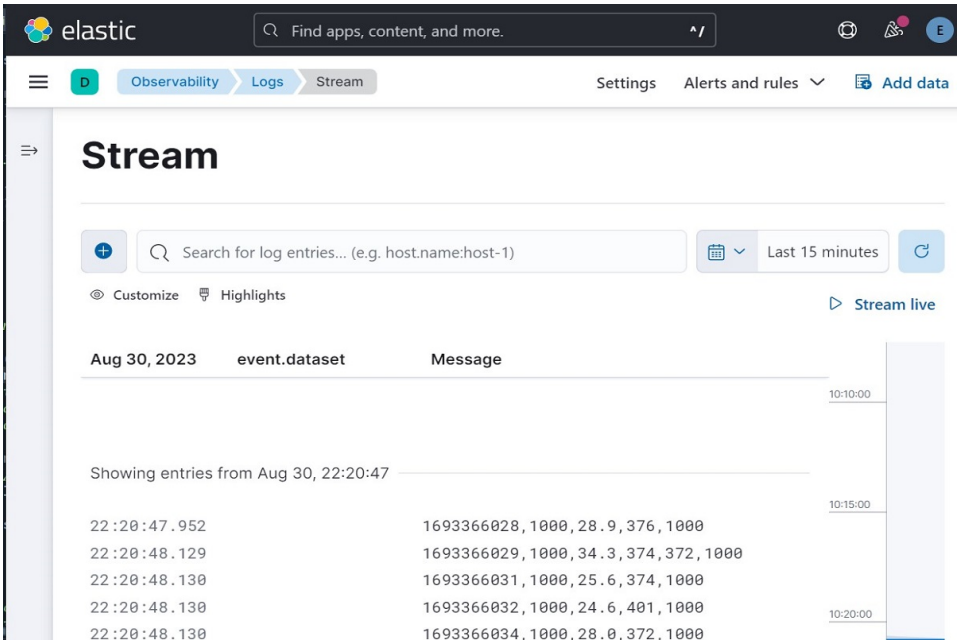


Figura 9.9: Visualização de dados de *logs* na Camada de *Fog*.

resultado de uma consulta feita sobre os dados de *log*, detalhando a latência entre o dispositivo IoT e alguns servidores de interesse na Internet.

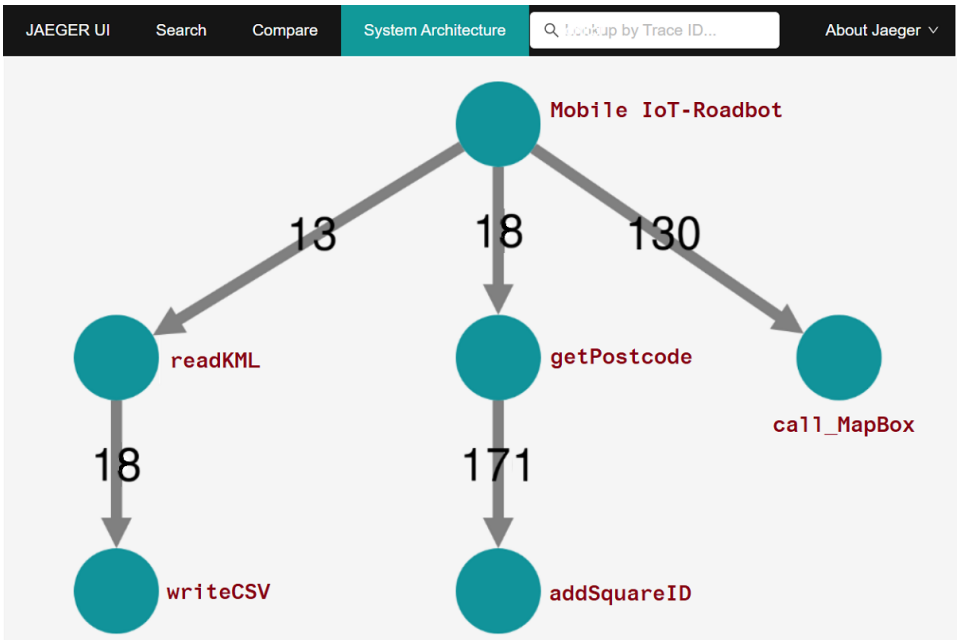


Figura 9.10: Visualização de dados de *traces* na Camada de *Fog*.

A Figura 9.10 mostra um gráfico de dependência, criado a partir dos *traces* coletados, em que é possível ver o atraso médio de cada componente dependente após centenas de solicitações, e identificar quais causam a maior parte do tempo de resposta. Esta informação é relevante para o planejamento de futuras melhorias de desempenho. Finalmente, a Figura 9.11 mostra os detalhes de um *trace* específico, em que é possível identificar os componentes que causam o maior tempo de resposta. Ao buscar a causa raiz de um problema identificado, essa informação é muito útil.

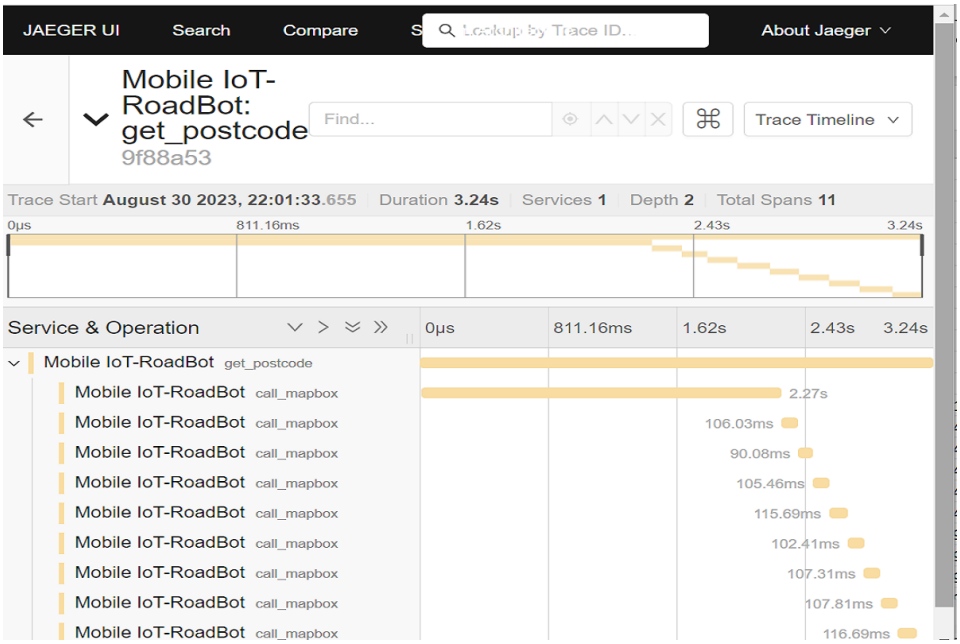


Figura 9.11: Visualização de dados de *traces* na Camada de *Fog*.

9.3.4 Discussão

O volume de dados de observabilidade gerenciado pelo ODLC durante os experimentos é apresentado na Tabela 9.3. O NodeExporter foi implementado com a configuração padrão. Embora seja uma ferramenta com um pequeno impacto em termos de uso de CPU e de memória [108], pode ter um impacto não negligenciável em termos do volume de dados que coleta. O conjunto padrão de métricas que ela expõe representa 65KB de informação. Esses dados são apresentados apenas quando o Prometheus os requisita usando uma chamada HTTP. Isso significa que não há armazenamento IoT para esses dados. Prometheus está configurado por padrão para coletar a página do NodeExporter a cada 5s, obtendo todas as métricas expostas e armazenando-as em seu TSDB no *Fog Node*. Considerando que há quatro dispositivos IoT expondo métricas, o volume de dados transmitido e armazenado no *Fog Node* é de cerca de 8,75GB ao longo de uma semana, o período em que esses dados estarão disponíveis para a tomada de decisões e outras análises na Camada *Fog*. Após atingir uma semana de idade, as informações são removidas do *Fog Node* e enviadas para a *Cloud* para armazenamento a longo prazo e análise histórica. Para fins de registro e de comparação, o volume na *Cloud* chegará a 75GB após dois meses de operação.

A saída padrão do NodeExporter fornece texto de ajuda para cada métrica, como mostrado na Figura 9.12. Essas informações representam pelo menos 20% do volume total de dados e devem ser removidas antes de se expor as métricas. O conjunto padrão de métricas é muito extenso e, provavelmente, nem todas

Tabela 9.3: Volume de dados de cada domínio de instrumentação na avaliação do Mobile IoT-Roadbot.

Ferramenta	**Domínio**	**Coleta de Dados**	**Período de Coleta**	**Volume por Hora**	**Armaz. em IoT**	**Armaz. em *Fog***	**Vol. em *Fog* (1 sem)**	**Armaz. na *Cloud***	**Vol. na *Cloud* (2 meses)**
Node Exporter	Métricas	65KB	5s	46 MB	Não	Sim	8.75 GB	Sim	75 GB
Filebeat	*Logs*	1KB	1s	3.50 MB	Sim	Sim	0.67 GB	Sim	5.77 GB
Open Telemetry	*Traces*	4KB	15s	1 MB	Não	Sim	0.2 GB	Sim	1.54 GB

as métricas são úteis para cada caso de uso. Por exemplo, o Node Exporter expõe dezenas de métricas do ambiente Go (Figura 9.12) que não são de interesse para o monitoramento do Mobile IoT-RoadBot, e podem ser removidas. Além de cortar as métricas que não são de interesse, soluções de Aprendizado de Máquina sobre dados históricos podem ser usadas para descobrir correlações entre métricas, e manter o conjunto de métricas alvo no mínimo [136]. Além disso, o período de coleta de métricas pode ser aumentado na configuração do Prometheus sem que haja perda relevante de oportunidades para atuação. Aumentar a espera para requisitar métricas para 10 segundos reduzirá à metade o volume de dados transmitidos para Camada de *Fog* e lá armazenados. Usando as estratégias de remover o texto de ajuda e mudar a configuração do Node Exporter para expor apenas métricas sobre CPU, memória, disco, rede e fonte de alimentação, e aumentando o período de coleta para 10 segundos no Prometheus, reduziu-se o volume de dados de métricas no *Fog Node* em 87%, o que também afetou positivamente o uso de CPU e de memória pelo Prometheus.

```
# TYPE go_memstats_frees_total counter
go_memstats_frees_total 697
# HELP go_memstats_gc_sys_bytes Number of bytes used for garbage collection system metadata.
# TYPE go_memstats_gc_sys_bytes gauge
go_memstats_gc_sys_bytes 2.991584e+06
# HELP go_memstats_heap_alloc_bytes Number of heap bytes allocated and still in use.
# TYPE go_memstats_heap_alloc_bytes gauge
```

Figura 9.12: Exemplo de saída padrão do Node Exporter.

Com relação aos *logs* gerados pelo Mobile IoT-Roadbot, enquanto os *SmartTrucks* se moviam pela cidade, eles registravam informações sobre a análise da rede 5G, tais como latência e *throughput*, além de informações contextuais (coordenadas GNSS, velocidade do *SmartTruck*, etc.). Assim, embora o aplicativo escreva informações nos *logs* a cada segundo, o volume de dados gravados é baixo (0,67 GB, em uma semana, conforme visto na Tabela 9.3), sendo menor do que o volume de dados gerados pelo Node Exporter após a aplicação de estratégias de redução de volume.

O Filebeat foi configurado para coletar apenas os *logs* escritos pelo Mobile IoT-RoadBot, e transmiti-los para a Camada *Fog*. Assim, foi necessário alterar a configuração padrão do Filebeat para desativar a função de descoberta automática. Quando essa função está ativa, o Filebeat recebe do gerenciador do Docker cada alteração de *status* de qualquer *container* no dispositivo, consumindo mais memória do que o necessário. Como o aplicativo não foi originalmente instrumentado para registrar *traces*, optou-se por incluir a geração de *traces* em uma função de relatório, utilizada para agregar dados 5G pelos subúrbios de Brimbank. Para realizar essa agregação, foram utilizadas coordenadas geográficas para encontrar o endereço completo da Austrália, usando um serviço chamado MapBox [156]. Dessa forma, conhecendo o volume de dados gerados por esses *traces*, estimou-se o volume de dados para gerar os *traces* da operação regular do Mobile IoT-RoadBot. Este caso de uso não tem a característica de lidar com rajadas de requisições, pois realiza o mesmo volume de operações enquanto está em serviço. Portanto, o volume de dados de *traces* é constante.

O volume de dados agregado, coletado pelos quatro dispositivos IoT, transmitido e armazenado no *Fog Node* por um período de uma semana, foi de aproximadamente 10GB, considerando a configuração padrão das ferramentas de código aberto utilizadas, conforme mostrado na Tabela 9.3. Usando as estratégias descritas nesta seção, o volume de dados agregado foi reduzido para 2GB, uma redução de 80%. Os quatro *SmartTrucks* cujos dados de observabilidade são reproduzidos pelos dispositivos IoT, neste experimento, transmitiram 291 GB de dados de vídeo utilizando a rede 5G em uma semana de operação no mundo real. Portanto, os dados de observabilidade (2GB) representariam um *overhead* de menos de 1% neste caso de

uso. Os experimentos mostram que é possível coletar os benefícios de alcançar um nível mais alto de observabilidade para um sistema em um ambiente de *Fog Computing*. Além disso, o *overhead* de implantar um ciclo de vida de dados de observabilidade pode ser baixo, quando gerenciado adequadamente.

A utilização de *container* Docker como o ambiente de execução para as ferramentas de observabilidade ajuda a enfrentar o desafio da heterogeneidade dos dispositivos de *Fog*. Devido à restrição de recursos dos dispositivos IoT, a coleta de dados de observabilidade deve ser feita por agentes leves. Além disso, o volume de dados deve ser minimizado para reduzir o risco de congestionamento da rede, e o aumento do *overhead* para coletar e transmitir os dados para a Camada de *Fog*. Cada domínio de instrumentação tem requisitos específicos de dados (Tabela 3.1) que devem ser atendidos para otimizar o armazenamento e minimizar o atraso médio na análise dos dados de observabilidade na Camada de *Fog* para tomada de decisão e atuação. Deixar na Camada de *Fog* apenas uma janela dos dados de observabilidade mais recentes é outra estratégia para lidar com a restrição de recursos dos *Fog Nodes*. Os dados que estão fora da janela de tempo permitida são enviados para a *Cloud* para armazenamento de longo prazo e análise histórica.

As Figuras 9.13 e 9.14 mostram como se deu a implementação da arquitetura FogObserver nesse estudo de caso com ferramentas de código aberto.

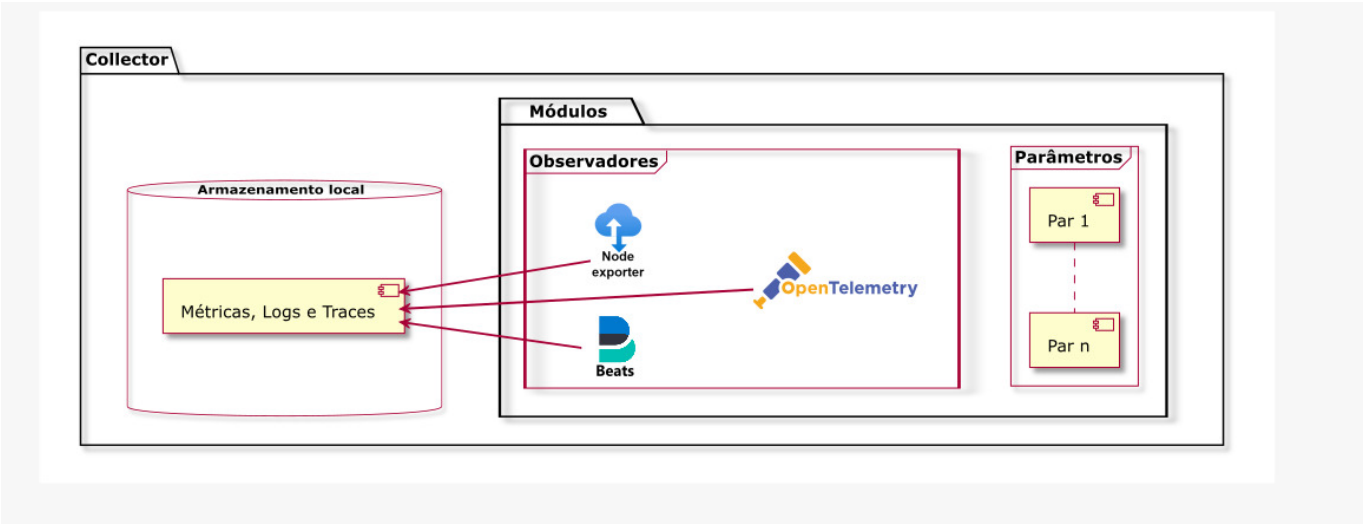


Figura 9.13: Componente *Collector* da arquitetura FogObserver implementado com ferramentas de código aberto.

No componente *Collector*, não houve um controle dinâmico de ativação e desativação dos módulos observadores. As ferramentas de código aberto selecionadas foram geridas de forma independente e manualmente. Na FogObserver, esse controle pode ser feito por meio dos parâmetros de execução, que são atualizados, quando oportuno, pelo módulo de comunicação. Esse cenário torna a atuação mais complexa e difícil de operar. Por exemplo, no ambiente dinâmico de *Fog Computing*, o Mobile IoT-RoadBot pode apresentar erros em alguns *SmartTrucks* enquanto está funcionando corretamente em outros. Nesses casos, se não houver recursos suficientes para transmitir todos os dados de observabilidade para a Camada *Fog*, uma decisão apropriada deve priorizar os dados desses *SmartTrucks* específicos e retornar à operação regular quando o problema for resolvido. Para implementar esse comportamento adaptativo e autônomo, pode ser necessário orquestrar o ciclo de vida dos dados de observabilidade e de seus observadores.

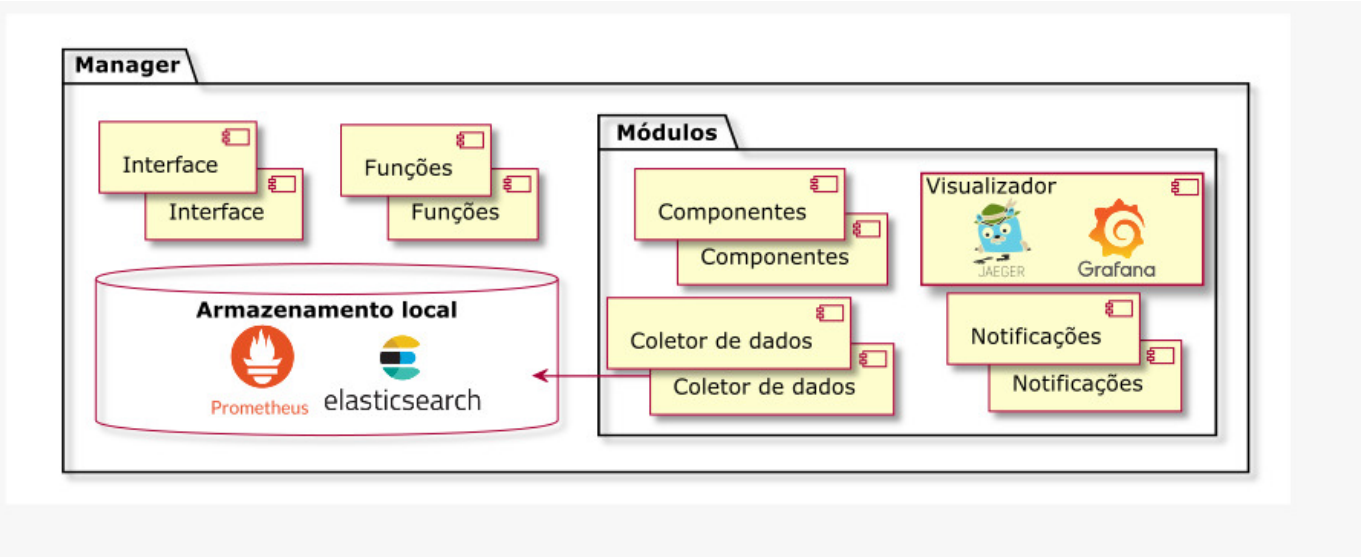


Figura 9.14: Componente *Manager* da arquitetura FogObserver implementado com ferramentas de código aberto.

No componente *Manager*, há uma superposição de funcionalidades que torna o ambiente mais complexo e com maior consumo de recursos. Enquanto Prometheus e ElasticSearch são complementares em relação aos tipos de armazenamento necessários para implantar a gestão de métricas, *logs* e *traces*, ambas as ferramentas possuem interfaces de gerenciamento próprias, que devem ser acessadas para configurar a aplicação de funções e o envio de notificações relativas à métricas (Prometheus), e relativas à *logs* e *traces* (ElasticSearch). Além disso, há duplicação dos módulos de componentes e de coleta de dados, uma vez que essas informações precisam ser atualizadas em cada uma das soluções. Em relação à visualização dos dados, também há superposição, uma vez que o Grafana se encarrega da visualização de métricas e de *logs*, mas no caso de *traces*, é o Jaeger quem faz essa ação.

O experimento descrito neste capítulo validou a hipótese de que é possível aumentar a observabilidade da infraestrutura e de sistemas em *Fog*. Com um caso de uso real de cidades inteligentes, foi possível implementar o ODLC para métricas, *logs* e *traces*, mantendo o *overhead* em nível baixo, o que viabilizou sua execução em um ambiente com restrição de recursos. No entanto, o uso de ferramentas independentes de código aberto agregou maior complexidade de operação e limitou a capacidade de reagir de forma autoadaptável às mudanças do ambiente.

9.4 Avaliação do *Framework* de Adaptabilidade

Esta seção apresenta a avaliação experimental do *framework* de adaptabilidade proposto nesta tese e apresentado no Capítulo 8.

9.4.1 Implementação do *Framework* de Adaptabilidade

A implementação do *framework* de adaptabilidade, detalhado na Seção 8.2, foi realizada em Python 3.11 [157]. Para comunicação entre os componentes do *framework*, foi utilizado o Mosquitto 2.0.20, um servidor de *Message Queuing Telemetry Transport* (MQTT). O MQTT é um protocolo leve de troca de mensagens,

baseado no modelo publicação/subscrição, ideal para aplicações IoT e redes de baixa largura de banda [158]. Diferentemente da seção anterior, em que um aplicativo real (Mobile IoT-RoadBot) é executado em um ambiente de testes, usando dados reais da operação dos *SmartTrucks*, esta implementação foca na validação do *framework* de adaptabilidade e de seus módulos, implementando o ciclo MAPE-K [51] para que reaja de forma autônoma a mudanças no ambiente. Assim sendo, para o *Collector*, os módulos implementam as seguintes funcionalidades:

- **Módulo de Decisão:** monitora os dados de observabilidade coletados e reage a mudanças com base em regras, criando eventos internos que vão ser processados pelo módulo de comportamento. Um arquivo de configuração contém as regras de valores mínimos de espaço em disco livre, de banda de rede e de bateria do dispositivo. O módulo verifica periodicamente (a cada 5 segundos, um valor ajustável) se a medição atual dos recursos está abaixo dos mínimos estabelecidos e cria eventos internos (Disco_Cheio, Rede_Insuficiente e Bateria_Baixa), caso esteja, que são enviados ao módulo de comportamento;
- **Módulo de Comportamento:** ao receber eventos internos a partir do módulo de decisão, reage efetuando modificações no ambiente. No caso de um dos três eventos internos implementados, escolhe a reação de alterar o *status* do *Collector* de Normal para Crítico, e cria um evento externo Alteracao_Status, notificando o módulo de comunicação para que envie mensagem ao *Manager*. No caso de o *Collector* estar com *status* Crítico e os valores dos recursos forem maiores que os valores mínimos constantes da regras, o processo é similar. Os eventos externos, enviados pelo *Manager*, que foram implementados são: Parar_Observadores, Executar_Observadores e Atualizar_Regras. Como a implementação apenas visa a validar a autoadaptabilidade que o *framework* provê, a ação associada aos dois primeiros eventos é modificar variáveis de controle, que representam a execução ou não de cada observador do *Collector*. No caso de Atualizar_Regras, os novos valores das regras são atualizados no arquivo de configuração e imediatamente aplicados, podendo gerar eventos internos, de acordo com o valor atual dos recursos;
- **Módulo de Comunicação:** envia ao *Manager*, e recebe dele, os eventos externos, fazendo uso de troca de mensagens MQTT. Antes de enviar a mensagem, inclui a identificação do *Collector* (chave única gerada na instalação). Verifica a identificação do *Manager* nas mensagens que recebe, descartando as que não contenham a chave correta. As mensagens contêm a ação a ser executada, os dados necessários à ação, o evento que a gerou, os valores atuais dos recursos e do *status*. Um registro da mensagem é feito no armazenamento local do *Collector* para viabilizar auditorias;
- **Interface do *Collector*:** exibe o identificador do dispositivo em que o *Collector* está executando e o *status* (Normal ou Crítico). Apresenta também se os observadores estão em execução ou parados, os valores atuais das regras vigentes e o *status* e valores dos recursos do *Manager* na última comunicação recebida. A interface do *Collector* pode ser visualizada na Figura 9.15. A interface permite alterar manualmente os valores dos recursos e das regras do próprio *Collector*, de forma a facilitar a validação da implementação e simular o funcionamento em um ambiente real de testes.

Em relação ao componente *Manager*, os módulos implementam as seguintes funcionalidades:

Figura 9.15: Interface do *Collector* evidenciando as informações usadas nas decisões adaptativas, assim como as mudanças de comportamento decorrentes.

- **Módulo de Decisão:** a implementação é similar ao módulo de decisão do *Collector*, mas com apenas dois recursos em acompanhamento: o percentual de disco livre e o percentual de CPU livre;
- **Módulo de Comportamento:** a implementação é similar à implementação do *Collector*, mas o *Manager* reage ao evento interno de Alterar_Status (de Normal para Crítico) criando o evento externo Parar_Observadores. Ao reconhecer que o *Manager* entrou em estado crítico, o objetivo desta implementação do *framework* de adaptabilidade é parar temporariamente a recepção de dados em *Fog*, de forma a preservar o *Fog Node* em que executa o *Manager*. Quando o *status* voltar para Normal, é criado o evento externo Executar_Observadores. Ao receber o evento externo de Alterar_*Status*, vindo de um *Collector*, a implementação atual deste módulo apenas atualiza a informação, de forma a permitir a visualização do *status* atualizado dos quatro *Collectors* sendo gerenciados;
- **Módulo de Comunicação:** implementação similar ao módulo de mesmo nome do *Collector*;
- **Interface do *Manager*:** exibe os valores atuais dos recursos em acompanhamento, o *status* e as regras vigentes. Exibe também os *status* e valores dos recursos dos *Collectors*, de acordo com a última comunicação recebida. A interface permite a atualização remota das regras que determinam os valores mínimos de recursos de um determinado dispositivo, que serão enviadas como evento externo ao dispositivo pelo módulo de comunicação. Para simular as mudanças de *status* autônomas, a interface permite a atualização dos valores atuais dos recursos do *Manager*, assim como o valor das regras vigentes para o *Manager*. A interface do *Manager* pode ser visualizada na Figura 9.16.

9.4.2 Resultados Alcançados

Os resultados alcançados com a implementação do *framework* de adaptabilidade validam a arquitetura proposta, demonstrando a viabilidade do ciclo MAPE-K [51] em *Fog* e a capacidade de resposta autônoma a mudanças no ambiente em um tempo curto e com baixo *overhead*. Dessa maneira, é possível destacar como principais resultados alcançados por meio do *framework* proposto nesta tese:

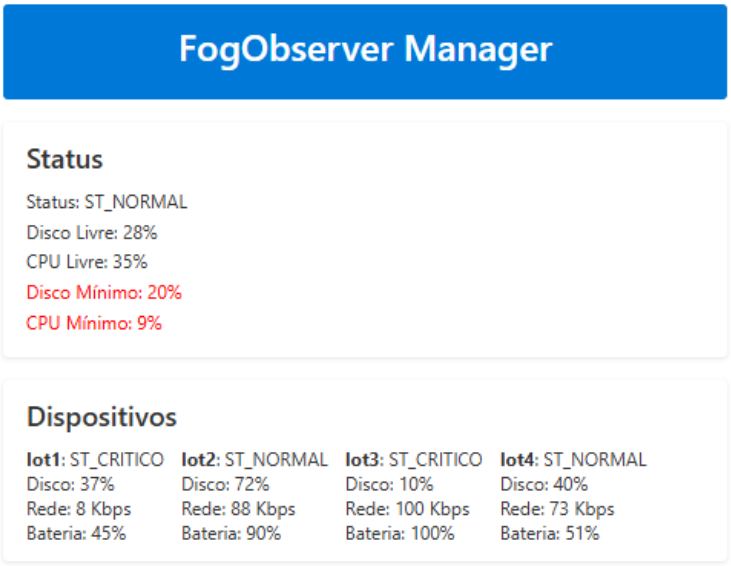


Figura 9.16: Interface do *Manager* evidenciando as informações recebidas dos *Collectors* gerenciados.

- **Comunicação Eficaz:** a troca de mensagens via MQTT entre os *Collectors* e o *Manager* provou ser eficiente, com baixo *overhead* e tempos de transmissão curtos. O tempo de transmissão das mensagens foi de 20 milessegundos, em média. É um reflexo da proximidade característica entre *Fog Nodes* e dispositivos IoT, garantindo respostas rápidas às mudanças no ambiente;
- **Adaptação Autônoma:** a implementação demonstrou a capacidade dos *Collectors* e do *Manager* de adaptarem seus comportamentos de forma autônoma, sem necessidade de intervenção manual, validando o conceito de autoadaptabilidade proposto;
- **Flexibilidade:** a capacidade de alterar dinamicamente os valores das regras de decisão dos *Collectors* valida a flexibilidade do *framework* e sua capacidade de adaptação a diferentes cenários e requisitos;
- **Baixo *Overhead***: o *overhead* introduzido pelo *framework* de adaptabilidade é negligenciável em comparação com o volume de dados de observabilidade. O tamanho médio da mensagem foi de 198 bytes, e o volume total de mensagens trocadas no período de teste foi de 20 Kilobytes, representando menos de 0,01% do volume de dados de observabilidade coletado e transmitido no mesmo período, conforme apresentado na Seção 9.3.4.

9.5 Considerações Finais

Este capítulo apresentou uma avaliação da proposta da tese em um contexto real de cidades inteligentes, utilizando o estudo de caso Mobile IoT-RoadBot. Assim, com os testes realizados, foi possível validar a implementação do Ciclo de Vida dos Dados de Observabilidade (ODLC) e da arquitetura FogObserver, em um ambiente de testes composto por dispositivos IoT, *Fog Nodes* e uma máquina virtual na *Cloud*. O *overhead* dos componentes de software foi mensurado, assim como o volume de dados coletados e transmitidos. Além disso, foi demonstrada uma implementação do *framework* de adaptabilidade, que reage de forma rápida e autônoma a mudanças no ambiente de execução, causando um *overhead* negligenciável.

Os resultados mostraram a viabilidade de aumentar a observabilidade de sistemas em um ambiente de *Fog Computing*, utilizando ferramentas de código aberto.

No próximo capítulo, serão apresentadas as conclusões desta pesquisa e as direções para trabalhos futuros, consolidando o conhecimento adquirido e abrindo novas possibilidades de investigação na área.

Parte III

Conclusão

Capítulo 10

Conclusão e Trabalhos Futuros

10.1 Conclusão

O objetivo geral desta tese foi investigar o aumento da observabilidade da infraestrutura e dos sistemas que executam em *Fog Computing*. A observabilidade é viabilizada nos sistemas distribuídos pela coleta de métricas, *logs* e *traces*. Ela permite compreender o desempenho dos sistemas de forma mais abrangente, agilizando a tomada de decisão e o retorno tempestivo dos sistemas a um estado operacional após uma falha. No entanto, aumentar a observabilidade em ambientes *Fog* apresenta desafios significativos devido à heterogeneidade dos dispositivos, à instabilidade das conexões e à restrição de recursos. Injetar um grande volume de dados adicionais de observabilidade na estrutura pode prejudicar o funcionamento do ambiente.

A pesquisa realizada nesta tese identificou uma lacuna na literatura, em que as propostas de orquestração de *Fog*, um processo fundamental para a gestão de serviços na plataforma, frequentemente supunham a existência de um serviço de gestão da observabilidade, sem detalhar sua implementação ou desafios. Por meio de uma revisão sistemática, disponível no Capítulo 2, identificou-se que a gestão da observabilidade era muito relevante para o funcionamento da orquestração, mas pouco estudada no contexto de *Fog*.

Essa lacuna conduziu a pesquisa desta tese para aprofundar o conhecimento sobre os modelos de operação, as funcionalidades disponíveis, e os desafios encontrados pelos sistemas de gestão da observabilidade de *Fog*, conteúdo disponível no Capítulo 3. Por meio de uma revisão sistemática, definiu-se uma nova taxonomia das características de soluções de gestão da observabilidade em *Fog Computing* (Capítulo 5). Essa taxonomia foi utilizada para caracterizar as 10 soluções de gestão da observabilidade de *Fog* disponíveis na literatura (Capítulo 6) e, com isso, entender quais desafios estavam sem atenção. Entre eles, o principal desafio era o aumento da observabilidade em *Fog*, alcançado por meio da coleta, armazenamento e disponibilização dos três domínios de instrumentação (métricas, *logs* e *traces*), simultaneamente.

Para viabilizar o aumento da observabilidade em *Fog*, foi mapeado o ciclo de vida dos dados de observabilidade, identificando suas etapas, os componentes do sistema de gestão da observabilidade que atuam em cada uma delas, assim como a camada em que elas ocorrem na arquitetura de *Fog*. Foi definido também um indicador para medir o nível da observabilidade de um sistema e feitas considerações acerca do controle do *overhead* da gestão da observabilidade, que deve ser mantido baixo para atender plenamente aos desafios de *Fog* (Capítulo 7).

Para endereçar os desafios encontrados na literatura, esta tese propõe FogObserver, uma arquitetura para sistemas de gestão da observabilidade em ambientes *Fog Computing*. Esta arquitetura, detalhada no Capítulo 8, promove a coleta, o gerenciamento e a entrega de dados de gestão da observabilidade, abrangendo métricas, *logs* e *traces*.

Para lidar com a dinamicidade e as restrições de recursos típicas dos ambientes *Fog*, a tese também apresenta um *framework* de adaptabilidade para a arquitetura FogObserver, que permite aos sistemas de gestão da observabilidade reagirem de forma autônoma a mudanças no ambiente. Esse *framework*, baseado no *loop* de controle MAPE-K, incorpora padrões de lógica de controle e é agnóstico quanto à topologia, podendo ser utilizado em diferentes configurações de sistemas de gestão da observabilidade.

Um estudo de caso com uma aplicação real de cidade inteligente foi realizado para avaliar o uso da arquitetura FogObserver. Os resultados demonstraram a viabilidade de aumentar a observabilidade em *Fog Computing* de forma eficaz, adicionando um *overhead* baixo à infraestrutura e aos canais de comunicação. Por meio de estratégias customizadas para o contexto da aplicação, conseguiu-se uma redução de 80% no volume de dados de observabilidade, transmitidos dos dispositivos IoT para *Fog*, e o volume resultante representou menos de 1% do volume de dados transmitidos pela aplicação quando em operação real. No entanto, o uso de ferramentas independentes de código aberto agregou maior complexidade de operação e limitou a capacidade de reagir de forma autoadaptável às mudanças do ambiente.

A implementação do *framework* de adaptabilidade demonstrou a capacidade de resposta autônoma a mudanças no ambiente com uso do *loop* de controle MAPE-K em *Fog*. Os resultados mostraram uma comunicação rápida e eficaz entre os componentes, com baixo *overhead*.

10.2 Trabalhos Futuros

A observabilidade não é um termo ou assunto novo [52], mas seu significado evoluiu no contexto dos serviços de IoT sustentados por ambientes de *Fog* [63, 146, 147]. É um assunto atual, e vários trabalhos estão tentando descobrir como usá-la para obter o maior valor com o menor custo, não apenas em termos financeiros, mas também em termos de uso de recursos. Esta seção apresenta algumas direções futuras para sistemas de gestão da observabilidade funcionando em ambientes de *Fog*.

Os recursos utilizados por um sistema de observabilidade competem com aqueles necessários para a aplicação de domínio e serviços [80]. Isso significa que um sistema de observabilidade para ambientes *Fog* deve manter uma baixa pegada de consumo de recursos computacionais, uma vez que dispositivos IoT e *Fog Nodes* podem ser restritos em recursos. Uma estratégia para reduzir o uso de recursos na fase de Coleta do ODLC é usar o Berkeley Packet Filter (eBPF) [159]. O eBPF permite a execução de programas personalizados em uma máquina virtual segura no *kernel* do sistema operacional do dispositivo de IoT, proporcionando um ponto de vista para monitorar atividades em nível de sistema. O eBPF pode ser uma ferramenta muito eficiente para monitorar o comportamento do sistema e detectar falhas [160]. O uso de *traces* é um esforço específico da aplicação, e implantar uma única infraestrutura de *traces* integrada em várias plataformas de aplicativos e componentes de *middleware* é desafiador [161]. Rezvani *et al.* [162] exploraram o uso do eBPF para fornecer não apenas métricas de hardware e rede, mas também métricas em nível de aplicativo para cargas de trabalho sensíveis à latência e que tenham interação direta com os

usuários. Assim sendo, pretende-se desenvolver um módulo observador, a ser integrado ao componente *Collector* da arquitetura FogObserver, que utilize o eBPF para coletar *traces* sem a necessidade de alterar as aplicações em execução nos dispositivos IoT.

Além disso, formatos de dados intercambiáveis podem permitir a conexão de várias ferramentas de coleta de dados de observabilidade, como por exemplo statsd [99], sensu [128], Amazon CloudWatch [163], com servidores de observabilidade comerciais ou de código aberto, que podem acumular dados de várias fontes distintas, proporcionando uma visão mais abrangente do ambiente para a equipe de manutenção e operacional [91]. Dessa forma, pretende-se adotar o padrão OpenTelemetry [104] na FogObserver para viabilizar a integração de diferentes ferramentas de coleta de dados de observabilidade.

A utilização da Inteligência Artificial é outra direção futura que pode ter um impacto relevante na área de observabilidade de *Fog*. Grandes volumes de dados podem dificultar a capacidade dos usuários de identificar gargalos da operação, e impactar o desempenho geral do sistema. Assim, selecionar apenas as métricas mais relevantes é essencial para reduzir o *overhead* de observabilidade, reduzir custos e melhorar a análise de desempenho [164]. No entanto, identificar essas métricas-chave, geralmente, depende da *expertise* profissional. Além disso, a importância de métricas específicas de desempenho pode flutuar ao longo do tempo com cargas de trabalho de aplicação variadas, tornando impraticável a seleção manual das métricas mais relevantes. Popiolek *et al.* [164] propuseram uma abordagem que utiliza correlação linear e análise de agrupamento hierárquico para reduzir a dimensionalidade dos dados por meio da seleção de métricas. Um menor número de métricas de maior qualidade pode ser mais valioso do que coletar uma infinidade de dados apenas porque é possível. Outra área na qual a Inteligência Artificial pode agregar valor é o apoio a comportamentos adaptativos [165], substituindo a atuação primária baseada em limiares por detecção de anomalias adaptativas. Uma estratégia de notificação baseada em detecção de anomalias apoia o estabelecimento de uma linha de base de utilização ao longo do tempo. Por exemplo, um pico de duas horas em 90% de uso de CPU, no último dia do mês, pode ser um evento regular do processamento de fim de mês e não deveria gerar um alerta, diferentemente de quando acontecesse no início do mês. Assim sendo, pretende-se usar Inteligência Artificial para identificar as métricas mais importantes para cada aplicação, assim como apontar os melhores limiares de acordo com o histórico de execução.

Em ambientes de *Fog Computing*, a capacidade de resposta rápida a eventos críticos é fundamental para garantir a disponibilidade e a confiabilidade dos serviços. No entanto, o fluxo constante de dados de observabilidade pode sobrecarregar o sistema, dificultando a identificação e o tratamento oportuno de informações de alta prioridade. Para mitigar esse problema, é essencial implementar mecanismos que permitam a priorização e o tratamento diferenciado de mensagens críticas, assegurando que recebam atenção imediata e que as decisões sejam tomadas em tempo hábil. Uma estratégia seria o estabelecimento de um canal bidirecional exclusivo para a troca de mensagens de alta prioridade e a modificação no processamento de mensagens, de forma que haja uma garantia de tratar exclusivamente o canal de mensagens prioritárias enquanto houver mensagens aguardando. Pretende-se alterar o módulo de comunicação dos componentes da arquitetura FogObserver para que reconheça e priorize as mensagens críticas, enviando-as através de canal exclusivo.

A arquitetura FogObserver foi definida e implementada usando topologias centralizada ou hierárquica. No entanto, em cenários específicos, uma abordagem distribuída, com uso de coreografia, por exemplo,

pode oferecer maior escalabilidade, resiliência e flexibilidade. Pretende-se explorar a adaptação da FogObserver para cenários de gestão da observabilidade com topologias distribuídas. Para viabilizar sua operação em um sistema que utiliza coreografia para implementar suas funcionalidades seria necessário a criação de novos módulos. Um módulo de negociação permitiria que os *Collectors* negociassem entre si os parâmetros de observabilidade, como a frequência de coleta de dados, os domínios de instrumentação a serem coletados e as políticas de compartilhamento de dados. Um modelo de consenso, baseado em algoritmos como Paxos [166], garantiria que todos os componentes concordassem com as decisões tomadas. Além disso, seria necessário alterar os módulos dos componentes atuais para retirar a dependência de um componente central. Para isso, é necessário que os componentes consigam descobrir outros componentes e que compartilhem os dados coletados entre si, por meio de sincronizações diferenciais de dados.

10.3 Produção Científica Derivada da Pesquisa Realizada Nesta Tese

A pesquisa realizada nesta tese permitiu ao autor participar do programa "Doutorado Sanduíche", financiado pela CAPES. Por seis meses, o autor fez pesquisas na Swinburne University of Technology (SUT), sediada em Melbourne, Austrália. O pesquisador Prem Prakash Jayaranam, diretor da Fábrica do Futuro [167], um laboratório de pesquisa aplicada ligado à SUT, foi o coorientador dos trabalhos realizados. O cenário motivador utilizado nesta tese (Seção 3.2) foi objeto real de pesquisa no período, que resultou em três artigos científicos publicados (Anexos III, IV e V).

A produção científica completa, derivada da pesquisa realizada nesta tese, está resumida na tabela 10.1. Esses artigos geraram até o momento[1] 243 citações, sendo o artigo "*Orchestration in Fog Computing: A Comprehensive Survey*" (Anexo I) o mais citado, com 213 citações.

Tabela 10.1: Produção científica gerada durante a pesquisa realizada nesta tese.

Anexo	Tipo	Local	Nome do Veículo de Publicação	Ano	Tema da Publicação
I [5]	A	Per	ACM Computing Surveys	2022	Orquestração em *Fog*
II [80]	A	Per	Computer Networks	2022	Taxonomia de sistemas de gestão da observabilidade
III [146]	A	Conf	Mobiquitous	2023	Ciclo de vida dos dados de observabilidade
IV [152]	A	Per	Internet of Things	2024	Estudo de caso de cidade inteligente
V [168]	C	Liv	Handbook of Data Engineering	2025	Arquitetura de sistemas de gestão da observabilidade

Abreviações – **A** = Artigo; **C** = Capítulo; **Conf** = Conferência; **Per** = Periódico; **Liv** = Livro;

[1] O número de citações foi coletado do Google Scholar em 20/04/2025

Referências

[1] Karumuri, Suman, Franco Solleza, Stan Zdonik e Nesime Tatbul: *Towards observability data management at scale*. ACM SIGMOD Record, 49(4):18–23, 2021. xiv, 2, 16, 18, 19, 22, 24, 27, 55, 59, 60, 66, 70, 71

[2] 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. ACM, 2012, ISBN 978-1-4503-1519-7. http://doi.acm.org/10.1145/2342509.2342513. 1, 7, 13

[3] Velasquez, K, D P Abreu, D Goncalves, L Bittencourt, M Curado, E Monteiro e E Madeira: *Service orchestration in fog environments*. Em *Proceedings - 2017 IEEE 5th International Conference on Future Internet of Things and Cloud, FiCloud 2017*, volume 2017-Janua, páginas 329–336, 2017, ISBN 978-1-5386-2074-8. 1

[4] Forti, Stefano, Marco Gaglianese e Antonio Brogi: *Lightweight self-organising distributed monitoring of fog infrastructures*. Future Generation Computer Systems, 114:605–618, 2021. 1, 52, 54, 55, 81

[5] Costa, Breno, Joao Bachiega, Leonardo Rebouças de Carvalho e Aleteia P. F. Araujo: *Orchestration in fog computing: A comprehensive survey*. ACM Comput. Surv., 55(2), jan 2022, ISSN 0360-0300. https://doi.org/10.1145/3486221. 1, 2, 6, 13, 14, 17, 80, 104

[6] Marie-Magdelaine, Nicolas: *Observability and resources managements in cloud-native environements*. Tese de Doutoramento, Université de Bordeaux, 2021. 2

[7] 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. 2, 22, 26, 29, 41, 43, 47

[8] Taherizadeh, Salman, Andrew C Jones, Ian Taylor, Zhiming Zhao e Vlado Stankovski: *Monitoring self-adaptive applications within edge computing frameworks: A state-of-the-art review*. Journal of Systems and Software, 136:19–38, 2018. 2, 27, 29, 47

[9] Abreha, Haftay Gebreslasie, Carlos J Bernardos, Antonio De La Oliva, Luca Cominardi e Arturo Azcorra: *Monitoring in fog computing: state-of-the-art and research challenges*. International Journal of Ad Hoc and Ubiquitous Computing, 36(2):114–130, 2021. 2, 47

[10] Battula, Sudheer Kumar, Saurabh Garg, James Montgomery e Byeong Kang: *An efficient resource monitoring service for fog computing environments*. IEEE Transactions on Services Computing, 13(4):709–722, 2019. 2, 38, 47, 51, 54, 55, 57, 81

[11] 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. 6

[12] 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. 6

[13] Foster, Ian, Yong Zhao, Ioan Raicu e Shiyong Lu: *Cloud computing and grid computing 360-degree compared*. Em *2008 grid computing environments workshop*, páginas 1–10. Ieee, 2008. 6

[14] 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. 7

[15] Mell, Peter e Tim Grance: *The nist definition of cloud computing*. National institute of science and technology, special publication, 800(2011):145, 2011. 7

[16] 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. 7

[17] 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. 7

[18] 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. 7

[19] Keahey, Katarzyna, Maurício Tsugawa, Andréa Matsunaga e José A. B. Fortes: *Sky computing*. Em *IEEE Internet Computing*, página 43–51. IEEE Computer Society, 2009. 7

[20] Shirinkin, Kirill: *Getting Started with Terraform*. Packt Publishing Ltd, 2017. 7

[21] Cloudify: *Getting started*, 2025. https://cloudify.co/getting-started/, acesso em 2025-02-23. 7

[22] Yi, Shanhe, Cheng Li e Qun Li: *A Survey of Fog Computing*. Proceedings of the 2015 Workshop on Mobile Big Data - Mobidata '15, páginas 37–42, 2015. http://dl.acm.org/citation.cfm?doid=2757384.2757397. 7, 53

[23] Mahmud, Md e Rajkumar Buyya: *Fog Computing: A Taxonomy, Survey and Future Directions*, páginas 1–10. Elsevier, novembro 2016, ISBN 978-981-10-5861-5. 7, 53

[24] Iorga, Michaela, Larry Feldman, Robert Barton, Michael Martin, Nedim Goren e Charif Mahmoudi: *Fog computing conceptual model*, 2018-03-14 2018. 7, 8

[25] 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. 8

[26] 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. 9

[27] 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. 9, 80

[28] Schwiegelshohn, Uwe, Rosa M Badia, Marian Bubak, Marco Danelutto, Schahram Dustdar, Fabrizio Gagliardi, Alfred Geiger, Ladislav Hluchy, Dieter Kranzlmüller, Erwin Laure *et al.*: *Perspectives on grid computing*. Future Generation Computer Systems, 26(8):1104–1115, 2010. 9

[29] 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, 98(December 2018), 2019, ISSN 13837621. 9

[30] 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. 9, 10

[31] Coulouris, George, Jean Dollimore e Tim Kindberg: *Distributed systems: Concepts and design. 3rd edition.* Addison Wesley, 2000. 9

[32] Mann, Zoltán Ádám: *Notions of architecture in fog computing*. Computing, 103(1):51–73, 2021. 9

[33] Goldberg, Robert P: *Survey of virtual machine research*. Computer, 7(6):34–45, 1974. 9, 10

[34] OpenFog: *Openfog consortium architecture working group*, February 2017. 10, 27

[35] Bachiega, Joao, Breno Gustavo Soares da Costa e Aleteia P. F. Araujo: *Computational perspective of the fog node*. 22nd International Conference on Internet Computing & IoT, 2021. 10, 60

[36] Dastjerdi, Amir Vahid, Harshit Gupta, Rodrigo N Calheiros, Soumya K Ghosh e Rajkumar Buyya: *Fog computing: Principles, architectures, and applications*. Em *Internet of things*, páginas 61–75. Elsevier, 2016. 10

[37] Dolui, Koustabh e Soumya Kanti Datta: *Comparison of edge computing implementations: Fog computing, cloudlet and mobile edge computing*. GIoTS 2017 - Global Internet of Things Summit, Proceedings, 2017. 10

[38] Ranaweera, Pasika, Anca Delia Jurcut e Madhusanka Liyanage: *Survey on multi-access edge computing security and privacy*. IEEE Communications Surveys & Tutorials, 23(2):1078–1124, 2021. 10

[39] Riaz, Nida, Saad Qaisar, Mudassar Ali e Muhammad Naeem: *Node selection and utility maximization for mobile edge computing–driven iot*. Transactions on Emerging Telecommunications Technologies, página e3704, 2019. 10

[40] Habibi, Pooyan, Mohammad Farhoudi, Sepehr Kazemian, Siavash Khorsandi e Alberto Leon-garcia: *Fog Computing : A Comprehensive Architectural Survey*. IEEE Access, PP:1, 2020. 11

[41] Hubaux, J P, Thomas Gross, J Y Le Boudec e Martin Vetterli: *Toward self-organized mobile ad hoc networks: the terminodes project*. IEEE Communications Magazine, 39(1):118–124, 2001. 11

[42] Ray, Partha Pratim: *An introduction to dew computing: Definition, concept and implications*. IEEE Access, 6:723–737, 2017. 11

[43] Xiao, Yong e Marwan Krunz: *Adaptivefog: A modelling and optimization framework for fog computing in intelligent transportation systems*. IEEE Transactions on Mobile Computing, 21(12):4187–4200, 2021. 12

[44] Anderson, John L: *Autonomous systems intelligence*. Em *Proceedings of the 1983 annual conference on Computers: Extending the human resource*, páginas 229–233, 1983. 13

[45] Peltz, C: *Web services orchestration and composition*. Computer, 36(10):46–52, 2003. 13

[46] Wen, Zhenyu, Renyu Yang, Peter Garraghan, Tao Lin, Jie Xu e Michael Rovatsos: *Fog orchestration for internet of things services*. IEEE Internet Computing, 21(2):16–24, 2017. 13

[47] Jiang, Yuxuan, Zhe Huang e Danny HK Tsang: *Challenges and solutions in fog computing orchestration*. IEEE Network, 32(3):122–129, 2017. 13

[48] Kitchenham, Barbara, O Pearl Brereton, David Budgen, Mark Turner, John Bailey e Stephen Linkman: *Systematic literature reviews in software engineering–a systematic literature review*. Information and software technology, 51(1):7–15, 2009. 13, 37

[49] Petersen, Kai, Robert Feldt, Shahid Mujtaba e Michael Mattsson: *Systematic mapping studies in software engineering.* Em *Ease*, volume 8, páginas 68–77, 2008. 13, 37

[50] 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. 15, 32

[51] Kephart, Jeffrey O e David M Chess: *The vision of autonomic computing*. Computer, 36(1):41–50, 2003. 15, 25, 32, 72, 75, 77, 81, 96, 97

[52] Kalman, Rudolf E: *On the general theory of control systems*. Em *Proceedings First International Conference on Automatic Control, Moscow, USSR*, páginas 481 – 492, 1960. 18, 102

[53] Marie-Magdelaine, Nicolas, Toufik Ahmed e Gauthier Astruc-Amato: *Demonstration of an observability framework for cloud native microservices*. Em *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, páginas 722–724. IEEE, 2019. 19

[54] Gharaibeh, Ammar, Mohammad A Salahuddin, Sayed Jahed Hussini, Abdallah Khreishah, Issa Khalil, Mohsen Guizani e Ala Al-Fuqaha: *Smart cities: A survey on data management, security, and enabling technologies*. IEEE Communications Surveys & Tutorials, 19(4):2456–2501, 2017. 19

[55] Forkan, Abdur Rahim Mohammad, Yong Bin Kang, Felip Marti, Shane Joachim, Abhik Banerjee, Josip Karabotic Milovac, Prem Prakash Jayaraman, Chris McCarthy, Hadi Ghaderi e Dimitrios Georgakopoulos: *Mobile IoT-RoadBot: An AI-Powered Mobile IoT Solution for Real-Time Roadside Asset Management*. Em *Proceedings of MobiCom 22*, página 883–885. ACM, 2022, ISBN 9781450391818. 19

[56] Fizza, Kaneez, Abhik Banerjee, Prem Prakash Jayaraman, Nitin Auluck, Rajiv Ranjan, Karan Mitra e Dimitrios Georgakopoulos: *A survey on evaluating the quality of autonomic internet of things applications*. IEEE Communications Surveys & Tutorials, 2022. 21

[57] 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. 21

[58] Syed, Hassan Jamil, Abdullah Gani, Raja Wasim Ahmad, Muhammad Khurram Khan e Abdelmuttlib Ibrahim Abdalla Ahmed: *Cloud monitoring: A review, taxonomy, and open research issues*. Journal of Network and Computer Applications, 98:11–26, 2017. 21, 38, 40

[59] 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. 22, 48, 54, 57, 80, 81

[60] CNCF - Cloud Native Computing Foundation: *Observability whitepaper.*, 2025. https://github.com/cncf/tag-observability/blob/whitepaper-v1.0.0/whitepaper.md, acesso em 23/02/2025. 22

[61] Hausenblas, Michael: *Cloud Observability in Action*. Manning, 2023. 22, 43

[62] Jordahl, Kelsey: *Geopandas documentation*, 2025. URL:http://sethc23.github.io/wiki/Python/geopandas, acesso em 23/02/2025. 23

[63] Sigelman, Benjamin H, Luiz André Barroso, Mike Burrows, Pat Stephenson, Manoj Plakal, Donald Beaver, Saul Jaspan e Chandan Shanbhag: *Dapper, a large-scale distributed systems tracing infrastructure*. Relatório Técnico, Google, 2010. 24, 102

[64] Khan, Mohd Ehmer e Farmeena Khan: *A comparative study of white box, black box and grey box testing techniques*. International Journal of Advanced Computer Science and Applications, 3(6), 2012. 24

[65] Ewaschuk, Rob e Betsy Beyer: *Monitoring distributed systems. site reliability engineering: How google runs production systems, chapter 6*, 2016. 25, 55

[66] Kubernetes: *Um sistema de código aberto para automatizar a implantação, dimensionamento e gerenciamento de aplicativos em contêiner.* https://kubernetes.io/, acesso em 23/02/2025. 25, 52

[67] Ifrah, Shimon: *Getting Started with Containers in Google Cloud Platform*, Chapter 8, páginas 221–258. Springer, 2021. 25

[68] Chang, Hsien Tsung, Yi Min Chang e Sheng Yuan Hsiao: *Scalable network file systems with load balancing and fault tolerance for web services*. Journal of Systems and Software, 93:102–109, 2014. 27

[69] Ahmed, Arif e Ejaz Ahmed: *A survey on mobile edge computing*. Em *2016 10th International Conference on Intelligent Systems and Control (ISCO)*, páginas 1–8, 2016. 27

[70] Xiao, Zheng, Pijun Liang, Zhao Tong, Kenli Li, Samee U Khan e Keqin Li: *Self-adaptation and mutual adaptation for distributed scheduling in benevolent clouds*. Concurrency and Computation: Practice and Experience, 29(5):e3939, 2017. 27

[71] Toosi, Adel Nadjaran, Rodrigo N Calheiros e Rajkumar Buyya: *Interconnected cloud computing environments: Challenges, taxonomy, and survey*. ACM Computing Surveys (CSUR), 47(1):1–47, 2014. 27

[72] EdgeXFoundry: *Uma plataforma de edge de código aberto.*, 2022. https://www.edgexfoundry.org/, acesso em 23/02/2025. 27

[73] OpenEdge: *Iniciativa para edge computing aberto.*, 2025. http://openedgecomputing.org/, acesso em 23/02/2025. 27

[74] Lee, In e Kyoochun Lee: *The internet of things (iot): Applications, investments, and challenges for enterprises*. Business horizons, 58(4):431–440, 2015. 27

[75] Fournier, Fabiana, Alexander Kofman, Inna Skarbovsky e Anastasios Skarlatidis: *Extending event-driven architecture for proactive systems.* Em *EDBT/ICDT Workshops*, páginas 104–110, 2015. 27

[76] Farris, Ivan, Tarik Taleb, Miloud Bagaa e Hannu Flick: *Optimizing service replication for mobile delay-sensitive applications in 5g edge network*. Em *2017 IEEE International Conference on Communications (ICC)*, páginas 1–6. IEEE, 2017. 28

[77] Sultan, Sari, Imtiaz Ahmad e Tassos Dimitriou: *Container security: Issues, challenges, and the road ahead*. IEEE Access, 7:52976–52996, 2019. 28

[78] Krupitzer, Christian, Felix Maximilian Roth, Sebastian VanSyckel, Gregor Schiele e Christian Becker: *A survey on engineering approaches for self-adaptive systems*. Pervasive and Mobile Computing, 17:184–206, 2015. 31

[79] Barba, Ademir José e Fernando Antonio de Castro Giorno: *A reference architecture for the iot services' adaptability-using agents to make iot services dynamically reconfigurable.* Em *IoTBDS*, páginas 187–194, 2018. 31, 32, 33, 45, 72, 77, 78, 80

[80] 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. 31, 37, 81, 85, 102, 104

[81] Muccini, Henry, Mohammad Sharaf e Danny Weyns: *Self-adaptation for cyber-physical systems: a systematic literature review*. Em *Proceedings of the 11th international symposium on software engineering for adaptive and self-managing systems*, páginas 75–81, 2016. 32

[82] Weyns, Danny, Bradley Schmerl, Vincenzo Grassi, Sam Malek, Raffaela Mirandola, Christian Prehofer, Jochen Wuttke, Jesper Andersson, Holger Giese e Karl M Göschka: *On patterns for decentralized control in self-adaptive systems*. Em *Software Engineering for Self-Adaptive Systems II: International Seminar, Dagstuhl Castle, Germany*, páginas 76–107. Springer, 2013. 32, 72

[83] Osinga, Frans PB: *Science, strategy and war: The strategic theory of John Boyd*. Routledge, 2007. 32

[84] Muccini, Henry e Mahyar Tourchi Moghaddam: *A cyber-physical space operational approach for crowd evacuation handling*. Em *Software Engineering for Resilient Systems: 9th International Workshop, SERENE 2017, Geneva, Switzerland, September 4–5, 2017, Proceedings 9*, páginas 81–95. Springer, 2017. 32

[85] Bali, Ahmed, Mahmud Al-Osta, Soufiene Ben Dahsen e Abdelouahed Gherbi: *Rule based autoscalability of iot services for efficient edge device resource utilization*. Journal of Ambient Intelligence and Humanized Computing, 11:5895–5912, 2020. 32

[86] Lam, An Ngoc, Oystein Haugen e Jerker Delsing: *Dynamical orchestration and configuration services in industrial iot systems: An autonomic approach*. IEEE Open Journal of the Industrial Electronics Society, 3:128–145, 2022. 32

[87] Serhani, M Adel, Hadeel T El-Kassabi, Khaled Shuaib, Alramzana N Navaz, Boualem Benatallah e Amine Beheshti: *Self-adapting cloud services orchestration for fulfilling intensive sensory data-driven iot workflows*. Future Generation Computer Systems, 108:583–597, 2020. 32

[88] McKinley, Philip K, Seyed Masoud Sadjadi, Eric P Kasten e Betty HC Cheng: *Composing adaptive software*. Computer, 37(7):56–64, 2004. 32

[89] Ramirez, Andres J e Betty HC Cheng: *Design patterns for developing dynamically adaptive systems*. Em *Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*, páginas 49–58, 2010. 32, 33, 80

[90] Aceto, Giuseppe, Alessio Botta, Walter De Donato e Antonio Pescapè: *Cloud monitoring: A survey*. Computer Networks, 57(9):2093–2115, 2013. 38

[91] Ward, Jonathan Stuart e Adam Barker: *Observing the clouds: a survey and taxonomy of cloud monitoring*. Journal of Cloud Computing, 3(1):1–30, 2014. 38, 55, 103

[92] Rosa Righi, Rodrigo da, Matheus Lehmann, Marcio Miguel Gomes, Jeferson Campos Nobre, Cristiano André da Costa, Sandro José Rigo, Marcio Lena, Rodrigo Fraga Mohr e Luiz Ricardo Bertoldi de Oliveira: *A survey on global management view: toward combining system monitoring, resource management, and load prediction*. Journal of Grid Computing, 17(3):473–502, 2019. 38

[93] Usman, Muhammad, Ricardo Britto, Jürgen Börstler e Emilia Mendes: *Taxonomies in software engineering: A systematic mapping study and a revised taxonomy development method*. Information and Software Technology, 85:43–59, 2017. 38

[94] Masip, Xavi, Eva Marín, Jordi Garcia e Sergi Sànchez: *Collaborative mechanism for hybrid fog-cloud scenarios*. Fog and Fogonomics: Challenges and Practices of Fog Computing, Communication, Networking, Strategy, and Economics, páginas 7–60, 2020. 40, 72

[95] Bachiega, João, Breno G. S. Costa e Aletéia Patricia Favacho Araújo: *Computational perspective of the fog node*. Em *2021 World Congress in Computer Science, Computer Engineering, & Applied Computing (CSCE'21)*, 2021. 42

[96] Madhavapeddy, Anil e David J Scott: *Unikernels: the rise of the virtual library operating system*. Communications of the ACM, 57(1):61–69, 2014. 42

[97] *Docker - empowering App Development for Developers*. `https://www.docker.com/`. 42, 86

[98] Morton, A: *Active and passive metrics and methods (with hybrid types in-between)*. Relatório Técnico, IETF, 2016. 44

[99] StatsD: *Statsd protocol*, 2025. `https://github.com/etsy/statsd/wiki`, acesso em 23/02/2025. 44, 51, 103

[100] Taherizadeh, Salman e Vlado Stankovski: *Auto-scaling applications in edge computing: Taxonomy and challenges*. Em *Proceedings of the International Conference on Big Data and Internet of Thing*, páginas 158–163, 2017. 44

[101] Popiolek, Pedro Freire e Odorico Machado Mendizabal: *Monitoring and analysis of performance impact in virtualized environments*. Journal of Applied Computing Research, 2:75–82, 2012. 44

[102] Popiolek, Pedro F, Karina S Machado e Odorico M Mendizabal: *Reducing monitoring overhead in virtualized environments through feature selection*. Em *Anais do XXXVI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, páginas 15–28. SBC, 2018. 44

[103] OpenMetrics: *Openmetrics*, 2025. `https://openmetrics.io/`, acesso em 23/02/2025. 45, 49

[104] OpenTelemetry: *Opentelemetry*, 2025. `https://opentelemetry.io/`, acesso em 23/02/2025. 45, 87, 103

[105] OpenCensus: *Opencensus*, 2025. `https://opencensus.io/`, acesso em 23/02/2025. 45

[106] OpenTracing: *Opentracing*, 2025. `https://github.com/opentracing/specification`, acesso em 23/02/2025. 45

[107] Okanović, Dušan, André van Hoorn, Christoph Heger, Alexander Wert e Stefan Siegl: *Towards performance tooling interoperability: An open format for representing execution traces*. Em *European Workshop on Performance Engineering*, páginas 94–108. Springer, 2016. 45

[108] Grossmann, Marcel e Christian Schenk: *A comparison of monitoring approaches for virtualized services at the network edge*. Em *2018 International Conference on Internet of Things, Embedded Systems and Communications (IINTEC)*, páginas 85–90. IEEE, 2018. 47, 49, 54, 57, 81, 92

[109] Nagios: *Nagios*, 2025. `http://www.nagios.org/`, acesso em 23/02/2025. 47

[110] Zabbix: *Zabbix*, 2025. `http://www.zabbix.com/`, acesso em 23/02/2025. 47

[111] Povedano-Molina, Javier, Jose M Lopez-Vega, Juan M Lopez-Soler, Antonio Corradi e Luca Foschini: *Dargos: A highly adaptable and scalable monitoring architecture for multi-tenant clouds*. Future Generation Computer Systems, 29(8):2041–2056, 2013. 47

[112] De Chaves, Shirlei Aparecida, Rafael Brundo Uriarte e Carlos Becker Westphall: *Toward an architecture for monitoring private clouds*. IEEE Communications Magazine, 49(12):130–137, 2011. 47

[113] Trihinas, Demetris, George Pallis e Marios D Dikaiakos: *Jcatascopia: Monitoring elastically adaptive applications in the cloud*. Em *2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, páginas 226–235. IEEE, 2014. 47

[114] Großmann, Marcel e Clemens Klug: *Monitoring container services at the network edge*. Em *2017 29th International Teletraffic Congress (ITC 29)*, volume 1, páginas 130–133. IEEE, 2017. 47, 48, 53, 54, 57, 79, 80, 81

[115] Monit: *Monit*, 2025. `https://mmonit.com/monit/`, acesso em 23/02/2025. 48

[116] PyMon: *Pymon online*, 2025. `https://github.com/whatever4711/PyMon`, acesso em 23/02/2025. 48

[117] *Marathon: A container orchestration platform for Mesos and DC/OS*, 2025. `https://github.com/d2iq-archive/marathon`, acesso em 23/02/2025. 48, 57

[118] Balouek, Daniel, Alexandra Carpen Amarie, Ghislain Charrier, Frédéric Desprez, Emmanuel Jeannot, Emmanuel Jeanvoine, Adrien Lèbre, David Margery, Nicolas Niclausse, Lucas Nussbaum *et al.*: *Adding virtualization capabilities to the grid'5000 testbed*. Em *International Conference on Cloud Computing and Services Science*, páginas 3–20. Springer, 2012. 48

[119] FMonE: *Fmone online*, 2025. `https://github.com/Brandonage/execo-utilities-g5k`, acesso em 23/02/2025. 49

[120] Prometheus: *Prometheus*, 2025. `https://prometheus.io/`, acesso em 23/02/2025. 49, 55, 79, 80, 86

[121] Prometheus: *Node exporter download page*, 2025. `https://prometheus.io/download/#node_exporter`, acesso em 23/02/2025. 49, 86, 87

[122] *Cadvisor*, 2025. `https://github.com/google/cadvisor`, acesso em 23/02/2025. 49

[123] Trakadas, Panagiotis, Panagiotis Karkazis, H C Leligou, Theodore Zahariadis, Wouter Tavernier, Thomas Soenen, Steven Van Rossem e L Miguel Contreras Murillo: *Scalable monitoring for multiple virtualized infrastructures for 5g services*. Em *SoftNetworking 2018, The International Symposium on Advances in SDN and NFV*, páginas 1–4, 2018. 49

[124] Stack, Prometheus: *Prometheus stack online*, 2025. https://github.com/uniba-ktr/docker-swarm-monitor, acesso em 23/02/2025. 49, 86

[125] Souza, Arthur, Nélio Cacho, Ayman Noor, Prem Prakash Jayaraman, Alexander Romanovsky e Rajiv Ranjan: *Osmotic monitoring of microservices between the edge and cloud*. Em *2018 IEEE 20th International Conference on High Performance Computing and Communications*, páginas 758–765. IEEE, 2018. 49, 53, 54, 55, 57, 81

[126] 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. 49

[127] Mourlin, Fabrice e Charif Mahmoudi: *Monitoring architecture for fog and mobile cloud*. Em *2018 17th International Symposium on Parallel and Distributed Computing (ISPDC)*, páginas 109–117. IEEE, 2018. 50, 54, 55, 57, 79, 81

[128] Porter, Sean: *Sensu and the art of monitoring.*, 2025. https://github.com/sensu/, acesso em 23/02/2025. 50, 57, 86, 103

[129] *Graphite*, 2025. https://graphiteapp.org/, acesso em 23/02/2025. 50

[130] Grafana Labs: *An open source visualisation tool.*, 2025. https://grafana.com/, acesso em 23/02/2025. 50, 87

[131] Taherizadeh, Salman, Vlado Stankovski e Marko Grobelnik: *A capillary computing architecture for dynamic internet of things: Orchestration of microservices from edge devices to fog and cloud providers*. Sensors, 18(9):2938, 2018. 50, 53, 54, 55, 57, 81

[132] IBM: *An architectural blueprint for autonomic computing*. Relatório Técnico, IBM, 2006. 50

[133] Switch: *The switch monitoring system*, 2025. https://github.com/salmant/ASAP/tree/master/SWITCH-Monitoring-System, acesso em 23/02/2025. 50, 51

[134] Apache: *Apache cassandra*, 2025. http://cassandra.apache.org/, acesso em 23/02/2025. 51

[135] Taherizadeh, Salman e Vlado Stankovski: *Dynamic multi-level auto-scaling rules for containerized applications*. The Computer Journal, 62(2):174–197, 2019. 51

[136] Bali, Ahmed e Abdelouahed Gherbi: *Rule based lightweight approach for resources monitoring on iot edge devices*. Em *Proceedings of the 5th International Workshop on Container Technologies and Container Clouds*, páginas 43–48, 2019. 51, 54, 55, 57, 81, 93

[137] Krahn, Robert, Donald Dragoti, Franz Gregor, Do Le Quoc, Valerio Schiavoni, Pascal Felber, Clenimar Souza, Andrey Brito e Christof Fetzer: *Teemon: A continuous performance monitoring framework for tees*. Em *Proceedings of the 21st International Middleware Conference*, páginas 178–192, 2020. 52, 54, 55, 57, 81

[138] TEEMon: *Teemon online*, 2025. https://sconedocs.github.io/teemon/, acesso em 23/02/2025. 52

[139] Brogi, Antonio, Stefano Forti e Marco Gaglianese: *Measuring the fog, gently*. Em *International Conference on Service-Oriented Computing*, páginas 523–538. Springer, 2019. 52, 55, 57

[140] Gaglianese, Marco, Stefano Forti, Federica Paganelli e Antonio Brogi: *Lightweight self-adaptive cloud-iot monitoring across fed4fire+ testbeds*. Em *IEEE INFOCOM 2022-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, páginas 1–6. IEEE, 2022. 53

[141] FogMon: *Fogmon online*, 2025. https://github.com/di-unipi-socc/FogMon/tree/liscio-2.0, acesso em 23/02/2025. 53

[142] Colombo, Vera, Alessandro Tundo, Michele Ciavotta e Leonardo Mariani: *Towards self-adaptive peer-to-peer monitoring for fog environments*. Em *17th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS '22)*. IEEE, 2022. 53, 55

[143] FogMon, Adaptive: *Adaptive fogmon online*, 2025. https://github.com/veracoo/FogMon/tree/adaptive-fogmon, acesso em 23/02/2025. 53

[144] Bittencourt, Luiz Fernando, Márcio Moraes Lopes, Ioan Petri e Omer F Rana: *Towards virtual machine migration in fog computing*. Em *2015 10th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC)*, páginas 1–8. IEEE, 2015. 53

[145] Kaldor, Jonathan, Jonathan Mace, Michał Bejda, Edison Gao, Wiktor Kuropatwa, Joe O'Neill, Kian Win Ong, Bill Schaller, Pingjia Shan, Brendan Viscomi *et al.*: *Canopy: An end-to-end performance tracing and analysis system*. Em *Proceedings of the 26th symposium on operating systems principles*, páginas 34–50, 2017. 55

[146] Costa, Breno, Abhik Banerjee, Prem Prakash Jayaraman, Leonardo R. Carvalho, João Bachiega e Aleteia Araujo: *Achieving Observability on Fog Computing with the Use of Open-Source Tools*. Em *Mobile and Ubiquitous Systems: Computing, Networking and Services*, páginas 319 – 340. Springer Nature Switzerland, 2024, ISBN 978-3-031-63992-0. 58, 72, 80, 102, 104

[147] Karumuri, Suman, Franco Solleza, Stan Zdonik e Nesime Tatbul: *Cloud observability: A melting pot for petabytes of heterogenous time series.* Em *CIDR*, 2021. 59, 61, 80, 102

[148] Dastjerdi, Amir Vahid e Rajkumar Buyya: *Fog computing: Helping the internet of things realize its potential*. Computer, 49(8):112–116, 2016. 60

[149] Hashem, Ibrahim Abaker Targio, Ibrar Yaqoob, Nor Badrul Anuar, Salimah Mokhtar, Abdullah Gani e Samee Ullah Khan: *The rise of "big data" on cloud computing: Review and open research issues*. Information systems, 47:98–115, 2015. 61

[150] McGrane, Martin e Simon K Poon: *A method to estimate high-dimensional synergistic interactions: A case study on information technology business value*, 2011. 62

[151] Elastic: *The ELK Stack*, 2025. https://www.elastic.co/what-is/elk-stack, acesso em 23/02/2025. 80, 87

[152] Banerjee, Abhik, Breno Costa, Abdur Rahim Mohammad Forkan, Yong Bin Kang, Felip Marti, Chris McCarthy, Hadi Ghaderi, Dimitrios Georgakopoulos e Prem Prakash Jayaraman: *5g enabled smart cities: A real-world evaluation and analysis of 5g using a pilot smart city application*. Internet of Things, 28:101326, 2024, ISSN 2542 - 6605. 84, 104

[153] Filebeat: *Lightweight shipper for logs.*, 2025. https://www.elastic.co/guide/en/beats/filebeat/current/filebeat-overview.html, acesso em 23/02/2025. 87

[154] Jaeger: *Open source, end-to-end distributed tracing.*, 2025. https://www.jaegertracing.io/, acesso em 23/02/2025. 87

[155] SysStat: *Sar - system activity report.*, 2025. https://github.com/sysstat/sysstat, acesso em 23/02/2025. 88

[156] MapBox, 2025. https://www.mapbox.com/, acesso em 23/02/2025. 93

[157] Python Software Foundation: *Python programming language.*, 2025. https://python.org/, acesso em 23/02/2025. 95

[158] Eclipse Foundation: *Mosquitto message broker*, 2025. https://mosquitto.org/, acesso em 23/02/2025. 96

[159] Gregg, Brendan: *BPF performance tools*. Addison-Wesley Professional, 2019. 102

[160] Levin, Joshua e Theophilus A Benson: *Viperprobe: Rethinking microservice observability with ebpf*. Em *2020 IEEE 9th International Conference on Cloud Networking (CloudNet)*, páginas 1–8, 2020. 102

[161] Neves, Francisco, Ricardo Vilaça e José Pereira: *Detailed black-box monitoring of distributed systems*. ACM SIGAPP Applied Computing Review, 21(1):24–36, 2021. 102

[162] Rezvani, Mohammadreza, Ali Jahanshahi e Daniel Wong: *Characterizing in-kernel observability of latency-sensitive request-level metrics with ebpf*. Em *2024 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, páginas 24 – 35. IEEE, 2024. 102

[163] Amazon Web Services: *Amazon cloudwatch*, 2025. https://aws.amazon.com/pt/cloudwatch/, acesso em 23/02/2025. 103

[164] Popiolek, Pedro Freire, Karina dos Santos Machado e Odorico Machado Mendizabal: *Low overhead performance monitoring for shared infrastructures*. Expert Systems with Applications, 171:114558, 2021. 103

[165] Bridges, Andre: *Monitoring and observability for infrastructure and applicationsl*, 2025. https://www.gartner.com/document/5486695, acesso em 23/02/2025. 103

[166] Lamport, Leslie: *Paxos made simple*. ACM SIGACT News (Distributed Computing Column) 32, 4 (Whole Number 121, December 2001), páginas 51–58, 2001. 104

[167] *Factory of the Future*, 2025. https://www.swinburne.edu.au/research/platforms-initiatives/factory-of-the-future/, acesso em 23/02/2025. 104

[168] Araujo, Aleteia, Breno Costa, Joao Bachiega Jr, Leonardo R Carvalho e Rajkumar Buyya: *Observability in fog computing*. arXiv preprint arXiv:2411.17753, 2024. 104

Anexo I

Artigo publicado na ACM Computing Surveys

Orchestration in Fog Computing: A Comprehensive Survey

BRENO COSTA, JOAO BACHIEGA JR., LEONARDO REBOUÇAS DE CARVALHO, and ALETEIA P. F. ARAUJO, University of Brasilia, Brazil

Fog computing is a paradigm that brings computational resources and services to the network edge in the vicinity of user devices, lowering latency and connecting with cloud computing resources. Unlike cloud computing, fog resources are based on constrained and heterogeneous nodes whose connectivity can be unstable. In this complex scenario, there is a need to define and implement orchestration processes to ensure that applications and services can be provided, considering the settled agreements. Although some publications have dealt with orchestration in fog computing, there are still some diverse definitions and functional intersection with other areas, such as resource management and monitoring. This article presents a systematic review of the literature with focus on orchestration in fog computing. A generic architecture of fog orchestration is presented, created from the consolidation of the analyzed proposals, bringing to light the essential functionalities addressed in the literature. This work also highlights the main challenges and open research questions.

CCS Concepts: • **General and reference** → **Surveys and overviews**; • **Computer systems organization** → **Cloud computing**; **Distributed architectures;**

Additional Key Words and Phrases: Fog computing, orchestration, monitoring, resource management

ACM Reference format:
Breno Costa, Joao Bachiega Jr., Leonardo Rebouças de Carvalho, and Aleteia P. F. Araujo. 2022. Orchestration in Fog Computing: A Comprehensive Survey. *ACM Comput. Surv.* 55, 2, Article 29 (January 2022), 34 pages.
https://doi.org/10.1145/3486221

1 INTRODUCTION

Cloud computing is already a mature paradigm that has been in place since 2006 [83]. It offers virtualized resources, created upon a huge shared infrastructure, that are consumed by the customers on a pay-per-use business model [139] and with a variety of cost options [19, 62]. Cloud computing is based on dozens of large datacenters, distributed around the world, that are placed in the center of the network and accessed by the Internet. Cloud computing also provides automatic scalability to its services.

Cloud computing datacenters are composed of thousands of resource-rich homogeneous physical servers that are interconnected by a redundant and stable network [157]. To optimize the infrastructure in use and comply with service and quality agreements made with the customers, a resource orchestration framework is in place.

Authors' address: B Costa, J. Bachiega Jr., L. R. de Carvalho, and A. P. F. Araujo, University of Brasilia, Department of Computer Science, Campus Darcy Ribeiro, Asa Norte, Brasilia, DF, 70910-900, Brazil; emails: {brenogcosta, joao.bachiega.jr, leouesb}@gmail.com, aleteia@unb.br.

0360-0300/2022/01-ART29 $15.00
https://doi.org/10.1145/3486221

Anexo II

Artigo publicado na Computer Networks

Computer Networks 215 (2022) 109189

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Review article

Monitoring fog computing: A review, taxonomy and open challenges

Breno Costa *, João Bachiega Jr., Leonardo Rebouças Carvalho, Michel Rosa, Aleteia Araujo

Department of Computer Science - University of Brasilia, Brazil

ARTICLE INFO

Keywords:
Monitoring
Orchestration
Fog computing
Taxonomy
Fog monitoring

ABSTRACT

Fog computing is a distributed paradigm that provides computational resources in the users' vicinity. Fog orchestration is a set of functionalities that coordinate the dynamic infrastructure and manage the services to guarantee the Service Level Agreements. Monitoring is an orchestration functionality of prime importance. It is the basis for resource management actions, collecting status of resource and service and delivering updated data to the orchestrator. There are several cloud monitoring solutions and tools, but none of them comply with fog characteristics and challenges. Fog monitoring solutions are scarce, and they may not be prepared to compose an orchestration service. This paper updates the knowledge base about fog monitoring, assessing recent subjects in this context like observability, data standardization and instrumentation domains. We propose a novel taxonomy of fog monitoring solutions, supported by a systematic review of the literature. Fog monitoring proposals are analyzed and categorized by this new taxonomy, offering researchers a comprehensive overview. This work also highlights the main challenges and open research questions.

1. Introduction

Fog computing is a computational paradigm that complements cloud computing, providing computational resources on the network edge, closer to the users. As a distributed infrastructure, fog computing must deal with heterogeneity of network links and processing capacity of its composing nodes [1]. These characteristics bring complexity to fog management, and it is addressed by the orchestration of services and resources. Orchestration is a management function, composed of several complementary functionalities. It is responsible for dealing with infrastructure dynamicity, for taking timely actions and for assuring that Service Level Agreements (SLAs) are respected [2]. There are several proposals of fog service orchestration in the literature, although most of them, only conceptual.

Monitoring is a functionality of prime importance and it is crucial to properly orchestrate fog services [3]. It collects updated status information about fog nodes and communication links and send them to the orchestrator. With an updated view of fog infrastructure and service execution, the orchestrator can take proper actions to guarantee the SLAs, e.g., offloading a service to a resource richer node and optimizing service placement according to historic data about node failures [4]. Besides the heterogeneity of nodes being monitored, there are other related concerns about frequency, topology, and communication model. There is a trade-off between the frequency of information updates and the overhead to the nodes and to the orchestrator related to generating, transmitting and processing status data. In such a dynamic scenario, adaptability of monitoring parameters can play an important role. In our previous work [4], we did a systematic literature review of fog service orchestration and analyzed 50 proposals. Most of them (40 out of 50) highlighted monitoring as a relevant process, but they frequently assumed that a fog monitoring solution would be available to deliver the information they needed, without presenting either implementation methods or insightful information on the subject.

Monitoring is not only about reporting availability, i.e. the capacity to answer the question of whether a node or a service is online and working properly. It is also about the capacity to explain why a node or service stopped working properly. The former is achieved by monitoring metrics, e.g. service response time. The latter is achieved by monitoring logs, i.e. unstructured strings of text, and traces, i.e. records of requests made by an user in a service. Metrics, logs and traces form what is called Instrumentation Domains of monitoring [5]. Different instrumentation domains can be used simultaneously by a monitoring solution to get different perspectives of a service. In such a scenario, there would be more capacity for decision-making on the server-side, but at the cost of increasing the complexity of monitoring, since their specific characteristics (e.g. life-cycle, data volume) would be managed accordingly. Another emergent concept that is being applied to monitoring microservices is Observability. It is referenced as a superset of monitoring that uses data analytics techniques on the collected

* Corresponding author.
E-mail addresses: brenogscosta@gmail.com (B. Costa), joao.bachiega.jr@gmail.com (J. Bachiega Jr.), leouesb@gmail.com (L.R. Carvalho), micheljunioferreira@gmail.com (M. Rosa), aleteia@unb.br (A. Araujo).

https://doi.org/10.1016/j.comnet.2022.109189
Received 5 April 2022; Received in revised form 16 June 2022; Accepted 13 July 2022
Available online 19 July 2022

Anexo III

Artigo publicado no Mobiquitous

Achieving Observability on Fog Computing with the Use of Open-Source Tools

Breno Costa[1,2(✉)], Abhik Banerjee[2], Prem Prakash Jayaraman[2], Leonardo R. Carvalho[1], João Bachiega Jr.[1], and Aleteia Araujo[1]

[1] Department of Computer Science, University of Brasília (UnB), Brasília, DF, Brazil
brenogscosta@gmail.com, aleteia@unb.br
[2] School of Science, Computing and Engineering Technologies, Swinburne University of Technology, Melbourne, VIC, Australia
{abanerjee,pjayaraman}@swin.edu.au

Abstract. Fog computing can provide computational resources and low-latency communication at the network edge. But with it comes uncertainties that must be managed in order to guarantee Service Level Agreements. Service observability can help the environment better deal with uncertainties, delivering relevant and up-to-date information in a timely manner to support decision making. Observability is considered a superset of monitoring since it uses not only performance metrics, but also other instrumentation domains such as logs and traces. However, as Fog Computing is typically characterised by resource-constrained nodes and network uncertainties, increasing observability in fog can be risky due to the additional load injected into a restricted environment. There is no work in the literature that evaluated fog observability. In this paper, we first outline the challenges of achieving observability in a Fog environment, based on which we present a formal definition of fog observability. Subsequently, a real-world Fog Computing testbed running a smart city use case is deployed, and an empirical evaluation of fog observability using open-source tools is presented. The results show that under certain conditions, it is viable to provide observability in a Fog Computing environment using open-source tools, although it is necessary to control the overhead modifying their default configuration according to the application characteristics.

Keywords: Observability · Fog Computing · Edge Computing · Metrics · Logs · Traces

1 Introduction

Fog computing is a computing model that brings computation and data storage closer to where it is needed, typically at the edge of the network. The main characteristics of Fog Computing are its distributed and decentralised nature,

A. Zaslavsky et al. (Eds.): MobiQuitous 2023, LNICST 594, pp. 319–340, 2024.
https://doi.org/10.1007/978-3-031-63992-0_21

Anexo IV

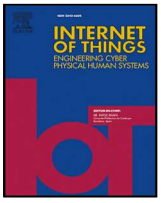
Artigo publicado na Internet of Things

Internet of Things 28 (2024) 101326

Contents lists available at ScienceDirect

Internet of Things

journal homepage: www.elsevier.com/locate/iot



5G enabled smart cities: A real-world evaluation and analysis of 5G using a pilot smart city application

Abhik Banerjee [a,*], Breno Costa [a,b], Abdur Rahim Mohammad Forkan [a], Yong-Bin Kang [a], Felip Marti [a], Chris McCarthy [a], Hadi Ghaderi [c], Dimitrios Georgakopoulos [a], Prem Prakash Jayaraman [a]

[a] School of Science, Computing and Engineering Technologies, Swinburne University of Technology, Melbourne, Victoria, Australia
[b] Department of Computer Science, University of Brasilia, Brasilia, Brazil
[c] School of Business, Law and Entrepreneurship, Swinburne University of Technology, Melbourne, Victoria, Australia

ARTICLE INFO

Keywords:
Smart cities
5G
Internet-of-things
Network performance

ABSTRACT

Ubiquitous sensing in smart cities is expected to be one of the key beneficiaries of the high bandwidth and low latency capabilities of 5G. However, current 5G deployments still have low population coverage, and are unlikely to reach global coverage of above 80% before 2028. This means that new smart city applications are likely to experience a combination of 4G and 5G, limiting their data-intensive capabilities. Thus, it is necessary to assess the ability of current 5G deployments to support emerging smart city applications, and how they perform in 5G environments. Existing performance evaluations focus either on the 5G core or use instantaneous speed tests, which do not effectively assess the suitability of 5G deployments for smart city applications. In this paper, we present a comprehensive evaluation and analysis of real-world 5G network performance observed through the outcomes of a pilot smart city application, an innovative mobile 5G Internet of Things (IoT) solution to automatically identify and report road assets requiring maintenance. The pilot smart city application was deployed on 11 waste collection service trucks over a 6 month period (June 2022–Dec 2022). We undertook both application-specific and application independent network performance evaluation to assess the ability of the 5G deployment to support uninterrupted smart city services. Our analysis shows that while 5G is capable of supporting mobile video streaming applications, there are significant variations in network performance, which may make it unsuitable for applications that require higher data intensive or near real-time responsiveness.

1. Introduction

The growing deployment of 5G networks worldwide is expected to benefit smart city applications deployment and services significantly. While Internet of Things (IoT) serves as a key backbone infrastructure for smart cities [1], the emergence of 5G technologies has enabled smart cities to integrate more effectively by allowing an enormous number of simultaneous connections and improving the network ubiquity [2], thereby enabling ubiquitous sensing that expand the scale and depth of IoT-based smart city applications and services [3]. The increasing focus on expanding 5G deployments is expected to enable new 5G smart city use cases in different verticals, such as transport, energy, manufacturing and healthcare [4]. The high bandwidth, low latency and high connection density capabilities of 5G make it suitable for deployment of novel IoT applications and services that require high data

* Corresponding author.
E-mail address: abanerjee@swin.edu.au (A. Banerjee).

https://doi.org/10.1016/j.iot.2024.101326
Received 2 October 2023; Received in revised form 27 March 2024; Accepted 9 August 2024
Available online 14 August 2024

Anexo V

Capítulo Springer Handbook of Data Engineering

Chapter 1

Observability in Fog Computing

Aleteia Araujo {1,2}, Breno Costa{1}, João Bachiega Jr.{1}, Leonardo R. Carvalho{1}, Rajkumar Buyya{2}

{1}Department of Computer Science - University of Brasília (UnB) - Brasília - DF - Brazil

email: aleteia@unb.br and {brenogscosta,joao.bachiega.jr,leouesb}@gmail.com

{2} Cloud Computing and Distributed Systems (CLOUDS) Labs - School of Computing and Information Systems - The University of Melbourne, Australia

email:{rbuyya@unimelb.edu.au}

Abstract - Fog Computing provides computational resources close to the end user, supporting low-latency and high-bandwidth communications. It supports IoT applications, enabling real-time data processing, analytics, and decision-making at the edge of the network. However, the high distribution of its constituent nodes and resource-restricted devices interconnected by heterogeneous and unreliable networks makes it challenging to execute service maintenance and troubleshooting, increasing the time to restore the application after failures and not guaranteeing the service level agreements. In such a scenario, increasing the observability of Fog applications and services may speed up troubleshooting and increase their availability. An observability system is a data-intensive service, and Fog Computing could have its nodes and channels saturated with an additional load. In this work, we detail the three pillars of observability (metrics, log, and traces), discuss the challenges, and clarify the approaches for increasing the observability of services in Fog environments. Furthermore, the system architecture that supports observability in Fog, related tools, and technologies are presented, providing a comprehensive discussion on this subject. An example of a solution shows how a real-world application can benefit from increased observability in this environment. Finally, there is a discussion about the future directions of Fog observability.

Keywords - Observability, Fog Computing, Edge Computing, Metrics, Logs, Traces

1.1 Introduction

Fog Computing is a model that seeks to bring processing and data storage closer to the location where they are required, usually at the network's edge. The primary features of Fog Computing include its distributed and decentralized nature, the emphasis on low latency and high bandwidth communication, and its capacity to support diverse devices and applications [1]. These features make it particularly well-suited for delivering high Quality of Service (QoS) in applications that