Towards a Observability Taxonomy for Microservices-Based Applications

Double-blind
Double-blind
Double-blind
Double-blind, Double-blind
Double-blind

Double-blind
Double-blind
Double-blind
Double-blind, Double-blind
Double-blind

Double-blind
Double-blind
Double-blind
Double-blind, Double-blind
Double-blind

ABSTRACT

Microservice-based applications have become widely adopted in enterprises due to their excellent scalability and timely update capabilities. However, while the fine-grained modularity and service orientation reduce the complexity of system development, they also increase the complexity of system operation and maintenance. This is because system faults have become more frequent and complex. With the increasing complexity of cloud-native applications, traditional monitoring solutions become inadequate, leading to higher risks of failures. Expanding monitoring to cloud-native applications is referred to as observability, which can be formally defined as the ability to understand and diagnose the internal behavior of a system by analyzing its external states, enables effective detection and resolution of issues. Although some studies in the literature address the concepts, tools, and challenges of observability, none of them presents an observability taxonomy. In this work, an observability taxonomy for microservices-based applications is proposed, supported by a systematic review of the literature. The authors of this work identified 26 studies published between 2019 and 2023. These selected studies were thoroughly analyzed and categorized by this taxonomy, providing researchers with a comprehensive overview about Observability.

CCS CONCEPTS

• **General and reference** → *Surveys and overviews*; *Reference works.*

KEYWORDS

Monitoring, Observability, Microservices, Taxonomy

ACM Reference Format:
Double-blind, Double-blind, and Double-blind. 2018. Towards a Observability Taxonomy for Microservices-Based Applications. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (SBES'24).* ACM, New York, NY, USA, 12 pages. https://doi.org/XXXXXXX.XXXXXXX

SBES'24, September 30 – October 04, 2024, Curitiba, PR

ACM ISBN 978-1-4503-XXXX-X/18/06
https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUÇÃO

Nos últimos anos, os sistemas de software modernos têm crescido em escala e complexidade, tornando cada vez mais desafiador incorporar novas funcionalidades. A emergência da arquitetura de microsserviços tem proporcionado uma solução fundamental para esse problema [50]. Tal abordagem ganhou ampla adoção para contornar bases de código monolíticas excessivamente complexas e dimensionar aplicativos de maneira transparente [12]. A partir de suas vantagens, cada vez mais empresas estão fazendo a transição para uma arquitetura de microsserviços, reconhecendo sua capacidade de aprimorar o desempenho e reduzir a mão de obra humana necessária para desenvolvimento, compensando assim os custos financeiros associados[47]. Infelizmente, Microsserviços não representam uma solução única e universal, e introduz novos desafios para os desenvolvedores [10].

À medida que os microsserviços e outros componentes arquiteturais inovadores tornam as aplicações nativas de nuvem mais distribuídas, complexas e imprevisíveis, há um aumento nos potenciais de falha e pontos que podem degradar o desempenho destas aplicações. O diagnóstico torna-se excepcionalmente desafiador quando uma aplicação nativa de nuvem consiste em centenas de microsserviços executando várias instâncias [12]. Soluções de monitoramento tradicionais falham em fornecer visibilidade completa e identificar proativamente desvios do comportamento esperado antes que interrompam os serviços. Medidas preventivas e detectivas dão origem a um novo conjunto de exigências de monitoramento desafiadoras que requerem um nível mais elevado de monitoramento, conhecido como observabilidade [21]. Observabilidade pode ser definida como a capacidade de inferir o estado de um sistema complexo com base em suas saídas [17]. Também se refere à propriedade de um sistema que pode ser observada e comparada com o estado esperado ao longo do ciclo de vida do desenvolvimento de software [18].

Alguns trabalhos analisaram soluções de observabilidade para microsserviços. Niedermaier *et al.* (2019) apresentaram um estudo baseado em entrevistas com a indústria para entender os desafios e melhores práticas no campo da observabilidade e monitoramento de sistemas distribuídos[34]. Li *et al.* (2022) conduziram um *survey* sobre microsserviços e observabilidade, enfatizando a complexidade dos sistemas de microsserviços industriais operando em infraestruturas de nuvem intricadas com instâncias de serviço criadas e destruídas dinamicamente[28]. Usman *et al.* (2022) realizaram um *survey* abrangente sobre ambientes de TI distribuídos e observabilidade de microsserviços, com o objetivo de explorar desafios, requisitos, melhores práticas e soluções atuais na observação de sistemas distribuídos[52]. Kosinska *et al.* (2023) apresentam um

mapeamento sistemático sobre observabilidade de aplicações nativas da nuvem, que consiste em um estudo sobre abordagens de engenharia de observabilidade, maturidade, eficiência, ferramentas, e também mostra direções de pesquisa futuras[21]. Até onde se sabe, nenhum trabalho realizou a categorização de soluções com relação aos propósitos, integração, como é realizada a coleta de dados, como utiliza a instrumentação, entre outros. Para abordar essas limitações, uma revisão abrangente de observabilidade em microsserviços é necessária.

Este trabalho analisa a literatura de observabilidade e propõe uma taxonomia de observabilidade para aplicações baseadas em microsserviços. O uso dessa taxonomia, na perspectiva de pesquisadores, profissionais ou desenvolvedores de ferramentas de observabilidade, é importante para fornecer uma estrutura comum e um vocabulário padronizado que facilita a comunicação, a organização e a análise de dados. Para os pesquisadores, essa taxonomia permite verificar se uma nova proposta de observabilidade é adequada para uso no contexto de microsserviços, bem como possibilitar a comparação de resultados entre diferentes estudos e a realização de meta-análises com maior precisão. Além disso, os desenvolvedores de soluções de observabilidade para microsserviços podem utilizá-la como um guia atualizado e abrangente para as características e recursos mais importantes de tal solução e, assim, auxiliar no desenvolvimento de ferramentas que atendam a padrões e necessidades do mercado. As principais contribuições deste trabalho são:

- Uma discussão atualizada e abrangente sobre as características e recursos da observabilidade para aplicações baseadas em microsserviços;
- Uma taxonomia de observabilidade para aplicações baseadas em microsserviços, descrevendo seus principais domínios e categorias;
- Uma categorização de soluções de observabilidade para aplicações baseadas em microsserviços encontradas na literatura usando a taxonomia proposta;

O restante deste artigo está organizado da seguinte forma: A Seção 2 contextualiza sobre aplicações baseadas em microsserviços e observabilidade. A Seção 3 descreve a metodologia da revisão utilizada. A Seção 4 descreve as categorias que compõem a taxonomia de observabilidade para aplicações baseadas em microsserviços. A Seção 5 apresenta os estudos selecionados na revisão da literatura, categorizados pela taxonomia proposta e as ameaças à validade. A Seção 6 apresenta os desafios do tema de observabilidade. Finalmente, a última seção mostra as conclusões e trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Esta Seção apresenta os conceitos, definições e características-chave relacionadas às áreas e subáreas que fundamentam esta pesquisa, as quais são de suma importância para a sua compreensão. Serão abordados os conceitos de Microsserviços e Observabilidade.

2.1 Microsserviços

O estilo arquitetural de microsserviços consiste no desenvolvimento de uma única aplicação como um conjunto de pequenos serviços, cada um operando de forma independente e se comunicando por meio de mecanismos leves, tipicamente uma API REST HTTP [27]. Esses serviços são construídos em torno das capacidades de negócios e podem ser implantados de forma independente, com mecanismos de implantação automatizados. Não há necessidade de gerenciamento centralizado, permitindo que sejam escritos em diferentes linguagens e usem diferentes tecnologias de armazenamento.

Este estilo, conforme descrito em [12], é composto por serviços autônomos com poucas responsabilidades, capazes de operar tanto de forma isolada quanto em conjunto uns com os outros. Pahl *et al* (2016) discutem a definição do tamanho apropriado para um microsserviço, indicando que um serviço é considerado muito grande quando possui muitas dependências ou executa muitas funções, justificando sua divisão em serviços menores e baixo acoplamento[36].

Alshuqayran *et al* (2016) descrevem a arquitetura de microsserviços como a separação de uma aplicação complexa em serviços distribuídos, cada um com funções específicas e utilizando comunicação leve [1]. No entanto, eles apontam que essa abordagem pode tornar as aplicações propensas a falhas devido à complexidade das dependências entre microsserviços.

Os microsserviços se tornaram a abordagem dominante para *Cloud Native Applications* (CNA) [24, 28]. Os sistemas industriais podem consistir em centenas a milhares de serviços, operando em infraestruturas de nuvem complexas, com instâncias de serviço criadas e destruídas dinamicamente [42, 55]. Diagnosticar problemas, como falhas de solicitações e latência alta, em arquiteturas de microsserviços é extremamente desafiador devido ao grande número de serviços envolvidos [40]. Em termos de desenvolvimento e implantação, os microsserviços são mais exigentes do que as aplicações tradicionais, compostas por vários componentes, como máquinas físicas, máquinas virtuais e contêineres [13, 16, 43]. Verificar a saúde, o desempenho e o comportamento de cada componente é essencial, destacando a importância do monitoramento como uma ferramenta crítica para os microsserviços [3, 15, 35].

2.2 Observabilidade

2.2.1 ***Introdução.*** Em um ambiente de produção, enfrentar os desafios de operar microsserviços requer a capacidade de mantê-los funcionando sem problemas e identificar prontamente quaisquer falhas ou degradação na qualidade do serviço. Tanto os provedores de serviços de nuvem quanto os usuários precisam monitorar constantemente o uso, o desempenho e a disponibilidade dos recursos [26, 41]. Em situações de falhas ou interrupções, eles dependem de dados de monitoramento para diagnosticar problemas e, em última análise, fazer reparos. Assim, uma nova estratégia conhecida como Observabilidade surgiu [11, 33].

No guia da Cloud Native Computing Foundation (CNCF) [9], um dos passos é a Observabilidade e Análises [21]. A Observabilidade é um conceito originado da Teoria do Controle [17], que define um sistema como observável se seu estado atual puder ser determinado em tempo finito usando apenas as saídas. Medir o desempenho geral de um microsserviço é crucial para fazer cumprir as métricas de Qualidade de Serviço (QoS) da aplicação. O sistema deve expor adequadamente seu estado por meio de técnicas de instrumentação para alcançar os parâmetros acordados no Acordo de Nível de Serviço (SLA). A Observabilidade é frequentemente composta por métricas, *logs* e *traces*, e é uma característica essencial em ambientes nativos de nuvem [22], pois seu principal objetivo é tornar a aplicação mais compreensível em vários níveis. Niedermaier *et al.* (2019)

afirmam que, em ambientes de nuvem, a observabilidade indica até que ponto a infraestrutura, as aplicações e suas interações podem ser monitoradas[34].

As motivações para equipar as aplicações baseadas em microsserviços com observabilidade derivam da fase de execução dessas aplicações [21]. Elas estão relacionadas ao provisionamento de recursos com foco no rastreamento em tempo real, medição do *overhead* do sistema e agendamento, incluindo aspectos como escalabilidade. Do ponto de vista da gestão, as motivações mais relevantes incluem detecção de causas-raiz (anomalias, previsão de falhas, gerenciamento de falhas, análise de dependências), gestão de SLA (carga, disponibilidade, QoS e violações) [38], bem como gestão de ponta a ponta (desempenho, tempo de resposta, análise de latência e rastreamento de ponta a ponta), entre outros [44].

2.2.2 ***Diferença entre Monitoramento e Observabilidade.*** Embora o monitoramento tenha desempenhado um papel central na TI ao longo das últimas décadas, sua eficácia foi questionada devido a uma série de fatores, incluindo o surgimento de metodologias de desenvolvimento ágil, implantações nativas da nuvem e novas práticas de DevOps. Esses fatores mudaram a maneira como todo o ecossistema de TI - infraestruturas, sistemas e aplicações - precisa ser monitorado para garantir uma resposta rápida a incidentes. Simplesmente aplicar ferramentas de monitoramento tradicionais não é mais eficaz por várias razões, como a presença de componentes em contêineres, ambientes mais dinâmicos e ciclos de implantação de aplicativos acelerados. A observabilidade complementa o monitoramento ajudando a identificar onde e por que uma falha ocorreu e o que desencadeou o problema. Ela não substitui o monitoramento ou elimina a necessidade dele; eles trabalham juntos.

Enquanto o monitoramento envolve o rastreamento da saúde de um sistema usando um conjunto predefinido de métricas e *logs*, focando na detecção de uma coleção específica de falhas, em um sistema distribuído, muitas mudanças são dinâmicas, também ocorrendo em paralelo, e podem ocorrer regularmente. Isso pode resultar em problemas que não se enquadram nas situações anteriormente definidas pelo monitoramento, escapando da detecção.

2.2.3 ***Definições de Observabilidade.*** Ao longo dos anos, vários autores ofereceram definições adicionais do conceito de observabilidade, complementando as ideias apresentadas anteriormente. De acordo com Usman *et al.* (2022), a observabilidade é um conjunto crescente de práticas combinadas com ferramentas que vão além do mero monitoramento, fornecendo *insights* sobre o estado interno de um sistema por meio da análise de suas saídas externas, predominantemente conhecidas como dados de telemetria, como métricas, *logs* e *traces*[52]. Esta abordagem oferece uma visão geral de alto nível da saúde do sistema, bem como *insights* detalhados sobre os modos de falha subjacentes. Além disso, um sistema com alto nível de observabilidade fornece um contexto abrangente para seu funcionamento interno, permitindo a identificação de falhas mais profundas no sistema.

Para Kratzke e Stage (2022), a observabilidade representa a base para o desenvolvimento de software orientado por dados[23]. Enquanto isso, para Marie *et al.* (2021) [30], a observabilidade é um conceito emergente que pode ser considerado um conjunto mais amplo do que o monitoramento, estendendo seus limites e aplicando técnicas de análise de dados. De certa forma, a observabilidade pode ser percebida como uma extensão necessária do monitoramento para a operação de aplicações nativas de nuvem.

De acordo com Picoreti *et al.* (2018), a observabilidade refere-se à capacidade de visualizar e entender o comportamento de um sistema reunindo e processando dados do sistema e apresentando-os de maneira adequada para diferentes tipos de usuários, como aplicações, desenvolvedores, administradores de sistema e sistemas de orquestração[37]. Esse processo não apenas permite a análise do comportamento presente, mas também a inferência do comportamento futuro do sistema.

2.2.4 ***Benefícios da Observabilidade.*** As aplicações baseadas em microsserviços se destacam quando seu ambiente de execução é equipado com recursos de observabilidade. Algumas vantagens do ecossistema de observabilidade incluem:

- **Aumento da Visibilidade**: Permite uma compreensão mais profunda do que, onde, como e quando ocorrem as atividades no sistema. Isso inclui identificar quais serviços estão ativos, avaliar seu desempenho, localizar componentes da aplicação e reconhecer condições inesperadas;
- **Detecção Antecipada de Problemas**: Facilita a identificação precoce de problemas, permitindo sua resolução antes que impactem a funcionalidade do sistema. Isso abrange não apenas questões de desempenho da aplicação, mas também aspectos relacionados à arquitetura e ao design do sistema;
- **Melhoria dos Processos DevOps e do Fluxo de Desenvolvimento**: Fornece *insights* valiosos sobre o comportamento do sistema, impulsionando o desenvolvimento de software. As decisões de design podem ser informadas por observações em tempo real do sistema em execução, potencialmente influenciando redesenhos ou ajustes na arquitetura;
- **Escalabilidade Aprimorada**: Permite que o sistema seja projetado com escalabilidade em mente, garantindo que subsistemas, especialmente aqueles relacionados à observabilidade, possam lidar com demandas de dados aumentadas à medida que o sistema cresce;
- **Possibilita a Automação**: Os dados de observabilidade podem ser usados para automatizar processos como escalonamento e *failover* automático, contribuindo para um controle mais autônomo da aplicação como um todo;
- **Informações Valiosas para Análise de Dados**: A coleta de dados sobre o sistema possibilita análises avançadas, incluindo estatísticas e aprendizado de máquina, fornecendo *insights* adicionais sobre o comportamento da aplicação e seu ambiente de execução; e
- **Melhoria da Experiência do Usuário**: Uma aplicação com observabilidade eficaz tende a oferecer uma melhor experiência ao usuário, aumentando a satisfação do cliente e contribuindo para os objetivos comerciais da aplicação.

2.2.5 ***Tipos de Dados de Observabilidade.*** Como mencionado anteriormente, existem três tipos principais de dados frequentemente referidos como os três pilares da observabilidade: *Traces*, Métricas e *Logs*.

Os ***Traces*** fornecem uma visão abrangente do fluxo de uma solicitação por meio de uma aplicação [38]. Os traces são essenciais para entender o caminho completo que uma solicitação percorre

pela aplicação. Um rastreamento distribuído consiste em uma série de eventos gerados em diferentes pontos em um sistema, conectados por um identificador único. Este identificador é propagado por todos os componentes envolvidos em qualquer operação necessária para completar a solicitação, permitindo que cada operação associe dados de evento com a solicitação original. Cada rastreamento representa uma solicitação única por meio de um sistema, que pode ser síncrona ou assíncrona. Solicitações síncronas ocorrem sequencialmente, com cada unidade de trabalho concluída antes de prosseguir. Já as solicitações assíncronas podem iniciar uma série de operações que podem ocorrer simultânea e independentemente. Cada operação registrada em um rastreamento é representada por um Span, uma única unidade de trabalho realizada no sistema.

As **Métricas**, assim como os rastreamentos distribuídos, fornecem informações sobre o estado de um sistema em execução para desenvolvedores e operadores. Os dados coletados por meio de métricas podem ser agregados ao longo do tempo para identificar tendências e padrões em aplicações, representados graficamente por meio de várias ferramentas e visualizações. O termo métricas tem uma ampla gama de aplicações, pois pode capturar desde métricas de sistema de baixo nível, como ciclos de CPU, até detalhes de nível mais alto, como o número de camisetas azuis vendidas atualmente. Além disso, as métricas são essenciais para monitorar a saúde de uma aplicação e decidir quando alertar sobre um possível problema. As métricas são frequentemente úteis por si só, mas quando correlacionadas com outros tipos de dados, como *logs* ou *traces*, podem fornecer informações ainda mais úteis sobre o estado do sistema.

Os ***Logs*** fornecem um registro das atividades que ocorreram e podem ser usados para debugar sistemas e investigar problemas. No contexto de observabilidade, logs frequentemente complementam *traces* e métricas, fornecendo uma visão mais detalhada de eventos específicos que ocorreram em um sistema. *Logs* são especialmente úteis para investigar problemas em tempo real, pois fornecem uma visão detalhada do que estava acontecendo em um sistema em um momento específico. Isso pode ser particularmente útil quando ocorrem falhas ou problemas de desempenho em um sistema, pois permite que os desenvolvedores vejam exatamente o que estava acontecendo em um sistema no momento de um problema.

3 METODOLOGIA DE PESQUISA

O trabalho de [45] inspira a metodologia de revisão sistemática utilizada neste trabalho. Como ilustrado na Figura 1, a pesquisa segue os seguintes passos: definição do escopo, busca, filtragem, classificação e extração de informações. No escopo do trabalho, foi elaborada uma questão orientadora para as buscas a fim de obter um resultado mais preciso sobre o tema a ser estudado. Ao final, apresentamos os resultados obtidos.

Figura 1: Fluxograma da metodologia aplicada na pesquisa.

Definição do Escopo: A primeira etapa é dedicada à Definição do Escopo, que contém a pergunta que guia a pesquisa. A pergunta define um limite para o escopo do trabalho, ou seja, um limite da área de estudo a ser mapeada. Posteriormente, as strings de busca são utilizadas para escolher os artigos. Vale ressaltar que, antes da elaboração da questão, houve um consenso para analisar tópicos úteis para orientar a pesquisa sobre "observabilidade em microsserviços". Portanto, este trabalho propõe uma **Questão de Pesquisa** a fim de delimitar o escopo do trabalho: *Quais são os diferentes domínios e categorias de observabilidade em microsserviços?* Esta questão visa identificar o real propósito da pesquisa em como a observabilidade pode ajudar as aplicações baseadas em microsserviços.

Busca: Dentro desta etapa, uma *String* de busca foi aplicado em quatro Mecanismos de Busca: *IEEE, ACM, Scopus* e *Web of Science*. Logo após, os trabalhos retornados foram coletados. Utilizamos palavras-chave, em inglês, mais relacionadas ao tema "Observabilidade em Microsserviços" para gerar a *string* de busca. Após analisar os termos, identificou-se que seria necessário utilizar a *string* "Observability". O termo "Monitoring" não foi utilizado como sinônimo porque os artigos retornados referem-se a métricas, não aos três tipos de dados, e também não abrangem a ideia geral de observabilidade. Além desses, a *string* "Microservices" precisava ser inserida, e sinônimos e formas no plural foram adicionados, como "Microservice", "Container" e "Containers". Após a busca e obtenção dos trabalhos, foi realizado um processo de filtragem, conforme mostrado na Figura 2. Também vale mencionar que a *string* de busca foi aplicada usando a opção de busca avançada fornecida pelos mecanismos de busca. Além disso, a busca foi ajustada para encontrar o trabalho pelo texto completo.

Filtragem: A estratégia adotada também considerou critérios de exclusão direta conforme a Tabela 1. A maneira adotada para incluir os trabalhos foi verificar se os trabalhos estavam fora dos critérios de exclusão usados. Se o trabalho não atender aos motivos explícitos de exclusão, o trabalho é automaticamente incluído e avança para as próximas etapas de filtragem. A Figura 2 apresenta o fluxograma usado para aplicar os critérios de exclusão e ilustra todas as fases de filtragem realizadas. Os trabalhos encontrados são de 2019 a 2023. No total, as buscas retornaram 1723 trabalhos pesquisados. O fluxograma mostra o número de trabalhos presentes e os excluídos com seus respectivos motivos de exclusão em cada fase de filtragem.

Tabela 1: Critérios de Exclusão Direta

Critérios de Exclusão Direta
O trabalho não está em inglês.
Trabalho com menos de seis páginas.
Artigos não científicos.
Survey/Mapeamento/Revisão.
Trabalhos fora do escopo pelo título.

Na primeira parte da filtragem, a remoção de trabalhos foi baseada em alguns critérios de exclusão definidos, como "Não ser um Artigo Científico" e "Não estar escrito em inglês", removendo assim 458 trabalhos. A segunda parte foi observar trabalhos duplicados, além de excluir trabalhos cujo objeto de estudo fosse um levantamento, mapeamento, revisão ou se o título não fosse informativo. Esta etapa excluiu um total de 1208 trabalhos que não estavam dentro do escopo do trabalho. A terceira parte analisou os 57 trabalhos restantes aplicando o critério de menos de seis páginas e fora do escopo de acordo com o texto completo, deixando 26 artigos restantes para classificação. Esta última etapa classificou os trabalhos

de acordo com a proposta de Taxonomia de Observabilidade para Aplicações Baseadas em Microsserviços (Seção 4).

Classificação: A quarta etapa tratou da Classificação, onde a categorização foi feita para extrair informações mais precisas e identificar lacunas na pesquisa. Esta etapa foi dividida em duas fases. A primeira fase consistiu em definir as categorias/subcategorias. Na primeira fase, foram definidos cinco tipos de categorias: *Ambiente, Domínio, Propósito, Integração e Coleta*. A categoria *Domínio* é dividida em 3 subcategorias: *Traces, Métricas e Logs*. A subcategoria *Traces* é dividida em 2 subcategorias: *Dado Extraído e Amostragem*. A subcategoria *Métricas* é dividida em 2 subcategorias: *Origem e Frequência*. A categoria *Coleta* possui uma subcategoria chamada de *Instrumentação*. Na segunda fase, foram criadas classificações para cada categoria/subcategoria. Todas as classificações utilizadas podem ser vistas na Figura 3 e foram declaradas com base nos trabalhos mapeados e na literatura.

Extração de Informações: A quinta etapa realizou a Extração de Informações. Neste estudo, a extração de informações sobre os trabalhos foi feita pela leitura direta dos trabalhos a serem mapeados de acordo com as classificações. O processo de classificação dos artigos selecionados foi feito por todos os autores deste artigo. O primeiro autor, aluno de doutorado, foi responsável pela leitura completa de cada um dos trabalhos, enquanto os demais, orientadores, acompanharam os achados e orientaram na construção da classificação, tanto nas categorias quanto no posicionamento dos artigos selecionados nessas categorias.

4 TAXONOMIA DE OBSERVABILIDADE

Esta Seção apresenta uma taxonomia de observabilidade para aplicações baseadas em microsserviços, criada a partir da revisão sistemática da literatura. A taxonomia consolida domínios e categorias relevantes do estado da arte de observabilidade. Eles são resumidos na Figura 3 e uma descrição detalhada deles é apresentada nas subseções seguintes.

4.1 Propósitos

Esta categoria define os objetivos de uma solução de observabilidade. A seguir, é apresentado os encontrados na literatura. **Uso de Recursos**: As soluções de observabilidade podem rastrear métricas de uso de recursos, como CPU, memória, disco e utilização de rede para garantir uma alocação eficiente de recursos e identificar possíveis gargalos de desempenho. **Escalonamento Automático**: Ao analisar métricas e tendências de desempenho, as soluções podem automatizar processos de escalonamento automático para ajustar dinamicamente a provisionamento de recursos com base na demanda, otimizando o desempenho do sistema e a eficiência de custos. **Agendamento**: As soluções podem ajudar a otimizar o agendamento de recursos e a distribuição de carga de trabalho. **Detecção de Anomalias**: As soluções podem empregar algoritmos de detecção de anomalias para identificar padrões ou desvios incomuns do comportamento normal. **Análise de Causa Raiz**: As soluções podem fornecer visibilidade detalhada nos componentes e dependências do sistema, facilitando a análise de causa raiz de problemas, rastreando eventos e identificando os fatores subjacentes que contribuem para a degradação de desempenho ou falhas. **Detecção de Gargalos**: Ao identificar gargalos de desempenho e restrições de recursos,

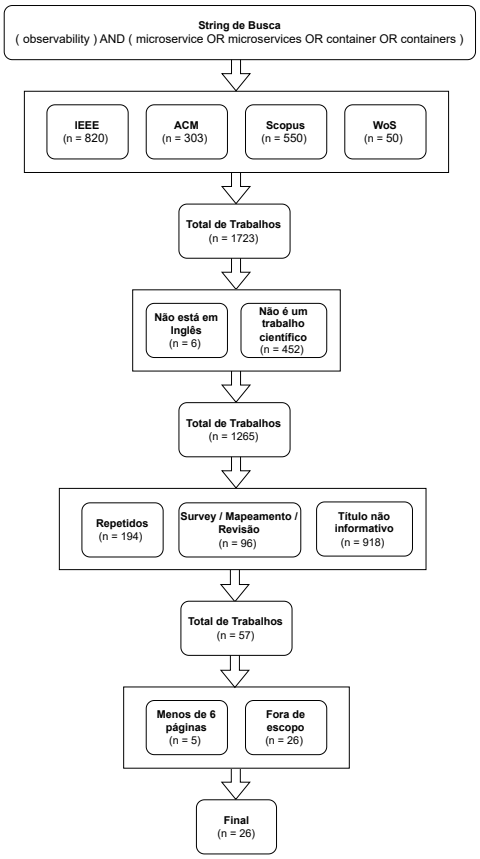

Figura 2: Fluxograma demonstrando as etapas do processo de filtragem.

as soluções permitem a otimização da arquitetura e configuração do sistema para melhorar a eficiência geral e a escalabilidade. **Desempenho**: As soluções podem monitorar e analisam métricas de desempenho para garantir que os sistemas atendam a objetivos de nível de serviço (SLOs) e metas de desempenho predefinidas. Isso inclui métricas como tempo de resposta, taxa de transferência e taxas de erro. **Custo**: As soluções podem rastrear métricas de uso de recursos para fornecer uma visão sobre alocação de custos e faturamento, permitindo que organizações otimizem a utilização de recursos e gerenciem os gastos em nuvem de forma eficaz. **Insights**: Ao agregar e analisar dados de várias fontes, as plataformas de observabilidade geram *insights* sobre o comportamento do sistema, tendências de desempenho e eficiência operacional, capacitando as organizações a tomar decisões e otimizações informadas. **Engenharia do Caos**: As soluções podem suportar práticas de engenharia do caos, fornecendo visibilidade sobre o comportamento do sistema em cenários de falha controlada, permitindo que as organizações avaliem a resiliência, identifiquem vulnerabilidades e melhorem a confiabilidade do sistema. **Segurança**: As soluções podem incluir capacidades de monitoramento e análise de segurança para detectar e mitigar ameaças de segurança, rastrear eventos de acesso e autorização e garantir conformidade com políticas e regulamentos de segurança.

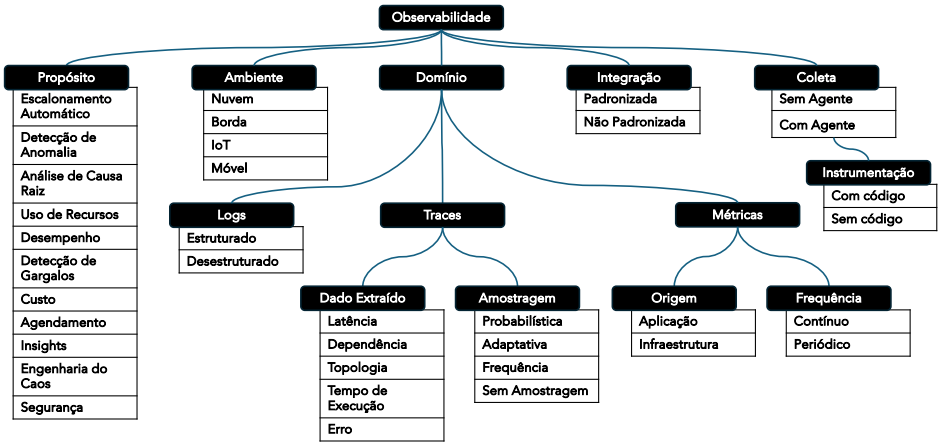

Figura 3: Taxonomia de Observabilidade.

4.2 Ambiente

A observabilidade pode ser aplicada em vários ambientes nos quais os microsserviços podem ser implantados, e requer abordagens personalizadas e ferramentas especializadas para abordar as características e desafios únicos de cada ambiente, tais como: Nuvem, Borda, IoT e Móvel. A observabilidade na **Nuvem** envolve o monitoramento de vários serviços nativos da nuvem, como orquestradores de contêineres (por exemplo, Kubernetes), plataformas sem servidor e bancos de dados em nuvem. A observabilidade em ambientes de **Borda** envolve o monitoramento de microsserviços distribuídos implantados em dispositivos de borda ou nós de computação de borda. A observabilidade em ambientes de *Internet of Things* **IoT** envolve o monitoramento de telemetria de dispositivos, dados de sensores e protocolos de comunicação em implantações distribuídas de IoT. A observabilidade em ambientes **Móveis** inclui o monitoramento do desempenho de aplicativos móveis, interações do usuário, chamadas de API de back-end e redes móveis.

4.3 Domínios

Como discutido na Seção 2.2.5, a observabilidade de sistemas distribuídos envolve três principais domínios de instrumentação: *logs*, *traces* e métricas.

4.4 *Logs*

Os formatos de *logs* podem ser amplamente categorizados em formatos estruturados e não estruturados com base em como os dados são organizados nos arquivos de *log*. *Logs* **estruturados** organizam os dados de *log* em um esquema ou formato predefinido, geralmente usando pares de chave-valor ou um formato de dados estruturado como JSON, XML ou CSV. *Logs* **Desestruturados** não seguem um esquema predefinido e, em vez disso, armazenam os dados de *log* em um formato de texto livre e legível por humanos.

4.5 *Traces*

4.5.1 ***Dado Extraído.*** Através dos *traces*, valiosos *insights* sobre a aplicação podem ser obtidos. O rastreamento distribuído permite a medição e análise da **latência**, que é o tempo necessário para uma solicitação percorrer os vários componentes de um sistema distribuído. Também fornece visibilidade das **dependências** entre diferentes serviços e componentes dentro do sistema, tornando possível entender o fluxo de dados e controle entre os serviços. Ao agregar dados de *trace* e visualizar as relações entre serviços e componentes, a **topologia** mostra a estrutura e arquitetura geral do sistema. O rastreamento distribuído facilita a detecção e diagnóstico de **erros** e falhas dentro de um sistema distribuído. Por fim, os *traces* fornecem *insights* em tempo real sobre o **tempo de execução** de um sistema distribuído.

4.5.2 ***Amostragem.*** A amostragem de *traces* é uma técnica usada em sistemas distribuídos para reduzir o *overhead* associado à coleta e armazenamento de dados de *traces* em escala. Em vez de registrar cada evento ou transação que ocorre no sistema, a amostragem de *traces* captura apenas uma amostra representativa de eventos para análise. Existem várias estratégias de amostragem que podem ser empregadas na amostragem de *traces*. **Frequência**: é definido uma frequência fixa para amostrar os *traces* (por exemplo, capturar um *trace* a cada 1000 eventos). **Probabilística**: cada evento tem uma certa probabilidade de ser amostrado (por exemplo, se for definido uma probabilidade de 0.1, isso significa que, em média, cerca de 10% dos eventos serão capturados). *Adaptativa*: Este método ajusta dinamicamente a taxa de amostragem com base em certas condições ou critérios (por exemplo, pode aumentar a taxa de amostragem quando certas métricas atingem um limiar específico, como o aumento do tráfego na aplicação).

4.6 Métricas

4.6.1 ***Origem.*** Os níveis de métricas referem-se às diferentes camadas ou categorias de métricas que são usadas para monitorar e avaliar vários aspectos do desempenho, comportamento e saúde de um sistema. Esses níveis fornecem diferentes perspectivas e granularidade de dados, possibilitando uma observabilidade abrangente e análise do sistema. As métricas de **Aplicação** concentram-se em monitorar o comportamento e o desempenho do aplicativo de software em si. Incluem métricas como tempo de resposta, taxa de transferência, taxa de erro e latência. As métricas de **Infraestrutura**

referem-se aos componentes de hardware e software subjacentes que suportam o aplicativo. Incluem métricas como utilização de CPU, uso de memória, E/S de disco e largura de banda de rede.

4.6.2 ***Frequência.*** A tomada de decisão eficaz depende de ter informações atualizadas sobre recursos e serviços. Uma frequência mais alta de atualizações, como receber notificações imediatas para cada mudança nos valores das métricas, reduz o risco de confiar em dados desatualizados. No entanto, uma alta frequência de atualização pode levar a sobrecarga computacional e congestionamento de rede, especialmente em cenários com um grande número de dispositivos monitorados. Portanto, é essencial equilibrar a frequência de monitoramento com a capacidade computacional e as restrições de rede. Existem duas abordagens para a frequência de monitoramento: Contínua e Periódica. O monitoramento **Contínuo** envolve o fluxo ininterrupto de dados de monitoramento para o servidor uma vez iniciado. O monitoramento **Periódico** permite configurar um período recorrente, com dados sendo enviados em intervalos regulares.

4.7 Integração

Esta categoria avalia se os estudos de observabilidade estão em conformidade com um formato de dados padronizado. Algumas soluções utilizam arquivos JSON como formatos de dados padronizados fundamentais. Nos últimos anos, iniciativas como o OpenTelemetry (OTel) [4], amplamente adotadas na indústria e consideradas o padrão de ouro para observabilidade, foram introduzidas, construindo sobre padrões anteriores como OpenCensus e OpenTracing. A adoção desses padrões facilita a troca de dados e a integração de vários componentes de monitoramento, mesmo que eles provenham de diferentes fornecedores ou operem em diferentes camadas da aplicação.

4.8 Coleta

Os dados gerados pelos microsserviços são continuamente coletados por um coletor centralizado. Duas técnicas principais são tipicamente empregadas para essa coleta: **Com Agente**, os dados são coletados de uma instância de serviço e depois encaminhados para o coletor; e **Sem Agente**, a instância de serviço envia seus dados gerados diretamente para o coletor. Esta abordagem é mais simples de implementar e não requer suporte de infraestrutura. No entanto, exige que os desenvolvedores integrem o mecanismo de coleta em sua base de código.

4.8.1 ***Instrumentação.*** Se caso a coleta seja realizada por um agente, para tornar um sistema observável, ele deve ser instrumentado. Os desenvolvedores podem instrumentar o código de duas maneiras principais: **Com código**, de maneira manual, permitem aos desenvolvedores obterem *insights* mais profundos e telemetria diretamente de suas aplicações; e **Sem código**, de maneira automática, são particularmente úteis para configurações iniciais ou situações em que modificar a aplicação não é viável. Essas soluções oferecem telemetria abrangente extraindo dados de bibliotecas utilizadas dentro da aplicação ou do ambiente de tempo de execução da aplicação.

5 DISCUSSÕES

Nesta Seção, é apresentado uma categorização das estudos selecionados de observabilidade para aplicações baseadas em microsserviços com base nas categorias/subcategorias definidas na taxonomia.

5.1 Classificação

A Tabela 2 apresenta os 26 estudos selecionados com a classificação deles de acordo com a taxonomia de observabilidade. Essa tabela tem as seguintes legendas: (i) Domínio: Métricas (M), *Logs* (L), Traces (T); (ii) Integração: se for utilizado alguma forma de integração, é marcado com o valor S; se não for utilizado, é marcado com o valor N; (iii) Coleta: se for utilizado um agente, é marcado com S; se não for utilizado, é marcado com N; (iv) Amostragem: Adaptativa (A), Sem amostragem (N), Frequência (F), Probabilística (P); (v) *Logs*: Estruturado (S), Desestruturado (N); (vi) Frequência: Periódico (P), Contínuo (C). No geral, caso não seja identificado um valor, fica marcado com um traço (-). A seguir, são apresentados os estudos selecionados na revisão sistemática em relação à taxonomia proposta.

Com relação aos propósitos apresentados na taxonomia, o escalonamento automático é verificado em cinco trabalhos ([51], [57], [31], [2], [26]). Tzanettis *et al.* (2022) fornecem uma abordagem de observabilidade e uma implementação de IoT para lidar com aspectos de fusão de dados em plataformas de orquestração de computação em borda e em nuvem [51]. Para auxiliar no escalonamento automático, são utilizados três tipos de domínios, sendo coletados por meio de agentes e realizada a instrumentação automática. As métricas coletadas são de infraestrutura (CPU / Memória) e aplicação (Tempo de resposta e Taxa de Erro) de maneira contínua. Os *logs* são estruturados no formato JSON para facilitar o armazenamento e a busca de dados para o escalonamento. Yu *et al.* (2020) apresentam o Microscaler para identificar automaticamente os serviços que necessitam de escalonamento e escaloná-los para atender ao SLA com um custo ótimo para sistemas de microsserviços. O Microscaler coleta apenas métricas, com foco em infraestrutura (CPU / Memória) e aplicação (Latência) de maneira periódica (a cada 10 segundos) [57]. Marie-Magdelaine *et al.* (2020) apresentam um *framework* de escalonamento automático proativo usando um modelo de previsão baseado em aprendizado para ajustar dinamicamente o conjunto de recursos, horizontalmente e verticalmente [31]. O trabalho utiliza apenas métricas, com foco em infraestrutura (CPU) e aplicação (Tempo de resposta) para verificar a necessidade do escalonamento. Baarzi *et al.* (2021) apresentam o SHOWAR, um *framework* que configura os recursos determinando o número de réplicas e a quantidade de CPU e memória para cada microsserviço. Para o escalonamento vertical, o SHOWAR utiliza a variação empírica no uso histórico de recursos para encontrar o tamanho ideal e mitigar o desperdício de recursos. Para o escalonamento horizontal, o SHOWAR utiliza teoria de controle juntamente com métricas de desempenho em nível de kernel, coletadas pelo agente periodicamente. Além disso, uma vez encontrado o tamanho ideal para cada microsserviço, o SHOWAR preenche a lacuna entre alocação ótima de recursos e agendamento, gerando regras de afinidade, a partir das dependências dos *traces*, para o agendador melhorar ainda mais o desempenho [2]. Levin *et al.* (2020) apresentam o ViperProbe, um *framework* de coleta de métricas para microsserviços baseado em

Tabela 2: Estudos Selecionados

#	Ref	Ambiente	Domínio	Propósito	Integração	Coleta	Agente Instrumentação	Traces Amostragem	Traces Dado Extraído	Métricas Origem	Métricas Frequência	Logs
1	[20]	Nuvem Móvel	M-L	Detecção de Anomalia	-	S	-	-	-	Infraestrutura Aplicação	-	-
2	[6]	Nuvem IoT	T	Detecção de Gargalos	S	S	-	-	Latência Dependências	-	-	-
3	[32]	Nuvem	M-L-T	Segurança	S	S	Com código	A	-	-	-	-
4	[22]	Nuvem	M	Insights	-	-	-	-	-	Infraestrutura	-	-
5	[23]	Nuvem	M-L-T	Insights	S	S	Com código	N	Tempo de execução	Infraestrutura	-	S
6	[8]	Nuvem	M-L-T	Custo	S	S	Sem código	-	Tempo de execução	Infraestrutura	P	-
7	[51]	Nuvem IoT	M-L-T	Escalonamento automático	S	S	-	-	Latência Erro	Infraestrutura Aplicação	C	S
8	[7]	Nuvem	M-T	Uso de Recursos	-	-	-	-	Topologia	Infraestrutura	-	-
9	[53]	Borda	M-L-T	Insights	S	S	-	-	Tempo de execução	Infraestrutura	-	N
10	[19]	Nuvem	M	Detecção de Anomalia	-	-	-	-	-	Infraestrutura Aplicação	-	-
11	[54]	Nuvem	M-L-T	Análise de causa raiz	-	S	-	-	Dependência	Infraestrutura	-	-
12	[48]	Nuvem	M-T	Desempenho Custo	-	-	-	-	Dependência	Infraestrutura Aplicação	P	-
13	[57]	Nuvem	M	Escalonamento automático	-	-	-	-	-	Infraestrutura Aplicação	P	-
14	[25]	Borda	M-L-T	Insights	S	S	Sem código	-	Latência	Infraestrutura Aplicação	-	-
15	[56]	Nuvem	M-L-T	Análise de causa raiz	S	S	Sem código	-	Latência	Infraestrutura Aplicação	C	N
16	[46]	Nuvem	M	Engenharia do caos	-	-	-	-	-	Infraestrutura Aplicação	-	-
17	[33]	IoT	M-L-T	Engenharia do caos	-	-	-	-	Topologia	Infraestrutura Aplicação	-	N
18	[31]	Nuvem	M	Escalonamento automático	-	-	-	-	-	Infraestrutura Aplicação	-	-
19	[49]	Nuvem Móvel	M-L-T	Detecção de Anomalia	-	S	-	-	Latência	Infraestrutura	-	S
20	[14]	Nuvem	M-T	Análise de causa raiz	-	S	-	F	Dependência	Infraestrutura	P	-
21	[2]	Nuvem	M-T	Escalonamento automático	-	S	-	-	Dependência	Infraestrutura Aplicação	P	-
22	[41]	Nuvem	M-L-T	Insights	S	S	-	N	Latência	Infraestrutura	P	S
23	[29]	Nuvem	M-T	Agendamento	-	-	-	P	Dependência	Aplicação	-	-
24	[39]	Nuvem	M-L-T	Detecção de Anomalia	-	S	-	-	Latência	Infraestrutura Aplicação	-	-
25	[5]	Nuvem	T	Uso de Recursos	S	S	-	-	Latência Dependências	-	-	-
26	[26]	Nuvem	M	Escalonamento automático	-	S	-	-	-	Aplicação	-	-

eBPF. Ele examina o perfil de desempenho antes da implantação em produção para reduzir efetivamente o conjunto de métricas de aplicação (Latência) coletadas por meio de agentes, melhorando assim a eficiência e a eficácia dessas métricas para escalonamento automático [26].

Com relação aos propósitos apresentados na taxonomia, os *insights* são verificados em cinco trabalhos ([22], [23], [25], [53], [41]). Kosińska *et al.* (2020) apresentam requisitos para o gerenciamento autonômico de CNA e a construção de um *framework* denominado AMoCNA, de acordo com os conceitos de Arquitetura Dirigida por Modelos, cujo principal objetivo é reduzir a complexidade do gerenciamento dessas aplicações [22]. O trabalho utiliza apenas métricas como domínio, focando em infraestrutura (CPU / Memória), para apresentar uma visão geral da aplicação. Kratzke *et al.* (2022) apresentam uma solução para a fusão dos três tipos de dados para observabilidade, de forma mais integrada e com uma abordagem de instrumentação direta, através de uma biblioteca de *logs* unificados e estruturados do tipo JSON [23]. O trabalho utiliza agentes para coleta dos dados com instrumentação manual para obter mais detalhes nos registros, com os *traces* mostrando o tempo de execução dos microsserviços. As métricas coletadas são de infraestrutura (CPU / Memória / Disco). Kumar *et al.* (2023) apresentam o NEOS, que simplifica o processo de coleta, análise e visualização das métricas, *logs* e *traces*. NEOS utiliza ferramentas de código aberto populares como OTel, Grafana, Prometheus, Jaeger e Loki [25]. Usman *et al.* (2023) apresentam um *framework* para integração de *workflow* de observabilidade em várias etapas, denominado DESK, cujo objetivo é melhorar os *workflows* para medição, coleta, fusão, armazenamento, visualização e notificação em observabilidade para os três domínios [53]. Esse trabalho também utiliza OTel, por meio de agentes e instrumentação automática para a coleta dos dados. Scrocca *et al.* (2020) investigam a observabilidade como um problema de pesquisa, discutindo os benefícios de eventos como uma abstração unificada para métricas, *logs* e dados de rastreamento, e as vantagens de empregar técnicas e ferramentas de processamento de fluxo de eventos nesse contexto [41]. Esse trabalho utiliza o padrão anterior ao OTel, conhecido como OpenTracing. A latência é extraída dos *traces*, bem como as métricas de infraestrutura são coletadas periodicamente e os *logs* são estruturados.

Com relação aos propósitos apresentados na taxonomia, a detecção de anomalias é verificada em quatro trabalhos ([20], [19], [39], [49]). Khichane *et al.* (2023) apresentam um *framework* chamado 5GC-Observer para a observabilidade dos serviços de rede 5G nativos de nuvem [20]. A solução utiliza um método estatístico baseado no Z-score para garantir a detecção de degradação de desempenho no sistema Core 5G, identificando as anomalias. 5GC-Observer utiliza um agente para a coleta tanto de métricas quanto de *logs* de tráfego de rede e relacionados ao sistema 5G. As métricas coletadas são de aplicação (Tempo de resposta) e infraestrutura (CPU/Memória). Kawasaki *et al.* (2023) apresentam uma solução que utiliza o eBPF para coletar informações detalhadas da infraestrutura e desenvolveram um modelo de previsão de falhas para a identificação de anomalias [19]. Esse trabalho realiza apenas a coleta de métricas, e elas são de aplicação (Latência) e infraestrutura (CPU). Ren *et al.*

(2023) apresentam o Triple, uma abordagem de detecção de anomalias baseada em *deep learning* para microsserviços. O Triple utiliza uma representação de grafo para descrever o relacionamento de dependência dos *traces*, métricas e *logs* coletados por um agente nos nós [39]. Com relação aos *traces*, são extraídas informações da latência das requisições. As métricas coletadas são de aplicação e infraestrutura. Sundqvist *et al.* (2023) apresentam um guia de rastreamento que permite uma divisão baseada em componentes e procedimentos dos registros do sistema para a Rede de Acesso por Rádio (RAN) 5G. À medida que o sistema pode ser dividido em partes menores, os modelos podem aprender com mais precisão o comportamento do sistema e usar o contexto para melhorar a detecção de anomalias e a observabilidade [49]. Esse trabalho utiliza um agente para realizar a coleta dos três domínios para prover essa detecção. Da mesma forma que o trabalho anterior, são extraídas informações da latência das requisições, pois através dela é possível medir o tempo entre os eventos em um procedimento para detectar os atrasos e, consequentemente, as anomalias. Esse trabalho utiliza métricas de infraestrutura (CPU e Memória) para medir o desempenho do sistema, e os *logs* são estruturados para facilitar os métodos de aprendizado de máquina utilizados na solução.

Com relação aos propósitos apresentados na taxonomia, a Análise de Causa Raiz (RCA) é verificada em três trabalhos ([54], [56], [14]). Wang *et al.* (2021) apresentam o Groot, uma abordagem baseada em grafo de causalidade em tempo real com base em eventos que resumem vários tipos de métricas, *logs* e *traces* no sistema em análise para RCA [54]. O trabalho utiliza de agente para realizar as coletas, sendo que as dependências entre os microsserviços são utilizados para indicar a possível causa raiz do problema. Yu *et al.* (2023) apresentam o Nezha, uma abordagem de RCA, que utiliza o OTel, que localiza as causas raiz no nível da região de código e tipo de recurso por meio da análise de dados multimodais (os três domínios). A ideia central de Nezha é comparar padrões de eventos na fase sem falhas com aqueles na fase de falha para localizar as causas raiz de forma interpretável [56]. Gan *et al.* (2021) apresentam o Sage, um sistema de RCA impulsionado por aprendizado de máquina para microsserviços em nuvem interativos que se concentra na praticidade e escalabilidade. O Sage utiliza modelos de ML não supervisionados para contornar o custo da rotulagem de rastreamento, captura periodicamente o impacto das dependências entre microsserviços para determinar a causa raiz de desempenho imprevisível, e aplica ações corretivas para recuperar a *QoS* de um serviço em nuvem [14].

Com relação aos propósitos apresentados na taxonomia, a Engenharia do Caos é verificada em dois trabalhos ([46], [33]). Simonsson *et al.* (2021) apresentam um sistema de injeção de falhas chamado ChaosOrca para chamadas de sistema em aplicações baseadas em contêiner. O ChaosOrca tem como objetivo avaliar a capacidade de autoproteção de uma aplicação específica em relação a erros de chamadas de sistema, através de experimentos sob uma carga de trabalho semelhante à produção sem instrumentar a aplicação [46]. O trabalho utiliza apenas o tipo métricas, focando em infraestrutura (CPU / Memória) e aplicação (Taxa de sucesso). Naqvi *et al.* (2022) apresentam o CHESS, uma abordagem para a avaliação sistemática de sistemas autoadaptativos e de autorreparo que se baseia na Engenharia do Caos, utilizando os três tipos de domínios para o entendimento da aplicação [33].

Os demais trabalhos utilizam a observabilidade para outros propósitos. Cassé *et al.* (2022) apresentam uma solução que utiliza rastreamento distribuído, baseado no OTel, para identificar gargalos [6], através de um grafo de propriedades hierárquicas para mostrar gargalos, tanto pela latência como pela dependência entre os microsserviços, em uma aplicação que segue o modelo de rede em camadas Nuvem-IoT. Monteiro *et al.* (2023) apresentam uma solução adaptativa de observabilidade, por meio do OTel, baseada em teoria dos jogos, que fornece capacidades de coleta de evidências para sistemas de microsserviços e se adapta continuamente para melhorar a prontidão forense dos microsserviços e, assim, observar os microsserviços antes da ocorrência de incidentes de segurança[32]. De Vries *et al.* (2023) apresentam uma solução baseada em um conjunto de monitoramento de desempenho de aplicativos de código aberto (OTel), que fornece *insights* sobre o perfil de custos dos vários componentes da aplicação e orientação para a tomada de decisões baseada nesse perfil [8]. Chow *et al.* (2022) apresentam o DeepRest, um sistema de estimativa de recursos impulsionado por *deep learning*. O DeepRest formula a estimativa de recursos como uma função do tráfego de API e aprende a causalidade entre as interações do usuário, utilizando a topologia do *trace*, e a utilização de recursos diretamente em um ambiente de produção, com métricas de infraestrutura (CPU / Memória) [7]. Son *et al.* (2023) apresentam o MicroBlend, um *framework* que fornece uma abordagem automatizada para combinar recursos levando em consideração as dependências de microsserviços, para melhorar o desempenho e gerenciar o custo da aplicação [48]. O desempenho é avaliado por métricas de infraestrutura (CPU/Memória) e aplicação (Latência) coletadas periodicamente. Li *et al.* (2023) apresentam um *framework* de Agendamento Orientado a Topologia de Microsserviços (MOTAS), que utiliza topologias de microsserviços e clusters para otimizar o *overhead* de rede de aplicações de microsserviços por meio de um algoritmo de mapeamento de grafos heurísticos, bem como detectar e lidar com violações de QoS (throughput) em aplicações de microsserviços [29]. Cassé *et al.* (2021) utilizam rastreamentos distribuídos do OTel e informações de alocação de microsserviços para detectar recursos ineficientes em aplicações baseadas em microsserviços [5].

De acordo com os resultados encontrados, é possível verificar que a observabilidade para aplicações baseadas em microsserviços pode ser utilizada para diversos propósitos, com destaque para o escalonamento automático e *insights* (38,46% das soluções). O principal ambiente de execução da observabilidade é a Nuvem (88,46% das soluções). Além disso, a maioria das soluções utiliza os três tipos de dados (Métricas, Logs e *Traces*) para entender o comportamento da aplicação e realizar as possíveis tomadas de decisões (46,15% das soluções). Também é possível verificar que algumas soluções utilizam OTel ou JSON para facilitar a integração e padronização com outras ferramentas. O OTel é utilizado por 7 soluções, além de ser citada por 12 trabalhos (46,15% das soluções), o que mostra a relevância desse padrão da indústria. A fim de facilitar a coleta de dados pelas soluções, a maioria delas utiliza agentes (65,38% das soluções). Quanto à instrumentação, algumas optam por uma abordagem sem código, em que não é necessário nenhuma alteração pelo desenvolvedor, enquanto outras preferem algo mais manual. Algumas soluções utilizam amostragem para diminuir a quantidade de *traces* coletados. Em relação às informações extraídas dos *traces*, pode-se destacar que a maioria das soluções utiliza a

latência e dependências entre os microsserviços (50% das soluções). Quanto às métricas, a maioria das soluções utiliza infraestrutura (com destaque para CPU e Memória) e aplicação (com destaque para Tempo de resposta). Em relação aos *logs*, algumas utilizam *logs* estruturados e outros não estruturados. Por fim, a frequência mais adotada para a coleta das métricas é a periódica. Estes resultados contribuem para responder a **Questão de Pesquisa**, ilustrando que existem diferentes domínios e categorias de observabilidade para aplicações baseadas em microsserviços, mostrando a necessidade de existir uma taxonomia que classifique as soluções, de tal forma que ajude os pesquisadores em novas propostas.

5.2 Ameaças à Validade

O presente trabalho apresenta ameaças à sua validade, sendo as mais notórias aquelas relacionadas (i) à identificação dos Estudos Primários, (ii) à extração dos dados utilizados e (iii) à síntese destes dados.

Em relação à primeira classe de ameaças, Alguns estudos podem não ser identificados durante o processo de seleção de artigos. No entanto, diferentes palavras-chave foram utilizadas e várias análises foram realizadas na *string* de busca. Em relação às bases de dados, foram escolhidas aquelas amplamente utilizadas em estudos secundários no campo da engenharia de software. Por fim, para mitigar o viés no processo de seleção, um protocolo de revisão foi definido.

Já em relação à extração de dados, estas informações dos artigos foram motivados pela pergunta de pesquisa do protocolo de revisão. Em caso de dúvidas, os revisores discutiram entre si até chegar a um consenso sobre as informações extraídas.

Por fim, a síntese foi mitigada na medida em que categorias foram utilizadas para agrupar os trabalhos identificados. Além disso, estatísticas descritivas foram usadas para resumir os resultados da Revisão de Literatura.

6 DESAFIOS

Essa Seção apresenta os desafios de pesquisa no tema de Observabilidade verificados na revisão sistemática realizada.

Aumento da Dinâmica e Complexidade: A tendência emergente de arquiteturas de microsserviços, implantações em nuvem e DevOps aumenta a complexidade dos sistemas distribuídos. Enquanto a complexidade individual de um microsserviço é reduzida, a complexidade das interdependências de microsserviços e dos componentes dinâmicos dentro de um sistema distribuído requer mais esforço operacional.

Grande quantidade de dados Espera-se que uma plataforma de observabilidade lide com grandes quantidades de dados. A gestão e o processamento de tais quantidades é uma questão crítica que necessita de investigação adicional. Além disso, diversas complexidades são encontradas pelos operadores ao correlacionar métricas e *logs* com carimbo de data/hora de vários serviços, o que é frequentemente acompanhado por metadados insuficientes. Da mesma forma, o uso combinado com rastreamento de sistema para inspecionar o que acontece dentro e entre microsserviços é um desafio.

Plataforma de Observabilidade Tudo-em-um: As plataformas que oferecem todas as funções de observabilidade como um único produto são essenciais. Assim, mais esforços em padrões abertos e tecnologias integradas para processamento de dados merecem investigações mais aprofundadas.

Fusão de Dados para Análise de *Traces*: As falhas de microsserviços podem estar enraizadas em configurações ambientais, implantação, interações de serviços e implementação de serviços. Incidentes de sistemas e infraestrutura em nuvem também podem influenciar os rastros de serviços. Portanto, a análise de rastreamento para detecção de anomalias e análise de causa raiz requer a combinação de outros dados, como configurações ambientais, métricas do sistema e logs de aplicativos. A fusão de dados é um desafio, pois diferentes tipos de dados são especificados em várias dimensões e gerados em diferentes frequências.

Escalonamento Automático baseado em Observabilidade: A natureza dinâmica do tamanho e da quantidade de recursos em CNA é altamente dependente de vários fatores. As necessidades de elasticidade podem ser extremamente voláteis, com picos esporádicos que exigem um tempo de reação muito rápido para serem transparentes para os usuários finais. O desafio consiste em verificar como a observabilidade pode permitir a detecção esses picos e indicar a necessidade do escalonamento automaticamente.

7 CONCLUSÃO E TRABALHOS FUTUROS

Medir o desempenho global de um microsserviço é crucial para garantir o cumprimento das métricas de QoS da aplicação. O sistema deve expor adequadamente seu estado por meio de técnicas de instrumentação, a fim de atender aos parâmetros estabelecidos no SLA. A Observabilidade é frequentemente composta por métricas, *logs* e *traces*, sendo uma característica essencial em ambientes nativos de nuvem, pois seu principal objetivo é tornar a aplicação mais compreensível em vários níveis. Este artigo detalha o papel da observabilidade para aplicações baseadas em microsserviços. As características da observabilidade para esse ambiente foram apresentadas e analisadas, com o objetivo de aumentar a base de conhecimento sobre esse novo campo de monitoramento. Uma taxonomia de soluções de observabilidade para aplicações baseadas em microsserviços foi criada a partir dos domínios e categorias mais relevantes na área. Para validar a taxonomia e oferecer aos pesquisadores uma análise abrangente das propostas de observabilidade para aplicações baseadas em microsserviços disponíveis, elas foram analisadas e categorizadas pela taxonomia, mostrando sua utilidade. Como trabalhos futuros, será (i) estendido esse estudo para um dos propósitos mais explorados pelas soluções, que foi o de escalonamento automático, a fim de realizar uma classificação dessa área e (ii) investigadas as ferramentas e aplicações *benchmarkings* mais utilizadas nos trabalhos, a fim de saber o que é mais utilizado nessa área.

DISPONIBILIDADE DE ARTEFATOS

A planilha contendo os dados referentes ao estudo realizado pode ser acessada em https://tinyurl.com/sbes2024-observability

AGRADECIMENTOS

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

REFERÊNCIAS

[1] Nuha Alshuqayran, Nour Ali, and Roger Evans. 2016. A systematic mapping study in microservice architecture. In *2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA)*. IEEE, 44–51.

[2] Ataollah Fatahi Baarzi and George Kesidis. 2021. Showar: Right-sizing and efficient scheduling of microservices. In *Proceedings of the ACM Symposium on Cloud Computing*. 427–441.

[3] Armin Balalaie, Abbas Heydarnoori, and Pooyan Jamshidi. 2016. Microservices architecture enables devops: Migration to a cloud-native architecture. *Ieee Software* 33, 3 (2016), 42–52.

[4] A. Boten and C. Majors. 2022. *Cloud-Native Observability with OpenTelemetry: Learn to gain visibility into systems by combining tracing, metrics, and logging with OpenTelemetry*. Packt Publishing. https://books.google.com.br/books?id=YZVsEAAAQBAJ

[5] Clément Cassé, Pascal Berthou, Philippe Owezarski, and Sébastien Josset. 2021. Using distributed tracing to identify inefficient resources composition in cloud applications. In *2021 IEEE 10th International Conference on Cloud Networking (CloudNet)*. IEEE, 40–47.

[6] Clément Cassé, Pascal Berthou, Philippe Owezarski, and Sébastien Josset. 2022. A tracing based model to identify bottlenecks in physically distributed applications. In *2022 International Conference on Information Networking (ICOIN)*. IEEE, 226–231.

[7] Ka-Ho Chow, Umesh Deshpande, Sangeetha Seshadri, and Ling Liu. 2022. DeepRest: deep resource estimation for interactive microservices. In *Proceedings of the Seventeenth European Conference on Computer Systems*. 181–198.

[8] Sjouke de Vries, Frank Blaauw, and Vasilios Andrikopoulos. 2023. Cost-Profiling Microservice Applications Using an APM Stack. *Future Internet* 15, 1 (2023), 37.

[9] Shuiguang Deng, Hailiang Zhao, Binbin Huang, Cheng Zhang, Feiyi Chen, Yinuo Deng, Jianwei Yin, Schahram Dustdar, and Albert Y Zomaya. 2023. Cloud-Native Computing: A Survey from the Perspective of Services. *arXiv preprint arXiv:2306.14402* (2023).

[10] Paolo Di Francesco, Ivano Malavolta, and Patricia Lago. 2017. Research on architecting microservices: Trends, focus, and potential for industrial adoption. In *2017 IEEE International conference on software architecture (ICSA)*. IEEE, 21–30.

[11] António Pedro dos Santos Carvalho. 2022. Observabilidade e telemetria em arquiteturas de micro-serviços. (2022).

[12] Nicola Dragoni, Saverio Giallorenzo, Alberto Lluch Lafuente, Manuel Mazzara, Fabrizio Montesi, Ruslan Mustafin, and Larisa Safina. 2017. Microservices: yesterday, today, and tomorrow. *Present and ulterior software engineering* (2017), 195–216.

[13] Paul M Duvall, Steve Matyas, and Andrew Glover. 2007. *Continuous integration: improving software quality and reducing risk*. Pearson Education.

[14] Yu Gan, Mingyu Liang, Sundar Dev, David Lo, and Christina Delimitrou. 2021. Sage: practical and scalable ML-driven performance debugging in microservices. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 135–151.

[15] Dennis Gannon, Roger Barga, and Neel Sundaresan. 2017. Cloud-native applications. *IEEE Cloud Computing* 4, 5 (2017), 16–21.

[16] Jez Humble and David Farley. 2010. *Continuous delivery: reliable software releases through build, test, and deployment automation*. Pearson Education.

[17] Rudolf E Kalman. 1960. On the general theory of control systems. In *Proceedings First International Conference on Automatic Control, Moscow, USSR*. 481–492.

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

[19] Junichi Kawasaki, Daiki Koyama, Takuya Miyasaka, and Tomohiro Otani. 2023. Failure Prediction in Cloud Native 5G Core With eBPF-based Observability. In *2023 IEEE 97th Vehicular Technology Conference (VTC2023-Spring)*. IEEE, 1–6.

[20] Abderaouf Khichane, Ilhem Fajjari, Nadjib Aitsaadi, and Mourad Gueroui. 2023. 5GC-Observer: a Non-intrusive Observability Framework for Cloud Native 5G System. In *NOMS 2023-2023 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 1–10.

[21] Joanna Kosińska, Bartosz Baliś, Marek Konieczny, Maciej Malawski, and Sławomir Zieliński. 2023. Towards the Observability of Cloud-native applications: The Overview of the State-of-the-Art. *IEEE Access* (2023).

[22] Joanna Kosińska and Krzysztof Zieliński. 2020. Autonomic management framework for cloud-native applications. *Journal of Grid Computing* 18 (2020), 779–796.

[23] Nane Kratzke. 2022. Cloud-native observability: the many-faceted benefits of structured and unified logging—a multi-case study. *Future Internet* 14, 10 (2022), 274.

[24] Nane Kratzke and Peter-Christian Quint. 2017. Understanding cloud-native applications after 10 years of cloud computing-a systematic mapping study. *Journal of Systems and Software* 126 (2017), 1–16.

[25] Abhijit Kumar, Tauseef Ahmed, Konica Saini, and Jay Kumar. 2023. NEOS: Non-intrusive Edge Observability stack based on Zero Trust security model for Ubiquitous Computing. In *2023 IEEE International Conference on Edge Computing and Communications (EDGE)*. IEEE, 79–84.

[26] Joshua Levin and Theophilus A Benson. 2020. ViperProbe: Rethinking microservice observability with eBPF. In *2020 IEEE 9th International Conference on Cloud Networking (CloudNet)*. IEEE, 1–8.

[27] James Lewis and Martin Fowler. 2014. Microservices: a definition of this new architectural term. *MartinFowler. com* 25, 14-26 (2014), 12.

[28] Bowen Li, Xin Peng, Qilin Xiang, Hanzhang Wang, Tao Xie, Jun Sun, and Xuanzhe Liu. 2022. Enjoy your observability: an industrial survey of microservice tracing and analysis. *Empirical Software Engineering* 27 (2022), 1–28.

[29] Xin Li, Junsong Zhou, Xin Wei, Dawei Li, Zhuzhong Qian, Jie Wu, Xiaolin Qin, and Sanglu Lu. 2023. Topology-Aware Scheduling Framework for Microservice Applications in Cloud. *IEEE Transactions on Parallel and Distributed Systems* 34, 5 (2023), 1635–1649.

[30] Nicolas Marie-Magdelaine. 2021. *Observability and resources managements in cloud-native environnements*. Ph. D. Dissertation. Bordeaux.

[31] Nicolas Marie-Magdelaine and Toufik Ahmed. 2020. Proactive autoscaling for cloud-native applications using machine learning. In *GLOBECOM 2020-2020 IEEE Global Communications Conference*. IEEE, 1–7.

[32] Davi Monteiro, Yijun Yu, Andrea Zisman, and Bashar Nuseibeh. 2023. Adaptive Observability for Forensic-Ready Microservice Systems. *IEEE Transactions on Services Computing* (2023).

[33] Moeen Ali Naqvi, Sehrish Malik, Merve Astekin, and Leon Moonen. 2022. On Evaluating Self-Adaptive and Self-Healing Systems using Chaos Engineering. In *2022 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS)*. IEEE, 1–10.

[34] Sina Niedermaier, Falko Koetter, Andreas Freymann, and Stefan Wagner. 2019. On observability and monitoring of distributed systems–an industry interview study. In *Service-Oriented Computing: 17th International Conference, ICSOC 2019, Toulouse, France, October 28–31, 2019, Proceedings 17*. Springer, 36–52.

[35] Claus Pahl, Antonio Brogi, Jacopo Soldani, and Pooyan Jamshidi. 2017. Cloud container technologies: a state-of-the-art review. *IEEE Transactions on Cloud Computing* 7, 3 (2017), 677–692.

[36] Claus Pahl and Pooyan Jamshidi. 2016. Microservices: A Systematic Mapping Study. *CLOSER (1)* (2016), 137–146.

[37] Rodolfo Picoreti, Alexandre Pereira do Carmo, Felippe Mendonca de Queiroz, Anilton Salles Garcia, Raquel Frizera Vassallo, and Dimitra Simeonidou. 2018. Multilevel observability in cloud orchestration. In *2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech)*. IEEE, 776–784.

[38] William Pourmajidi, Lei Zhang, John Steinbacher, Tony Erwin, and Andriy Miranskyy. 2023. A Reference Architecture for Observability and Compliance of Cloud Native Applications. *arXiv preprint arXiv:2302.11617* (2023).

[39] Rui Ren, Yang Wang, Fengrui Liu, Zhenyu Li, and Gaogang Xie. 2023. Triple: The Interpretable Deep Learning Anomaly Detection Framework based on Trace-Metric-Log of Microservice. In *2023 IEEE/ACM 31st International Symposium on Quality of Service (IWQoS)*. IEEE, 1–10.

[40] Chris Richardson. 2018. *Microservices patterns: with examples in Java*. Simon and Schuster.

[41] Mario Scrocca, Riccardo Tommasini, Alessandro Margara, Emanuele Della Valle, and Sherif Sakr. 2020. The Kaiju project: enabling event-driven observability. In *Proceedings of the 14th ACM International Conference on Distributed and Event-Based Systems*. 85–96.

[42] Mali Senapathi, Jim Buchan, and Hady Osman. 2018. DevOps capabilities, practices, and challenges: Insights from a case study. In *Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018*. 57–67.

[43] Mojtaba Shahin, Muhammad Ali Babar, and Liming Zhu. 2017. Continuous integration, delivery and deployment: a systematic review on approaches, tools, challenges and practices. *IEEE access* 5 (2017), 3909–3943.

[44] Anas Shatnawi, Bachar Rima, Zakarea Alshara, Gabriel Darbord, Abdelhak-Djamel Seriai, and Christophe Bortolaso. 2023. Telemetry of Legacy Web Applications: An Industrial Case Study. (2023).

[45] Francisco Silva, Valéria Lelli, Ismayle Santos, and Rossana Andrade. 2022. Towards a fault taxonomy for microservices-based applications. In *Proceedings of the XXXVI Brazilian Symposium on Software Engineering*. 247–256.

[46] Jesper Simonsson, Long Zhang, Brice Morin, Benoit Baudry, and Martin Monperrus. 2021. Observability and chaos engineering on system calls for containerized applications in docker. *Future Generation Computer Systems* 122 (2021), 117–129.

[47] Andy Singleton. 2016. The economics of microservices. *IEEE Cloud Computing* 3, 5 (2016), 16–20.

[48] Myungjun Son, Shruti Mohanty, Jashwant Raj Gunasekaran, and Mahmut Kandemir. 2023. MicroBlend: An Automated Service-Blending Framework for Microservice-Based Cloud Applications. In *2023 IEEE 16th International Conference on Cloud Computing (CLOUD)*. IEEE, 460–470.

[49] Tobias Sundqvist, Monowar Bhuyan, and Erik Elmroth. 2023. Robust procedural learning for anomaly detection and observability in 5G RAN. *IEEE Transactions on Network and Service Management* (2023).

[50] Johannes Thönes. 2015. Microservices. *IEEE software* 32, 1 (2015), 116–116.
[51] Ioannis Tzanettis, Christina-Maria Androna, Anastasios Zafeiropoulos, Eleni Fotopoulou, and Symeon Papavassiliou. 2022. Data Fusion of Observability Signals for Assisting Orchestration of Distributed Applications. *Sensors* 22, 5 (2022), 2061.
[52] Muhammad Usman, Simone Ferlin, Anna Brunstrom, and Javid Taheri. 2022. A Survey on Observability of Distributed Edge "&"Container-Based Microservices. *IEEE Access* 10 (2022), 86904–86919. https://doi.org/10.1109/ACCESS.2022.3193102
[53] Muhammad Usman, Simone Ferlin, Anna Brunstrom, and Javid Taheri. 2023. DESK: Distributed Observability Framework for Edge-Based Containerized Microservices. In *2023 Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit)*. IEEE, 617–622.
[54] Hanzhang Wang, Zhengkai Wu, Huai Jiang, Yichao Huang, Jiamu Wang, Selcuk Kopru, and Tao Xie. 2021. Groot: An event-graph-based approach for root cause analysis in industrial settings. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 419–429.
[55] Johannes Wettinger, Vasilios Andrikopoulos, Frank Leymann, and Steve Strauch. 2016. Middleware-oriented deployment automation for cloud applications. *IEEE Transactions on Cloud Computing* 6, 4 (2016), 1054–1066.
[56] Guangba Yu, Pengfei Chen, Yufeng Li, Hongyang Chen, Xiaoyun Li, and Zibin Zheng. 2023. Nezha: Interpretable Fine-Grained Root Causes Analysis for Microservices on Multi-modal Observability Data. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 553–565.
[57] Guangba Yu, Pengfei Chen, and Zibin Zheng. 2020. Microscaler: Cost-effective scaling for microservice applications in the cloud with an online learning approach. *IEEE Transactions on Cloud Computing* 10, 2 (2020), 1100–1116.