UNIVERSIDADE DE SÃO PAULO
ESCOLA POLITÉCNICA

Kayque Araujo Souza

Observabilidade em microserviços: Estratégias e Trade-offs para uma Gestão Eficiente

São Paulo
2025
Kayque Araujo Souza

Observabilidade em microserviços: estratégias e trade-offs para uma gestão eficiente

Trabalho de Conclusão de Curso apresentado a Escola Politécnica da Universidade de São Paulo, como parte dos requisitos para a obtenção do título de Mestre em Engenharia de Software

Orientadora: Profa. Dra. Gabriela Cabel Barbarán

São Paulo

2025

Nome: SOUZA, Kayque Araujo

Título: Observabilidade em microsserviços: estratégias e *trade-off* para uma gestão eficiente

Trabalho de Conclusão de Curso apresentado a Escola Politécnica da Universidade de São Paulo, como parte dos requisitos para a obtenção do título de Mestre em Engenharia de Software

Aprovado em: / /

Banca Examinadora

Prof(a). Dr(a).: _____

Instituição: _____

Julgamento: _____

Prof(a). Dr(a).: _____

Instituição: _____

Julgamento: _____

Prof(a). Dr(a).: _____

Instituição: _____

Julgamento: _____

Prof(a). Dr(a).: _____

Instituição: _____

Julgamento: _____

RESUMO

SOUZA, K. A. **Observabilidade em microsserviços:** estratégias e trade-offs para uma gestão eficiente. 2025. Monografia (MBA em Engenharia de Software) – Programa de Educação Continuada em Engenharia da Escola Politécnica da Universidade de São Paulo, São Paulo, 2025.

Com a adoção em larga escala da arquitetura de microsserviços, surgiram a necessidade de mudanças significativas nos ambientes de desenvolvimento. A expansão da cultura DevOps levou a um monitoramento intensificado do software, impulsionado pela complexidade desses novos sistemas. Nesse contexto, a Observabilidade tornou-se um elemento fundamental para compreender o estado de aplicações descentralizadas, abordando não apenas a coleta de dados, mas também a forma de agir com base nas informações obtidas. Este estudo tem como objetivo apresentar abordagens para a Observabilidade em ambientes de desenvolvimento de software, permitindo que, a partir das informações coletadas, sejam realizadas adaptações e implementações eficazes. Além disso, busca-se discutir os trade-offs associados às diferentes estratégias de Observabilidade, visando otimizar a utilização de recursos e aprimorar o desempenho das aplicações. A análise das melhores práticas proporcionará às equipes de desenvolvimento a capacidade de tomar decisões informadas, contribuindo não apenas para a estabilidade, mas também para a eficiência no consumo de recursos, resultando em um sistema mais eficaz em um cenário de constante evolução tecnológica. Para isso, foram revisados artigos recentes, publicados a partir de 2022, que abordam diferentes estratégias e considerações importantes para a tomada de decisão no design de Observabilidade.

Palavras-chave: Observabilidade; DevOps; Coleta de Dados; Microsserviços.

ABSTRACT

SOUZA, K. A. **Observability in Microservices:** Strategies and Trade-offs for Efficient Management. 2025. Monograph (MBA em Engenharia de Software) – Escola Politécnica, Universidade de São Paulo, São Paulo, 2025.

With the widespread adoption of microservices architecture, there has been a need for significant changes in development environments. The expansion of the DevOps culture has led to intensified monitoring of software, driven by the complexity of these new systems. In this context, observability has become a fundamental element for understanding the state of decentralized applications, addressing not only data collection but also how to act based on the information obtained. This study aims to present approaches to observability in software development environments, allowing for adaptations and effective implementations based on the collected information. Additionally, it seeks to discuss the trade-offs associated with different observability strategies, aiming to optimize resource utilization and enhance application performance. The analysis of best practices will provide development teams with the ability to make informed decisions, contributing not only to stability but also to efficiency in resource consumption, resulting in a more effective system in a constantly evolving technological landscape. To this end, recent articles published from 2022 onwards were reviewed, addressing different strategies and important considerations for decision-making in observability design.

Keywords: Observability; DevOps; Data Collection; Microservices.

SUMÁRIO

1 INTRODUÇÃO

1.1 Motivação

Nas últimas décadas, a área de desenvolvimento de software tem observado uma crescente adoção da arquitetura baseada em microsserviços e da cultura DevOps, impulsionada pela necessidade de atualização constante e entrega ágil de novas funcionalidades com aplicações cada vez mais decentralizadas e complexas. (BROWN, 2024).

Essa nova abordagem trouxe à tona uma série de boas práticas que capacitam os times de desenvolvimento. Um estudo realizado pela empresa Puppet (PUPPET, 2024) estima que, em 2023, mais de 80% das empresas já implementaram processos baseados em DevOps. Um dos conceitos que ganhou destaque nos últimos anos, como resultado dessa nova mentalidade em relação ao desenvolvimento de software, é o da "Observabilidade". Este conceito se refere à capacidade de monitorar o software com o objetivo de coletar dados que auxiliem na manutenção da estabilidade, na análise de custos, na identificação de gargalos, entre outros (AWS, [s.d.]).

Diversos trabalhos acadêmicos têm analisado e avaliado abordagens de observabilidade em arquiteturas de microsserviços. O estudo (GOMES, REGO e TRINTA, 2024) apresenta uma taxonomia de observabilidade de software dentro desse ambiente, servindo como um cardápio de opções que podem ser empregadas em sistemas monitorados. O trabalho (CHAVA, 2022) discute formas de aprimorar a observabilidade e sua eficácia, além de justificar as vantagens percebidas ao implementar tais práticas em ambientes de software. O estudo (BORGES et al., 2024) demonstra uma abordagem para organizar os atributos observados, permitindo a análise de redundâncias nas informações coletadas. Por fim, a pesquisa (USMAN et al., 2022) compila informações de mercado, catalogando os principais gargalos relacionados à implementação de observabilidade.

1.2 Objetivo

Este trabalho tem como objetivo analisar os estudos mais recentes sobre observabilidade em ambientes de software baseados em microsserviços, apresentando uma visão geral das decisões que podem ser tomadas durante a manutenção desses ambientes.

Serão discutidos os possíveis trade-offs de cada abordagem, sempre com foco na eficácia e na minimização do desperdício de recursos. Com essas informações, espera-se que o leitor obtenha uma introdução aos principais pontos de discussão sobre observabilidade, além de uma apresentação de uma gama de decisões que podem ser adotadas para direcionar os esforços, tanto no ambiente de software em questão quanto em sistemas de terceiros.

1.3 Justificativa

Diversas ferramentas de observabilidade, como DataDog, Grafana e SolarWinds, oferecem uma ampla gama de opções para métricas e monitoramento. No entanto, o custo associado à coleta e análise de todas essas informações é elevado, e pouco se discute sobre boas práticas que possam proporcionar um bom custo-benefício. A abordagem mais comum é a de tentativa e erro, onde as equipes optam por não monitorar determinadas características do software na esperança de que essas informações não sejam críticas, até que, eventualmente, um bug ou erro no sistema prove o contrário (USMAN et al., 2022).

Essa falta de clareza sobre o que deve ser monitorado de forma eficaz tem gerado problemas significativos. A tentativa de monitorar todos os aspectos de um sistema é inviável devido ao alto custo, e a única solução viável é priorizar quais informações devem ser observadas. Contudo, essa priorização é complexa, especialmente em um cenário onde muitas aplicações são compostas por microsserviços, cada um com seu próprio contexto, exigências e importância dentro da arquitetura do sistema.

1.4 Método de Pesquisa

Este trabalho consiste em uma revisão sistemática de artigos que abordam primordialmente o papel e os desafios da observabilidade de software dentro da cultura DevOps de desenvolvimento de software. Muitos estudos relatam como implementar a observabilidade em uma arquitetura de microsserviços, enquanto outros discutem as vantagens e os benefícios que uma organização pode obter ao adotar tais práticas. Além disso, há trabalhos que avaliam as principais ferramentas de observabilidade atualmente disponíveis no mercado. No entanto, estamos em um momento em que o padrão é já ter, ao menos, algum nível de observabilidade do

software. Assim, este trabalho tem como objetivo apresentar decisões de arquitetura que visem melhorar a eficácia de um sistema já implementado.

1.4.1 Escopo

O escopo da pesquisa prioriza artigos que discutem as possíveis técnicas que podem ser aplicadas para aprimorar a eficácia de um ambiente observado, dando preferência a produções mais recentes, especialmente aquelas publicadas a partir de 2022. Contudo, essa não é uma exigência absoluta.

1.4.2 Pesquisa

Os artigos foram pesquisados em mecanismos de busca acadêmicos amplamente utilizados, como IEEE Xplore, Google Acadêmico e o Portal de Periódicos da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), seguindo as diretrizes estabelecidas no escopo da pesquisa. Diversas palavras-chave foram utilizadas para a busca dos documentos, garantindo uma abrangência adequada na coleta de informações relevantes para o tema em questão:

a) Observabilidade: Sendo este o tema central do trabalho, essa palavra-chave foi, de longe, a mais utilizada na pesquisa dos artigos.
b) Microsserviços: Atualmente, a arquitetura de microsserviços é o padrão de mercado. Portanto, é importante que os artigos abordem os problemas e desafios contemporâneos relacionados a essa abordagem.
c) DevOps: A observabilidade é uma característica fundamental da cultura DevOps. Assim, artigos que mencionam essa palavra-chave estão intimamente relacionados ao tipo de observabilidade que esta pesquisa discute.
d) Overhead: Um dos problemas mais comuns apresentados na literatura é o overhead associado à coleta de informações. Artigos que abordam esse problema e discutem suas implicações também oferecem possíveis soluções, não apenas para essa questão, mas para outros desafios relacionados à observabilidade.

É importante observar que as palavras-chave mencionadas foram pesquisadas tanto em inglês quanto em português, exceto em casos em que o termo em português mais utilizado também é uma palavra em inglês, como "*overhead*" ou "*DevOps*". Além

disso, algumas palavras foram excluídas da pesquisa, como "*log*", "*trace*" ou "*monitoring*", a fim de evitar que as buscas fossem direcionadas e focadas apenas em um desses aspectos da observabilidade. Essa abordagem visa garantir uma abrangência maior na coleta de artigos relevantes para a pesquisa.

1.4.3 **Seleção dos Artigos**

Foram analisados pouco mais de 15 artigos, todos lidos com o objetivo de avaliar se apresentavam soluções e decisões arquitetônicas relevantes que poderiam ser implementadas em um ambiente de software. Dentre eles, os principais selecionados foram:

(GOMES, REGO e TRINTA, 2024): Este estudo catalogou uma forma de taxonomia de observabilidade, utilizando também uma revisão sistemática. Os autores avaliaram diversos artigos que tratam exclusivamente da observabilidade, organizando-os em categorias e identificando semelhanças, o que permitiu extrair uma visão abrangente dos diversos temas e variações que compõem os processos de observação do software.

(CHAVA, 2022): Este artigo apresentou uma visão com diversos pontos de melhoria dentro da observabilidade, focando nas principais práticas que devem ser priorizadas para alcançar o sucesso na implementação. Além disso, discutiu os principais erros e problemas que podem surgir durante a implementação e como abordá-los. Embora seja um trabalho conciso, ele destaca pontos-chave que merecem atenção.

(USMAN et al., 2022): Os autores realizaram uma pesquisa sobre diversos artigos recentes que tratam da observabilidade em software distribuído baseado em microsserviços, catalogando cada uma das pesquisas por categoria. Após essa organização, observaram as ideias mais reforçadas por esses artigos e as apresentaram com o intuito de compartilhar os principais aprendizados do mercado.

Os artigos selecionados foram lidos na íntegra, e seus principais pontos, especialmente aqueles que mais se relacionam com o tema central deste trabalho, foram extraídos para análise e apresentação.

1.5 Estrutura do Trabalho

Nesta sessão será apresentado como a monografia foi estruturada. Será descrito capítulo por capítulo.

a) Introdução: A introdução estabelece o contexto do trabalho, apresentando a relevância da observabilidade em microsserviços. Discute a motivação para o estudo, os objetivos a serem alcançados, a justificativa da pesquisa, o método utilizado para a coleta de dados e a estrutura do trabalho.
b) Fundamentação teórica: A fundamentação teórica explora os conceitos essenciais que sustentam a observabilidade em microsserviços. Aborda temas como Software as a Service (SaaS), arquitetura de microsserviços, cultura DevOps, a definição de observabilidade, as diferenças entre observabilidade e monitoração, e os aspectos chave da observabilidade, incluindo logs, traces e métricas.
c) Decisões de design para melhora da eficácia da observabilidade em sistemas distribuídos: Esta seção discute as decisões de design que podem ser adotadas para aprimorar a eficácia da observabilidade. Inclui uma visão geral das considerações necessárias, a escolha de ferramentas, os dados principais a serem coletados, estratégias para evitar redundância na coleta de dados, a importância dos pilares da observabilidade e a coleta centralizada de dados.
d) Conclusão e trabalhos futuros: A conclusão resume os principais achados do trabalho, destacando a importância da observabilidade em microsserviços. Também sugere direções para pesquisas futuras, identificando áreas que podem ser exploradas para aprofundar o conhecimento sobre o tema.
e) Referências: Esta seção lista todas as fontes citadas ao longo do trabalho, garantindo a credibilidade e a rastreabilidade das informações apresentadas.

2 FUNDAMENTAÇÃO TEORICA

Neste capítulo, serão discutidos os conceitos e definições fundamentais que servem de base para a pesquisa em questão. A compreensão adequada desses temas é essencial para o desenvolvimento do trabalho. Em particular, serão abordados os conceitos de microsserviços, DEVOPS e observabilidade, que são relevantes para a análise proposta.

2.1 Software as a Service (SaaS)

O Software como Serviço (SaaS) é um modelo de software baseado em nuvem que permite o acesso a aplicações por meio de navegadores da Internet, sem a necessidade de gerenciar a infraestrutura subjacente. Os fornecedores de SaaS hospedam e mantêm as aplicações, oferecendo acesso sob demanda, geralmente por meio de assinaturas.

As principais vantagens do SaaS incluem:

a) Acessibilidade: Acesso de qualquer dispositivo com Internet.
b) Custos reduzidos: Menores despesas iniciais e contínuas, com manutenção coberta pela assinatura.
c) Implantação rápida: Eliminação de instalação complexa.
d) Escalabilidade: Facilidade para adicionar serviços conforme necessário.
e) Confiabilidade: Alta disponibilidade e segurança garantidas pelos provedores.
f) Atualizações automáticas: Atualizações regulares sem necessidade de suporte técnico.

Esse modelo de negócio de software exige mudanças mais rápidas, o que torna a arquitetura de microsserviços uma escolha ideal. Com a capacidade de desenvolver, implantar e escalar componentes de forma independente, os microsserviços permitem que as empresas respondam rapidamente às demandas do mercado e implementem novas funcionalidades sem interrupções significativas. Essa agilidade é crucial para provedores de SaaS, que precisam garantir que suas aplicações permaneçam competitivas e relevantes em um ambiente em constante evolução.

2.2 Arquitetura de microserviços

Microsserviços são um estilo de arquitetura de software no qual uma aplicação é composta por diversos microsserviços que operam de forma independente, mas que se comunicam, geralmente, por meio de protocolos como HTTP ou RPC. Cada um desses serviços é responsável por uma parte lógica do sistema como um todo. Não é necessário que todos os microsserviços utilizem a mesma linguagem de programação; cada um pode ser implementado de acordo com suas necessidades específicas, utilizando diferentes linguagens, tecnologias e até mesmo bancos de dados distintos, o que proporciona maior liberdade para que cada parte lógica seja desenvolvida de maneira otimizada (FOWLER, 2019).

Esse modelo arquitetônico tem ganhado destaque no mercado, impulsionado pela necessidade de atualizações constantes e pela importância de manter os serviços sempre disponíveis.

Os principais benefícios da implementação de microsserviços incluem (AWS, [s.d.]).:

a) Resiliência: Uma das características mais relevantes dos microsserviços é sua capacidade de resistir a falhas. Em um sistema monolítico, a falha de um único componente pode comprometer toda a aplicação. Com a arquitetura de microsserviços, a independência entre os serviços permite que a aplicação continue operando, mesmo que uma parte falhe, resultando em uma degradação controlada da funcionalidade.
b) Agilidade: A estrutura de microsserviços favorece a formação de equipes pequenas e autônomas, que assumem a responsabilidade total sobre seus serviços. Isso possibilita que as equipes trabalhem de maneira mais rápida e eficiente, encurtando os ciclos de desenvolvimento e aumentando a produtividade geral da organização.
c) Escalabilidade Flexível: Cada microsserviços pode ser escalado de forma independente, permitindo que as equipes ajustem a infraestrutura de acordo com a demanda específica de cada funcionalidade. Essa abordagem não apenas otimiza os recursos, mas também assegura que a aplicação permaneça disponível durante picos de uso.

d) Implantação Simples: A arquitetura de microsserviços facilita a adoção de práticas de integração e entrega contínuas. Isso permite que as equipes testem novas ideias rapidamente e revertam alterações que não funcionem, reduzindo o custo de falhas e acelerando o lançamento de novas funcionalidades.
e) Liberdade Tecnológica: Com a adoção de microsserviços, as equipes não estão restritas a uma única tecnologia ou ferramenta. Elas têm a liberdade de escolher as melhores soluções para os problemas específicos que enfrentam, o que pode resultar em resultados mais eficazes.
f) Código Reutilizável: A divisão do software em módulos bem definidos permite que as equipes reutilizem funções em diferentes contextos. Isso significa que um serviço desenvolvido para uma finalidade pode ser utilizado como base para outras funcionalidades, economizando tempo e esforço no desenvolvimento.

Essas vantagens associadas a uma arquitetura de software distribuída se alinham de maneira eficaz com a cultura DevOps, que está sendo cada vez mais implantada nos ambientes de desenvolvimento de software.

2.3 Cultura DevOps

O movimento DevOps teve seu início por volta de 2007 e 2008, em um contexto em que a comunidade de desenvolvimento estava insatisfeita com a cultura de desenvolvimento de software predominante na época (BUCHANAN, 2025). A crescente necessidade de entregar software de forma mais rápida estava se intensificando, impulsionada pela ascensão do modelo de Software como Serviço (SaaS), que demandava um aumento significativo na frequência de atualizações. No entanto, os processos de desenvolvimento existentes eram rígidos e dificultavam o progresso.

Um dos principais fatores que contribuíam para essa situação era a separação acentuada entre as áreas de desenvolvimento e operações/infraestrutura. Cada área possuía indicadores e objetivos distintos, que muitas vezes se contradiziam. Em algumas organizações, as equipes de desenvolvimento e operações estavam até localizadas em prédios separados, operando em mundos distintos.

Com o tempo, começou a ocorrer uma convergência entre essas duas áreas. As comunidades de desenvolvimento e operações começaram a se reunir em fóruns e eventos menores para discutir possíveis soluções para esse distanciamento, com o objetivo de entregar software com maior qualidade e agilidade. Esse movimento culminou na formação da cultura de desenvolvimento de software que hoje conhecemos como DevOps, que integra as áreas de DEVelopment e OPerationS.

Diversos fundamentos e conceitos foram estabelecidos para definir essa cultura DevOps, entre os quais se destacam:

a) Metodologias Ágeis: As metodologias de desenvolvimento de software que preveem entregas menores e contínuas se alinharam bem ao movimento DevOps, com destaque para o Scrum e o Extreme Programming (XP).
b) CI/CD (Integração e Entrega Contínuas): Este conceito baseia-se na prática de mesclar e implantar aplicações de forma automatizada, eliminando o atrito que existia anteriormente entre o desenvolvimento e o lançamento do produto. Atualmente, existem pipelines (esteiras de produção) que integram, constroem, testam o código e realizam o deploy automaticamente, desde que todos os testes sejam bem-sucedidos, sem a necessidade de intervenção manual.
c) Controle de Versão Automatizado: O GIT trouxe uma evolução significativa ao controle de versão de código, permitindo a automação de processos de deploy com base nas versões controladas. Além disso, facilitou consideravelmente o processo de mesclagem (merge), que anteriormente era uma fonte de dificuldades para as equipes de desenvolvimento (AKITA, 2020).

Esses elementos fundamentais contribuíram para a consolidação da cultura DevOps, promovendo uma colaboração mais eficaz entre as equipes de desenvolvimento e operações, e resultando em um ciclo de desenvolvimento de software mais ágil e eficiente.

2.4 Observabilidade

Segundo (RED HAT, 2023), observabilidade de software diz respeito a habilidade de observação do estado da aplicação, seja o estado atual, como métricas

de avaliações de performance, seja o estado passado com logs gerados pelo sistema e armazenados, ou seja, seu estado futuro com gráfico que calculam as previsões de uso de RAM de um sistema.

Já havia maneiras de observar o estado do software a muito tempo na área, porém com a cultura DevOps a observabilidade deu um passo adiante sendo implementado em massa e fazendo parte essencial de qualquer ambiente de software.

Além disso a mudança do mercado que passou a adotar microsserviços como padrão, também colaborou para esse desenvolvimento das ferramentas de observação, isso porque os ambientes ficaram decentralizados, com diversos serviços comunicando entre si gerando uma “teia” de comunicações bem mais complexa, que dificulta a manutenção.

A implementação de técnicas de observabilidade ajuda no rastreio e entendimento do sistema em aspectos micro, como o estado um dos microsserviços, e em aspectos macro como a caminho percorrido de uma requisição por diversos microsserviços.

2.5 Observabilidade vs monitoração

É fundamental destacar a diferença entre os conceitos de Observabilidade e Monitoração, uma vez que há uma confusão recorrente entre os termos. Ambos se referem ao entendimento do estado de uma aplicação, mas a observabilidade é um conceito significativamente mais amplo do que a monitoração (IBM EDUCATION, 2022).

A monitoração refere-se ao processo de acompanhar o software, armazenar e apresentar seus outputs relativos ao estado atual e passado. Para isso, é necessário definir quais aspectos serão monitorados e quais características da aplicação são suficientemente relevantes para serem coletadas. Exemplos dessas características incluem a quantidade de requisições, o registro de logs de um determinado *endpoint* e o uso diário de CPU.

Por outro lado, a observabilidade vai além da simples monitoração. Ela envolve não apenas a coleta de dados, mas também a capacidade de tomar ações com base nesses dados, realizando correlações que permitam prever ou diagnosticar possíveis problemas. A observabilidade busca entender de forma mais profunda o que está acontecendo com o sistema, transformando os dados coletados em informações significativas. Essa capacidade de análise e interpretação é crucial para a manutenção

da saúde e da eficiência operacional de sistemas complexos, especialmente em ambientes que utilizam arquiteturas de microsserviços.

2.6 Aspectos chave da observabilidade: logs, traces e métricas

A observabilidade em ambientes computacionais envolve a coleta e o monitoramento de diversos tipos de dados, sendo que três categorias fundamentais sustentam essa prática: logs, traces e métricas. A seguir, será apresentada uma breve descrição de cada um desses elementos essenciais.

a) Logs: Os logs consistem em registros temporais que documentam informações detalhadas sobre respostas ou ações que ocorrem dentro de um sistema. Esses registros podem ser estruturados ou não, apresentando uma variedade de formatos, sendo o JavaScript Object Notation (JSON) o mais comum. A coleta de logs é crucial para a compreensão do estado anterior da aplicação, permitindo, por exemplo, rastrear o momento em que um sistema começou a apresentar alterações em seu comportamento. Essa capacidade de rastreamento facilita o diagnóstico de mudanças inesperadas no sistema, proporcionando insights valiosos para a resolução de problemas.

b) Métricas: As métricas, por sua vez, refletem o estado atual da aplicação, consistindo em dados coletados geralmente em tempo real. Em algumas situações, pode ser vantajoso realizar coletas em intervalos específicos, um aspecto que será discutido mais adiante neste estudo. Exemplos de métricas incluem a utilização atual de RAM ou CPU, o número de requisições por minuto e o tempo médio de resposta da aplicação. Através da análise de métricas, a identificação de anomalias torna-se significativamente mais ágil. Equipes que monitoram métricas podem diagnosticar problemas e resolvê-los rapidamente. Além disso, com o advento da computação em nuvem, é possível implementar automações baseadas nessas métricas, como a alocação dinâmica de recursos computacionais em resposta à demanda sobre a aplicação.

c) Traces: Os traces, ou rastreamentos, são registros que capturam o percurso de uma solicitação em um sistema distribuído. Eles consistem em uma série de spans, que representam operações individuais realizadas durante o processamento. Cada span inclui informações

> sobre o tempo de início e término da operação, além de metadados relevantes. A coleta de traces é essencial para identificar gargalos de desempenho e analisar a latência em sistemas complexos. Ao mapear a jornada de uma solicitação, é possível localizar atrasos e falhas, facilitando a resolução de problemas e a otimização do sistema.

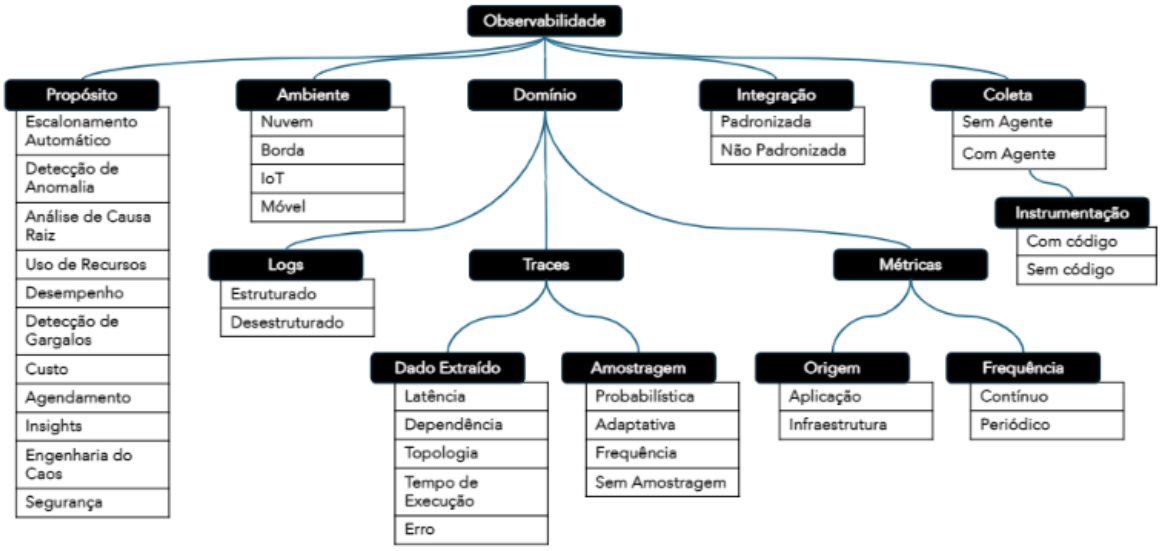
Esses três elementos — logs, métricas e traces — são interdependentes e, juntos, fornecem uma visão abrangente do estado e do comportamento de um sistema. A integração eficaz desses aspectos é fundamental para a implementação de práticas de observabilidade que garantam a estabilidade e a eficiência operacional em ambientes de software complexos.

3 DECISÕES DE DESIGN PARA MELHORA DA EFICÁCIA DA OBSERVABILIDADE EM SISTEMAS DISTRIBUÍDOS

3.1 Visão Geral

As possíveis decisões de design e arquitetura estão divididas em categorias, começando com uma visão geral das ferramentas disponíveis e, em seguida, focando nos três principais aspectos da observabilidade: logs, traces e monitoramento.

Figura 1 – Taxonomia de Observabilidade



Fonte: (GOMES; REGO; TRINTA, 2024)

Além disso, serão apresentadas informações sobre temas secundários que complementam ainda mais o estudo. Para a seleção dos temas abordados em cada ponto do estudo, foi utilizada a taxonomia proposta por (GOMES; REGO; TRINTA, 2024), conforme ilustrado na Figura 1. Essa abordagem permite uma organização sistemática e coerente das decisões de design, facilitando a compreensão das melhores práticas para a melhoria da eficácia da observabilidade em sistemas distribuídos.

3.2 Discussões

Nesta seção, serão apresentadas as discussões geradas para cada categoria abordada, com foco na aplicação dos conceitos dentro da observabilidade em sistemas distribuídos.

3.2.1 Ferramentas

Um erro comum identificado na pesquisa de (USMAN et al., 2022) ao selecionar uma ferramenta de observabilidade é a suposição de que ela será sempre suficiente. A realidade é que nenhuma ferramenta é capaz de monitorar todos os aspectos de um software; eventualmente, a ferramenta escolhida pode não atender mais às necessidades. Quando esse momento chega, a ideia geral é que é necessário trocar de ferramenta. No entanto, essa prática pode levar à mesma falácia, pois a segunda ferramenta também pode não ser suficiente para monitorar todos os aspectos do software.

Ao escolher uma ferramenta, é recomendável optar por aquelas que possuam protocolos de comunicação abertos. Essa abordagem oferece as seguintes vantagens:

a) Liberdade de Escolha: Não estar vinculado a uma solução ou ecossistema específico proporciona mais oportunidades para testar e adaptar ferramentas ao contexto em que se busca aplicar.
b) Custo-Benefício: Estar dentro de um ecossistema fechado pode limitar a escolha de soluções mais acessíveis, obrigando o usuário a se restringir ao que a solução oferece.

Por outro lado, permanecer em um ecossistema fechado pode facilitar o manuseio. O profissional atuante não precisará aprender a utilizar diversas ferramentas que funcionam de maneiras diferentes. Contudo, com o advento da inteligência artificial, um profissional generalista pode trabalhar eficientemente com várias ferramentas e alcançar a mesma eficácia que um especialista.

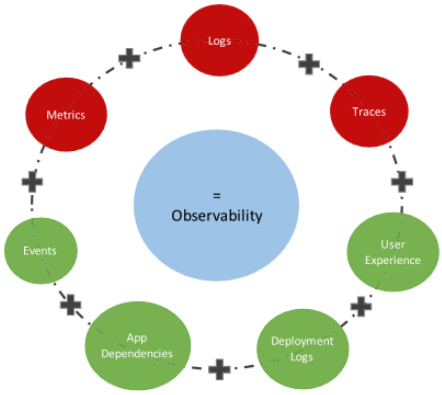
O artigo (CHAVA, 2022) apresenta uma lista de quatro itens-chave a serem considerados na escolha de uma ferramenta, sendo essas as principais funcionalidades que a ferramenta deve oferecer para garantir o sucesso no processo de observação:

a) Monitoração em Tempo Real: O rastreamento contínuo e rápido da performance e estabilidade do sistema é essencial para que as respostas a incidentes sejam detectadas o mais rapidamente possível.
b) Alertas: A capacidade de notificar sobre possíveis inconsistências no sistema é fundamental. Quanto maior a gama de opções de notificação, mais adaptável a solução será ao contexto em que é aplicada.
c) Dashboards Personalizáveis: A forma como as informações são exibidas e o nível de personalização disponível são cruciais. Diferentes ecossistemas de microsserviços requerem dashboards distintos, focando em diferentes aspectos, além de possivelmente mudar o foco de observação conforme o sistema evolui.
d) Escalabilidade: A escalabilidade garante que a ferramenta se expanda conforme o sistema monitorado também se expande, o que é importante para manter um suporte a longo prazo da ferramenta.

Como mencionado anteriormente, ao realizar a pesquisa de ferramentas, pode não ser possível encontrar uma que possua todas essas funcionalidades e que se encaixe perfeitamente no contexto em questão. No entanto, é interessante considerar a combinação de ferramentas para alcançar um alto nível de qualidade na observação. Como prática de mercado, é comum, por exemplo, utilizar uma ferramenta para coletar dados e outra para exibi-los.

3.2.2 Dados principais a serem coletados

Figura 2 – Os três pilares fundamentais da observabilidade (em vermelho) e outros tipos de dados considerados vitais para a observabilidade por diferentes fornecedores



Fonte: (BORGES et al., 2024)

Existem inúmeras informações que podem ser coletadas de um software (Figura 2). Atualmente, cada ferramenta de observabilidade oferece uma vasta gama de opções de monitoramento. No entanto, a coleta de dados pode ser onerosa, especialmente em um sistema de microsserviços, onde cada serviço deve ser monitorado individualmente. Além disso, a tentativa de observar todos os aspectos do sistema pode resultar na coleta de informações que não são necessariamente relevantes para o contexto, o que pode "sujar" a visualização dos dashboards com dados pouco significativos.

Esse tipo de problema é conhecido como "overhead", que ocorre quando a coleta excessiva de dados não agrega valor à observabilidade do sistema. O ideal é focar na coleta de dados que realmente geram valor, permitindo uma análise mais eficaz e uma visualização mais clara das informações relevantes para a operação e manutenção do software. Essa abordagem não apenas otimiza os recursos, mas também melhora a capacidade de resposta a incidentes e a tomada de decisões informadas.

> (...) é importante identificar precisamente *o que* queremos e precisamos monitorar (...) O essencial é identificar quais propriedades de um microsserviço são necessárias e suficiente para descrever seu comportamento e então determinar o que as mudanças nessas propriedades nos dizem sobre o status geral e a saúde do microsserviço. Chamaremos estas propriedades de *métricas principais*. (FOWLER, 2017)

Neste contexto de sobrecarga (*overheat*), alguns profissionais optam por trocar de ferramenta por uma alternativa mais econômica. No entanto, essa abordagem, além de ser dispendiosa, apenas resolve temporariamente a questão dos custos, pois, à medida que a aplicação escalar, o problema tende a se repetir.

Dessa forma, é fundamental avaliar quais são as principais métricas e concentrar esforços nelas. A literatura já delineia um caminho a ser seguido, que consiste em focar nos pilares da observabilidade de software: logs, métricas e traces. É essencial coletar, principalmente, dados relacionados ao servidor e à infraestrutura, os quais

são padrão em todo tipo de microsserviços, independentemente da linguagem de programação ou do sistema operacional utilizado para executar a aplicação.

Os principais dados de servidor e infraestrutura que devem ser coletados de um microsserviço incluem (FOWLER, 2017):

a) CPU utilizada
b) RAM utilizada
c) Threads disponíveis
d) Número de conexões com o banco de dados
e) Descritores de arquivos abertos

Além desses dados, é pertinente coletar informações específicas de cada aplicação, as quais dependerão do contexto, da linguagem utilizada e do tipo de aplicação desenvolvida. Supondo que se trate de um sistema de microsserviços que se comunicam via Hyper Text Transfer Protocol (HTTP), o livro também estabelece uma base de informações que devem ser coletadas:

a) Métricas específicas da linguagem
b) Disponibilidade
c) Acordo de Nível de Serviço (SLA)
d) Latência
e) Taxa de sucesso do *endpoint*
f) Resposta do *endpoint*
g) Tempo de resposta do *endpoint*
h) Cliente
i) Erros e exceções
j) Dependências

A combinação desses dados será suficiente para coletar e fornecer as principais métricas de ouro da Engenharia de Confiabilidade de Sites (SRE). Conforme descrito na referência, essas métricas são:

Latência: Refere-se ao tempo de resposta de uma requisição, sendo um indicador crucial da performance da aplicação e permitindo a identificação de possíveis gargalos. É fundamental estabelecer uma meta de performance, a qual pode variar de acordo com a aplicação e, em algumas situações, com o *endpoint* específico.

O monitoramento deve ser realizado para identificar os momentos em que esse tempo pré-estabelecido não é cumprido.

Tráfego (demanda de uso): Consiste no registro do número de solicitações realizadas na aplicação em um determinado intervalo de tempo. Através desse tipo de dado, é possível identificar os horários de pico de utilização da aplicação, permitindo a formulação de estratégias que garantam a estabilidade do sistema.

Erros: A coleta de dados sobre erros facilita a identificação de possíveis problemas na aplicação por parte dos desenvolvedores. É importante destacar que existem erros explícitos (como o retorno 500 HTTP) e erros implícitos (por exemplo, uma atualização que retorna 204, mas não realiza a alteração no banco de dados). Com as informações sugeridas para coleta, é viável rastrear ambos os tipos de erro.

Saturação: Refere-se às métricas de servidor e infraestrutura, indicando como a aplicação está sendo executada e quanto de recursos está consumindo. Com essas informações, é possível desenvolver estratégias que aumentem automaticamente os recursos da máquina quando uma quantidade específica de recursos computacionais estiver sendo utilizada.

3.2.3 Evitar redundância na coleta de dados

Inevitavelmente, em alguns momentos, dois tipos de dados coletados podem fornecer a mesma informação. Por exemplo, a informação de que "o sistema está fora do ar" pode ser evidenciada tanto pelo consumo de RAM zerado quanto pelo consumo de CPU também zerado. Embora essas sejam duas métricas distintas, ambas indicam a mesma condição: "o sistema está fora do ar".

Até certo ponto, é normal e aceitável que haja essa redundância. No entanto, é interessante mapear essas situações para identificar momentos em que é possível descontinuar a coleta de determinados dados, uma vez que outra coleta já fornece a mesma informação.

O artigo de (BORGES et al., 2024) propõe uma tabela que permite visualizar esse tipo de redundância, facilitando a gestão e a otimização da coleta de dados.

Tabela 1 - Métricas de visibilidade de falhas para a configuração de observabilidade padrão

Falhas	Utiliz. CPU	Duraç. Trace	Req. Per minute	FC
Pausa	1	1	1	3/3
Perda de pacote (15%)	1	0	0	1/3
Delay de rede (0 - 90ms)	0	0	0	0/3
				OFO = 2/3

Fonte: (BORGES et al., 2024)

A Tabela 1 apresenta, à esquerda, as informações a serem obtidas sobre o software, enquanto as outras três colunas correspondem às métricas estabelecidas: utilização de CPU, duração de trace e requisições por minuto. Os números 1 e 0 indicam se uma determinada métrica observa uma falha específica, sendo 1 para "observado" e 0 para "não observado". No caso, a pausa da aplicação é observada pelas três métricas propostas, enquanto a perda de pacotes é observada apenas pela utilização de CPU.

À direita, o termo Fault Coverage (FC) representa o coeficiente de percepção da falha. A pausa apresenta uma cobertura de 3/3, o que significa que todas as três métricas observam a interrupção do sistema. Por outro lado, o delay de rede apresenta uma cobertura de 0/3, indicando que nenhuma das métricas observa essa falha.

Overall Fault Observability (OFO) refere-se à quantidade de falhas observadas no contexto geral da observabilidade. Neste caso, a pausa e a perda de pacotes são observadas, enquanto o delay de rede não é, resultando em uma cobertura de 2/3 das falhas.

Com esse tipo de análise, é possível realizar trade-offs, optando por deixar de monitorar uma informação em favor de outra. Por exemplo, pode-se descontinuar o monitoramento da duração do trace e passar a monitorar uma métrica que identifique o delay de rede. Dessa forma, todas as 3/3 falhas seriam cobertas, melhorando a cobertura geral e a eficácia da observação.

3.2.4 Pilares da observabilidade

Conforme mencionado anteriormente, os três pilares da observabilidade são logs, traces e métricas. Embora esses elementos se inter-relacionem e se complementem, cada um deles possui suas próprias considerações de design. A seguir, serão abordadas maneiras de pensar sobre a coleta desses dados, com foco na eficácia.

3.2.4.1 Logs

Segundo (GOMES; REGO; TRINTA, 2024), podemos separar os logs em dois grandes grupos, estruturados e desestruturados:

a) Desestruturado: informações sem estrutura ou padrão, apenas grandes *strings* de texto sem formatação.
b) Estruturados: as informações são enviadas numa estrutura definida, geralmente um objeto com as informações separadas nas propriedades.

Logs desestruturadas são mais fáceis de implementar pois requerem pouca configuração prévia, porém dificultam a analisa de possíveis falhas. A busca fica dificultada pela falta de categorização e isso pode aumentar o tempo de resposta a uma anomalia no software.

Por outro lado, os logs estruturados são mais difíceis de implementar quando comparado a logs desestruturados, entretanto, facilita em muito a atuação no caso de anomalias. O tempo de resposta reduz bastante já estando estruturado, a pesquisa e a localização ficam facilitadas, basta o profissional filtrar pela propriedade que fornece a informação que ele procura. O mesmo caso com dados desestruturados, teria que haver uma etapa de conversão para identificar as informações ou para fazer algum tipo de query dinâmica na base de logs.

Além do formato da informação, é necessário definir um tempo de armazenamento dos logs e isso vai depender da quantidade logs gerados e o nível de importância do sistema. Armazenar logs gera um custo considerável além disso, quanto mais logs um sistema possui, mais lento será o tempo de execução das queries de busca.

Para contornar esse problema de armazenamento de logs há algumas estratégias.

a) Coleta periódica: A coleta de logs pode ser periódica, uma proporção deve ser definida como “duas a cada cinco requisições serão salvas”. Isto diminui a quantidade de informações armazenadas, porém pode se perder o rastreio de uma requisição. Neste mesmo caso, se uma requisição gera 3 tipos de logs, mas apenas duas de cinco serão salvas, então pode acontecer casos em que há o primeiro e o segundo log, mas o terceiro não foi salvo.
b) Definir um tempo de retenção de logs: Um log coletado pode ter uma data de expiração de quando ele será apagado ou transferido para uma base secundaria. Assim é possível garantir uma base menor com dados recentes que responde rapidamente a queries de pesquisa e caso necessário haverá uma segunda base mais lenta com todos os logs históricos. Essa estratégia garante também um custo menor no armazenamento de logs, em algumas ferramentas de observabilidade, esse custo pode ser bem alto.

3.2.4.2 Traces

Os traces são fundamentais para compreender todo o ciclo de funcionamento do sistema, especialmente em ambientes de sistemas distribuídos. A capacidade de rastrear uma requisição entre os diversos microsserviços proporciona uma compreensão mais abrangente do funcionamento total do sistema. No entanto, assim como nos logs, algumas técnicas podem ser aplicadas para reduzir a carga de coleta de dados, como a amostragem, conforme relatado no artigo de (GOMES; REGO; TRINTA, 2024).

A amostragem de traces diminui a frequência de coleta, e algumas estratégias comuns de amostragem incluem:

Frequência: Define-se uma determinada frequência de coleta baseada em uma razão, como 1/3 (um trace coletado a cada três requisições) ou 1/8 (um trace coletado a cada oito requisições).

Probabilística: Diferente da abordagem de frequência, a amostragem probabilística baseia-se na definição de uma porcentagem específica de chance de

um trace ser coletado, tornando o processo mais aleatório. Por exemplo, pode-se definir que cada requisição tem 30% de chance de ser coletada.

Adaptativa: Essa abordagem coleta a amostra dependendo de determinadas condições. Por exemplo, pode-se definir que serão coletados traces apenas quando um determinado *endpoint* HTTP retornar uma quantidade específica de erros.

Temporal: Rastreia requisições em intervalos de tempo específicos, como durante picos de tráfego ou em horários de manutenção, para obter uma visão mais clara do desempenho em momentos críticos.

3.2.4.3 Métricas

As métricas são essenciais para monitorar o funcionamento do software, coletando tanto dados de infraestrutura quanto dados específicos do próprio software. Uma parte significativa dos dados coletados é crucial para compreender o desempenho do sistema, sendo ideal que essas informações sejam disponibilizadas o mais rapidamente possível, preferêncialmente em tempo real.

No entanto, algumas dessas métricas podem ser coletadas em lote durante horários em que a aplicação está menos utilizada. Dessa forma, é possível reservar o processamento de coleta para momentos de menor tráfego, quando há maior disponibilidade de recursos. Assim, podemos realizar uma:

a) Coleta Contínua: Nesse modelo, as informações são coletadas em tempo real de forma contínua. Essa abordagem é preferível para dados mais críticos.
b) Coleta Periódica: Nesta estratégia, a aplicação realiza a coleta de algumas métricas secundárias em intervalos de tempo determinados, otimizando assim o uso dos recursos computacionais da máquina.

As métricas também desempenham um papel fundamental no tempo de resposta a anomalias, uma vez que as configurações de notificação devem ser baseadas nelas. Por conta disso, a coleta periódica deve ser utilizada apenas para informações que realmente possuem menor importância. É necessária cautela ao adotar essa estratégia.

3.2.5 Coleta centralizada

Independentemente do tipo de dado coletado, é fundamental dispor de um sistema centralizador, um local onde todas as informações possam ser observadas e correlacionadas, permitindo também que se forneça o devido contexto aos dados.

A utilização de ferramentas de observação que possam receber dados de diversas fontes é extremamente importante nesse contexto. O padrão é que os ambientes de software incluam tanto aplicações legadas quanto novas aplicações. As aplicações legadas podem necessitar de ferramentas de monitoramento específicas, e um coletor de dados que apresente maior diversidade de métodos de comunicação facilita a centralização das informações.

Para realizar essa coleta, existem duas abordagens, com agente e sem agente (GOMES; REGO; TRINTA, 2024):

a) Com Agente: Nesta abordagem, um agente intermediário é posicionado entre o sistema e o coletor, realizando a tradução das informações para que possam ser processadas adequadamente pelo coletor.
b) Sem Agente: Os dados são enviados diretamente ao coletor, mas isso requer um esforço para padronizar cada serviço, garantindo que os dados sejam fornecidos corretamente ao coletor.

3.2.6 Dados extras a serem coletados

Há uma ampla gama de informações que podem ser coletadas. Em determinado momento, o ambiente pode estar estruturado e maduro o suficiente para não apenas coletar as informações essenciais, mas também outros dados que podem agregar valor e contribuir para o processo de evolução do software. Entre esses dados, destacam-se:

a) Experiência do Usuário: Ferramentas que observam e armazenam as interações dos usuários com o sistema são importantes para identificar possíveis melhorias na usabilidade.
b) Logs de Deploy: É interessante monitorar não apenas o software em execução, mas também o processo de deploy. Com esse tipo de informação, é possível aprimorar o processo de implantação, reduzindo os empecilhos e problemas durante as atualizações para produção.

c) Monitoramento de Dependências do Projeto: A vigilância sobre as dependências do projeto protege a aplicação de possíveis falhas de execução ou de segurança decorrentes da atualização de bibliotecas.
d) Feedback do Usuário: Sistemas podem coletar informações fornecidas diretamente pelos usuários. Essa comunicação direta pode ser crucial para descobrir erros implícitos que podem não ser identificados por meio da monitoração de métricas.

4 CONCLUSÃO E TRABALHOS FUTUROS

Ao longo da última década, com a implementação em massa do SaaS e da cultura DEVOPS, o software tornou-se mais complexo, com ciclos de atualização mais rápidos e uma necessidade crescente de monitoramento constante e resposta ágil a problemas emergentes. Como resultado, a observabilidade tornou-se padrão nos ambientes de desenvolvimento de software.

Neste novo momento, é essencial que os profissionais responsáveis pela arquitetura de ambientes de observabilidade possuam um conhecimento sólido sobre as diferentes abordagens de coleta de dados e seus trade-offs. Decisões de design tomadas sem esse entendimento podem atrasar a adoção da observabilidade, resultando em custos excessivos ou em uma coleta de dados irrelevante.

Esta monografia apresentou as principais técnicas e abordagens de monitoramento e observação de dados, oferecendo uma visão abrangente sobre como configurar um ambiente de observabilidade eficaz. Com esse conhecimento, é possível adaptar soluções conforme o contexto, seja em ambientes mais permissivos ou em cenários mais restritos.

Foram discutidas as decisões de design mais relevantes, incluindo a escolha de ferramentas, a seleção de dados a serem rastreados e as três fontes principais de dados em um ambiente observado: logs, métricas e traces. A análise dessas questões revelou que a escolha adequada das ferramentas e a definição clara das métricas a serem monitoradas são cruciais para o sucesso da observabilidade.

Embora este estudo tenha abordado questões de alto nível, ele serve como um ponto de partida para futuras investigações. Trabalhos futuros podem aprofundar-se em temas específicos, realizar benchmarks ou explorar a adoção de técnicas de observabilidade no mercado, contribuindo assim para o avanço contínuo dessa área.

5 REFERÊNCIAS

BROWN, Kyle. **Beyond buzzwords: A brief history of microservices patterns**. IBM Developer. 10 de outubro de 2024 https://developer.ibm.com/articles/cl-evolution-microservices-patterns/

PUPPET**.** ***The State of DevOps Report: The Evolution of Platform Engineering.*** **Puppet**, 2024. Disponível em: https://www.puppet.com/resources/state-of-platform-engineering.

AWS. **What are Microservices?** Amazon Web Services, [s.d.]. Disponível em: https://aws.amazon.com/microservices/.

GOMES, Francisco A. A.; REGO, Paulo A. L.; TRINTA, Fernando A. M. **Rumo a uma Taxonomia de Observabilidade para Aplicações Baseadas em Microsserviços**. Universidade Federal do Ceará, Ceará, 2024.

CHAVA, Ajay. **Enhancing Software Performance and Reliability through Observability in DevOps**. International Journal of Science and Research (IJSR). 2022. Disponível em: https://www.ijsr.net/archive/v13i10/MS241012104833.pdf.

BORGES, Maria C.; BAUER, Joshua; WERNER, Sebastian; GEBAUER, Michael; TAI, Stefan. **Informed and Assessable Observability Design Decisions in Cloud-native Microservice Applications**. Preprint, compilado em 16 jul. 2024. Information Systems Engineering, Technische Universität Berlin, Alemanha.

USMAN, Muhammad; FERLIN, Simone; BRUNSTROM, Anna; TAHERI, Javid. **A Survey on Observability of Distributed Edge & Container-Based Microservices**. IEEE Access, 15 jun. 2022, aceito em 15 jul. 2022, publicado em 21 jul. 2022, versão atual em 24 ago. 2022. DOI: 10.1109/ACCESS.2022.3193102. Department of Computer Science, Karlstad University, Karlstad, Suécia; Red Hat, Estocolmo, Suécia. Disponível em: https://ieeexplore.ieee.org/document/9837035.

FOWLER, Susan J. **Microserviços prontos para a produção**. Capítulo: Monitoramento de microserviços. Novatec Editora, 5 out. 2017.

AWS. **O que é SaaS (software como serviço)?** Amazon Web Services. Disponível em: https://aws.amazon.com/pt/what-is/saas/.

BUCHANAN, Ian. **History of DevOps: How development and operations teams came together to solve dysfunction in the industry**. Atlassian. Disponível em: https://www.atlassian.com/devops/what-is-devops/history-of-devops.

AKITA, Fabio**. [Akitando] #70 - Entendendo GIT | (não é um tutorial!)**. 5 fev. 2020. Disponível em: https://www.akitaonrails.com/2020/02/05/akitando-70-entendendo-git-nao-e-um-tutorial.

RED HAT. **What is Observability?** 21 dez. 2023. Disponível em: https://www.redhat.com/en/topics/devops/what-is-observability.

IBM EDUCATION. **Observability vs. monitoring: What's the difference**? 29 ago. 2022. Disponível em: https://www.ibm.com/think/topics/observability-vs-monitoring.

ISAIAH, Ayooluwa. **Why Structured Logging is Fundamental to Observability**. BetterStack, atualizado em 16 out. 2024. Disponível em: https://betterstack.com/community/guides/logging/structured-logging

BETTERSTACK TEAM. **7 Steps to Reducing Your Logging Costs: Best Practices**. BetterStack, atualizado em 15 jan. 2024. Disponível em: https://betterstack.com/community/guides/logging/reduce-logging-costs/.

FOWLER, Martin. **Microservices Guide**. Martin Fowler, 21 ago. 2019. Disponível em: https://martinfowler.com/microservices/.

TEBALDI, Pedro César. **Qual a diferença entre monitoramento e observabilidade?** 20 nov. 2020. Disponível em: https://www.opservices.com.br/diferenca-entre-monitoramento-e-observabilidade/.