UNIVERSIDADE FEDERAL DO CEARÁ

CENTRO DE CIÊNCIAS (CC)

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

DOUTORADO EM CIÊNCIA DA COMPUTAÇÃO

FRANCISCO ANDERSON DE ALMADA GOMES

UMA ABORDAGEM ESTRUTURADA PARA OBSERVABILIDADE EM MICROSSERVIÇOS: TAXONOMIA, CATÁLOGO E *FRAMEWORK* DE DETECÇÃO DE ANTI-PADRÕES

FORTALEZA

2026

FRANCISCO ANDERSON DE ALMADA GOMES

UMA ABORDAGEM ESTRUTURADA PARA OBSERVABILIDADE EM MICROSSERVIÇOS: TAXONOMIA, CATÁLOGO E *FRAMEWORK* DE DETECÇÃO DE ANTI-PADRÕES

Tese apresentada ao Programa de Pós-Graduação em Ciência da Computação do Centro de Ciências (CC) da Universidade Federal do Ceará, como requisito parcial à obtenção do título de doutor em Ciência da Computação. Área de Concentração: Engenharia de Software

Orientador: Dr. Fernando Antonio Mota Trinta

Coorientador: Dr. Paulo Antônio Leal Rego

FORTALEZA

2026

FRANCISCO ANDERSON DE ALMADA GOMES

UMA ABORDAGEM ESTRUTURADA PARA OBSERVABILIDADE EM MICROSSERVIÇOS: TAXONOMIA, CATÁLOGO E *FRAMEWORK* DE DETECÇÃO DE ANTI-PADRÕES

Tese apresentada ao Programa de Pós-Graduação em Ciência da Computação do Centro de Ciências (CC) da Universidade Federal do Ceará, como requisito parcial à obtenção do título de doutor em Ciência da Computação. Área de Concentração: Engenharia de Software

Aprovada em: 30 de Abril de 2026

BANCA EXAMINADORA

Dr. Fernando Antonio Mota Trinta (Orientador)
Universidade Federal do Ceará (UFC)

Dr. Paulo Antônio Leal Rego (Coorientador)
Universidade Federal do Ceará (UFC)

Dr. Lincoln Souza Rocha
Universidade Federal do Ceará (UFC)

Dr. Nabor das Chagas Mendonça
Universidade de Fortaleza (Unifor)

Dra. Aletéia Patrícia Favacho de Araújo von Paumgartten
Universidade de Brasília (UnB)

Dedico este trabalho a todos que, de alguma forma, contribuíram para minha jornada, com apoio, ensinamentos e incentivo.

AGRADECIMENTOS

Primeiramente, agradeço a Deus por todas as bênçãos concedidas ao longo da minha vida.

À minha filha, Ana Yasmin, por ser o maior presente da minha vida.

À minha esposa, Vitória Regina, pelo amor incondicional e pelo constante incentivo.

Aos meus pais, Alexandre e Jacqueline, pelo carinho, apoio incondicional e por sempre acreditarem em mim.

Ao meu irmão e à minha cunhada, Alex e Rita, pelo suporte e pela motivação ao longo desta jornada.

Aos meus orientadores, professores Fernando Trinta e Paulo Rego, pela orientação, paciência e dedicação durante o desenvolvimento deste trabalho.

Aos professores Lincoln, Nabor e Aletéia, membros da banca examinadora, pela disponibilidade e pelas valiosas contribuições para o aprimoramento desta pesquisa.

À Universidade Federal do Ceará, pela oportunidade de realização do doutorado.

Ao Campus de Crateús da Universidade Federal do Ceará, pelo apoio institucional e pela liberação para cursar o doutorado.

Aos amigos e colegas que compartilharam esta caminhada, pelas trocas de experiências e pelo apoio nos momentos desafiadores.

A todos que, de alguma forma, contribuíram para minha formação, expresso o meu mais sincero agradecimento.

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

“O sucesso é a soma de pequenos esforços repetidos dia após dia.” (Robert Collier)

RESUMO

Sistemas de software dependem cada vez mais de arquiteturas baseadas em microsserviços para aprimorar escalabilidade, modularidade e implantação contínua. Embora essa abordagem simplifique o desenvolvimento ao promover uma divisão funcional entre componentes, ela também introduz uma complexidade operacional significativa, tornando falhas mais frequentes e difíceis de diagnosticar. Nesse contexto, a observabilidade surge como um conceito fundamental, definido como a capacidade de compreender e diagnosticar o comportamento interno de um sistema a partir de seus estados externos, como métricas, *logs* e *traces*. No entanto, apesar de sua importância, a observabilidade é frequentemente mal implementada devido à ausência de práticas padronizadas, o que resulta em monitoramento ineficaz, fadiga de alertas e baixa eficiência na resposta a incidentes. Embora existam estudos que discutem conceitos, ferramentas e desafios relacionados à observabilidade, nenhum trabalho anterior se concentrou especificamente nos anti-padrões de observabilidade, práticas recorrentes que comprometem a eficácia do monitoramento, tampouco há soluções capazes de detectá-los automaticamente. Além disso, observa-se a ausência de uma taxonomia abrangente que permita classificar e organizar os estudos existentes sobre observabilidade. Diante disso, esta tese apresenta uma taxonomia voltada à observabilidade em aplicações baseadas em microsserviços, construída a partir de um mapeamento sistemático da literatura. Assim, foram analisados 84 estudos relevantes publicados entre 2019 e 2025, oferecendo uma visão abrangente sobre o campo. A revisão também identifica ferramentas, aplicações de *benchmarking* e conjuntos de dados reais utilizados nos estudos selecionados. Complementando essa contribuição, o trabalho desenvolveu um catálogo sistematizado de anti-padrões de observabilidade, oferecendo uma abordagem para identificar e mitigar práticas prejudiciais. Este catálogo serve como um guia prático para apoiar equipes na construção de sistemas mais confiáveis e eficientes. Ao todo, foram identificados 37 anti-padrões, cuja relevância foi avaliada por 60 especialistas, com um índice de concordância de 95%. Por fim, a última contribuição consiste no desenvolvimento do *framework* Observa, projetado para detectar automaticamente anti-padrões de observabilidade. O seu funcionamento foi avaliado por meio de três experimentos, além de uma prova de conceito, que demonstraram sua viabilidade técnica e aprovaram sua adoção, com recomendação de excelência pelos avaliadores.

Palavras-chave: monitoramento; observabilidade; microsserviços; taxonomia; catálogo; anti-padrão.

ABSTRACT

Software systems increasingly rely on microservices-based architectures to enhance scalability, modularity, and continuous deployment. Although this approach simplifies development by promoting a functional decomposition of components, it also introduces significant operational complexity, making failures more frequent and harder to diagnose. In this context, observability emerges as a fundamental concept, defined as the ability to understand and diagnose the internal behavior of a system based on its external outputs, such as metrics, logs, and traces. However, despite its importance, observability is often poorly implemented due to the lack of standardized practices, resulting in ineffective monitoring, alert fatigue, and low efficiency in incident response. Although existing studies discuss concepts, tools, and challenges related to observability, no prior work has focused specifically on observability anti-patterns, recurrent practices that undermine monitoring effectiveness, nor proposed solutions capable of detecting them automatically. Furthermore, there is a lack of a comprehensive taxonomy to classify and organize existing studies on observability. To address these gaps, this thesis presents a taxonomy focused on observability in microservices-based applications, constructed through a systematic mapping of the literature. A total of 84 relevant studies published between 2019 and 2025 were analyzed, providing a comprehensive overview of the field. The review also identifies tools, benchmarking applications, and real-world datasets used in the selected studies. Complementing this contribution, the thesis develops a systematized catalog of observability anti-patterns, offering an approach to identify and mitigate harmful practices. This catalog serves as a practical guide to support teams in building more reliable and efficient systems. In total, 37 anti-patterns were identified, whose relevance was evaluated by 60 experts, achieving an agreement rate of 95%. Finally, the thesis introduces the Observa framework, designed to automatically detect observability anti-patterns. Its operation was evaluated through three experiments and a proof of concept, which demonstrated its technical feasibility and approved its adoption, receiving a recommendation of excellence from the evaluators.

Keywords: monitoring; observability; microservices; taxonomy; catalog; anti-pattern

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE CÓDIGOS-FONTE

LISTA DE ABREVIATURAS E SIGLAS

AP	Anti-Padrão
API	*Application Programming Interface*
CNA	*Cloud-Native Application*
CNCF	*Cloud Native Computing Foundation*
HTTP	*Hypertext Transfer Protocol*
MLC	Mapeamento da Literatura Cinzenta
MSL	Mapeamento Sistemático da Literatura
OTel	*OpenTelemetry*
QoS	*Quality of Service*
REST	*Representational State Transfer*
SLA	*Service Level Agreement*

SUMÁRIO

1 INTRODUÇÃO

1.1 Contextualização

Nos últimos anos, os sistemas de software aumentaram significativamente em escala e complexidade, tornando cada vez mais desafiador a incorporação de novas funcionalidades. A adoção da arquitetura de microsserviços emergiu como uma solução central para esse problema (THÖNES, 2015), e ganhou ampla aceitação por mitigar os desafios de grandes bases de código monolíticas e permitir a escalabilidade das aplicações de forma mais transparente (DRAGONI *et al.*, 2017). Em virtude dessas vantagens, um número crescente de empresas tem migrado para arquiteturas baseadas em microsserviços, reconhecendo seu potencial para melhorar o desempenho, reduzir o esforço humano em tarefas de desenvolvimento e, consequentemente, diminuir os custos financeiros associados (SINGLETON, 2016).

Apesar de suas contribuições positivas, os microsserviços também introduzem novos desafios (SINGLETON, 2016; FRANCESCO *et al.*, 2017). À medida que os sistemas se tornam mais distribuídos, dinâmicos e imprevisíveis, aumenta o número de possíveis pontos de falha e áreas sujeitas à degradação de desempenho. O diagnóstico torna-se especialmente complexo quando aplicações nativas da nuvem são compostas por centenas de microsserviços, cada um executando múltiplas instâncias em paralelo (DRAGONI *et al.*, 2017).

Nesse contexto, soluções tradicionais de monitoramento revelam-se insuficientes, pois não oferecem visibilidade completa e tampouco identificam proativamente desvios em relação ao comportamento esperado, o que pode levar a interrupções nos serviços. A necessidade de medidas preventivas e capacidades avançadas de detecção impõe um novo conjunto de requisitos para o monitoramento de sistemas modernos, exigindo um nível mais elevado de visibilidade, conhecido como observabilidade (KOSIŃSKA *et al.*, 2023).

Observabilidade é um termo originalmente definido no contexto de sistemas lineares, como a capacidade de inferir o estado interno de um sistema complexo a partir de suas saídas externas (KALMAN, 1959). Mais recentemente, passou a ser utilizado também no domínio de sistemas computacionais, consistindo de um conjunto de práticas em expansão, combinadas com ferramentas que vão além do mero monitoramento, proporcionando observações sobre o estado interno de um sistema por meio da análise de suas saídas externas, predominantemente conhecidas como dados de telemetria, que são as métricas, *logs* e *traces*.

A observabilidade também se refere à capacidade de um sistema fornecer dados

que possibilitem a análise contínua de seu comportamento ao longo de todo o ciclo de vida do desenvolvimento (KARUMURI *et al.*, 2021). Quando bem implementada, ela proporciona uma visão aprofundada do fluxo de execução do sistema, facilitando atividades como depuração, identificação de comportamentos inesperados, detecção de gargalos e ineficiências, além de contribuir para a melhoria dos tempos de resposta e para o uso mais eficiente dos recursos computacionais.

1.2 Motivação

Apesar dos benefícios proporcionados pela observabilidade, muitas equipes acabam adotando anti-padrões (APs) ao implementá-la (KOSIŃSKA *et al.*, 2023; LI *et al.*, 2022; CHEVRE, 2024). Esses APs consistem em práticas que, embora inicialmente pareçam vantajosas, comprometem a eficiência da observabilidade e podem resultar em interpretações equivocadas, ineficiências operacionais e atrasos na identificação e resolução de falhas. Um exemplo recorrente ocorre quando o sistema de monitoramento gera um volume excessivo de alertas, frequentemente decorrente de regras redundantes, ausência de mecanismos de agregação ou notificações excessivamente granulares (TOWNSHEND, 2023). Esse cenário pode levar à chamada fadiga de alertas, em que os engenheiros passam a ignorar os avisos, inclusive os que sinalizam falhas críticas, devido à sobrecarga de notificações irrelevantes.

Diversos estudos têm se dedicado à análise de soluções de observabilidade. Niedermaier *et al.* (2019) realizaram um estudo empírico com base em entrevistas na indústria, com o objetivo de compreender os principais desafios e boas práticas relacionados à observabilidade e ao monitoramento de sistemas distribuídos. Li *et al.* (2022) conduziram uma investigação sobre observabilidade em arquiteturas de microsserviços, destacando a complexidade dos ambientes industriais em nuvem, nos quais instâncias de serviço são criadas e destruídas dinamicamente. Usman *et al.* (2022) também realizaram uma revisão abrangente sobre ambientes de TI distribuídos, focando nos desafios, requisitos, boas práticas e soluções voltadas à observação de sistemas modernos baseados em microsserviços. Complementando esses estudos, Kosińska *et al.* (2023) apresentaram um mapeamento sistemático sobre observabilidade em aplicações nativas da nuvem, discutindo abordagens de engenharia, níveis de maturidade, eficiência, ferramentas disponíveis e lacunas de pesquisa.

No entanto, até onde se tem conhecimento, não há estudos anteriores que proponham uma taxonomia voltada à classificação de pesquisas sobre observabilidade, considerando aspectos

como propósito, integração, coleta de dados e estratégias de instrumentação. Ademais, não foram identificados trabalhos que realizem o mapeamento sistemático de práticas inadequadas no uso da observabilidade, representadas por APs recorrentes enfrentados por profissionais da área. Tampouco foram encontradas propostas que permitam a identificação automática desses APs em sistemas computacionais.

1.3 Questões de Pesquisa

O impacto dos APs de observabilidade em aplicações baseadas em microsserviços é o tema central deste trabalho de doutorado. Os estudos aqui descritos são orientados à um estudo prático sobre o impacto que más práticas trazem à este tipo de aplicação, e soluções para mitigar tal impacto. De modo a estabelecer um ponto de partida, foi elaborada uma questão de pesquisa que busca orientar o trabalho de investigação realizado. Esta pergunta é apresentada a seguir:

> **QPG:** ***Como estruturar, sistematizar e operacionalizar o conhecimento sobre observabilidade em aplicações baseadas em microsserviços de modo a apoiar a identificação e mitigação de práticas inadequadas?***

Esta pergunta é ampla e impõe refinamentos. Estes refinamentos podem ser materializados por questões mais específicas sobre (i) a sistematização sobre o conhecimento a respeito de observabilidade e (ii) o apoio à identificação de práticas inadequadas em aplicações baseadas em microsserviços, especialmente, os chamados anti-padrões.

Em relação à organização sobre conhecimento, um novo conjunto de perguntas pode então ser formulada, conforme quadro a seguir:

> **QP1:** Quais são os diferentes domínios e categorias de observabilidade em microsserviços?
> **QP2:** Quais são as diferentes ferramentas, aplicações *benchmarking* e conjunto de dados relacionados à observabilidade em microsserviços?

Estas primeiras questões relacionam-se com a compreensão e a análise do campo de estudo principal deste trabalho. A partir das respostas à estas questões, surgiram às principais oportunidades de pesquisa e contribuições a serem alcançadas.

Já em relação à identificação das más práticas, surgem as seguintes questões:

QP3: Quais são os APs de observabilidade mais comuns?
QP4: Como os APs de observabilidade identificados podem ser mitigados ou evitados por meio de boas práticas?
QP5: É possível propor uma solução para automatizar o processo de detecção de APs que seja extensível e de fácil utilização por desenvolvedores e operadores de aplicações baseadas em microsserviços?

Este segundo grupo é consequência do escopo mais reduzido de investigação do trabalho. Uma vez definido o foco para as más práticas, as respostas aqui buscam agrupar APs de acordo com frequência de ocorrência, impacto e viabilidade de detecção automática. De certo modo, as respostas às perguntas guiam os objetivos e principais contribuições desta tese.

1.4 Objetivos e Contribuições

Este trabalho tem três objetivos principais: (i) conduzir uma revisão abrangente sobre observabilidade em aplicações baseadas em microsserviços, com o objetivo de desenvolver uma taxonomia de classificação que sirva como estrutura comum para posicionamento e análise dos estudos da área; (ii) mapear práticas inadequadas no uso da observabilidade por meio da elaboração de um catálogo estruturado, com o intuito de documentar os principais problemas enfrentados por profissionais no uso cotidiano de mecanismos de observabilidade; e (iii) desenvolver um *framework* para ajudar as equipes de observabilidade para detectar automaticamente os anti-padrões de observabilidade. Assim, as principais contribuições deste trabalho são:

- Uma taxonomia sobre observabilidade em aplicações baseadas em microsserviços, contemplando as principais categorias do tema e possibilitando a classificação sistemática dos estudos identificados;
- Uma visão geral das ferramentas utilizadas, aplicações *benchmarking* e conjunto de dados de *workload* do mundo real presentes nos estudos analisados;
- A documentação de más práticas de observabilidade por meio de um catálogo estruturado e sistematizado, contendo 37 APs que comprometem a efetividade das estratégias de observabilidade;
- A implementação de um portal de acesso aberto que disponibiliza todas as informações sobre os APs identificados, bem como a validação do catálogo por especialistas da área, a fim de garantir a relevância do que foi desenvolvido; e

- O projeto e a implementação de um *framework*, intitulado Observa, para facilitar a detecção automática de anti-padrões de observabilidade.

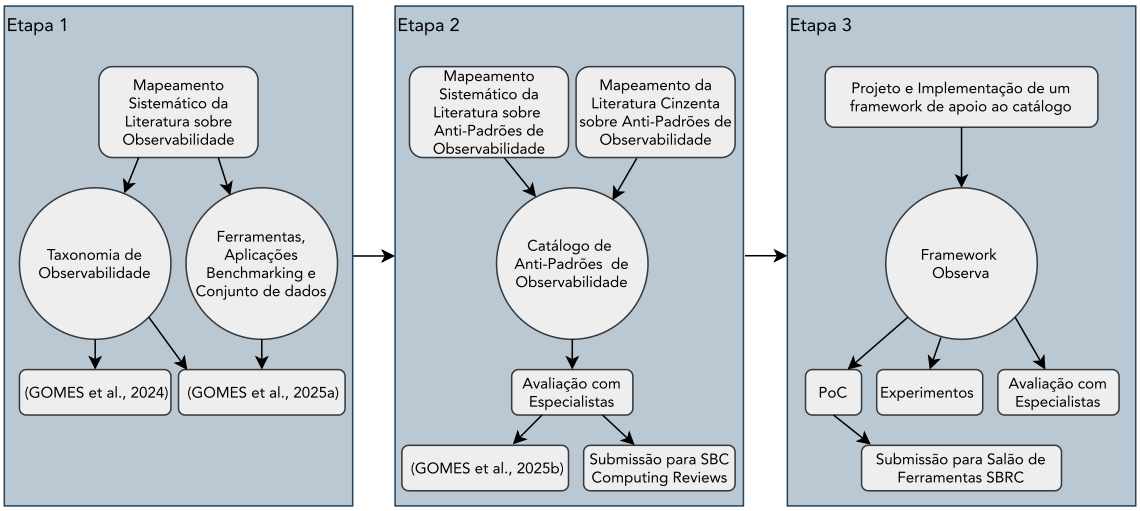
Esta pesquisa delimita-se ao estudo da observabilidade em aplicações baseadas em microsserviços. Não são abordados outros estilos arquiteturais, como sistemas monolíticos ou *serverless*. O catálogo desenvolvido contempla apenas os APs identificados na literatura, não pretendendo esgotar todas as possibilidades existentes.

1.5 Metodologia da Pesquisa

A metodologia adotada nesta tese caracteriza-se como aplicada, de natureza exploratória e explicativa, fundamentada em uma abordagem de métodos mistos, combinando técnicas qualitativas e quantitativas. Essa escolha decorre do próprio objetivo central do trabalho, que não se limita à compreensão teórica do fenômeno da observabilidade em arquiteturas de microsserviços, mas visa também à proposição, implementação e avaliação de artefatos concretos capazes de apoiar a prática profissional e a pesquisa científica na área.

Para isso, o desenvolvimento da tese foi realizada em etapas alinhadas aos objetivos descritos na seção anterior: (*i*) a construção da taxonomia de observabilidade; (*ii*) a elaboração de um catálogo de APs; e (*iii*) o desenvolvimento do *framework*. A Figura 1 ilustra as atividades específicas associadas a cada uma dessas etapas, detalhando os métodos empregados, os objetivos de cada fase do processo e as respectivas publicações realizadas.

Figura 1 – Metodologia de pesquisa.



Fonte: Autoria própria.

1.5.1 Taxonomia de Observabilidade

Como mencionado anteriormente, foi realizada uma busca inicial na literatura por estudos sobre observabilidade. A partir dessa busca, verifou-se a ausência de uma taxonomia capaz de classificar estudos de observabilidade em aplicações baseadas em microsserviços.

Assim, na Etapa 1, foi conduzido um Mapeamento Sistemático da Literatura (MSL), de acordo com diretrizes consolidadas para revisões na Engenharia de Software (PETERSEN *et al.*, 2008), com o objetivo de identificar, organizar e analisar estudos primários que abordassem aspectos da observabilidade em sistemas distribuídos, com ênfase em arquiteturas de microsserviços. Foram definidos critérios de inclusão e exclusão, termos de busca, *string* de consulta, bases de dados e protocolos de seleção, assegurando a qualidade e a relevância dos estudos analisados.

A partir da análise dos trabalhos selecionados, foram extraídas categorias conceituais e operacionais, organizadas em um esquema de classificação hierárquico para a construção da **taxonomia de observabilidade**. Essa estrutura foi concebida com o propósito de oferecer uma linguagem comum e sistematizar o conhecimento existente na área.

Posteriormente, os estudos incluídos na MSL foram classificados com base na taxonomia proposta, de acordo com as categorias estabelecidas. Esse processo permitiu avaliar sua aplicabilidade e identificar padrões recorrentes na literatura, cujos resultados foram publicados no Simpósio Brasileiro de Engenharia de Software (GOMES *et al.*, 2024).

Além disso, foi conduzido o mapeamento de **ferramentas, aplicações de *benchmarking* e conjuntos de dados (*datasets*) de *workload* do mundo real** empregados nos trabalhos analisados, com o objetivo de identificar os recursos experimentais mais recorrentes na literatura. Essa análise, juntamente com a extensão da taxonomia, foi publicada em um periódico especializado (GOMES *et al.*, 2025b).

1.5.2 Catálogo de Anti-padrões de Observabilidade

A partir do estudo da taxonomia de observabilidade foi verificada a ausência de um estudo que focasse em mapear as más práticas da observabilidade. Assim, a Etapa 2 concentrou-se na identificação, estruturação e validação de um catálogo de APs de observabilidade, com o intuito de mapear práticas recorrentes que impactam negativamente a capacidade de observar, monitorar e diagnosticar aplicações de microsserviços.

Um novo Mapeamento Sistemático da Literatura (MSL) foi realizado, e cuja revisão

possibilitou a identificação de padrões problemáticos presentes em implementações descritas na literatura. Complementarmente, realizou-se um Mapeamento da Literatura Cinzenta (MLC) com o intuito de capturar relatos e experiências de profissionais da área, presentes em *blogs* técnicos, fóruns especializados e vídeos.

Posteriormente, foi realizada a identificação de APs de acordo com a literatura. As informações extraídas foram organizadas em um **catálogo de anti-padrões de observabilidade**, que é estruturado com base em um *template* padronizado, contendo atributos como nome, categoria, relevância, contexto, problema, exemplo, soluções e consequências. Em seguida, para facilitar o acesso ao conteúdo produzido, foi realizado o desenvolvimento de um portal digital com funcionalidades de navegação e filtragem dos APs, visando tornar o catálogo acessível e útil para profissionais e pesquisadores.

Por fim, a validade do catálogo foi verificada por meio de uma **avaliação com especialistas** da área de observabilidade, os quais contribuíram com *feedbacks* sobre clareza, legibilidade, utilidade, relevância e completude dos itens descritos. Os resultados dessa etapa foram publicados em (GOMES *et al.*, 2025a). A partir dessa publicação, o trabalho recebeu convite para extensão em *Special Issue* de Engenharia de Software do periódico científico *SBC Computing Reviews*, resultando em uma nova submissão em versão ampliada, a qual se encontra atualmente em processo de revisão.

1.5.3 Framework Observa

Para apoiar o catálogo de APs de observabilidade, a Etapa 3 concentrou-se no desenvolvimento de um *framework* intitulado Observa, que é uma solução capaz de identificar automaticamente APs de observabilidade.

Inicialmente, o projeto do Observa contemplou a definição e a análise das funcionalidades necessárias para atender aos objetivos propostos. Complementarmente, foi definida a arquitetura da solução, abrangendo todos os módulos responsáveis por suportar as funcionalidades previstas no projeto. Em seguida, procedeu-se à implementação da solução proposta, considerando os requisitos funcionais e arquiteturais previamente estabelecidos. Depois, com o intuito de validar o funcionamento do Observa e assegurar sua capacidade de detecção dos APs, foi desenvolvida uma prova de conceito.

Por fim, foram conduzidos experimentos utilizando dados reais provenientes de empresas, bem como dados sintéticos, além da aplicação de uma avaliação com especialistas,

com o objetivo de avaliar a capacidade de detecção, a usabilidade e o potencial de recomendação da solução. O Observa com a prova de conceito foi submetido para o Salão de Ferramentas do Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC), e encontra-se atualmente em processo de revisão.

A abordagem adotada, que combina mapeamento sistemático, análise da literatura cinzenta, estruturação conceitual, validação empírica e planejamento de aplicação prática, visa assegurar a robustez metodológica da tese, bem como resultados obtidos e a relevância dos artefatos para a comunidade técnica e científica. Cada uma dessas etapas é detalhada nos capítulos 4, 5 e 6 desta tese, com ênfase nos métodos aplicados, nos dados coletados e nas análises realizadas.

1.6 Organização da Tese

Esta tese está organizada em nove capítulos. Este capítulo introdutório apresentou uma visão geral do tema, contextualizando o problema de pesquisa, a motivação, os objetivos e as contribuições, bem como a metodologia adotada.

O Capítulo 2 trata da fundamentação teórica, abordando os principais conceitos utilizados ao longo do trabalho, como microsserviços, observabilidade e APs.

O Capítulo 3 apresenta os trabalhos relacionados, com foco em estudos secundários que investigam o estado da arte sobre monitoramento e observabilidade em sistemas de software.

O Capítulo 4 detalha a taxonomia construída, apresentando sua metodologia de desenvolvimento, os domínios e categorias definidos, a classificação dos estudos selecionados, bem como o mapeamento de ferramentas, aplicações *benchmarking* e conjuntos de dados do mundo real identificados na literatura.

O Capítulo 5 descreve o catálogo de APs elaborado, incluindo o processo metodológico, as fontes consultadas, o *template* utilizado para sistematizar os dados, o portal digital e aberto criado para disponibilização pública do conteúdo, além da avaliação realizada com especialistas da área. De forma complementar, é apresentada uma solução para o problema de alertas excessivos, com o objetivo de reduzir a sobrecarga de informações a serem analisadas pelos operadores.

O Capítulo 6 descreve em detalhes o *framework* desenvolvido para a detecção de anti-padrões de observabilidade, apresentando sua arquitetura, componentes principais e o funcionamento dos mecanismos responsáveis pela identificação automatizada desses problemas.

O Capítulo 7 apresenta os experimentos conduzidos para validar o *framework*, detalhando o ambiente de avaliação, os cenários de teste, bem como os métodos de coleta e análise dos resultados, além da discussão das evidências obtidas a partir de sua aplicação prática.

Por fim, o Capítulo 8 reúne as conclusões deste trabalho, destacando as principais contribuições alcançadas, as limitações identificadas durante o processo de pesquisa e as perspectivas para trabalhos futuros, que buscam aprimorar e expandir as abordagens aqui desenvolvidas.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta os conceitos, as definições e as principais características das áreas e subáreas que fundamentam esta pesquisa, e são essenciais para sua compreensão. A Seção 2.1 aborda os conceitos relacionados a microsserviços, incluindo as vantagens e as desvantagens dessa arquitetura para aplicações. A Seção 2.2 explora a observabilidade, destacando suas definições, a diferença entre monitoramento e observabilidade, os tipos de dados envolvidos, bem como algumas ferramentas, e o principal *framework open-source* para dados de telemetria, o *OpenTelemetry*. Por fim, a Seção 2.3 trata dos conceitos de Anti-Padrões (APs), com ênfase na observabilidade.

2.1 Microsserviços

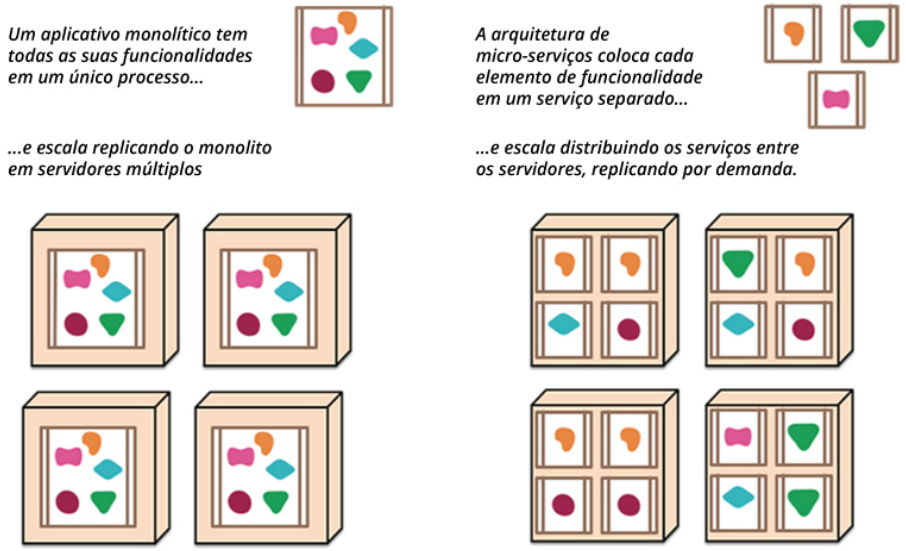
A arquitetura de microsserviços se distingue das arquiteturas monolíticas tradicionais por sua abordagem modular e distribuída de desenvolvimento. Enquanto sistemas monolíticos enfrentam desafios de escalabilidade e limitações na autonomia das equipes de desenvolvimento para adotar novas tecnologias (AL-DEBAGY; MARTINEK, 2018), os microsserviços proporcionam maior flexibilidade e independência. Em um sistema monolítico, escalar a aplicação implica custos de tempo significativos e exige uma equipe altamente madura. Mesmo quando os módulos são minimizados para reduzir a complexidade do sistema, eles geralmente são projetados para operar como um único bloco. Assim, a escalabilidade se torna um desafio, pois todos os serviços da aplicação são escalados simultaneamente, independentemente da demanda (BLINOWSKI *et al.*, 2022).

O estilo arquitetural de microsserviços envolve o desenvolvimento de uma única aplicação como um conjunto de pequenos serviços, cada um operando de forma independente e comunicando-se por meio de mecanismos leves, geralmente, uma API REST baseada em HTTP (RAZZAQ; GHAYYUR, 2023). Esses serviços são construídos em torno de capacidades de negócio e podem ser implantados de forma independente, com mecanismos automatizados de implantação. Também não há a imposição de um gerenciamento centralizado, permitindo que sejam escritos em diferentes linguagens e utilizem diferentes tecnologias de armazenamento.

A Figura 2, adaptada de Lewis e Fowler (2014), mostra o contraste entre aplicações monolíticas e arquiteturas baseadas em microsserviços. No lado monolítico, uma única caixa engloba todas as funcionalidades da aplicação, em que normalmente, esse monólito centraliza

diversas camadas (interface gráfica, lógica de negócio, acesso a dados). Caso necessite ser escalado, toda a aplicação será replicada. No lado microsserviços, a aplicação é dividida em múltiplas caixas menores, cada uma representando um microsserviço com escopo claro do negócio. Cada microsserviço inclui sua própria interface gráfica (quando necessário), lógica e armazenamento de dados. Caso necessite ser escalado, apenas o que precisa será, efetivamente, replicado.

Figura 2 – Diferenças entre arquitetura monolítica e baseada em microsserviços.



Fonte: Adaptado de Lewis e Fowler (2014).

Esse estilo de microsserviços, conforme descrito em (DRAGONI *et al.*, 2017), consiste em serviços autônomos com poucas responsabilidades, capazes de operar tanto isoladamente quanto em conjunto com outros serviços. Abgaz *et al.* (2023) descrevem a arquitetura de microsserviços como a separação de uma aplicação complexa em serviços distribuídos, cada um com funções específicas e utilizando comunicação leve. No entanto, tais autores também apontam que essa abordagem pode tornar as aplicações propensas a falhas devido à complexidade das dependências entre os microsserviços.

2.1.1 Vantagens e Desvantagens dos Microsserviços

Algumas das principais vantagens associadas à tecnologia de microsserviços podem ser destacadas a partir da literatura (SHADIJA *et al.*, 2017; RAZZAQ; GHAYYUR, 2023; FRANCESCO *et al.*, 2019). Uma das características mais relevantes é a **heterogeneidade tecnológica**, já que os microsserviços permitem a construção de aplicações compostas por

diversos serviços colaborativos, cada um desenvolvido em diferentes linguagens e plataformas. Essa abordagem concede aos desenvolvedores maior flexibilidade para selecionar as tecnologias mais adequadas a cada necessidade e possibilita a experimentação de novas soluções com riscos reduzidos.

Outro benefício central refere-se à **resiliência**, em que diferentemente de sistemas monolíticos, nos quais a falha de um único componente pode comprometer toda a aplicação, em arquiteturas de microsserviços é possível isolar serviços com problemas, permitindo que os demais continuem operando com impacto mínimo. A **escalabilidade de recursos** também é favorecida, pois enquanto em sistemas monolíticos o processo de escalar recursos pode ser complexo e demandar arquiteturas cuidadosamente planejadas para evitar desperdícios, os microsserviços permitem escalar individualmente cada serviço, o que simplifica e torna mais eficiente a alocação de recursos.

Adicionalmente, destaca-se a **facilidade de implantação**, como cada microsserviço pode ser implantado de forma independente, novas funcionalidades podem ser introduzidas sem comprometer a aplicação como um todo. Essa característica reduz o tempo de inatividade, aumenta a agilidade no processo de entrega e permite reverter rapidamente serviços problemáticos por meio de *rollback*, sem afetar os demais. Em contraste, arquiteturas monolíticas exigem que toda a aplicação seja implantada de uma só vez, o que torna o processo mais lento e arriscado, já que falhas em componentes isolados podem impactar a aplicação inteira.

Embora o ambiente de microsserviços ofereça diversas vantagens, ele também apresenta algumas desvantagens que precisam ser consideradas. Uma delas refere-se ao **gerenciamento de banco de dados**, uma vez que a administração de múltiplos repositórios distribuídos exige esforço significativo tanto para configuração quanto para manutenção da integridade dos dados. Além disso, o controle de transações em ambientes distribuídos torna-se particularmente desafiador devido à sua natureza descentralizada.

Outro aspecto crítico diz respeito à **comunicação entre serviços**, pois chamadas de rede são mais custosas do que chamadas internas, então a troca de informações entre microsserviços pode comprometer o desempenho se não for devidamente escalada. Nesse sentido, torna-se essencial avaliar criteriosamente quais funcionalidades devem, de fato, ser implementadas como microsserviços.

Por fim, a **complexidade do desenvolvimento** também aumenta nesse paradigma, pois apesar do suporte de ferramentas que automatizam processos de *build* e gerenciamento,

ainda é necessário manter versões atualizadas de cada serviço e realizar extensivos testes de integração, o que amplia o esforço de coordenação e dificulta a manutenção da solução.

2.1.2 Tecnologias para Microsserviços

Implantar e gerenciar a arquitetura de microsserviços na nuvem é uma preocupação constante para os administradores de ambientes de microsserviço. Contudo, essa preocupação é um pouco amenizada devido ao uso de tecnologias que facilitam a entrega das aplicações. A seguir, será apresentado algumas dessas tecnologias.

2.1.2.1 Contêineres

Segundo Pahl *et al.* (2017), os contêineres representam uma alternativa mais leve em relação às máquinas virtuais. Eles criam ambientes que encapsulam aplicações, oferecendo apenas os recursos indispensáveis para sua execução. Esses ambientes garantem isolamento em níveis de rede, disco, processamento e memória, o que possibilita elevada flexibilidade e a execução simultânea de diversas aplicações em um mesmo hospedeiro.

A principal distinção entre contêineres e máquinas virtuais está no fato de que estas últimas necessitam de um sistema operacional completo para cada instância criada e dependem de uma camada de software adicional, o hipervisor, responsável por intermediar a comunicação com o sistema operacional da máquina física (RODRIGUEZ; BUYYA, 2019). Já os contêineres acessam diretamente os recursos do sistema hospedeiro, como memória, disco e rede. Assim, para utilizá-los basta instalar apenas os requisitos específicos do microsserviço (bibliotecas, dependências e arquivos de configuração), sem a necessidade de um sistema operacional adicional e sem a sobrecarga da camada de hipervisor. Nesse contexto, surge o *Docker*[1], tem se consolidado como a tecnologia mais popular para viabilizar a utilização de contêineres em ambientes de desenvolvimento e produção.

2.1.2.2 Docker

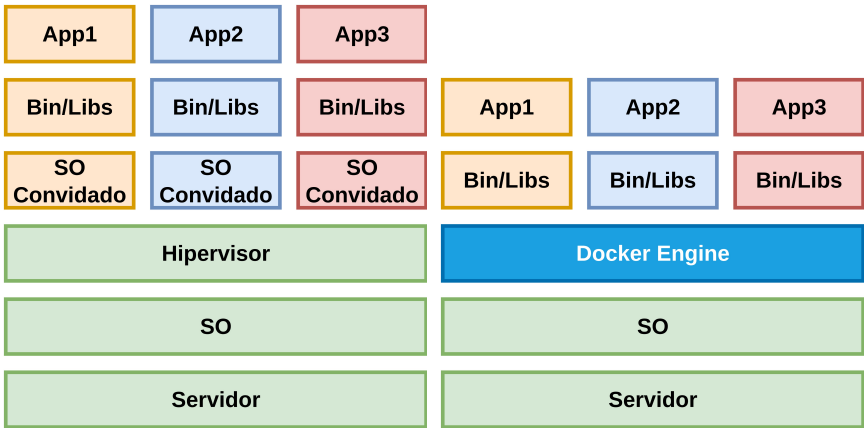
O *Docker* é um projeto mantido pela comunidade *open source*, amplamente reconhecido como a solução de contêineres mais popular. A ferramenta permite criar e gerenciar múltiplos contêineres em um mesmo ambiente de forma prática. Entre as vantagens proporciona-

[1] <https://www.docker.com/>

das estão a possibilidade de compartilhamento de ambientes entre equipes de desenvolvimento, além da leveza e da simplicidade no processo de configuração.

A Figura 3 apresenta uma comparação entre a arquitetura baseada em máquinas virtuais e a arquitetura em contêineres com *Docker*. Enquanto as máquinas virtuais executam um sistema operacional completo em cada instância, o *Docker* utiliza o *kernel* do hospedeiro e isola as aplicações em processos independentes. Essa abordagem reduz o consumo de recursos e torna o ciclo de desenvolvimento e implantação significativamente mais eficiente em relação às máquinas virtuais.

Figura 3 – Arquitetura com máquinas virtuais X *Docker*.



(a) Máquinas Virtuais (b) Contêineres *Docker*

Fonte: Autoria própria.

Embora o *Docker* seja eficaz no gerenciamento de contêineres individuais, quando a aplicação é dividida em inúmeros contêineres seguindo o paradigma de microsserviços, surgem desafios relacionados à administração e à orquestração desse ecossistema. Para resolver essas limitações, surgiram ferramentas específicas de orquestração, sendo o *Kubernetes*[2] a mais consolidada no cenário atual.

2.1.2.3 Kubernetes

A ferramenta *open source* Kubernetes é amplamente utilizada para automatizar a implantação, o dimensionamento e o gerenciamento de aplicações em contêineres por meio de um *cluster* (BURNS *et al.*, 2019). Trata-se de uma plataforma projetada para administrar todo o ciclo de vida de serviços conteinerizados. No Kubernetes, a menor unidade de implantação é o *Pod*, que pode conter um ou mais contêineres logicamente relacionados, compartilhando o

[2] <https://kubernetes.io/pt-br/>

mesmo endereço IP, *namespace* de rede e volumes de armazenamento. Os *Pods* representam a unidade básica de execução no *cluster* e são efêmeros por natureza, podendo ser recriados automaticamente conforme políticas de disponibilidade e escalabilidade.

No contexto de arquiteturas baseadas em microsserviços, cada serviço é tipicamente empacotado como um contêiner e implantado em um ou mais *Pods*. Essa abordagem permite que cada microsserviço seja escalado, atualizado e monitorado de forma independente, respeitando o princípio de baixo acoplamento e alta coesão característico desse estilo arquitetural. O Kubernetes orquestra essas unidades de execução, possibilitando a comunicação entre microsserviços por meio de mecanismos internos de rede e descoberta de serviços, além de suportar estratégias de implantação contínua, como *rolling updates* e *rollback*, fundamentais para ambientes distribuídos e altamente dinâmicos.

Suas funcionalidades centrais incluem o agendamento de execução (*scheduling*), que distribui Pods de forma inteligente entre múltiplos nós do *cluster*, equilibrando a carga de trabalho e otimizando o uso de recursos computacionais. Além disso, o Kubernetes possui mecanismos de verificação de integridade (*liveness* e *readiness probes*), que realizam monitoramento contínuo dos *Pods* para assegurar alta disponibilidade e detecção precoce de falhas. Ele também disponibiliza mecanismos de *Domain Name System* (DNS) e descoberta de serviços, permitindo o roteamento dinâmico e a comunicação eficiente entre microsserviços. Por fim, ele realiza o gerenciamento de recursos, acompanhando em tempo real métricas essenciais como uso de CPU, memória, armazenamento e largura de banda, possibilitando ajustes automáticos e garantindo o desempenho consistente de aplicações distribuídas.

Apesar da evolução das tecnologias que facilitam a execução e o gerenciamento de microsserviços em ambientes distribuídos, ainda existem desafios significativos relacionados à observação de falhas e degradação de desempenho. Os microsserviços se tornaram a abordagem dominante para *Cloud-Native Application* (CNA) (LI *et al.*, 2022; KRATZKE; QUINT, 2017). Estas aplicações, geralmente, representam sistemas industriais que podem consistir em centenas a milhares de serviços, operando em infraestruturas complexas na nuvem, com instâncias de serviços sendo criadas e destruídas dinamicamente (SENAPATHI *et al.*, 2018; WETTINGER *et al.*, 2016). Diagnosticar problemas como falhas em requisições e alta latência em arquiteturas de microsserviços é extremamente desafiador devido ao grande número de serviços envolvidos (RICHARDSON, 2018). Em termos de desenvolvimento e implantação, microsserviços são mais exigentes do que aplicações tradicionais, pois são compostos por diversos componentes,

como máquinas físicas, máquinas virtuais e contêineres (DUVALL *et al.*, 2007; HUMBLE; FARLEY, 2010; SHAHIN *et al.*, 2017). Dessa forma, verificar a integridade, o desempenho e o comportamento de cada componente é essencial, destacando a importância do monitoramento como uma ferramenta crítica para microsserviços (PAHL *et al.*, 2017; GANNON *et al.*, 2017; BALALAIE *et al.*, 2016).

2.2 Observabilidade

Embora o monitoramento tenha desempenhado um papel central na TI ao longo das últimas décadas, sua eficácia tem sido questionada devido a vários fatores, incluindo o surgimento de metodologias ágeis de desenvolvimento, implantações nativas em nuvem e novas práticas DevOps. Esses fatores mudaram a forma como todo o ecossistema de TI (infraestruturas, sistemas e aplicações) precisa ser monitorado. No entanto, a simples aplicação de ferramentas tradicionais de monitoramento, normalmente, baseadas em métricas isoladas e alertas estáticos, não é suficiente para lidar com a alta dinamicidade, a efemeridade dos componentes e a complexidade das dependências em arquiteturas modernas. Como consequência, torna-se difícil identificar a causa raiz de falhas e compreender o comportamento sistêmico de aplicações distribuídas.

Garantir o funcionamento adequado dos serviços e manter a qualidade do serviço exige mais do que monitoramento básico. Provedores de serviços em nuvem e usuários precisam acompanhar continuamente o uso, o desempenho e a disponibilidade dos recursos (SCROCCA *et al.*, 2020; LEVIN; BENSON, 2020). Em situações de falhas ou degradações, os dados monitorados podem não fornecer informação suficiente para diagnosticar problemas complexos. Essa crescente complexidade operacional impulsionou o desenvolvimento de uma abordagem mais abrangente, conhecida como observabilidade (CARVALHO, 2022; NAQVI *et al.*, 2022; WIDERBERG; JOHANSSON, 2021; NI, 2023).

O conceito de observabilidade tem origem na Teoria de Controle (KALMAN, 1959), que define um sistema como observável se seu estado atual puder ser determinado em um tempo finito apenas por meio de suas saídas. Medir o desempenho geral de um microsserviço é crucial para garantir o cumprimento dos parâmetros de *Quality of Service* (QoS) estabelecidos no *Service Level Agreement* (SLA). Para alcançar esses parâmetros, o sistema deve expor adequadamente seu estado por meio de técnicas de instrumentação.

A observabilidade é frequentemente composta por métricas, *logs* e *traces* e é uma característica essencial para aplicações modernas, como as CNA (KOSIŃSKA; ZIELIŃSKI,

2020), pois o seu principal objetivo é tornar a aplicação mais compreensível em vários níveis. Niedermaier *et al.* (2019) afirmam que, em ambientes em nuvem, a observabilidade indica o grau em que a infraestrutura, as aplicações e suas interações podem ser monitoradas.

Ao longo dos anos, diversos autores ofereceram definições complementares sobre o conceito de observabilidade. Marie-Magdelaine (2021) consideram observabilidade como um conceito emergente que amplia os limites do monitoramento e incorpora técnicas de análise de dados. Já Kratzke (2022) descreve a observabilidade como a base para o desenvolvimento de software orientado a dados.

Segundo Usman *et al.* (2022), a observabilidade é um conjunto de práticas em expansão, combinadas com ferramentas que vão além do mero monitoramento, proporcionando observações sobre o estado interno de um sistema por meio da análise de suas saídas externas, predominantemente conhecidas como dados de telemetria, tais como métricas, *logs* e *traces*. Essa abordagem permite uma visão de alto nível da saúde do sistema, além de oferecer observações detalhadas sobre modos de falha subjacentes. O presente trabalho adota a definição de Usman *et al.* (2022) por sua abrangência e por enfatizar o uso de saídas externas, em alinhamento com (KALMAN, 1959).

As motivações para equipar aplicações baseadas em microsserviços com observabilidade derivam da fase de execução dessas aplicações (KOSIŃSKA *et al.*, 2023). Elas estão relacionadas ao provisionamento de recursos com foco no rastreamento em tempo real, na medição da sobrecarga do sistema e no escalonamento, incluindo aspectos como escalabilidade. Do ponto de vista da gestão, as motivações mais relevantes incluem a detecção da causa raiz (análise de falhas, previsão de falhas, gerenciamento de falhas, análise de dependências), o gerenciamento de SLA (carga, disponibilidade, QoS e violações) (POURMAJIDI *et al.*, 2025), bem como a gestão de ponta a ponta (desempenho, tempo de resposta, análise de latência e rastreamento ponta a ponta), entre outros (SHATNAWI *et al.*, 2023).

2.2.1 Diferença entre Monitoramento e Observabilidade

Um ponto importante ao se falar sobre observabilidade é a diferença relacionada ao conhecido termo de “monitoramento” de sistemas. O monitoramento envolve o rastreamento da integridade de um sistema usando um conjunto predefinido de métricas e *logs*, focando na detecção de falhas previamente conhecidas. No entanto, em um sistema distribuído, muitas mudanças são dinâmicas, frequentemente ocorrem em paralelo e podem acontecer de maneira

imprevisível. Isso pode levar a problemas que não se encaixam nas categorias previamente definidas pelo monitoramento, escapando da sua detecção. A observabilidade, portanto, complementa o monitoramento ao fornecer uma análise mais profunda sobre as falhas e suas causas, ajudando a identificar onde e por que uma falha ocorreu, e o que desencadeou o problema. No entanto, ela não substitui o monitoramento nem elimina sua necessidade, pois ambos trabalham em conjunto.

Para ilustrar melhor essa diferença, considere uma plataforma de *e-commerce* que acaba de lançar um novo recurso de *checkout*. Pela manhã, a equipe de operações de TI verifica seus *dashboards* e percebe uma queda significativa de desempenho no serviço responsável pela taxa de conversão dos pedidos dos clientes. O sistema de monitoramento alerta que a latência do serviço de pagamento está acima do normal, e a equipe rapidamente identifica que o problema pode estar relacionado a esse serviço. No entanto, para obter mais detalhes, recorrem a uma plataforma de observabilidade, explorando *logs* e *traces*. Isso permite não apenas identificar a alta latência, mas também localizar as partes do código responsáveis pelo problema. Ao analisar requisições individuais, descobrem que um novo *endpoint* falha sob carga elevada. Enquanto o monitoramento fornece uma visão do que está acontecendo, a observabilidade oferece uma compreensão do porquê. Com essas observações, a equipe otimiza o *endpoint* e restaura o serviço de taxa de conversão, evidenciando como monitoramento e observabilidade são distintos, mas complementares em ecossistemas complexos.

Embora essa distinção seja clara, a implementação e o uso prático podem variar. Algumas empresas podem usar os termos de forma intercambiável ou ter uma compreensão superficial de como esses conceitos se aplicam aos seus ambientes. Portanto, apesar da diferenciação ser amplamente aceita, sua aplicação e interpretação podem diferir de uma organização para outra.

2.2.2 Benefícios da Observabilidade

Aplicações baseadas em microsserviços podem alcançar melhorias significativas de desempenho quando seus ambientes de execução são equipados com capacidades avançadas de observabilidade (KOSIŃSKA *et al.*, 2023). Entre os principais benefícios associados a um ecossistema de observabilidade, destaca-se o **aumento da visibilidade**, que proporciona uma compreensão abrangente das atividades do sistema, permitindo identificar serviços ativos, avaliar seu desempenho, localizar componentes e detectar condições inesperadas. Outro aspecto relevante é a **detecção precoce de problemas**, que viabiliza a resolução de falhas antes que

impactem a funcionalidade da aplicação, abrangendo tanto questões de desempenho quanto aspectos arquiteturais e de *design*.

A observabilidade também contribui para a **melhoria dos processos de *DevOps* e do fluxo de desenvolvimento**, oferecendo informações valiosas sobre o comportamento do sistema em tempo real que podem orientar decisões de *design* e até mesmo motivar ajustes arquiteturais. Além disso, ela apoia a **escalabilidade** ao garantir que subsistemas, em especial aqueles relacionados ao monitoramento, sejam capazes de lidar com o aumento no volume de dados à medida que o sistema evolui.

Outro benefício importante consiste na **habilitação de automação**, uma vez que os dados de observabilidade podem sustentar mecanismos como escalabilidade dinâmica e *failover* automático, favorecendo um controle mais autônomo do ambiente. Esses **dados também têm grande valor analítico**, podendo ser utilizados em avaliações estatísticas e em técnicas de aprendizado de máquina, o que permite observações mais profundas sobre o comportamento da aplicação e seu contexto de execução. Por fim, destaca-se a contribuição direta da observabilidade para a **experiência do usuário**, em que aplicações com mecanismos de observabilidade tendem a oferecer maior confiabilidade e qualidade de serviço, resultando em níveis superiores de satisfação do cliente e, consequentemente, no fortalecimento dos objetivos de negócio.

2.2.3 Tipos de Dados de Observabilidade

Como mencionado anteriormente e ilustrado na Figura 4, existem três tipos principais de dados, também denominados domínios, frequentemente referidos como os pilares da observabilidade (COSTA *et al.*, 2022): *Traces*, Métricas e *Logs*.

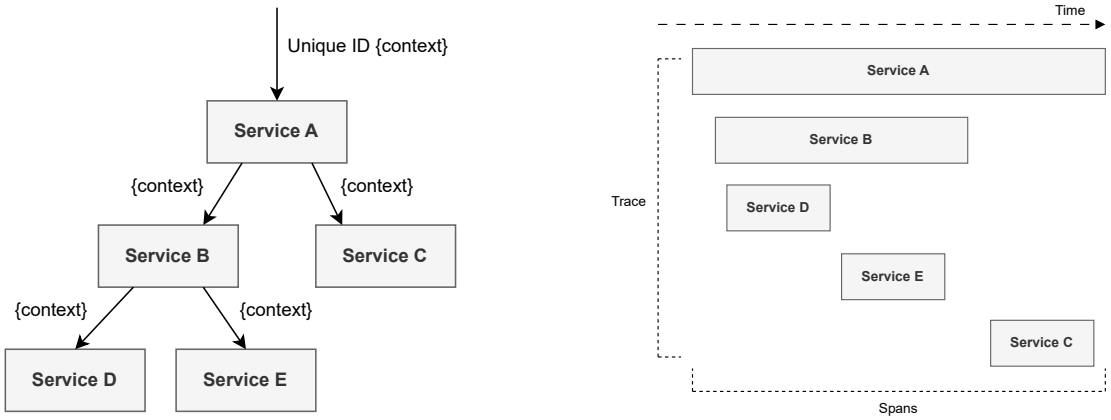
2.2.3.1 Traces

Traces fornecem uma visão abrangente do fluxo de uma solicitação por meio de uma aplicação (POURMAJIDI *et al.*, 2025). Os *traces* são essenciais para entender o caminho completo que uma solicitação percorre na aplicação. Um *trace* distribuído consiste em uma série de eventos gerados em diferentes pontos de um sistema e possui uma estrutura em árvore composta por invocações de serviços (como mostrado na Figura 5a), conectados por um identificador único. Esse identificador é propagado por todos os componentes envolvidos em qualquer operação necessária para concluir a solicitação, permitindo que cada operação associe dados de evento com a solicitação original.

Figura 4 – Três pilares da observabilidade.

Fonte: Autoria própria.

Figura 5 – Representação de um *trace* da requisição.



(a) Topologia da requisição.

(b) Linha do tempo da requisição.

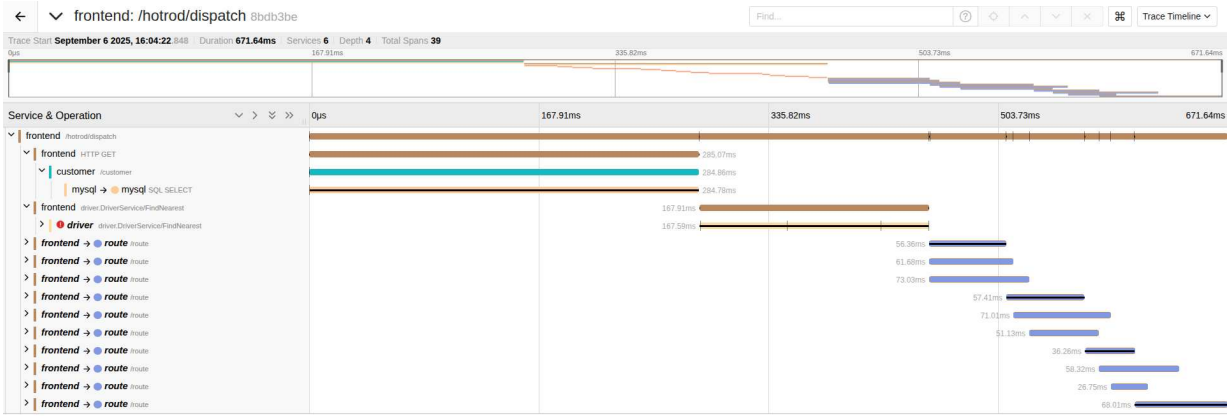
Fonte: Li *et al.* (2022).

Cada *trace* representa uma solicitação única por meio de um sistema, que pode ser tanto síncrona quanto assíncrona. As solicitações síncronas ocorrem sequencialmente, com cada unidade de trabalho concluída antes de prosseguir. Solicitações assíncronas podem iniciar uma série de operações que podem ocorrer simultaneamente e de forma independente. Cada operação registrada em um *trace* é representada por um *span*, uma única unidade de trabalho executada no sistema, como mostrado na Figura 5b, que é uma operação nomeada e cronometrada (isto é, uma invocação de serviço) representando uma parte do fluxo.

Um *span* possui um identificador único denominado *span ID*. Já o *context* corresponde ao conjunto de informações de *trace* que acompanham a transação distribuída, inclusive quando esta é propagada entre serviços pela rede. Esse *context* inclui o *trace ID*, o *span ID* e outros metadados necessários para que o sistema de rastreamento mantenha a continuidade das informações entre serviços. Quando um ponto de instrumentação é acionado em uma instância de serviço, um registro é gerado como um *span log*. Os *span logs* são, geralmente, persistidos de forma assíncrona antes de serem enviados a um coletor, sendo então reconstruídos em um *trace* com base nos diferentes *logs* emitidos por distintos serviços (LI *et al.*, 2022).

Com relação às ferramentas de *traces*, o *Jaeger*[3] é uma plataforma popular de rastreamento distribuído que permite monitorar e analisar transações complexas em arquiteturas baseadas em microsserviços, fornecendo recursos como rastreamento de dependências, análise de latência e detecção de gargalos de desempenho. A Figura 6 ilustra o uso do *Jaeger*, que informa a duração da requisição, o acesso a cada um dos serviços, bem como a duração de cada *span*. Cada *span* contém os detalhes da requisição naquele serviço.

Figura 6 – Exemplo de uso do Jaeger.



Fonte: Autoria própria.

[3] <https://www.jaegertracing.io/>

2.2.3.2 Métricas

Métricas 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 nas 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 em um intervalo de tempo. 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.

Com relação as ferramentas para o monitoramento de métricas, o *Prometheus*[4] é uma das ferramentas mais populares no ecossistema de microsserviços e infraestrutura em nuvem. Ele coleta, armazena e consulta métricas em tempo real utilizando um modelo de séries temporais, oferecendo recursos de alerta e integração com diversas ferramentas de visualização. A Figura 7 mostra o uso do *Prometheus*, realizando a consulta de uma métrica chamada “kubelet_running_containers{container_state="running"}”, que verifica a quantidade de *pods* em execução em cada nó.

Figura 7 – Exemplo de uso do Prometheus.

Fonte: Autoria própria.

[4] <https://prometheus.io/>

2.2.3.3 Logs

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

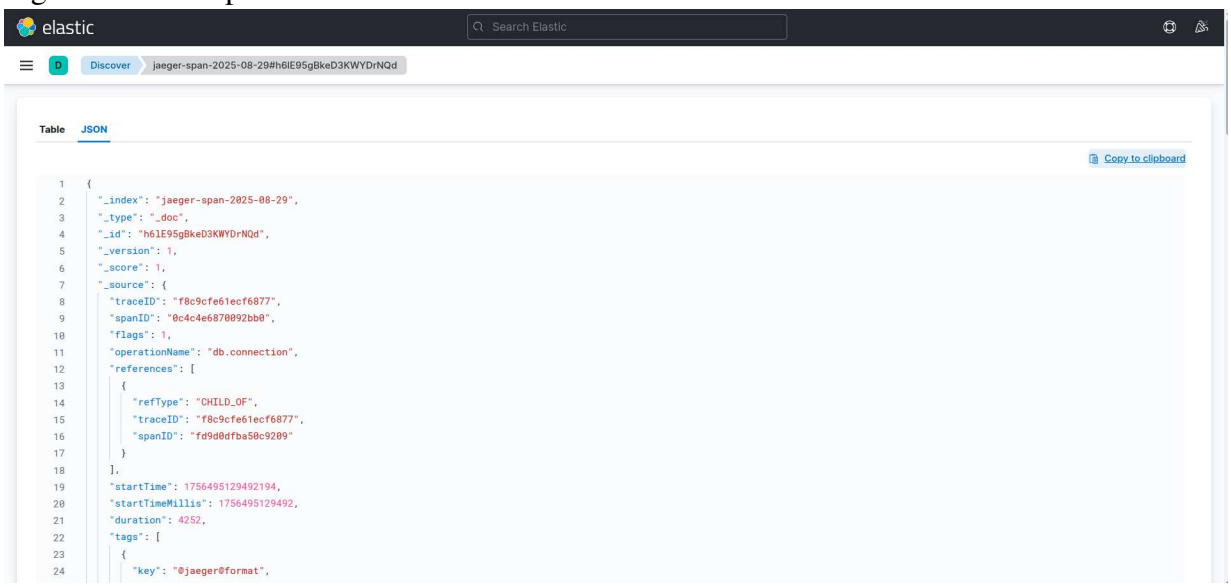
Com relação às ferramentas de *logs*, a *Elastic Stack*[5] constitui um popular ecossistema voltado para a coleta, armazenamento, análise e visualização de dados em larga escala. No centro desse ecossistema, o *Elasticsearch* atua como motor de busca e análise distribuído, projetado para lidar com grandes volumes de dados e oferecer consultas de baixa latência. Complementarmente, o *Kibana* funciona como a camada de visualização, disponibilizando recursos avançados de exploração interativa por meio de *dashboards* dinâmicos, gráficos e relatórios customizáveis.

A Figura 8 mostra o uso do *Kibana*, apresentando um *log* em formato estruturado, do tipo *JavaScript Object Notation* (JSON), que representa uma requisição coletada pelo *Jaeger*, e armazenada no *Elasticsearch*. Esse formato, baseado em pares chave–valor, facilita a indexação e a aplicação de filtros, agregações e análises no *Kibana*, favorecendo a identificação de padrões e a correlação de eventos em sistemas distribuídos.

A correlação entre métricas, *logs* e *traces* é essencial para fornecer uma visão completa e integrada do comportamento do sistema. Enquanto cada fonte oferece uma perspectiva distinta, é a análise conjunta que permite identificar padrões anômalos, diagnosticar causas raiz de falhas, compreender dependências entre componentes e avaliar impactos em tempo real. Sem essa correlação, problemas complexos podem passar despercebidos ou levar a diagnósticos incompletos, especialmente em sistemas distribuídos e altamente dinâmicos. A capacidade de cruzar informações entre métricas, *logs* e *traces* aumenta significativamente a precisão na detecção de incidentes, possibilita ações corretivas mais rápidas e contribui para manter a confiabilidade, o desempenho e a disponibilidade de aplicações. A próxima subseção aborda o

[5] <https://www.elastic.co/pt/elastic-stack>

Figura 8 – Exemplo de uso do Kibana.



Fonte: Autoria própria.

OpenTelemetry (OTel)[6], uma ferramenta de observabilidade que permite realizar essa coleta e correlação de dados de forma integrada.

2.2.4 OpenTelemetry (OTel)

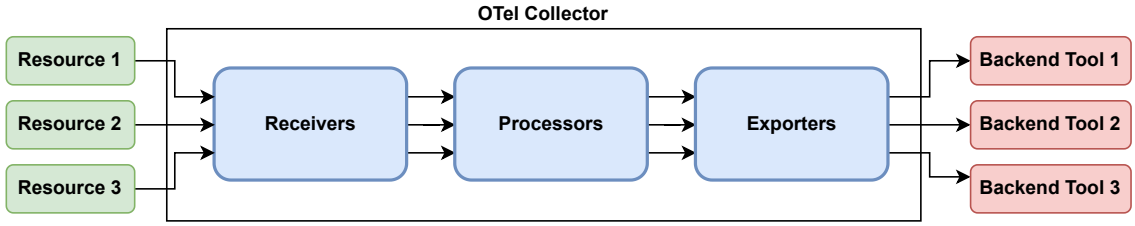
OTel é um projeto da *Cloud Native Computing Foundation* (CNCF) que fornece um *framework* e um conjunto de ferramentas de observabilidade projetados para criar e gerenciar dados de telemetria, como *traces*, métricas e *logs* (BOTEN; MAJORS, 2022). OTel é independente de fornecedor, o que significa que pode ser usado com uma ampla variedade de *backends* de observabilidade, como Jaeger e Prometheus. OTel definiu o protocolo OpenTelemetry (OTLP) para garantir que os dados de telemetria sejam transmitidos da maneira mais eficiente e confiável possível, sendo baseado em arquivos *Protocol Buffers*. Isso significa que qualquer cliente ou servidor interessado em enviar ou receber OTLP precisa apenas implementar essas definições para oferecer suporte a ele. Além disso, ele pode utilizar tanto gRPC quanto HTTP para o transporte de dados.

O OTel simplifica o gerenciamento de telemetria com seu componente *Collector*, que oferece uma abordagem independente de fornecedor para receber, processar e exportar dados. Isso elimina a necessidade de gerenciar múltiplos agentes/coletadores, como mostrado na Figura 9. O *Collector* atua como intermediário entre a fonte de telemetria e o *backend* que armazena os dados para análise. É importante notar que, apesar dos benefícios, recursos adicio-

[6] <https://opentelemetry.io/>

nais são necessários para a execução, operação e monitoramento do *Collector*. O componente de entrada de dados, chamado *Receiver*, permite que os dados cheguem ao *Collector*, sendo capaz de suportar várias fontes, formatos e converter para o formato interno. Geralmente, um *Receiver* registra um ouvinte que expõe uma porta no *Collector* para os protocolos suportados. Após os dados serem recebidos, eles passam por *Processors* para tarefas adicionais, como filtragem ou inserção de atributos. O componente de saída, o *Exporter*, encaminha os dados para um ou mais *backends*/destinos, suportando várias fontes e permitindo a configuração de diferentes *Exporters* para destinos distintos conforme necessário.

Figura 9 – Coletor do OpenTelemetry.



Fonte: Autoria própria.

2.3 Anti-padrões

Uma das maneiras pelas quais os seres humanos resolvem problemas inéditos é por meio da aplicação subconsciente de soluções que foram eficazes em situações semelhantes ou relacionadas. Esse processo, amplamente reconhecido, é chamado de raciocínio analógico, alegórico ou baseado em casos (NEILL; LAPLANTE, 2005). Trata-se de uma abordagem consolidada no Aprendizado de Máquina e em sistemas de Inteligência Artificial. O arquiteto e professor Christopher Alexander (ALEXANDER, 1977) materializou essa ideia ao registrar e compartilhar experiências individuais de forma clara, fundamentada e contextualizada, permitindo sua aplicação eficaz na solução de novos problemas. Ele explorou o conceito de que a arquitetura bem-sucedida se baseia essencialmente na aplicação de padrões de *design* que, embora não formalmente registrados, existiam há milênios. Esses padrões foram criados para que futuros arquitetos pudessem facilmente reconhecê-los e integrá-los em seus próprios projetos. Isso define o conceito de "padrão".

Alexander define um padrão como a descrição de um problema recorrente em nosso ambiente, seguida pela explicação de sua solução central, de modo que essa solução possa ser reutilizada milhões de vezes, sem que se repita de forma idêntica em cada aplicação (ALEXANDER,

1977).

Cada padrão possui um nome específico, que serve para transmitir, de maneira concisa e precisa, a essência e a intenção do padrão. A escolha desse nome é crucial, pois o conjunto de padrões dentro de um determinado contexto forma um verdadeiro vocabulário de comunicação (i.e., uma linguagem) (ALEXANDER, 1977). Essas linguagens possibilitam que grupos compartilhem ideias amplas e soluções complexas para problemas de maneira eficiente. Em vez de detalhar minuciosamente os aspectos de um problema ou a lógica por trás de uma solução, os grupos podem simplesmente utilizar os nomes dos padrões, permitindo uma compreensão clara e imediata do trabalho em questão por todos os envolvidos. Além do nome, do conflito (ou problema) e da resolução (ou solução), formatos mais abrangentes incluem seções que descrevem as consequências do uso do padrão, tanto positivas quanto negativas e o escopo do problema que o padrão busca solucionar.

Embora seja útil estudar as soluções bem-sucedidas para resolver problemas, a análise das falhas pode trazer aprendizados ainda mais valiosos. Esse é o conceito subjacente aos padrões negativos, ou Anti-padrões (APs) (NEILL; LAPLANTE, 2005). Enquanto os padrões descrevem um problema recorrente e sua respectiva solução, os APs descrevem soluções cujas consequências negativas superam os benefícios. Em essência, eles representam abordagens ineficazes para a resolução de problemas, seguidas das modificações necessárias para corrigir essa falha. Evidentemente, não se pretende criar essas situações problemáticas; elas surgem por negligência, ignorância ou diversos outros motivos. Uma vez preso a esses cenários falhos, é necessário saber como sair deles e evitar que se repitam. Surge, assim, a necessidade de existir um catálogo de APs.

A ideia dos APs surgiu logo após a dos padrões. Koenig (1998) publicou um artigo utilizando o termo, em que diz que o Anti-Padrão (AP) é como um padrão, exceto que, em vez de uma solução, ele fornece algo que parece superficialmente uma solução, mas não é. A popularização do conceito é oriunda do livro de Brown *et al.* (1998).

2.3.1 Anti-padrões da Observabilidade

A implementação inadequada da observabilidade pode resultar em efeitos adversos, gerando sobrecarga de dados e dificultando a compreensão do comportamento do sistema. Esse cenário de falsa percepção de controle do sistema ocorre justamente quando APs de observabilidade são adotados (TOWNSHEND, 2023). Em Engenharia de Software, um AP

refere-se a uma solução aparentemente eficaz que, com o tempo, revela-se prejudicial, trazendo impactos negativos inesperados ao sistema. Essas falhas decorrem da busca por soluções imediatas, da falta de experiência ou do desconhecimento sobre as consequências de determinadas escolhas (MITTAL, 2022).

Os APs de observabilidade comprometem a visibilidade do sistema, tornando mais difícil diagnosticar falhas, realizar manutenções e otimizar o desempenho das aplicações. Em vez de aprimorar a compreensão sobre o estado do sistema, essas práticas geram ruído, dificultando a extração de informações relevantes (CHEVRE, 2024). Assim como ocorre em outras áreas da engenharia de software, sua adoção geralmente resulta da necessidade de respostas rápidas ou da falta de consciência sobre os efeitos adversos dessas decisões. Quando mal implementada, a observabilidade não apenas falha em seu propósito, mas também se torna um obstáculo à gestão eficiente do sistema.

Um dos maiores problemas associado aos APs da observabilidade é o excesso de dados sem valor analítico significativo (YANG *et al.*, 2022; MHAISEKAR, 2024). Muitas organizações tentam compensar a falta de visibilidade acumulando registros de *logs*, métricas e *traces* indiscriminadamente, sob a suposição equivocada de que um maior volume de dados resulta em um controle mais preciso. Entretanto, essa abordagem leva à poluição informacional, na qual eventos triviais e irrelevantes são registrados sem critério adequado, mascarando informações essenciais. Como consequência, as equipes de engenharia desperdiçam tempo filtrando dados em vez de solucionarem problemas de forma objetiva. Além disso, a coleta e o armazenamento desnecessários elevam os custos operacionais e podem comprometer a escalabilidade do sistema.

Um exemplo prático desse problema pode ser observado em plataformas de *e-commerce* que registram *logs* detalhados de cada operação com o objetivo de garantir rastreabilidade, mas acabam gerando um volume excessivo de informações. Cada ação do usuário, desde a adição de itens ao carrinho até a conclusão do pagamento, gera múltiplas entradas, muitas delas redundantes. Durante períodos de alto tráfego, como a *Black Friday*, o armazenamento dessas informações torna-se um gargalo, elevando custos e reduzindo o desempenho da plataforma. Erros críticos acabam ocultos entre registros irrelevantes, dificultando a identificação e a resolução de problemas. Como resultado, a equipe de suporte enfrenta dificuldades para diagnosticar falhas, tornando a resposta a incidentes mais lenta e comprometendo a eficiência do sistema.

A observabilidade eficaz não se resume apenas à coleta de dados, mas à capacidade de interpretar essas informações de maneira precisa e contextualizada. No entanto, um dos

APs mais comuns nesse processo é a ausência de contexto[7] nos registros de *logs*, métricas e *traces*, comprometendo a utilidade das informações coletadas (USMAN *et al.*, 2022). A falta de contexto nos dados de observabilidade gera dificuldades significativas na detecção e análise de incidentes. Em muitos casos, *logs* são gerados sem identificadores essenciais, como IDs de requisição, identificadores de usuário ou detalhes sobre a origem da chamada. Como resultado, quando uma falha ocorre, as equipes responsáveis pelo suporte técnico enfrentam obstáculos para correlacionar eventos dispersos, tornando a investigação mais lenta e menos eficiente. A ausência de um encadeamento claro entre métricas e *traces* também compromete a análise de desempenho, dificultando a identificação das causas raízes dos problemas. Assim, em vez de proporcionar visibilidade clara sobre o funcionamento do sistema, a observabilidade deficiente gera um volume de dados fragmentado e de difícil interpretação.

A ausência de contexto pode ser exemplificado em um sistema de pagamentos *online* que registra falhas nas transações sem incluir informações cruciais, como o identificador do pedido, a origem da requisição ou o status das tentativas anteriores. Quando uma transação falha, o *log* simplesmente informa que o pagamento não foi concluído, sem detalhar se o erro ocorreu por uma falha na comunicação com a operadora, por saldo insuficiente do usuário ou por um problema interno na aplicação. Sem esses dados adicionais, a equipe responsável precisa buscar manualmente outras fontes de informação para compreender o ocorrido, tornando o processo de diagnóstico e correção mais demorado. Em cenários de alto volume de transações, como datas promocionais, essa falta de contexto pode resultar em atrasos operacionais e prejuízos financeiros significativos.

Além disso, a falta de integração entre os diferentes tipos de dados de observabilidade, tais como *logs*, métricas e *traces*, é outro AP comum (USMAN *et al.*, 2022; NIEDERMAIER *et al.*, 2019). Quando esses elementos são tratados de maneira isolada e sem uma correlação clara entre eles, cria-se um fragmento de visibilidade, no qual as equipes de engenharia têm que recorrer a várias ferramentas para tentar entender a totalidade do comportamento do sistema. Por exemplo, os *logs* podem indicar um erro, mas sem a métrica de desempenho ou o *trace* adequado, pode ser impossível entender se esse erro é um reflexo de uma sobrecarga do sistema ou de uma falha em uma interação com outro serviço. A falta de um padrão unificado para coletar e interpretar esses dados torna a análise de incidentes muito mais difícil, além de aumentar o tempo de resolução e a probabilidade de cometer erros ao tentar fazer correlações manuais.

[7] Conjunto de metadados que tornam os dados observáveis mais úteis, correlacionáveis e interpretáveis.

Um exemplo prático do AP pode ser dado a partir de um aplicativo de *delivery* que começou a apresentar falhas recorrentes nos momentos mais críticos. O sistema registrava mensagens genéricas de erro nos *logs*, enquanto as métricas apenas indicavam um aumento no consumo de recursos. Sem um rastreamento completo das transações, a equipe passou horas investigando componentes errados, reiniciando serviços e analisando bancos de dados desnecessariamente. Quando o problema foi finalmente identificado, descobriu-se que vinha de um serviço secundário que não estava sendo monitorado de forma adequada. Se houvesse uma solução de observabilidade integrada, mostrando claramente o fluxo completo das requisições, o diagnóstico teria sido imediato e a resolução, rápida.

No nível operacional, um dos maiores erros que pode ser cometido é a falta de alinhamento entre os alertas e os objetivos de serviço (YANG *et al.*, 2022). O sistema pode gerar alertas a todo momento, mas se esses alertas não são baseados em metas claras de desempenho ou indicadores de serviço, eles acabam se tornando um ruído, sem valor real para a equipe. Uma grande quantidade de alertas irrelevantes leva à fadiga dos engenheiros, que acabam ignorando notificações importantes por já estarem acostumados com a constante avalanche de alarmes. O problema se agrava quando os alertas são configurados sem uma análise detalhada do impacto real que uma falha pode ter no serviço. Nesse cenário, alertas podem ser acionados para problemas que, na prática, não afetam a experiência do usuário, enquanto problemas realmente críticos ficam ocultos, sem o devido destaque.

Um exemplo prático ocorre, por exemplo, quando uma equipe de SRE configura alertas para cada pequena variação na utilização de CPU em servidores individuais, gerando um volume massivo de notificações. Como essas flutuações são normais e nem sempre indicam degradação real do serviço, os engenheiros passam a desconsiderar os avisos. Quando, finalmente, um incidente grave compromete a experiência dos usuários, a falha não é detectada de imediato, pois o alerta crítico se perde em meio a um histórico de notificações irrelevantes. Isso retarda a resposta ao problema e impacta negativamente a confiabilidade do sistema.

O excesso de *dashboards* em uma organização pode comprometer significativamente a eficácia do monitoramento e a resolução de problemas, sendo considerado um AP (VILLELA, 2022). Ele ocorre quando equipes criam múltiplos *dashboards* para propósitos semelhantes, sem uma estratégia clara de governança ou consolidação. A falta de um controle centralizado leva à proliferação de painéis redundantes, desatualizados e, muitas vezes, inconsistentes. Como consequência, os usuários enfrentam dificuldades para identificar quais *dashboards* são confiáveis

e relevantes, tornando o processo de análise de métricas confuso e ineficiente.

Esse AP pode ser exemplificado quando uma equipe de SRE precisa investigar um problema de desempenho e encontra diversos *dashboards* com nomes semelhantes, cada um exibindo métricas ligeiramente diferentes para o mesmo serviço. Incapazes de identificar rapidamente qual painel apresenta os dados mais precisos, os engenheiros perdem tempo valioso analisando informações conflitantes, retardando a identificação da causa raiz do problema. Em momentos críticos, essa ineficiência pode resultar em falhas prolongadas e impactos negativos para os usuários finais.

Assim, a implementação inadequada da observabilidade cria uma falsa sensação de controle, na qual os dados disponíveis parecem fornecer visibilidade sobre o sistema, mas na prática, ocultam falhas ou geram diagnósticos imprecisos. A verdadeira observabilidade não se trata de coletar mais dados, mas de coletar dados relevantes, úteis e bem estruturados, que realmente permitam à equipe entender e controlar o comportamento do sistema. Sem uma abordagem bem planejada e uma visão holística dos dados, a observabilidade deixa de ser uma ferramenta eficiente e se transforma em mais um desafio a ser gerido.

2.4 Considerações Finais

Esse capítulo apresentou a fundamentação teórica que sustenta esta pesquisa, abordando conceitos essenciais para a compreensão do estudo. Inicialmente, foram discutidos os microsserviços, evidenciando suas vantagens, como flexibilidade, escalabilidade e resiliência, bem como desafios relacionados à comunicação, gerenciamento de dados e complexidade do desenvolvimento. Em seguida, foram exploradas as tecnologias que suportam esse estilo arquitetural, incluindo contêineres, *Docker* e *Kubernetes*, destacando como essas ferramentas facilitam a implantação, a orquestração e a operação de serviços distribuídos.

Na sequência, abordou-se a observabilidade, diferenciando-a do monitoramento e detalhando seus principais pilares: métricas, *logs* e *traces*. Foram apresentados os benefícios de um ecossistema observável, como aumento da visibilidade, detecção precoce de problemas, suporte a processos de DevOps, escalabilidade e melhoria da experiência do usuário, bem como o uso das ferramentas de observabilidade como *Prometheus*, *Jaeger*, *Elastic Stack* e *OTel*.

Por fim, a discussão sobre APs de observabilidade evidenciou como práticas inadequadas podem gerar excesso de dados, ausência de contexto, fragmentação de informações, alertas irrelevantes e proliferação de *dashboards*, comprometendo a capacidade de análise, di-

agnóstico e resposta a incidentes. Esses exemplos reforçam a necessidade de uma abordagem planejada, que priorize a coleta de dados relevantes e sua correlação adequada, garantindo que a observabilidade cumpra seu papel de fornecer compreensão sobre o comportamento do sistema.

O conhecimento apresentado neste capítulo fornece a base para os trabalhos relacionados, discutidos como estudos secundários no Capítulo 3, com foco em monitoramento e observabilidade. Esses estudos permitem identificar práticas consolidadas, lacunas na literatura e oportunidades para aprofundar a compreensão e propor soluções para APs de observabilidade em microsserviços.

3 TRABALHOS RELACIONADOS

Este capítulo apresenta os trabalhos relacionados a esta pesquisa, com ênfase em estudos secundários que analisam o estado da arte em monitoramento e observabilidade de sistemas de software. A priorização desses estudos decorre do objetivo central desta tese, que consiste em investigar e sistematizar o conhecimento existente sobre observabilidade, bem como identificar e consolidar evidências relacionadas a anti-padrões nesse contexto.

Uma pesquisa preliminar foi realizada para identificar tais estudos, utilizando o *Google Scholar*[1] para localizar revisões, mapeamentos sistemáticos e *surveys*. Essa revisão inicial evidenciou a ausência de uma taxonomia como a que aqui se propõe, bem como a falta de informações consolidadas sobre ferramentas, conjuntos de dados e aplicações empregadas em avaliações, além da inexistência de estudos que mapeiem Anti-Padrões (APs) de observabilidade, que são fatores que motivaram o desenvolvimento desta tese. A revisão conduzida ao longo do trabalho complementou essa etapa preliminar e fundamenta a seleção dos estudos relacionados que serão apresentados.

Assim, embora o foco esteja direcionado a aplicações baseadas em microsserviços, os estudos considerados abrangem contextos mais amplos. Em vez de priorizar pesquisas primárias que apresentam soluções específicas, optou-se por destacar investigações de caráter secundário, por fornecerem uma visão consolidada da área.

Logo, para apresentar a pesquisa realizada, a Seção 3.1 apresenta trabalhos que tratam do monitoramento em um contexto mais amplo, destacando investigações sobre ferramentas e desafios associados. Já a Seção 3.2 concentra-se em estudos voltados à observabilidade, abrangendo seus fundamentos, práticas, ferramentas e os principais desafios relacionados ao tema.

3.1 Monitoramento

Costa *et al.* (2022) propõem uma taxonomia para soluções de monitoramento em *Fog*, apoiada por uma revisão sistemática da literatura, e destacam os principais desafios e questões de pesquisa abertas na área. Os autores afirmam que o monitoramento é uma funcionalidade fundamental de orquestração e é a base para ações de gerenciamento de recursos, coletando o *status* de recursos e serviços e fornecendo dados atualizados para o orquestrador. Além disso,

[1] <https://scholar.google.com.br/>

observam que, embora existam várias soluções e ferramentas de monitoramento na nuvem, nenhuma delas aborda completamente as características e os desafios dos ambientes *Fog*. As soluções de monitoramento em computação *Fog* são escassas e podem não estar prontas para compor um serviço de orquestração. Eles discutem tópicos recentes neste contexto, como observabilidade, padronização de dados e domínios de instrumentação.

O estudo de Costa *et al.* (2022) serviu como referência inicial para a construção da taxonomia proposta nesta tese, da qual foram selecionadas classificações aplicáveis ao contexto de observabilidade, como propósito, domínios e integração das soluções (detalhados na Seção 4.2). No entanto, o estudo não contempla um levantamento das más práticas de monitoramento, que podem impactar a compreensão do comportamento das aplicações.

Janes *et al.* (2023) apresentam uma visão geral das ferramentas de rastreamento *open-source* populares no mercado por meio de uma comparação crítica. Eles identificaram trinta ferramentas usando um protocolo de Revisão Sistemática Multivocal e realizaram sua classificação. As características incluem informações gerais, como licença e linguagem de programação, implantação, uso, dados coletados e interoperabilidade, destacando aspectos importantes para garantir um alto grau de operabilidade, como a disponibilidade de API e suporte a OTel.

O estudo de Janes *et al.* (2023) serviu como referência para identificar as ferramentas de rastreamento atualmente disponíveis. Entretanto, não é indicado quais dessas ferramentas são efetivamente empregadas em estudos acadêmicos, informação que poderia ser obtida por meio de um mapeamento sistemático. Esse mapeamento é importante porque permite identificar quais delas são efetivamente utilizadas na prática científica, indicando seu grau de consolidação e relevância na área. Além de evidenciar lacunas e tendências na adoção e no desenvolvimento dessas ferramentas no contexto acadêmico, esse levantamento contribui para comparabilidade entre estudos e tomada de decisão mais fundamentada por parte dos pesquisadores.

Giamattei *et al.* (2023) apresentam os resultados de uma revisão da literatura cinza conduzida para identificar, classificar e analisar ferramentas de monitoramento disponíveis para DevOps e microsserviços. O estudo teve como objetivo fornecer uma visão geral das opções de ferramentas disponíveis e analisar os esforços, desafios e direções em andamento para oferecer melhores ferramentas. Ao todo, setenta e uma ferramentas foram analisadas, e um mapa das principais características das ferramentas existentes foi desenvolvido, incluindo seus benefícios e desvantagens, pressupostos para seu uso, as informações que coletam, as técnicas usadas para

coleta eficiente de dados, como apresentam resultados e várias outras características.

O estudo de Giamattei *et al.* (2023) serviu como referência para identificar ferramentas de monitoramento atualmente disponíveis. Entretanto, assim como observado em Janes *et al.* (2023), não há investigação sobre o uso efetivo dessas ferramentas no contexto acadêmico, informação que poderia ser obtida por meio de um mapeamento sistemático.

3.2 Observabilidade

Niedermaier *et al.* (2019) apresentam um estudo qualitativo para entender os desafios e as melhores práticas no campo de observabilidade e monitoramento de sistemas distribuídos. Eles realizaram 28 entrevistas semiestruturadas na indústria com profissionais de software envolvidos com monitoramento, incluindo gerentes de serviço, engenheiros de DevOps, fornecedores de software e consultores. A partir dessas entrevistas, foram extraídos desafios, requisitos e soluções. Eles encontraram crescente complexidade e dinamicidade neste campo. A observabilidade está se tornando um pré-requisito essencial para garantir serviços estáveis e o desenvolvimento contínuo das aplicações clientes. No entanto, os participantes mencionaram uma discrepância na conscientização sobre a importância do tema tanto na perspectiva da gestão quanto dos desenvolvedores. Além dos desafios técnicos, eles identificaram a necessidade de um conceito organizacional incluindo estratégia, papéis e responsabilidades para os profissionais no desenvolvimento sistemático e implementação da observabilidade e monitoramento de sistemas distribuídos.

Todavia, enquanto o estudo de Niedermaier *et al.* (2019) enfatiza a visão prática da indústria, apresentando boas práticas para observabilidade, não são abordadas as más práticas enfrentadas pelos profissionais, nem apresentadas soluções para mitigá-las.

Li *et al.* (2022) realizaram uma pesquisa sobre microsserviços e observabilidade, destacando a complexidade dos sistemas industriais de microsserviços que operam em infraestruturas complexas na nuvem, com instâncias de serviço criadas e destruídas dinamicamente. Os autores destacam que o desafio das aplicações baseadas em microsserviços está em entender e diagnosticar problemas dentro dessa arquitetura, como falhas em requisições e alta latência, devido ao envolvimento de numerosos serviços e a um ambiente de execução diverso. A pesquisa destaca que rastrear e analisar microsserviços introduz um novo grande desafio de *Big data* para a engenharia de software, apresentando oportunidades e desafios como amostragem adaptativa de *logs*, fusão de dados para análise de *traces*, análise mais inteligente de *traces* e inteligência de

negócios por meio da análise de *traces*.

O estudo de Li *et al.* (2022) concentra-se na caracterização dos desafios técnicos relacionados ao rastreamento e à análise de microsserviços, com base em entrevistas, enfatizando boas práticas. No entanto, não são discutidas as más práticas de observabilidade enfrentadas pelos profissionais.

Usman *et al.* (2022) realizaram uma pesquisa abrangente sobre ambientes distribuídos de TI e observabilidade de microsserviços, com o objetivo de explorar desafios, requisitos, melhores práticas e soluções atuais para observar sistemas distribuídos. Os pesquisadores compilaram estudos e artigos relevantes, apresentando diferentes tipos de dados de telemetria cruciais para a resolução de problemas operacionais. O artigo identifica e discute funcionalidades essenciais para a observabilidade, abordando questões de pesquisa abertas relacionadas a infraestruturas heterogêneas e arquiteturas de microsserviços em distintos casos de uso. Também são discutidos desafios relacionados ao grande volume de dados gerados pela observabilidade e à necessidade de uma plataforma unificada capaz de coletar, processar e analisar esses dados. Além disso, o estudo evidencia a influência da cultura organizacional e das mentalidades individuais na adoção de novas ferramentas e práticas, assim como preocupações com segurança e conformidade.

Todavia, enquanto o estudo de Usman *et al.* (2022) fornece uma visão consolidada da observabilidade por meio de um *survey*, destacando boas práticas nesse contexto, não realiza uma análise estruturada e sistematizada dos principais problemas enfrentados na utilização da observabilidade.

Kosińska *et al.* (2023) apresentam um mapeamento sistemático da observabilidade de aplicações nativas da nuvem, que consiste em um estudo sobre abordagens de engenharia de observabilidade, maturidade, eficiência, ferramentas e direções de pesquisa futuras. Ao todo, cinquenta e seis estudos principais dos últimos cinco anos foram analisados. As principais áreas de pesquisa dos trabalhos são tradicionalmente computação em nuvem e distribuída, mas novas áreas, incluindo *IoT*, *Edge* e *Fog Computing*, também são abordadas. Uma observação importante é a relação dos estudos de observabilidade com *Big Data*, Segurança e temas de *Machine Learning* (ML)/Inteligência Artificial (AI). Além das áreas, outras classificações foram feitas, onde aspectos como requisitos funcionais, requisitos não funcionais, técnicas de conteinerização, orquestradores, provedores de infraestrutura como serviço, software de observabilidade, ferramentas específicas de observação de infraestrutura usadas e métodos de armazenamento de dados podem ser destacados.

Apesar das classificações realizadas por Kosińska *et al.* (2023), o estudo não apresenta uma taxonomia de observabilidade, nem realiza levantamento dos conjuntos de dados e das aplicações utilizadas nos trabalhos analisados.

3.3 Comparação entre os Trabalhos Relacionados

A Tabela 1 apresenta uma comparação dos estudos mencionados. Pode-se observar que todos os estudos apresentaram algum tipo de classificação, tanto em monitoramento quanto em observabilidade. No entanto, nenhum deles apresentou uma taxonomia e um catálogo de APs de observabilidade para aplicações baseadas em microsserviços. Além disso, nenhum deles apresenta as aplicações *benchmarking* ou conjunto de dados de *workload* do mundo real utilizados em observabilidade.

Tabela 1 – Comparação dos trabalhos relacionados.

Estudo	**Ano**	**Contexto**	**Critérios**	**Tipo de Estudo**
Niedermaier *et al.*	2019	Observabilidade e Monitoramento de Sistemas Distribuídos	Desafios/ Requisitos/ Soluções	Entrevistas na Indústria
Costa *et al.*	2022	Monitoramento em *Fog*	Contexto/ Taxonomia	Revisão Sistemática
Usman *et al.*	2022	Observabilidade de Microsserviços Baseados em *Edge* e Contêineres	Contexto/ Ferramentas	*Survey*
Li *et al.*	2022	Rastreamento e Análise de Microsserviços	Contexto/ Desafios	Pesquisa na Indústria
Janes *et al.*	2023	Ferramentas de Rastreamento *Open-Source*	Contexto/ Soluções	Revisão Sistemática
Giamattei *et al.*	2023	Ferramentas de Monitoramento	Contexto/ Soluções	Revisão Sistemática
Kosinska *et al.*	2023	Observabilidade de CNA	Contexto/ Requisitos/ Ferramentas	Mapeamento Sistemático
Tese	**2026**	**Observabilidade em Microsserviços**	**Contexto/ Taxonomia/ Catálogo de APs/ Ferramentas/ Aplicações de *Benchmarking*/ *Datasets*/ *Framework* de Detecção**	**Mapeamento Sistemático e Literatura Cinzenta**

Fonte: Autoria própria.

3.4 Considerações Finais

Em síntese, os trabalhos relacionados analisados neste capítulo fornecem importantes referências sobre monitoramento e observabilidade, abrangendo desde classificações de ferramentas até mapeamentos sistemáticos e estudos empíricos sobre desafios e boas práticas. No entanto, observa-se que tais investigações tendem a enfatizar aspectos positivos, como requisitos, funcionalidades e soluções, deixando em aberto a sistematização das más práticas que comprometem a efetividade da observabilidade em sistemas distribuídos.

Além disso, não foi identificada nenhuma solução capaz de detectar automaticamente APs de observabilidade. Trabalhos existentes concentram-se em outros níveis de análise ou tratam apenas sintomas específicos do problema. Por exemplo, Monsef e Ashtiani (2026) propõem uma ferramenta para detecção automática de APs arquiteturais em microsserviços, utilizando *Graph Neural Networks* (GNNs) e *Large Language Models* (LLMs) para modelar a arquitetura como grafos e identificar padrões como *cyclic dependencies, enterprise service bus (ESB) usage, microservice greediness* e *inappropriate service intimacy*. Embora relevante para a qualidade arquitetural, essa abordagem foca apenas nos microsserviços e não na observabilidade. De forma complementar, Min *et al.* (2023) investigam a fadiga de alertas sob a perspectiva de sistemas de detecção de anomalias, propondo estratégias de otimização de limiares e agregação de notificações para reduzir falsos positivos e sobrecarga operacional. Contudo, essa linha de trabalho foca apenas em um problema, sem a detecção automática de APs possíveis.

Nesse sentido, a presente tese diferencia-se ao reunir, de forma estruturada, uma taxonomia de classificação e um catálogo de APs de observabilidade, constituindo, até onde se tem conhecimento, o primeiro esforço integrado nesse domínio, com foco particular em aplicações baseadas em microsserviços. De forma complementar, também são apresentados os principais conjuntos de ferramentas, aplicações *benchmarking* e conjunto de dados de *workload* do mundo real utilizados nos estudos analisados. Por fim, a tese apresenta um *framework* para detecção automática de APs de observabilidade.

O próximo capítulo apresenta a metodologia de pesquisa adotada para a construção da taxonomia, detalhando as etapas do mapeamento sistemático e a análise realizada. Em seguida, são descritas a taxonomia propriamente dita, com suas categorias e a classificação dos estudos selecionados, bem como o levantamento de soluções utilizadas nas experimentações dos estudos analisados.

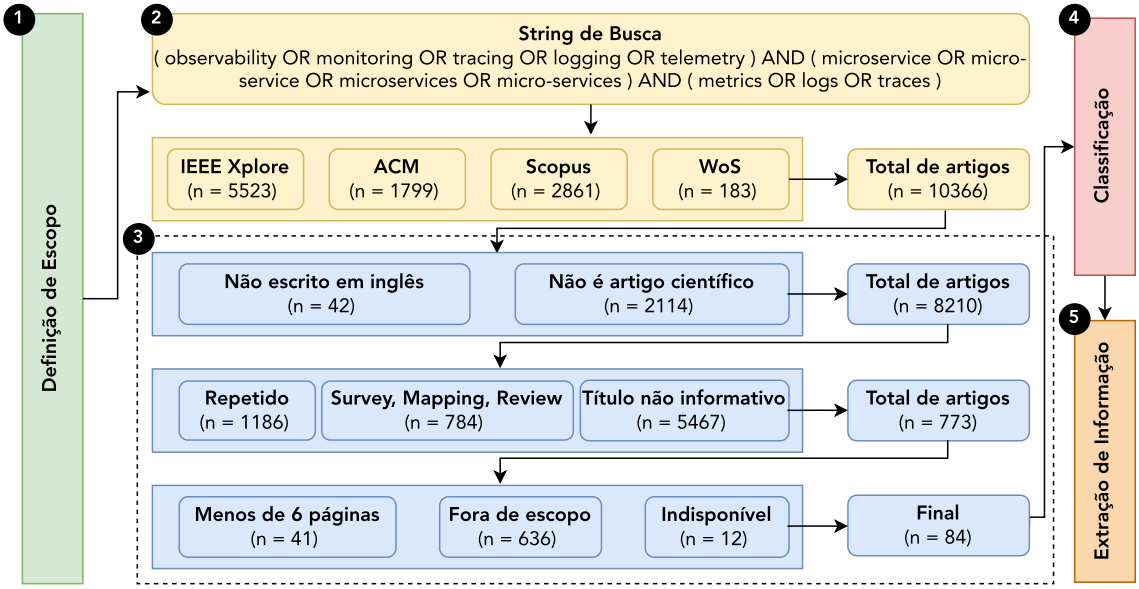
4 UMA TAXONOMIA DE OBSERVABILIDADE

Este capítulo apresenta a taxonomia de observabilidade para aplicações baseadas em microsserviços (GOMES *et al.*, 2025b). Na Seção 4.1, descreve-se a metodologia adotada para a condução do mapeamento sistemático. A Seção 4.2 oferece uma visão geral da taxonomia desenvolvida, destacando as categorias identificadas a partir do mapeamento. Em seguida, a Seção 4.3 detalha a classificação dos estudos selecionados, estruturada de acordo com a taxonomia desenvolvida. A Seção 4.4 apresenta as ferramentas, as aplicações *benchmarking* e o conjunto de dados empregados nos estudos incluídos. Por fim, a Seção 4.5 discute as potenciais ameaças à validade da taxonomia.

4.1 Metodologia do Mapeamento Sistemático da Literatura

Para a construção da taxonomia de observabilidade foi realizado um Mapeamento Sistemático da Literatura (MSL), seguindo os princípios detalhados em (PETERSEN *et al.*, 2008). Assim, a metodologia, ilustrada na Figura 10, tem as seguintes etapas: (1) definição do escopo, (2) busca, (3) filtragem, (4) classificação e (5) extração de informações, culminando na apresentação dos resultados. O processo de revisão foi realizado entre março de 2024 e novembro de 2024, para complementar os resultados, foi realizado novamente no mês de junho de 2025.

Figura 10 – Etapas da metodologia da taxonomia.



Fonte: Autoria própria.

4.1.1 Definição do Escopo

A pesquisa foi conduzida em conformidade com as diretrizes de um MSL. Inicialmente, definiu-se o escopo do estudo com base nas questões de pesquisa **QP1** e **QP2**, que orientaram todo o processo e delimitaram a área de investigação a ser mapeada. Esse passo foi precedido por uma discussão que buscou consenso sobre os temas mais relevantes para a tese, em particular aqueles relacionados à "observabilidade em microsserviços". Em seguida, elaborou-se a *string* de busca, utilizada para a seleção sistemática dos artigos que compuseram a base de análise.

4.1.2 Busca

Nesta etapa, o processo de busca foi conduzido de acordo com as orientações de um MSL, envolvendo a aplicação de uma *string* de busca em quatro mecanismos amplamente reconhecidos pela comunidade acadêmica: *IEEE Xplore*, *ACM Digital Library*, *Scopus* e *Web of Science (WoS)*. A adoção de múltiplas bases garantiu uma cobertura abrangente, contemplando tanto artigos de conferência (*IEEE Xplore* e *ACM Digital Library*) quanto periódicos científicos de grande impacto (*Scopus* e *WoS*).

A definição da *string* de busca foi cuidadosamente elaborada a partir da análise de termos diretamente associados ao tema da tese, centrado na observabilidade em microsserviços. O termo "*Observability*" foi considerado essencial para capturar o conceito central, sendo complementado por sinônimos e termos correlatos, como "*Monitoring*", "*Tracing*" e "*Logging*", de modo a ampliar o espectro de trabalhos recuperados. Incluiu-se também o termo "*Telemetry*", dada a sua estreita relação com práticas de coleta, transmissão e análise de dados no contexto da observabilidade. Além disso, a presença da palavra "*Microservice*", em suas diferentes variações, foi considerada obrigatória, enquanto termos como "*Metrics*", "*Logs*" e "*Traces*" foram acrescentados por representarem os três pilares fundamentais do conceito em questão.

O resultado desse processo foi a construção de uma *string* final de busca, a qual foi, posteriormente, utilizada em cada uma das bases selecionadas. Nos mecanismos *Scopus* e *WoS*, a *string* foi executada diretamente, ao passo que, no *IEEE Xplore* e no *ACM Digital Library*, os termos entre parênteses foram aplicados separadamente. Em todos os casos, empregou-se a opção de busca avançada, a fim de assegurar que os trabalhos recuperados contemplassem o texto completo e refletissem, de maneira fidedigna, o escopo previamente definido.

4.1.3 Filtragem

A etapa de filtragem foi conduzida de forma sistemática, com base em critérios de exclusão previamente definidos. Foram eliminados os trabalhos que: (i) não estavam redigidos em inglês; (ii) não se caracterizavam como artigos científicos, como pôsteres, tutoriais, palestras principais, livros ou cursos; (iii) correspondiam a estudos secundários, como revisões, mapeamentos ou *surveys*; (iv) apresentavam-se fora do escopo da pesquisa a partir da análise do título; ou (v) possuíam menos de seis páginas, sendo classificados como artigos curtos. Apenas os trabalhos que não se enquadraram em nenhum desses critérios foram mantidos para as etapas subsequentes.

Complementarmente, os critérios de inclusão exigiram que os estudos selecionados estivessem disponíveis em texto completo e que tivessem sido publicados no período compreendido entre janeiro de 2019 e maio de 2025. Essa estratégia visou garantir que apenas estudos primários, relevantes e suficientemente detalhados fossem considerados no mapeamento.

O processo de filtragem está representado no fluxograma da Figura 10, que apresenta o número inicial de trabalhos recuperados (10.366 no total), e a quantidade de artigos excluídos em cada fase. Na primeira etapa, aplicaram-se os critérios (i) e (ii), resultando na exclusão de 2.156 estudos. A segunda etapa concentrou-se na remoção de trabalhos duplicados e na aplicação dos critérios (iii) e (iv), o que levou à eliminação de 7.437 registros. Em seguida, a terceira etapa avaliou os 773 trabalhos remanescentes, considerando o critério (v), a disponibilidade de acesso ao texto completo e a pertinência temática após a leitura integral. Como resultado desse processo progressivo, obteve-se um conjunto final de 84 artigos, que constituíram a base para a etapa de classificação.

4.1.4 Classificação

Com base na leitura dos estudos selecionados, foram extraídas categorias para classificar os estudos e, assim, obter informações mais precisas. O processo de construção da taxonomia foi conduzido em duas fases complementares. Na primeira, estabeleceu-se a estrutura inicial por meio da definição das categorias principais e da identificação da necessidade de desdobrá-las em subcategorias. Na segunda fase, procedeu-se à atribuição das classificações correspondentes a cada uma dessas categorias e subcategorias, de forma a refinar e consolidar a organização proposta. Como resultado, obteve-se a taxonomia de observabilidade voltada

especificamente para aplicações baseadas em microsserviços, que servirá de referência para as análises subsequentes (ver Seção 4.2).

4.1.5 Extração de Informações

Por fim, essa etapa correspondeu ao processo de extração de informações, conduzido por meio da leitura integral dos estudos selecionados e de seu mapeamento segundo as classificações previamente estabelecidas. O autor do documento e os orientadores participaram ativamente dessa fase, de modo colaborativo. O autor do documento assumiu a responsabilidade pela leitura completa de cada estudo e pela sistematização inicial das evidências, enquanto os demais acompanharam os achados, contribuindo para a definição das categorias e subcategorias, bem como para a adequada classificação e posicionamento dos estudos no escopo da taxonomia. Posteriormente, o autor do documento também redigiu o manuscrito e consolidou os resultados obtidos, que foram revisados criticamente pelos orientadores. O processo metodológico e as classificações resultantes encontram-se sintetizados no *link* disponibilizado: <https://doi.org/10.5281/zenodo.17108406>.

4.2 Visão Detalhada da Taxonomia

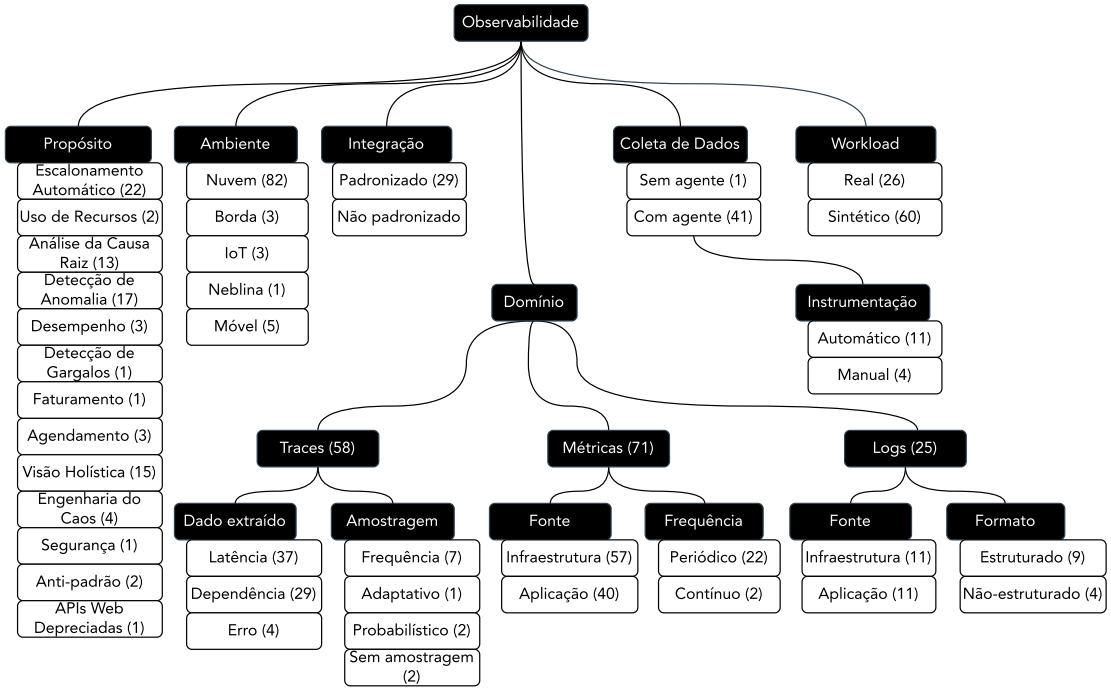
A partir do MSL realizado e descrito na seção anterior, foi possível propor uma taxonomia de observabilidade para aplicações baseadas em microsserviços, como ilustrado na Figura 11. Esta taxonomia busca consolidar os principais domínios e categorias presentes no estado da arte, que são detalhados nas subseções seguintes.

O desenvolvimento da taxonomia ocorreu de maneira progressiva e estruturada, contemplando diferentes etapas metodológicas. Inicialmente, tomou-se como referência o trabalho de Costa *et al.* (2022), o qual serviu de base para a leitura integral dos 84 estudos selecionados. A partir dessa análise, foram identificadas as primeiras categorias, relacionadas a propósito (Seção 4.2.1), domínios (Seção 4.2.3), *logs* (Seção 4.2.4), *traces* (Seção 4.2.5), métricas (Seção 4.2.6) e integração (Seção 4.2.7).

Em um segundo momento, considerando os resultados preliminares, os orientadores recomendaram uma reanálise detalhada dos estudos, com o objetivo de identificar elementos experimentais relevantes. Essa etapa levou à incorporação de três novas categorias: ambiente (Seção 4.2.2), coleta de dados (Seção 4.2.8) e *workload* (Seção 4.2.9).

Na sequência, verificou-se a necessidade de maior refinamento, o que motivou a subdivisão de algumas categorias em subcategorias específicas, de modo a captar nuances presentes nos estudos. Por fim, a classificação das categorias e subcategorias foi realizada de forma iterativa, com base em sua ocorrência nos artigos analisados, sendo constantemente discutida e validada entre os orientadores a fim de assegurar consistência e rigor metodológico. O número de ocorrências de cada categorização está indicado entre parênteses ao longo das seções correspondentes e na Figura 11.

Figura 11 – Taxonomia de observabilidade.



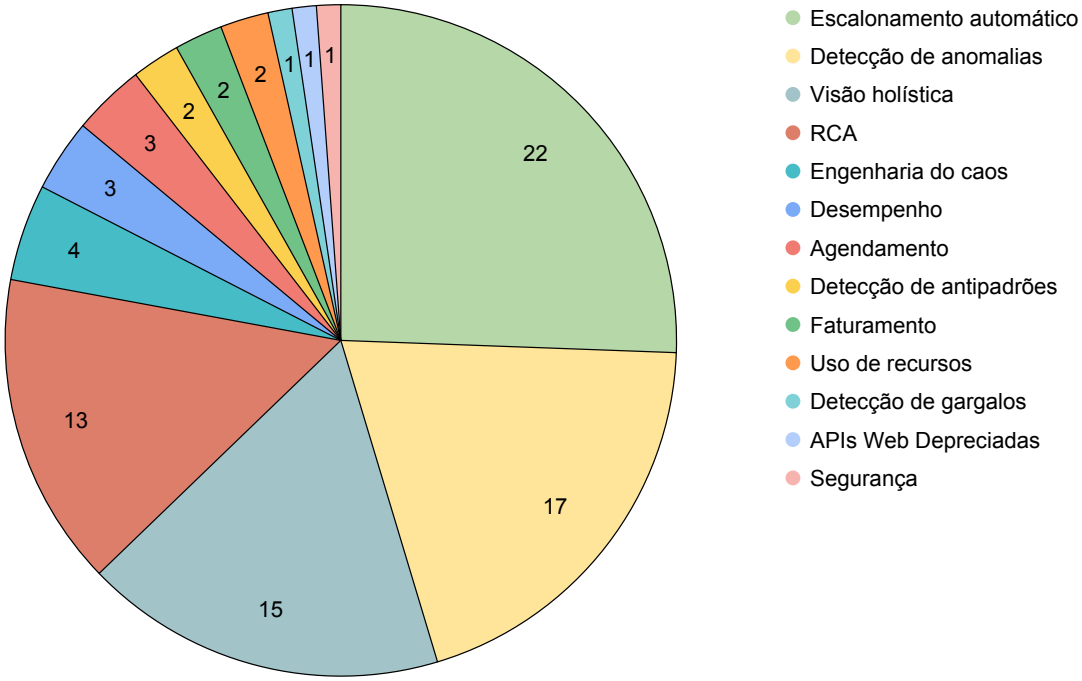
Fonte: Autoria própria.

4.2.1 Propósito

Esta primeira categoria define os objetivos de uma solução de observabilidade. Diversos propósitos foram identificados na literatura e são ilustrados na Figura 12, que apresenta a quantidade de trabalhos de observabilidade que os abordam.

As soluções de observabilidade podem rastrear métricas de **uso de recursos** (2), como CPU, memória, disco e utilização da rede, para garantir a alocação eficiente de recursos e identificar possíveis gargalos de desempenho. Ao analisar métricas e tendências de desempenho, elas também podem automatizar o processo de **escalonamento automático (EA)** (22), ajustando dinamicamente o provisionamento de recursos com base na demanda para otimizar tanto o

Figura 12 – Propósitos da observabilidade.



Fonte: Autoria própria.

desempenho do sistema quanto a eficiência de custos. Além disso, essas soluções ajudam a otimizar o **agendamento** (3) de recursos e a distribuição de carga. Algoritmos de **detecção de anomalias** (17) podem ser empregados para identificar padrões incomuns ou desvios do comportamento normal. As soluções de observabilidade também fornecem visibilidade detalhada dos componentes e dependências do sistema, facilitando a **análise de causa raiz (RCA)** (13) ao rastrear eventos e identificar fatores subjacentes que contribuem para a degradação do desempenho ou falhas. Além disso, ao identificar **gargalos** (1) de desempenho e restrições de recursos, elas permitem a otimização da arquitetura e configuração do sistema para melhorar a eficiência e escalabilidade geral.

Monitorar e analisar métricas de **desempenho** (3) é outra função crucial das soluções de observabilidade, garantindo que os sistemas atendam aos SLOs e metas de desempenho predefinidas, como tempo de resposta, taxa de transferência e taxas de erro. Essas soluções também podem rastrear métricas de uso de recursos para oferecer observações sobre alocação de custos e **faturamento** (1), ajudando as organizações a otimizar a utilização de recursos e gerenciar efetivamente os gastos em nuvem. Ao agregar e analisar dados de várias fontes, a **visão holística (VS)** (15) na observabilidade fornece uma perspectiva integrada de todos os componentes do sistema, permitindo uma compreensão clara de suas interações e impacto geral.

Além disso, as soluções de observabilidade podem apoiar práticas de **Engenharia do Caos** (4) ao fornecer visibilidade do comportamento do sistema durante cenários controlados de falha. Isso permite que as organizações avaliem a resiliência, identifiquem vulnerabilidades e melhorem a confiabilidade do sistema.

Além disso, a **Detecção de Anti-Padrões** (APs) (2) em microsserviços envolve a identificação de práticas prejudiciais, como acoplamento excessivo de serviços, dependência excessiva de comunicação síncrona e gerenciamento inadequado da consistência de dados, que podem prejudicar a escalabilidade e o desempenho. Da mesma forma, o monitoramento de **APIs Web Depreciadas** (1) é crucial, pois seu uso contínuo pode levar a problemas de compatibilidade e dificultar futuras atualizações do sistema. Por fim, capacidades de **segurança** (1) são frequentemente incluídas, permitindo a detecção e mitigação de ameaças de segurança, o rastreamento de eventos de acesso e autorização, e a garantia de conformidade com políticas e regulamentos de segurança.

4.2.2 Ambiente

A observabilidade pode ser aplicada a vários ambientes onde os microsserviços podem ser implantados, exigindo abordagens personalizadas e ferramentas especializadas para abordar as características e desafios únicos de cada ambiente, como **Nuvem** (82), **Borda** (3), **Neblina** (1), **IoT** (3) e **Móvel** (5).

Essas classificações de ambiente são baseadas em menções explícitas nos estudos selecionados, tais como (KUMAR *et al.*, 2023), (COSTA *et al.*, 2024), (TZANETTIS *et al.*, 2022) e (KHICHANE *et al.*, 2023).

4.2.3 Domínio

Como discutido na Seção 2.2.3, a observabilidade de sistemas distribuídos envolve três principais domínios de instrumentação: **Logs** (25), **Traces** (58) e **Métricas** (71). Verifica-se que existem muitos trabalhos sobre cada um destes aspectos, porém com uma forte tendência no uso de métricas nas pesquisas encontradas. A seguir, detalha-se cada um dos domínios encontrados.

4.2.4 Logs

Logs são registros que capturam eventos e estados do sistema. Esta seção apresenta as fontes de coleta de dados de *log* e a forma como esses registros são estruturados.

4.2.4.1 Fonte

Esta categoria refere-se ao tipo de *log* utilizado nos estudos analisados. Os *logs* de infraestrutura (11) registram eventos em nível de sistema, como utilização de CPU, atividade de disco e tráfego de rede, oferecendo informações relevantes sobre o estado e o desempenho de recursos físicos e virtuais. Já os *logs* de aplicação (11) concentram-se em eventos internos ao software, incluindo interações dos usuários, ocorrência de erros e dados de depuração, fornecendo subsídios para a análise do comportamento funcional e para a identificação de falhas.

4.2.4.2 Formato

Os formatos de *log* identificados na literatura podem ser classificados em duas categorias principais, de acordo com a forma como os dados são organizados nos arquivos: estruturados e não estruturados. Os *logs* estruturados (9) apresentam seus registros segundo um esquema predefinido, comumente organizado em pares chave–valor ou em formatos padronizados como JSON, XML ou CSV, o que facilita a análise automatizada e a integração com ferramentas de monitoramento. Em contraste, os *logs* não estruturados (4) não seguem um padrão rígido, sendo armazenados em formato de texto livre, geralmente voltado para leitura humana, o que pode dificultar o processamento automático, mas oferece maior flexibilidade na documentação de eventos.

4.2.5 Traces

Traces representam o percurso das requisições por meio de um sistema. Esta seção aborda os dados extraídos e o processo de amostragem.

4.2.5.1 Dados Extraídos

Por meio de rastreamentos distribuídos, é possível obter observações valiosas sobre o comportamento e a saúde de uma aplicação. O rastreamento permite a medição e análise

da **latência** (37), que corresponde ao tempo que uma solicitação leva para percorrer vários componentes de um sistema distribuído, possibilitando a identificação de gargalos e pontos críticos de desempenho. Além disso, ele oferece visibilidade das **dependências** (29) entre diferentes serviços e componentes, facilitando o entendimento do fluxo de dados e controle entre eles. Outro aspecto fundamental é a detecção e o diagnóstico de **erros** (4) no sistema. Por meio dos rastreamentos, é possível identificar spans que apresentaram falhas, como exceções lançadas, códigos de status indicativos de erro (ex: HTTP 4xx ou 5xx), *timeouts* e *retries* automáticos.

4.2.5.2 Amostragem

A amostragem de rastreamento constitui uma técnica amplamente utilizada em sistemas distribuídos com o objetivo de reduzir a sobrecarga decorrente da coleta e do armazenamento de dados de rastreamento em larga escala. Em vez de registrar todos os eventos ou transações ocorridos, essa abordagem seleciona apenas um subconjunto representativo, suficiente para fins de análise. Diversas estratégias de amostragem podem ser empregadas nesse processo, cada uma com características específicas.

Na **amostragem baseada em frequência** (7), define-se um intervalo fixo para captura dos rastreamentos, como, por exemplo, registrar um a cada mil eventos. Já a **amostragem probabilística** (2) associa a cada evento uma probabilidade de ser amostrado, de modo que, ao definir a probabilidade de 0,1, cerca de 10% dos eventos são capturados em média. Por sua vez, a **amostragem adaptativa** (1) ajusta a taxa de amostragem de maneira dinâmica, em função de condições previamente estabelecidas. Nesse caso, a frequência de captura pode aumentar quando determinadas métricas, como o tráfego da aplicação, atingem limites predefinidos, permitindo maior granularidade da análise em situações críticas.

4.2.6 Métricas

Métricas são representações numéricas do comportamento do sistema ao longo do tempo. Esta seção apresenta as fontes de coleta de dados de métricas e a frequência com que são coletadas.

4.2.6.1 Fonte

Os níveis de métricas referem-se às diferentes camadas ou categorias de métricas usadas para monitorar e avaliar vários aspectos do desempenho, comportamento e saúde de um sistema. Esses níveis fornecem diferentes perspectivas e granularidade de dados, permitindo uma observabilidade abrangente e análise do sistema. As métricas de **aplicação** (40) focam no monitoramento do comportamento e desempenho do software em si. Elas incluem métricas como tempo de resposta, taxa de transferência, taxa de erro e latência. As métricas de **infraestrutura** (57) referem-se aos componentes de hardware e software subjacentes que suportam a aplicação. Elas incluem métricas como utilização de CPU, uso de memória, E/S de disco e largura de banda de rede.

4.2.6.2 Frequência

A tomada de decisão eficaz depende de informações atualizadas sobre recursos e serviços. Uma frequência maior de atualizações, como receber notificações imediatas para cada mudança nos valores das métricas, reduz o risco de depender de dados desatualizados. No entanto, uma alta frequência de atualização pode levar a sobrecarga computacional e congestionamento da 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 da rede. Existem duas abordagens para a frequência de monitoramento: Contínua e Periódica. O monitoramento **contínuo** (2) envolve o fluxo ininterrupto de dados de monitoramento para o servidor uma vez iniciado. O monitoramento **periódico** (22) permite configurar um período recorrente, com os dados sendo enviados em intervalos regulares.

4.2.7 Integração

Esta categoria avalia se os estudos de observabilidade estão em conformidade com um formato de dados padronizado. Algumas soluções usam arquivos JSON como formatos de dados padronizados fundamentais. Nos últimos anos, iniciativas como o OpenTelemetry (OTel) (23) (ver Seção 2.2.4), amplamente adotadas na indústria e consideradas o padrão ouro para observabilidade, foram introduzidas, construindo sobre padrões anteriores como OpenCensus e OpenTracing (6) (BOTEN; MAJORS, 2022). A adoção desses padrões facilita a troca de dados e a integração de vários componentes de monitoramento, mesmo que venham de diferentes

fornecedores ou operem em diferentes camadas de aplicação.

4.2.8 Coleta de Dados

Os dados gerados pelos microsserviços são coletados continuamente por um coletor centralizado. Duas técnicas principais são comumente usadas para essa coleta. A abordagem **com agente** (41) envolve a coleta de dados de uma instância de serviço e, em seguida, o encaminhamento para o coletor. Em contraste, a abordagem **sem agente** (1) permite que a instância de serviço envie seus dados gerados diretamente para o coletor. Embora o método sem agente seja mais simples de implementar e não exija suporte de infraestrutura adicional, ele requer que os desenvolvedores integrem o mecanismo de coleta de dados em sua base de código.

4.2.8.1 Instrumentação

Se a coleta for feita por um agente para tornar um sistema observável, o sistema deve ser instrumentado. Os desenvolvedores podem instrumentar o código de duas maneiras principais. A abordagem **manual** (4) permite que os desenvolvedores obtenham observações mais profundas e coletem telemetria diretamente de suas aplicações. A abordagem **automática** (11) é particularmente útil para configurações iniciais ou em situações onde a modificação da aplicação não é viável. Ambas as soluções oferecem telemetria abrangente ao extrair dados das bibliotecas usadas dentro da aplicação ou do ambiente de execução da aplicação.

4.2.9 Workload

Um *workload* refere-se às tarefas ou processos que um sistema de computação precisa executar e é fundamental para testar e avaliar o desempenho do sistema. Existem dois principais tipos de *workloads* usados em *benchmarks* e testes de desempenho: reais e sintéticos. Os *workloads* **reais** (26) são baseadas nas operações reais que um sistema executa em um ambiente de produção, representando as tarefas e processos que os usuários finais executam diariamente. Em contraste, *workloads* **sintéticos** (60) são criadas artificialmente para simular as operações de um sistema real. Esses *workloads* são geradas por ferramentas de *benchmark* e são projetadas para replicar padrões de uso específicos.

4.3 Classificação dos Estudos Selecionados com Base na Taxonomia

Esta seção categoriza os estudos de observabilidade selecionados para aplicações baseadas em microsserviços, com base na taxonomia definida de categorias e subcategorias. A Tabela 2 lista os 84 estudos selecionados, juntamente com sua classificação de acordo com a taxonomia de observabilidade. Na tabela, o *Domínio* refere-se a Métricas (M), *Logs* (L) ou *Traces* (T). Para *Integração (Int.)*, um valor de **S** indica que alguma forma de integração é utilizada, enquanto **N** mostra que não é. A *Coleta de Dados (Agente)* é marcada com **S** se um agente é utilizado, ou **N** se não é. Para *Logs*, as entradas são classificadas como Estruturados (**E**) ou Não Estruturados (**N**). Se nenhum valor for identificado, um traço (**-**) é utilizado. Abaixo, os estudos selecionados da MSL são apresentados de acordo com a taxonomia. As subseções a seguir visam discutir os detalhes dos estudos, com foco na divisão por propósito com o maior número de estudos: escalonamento automático (Seção 4.3.1), visão holística (Seção 4.3.3), detecção de anomalia (Seção 4.3.2) e análise da causa raiz (Seção 4.3.4). A Seção 4.3.5 cobre os estudos restantes com diversos propósitos. Finalmente, a Seção 4.3.6 fornece uma visão geral das descobertas dos estudos selecionados na MSL em relação à taxonomia.

Tabela 2 – Estudos selecionados.

#	Referência	Ambiente	Domínio	Propósito	Int.	Agente	Agente I.	Workload	T A.	T DE.	M Fonte	M Freq	L Formato	L Fonte
1	Cinque *et al.* (2019)	Nuvem	L-T	VH	-	-	-	Sintético	-	Lat	-	-	E	Infra
2	Nedelkoski *et al.* (2019)	Nuvem	T	Anomalia	-	-	-	Real	-	Lat	-	-	-	-
3	Wu *et al.* (2020)	Nuvem	M	RCA	-	-	-	Sintético	-	-	Infra/App	-	-	-
4	Kosińska e Zieliński (2020)	Nuvem	M-T	VH	S	-	-	Real	-	Lat	Infra/App	-	-	-
5	Scrocca *et al.* (2020)	Nuvem	M-L-T	VH	S	S	-	Sintético	Sem	Lat	Infra	Periódico	E	-
6	Qiu *et al.* (2020)	Nuvem	M	EA	-	-	-	Sintético	-	-	Infra/App	Periódico	-	-
7	Chowdhury *et al.* (2020)	Nuvem	M	EA	-	-	-	Sintético	-	-	Infra	-	-	-
8	Marie-Magdelaine e Ahmed (2020)	Nuvem	M	EA	-	-	-	Sintético	-	-	Infra/App	-	-	-
9	Yu *et al.* (2020)	Nuvem	M	EA	-	-	-	Sintético	-	-	Infra/App	Periódico	-	-
10	Levin e Benson (2020)	Nuvem	M	EA	-	S	-	Sintético	-	-	App	-	-	-
11	Wang *et al.* (2021)	Nuvem	M-L-T	RCA	-	S	-	Real	-	Dep	Infra	-	-	App
12	Simonsson *et al.* (2021)	Nuvem	M	Engenharia do Caos	-	-	-	Sintético	-	-	Infra/App	-	-	-
13	Gan *et al.* (2021)	Nuvem	M-T	RCA	S	S	-	Sintético	Frequência	Dep	Infra	Periódico	-	-
14	Cassé *et al.* (2021)	Nuvem	T	Uso de Recur-sos	S	S	-	Sintético	-	Dep/ Lat	-	-	-	-
15	Park *et al.* (2021)	Nuvem	M-T	EA	-	-	-	Real	Frequência	Dep/ Lat	Infra	Periódico	-	-
16	Zhang *et al.* (2021)	Nuvem	M-T	EA	-	S	-	Sintético	-	Dep	Infra	-	-	-
17	Bhasi *et al.* (2021)	Nuvem	M-T	EA	-	-	-	Real/ Sin-tético	-	Dep	App	-	-	-
18	Baarzi e Kesidis (2021)	Nuvem	M-T	EA	-	S	-	Real	-	Dep	Infra/App	Periódico	-	-
19	Li *et al.* (2021)	Nuvem	M-T	Anomalia	-	-	-	Real	-	Dep	Infra	-	-	-
20	Liu *et al.* (2021)	Nuvem	M	RCA	-	-	-	Sintético	-	-	App	-	-	-
21	Toka *et al.* (2021)	Nuvem	M	Anomalia	-	S	-	Sintético	-	-	Infra	-	-	-
22	Li *et al.* (2021)	Nuvem	M-T	RCA	-	-	-	Sintético	-	Lat	Infra/App	-	-	-
23	Chow *et al.* (2022)	Nuvem	M-T	Uso de Recur-sos	-	-	-	Real	-	Dep	Infra	-	-	-

A tabela continua na próxima página

#	Referência	Ambiente	Domínio	Propósito	Int.	Agente	Agente I.	Workload	T A.	T DE.	M Fonte	M Freq	L Formato	L Fonte
24	Cassé *et al.* (2022)	Nuvem/ IoT	T	Gargalos	S	S	A	Sintético	-	Dep/ Lat	-	-	-	-
25	Kratzke (2022)	Nuvem	M-L-T	VH	S	S	M	Sintético	Sem	-	Infra	-	E	Infra/App
26	Naqvi *et al.* (2022)	Nuvem/ IoT	M-L-T	Engenharia do Caos	-	-	-	Sintético	-	Dep	Infra/App	-	N	Infra
27	Zhu *et al.* (2022)	Nuvem	M-T	Agendamento	-	S	A	Sintético	-	Dep/ Lat	Infra	-	-	-
28	Tzanettis *et al.* (2022)	Nuvem/ IoT	M-L-T	EA	S	S	A	Sintético	-	Lat	Infra/App	Contínuo	E	Infra/App
29	Vu *et al.* (2022)	Nuvem	M-T	EA	-	-	-	Real	-	Lat	Infra/App	-	-	-
30	Song *et al.* (2022)	Nuvem/ IoT	M-T	EA	-	-	-	Real/ Sintético	-	Dep	Infra	Periódico	-	-
31	Maruf *et al.* (2022)	Nuvem	L	AP	S	-	-	Sintético	-	-	-	-	E	Infra/App
32	Rios *et al.* (2022)	Nuvem	L-T	RCA	-	-	-	Sintético	-	Dep/ Erro	-	-	-	-
33	Afshinpour *et al.* (2022)	Nuvem	L	Anomalia	-	-	-	Sintético	-	-	-	-	-	Infra/App
34	SP *et al.* (2022)	Nuvem	M	VH	-	-	-	Sintético	-	-	Infra	-	-	-
35	Ren *et al.* (2023)	Nuvem	M-L-T	Anomalia	-	S	-	Real	-	Lat	Infra/App	-	-	App
36	Kawasaki *et al.* (2023)	Nuvem	M	Anomalia	-	-	-	Sintético	-	-	Infra/App	-	-	-
37	Khichane *et al.* (2023)	Nuvem/ Móvel	M-L	Anomalia	-	S	-	Sintético	-	-	Infra/App	-	-	Infra/App
38	Monteiro *et al.* (2023)	Nuvem	M	Segurança	S	S	M	Sintético	Adaptativo	-	-	-	-	-
39	Li *et al.* (2023)	Nuvem	M-T	Agendamento	-	-	-	Sintético	Probab.	Dep	App	-	-	-
40	Yu *et al.* (2023)	Nuvem	M-L-T	RCA	S	S	A	Sintético	-	Lat	Infra/App	Contínuo	N	App
41	Kumar *et al.* (2023)	Borda	M-L-T	VH	S	S	-	Sintético	-	Lat	Infra/App	-	-	App
42	Usman *et al.* (2023)	Borda	M-L-T	VH	S	S	A	Sintético	-	-	Infra	-	N	-
43	Son *et al.* (2023)	Nuvem	M-T	EA	-	-	-	Real	-	Dep	Infra/App	Periódico	-	-
44	Liao *et al.* (2023)	Nuvem/ Móvel	M-T	EA	-	-	-	Real	-	Dep	Infra	-	-	-
45	Cai *et al.* (2023)	Nuvem	M-T	EA	-	S	-	Real	Frequência	Dep	Infra/App	Periódico	-	-
46	Liu *et al.* (2023b)	Nuvem	M-T	EA	S	-	-	Real	Frequência	Lat	Infra/App	-	-	-

A tabela continua na próxima página

#	Referência	Ambiente	Domínio	Propósito	Int.	Agente	Agente I.	Workload	T A.	T DE.	M Fonte	M Freq	L Formato	L Fonte
47	Liu *et al.* (2023a)	Nuvem/ Móvel	M-T	EA	S	S	-	Real	-	Lat	Infra/App	-	-	-
48	Meng *et al.* (2024)	Nuvem	M-T	EA	-	-	-	Sintético	Frequência	Dep	Infra/App	Periódico	-	-
49	Bi *et al.* (2023)	Nuvem	M-L-T	EA	-	-	-	Sintético	-	Lat	Infra/App	Periódico	-	-
50	Gu *et al.* (2023)	Nuvem	M-L-T	RCA	S	S	-	Real	-	Dep/ Error	Infra/App	-	-	App
51	Bruno *et al.* (2023)	Nuvem/ Móvel	M-L	Anomalia	-	-	-	Sintético	-	-	Infra	Periódico	E	-
52	Shen *et al.* (2023)	Nuvem	M-T	VH	-	S	A	Sintético	-	Lat	Infra	Periódico	-	-
53	Soldani *et al.* (2023)	Nuvem/ Móvel	M-T	VH	-	-	-	Sintético	-	Lat	Infra	-	-	-
54	Yang *et al.* (2023)	Nuvem	M-T	RCA	-	-	-	Real	-	Dep/ Lat	Infra/App	-	-	-
55	Zeng *et al.* (2023)	Nuvem	M	EA	-	S	-	Real	-	-	Infra/App	-	-	-
56	Raeiszadeh *et al.* (2023)	Nuvem	T	Anomalia	S	-	-	Sintético	-	Dep/ Lat	-	-	-	-
57	Costa *et al.* (2024)	Nuvem/ Neblina	M-L-T	VH	S	S	-	Real	Frequência	Dep/ Lat	Infra/ App	Periódico	-	Infra/ App
58	Akbari *et al.* (2024)	Nuvem/ Móvel	M	Anomalia	-	-	-	Sintético	-	-	Infra	Periódico	-	-
59	Sreemathy *et al.* (2024)	Nuvem	M	VH	S	S	-	Sintético	-	-	Infra	Periódico	-	-
60	Chow *et al.* (2024)	Nuvem	M-T	Desempenho/ Faturamento	S	S	-	Real	-	Lat	Infra	Periódico	-	-
61	Rezvani *et al.* (2024)	Nuvem	M-T	Desempenho	-	-	-	Sintético	-	Lat	App	-	-	-
62	Kannan *et al.* (2024)	Nuvem	M	VH	-	S	-	Sintético	-	-	Infra	-	-	-
63	Bonorden e Hoorn (2024)	Nuvem	T	APIs Web Depreciadas	S	S	A	Sintético	-	-	-	-	-	-
64	Talaver e Vakaliuk (2023)	Nuvem	M-L-T	Anomalia	S	S	-	Sintético	-	Lat/ Error	App	-	-	-
65	Sharma e Nadig (2024)	Nuvem	M-L-T	VH	S	S	M	Sintético	-	Lat	Infra	Periódico	-	-
66	Yokelson *et al.* (2024)	Nuvem	M	Desempenho	-	S	-	Sintético	-	-	Infra	Periódico	-	-
67	Noetzold *et al.* (2024)	Nuvem	M-L	Anomalia	-	N	-	Real	-	-	Infra/App	-	-	Infra
68	Han *et al.* (2024)	Nuvem	M-L-T	RCA	-	-	-	Real	-	Lat	Infra	-	E	-

A tabela continua na próxima página

#	Referência	Ambiente	Domínio	Propósito	Int.	Agente	Agente I.	Workload	T A.	T DE.	M Fonte	M Freq	L Formato	L Fonte
69	Borges *et al.* (2024)	Nuvem	M-T	Engenharia do Caos	S	S	A-M	Sintético	Probabilistic	Lat	Infra/App	Periódico	-	-
70	Chakraborty *et al.* (2024)	Nuvem	M-L	EA	S	S	-	Sintético	-	-	Infra/App	Periódico	E	Infra
71	Tian *et al.* (2024)	Nuvem	T	VH	S	-	-	Sintético	-	Dep/ Lat	-	-	-	-
72	Chen *et al.* (2024)	Nuvem	T	Engenharia do Caos	S	S	A	Sintético	-	Lat	-	-	-	-
73	Xie *et al.* (2024b)	Nuvem	M	RCA	-	-	-	Real	-	-	-	-	-	-
74	Sundqvist *et al.* (2023)	Nuvem/ Móvel	M-L-T	Anomalia	-	S	-	Sintético	-	Lat	Infra	-	E	Infra
75	Marchese e Tomarchio (2024)	Nuvem/ Borda	M-T	Agendamento	S	S	-	Sintético	-	Lat	Infra/App	Periódico	-	-
76	Otero *et al.* (2024)	Nuvem	T	VH	S	S	A	Sintético	-	Lat	-	-	-	-
77	Saha *et al.* (2024)	Nuvem	M-L-T	Anomalia	-	S	-	Sintético	-	Dep	-	-	N	-
78	Xie *et al.* (2024a)	Nuvem	T	EA	S	S	-	Sintético	-	Dep/ Lat	-	-	-	-
79	Gomes *et al.* (2024)	Nuvem	M-T	AP	S	S	A	Sintético	-	Dep/ Lat	Infra/App	-	-	-
80	Talaver e Vakaliuk (2024)	Nuvem	M-T	Anomalia	S	S	-	Sintético	-	Dep	App	-	-	-
81	Wu *et al.* (2024)	Nuvem	M	RCA	-	-	-	Real	-	-	Infra/App	-	-	-
82	Denaro *et al.* (2024)	Nuvem	M	Anomalia	-	-	-	Sintético	-	-	Infra/App	-	-	-
83	Kaushik *et al.* (2024)	Nuvem	M-T	Anomalia	-	-	-	Sintético	-	Dep/ Lat	Infra	-	-	-
84	Erakovic e Pahl (2025)	Nuvem	M-T	RCA	-	-	-	Sintético	-	Lat	Infra	-	-	-

Fonte: Autoria própria.

4.3.1 Escalonamento Automático

Com relação ao escalonamento automático, 22 trabalhos com este propósito foram identificados. Tzanettis *et al.* (2022) propõem uma abordagem de observabilidade integrada a uma implementação em IoT, voltada para aspectos de fusão de dados em plataformas de orquestração entre borda e nuvem, contemplando diferentes tipos de domínios para apoiar o escalonamento automático. Nesse mesmo contexto, Yu *et al.* (2020) apresentam o Microscaler, uma solução capaz de identificar automaticamente os serviços que demandam escalonamento e ajustá-los de forma a atender aos acordos de nível de serviço (SLAs) com custo otimizado. O Microscaler realiza a coleta periódica (a cada 10 segundos) de métricas de infraestrutura (CPU e memória) e de aplicação (latência), concentrando-se em informações essenciais para a tomada de decisão.

De forma complementar, Marie-Magdelaine e Ahmed (2020) propõem um *framework* de escalonamento automático proativo, fundamentado em modelos de previsão baseados em aprendizado de máquina, que ajusta dinamicamente o pool de recursos, tanto na dimensão horizontal quanto vertical. Em linha semelhante, Chowdhury *et al.* (2020) apresentam um *framework* que, a partir da análise de métricas relacionadas à aplicação, realiza orquestração automatizada proativa com o uso de técnicas de aprendizado de máquina para promover escalabilidade.

Por outro lado, Levin e Benson (2020) introduzem o ViperProbe, um *framework* de coleta de métricas para microsserviços baseado em eBPF, que analisa perfis de desempenho previamente à implantação em produção. O objetivo é reduzir de forma eficaz o conjunto de métricas de aplicação (como latência) a serem coletadas via agentes, diminuindo a sobrecarga do processo de monitoramento. Além disso, Qiu *et al.* (2020) apresentam o FIRM, um *framework* de gerenciamento inteligente e granular de recursos que busca proporcionar um compartilhamento previsível entre microsserviços, aumentando, assim, a eficiência na utilização global da infraestrutura.

Adicionalmente, Baarzi e Kesidis (2021) apresentam o SHOWAR, um *framework* que configura recursos determinando o número de réplicas e a quantidade de CPU e memória para cada microsserviço. O SHOWAR preenche a lacuna entre a alocação e o agendamento de recursos ideais, gerando regras de afinidade a partir de dependências de *traces* para que o agendador melhore ainda mais o desempenho. Além disso, Bhasi *et al.* (2021) apresentam o Kraken, um *framework* de gerenciamento de recursos de fluxo de trabalho para minimizar o

número de contêineres provisionados, garantindo a conformidade com o SLO.

Em complemento, Park *et al.* (2021) apresentam o GRAF, um *framework* de alocação de recursos proativo baseado em redes neurais de grafos para minimizar os recursos totais de CPU, satisfazendo o SLO de latência. Zhang *et al.* (2021) apresentam o Sinan, um gerenciador de cluster orientado a dados online e consciente da QoS para microsserviços na nuvem. Song *et al.* (2022) apresentam um sistema de escalonamento automático que identifica os serviços que necessitam de escalonamento e toma decisões sobre a alocação correta de recursos para eles. Vu *et al.* (2022) apresentam um método de escalonamento automático híbrido e consciente de picos para aplicações conteinerizadas para atender aos requisitos de QoS e otimizar a utilização de contêineres.

Nesse mesmo contexto, Son *et al.* (2023) apresentam o MicroBlend, um *framework* que fornece uma abordagem automatizada para combinar recursos, considerando dependências de microsserviços, para melhorar o desempenho e gerenciar os custos da aplicação. Bi *et al.* (2023) apresentam um *framework* de ajuste de microsserviços baseado no modelo de controle MAPE. O *framework* coleta regularmente dados de serviço em tempo de execução e identifica e ajusta proativamente serviços anormais. Cai *et al.* (2023) apresentam o AutoMan, um gerenciador de recursos orientado por aprendizado para microsserviços que permite o provisionamento de recursos, garantindo a conformidade com o SLO. Durante a execução, ele identifica proativamente os microsserviços críticos responsáveis por anomalias de desempenho.

Nessa mesma perspectiva, Liu *et al.* (2023a) apresentam o Sora, um *framework* de gerenciamento de adaptação de recursos flexíveis para identificar e ajustar o nível de concorrência ideal de microsserviços críticos para mitigar violações de SLO. Liao *et al.* (2023) apresentam o STAAF, um *framework* de escalonamento automático proativo para microsserviços que considera as correlações espaço-temporais de microsserviços para provisionamento de recursos global. Liu *et al.* (2023b) apresentam o μConAdapter, um *framework* de gerenciamento de adaptação de concorrência. Ele identifica rapidamente alocações de recursos flexíveis ideais para microsserviços críticos e os ajusta para mitigar violações de SLO.

De forma complementar, Meng *et al.* (2024) apresentam o DeepScaler, uma abordagem de escalonamento automático holística baseada em aprendizado profundo para microsserviços que se concentra no tratamento de dependências de serviço para otimizar o SLA e a eficiência de custos. Todavia, Chakraborty *et al.* (2024) propõem o Octopus, um *framework* de processamento de observabilidade multi-nuvem e multi-motor. No Octopus, fluxos de dados de

observabilidade declarativos (DODs) servem como uma abstração orientada por intenção para engenheiros de confiabilidade de sites (SREs) especificarem fluxos de dados de observabilidade auto-gerenciados.

Além disso, Xie *et al.* (2024a) propõem uma estratégia de implantação de microsserviços baseada em uma modelagem refinada e abrangente de dependências de microsserviços, fundamentada em uma análise aprofundada das características do serviço. Nesta abordagem, as dependências são medidas no nível de requisição e combinadas com relações de compartilhamento de dados para determinar conjuntamente a divisão de microsserviços, visando aprimorar a comunicação entre os serviços. Por fim, Zeng *et al.* (2023) apresenta uma estratégia de provisionamento de recursos auto-adaptativa e consciente da topologia para microsserviços. Primeiramente, é proposto um grafo de estado dos microsserviços para caracterizar o *status* de cada serviço em uma aplicação. Em seguida, utilizam-se redes neurais gráficas com mecanismos de atenção para extrair os requisitos de recursos e as correlações entre os microsserviços. Por fim, uma abordagem baseada em aprendizado por reforço é empregada para alocar recursos de forma uniforme entre os microsserviços.

4.3.2 Detecção de Anomalias

Em relação aos propósitos apresentados na taxonomia, a detecção de anomalias é verificada em 17 trabalhos. Khichane *et al.* (2023) apresentam um *framework* chamado 5GC-Observer para a observabilidade de serviços de rede 5G nativos da nuvem. A solução utiliza um método estatístico baseado no escore Z para garantir a detecção de degradação de desempenho no sistema 5G Core, identificando anomalias. Kawasaki *et al.* (2023) apresentam uma solução que utiliza eBPF para coletar informações detalhadas de infraestrutura e desenvolver um modelo de previsão de falhas para identificação de anomalias.

Além disso, Ren *et al.* (2023) apresentam o Triple, uma abordagem de detecção de anomalias baseada em aprendizado profundo para microsserviços. O Triple utiliza uma representação gráfica para descrever a relação de dependência de *traces*, métricas e *logs* coletados por um agente nos nós. Contudo, Sundqvist *et al.* (2023) propõem uma diretriz de *traces* que permite uma divisão baseada em componentes e procedimentos de *logs* de sistema para a futura Rede de Acesso de Rádio (RAN) 5G. No entanto, Akbari *et al.* (2024) apresentam um *framework* de monitoramento, observabilidade e análise para redes B5G mais verdes e convenientes.

Adicionalmente, Talaver e Vakaliuk (2023) utilizam telemetria para a análise de

sistemas de informação distribuídos, a detecção de práticas arquitetônicas prejudiciais e eventos anômalos. Noetzold *et al.* (2024) integram modelos de aprendizado de máquina em plataformas de observabilidade para aprimorar a precisão do monitoramento em tempo real. O sistema proposto facilita a detecção de anomalias rápida e proativa. Saha *et al.* (2024) propõem uma abordagem de solução para a Observabilidade de Processos de Negócio, detectando primeiro eventos anômalos que impactam o processo de negócio.

Em complemento, Nedelkoski *et al.* (2019) propõem um método de detecção de anomalias que utiliza uma rede neural multimodal com LSTM com dados de *traces*. Li *et al.* (2021) apresenta uma solução que detecta falhas em microsserviços usando uma janela deslizante dinâmica, e as causas raiz foram localizadas por meio de um algoritmo de ordenação baseado no rastreamento de chamadas. No entanto, Toka *et al.* (2021) propõem um pipeline de IA que processa dados de monitoramento coletados e compactação do conjunto de dados para uma operação de detecção de anomalias em métricas de infraestrutura e da aplicação. Afshinpour *et al.* (2022) utilizam aprendizado de máquina para criar um modelo a partir dos *logs* de monitoramento e apresentam algumas etapas para correlacionar eventos de entrada com as anomalias detectadas, a fim de prever falhas iminentes no sistema.

De forma complementar, Bruno *et al.* (2023) propõem uma abordagem multifacetada que combina métricas, *logs* e uma solução de aprendizado de máquina para diagnosticar anomalias em sistemas B5G. No entanto, Raeiszadeh *et al.* (2023) apresentam uma abordagem de detecção de anomalias guiada por *traces*, baseada em uma representação chamada Grafo Causal de Spans. O modelo é treinado utilizando Redes Neurais Gráficas (GNN) em conjunto com aprendizado positivo e não rotulado (PU learning), permitindo otimizar os parâmetros do modelo por meio da estimativa da distribuição subjacente dos dados. No entanto, Talaver e Vakaliuk (2024) apresentam um estudo para verificar a viabilidade de utilizar o padrão OpenTelemetry para analisar um sistema de informação distribuído, com foco na detecção e resposta rápida a práticas arquiteturais nocivas e eventos anômalos.

Em complemento, Denaro *et al.* (2024) propõem o Preface, uma abordagem para prever falhas em aplicações distribuídas com escalonamento automático. O Preface combina estatísticas descritivas com uma rede neural para identificar valores anômalos de KPIs que indicam sintomas de falhas iminentes no sistema, além de classificar os microsserviços que provavelmente são responsáveis por essas falhas. Por fim, Kaushik *et al.* (2024) apresenta uma solução que utiliza algoritmos de aprendizado de máquina e aprendizado profundo para

identificar falhas em aplicações baseadas em microsserviços. Ele utiliza de métricas e *traces* para prever e localizar potenciais falhas em sistemas baseados na arquitetura de microsserviços.

4.3.3 Visão Holística

Em relação aos propósitos apresentados na taxonomia, a visão holística é verificada em 15 trabalhos. Kosińska e Zieliński (2020) apresentam requisitos para o gerenciamento autônomo de CNA e a construção de um *framework* chamado AMoCNA, baseado em conceitos de Arquitetura Orientada a Modelos, visando reduzir a complexidade do gerenciamento dessas aplicações. Kratzke (2022) apresenta uma solução para a fusão dos três tipos de dados de observabilidade, de forma mais integrada e com uma abordagem de instrumentação direta, através de uma biblioteca de *logs* JSON unificados e estruturados.

Além disso, Kumar *et al.* (2023) apresentam o NEOS, que simplifica o processo de coleta, análise e visualização de métricas, *logs* e *traces*. Usman *et al.* (2023) apresentam um *framework* de integração de fluxo de trabalho de observabilidade multi-passos chamado DESK, destinado a melhorar os fluxos de trabalho para medição, coleta, fusão, armazenamento, visualização e notificação na observabilidade para os três domínios. Este trabalho também utiliza OTel, através de agentes e instrumentação automática para coleta de dados. Scrocca *et al.* (2020) investigam a observabilidade como um problema de pesquisa, discutindo os benefícios dos eventos como uma abstração unificada para métricas, *logs* e *traces*, e as vantagens de empregar técnicas e ferramentas de processamento de fluxo de eventos neste contexto.

Adicionalmente, Costa *et al.* (2024) fornecem observabilidade em um ambiente de Computação de Neblina usando ferramentas de código aberto. Sreemathy *et al.* (2024) destacam o papel crucial da observabilidade, monitoramento, *streaming* de dados e visualização de dados nas práticas modernas de engenharia de software. Todavia, Kannan *et al.* (2024) propõem um agente de observabilidade de rede leve, *netobserv-ebpf-agent*, construído usando eBPF, que pode ser implantado em vários ambientes e opera independentemente do *datapath* de rede subjacente ou das Interfaces de Rede de Contêiner implantadas pelo orquestrador.

Em complemento, Sharma e Nadig (2024) propõem uma solução baseada em eBPF que oferece observabilidade completa para implantações nativas da nuvem. Contudo, Tian *et al.* (2024) propõem o iTCRL, uma abordagem de aprendizado de representação contrastiva de *traces* baseada em intervenção causal. Esta abordagem primeiro constrói uma representação gráfica unificada para cada *trace*, para descrever o estado de tempo de execução dos eventos de serviço

em *traces* e as relações complexas entre eles. Otero *et al.* (2024) propõem uma abordagem leve e distribuída para telemetria que capitaliza as informações da API fornecidas pela Especificação OpenAPI.

Além disso, Cinque *et al.* (2019) apresenta uma abordagem para acompanhar os *logs* de microsserviços com rastreamento em caixa preta a fim de auxiliar os profissionais na tomada de decisões informadas durante a resolução de problemas. No entanto, SP *et al.* (2022) desenvolveram um operador para Kubernetes que automatiza o implantação e o monitoramento de aplicações, notificando comportamentos indesejados em tempo real. Ele também permite a visualização das métricas geradas pela aplicação e possibilita a padronização desses painéis de visualização para cada tipo de aplicação.

De forma complementar Shen *et al.* (2023) apresentam o DeepFlow, um *framework* para *traces* que facilita a solução de problemas centrada na rede em microsserviços. Ele oferece instrumentação automática e não intrusiva, elimina pontos cegos na rede e correlaciona efetivamente métricas de rede com *traces* da aplicação. Por fim, Soldani *et al.* (2023) propõem uma plataforma eBPF, denominada Sauron, com módulos que abrangem observabilidade de rede, segurança em tempo de execução e monitoramento de dissipação de energia.

4.3.4 Análise da Causa Raiz (RCA)

Em relação aos propósitos apresentados na taxonomia, a RCA é verificada em 13 trabalhos. Wang *et al.* (2021) apresentam o Groot, uma abordagem em tempo real baseada em gráficos de causalidade, usando resumos baseados em eventos de vários tipos de métricas, *logs* e *traces* no sistema em análise para RCA. Yu *et al.* (2023) apresentam o Nezha, uma abordagem de RCA que utiliza OTel para localizar causas raiz no nível da região do código e do tipo de recurso, analisando dados multimodais (os três domínios).

Contudo, Gan *et al.* (2021) apresentam o Sage, um sistema de RCA orientado por aprendizado de máquina para microsserviços de nuvem interativos, com foco na praticidade e escalabilidade. O Sage utiliza modelos de ML não supervisionados para contornar o custo da rotulagem de *traces*, captura periodicamente o impacto das dependências entre microsserviços para determinar a causa raiz do desempenho imprevisível e aplica ações corretivas para recuperar a QoS de um serviço de nuvem. No entanto, Gu *et al.* (2023) propõem a solução de RCA TrinityRCL, que é capaz de localizar as causas raiz de anomalias em múltiplos níveis de granularidade, incluindo nível de aplicação, nível de serviço, nível de host e nível de métrica.

Além disso, Han *et al.* (2024) propõem o HolisticRCA, um *framework* de análise de causa raiz para sistemas nativos da nuvem a partir de uma perspectiva holística. Xie *et al.* (2024b) propõem o LatentScope, um *framework* de RCA não supervisionado que modela Candidatos de Causa Raiz (RCCs) como variáveis latentes. A ideia é inferir o *status* dos RCCs como variáveis latentes usando métricas de monitoramento relacionadas, em vez de extrair diretamente recursos apenas das métricas observáveis. Wu *et al.* (2020) apresenta o MicroRCA, um sistema para localizar causas raiz de problemas de desempenho em microsserviços. O MicroRCA infere as causas raiz em tempo real ao correlacionar sintomas de desempenho da aplicação com a utilização correspondente de recursos do sistema, sem a necessidade de instrumentação da aplicação.

Em complemento, Liu *et al.* (2021) propõem um o MicroHECL, que baseia-se em grafo de chamadas de serviços construído dinamicamente e analisa possíveis cadeias de propagação de anomalias e ranqueia as causas raiz candidatas com base em uma análise de correlação. Todavia, Li *et al.* (2021) propõem o TraceRCA, que é uma solução que possui um mecanismo para avaliar se o microsserviço é a causa raiz baseado na quantidade de *traces* anômalos e *traces* normais que passam por ele. Ele é composto por detecção de anomalias em *traces*, mineração de um conjunto de microsserviços suspeitos e ranqueamento dos microsserviços. Rios *et al.* (2022) apresenta uma abordagem que utiliza *traces* para detectar, localizar e auxiliar automaticamente na explicação de falhas em nível de aplicação.

Adicionalmente, Yang *et al.* (2023) propõem o TraceNet, um algoritmo para localizar causas raiz, baseado em uma construção detalhada das dependências entre microsserviços no nível de operação. Wu *et al.* (2024) apresentam o *framework* FlowRCA que possui um modelo de fluxo para construir grafos de causalidade entre métricas e quantifica impactos causais por meio de predições de intervenção. Por fim, Erakovic e Pahl (2025) desenvolveram um *framework* híbrido de RCA que integra mineração de arquitetura, detecção de anomalias e análise baseada em regras. Ele incluiu um conjunto de regras para a RCA que abrange uma variedade de anomalias relacionadas a CPU, rede e armazenamento.

4.3.5 Outros Propósitos

Os outros estudos utilizam a observabilidade para diferentes propósitos. Cassé *et al.* (2022) apresentam uma solução que utiliza rastreamento distribuído OTel para identificar gargalos, por meio de um gráfico de propriedades hierárquicas com latência e dependência entre

microsserviços, no modelo Nuvem-IoT. Chow *et al.* (2022) apresentam o DeepRest, um sistema de estimativa de recursos orientado por aprendizado profundo. O DeepRest formula a estimativa de recursos como uma função do tráfego da API e aprende a causalidade entre as interações do usuário, utilizando a topologia de *traces* e a utilização de recursos diretamente em um ambiente de produção.

Adicionalmente, Maruf *et al.* (2022) propõem utilizar dados de telemetria para identificar a comunicação entre serviços e construir o Grafo de Dependência de Software. Em seguida, essa informação é usada para detectar *smells* arquiteturais e calcular métricas de APs automaticamente. Gomes *et al.* (2024) propõem uma abordagem para detectar APs de aplicação baseadas em microsserviços por meio dos *traces*. Cassé *et al.* (2021) utilizam traces distribuídos OTel e informações de alocação de microsserviços para detectar recursos ineficientes em aplicações baseadas em microsserviços.

No entanto, Zhu *et al.* (2022) apresentam o TOPOSCH, um *framework* de agendamento e execução para priorizar a QoS, equilibrando o desempenho de trabalhos em lote e mantendo a alta utilização do cluster através da coleta de recursos ociosos. Li *et al.* (2023) apresentam um *framework* de Agendamento de Microsserviços Orientado à Topologia (MOTAS), que utiliza topologias de microsserviços e *clusters* para otimizar a sobrecarga de rede de aplicações de microsserviços através de um algoritmo de mapeamento de grafos heurístico, bem como detectar e lidar com violações de QoS (throughput) em aplicações de microsserviços.

Em complemento, Marchese e Tomarchio (2024) propõem estender o Kubernetes com uma estratégia de agendamento orientada por telemetria que orquestra dinamicamente as aplicações com base nos estados dinâmicos da infraestrutura e da aplicação, ajustando continuamente sua colocação para aprimorar o desempenho. Simonsson *et al.* (2021) apresentam um sistema de injeção de falhas chamado ChaosOrca para chamadas de sistema em aplicações baseadas em contêineres. O ChaosOrca visa avaliar a capacidade de autoproteção de uma aplicação específica contra erros de chamada de sistema através de experimentos sob carga de trabalho semelhante à produção, sem instrumentar a aplicação.

De forma complementar, Naqvi *et al.* (2022) apresentam o CHESS, uma abordagem para a avaliação sistemática de sistemas auto adaptativos e auto curativos baseada em Engenharia do Caos, utilizando os três tipos de domínios para a compreensão da aplicação. Borges *et al.* (2024) propõem uma avaliação sistemática, reproduzível e comparativa de decisões de design de observabilidade, com foco especificamente na observabilidade de falhas. Chen *et al.* (2024)

apresentam o MicroFI, um *framework* de injeção de falhas não intrusivo com o objetivo de testar eficientemente diferentes funções da aplicação com injeção no nível de requisição.

Além disso, Bonorden e Hoorn (2024) apresentam uma abordagem dinâmica utilizando rastreamento para detectar chamadas para APIs web. Posteriormente, eles verificam se as APIs chamadas estão depreciadas usando uma especificação de API, metadados de resposta ou uma base de conhecimento. Monteiro *et al.* (2023) propõem uma abordagem de observabilidade adaptativa baseada na teoria dos jogos. A abordagem aborda o desafio de implementar sistemas de microsserviços prontos para forense, considerando as incertezas em incidentes de segurança. Chow *et al.* (2024) apresentam o Atlas, um gerenciador de migração de nuvem híbrida. O Atlas utiliza uma abordagem orientada por dados para aprender como cada API voltada para o usuário utiliza diferentes componentes e suas pegadas de rede para orientar as decisões de migração.

Adicionalmente, Rezvani *et al.* (2024) propõem uma abordagem de observabilidade de servidor utilizando eBPF para métricas de desempenho detalhadas no nível de requisição de aplicações sensíveis à latência de data center. Por fim, Yokelson *et al.* (2024) apresentam um *framework* baseado em serviço para monitoramento de aplicações HPC (SOMA). Eles demonstram que arquiteturas baseadas em serviço, juntamente com um modelo de dados apropriado, podem atender às necessidades de monitoramento de desempenho de fluxos de trabalho de conjunto em larga escala com baixa sobrecarga.

4.3.6 Análise Geral da Taxonomia

Com todo esse levantamento apresentado, é possível então revisitar a **QP1**. De acordo com as descobertas, pode-se verificar que a observabilidade para aplicações baseadas em microsserviços pode ser utilizada para diversos propósitos, com ênfase em escalonamento automático (26,19%).

Adicionalmente, o principal ambiente de execução para observabilidade é a nuvem (97,61%). Além disso, a maioria das soluções utiliza métricas como o principal tipo de dado (84,52%), com 20,23% utilizando os três tipos de dados. Também é evidente que algumas soluções utilizam OTel/OpenTracing/JSON para facilitar a integração e padronização com outras ferramentas. OTel é utilizado em 23 estudos e citado em outros 9, indicando a relevância deste padrão da indústria. Em relação ao tipo de carga de trabalho, 71,42% utilizam sintética. Para facilitar a coleta de dados pelas soluções, 48,8% utilizam agentes.

Em complemento, com relação à instrumentação, alguns optam por uma abordagem

automática, na qual nenhuma alteração é exigida pelo desenvolvedor, enquanto outros preferem uma abordagem mais manual. Algumas soluções utilizam amostragem para reduzir a quantidade de *traces* coletados, sendo a frequência a mais utilizada. Em relação às informações extraídas dos *traces*, é notável que a maioria das soluções utiliza latência e dependências entre microsserviços (98,21%). Em relação às métricas, as soluções utilizam métricas de infraestrutura (destacando CPU e memória) e métricas de aplicação (destacando tempo de resposta). Entre os 24 estudos que especificam a frequência de coleta de métricas, 22 adotam uma frequência periódica.

Além disso, em relação aos *logs*, entre os 13 estudos que especificam o formato dos *logs*, 9 utilizam *logs* estruturados e 4 utilizam *logs* não estruturados. Entre os 16 estudos que especificam a fonte dos *logs*, 6 utilizam *logs* de infraestrutura e aplicação.

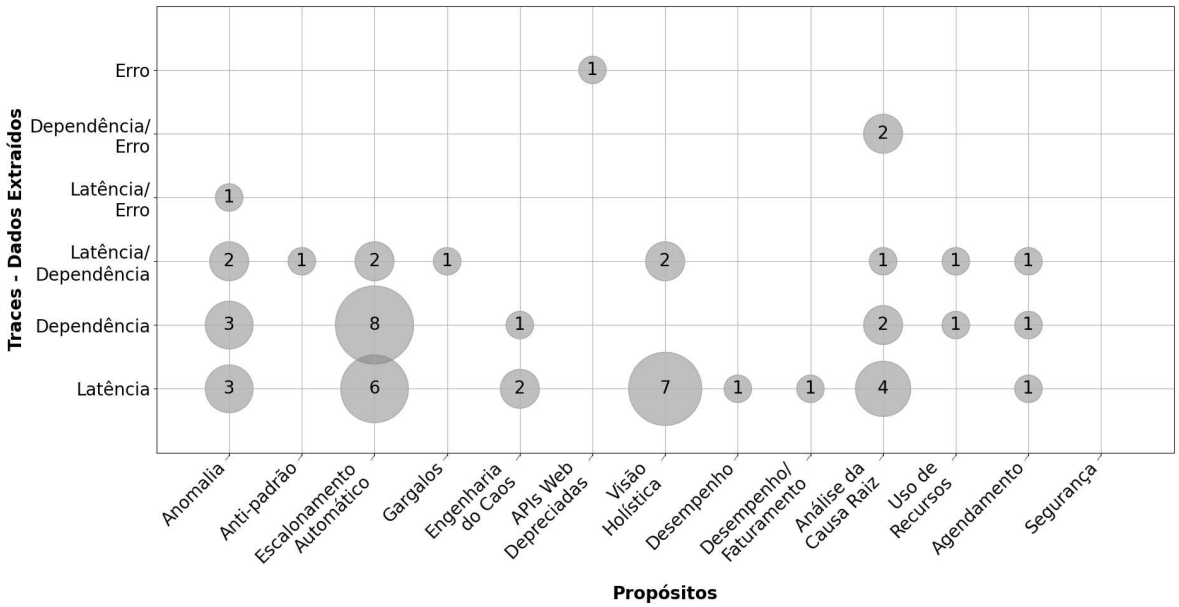
Neste trabalho, também foi realizado o cruzamento entre categorias com base na frequência dos estudos selecionados. Os gráficos exibem, nas interseções dos eixos X e Y, o número de artigos correspondentes a cada combinação das classificações, permitindo visualizar quantos desses trabalhos abordam ambos os critérios simultaneamente. O tamanho das bolhas é proporcional ao número de estudos que abordam cada combinação, e os valores numéricos correspondem às respectivas quantidades. Essa análise cruzada possibilita identificar padrões de coocorrência entre as classificações, evidenciando relações temáticas relevantes e tendências presentes no conjunto de estudos analisados.

A Figura 13 apresenta as classificações relacionadas à interseção dos critérios Propósito e *Traces* - Dados Extraídos. O eixo X mostra os tipos de propósitos. O eixo Y mostra os tipos de dados extraídos dos *traces*. Observa-se que a Latência é a dado mais frequentemente extraída, especialmente em estudos voltados a Visão Holística (7), Escalonamento Automático (6) e Análise da Causa Raiz (4). A classificação Dependência também apresenta representatividade relevante, com destaque para Escalonamento Automático (8) e Detecção de Anomalia (3).

As combinações de diferentes tipos de dados, como Latência/Dependência e Latência/Erro, ainda são pouco frequentes, o que indica que abordagens multivaloradas permanecem subexploradas. Esse cenário revela um potencial promissor de pesquisa voltado ao aprimoramento de técnicas para detecção de anomalias, análise de desempenho e compreensão holística de sistemas baseados em microsserviços.

A Figura 14 apresenta as classificações relacionadas à interseção dos critérios Propósito e *Workload*. Conforme mencionado anteriormente, o eixo X mostra os tipos de propósitos. O eixo Y mostra os tipos de cargas de trabalho utilizadas, tais como Real, Sintético e Real/Sintético.

Figura 13 – Propósitos x Trace - dados extraídos.



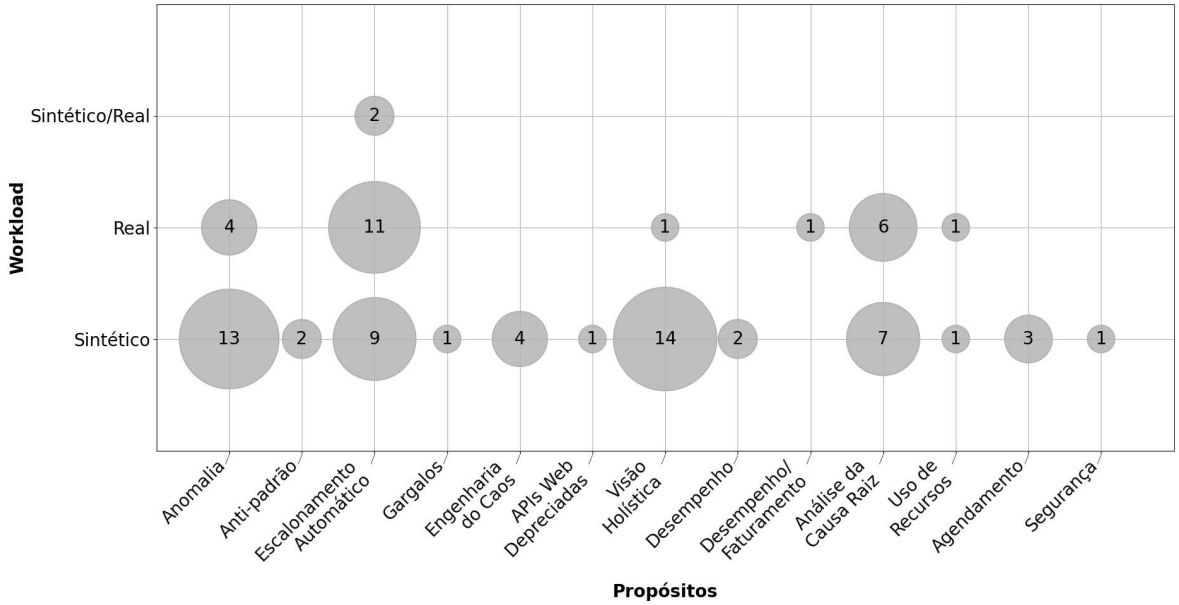
Fonte: Autoria própria.

Observa-se que os *workloads* sintéticos são amplamente predominantes, sendo utilizados na maioria dos propósitos analisados, com destaque para Visão Holística (14 estudos), Anomalia (13) e Escalonamento Automático (9). Esse resultado sugere uma preferência por experimentos controlados e reproduzíveis, típicos de cenários sintéticos.

Por outro lado, os *workloads* reais apresentam menor representatividade, embora se destaquem em propósitos como Escalonamento Automático (11) e Análise da Causa Raiz (6), indicando um esforço de parte da comunidade em validar resultados em ambientes mais próximos da prática. Os *workloads* híbridos são pouco explorados, aparecendo apenas no propósito Escalonamento Automático (2), o que evidencia uma lacuna na combinação de cenários controlados e reais para avaliação de soluções.

A Figura 15 apresenta as classificações relacionadas à interseção dos critérios *Purpose* e *Domains*. Como mencionado anteriormente, o eixo X mostra os tipos de propósitos, enquanto o eixo Y mostra os tipos de domínios, como M, T, M-T, M-L e M-L-T. Observa-se que o domínio de Métricas (M) é amplamente predominante, especialmente quando combinado com *Traces* (M-T) e *Logs* e *Traces* (M-L-T), o que evidencia o foco crescente em soluções de observabilidade para microsserviços. Destacam-se os propósitos de Escalonamento Automático (16), Detecção de Anomalia (10), Visão Holística (9), e Análise da Causa Raiz (9), que aparecem associados a multidomínio, refletindo a necessidade de integrar diferentes fontes de dados para diagnóstico e compreensão do comportamento do sistema.

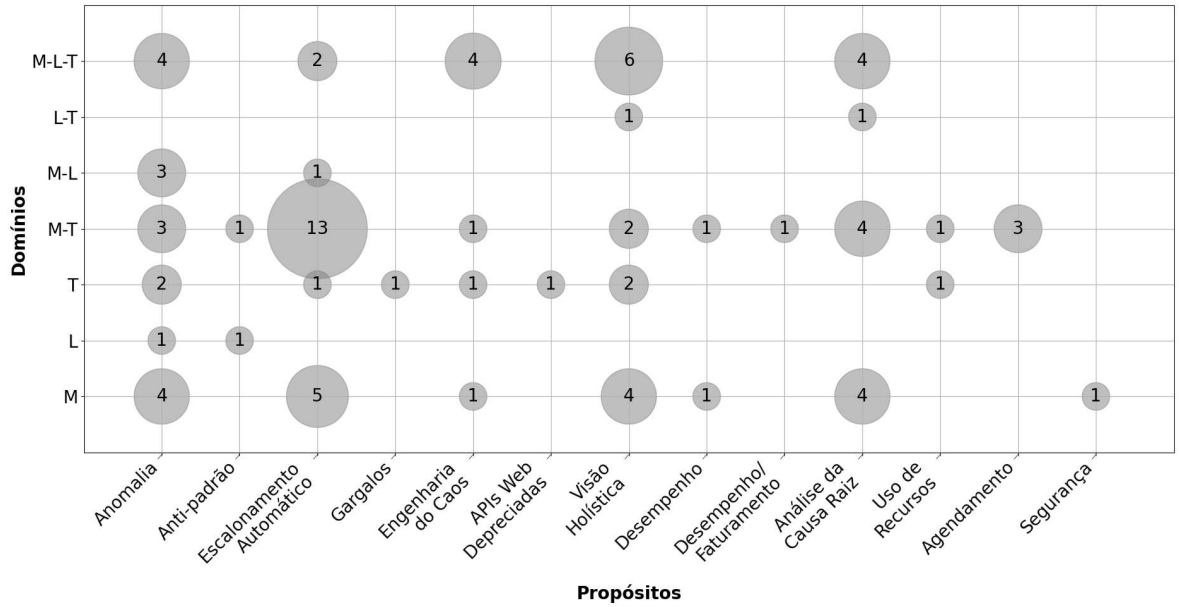
Figura 14 – Propósitos x Workload.



Fonte: Autoria própria.

Por outro lado, domínios isolados, como *Logs* (L) ou *Traces* (T), aparecem em menor escala, sugerindo que estudos baseados em um único tipo de dado têm perdido espaço para abordagens mais integradas. Essa tendência reforça a importância de análises multidomínio, que combinam métricas, *logs* e *traces* para aprimorar a detecção de gargalos, a investigação de causas raiz e a visão holística do sistema. Esses resultados ajudam a responder à **QP1**, ao demonstrarem a diversidade de domínios e categorias de observabilidade, reforçando a necessidade de uma taxonomia estruturada que sirva de base para pesquisas futuras.

Figura 15 – Propósitos x Domínios.



Fonte: Autoria própria.

4.4 Ferramentas, Aplicações *Benchmarking* e *Workloads*

Esta seção apresenta as ferramentas, as aplicações *bencmarking* e os conjunto de dados de *workloads* do mundo real mapeadas dos estudos selecionados na MSL.

4.4.1 Conjunto de Dados de Workload do Mundo Real

Esta seção detalha as *workloads* do mundo real utilizadas nos estudos selecionados, que incluem *traces* de usuários reais. O uso de *workloads* do mundo real é essencial para avaliar e otimizar sistemas em ambientes que espelham de perto seus contextos operacionais pretendidos. A Tabela 3 apresenta a distribuição dos conjuntos de dados de *workloads* do mundo real dos estudos selecionados. Os conjuntos de dados de *workloads* do mundo real utilizados nos artigos selecionados são apresentados abaixo.

A **Wikipedia**[1] disponibilizou, por meio da Wikimedia Foundation, uma amostra de 10% de todas as solicitações direcionadas a todos os projetos wiki (URDANETA *et al.*, 2009). A amostra gerada contém 20,6 bilhões de solicitações HTTP correspondentes ao período de 19 de setembro de 2007 a 2 de janeiro de 2008. Cada solicitação em seu traço é caracterizada por um ID único, um carimbo de data/hora, o URL solicitado e um campo que indica se a solicitação resultou em uma operação de salvamento. Já o **WorldCup98**[2] consiste em todas as solicitações feitas ao site da Copa do Mundo de 1998 entre 30 de abril e 26 de julho daquele ano. Durante este período de 88 dias, 1.352.804.107 solicitações foram recebidas pelo site da Copa do Mundo. Não há informações disponíveis sobre quantas solicitações não foram registradas, embora se acredite que não houve interrupções no sistema durante o período de coleta. Os campos da estrutura de solicitação contêm as seguintes informações: carimbo de data/hora, identificador do cliente (IP); URL solicitado; o método HTTP; o número de bytes na resposta; o código de *status* da resposta HTTP; o tipo de arquivo solicitado; localização do servidor que tratou a solicitação.

O **AzurePublicDatasetV2**[3] é o repositório da Azure que contém *traces* do Microsoft Azure para o benefício da comunidade de pesquisa e acadêmica. O conjunto de dados de traços tem dois traços representativos da carga de trabalho de máquinas virtuais (VM) do Microsoft Azure coletados em 2017 e 2019. AzurePublicDatasetV1 - *traces* criados usando dados da carga de trabalho de VM do Azure de 2017 contendo informações sobre 2M VMs e 1,2B leituras de

[1] <http://www.wikibench.eu/?page_id=60>
[2] <https://ita.ee.lbl.gov/html/contrib/WorldCup.html>
[3] <https://github.com/Azure/AzurePublicDataset>

utilização (CORTEZ *et al.*, 2017). AzurePublicDatasetV2 - *traces* criados usando dados da carga de trabalho de VM do Azure de 2019 contendo informações sobre 2,6M VMs e 1,9B leituras de utilização. O **AOL Query Logs**[4], lançou um arquivo de texto compactado em um de seus sites contendo 20 milhões de consultas de pesquisa para mais de 650.000 usuários durante um período de 3 meses, destinado a pesquisa, em 4 de agosto de 2006. A AOL removeu o arquivo de seu site até 7 de agosto, mas não antes de ter sido copiado e distribuído na Internet. A AOL não identificou os usuários no relatório; no entanto, informações pessoalmente identificáveis estavam presentes em muitas das consultas.

O **ClarkNet**[5] contêm *traces* de duas semanas de todas as solicitações HTTP ao servidor ClarkNet. ClarkNet é um provedor de acesso completo à Internet para a área metropolitana de Baltimore-Washington DC. O primeiro log foi coletado de 00:00:00 28 de agosto de 1995 até 23:59:59 3 de setembro de 1995, totalizando 7 dias. O segundo log foi coletado de 00:00:00 4 de setembro de 1995 até 23:59:59 10 de setembro de 1995, totalizando 7 dias. Nesse período de duas semanas, houve 3.328.587 solicitações. Os carimbos de data/hora têm resolução de 1 segundo. O conjunto de dados do **Facebook**[6] é fornecido pelo *Network Repository*, que é o primeiro repositório de dados interativo com análise e visualização de grafos em tempo real (ROSSI; AHMED, 2015), com interações e postagens.

O conjunto de dados do **INRIA**[7] possui rotulagem de imagens aéreas do INRIA, que é composto por 360 tiles RGB de 5000×5000px com uma resolução espacial de 30cm/px em 10 cidades ao redor do globo (MAGGIORI *et al.*, 2017), e possui vista terrestre pública de contornos de edifícios. O **Twitter**[8] possui uma coleção simples de JSON capturado do fluxo geral do Twitter, para fins de pesquisa, história, teste e memória. O traço do Twitter tem uma grande variação em picos (média = 3332 rps, pico = 6978 rps). Twitter é um serviço de rede social online e *microblogging* que permite aos usuários enviar e ler mensagens de texto de até 140 caracteres, conhecidas como “tweets”.

O **WITS**[9] disponibilizou um conjunto de dados que consiste em *traces* de tráfego capturados pelo grupo WAND na Nova Zelândia. Como a largura de banda é cara na Nova Zelândia, é muito benéfico para o grupo de pesquisa WAND ter esses dados hospedados em outro lugar. O ponto de captura está no link que interconecta a Universidade com a Internet. Já o

[4] <https://www.kaggle.com/datasets/dineshydv/aol-user-session-collection-500k>
[5] <https://ita.ee.lbl.gov/html/contrib/ClarkNet-HTTP.html>
[6] <https://networkrepository.com/socfb.php>
[7] <https://project.inria.fr/aerialimagelabeling/>
[8] <https://archive.org/details/twitterstream>
[9] <https://wand.net.nz/wits/>

Alibaba[10] reúne métricas detalhadas de tempo de execução de quase vinte mil microsserviços, coletadas em *clusters* de produção com mais de dez mil nós bare-metal durante 13 dias em 2022. Em relação à versão anterior (v2021), o traço atualizado possui duração maior e inclui informações adicionais, como o ID do serviço no grafo de chamadas (LUO *et al.*, 2022).

Por fim, o **AIOps**[11] contém dados do campeonato internacional AIOPS 2022, cujo tema foi a identificação e classificação de falhas em arquiteturas de microsserviços em sistemas de comércio eletrônico. O objetivo da competição foi explorar modos de falha e desenvolver modelos de classificação para cenários de mitigação de perdas, exigindo que os participantes projetassem algoritmos de detecção de anomalias e classificação de falhas eficientes e precisos.

Tabela 3 – Distribuição dos conjuntos de dados de *workload* do mundo real.

Nome	Artigo #
AIOps	19, 35, 54, 68, 81
Wikipedia	17, 18, 29, 43
WorldCup98	29, 30
AzurePublicDatasetV2	15, 44
Facebook	23, 60
INRIA	23, 60
Alibaba	55, 78
AOL Query Logs	30
ClarkNet	45
Twitter	17
WITS	43

Fonte: Autoria própria.

4.4.2 Aplicações Benchmarking

Uma aplicação *benchmarking* serve como uma implementação de referência para avaliar vários aspectos de arquiteturas de microsserviços. Esse tipo de aplicação simula um cenário do mundo real, geralmente envolvendo vários serviços que interagem entre si para realizar tarefas complexas. O uso de tais aplicações de referência ajuda a identificar melhores práticas, possíveis gargalos e áreas de melhoria em arquiteturas de microsserviços, fornecendo observações valiosas para a construção de sistemas mais eficientes e resilientes. A Tabela 4 mostra as aplicações *benchmarking* usadas nos estudos. A última linha compreende aplicações desconhecidas, que não foram identificadas pelos autores deste trabalho ou são aplicações proprietárias dos estudos selecionados. As aplicações *benchmarking* utilizadas nos artigos selecionados são apresentadas abaixo.

[10] <https://github.com/alibaba/clusterdata/tree/master/cluster-trace-microservices-v2022>
[11] <https://competition.aiops-challenge.com/home/competition/1496398526429724760>

A **TrainTicket**[12] fornece funcionalidades típicas de reserva de passagens de trem, como consulta de passagens, reserva, pagamento, alteração e notificação do usuário (ZHOU *et al.*, 2018). Ele é projetado usando princípios de design de microsserviços e cobre diferentes modos de interação, como invocações síncronas, invocações assíncronas e filas de mensagens. Foi desenvolvido em 2018. O sistema contém 41 microsserviços relacionados à lógica de negócios. Ele usa quatro linguagens de programação: Java, Python, Node.js e Go; e usa dois bancos de dados: Mongo e MySQL. Já a **Online Boutique**[13] é uma aplicação de demonstração de microsserviços voltada para a nuvem. A aplicação é um aplicativo de comércio eletrônico baseado na web onde os usuários podem navegar por itens, adicioná-los ao carrinho e comprá-los. Esta aplicação foi desenvolvida pelo Google para demonstrar como os desenvolvedores podem modernizar aplicações empresariais. Foi desenvolvida em 2018. É composta por 10 microsserviços escritos em diferentes linguagens que se comunicam entre si via gRPC. Ele usa cinco linguagens de programação: Java, Python, Node.js, Go e Javascript; e usa um banco de dados: Redis. **Hipster Shop**[14] é um *fork* do repositório do Online Boutique e demonstra como instrumentar para rastreamento distribuído e monitoramento usando múltiplas bibliotecas de rastreamento, incluindo OpenTelemetry, OpenTracing e *tracers* da LightStep, Zipkin e Jaeger. Esta aplicação tem os mesmos microsserviços.

A **DeathStarBench**[15] é um conjunto de benchmarks de código aberto construído com microsserviços que é representativo de grandes serviços de ponta a ponta, modular e extensível (GAN *et al.*, 2019). Foi desenvolvido em 2019. DeathStarBench inclui uma Rede Social, um Serviço de Mídia e Reserva de Hotel. A **Social Network** é uma rede social com relações de seguimento unidirecionais, implementada com microsserviços fracamente acoplados, que se comunicam entre si via Thrift RPCs. Ações suportadas: Criar postagem de texto, Ler postagem, Ler toda a linha do tempo do usuário, Receber recomendações sobre quais usuários seguir, Pesquisar no banco de dados por usuário ou postagem, Registrar/Login usando credenciais do usuário, Seguir/Deixar de seguir usuário. A aplicação contém 36 microsserviços. Ele usa nove linguagens de programação: C, C++, Java, Node.Js, Python, Scala, PHP, Javascript e Go; e usa dois bancos de dados: Memcached e Mongo.

A **Media Service** é uma aplicação que implementa um serviço de ponta a ponta para navegar por informações de filmes, bem como revisar, avaliar, alugar e transmitir filmes. A

[12] <https://github.com/FudanSELab/train-ticket>
[13] <https://github.com/GoogleCloudPlatform/microservices-demo>
[14] <https://github.com/lightstep/hipster-shop>
[15] <https://github.com/delimitrou/DeathStarBench/>

aplicação contém 38 microsserviços. Ele usa oito linguagens de programação: C, C++, Java, Node.Js, Python, Scala, PHP e Javascript; e usa três bancos de dados: Memcached, Mongo e MySQL. Já a **Hotel Reservation** é uma aplicação que implementa um serviço de reserva de hotel, construído com Go e gRPC. Ações suportadas: Obter perfil e taxas de hotéis próximos disponíveis durante determinados períodos de tempo, Recomendar hotéis com base em métricas fornecidas pelo usuário, Fazer reservas. A aplicação contém 15 microsserviços. Ele usa uma linguagem de programação: Go; e usa dois bancos de dados: Memcached e Mongo.

A **Sock Shop**[16] é um site de comércio eletrônico que permite aos clientes navegar e comprar diferentes tipos de meias, desenvolvido inicialmente pela Weaveworks para fins de demonstração e teste. Foi desenvolvido em 2016. A aplicação contém 8 microsserviços. Utiliza quatro linguagens de programação: Java, .Net, Go e Node.Js; e utiliza três bancos de dados: Redis, Mongo, MySql. Já a **BookInfo**[17] é uma aplicação composta por quatro microsserviços separados, utilizada para demonstrar várias funcionalidades do Istio. A aplicação exibe informações sobre um livro, semelhante a uma única entrada de catálogo de uma livraria online. A página exibe uma descrição do livro, detalhes do livro (ISBN, número de páginas, etc.) e algumas resenhas do livro. Foi desenvolvido em 2017. Utiliza quatro linguagens de programação: Java, Python, Ruby e Node.Js; e utiliza dois bancos de dados: Mongo, MySql.

Por fim, a **OpenTelemetry Demo**[18] é um sistema distribuído baseado em microsserviços, destinado a ilustrar a implementação do OpenTelemetry em um ambiente quase real. A aplicação é composta por microsserviços escritos em diferentes linguagens de programação que se comunicam entre si via gRPC e HTTP; e um gerador de carga que utiliza o Locust para simular tráfego de usuários. Foi desenvolvido em 2022. A aplicação contém 13 microsserviços. Utiliza onze linguagens de programação: .NET, C++, Go, Java, JavaScript, Kotlin, PHP, Python, Ruby, Rust e TypeScript; e utiliza um banco de dados: Redis.

As Figuras 16 e 17 apresentam o cruzamento entre duas categorias (propósitos e domínios) com base na frequência dos estudos selecionados que empregam aplicações *benchmarking*. Os gráficos mostram, nas interseções dos eixos X e Y, o número de artigos correspondentes a cada combinação, de forma semelhante às representadas anteriormente nas Figuras 13, 14 e 15.

Na Figura 16, observa-se que diversas aplicações foram empregadas para os propósitos de Escalonamento Automático e Análise da Causa Raiz. Assim, destaca-se o uso

[16] <https://github.com/microservices-demo/microservices-demo>

[17] <https://istio.io/latest/docs/examples/bookinfo/>

[18] <https://opentelemetry.io/docs/demo/architecture/>

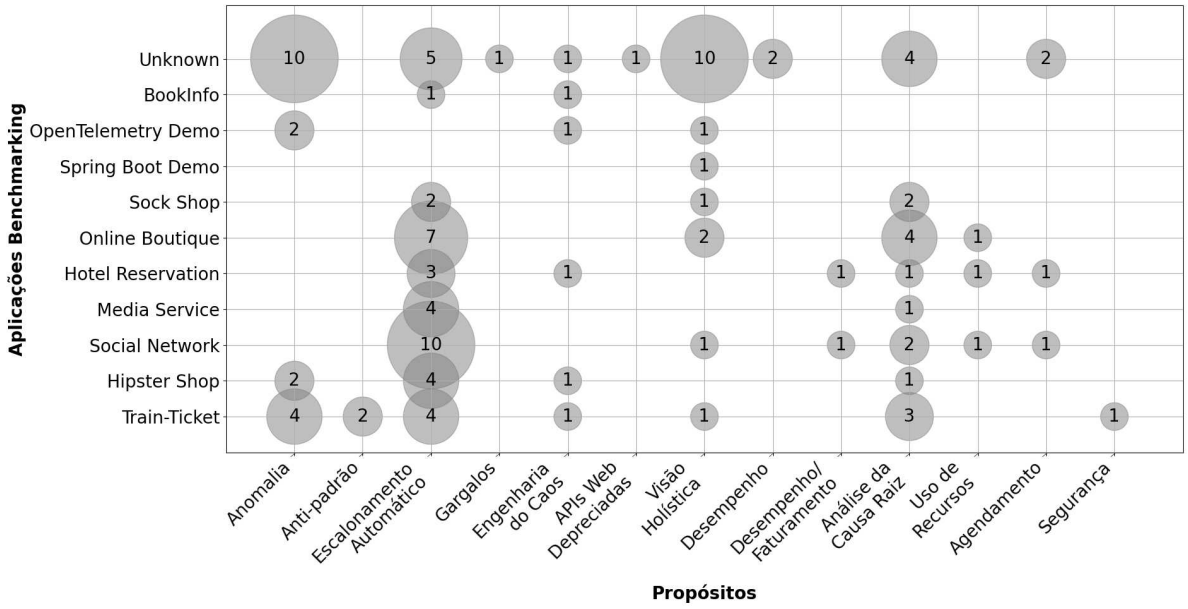
Tabela 4 – Distribuição das aplicações *benchmarking*.

Aplicação *Benchmarking*	Artigo #
Train-Ticket	6, 18, 22, 31, 32, 33, 35, 38, 40, 48, 55, 56, 71, 72, 79, 82
Social Network	6, 13, 15, 16, 17, 18, 23, 39, 43, 45, 46, 47, 55, 60, 68, 71
Online Boutique	8, 14, 15, 18, 22, 40, 43, 44, 48, 53, 71, 73, 78, 81
Hotel Reservation	6, 13, 16, 17, 23, 39, 60, 72
Hipster Shop	9, 10, 30, 35, 54, 68, 72, 83
Media Service	6, 13, 17, 43, 55
Sock Shop	3, 4, 46, 47, 81
OpenTelemetry Demo	64, 65, 69, 80
BookInfo	12, 48
Desconhecida	1, 2, 5, 7, 11, 19, 20, 21, 24, 25, 26, 27, 28, 29, 34, 36, 37, 41, 42, 49, 50, 51, 52, 57, 58, 59, 61, 62, 63, 66, 67, 70, 74, 75, 76, 77

Fonte: Autoria própria.

da *Train-Ticket*, que aparece associada a múltiplos propósitos, como Detecção de Anomalias, Identificação de Anti-padrões e Escalonamento Automático, entre outros.

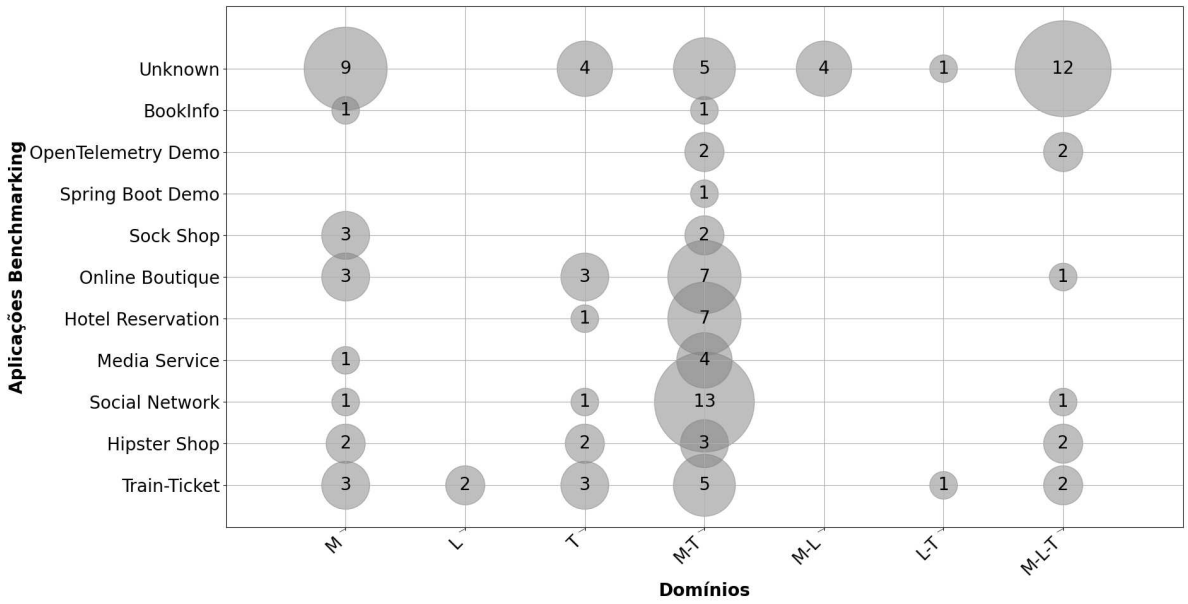
Figura 16 – Aplicações x Propósitos



Fonte: Autoria própria

Já na Figura 17, observa-se que a combinação de Métricas e *Traces* (M–T) foi explorada por todas as aplicações, com destaque para a *Social Network*, presente em 13 estudos. A *Train-Ticket*, por sua vez, abrange todas as combinações entre os diferentes domínios, com exceção da associação entre Métricas e *Logs* (M–L).

Figura 17 – Aplicações x Domínios.



Fonte: Autoria própria.

4.4.3 Ferramentas

Esta seção apresenta o mapeamento das ferramentas utilizadas nos estudos selecionados. Essas ferramentas têm como foco testes de carga, alerta, monitoramento, *traces*, *logs*, visualização e *service meshs*.

4.4.4 Análise Geral das Ferramentas, Aplicações Benchmarking e Workloads

Essas descobertas contribuem para responder à **QP2**, apresentando as ferramentas, aplicações e conjuntos de dados utilizados, revelando uma variedade de abordagens no contexto da observabilidade em arquiteturas de microsserviços. O número de ocorrências em cada uma dessas abordagens está indicado entre parênteses.

A ferramenta mais recorrente para a realização de testes de carga é o Locust (20). No que se refere ao monitoramento, destaca-se o Prometheus (41), enquanto o gerenciamento de alertas é predominantemente conduzido pelo Alert Manager (5) nos estudos analisados. Para a análise de *traces*, o Jaeger (21) figura como a solução mais utilizada. De modo complementar, o Elasticsearch (8) é amplamente adotado para o tratamento de *logs*, ao passo que, na visualização, sobressai-se o Grafana (16). Por fim, no âmbito de *service mesh*, o Istio (12) constitui a principal ferramenta identificada.

No que concerne às aplicações *benchmarking*, as mais utilizadas nos estudos se-

Tabela 5 – Ferramentas usadas nos estudos selecionados.

Tipo	Nome	Artigo #
Teste de Carga	Locust[42]	3, 8, 10, 15, 16, 18, 23, 29, 44, 48, 60, 65, 69, 72, 75, 78, 79, 80, 82, 83
	Wrk2[43]	6, 13, 39
	JMeter[44]	27
	Gatling[45]	38
	Vegeta[46]	28
Alerta	AlertManager[47]	5, 17, 34, 40, 42
Monitoramento	Prometheus[48]	3, 4, 5, 6, 7, 8, 9, 12, 13, 15, 17, 18, 21, 23, 26, 28, 29, 30, 34, 37, 40, 41, 42, 43, 44, 48, 51, 53, 55, 57, 58, 59, 60, 65, 67, 69, 73, 75, 79, 82, 83
	cAdvisor[49]	4, 5, 6, 12, 13, 15, 30, 36, 40, 43, 44, 49, 47, 48, 57, 58, 60, 75
	Elastic Beats[50]	25, 57, 73
	NetData[51]	21
	Heapster[52]	49
	BCC[53]	10
Traces	Jaeger[54]	5, 6, 13, 14, 15, 16, 23, 24, 41, 42, 45, 47, 48, 56, 57, 60, 69, 72, 75, 78, 79
	Zipkin[55]	2, 28, 49, 83
	Kiali [56]	26, 34
Logs	Elasticsearch[57]	21, 25, 42, 49, 51, 57, 72, 78
	Loki[58]	36, 40, 41, 42
	Fluentd[59]	5, 28, 51
	Logstash[60]	41, 49
Visualização	Grafana[61]	4, 10, 12, 21, 26, 28, 34, 37, 41, 42, 51, 57, 59, 65, 67, 83
	Kibana[62]	25, 51
Service Mesh	Istio[63]	3, 9, 26, 34, 30, 31, 34, 48, 55, 60, 72, 75
	Linkerd[64]	8, 15, 44

Fonte: Autoria própria.

lecionados para a condução dos testes são o TrainTicket (16) e o Social Network (16). Essas aplicações se destacam não apenas pela diversidade de tecnologias envolvidas, mas também pelo elevado número de microsserviços que compõem suas arquiteturas, o que proporciona cenários complexos e representativos para avaliação. Essa complexidade permite investigar aspectos cruciais do desempenho, da escalabilidade e da resiliência das soluções testadas, tornando-as referências consolidadas para experimentos que visam compreender o comportamento de sistemas distribuídos sob diferentes condições de carga e interação entre serviços.

No que se refere aos conjuntos de dados de *workload* do mundo real, os mais empregados para a condução de testes nos estudos selecionados são o AIOps (5) e o Wikipedia (4). Enquanto o primeiro fornece aspectos de confiabilidade e resiliência em situações de falhas operacionais, o segundo enfatiza um cenário de carga em termos de acesso da Web.

4.5 Ameaças à Validade da Taxonomia

Esta seção discute potenciais ameaças à validade do estudo da taxonomia, organizadas segundo as dimensões de validade de construção, interna, externa e de conclusão. Embora diversas estratégias tenham sido adotadas para reduzir riscos, é importante reconhecer as limitações que podem influenciar tanto a interpretação quanto a generalização dos achados.

Validade de Construção: As questões de pesquisa formuladas podem não abranger integralmente a complexidade do fenômeno da observabilidade em microsserviços. A taxonomia proposta foi cuidadosamente construída a partir de elementos recorrentes na literatura. Entretanto, aspectos emergentes ou ainda pouco explorados podem ter sido desconsiderados. Essa lacuna pode restringir a abrangência da classificação e comprometer sua capacidade de representar de maneira plena e holística o tema investigado.

Validade Interna: Para mitigar vieses no processo de seleção dos estudos, foi utilizada uma *string* de busca previamente definida e aplicada de forma consistente em quatro bases de relevância. Além disso, os critérios de inclusão e de exclusão foram explicitamente especificados, e o processo de triagem passou por revisão. Apesar desses cuidados, existe a possibilidade de que estudos pertinentes tenham sido excluídos em virtude de variações terminológicas ou limitações nos mecanismos de indexação.

Validade Externa: Os resultados apresentados derivam da análise de 84 publicações produzidas entre 2019 e 2025. Embora esse recorte temporal assegure o alinhamento com debates contemporâneos e avanços recentes, ele implica a exclusão de trabalhos anteriores a 2019, bem como de estudos muito recentes que ainda não foram indexados.

Validade de Conclusão: Entre as ameaças potenciais encontram-se vieses na extração dos dados, decisões de caráter subjetivo durante o processo de classificação e limitações inerentes ao escopo do conjunto analisado. Para enfrentar tais riscos, o estudo seguiu um protocolo estruturado de revisão e disponibilizou publicamente o conjunto de dados utilizado. Essas medidas não apenas reforçam a transparência e a reprodutibilidade, como também possibilitam que outros pesquisadores repliquem, validem ou expandam os resultados aqui apresentados.

4.6 Considerações Finais

Nesse capítulo, apresentou-se a taxonomia de observabilidade para aplicações baseadas em microsserviços, construída a partir de um mapeamento sistemático da literatura. O

desenvolvimento da taxonomia possibilitou organizar e categorizar os elementos recorrentes identificados nos estudos selecionados, fornecendo uma visão estruturada da observabilidade atualmente discutidas na área.

A classificação detalhada dos estudos, bem como a análise das ferramentas, aplicações *benchmarking* e conjuntos de dados, contribuiu para evidenciar padrões, lacunas e oportunidades de investigação futura. Além disso, a discussão sobre as ameaças à validade do estudo permitiu contextualizar limitações metodológicas e fortalecer a transparência do processo de construção da taxonomia.

De maneira geral, a taxonomia proposta oferece uma base conceitual sólida que pode servir tanto como referência para pesquisadores que busquem compreender melhor a observabilidade em microsserviços quanto como guia para profissionais que desejem avaliar, e implementar práticas de monitoramento e análise em ambientes distribuídos.

O próximo capítulo apresenta o catálogo de anti-padrões de observabilidade para aplicações baseadas em microsserviços desenvolvido nesta tese. São descritas a metodologia utilizada para sua construção, as fontes consultadas, o catálogo propriamente dito, bem como a avaliação realizada com especialistas para validar o conteúdo.

5 UM CATÁLOGO DE ANTI-PADRÕES DE OBSERVABILIDADE

Este capítulo apresenta o Catálogo de Anti-Padrões (APs) de observabilidade para aplicações baseadas em microsserviços desenvolvido na tese (GOMES *et al.*, 2025a). Primeiramente, a Seção 5.1 apresenta a metodologia utilizada no mapeamento sistemático e na literatura cinzenta. Depois, a Seção 5.1.3 apresenta as fontes de APs de observabilidade, considerando o mapeamento e a literatura cinzenta. Em seguida, a Seção 5.1.6 detalha o catálogo desenvolvido, com o *template* para a estruturar cada um dos APs, descrição do portal digital, e a lista com todos os APs identificados. Posteriormente, a Seção 5.3 mostra a avaliação com especialistas realizado para validar o catálogo. Por fim, a Seção 5.4 discute as potenciais ameaças à validade do catálogo.

5.1 Metodologia do Mapeamento Sistemático da Literatura e Mapeamento da Literatura Cinzenta

Para o desenvolvimento do catálogo de APs de observabilidade, inspirado em (TIGHILT *et al.*, 2020), foi realizado um Mapeamento Sistemático da Literatura (MSL), com o objetivo de identificar e sintetizar o conhecimento acadêmico disponível sobre o tema. A Figura 18 ilustra os passos seguidos para o MSL, que contempla: (1) Definição do escopo, (2) Busca, (3) Filtragem, (5) Classificação e (6) Extração de informações.

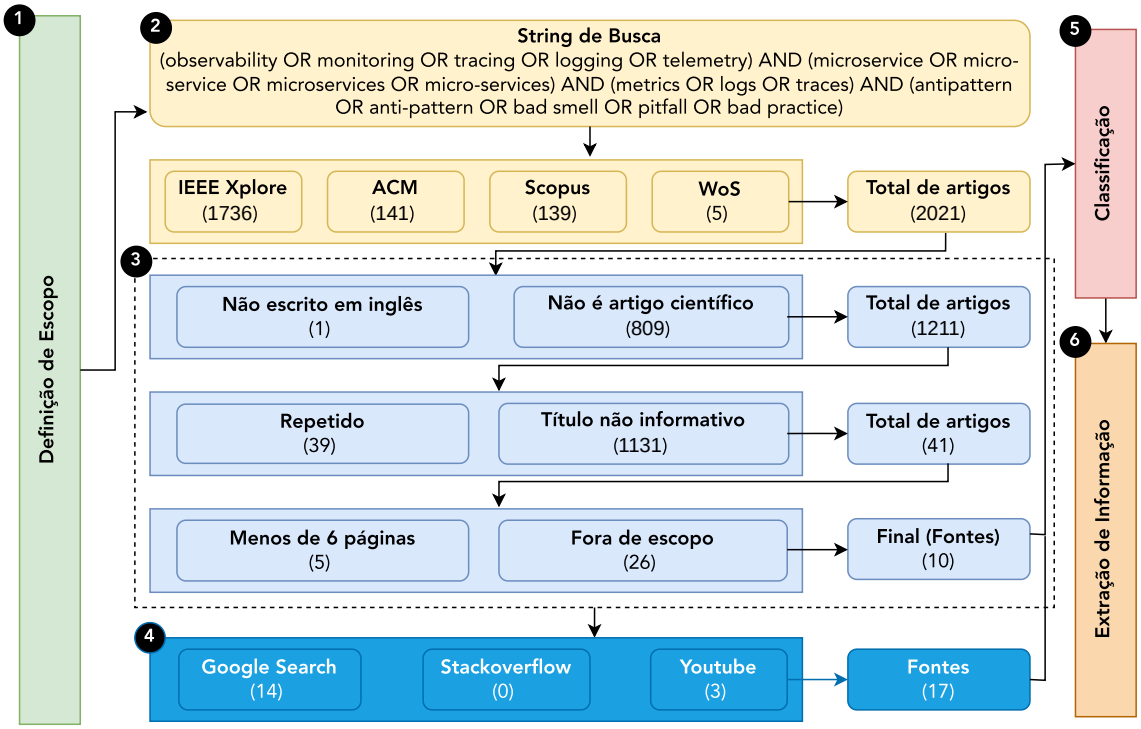
Adicionalmente, foi conduzido um Mapeamento da Literatura Cinzenta (MLC), ilustrado na Figura 18 (4), abrangendo documentação técnica, *blogs*, vídeos e outras fontes não acadêmicas, a fim de capturar percepções práticas e desafios enfrentados pela indústria.

A combinação dessas abordagens proporcionou uma perspectiva abrangente e bem fundamentada, equilibrando rigor científico com aplicabilidade prática no contexto da observabilidade. O processo de revisão foi realizado entre fevereiro de 2025 e abril de 2025 e, para complementar os resultados, foi realizado novamente em junho de 2025.

5.1.1 Mapeamento Sistemático da Literatura

Como apresentado anteriormente, a elaboração da MSL referente ao catálogo foi realizada de forma metodológica, contemplando os passos descritos em cada subseção seguinte.

Figura 18 – Etapas da metodologia do catálogo.



Fonte: Autoria própria.

5.1.1.1 Definição do Escopo

Na fase de definição do escopo, as questões de pesquisa **QP3** e **QP4** foram norteadoras para a obtenção de resultados mais precisos sobre o tema em estudo. Para a formulação dessas perguntas, houve consenso entre o autor da tese e os orientadores quanto à análise de tópicos relevantes para orientar pesquisas sobre APs de observabilidade.

5.1.1.2 Busca

Nesta etapa, foi aplicada uma *string* de busca nos mesmos quatro mecanismos de pesquisa: **IEEE Xplore, ACM, Scopus** e **Web of Science**. Como o catálogo é complementar em relação a taxonomia, a *string* utilizada é a mesma acrescentando termos relacionados a ao tema "Anti-padrões". Assim, o termo "*anti-pattern*" foi incluído, juntamente com sinônimos como "*antipattern*, *bad smell*, *pitfall*" e "*bad practice*". Após a busca e a recuperação dos documentos, foi realizado um processo de filtragem.

5.1.1.3 Filtragem

A estratégia de filtragem também considerou critérios diretos de exclusão da taxonomia, com exceção do relacionado a estudos secundários, pois estes trazem uma visão geral do tema. Os artigos que não atenderam a nenhum dos critérios de exclusão foram incluídos e avançaram para as etapas seguintes. Os critérios de inclusão também são os mesmos. O fluxograma na Figura 18 mostra o processo de filtragem, sendo que no total as buscas retornaram 2021 artigos. O fluxograma mostra o número de artigos excluídos e seus respectivos motivos em cada fase de filtragem. No total, restaram 10 artigos que serviram de base para o catálogo de AP de observabilidade.

5.1.1.4 Classificação

Na quarta etapa, foram analisados os artigos e identificamos os AP de observabilidade presentes neles. Esse processo permitiu extrair informações relevantes e agrupar os AP conforme suas categorias funcionais, contribuindo para o desenvolvimento do catálogo. Inicialmente, definimos as principais categorias. Especificamente, estabelecemos seis categorias: **métricas,** ***logs*****,** ***traces*****, alertas,** ***dashboards*** e **geral**. As três primeiras refletem os tipos de dados de observabilidade.

Uma categoria foi criada para "alertas" e "*dashboards*" porque foram identificados AP relacionados a alertas gerados por ferramentas, bem como *dashboards* desenvolvidos para exibir os dados. Por fim, a categoria "geral" abrange aspectos relacionados a processos, uso de ferramentas e outros fatores. Adicionalmente, os AP foram classificados com base no número de artigos que discutem cada um, destacando sua relevância.

5.1.1.5 Extração de Informações

Na quinta etapa, foi realizada a extração de informações com base na leitura integral dos artigos selecionados, mapeando os dados conforme as categorias previamente definidas. O processo de classificação foi conduzido de forma colaborativa pelo autor da tese e os orientadores. O autor da tese foi responsável pela leitura completa dos estudos, enquanto os orientadores acompanharam os achados e ofereceram direcionamentos na construção das classificações. Além disso, orientaram o processo de extração das informações e participaram de discussões para o preenchimento do *template* utilizado na descrição dos AP de observabilidade (ver Seção 5.1.4).

5.1.2 Mapeamento da Literatura Cinzenta

A MLC foi realizada com o objetivo de identificar informações relevantes que não estavam disponíveis em fontes acadêmicas tradicionais. Como dito anteriormente, ela seguiu os mesmos passos da MSL. Para isso, foram feitas buscas em três plataformas: Google Search[1], StackOverflow[2] e YouTube[3], utilizando os mesmos termos da MSL. No Google, as buscas foram feitas até a décima página de resultados, no qual foi encontrado um vídeo relevante sobre o tema. No StackOverflow, não foram encontrados resultados significativos. No YouTube, foram identificados três vídeos pertinentes. No total, a MLC resultou em 4 vídeos, 9 *blogs* técnicos e 2 sites oficiais. Além disso, dois artigos sobre observabilidade foram incluídos, ambos sendo pesquisas industriais. Esses resultados complementam as informações obtidas na MSL, oferecendo uma visão mais ampla sobre os AP de observabilidade. O processo metodológico, bem como as classificações, pode ser visualizado no seguinte link: <https://doi.org/10.5281/zenodo.17114298>.

5.1.3 Fontes de Anti-padrões de Observabilidade

A Tabela 6 apresenta a distribuição das fontes de AP de observabilidade, considerando a MSL e a MLC. Essa tabela é dividida de acordo com o tipo de fonte (Artigo Científico, Vídeo, Blog Técnico e Site Oficial), título da fonte, ano, referência e identificação (ID).

[1] <https://www.google.com/>
[2] <https://stackoverflow.com/>
[3] <https://www.youtube.com/>

Tabela 6 – Fontes dos anti-padrões de observabilidade.

Tipo	Título	Ano	Referência	ID
Artigo Científico	On Observability and Monitoring of Distributed Systems – An Industry Interview Study	2019	Niedermaier *et al.* (2019)	1
	On the Study of Microservices Antipatterns: a Catalog Proposal	2020	Tighilt *et al.* (2020)	2
	A Survey of Software Log Instrumentation	2021	Chen e Jiang (2021)	3
	A Survey on Automated Log Analysis for Reliability Engineering	2021	He *et al.* (2021)	4
	Enjoy your observability: an industrial survey of microservice tracing and analysis	2021	Li *et al.* (2022)	5
	A Survey on Observability of Distributed Edge & Container-Based Microservices	2022	Usman *et al.* (2022)	6
	Visualizing Microservice Architecture in the Dynamic Perspective: A Systematic Mapping Study	2022	Gortney *et al.* (2022)	7
	Characterizing and Mitigating Anti-patterns of Alerts in Industrial Cloud Systems	2022	Yang *et al.* (2022)	8
	Logging Practices in Software Engineering: A Systematic Mapping Study	2022	Gu *et al.* (2022)	9
	Serverless: From Bad Practices to Good Solutions	2022	Taibi *et al.* (2022)	10
	Catalog and detection techniques of microservice anti-patterns and bad smells: A tertiary study	2023	Cerny *et al.* (2023)	11
	Toward the Observability of Cloud-Native Applications: The Overview of the State-of-the-Art	2023	Kosińska *et al.* (2023)	12
Vídeo	10 tips to BAD Observability by Sameer Mhaisekar	2024	Mhaisekar (2024)	13
	Bad API observability: 5 anti-patterns in 5 minutes	2024	Bhattacharya (2024)	14
	AWS: Your Ally Against Observability Anti-Patterns | Indika Wimalasuriya | Conf42 Observability 2024	2024	Wimalasuriya (2024)	15
	Antipatterns in Observability: Lessons Learned and How OpenTelemetry Solves Them - Steve Flanders	2025	Flanders (2025)	16
Blog Técnico	Observability Mythbusters: Observability Anti-Patterns	2022	Villela (2022)	17
	Mistakes to avoid in Observability	2022	Mittal (2022)	18
	Bad Observability	2023	Townshend (2023)	19
	Everything You Know About Observability is Wrong	2023	LaRock (2023)	20
	OpenTelemetry Collector Anti-Patterns	2024	Villela (2024)	21
	How to Resolve Bad Observability Data Quality	2024	Patel (2024)	22
	How to Solve 3 Data Flow Issues in Observability	2024	Patel (2024)	23
	Top 10 Mistakes People Make When Building Observability Dashboards	2024	O'Donnell (2024)	24
	7 API Observability Anti-Patterns to Avoid	2024	Chevre (2024)	25
Website Oficial	Anti-patterns for continuous monitoring	2025	AWS (2025)	26
	AWS Observability Best Practices	2025	AWS (2025)	27

Fonte: Autoria própria.

5.1.4 Template de Documentação dos Anti-padrões de Observabilidade

O modelo adotado para a documentação dos AP de observabilidade foi adaptado a partir do *template* disponibilizado na C2 Wiki (C2 Wiki, 2024). Tal estrutura propicia uma abordagem sistemática para a descrição dos AP, promovendo a uniformização da apresentação e facilitando a compreensão e mitigação dos problemas relacionados à observabilidade em arquiteturas baseadas em microsserviços. O *template* contempla os seguintes campos essenciais:

- **Nome**: Denominação concisa e descritiva do AP;
- **Categoria**: Classificação do AP com base em categorias predefinidas (métricas, *logs*, *traces*, alertas, *dashboards* ou geral);
- **Relevância**: Classificação do AP de acordo com níveis de relevância (alta, média e baixa). Detalhes adicionais sobre essa classificação podem ser encontrados na Seção 5.1.5;
- **Contexto**: Descrição do cenário em que o AP ocorre, incluindo as condições típicas que favorecem seu surgimento;
- **Problema**: Principais implicações do AP, com destaque para seus impactos sobre a observabilidade;
- **Exemplo**: Ilustração prática, real ou hipotética, que evidencia a manifestação do AP;
- **Solução Recomendada**: Conjunto de boas práticas e diretrizes para a mitigação ou prevenção do AP; e
- **Consequências**:
  - **Positivas**: Benefícios decorrentes da aplicação das soluções recomendadas.
  - **Negativas**: Possíveis dificuldades na adoção das soluções recomendadas, como o aumento da complexidade ou a elevação de custos operacionais.

5.1.5 Classificação de Relevância

Após a consolidação dos AP de observabilidade a partir das fontes levantadas na revisão de literatura, procedeu-se à análise do número de citações de cada AP com o intuito de aferir sua relevância. Na sequência, foi aplicado um processo de ranqueamento com base em quartis, assegurando uma classificação estatisticamente fundamentada que refletisse a distribuição das frequências de citação.

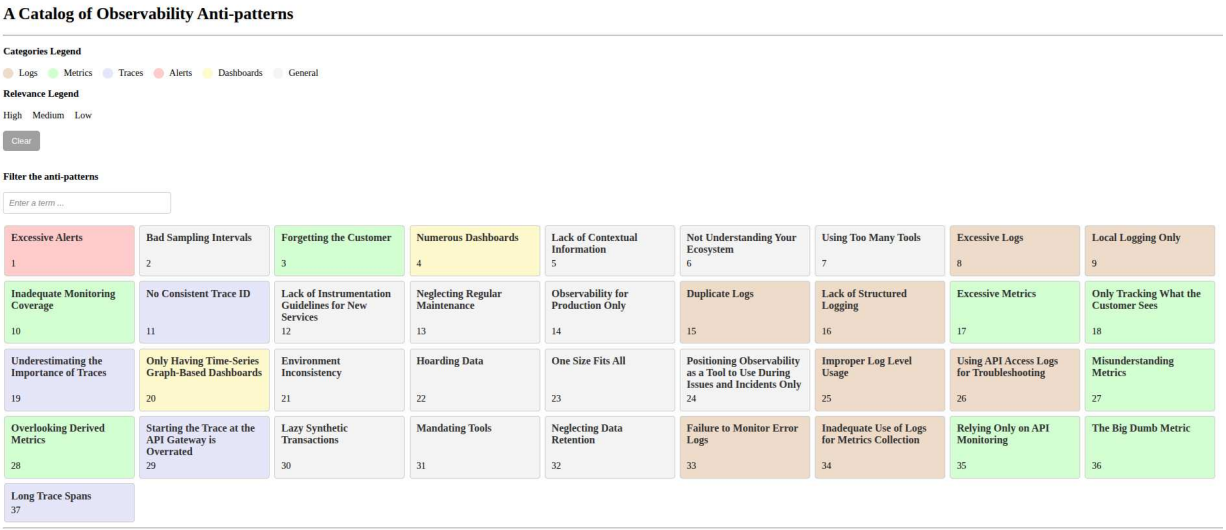
Os valores de corte foram estabelecidos da seguinte forma: primeiro quartil (Q1) com 2 citações, segundo quartil (Q2) com 3 citações, terceiro quartil (Q3) com 4 citações e quarto quartil (Q4) com 6 ou mais citações. Com base nessa segmentação, a classificação de relevância foi definida da seguinte forma: (i) Baixa relevância: 1 ou 2 citações; (ii) Relevância média: 3 ou 4 citações; e (iii) Alta relevância: de 6 a 8 citações.

5.1.6 Portal do Catálogo

Com o intuito de disponibilizar de forma acessível e interativa os resultados obtidos nesta pesquisa, foi desenvolvido um portal digital do catálogo de APs, disponível no endereço: <https://observability-antipatterns.github.io/>. Esse portal funciona como um repositório centralizado, reunindo todos os APs identificados durante o estudo e apresentando-os em uma interface visual organizada no formato de grade. Cada AP é exibido de acordo com o *template* previamente estabelecido (Seção 5.1.4).

A Figura 19 ilustra a página inicial do portal, evidenciando a organização gráfica dos APs. Essa interface visual não apenas facilita a navegação, mas também oferece uma visão panorâmica do conjunto de problemas mais recorrentes em observabilidade, organizados de modo a promover uma melhor compreensão por parte de profissionais, pesquisadores e estudantes interessados no tema.

Figura 19 – Página inicial do portal.



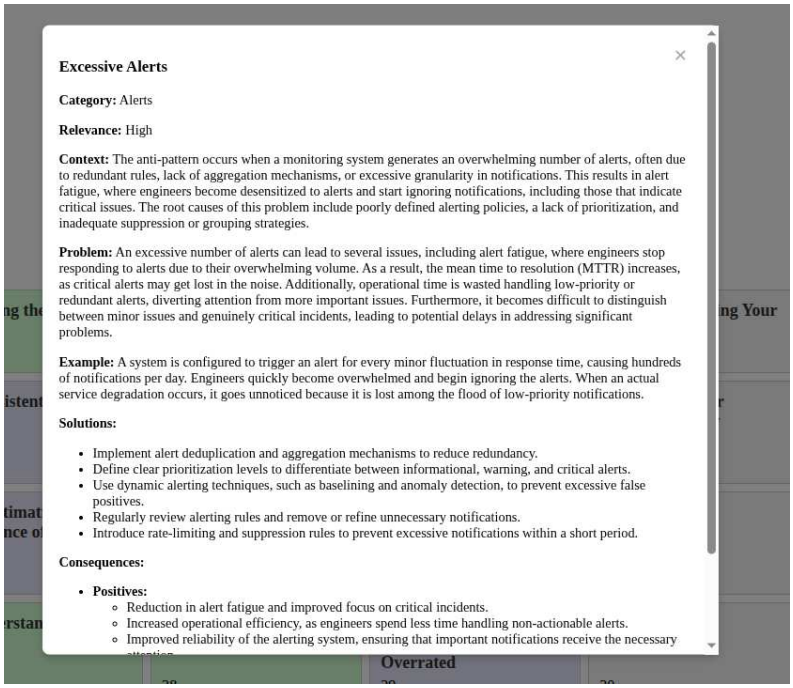
Fonte: Autoria própria.

Os APs apresentados no portal estão ordenados de acordo com o número de citações encontradas na literatura (conforme apresentado na Tabela 7), refletindo assim sua relevância no

estado da arte. Além disso, cada AP possui uma coloração distinta que o associa à respectiva categoria, permitindo ao usuário identificar rapidamente a área à qual pertence e favorecendo comparações entre diferentes classes de problemas.

Para enriquecer a experiência de navegação, ao clicar em qualquer AP, o portal exibe suas informações detalhadas por meio de uma janela modal, como demonstrado na Figura 20, que exemplifica a descrição completa do AP *Alertas Excessivos* baseado no *template*. Essa funcionalidade possibilita que o usuário acesse rapidamente informações aprofundadas sem perder a visão geral do catálogo.

Figura 20 – Exemplo de descrição detalhada no portal do AP alertas excessivos.



Fonte: Autoria própria.

Outro recurso fundamental do portal é a presença de mecanismos de filtragem e busca, apresentados na Figura 21. Os filtros permitem a seleção de APs por categoria (métricas, *logs*, *traces*, alertas, *dashboards* ou geral) ou por grau de relevância (alto, médio ou baixo), oferecendo ao usuário diferentes perspectivas de análise. Paralelamente, o campo de busca possibilita localizar APs específicos por meio de termos-chave, realizando uma varredura em todos os campos do *template*. Esse mecanismo foi concebido para aumentar a precisão na recuperação da informação e, consequentemente, apoiar tanto atividades de pesquisa quanto a aplicação prática no contexto da engenharia de software.

Dessa forma, o portal do catálogo constitui-se não apenas como um instrumento de documentação, mas também como uma ferramenta de apoio à decisão e à aprendizagem. Ele

Figura 21 – Filtros do portal.

Fonte: Autoria própria.

promove a disseminação do conhecimento sobre APs de observabilidade, facilita a identificação de problemas recorrentes e suas soluções recomendadas, e contribui para o desenvolvimento de práticas mais eficazes no desenho, implementação e manutenção de arquiteturas de observabilidade.

A Tabela 7 lista os 37 AP identificados, acompanhados de suas respectivas categorias e classificações de relevância. O conteúdo completo de cada um dos AP, seguindo o *template*, pode ser visto no Apêndice A.

5.1.7 Análise Geral do Catálogo

De acordo com o estudo, AP de observabilidade foram identificados em diversas fontes, tanto acadêmicas quanto da indústria. Os AP mais frequentemente citados foram Alertas Excessivos, Intervalos de Amostragem Inadequados e Esquecer o Cliente, cada um com oito citações, o que os posiciona como os mais relevantes. Além disso, destacam-se os AP *Dashboards* em excesso e Uso excessivo de ferramentas, citado exclusivamente por fontes não acadêmicas, totalizando sete citações. Esse dado evidencia sua relevância prática e a necessidade de uma atenção especial por parte dos usuários. A categoria com o maior número de APs foi a categoria Geral, que representa 37,83% (14 de 37) do total identificado. As categorias *Logs* e Métricas apareceram em seguida, cada uma com 21,62% (8 de 37) do total.

Por fim, tais resultados contribuem para a resposta à **QP3**, ao demonstrar que o catálogo mapeou os APs de observabilidade mais recorrentes, os quais demandam maior atenção

Tabela 7 – Lista com os 37 anti-padrões identificados.

ID	Nome	Categoria	Relevância	Fonte
1	Alertas excessivos	Alerta	Alta	8, 15, 18, 19, 20, 24, 26, 27
2	Intervalos de amostragem inadequados	Geral	Alta	3, 4, 5, 13, 15, 18, 19, 22
3	Esquecer o cliente	Métrica	Alta	1, 13, 14, 15, 18, 19, 25, 27
4	*Dashboards* em excesso	*Dashboard*	Alta	13, 15, 17, 18, 19, 20, 24
5	Uso excessivo de ferramentas	Geral	Alta	13, 15, 16, 17, 18, 19, 25
6	Falta de informações contextuais	Geral	Alta	1, 6, 15, 20, 24, 27
7	Não entender seu ecossistema	Geral	Alta	1, 6, 12, 15, 17, 19
8	*Logs* excessivos	*Log*	Alta	3, 4, 9, 13, 15, 27
9	Cobertura de monitoramento inadequada	Métrica	Média	2, 11, 16, 21, 26
10	*Log* apenas local	*Log*	Média	2, 7, 11, 27
11	Ausência de ID de rastreamento consistente	Trace	Média	7, 13, 15, 19
12	Falta de diretrizes de instrumentação para novos serviços	Geral	Média	15, 17, 18
13	Negligência na manutenção regular	Geral	Média	12, 23, 24
14	Observabilidade apenas para produção	Geral	Média	1, 14, 25
15	*Logs* duplicados	*Log*	Média	9, 3, 4
16	Falta de *logging* estruturado	*Log*	Média	3, 15, 27
17	Métricas excessivas	Métrica	Média	10, 13, 20
18	Monitorar apenas o que o cliente vê	Métrica	Média	18, 22, 27
19	Subestimar a importância dos traces	Trace	Média	15, 17, 18
20	Acumular dados desnecessariamente	Geral	Baixa	16, 18, 19
21	Ter apenas *dashboards* baseados em gráficos de séries temporais	*Dashboard*	Baixa	7, 18
22	Inconsistência entre ambientes	Geral	Baixa	15, 19
23	Solução única para todos	Geral	Baixa	14, 25
24	Tratar a observabilidade como uma ferramenta apenas para incidentes	Geral	Baixa	18, 24
25	Uso inadequado de níveis de *log*	*Log*	Baixa	9, 27
26	Usar *logs* de acesso à API para depuração	*Log*	Baixa	14, 25
27	Má compreensão das métricas	Métrica	Baixa	13, 19
28	Ignorar métricas derivadas	Métrica	Baixa	26, 27
29	Começar o trace no API *Gateway* é superestimado	Trace	Baixa	14, 25
30	Transações sintéticas preguiçosas	Geral	Baixa	19
31	Imposição de ferramentas	Geral	Baixa	19
32	Negligência na retenção de dados	Geral	Baixa	24
33	Falha ao monitorar *logs* de erro	*Log*	Baixa	23
34	Uso inadequado de *logs* para coleta de métricas	*Log*	Baixa	27
35	Confiar apenas no monitoramento de APIs	Métrica	Baixa	25
36	A métrica burra e gigante	Métrica	Baixa	19
37	Spans de trace muito longos	Trace	Baixa	15

Fonte: Autoria própria.

por parte dos usuários. Ademais, a **QP4** é também contemplada por meio do catálogo, uma vez que foram apresentadas soluções voltadas à mitigação ou prevenção dos APs de observabilidade, por meio da aplicação de boas práticas, promovendo, assim, uma observabilidade mais eficaz e confiável.

5.2 Solução do Anti-padrão Alertas Excessivos

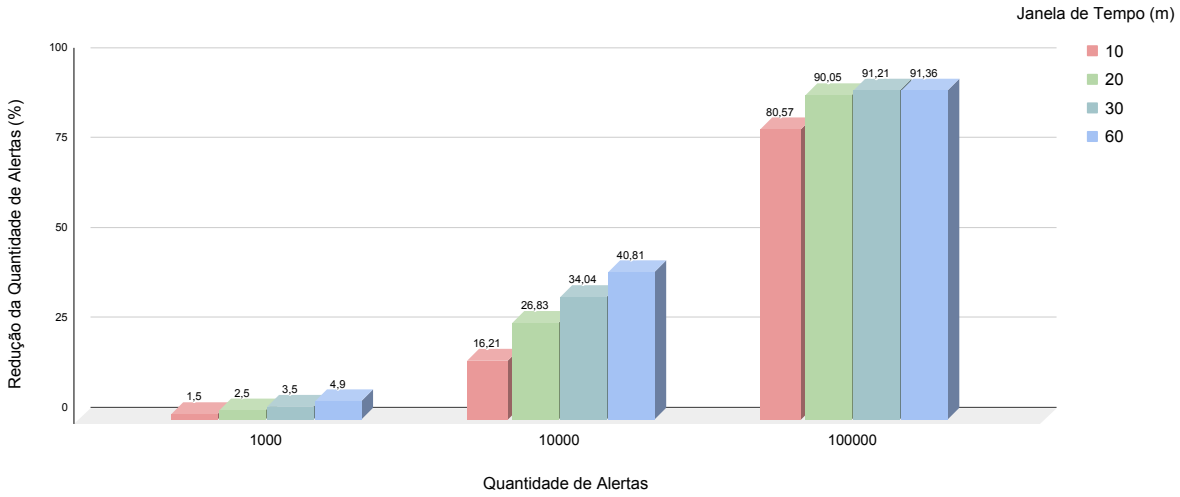
Com o intuito de complementar os estudos em torno dos APs de Observabilidade e mitigar as problemáticas envolvidas nesse tema, ainda que essa não seja a contribuição central desta tese, foi desenvolvida uma solução experimental que busca reduzir a quantidade de alertas apresentados às equipes responsáveis pelo monitoramento. Dessa forma, objetiva-se minimizar o AP de Alertas Excessivos. A solução desenvolvida[4] baseia-se no agrupamento dos alertas, mecanismo previamente descrito no catálogo, considerando uma janela de tempo configurável e os campos que compõem a estrutura de cada alerta. Utilizou-se a estrutura padrão apresentada nos exemplos anteriores, contendo: `host`, `name`, `service`, `severity`, `value` e *timestamp*. O agrupamento é realizado com base nos quatro primeiros campos, que conjuntamente representam a identidade semântica do alerta, enquanto os campos subsequentes variam unicamente quanto ao valor observado e ao instante de captura. A janela de tempo atua como delimitadora do conjunto, reunindo múltiplos alertas equivalentes emitidos dentro do mesmo intervalo em grupos únicos.

Para validar o funcionamento da solução, foram utilizados três conjuntos de dados sintéticos contendo diferentes volumes de alertas: (i) 1.000, (ii) 10.000 e (iii) 100.000 alertas. Adicionalmente, foi realizada a variação da janela temporal do algoritmo em quatro configurações distintas: (i) 10, (ii) 20, (iii) 30 e (iv) 60 minutos. Os *datasets* podem ser visualizados no seguinte link: <https://tinyurl.com/alerts-datasets>

A Figura 22 apresenta os resultados obtidos para cada combinação de volume de alertas e janela temporal. Conforme observado, janelas menores resultam em menor redução, dado que produzem maior quantidade de grupos distintos. Em contrapartida, à medida que o volume de alertas aumenta, o impacto da redução se torna proporcionalmente mais expressivo. Ou seja, quanto mais elevada a carga de alertas enfrentada pelas equipes de Observabilidade, maior tende a ser o ganho obtido pelo agrupamento, contribuindo para diminuir o desgaste decorrente da sobrecarga de informações e, portanto, mitigando a incidência do AP de Alertas Excessivos.

[4] <https://github.com/andersonalmada/detect-antipatterns/blob/master/alerts/alert_solution.py>

Figura 22 – Redução da quantidade de alertas.



Fonte: Autoria própria.

No experimento realizado, a redução da quantidade de alertas variou entre 1,5% e 91,36%, a depender do volume inicial de dados e da configuração da janela temporal. Esses resultados demonstram que mesmo uma solução simples, pode gerar benefícios tangíveis na prática de observabilidade. Além disso, os achados indicam que o agrupamento é proporcionalmente mais eficaz em cenários de maior sobrecarga de alertas, sugerindo que sua adoção pode ser particularmente relevante para sistemas distribuídos de larga escala.

5.3 Avaliação com Especialistas

Com o intuito de avaliar o catálogo, elaborou-se um formulário de avaliação baseado na metodologia de Duarte *et al.* (2024). O formulário é composto por uma seção introdutória, na qual, apresenta-se os autores da pesquisa, os objetivos do formulário e do catálogo, a descrição das seções e das questões, bem como a definição do público-alvo e o tempo estimado para responder. Também foi disponibilizado aos avaliadores o *link* de acesso ao portal contendo o catálogo. O formulário avaliou qualitativamente o catálogo em termos de clareza, legibilidade, relevância, utilidade e avaliação geral. Ele foi estruturado em duas partes: a primeira voltada à autoavaliação dos participantes e a segunda com questões qualitativas objetivas sobre o catálogo. As respostas a essas questões seguem a escala de Likert (concordo totalmente, concordo, neutro, discordo e discordo totalmente). Se caso o participante escolher “discordo” ou “discordo totalmente”, era pedido para ele justificar o porque da escolha. Além disso, uma pergunta aberta optativa foi fornecida para os participantes, no sentido de sugerir melhorias para o catálogo ou

realizar alguma observação.

Além disso, foi distribuído o formulário por meio do Google Forms, selecionando os participantes tanto por contatos pessoais dos autores quanto em grupos especializados em observabilidade e redes de profissionais da área. A seleção via contatos pessoais foi direcionada a profissionais com experiência prática com observabilidade. Adicionalmente, o primeiro autor realizou contatos diretos a profissionais da área via LinkedIn. Essa abordagem incluiu reuniões por videoconferência para apresentar o estudo e solicitar apoio na disseminação do formulário. Posteriormente, ainda pelo LinkedIn, outros profissionais foram abordados, especialmente SREs e DevOps, todos com vivência prática em observabilidade. Como o objetivo era contar apenas com especialistas na avaliação, foi desconsiderado as respostas de participantes com menos de um ano de experiência na área (4 participantes). Ao todo, 60 avaliadores com mais de um ano de experiência em observabilidade participaram da avaliação.

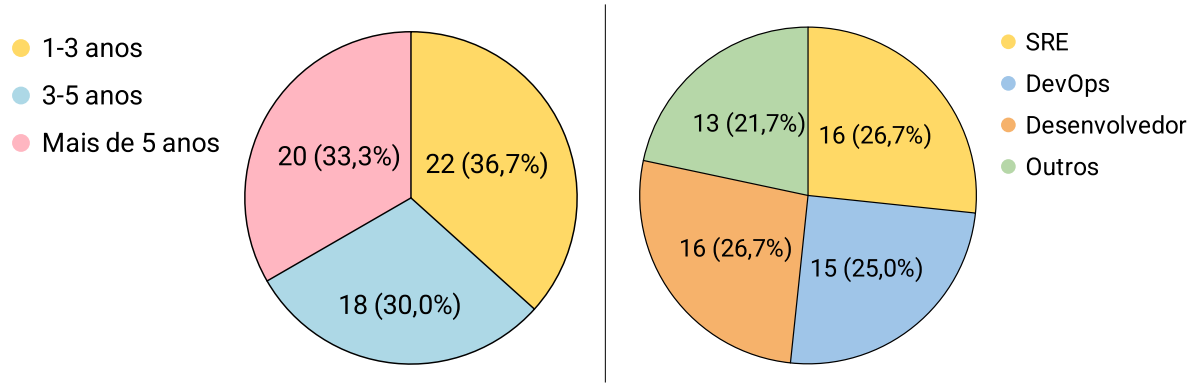
5.3.1 Perfil dos Avaliadores

A primeira parte do questionário teve como objetivo analisar o perfil dos avaliadores. Assim, profissionais *Site Reliability Engineering* (SRE) representam 26,7% dos avaliadores, enquanto profissionais de DevOps correspondem a 25% e desenvolvedores de software a 26,7%. O restante varia entre profissionais de Gestão de TI, Infraestrutura, Pesquisa acadêmica, Qualidade de Software, Segurança da Informação e Arquitetura de software. Além disso, pelo menos 80% dos avaliadores atuam no mercado (foi colocado esse campo como opcional), em empresas nacionais e internacionais, que atuam nos mais variados ramos, desde telecomunicação, serviços financeiros, desenvolvimento de software, recursos humanos, entre outros. Com relação à experiência com observabilidade, 36,7% dos avaliadores afirmaram ter entre 1 e 3 anos de experiência. Cerca de 30% indicaram ter entre 3 e 5 anos de atuação na área. Por fim, 33,33% afirmaram possuir mais de 5 anos de experiência trabalhando com observabilidade.

5.3.2 Resultados Obtidos na Avaliação

A Figura 24 resume os resultados das perguntas qualitativas em escala de Likert respondidas pelos participantes. Somando-se as respostas positivas (por exemplo, “concordo” e “concordo totalmente”), obteve-se uma taxa de aprovação de 93,33% para a avaliação geral, 95% para relevância, 96,67% para utilidade, 83,33% para legibilidade e 83,33% para clareza. Tanto para legibilidade quanto para clareza, houve críticas negativas (respostas “discordo”) de

Figura 23 – Perfil dos avaliadores.



(a) Experiência com observabilidade. (b) Atuação profissional.

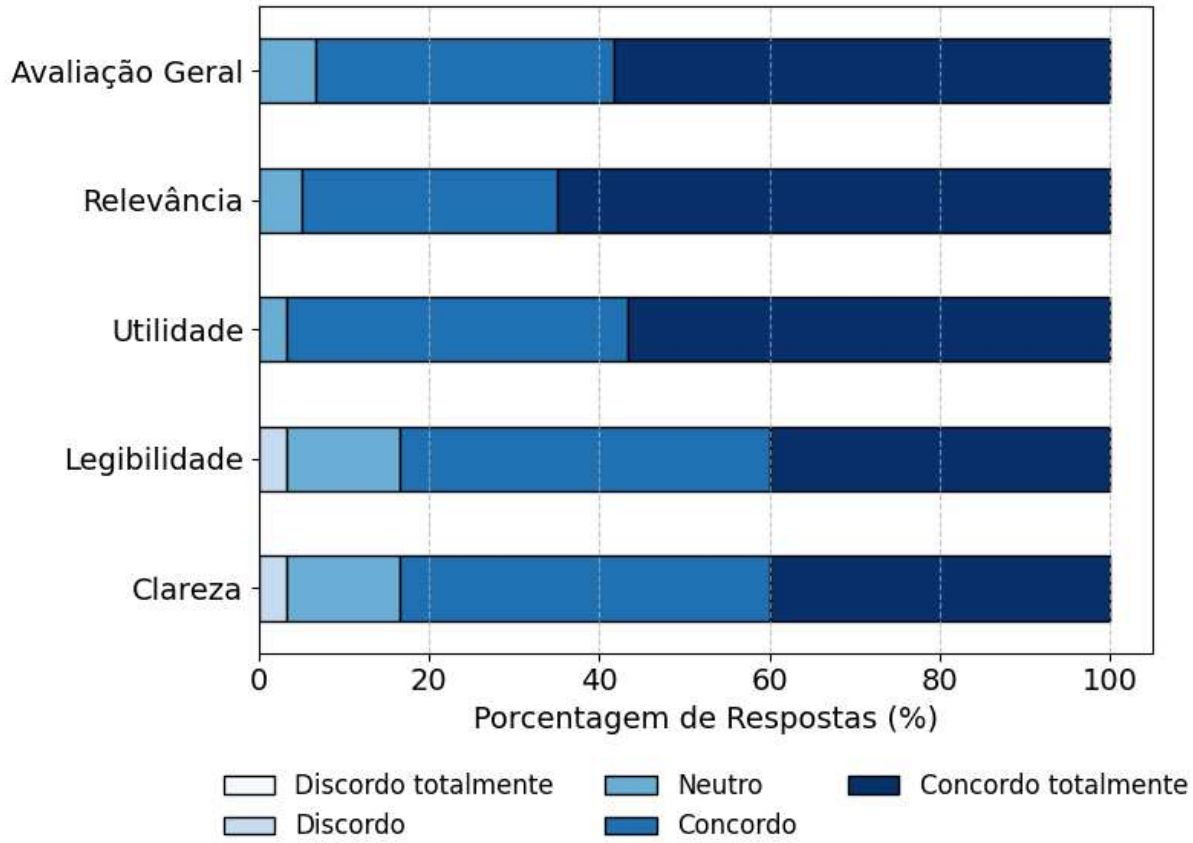
Fonte: Autoria própria.

dois participantes, os quais sugeriram tornar o portal mais intuitivo para facilitar o acesso às informações. Essa sugestão foi incorporada com a adição de novos filtros, tanto por categoria quanto por relevância, visando otimizar a navegação. Os resultados obtidos evidenciam uma forte aceitação por parte dos participantes em relação ao catálogo de AP de observabilidade. As altas taxas de aprovação, especialmente nos critérios de utilidade e relevância, indicam que o conteúdo atende às expectativas do público-alvo e possui aplicabilidade prática. A avaliação geral reforça essa percepção positiva, e as críticas pontuais demonstram um engajamento construtivo por parte dos usuários.

Com relação à pergunta aberta, alguns participantes deixaram comentários adicionais que reforçam os resultados. Um deles afirmou que o catálogo possui um excelente conteúdo de pesquisa, sendo relevante, abrangente e com alto nível de qualidade, detalhamento e profundidade. Outro destacou que o material está muito bom e aborda diversas dores enfrentadas diariamente por profissionais de SRE. Também foi ressaltado que, para quem deseja iniciar na área de monitoramento de aplicações, é essencial conhecer os possíveis erros desde o início do desenvolvimento, a fim de preveni-los. Um participante comentou: “o tema é superinteressante, frequentemente me deparo com situações de *logs* sem informações úteis sobre a realidade da exceção, ou até mesmo com a ausência completa desses registros. Entendo completamente a importância do estudo”.

Por fim, houve quem enxergasse o catálogo como uma verdadeira “bússola” para empresas em diferentes níveis de maturidade na adoção da observabilidade, o que evidencia o potencial do material como referência prática e estratégica para o setor. Os comentários reforçam o valor do catálogo, destacando sua profundidade, aplicabilidade no cotidiano de profissionais

Figura 24 – Avaliação do catálogo.



Fonte: Autoria própria.

da área e importância para iniciantes no campo da observabilidade.

5.4 Ameaças à Validade do Catálogo

Esta seção apresenta as potenciais ameaças à validade do estudo do catálogo, estruturadas de acordo com a mesma classificação utilizada para a taxonomia (ver Seção 4.5). Apesar da adoção de estratégias voltadas à mitigação de riscos, é necessário reconhecer a existência de limitações que podem influenciar tanto a interpretação dos resultados quanto a abrangência de sua generalização.

Validade de Construção: Existe o risco de que algumas fontes relevantes não tenham sido identificadas durante o processo de seleção. Para mitigar esse problema, foram conduzidos um Mapeamento Sistemático da Literatura (MSL) e um Mapeamento da Literatura Cinzenta (MLC), ambas fundamentadas em um conjunto diversificado de termos de busca. No caso da MSL, a busca foi realizada em bases de dados acadêmicas consolidadas e amplamente utilizadas na área de computação. A MLC, por sua vez, incluiu múltiplas fontes não acadêmicas, como

blogs técnicos, sites oficiais e vídeos especializados, o que permitiu ampliar a cobertura e incluir diferentes perspectivas práticas. Além disso, a definição de um protocolo de revisão contribuiu para reduzir vieses durante a etapa de seleção.

Validade Interna: A extração de dados pode ter sido afetada por ambiguidades ou inconsistências nas fontes analisadas. Para lidar com esse risco, as informações recuperadas do MSL e do MLC foram estruturadas a partir das questões de pesquisa previamente estabelecidas no protocolo de revisão. Nos casos em que surgiram dúvidas ou divergências, os orientadores discutiram coletivamente até alcançar consenso, de modo a assegurar maior confiabilidade nos dados extraídos.

Validade Externa: A generalização dos resultados está condicionada ao escopo das fontes selecionadas. Embora o uso combinado de MSL e MLC tenha ampliado a diversidade das evidências, é possível que determinados contextos, práticas ou abordagens de observabilidade não tenham sido contemplados.

Validade de Conclusão: Entre as ameaças potenciais estão a síntese dos dados e a avaliação dos especialistas. O risco relacionado à síntese foi mitigado por meio da utilização de categorias para agrupar os APs identificados, complementadas por estatísticas descritivas para sumarizar os resultados. Quanto à avaliação dos especialistas, ainda que o número de participantes tenha sido limitado, buscou-se incluir apenas profissionais com experiência prévia significativa em observabilidade, excluindo aqueles com menos de um ano de atuação na área. Essa escolha teve como objetivo assegurar a relevância e a qualidade das contribuições fornecidas.

5.5 Considerações Finais

Este capítulo apresentou o catálogo de APs de observabilidade para aplicações baseadas em microsserviços desenvolvido nesta tese, destacando sua metodologia de construção, fontes de dados utilizadas, estrutura de documentação e portal digital de acesso. Foram detalhados os passos do mapeamento sistemático da literatura e da literatura cinzenta, permitindo identificar 37 APs de observabilidade, classificados por categoria e relevância.

Além disso, foi apresentada uma solução experimental complementar para o AP de alertas excessivos, cujo agrupamento reduziu de forma expressiva a quantidade de alertas em cenários com alta carga operacional. A avaliação do catálogo realizada com 60 especialistas demonstrou ampla aceitação do catálogo, reforçando seu valor prático para profissionais e pesquisadores da área. Os comentários recebidos também indicam que o catálogo oferece uma

ferramenta eficaz para apoiar decisões, disseminar conhecimento e auxiliar na mitigação de problemas recorrentes em observabilidade.

Por fim, as potenciais ameaças à validade do estudo foram discutidas, contemplando aspectos de construção, interna, externa e de conclusão. Em síntese, o catálogo consolidado neste capítulo constitui um recurso relevante e estruturado, capaz de orientar boas práticas e prevenir erros comuns em observabilidade, servindo como um complemento essencial à taxonomia apresentada no capítulo anterior.

Assim, permanece uma lacuna na literatura no que diz respeito à detecção automática de APs de observabilidade. É justamente essa lacuna que motiva o desenvolvimento do Observa. O próximo capítulo apresenta o *framework* Observa desenvolvido nesta tese, detalhando sua arquitetura, funcionamento, implementação, implantação e uma prova de conceito para demonstrar sua viabilidade técnica e aplicabilidade prática.

6 OBSERVA: *FRAMEWORK* PARA DETECÇÃO AUTOMÁTICA DE ANTI-PADRÕES DE OBSERVABILIDADE

Este capítulo apresenta o Observa de forma detalhada. Na Seção 6.1, é descrita a arquitetura do *framework*, enquanto na Seção 6.2 são apresentadas as abstrações fundamentais que o sustentam. Já na Seção 6.3, discutem-se os princípios de *design* que orientaram seu desenvolvimento. Na Seção 6.4, é abordado o funcionamento operacional do Observa. Por fim, o capítulo apresenta a interação do usuário com o *framework* por meio da interface gráfica (Seção 6.5). Neste trabalho, os usuários são profissionais que trabalham com observabilidade.

6.1 Visão Geral da Arquitetura

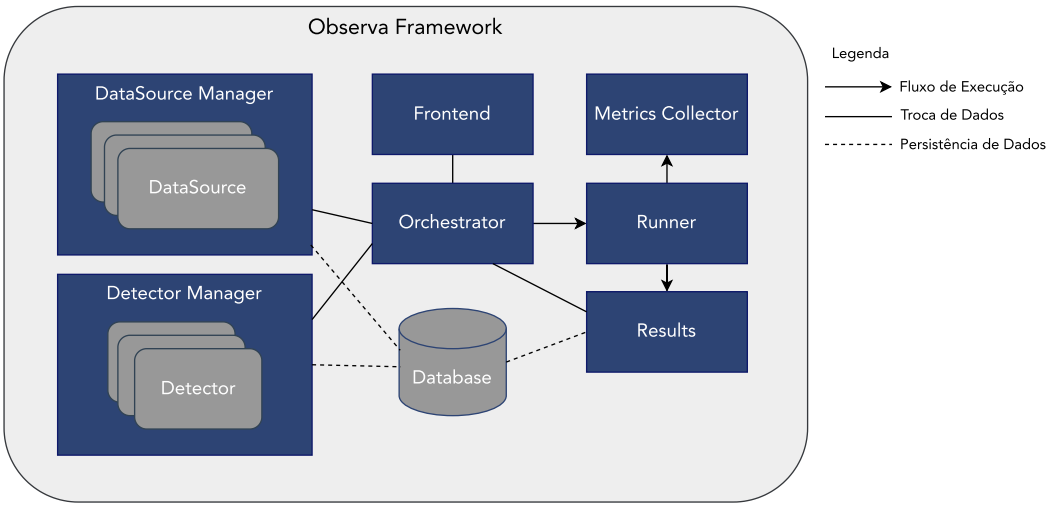
O Observa é um *framework* projetado para apoiar a Detecção Automática de Anti-Padrões (APs) de Observabilidade em aplicações baseadas em microsserviços. Diferentemente de ferramentas tradicionais de observabilidade, que se concentram na coleta, visualização ou correlação de sinais, o Observa avalia a qualidade da própria observabilidade implementada, identificando más práticas relacionadas ao uso de métricas, *logs*, *traces*, alertas e *dashboards*. Para isso, o *framework* integra fontes heterogêneas de dados de telemetria e aplica detectores especializados, cada um responsável por analisar indícios de um AP específico.

Do ponto de vista arquitetural, o Observa foi concebido como um *framework* extensível e tecnologicamente neutro, permitindo a incorporação progressiva de novas fontes de dados e estratégias de detecção sem alterações em seu núcleo. Além de apoiar o uso prático do catálogo de APs, o Observa também foi projetado para atender a requisitos científicos, como reprodutibilidade experimental, rastreabilidade das análises e comparação sistemática de resultados.

A Figura 25 apresenta a visão geral da arquitetura do *framework* Observa, evidenciando seus principais módulos e as interações estabelecidas entre eles. A arquitetura é organizada de forma modular, com uma separação de responsabilidades entre os componentes, o que permite maior controle sobre o processo de análise e favorece a extensibilidade do *framework*. A arquitetura é composta pelos seguintes componentes: `Orchestrator`, `Runner`, `DataSource Manager`, `DataSource`, `Detector Manager`, `Detector`, `Metrics Collector`, `Results`, `Database` e `Frontend`. A descrição detalhada do papel e das responsabilidades de cada um desses componentes é apresentada a seguir.

O `Orchestrator` é o componente central da arquitetura e o responsável por coorde-

Figura 25 – Arquitetura do Observa.



Fonte: Autoria própria.

nar o fluxo de execução do *framework* Observa. Ele atua como o núcleo de controle lógico do sistema, sendo encarregado de receber as interações dos usuários, por meio do `Frontend`, e de iniciar o funcionamento dos demais módulos.

O `DataSource Manager` é o módulo responsável por gerenciar as fontes de dados utilizadas, que são representadas pelo `DataSource` no *framework*, bem como de ter o registro das fontes disponíveis. Esse componente é acionado pelo `Orchestrator`.

O `DataSource` representa uma fonte concreta de dados de observabilidade integrada ao Observa. Esse componente encapsula a lógica específica de acesso a uma determinada fonte, como métricas, *traces* ou *logs*, sendo responsável pela coleta e preparação dos dados para análise. A utilização do `DataSource` permite desacoplar os mecanismos de coleta do restante da arquitetura, favorecendo a extensibilidade do *framework*.

O `Detector Manager` é responsável por gerenciar o conjunto de detectores disponíveis no *framework*. Esse componente seleciona e organiza os detectores que devem ser executados com base nos `DataSource` e no objetivo da análise. Esse componente é acionado pelo `Orchestrator`.

O `Detector` é o componente encarregado da análise dos dados de observabilidade. Cada detector implementa uma lógica específica de detecção, voltada à identificação de APs. Esses detectores constituem o núcleo analítico do Observa, podendo empregar heurísticas baseadas em regras ou estratégias fundamentadas na literatura.

O `Runner` é responsável pela execução efetiva das detecções definidas pelo usuário. Esse componente materializa o plano de execução estabelecido, realizando as operações de coleta

de dados, acionamento dos detectores e persistência dos resultados. Ao separar a coordenação da execução, o `Runner` contribui para uma arquitetura mais flexível e extensível, permitindo que diferentes estratégias de execução sejam adotadas sem impactar no *framework*.

O `Results` representa as saídas finais do processo de análise realizado pelo Observa. Esse componente consolida os achados produzidos pelos detectores, como a identificação de APs e as razões dessa detecção nos elementos enviados pelos `DataSource`. Os resultados podem ser enviados ao `Frontend` para visualização e avaliação.

O `Metrics Collector` é responsável pela coleta de métricas provenientes das execuções do Observa. Esse componente atua na captura sistemática de dados quantitativos relacionados ao comportamento da detecção. Atualmente, a métrica coletada é a de tempo de execução das detecções pelo Observa.

O `Database` é responsável pela persistência dos dados coletados e dos resultados gerados pelo *framework*. Esse componente garante o armazenamento estruturado das informações, possibilitando análises posteriores, comparações entre diferentes execuções e a reprodutibilidade dos experimentos conduzidos.

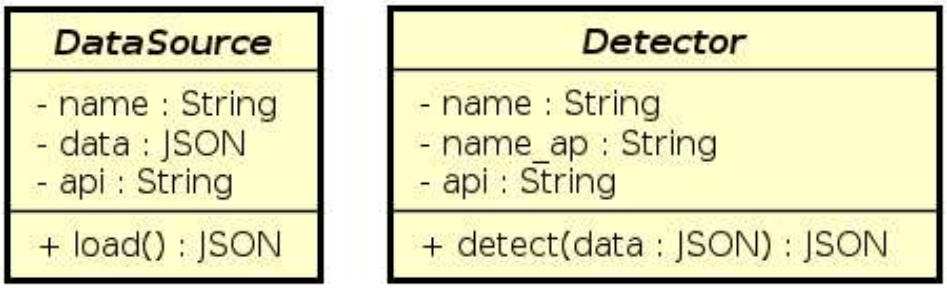
Por fim, o `Frontend` é um componente dedicado à interação do usuário com o Observa, atuando como a camada de apresentação do *framework*. Por meio desse componente, é possível realizar o gerenciamento completo dos `DataSources` e dos `Detectors`, incluindo a seleção dos módulos utilizados nas detecções, a configuração de novas fontes de dados e detectores, o disparo das análises de APs, bem como a visualização dos resultados e do histórico de execuções.

6.2 Abstrações Arquiteturais do Observa

Enquanto a seção anterior apresentou a organização estrutural do Observa em termos de seus componentes arquiteturais, esta seção descreve as principais abstrações que fundamentam o projeto do *framework*. Essas abstrações visam garantir independência tecnológica, modularidade e extensibilidade, por meio da definição de contratos entre fontes de dados e mecanismos de detecção.

As principais abstrações do Observa são representadas pelas interfaces `DataSource` e `Detector`. A abstração `DataSource` (ver Figura 26a) padroniza o acesso a diferentes origens de dados de observabilidade, como métricas, *logs*, *traces* e alertas, independentemente da tecnologia subjacente.

Figura 26 – Classes abstratas do Observa.



(a) Fonte de dados. (b) Detector.

Fonte: Autoria própria.

Conforme ilustrado na Figura 26a, a abstração `DataSource` encapsula as informações necessárias para a identificação e configuração de uma fonte de dados, incluindo um identificador (`name`) e os dados da fonte, que podem ser fornecidos de forma estática, em formato `JSON` (`data`), ou de forma dinâmica, por meio de um *endpoint* (`API`). Os dados coletados, independentemente da forma, devem estar no formato `JSON`. A independência tecnológica das fontes está relacionada à possibilidade de utilização de qualquer tecnologia capaz de fornecer os dados nesse formato. Além disso, essa abstração define a operação `load()`, responsável pela coleta dos dados de observabilidade e pela sua disponibilização estruturada aos demais componentes do *framework*.

De forma complementar, a abstração `Detector` (ver Figura 26b) define o contrato responsável pela análise dos dados de observabilidade e pela identificação de APs. Essa abstração encapsula a lógica analítica necessária para avaliar conjuntos específicos de dados, operando de maneira isolada em relação às fontes de coleta.

A abstração `Detector` é composta por atributos que identificam o detector (`name`), o AP associado à análise (`name_ap`) e o endereço da interface de comunicação (`API`), caso seja remoto. O método `detect(data:JSON)` define a operação central dessa abstração, sendo responsável por processar os dados fornecidos pelas instâncias de `DataSource` e retornar, em formato estruturado (ver Seção B.1.2), os resultados da detecção.

Em conjunto, essas abstrações permitem a incorporação de novas fontes de dados e de diferentes estratégias de detecção de forma independente, sem a necessidade de modificações no núcleo arquitetural do Observa, reforçando os princípios de extensibilidade e baixo acoplamento do *framework*.

6.3 Princípios de *Design*

O desenvolvimento do Observa foi orientado por um conjunto de princípios arquiteturais que busca ofertar robustez, extensibilidade e alinhamento com objetivos científicos e práticos. Diferentemente das seções anteriores, que descreveram a arquitetura e as abstrações do *framework*, esta seção tem como objetivo explicitar os critérios conceituais que fundamentaram as decisões de *design* adotadas ao longo de seu desenvolvimento. Os principais princípios adotados são apresentados a seguir.

O Observa adota o princípio da **Modularidade e Separação de Preocupações**, no qual responsabilidades como coleta de dados, análise de APs, execução e persistência são tratadas de forma independente. Essa separação reduz o acoplamento entre componentes, favorece a clareza arquitetural e cria uma base adequada para a evolução incremental do *framework* sem comprometer sua estrutura central.

A partir dessa organização modular, o Observa incorpora o princípio da **Extensibilidade Orientada a Anti-padrões**. A arquitetura foi concebida para permitir a incorporação progressiva de novos APs, fontes de dados e estratégias de detecção. Cada AP é tratado como uma unidade analítica autônoma, encapsulada em detectores independentes, o que possibilita a ampliação do *framework* sem a necessidade de reestruturações no núcleo arquitetural.

Considerando a heterogeneidade de ferramentas e ecossistemas de observabilidade, o Observa também adota o princípio da **Independência Tecnológica**. A interação com dados de telemetria ocorre por meio de abstrações padronizadas, assegurando compatibilidade com diferentes ambientes e preservando a neutralidade tecnológica da solução, aspecto essencial para sua adoção em cenários diversos.

No contexto científico desta pesquisa, a arquitetura do Observa foi projetada para suportar execuções controladas e repetíveis, materializando o princípio da **Reprodutibilidade Experimental**. Esse princípio permite que a detecção de APs produza resultados consistentes independentemente do ambiente de execução, sendo fundamental para a validade experimental de estudos empíricos e para a comparação sistemática de resultados entre diferentes cenários e configurações.

Complementarmente, o Observa adota o princípio da **Transparência e Rastreabilidade**. Todas as etapas do processo analítico, incluindo as fontes utilizadas, os detectores selecionados, os resultados obtidos e os metadados de execução, são explicitamente registradas. Essa característica garante auditabilidade, possibilita análises retrospectivas e sustenta estudos

longitudinais sobre a evolução da qualidade da observabilidade ao longo do tempo.

6.4 Funcionamento Operacional

A compreensão do funcionamento interno do Observa requer não apenas a análise de sua arquitetura, mas também a explicitação dos fluxos operacionais que estruturam seu comportamento. Nesse sentido, os diagramas de atividades UML apresentados nesta seção desempenham um papel fundamental ao ilustrar as sequências de ações executadas pelo *framework* durante suas principais operações.

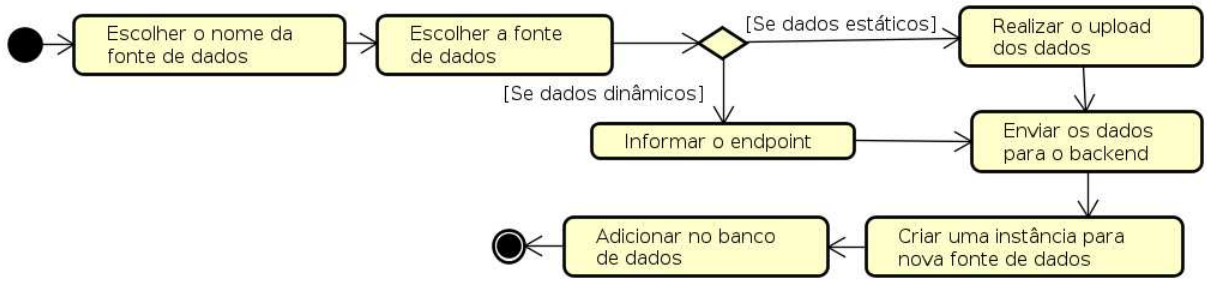
Esses diagramas capturam tanto os fluxos de uso voltados ao usuário, como a execução de detecções ou baseadas em histórico, quanto os processos internos responsáveis pela configuração e extensibilidade do *framework*, evidenciados na adição de novas fontes de dados e novos detectores. Cada diagrama representa uma perspectiva complementar do funcionamento do Observa, permitindo visualizar como seus componentes interagem, quais decisões condicionam o comportamento do sistema e de que maneira as atividades contribuem para a construção do resultado final.

6.4.1 Configurar uma Nova Fonte de Dados

A Figura 27 descreve o processo de cadastro de novas fontes de dados no Observa, elemento essencial para permitir que o *framework* se conecte a diferentes ambientes, aplicações ou mecanismos de monitoramento. O fluxo inicia-se com a **definição do nome da fonte de dados**, que servirá como identificador interno no sistema. Em seguida, o usuário escolhe o **tipo de fonte**. Caso a fonte seja composta por dados estáticos, o usuário deve realizar o ***upload* de um arquivo JSON** contendo as informações estruturadas necessárias. Esse arquivo é então enviado ao *backend*, que valida e processa seu conteúdo. Caso a fonte seja composta por dados dinâmicos, o usuário deve **informar o endpoint (API)** responsável por fornecer dados continuamente ou sob demanda. Da mesma forma, essas informações são **enviadas ao *backend*** para processamento.

Independentemente do tipo da fonte, o *backend* cria uma instância de uma classe que herda de `DataSource`, abstração responsável por padronizar a interação entre o *framework*, apresentado anteriormente, e diferentes origens de dados. Finalmente, a nova fonte é **adicionada ao banco de dados**, completando o fluxo. Este processo busca ofertar a extensibilidade do *framework* para novas fontes de dados.

Figura 27 – Diagrama de atividades de adicionar uma nova fonte de dados.



Fonte: Autoria própria.

6.4.2 Configurar um Novo Detector

A Figura 28 descreve o fluxo de cadastro de um novo detector no Observa, elemento que permite a ampliação progressiva do conjunto de APs que o *framework* é capaz de identificar. O processo tem início com a **definição do nome do anti-padrão** a ser representado pelo novo detector. Em seguida, o usuário **informa o nome do detector**, geralmente associado a uma implementação específica da lógica de análise.

Na etapa seguinte, o usuário **fornece o endpoint da API** em que o detector se encontra para realizar as heurísticas de detecção. Essas informações são então **enviadas ao *backend***. O *backend* cria, a partir dos dados recebidos, uma instância de uma classe que herda de `Detector`, abstração fundamental que padroniza a interface de detecção dentro do *framework*, apresentado anteriormente. A nova instância é validada e **registrada no banco de dados**, permitindo sua utilização nos fluxos de detecção. Da mesma forma que na fonte de dados, esse diagrama evidencia a natureza extensível do Observa, na qual novos detectores, e, por consequência, novos APs, podem ser incorporados sem comprometer a arquitetura existente.

Figura 28 – Diagrama de atividades de adicionar um novo detector.

Fonte: Autoria própria.

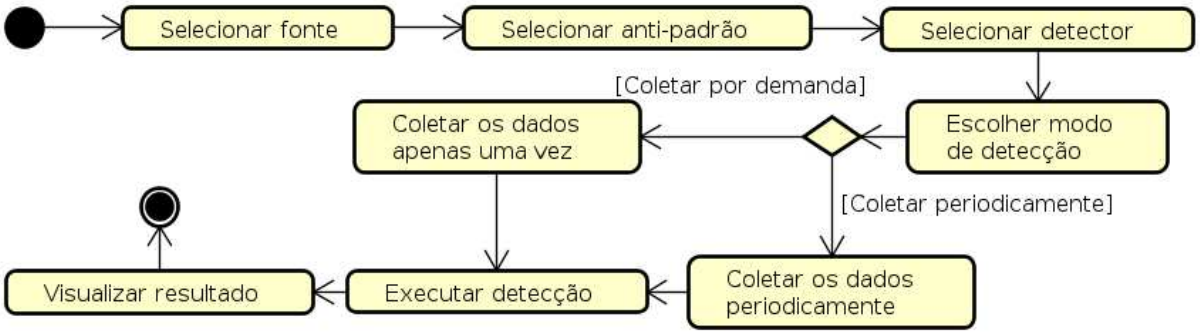
6.4.3 Execução pelo Usuário

A Figura 29 representa o fluxo básico de uso do Observa quando o objetivo é realizar uma detecção de um AP, utilizando uma fonte e um detector já configurados. O fluxo inicia-se com a seleção da fonte de dados, etapa na qual o **usuário escolhe um conjunto** específico de métricas, *logs* ou informações estruturadas que servirão como entrada para o processo de análise. Em seguida, o usuário **seleciona o anti-padrão** que deseja investigar. Cada anti-padrão possui um ou mais detectores associados, de modo que o passo subsequente consiste em **escolher o detector** a ser empregado.

Após a seleção, o usuário escolhe como será o modo de detecção que ele prefere. Tem duas opções: ele pode escolher que o coletor busque os dados **apenas uma vez**, a partir da fonte de dados escolhida, ou ele pode escolher que aconteça de **maneira periódica** (por exemplo, a cada um minuto).

Após toda a coleta de dados, o detector é invocado por meio da atividade **Executar detecção**, que aplica as regras, heurísticas ou algoritmos definidos para identificar indícios ou ocorrências do AP selecionado. O resultado desse processamento é então encaminhado para a etapa final do fluxo, **Visualizar resultado**, na qual o usuário recebe um relatório contendo a conclusão da análise.

Figura 29 – Diagrama de atividades da execução pelo usuário.



Fonte: Autoria própria.

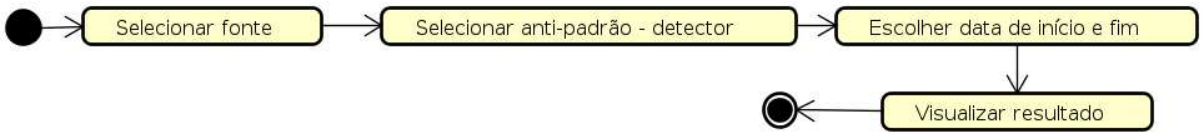
6.4.4 Visualizar o Histórico de Execuções

A Figura 29 representa como o usuário pode visualizar as detecções realizadas no *framework*. O processo se inicia com a **seleção da fonte** e a **definição do anti-padrão e detector** a ser utilizado. Além disso, o usuário precisa escolher em qual período ele deseja visualizar as

execuções realizadas. Esse intervalo pode corresponder, por exemplo, a períodos de instabilidade, janelas de manutenção, picos de uso ou qualquer outra situação relevante para a observabilidade do sistema.

Depois do período escolhido, o usuário consegue visualizar o resultado, que consiste de uma linha do tempo contendo todas as execuções envolvendo a fonte de dados e detector realizadas. E caso queira, ele pode visualizar o relatório completo de cada execução. Esse fluxo é fundamental para estudos evolutivos, auditorias técnicas e identificação de APs que se manifestam apenas ao longo do tempo.

Figura 30 – Diagrama de atividades de visualizar histórico.



Fonte: Autoria própria.

6.5 Interface Gráfica do Observa

Embora a interface gráfica não constitua o foco principal da contribuição científica desta tese, a sua implementação desempenha um papel relevante ao materializar, de forma operacional, as abstrações e os mecanismos arquiteturais desenvolvidos para o Observa. A interface atua como uma camada de interação que permite aos usuários configurar fontes de dados e detectores, iniciar execuções de detecção e analisar os resultados produzidos pelo *framework*, bem como visualizar o histórico dos resultados.

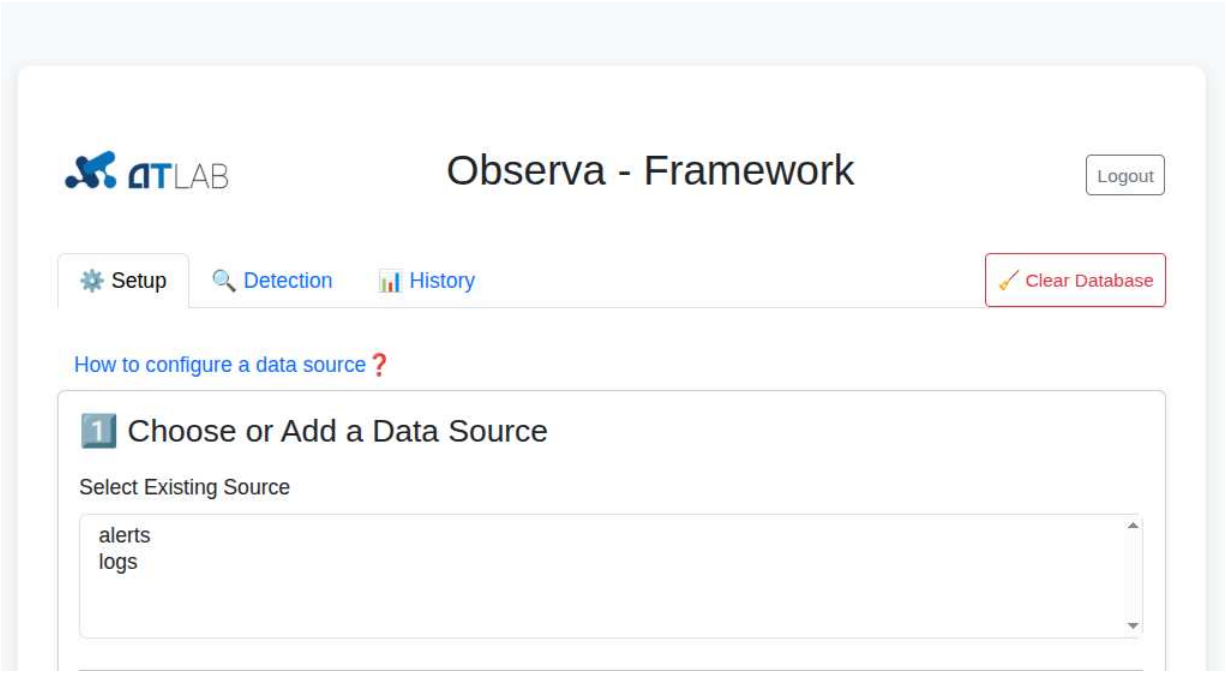
A implementação do Observa foi realizada utilizando tecnologias do ecossistema *Python* que, em conjunto, permitem a construção de um serviço leve, modular e facilmente integrável com ambientes de microsserviços. Os detalhes de implementação e implantação podem ser visualizados no Apêndice B. O repositório contendo o Observa está disponível em: <https://github.com/andersonalmada/detect-antipatterns>.

6.5.1 Escolha de Fontes de Dados

Nessa etapa, o usuário pode selecionar a fonte de dados a ser utilizada na execução das detecções de APs. Para isso, a interface apresenta exclusivamente as fontes de dados que já foram previamente cadastradas e implantadas no Observa.

Essa abordagem reflete o modelo operacional adotado pelo *framework*, no qual a definição e o registro das fontes de dados ocorrem em um momento anterior ao uso analítico propriamente dito. A Figura 31 ilustra as fontes de dados já cadastradas no sistema, sendo uma destinada à coleta de alertas e outra à coleta de *logs*.

Figura 31 – Tela de configuração para escolha de fontes de dados já cadastradas no Observa.



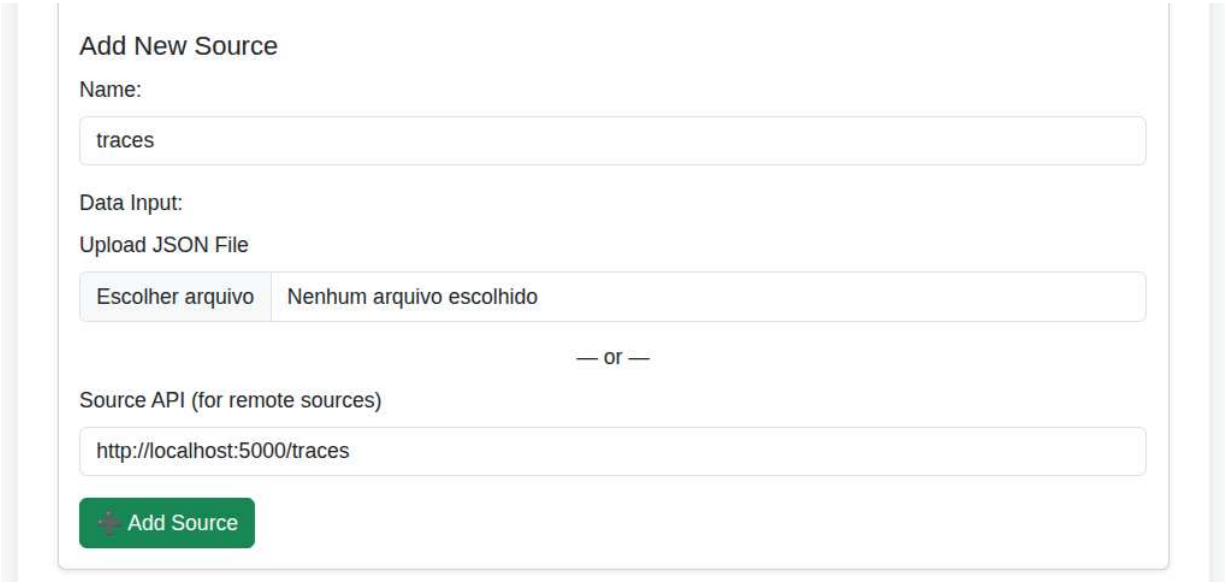
Fonte: Autoria própria.

6.5.2 Configuração de Novas Fontes de Dados

Além da seleção de fontes previamente cadastradas, o Observa disponibiliza um fluxo específico para a configuração e o registro de novas fontes de dados de observabilidade. Essa funcionalidade permite que o *framework* seja adaptado a diferentes ambientes, aplicações e tecnologias de monitoramento, reforçando sua natureza extensível e independente de fornecedores.

Durante o processo de configuração, o usuário define um identificador para a nova fonte de dados e seleciona o tipo de origem a ser utilizada. Como dito anteriormente, pode ser cadastrado tanto fontes de dados estáticas quanto fontes dinâmicas acessadas por meio de *endpoints*. Em ambos os casos, os parâmetros necessários para a coleta dos dados são informados pelo usuário e enviados ao *backend* para validação e processamento. A Figura 32 mostra a tela de criação de uma nova fonte de dados, denominada `traces`, configurada a partir do *endpoint* <http://localhost:5000/traces>.

Figura 32 – Tela de configuração de nova fonte de dados no Observa.

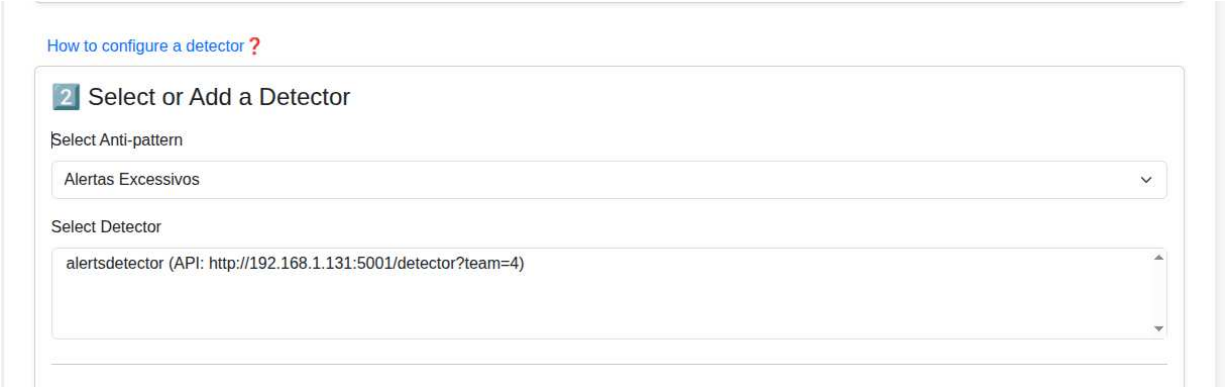


Fonte: Autoria própria.

6.5.3 Escolha de Detector

Após a seleção da fonte de dados ou o cadastro de uma nova fonte, o usuário deve escolher o detector de AP a ser utilizado na execução da análise. Nessa etapa, a interface do Observa apresenta os detectores que já foram previamente configurados e registrados no *framework* de acordo com cada AP. A Figura 33 ilustra a interface de seleção de detectores, na qual o usuário inicialmente escolhe o AP e, em seguida, seleciona o detector associado. Nesse exemplo, o AP de *alertas excessivos* é selecionado, bem como o detector vinculado a ele. A concepção do Observa prevê a possibilidade de coexistência de múltiplos algoritmos de detecção para um mesmo AP, permitindo a comparação de abordagens analíticas distintas.

Figura 33 – Tela de configuração de anti-padrão alertas excessivo e detector alertsdetector no Observa.



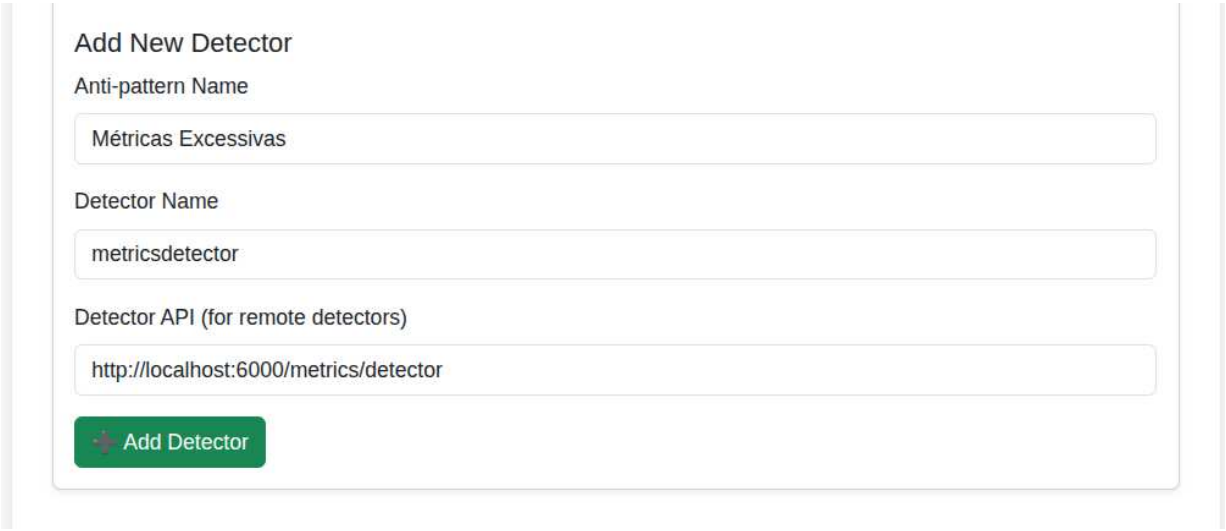
Fonte: Autoria própria.

6.5.4 Configuração de Novos Detectores

Além da seleção de detectores previamente cadastrados, o Observa disponibiliza um fluxo específico para a configuração e o registro de novos detectores de AP de observabilidade. Durante o processo de configuração, o usuário define o nome do AP ao qual o detector estará associado, bem como um identificador para o próprio detector. Em seguida, o usuário informa a origem da lógica de detecção, o qual é disponibilizado por meio de uma API externa. Os parâmetros necessários para o acesso ao detector são então enviados ao *backend* para validação e registro.

A Figura 34 mostra a tela de criação de um novo detector, voltado para a detecção do AP de *Métricas Excessivas*, denominado `metricsdetector` e configurado a partir do *endpoint* <http://localhost:6000/metrics/detector>.

Figura 34 – Tela de configuração de novo detector no Observa.



Fonte: Autoria própria.

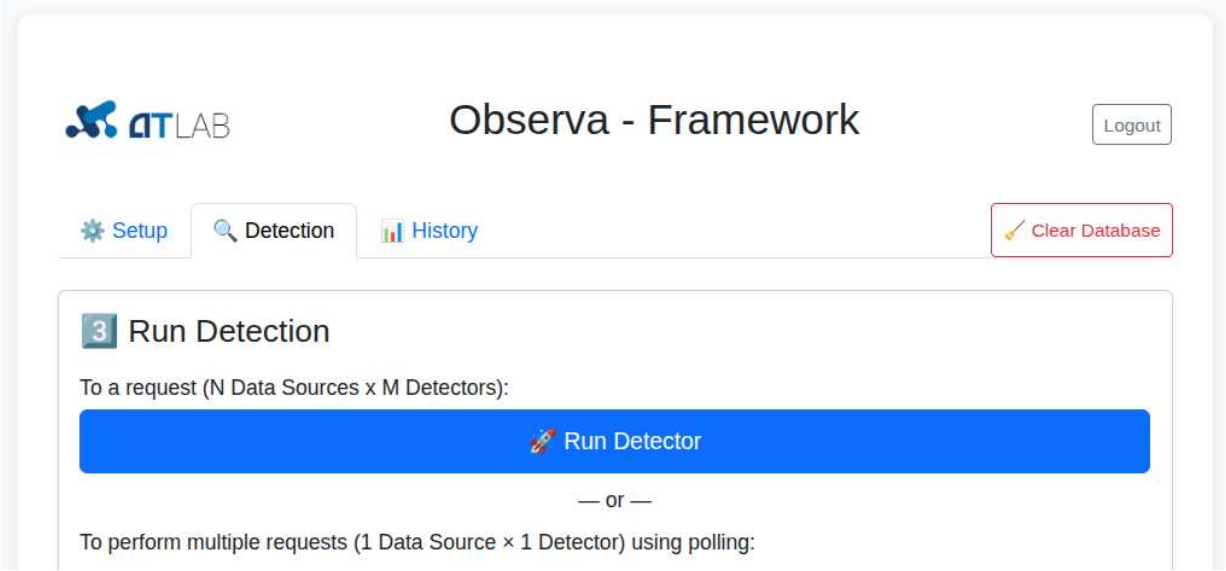
6.5.5 Execução da Detecção

A etapa de execução da detecção corresponde ao momento em que o Observa executa a coleta dos dados e a aplicação dos detectores de AP, com base nas fontes e detectores previamente selecionados. Essa execução pode ocorrer segundo dois modos distintos: execução por demanda e execução em modo de *polling* na coleta de dados.

Na execução por demanda (ver Figura 35), o usuário pode selecionar múltiplas fontes de dados e múltiplos detectores, desde que sejam semanticamente compatíveis, isto é, os

detectores escolhidos devem ser capazes de processar o tipo de dados fornecido pelas fontes selecionadas. Nesse modo, o Observa realiza a coleta dos dados de cada fonte de acordo com a demanda do usuário, encaminha os dados coletados aos respectivos detectores e executa a análise de forma imediata. Essa abordagem é adequada para inspeções pontuais e validações exploratórias, nas quais o objetivo é obter rapidamente um diagnóstico sobre a presença de APs.

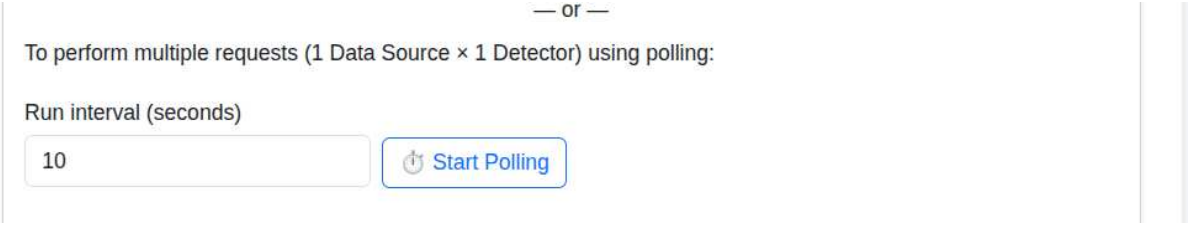
Figura 35 – Tela de escolha de detecção por demanda.



Fonte: Autoria própria.

O segundo modo de execução corresponde à detecção em modo de *polling* na coleta de dados (ver Figura 36). Nesse caso, a execução é restrita à seleção de uma única fonte de dados e um único detector. O Observa passa a realizar coletas periódicas a partir da fonte selecionada, em intervalos de tempo previamente definidos, acumulando os dados coletados ao longo do período de observação e executando a detecção.

Figura 36 – Tela de escolha de detecção em modo *polling*.



Fonte: Autoria própria.

O modo *polling* é particularmente adequado para a identificação de APs que se manifestam de maneira progressiva ou dependem de comportamento temporal, como crescimento excessivo de *logs*, aumento gradual no volume de métricas ou padrões repetitivos ao longo do

tempo.

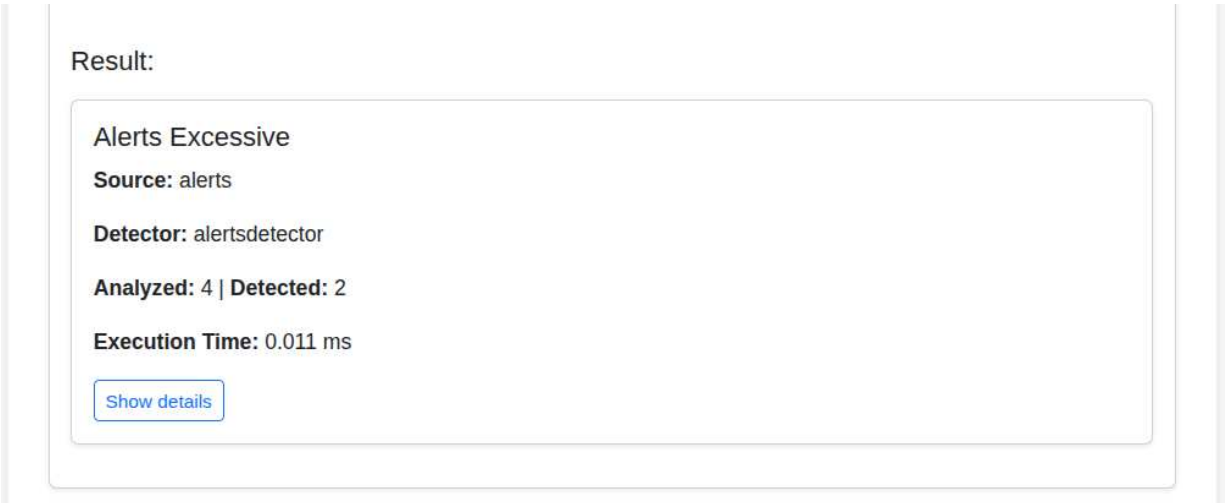
A distinção entre os dois modos de execução reflete uma decisão arquitetural do Observa voltada à flexibilidade operacional. Enquanto a execução por demanda privilegia agilidade e exploração, o modo de *polling* favorece análises longitudinais e a observação de tendências, ampliando o espectro de cenários em que o *framework* pode ser aplicado.

6.5.6 Visualização dos Resultados

Após a execução da detecção, o Observa disponibiliza os resultados ao usuário por meio de uma interface de visualização estruturada, cujo objetivo principal é apoiar a interpretação dos diagnósticos produzidos pelos detectores de APs. Essa visualização atua como a camada final do fluxo de análise, conectando os dados coletados, a lógica de detecção aplicada e as evidências geradas durante a execução.

Os resultados são apresentados de forma consolidada, incluindo informações quantitativas, como o número total de elementos analisados e a quantidade de ocorrências identificadas como AP, conforme ilustrado na Figura 37. Nessa execução, a fonte de dados `alerts` foi analisada pelo detector `alertsdetector`, associado ao AP de *alertas excessivos*. Ao todo, foram processados quatro itens, dos quais dois foram classificados como AP. Adicionalmente, o Observa registra o tempo de execução de cada detecção; nesse exemplo, o processamento foi concluído em 0.011 ms.

Figura 37 – Tela de visualização dos resultados.

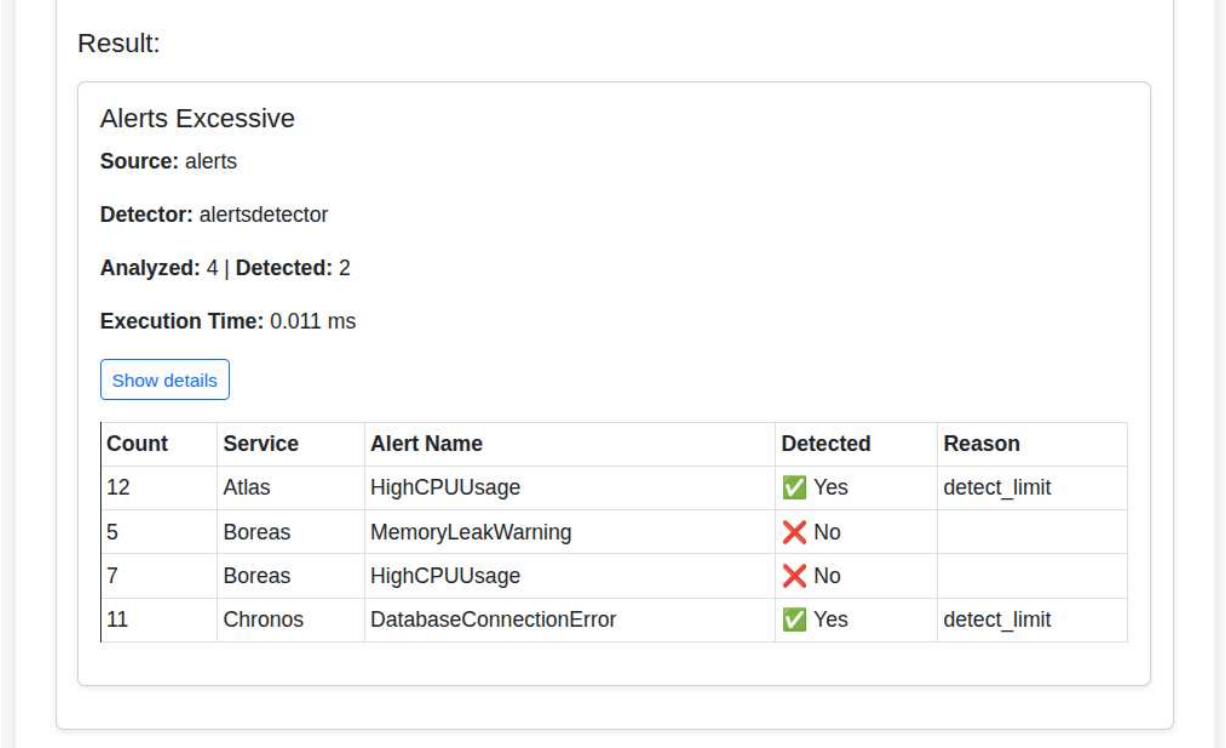


Fonte: Autoria própria.

Para cada elemento analisado, a interface mostra a justificativa ou a heurística que motivou a classificação como AP, conforme exemplificado na Figura 38, por meio da coluna

`reason`. Nesse caso, o valor apresentado indica que os dados de alertas enviados ao detector ultrapassaram o limite de dez alertas por hora (`detect_limit`), de acordo com o algoritmo de detecção implementado.

Figura 38 – Tela de visualização de detalhes dos resultados.



Fonte: Autoria própria.

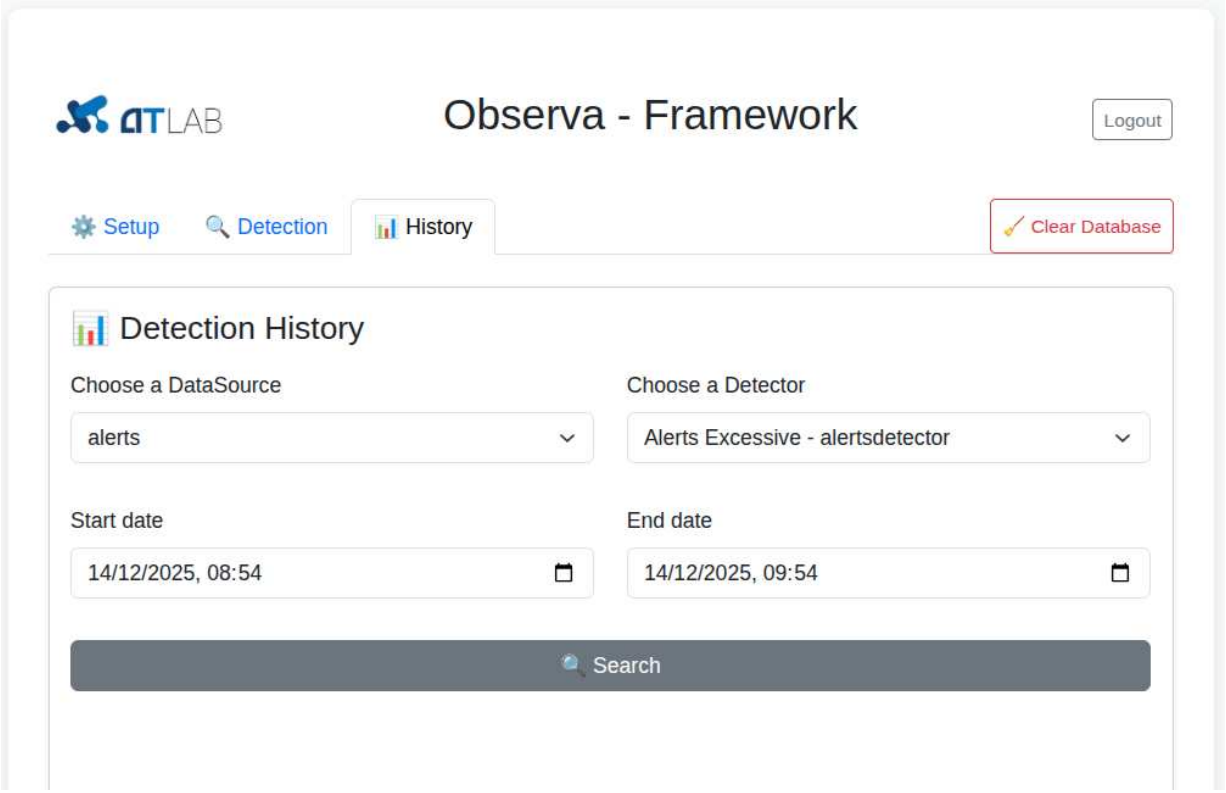
A visualização dos resultados, portanto, não se limita à apresentação de saídas dos detectores, mas desempenha um papel central na rastreabilidade e na interpretação dos diagnósticos.

6.5.7 Histórico das Detecções

Além da visualização imediata dos resultados, como dito anteriormente, o Observa mantém o registro de todas as execuções de detecção em um repositório persistente, possibilitando ao usuário consultar o histórico completo das análises realizadas ao longo do tempo. Cada execução registrada contém informações sobre a fonte de dados utilizada, o detector aplicado, o anti-padrão analisado, os resultados obtidos e os metadados associados à execução, como o instante e a duração do processamento.

A interface de histórico permite ao usuário selecionar as execuções associadas a cada par formado por fonte de dados e detector (ver Figura 39), bem como definir um intervalo

Figura 39 – Tela para escolher o par fonte de dados e detector e intervalo temporal do histórico das execuções.



Fonte: Autoria própria.

temporal específico para recuperar todas as execuções de detecção correspondentes, organizadas de forma cronológica. Essas execuções, como podem ser vistas na Figura 40, são apresentadas em formato de barras verticais, nas quais a altura total representa a quantidade de elementos analisados, enquanto a porção destacada em vermelho corresponde à quantidade de elementos classificados como AP pelo detector e a verde corresponde a quantidade de elementos não classificados como AP.

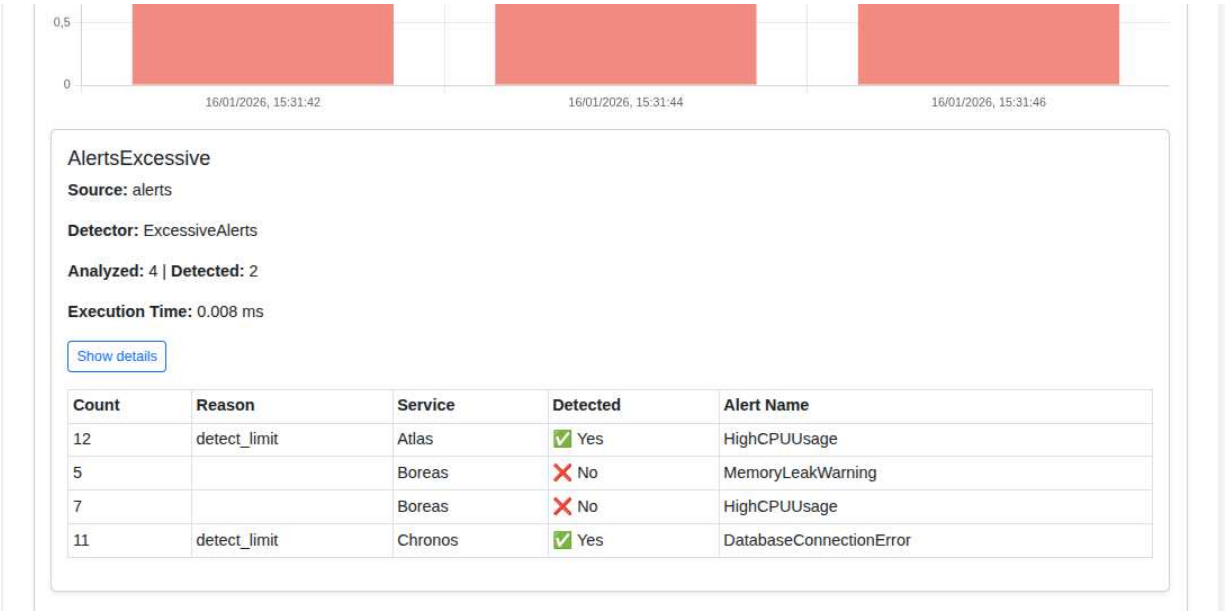
Além disso, é possível visualizar os detalhes de uma execução específica selecionada (ver Figura 41). Essas funcionalidades viabilizam a análise longitudinal do comportamento da observabilidade, permitindo a identificação de tendências, recorrências e padrões que não seriam perceptíveis em execuções pontuais.

O acesso ao histórico das detecções é, particularmente, relevante para auditorias técnicas, estudos evolutivos e comparações entre diferentes períodos de observação, como janelas de instabilidade, picos de carga ou alterações na instrumentação dos serviços. Ao preservar os resultados e as evidências associadas a cada execução, o Observa contribui para a rastreabilidade das análises e para a avaliação contínua da qualidade da observabilidade em sistemas distribuídos.

Figura 40 – Tela para visualizar todo o histórico das execuções.

Fonte: Autoria própria.

Figura 41 – Tela para visualizar os detalhes de uma execução específica



Fonte: Autoria própria.

6.6 Considerações Finais

Este capítulo apresentou o Observa, um *framework* projetado para operacionalizar a detecção automática de APs de observabilidade em aplicações baseadas em microsserviços. A partir de uma arquitetura modular, extensível e tecnologicamente neutra, o Observa materializa os conceitos teóricos discutidos nos capítulos anteriores, permitindo que APs de observabilidade sejam identificados de forma sistemática, reprodutível e apoiada por evidências.

A descrição detalhada da arquitetura, das abstrações e dos fluxos operacionais evidenciou como o *framework* integra fontes heterogêneas de dados e executa detectores independentes de maneira transparente. Além disso, os aspectos de implementação e implantação demonstram que o Observa pode ser executado em ambientes controlados com baixo esforço de configuração, favorecendo sua adoção tanto em contextos acadêmicos quanto experimentais.

Em síntese, o Observa estabelece uma ponte entre o conhecimento conceitual sobre APs de observabilidade e sua aplicação prática, constituindo uma base sólida para avaliações empíricas mais amplas, para a incorporação de novos APs e para a evolução de abordagens automatizadas voltadas à melhoria contínua da observabilidade em ambientes de microsserviços.

O capítulo seguinte apresenta uma avaliação empírica do *framework*, conduzida por meio de experimentos e por uma análise de usabilidade baseada em *survey*, com o objetivo de fornecer evidências adicionais sobre a aplicabilidade, robustez e utilidade prática do Observa, e examinando seu potencial de adoção em ambientes de observabilidade.

7 AVALIAÇÕES E RESULTADOS COM O OBSERVA

Este capítulo apresenta as avaliações realizadas com o Observa, com o objetivo de analisar seu comportamento, desempenho e aceitação em diferentes cenários de uso. Para alcançar esses objetivos, a avaliação do Observa foi conduzida por meio de quatro abordagens complementares: (i) uma prova de conceito que demonstra sua viabilidade técnica e aplicabilidade prática; (ii) uma avaliação experimental com *datasets* reais, (iii) uma avaliação experimental controlada com dados sintéticos; e (iv) uma avaliação empírica de usabilidade baseada em *survey*. Em conjunto, essas abordagens permitem analisar o Observa sob perspectivas quantitativas e qualitativas, fornecendo uma visão abrangente de suas capacidades e limitações.

7.1 Objetivos e Questões de Avaliação

As avaliações conduzidas neste capítulo foram orientadas por objetivos específicos, derivados das contribuições propostas nesta tese. De forma geral, buscou-se responder se o Observa, enquanto artefato científico, é capaz de apoiar de maneira efetiva a detecção automática de Anti-Padrões (APs) de observabilidade em sistemas baseados em microsserviços. A partir desse objetivo geral, as avaliações foram organizadas para responder às seguintes questões de avaliação (QA):

- **QA1:** O Observa é capaz de detectar automaticamente APs de observabilidade a partir de dados de telemetria?
- **QA2:** O Observa é capaz de processar e analisar dados reais de observabilidade, produzindo indicações estruturadas da existência de APs quando estes estão presentes?
- **QA3:** O Observa é capaz de lidar com a evolução temporal dos dados de observabilidade em cenários controlados, permitindo a detecção progressiva de APs ao longo de ciclos de coleta?
- **QA4:** O Observa é percebido como uma solução usável e útil por usuários com experiência em observabilidade, considerando tarefas típicas de configuração, execução e análise de resultados?

Essas questões não têm como objetivo avaliar a validação semântica exaustiva dos detectores, mas sim analisar o comportamento do *framework* enquanto infraestrutura de apoio à detecção de APs.

7.2 Configuração Experimental

Esta seção descreve a configuração experimental adotada nas avaliações, com o objetivo de garantir clareza, reprodutibilidade e transparência metodológica. Todas as avaliações experimentais foram conduzidas em um ambiente controlado, utilizando a versão empacotada do Observa por meio de contêineres Docker (ver Apêndice B). Essa abordagem assegura consistência no ambiente de execução e reduz variações decorrentes de configurações específicas do sistema hospedeiro.

Os experimentos foram executados em um notebook equipado com processador Intel(R) Core(TM) i5-9300H, com 8 núcleos de processamento e frequência de 2.40GHz, com 16GB de memória RAM, e executando o sistema operacional Linux Mint 21.3. Em todas as avaliações, o Observa foi configurado com fontes de dados e detectores registrados por meio da interface gráfica. Além disso, a persistência dos resultados e do histórico de execuções foi mantida ativa, assim como a coleta da métrica de tempo de execução das detecções. Essa configuração foi mantida constante ao longo de todos os experimentos, garantindo que as diferenças observadas nos resultados estivessem relacionadas exclusivamente aos dados analisados e aos modos de execução adotados, e não a variações na infraestrutura do *framework*.

As avaliações experimentais consideraram, principalmente, métricas relacionadas ao comportamento do Observa enquanto *framework*, incluindo o tempo de execução das detecções, o volume de dados analisados e a quantidade de ocorrências classificadas como AP. Na avaliação por *survey*, foram analisadas métricas perceptivas, relacionadas à usabilidade e utilidade do Observa, conforme descrito na Seção 7.6.

7.3 Prova de Conceito

Esta seção apresenta a *Proof of Concept* (PoC) do *framework* Observa, com o objetivo de demonstrar sua viabilidade técnica, extensibilidade e aplicabilidade prática no contexto da detecção automática de APs de observabilidade. Diferentemente das seções anteriores, que detalharam a arquitetura, as abstrações e os mecanismos internos do *framework*, a PoC concentra-se em sua execução em um cenário controlado, evidenciando seu funcionamento ponta a ponta.

7.3.1 Cenário Motivador

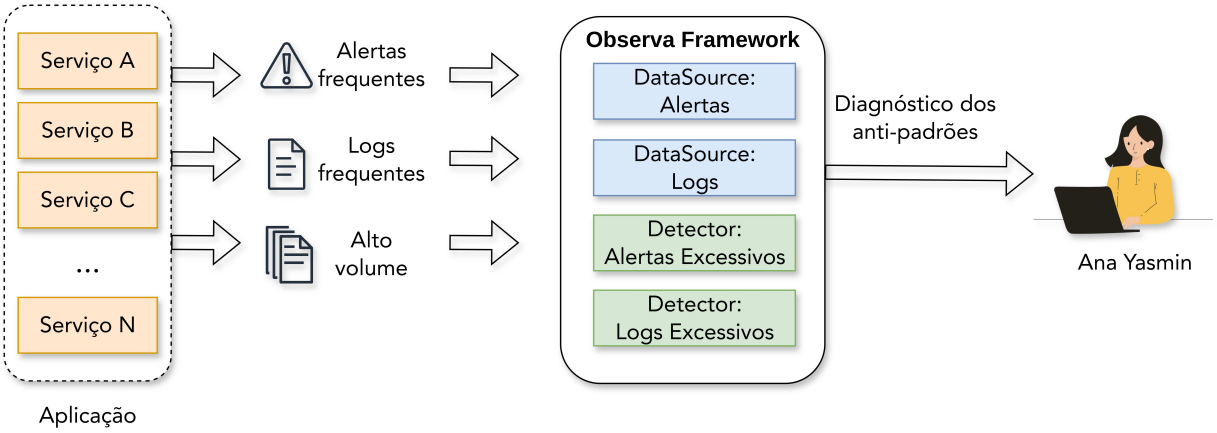
A equipe de SRE, da qual Ana Yasmin faz parte, é responsável pelo monitoramento e operação de uma aplicação distribuída composta por dezenas de microsserviços. Para garantir a confiabilidade e a disponibilidade do sistema em produção, a equipe de SRE utiliza ferramentas consolidadas de observabilidade para coleta de métricas, geração de alertas e análise de *logs*.

Com a evolução contínua da aplicação e a instrumentação progressiva dos serviços, o volume de dados de observabilidade cresce de forma significativa. Nesse contexto, Ana Yasmin passa a lidar com um número elevado de alertas redundantes ou pouco informativos, disparados em curtos intervalos de tempo, o que dificulta a identificação rápida de eventos realmente críticos. Esse comportamento caracteriza o AP de *alertas excessivos* e compromete a eficiência da resposta operacional.

De forma semelhante, a análise dos *logs* gerados pelos microsserviços torna-se progressivamente mais complexa. Mensagens repetitivas, volumes elevados de registros e níveis de severidade inadequados dificultam a investigação de falhas e reduzem o valor diagnóstico das informações coletadas, evidenciando a ocorrência do AP de *logs excessivos*.

Embora a equipe disponha de ferramentas maduras para coleta e visualização de dados de observabilidade, a identificação sistemática desses APs depende majoritariamente da experiência individual dos engenheiros. A ausência de mecanismos automatizados para avaliar a qualidade da observabilidade torna esse processo subjetivo, pouco reprodutível e difícil de escalar.

Figura 42 – Cenário motivador.



Fonte: Autoria própria.

Diante desse contexto e a partir do cenário motivador, a PoC apresentada neste

capítulo concentra-se em dois cenários de validação do Observa. O primeiro é voltado à detecção automática do AP de alertas excessivos, enquanto o segundo é direcionado à detecção do AP de *Logs* Excessivos. Esses APs foram selecionados por sua relevância prática e por poderem ser caracterizados a partir de critérios objetivos, tais como frequência, volume e repetição de eventos. Para fins de controle experimental e garantia de reprodutibilidade, os dados utilizados foram fornecidos por fontes estáticas previamente cadastradas no Observa, as quais simulam dados de alertas e *logs* coletados de uma aplicação baseada em microsserviços instrumentada com mecanismos de observabilidade.

7.3.2 Fontes de Dados Utilizadas

Conforme apresentado anteriormente, foram configuradas duas fontes de dados principais: (i) uma fonte de alertas, contendo registros agregados de alertas disparados por diferentes serviços ao longo de um período de 24 horas de coleta, totalizando 1000 alertas, que é uma quantidade que 63% das organizações enfrentam diariamente[1]; e (ii) uma fonte de *logs*, composta por mensagens geradas por múltiplos serviços, com diferentes níveis de severidade, em que cada registro apresenta um incremento temporal de um segundo em relação ao anterior, totalizando um *dataset* com 1000 *logs*. Para a geração desses dados, foi utilizado um gerador alertas e *logs*, que podem ser visualizados nos seguintes links: <https://github.com/andersonalmada/detect-antipatterns/blob/master/alerts/fake.py> e <https://github.com/andersonalmada/detect-antipatterns/blob/master/logs/fake.py>.

Essas fontes foram integradas ao *framework* por meio da classe `JSONDataSource` (ver Código-fonte 6), utilizando o *upload* dos arquivos *JSON* disponibilizados nos seguintes links: (i) para os alertas – <https://github.com/andersonalmada/detect-antipatterns/blob/master/alerts/alerts-1000-original.json>; e (ii) para os *logs* – <https://github.com/andersonalmada/detect-antipatterns/blob/master/logs/logs.json>.

Para fins de exemplificação, apresenta-se a seguir uma amostra da fonte de dados de alertas, a fim de evidenciar a estrutura dos dados empregados no processo de detecção.

```
[
  {
    "host": "app-server-01",
    "name": "TestAlert1",
    "service": "auth-service",
    "severity": "warning",
```

[1] <https://www.netdata.cloud/resources/research/monitor-everything-anti-pattern/>

```
    "timestamp": "2025-12-16T23:02:41Z",
    "value": 51
  },
  ...
]
```

Código-fonte 1 – Amostra da fonte de dados de alertas.

De forma complementar, apresenta-se a seguir uma amostra da fonte de dados de *logs*, ilustrando a estrutura dos dados empregados no processo de detecção.

```
[
  {
    "level": "DEBUG",
    "message": "Order created successfully",
    "service": "payment-service",
    "timestamp": "2025-12-17T10:49:19.721460Z",
    "trace_id": "aaa111"
  },
  ...
]
```

Código-fonte 2 – Amostra da fonte de dados de *logs*

7.3.3 Detectores Utilizados

Para a execução da PoC, foram empregados dois APs previamente implementados e cadastrados no Observa: (i) um detector de *alertas excessivos* (*alertsdetector*) e (ii) um detector de *logs excessivos* (*logsdetector*).

7.3.3.1 Alertas Excessivos

O detector de alertas excessivos foi implementado como um serviço remoto utilizando a tecnologia Flask em Python. O código-fonte está disponível em: <https://github.com/andersonalmada/detect-antipatterns/blob/master/alerts/alert_detector.py>. Esse detector realiza a análise de fluxos de alertas com o objetivo de identificar padrões de excesso e redundância temporal, comumente associados a cenários de excessos de alertas.

A lógica de análise é estruturada em três etapas principais: (i) agrupamento temporal,

(ii) detecção de excesso de volume e (iii) detecção de redundância temporal. Inicialmente, os alertas são organizados em grupos horários com base em seus *timestamps*, formando janelas temporais fixas de uma hora.

Na etapa seguinte, o detector avalia o volume total de alertas em cada grupo horário. Caso esse volume ultrapasse um limiar configurável, ajustado de acordo com o tamanho da equipe responsável, o grupo é classificado como excessivo. Nesse exemplo, foi utilizado um limiar de dez alertas por membro por hora. Nessa situação, todos os alertas pertencentes ao grupo são marcados como tal, caracterizando cenários em que o elevado volume de notificações pode comprometer a capacidade de resposta operacional.

Além da análise de volume, o detector realiza a identificação de redundância temporal dentro de cada grupo horário. Para isso, os alertas são agrupados com base em atributos semânticos, como `host`, `service` e `name`, e ordenados cronologicamente. Alertas consecutivos que ocorrem dentro de uma janela de tempo predefinida são considerados redundantes, caracterizando padrões de repetição excessiva em curtos intervalos. Nesse exemplo, foi utilizado uma janela de 60 segundos. Esse tipo de análise permite identificar problemas que não seriam capturados apenas por métricas agregadas de volume, como a reemissão frequente de alertas idênticos causada por falhas persistentes ou configurações inadequadas de monitoramento.

O serviço expõe um *endpoint* que recebe os alertas em formato JSON, aplica as etapas de análise descritas e retorna uma resposta estruturada contendo: (i) o número total de alertas analisados; (ii) o número total de alertas classificados como problemáticos; e (iii) os grupos detalhados, incluindo os alertas marcados e o motivo da classificação.

7.3.3.2 Logs Excessivos

O detector de *Logs* Excessivos também foi implementado como um serviço remoto, utilizando a tecnologia Spring Boot em Java. O seu código-fonte está disponível em: <https://github.com/andersonalmada/detect-antipatterns/tree/master/logs/logsdetector>. O objetivo desse detector é identificar padrões de geração anômala de *logs* em curtos intervalos de tempo, frequentemente associados a falhas recorrentes, laços de erro, níveis de *log* inadequados.

Diferentemente do detector de alertas, que opera em janelas horárias, o detector de *logs* adota uma granularidade temporal mais fina, agrupando as mensagens em janelas de um segundo, permitindo a identificação de picos súbitos de geração de *logs*.

Após o agrupamento temporal, o serviço avalia o volume de *logs* gerados. Caso o

número de mensagens ultrapasse um limiar previamente definido, o intervalo é classificado como excessivo, e os *logs* correspondentes são explicitamente marcados. Nesse exemplo, foi utilizado um limitar de 5 *logs* por segundo. Além disso, esse detector avalia o tamanho dos *logs* baseado no volume. Nesse exemplo, foi utilizado um limiar de 1 KB por segundo[2]. E por fim, o detector avalia o uso inadequado de níveis de severidade em produção, que seria *DEBUG* ou *TRACE*, como explicado no catálogo.

O detector disponibiliza um *endpoint* que recebe uma lista de *logs* estruturados em formato JSON e retorna uma resposta consolidada contendo: (i) o total de *logs* analisados; (ii) o total de *logs* classificados como problemáticos; e (iii) os *logs* individualmente associados a esses problemas.

7.3.4 Execução da PoC

Inicialmente, foi selecionado a fonte de dados previamente cadastrada e o AP de observabilidade a ser analisado. Em seguida, foi escolhido o detector correspondente e configurado o modo de execução, optando-se pela execução por demanda da coleta e análise.

Após o início da execução, o Observa realizou automaticamente a coleta dos dados, a partir da fonte configurada e encaminhou os dados ao detector selecionado. O detector aplicou suas regras de análise específicas e retornou os resultados ao *framework*, que consolidou as informações, registrou os metadados da execução e persistiu os resultados no banco de dados interno. Esse processo foi realizado tanto para alertas quanto para os *logs*.

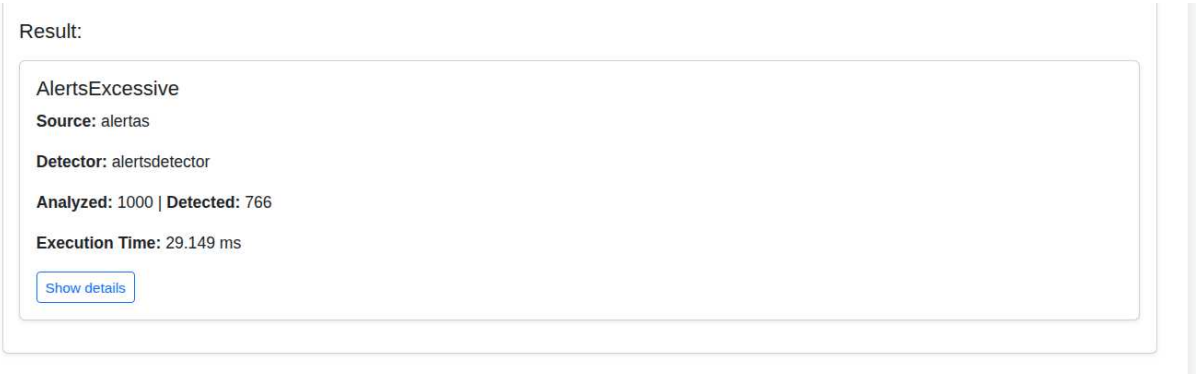
Todo o processo de execução, incluindo a seleção dos parâmetros, a comunicação com detectores remotos e a apresentação dos resultados, foi registrado em vídeo, disponibilizado no seguinte endereço: <https://youtu.be/C42I-gwNA50>, a fim de evidenciar o funcionamento ponta a ponta do Observa em um cenário controlado de uso.

7.3.5 Resultados Obtidos

A Figura 43 apresenta os resultados obtidos no Observa para a fonte de dados de alertas, no contexto da detecção do AP de alertas excessivos. Conforme descrito anteriormente, o *dataset* de alertas é composto por 1000 registros, conforme visualizado na figura. Desse total, após a aplicação dos algoritmos de detecção descritos na Subseção 7.3.3.1, foram identificados 766 alertas classificados como ocorrência do AP.

[2] <https://docs.cloudfoundry.org/loggregator/app-log-rate-limits.html>

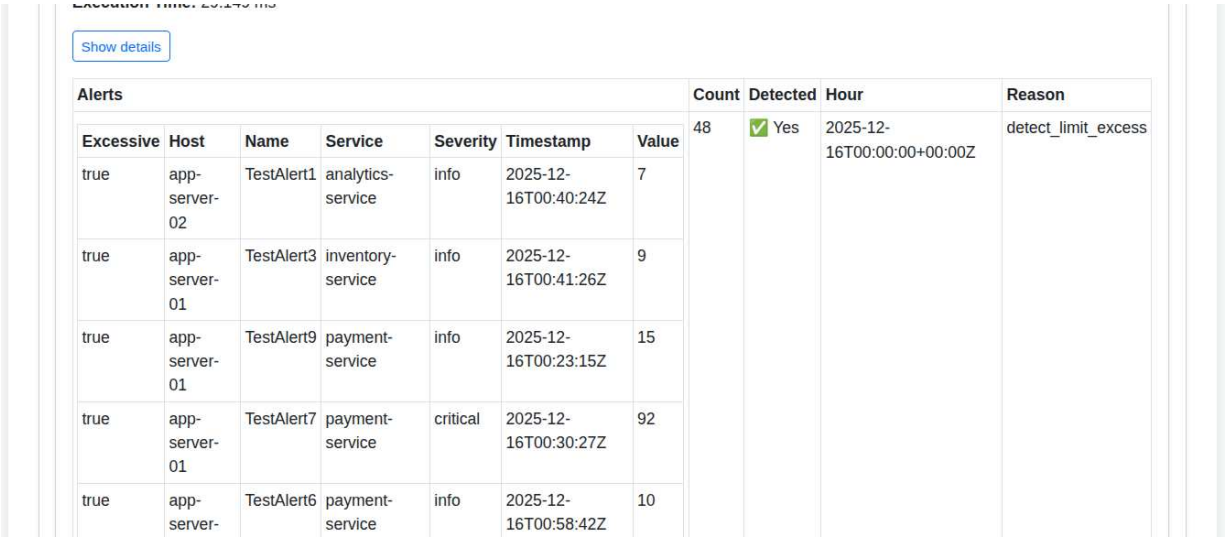
Figura 43 – Visão geral dos resultados da detecção de alertas excessivos.



Fonte: Autoria própria.

A partir desses resultados, foram analisados os detalhes da detecção, observando-se a identificação de janelas temporais em que o volume total de alertas ultrapassou os limiares configurados no detector, como dito anteriormente, definidos como dez alertas por membro da equipe de observabilidade. Nesses intervalos, os alertas associados foram classificados como excessivos, caracterizando situações com potencial de fadiga de alertas, com a quantidade de alertas (`count`) igual ou superior a quarenta, correspondente a dez alertas para cada um dos quatro membros da equipe, como pode ser visto na Figura 44. Além da análise da quantidade, o detector também identificou padrões de redundância temporal, nos quais alertas semanticamente equivalentes, isto é, provenientes do mesmo `host`, `service` e `name`, foram emitidos de forma repetitiva em curtos intervalos de tempo, especificamente em cinco ocorrências dentro de um período de um minuto.

Figura 44 – Detalhes de alertas detectados como excessivos.



Fonte: Autoria própria.

A Figura 45 mostra um cenário no qual o número total de alertas foi inferior a

quarenta, os alertas não foram classificados como excessivos e, consequentemente, não houve a detecção do AP.

Figura 45 – Detalhes de alertas não detectados como excessivos.

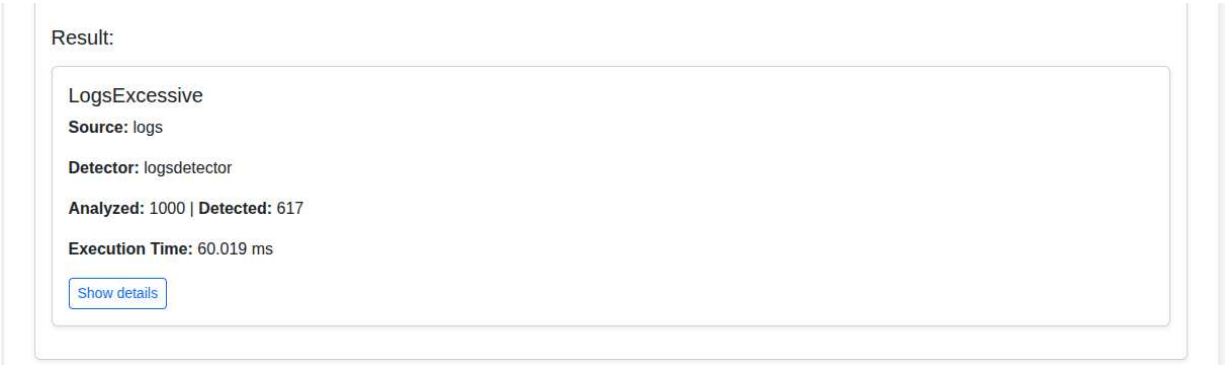
Excessive	Host	Name	Service	Severity	Timestamp	Value
false	app-server-02	TestAlert9	analytics-service	warning	2025-12-16T15:08:34Z	45
false	app-server-02	TestAlert4	notification-service	info	2025-12-16T15:57:04Z	2
false	app-server-02	TestAlert8	payment-service	info	2025-12-16T15:18:32Z	2
false	app-server-	TestAlert0	notification-service	warning	2025-12-16T15:07:48Z	70

34 ❌ No 2025-12-16T15:00:00+00:00Z

Fonte: Autoria própria.

A Figura 46 apresenta os resultados obtidos no Observa para a fonte de dados de *logs*, no contexto da detecção do AP de *Logs* Excessivos. Conforme descrito anteriormente, o *dataset* de *logs* é composto por 1000 registros, conforme visualizado na figura. Desse total, após a aplicação dos algoritmos de detecção descritos na Seção 7.3.3.2, foram identificados 617 *logs* classificados como ocorrência do AP.

Figura 46 – Visão geral dos resultados da detecção de *logs* excessivos.

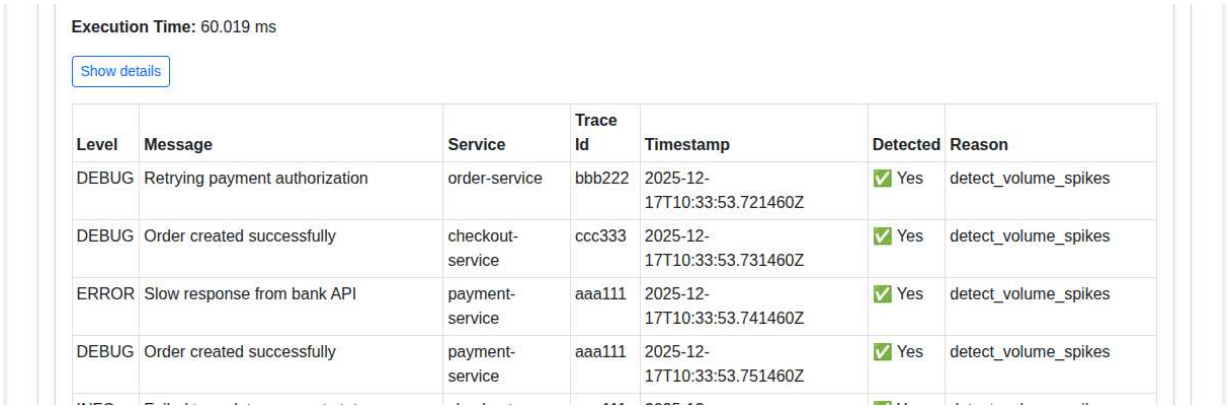


Fonte: Autoria própria.

A partir desses resultados, foram analisados os detalhes da detecção. Alguns apresentaram picos súbitos de volume (ver Figura 47), outros repetição recorrente de mensagens (ver Figura 48), isto é, cinco mensagens repetidas com mesmo `service`, `trace id`, e `severity`, e uso inadequado de níveis de severidade (ver Figura 49), características associadas a cenários de excessos de *logs*.

A análise dos resultados obtidos indica que o Observa foi capaz de integrar fontes heterogêneas de dados de observabilidade, mesmo quando essas fontes apresentam diferenças

Figura 47 – Detalhes de *logs* detectados como excessivos - *volume spikes*.



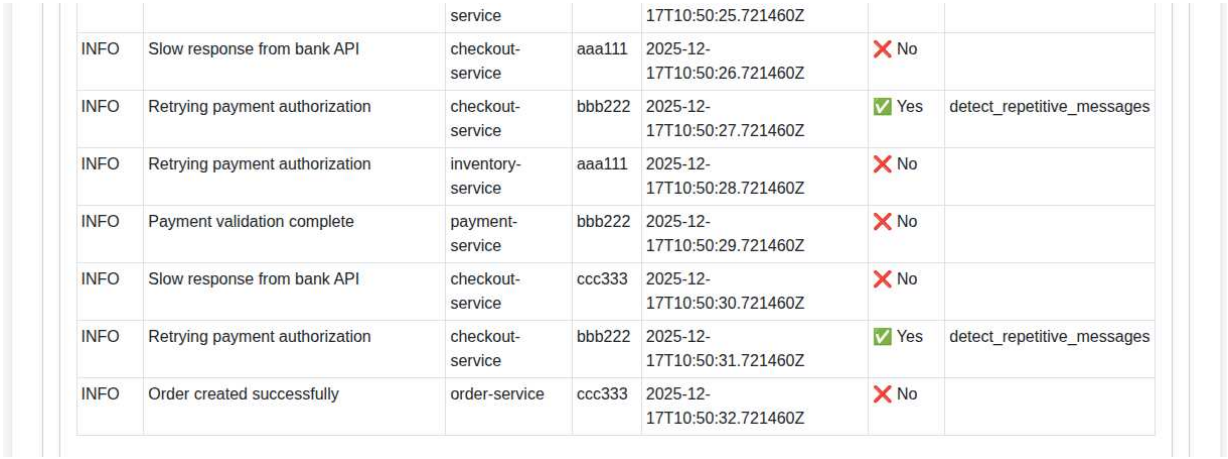
Execution Time: 60.019 ms

Show details

Level	Message	Service	Trace Id	Timestamp	Detected	Reason
DEBUG	Retrying payment authorization	order-service	bbb222	2025-12-17T10:33:53.721460Z	✅ Yes	detect_volume_spikes
DEBUG	Order created successfully	checkout-service	ccc333	2025-12-17T10:33:53.731460Z	✅ Yes	detect_volume_spikes
ERROR	Slow response from bank API	payment-service	aaa111	2025-12-17T10:33:53.741460Z	✅ Yes	detect_volume_spikes
DEBUG	Order created successfully	payment-service	aaa111	2025-12-17T10:33:53.751460Z	✅ Yes	detect_volume_spikes

Fonte: Autoria própria.

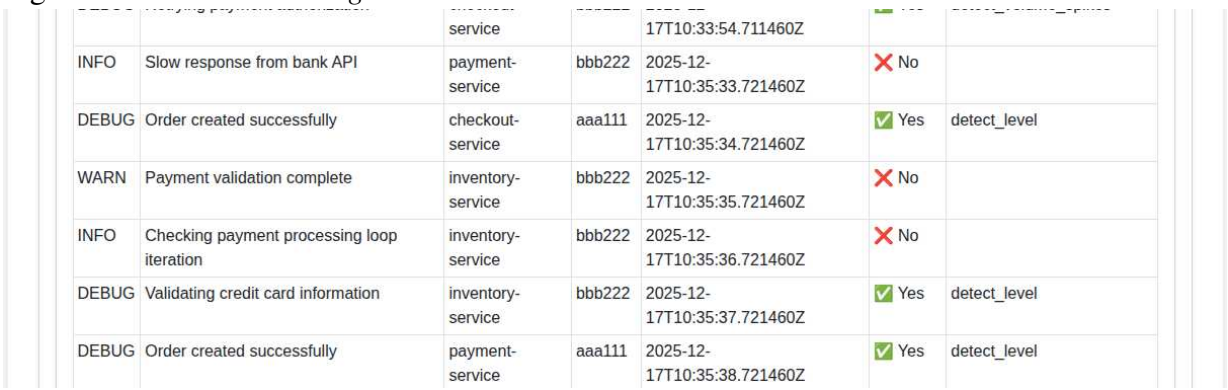
Figura 48 – Detalhes de *logs* detectados como excessivos - *repetitive messages*.



		service		17T10:50:25.721460Z		
INFO	Slow response from bank API	checkout-service	aaa111	2025-12-17T10:50:26.721460Z	❌ No	
INFO	Retrying payment authorization	checkout-service	bbb222	2025-12-17T10:50:27.721460Z	✅ Yes	detect_repetitive_messages
INFO	Retrying payment authorization	inventory-service	aaa111	2025-12-17T10:50:28.721460Z	❌ No	
INFO	Payment validation complete	payment-service	bbb222	2025-12-17T10:50:29.721460Z	❌ No	
INFO	Slow response from bank API	checkout-service	ccc333	2025-12-17T10:50:30.721460Z	❌ No	
INFO	Retrying payment authorization	checkout-service	bbb222	2025-12-17T10:50:31.721460Z	✅ Yes	detect_repetitive_messages
INFO	Order created successfully	order-service	ccc333	2025-12-17T10:50:32.721460Z	❌ No	

Fonte: Autoria própria.

Figura 49 – Detalhes de *logs* detectados como excessivos - *level*.



		service		17T10:33:54.711460Z		
INFO	Slow response from bank API	payment-service	bbb222	2025-12-17T10:35:33.721460Z	❌ No	
DEBUG	Order created successfully	checkout-service	aaa111	2025-12-17T10:35:34.721460Z	✅ Yes	detect_level
WARN	Payment validation complete	inventory-service	bbb222	2025-12-17T10:35:35.721460Z	❌ No	
INFO	Checking payment processing loop iteration	inventory-service	bbb222	2025-12-17T10:35:36.721460Z	❌ No	
DEBUG	Validating credit card information	inventory-service	bbb222	2025-12-17T10:35:37.721460Z	✅ Yes	detect_level
DEBUG	Order created successfully	payment-service	aaa111	2025-12-17T10:35:38.721460Z	✅ Yes	detect_level

Fonte: Autoria própria.

em termos de estrutura e semântica. Essa integração ocorreu sem a necessidade de adaptações no *framework*.

Adicionalmente, os experimentos demonstraram que detectores implementados como serviços independentes, desenvolvidos em tecnologias distintas e executados remotamente, puderam ser acionados de forma transparente a partir de um mesmo fluxo de análise. Do ponto de vista do usuário, o processo de execução manteve-se uniforme, independentemente da natureza do detector, o que reforça a flexibilidade e a extensibilidade da arquitetura desenvolvida.

Por fim, os resultados produzidos pelo Observa foram organizados de maneira estruturada e acompanhados de metadados que permitem rastrear a origem dos dados, o detector responsável, os critérios de classificação aplicados e o instante da análise. Essa característica possibilita não apenas a identificação automática de APs de observabilidade, mas também a auditoria das detecções e sua reutilização em análises posteriores.

7.3.6 Discussão

Os resultados obtidos na Prova de Conceito fornecem evidências de que o *framework* Observa é capaz de operar de forma consistente em um cenário controlado de observabilidade em microsserviços. A integração de diferentes fontes de dados, a execução de detectores independentes e a geração de resultados estruturados indicam que as abstrações arquiteturais propostas são adequadas para o objetivo de identificar automaticamente APs de observabilidade, respondendo a **QA1**.

Cabe destacar que a PoC foi conduzida em um ambiente controlado, com fontes de dados estáticas e detectores baseados em limiares e heurísticas previamente definidos. Embora essa abordagem seja adequada para validar a viabilidade técnica e a arquitetura do *framework*, ela impõe limitações quanto à generalização dos resultados. Em cenários mais complexos e dinâmicos, ajustes manuais de parâmetros podem não ser suficientes, o que aponta para a necessidade de mecanismos adaptativos ou técnicas mais avançadas de análise como trabalhos futuros.

Adicionalmente, o Observa adota uma separação explícita entre a detecção de APs e a aplicação de ações corretivas. Quando um pico de volume de *logs* é identificado, por exemplo, o *framework* registra e caracteriza a ocorrência, preservando evidências e metadados para análise posterior, sem executar intervenções automáticas. Essa decisão de projeto visa reduzir riscos operacionais e reforça o foco do trabalho na avaliação da qualidade da observabilidade, e não no

controle direto do comportamento dos sistemas monitorados.

7.4 Avaliação Experimental com *Datasets* Reais

A primeira avaliação experimental foi conduzida utilizando dois *datasets* reais de observabilidade. O objetivo dessa avaliação foi analisar o comportamento do Observa frente a dados reais de sistemas, caracterizados por variabilidade, ruído e heterogeneidade típicos de cenários de produção. Ambos os *datasets* são compostos por dados de observabilidade com foco em alertas.

O primeiro *dataset* (Empresa 1) foi coletado, em um período de cinco dias, de uma empresa de desenvolvimento de software sediada nos Estados Unidos, com porte estimado entre 51 e 200 funcionários, conforme o LinkedIn. A aplicação analisada é voltada ao modelo *Business-to-Business* (B2B), com foco no setor da construção civil, oferecendo suporte principalmente às atividades financeiras associadas à gestão de obras. Os alertas presentes nesse conjunto de dados refletem eventos operacionais e comportamentos anômalos observados durante a execução dos serviços.

O segundo *dataset* (Empresa 2) foi obtido, em um período também de cinco dias, a partir de uma empresa do setor de telecomunicações no Brasil, cujo porte organizacional é significativamente maior, com aproximadamente 5000 a 10000 funcionários, conforme o LinkedIn. Os alertas gerados são oriundos das infraestruturas que executam as aplicações acessadas pelos clientes da empresa. Os *datasets* estão disponíveis em <https://github.com/andersonalmada/detect-antipatterns/blob/master/alerts/alerts-company-1.json> e <https://github.com/andersonalmada/detect-antipatterns/blob/master/alerts/alerts-company-2.json>.

Não foi imposta uma estrutura pré-definida para a representação dos dados de alertas fornecidos pelas empresas participantes, permitindo que cada uma disponibilizasse seus dados conforme os formatos utilizados em suas respectivas aplicações de monitoramento. Como consequência, os *datasets* apresentam estruturas distintas, o que exigiu adaptações nos detectores previamente apresentados para acomodar essas variações.

Para a fonte de dados da Empresa 1, foi realizada uma modificação no algoritmo descrito na Seção 7.3.3.1, com o objetivo de normalizar o campo que representa o instante de geração do alerta. O formato original da data, expresso como 11/30/2025 4:58 AM, foi convertido para o padrão ISO 8601, resultando em 2025-11-30T04:58:00. De forma semelhante, para a fonte de dados da Empresa 2, foi aplicada uma normalização no campo de data, convertendo,

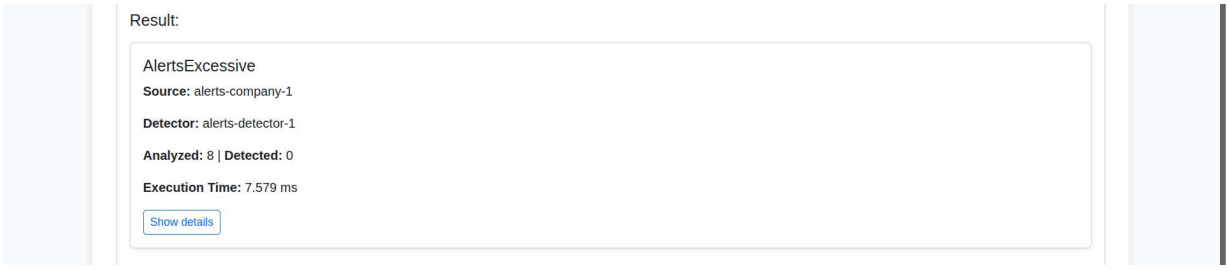
por exemplo, o valor Quarta, 17 de dezembro de 2025 15:31 para 2025-12-17T15:31:00.

Adicionalmente, o *dataset* da Empresa 2 disponibiliza um campo que indica a duração do alerta, o que possibilitou a implementação de um novo algoritmo para a detecção de problemas baseados em características temporais. Esse algoritmo analisa a duração de lembretes de alerta a fim de identificar alertas excessivos e classificá-los como instáveis, persistentes ou crônicos. Alertas de longa duração indicam, em geral, condições não acionáveis ou cenários de degradação contínua do sistema. Tais padrões contribuem para o aumento do ruído operacional e favorecem o fenômeno de fadiga de alertas. Dessa forma, o algoritmo evidencia configurações inadequadas de monitoramento e potenciais falhas operacionais em sistemas distribuídos.

Os detectores modificados e implementados para a identificação do AP de alertas excessivos estão disponíveis em: <https://github.com/andersonalmada/detect-antipatterns/blob/master/alerts/alerts_detector_company_1.py> e <https://github.com/andersonalmada/detect-antipatterns/blob/master/alerts/alerts_detector_company_2.py>. A Figura 50 apresenta o resultado da detecção para o *dataset* da Empresa 1. Verifica-se que não foi identificada nenhuma detecção de AP nesse *dataset*. Um aspecto relevante é que apenas oito alertas foram gerados ao longo de um período de cinco dias, o que inviabiliza o acionamento do algoritmo baseado na quantidade de alertas. Adicionalmente, o algoritmo de detecção de redundância também não foi ativado, uma vez que não há, pelo menos, dois alertas gerados dentro de uma janela temporal de sessenta segundos. Dessa forma, o resultado da detecção mostra-se coerente com as características do *dataset* analisado.

Questionada sobre o baixo volume de alertas, a empresa afirmou que “caso houvesse uma quantidade elevada de alertas, isso sim seria preocupante, pois indicaria configurações inadequadas e a necessidade de ajuste dos limiares das métricas monitoradas”.

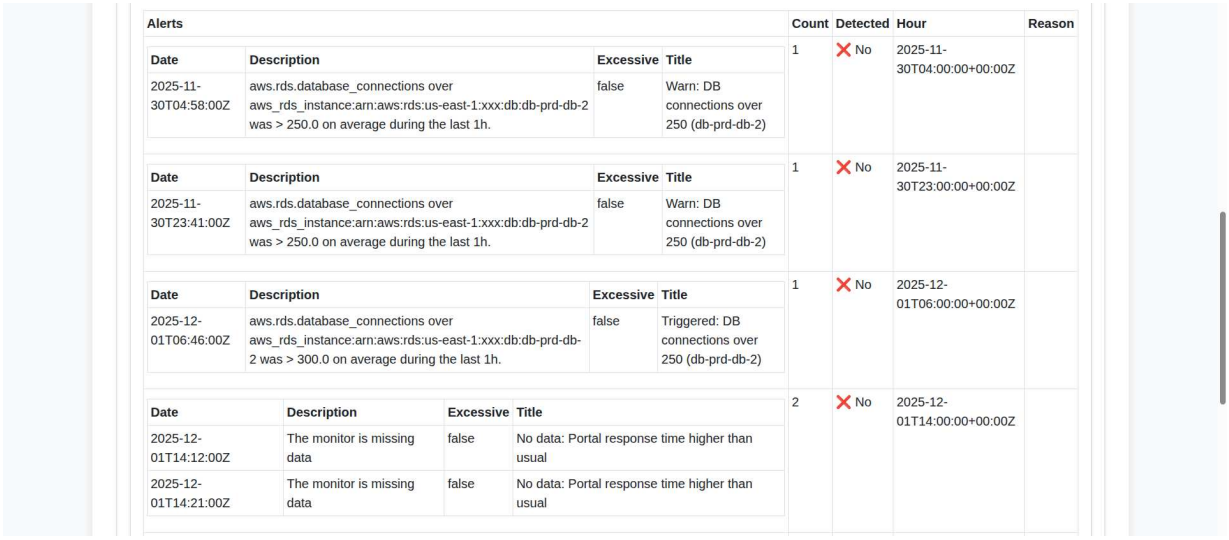
Figura 50 – Resultado da detecção para fonte da Empresa 1.



Fonte: Autoria própria.

Com finalidade ilustrativa, a Figura 51 apresenta os detalhes da detecção, demonstrando que não foi identificado nenhum caso de alertas excessivos.

Figura 51 – Detalhes dos resultados da detecção para fonte da Empresa 1.

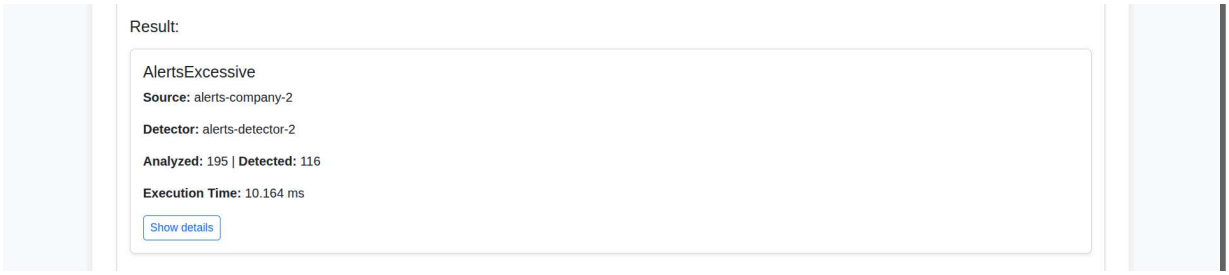


Alerts				Count	Detected	Hour	Reason
Date	Description	Excessive	Title	1	❌ No	2025-11-30T04:00:00+00:00Z	
2025-11-30T04:58:00Z	aws.rds.database_connections over aws_rds_instance:arn:aws:rds:us-east-1:xxx:db:db-prd-db-2 was > 250.0 on average during the last 1h.	false	Warn: DB connections over 250 (db-prd-db-2)				
Date	Description	Excessive	Title	1	❌ No	2025-11-30T23:00:00+00:00Z	
2025-11-30T23:41:00Z	aws.rds.database_connections over aws_rds_instance:arn:aws:rds:us-east-1:xxx:db:db-prd-db-2 was > 250.0 on average during the last 1h.	false	Warn: DB connections over 250 (db-prd-db-2)				
Date	Description	Excessive	Title	1	❌ No	2025-12-01T06:00:00+00:00Z	
2025-12-01T06:46:00Z	aws.rds.database_connections over aws_rds_instance:arn:aws:rds:us-east-1:xxx:db:db-prd-db-2 was > 300.0 on average during the last 1h.	false	Triggered: DB connections over 250 (db-prd-db-2)				
Date	Description	Excessive	Title	2	❌ No	2025-12-01T14:00:00+00:00Z	
2025-12-01T14:12:00Z	The monitor is missing data	false	No data: Portal response time higher than usual				
2025-12-01T14:21:00Z	The monitor is missing data	false	No data: Portal response time higher than usual				

Fonte: Autoria própria.

A Figura 52 apresenta o resultado da detecção para a Empresa 2. Verifica-se que foram identificados APs nesse *dataset*. No total, foram gerados 195 alertas, de um período de cinco dias, no algoritmo e detectado um total de 116 ocorrências de excessos.

Figura 52 – Resultado da detecção para fonte da Empresa 2.



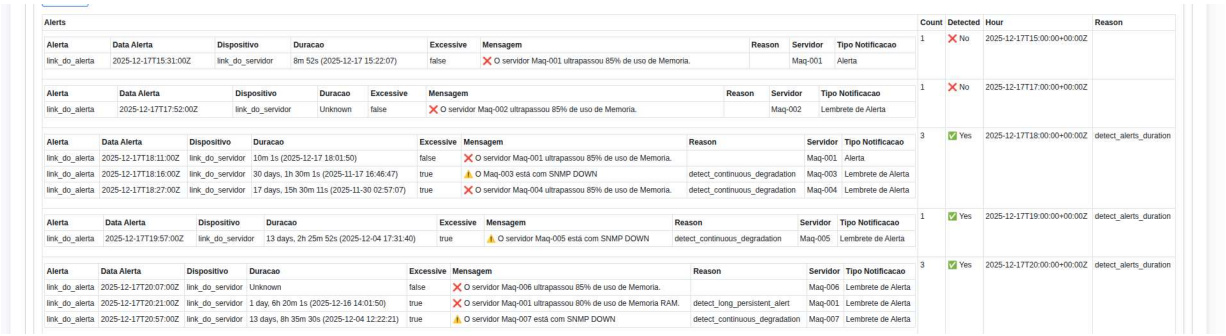
Fonte: Autoria própria.

A partir da visão detalhada dos resultados no Observa, foi possível analisar quais dados foram caracterizados como APs. Parte dos resultados obtidos pode ser observada na Figura 53. De acordo com a análise realizada, o algoritmo não identificou a ocorrência de alertas excessivos com base na quantidade por hora, sendo observado um máximo de quatro alertas por hora. Ademais, não foram detectados indícios de redundância nos alertas analisados.

Por outro lado, os APs efetivamente identificados estão relacionados à duração com que os alertas permanecem ativos no sistema de notificação da empresa, caracterizando comportamentos persistentes ao longo do tempo. Em alguns casos, foram observadas situações de degradação contínua, nas quais os alertas permaneceram ativos por períodos superiores a sete dias, o que evidencia a presença de problemas crônicos associados a configurações inadequadas. O cenário mais crítico refere-se a um problema identificado no servidor Maq-008, no qual o uso

de memória ultrapassou 85% por um período superior a 107 dias consecutivos, configurando um caso evidente de falha persistente e ausência de tratamento corretivo adequado.

Figura 53 – Detalhes dos resultados da detecção para fonte da Empresa 2.



Alerts

Alerta	Data Alerta	Dispositivo	Duracao	Excessive	Mensagem	Reason	Servidor	Tipo Notificacao
link_do_alerta	2025-12-17T15:31:00Z	link_do_servidor	8m 52s (2025-12-17 15:22:07)	false	❌ O servidor Maq-001 ultrapassou 85% de uso de Memoria.		Maq-001	Alerta

Alerta	Data Alerta	Dispositivo	Duracao	Excessive	Mensagem	Reason	Servidor	Tipo Notificacao
link_do_alerta	2025-12-17T17:52:00Z	link_do_servidor	Unknown	false	❌ O servidor Maq-002 ultrapassou 85% de uso de Memoria.		Maq-002	Lembrete de Alerta

Alerta	Data Alerta	Dispositivo	Duracao	Excessive	Mensagem	Reason	Servidor	Tipo Notificacao
link_do_alerta	2025-12-17T18:11:00Z	link_do_servidor	10m 1s (2025-12-17 18:01:50)	false	❌ O servidor Maq-001 ultrapassou 85% de uso de Memoria.		Maq-001	Alerta
link_do_alerta	2025-12-17T18:16:00Z	link_do_servidor	30 days, 1h 30m 1s (2025-11-17 16:46:47)	true	⚠️ O Maq-003 está com SNMP DOWN	detect_continuous_degradation	Maq-003	Lembrete de Alerta
link_do_alerta	2025-12-17T18:27:00Z	link_do_servidor	17 days, 15h 30m 11s (2025-11-30 02:57:07)	true	❌ O servidor Maq-004 ultrapassou 85% de uso de Memoria.	detect_continuous_degradation	Maq-004	Lembrete de Alerta

Alerta	Data Alerta	Dispositivo	Duracao	Excessive	Mensagem	Reason	Servidor	Tipo Notificacao
link_do_alerta	2025-12-17T19:57:00Z	link_do_servidor	13 days, 2h 25m 52s (2025-12-04 17:31:40)	true	⚠️ O servidor Maq-005 está com SNMP DOWN	detect_continuous_degradation	Maq-005	Lembrete de Alerta

Alerta	Data Alerta	Dispositivo	Duracao	Excessive	Mensagem	Reason	Servidor	Tipo Notificacao
link_do_alerta	2025-12-17T20:07:00Z	link_do_servidor	Unknown	false	❌ O servidor Maq-006 ultrapassou 85% de uso de Memoria.		Maq-006	Lembrete de Alerta
link_do_alerta	2025-12-17T20:21:00Z	link_do_servidor	1 day, 6h 20m 1s (2025-12-16 14:01:50)	true	❌ O servidor Maq-001 ultrapassou 80% de uso de Memoria RAM.	detect_long_persistent_alert	Maq-001	Lembrete de Alerta
link_do_alerta	2025-12-17T20:57:00Z	link_do_servidor	13 days, 8h 35m 30s (2025-12-04 12:22:21)	true	⚠️ O servidor Maq-007 está com SNMP DOWN	detect_continuous_degradation	Maq-007	Lembrete de Alerta

Count	Detected	Hour	Reason
1	❌ No	2025-12-17T15:00:00+00:00Z	
1	❌ No	2025-12-17T17:00:00+00:00Z	
3	✅ Yes	2025-12-17T18:00:00+00:00Z	detect_alerts_duration
1	✅ Yes	2025-12-17T19:00:00+00:00Z	detect_alerts_duration
3	✅ Yes	2025-12-17T20:00:00+00:00Z	detect_alerts_duration

Fonte: Autoria própria.

Os resultados obtidos a partir da avaliação experimental com *datasets* reais evidenciam a capacidade do Observa em lidar com dados provenientes de ambientes de produção distintos, caracterizados por diferentes volumes, estruturas e comportamentos operacionais, respondendo a **QA2**. A ausência de detecções no conjunto de dados da Empresa 1 mostrou-se consistente com o baixo volume de alertas observados, reforçando que o Observa não introduz falsos positivos em cenários estáveis. Por outro lado, a aplicação do método sobre o *dataset* da Empresa 2 demonstrou a efetividade do Observa na identificação de APs relevantes, especialmente aqueles associados à persistência prolongada de alertas, os quais são frequentemente negligenciados por abordagens baseadas apenas em volume ou redundância. Além disso, os resultados indicam que o Observa pode atuar como uma ferramenta de apoio à tomada de decisão operacional, auxiliando equipes na identificação de falhas persistentes.

7.5 Avaliação Experimental com Dados Sintéticos em Modo *Polling*

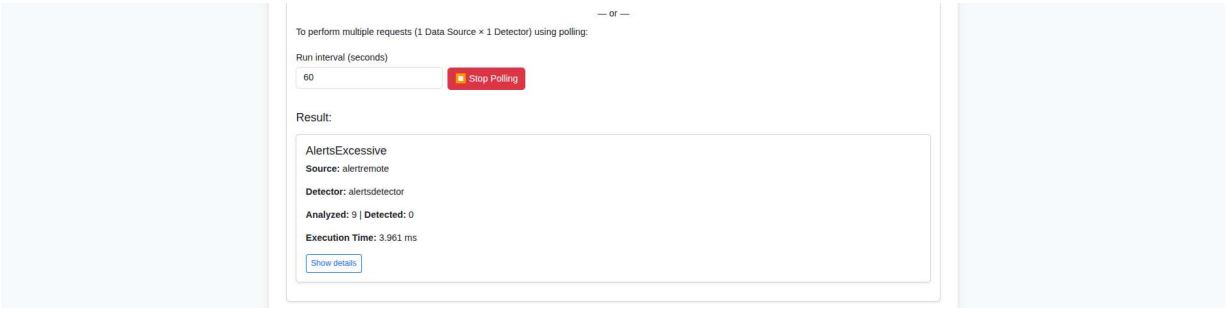
A segunda avaliação experimental teve como objetivo analisar o comportamento temporal do Observa, utilizando dados sintéticos gerados de forma controlada e coletados em modo de *polling*. Essa abordagem permitiu explorar cenários nos quais os APs se manifestam progressivamente ao longo do tempo.

Os dados sintéticos foram gerados, utilizando o gerador de alertas apresentado no capítulo anterior, com o propósito de simular padrões específicos de comportamento, tais como o crescimento gradual no volume de eventos e a recorrência periódica de registros de alerta. Assim sendo, foram conduzidos dois experimentos. No primeiro, um alerta foi gerado a cada minuto,

com o objetivo de avaliar a capacidade do Observa, por meio de seu detector, de identificar o AP de alertas excessivos com base no aumento progressivo do número de alertas ao longo do tempo. No segundo experimento, um alerta foi gerado a cada 0,01 segundos, visando verificar a capacidade do detector de identificar alertas redundantes.

Com relação ao primeiro experimento, a Figura 54 apresenta o resultado obtido pelo Observa, evidenciando um total de nove alertas capturados, o que corresponde a nove minutos de execução do teste. Nesse cenário, como esperado, ainda não há o AP de alertas excessivos, uma vez que o limite estabelecido para caracterização de fadiga de alertas, adotado neste trabalho, é de dez alertas em um intervalo de uma hora.

Figura 54 – Resultado da detecção de alertas excessivos para o experimento 1.



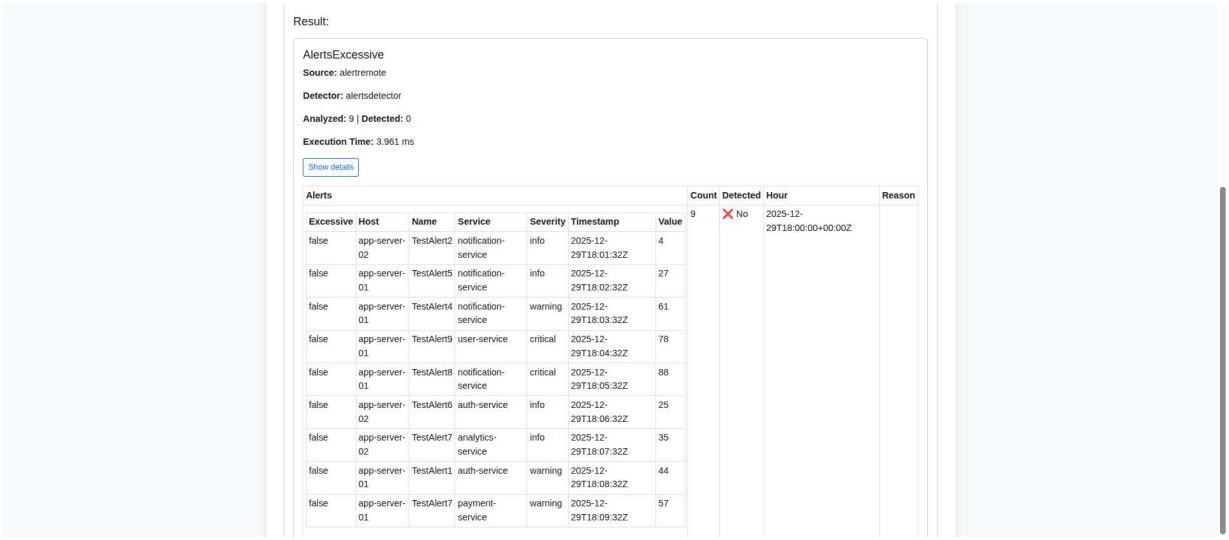
Fonte: Autoria própria.

A Figura 55 detalha esse resultado, comprovando a ausência de detecções do AP, demonstrando que o detector permaneceu estável diante de volumes de alerta que ainda não representam sobrecarga operacional.Por sua vez, a Figura 56 apresenta o resultado correspondente ao momento em que o volume de alertas alcançou dez ocorrências. A partir desse ponto, tem início a detecção do AP, uma vez que esse valor representa o limiar adotado nesta pesquisa como indicativo de fadiga para operadores em um intervalo de uma hora.

Complementarmente, a Figura 57 evidencia a evolução temporal do processo de detecção. É possível observar que, até o nono alerta, as barras permanecem destacadas em verde, indicando que o volume de notificações ainda se encontra em um nível aceitável. A partir do décimo alerta, contudo, o detector identifica a caracterização do AP, destacando o excesso de forma explícita.

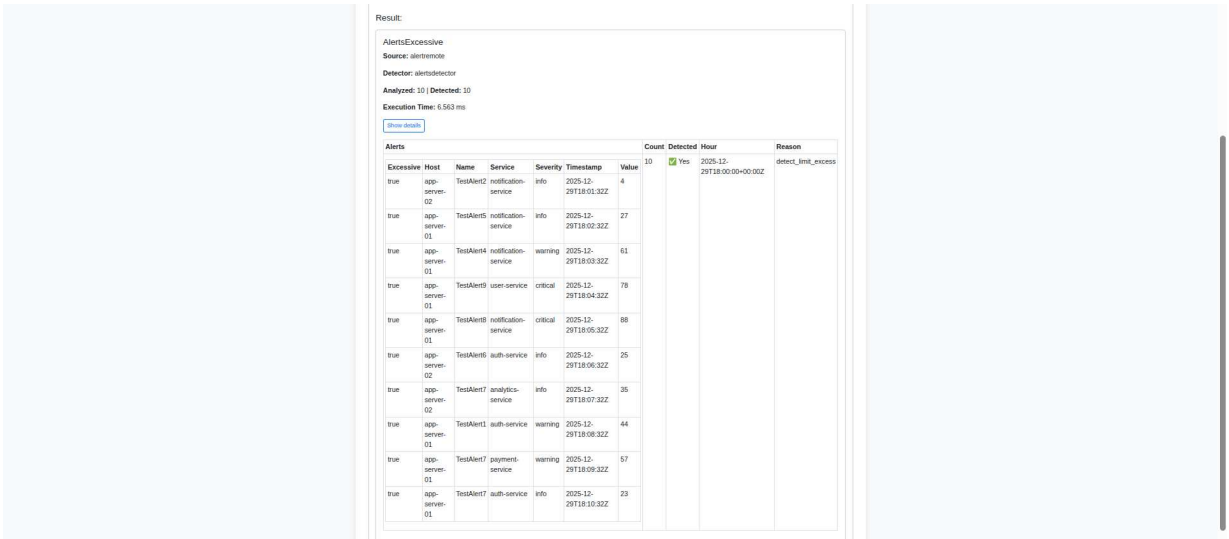
No que se refere ao segundo experimento, a Figura 58 apresenta a detecção do AP, observado neste cenário por meio da redundância de mensagens. Foi realizado a captura de 2470 alertas ao longo de um período de quatro minutos. Como previamente mencionado, a técnica de detecção de redundância do algoritmo proposto baseia-se na ocorrência de, pelo menos, dois

Figura 55 – Detalhes da detecção de alertas excessivos para o experimento 1.



Fonte: Autoria própria.

Figura 56 – Ocorrência de alertas excessivos.

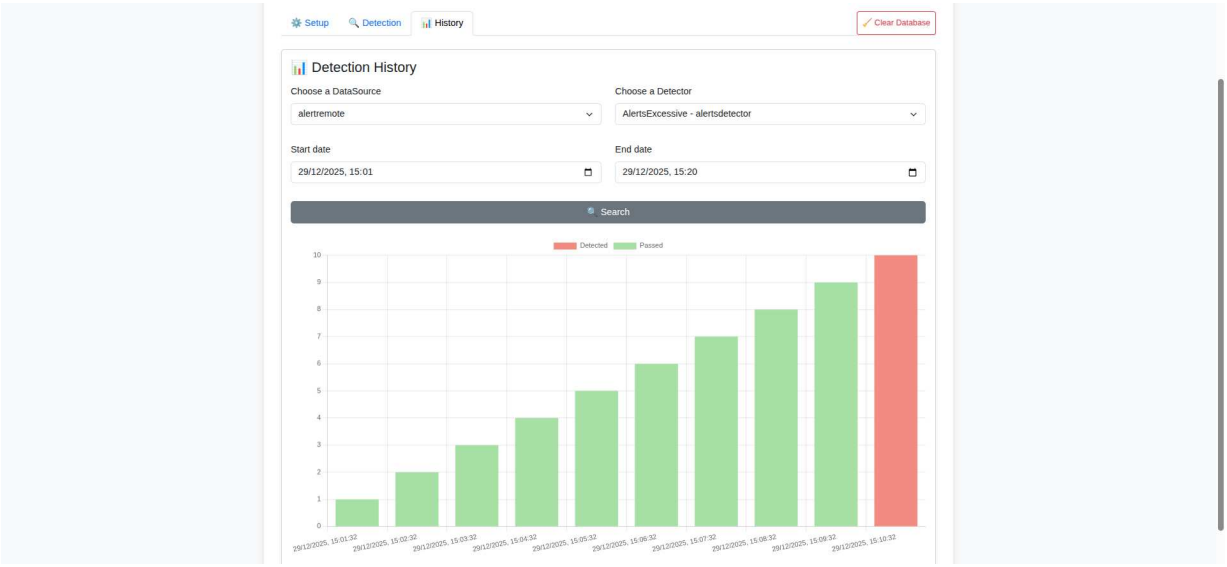


Fonte: Autoria própria.

alertas idênticos dentro de uma janela temporal de 60 segundos. Considerando a taxa registrada de mais de dez alertas por segundo durante a coleta, o Observa conseguiu identificar que havia pares de alertas redundantes.

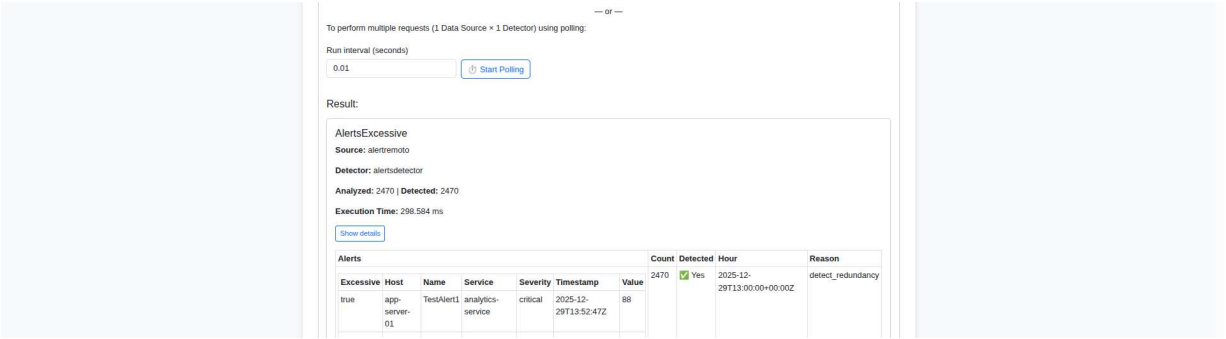
De forma geral, os resultados indicam que o Observa foi capaz de lidar adequadamente com a coleta incremental de dados e conduzir análises consolidadas ao final de cada execução, respondendo a **QA3**. O *framework* manteve consistência nos resultados, registrando corretamente o histórico das execuções e permitindo análises longitudinais sem perda de informações, o que valida sua adequação para cenários operacionais contínuos e dependentes de evolução temporal de eventos.

Figura 57 – Histórico da detecção de alertas excessivos.



Fonte: Autoria própria.

Figura 58 – Resultado da detecção de alertas excessivos por meio da redundância.



Fonte: Autoria própria.

7.6 Avaliação de Usabilidade Baseada em *Survey*

Além das avaliações quantitativas conduzidas com *datasets* reais e sintéticos, foi realizada uma avaliação empírica de caráter subjetivo, com o objetivo de analisar a percepção de usuários com experiência prévia em observabilidade quanto à usabilidade do Observa. Essa avaliação complementa os experimentos anteriores ao fornecer evidências qualitativas relacionadas à aceitação do *framework* enquanto solução prática para o apoio à detecção de APs.

7.6.1 Perfil dos Participantes e Instrumento

O instrumento adotado foi um *survey* estruturado, composto por questões baseadas no padrão *System Usability Scale* (SUS) (BROOKE *et al.*, 1996) e por uma questão adicional de recomendação no formato *Net Promoter Score* (NPS) (REICHHELD, 2003). O SUS é uma métrica padronizada amplamente utilizada para avaliar a usabilidade de sistemas interativos,

composta por dez afirmações respondidas em escala Likert de cinco pontos (concordo totalmente, concordo, neutro, discordo e discordo totalmente). O escore resultante é convertido para uma escala entre 0 e 100, permitindo quantificar de forma sintética a percepção global de usabilidade, sendo que valores acima de 68 são tradicionalmente interpretados como superiores à média[3]. Complementarmente, o NPS é um indicador destinado a mensurar a satisfação e a lealdade dos usuários, fundamentado em uma única pergunta sobre a probabilidade de recomendação do produto ou serviço a terceiros, com a escala de resposta adotada foi de 0 a 10 (em que 0 representa "Nada provável recomendar" e 10 corresponde a "Extremamente provável recomendar")[4].

No total, foram selecionados quatro participantes. Os participantes possuem experiência entre 1 e mais de 5 anos em observabilidade de sistemas distribuídos, atuando como DevOps, Desenvolvimento e Arquitetura de Software. Os participantes foram selecionados por contatos pessoais do autor da tese. Antes de responderem ao questionário, os participantes utilizaram o Observa para executar um fluxo completo de uso, guiados por um roteiro disponibilizado (Apêndice C).

A primeira tarefa consistiu na utilização das funcionalidades do Observa a partir de fontes e detectores previamente cadastrados, com o propósito de avaliar a consistência do *framework* e verificar se, de fato, ele é capaz de identificar os APs presentes nos dados. O fluxo dessa tarefa envolveu: (i) selecionar a fonte de dados; (ii) escolher o AP e o detector; (iii) executar o processo de detecção; (iv) analisar o resultado e seus detalhes; e, por fim, (v) consultar o histórico de execuções. Essa etapa foi essencial para garantir que as respostas fornecidas no questionário estivessem fundamentadas em uma experiência prática e direta de interação com o artefato.

Complementarmente, a segunda tarefa buscou avaliar a extensibilidade do Observa, ou seja, sua capacidade de incorporar novos componentes. Nessa etapa, cada participante foi orientado a cadastrar uma nova fonte de dados e criar um detector de APs, desenvolvido na tecnologia de sua preferência. Para apoiar a realização da tarefa, foram disponibilizados exemplos de código tanto da fonte quanto do detector, de forma a facilitar o desenvolvimento e execução dos componentes no ambiente do Observa.

Após a execução das tarefas, os participantes foram convidados a responder um formulário que foi utilizado para avaliar qualitativamente o Observa, e foi estruturado em três partes: (i) uma seção destinada à autoavaliação dos participantes; (ii) uma seção com questões

[3] <https://measuringu.com/interpret-sus-score/>
[4] <https://www.retently.com/blog/good-net-promoter-score/>

relacionadas às tarefas realizadas; e (iii) uma seção composta por perguntas qualitativas referentes ao SUS e ao NPS. Além dessas seções, o formulário também disponibilizou uma pergunta aberta opcional, com o propósito de permitir que os participantes sugerissem melhorias para o Observa ou registrassem comentários adicionais. O formulário, utilizado para coleta das respostas, foi disponibilizado aos participantes por meio da plataforma Google Forms.

7.6.2 Resultados do Survey

A Tabela 8 condensa os dados obtidos na aplicação do *survey*. Essa tabela possui a experiência de cada participante em anos, o tempo gasto em cada tarefa em minutos, o valor atribuído no SUS em porcentagem, de acordo com as respostas de cada um, bem como a recomendação do Observa, considerando o NPS. No que se refere à segunda seção, dedicada à avaliação das tarefas realizadas pelos participantes, verificou-se que, na primeira tarefa, todos os envolvidos conseguiram executar integralmente as atividades previstas. Cada participante informou, ainda, o tempo necessário para concluir a tarefa. O participante P1 registrou o menor tempo, finalizando em dois minutos (m), enquanto P4 apresentou o maior tempo, com oito minutos. Em média, considerando os quatro participantes, a execução da primeira tarefa demandou aproximadamente 4,5 $\pm$ 2,65 minutos. Esse resultado preliminar sugere que a curva de aprendizagem para utilização das funcionalidades existentes do Observa é reduzida, permitindo que novos usuários consigam utilizá-lo rapidamente e sem barreiras significativas.

Tabela 8 – Resultados do *survey* de avaliação de usabilidade.

Participante	**Experiência (ano)**	Tarefa 1 (m)	Tarefa 2 (m)	**SUS (%)**	**Recomendação (NPS)**
P1	1–3	2	15	85	8
P2	Mais de 5	3	10	95	9
P3	3–5	5	20	90	10
P4	Mais de 5	8	12	62,5	10

Fonte: Autoria própria.

De modo semelhante, na segunda tarefa todos os participantes também obtiveram êxito, sendo possível observar que os detectores desenvolvidos foram eficazes na identificação de APs. Em relação ao tempo gasto, o participante P2 foi o mais rápido, necessitando cerca de dez minutos para criar e executar os componentes necessários, ao passo que o participante P3 demandou o maior intervalo, aproximadamente vinte minutos. Em média, considerando os quatro participantes, a execução da segunda tarefa levou cerca de 14,25 $\pm$ 4,35 minutos. Esses tempos de execução, mais elevados quando comparados à primeira tarefa, indicam maior

complexidade na etapa de extensão do Observa, o que é esperado, dado que exige habilidades adicionais de programação e entendimento da arquitetura do *framework*. Ainda assim, o fato de todos os participantes terem conseguido desenvolver e integrar novos componentes evidencia a extensibilidade do Observa e sua capacidade de ser adotado em cenários reais, nos quais detectores personalizados podem ser necessários.

Além das questões objetivas, o formulário disponibilizou um campo aberto destinado à coleta de percepções livres dos participantes. De maneira geral, os comentários revelam uma experiência positiva com o Observa. Os participantes destacaram que o processo de configuração e execução das tarefas se mostrou intuitivo, com uma curva de aprendizagem reduzida, facilitando o funcionamento do *framework* e realização dos testes. Também foi ressaltado que a separação clara entre fontes de dados e detectores contribui significativamente para a flexibilidade e a extensibilidade da solução. P3 relatou que: "A experiência com o *framework* Observa foi bastante positiva. A possibilidade de integrar detectores externos por meio de APIs se destaca como um ponto forte, permitindo a criação de regras personalizadas de forma simples."

Os participantes também notaram que o Observa foi eficaz na identificação de APs de observabilidade durante a execução das tarefas, evidenciando situações que poderiam passar despercebidas em análises manuais. Ao mesmo tempo, indicaram sugestões que demonstram expectativas de evolução do sistema. P3 sugeriu que a geração de detectores pudesse ser parcialmente automatizada, utilizando Inteligência Artificial Generativa (IAG), de modo que, a partir das características das fontes de dados e de informações fornecidas pelo usuário, o sistema pudesse criar detectores iniciais, reduzindo o esforço necessário para usuários com menor experiência técnica. Também foram mencionadas possíveis melhorias na apresentação dos resultados, como o fornecimento de explicações mais detalhadas ou o uso de *feedbacks* visuais, igualmente com potencial apoio de IAG.

Apesar dos elogios, foram registradas também percepções negativas e dificuldades enfrentadas durante a execução. P4 relatou a necessidade de mudar de aba para acionar o botão de execução e foi percebida como problemática por provocar a sensação de perda de contexto e insegurança quanto à preservação dos parâmetros previamente selecionados. Outros comentários negativos envolveram a ausência de mecanismos de edição de fontes ou detectores já cadastrados. Ainda que tais dificuldades tenham sido citadas, o tom geral das mensagens manteve-se positivo.

Com relação à terceira seção do formulário, destinada à avaliação por meio do SUS e do NPS, os participantes atribuíram as pontuações apresentadas na Tabela 8. Para realizar

o cálculo do SUS, foi utilizado esta calculadora: <https://stuart-cunningham.github.io/sus/>. Com base nos resultados apresentados, o Observa obteve um escore SUS médio de **81,25%**, o qual, segundo a literatura de referência[5], caracteriza-se como **A – excelente**. Esse resultado indica percepção favorável quanto à facilidade de uso, aprendizado e fluidez geral de interação do sistema.

No que diz respeito ao NPS, conforme sua metodologia, as respostas são classificadas em três categorias: (i) Promotores (nota 9–10), (ii) Neutros (nota 7–8) e (iii) Detratores (nota 0–6). O valor do NPS é obtido subtraindo-se o percentual de detratores do percentual de promotores. No presente estudo, três participantes foram caracterizados como promotores e um como neutro, resultando em um NPS de **75%**, classificação considerada excelente[6]. Esse índice sugere forte predisposição de profissionais a recomendar o Observa como ferramenta de apoio operacional.

Essa avaliação complementa os resultados quantitativos anteriores ao fornecer evidências empíricas sobre a aceitação do Observa como artefato aplicável ao uso em campo, respondendo a **QA4**. Dessa forma, os achados permitem responder à **QP5**, ao indicar que a solução desenvolvida é extensível e de fácil utilização por desenvolvedores e operadores de aplicações baseadas em microsserviços.

7.7 Ameaças à Validade das Avaliações do Observa

As avaliações realizadas com o Observa foram conduzidas com rigor metodológico, porém algumas ameaças à validade devem ser consideradas ao interpretar os resultados obtidos.

7.7.1 Validade de Construção

As métricas utilizadas para caracterizar APs (por exemplo, volume por hora, redundância temporal e duração de alertas) foram definidas com base em literatura e em práticas reportadas, porém ainda dependem de limiares e configurações parametrizáveis. Assim, a caracterização operacional dos APs pode variar conforme ajustes específicos ou diferentes interpretações organizacionais do que constitui fadiga de alertas.

[5] <https://measuringu.com/interpret-sus-score/>
[6] <https://www.retently.com/blog/good-net-promoter-score/>

7.7.2 Validade Interna

Os experimentos com *datasets* reais e sintéticos foram executados pelo próprio autor da tese, o que pode introduzir viés na condução e interpretação desses resultados. Em contrapartida, a avaliação de usabilidade foi realizada diretamente pelos participantes, o que reduz essa ameaça para essa etapa específica.

7.7.3 Validade Externa

A generalização dos resultados é limitada pelo número de empresas participantes (duas organizações) e pela natureza dos conjuntos de dados fornecidos. Empresas com outros perfis setoriais, maturidade em observabilidade ou arquiteturas tecnológicas distintas podem produzir alertas com estruturas e padrões diferentes. Assim, os resultados obtidos não podem ser generalizados para todos os contextos industriais sem replicação empírica adicional.

7.7.4 Validade de Conclusão

Embora os resultados apontem para a efetividade do Observa, o tamanho reduzido da amostra no *survey* (quatro participantes) limita a força das conclusões relacionadas à usabilidade e aceitação. Ademais, os experimentos com *datasets* reais e sintéticos geraram resultados determinísticos a partir de execuções únicas, o que impede a análise de variabilidade.

7.8 Considerações Finais

Este capítulo apresentou a avaliação empírica do Observa em diferentes cenários de uso, complementando a prova de conceito do capítulo anterior e fornecendo evidências adicionais sobre sua aplicabilidade. As três abordagens adicionais adotadas permitiram responder às questões levantadas nesse capítulo.

A prova de conceito apresentada ao final do capítulo confirmou a viabilidade técnica do Observa e sua capacidade de identificar automaticamente ocorrências de APs de observabilidade, como alertas excessivos e *Logs* Excessivos. Os resultados obtidos indicam que o *framework* produz diagnósticos estruturados, rastreáveis e auditáveis, fornecendo subsídios concretos para a avaliação da qualidade da observabilidade em sistemas distribuídos.

No que se refere à QA1, observou-se que o Observa é capaz de processar dados reais

de observabilidade provenientes de diferentes organizações, identificando APs quando presentes e não produzindo falsos positivos em cenários estáveis. Em relação à QA2, os experimentos sintéticos demonstraram que o *framework* consegue lidar com a evolução temporal dos eventos, realizando análises incrementais e identificando a manifestação progressiva de APs ao longo do tempo. Por fim, a QA3 foi atendida ao se verificar que participantes com experiência prévia em observabilidade perceberam o Observa como uma solução útil e com boa usabilidade, apontando ainda potenciais caminhos de evolução.

Por fim, foram discutidas ameaças à validade que contextualizam a interpretação dos resultados, reforçando a necessidade de replicações e estudos adicionais para ampliar o grau de generalização das evidências aqui obtidas. No conjunto, os resultados demonstram que o Observa não apenas é tecnicamente viável, como também apresenta potencial concreto de adoção em ambientes reais de operação.

O capítulo seguinte apresenta as conclusões desta pesquisa, sintetizando suas contribuições, limitações e oportunidades de trabalhos futuros.

8 CONCLUSÃO E TRABALHOS FUTUROS

Este capítulo resume os principais elementos discutidos e desenvolvidos ao longo desta tese de doutorado. A Seção 8.1 apresenta os resultados alcançados, enquanto a Seção 8.2 descreve limitações identificadas no processo de desenvolvimento da tese. Por fim, a Seção 8.3 discorre sobre os trabalhos futuros e as perspectivas de continuidade desta linha de pesquisa.

8.1 Resultados Alcançados

As contribuições desta tese podem ser consolidadas em três eixos principais, que sintetizam os avanços conceituais, tecnológicos e práticos alcançados ao longo da pesquisa.

O primeiro eixo diz respeito à fundamentação conceitual, representada pela elaboração de uma taxonomia abrangente sobre observabilidade aplicada a sistemas distribuídos baseados em microsserviços. Essa taxonomia permitiu a classificação sistemática dos estudos existentes, consolidou categorias conceituais relevantes para a área e possibilitou compreender de forma estruturada os elementos que compõem o ecossistema de observabilidade. Integrado a esse eixo, foi conduzido um levantamento sobre o estado da arte, incluindo ferramentas de observabilidade, aplicações de *benchmarking* e conjuntos de dados de *workload* do mundo real empregados nos estudos analisados no contexto da experimentação. Esse panorama forneceu a base teórica e contextual necessária para sustentar e justificar a solução desenvolvida.

O segundo eixo consiste em um catálogo estruturado contendo 37 Anti-Padrões (APs) que comprometem a efetividade do monitoramento, diagnóstico, rastreamento e análise de sistemas distribuídos. Esse catálogo foi complementado pela implementação de um portal de acesso aberto, que reúne todo o conteúdo produzido, disponibiliza mecanismos de busca e filtragem e promove sua disseminação para profissionais e pesquisadores. Além disso, esse eixo inclui a validação qualitativa realizada com especialistas atuantes nas áreas de desenvolvimento, arquitetura e operação, assegurando a relevância, utilidade prática e das recomendações apresentadas.

Por fim, o terceiro eixo contempla o desenvolvimento e avaliação do Observa. O *framework* constitui a materialização prática da tese, demonstrando sua viabilidade prática e sua capacidade de apoiar equipes na identificação automática de problemas na utilização da observabilidade. Sua avaliação em ambiente experimental permitiu validar seu funcionamento e evidenciar seu potencial como ferramenta de apoio à observabilidade.

8.2 Limitações

Essa tese possui limitações decorrentes de suas escolhas metodológicas e de escopo. A primeira limitação refere-se ao ambiente de experimentação utilizado para avaliação do Observa, desenvolvido e testado em um cenário controlado, com cargas e topologias compatíveis com aplicações de porte médio. Embora adequado para validar a viabilidade técnica e o funcionamento do *framework*, esse ambiente não representa a diversidade de contextos encontrados em organizações de grande escala, o que limita a extrapolação imediata dos resultados.

Outra limitação está relacionada ao próprio escopo conceitual desta pesquisa, que se concentra exclusivamente em observabilidade no contexto de microsserviços. Não foram incluídas na taxonomia nem no catálogo toda a abrangência de sistemas distribuídos, o que abre espaço para ampliação em pesquisas subsequentes.

Além disso, apesar da quantidade de especialistas que avaliaram o catálogo e o Observa, é um conjunto reduzido de pessoas, embora qualificados e atuantes profissionalmente na área. Essa limitação quantitativa pode restringir a generalização dos resultados, ainda que o objetivo da validação tenha sido de natureza qualitativa.

Mesmo diante dessas limitações, os resultados obtidos demonstram que a abordagem é viável, aplicável e potencialmente útil para organizações.

8.3 Trabalhos Futuros

Com base nos resultados obtidos e nas limitações identificadas, esta tese abre espaço para diferentes possibilidades de continuidade. Entre os principais trabalhos futuros, destacam-se:

- **Ampliação do catálogo**: inclusão de novos APs emergentes, especialmente relacionados a arquiteturas orientadas a eventos, *service mesh* e *serverless*;
- **Automação avançada**: evolução do Observa com técnicas de correlação temporal e aprendizado de máquina, permitindo inferência preditiva e identificação baseada em padrões comportamentais;
- **Integração com ecossistemas DevOps**: desenvolvimento de extensões para uso direto dentro de pipelines de CI/CD, ambientes Kubernetes e painéis Grafana/Prometheus, possibilitando análise contínua de observabilidade no ciclo de entrega de software;
- **Validação industrial em grande escala**: condução de estudos em empresas com alta demanda transacional, avaliando o impacto do *framework* em indicadores organizacionais

como TTR (*time to repair*), MTBF (*mean time between failures*) e eficiência operacional.

- **Implantação de Soluções**: implementação de um módulo de soluções para os APs dentro do Observa, possibilitando além da identificação automática, a mitigação de problemas das infraestruturas de observabilidade.

REFERÊNCIAS

ABGAZ, Y.; MCCARREN, A.; ELGER, P.; SOLAN, D.; LAPUZ, N.; BIVOL, M.; JACKSON, G.; YILMAZ, M.; BUCKLEY, J.; CLARKE, P. Decomposition of monolith applications into microservices architectures: A systematic review. **IEEE Transactions on Software Engineering**, IEEE, v. 49, n. 8, p. 4213–4242, 2023.

AFSHINPOUR, B.; GROZ, R.; AMINI, M.-R. Telemetry-based software failure prediction by concept-space model creation. In: IEEE. **2022 IEEE 22nd International Conference on Software Quality, Reliability and Security (QRS)**. Guangzhou, China, 2022. p. 199–208.

AKBARI, M.; BOLLA, R.; BRUSCHI, R.; DAVOLI, F.; LOMBARDO, C.; SICCARDI, B. A monitoring, observability and analytics framework to improve the sustainability of b5g technologies. In: IEEE. **2024 IEEE International Conference on Communications Workshops (ICC Workshops)**. Denver, CO, USA, 2024. p. 969–975.

AL-DEBAGY, O.; MARTINEK, P. A comparative review of microservices and monolithic architectures. In: IEEE. **2018 IEEE 18th International Symposium on Computational Intelligence and Informatics (CINTI)**. Budapest, Hungary, 2018. p. 000149–000154.

ALEXANDER, C. **A pattern language: towns, buildings, construction**. New York, USA: Oxford university press, 1977.

AWS. **Anti-patterns for continuous monitoring**. 2025. Disponível em: <https://docs.aws.amazon.com/wellarchitected/latest/devops-guidance/anti-patterns-for-continuous-monitoring.html>. Acesso em: 10-06-2025.

AWS. **AWS Observability Best Practices**. 2025. Disponível em: <https://aws-observability.github.io/observability-best-practices/signals/logs>. Acesso em: 10-06-2025.

BAARZI, A. F.; KESIDIS, G. Showar: Right-sizing and efficient scheduling of microservices. In: **Proceedings of the ACM Symposium on Cloud Computing**. New York, NY, USA: Association for Computing Machinery, 2021. (SoCC '21), p. 427–441. ISBN 9781450386388. Disponível em: <https://doi.org/10.1145/3472883.3486999>.

BALALAIE, A.; HEYDARNOORI, A.; JAMSHIDI, P. Microservices architecture enables devops: Migration to a cloud-native architecture. **IEEE Software**, v. 33, n. 3, p. 42–52, 2016.

BHASI, V. M.; GUNASEKARAN, J. R.; THINAKARAN, P.; MISHRA, C. S.; KANDEMIR, M. T.; DAS, C. Kraken: Adaptive container provisioning for deploying dynamic dags in serverless platforms. In: **Proceedings of the ACM Symposium on Cloud Computing**. New York, NY, USA: Association for Computing Machinery, 2021. (SoCC '21), p. 153–167. ISBN 9781450386388. Disponível em: <https://doi.org/10.1145/3472883.3486992>.

BHATTACHARYA, B. **Bad API observability: 5 anti-patterns in 5 minutes**. 2024. Disponível em: <https://devopsdays.org/events/2024-houston/program/budhaditya-bhattacharya-ignite/>. Acesso em: 10-06-2025.

BI, Q.; LIU, Y.; ZHANG, L.; KANG, K.; YANG, D.; MIN, S. Dynamic scalability mechanisms for microservices in federated cloud platform. In: IEEE. **2023 IEEE 5th International Conference on Civil Aviation Safety and Information Technology (ICCASIT)**. Dali, China, 2023. p. 732–737.

BLINOWSKI, G.; OJDOWSKA, A.; PRZYBYŁEK, A. Monolithic vs. microservice architecture: A performance and scalability evaluation. **IEEE Access**, IEEE, v. 10, p. 20357–20374, 2022.

BONORDEN, L.; HOORN, A. van. Detecting usage of deprecated web apis via tracing. In: IEEE. **2024 IEEE 21st International Conference on Software Architecture (ICSA)**. Hyderabad, India, 2024. p. 23–33.

BORGES, M. C.; BAUER, J.; WERNER, S.; GEBAUER, M.; TAI, S. Informed and assessable observability design decisions in cloud-native microservice applications. In: IEEE. **2024 IEEE 21st International Conference on Software Architecture (ICSA)**. Hyderabad, India, 2024. p. 69–78.

BOTEN, A.; MAJORS, C. **Cloud-Native Observability with OpenTelemetry: Learn to gain visibility into systems by combining tracing, metrics, and logging with OpenTelemetry**. Packt Publishing, 2022. ISBN 9781801071901. Disponível em: <https://books.google.com.br/books?id=YZVsEAAAQBAJ>.

BROOKE, J. *et al.* Sus-a quick and dirty usability scale. **Usability evaluation in industry**, London, England, v. 189, n. 194, p. 4–7, 1996.

BROWN, W. H.; MALVEAU, R. C.; MCCORMICK, H. W. S.; MOWBRAY, T. J. **AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis**. 1st. ed. USA: John Wiley & Sons, Inc., 1998. ISBN 0471197130.

BRUNO, G. Z.; RODRIGUES, K. B. C.; CARDOSO, K. V.; CORREA, S. L.; BOTH, C. B. Anomaly detection in cloud-native b5g systems using observability and machine learning cots solutions. **Journal of Internet Services and Applications**, v. 14, n. 1, p. 189–199, 2023.

BURNS, B.; BEDA, J.; HIGHTOWER, K. **Kubernetes: up and running: dive into the future of infrastructure**. Sebastopol, CA, USA: O'Reilly Media, 2019.

C2 Wiki. **Anti-Pattern Template**. 2024. Disponível em: <https://wiki.c2.com/?AntiPatternTemplate>. Acesso em: 10-06-2025.

CAI, B.; WANG, B.; YANG, M.; GUO, Q. Automan: Resource-efficient provisioning with tail latency guarantees for microservices. **Future Generation Computer Systems**, Elsevier, v. 143, p. 61–75, 2023.

CARVALHO, A. P. dos S. **Observabilidade e telemetria em arquiteturas de micro-serviços**. Dissertação (Mestrado) — Universidade do Porto (Portugal), 2022.

CASSÉ, C.; BERTHOU, P.; OWEZARSKI, P.; JOSSET, S. Using distributed tracing to identify inefficient resources composition in cloud applications. In: IEEE. **2021 IEEE 10th International Conference on Cloud Networking (CloudNet)**. Cookeville, TN, USA, 2021. p. 40–47.

CASSÉ, C.; BERTHOU, P.; OWEZARSKI, P.; JOSSET, S. A tracing based model to identify bottlenecks in physically distributed applications. In: IEEE. **2022 International Conference on Information Networking (ICOIN)**. Jeju-si, Republic of Korea, 2022. p. 226–231.

CERNY, T.; ABDELFATTAH, A. S.; MARUF, A. A.; JANES, A.; TAIBI, D. Catalog and detection techniques of microservice anti-patterns and bad smells: A tertiary study. **Journal of Systems and Software**, Elsevier, v. 206, p. 111829, 2023.

CHAKRABORTY, A.; ESWARAN, A.; THORAT, P.; VERMA, M.; GUPTA, P.; JAYACHANDRAN, P. Intent-driven multi-engine observability dataflows for heterogeneous geo-distributed clouds. In: IEEE. **2024 IEEE 17th International Conference on Cloud Computing (CLOUD)**. Shenzhen, China, 2024. p. 30–41.

CHEN, B.; JIANG, Z. M. A survey of software log instrumentation. **ACM Computing Surveys (CSUR)**, ACM New York, NY, USA, v. 54, n. 4, p. 1–34, 2021.

CHEN, H.; CHEN, P.; YU, G.; LI, X.; HE, Z.; ZHANG, H. Microfi: Non-intrusive and prioritized request-level fault injection for microservice applications. **IEEE Transactions on Dependable and Secure Computing**, IEEE, 2024.

CHEVRE, S. **7 API Observability Anti-Patterns to Avoid**. 2024. Disponível em: <https://devops.com/7-api-observability-anti-patterns-to-avoid/>. Acesso em: 10-06-2025.

CHOW, K.-H.; DESHPANDE, U.; DEENADAYALAN, V.; SESHADRI, S.; LIU, L. Atlas: Hybrid cloud migration advisor for interactive microservices. In: **Proceedings of the Nineteenth European Conference on Computer Systems**. New York, NY, USA: Association for Computing Machinery, 2024. (EuroSys '24), p. 870–887. ISBN 9798400704376. Disponível em: <https://doi.org/10.1145/3627703.3629587>.

CHOW, K.-H.; DESHPANDE, U.; SESHADRI, S.; LIU, L. Deeprest: deep resource estimation for interactive microservices. In: **Proceedings of the Seventeenth European Conference on Computer Systems**. New York, NY, USA: Association for Computing Machinery, 2022. (EuroSys '22), p. 181–198. ISBN 9781450391627. Disponível em: <https://doi.org/10.1145/3492321.3519564>.

CHOWDHURY, R.; TALHI, C.; OULD-SLIMANE, H.; MOURAD, A. A framework for automated monitoring and orchestration of cloud-native applications. In: IEEE. **2020 International Symposium on Networks, Computers and Communications (ISNCC)**. Montreal, QC, Canada, 2020. p. 1–6.

CINQUE, M.; CORTE, R. D.; PECCHIA, A. Microservices monitoring with event logs and black box execution tracing. **IEEE transactions on services computing**, IEEE, v. 15, n. 1, p. 294–307, 2019.

CORTEZ, E.; BONDE, A.; MUZIO, A.; RUSSINOVICH, M.; FONTOURA, M.; BIANCHINI, R. Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms. In: **Proceedings of the 26th Symposium on Operating Systems Principles**. New York, NY, USA: Association for Computing Machinery, 2017. (SOSP '17), p. 153–167. ISBN 9781450350853. Disponível em: <https://doi.org/10.1145/3132747.3132772>.

COSTA, B.; BACHIEGA, J.; CARVALHO, L. R.; ROSA, M.; ARAUJO, A. Monitoring fog computing: A review, taxonomy and open challenges. **Computer Networks**, v. 215, p. 109189, 2022. ISSN 1389-1286. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1389128622002845>.

COSTA, B.; BANERJEE, A.; JAYARAMAN, P. P.; CARVALHO, L. R.; BACHIEGA, J.; ARAUJO, A. Achieving observability on fog computing with the use of open-source tools. In: ZASLAVSKY, A.; NING, Z.; KALOGERAKI, V.; GEORGAKOPOULOS, D.; CHRYSANTHIS, P. K. (Ed.). **Mobile and Ubiquitous Systems: Computing, Networking and Services**. Cham: Springer Nature Switzerland, 2024. p. 319–340. ISBN 978-3-031-63992-0.

DENARO, G.; MOUSSA, N. E.; HEYDAROV, R.; LOMIO, F.; PEZZÈ, M.; QIU, K. Predicting failures of autoscaling distributed applications. **Proceedings of the ACM on Software Engineering**, ACM New York, NY, USA, v. 1, n. FSE, p. 1960–1981, 2024.

DRAGONI, N.; GIALLORENZO, S.; LAFUENTE, A. L.; MAZZARA, M.; MONTESI, F.; MUSTAFIN, R.; SAFINA, L. Microservices: yesterday, today, and tomorrow. **Present and ulterior software engineering**, Springer, p. 195–216, 2017.

DUARTE, P.; CARVALHO, R.; VIANA, W. A catalog of interoperability solutions for ambient assisted living. In: **Anais do XVIII Simpósio Brasileiro de Componentes, Arquiteturas e Reutilização de Software**. Porto Alegre, RS, Brasil: SBC, 2024. p. 61–70. ISSN 0000-0000. Disponível em: <https://sol.sbc.org.br/index.php/sbcars/article/view/30233>.

DUVALL, P. M.; MATYAS, S.; GLOVER, A. **Continuous integration: improving software quality and reducing risk**. Boston, USA: Pearson Education, 2007.

ERAKOVIC, I.; PAHL, C. Hybrid root cause analysis for partially observable microservices based on architecture profiling. In: INSTICC. **Proceedings of the 15th International Conference on Cloud Computing and Services Science - Volume 1: CLOSER**. Porto, Portugal: SciTePress, 2025. p. 255–263. ISBN 978-989-758-747-4.

FLANDERS, S. **Antipatterns in Observability: Lessons Learned and How OpenTelemetry Solves Them**. 2025. Disponível em: <https://www.youtube.com/watch?v=2xX3Fx_WVSo>. Acesso em: 10-06-2025.

FRANCESCO, P. D.; LAGO, P.; MALAVOLTA, I. Architecting with microservices: A systematic mapping study. **Journal of Systems and Software**, Elsevier, v. 150, p. 77–97, 2019.

FRANCESCO, P. D.; MALAVOLTA, I.; LAGO, P. Research on architecting microservices: Trends, focus, and potential for industrial adoption. In: IEEE. **2017 IEEE International conference on software architecture (ICSA)**. Gothenburg, Sweden, 2017. p. 21–30.

GAN, Y.; LIANG, M.; DEV, S.; LO, D.; DELIMITROU, C. 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**. New York, NY, USA: Association for Computing Machinery, 2021. (ASPLOS '21), p. 135–151. ISBN 9781450383172. Disponível em: <https://doi.org/10.1145/3445814.3446700>.

GAN, Y.; ZHANG, Y.; CHENG, D.; SHETTY, A.; RATHI, P.; KATARKI, N.; BRUNO, A.; HU, J.; RITCHKEN, B.; JACKSON, B.; HU, K.; PANCHOLI, M.; HE, Y.; CLANCY, B.; COLEN, C.; WEN, F.; LEUNG, C.; WANG, S.; ZARUVINSKY, L.; ESPINOSA, M.; LIN, R.; LIU, Z.; PADILLA, J.; DELIMITROU, C. An open-source benchmark suite for microservices and their hardware-software implications for cloud & edge systems. In: **Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems**. New York, NY, USA: Association for Computing Machinery, 2019. (ASPLOS '19), p. 3–18. ISBN 9781450362405. Disponível em: <https://doi.org/10.1145/3297858.3304013>.

GANNON, D.; BARGA, R.; SUNDARESAN, N. Cloud-native applications. **IEEE Cloud Computing**, IEEE, v. 4, n. 5, p. 16–21, 2017.

GIAMATTEI, L.; GUERRIERO, A.; PIETRANTUONO, R.; RUSSO, S.; MALAVOLTA, I.; ISLAM, T.; DÎNGA, M.; KOZIOLEK, A.; SINGH, S.; ARMBRUSTER, M. *et al.* Monitoring tools for devops and microservices: A systematic grey literature review. **Journal of Systems and Software**, Elsevier, p. 111906, 2023.

GOMES, F.; REGO, P.; TRINTA, F. Rumo a uma taxonomia de observabilidade para aplicações baseadas em microsserviços. In: **Anais do XXXVIII Simpósio Brasileiro de Engenharia de Software**. Porto Alegre, RS, Brasil: SBC, 2024. p. 234–245. ISSN 2833-0633. Disponível em: <https://sol.sbc.org.br/index.php/sbes/article/view/30365>.

GOMES, F.; REGO, P.; TRINTA, F. Rumo a um catálogo de anti-padrões de observabilidade: Uma revisão da literatura. In: **Anais do XXXIX Simpósio Brasileiro de Engenharia de Software**. Porto Alegre, RS, Brasil: SBC, 2025. p. 104–114. ISSN 2833-0633. Disponível em: <https://sol.sbc.org.br/index.php/sbes/article/view/36990>.

GOMES, F.; REGO, P.; TRINTA, F. A systematic mapping study on observability of microservices-based applications: fundamentals, classifications, and challenges. **Computing**, Springer, v. 107, n. 9, p. 1–25, 2025.

GOMES, F. A.; ROCHA, L. S.; REGO, P. A.; TRINTA, F. A. Using observability to detect anti-patterns in benchmarking application with opentelemetry. In: IEEE. **2024 IEEE 13th International Conference on Cloud Networking (CloudNet)**. Rio de Janeiro, Brazil, 2024. p. 1–8.

GORTNEY, M. E.; HARRIS, P. E.; CERNY, T.; MARUF, A. A.; BURES, M.; TAIBI, D.; TISNOVSKY, P. Visualizing microservice architecture in the dynamic perspective: A systematic mapping study. **IEEE Access**, IEEE, v. 10, p. 119999–120012, 2022.

GU, S.; RONG, G.; REN, T.; ZHANG, H.; SHEN, H.; YU, Y.; LI, X.; OUYANG, J.; CHEN, C. Trinityrcl: Multi-granular and code-level root cause localization using multiple types of telemetry data in microservice systems. **IEEE Transactions on Software Engineering**, IEEE, v. 49, n. 5, p. 3071–3088, 2023.

GU, S.; RONG, G.; ZHANG, H.; SHEN, H. Logging practices in software engineering: A systematic mapping study. **IEEE transactions on software engineering**, IEEE, v. 49, n. 2, p. 902–923, 2022.

HAN, Y.; DU, Q.; HUANG, Y.; LI, P.; SHI, X.; WU, J.; FANG, P.; TIAN, F.; HE, C. Holistic root cause analysis for failures in cloud-native systems through observability data. **IEEE Transactions on Services Computing**, IEEE, 2024.

HE, S.; HE, P.; CHEN, Z.; YANG, T.; SU, Y.; LYU, M. R. A survey on automated log analysis for reliability engineering. **ACM computing surveys (CSUR)**, ACM New York, NY, USA, v. 54, n. 6, p. 1–37, 2021.

HUMBLE, J.; FARLEY, D. **Continuous delivery: reliable software releases through build, test, and deployment automation**. London, United Kingdom: Pearson Education, 2010.

JANES, A.; LI, X.; LENARDUZZI, V. Open tracing tools: Overview and critical comparison. **Journal of Systems and Software**, Elsevier, p. 111793, 2023.

KALMAN, R. On the general theory of control systems. **IRE Transactions on Automatic Control**, v. 4, n. 3, p. 110–110, 1959.

KANNAN, P. G.; GUPTA, S. M.; BEHL, D.; RAICHSTEIN, E.; TAKVORIAN, J. Designing a lightweight network observability agent for cloud applications. In: **Passive and Active Measurement: 25th International Conference, PAM 2024, Virtual Event, March 11–13, 2024, Proceedings, Part I**. Berlin, Heidelberg: Springer-Verlag, 2024. p. 262–276. ISBN 978-3-031-56248-8. Disponível em: <https://doi.org/10.1007/978-3-031-56249-5_11>.

KARUMURI, S.; SOLLEZA, F.; ZDONIK, S.; TATBUL, N. Towards observability data management at scale. **SIGMOD Rec.**, Association for Computing Machinery, New York, NY, USA, v. 49, n. 4, p. 18–23, mar. 2021. ISSN 0163-5808. Disponível em: <https://doi.org/10.1145/3456859.3456863>.

KAUSHIK, N.; KUMAR, H.; RAJ, V.; GARG, P. Proactive fault prediction in microservices applications using trace logs and monitoring metrics. In: IEEE. **2024 International Conference on Progressive Innovations in Intelligent Systems and Data Science (ICPIDS)**. Pattaya, Thailand, 2024. p. 410–415.

KAWASAKI, J.; KOYAMA, D.; MIYASAKA, T.; OTANI, T. Failure prediction in cloud native 5g core with ebpf-based observability. In: IEEE. **2023 IEEE 97th Vehicular Technology Conference (VTC2023-Spring)**. Florence, Italy, 2023. p. 1–6.

KHICHANE, A.; FAJJARI, I.; AITSAADI, N.; GUEROUI, M. 5gc-observer: a non-intrusive observability framework for cloud native 5g system. In: IEEE. **NOMS 2023-2023 IEEE/IFIP Network Operations and Management Symposium**. Miami, FL, USA, 2023. p. 1–10.

KOENIG, A. **Patterns and antipatterns**. USA: Cambridge University Press, 1998. 383–389 p. ISBN 0521648181.

KOSIŃSKA, J.; BALIŚ, B.; KONIECZNY, M.; MALAWSKI, M.; ZIELIŃSKI, S. Towards the observability of cloud-native applications: The overview of the state-of-the-art. **IEEE Access**, IEEE, 2023.

KOSIŃSKA, J.; ZIELIŃSKI, K. Autonomic management framework for cloud-native applications. **Journal of Grid Computing**, Springer, v. 18, p. 779–796, 2020.

KRATZKE, N. Cloud-native observability: the many-faceted benefits of structured and unified logging—a multi-case study. **Future Internet**, MDPI, v. 14, n. 10, p. 274, 2022.

KRATZKE, N.; QUINT, P.-C. Understanding cloud-native applications after 10 years of cloud computing-a systematic mapping study. **Journal of Systems and Software**, Elsevier, v. 126, p. 1–16, 2017.

KUMAR, A.; AHMED, T.; SAINI, K.; KUMAR, J. Neos: Non-intrusive edge observability stack based on zero trust security model for ubiquitous computing. In: IEEE. **2023 IEEE International Conference on Edge Computing and Communications (EDGE)**. Chicago, IL, USA, 2023. p. 79–84.

LAROCK, T. **Everything You Know About Observability is Wrong**. 2023. Disponível em: <https://www.selector.ai/blog/everything-you-know-about-observability-is-wrong/>. Acesso em: 10-06-2025.

LEVIN, J.; BENSON, T. A. Viperprobe: Rethinking microservice observability with ebpf. In: IEEE. **2020 IEEE 9th International Conference on Cloud Networking (CloudNet)**. Piscataway, NJ, USA, 2020. p. 1–8.

LEWIS, J.; FOWLER, M. **Microservices: A Definition of This New Architectural Term**. 2014. Disponível em: <https://martinfowler.com/articles/microservices.html>. Acesso em: 10-06-2025.

LI, B.; PENG, X.; XIANG, Q.; WANG, H.; XIE, T.; SUN, J.; LIU, X. Enjoy your observability: an industrial survey of microservice tracing and analysis. **Empirical Software Engineering**, Springer, v. 27, p. 1–28, 2022.

LI, M.; TANG, D.; WEN, Z.; CHENG, Y. Microservice anomaly detection based on tracing data using semi-supervised learning. In: IEEE. **2021 4th International Conference on Artificial Intelligence and Big Data (ICAIBD)**. Chengdu, China, 2021. p. 38–44.

LI, X.; ZHOU, J.; WEI, X.; LI, D.; QIAN, Z.; WU, J.; QIN, X.; LU, S. Topology-aware scheduling framework for microservice applications in cloud. **IEEE Transactions on Parallel and Distributed Systems**, IEEE, v. 34, n. 5, p. 1635–1649, 2023.

LI, Z.; CHEN, J.; JIAO, R.; ZHAO, N.; WANG, Z.; ZHANG, S.; WU, Y.; JIANG, L.; YAN, L.; WANG, Z. *et al.* Practical root cause localization for microservice systems via trace analysis. In: IEEE. **2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQOS)**. Tokyo, Japan, 2021. p. 1–10.

LIAO, J.; ZHOU, Z.; XU, F.; CHEN, X. Staaf: Spatial-temporal correlations aware autoscaling framework for microservices. In: IEEE. **2023 IEEE Smart World Congress (SWC)**. Portsmouth, United Kingdom, 2023. p. 1–9.

LIU, D.; HE, C.; PENG, X.; LIN, F.; ZHANG, C.; GONG, S.; LI, Z.; OU, J.; WU, Z. Microhecl: high-efficient root cause localization in large-scale microservice systems. In: **Proceedings of the 43rd International Conference on Software Engineering: Software Engineering in Practice**. IEEE Press, 2021. (ICSE-SEIP '21), p. 338–347. ISBN 9780738146690. Disponível em: <https://doi.org/10.1109/ICSE-SEIP52600.2021.00043>.

LIU, J.; WANG, Q.; ZHANG, S.; HU, L.; SILVA, D. D. Sora: A latency sensitive approach for microservice soft resource adaptation. In: **Proceedings of the 24th International Middleware Conference**. New York, NY, USA: Association for Computing Machinery, 2023. (Middleware '23), p. 43–56. ISBN 9798400701771. Disponível em: <https://doi.org/10.1145/3590140.3592851>.

LIU, J.; ZHANG, S.; WANG, Q. μconadapter: Reinforcement learning-based fast concurrency adaptation for microservices in cloud. In: **Proceedings of the 2023 ACM Symposium on Cloud Computing**. New York, NY, USA: Association for Computing Machinery, 2023. (SoCC '23), p. 427–442. ISBN 9798400703874. Disponível em: <https://doi.org/10.1145/3620678.3624980>.

LUO, S.; XU, H.; YE, K.; XU, G.; ZHANG, L.; YANG, G.; XU, C. The power of prediction: microservice auto scaling via workload learning. In: **Proceedings of the 13th Symposium on Cloud Computing**. New York, NY, USA: Association for Computing Machinery, 2022. (SoCC '22), p. 355–369. ISBN 9781450394147. Disponível em: <https://doi.org/10.1145/3542929.3563477>.

MAGGIORI, E.; TARABALKA, Y.; CHARPIAT, G.; ALLIEZ, P. Can semantic labeling methods generalize to any city? the inria aerial image labeling benchmark. In: IEEE. **2017 IEEE International geoscience and remote sensing symposium (IGARSS)**. Fort Worth, TX, USA, 2017. p. 3226–3229.

MARCHESE, A.; TOMARCHIO, O. Telemetry-driven microservices orchestration in cloud-edge environments. In: IEEE. **2024 IEEE 17th International Conference on Cloud Computing (CLOUD)**. Shenzhen, China, 2024. p. 91–101.

MARIE-MAGDELAINE, N. **Observability and resources managements in cloud-native environnements**. Tese (Theses) — Université de Bordeaux, nov. 2021. Disponível em: <https://theses.hal.science/tel-03486157>.

MARIE-MAGDELAINE, N.; AHMED, T. Proactive autoscaling for cloud-native applications using machine learning. In: **GLOBECOM 2020 - 2020 IEEE Global Communications Conference**. IEEE Press, 2020. p. 1–7. Disponível em: <https://doi.org/10.1109/GLOBECOM42002.2020.9322147>.

MARUF, A. A.; BAKHTIN, A.; CERNY, T.; TAIBI, D. Using microservice telemetry data for system dynamic analysis. In: IEEE. **2022 IEEE International Conference on Service-Oriented System Engineering (SOSE)**. Newark, CA, USA, 2022. p. 29–38.

MENG, C.; SONG, S.; TONG, H.; PAN, M.; YU, Y. Deepscaler: Holistic autoscaling for microservices based on spatiotemporal gnn with adaptive graph learning. In: **Proceedings of the 38th IEEE/ACM International Conference on Automated Software Engineering**. IEEE Press, 2024. (ASE '23), p. 53–65. ISBN 9798350329964. Disponível em: <https://doi.org/10.1109/ASE56229.2023.00038>.

MHAISEKAR, S. **10 tips to BAD Observability**. 2024. Disponível em: <https://www.youtube.com/watch?v=ljyMJ1i-zxI>. Acesso em: 10-06-2025.

MIN, S.; GUO, L.; WENG, G. Alert fatigue mitigation in anomaly detection systems: A comparative study of threshold optimization and alert aggregation strategies. **Journal of Computing Innovations and Applications**, v. 1, n. 2, p. 59–73, 2023.

MITTAL, D. **Mistakes to avoid in Observability**. 2022. Disponível em: <https://notes.drdroid.io/mistakes-to-avoid-in-observability>. Acesso em: 10-06-2025.

MONSEF, T.; ASHTIANI, M. Detecting microservice's architectural anti-pattern indicators using graph neural networks. **Software: Practice and Experience**, v. 56, n. 3, p. 273–288, 2026. Disponível em: <https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.70035>.

MONTEIRO, D.; YU, Y.; ZISMAN, A.; NUSEIBEH, B. Adaptive observability for forensic-ready microservice systems. **IEEE Transactions on Services Computing**, IEEE, 2023.

NAQVI, M. A.; MALIK, S.; ASTEKIN, M.; MOONEN, L. On evaluating self-adaptive and self-healing systems using chaos engineering. In: IEEE. **2022 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS)**. CA, USA, 2022. p. 1–10.

NEDELKOSKI, S.; CARDOSO, J.; KAO, O. Anomaly detection from system tracing data using multimodal deep learning. In: IEEE. **2019 IEEE 12th International Conference on Cloud Computing (CLOUD)**. Milan, Italy, 2019. p. 179–186.

NEILL, C. J.; LAPLANTE, P. A. **Antipatterns: identification, refactoring, and management**. Florida, USA: Auerbach Publications, 2005.

NI, C. **Adopting Observability-Driven Development for Cloud-Native Applications : Designing End-to-end Observability Pipeline using Open-source Software**. 62 p. Dissertação (Mestrado) — KTH, School of Electrical Engineering and Computer Science (EECS), 2023.

NIEDERMAIER, S.; KOETTER, F.; FREYMANN, A.; WAGNER, S. On observability and monitoring of distributed systems – an industry interview study. In: YANGUI, S.; RODRIGUEZ, I. B.; DRIRA, K.; TARI, Z. (Ed.). **Service-Oriented Computing**. Cham: Springer International Publishing, 2019. p. 36–52. ISBN 978-3-030-33702-5.

NOETZOLD, D.; ROSSETTO, A. G. D. M.; LEITHARDT, V. R. Q.; COSTA, H. J. d. M. Enhancing infrastructure observability: Machine learning for proactive monitoring and anomaly detection. **Journal of Internet Services and Applications**, v. 15, n. 1, p. 508–522, Oct. 2024. Disponível em: <https://journals-sol.sbc.org.br/index.php/jisa/article/view/4509>.

O'DONNELL, J. **Top 10 Mistakes People Make When Building Observability Dashboards**. 2024. Disponível em: <https://logz.io/blog/top-10-mistakes-building-observability-dashboards/>. Acesso em: 10-06-2025.

OTERO, M.; GARCIA, J. M.; FERNANDEZ, P. Towards a lightweight distributed telemetry for microservices. In: IEEE. **2024 IEEE 44th International Conference on Distributed Computing Systems Workshops (ICDCSW)**. Jersey City, NJ, USA, 2024. p. 75–82.

PAHL, C.; BROGI, A.; SOLDANI, J.; JAMSHIDI, P. Cloud container technologies: a state-of-the-art review. **IEEE Transactions on Cloud Computing**, IEEE, v. 7, n. 3, p. 677–692, 2017.

PARK, J.; CHOI, B.; LEE, C.; HAN, D. Graf: a graph neural network based proactive resource allocation framework for slo-oriented microservices. In: **Proceedings of the 17th International Conference on Emerging Networking EXperiments and Technologies**. New York, NY, USA: Association for Computing Machinery, 2021. (CoNEXT '21), p. 154–167. ISBN 9781450390989. Disponível em: <https://doi.org/10.1145/3485983.3494866>.

PATEL, A. **How to Resolve Bad Observability Data Quality**. 2024. Disponível em: <https://read.srepath.com/p/how-to-resolve-bad-observability>. Acesso em: 10-06-2025.

PATEL, A. **How to Solve 3 Data Flow Issues in Observability**. 2024. Disponível em: <https://read.srepath.com/p/how-to-solve-3-data-flow-issues-in>. Acesso em: 10-06-2025.

PETERSEN, K.; FELDT, R.; MUJTABA, S.; MATTSSON, M. Systematic mapping studies in software engineering. In: **Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering**. Swindon, GBR: BCS Learning & Development Ltd., 2008. (EASE'08), p. 68–77.

POURMAJIDI, W.; ZHANG, L.; STEINBACHER, J.; ERWIN, T.; MIRANSKYY, A. A Reference Architecture for Governance of Cloud Native Applications . **IEEE Transactions on Cloud Computing**, IEEE Computer Society, Los Alamitos, CA, USA, v. 13, n. 03, p. 935–952, jul. 2025. ISSN 2168-7161. Disponível em: <https://doi.ieeecomputersociety.org/10.1109/TCC.2025.3578557>.

QIU, H.; BANERJEE, S. S.; JHA, S.; KALBARCZYK, Z. T.; IYER, R. K. Firm: an intelligent fine-grained resource management framework for slo-oriented microservices. In: **Proceedings of the 14th USENIX Conference on Operating Systems Design and Implementation**. USA: USENIX Association, 2020. (OSDI'20). ISBN 978-1-939133-19-9.

RAEISZADEH, M.; EBRAHIMZADEH, A.; SALEEM, A.; GLITHO, R. H.; EKER, J.; MINI, R. A. Real-time anomaly detection using distributed tracing in microservice cloud applications.

In: IEEE. **2023 IEEE 12th International Conference on Cloud Networking (CloudNet)**. Hoboken, NJ, USA, 2023. p. 36–44.

RAZZAQ, A.; GHAYYUR, S. A. A systematic mapping study: The new age of software architecture from monolithic to microservice architecture—awareness and challenges. **Computer Applications in Engineering Education**, Wiley Online Library, v. 31, n. 2, p. 421–451, 2023.

REICHHELD, F. F. The one number you need to grow. **Harvard business review**, v. 81, n. 12, p. 46–55, 2003.

REN, R.; WANG, Y.; LIU, F.; LI, Z.; XIE, G. Triple: The interpretable deep learning anomaly detection framework based on trace-metric-log of microservice. In: IEEE. **2023 IEEE/ACM 31st International Symposium on Quality of Service (IWQoS)**. Orlando, FL, USA, 2023. p. 1–10.

REZVANI, M.; JAHANSHAHI, A.; WONG, D. Characterizing in-kernel observability of latency-sensitive request-level metrics with ebpf. In: IEEE. **2024 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)**. Indianapolis, IN, USA, 2024. p. 24–35.

RICHARDSON, C. **Microservices patterns: with examples in Java**. New York, USA: Simon and Schuster, 2018.

RIOS, J.; JHA, S.; SHWARTZ, L. Localizing and explaining faults in microservices using distributed tracing. In: IEEE. **2022 IEEE 15th International Conference on Cloud Computing (CLOUD)**. Barcelona, Spain, 2022. p. 489–499.

RODRIGUEZ, M. A.; BUYYA, R. Container-based cluster orchestration systems: A taxonomy and future directions. **Software: Practice and Experience**, Wiley Online Library, v. 49, n. 5, p. 698–719, 2019.

ROSSI, R. A.; AHMED, N. K. The network data repository with interactive graph analytics and visualization. In: **Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence**. Austin, Texas: AAAI Press, 2015. (AAAI'15), p. 4292–4293. ISBN 0262511290.

SAHA, A.; AGARWAL, P.; GHOSH, S.; GANTAYAT, N.; SINDHGATTA, R. Towards business process observability. In: **Proceedings of the 7th Joint International Conference on Data Science & Management of Data (11th ACM IKDD CODS and 29th COMAD)**. New York, NY, USA: Association for Computing Machinery, 2024. (CODS-COMAD '24), p. 257–265. ISBN 9798400716348. Disponível em: <https://doi.org/10.1145/3632410.3632435>.

SCROCCA, M.; TOMMASINI, R.; MARGARA, A.; VALLE, E. D.; SAKR, S. The kaiju project: enabling event-driven observability. In: **Proceedings of the 14th ACM International Conference on Distributed and Event-Based Systems**. New York, NY, USA: Association for Computing Machinery, 2020. (DEBS '20), p. 85–96. ISBN 9781450380287. Disponível em: <https://doi.org/10.1145/3401025.3401740>.

SENAPATHI, M.; BUCHAN, J.; OSMAN, H. 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**. New York, NY, USA: Association for Computing Machinery, 2018. (EASE '18), p. 57–67. ISBN 9781450364034. Disponível em: <https://doi.org/10.1145/3210459.3210465>.

SHADIJA, D.; REZAI, M.; HILL, R. Towards an understanding of microservices. In: **2017 23rd International Conference on Automation and Computing (ICAC)**. Huddersfield, UK: IEEE, 2017. p. 1–6.

SHAHIN, M.; BABAR, M. A.; ZHU, L. Continuous integration, delivery and deployment: a systematic review on approaches, tools, challenges and practices. **IEEE access**, IEEE, v. 5, p. 3909–3943, 2017.

SHARMA, B.; NADIG, D. ebpf-enhanced complete observability solution for cloud-native microservices. In: IEEE. **ICC 2024-IEEE International Conference on Communications**. Denver, CO, USA, 2024. p. 1980–1985.

SHATNAWI, A.; RIMA, B.; ALSHARA, Z.; DARBORD, G.; SERIAI, A.-D.; BORTOLASO, C. Telemetry of legacy web applications: An industrial case study. **TechRxiv**, v. 2023, n. 1101, 2023. Disponível em: <https://www.techrxiv.org/doi/abs/10.36227/techrxiv.24449092.v1>.

SHEN, J.; ZHANG, H.; XIANG, Y.; SHI, X.; LI, X.; SHEN, Y.; ZHANG, Z.; WU, Y.; YIN, X.; WANG, J.; XU, M.; LI, Y.; YIN, J.; SONG, J.; LI, Z.; NIE, R. Network-centric distributed tracing with deepflow: Troubleshooting your microservices in zero code. In: **Proceedings of the ACM SIGCOMM 2023 Conference**. New York, NY, USA: Association for Computing Machinery, 2023. (ACM SIGCOMM '23), p. 420–437. ISBN 9798400702365. Disponível em: <https://doi.org/10.1145/3603269.3604823>.

SIMONSSON, J.; ZHANG, L.; MORIN, B.; BAUDRY, B.; MONPERRUS, M. Observability and chaos engineering on system calls for containerized applications in docker. **Future Generation Computer Systems**, v. 122, p. 117–129, 2021. ISSN 0167-739X. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0167739X21001163>.

SINGLETON, A. The economics of microservices. **IEEE Cloud Computing**, IEEE, v. 3, n. 5, p. 16–20, 2016.

SOLDANI, D.; NAHI, P.; BOUR, H.; JAFARIZADEH, S.; SOLIMAN, M. F.; GIOVANNA, L. D.; MONACO, F.; OGNIBENE, G.; RISSO, F. ebpf: A new approach to cloud-native observability, networking and security for current (5g) and future mobile networks (6g and beyond). **IEEE Access**, IEEE, v. 11, p. 57174–57202, 2023.

SON, M.; MOHANTY, S.; GUNASEKARAN, J. R.; KANDEMIR, M. Microblend: An automated service-blending framework for microservice-based cloud applications. In: IEEE. **2023 IEEE 16th International Conference on Cloud Computing (CLOUD)**. [S.l.], 2023. p. 460–470.

SONG, Y.; LI, C.; ZHUANG, K.; MA, T.; WO, T. An automatic scaling system for online application with microservices architecture. In: IEEE. **2022 IEEE International Conference on Joint Cloud Computing (JCC)**. Fremont, CA, USA, 2022. p. 73–78.

SP, P. S.; VISHNU, S. S.; KUMAR, R.; BAILUGUTTU, S. Enhancement of observability using kubernetes operator. **Indonesian Journal of Electrical Engineering and Computer Science**, v. 25, n. 1, p. 496–503, 2022.

SREEMATHY, S.; KUMAR, V.; PRIYADHARSHINI, S. Application monitoring and telemetry analytics. In: IEEE. **2024 7th International Conference on Circuit Power and Computing Technologies (ICCPCT)**. Kollam, India, 2024. v. 1, p. 1559–1565.

SUNDQVIST, T.; BHUYAN, M.; ELMROTH, E. Robust procedural learning for anomaly detection and observability in 5g ran. **IEEE Transactions on Network and Service Management**, IEEE, 2023.

TAIBI, D.; KEHOE, B.; POCCIA, D. Serverless: from bad practices to good solutions. In: IEEE. **2022 IEEE International Conference on Service-Oriented System Engineering (SOSE)**. Newark, CA, USA, 2022. p. 85–92.

TALAVER, O. V.; VAKALIUK, T. A. Dynamic system analysis using telemetry. In: CEUR-WS. **CS&SE@ SW: 6th Workshop for Young Scientists in Computer Science & Software Engineering**. Kryvyi Rih, Ukraine, 2023. p. 193–209.

TALAVER, V.; VAKALIUK, T. A. Telemetry to solve dynamic analysis of a distributed system. **Journal of Edge Computing**, v. 3, n. 1, p. 87–109, 2024.

THÖNES, J. Microservices. **IEEE software**, IEEE, v. 32, n. 1, p. 116–116, 2015.

TIAN, X.; YING, S.; LI, T.; YUAN, M.; WANG, R.; ZHAO, Y.; SHANG, J. itcrl: Causal-intervention-based trace contrastive representation learning for microservice systems. **IEEE Transactions on Software Engineering**, v. 50, n. 10, p. 2583–2601, 2024.

TIGHILT, R.; ABDELLATIF, M.; MOHA, N.; MILI, H.; BOUSSAIDI, G. E.; PRIVAT, J.; GUéHéNEUC, Y.-G. On the study of microservices antipatterns: a catalog proposal. In: **Proceedings of the European Conference on Pattern Languages of Programs 2020**. New York, NY, USA: Association for Computing Machinery, 2020. (EuroPLoP '20). ISBN 9781450377690. Disponível em: <https://doi.org/10.1145/3424771.3424812>.

TOKA, L.; DOBREFF, G.; HAJA, D.; SZALAY, M. Predicting cloud-native application failures based on monitoring data of cloud infrastructure. In: IEEE. **2021 IFIP/IEEE International Symposium on Integrated Network Management (IM)**. Bordeaux, France, 2021. p. 842–847.

TOWNSHEND, S. **Bad Observability**. 2023. Disponível em: <https://squaredup.com/blog/bad-observability/>. Acesso em: 10-06-2025.

TZANETTIS, I.; ANDRONA, C.-M.; ZAFEIROPOULOS, A.; FOTOPOULOU, E.; PAPAVASSILIOU, S. Data fusion of observability signals for assisting orchestration of distributed applications. **Sensors**, MDPI, v. 22, n. 5, p. 2061, 2022.

URDANETA, G.; PIERRE, G.; STEEN, M. V. Wikipedia workload analysis for decentralized hosting. **Computer Networks**, Elsevier, v. 53, n. 11, p. 1830–1845, 2009.

USMAN, M.; FERLIN, S.; BRUNSTROM, A.; TAHERI, J. A survey on observability of distributed edge & container-based microservices. **IEEE Access**, IEEE, v. 10, p. 86904–86919, 2022.

USMAN, M.; FERLIN, S.; BRUNSTROM, A.; TAHERI, J. Desk: Distributed observability framework for edge-based containerized microservices. In: IEEE. **2023 Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit)**. Gothenburg, Sweden, 2023. p. 617–622.

VILLELA, A. **Observability Mythbusters: Observability Anti-Patterns**. 2022. Disponível em: <https://faun.pub/observability-mythbusters-observability-anti-patterns-2b3062405b54>. Acesso em: 10-06-2025.

VILLELA, A. **OpenTelemetry Collector Anti-Patterns**. 2024. Disponível em: <https://geekingoutpodcast.substack.com/p/opentelemetry-collector-anti-patterns>. Acesso em: 10-06-2025.

VU, D.-D.; TRAN, M.-N.; KIM, Y. Predictive hybrid autoscaling for containerized applications. **IEEE Access**, IEEE, v. 10, p. 109768–109778, 2022.

WANG, H.; WU, Z.; JIANG, H.; HUANG, Y.; WANG, J.; KOPRU, S.; XIE, T. 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)**. Melbourne, Australia: IEEE/ACM, 2021. p. 419–429.

WETTINGER, J.; ANDRIKOPOULOS, V.; LEYMANN, F.; STRAUCH, S. Middleware-oriented deployment automation for cloud applications. **IEEE Transactions on Cloud Computing**, IEEE, v. 6, n. 4, p. 1054–1066, 2016.

WIDERBERG, A.; JOHANSSON, E. **Observability of Cloud Native Systems: : An industrial case study of system comprehension with Prometheus & knowledge transfer**. 2021. Disponível em: <https://www.diva-portal.org/smash/get/diva2:1586397/FULLTEXT02.pdf>. Acesso em: 10-06-2025.

WIMALASURIYA, I. **AWS: Your Ally Against Observability Anti-Patterns | Conf42 Observability 2024**. 2024. Disponível em: <https://www.youtube.com/watch?v=Mn4eqXiZoHI>. Acesso em: 10-06-2025.

WU, L.; TORDSSON, J.; ELMROTH, E.; KAO, O. Microrca: Root cause localization of performance issues in microservices. In: IEEE. **NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium**. Budapest, Hungary, 2020. p. 1–9.

WU, Z.; WANG, J.; QI, Q.; SHU, M.-G.; CHU, R.; LI, J.-B.; JIN, J.; CHEN, D. Flowrca: Enhancing microservice reliability with non-invasive root cause analysis. In: IEEE. **2024 IEEE International Conference on Web Services (ICWS)**. Shenzhen, China, 2024. p. 1251–1258.

XIE, R.; WANG, L.; SONG, C. Towards minimum latency in cloud-native applications via service-characteristic-aware microservice deployment. In: IEEE. **2024 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)**. Rovaniemi, Finland, 2024. p. 35–46.

XIE, Z.; ZHANG, S.; GENG, Y.; ZHANG, Y.; MA, M.; NIE, X.; YAO, Z.; XU, L.; SUN, Y.; LI, W.; PEI, D. Microservice root cause analysis with limited observability through intervention recognition in the latent space. In: **Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining**. New York, NY, USA: Association for Computing Machinery, 2024. (KDD '24), p. 6049–6060. ISBN 9798400704901. Disponível em: <https://doi.org/10.1145/3637528.3671530>.

YANG, J.; GUO, Y.; CHEN, Y.; ZHAO, Y. Tracenet: Operation aware root cause localization of microservice system anomalies. In: IEEE. **2023 IEEE International Conference on Communications Workshops (ICC Workshops)**. Rome, Italy, 2023. p. 758–763.

YANG, T.; SHEN, J.; SU, Y.; REN, X.; YANG, Y.; LYU, M. R. Characterizing and mitigating anti-patterns of alerts in industrial cloud systems. In: IEEE. **2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)**. Baltimore, MD, USA, 2022. p. 393–401.

YOKELSON, D.; TITOV, M.; RAMESH, S.; KILIC, O.; TURILLI, M.; JHA, S.; MALONY, A. Enabling performance observability for heterogeneous hpc workflows with soma. In: **Proceedings of the 53rd International Conference on Parallel Processing**. New York, NY, USA: Association for Computing Machinery, 2024. (ICPP '24), p. 220–230. ISBN 9798400717932. Disponível em: <https://doi.org/10.1145/3673038.3673100>.

YU, G.; CHEN, P.; LI, Y.; CHEN, H.; LI, X.; ZHENG, Z. 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**. New York, NY, USA: Association for Computing Machinery, 2023. (ESEC/FSE 2023), p. 553–565. ISBN 9798400703270. Disponível em: <https://doi.org/10.1145/3611643.3616249>.

YU, G.; CHEN, P.; ZHENG, Z. Microscaler: Cost-effective scaling for microservice applications in the cloud with an online learning approach. **IEEE Transactions on Cloud Computing**, IEEE, v. 10, n. 2, p. 1100–1116, 2020.

ZENG, H.; WANG, T.; LI, A.; WU, Y.; WU, H.; ZHANG, W. Topology-aware self-adaptive resource provisioning for microservices. In: IEEE. **2023 IEEE International Conference on Web Services (ICWS)**. Chicago, IL, USA, 2023. p. 28–35.

ZHANG, Y.; HUA, W.; ZHOU, Z.; SUH, G. E.; DELIMITROU, C. Sinan: Ml-based and qos-aware resource management for cloud microservices. In: **Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems**. New York, NY, USA: Association for Computing Machinery, 2021. (ASPLOS '21), p. 167–181. ISBN 9781450383172. Disponível em: <https://doi.org/10.1145/3445814.3446693>.

ZHOU, X.; PENG, X.; XIE, T.; SUN, J.; JI, C.; LI, W.; DING, D. Fault analysis and debugging of microservice systems: Industrial survey, benchmark system, and empirical study. **IEEE Transactions on Software Engineering**, IEEE, v. 47, n. 2, p. 243–260, 2018.

ZHU, J.; YANG, R.; SUN, X.; WO, T.; HU, C.; PENG, H.; XIAO, J.; ZOMAYA, A. Y.; XU, J. Qos-aware co-scheduling for distributed long-running applications on shared clusters. **IEEE Transactions on Parallel and Distributed Systems**, IEEE, v. 33, n. 12, p. 4818–4834, 2022.

APÊNDICE A – CATÁLOGO DE ANTI-PADRÕES DE OBSERVABILIDADE

A descrição completa de todos os anti-padrões de observabilidade é apresentada a seguir:

- **Nome**: Alertas Excessivos
- **Relevância**: Alta
- **Categoria**: Alertas
- **Contexto**: Alertas excessivos ocorrem quando um sistema de monitoramento gera um volume elevado de notificações, dificultando a distinção entre problemas críticos e eventos menores. Esse problema geralmente resulta de regras redundantes, granularidade excessiva ou falta de mecanismos de agregação. Como consequência, os engenheiros podem se tornar insensíveis às notificações, incluindo aquelas que indicam falhas significativas. A ausência de políticas bem definidas para filtragem e priorização de alertas agrava esse anti-padrão.
- **Problema**: O excesso de alertas pode causar fadiga de alerta, levando os engenheiros a desconsiderarem notificações devido ao elevado volume. Isso resulta no aumento do tempo médio de resolução, pois alertas críticos podem se perder no excesso de notificações. Além disso, há desperdício de tempo operacional com alertas de baixa prioridade ou redundantes, o que desvia a atenção de incidentes realmente críticos. Como resultado, falhas significativas podem passar despercebidas, comprometendo a confiabilidade do sistema e impactando a experiência dos usuários finais.
- **Exemplo**: Uma equipe de SRE configura alertas para cada pequena variação no tempo de resposta, resultando em centenas de notificações diárias. Com o tempo, os engenheiros começam a ignorar essas notificações devido ao alto volume. Quando um problema real ocorre, causando uma degradação significativa no desempenho do sistema, ele passa despercebido até que a falha se agrave, afetando milhares de usuários antes que qualquer ação corretiva seja tomada.
- **Solução Recomendada**:
  - Implementar mecanismos de deduplicação, ou seja, eliminar alertas repetidos ou duplicados, mesmo que o evento seja detectado mais de uma vez em um curto período,

para reduzir a redundância de alertas;

- Implementar mecanismos de agregação, que consiste em um processo de agrupar múltiplos alertas relacionados em um único alerta mais geral, para reduzir a redundância de alertas. Isso é útil quando um conjunto de alertas está relacionado a uma mesma causa ou problema. Em vez de gerar vários alertas isolados para cada métrica ou falha, um alerta agregado pode fornecer uma visão consolidada do problema, o que torna a resposta mais eficiente;
- Definir níveis claros de priorização, classificando alertas como informativos, de aviso e críticos, e consequentemente, estabelecendo uma hierarquia que categorize os alertas com base na gravidade e no impacto que eles têm no sistema ou no serviço monitorado. Isso ajuda as equipes a se concentrarem nas notificações mais importantes e a responder de maneira mais eficaz a incidentes. Como exemplo, imagine que você tenha um sistema de monitoramento que verifica a disponibilidade de servidores e bancos de dados. Se a utilização de CPU de um servidor atingir 85%, o alerta seria classificado como “Aviso”. Caso o servidor fique offline, seria classificado como “Crítico”. Por fim, se uma tarefa de rotina for executada com sucesso, o alerta seria “Informativo”. Essa classificação ajuda a equipe a entender rapidamente a urgência de cada situação e a priorizar suas ações;
- Utilizar técnicas de alerta dinâmico, que se baseiam em comportamentos adaptáveis, ajustados às condições reais e variáveis do sistema ao longo do tempo, em vez de valores fixos ou predefinidos. Isso torna os alertas mais flexíveis e relevantes, respondendo de forma precisa às flutuações do sistema e evitando a geração de alertas desnecessários (falsos positivos). Esses alertas podem incorporar a definição de *baseline* (referência de comportamento esperado) e a detecção de anomalias para identificar desvios significativos;
- A equipe de monitoramento deve revisar regularmente as regras de alerta para garantir que estão alinhadas com as necessidades atuais do sistema e da equipe. Isso envolve remover ou ajustar alertas que já não fazem sentido devido a mudanças no sistema ou na infraestrutura; e
- Implementar limitação de taxa para evitar notificações excessivas em curtos períodos de tempo. A limitação de taxa tem como objetivo controlar a frequência de notificações geradas por alertas em um curto intervalo de tempo, evitando que múltiplos

alertas sobre o mesmo problema sejam enviados para a equipe de operações. Como exemplo, suponha que um sistema de microsserviços esteja gerando alertas a cada segundo devido a falhas temporárias de rede. Para evitar que a equipe seja inundada com notificações, a limitação de taxa pode ser configurada para enviar apenas uma notificação a cada 30 segundos, mesmo que o problema continue ocorrendo nesse intervalo. Isso garante que a equipe não receba um volume excessivo de alertas repetitivos e possa focar melhor na resolução de problemas importantes.

- **Consequências**:
  - **Positivas**:
    - **Redução da fadiga de alerta**: A implementação dessas soluções resulta em uma diminuição significativa da sobrecarga de notificações, permitindo que os engenheiros se concentrem nos alertas que realmente necessitam de atenção e ação;
    - **Maior eficiência operacional**: Com um número reduzido de alertas irrelevantes, as equipes conseguem otimizar o tempo, priorizando problemas mais críticos e realizando ações corretivas de forma mais rápida e eficaz; e
    - **Maior confiabilidade do sistema de monitoramento**: Ao melhorar a qualidade e a relevância das notificações, o sistema de monitoramento se torna mais confiável, garantindo que as alertas importantes sejam detectados e tratados de forma adequada, aumentando a confiabilidade do sistema como um todo.
  - **Negativas**:
    - **Investimento inicial significativo**: A configuração de mecanismos de deduplicação, agregação e limitação de taxa de alertas pode demandar um esforço inicial considerável, incluindo a adaptação de ferramentas e a definição de novas regras de alerta;
    - **Necessidade de ferramentas avançadas de monitoramento**: Implementar essas estratégias pode exigir o uso de ferramentas mais sofisticadas, com recursos de correlação inteligente de eventos e análise dinâmica, o que pode representar um desafio em termos de custo e integração; e
    - **Necessidade de ajustes contínuos**: O sistema de alertas precisará de manutenção regular e ajustes contínuos para garantir que a configuração permaneça eficaz, ajustando-se a mudanças no sistema e nos padrões de comportamento, o

que pode gerar um trabalho contínuo de monitoramento e revisão.

- **Nome**: Dependência Exclusiva de *Dashboards* Baseados em Séries Temporais
- **Relevância**: Baixa
- **Categoria**: *Dashboards*
- **Contexto**: Esse anti-padrão ocorre quando os *dashboards* de monitoramento dependem exclusivamente de gráficos de séries temporais para apresentar dados. Embora esses gráficos sejam úteis para visualizar tendências ao longo do tempo, eles não oferecem uma visão completa da saúde do sistema. A ausência de métodos complementares de visualização, como tabelas, *heatmaps*, mapas topológicos ou resumos de alertas, dificulta a extração de *insights* acionáveis e torna o diagnóstico de problemas mais demorado.
- **Problema**: A dependência exclusiva de gráficos de séries temporais pode gerar diversos problemas, incluindo dificuldade na correlação de múltiplas fontes de dados, como *logs*, *traces* e alertas. Além disso, alguns problemas, como degradação gradual do desempenho ou padrões cíclicos anômalos, podem não ser detectados facilmente em gráficos de séries temporais. Esse modelo também torna a análise de incidentes menos eficiente, pois exige que os engenheiros interpretem tendências manualmente, sem suporte de indicadores mais diretos. A identificação de padrões em diferentes componentes do sistema torna-se mais complexa, dificultando a rápida detecção e resolução de problemas.
- **Exemplo**: Um *dashboard* exibe a utilização de CPU ao longo do tempo, mas não fornece informações contextuais, como quais serviços foram afetados, quais *logs* foram gerados durante picos de uso ou se incidentes similares ocorreram no passado. Como resultado, os engenheiros precisam alternar entre múltiplas ferramentas e correlacionar dados manualmente para diagnosticar problemas de desempenho.
- **Solução Recomendada**:
  - Os *dashboards* devem fornecer uma visão holística do sistema, combinando diferentes formas de exibição de dados. Algumas alternativas incluem: **Heatmaps**: São úteis para identificar padrões e variações em grande volume de dados. Por exemplo, um *heatmap* pode mostrar quais servidores estão apresentando alta latência ao longo do dia, facilitando a identificação de padrões sazonais. **Tabelas de métricas**: Em vez de apenas gráficos, exibir métricas-chave organizadas em tabelas permite uma

comparação rápida. Exemplo: uma tabela pode mostrar os top 5 serviços mais lentos no último período de monitoramento. **Mapas topológicos e de dependência**: Permitem visualizar a interação entre serviços. Por exemplo, se um serviço A está com latência alta, o mapa pode mostrar que essa lentidão se propaga para um serviço B que depende dele. **Correlação entre *logs*, métricas e traces**: Um problema identificado em um gráfico de série temporal pode ser correlacionado com *logs* e *traces* para entender sua causa raiz; e

- Um erro comum é criar *dashboards* que apenas exibem dados brutos, sem fornecer *insights* práticos. Para evitar isso, é essencial que os *dashboards* não apenas apresentem métricas, mas também auxiliem na interpretação dos dados e na tomada de decisão. Uma abordagem eficaz é adicionar alertas inteligentes diretamente na interface, destacando automaticamente padrões anômalos. Por exemplo, em vez de apenas exibir gráficos de séries temporais, a ferramenta de monitoramento pode identificar pontos fora do padrão usando aprendizado de máquina ou regras estatísticas, sinalizando eventos atípicos e sugerindo possíveis causas ou ações corretivas. Isso reduz a necessidade de análise manual e torna o diagnóstico mais eficiente.

- **Consequências**:
  - **Positivas**:
    - **Eficiência na resolução de problemas**: A integração de diferentes tipos de visualizações nos *dashboards* reduz o tempo necessário para diagnosticar falhas. Engenheiros podem identificar rapidamente padrões e anomalias, facilitando a resolução de problemas antes que eles se agravem;
    - **Melhor visibilidade e contextualização dos dados**: A combinação de gráficos de séries temporais com outros tipos de visualizações, como heatmaps ou mapas topológicos, oferece uma visão mais completa e contextualizada dos sistemas, permitindo uma análise mais precisa e ações corretivas mais rápidas; e
    - **Redução da carga cognitiva**: Com a organização das informações em *dashboards* mais intuitivos e fáceis de interpretar, os engenheiros conseguem processar dados mais rapidamente, reduzindo a sobrecarga cognitiva e aumentando a produtividade durante a análise de incidentes.
  - **Negativas**:
    - **Esforço adicional para *redesign* de *dashboards***: Implementar essas mudanças

exige um investimento inicial significativo para criar *dashboards* mais eficazes, incluindo o design de novas visualizações e a implementação de uma estrutura de dados mais integrada;

* **Necessidade de treinamento**: Equipes que estão acostumadas com dashboards tradicionais, baseados apenas em gráficos de séries temporais, podem precisar de treinamento ou uma adaptação gradual para entender e utilizar as novas funcionalidades de visualização; e
* **Manutenção contínua e ajustes**: Os *dashboards* precisam de refinamentos regulares para garantir que as visualizações permaneçam úteis e alinhadas às necessidades dos engenheiros. Isso envolve descontinuar informações obsoletas e ajustar os *dashboards* conforme as mudanças nos sistemas e nas métricas monitoradas.

- **Nome**: *Dashboards* em excesso
- **Relevância**: Alta
- **Categoria**: *Dashboards*
- **Contexto**: Este anti-padrão ocorre quando uma organização acumula um número excessivo de *dashboards*, frequentemente sem uma governança adequada ou uma estratégia de consolidação. Isso geralmente acontece quando equipes diferentes criam *dashboards* para propósitos semelhantes, *dashboards* desatualizados não são removidos ou quando não há um plano claro de gestão dos *dashboards*. Como resultado, os usuários têm dificuldade em identificar quais *dashboards* são relevantes e confiáveis, gerando ineficiências tanto no monitoramento quanto na solução de problemas.
- **Problema**: Ter uma grande quantidade de *dashboards* pode causar vários problemas, como a dificuldade em localizar o *dashboard* correto para uma situação específica. Além disso, isso pode levar à inconsistência ou dados conflitantes entre diferentes *dashboards*, o que diminui a confiança no sistema de monitoramento. A sobrecarga de manutenção aumenta à medida que *dashboards* obsoletos ou redundantes continuam em uso. Os engenheiros também podem enfrentar sobrecarga cognitiva ao navegar por vários *dashboards*, o que dificulta a solução de problemas de forma eficaz e ágil.
- **Exemplo**: Uma equipe de SRE precisa investigar um problema de desempenho, mas

encontra múltiplos *dashboards* com nomes semelhantes, cada um exibindo métricas diferentes para o mesmo serviço. Como resultado, os engenheiros perdem tempo precioso tentando determinar qual *dashboard* contém os dados mais relevantes e precisos.

- **Solução Recomendada**:
  - Estabelecer uma governança clara para a criação e manutenção de *dashboards*, definindo diretrizes claras sobre quando e como criar novos *dashboards*, assegurando que cada *dashboard* tenha uma função específica e bem definida;
  - Reduzir a redundância consolidando *dashboards* que oferecem informações semelhantes, garantindo que as visualizações sejam mais integradas e fáceis de acessar;
  - Realizar análises periodicamente nos *dashboards*, removendo ou atualizando aqueles que não estão mais em uso ou que apresentam informações desatualizadas; e
  - Organizar os *dashboards* com *tags* ou categorias que permitam que os usuários encontrem rapidamente as visualizações que correspondem às suas necessidades de análise;
- **Consequências**:
  - **Positivas**:
    - **Maior eficiência no acesso a *dashboards* relevantes**: Ao reduzir o número de *dashboards* e consolidar informações de forma clara, os engenheiros conseguem localizar mais rapidamente as visualizações necessárias, economizando tempo durante a análise e resolução de problemas;
    - **Redução da confusão e sobrecarga cognitiva**: Com *dashboards* mais organizados e menos redundantes, a carga cognitiva dos engenheiros diminui, tornando a solução de incidentes mais ágil e eficiente; e
    - **Menor sobrecarga de manutenção**: A eliminação de *dashboards* redundantes ou desatualizados reduz o trabalho contínuo de manutenção e ajuda a garantir que apenas as visualizações mais relevantes sejam mantidas e aprimoradas ao longo do tempo.
  - **Negativas**:
    - **Esforço inicial para revisão e consolidação**: Realizar uma análise completa dos *dashboards* existentes e consolidá-los em visualizações mais eficientes requer um esforço inicial considerável. Isso pode envolver reconfigurações e ajustes em várias ferramentas e processos;

    * **Resistência à mudança**: Algumas equipes podem se sentir relutantes em abandonar seus *dashboards* personalizados ou modificados, resultando em resistência à implementação de *dashboards* padronizados; e
    * **Necessidade de governança contínua**: Para evitar que o descontrole da quantidade de *dashboards*) se repita, é necessário um processo de governança contínuo. Isso envolve monitorar a criação de novos *dashboards* e garantir que não se acumulem *dashboards* desnecessários ou obsoletos ao longo do tempo.

- **Nome**: Transações Sintéticas Preguiçosas
- **Relevância**: Baixa
- **Categoria**: Geral
- **Contexto**: Esse anti-padrão ocorre quando as transações sintéticas são executadas de forma preguiçosa, ou seja, elas são acionadas sob demanda ou quando condições específicas são atendidas, em vez de serem executadas de forma contínua ou em intervalos regulares. Esse comportamento é comum em sistemas que buscam reduzir a sobrecarga ou em ambientes com recursos limitados. As equipes podem adotar essa prática para economizar recursos, mas sem perceber que isso compromete a cobertura do sistema, deixando partes críticas mal monitoradas.
- **Problema**: Transações sintéticas preguiçosas podem não cobrir todas as áreas críticas do sistema de forma contínua, o que pode resultar em falhas não detectadas. O monitoramento intermitente enfraquece as capacidades de diagnóstico e aumenta o tempo necessário para identificar e resolver incidentes, dificultando a detecção de problemas que ocorrem entre os períodos de verificação.
- **Exemplo**: Em uma plataforma de *e-commerce*, as transações sintéticas são executadas apenas durante picos de tráfego ou quando o sistema detecta degradação de performance. Durante períodos de baixo tráfego, não há monitoramento contínuo. Quando uma falha ocorre fora desses períodos, a equipe demora mais para identificar a causa, pois não há cobertura constante das transações críticas.
- **Solução Recomendada**:
    - Realização periódica das transações sintéticas com intervalos reduzidos, assegurando uma cobertura abrangente e constante de todas as funcionalidades críticas do sistema,

de modo a identificar falhas de forma proativa e evitar lacunas no monitoramento, especialmente em períodos de baixo tráfego ou carga; e

  - Modificar a execução das transações sintéticas conforme o contexto operacional e as condições do sistema, aumentando a frequência de verificação nas áreas mais críticas e sensíveis, enquanto reduz a intensidade de monitoramento em segmentos de menor risco, garantindo uma alocação eficiente dos recursos de monitoramento.

- **Consequências**:
  - **Positivas**:
    - **Cobertura aprimorada**: Proporciona a detecção contínua de problemas em tempo real, em vez de limitar-se a momentos específicos, permitindo uma resposta mais ágil e eficaz a falhas e eventos críticos; e
    - **Diagnósticos rápidos**: Reduz o tempo de resolução de incidentes, melhorando a eficiência na identificação de causas e soluções.
  - **Negativas**:
    - **Sobrecarga de recursos**: Aumento do consumo de recursos, especialmente se a execução das transações sintéticas não for adequadamente balanceada;
    - **Esforço adicional**: Ajustes contínuos no processo de monitoramento podem exigir esforço adicional na configuração e adaptação das ferramentas.

- **Nome**: Imposição de Ferramentas
- **Relevância**: Baixa
- **Categoria**: Geral
- **Contexto**: Esse anti-padrão ocorre quando organizações ou equipes impõem o uso de uma ferramenta específica ou um conjunto de ferramentas em toda a sua infraestrutura ou organização, independentemente das necessidades ou contextos distintos de diferentes equipes ou projetos. Frequentemente, surge do desejo de padronizar processos, simplificar a gestão de ferramentas ou reduzir custos. Embora possa proporcionar consistência, essa prática pode gerar ineficiências quando as ferramentas impostas não se alinham aos requisitos ou fluxos de trabalho específicos de algumas equipes.
- **Problema**: Impor o uso de uma única ferramenta ou plataforma pode resultar em desempenho abaixo do esperado, frustração e ineficiência. As equipes podem ser forçadas a

utilizar ferramentas que não atendem às suas necessidades específicas, o que aumenta a complexidade operacional e reduz a produtividade. Além disso, essa imposição pode gerar resistência por parte dos membros da equipe, que podem preferir ou precisar de ferramentas mais adequadas para realizar suas tarefas de forma mais eficaz, prejudicando o desempenho geral e a colaboração.

- **Exemplo**: Uma empresa impõe o uso de uma única ferramenta de *logging* em todas as equipes de engenharia. Embora a ferramenta funcione bem para monitoramento de infraestrutura, as equipes de desenvolvimento enfrentam dificuldades devido às opções limitadas de personalização para *logging* em nível de aplicação, como dados de transações de usuários, rastreamento de sessões e *logs* de performance do código. Essas limitações dificultam a solução de problemas e a otimização do código. Como resultado, as equipes acabam gastando mais tempo configurando e solucionando problemas com a ferramenta do que se dedicando às suas tarefas principais.
- **Solução Recomendada**:
  - Permitir flexibilidade na escolha das ferramentas, considerando as necessidades específicas e a expertise das equipes, enquanto se padronizam os processos para promover as melhores práticas. Isso garante consistência na abordagem sem impor uma ferramenta específica a todas as equipes, permitindo que cada uma utilize as ferramentas mais adequadas para suas necessidades.
  - Avaliar e adaptar as ferramentas para garantir que atendam aos fluxos de trabalho e requisitos específicos de cada equipe, permitindo flexibilidade para a adoção e experimentação de novas ferramentas quando necessário, sem comprometer a consistência dos processos.
- **Consequências**:
  - **Positivas**:
    - **Maior satisfação e eficiência**: As equipes têm a flexibilidade de escolher ferramentas que atendem melhor às suas necessidades específicas, o que contribui para um aumento na produtividade, motivação e engajamento.
    - **Fomento à inovação**: Ao permitir a experimentação com novas ferramentas e tecnologias, as equipes podem adotar soluções mais adequadas e inovadoras, abordando problemas específicos de forma mais eficaz.
  - **Negativas**:

    * **Falta de consistência**: A utilização de ferramentas diferentes entre as equipes pode dificultar a colaboração e a resolução de problemas, especialmente quando há necessidade de integração entre áreas distintas.
    * **Sobrecarga de gestão**: O suporte de múltiplas ferramentas pode aumentar os custos e o esforço necessário para gerenciar toda a infraestrutura, além de complicar a manutenção e a escalabilidade a longo prazo.

- **Nome**: Negligência na Retenção de Dados
- **Relevância**: Baixa
- **Categoria**: General
- **Contexto**: Este anti-padrão ocorre quando as organizações não gerenciam adequadamente a retenção de dados de observabilidade, como *logs*, métricas e traces. Isso pode acontecer quando as equipes não definem políticas claras sobre o tempo de retenção dos dados ou não monitoram periodicamente o cumprimento dessas políticas. Frequentemente, a negligência ocorre devido à falta de conscientização sobre os impactos da retenção excessiva ou insuficiente de dados ou pela priorização das necessidades imediatas em detrimento da gestão a longo prazo.
- **Problema**: A negligência na retenção de dados pode acarretar diversos problemas, como a perda de informações cruciais, que podem ser excluídas ou se tornar inacessíveis antes de serem necessárias para a solução de problemas ou análise. Também pode resultar em custos excessivos com armazenamento, devido à retenção prolongada de dados desnecessários, o que leva a um uso ineficiente dos recursos. Além disso, a organização pode ter dificuldades em cumprir os requisitos legais ou regulatórios de retenção de dados, caso os dados necessários sejam excluídos ou retidos por períodos inadequados.
- **Exemplo**: Uma organização coleta *logs* de aplicação para monitoramento de desempenho, mas não configura adequadamente a política de retenção desses dados. Os *logs* são retidos pelo período padrão de 30 dias, sem considerar a importância de manter informações críticas por mais tempo. Após esse período, os *logs* mais antigos são automaticamente excluídos. Quando ocorre um incidente de performance grave no sistema, a equipe precisa analisar os *logs* de dias anteriores para entender a causa. No entanto, os *logs* relevantes já foram apagados, pois ultrapassaram o período de retenção. Sem essas informações, a

equipe não consegue identificar a origem do problema e demora mais tempo para resolver o incidente, o que afeta a continuidade dos serviços e a experiência do usuário.

- **Solução Recomendada**:
  - Definir e implementar políticas de retenção de dados baseadas no tipo de dado, seu valor, requisitos regulatórios e a necessidade de armazenamento, assegurando que os dados sejam mantidos pelo tempo necessário e excluídos automaticamente quando não forem mais úteis, garantindo eficiência e conformidade com padrões legais ou da indústria; e
  - Monitorar regularmente o uso do armazenamento para garantir que as políticas de retenção sejam seguidas e ajustar os períodos conforme necessário, equilibrando custos e valor para a organização.
- **Consequências**:
  - **Positivas**:
    - **Redução de custos**: A otimização dos períodos de retenção de dados permite reduzir os custos com armazenamento, eliminando dados desnecessários;
    - **Melhora na resposta a incidentes**: Dados mais relevantes estão disponíveis para solução de problemas, acelerando a resposta a incidentes e facilitando a investigação; e
    - **Conformidade com requisitos legais**: A definição adequada dos períodos de retenção melhora a conformidade com as exigências legais e regulatórias para retenção de dados.
  - **Negativas**:
    - **Esforço inicial**: O processo de definição e implementação das políticas de retenção exige um esforço inicial significativo para garantir que os diferentes tipos de dados sejam adequadamente gerenciados; e
    - **Risco de perda de dados**: Se os períodos de retenção forem definidos de forma muito curta, pode haver a possibilidade de perder dados relevantes, o que pode dificultar a análise e a resolução de incidentes futuros.

- **Nome**: Tratando a Observabilidade Apenas como Ferramenta para Incidentes
- **Relevância**: Baixa

- **Categoria**: Geral
- **Contexto**: Esse anti-padrão ocorre quando equipes tratam a observabilidade apenas como uma ferramenta reativa para incidentes, negligenciando seu valor para monitoramento contínuo e otimização do sistema. Isso dificulta a detecção precoce de falhas e degradações, tornando a investigação pós-falha a principal estratégia para lidar com problemas.
- **Problema**: O uso limitado da observabilidade a momentos de crise aumenta o tempo de detecção de problemas e reduz a eficiência operacional. Sem monitoramento proativo, sinais de alerta passam despercebidos até afetarem os usuários, dificultando a análise da causa raiz e restringindo oportunidades de otimização do desempenho.
- **Exemplo**: Um time de SRE só consulta *dashboards* de observabilidade quando recebe reclamações de usuários sobre falhas no serviço. Como não há monitoramento contínuo, um aumento gradual na taxa de erros e na utilização de recursos passa despercebido, levando a uma interrupção evitável que poderia ter sido mitigada com ações proativas.
- **Solução Recomendada**:
  - Adotar observabilidade contínua, integrando métricas, *logs* e *traces* ao monitoramento diário para identificar falhas de forma proativa.
  - Configurar alertas precisos para detectar anomalias e prevenir incidentes antes que se agravem; e
  - Utilizar dados de observabilidade para otimizar o desempenho e a eficiência do sistema de forma constante.
- **Consequências**:
  - **Positivas**:
    - **Detecção antecipada e maior confiabilidade**: A observabilidade contínua permite a identificação precoce de falhas, reduzindo o impacto de incidentes e aumentando a estabilidade do sistema; e
    - **Melhoria contínua**: *Insights* gerados pela observabilidade otimizam constantemente o desempenho e a eficiência do sistema.
  - **Negativas**:
    - **Mudança cultural**: A equipe precisa adotar uma mentalidade de monitoramento contínuo, o que pode ser desafiador; e
    - **Custo inicial**: A implementação de observabilidade contínua demanda investimentos iniciais significativos e maior consumo de recursos.

- **Nome**: Inconsistência de Ambiente
- **Relevância**: Baixa
- **Categoria**: Geral
- **Contexto**: Esse anti-padrão ocorre quando há discrepâncias entre os ambientes de desenvolvimento, *staging* e produção, o que pode levar a comportamentos imprevisíveis nas aplicações ou sistemas. Essas diferenças, que podem envolver configurações, versões de bibliotecas ou até a infraestrutura, dificultam a replicação fiel do ambiente de produção para testes, resultando em falhas que só são identificadas em produção.
- **Problema**: A inconsistência entre ambientes eleva o risco de falhas na implantação, pois alterações que funcionam em *staging* ou desenvolvimento podem falhar em produção devido a diferenças não detectadas. Além disso, essa inconsistência dificulta a reprodução de problemas, retardando a resolução de incidentes e aumentando o tempo médio de recuperação.
- **Exemplo**: Uma equipe desenvolve e testa uma nova funcionalidade em um ambiente de desenvolvimento local, mas devido a diferenças de configuração, o comportamento da funcionalidade é diferente em *staging* ou produção. Em produção, o recurso falha devido a uma discrepância na versão de uma biblioteca de terceiros, que estava instalada no ambiente de *staging*, mas foi esquecida na produção. O time gasta tempo extra corrigindo o problema, que poderia ter sido identificado antes no pipeline de desenvolvimento.
- **Solução Recomendada**:
  - Definir e versionar todas as configurações (desenvolvimento, staging e produção) e dependências dos ambientes para garantir a replicabilidade e consistência;
  - Assegurar a máxima semelhança entre os ambientes (desenvolvimento, staging e produção), implementando validações automatizadas para detectar e corrigir discrepâncias antes da implantação; e
  - Realizar testes regulares nos ambientes de staging e desenvolvimento para garantir que coincidam o máximo possível com a produção, detectando discrepâncias de forma antecipada.
- **Consequências**:
  - **Positivas**:
    - **Implantações mais confiáveis**: Redução significativa de problemas inesperados

após as liberações em produção;

    * **Solução de problemas mais rápida**: Facilidade para a equipe reproduzir problemas de produção em ambientes consistentes de *staging* ou desenvolvimento;

  – **Negativas**:

    * **Esforço inicial**: A configuração e manutenção de ambientes consistentes pode exigir mais recursos e atenção; e
    * **Complexidade de gerenciar a paridade de infraestrutura**: Em sistemas maiores e mais complexos, com múltiplas dependências, manter a paridade entre ambientes pode ser desafiador.

- **Nome**: Não entender seu ecossistema
- **Relevância**: Alta
- **Categoria**: Geral
- **Contexto**: Este anti-padrão ocorre quando as equipes falham em obter um entendimento profundo de seu próprio ecossistema. Isso inclui a falta de compreensão das relações entre vários serviços, componentes ou dependências, e o comportamento do sistema como um todo. As equipes podem se concentrar demais em componentes individuais, sem considerar como as mudanças afetam o sistema mais amplo, ou podem negligenciar o monitoramento das interações entre os serviços, dificultando a previsão de como uma parte do sistema se comportará em diferentes condições.
- **Problema**: Quando as equipes não entendem seu ecossistema, enfrentam um risco aumentado de falhas no sistema, pois as mudanças em uma parte do sistema podem causar consequências imprevistas em outras. A solução de problemas torna-se ineficiente, com tempos de resolução mais longos devido à falta de visibilidade nas interações do sistema. Além disso, comportamento inconsistente também pode surgir, levando a discrepâncias entre os ambientes de *staging* e produção ou respostas inesperadas sob carga.
- **Exemplo**: Uma equipe de desenvolvimento introduz um novo recurso que depende de um microsserviço. No entanto, eles não levam em consideração a forte dependência do serviço em uma API externa que possui limitação de taxa. Quando o recurso é implantado em produção, causa um gargalo e o sistema desacelera inesperadamente. Como a equipe não entendeu a dependência do serviço no ecossistema, não conseguiu prever a falha.

- **Solução Recomendada**:
  - Manter uma visão sempre atualizada do ecossistema, incluindo serviços, dependências e interconexões, para compreender melhor o comportamento do sistema e antecipar possíveis impactos;
  - Incentivar a comunicação contínua e a colaboração entre as equipes responsáveis por diferentes serviços, garantindo alinhamento sobre o ecossistema, suas interdependências e impactos operacionais;
  - Utilizar ferramentas avançadas de observabilidade para rastrear, em tempo real, as interações entre serviços, identificando padrões de comportamento, mapeando dependências e permitindo que as equipes antecipem e mitiguem potenciais falhas antes que impactem o sistema;
  - Antes de realizar alterações no sistema, conduzir uma análise profunda dos impactos nos diversos componentes e serviços interconectados, avaliando como essas mudanças podem afetar o comportamento global do ecossistema e identificando riscos que possam comprometer a estabilidade e a performance do sistema.
- **Consequências**:
  - **Positivas**:
    - **Solução de problemas e resolução de incidentes mais rápidas**: Compreensão clara de como os serviços interagem e se afetam, acelerando a resolução de problemas;
    - **Melhoria da estabilidade e otimização do sistema**: Compreensão completa do ecossistema, levando a uma redução significativa de falhas imprevistas e aumentando a estabilidade do sistema, bem como melhorias de desempenho.
  - **Negativas**:
    - **Esforço inicial**: A adoção de ferramentas avançadas de observabilidade e a análise detalhada das interações entre serviços podem introduzir uma complexidade adicional e custo no gerenciamento do sistema. Essas ferramentas podem gerar uma quantidade excessiva de dados e alertas, tornando difícil para as equipes filtrarem as informações realmente relevantes e aumentando a carga de trabalho para configurar e manter essas ferramentas; e
    - **Atraso na Implementação de Mudanças**: A análise profunda dos impactos de alterações no sistema, embora crucial para evitar falhas, pode levar a atrasos

significativos na implementação de novas funcionalidades. O processo de avaliar extensivamente os efeitos em todos os componentes e serviços pode resultar em ciclos de desenvolvimento mais longos, retardando a entrega de novas melhorias e inovações no sistema;

- **Nome**: Acúmulo de Dados
- **Relevância**: Baixa
- **Categoria**: Geral
- **Contexto**: Este anti-padrão ocorre quando diferentes equipes dentro de uma organização mantêm suas próprias plataformas ou conjuntos de dados de observabilidade sem compartilhá-los com outras equipes. Isso frequentemente acontece devido à falta de confiança, medo de falhas ou simples resistência à mudança. Esse comportamento cria barreiras de comunicação e colaboração entre as equipes, dificultando a resolução de problemas e a melhoria contínua do sistema.
- **Problema**: Quando os dados de observabilidade não são compartilhados, isso leva à falta de transparência, impedindo que as equipes tenham uma visão clara do sistema como um todo, o que resulta em diagnósticos imprecisos e tempos de resolução de incidentes mais longos. A ausência de compartilhamento de dados também torna a resolução de problemas ineficaz, pois as equipes não têm acesso às informações necessárias para resolver questões em sistemas distribuídos ou complexos. Isso pode fomentar desconfiança entre as equipes, prejudicando as relações e a colaboração. Além disso, sem o fluxo de informações, a capacidade da organização de identificar rapidamente áreas para melhorias é prejudicada, comprometendo, por fim, o desempenho do sistema.
- **Exemplo**: Uma equipe de engenharia mantém sua própria solução de *logs* e métricas e não compartilha essas informações com outras equipes, como operações ou segurança. Como resultado, quando ocorre um problema na produção, ninguém tem acesso aos dados completos necessários para investigar a questão, e a solução é retardada enquanto as equipes procuram por informações isoladas em suas próprias plataformas.
- **Solução Recomendada**:
  - Incentivar a transparência e colaboração entre as equipes, garantindo que todos tenham acesso aos dados relevantes de observabilidade, através de uma cultura de

compartilhamento, promovendo, assim, um ambiente mais colaborativo e aberto; e
  - Usar uma plataforma centralizada de observabilidade onde todos os dados possam ser facilmente acessados e analisados por qualquer equipe, de maneira segura e eficiente.
- **Consequências**:
  - **Positivas**:
    * **Melhoria na colaboração**: Equipes que compartilham dados têm melhor comunicação e colaboração, levando a um desempenho mais eficaz; e
    * **Resolução de incidentes mais rápida**: Com o acesso compartilhado aos dados, as equipes conseguem resolver problemas de maneira mais rápida e precisa;
  - **Negativas**:
    * **Resistência cultural**: Mudar uma cultura acostumada ao isolamento de dados pode ser desafiador e demorado; e
    * **Garantir segurança e conformidade**: Garantir que dados sensíveis, como informações de clientes, sejam tratados adequadamente durante o processo de compartilhamento.

- **Nome**: Solução Única para Todos
- **Relevância**: Baixa
- **Categoria**: Geral
- **Contexto**: Esse anti-padrão ocorre quando as organizações tentam implementar uma solução, ferramenta ou abordagem única em toda a estrutura, sem levar em conta as necessidades específicas de diferentes equipes, serviços ou casos de uso. Essa abordagem geralmente surge devido a um impulso por padronização ou redução de custos, mas muitas vezes falha em atender aos requisitos distintos de cada área. Embora a intenção seja simplificar processos ou reduzir a complexidade, ela pode resultar em ineficiências, frustração e impacto negativo no desempenho, pois soluções inadequadas são aplicadas de forma generalizada.
- **Problema**: Quando uma solução única é aplicada de maneira universal, ela compromete a flexibilidade, pois diferentes equipes ou sistemas podem ter necessidades específicas que não são atendidas, gerando soluções improvisadas ou ineficiências. Isso pode resultar em desempenho insatisfatório, já que ferramentas ou práticas eficazes para uma equipe podem

não ser adequadas para outras, levando ao desperdício de recursos. Além disso, as equipes podem resistir à adoção de ferramentas ou abordagens que não se alinham aos seus fluxos de trabalho, o que diminui a colaboração e a eficiência organizacional.

- **Exemplo**: Uma empresa obriga o uso de uma única plataforma de agregação de *logs* para todas as equipes, sem considerar as necessidades específicas de engenharia, ciência de dados e suporte. Enquanto a plataforma funciona bem para algumas, outras enfrentam limitações. A equipe de engenharia precisa de consultas avançadas para depuração, mas a ferramenta não oferece esses recursos. A equipe de suporte encontra dificuldades para analisar e navegar pelos *logs* devido à complexidade da plataforma. Como resultado, ambas as equipes perdem tempo tentando adaptar a ferramenta, o que compromete a eficiência.
- **Solução Recomendada**:
  - Permitir que as equipes escolham ferramentas e práticas que atendam às suas necessidades específicas, ao mesmo tempo em que se mantêm padrões e integração, e escolher soluções adaptáveis e expansíveis para diferentes casos de uso, garantindo que a organização se beneficie tanto da padronização quanto da personalização; e
  - Fomentar a comunicação entre as equipes para compartilhar conhecimentos e alinhar as melhores práticas, respeitando as necessidades únicas de cada equipe.
- **Consequências**:
  - **Positivas**:
    - **Maior satisfação e produtividade das equipes**: As equipes podem usar ferramentas e processos que atendem melhor às suas necessidades, otimizando as tarefas e fluxos de trabalho específicos; e
    - **Melhor adoção de ferramentas e práticas**: As equipes têm mais probabilidade de se envolver com soluções adaptadas ao seu contexto.
  - **Negativas**:
    - **Potencial fragmentação entre as equipes**: Se não houver uma coordenação adequada, pode resultar em dificuldades de integração e inconsistências no sistema; e
    - **Aumento da complexidade**: A gestão de diversas ferramentas ou abordagens exige maior coordenação e governança, o que pode aumentar a complexidade operacional.

- **Nome**: Falta de Diretrizes de Instrumentação para Novos Serviços
- **Relevância**: Média
- **Categoria**: Geral
- **Contexto**: Este anti-padrão ocorre quando as organizações não estabelecem diretrizes claras e consistentes para a instrumentação de novos serviços com as práticas essenciais de observabilidade. Isso pode ocorrer devido à ausência de um processo formal de integração de observabilidade ou à adoção de abordagens variadas por diferentes equipes, resultando em lacunas no monitoramento e em dificuldades para manter a visibilidade do sistema como um todo.
- **Problema**: Quando não há diretrizes claras de instrumentação, isso leva a uma observabilidade inconsistente, pois alguns serviços podem estar bem instrumentados enquanto outros carecem de visibilidade suficiente, dificultando o monitoramento do sistema como um todo. Isso pode atrasar a detecção de problemas, pois problemas em novos serviços podem passar despercebidos por mais tempo, aumentando o tempo de resolução e o impacto no usuário. A depuração se torna mais difícil quando os serviços não são consistentemente instrumentados, devido à falta de dados para entender o comportamento do sistema.
- **Exemplo**: Um novo microsserviço é implantado sem o rastreamento padrão de *logs* e métricas, pois não foram fornecidas diretrizes para a observabilidade. Quando ocorre um problema em produção, a equipe de engenharia luta para reunir os dados relevantes para a depuração, já que os *logs* são escassos e as métricas importantes não foram capturadas. Em contraste, outros serviços no sistema estão bem instrumentados e podem ser monitorados facilmente, mas a falta de consistência entre os serviços torna a solução de problemas e a análise de causa raiz mais complexas.
- **Solução Recomendada**:
  - Estabelecer padrões para toda a empresa sobre como os serviços devem ser instrumentados, incluindo formatos de *logs*, definições de métricas e padrões de traces;
  - Tornar a observabilidade uma parte integral do ciclo de vida de desenvolvimento dos serviços. Garantir que novos serviços sejam instrumentados antes de serem implantados;
  - Incentivar as equipes a compartilharem estratégias de instrumentação e melhores práticas, garantindo que todos os serviços sejam rastreados de maneira padronizada.

- **Consequências**:
  - **Positivas**:
    - **Melhoria da visibilidade do sistema**: Garante que todos os serviços sejam consistentemente instrumentados com *logs*, métricas e traces;
    - **Detecção e resolução de problemas mais rápidas**: Os serviços são devidamente monitorados desde o início, facilitando a identificação e a correção de problemas; e
    - **Depuração simplificada**: As mesmas práticas de instrumentação serão usadas em todos os serviços, permitindo uma identificação mais rápida da causa raiz.
  - **Negativas**:
    - **Esforço inicial**: Estabelecer diretrizes claras e engajar as equipes na adoção de práticas consistentes, o que pode demandar treinamento e um esforço coordenado entre diferentes equipes; e

      Sobrecarga para garantir a conformidade: Monitorar e garantir que as diretrizes sejam seguidas de forma consistente, especialmente em organizações grandes ou em rápido crescimento, pode gerar sobrecarga operacional.

- **Nome**: Uso Excessivo de Ferramentas
- **Relevância**: Alta
- **Categoria**: Geral
- **Contexto**: Este anti-padrão ocorre quando uma organização ou equipe adota um número excessivo de ferramentas para resolver problemas semelhantes ou sobrepostos sem avaliar completamente se as ferramentas são necessárias ou complementares. Isso geralmente resulta do desejo de testar cada nova tecnologia ou da falta de coordenação entre equipes, levando a redundâncias e ineficiência. Embora seja tentador integrar uma nova ferramenta para cada problema, usar ferramentas em excesso pode sobrecarregar as equipes, aumentar a complexidade e dificultar a manutenção e o gerenciamento dos sistemas.
- **Problema**: Quando muitas ferramentas são usadas, o *overhead* operacional aumenta, pois gerenciar um grande número de ferramentas adiciona tarefas administrativas, como configuração, monitoramento e manutenção, desviando as equipes do foco em resolver problemas reais. Problemas de integração podem surgir quando as ferramentas não funcionam

bem juntas ou exigem integrações complexas, resultando em observabilidade fragmentada e dificuldade para correlacionar dados de diferentes fontes. A curva de aprendizado também se torna mais acentuada, já que as equipes precisam gastar tempo aprendendo e gerenciando várias ferramentas, levando a ineficiências e falta de especialização em qualquer ferramenta específica. Além disso, o uso de ferramentas sobrepostas pode resultar em custos mais altos, desperdiçando investimentos em licenças de software e recursos.

- **Exemplo**: Uma empresa usa ferramentas separadas para monitoramento de infraestrutura, *logs*, alertas e *traces*. Essas ferramentas não se integram bem, e cada equipe é responsável por um conjunto diferente de ferramentas. Como resultado, quando ocorre um problema de desempenho, a equipe de operações tem que verificar uma ferramenta para monitoramento de infraestrutura, outra para *logs* de aplicativo e outra para dados de *traces*. Essa abordagem fragmentada leva a tempos de resolução mais longos, aumento do *overhead* na gestão de cada ferramenta e custos mais altos com assinaturas e manutenção.
- **Solução Recomendada**:
  - Avaliar as ferramentas em uso e consolidá-las onde possível. Usar ferramentas multifuncionais que atendem a várias necessidades (ex.: uma plataforma que combine *logs*, métricas e *traces* em uma interface unificada).
  - Estabelecer um padrão na empresa ou equipe para as ferramentas a serem usadas, garantindo que todos utilizem um conjunto consistente de ferramentas para tarefas semelhantes.
  - Antes de adotar uma nova ferramenta, avaliar cuidadosamente se ela oferece um valor único ou se sobrepõe a soluções existentes, considerando a escalabilidade e o suporte a longo prazo.
- **Consequências**:
  - **Positivas**:
    - **Redução da complexidade**: Minimizar o número de ferramentas em uso facilita a gestão de sistemas e fluxos de trabalho;
    - **Redução de custos**: Menos ferramentas implicam em menores gastos com assinaturas, licenças e recursos específicos; e
    - **Maior eficiência**: Ao focar em menos ferramentas, as equipes podem dominá-las melhor e utilizá-las de forma mais eficaz, além de facilitar a correlação de dados.

  - **Negativas**:
    * **Investimento inicial**: O tempo necessário para avaliar as ferramentas, consolidar sistemas e migrar para um número reduzido de ferramentas, o que pode demandar algum tempo de inatividade ou recursos extras.
    * **Resistência à mudança**: Equipes acostumadas a ferramentas ou fluxos de trabalho estabelecidos podem resistir à adoção de novas abordagens, especialmente se já estiverem profundamente integradas em práticas existentes.

- **Nome**: Negligenciar a Manutenção Regular
- **Relevância**: Média
- **Categoria**: Geral
- **Contexto**: Este anti-padrão ocorre quando equipes ou organizações falham em realizar a manutenção regular de seus sistemas de observabilidade, como atualização de ferramentas, limpeza de dados antigos ou revisão de configurações. Isso pode acontecer quando as equipes priorizam problemas imediatos em vez da saúde a longo prazo do sistema, levando à degradação gradual. A negligência da manutenção pode ser resultado de falta de tempo, responsabilidades pouco claras ou uma mentalidade reativa em relação à manutenção do sistema.
- **Problema**: Quando a manutenção regular é negligenciada, a confiabilidade do sistema diminui, pois softwares, configurações ou infraestruturas desatualizadas podem causar falhas inesperadas, degradação de desempenho ou vulnerabilidades de segurança. O tempo de solução de problemas aumenta, pois eles podem passar despercebidos ou se tornar mais difíceis de diagnosticar devido a ferramentas e sistemas desatualizados. Além disso, os recursos são desperdiçados com dados desnecessários, ferramentas ou configurações não utilizadas, levando a um uso ineficiente de armazenamento e recursos computacionais. Com o tempo, os sistemas podem se tornar fragmentados, com dados, configurações ou versões de ferramentas inconsistentes entre as equipes. A negligência na manutenção também deixa os sistemas vulneráveis a riscos de segurança, podendo resultar em violações de dados ou problemas de desempenho.
- **Exemplo**: Uma equipe continua utilizando uma plataforma de *logs* desatualizada, com suporte limitado, e não tem tempo para atualizar para uma solução mais moderna. À

medida que novos recursos são adicionados ao sistema, a antiga ferramenta de *logs* se torna menos eficiente, levando a processamentos lentos e *logs* imprecisos. Quando ocorre um incidente, a equipe encontra dificuldades para extrair dados significativos dos *logs*, pois a ferramenta não foi atualizada para suportar novos formatos de *logs* ou integrar-se com outros sistemas de monitoramento.

- **Solução Recomendada**:
  - Atribua membros ou funções dedicadas à manutenção da infraestrutura de observabilidade, garantindo responsabilidade e acompanhamento contínuo.
  - Audite regularmente as ferramentas e configurações de observabilidade para identificar áreas de melhoria ou modernização.
  - Certifique-se de que o sistema de observabilidade possa evoluir com a organização, incluindo o planejamento para futuras atualizações de ferramentas e expansão de capacidade.
- **Consequências**:
  - **Positivas**:
    - **Maior estabilidade do sistema**: Através de atualizações regulares e detecção proativa de problemas, resultando em menos falhas no sistema e resolução mais rápida de incidentes;
    - **Menor risco operacional**: Menor chance de vulnerabilidades de segurança ou sistemas desatualizados serem explorados; e
    - **Melhor gestão de recursos**: Eliminando dados ou ferramentas redundantes, garantindo que os recursos sejam utilizados de forma mais eficiente.
  - **Negativas**:
    - **Investimento de tempo**: É necessário para agendar e executar tarefas de manutenção, especialmente em ambientes dinâmicos com diversas prioridades concorrentes; e
    - **Resistência à mudança**: Existe uma resistência a mudança por equipes habituadas a ferramentas ou sistemas antigos, o que demanda uma gestão de mudanças eficaz para garantir a adoção das novas práticas.

- **Nome**: Observabilidade Somente para Produção

- **Relevância**: Média
- **Categoria**: Geral
- **Contexto**: Esse anti-padrão ocorre quando as organizações concentram seus esforços de observabilidade exclusivamente nos ambientes de produção, negligenciando ambientes não-produção, como *staging*, testes ou desenvolvimento. Embora a observabilidade seja crucial na produção para detectar e resolver problemas em tempo real, limitar a observabilidade à produção pode resultar em problemas críticos não identificados antes de chegarem a produção. Esse comportamento geralmente é causado por limitações de recursos, falta de conscientização ou a crença equivocada de que os problemas são menos prováveis de ocorrer em ambientes não-produção.
- **Problema**: Quando a observabilidade se limita à produção, aumenta-se o risco de problemas chegarem à produção, pois falhas que poderiam ser detectadas mais cedo no desenvolvimento ou *staging* permanecem indetectadas, escalando para a produção, onde são mais difíceis e caras de corrigir. A depuração se torna ineficiente, já que problemas descobertos em produção não têm o histórico de dados de ambientes de teste ou *staging*, tornando o processo mais demorado e complexo. Sem observabilidade consistente em ambientes não-produção, as equipes perdem a oportunidade de otimizar serviços antes do lançamento, resultando em uma qualidade de lançamento inferior. Isso também gera frustração para desenvolvedores e testadores, que ficam sem visibilidade sobre o desempenho e o comportamento do sistema em seus ambientes.
- **Exemplo**: Uma equipe de desenvolvimento implanta uma nova funcionalidade em um ambiente de *staging* sem um monitoramento ou *logs* adequados. Quando a funcionalidade chega à produção, surgem problemas de desempenho, mas a equipe tem dificuldades em identificar a causa raiz, pois não havia visibilidade suficiente no ambiente de *staging*. Como resultado, a equipe precisa investigar o problema em produção, onde é mais difícil reproduzir e o impacto é maior. Enquanto isso, o restante do sistema funciona bem, mas a falta de observabilidade nos ambientes não-produção causa um atraso significativo na resolução do problema.
- **Solução Recomendada**:
  - Implemente a observabilidade em ambientes não-produção para garantir que os problemas possam ser detectados e resolvidos antes de chegarem à produção, assim como as mesmas ferramentas em todos os ambientes, a afim de padronizar a observa-

bilidade entre os diferentes ambientes e garantir que as equipes possam monitorar e diagnosticar problemas de forma consistente.

  - Incorpore testes automatizados e monitoramento no pipeline CI/CD, permitindo a identificação e resolução rápida de problemas nos ambientes não produtivos antes de chegarem à produção.
  - Assegure que os ambientes de *staging* ou teste sejam dimensionados e instrumentados corretamente para simular as cargas de produção, permitindo a detecção de problemas que possam não ser visíveis sob cargas de teste menores.

- **Consequências**:
  - **Positivas**:
    - **Detecção precoce de falhas em ambientes não-produção**: Ampliar a observabilidade para ambientes como desenvolvimento e *staging* permite identificar problemas antes que cheguem à produção, melhorando a qualidade das versões lançadas e acelerando a resolução de falhas; e
    - **Testes mais eficazes**: Ambientes de *staging* bem instrumentados, replicando as condições de produção, ajudam a detectar problemas de desempenho e falhas que não seriam visíveis em testes mais simples, tornando os testes mais representativos e eficientes.
  - **Negativas**:
    - **Maior consumo de recursos**: A inclusão de ferramentas em todos os ambientes pode impactar o desempenho do sistema e aumentar a demanda por armazenamento, especialmente em ambientes de teste e desenvolvimento; e
    - **Configuração mais complexa**: A padronização de ferramentas e práticas entre diversos ambientes exige uma coordenação cuidadosa e torna a configuração mais desafiadora, demandando maior esforço para garantir consistência.

- **Nome**: Falta de Informações Contextuais
- **Relevância**: Alta
- **Categoria**: Geral
- **Contexto**: Esse anti-padrão ocorre quando os dados capturados por ferramentas de observabilidade, como *logs*, métricas ou traces, não possuem informações contextuais suficientes

para ajudar as equipes a compreenderem a situação de forma completa. Sem informações relevantes, torna-se difícil interpretar corretamente os dados, levando à confusão, resolução mais lenta de problemas e perda de oportunidades para ações proativas. Esse problema pode ocorrer quando *logs* ou métricas são pouco detalhados ou quando relações importantes e metadados (com o *IDs* de requisição, sessões de usuário ou dependências de serviço) não são capturados.

- **Problema**: A ausência de informações contextuais dificulta a identificação da causa e do impacto de problemas, tornando a análise mais lenta e propensa a erros. *Logs* e métricas sem detalhes suficientes, como *IDs* de *trace* ou dados de sessão, tornam o diagnóstico mais trabalhoso e podem levar a investigações fragmentadas, onde os times precisam correlacionar manualmente múltiplas fontes de dados. Isso reduz a eficiência na solução de problemas e impede ações preventivas, forçando uma abordagem reativa ao invés de proativa. Além disso, alertas sem contexto adequado podem resultar em falsos positivos ou na perda de incidentes críticos, contribuindo para fadiga de alertas e respostas mais lentas.
- **Exemplo**: Um serviço começa a apresentar uma taxa elevada de erros, mas os *logs* não incluem o ID da requisição ou as dependências do serviço envolvidas no erro. A equipe percebe os picos de erro nos *logs*, mas não consegue identificar quais usuários ou transações foram afetados nem se o problema está relacionado a outros componentes do sistema. Como resultado, a equipe perde tempo tentando correlacionar os erros com outros dados do sistema e reproduzir a falha, tornando a investigação mais lenta e prolongando a resolução do incidente.
- **Solução Recomendada**:
  - Incluir metadados relevantes e capturar o contexto de ponta a ponta, registrando IDs de requisição, sessões de usuário, mensagens de erro, versões dos serviços, dependências externas, interações entre serviços, latências e propagação de erros. Isso facilita a correlação de eventos e oferece uma visão mais clara do fluxo de requisições; e
  - Criar regras de alerta que não apenas notifiquem sobre problemas, mas também incluam detalhes como serviço afetado, usuários impactados e possíveis causas raízes.
- **Consequências**:
  - **Positivas**:

    * **Resolução mais rápida de incidentes**: Com informações contextuais adequadas, as equipes podem compreender o escopo e a causa raiz dos problemas mais rapidamente; e
    * **Menos fadiga de alertas**: Alertas enriquecidos com contexto reduzem notificações irrelevantes, evitando sobrecarga cognitiva e melhorando a eficiência na resposta.
  - **Negativas**:
    * **Maior volume de dados**: A coleta de mais contexto pode aumentar significativamente o volume de dados, exigindo mais armazenamento e processamento; e
    * **Complexidade na implementação**: Garantir a captura consistente de contexto em todos os componentes pode demandar mudanças significativas nos sistemas e fluxos de trabalho.

- **Nome**: Falha no Monitoramento de *Logs* de Erro
- **Relevância**: Baixa
- **Categoria**: *Logs*
- **Contexto**: Este anti-padrão ocorre quando as equipes ou organizações negligenciam o monitoramento de *logs* de erro ou falham na configuração adequada de alertas para erros no sistema. *Logs* de erro contêm informações cruciais sobre falhas do sistema, exceções e comportamentos inesperados. No entanto, se não forem monitorados ou analisados ativamente, problemas potenciais podem passar despercebidos, resultando em tempo de inatividade, impacto para o usuário e dificuldades na resolução de incidentes. Isso pode acontecer quando as equipes assumem que os erros serão detectados automaticamente ou quando há uma dependência excessiva de outros sinais de monitoramento (como métricas ou *traces*) que não capturam todos os modos de falha.
- **Problema**: Quando os *logs* de erro não são monitorados de forma ativa, incidentes críticos podem ser negligenciados, resultando em falhas não detectadas que afetam a estabilidade do sistema, reduzem o desempenho e prejudicam a experiência do usuário. Sem monitoramento adequado, problemas que deveriam ser identificados por meio dos *logs* de erro permanecem ocultos por períodos prolongados, elevando os tempos de

recuperação. Os *logs* de erro são fontes essenciais para a análise da causa raiz; sem monitoramento contínuo, identificar a origem dos problemas se torna mais complexo e demorado, estendendo o tempo de resolução e impactando negativamente o tempo médio de recuperação.

- **Exemplo**: Em uma plataforma de *e-commerce*, os *logs* de erro indicam falhas intermitentes no sistema de pagamento, mas não há monitoramento adequado configurado para alertar a equipe sobre essas falhas. Durante semanas, os *logs* registram essas falhas, mas ninguém é notificado a tempo. Como resultado, os clientes começam a relatar problemas ao tentar realizar compras, mas a equipe de suporte só detecta a gravidade do problema depois que a quantidade de reclamações aumenta significativamente. Nesse período, o impacto sobre os usuários cresce, e o tempo de recuperação do sistema, que poderia ter sido imediato, se arrasta por várias horas até que a equipe consiga corrigir o problema.
- **Solução Recomendada**:
  - Garantir que todos os *logs* de erro críticos sejam monitorados ativamente e que alertas sejam configurados para notificar imediatamente a equipe quando as taxas de erro ultrapassarem os limites predefinidos.
  - Estabelecer processos para revisar regularmente os *logs* de erro mais comuns ou críticos, permitindo que as equipes identifiquem padrões ou problemas recorrentes e os abordem proativamente.
  - Usar ferramentas que visualizem tendências de erros ao longo do tempo, permitindo que as equipes detectem problemas emergentes e evitem falhas generalizadas.
- **Consequências**:
  - **Positiva**:
    - **Detecção mais rápida de incidentes**: Com o monitoramento adequado dos *logs* de erro, os problemas podem ser detectados mais cedo, permitindo que as equipes respondam rapidamente e reduzam o tempo de inatividade.
  - **Negativas**:
    - **Sobrecarga de dados**: *Logs* de erro podem se acumular rapidamente, especialmente em sistemas complexos, o que pode resultar em fadiga de alertas se não forem gerenciados e filtrados adequadamente.
    - **Tempo de configuração inicial**: Configurar sistemas eficazes de monitoramento e alerta para *logs* de erro pode levar tempo, especialmente em sistemas grandes

ou legados, onde os formatos e o armazenamento dos *logs* podem precisar ser padronizados.

- **Nome**: Uso de *Logs* de Acesso à API para Solução de Problemas
- **Relevância**: Baixa
- **Categoria**: *Logs*
- **Contexto**: Este anti-padrão ocorre quando equipes dependem dos *logs* de acesso de API (como *logs* de requisições HTTP) para solucionar problemas, sem usar ferramentas diagnósticas mais profundas, como *logs* de aplicação, *logs* de erro ou traces. Embora os *logs* de acesso de API sejam úteis para rastrear o tráfego de requisições e fornecer informações básicas (como códigos de status de resposta, carimbos de data/hora e caminhos das requisições), muitas vezes eles carecem do contexto necessário para diagnosticar problemas complexos, como *bugs* na aplicação, falhas em serviços ou gargalos de desempenho.
- **Problema**: Quando as equipes usam apenas os *logs* de acesso de API para solução de problemas, isso resulta em um diagnóstico limitado, já que esses *logs* geralmente fornecem apenas dados superficiais e não revelam informações detalhadas sobre a lógica interna da aplicação, erros ou gargalos de desempenho. Identificar as causas raiz torna-se difícil, pois os *logs* de acesso de API podem indicar problemas, como altas taxas de erro ou respostas lentas, mas não oferecem os detalhes necessários, como rastreamentos de pilha, exceções ou o contexto necessário para entender a causa do problema. Além disso, as dependências de serviços frequentemente são negligenciadas, pois os *logs* de acesso de API não capturam como os serviços interagem entre si.
- **Exemplo**: Um serviço web tem apresentado altas taxas de erro 5xx, visíveis nos *logs* de acesso de API, mas os *logs* não fornecem mais detalhes sobre a causa raiz dos erros. A equipe usa esses *logs* para concluir que o problema provavelmente está relacionado ao processamento de *back-end* da API, mas não examina os *logs* da aplicação nem implementa os traces. O problema, na verdade, estava em um serviço *downstream* que falhou, mas os *logs* de acesso de API não ofereceram contexto suficiente para identificar isso. A equipe passou um tempo considerável resolvendo o problema na parte errada do sistema.
- **Solução Recomendada**:
  - Use uma combinação de *logs* de acesso de API, *logs* de aplicação e *logs* de erro para

solução de problemas. Os *logs* de aplicação fornecem informações mais profundas sobre o comportamento interno dos serviços, como exceções, mensagens de erro e lógica de processamento.

  - Utilize os *traces* para obter uma visão completa de ponta a ponta de como as requisições fluem pelo sistema, incluindo interações com outros serviços, bancos de dados e APIs externas. Isso proporcionará um contexto mais abrangente para diagnosticar problemas.

- **Consequências**:
  - **Positiva**:
    * Ao usar uma variedade de *logs* e traces, as equipes podem identificar rapidamente a causa raiz dos problemas, reduzindo o tempo gasto na solução.
  - **Negativas**:
    * **Aumento da complexidade**: Implementar uma estratégia de *logs* e *traces* mais abrangente pode exigir configuração adicional, além de ferramentas mais avançadas e integração entre diferentes serviços; e
    * **Potencial para ruído**: Coletar e correlacionar dados de várias fontes pode aumentar o volume de informações, o que pode sobrecarregar as equipes se não for bem gerido, levando a fadiga de alertas ou sobrecarga de dados.

- **Nome**: Uso inadequado de Níveis de *Log*
- **Relevância**: Baixa
- **Categoria**: *Logs*
- **Contexto**: Este anti-padrão ocorre quando os *logs* são gerados com níveis de *log* inadequados, como usar "INFO" para erros críticos ou "DEBUG" para o fluxo geral da aplicação. Níveis de *log* como ERROR, WARN, INFO e DEBUG são usados para indicar a gravidade ou importância dos eventos. No entanto, equipes frequentemente usam níveis de *log* de forma errada, configurando-os muito baixos (por exemplo, logando erros críticos como INFO) ou muito altos (por exemplo, logando informações rotineiras como ERROR). Essa classificação inadequada leva a confusão, volume excessivo de *logs* e torna difícil identificar rapidamente a gravidade dos problemas, especialmente ao revisar *logs* durante incidentes ou no processo de resolução de problemas.

- **Problema**: O uso inadequado dos níveis de *log* gera uma série de problemas. Erros críticos podem ser escondidos pelo excesso de informações se eventos importantes, como falhas no sistema ou quedas de aplicação, forem registrados com níveis inadequados (por exemplo, INFO ou DEBUG). Isso pode levar a equipe a desconsiderar ou não priorizar corretamente esses eventos. Além disso, o uso incorreto dos níveis de *log* resulta em perdas de oportunidades para monitoramento proativo, já que eventos significativos podem não acionar alertas, dificultando a identificação e resolução de problemas antes que se agravem. Por fim, a resolução de incidentes se torna mais demorada e ineficiente, pois as equipes precisam filtrar grandes volumes de dados irrelevantes para encontrar a causa raiz.
- **Exemplo**: Um serviço apresenta uma falha inesperada devido a um tempo de espera excedido na conexão com o banco de dados. O erro é registrado como INFO e um aviso sobre a falha ao tentar reconectar ao banco de dados é registrado como DEBUG. Quando a equipe revisa os *logs*, o que foi definido como INFO é considerado *logs* operacionais regulares, e o DEBUG sobre o problema do banco de dados é ignorado, pois não foi corretamente sinalizado como erro. Como resultado, a equipe só percebe a gravidade do problema muito depois, quando o serviço se torna instável.
- **Solução Recomendada**:
  - Garanta que os *logs* sejam registrados com o nível adequado, alinhado à gravidade e importância do evento;
  - Implemente políticas e diretrizes claras para padronizar os níveis de *log* entre as equipes e serviços, promovendo consistência na classificação dos *logs*; e
  - Realize auditorias regulares nos *logs* para assegurar o uso correto dos níveis e ajuste-os conforme necessário, otimizando a utilidade e a ação dos *logs*.
- **Consequências**:
  - **Positivas**:
    - **Detecção mais rápida de incidentes**: O uso adequado dos níveis de *log* garante que problemas críticos sejam rapidamente identificados, permitindo a detecção e resolução de incidentes antes que se agravem; e
    - **Monitoramento e alerta aprimorados**: *logs* bem estruturados facilitam a integração com sistemas de alerta, possibilitando notificações precisas e oportunas conforme a gravidade do evento..
  - **Negativas**:

    * **Esforço inicial de configuração**: Padronizar os níveis de *log* em toda a organização pode exigir esforço para auditar os *logs* existentes e refatorá-los onde necessário, o que pode envolver mudanças no código e coordenação; e
    * **Mudança de hábitos das equipes**: Equipes acostumadas ao uso inconsistente ou inadequado dos níveis de *log* podem encontrar resistência ao adotar práticas mais estruturadas.

- **Nome**: Falta de *Logs* Estruturados
- **Relevância**: Média
- **Categoria**: Logs
- **Contexto**: Este anti-padrão ocorre quando os *logs* são escritos em formato não estruturado ou de texto livre, o que torna difícil buscar, analisar e correlacionar entradas de *log* entre sistemas. *Logs* não estruturados geralmente consistem em mensagens de texto simples com contexto ou metadados mínimos, dificultando seu processamento programático. Isso acontece quando as equipes falham em implementar *logs* estruturados, nos quais os *logs* são gerados em um formato consistente e legível por máquinas (por exemplo, JSON ou pares chave-valor). A falta de estrutura aumenta o tempo e o esforço necessários para extrair informações significativas dos *logs*, especialmente em sistemas complexos ou durante a resposta a incidentes.
- **Problema**: Quando os *logs* não são estruturados, eles se tornam difíceis de pesquisar e analisar de maneira eficiente, pois os *logs* não estruturados frequentemente consistem em entradas de texto livre que dificultam a identificação de eventos ou padrões específicos. Isso exige uma análise manual ou o uso de expressões regulares, o que é demorado e sujeito a erros. Sem *logs* estruturados, ferramentas de automação (por exemplo, sistemas de alerta, plataformas de agregação de logs) não conseguem analisar ou processar facilmente os dados de log, perdendo oportunidades para monitoramento proativo ou remediação automatizada. A falta de estrutura também leva a práticas de registro inconsistentes, onde cada mensagem de *log* segue um formato diferente ou contém informações inconsistentes, dificultando a correlação de *logs* entre serviços ou sistemas. Isso aumenta a sobrecarga operacional, pois as equipes precisam gastar mais tempo examinando manualmente os *logs* para encontrar informações relevantes, criando ineficiências tanto nas operações

rotineiras quanto nas investigações de incidentes. Finalmente, a ausência de estrutura prejudica a resposta a incidentes, pois se torna mais difícil identificar rapidamente as informações-chave, atrasando o processo de investigação e recuperação.

- **Exemplo**: Uma plataforma de e-commerce que depende de microsserviços para processamento de pedidos registra *logs* de erros de forma não estruturada. Em um incidente crítico, onde os pagamentos falham em uma grande quantidade de pedidos, os *logs* são registrados como simples mensagens de erro, como "Falha na transação de pagamento". Não há informações adicionais, como o ID do pedido, a transação de pagamento, ou o nome do serviço que falhou. Durante a investigação, a equipe tem dificuldade para identificar quais pedidos foram afetados e em qual ponto do processo ocorreu a falha. Isso atrasa a correção do problema e a comunicação com os clientes afetados, além de dificultar a análise da causa raiz.
- **Solução Recomendada**:
  - Adote um formato consistente e legível por máquina (por exemplo, JSON ou pares chave-valor) que inclua metadados relevantes, como carimbos de data/hora, níveis de log, nomes de serviços, IDs de solicitação, códigos de erro e informações contextuais; e
  - Implemente plataformas centralizadas de agregação de *logs* (por exemplo, ELK Stack, Splunk) que possam manipular e consultar *logs* estruturados de forma eficiente. Essas ferramentas permitem identificar rapidamente tendências, anomalias ou erros em todo o sistema.
- **Consequências**:
  - **Positivas**:
    - **Análise de *logs* mais fácil**: *Logs* estruturados são mais fáceis de pesquisar, filtrar e analisar, permitindo que as equipes identifiquem rapidamente padrões, problemas e tendências;
    - **Melhor automação**: Ferramentas de automação podem processar *logs* estruturados de forma mais eficaz, possibilitando recursos como alertas automatizados e detecção de anomalias; e
    - **Melhor monitoramento e relatórios**: *Logs* estruturados se integram melhor com ferramentas de monitoramento e relatórios, ajudando a gerar métricas que impulsionam melhorias no sistema.

  - **Negativas**:
    * **Esforço inicial**: Implementar registros estruturados em todos os sistemas exige um esforço significativo de refatoração dos *logs* existentes, padronização de formatos e garantia de que as ferramentas adequadas estão em funcionamento;
    * **Aumento nos requisitos de armazenamento**: *Logs* estruturados podem aumentar o volume de dados de *log* devido aos metadados adicionais e informações mais detalhadas, resultando em custos mais elevados de armazenamento e processamento; e
    * **Curva de aprendizado**: As equipes podem precisar de treinamento ou documentação para entender a nova abordagem de registros estruturados e ferramentas, especialmente se estiverem acostumadas a trabalhar com *logs* não estruturados.

- **Nome**: *Logs* Duplicados
- **Relevância**: Média
- **Categoria**: *Logs*
- **Contexto**: Este anti-padrão ocorre quando a mesma mensagem de *log* é gerada várias vezes em diferentes partes de um sistema, muitas vezes devido a configurações inadequadas ou práticas de registro incorretas. *Logs* duplicados podem ocorrer em várias situações, como quando o mesmo evento é registrado por múltiplos serviços, componentes ou camadas dentro de uma aplicação.
- **Problema**: Quando *logs* duplicados são gerados, isso resulta em um aumento no volume de *logs*, consumindo mais espaço de armazenamento e exigindo mais recursos para processar, armazenar e analisar. A presença de *logs* duplicados causa fadiga de *logs*, já que as equipes ficam sobrecarregadas com o grande volume de entradas, dificultando a identificação de informações relevantes e aumentando a probabilidade de eventos importantes passarem despercebidos. Além disso, os *logs* duplicados distorcem a análise e as métricas, levando a relatórios e tendências imprecisas ou à falha na identificação de problemas reais, já que a mesma mensagem de *log* é contada várias vezes. Durante a investigação de incidentes, os *logs* duplicados atrasam a resolução de problemas e a análise da causa raiz, pois as equipes precisam filtrar entradas idênticas em vez de focar nos *logs* únicos e acionáveis.
- **Exemplo**: Um serviço web está configurado para registrar erros quando um usuário tenta

realizar uma ação que falha devido a permissões insuficientes. No entanto, o erro é registrado tanto pela camada da API (que recebe a solicitação) quanto pela camada de autorização (que gerencia o controle de acesso). Como resultado, cada ação que falha gera duas entradas idênticas de *log* com a mesma mensagem de erro e contexto, causando duplicação desnecessária dos dados de log. Quando a equipe revisa os *logs*, ela é forçada a ignorar as entradas redundantes e se concentrar apenas nos *logs* únicos, o que resulta em uma perda de tempo valioso durante a resposta ao incidente.

- **Solução Recomendada**:
  - Garanta que os *logs* sejam gerados apenas por um componente ou serviço por evento. Por exemplo, ou a camada da API ou a camada de autorização deve registrar o erro, mas não ambas. Designe qual camada será responsável por registrar cada tipo de evento para evitar redundância; e
  - Use plataformas de agregação de *logs* (ex.: ELK Stack, Splunk) que suportem deduplicação automática ou filtragem de entradas de *logs* duplicadas com base em atributos comuns, como *timestamps*, IDs de solicitação ou códigos de erro.
- **Consequências**:
  - **Positivas**:
    - **Redução do volume de *logs***: Eliminar *logs* duplicados reduz os dados desnecessários, diminuindo os requisitos de armazenamento e agilizando o processamento e análise de *logs*; e
    - **Melhor resposta a incidentes**: As equipes podem gastar menos tempo filtrando *logs* redundantes e mais tempo investigando e resolvendo problemas, levando a uma recuperação mais rápida e menos tempo de inatividade.
  - **Negativa**:
    - **Esforço inicial de limpeza**: Identificar e remover *logs* duplicados em um sistema grande pode exigir tempo e esforço significativos, especialmente se a duplicação for generalizada.

- **Nome**: Somente *Logs* Locais
- **Relevância**: Média
- **Categoria**: *Logs*

- **Contexto**: Este anti-padrão ocorre quando os *logs* são gerados e armazenados apenas localmente dentro de serviços individuais, em vez de serem agregados ou centralizados em um único local para todo o sistema. Os serviços podem registrar eventos em arquivos de disco locais ou sistemas de *logging* locais que não são acessíveis por outras partes da infraestrutura. Essa prática geralmente é resultado de um design inadequado ou da falta de políticas de *logging* centralizadas, onde as equipes acreditam que os *logs* locais são suficientes para solução de problemas e monitoramento. No entanto, ela causa desafios significativos em sistemas de grande escala, onde a visibilidade e rastreabilidade entre serviços são essenciais.
- **Problema**: Quando os *logs* são armazenados apenas localmente, a visibilidade é limitada, pois os *logs* ficam isolados em máquinas ou serviços individuais, dificultando a obtenção de uma visão holística do sistema. A falta de centralização impede a correlação de eventos entre múltiplos serviços ou a identificação de problemas em nível de sistema. Durante incidentes, a solução de problemas se torna difícil, pois as equipes precisam acessar manualmente os *logs* de diferentes locais, o que é demorado e propenso a erros. Além disso, sem armazenamento centralizado, os dados de *log* não podem ser facilmente ingeridos em sistemas de monitoramento ou alerta, o que significa que eventos críticos, anomalias ou tendências podem passar despercebidos. Finalmente, armazenar *logs* localmente cria desafios de conformidade e auditoria, pois o *logging* centralizado é frequentemente exigido para retenção de dados, segurança e controle de acesso.
- **Exemplo**: Uma aplicação baseada em microsserviços registra todos os eventos localmente em arquivos de texto em cada máquina de serviço. Quando ocorre um erro no sistema visível ao usuário, a equipe não consegue correlacionar facilmente os *logs* de múltiplos serviços (por exemplo, o serviço de front-end, o serviço de autenticação e o banco de dados) para identificar a causa. Os desenvolvedores precisam acessar manualmente as máquinas de cada serviço para recuperar os *logs*, resultando em atrasos significativos e frustração. Além disso, alguns *logs* são perdidos porque o disco local se enche, fazendo com que o serviço substitua *logs* mais antigos.
- **Solução Recomendada**:
  - Utilize sistemas de *logging* centralizados (como ELK Stack e Splunk) para agregar *logs* de todos os serviços em um único repositório. Isso facilita a análise, a busca e a correlação dos *logs* em todo o sistema; e

  - Em sistemas distribuídos, garanta que os *logs* sejam coletados em todos os serviços e possam ser correlacionados usando identificadores únicos, como IDs de requisição ou IDs de transação, para rastrear eventos enquanto fluem pelo sistema.
- **Consequências**:
  - **Positivas**:
    * **Melhor visibilidade**: A centralização de *logs* permite que as equipes obtenham uma visão completa do sistema, facilitando a identificação de problemas e o entendimento de como diferentes serviços interagem; e
    * **Melhor monitoramento e alerta**: *Logs* centralizados podem ser ingeridos em ferramentas de monitoramento e configurados com sistemas de alerta automatizados para detectar proativamente problemas antes que se agravem.
  - **Negativas**:
    * **Configuração inicial**: Configurar a infraestrutura de *logging* centralizado pode exigir configuração significativa, especialmente em sistemas grandes com muitos serviços.
    * **Custo**: Sistemas de *logging* centralizado, especialmente serviços baseados em nuvem, podem gerar custos adicionais para armazenamento, processamento e retenção de *logs*, especialmente se os *logs* forem gerados em grande volume.
    * **Mudança cultural**: A transição de *logs* locais para centralizados pode exigir uma mudança nas práticas e ferramentas das equipes, e as equipes podem precisar de treinamento sobre como gerenciar e interpretar *logs* centralizados.
    *
    *

- **Nome**: *Logs* Excessivos
- **Relevância**: Alta
- **Categoria**: *Logs*
- **Contexto**: Este anti-padrão ocorre quando os sistemas registram informações demais, muitas vezes em um nível muito granular. Embora o *logging* seja essencial para entender o comportamento do sistema, *logs* excessivos podem inundar o sistema com dados desnecessários, ofuscando informações valiosas. Isso pode acontecer quando as equipes registram

cada ação, detalhe ou evento menor. Também ocorre quando os *logs* são configurados para capturar cada informação durante as fases de desenvolvimento ou depuração e nunca são ajustados adequadamente para os ambientes de produção.

- **Problema**: Quando *logs* excessivos são gerados, os custos de armazenamento aumentam devido à necessidade de armazenar grandes quantidades de dados de *log* que podem não fornecer valor significativo. Isso também pode degradar o desempenho, especialmente se os *logs* forem gravados de maneira síncrona. O *log* excessivo cria ruído nos dados, tornando mais difícil extrair *insights* valiosos, o que retarda a resposta a incidentes. Além disso, o grande volume de *logs* dificulta a análise e a identificação de problemas, sobrecarregando os sistemas de gerenciamento de *logs*.
- **Exemplo**: Em uma aplicação de *e-commerce*, todas as interações do usuário, incluindo navegação pelas páginas, visualização de produtos e ações de adição ao carrinho, são registradas como *logs* de DEBUG. Isso resulta em uma quantidade massiva de *logs* sendo gerada constantemente, com informações excessivas que não são úteis para monitoramento ou diagnóstico em produção. Quando ocorre uma falha, a equipe de operações encontra dificuldades em localizar os *logs* realmente relevantes, pois está inundada por centenas de milhares de entradas relacionadas a atividades de usuários, como cliques e navegação, dificultando a análise do incidente.
- **Solução Recomendada**:
  - Configure as bibliotecas de *logs* para usar níveis de *log* apropriados. Para ambientes de produção, garanta que o nível de *log* padrão seja configurado para INFO ou superior, e evite registrar *logs* detalhados de nível DEBUG, a menos que seja necessário para solucionar problemas específicos;
  - Priorize o registro de eventos que forneçam *insights* acionáveis, como mensagens de erro, avisos, gargalos de desempenho e ações chave de usuários. Evite registrar detalhes excessivos que não contribuam para a compreensão do comportamento do sistema;
  - Em sistemas de alto volume, considere usar técnicas de amostragem de *logs* para limitar a quantidade de dados registrados, enquanto ainda captura informações suficientes para detectar anomalias ou tendências; e
  - Use plataformas de agregação de *logs* (como ELK Stack e Splunk) para centralizar *logs* e facilitar o filtro, análise e monitoramento dos dados de *logs*. Essas platafor-

mas permitem um melhor gerenciamento de *logs* excessivos através de filtragem, indexação e configuração de alertas para eventos relevantes.

- **Consequências**:
  - **Positivas**:
    - **Redução do volume de logs**: Limitar *logs* excessivos reduz custos de armazenamento e processamento, além de aliviar a carga sobre os sistemas de gerenciamento de *logs*;
    - **Resposta mais rápida a incidentes**: Com um conjunto de *logs* mais focado, as equipes podem identificar informações relevantes mais rapidamente e resolver incidentes, melhorando a confiabilidade do sistema e reduzindo o tempo de inatividade; e
    - **Maior eficiência operacional**: As equipes gastam menos tempo filtrando *logs* excessivos e mais tempo lidando com problemas importantes, o que leva a uma maior eficiência operacional.
  - **Negativas**:
    - **Esforço inicial para limpeza**: Reduzir *logs* excessivos pode exigir um esforço significativo para identificar quais *logs* são desnecessários e modificar as configurações de *logging*, especialmente em sistemas grandes com muitos serviços; e
    - **Risco de sub-*logging***: Reduzir a verbosidade dos *logs* demais pode resultar em informações cruciais sendo perdidas, dificultando a solução de problemas. É importante garantir que os *logs* mantidos forneçam contexto suficiente para diagnosticar problemas de maneira eficaz.

- **Nome**: Uso Inadequado de *Logs* para Coleta de Métricas
- **Relevância**: Baixa
- **Categoria**: *Logs*
- **Contexto**: Este anti-padrão ocorre quando *logs* são usados de maneira inadequada ou ineficiente para coleta de métricas. *Logs* são frequentemente vistos como uma fonte primária de dados para observabilidade, mas usá-los como meio único ou principal para coleta de métricas pode levar a ineficiências significativas. As equipes podem registrar

informações detalhadas, mas não conseguem extrair as métricas apropriadas desses *logs*, ou podem depender de *logs* para métricas que poderiam ser mais eficazmente rastreadas como métricas dedicadas. Essa situação pode surgir devido à falta de compreensão das diferenças entre *logs* e métricas, ou por equipes tentarem usar *logs* como uma solução abrangente para monitoramento, em vez de utilizar a ferramenta certa para o propósito certo.

- **Problema**: Quando os *logs* são usados de maneira inadequada para a coleta de métricas, as equipes enfrentam ineficiência devido à verbosidade dos *logs*, que podem conter informações desnecessárias, dificultando a extração de métricas significativas. *Logs* capturam dados em um ponto específico no tempo e muitas vezes não são projetados para análise em tempo real, ao contrário dos sistemas dedicados de métricas que podem fornecer insights contínuos. Além disso, enquanto os *logs* podem fornecer dados detalhados, frequentemente faltam as métricas específicas necessárias para monitorar a saúde e o desempenho do sistema, resultando em sobrecarga de armazenamento e processamento.
- **Exemplo**: Uma equipe registra todas as requisições HTTP com dados detalhados sobre os cabeçalhos de requisição e resposta, conteúdo do corpo e carimbos de tempo. Embora esses *logs* forneçam informações ricas, a equipe também tenta extrair métricas de desempenho, como tempos de resposta ou taxas de erro a partir deles do corpo. Isso leva a um processamento lento, armazenamento excessivo de *logs* e dificuldade para correlacionar os dados necessários para uma análise significativa.
- **Solução Recomendada**:
  - Utilize ferramentas e sistemas especializados em métricas para coletar dados sobre o desempenho do sistema (por exemplo, latência e throughput).
- **Consequências**:
  - **Positivas**:
    - **Melhoria no monitoramento de desempenho do sistema**: Usando as ferramentas certas para o trabalho certo (métricas para desempenho, *logs* para eventos detalhados), as equipes podem obter melhor visibilidade sobre a saúde e o desempenho do sistema;
    - **Redução da sobrecarga**: Com as métricas coletadas separadas dos *logs*, o sistema pode processar e armazenar dados relevantes de maneira mais eficiente; e

    - **Facilita a correlação de dados**: *Logs* estruturados e métricas dedicadas permitem correlação mais fácil e precisa entre sistemas.
  - **Negativas**:
    - **Aumento da complexidade**: Implementar sistemas separados para *logs* e métricas exige mais configuração e manutenção; e
    - **Migração de dados**: Se *logs* foram usados inadequadamente para coleta de métricas no passado, migrar para um sistema mais adequado pode exigir a reformulação dos dados de *log* existentes.

- **Nome**: Intervalos de Amostragem Inadequados
- **Relevância**: Alta
- **Categoria**: Geral
- **Contexto**: Este anti-padrão ocorre quando os intervalos de amostragem usados para coletar os dados são mal escolhidos. Se o intervalo for muito curto, o sistema pode gerar dados demais, levando ao consumo excessivo de recursos, custos de armazenamento e, potencialmente, sobrecarregando a infraestrutura de monitoramento. Por outro lado, se o intervalo de amostragem for muito longo, os dados coletados podem não capturar flutuações importantes ou padrões no desempenho do sistema, deixando de identificar problemas críticos que ocorrem entre os períodos de amostragem. Esse problema geralmente ocorre quando as equipes definem os intervalos de amostragem com base na conveniência ou limitações técnicas, em vez de considerar as necessidades específicas do sistema.
- **Problema**: Quando os intervalos de amostragem são mal escolhidos, vários problemas podem ocorrer. A coleta excessiva de dados pode acontecer quando a frequência de amostragem é muito alta, resultando em um volume de dados que sobrecarrega os sistemas de armazenamento e as capacidades de processamento. Por outro lado, intervalos de amostragem longos podem resultar em dados imprecisos, pois picos, quedas ou anomalias críticas no desempenho podem ser perdidos. Isso também leva a desperdício de recursos, pois coletar mais dados do que o necessário consome recursos computacionais, largura de banda e espaço de armazenamento desnecessários. Resultados de monitoramento inconsistentes podem surgir quando os intervalos de amostragem variam entre diferentes partes do sistema, levando a dados fragmentados ou enganosos.

- **Exemplo**: Uma equipe define um intervalo de amostragem de 10 minutos para monitorar o uso de disco de um servidor. No entanto, durante esse período, o disco atinge 100% de uso em 2 minutos, causando lentidão no sistema. Como o intervalo de amostragem é longo demais, o pico de uso de disco não é detectado, e o problema só é identificado quando a performance do sistema já está comprometida.
- **Solução Recomendada**:
  - Ajuste a frequência de coleta com base na importância dos dados para a detecção de problemas críticos no sistema;
  - Utilize técnicas que ajustem automaticamente os intervalos de amostragem, dependendo da carga do sistema ou da identificação de anomalias, garantindo dados mais precisos quando necessário; e
  - Avalie continuamente a eficácia dos intervalos escolhidos, analisando se os dados coletados são relevantes e se estão proporcionando *insights* úteis para a operação do sistema.
- **Consequências**:
  - **Positivas**:
    - **Coleta de dados mais eficiente**: Ao escolher o intervalo de amostragem certo, o sistema pode coletar dados suficientes sem sobrecarregar os recursos, e ajudar as equipes a identificar rapidamente problemas de desempenho e anomalias; e
    - **Economia de custos**: Reduzir a coleta de dados desnecessários ajuda a minimizar os custos de armazenamento e processamento.
  - **Negativas**:
    - **Ajuste dos intervalos**: Determinar o intervalo de amostragem ideal pode demandar experimentação contínua e ajustes, pois depende das necessidades específicas de monitoramento e das características do sistema; e
    - **Balanceamento da granularidade**: Encontrar o equilíbrio entre dados detalhados, que fornecem *insights* profundos, e métricas de alto nível, que asseguram a visibilidade da saúde geral do sistema, pode ser um desafio contínuo.

- **Nome**: Métricas Excessivas
- **Relevância**: Média

- **Categoria**: Métricas
- **Contexto**: Esse anti-padrão ocorre quando as equipes monitoram um número excessivo de métricas, muitas vezes sem um propósito claro ou priorização. Embora as métricas sejam fundamentais para a observabilidade, coletar métricas demais pode resultar em sobrecarga de dados, dificultando o foco nos indicadores mais importantes de saúde e performance do sistema. As equipes podem cair nessa armadilha ao tentar monitorar todos os pontos de dados possíveis ou por se sentirem obrigadas a coletar métricas de todas as partes do sistema, sem analisar se são realmente úteis, resultando em desperdício de recursos.
- **Problema**: Quando as equipes monitoram métricas excessivas, vários problemas podem surgir. Primeiro, ocorre sobrecarga de informações, pois muitas métricas tornam difícil identificar e focar nas que realmente importam. Isso leva ao desperdício de recursos, já que coletar e armazenar métricas excessivas consome armazenamento e poder de processamento desnecessários. O grande número de métricas também dificulta a priorização do que é importante, complicando a tomada de decisões. Além disso, a resolução de problemas fica mais lenta, pois encontrar a causa raiz de um problema se torna mais desafiador com tantas métricas para filtrar. Finalmente, as métricas excessivas muitas vezes não geram *insights* acionáveis, pois são muito granulares ou irrelevantes para os objetivos atuais do sistema, tornando-se inúteis para decisões ou ações significativas.
- **Exemplo**: Uma equipe monitora centenas de métricas em diferentes serviços, incluindo métricas para cada *endpoint* de API individual, método de requisição e até mesmo consultas específicas ao banco de dados. Embora algumas dessas métricas forneçam informações úteis, outras não agregam valor e geram ruído desnecessário. Como resultado, a solução do problema é atrasada, e o incidente leva mais tempo para ser resolvido.
- **Solução Recomendada**:
  - Identifique e monitore apenas as métricas mais críticas que estão alinhadas com os objetivos do sistema e metas de performance;
  - Antes de coletar uma nova métrica, defina seu propósito e como ela ajudará a equipe a tomar decisões ou melhorar a performance do sistema; e
  - Avalie continuamente quais métricas estão sendo monitoradas e remova ou consolide aquelas que não fornecem *insights* acionáveis.
- **Consequências**:
  - **Positivas**:

    * **Foco aprimorado**: Ao monitorar apenas as métricas mais relevantes, a equipe pode concentrar sua atenção nos indicadores críticos de saúde do sistema;
    * **Redução no consumo de recursos**: Coletar e armazenar menos métricas reduz o custo de recursos do sistema; e
    * **Resolução de problemas mais rápida**: Com um conjunto mais focado de métricas, fica mais fácil e rápido identificar a causa raiz dos problemas.
  - **Negativas**:
    * **Balanceamento entre detalhes e visão geral**: Encontrar o equilíbrio entre monitorar dados detalhados para resolução de problemas e evitar granularidade excessiva pode ser desafiador; e
    * **Lidar com dados legados**: Se o sistema acumulou métricas excessivas ao longo do tempo, pode ser desafiador limpar e reorganizar a coleta de dados.

- **Nome**: Esquecer o Cliente
- **Relevância**: Alta
- **Categoria**: Métricas
- **Contexto**: Esse anti-padrão ocorre quando as equipes focam excessivamente no desempenho interno do sistema ou em métricas de infraestrutura e negligenciam a experiência do cliente. Isso geralmente acontece quando há uma ênfase excessiva em métricas técnicas (como saúde dos servidores e performance do banco de dados) em detrimento de métricas relacionadas ao usuário, como responsividade da aplicação, usabilidade ou satisfação do usuário.
- **Problema**: Quando as equipes não monitoram ou priorizam métricas voltadas para o cliente, elas perdem visibilidade sobre a experiência real dos usuários. Isso pode resultar na ausência de métricas cruciais, como tempos de resposta, taxas de erro e satisfação do usuário, que poderiam indicar problemas de desempenho que impactam diretamente os clientes. Enquanto as métricas internas do sistema podem indicar um funcionamento ideal, a falta de monitoramento da experiência do usuário pode fazer com que as equipes não percebam problemas até que os clientes os reportem. Esse atraso na detecção de problemas pode levar a uma priorização inadequada de correções técnicas que não melhoram a experiência do usuário, resultando em uma queda na satisfação do cliente ao longo do

tempo.

- **Exemplo**: Uma equipe monitora o uso da CPU do servidor, a performance do banco de dados e as latências internas das APIs, mas não acompanha métricas voltadas para o usuário, como tempo de carregamento de páginas ou taxa de sucesso de transações. Embora o sistema interno apresente um bom desempenho, os usuários começam a enfrentar lentidão e falhas intermitentes ao acessar o site. A equipe só percebe o impacto quando recebe várias reclamações de clientes, atrasando a resposta e a resolução do problema.
- **Solução Recomendada**:
  - Certifique-se de acompanhar métricas relacionadas à experiência do cliente, como tempo de carregamento de páginas, tempos de resposta de APIs, taxas de sucesso de transações e taxas de erro.
  - Construa *dashboards* que priorizem métricas voltadas para o usuário, garantindo que a equipe possa identificar rapidamente problemas que impactam diretamente os clientes.
- **Consequências**:
  - **Positiva**:
    - **Experiência do cliente aprimorada**: Ao focar em métricas voltadas para o usuário, as equipes garantem que problemas que afetam os clientes sejam detectados e resolvidos rapidamente.
  - **Negativas**:
    - **Complexidade adicional no monitoramento**: Monitorar tanto métricas internas do sistema quanto métricas voltadas para o usuário pode aumentar a complexidade da configuração de observabilidade; e
    - **Maior demanda por recursos**: O foco em métricas da experiência do usuário pode exigir ferramentas ou infraestrutura adicionais para coletar e processar os dados.

- **Nome**: Cobertura Inadequada de Monitoramento
- **Relevância**: Média
- **Categoria**: Métricas
- **Contexto**: Este anti-padrão ocorre quando componentes ou partes críticas do sistema não

são adequadamente monitorados, deixando lacunas na observabilidade. Essas lacunas podem surgir devido ao escopo limitado de monitoramento, focando apenas em certos serviços ou sistemas, ou assumindo que algumas partes do sistema não necessitam de atenção. A cobertura inadequada de monitoramento diminui a capacidade de detectar e responder a incidentes.

- **Problema**: Quando as equipes falham em monitorar adequadamente todos os componentes críticos de um sistema, vários problemas podem surgir. Podem aparecer pontos cegos na observabilidade do sistema, já que serviços, bancos de dados ou serviços auxiliares, não são monitorados corretamente. Isso leva a uma detecção tardia de problemas, com áreas não monitoradas passando despercebidas até que se transformem em incidentes graves. A resposta a incidentes também fica mais lenta, pois identificar a causa raiz dos problemas sem monitoramento adequado torna-se mais desafiador. Além disso, a cobertura inadequada de monitoramento limita a capacidade de prever falhas do sistema ou problemas de desempenho antes que ocorram.
- **Exemplo**: Uma equipe foca seu monitoramento no *frontend* da aplicação voltado para o usuário e nas APIs principais, mas negligencia o monitoramento dos sistemas de *backend*, como o banco de dados, camada de cache ou microsserviços que dão suporte ao *frontend*. Em um dia, o banco de dados sofre com aumento de latência nas consultas, mas como não é monitorado corretamente, o problema só é notado quando as aplicações voltadas para o cliente começam a apresentar lentidão.
- **Solução Recomendada**:
  - Certifique-se de que todos os componentes críticos, incluindo *frontend*, *backend*, bancos de dados, camadas de cache, filas de mensagens e infraestrutura, estão cobertos pelo monitoramento;
  - Realize auditorias regulares no sistema de monitoramento para identificar áreas que possam estar negligenciadas.
- **Consequências**:
  - **Positivas**:
    - **Detecção mais rápida de problemas**: Com uma cobertura abrangente de monitoramento, as equipes podem identificar e resolver problemas muito mais rápido; e
    - **Maior confiabilidade do sistema**: Ao ter visibilidade total de todos os compo-

nentes do sistema, as equipes podem garantir que cada parte do sistema esteja funcionando corretamente

- **Negativas**:
  - **Aumento da complexidade de monitoramento**: Expandir a cobertura de monitoramento para todos os componentes do sistema exige configurações de monitoramento mais sofisticadas;
  - **Maior exigência de recursos**: Um monitoramento mais abrangente pode resultar em um consumo maior de recursos; e
  - **Equilibrar a granularidade do monitoramento**: Encontrar o equilíbrio certo entre monitorar sistemas críticos sem sobrecarregar a equipe com dados ou alertas desnecessários pode ser desafiador.

- **Nome**: Mal-Interpretação de Métricas
- **Relevância**: Baixa
- **Categoria**: Métricas
- **Contexto**: Este anti-padrão ocorre quando equipes ou organizações interpretam incorretamente ou utilizam métricas de forma inadequada, levando a decisões e ações equivocadas. Isso pode envolver a compreensão errada do que uma métrica realmente está medindo, como ela deve ser usada ou o que ela indica sobre a saúde ou o desempenho do sistema. As equipes podem se concentrar nas métricas erradas, interpretar seu significado de forma equivocada ou não entender seu contexto, resultando em tomadas de decisão ruins e ações ineficazes.
- **Problema**: Quando as métricas são mal interpretadas, podem surgir diversos desafios significativos para as equipes que tentam manter a saúde do sistema. Interpretações incorretas das métricas podem levar as equipes a tirar conclusões erradas, resultando em uma priorização incorreta dos problemas. Elas podem se concentrar em problemas não críticos enquanto deixam de observar métricas importantes que indicam que componentes do sistema necessitam de atenção. A má interpretação do significado ou da importância de uma métrica também pode criar uma falsa sensação de segurança, fazendo com que as equipes acreditem que o sistema está funcionando bem, quando na realidade existem problemas ocultos. Essa falta de clareza resulta em uma priorização incorreta, com recursos

alocados para problemas menos importantes. Além disso, a incapacidade de interpretar corretamente as métricas pode retardar os esforços de solução de problemas, atrasando a resolução de questões reais e impactando a confiabilidade do sistema.

- **Exemplo**: Uma equipe monitora a métrica "Contagem de Solicitações" para sua API e observa um aumento constante nas solicitações. Eles assumem que tudo está funcionando bem, mas ao analisar mais de perto, percebem que a "Taxa de Erros" também aumentou significativamente. Apenas a "Contagem de Solicitações" os levou a ignorar um problema crescente, que era um aumento nas falhas de solicitação. Como resultado, eles não tomaram providências para resolver o problema a tempo, impactando a experiência do usuário.
- **Solução Recomendada**:
  - Garanta que as métricas sejam interpretadas dentro do contexto do sistema que estão monitorando, combinando-as, quando necessário, para obter uma visão mais completa e precisa do desempenho geral;
  - Certifique-se de que as métricas escolhidas estão alinhadas com os principais indicadores de desempenho que refletem os resultados do negócio.
- **Consequências**:
  - **Positivas**:
    - **Melhor tomada de decisões**: Uma compreensão clara das métricas facilita a identificação de problemas mais rapidamente; e
    - **Melhor desempenho do sistema**: Com a interpretação correta das métricas, as equipes podem focar na otimização das áreas certas do sistema.
  - **Negativas**:
    - **Treinamento e especialização**: Garantir que todos saibam interpretar e utilizar as métricas corretamente pode exigir treinamentos adicionais; e
    - **Definição de métricas**: Pode levar tempo para definir adequadamente métricas significativas que estejam alinhadas com os objetivos do negócio e o comportamento do sistema.

- **Nome**: Apenas Rastrear o Que o Cliente Vê
- **Relevância**: Média
- **Categoria**: Métricas

- **Contexto**: Esse anti-padrão surge quando as equipes focam apenas em rastrear métricas voltadas para o usuário (como tempos de resposta, disponibilidade ou tempos de carregamento de página), negligenciando métricas internas críticas que refletem a saúde e o desempenho do sistema subjacente. Isso ocorre tipicamente porque as equipes priorizam métricas visíveis aos clientes, assumindo que são os únicos indicadores importantes de desempenho.
- **Problema**: Quando as equipes rastreiam apenas métricas voltadas para o cliente, frequentemente ignoram questões internas importantes que podem afetar o desempenho e a estabilidade do sistema. Focar exclusivamente em métricas como desempenho do *frontend*, tempos de resposta ou erros visíveis ao cliente significa que gargalos no *backend*, esgotamento de recursos ou falhas em serviços internos podem passar despercebidos. Esse escopo limitado impede as equipes de identificar problemas, atrasando a resposta até que impactem o usuário final. Além disso, sem visibilidade sobre o funcionamento interno do sistema, a análise de causa raiz se torna desafiadora, pois as equipes podem não conseguir rastrear problemas até os serviços ou componentes específicos que os causam.
- **Exemplo**: Uma equipe foca principalmente em rastrear os tempos de carregamento de seu site e o número de transações bem-sucedidas processadas por suas APIs voltadas ao cliente. No entanto, não monitoram métricas internas, como o desempenho de consultas ao banco de dados, latências de serviços backend ou uso de CPU e memória em servidores críticos. Com o tempo, o banco de dados sofre degradação de desempenho devido a uma alta carga de consultas, mas os usuários não percebem imediatamente a lentidão, pois a equipe não está rastreando os gargalos internos.
- **Solução Recomendada**:
  - Além das métricas voltadas para o cliente, monitorar métricas internas como desempenho de serviços *backend*, tempos de consulta ao banco de dados, uso de memória, carga de CPU e latência de rede; e
  - Configurar alertas automáticos para métricas internas, como consultas lentas ao banco de dados, picos de CPU no servidor ou vazamentos de memória.
- **Consequências**:
  - **Positivas**:
    - **Melhor detecção proativa de problemas**: Ao rastrear tanto métricas internas quanto externas, as equipes podem detectar problemas antes que afetem o cliente,

identificando a causa raiz mais rapidamente; e

    * **Maior confiabilidade**: Ao tratar problemas internos antes que impactem a experiência do usuário, as equipes garantem serviços mais estáveis e confiáveis.

  – **Negativas**:

    * **Maior complexidade no monitoramento**: Monitorar tanto métricas voltadas ao usuário quanto internas requer uma configuração mais sofisticada, bem como ter uma sobrecarga de dados.
    * **Alocação de recursos**: Garantir um monitoramento interno eficiente pode exigir investimentos significativos em infraestrutura, ferramentas especializadas e equipe qualificada para análise e manutenção contínua.

- **Nome**: Ignorar Métricas Derivadas
- **Relevância**: Baixa
- **Categoria**: Métricas
- **Contexto**: Esse anti-padrão ocorre quando as equipes deixam de rastrear ou considerar métricas derivadas que poderiam fornecer *insights* mais profundos sobre o comportamento do sistema. Métricas derivadas são calculadas a partir de métricas brutas e oferecem uma compreensão mais significativa da saúde, desempenho ou padrões de uso do sistema.
- **Problema**: Quando as equipes ignoram métricas derivadas, podem perder *insights* críticos que ajudam a avaliar com precisão a saúde e o desempenho do sistema. Métricas brutas, como latência básica ou contagem de erros, geralmente não fornecem contexto suficiente para entender completamente o comportamento do sistema ou detectar problemas em tempo hábil. Métricas derivadas, que combinam métricas brutas, podem revelar tendências e anomalias que podem passar despercebidas, fornecendo uma visão mais holística do sistema. Além disso, métricas derivadas frequentemente fornecem dados mais claros e acionáveis, ajudando as equipes a tomar decisões informadas. Ao ignorar essas métricas, as equipes correm o risco de perder problemas sistêmicos maiores, como problemas de escalabilidade ou gargalos de desempenho, que podem impactar a confiabilidade e o desempenho do sistema.
- **Exemplo**: Uma equipe monitora o tempo de resposta bruto de um aplicativo web e nota picos ocasionais de latência. No entanto, não rastreiam o percentil 95 da latência, o que

significa que, se o percentil 95 for de 500 ms, 95% das requisições foram respondidas em até 500 ms. Contudo, as outras 5% podem ter demorado mais que isso, indicando que há picos de latência que afetam um número pequeno de usuários. Mais tarde, ao receber reclamações, a equipe descobre que os picos de latência impactavam apenas uma parte específica dos usuários, o que só seria visível com a análise de métricas derivadas, como os percentis.

- **Solução Recomendada**:
  - Além de monitorar métricas brutas, garantir que métricas derivadas (como médias, percentis, taxas e proporções) também sejam acompanhadas, bem como combinar métricas para entender o desempenho do sistema de forma mais detalhada; e
  - Métricas derivadas como taxa de sucesso (requisições bem-sucedidas / total de requisições) ou taxa de erro (erros / total de requisições) fornecem uma melhor compreensão da confiabilidade do sistema.
- **Consequências**:
  - **Positivas**:
    - ***Insights* mais profundos**: O rastreamento de métricas derivadas proporciona uma compreensão mais detalhada do comportamento do sistema; e
    - **Melhor priorização de problemas**: Métricas derivadas ajudam as equipes a priorizar incidentes com base no impacto real do problema, detectando tendências, anomalias ou possíveis problemas precocemente.
  - **Negativas**:
    - **Maior complexidade**: Rastrear e gerenciar métricas derivadas adiciona complexidade aos sistemas de monitoramento, assim como há um risco de sobrecarga de informações; e
    - **Potencial para interpretação equivocada**: Sem o devido entendimento e contexto, métricas derivadas podem ser mal interpretadas ou levar a conclusões erradas.

- **Nome**: Confiar Apenas no Monitoramento de APIs
- **Relevância**: Baixa
- **Categoria**: Métricas

- **Contexto**: Esse anti-padrão ocorre quando equipes ou organizações confiam exclusivamente no monitoramento das APIs do sistema, sem considerar outras áreas cruciais da infraestrutura ou aplicação. Embora o monitoramento das APIs seja essencial, ele fornece apenas visibilidade sobre a disponibilidade e o desempenho dos *endpoints* das APIs, sem cobrir outras áreas críticas, como o uso de recursos do sistema, a comunicação interna entre serviços ou a saúde de outros componentes do ecossistema.
- **Problema**: Quando as equipes dependem apenas do monitoramento das APIs, enfrentam várias limitações para entender a saúde geral do sistema. O monitoramento das APIs oferece principalmente visões sobre o desempenho dos *endpoints*, mas não captura a saúde interna do sistema, como o desempenho do banco de dados, a comunicação entre serviços ou a utilização de recursos. Problemas como alto uso de CPU, vazamentos de memória, esgotamento de espaço em disco ou latência de rede podem impactar o desempenho da API, mas podem passar despercebidos caso apenas a API seja monitorada. Além disso, esse foco limitado dificulta o diagnóstico de problemas complexos que afetam os sistemas *backend* ou dependências, tornando mais difícil identificar e resolver as causas raiz dos gargalos de desempenho. Dependendo apenas do monitoramento das APIs, a detecção e a resposta a incidentes podem ser retardadas, já que problemas com serviços dependentes ou bancos de dados podem passar despercebidos até afetar diretamente a API.
- **Exemplo**: Uma plataforma de *e-commerce* depende exclusivamente do monitoramento da API de pedidos para acompanhar o desempenho. Durante os horários de pico de compras, os clientes experimentam tempos de carregamento lentos e falhas ocasionais ao tentar finalizar as compras. No entanto, o monitoramento está funcionando perfeitamente, com todas as requisições sendo completadas com sucesso e dentro dos tempos de latência normais. A causa raiz é identificada mais tarde como um gargalo no banco de dados, que não foi capturado pelo monitoramento da API sozinho, levando a transações de clientes com atraso.
- **Solução Recomendada**:
  - Implementar monitoramento completo, que inclua o monitoramento das APIs, o monitoramento de recursos do sistema (CPU, memória, disco, rede), o desempenho do banco de dados, a saúde dos serviços internos e o monitoramento da infraestrutura (ex. balanceadores de carga, caches, filas). Isso fornecerá uma visão completa da saúde do sistema.

  - Utilizar *traces* para monitorar como as requisições fluem por todo o sistema. Isso permite visualizar como os microsserviços interagem entre si e identificar onde ocorrem os atrasos ou falhas, mesmo que não impactem diretamente a API.
  - Utilizar os *logs* para obter mais informações sobre falhas. *Logs* de serviços de *backend* ou componentes da infraestrutura podem fornecer informações adicionais sobre as métricas de desempenho da API e ajudar a rastrear problemas entre os sistemas.
- **Consequências**:
  - **Positivas**:
    * **Observabilidade abrangente**: O monitoramento além das APIs permite que as equipes compreendam todo o sistema, desde a infraestrutura e os serviços até as interações e o desempenho do usuário; e
    * **Identificação mais rápida das causas raiz**: Com a observabilidade completa, as equipes podem diagnosticar mais rapidamente os problemas que afetam tanto as APIs quanto os componentes subjacentes, reduzindo o tempo de resolução durante incidentes.
  - **Negativas**:
    * **Complexidade aumentada**: Implementar monitoramento em todas as camadas do sistema pode aumentar a complexidade e exigir mais recursos para configuração, gerenciamento e análise; e
    * **Sobrecarga de dados**: Com a cobertura ampliada de monitoramento, há o risco de sobrecarga de dados. Estratégias adequadas de filtragem, agregação e alerta são necessárias para evitar a fadiga com alertas ou dados desnecessários.

- **Nome**: A Grande Métrica Boba
- **Relevância**: Baixa
- **Categoria**: Métricas
- **Contexto**: Este anti-padrão ocorre quando organizações focam em uma única métrica excessivamente simplista, que não oferece uma visão significativa da saúde ou desempenho do sistema. Um exemplo comum inclui a métrica “Uso de CPU”, sem entender o contexto mais amplo. Essas métricas são fáceis de coletar, mas muitas vezes falham em refletir o

verdadeiro estado do sistema, levando a decisões equivocadas.

- **Problema**: Quando essa métrica é usada, ela pode levar a conclusões enganosas, pois uma única métrica pode não capturar toda a complexidade do desempenho ou da saúde do sistema. Além disso, confiar apenas em uma métrica pode resultar em uma estratégia de monitoramento inadequada, onde informações valiosas de outras métricas são ignoradas, levando a práticas de visibilidade ineficazes. Essa falsa sensação de segurança pode fazer com que as equipes acreditem que o sistema está saudável quando problemas subjacentes persistem, atrasando as respostas durante incidentes.
- **Exemplo**: Uma empresa de *e-commerce* monitora sua infraestrutura e foca exclusivamente no uso total de CPU dos servidores. Embora o uso de CPU seja uma métrica importante, a equipe ignora outros recursos essenciais como memória, disco e rede. Durante um pico de tráfego, o uso da CPU está dentro de níveis aceitáveis, então a equipe assume que tudo está funcionando bem. No entanto, os usuários começam a relatar lentidão no site. A equipe só descobre a causa quando começa a olhar outras métricas, como o uso de memória e a performance do disco. Eles percebem que o sistema estava esgotando a memória e começando a usar swap, o que causava a lentidão. Além disso, a análise da CPU revelou que o gargalo estava em um processo específico que estava consumindo muito mais CPU do que o esperado. A equipe demorou a identificar o problema, pois estava monitorando cegamente apenas uma métrica, o uso total de CPU, sem considerar o panorama completo da infraestrutura.
- **Solução Recomendada**:
  - Use uma combinação de métricas relevantes que forneçam uma compreensão mais completa da saúde do sistema. Por exemplo, ao invés de monitorar apenas a "Contagem de Requisições", monitore também "Latência de Requisições", "Taxas de Erro", "Utilização de Recursos" e "Dependências de Serviços"; e
  - Combine múltiplos sinais em uma única métrica que reflita mais do comportamento do sistema, como combinar tempo de resposta e taxa de erro para criar uma visão mais holística da saúde do serviço.
- **Consequências**:
  - **Positivas**:
    - **Visibilidade aprimorada**: Ao diversificar as métricas, as equipes podem monitorar o sistema de forma mais eficaz, obtendo *insights* sobre vários aspectos do

desempenho e da saúde, e não apenas um único ponto de dados; e

    * **Identificação mais rápida de problemas**: Um conjunto de métricas mais detalhado permite detectar mais rapidamente problemas potenciais em diferentes componentes do sistema, levando a tempos de resposta mais rápidos e resolução mais eficaz de incidentes.

  – **Negativas**:

    * **Complexidade aumentada**: Adicionar mais métricas pode aumentar a complexidade nos sistemas de monitoramento e alerta, dificultando o acompanhamento e gerenciamento dos dados; e
    * **Sobrecarga**: Coletar e analisar uma maior variedade de métricas pode exigir mais recursos, tanto em termos de desempenho do sistema (por exemplo, sobrecarga de coleta) quanto sobrecarga operacional (por exemplo, mais alertas, mais dados para interpretar).

- **Nome**: *Spans* de *Trace* Longos
- **Relevância**: Baixa
- **Categoria**: *Traces*
- **Contexto**: Este anti-padrão ocorre quando os *spans* de *trace* são excessivamente longos, frequentemente capturando mais dados do que o necessário. No rastreamento distribuído, os *spans* são usados para medir e acompanhar a duração de operações, requisições ou interações específicas dentro de um sistema. Quando esses *spans* são muito longos, podem causar vários problemas, como a incapacidade de detectar gargalos e pouca visibilidade sobre o desempenho.
- **Problema**: Quando os *spans* são muito longos, as equipes podem se deparar com um volume excessivo de dados de trace, o que dificulta a obtenção de *insights* valiosos. Esse excesso de dados pode mascarar a identificação de gargalos, já que os *spans* longos abrangem múltiplas operações, mascarando a origem exata dos atrasos. Isso geralmente acontece quando o rastreamento é aplicado a tarefas ou processos de longa duração, que seriam mais eficazmente divididos em *spans* menores e mais fáceis de analisar.
- **Exemplo**: Uma equipe está realizando o *trace* no sistema, que inclui vários serviços de *backend*, como autenticação, armazenamento de dados do usuário e verificação de

email. Em vez de rastrear cada serviço separadamente, todo o fluxo é capturado como um único *span* longo. Isso resulta em um único *trace* que dura vários minutos, dificultando a identificação de qual parte do processo está atrasando o sistema (por exemplo, a operação de gravação no banco de dados ou a chamada à API de email).

- **Solução Recomendada**:
  - Em vez de rastrear os fluxos como *spans* únicos, divida-os em *spans* menores e mais gerenciáveis para cada serviço ou operação individual; e
  - Garanta que o contexto de *trace* seja propagado corretamente entre os serviços, de forma que cada *span* represente uma unidade lógica de trabalho;
- **Consequências**:
  - **Positivas**:
    - **Granularidade aprimorada do *trace***: Ao dividir *spans* longos em unidades menores, as equipes podem obter melhores *insights* sobre áreas específicas do sistema; e
    - **Resolução de problemas de desempenho mais rápida**: *spans* menores e bem definidos tornam mais fácil isolar e identificar a causa exata de atrasos ou gargalos;
  - **Negativas**:
    - **Complexidade aumentada**: *Spans* mais granulares podem aumentar a complexidade na gestão de *traces* e tornar mais difícil manter a consistência em sistemas grandes; e
    - **Potencial para sobrecarga de *traces***: Embora *spans* mais granulares sejam úteis, eles também podem levar a uma quantidade excessiva de dados de trace, especialmente em sistemas de alto *throughput*.

- **Nome**: Sem ID de *Trace* Consistente
- **Relevância**: Média
- **Categoria**: Traces
- **Contexto**: Este anti-padrão ocorre quando um sistema distribuído falha ao propagar um ID de *trace* entre todos os serviços e componentes envolvidos em uma transação. Um ID de *trace* é um identificador único atribuído a uma única requisição ou transação que percorre

vários serviços, permitindo que as equipes sigam seu fluxo e diagnostiquem problemas.

- **Problema**: Quando os IDs de *trace* não são propagados de maneira consistente, as equipes enfrentam dificuldades em correlacionar dados entre *logs*, métricas e *traces*, prejudicando uma visão abrangente do comportamento do sistema. O tempo de resolução de problemas aumenta porque os engenheiros precisam consultar manualmente dados de vários serviços sem o benefício de um ID de *trace*. Isso resulta em visibilidade incompleta sobre o fluxo das requisições entre os serviços, dificultando a análise de desempenho.
- **Exemplo**: Um usuário envia uma requisição para comprar um item em uma plataforma de *e-commerce*. A requisição passa por vários serviços: autenticação, inventário, processamento de pagamento e execução do pedido. Cada serviço registra seus próprios dados, mas como o ID de *trace* não é consistentemente passado de um serviço para o próximo, os logs de cada serviço não podem ser conectados. A equipe de engenharia tem dificuldade em diagnosticar o motivo do atraso no processamento de pagamento, já que não há um *trace* unificado mostrando o fluxo da requisição, o que leva a um aumento no tempo de resolução de problemas.
- **Solução Recomendada**:
  - Garanta que todos os serviços envolvidos no manuseio de uma requisição propaguem o mesmo ID de *trace* através de seus *logs*, métricas e *traces*; e
  - Utilize ferramentas de rastreamento distribuído (por exemplo, OTel e Jaeger) que geram e propagam automaticamente os IDs de *trace* entre os serviços;
- **Consequências**:
  - **Positivas**:
    - **Melhor correlação**: Com um ID de *trace* consistente, é mais fácil correlacionar *logs*, métricas e *traces*, e consequentemente permite uma identificação mais rápida da causa raiz de problemas de desempenho ou falhas; e
    - **Melhor visibilidade do sistema**: Ao rastrear consistentemente as requisições através de todos os serviços, as equipes ganham visibilidade completa sobre o ciclo de vida de uma requisição.
  - **Negativas**:
    - **Configuração inicial e instrumentação**: Implementar a propagação consistente do ID de *trace* em todos os serviços pode exigir um esforço significativo; e
    - **Sobrecarga de dados**: Propagar os IDs de *trace* através de todos os serviços

pode aumentar o volume de dados de *trace*, o que precisa ser gerido de forma eficaz.

- **Nome**: Subestimar a Importância dos Traces
- **Relevância**: Média
- **Categoria**: Traces
- **Contexto**: Esse anti-padrão ocorre quando equipes ou organizações não reconhecem completamente o valor dos *traces* como parte de sua estratégia de observabilidade. O *trace* é essencial para fornecer visibilidade no ciclo de vida de requisições ou transações através de múltiplos serviços, ajudando a detectar gargalos, problemas de latência e falhas em sistemas distribuídos. No entanto, algumas equipes podem subutilizar ou negligenciar o *trace* porque confiam demais em métricas e *logs*, assumindo que esses são suficientes para monitorar o desempenho do sistema e solucionar problemas.
- **Problema**: Quando a importância dos *traces* é subestimada, diversos problemas podem surgir. Primeiramente, há visibilidade limitada, já que as equipes ficam com uma visão fragmentada do comportamento do sistema, dificultando a compreensão de como as requisições fluem entre os serviços e onde ocorrem falhas ou lentidões. Isso leva à incapacidade de identificar gargalos, pois métricas como tempos de resposta ou *throughput* podem não fornecer detalhes suficientes para apontar problemas específicos de desempenho, especialmente em sistemas distribuídos complexos. Como resultado, os tempos de resolução de incidentes são mais longo. Além disso, o processo de depuração se torna mais complexo sem *traces*, já que as equipes não têm visibilidade completa das operações e interações. Por fim, negligenciar o *traces* pode resultar em oportunidades perdidas de otimização, como identificar operações de longa duração, chamadas de serviço ineficientes ou consultas de banco de dados de baixo desempenho.
- **Exemplo**: Uma empresa depende de métricas como tempos de resposta e taxas de erro para monitorar o desempenho de seu aplicativo web. No entanto, quando ocorre um problema crítico, como uma desaceleração significativa no processo de registro, as métricas sozinhas não são suficientes para identificar a causa. A equipe fica analisando *logs* e métricas de serviços individuais, mas não consegue ver o fluxo completo das requisições no sistema distribuído. Se tivessem implementado os *traces*, poderiam rapidamente identificar que

uma consulta ao banco de dados no serviço de autenticação estava causando o atraso.

- **Solução Recomendada**:
  - Faça dos traces uma parte fundamental de sua estratégia de observabilidade. Além disso, combine-os com *logs* e métricas para obter *insights* mais profundos sobre o desempenho do sistema; e
  - Explique as equipes sobre os benefícios dos traces e como ele pode ajudar na resolução de problemas, otimização de desempenho e melhoria da confiabilidade do sistema.
- **Consequências**:
  - **Positivas**:
    - **Resolução mais rápida de incidentes**: Com os *traces* implementado, os incidentes podem ser diagnosticados e resolvidos mais rapidamente; e
    - **Melhor observabilidade**: Quando os *traces* são integrados com métricas e *logs*, ele oferece uma visão mais holística da saúde do sistema.
  - **Negativas**:
    - **Complexidade na configuração inicial**: Configurar os *traces* requer um investimento inicial em instrumentação, configuração e integração de ferramentas.
    - **Volume de dados e custos de armazenamento**: Os *traces* pode gerar um grande volume de dados, especialmente em sistemas de alto *throughput*, o que pode levar a custos maiores de armazenamento e processamento.

- **Nome**: Começar o *Trace* no API *Gateway* é Superestimado
- **Relevância**: Baixa
- **Categoria**: *Traces*
- **Contexto**: Esse anti-padrão ocorre quando organizações dependem exclusivamente do API Gateway como ponto de partida para rastrear requisições em seu sistema. O API Gateway frequentemente serve como ponto de entrada para as requisições dos clientes, e muitos sistemas o instrumentam para iniciar o *trace* quando uma requisição chega ao gateway. No entanto, isso pode ser problemático porque não oferece visibilidade completa na jornada do usuário ou no funcionamento interno dos serviços que ocorrem antes do gateway.

- **Problema**: Quando os *traces* começam apenas no API *Gateway*, as equipes perdem visibilidade sobre todo o fluxo da requisição antes deste ponto. Isso resulta na perda de contexto para possíveis problemas que podem ocorrer, como falhas de resolução DNS, problemas com proxies ou questões de rede. Também dificulta a análise da causa raiz, pois erros ou problemas de desempenho originados em estágios anteriores se tornam mais difíceis de identificar.
- **Exemplo**: Um usuário envia uma requisição para uma aplicação web que passa por um API *Gateway* antes de chegar aos serviços *backend*. O API *Gateway* inicia o *trace*. No entanto, antes de a requisição atingir o API *Gateway*, ela passa por um balanceador de carga interno que ocasionalmente descarta requisições sob alta carga, causando falhas intermitentes. Como o *trace* começa no API *Gateway*, a falha no balanceador de carga não é capturada, levando a equipe de engenharia a gastar tempo desnecessário solucionando problemas nos serviços *backend*, sem perceber o problema que ocorre anteriormente no processo.
- **Solução Recomendada**:
  - Em vez de iniciar o *trace* exclusivamente no API *Gateway*, garanta que os *traces* sejam iniciados o mais cedo possível no ciclo de vida da requisição; e
  - Além do *trace* no lado do servidor, considere implementar o *trace* no lado do cliente para capturar toda a jornada de uma requisição desde o dispositivo do usuário até o *backend*.
- **Consequências**:
  - **Positivas**:
    - **Visibilidade completa**: Ao começar o *trace* mais cedo no fluxo da requisição, você ganha visibilidade sobre todos os componentes que influenciam a requisição; e

      Resolução proativa de problemas: Com a cobertura completa do *trace*, você pode identificar e resolver problemas em estágios mais iniciais do fluxo da requisição.
  - **Negativas**:
    - **Complexidade aumentada**: Instrumentar todas as camadas do sistema, pode aumentar a complexidade da configuração do *trace*; e
    - **Sobrecarga de dados**: Rastrear toda a jornada de uma requisição pode gerar

mais dados de trace, o que pode exigir recursos adicionais de armazenamento e processamento.

APÊNDICE B – DETALHES DE IMPLEMENTAÇÃO E IMPLANTAÇÃO DO OBSERVA

A implementação do Observa foi realizada utilizando tecnologias do ecossistema *Python* que, em conjunto, permitem a construção de um serviço leve, modular e facilmente integrável com ambientes de microsserviços. O repositório contendo o Observa está disponível em: <https://github.com/andersonalmada/detect-antipatterns>.

A base arquitetural do Observa utiliza o *FastAPI*[1], que é um *framework* web moderno e de alto desempenho para a construção de APIs com Python. O Observa é executado com *Uvicorn*[2], servidor ASGI[3] que oferece alta eficiência no tratamento simultâneo de requisições. A arquitetura resultante é modular e orientada à componibilidade, característica essencial para um *framework* cujo papel envolve conectar-se a diversos serviços, processar informações heterogêneas e aplicar heurísticas ou regras de inferência para detectar APs de observabilidade.

A configuração do Observa é realizada por meio da biblioteca *python-dotenv*[4], que carrega variáveis de ambiente definidas em arquivos específicos para cada contexto de execução, promovendo a separação entre lógica e configuração. Entre os parâmetros configuráveis incluem-se: (i) credenciais do banco de dados, (ii) adição de fonte de dados e (iii) adição de detectores. Esse mecanismo contribui para que o Observa seja facilmente adaptado a diferentes cenários sem modificação do código-fonte.

A camada de persistência emprega *SQLAlchemy*[5], com o banco de dados PostgreSQL acessado via *psycopg2-binary*[6], para armazenar os dados coletados e os resultados das análises. A modelagem cobre tanto registros brutos quanto informações derivadas, como: dados provenientes de diferentes fontes, avaliações dos detectores de anti-padrões e histórico temporal de ocorrências, permitindo análises evolutivas. Os *schemas* utilizados para validação e transporte de dados são definidos com *Pydantic*[7], garantindo consistência estrutural tanto nas entradas quanto nas saídas da API.

A etapa de coleta de dados e envio para API de detectores é realizada utilizando a biblioteca *Requests*[8], responsável por executar requisições HTTP. A camada de apresentação uti-

[1] <https://fastapi.tiangolo.com/>
[2] <https://uvicorn.dev/>
[3] <https://asgi.readthedocs.io/en/latest/>
[4] <https://github.com/theskumar/python-dotenv>
[5] <https://www.sqlalchemy.org/>
[6] <https://pypi.org/project/psycopg2-binary/>
[7] <https://docs.pydantic.dev/>
[8] <https://requests.readthedocs.io/en/latest/>

liza *Jinja2*[9] para geração dinâmica de interfaces HTML. A escolha por ferramentas amplamente consolidadas, assegura longevidade ao projeto e reduz barreiras para futuros pesquisadores ou desenvolvedores que desejem estender seus componentes.

Com o objetivo de simplificar o processo de implantação e reduzir a complexidade associada à configuração manual do ambiente de execução, o Observa foi empacotado como uma imagem *Docker* (ver Código-fonte 3). Essa abordagem assegura portabilidade, reprodutibilidade e isolamento do ambiente, permitindo que o *framework* seja executado de forma consistente em diferentes infraestruturas, independentemente do sistema operacional subjacente.

```
# Base Python image
FROM python:3.11-slim

# Set working directory
WORKDIR /app

# Install dependencies
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# Copy project files
COPY . .

# Expose FastAPI default port
EXPOSE 8000

# Command to run the API
CMD ["uvicorn", "observa.app:app", "--host", "0.0.0.0", "--port", "8000"]
```

Código-fonte 3 – Dockerfile utilizado para a construção da imagem do framework Observa

Para viabilizar a execução integrada do Observa com seus componentes de suporte, foi definido um arquivo *Docker Compose* (ver Código-fonte 4), responsável por orquestrar os contêineres necessários ao funcionamento completo do *framework*. Essa abordagem facilita a inicialização coordenada dos serviços e assegura que suas interdependências sejam corretamente resolvidas.

O serviço principal, identificado como *observa*, corresponde à instância do Observa propriamente dita. Ele é baseado na imagem *Docker* previamente construída e expõe a aplicação por meio da porta 8000, possibilitando o acesso à API e à interface de visualização. Variáveis

[9] <https://jinja.palletsprojects.com/en/stable/>

de ambiente são utilizadas para configurar dinamicamente aspectos do *framework*, incluindo: (i) parâmetros de conexão com o banco de dados PostgreSQL; (ii) definição de fontes de dados locais e (iii) definição dos detectores disponíveis.

```
version: "3.9"
services:
  observa:
    image: andersonalmada/observa
    container_name: observa
    ports:
      - "8000:8000"
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: postgres
      POSTGRES_DB: antipatterns
      POSTGRES_HOST: postgres
      POSTGRES_PORT: 5432
      SOURCES_LOCAL_NAME: alerts, logs
      SOURCES_LOCAL_PACKAGE: observa.sources.alerts.AlertsDataSource, observa
          .sources.logs.LogsDataSource
      DETECTOR_LOCAL_NAME: alertsdetector
      DETECTOR_LOCAL_PATH: observa.detectors.excessive_alerts.
          ExcessiveAlertsDetector
      DETECTOR_LOCAL_AP: Alerts Excessive
    depends_on:
      - db
    restart: always
  db:
    image: postgres:15
    container_name: postgres
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: postgres
      POSTGRES_DB: antipatterns
    ports:
      - "5432:5432"
    volumes:
      - postgres_data:/var/lib/postgresql/data
    restart: always
volumes:
  postgres_data:
```

Código-fonte 4 – Arquivo Docker Compose utilizado para a implantação do framework Observa

Nesse exemplo `SOURCES_LOCAL_NAME` e `SOURCES_LOCAL_PACKAGE` contemplam

quais fontes serão instaladas automaticamente. No caso, são duas fontes, sendo a primeira relacionado a alertas e a segunda relacionada a *logs*. O *Python* permite importar módulos e classes em tempo de execução, o que viabiliza a descoberta e o carregamento dinâmico de componentes. Cada um dele são instâncias de classes que herdam `DataSource`, como explicado na Seção 6.4.1. Para os alertas, foi implantado a classe `AlertsDataSource`, que está no pacote *observa.sources.alerts*. Já para os *logs*, foi implantado a classe `LogsDataSource`, que está no pacote *observa.sources.logs*. Além disso, nesse exemplo, tem o `DETECTOR_LOCAL_NAME`, `DETECTOR_LOCAL_PACKAGE` e `DETECTOR_LOCAL_AP` que contemplam quais detectores serão instaladas automaticamente. No caso, é um detector relacionado ao AP de alertas excessivos. Cada detector são instâncias de classes que herdam `Detector`, como explicado na Seção 6.4.2.

O serviço de persistência é provido por um contêiner independente baseado na imagem oficial do *PostgreSQL*[10], configurado para armazenar os dados utilizados pelo Observa, tais como fontes cadastradas, detectores registrados, resultados de detecção e histórico de análises. A persistência dos dados é garantida por meio de um volume nomeado, que assegura a preservação das informações mesmo em cenários de reinicialização ou atualização dos contêineres. A dependência explícita entre o serviço *observa* e o serviço de *banco de dados* garante que o *framework* seja iniciado apenas quando o mecanismo de persistência estiver disponível, reduzindo falhas durante o processo de inicialização.

A utilização de variáveis de ambiente reforça a flexibilidade do processo de implantação, permitindo ajustes de comportamento do Observa sem a necessidade de reconstrução da imagem. Em conjunto com o uso de *Docker* e *Docker Compose*, essa abordagem promove a padronização do ambiente de execução, facilita a reprodução experimental em contextos acadêmicos, reduz o esforço de configuração para novos usuários e oferece suporte tanto a implantações locais quanto a ambientes mais complexos, como servidores dedicados ou infraestruturas conteinerizadas.

Além disso, tal estratégia encontra-se alinhada com práticas modernas de engenharia de software e DevOps, tornando o Observa particularmente adequado para estudos empíricos, experimentos controlados e cenários de avaliação comparativa, nos quais a consistência do ambiente de execução constitui um fator crítico para a validade dos resultados obtidos.

[10] <https://www.postgresql.org/>

B.1 Detalhes de Implementação das Fontes de Dados e Detectores no Observa

Nesta seção, são detalhados os aspectos de implementação das fontes de dados e dos detectores no Observa, com foco em como esses componentes são criados e configurados por meio de código. Descrevem-se as estruturas, abstrações e mecanismos utilizados para definir as fontes de coleta e os detectores, evidenciando como o uso de código permite flexibilidade, extensibilidade e integração com diferentes cenários de monitoramento e detecção de APs.

B.1.1 Criação de uma Fonte de Dados

Para demonstrar a extensibilidade do Observa, esta subseção apresenta um exemplo de criação de uma nova fonte de dados por meio da implementação da abstração `DataSource`. O objetivo do exemplo é evidenciar como diferentes tecnologias de observabilidade podem ser integradas ao *framework* por meio de um contrato comum, sem dependências diretas de fornecedores ou ferramentas específicas. A lógica de acesso à tecnologia subjacente permanece encapsulada na classe concreta, enquanto o restante do *framework* interage exclusivamente com a abstração definida.

Para a criação de uma fonte de dados, o Observa oferece três possibilidades: (i) seleção de um arquivo no formato JSON; (ii) implementação explícita de uma classe que estenda a classe abstrata `DataSource`; e (iii) disponibilização de uma API que exponha um método HTTP do tipo GET. Em todos os casos, são criadas instâncias de classes concretas que estendem a abstração `DataSource`, assegurando uniformidade na interação com o restante do *framework*.

Para ilustrar o primeiro caso, o Código-fonte 6 apresenta o conteúdo de um arquivo JSON utilizado como fonte de dados. Nesse exemplo, a fonte representa a coleta estática de alertas gerados por um sistema, em que cada item corresponde ao conjunto de alertas produzidos em um intervalo de uma hora, totalizando quatro horas de coleta. Dessa forma, são gerados 35 alertas no total (12 + 5 + 7 + 11), variando tanto o nome do alerta quanto o serviço responsável por sua emissão.

Internamente ao Observa, quando o usuário define um nome para a fonte de dados, seleciona o arquivo JSON e realiza o cadastro por meio da interface gráfica, é criada uma instância da classe `JSONDataSource` (ver Código-fonte 6) e adicionado no banco de dados do Observa.

```
[
    {
        "alert_name": "HighCPUUsage",
        "service": "Atlas",
        "count": 12
    },
    {
        "alert_name": "MemoryLeakWarning",
        "service": "Boreas",
        "count": 5
    },
    {
        "alert_name": "HighCPUUsage",
        "service": "Boreas",
        "count": 7
    },
    {
        "alert_name": "DatabaseConnectionError",
        "service": "Chronos",
        "count": 11
    }
]
```

Código-fonte 5 – Exemplo de implementação de uma fonte de dados estática no Observa

```
from typing import Any
from observa.framework.base import DataSource

class JSONDataSource(DataSource):
    def __init__(self, name: String = None, data: Any = None):
        self.name = name
        self.data = data

    def load(self) -> Any:
        return self.data
```

Código-fonte 6 – Implementação da classe JSONDataSource que herda DataSource para fontes de dados estática

Para ilustrar o segundo caso, o Código-fonte 7 ilustra uma implementação simplificada de uma fonte de dados. A classe `LogsLocal` herda da abstração `DataSource` e implementa os métodos necessários para realizar a coleta, normalização e disponibilização dos dados para

a detecção. A lógica específica de acesso à tecnologia subjacente permanece encapsulada na classe concreta, enquanto o restante do *framework* interage apenas com a abstração. Essa fonte representa a coleta estática dos *logs* gerados por um sistema, onde cada item do *array* corresponde a um *log*. Nesse caso, internamente, o Observa pode carregar essa fonte de dados, a partir da implantação pelo *Docker Compose*.

```
from typing import Any
from observa.framework.base import DataSource

class LogsLocal(DataSource):
    def load(self) -> Any:
        logs = [
            [
                {
                    "timestamp": "2025-03-01T10:20:00Z",
                    "level": "DEBUG",
                    "service": "payment-service",
                    "trace_id": "aaa111",
                    "message": "Checking payment processing loop iteration"
                },
                {
                    "timestamp": "2025-03-01T10:20:01Z",
                    "level": "INFO",
                    "service": "checkout-service",
                    "trace_id": "aaa111",
                    "message": "Checkout complete"
                },
                {
                    "timestamp": "2025-03-01T10:20:02Z",
                    "level": "DEBUG",
                    "service": "payment-service",
                    "trace_id": "aaa111",
                    "message": "Checking payment processing loop iteration"
                },
            ]

        return logs
```

Código-fonte 7 – Exemplo de implementação de uma fonte de dados estática no Observa em uma classe filha

Por fim, no terceiro caso, é possível adicionar uma fonte de dados a partir de uma API de um serviço responsável pela coleta dos dados. O Código-fonte 8 ilustra uma implementação

de uma API que expõe dados de alertas. Nesse exemplo, foi utilizado o *Flask* para a criação de uma aplicação web em Python, executada na porta 9091, na qual é definida uma rota HTTP do tipo GET para o *endpoint* `/alerts`.

Uma vez que a aplicação esteja em execução, o usuário do Observa deve configurar essa API por meio da interface gráfica do *framework*. Embora o exemplo apresentado utilize a tecnologia Python, assim como o próprio Observa, a arquitetura não impõe restrições quanto à tecnologia empregada na implementação das fontes de dados remotas, desde que os dados sejam disponibilizados por meio de uma API acessível.

```
from flask import Flask, jsonify, request

app = Flask(__name__)

def generate_alerts(count: int, days=7, hours=24):
...

@app.route("/alerts", methods=["GET"])
def get_alerts():
    try:
        count = int(request.args.get("count", 1000))
        return jsonify(generate_alerts(count))

    except Exception as e:
        return jsonify({"error": str(e)}), 400

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=9091, debug=True)
```

Código-fonte 8 – Exemplo de implementação de uma fonte de dados remota no Observa

Para cada fonte de dados remota adicionada ao Observa e baseada em uma API, o *framework* cria internamente uma instância da classe `RemoteDataSource` (ver Código-fonte 9), que herda da abstração `DataSource` e implementa o método `load` por meio de uma requisição HTTP do tipo GET ao endpoint informado pelo usuário.

Essa estratégia preserva a independência entre o mecanismo de coleta e a tecnologia subjacente, uma vez que a lógica específica de acesso aos dados permanece encapsulada na classe concreta.

```
from observa.framework.base import DataSource
from typing import Any
import requests

class RemoteDataSource(DataSource):
    def __init__(self, name: String = None, api: String = None):
        self.name = name
        self.api = api

    def load(self) -> Any:
        response = requests.get(self.api)
        if response.status_code == 200:
            return response.json()
        else:
            print(f"Erro {response.status_code}: {response.text}")
```

Código-fonte 9 – Implementação da classe RemoteDataSource que herda DataSource para fontes de dados remotas

Assim, essa abordagem permite que novas fontes sejam adicionadas de forma incremental, favorecendo a evolução do *framework* e sua adaptação a diferentes ecossistemas de observabilidade, sem a necessidade de modificações no núcleo arquitetural do Observa.

B.1.2 Criação de um Detector de Anti-padrão

De forma análoga à criação de fontes de dados, a adição de novos detectores no Observa ocorre por meio da implementação da abstração `Detector`. Cada detector representa uma unidade analítica autônoma, responsável por identificar um AP específico a partir de um conjunto de dados.

Para a criação de um detector de AP, o Observa oferece duas possibilidades: (i) implementação explícita de uma classe que estenda a classe abstrata `Detector`; e (ii) disponibilização de uma API que exponha um método HTTP do tipo POST. Em todos os casos, são criadas instâncias de classes concretas que estendem a abstração `Detector`, assegurando uniformidade na interação com o restante do *framework*.

Para ilustrar o primeiro caso, o Código-fonte 10 apresenta um exemplo de implementação de um detector de anti-padrão de Alertas Excessivos. A classe herda da abstração `Detector` e implementa a lógica necessária para analisar os dados fornecidos pela fonte, produzindo um

resultado estruturado que inclui evidências sobre a detecção.

Todo detector precisa receber dados no formato JSON, que podem ter qualquer formato, aplicar as regras ou heurísticas de detecção de acordo com os dados recebidos e retornar uma resposta também em formato JSON. Essa resposta inclui, entre as linhas 14 e 18, os seguintes campos: `analyzed`, que indica a quantidade total de elementos analisados; `detected`, que informa quantos desses elementos foram classificados como AP; e `data`, que contém os dados modificados, refletindo o resultado da detecção.

Nesse exemplo, para cada alerta recebido, é avaliado se o valor de `count` é superior a 10. Se sim, o item é marcado como AP (`detected`) e informado o motivo para que isso tenha ocorrido (`reason`). Esse valor 10, é utilizado como referência ao longo da tese, pois é o valor considerado como o limiar para cada hora de trabalho de um engenheiro de observabilidade para evitar a fadiga de alertas[11]. Nesse caso, internamente, o Observa pode carregar esse detector, a partir da implantação pelo *Docker Compose*.

```
1  from observa.framework.base import Detector
2  from typing import Any, Dict
3
4  class ExcessiveAlertsDetector(Detector):
5      def detect(self, data: Any) -> Dict:
6          for item in data:
7              if int(item.get('count', 0)) > 10:
8                  item["detected"] = True
9                  item["reason"] = "detect_limit"
10             else:
11                 item["detected"] = False
12                 item["reason"] = ""
13
14         count = sum(1 for x in data if x["detected"])
15         response = {
16             "analyzed": len(data),
17             "detected": count,
18             "data": data
19         }
20
21         return response
```

Código-fonte 10 – Retorno de um detector de anti-padrão no Observa

Para ilustrar o segundo caso, o Código-fonte 11 apresenta um detector que recebe

[11] <https://www.suprsend.com/post/alert-fatigue>

dados no formato JSON por meio do método HTTP POST, aplica as regras ou heurísticas de detecção de acordo com os dados recebidos e retorna uma resposta, também em formato JSON, seguindo a estrutura exigida pelo Observa. Nesse exemplo, o detector verifica se há ocorrência de *logs* excessivos nos dados enviados.

```
from flask import Flask, request, jsonify
from collections import Counter, defaultdict
from datetime import datetime
import json

app = Flask(__name__)

def detect_volume_spikes(data, max_kbytes=1):
...
def detect_repetitive_messages(data, repetition_threshold=5):
...
def detect_level(data):
...
@app.post("/detector")
def detect():
    input_data = request.get_json()
    for item in input_data:
        item["detected"] = False
        item["reason"] = ""

    output_data = detect_volume_spikes(data=input_data, max_kbytes=1)
    output_data = detect_repetitive_messages(data=output_data,
        repetition_threshold=5)
    output_data = detect_level(data=output_data)

    count = sum(1 for x in output_data if x["detected"])

    response = {
        "analyzed": len(input_data),
        "detected": count,
        "data": output_data
    }

    return jsonify(response)

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=5001)
```

Código-fonte 11 – Implementação de um detector de anti-padrão de Logs Excessivos no Observa

Para cada detector remoto adicionado ao Observa e baseada em uma API, o *framework* cria internamente uma instância da classe `RemoteDetector` (ver Código-fonte 12), que herda da abstração `Detector` e implementa o método `detect` por meio de uma requisição HTTP do tipo POST ao endpoint informado pelo usuário.

```
from observa.framework.base import Detector
from typing import Any, Dict
import requests

class RemoteDetector(Detector):
    def detect(self, data: Any) -> Dict:
        response = requests.post(self.api_url, json=data)
        if response.status_code == 200:
            return response.json()
        else:
            print(f"Erro {response.status_code}: {response.text}")
```

Código-fonte 12 – Implementação da classe RemoteDetector que herda Detector para detectores remotos

Essa estratégia preserva a independência entre o mecanismo de detecção e a tecnologia subjacente, uma vez que as regras de detecção permanecem encapsuladas na classe concreta.

A partir da apresentação dos conceitos e códigos-fonte do Observa, é possível afirmar que a separação entre fontes de dados e detectores garante que a lógica de detecção permaneça desacoplada da forma como os dados são coletados. Além disso, a modularidade dos detectores facilita a incorporação de novos APs ao *framework*, à medida que podem surgir novas más práticas no contexto de observabilidade.

APÊNDICE C – ROTEIRO DE AVALIAÇÃO DE USABILIDADE DO OBSERVA

O Observa (<https://github.com/andersonalmada/detect-antipatterns>) é um *framework* desenvolvido para facilitar a detecção automática de anti-padrões de observabilidade pelos usuários. Os anti-padrões foram catalogados anteriormente e podem ser visualizados no seguinte link: <https://observability-antipatterns.github.io/>. Ele atua intermediando fontes de dados e detectores de anti-padrões. Por meio do *framework*, é possível criar qualquer tipo de fonte de dados, desde que estruturada em formato JSON, como por exemplo dados de alertas, *logs*, *traces*, *dashboards*, entre outros. Uma *playlist* foi criada para facilitar o entendimento da prática: <https://tinyurl.com/playlist-observa>

Na configuração dessas fontes, existem duas possibilidades:

1. Criar uma nova fonte de dados manualmente, em um arquivo JSON; ou
2. Adicionar uma fonte de dados existente, vinculando-a por meio de uma API.

No primeiro caso, tratam-se de dados estáticos, portanto, o detector deve sempre produzir a mesma resposta ao analisá-los. Já no segundo caso, como a fonte está ligada a uma API, o *endpoint* pode fornecer dados dinâmicos. Como pré-requisito, essa API deve disponibilizar os dados via requisição GET.

Independentemente da origem, uma vez obtidos os dados, o Observa os envia ao detector selecionado pelo usuário. Cada detector deve ser uma API capaz de receber um JSON via método POST. Além disso, é obrigatório que o detector retorne uma resposta também em formato JSON, seguindo a estrutura abaixo:

```
response = {
    "analyzed": len(input_data),
    "detected": count,
    "data": output_data
}
```

Código-fonte 13 – Resposta dos Detectores do Observa

Sendo `input_data` o conjunto de dados recebido pelo detector (ou seja, o JSON proveniente da fonte de dados), o campo `count` representa a quantidade de elementos identificados como um anti-padrão. Já `output_data` corresponde aos dados após o processamento do detector, normalmente, para cada entrada do JSON original, é adicionado um novo campo indicando que um anti-padrão foi detectado. Dessa forma:

- “analyzed” indica a quantidade total de elementos analisados;
- “detected” indica quantos desses elementos foram classificados como anti-padrão;
- “data” contém os dados modificados (output_data), refletindo o resultado da detecção.

C.1 Orientações para Execução

Esta seção apresenta as instruções necessárias para a realização das atividades com o *framework* Observa. Siga os passos descritos a seguir para garantir uma execução consistente.

- Serão executadas duas tarefas no *framework*: uma relacionada ao uso dos recursos disponíveis e outra referente à criação desses recursos.
- Como pré-requisito, é necessário ter o **Docker Compose** instalado. A documentação oficial está disponível em: https://docs.docker.com/compose/
- Em seguida, realize o download do arquivo docker-compose.yml por meio do link: https://tinyurl.com/observa-docker
- Após finalizar o download, execute o Docker Compose (verifique se as portas configuradas estão livres no seu ambiente): docker compose up -d
- Para cada uma das tarefas realizadas, **meça o tempo necessário** para concluí-las e, caso identifique alguma oportunidade de melhoria, registre suas observações referentes ao funcionamento do *framework* Observa.
- Após as tarefas, responda o seguinte formulário: https://forms.gle/BmAV5UCW6K7gGZbq9

C.2 Tarefa 1 - Usar o *Framework* Observa

Nesta etapa, você utilizará o Observa exclusivamente com os recursos já disponibilizados. O objetivo é verificar sua capacidade de localizar as funcionalidades existentes, executar a tarefa proposta e identificar os anti-padrões que já estão configurados no sistema.

1. Com o Observa em execução, abra o navegador e acesse: http://localhost:8000
2. Realize o login com usuário e senha admin
3. Na aba Setup, no Passo 1, selecione a fonte alertas. Essa fonte representa os alertas gerados por um sistema, onde cada item corresponde aos alertas produzidos em um intervalo de 1 hora (totalizando 4 horas de dados). Ela é composta pelos seguintes dados:

```
[
 {
      "alert_name":"HighCPUUsage",
```

```
        "service":"Atlas",
        "count":12
    },
    {
        "alert_name":"MemoryLeakWarning",
        "service":"Boreas",
        "count":5
    },
    {
        "alert_name":"HighCPUUsage",
        "service":"Boreas",
        "count":7
    },
    {
        "alert_name":"DatabaseConnectionError",
        "service":"Chronos",
        "count":11
    }
]
}
```

Código-fonte 14 – Dados de Alertas

4. No Passo 2, selecione o anti-pattern Alerts Excessive.
5. Em seguida, escolha o detector alertsdetector. Esse detector analisa a cada 1 hora se a quantidade de alertas é maior ou igual a 10, se sim é considerado um anti-padrão, pois excede a quantidade de alertas que um operador consegue resolver nesse tempo e pode levar a uma fadiga de alertas se acontecer muitas notificações (acima de 10). Em código, esse detector analisa cada elemento da fonte e verifica se o valor de count é maior ou igual a 10. Caso essa condição seja satisfeita, o item é marcado como detected=True.
6. Na aba Detection, no Passo 3, pressione Run Detector.
7. O sistema exibirá quantos itens foram analisados e quantos foram detectados como anti-padrão.
8. Conforme mencionado anteriormente, registre o tempo total necessário para realizar a atividade e anote quaisquer problemas encontrados durante o processo.

C.3 Tarefa 2 - Criar fonte + detector

Nesta etapa, você criará uma nova fonte de dados e um novo detector de anti-padrões no *framework* Observa. O objetivo é verificar sua capacidade de configurar e executar essas funcionalidades corretamente. A seguir, apresentamos um exemplo de fonte de dados e um exemplo de detector para a detecção de *logs* excessivos, que servirão como referência para a sua implementação.

1. Com o Observa em execução, abra o navegador e acesse: http://localhost:8000
2. Na aba Setup, no Passo 1, acesse a seção Adicionar uma nova fonte de dados. Escolha um nome de sua preferência para a fonte (por exemplo, *logs*). Em Entrada de dados, selecione a opção Upload.
3. Caso deseje, um JSON de exemplo foi disponibilizado abaixo, basta copiá-lo e salvá-lo em um arquivo com extensão .json (por exemplo, logs.json).


```
[
   {
      "timestamp":"2025-03-01T10:20:00.001Z",
      "level":"DEBUG",
      "service":"payment-service",
      "trace_id":"aaa111",
      "message":"Checking payment processing loop iteration"
   },
   {
      "timestamp":"2025-03-01T10:20:00.002Z",
      "level":"DEBUG",
      "service":"payment-service",
      "trace_id":"aaa111",
      "message":"Checking payment processing loop iteration"
   },
   {
      "timestamp":"2025-03-01T10:20:00.003Z",
      "level":"DEBUG",
      "service":"payment-service",
      "trace_id":"aaa111",
      "message":"Checking payment processing loop iteration"
   },
   {
      "timestamp":"2025-03-01T10:20:00.004Z",
      "level":"INFO",
      "service":"payment-service",
      "trace_id":"aaa111",
      "message":"Payment validation complete"
```

```
    },
    {
        "timestamp":"2025-03-01T10:20:00.005Z",
        "level":"DEBUG",
        "service":"payment-service",
        "trace_id":"aaa111",
        "message":"Checking payment processing loop iteration"
    },
    {
        "timestamp":"2025-03-01T10:20:00.006Z",
        "level":"WARN",
        "service":"payment-service",
        "trace_id":"aaa111",
        "message":"Slow response from bank API"
    },
    {
        "timestamp":"2025-03-01T10:20:00.007Z",
        "level":"DEBUG",
        "service":"payment-service",
        "trace_id":"aaa111",
        "message":"Checking payment processing loop iteration"
    },
    {
        "timestamp":"2025-03-01T10:20:00.008Z",
        "level":"DEBUG",
        "service":"payment-service",
        "trace_id":"aaa111",
        "message":"Checking payment processing loop iteration"
    },
    {
        "timestamp":"2025-03-01T10:20:00.009Z",
        "level":"DEBUG",
        "service":"payment-service",
        "trace_id":"aaa111",
        "message":"Checking payment processing loop iteration"
    },
    {
        "timestamp":"2025-03-01T10:20:00.010Z",
        "level":"ERROR",
        "service":"payment-service",
        "trace_id":"aaa111",
        "message":"Failed to update payment status"
    }
]
```

Código-fonte 15 – Dados de Logs

4. Após selecionar o arquivo, clique em Add Source.
5. Neste exemplo de criação de um detector, foi elaborado um código em Python, utilizando o *framework* Flask. Como mencionado anteriormente, o detector deve expor uma rota com método POST, receber os dados provenientes da fonte, realizar o processamento necessário para detectar o anti-padrão e retornar um JSON no formato exigido pelo Observa.
6. Para criar e executar esse exemplo, siga os passos abaixo:
   a) Crie um projeto e no terminal, execute os seguintes comandos para instalação dos pacotes:

```
python3 -m venv venv
source venv/bin/activate
pip install Flask
```

   b) Crie um arquivo app.py e coloque o seguinte conteúdo:

```
from flask import Flask, request, jsonify
from collections import Counter

app = Flask(__name__)

def detect_repetitive_messages(data, repetition_threshold=5):
    messages = [log["message"] for log in data]
    counter = Counter(messages)

    repetitive = {msg: count for msg, count in counter.items() if
        count >= repetition_threshold}

    for log in data:
        if log["message"] in repetitive:
            log["detected"] = True
        else:
            log["detected"] = False

    return data

def detect_level(data):
    for item in data:
      if item.get('level') in ("DEBUG", "TRACE") and item["detected
          "] == False:
        item["detected"] = True

    return data

@app.post("/detector")
```

```
28 def detect():
29     input_data = request.get_json()
30     input_data = detect_repetitive_messages(data=input_data,
           repetition_threshold=5)
31     input_data = detect_level(data=input_data)
32
33     count = sum(1 for x in input_data if x["detected"])
34
35     response = {
36         "analyzed": len(input_data),
37         "detected": count,
38         "data": input_data
39     }
40
41     return jsonify(response)
42
43 if __name__ == "__main__":
44     app.run(host="0.0.0.0", port=5001)
```

Código-fonte 16 – Detector de Logs Excessivos em Python

c) Execute a aplicação:

```
python3 app.py
```

d) A API da aplicação está no seguinte endereço: http://localhost:5001/detector

e) Caso queria testar, antes de submeter no Observa, utilize o Postman com a mesma carga (logs.json).

f) Como o observa está em um docker, é necessário encontrar o IP da máquina. Execute o seguinte comando no terminal:

```
Se Linux   -   ifconfig
Se Windows -   ipconfig
```

g) Exemplo do endereço com o IP da máquina:

```
http://192.168.1.131:5001/detector
```

7. Na aba Setup, no Passo 2, vá para a parte de Adicionar um novo detector. Escolha o nome do anti-padrão (exemplo, *Logs Excessives*). Escolha o nome do detector desse anti-padrão

(exemplo, logsexcessivos). Em API do detector, coloque o endereço do item 6.6.

8. Clique em Add Detector.
9. Selecione tanto o source quanto o detector adicionados na aba Setup.
10. Após cadastrar a nova fonte e o novo detector, acesse a aba Detection e execute o processo normalmente.
11. Conforme mencionado anteriormente, registre o tempo total necessário para realizar a atividade e anote quaisquer problemas encontrados durante o processo.

ANEXO A – AVALIAÇÃO DE USABILIDADE DO OBSERVA

A.1 System Usability Scale (SUS)

O System Usability Scale (SUS) (BROOKE *et al.*, 1996) é um instrumento amplamente reconhecido e validado internacionalmente, utilizado para avaliar a usabilidade de sistemas computacionais, aplicações, websites, interfaces digitais e tecnologias em geral.

Na avaliação de usabilidade do Observa, cada participante respondeu a um conjunto de 10 afirmações relacionadas à sua experiência de uso do *framework*. As respostas foram registradas em uma escala do tipo *Likert* de cinco pontos, variando de 1 a 5, em que: 1 = Discordo totalmente, 2 = Discordo, 3 = Neutro, 4 = Concordo e 5 = Concordo totalmente.

1. Eu acho o observa fácil de usar.
2. Eu acho o observa mais complexo do que deveria ser.
3. Eu achei o observa simples e consistente.
4. Eu acho que precisaria de ajuda de uma pessoa técnica para usar o observa.
5. Eu achei que as funcionalidades do observa estão bem integradas.
6. Eu acho que existem muitas inconsistências no observa.
7. Eu imagino que a maioria das pessoas aprenderia a usar o observa rapidamente.
8. Eu considero o observa difícil de aprender.
9. Eu me senti confiante ao usar o observa.
10. Eu precisei aprender vários conceitos antes de conseguir usar o observa.

A.2 Net Promoter Score (NPS)

O Net Promoter Score (NPS) (REICHHELD, 2003) é uma métrica amplamente utilizada para mensurar o grau de satisfação e lealdade dos usuários, indicando a probabilidade de recomendação de um sistema a terceiros com base na experiência geral de uso.

Também na avaliação, os participantes responderam uma questão objetiva para responder o NPS, utilizando uma escala numérica que varia de 0 a 10, em que 0 representa “Nada provável recomendar” e 10 corresponde a “Extremamente provável recomendar”.

Em uma escala de 0 a 10, o quanto você recomendaria o observa para outra pessoa?