Observa: *Framework* para Detecção Automática de Anti-padrões de Observabilidade

Francisco A. A. Gomes
Universidade Federal do Ceará
Crateús, CE, Brasil
almada@crateus.ufc.br

Paulo A. L. Rego
Universidade Federal do Ceará
Fortaleza, CE, Brasil
paulo@dc.ufc.br

Fernando A. M. Trinta
Universidade Federal do Ceará
Fortaleza, CE, Brasil
fernando.trinta@dc.ufc.br

Resumo

Sistemas modernos adotam microsserviços para alcançar escalabilidade e entrega contínua. Embora esse estilo arquitetural simplifique o desenvolvimento, ele aumenta a complexidade operacional, tornando falhas mais frequentes e difíceis de diagnosticar, o que exige a adoção da observabilidade. Contudo, a observabilidade é frequentemente implementada sem diretrizes padronizadas, resultando em monitoramento ineficaz e na ocorrência de anti-padrões. Este artigo propõe o Observa, um *framework* para apoiar a detecção automática desses anti-padrões, e apresenta uma prova de conceito para validar a solução, evidenciando extensibilidade, neutralidade tecnológica, reprodutibilidade experimental, rastreabilidade das análises e possibilidade de comparação sistemática dos resultados.
Demo video: https://doi.org/10.5281/zenodo.20387190

Palavras-chave

Monitoramento, Observabilidade, Anti-padrões, Detecção

1 Introdução

A arquitetura de microsserviços possibilitou a evolução de sistemas monolíticos para aplicações mais escaláveis e flexíveis [3, 16]. Apesar dos ganhos em desempenho e produtividade, esse modelo introduz desafios de integração, maior risco de falhas e dificuldades de diagnóstico em ambientes distribuídos [2, 3, 15]. Nesse contexto, o monitoramento tradicional mostra-se limitado, reforçando a necessidade da adoção de observabilidade [10].

De acordo com [18], a observabilidade compreende um conjunto de práticas e ferramentas que vão além do monitoramento convencional, fornecendo *insights* sobre o estado interno de um sistema a partir da análise de suas saídas externas, predominantemente representadas por dados de telemetria, como métricas, *logs* e *traces*. Seu emprego amplia a visibilidade do fluxo de execução, favorecendo a depuração, a detecção de anomalias e a identificação de gargalos, o que contribui para melhores tempos de resposta e uso mais eficiente dos recursos [8, 9]. Apesar dos benefícios, a implementação da observabilidade frequentemente incorre em anti-padrões (APs) [1, 10, 11, 19], práticas que reduzem sua eficácia e podem levar a interpretações equivocadas e atrasos na resolução de incidentes. Um exemplo recorrente é a geração excessiva de alertas, decorrente de regras redundantes ou notificações demasiadamente granulares, o que provoca fadiga de alertas e faz com que engenheiros passem a ignorar até avisos críticos [17].

Diversos estudos investigaram soluções de observabilidade. Em [14], foi conduzido um estudo com a indústria para compreender desafios e melhores práticas no monitoramento de sistemas distribuídos. Em [11], apresentou-se uma pesquisa sobre microsserviços e observabilidade, com ênfase em ferramentas e desafios. O trabalho de [18] discutiu requisitos, boas práticas e soluções para observabilidade em ambientes distribuídos. Já [10] realizou um mapeamento sistemático, abordando maturidade, eficiência e direções futuras. Em [5], foi apresentado uma taxonomia de observabilidade para microsserviços, enquanto [6] apresentou um catálogo de APs que mapeia práticas inadequadas nesse contexto. Contudo, não foram identificados estudos que proponham uma solução capaz de detectar automaticamente tais APs de observabilidade.

Motivado por essa lacuna, este trabalho propõe o Observa, um *framework* destinado a apoiar equipes de monitoramento na detecção automática de APs de observabilidade. O Observa avalia a observabilidade implementada, identificando más práticas a partir dos dados de telemetria analisados. Para isso, o *framework* integra fontes heterogêneas de dados e aplica detectores especializados, cada um responsável por analisar indícios de um AP específico. Para validação, foi desenvolvida uma prova de conceito que demonstrou a capacidade do Observa de identificar APs, evidenciando características de extensibilidade, neutralidade tecnológica, reprodutibilidade experimental, rastreabilidade das análises e possibilidade de comparação sistemática de resultados.

O restante deste artigo está organizado da seguinte forma: a Seção 2 apresenta o Observa, incluindo arquitetura, princípios de projeto, detalhes de implementação, manuais e requisitos para demonstração; a Seção 3 descreve a prova de conceito utilizada na validação; a Seção 4 apresenta os trabalhos relacionados; por fim, a Seção 5 apresenta a conclusão e trabalhos futuros.

2 *Framework* Observa

O Observa é um *framework* projetado para apoiar a detecção automática de APs de observabilidade em aplicações distribuídas. Diferentemente das ferramentas convencionais de observabilidade que tradicionalmente são voltadas à coleta, visualização e correlação de sinais, o Observa tem como foco avaliar a qualidade da observabilidade efetivamente implementada, identificando más práticas associadas ao uso de métricas, *logs*, *traces*, alertas e *dashboards*. Para isso, o *framework* integra fontes heterogêneas de dados de telemetria e emprega detectores especializados, cada um responsável por analisar evidências de um AP específico.

2.1 Arquitetura

A Figura 1 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 coordenar 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 que tratadas pelo `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 de dados de observabilidade integrada ao Observa, e 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*. Os dados do `DataSource` podem ser estáticos ou dinâmicos. No caso estático, consistem em um *dataset* em formato JSON, contendo os dados a serem analisados pelo algoritmo de detecção. No caso dinâmico, consistem em um coleta a partir de um *endpoint* (API HTTP) que retorna dados em JSON, à qual o Observa se conecta por meio de uma interface para recuperar as informações e aplicar a detecção.

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. Na prática, um detector consiste em um serviço que pode ser desenvolvido em qualquer tecnologia que expõe uma API HTTP contendo o algoritmo de detecção, e que o Observa se conecta através de uma interface interna.

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 o *framework*. No Observa, estão implementadas duas estratégias de execução: simples e periódica. Na abordagem simples, a coleta de dados e a detecção são realizadas pelo Observa de forma sob demanda, conforme solicitação do usuário. Na estratégia periódica, o Observa realiza coletas recorrentes a partir da fonte selecionada, em intervalos de tempo previamente definidos, acumulando os dados ao longo do período de observação e, posteriormente, executando a detecção.

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 que levaram à detecção dos mesmos, de acordo com os dados 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.

2.2 Princípios de Design

O desenvolvimento do Observa foi orientado por um conjunto de princípios de *design* que garantem sua robustez, extensibilidade e alinhamento com objetivos científicos e práticos. 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 anti-padrões, 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 métricas, *logs*, *traces* e alertas 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**.

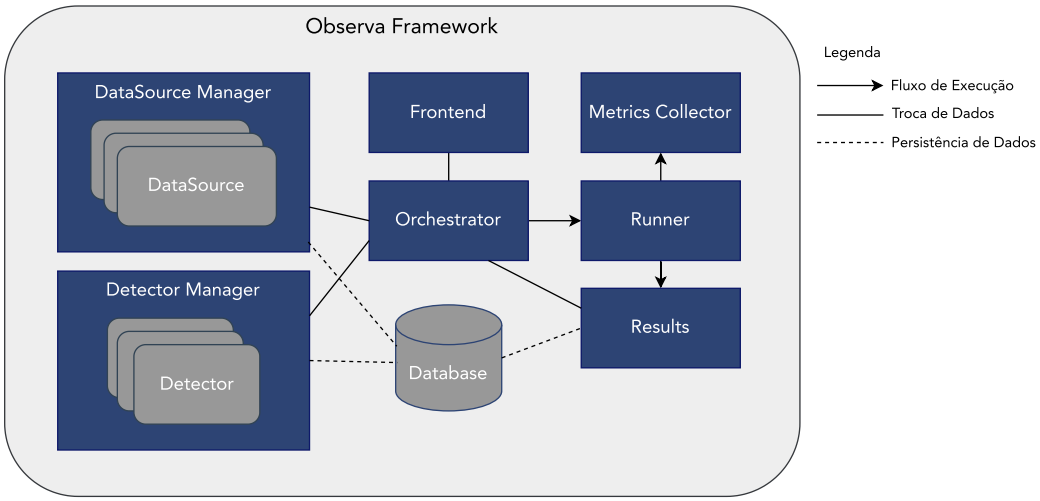

Figura 1: Arquitetura do Observa

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.

2.3 Detalhes de Implementação

O Observa foi desenvolvido em *Python*, utilizando um ambiente conteinerizado com Docker para garantir replicabilidade e facilitar a implantação em diferentes ambientes.

A camada de comunicação da aplicação foi implementada com *FastAPI*[1], utilizada na construção da API REST responsável pelo gerenciamento das análises, configuração dos detectores de APs e integração com os componentes de observabilidade. A escolha dessa tecnologia ocorreu devido ao suporte nativo à programação assíncrona, validação automática de dados e geração automática de documentação baseada no padrão OpenAPI.

Para persistência dos dados, utilizou-se o *SQLAlchemy*[2] como camada de abstração ORM, juntamente ao banco de dados PostgreSQL, acessado via *psycopg2-binary*[3]. Essa abordagem facilita a manutenção do sistema e a evolução do modelo de dados. Além disso, a validação, serialização e transporte de dados entre os componentes são realizados com o *Pydantic*[4], garantindo consistência nas requisições e respostas da API.

3 Prova de Conceito

Esta seção apresenta a prova de conceito (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. Nessa PoC, são realizadas as principais funcionalidades do Observa, como adicionar as fontes de dados, adicionar detectores dos APs, realizar a detecção, visualizar os resultados e históricos das detecções. Para efeito de demonstração, são utilizados os dados de alertas como fontes de dados e realizado a detecção de APs de Alertas Excessivos.

3.1 Fonte de Dados: Alertas

A fonte de dados utilizada na prova de conceito é estática, visando à replicabilidade, e consiste em um *dataset* com registros agregados de alertas coletados ao longo de 24 horas, totalizando 1000 alertas, volume enfrentado diariamente por 63% das organizações[5]. O *dataset* está disponível em: https://github.com/andersonalmada/detect-antipatterns/blob/master/alerts/alerts-1000.json. Os dados foram gerados por um simulador, disponível em: https://github.com/andersonalmada/detect-antipatterns/blob/master/alerts/fake.py.

3.2 Detector: Alertas Excessivos

O detector de Alertas Excessivos foi implementado como um serviço remoto em Flask (Python), com código-fonte disponível em: https://github.com/andersonalmada/detect-antipatterns/blob/master/alerts/docker/app.py . Ele analisa fluxos de alertas para identificar padrões de excesso e redundância temporal, associados a cenários de sobrecarga. A lógica de análise é composta por três etapas: (i) agrupamento temporal, (ii) detecção de excesso de volume e (iii) detecção de redundância temporal. Inicialmente, os alertas são organizados em janelas fixas de uma hora com base em seus *timestamps*.

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 10 alertas por membro por hora. Esse valor de 10, é utilizado como referência, pois é o valor considerado como o limiar

[1] https://fastapi.tiangolo.com/
[2] https://www.sqlalchemy.org/
[3] https://pypi.org/project/psycopg2-binary/
[4] https://docs.pydantic.dev/latest/
[5] https://www.netdata.cloud/resources/research/monitor-everything-anti-pattern/

para cada hora de trabalho de um engenheiro de observabilidade para evitar a fadiga de alertas[6]. 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 identifica redundâncias temporais dentro de cada grupo horário. Nesta PoC, os alertas são agrupados com base em atributos semânticos, como `host`, `service`, `name` e `severity`, e ordenados cronologicamente. Alertas consecutivos dentro de uma janela de tempo predefinida são considerados redundantes, caracterizando padrões de repetição em curtos intervalos. Neste exemplo, foi utilizada uma janela de 60 segundos. Essa análise permite identificar problemas não capturados apenas por métricas agregadas, como a reemissão frequente de alertas idênticos causada por falhas persistentes ou configurações inadequadas.

O serviço expõe um *endpoint* responsável por receber os alertas em formato JSON, correspondentes ao *dataset* analisado. O Observa espera como entrada dados estruturados em JSON, independentemente do formato adotado pela fonte de dados. Em cenários dinâmicos, como dito anteriormente, a coleta pode ser realizada automaticamente por meio de requisições HTTP GET. Neste exemplo, os dados são representados como um *array* JSON contendo os alertas a serem processados.

Após o recebimento dos dados, o Observa encaminha o conteúdo para o detector correspondente por meio de uma requisição HTTP POST. O detector recebe o JSON, executa as etapas de análise descritas e retorna os resultados em um formato padronizado definido pelo Observa. A resposta estruturada contém: (i) o número total de alertas analisados; (ii) a quantidade de alertas classificados como problemáticos; e (iii) os grupos detalhados, incluindo os alertas marcados e os respectivos motivos da classificação.

Tanto a fonte de dados quanto os detectores são independentes do Observa, podendo ser adicionados ou substituídos conforme a necessidade.

3.3 Execução da PoC

Na interface gráfica do Observa, são selecionados a fonte de dados e o detector. Nesse exemplo, é o *dataset* de alertas e o detector de AP de Alertas Excessivos, descrito anteriormente. Em seguida, é e configurado o modo de execução, optando-se pela `execução simples` de 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 as regras de análise específicas e retornou os resultados ao *framework*, que consolidou as informações, registrou os metadados da execução e persistiu os resultados.

Todo o processo de execução, incluindo a seleção dos parâmetros, a comunicação com detectores e a apresentação dos resultados, foi registrado no vídeo disponibilizado, a fim de evidenciar o funcionamento ponta a ponta do Observa em um cenário controlado de uso.

3.4 Resultados Obtidos

A Figura 2 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 algoritmo de detecção descrito anteriormente, foram identificados 767 alertas classificados como ocorrência do AP.

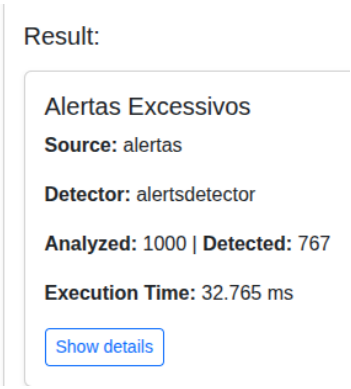

Figura 2: Visão Geral dos Resultados da Detecção de Alertas Excessivos

Uma vez que clica em `Show details`, detalhes da detecção são apresentados, como pode ser visto na Figura 3. A coluna `Alerts` é o *array* JSON processado; a coluna `Count` é a quantidade de itens agrupados em cada hora, representado pela coluna `Hour`; a coluna `Detected` identifica se naquela hora tem uma ocorrência de AP; e a coluna `Reason` indica a razão pela qual aquela hora foi marcada como AP.

Os resultados mostram que foi identificado 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, nesse exemplo, igual ou superior a 40, correspondente a dez alertas para cada um dos quatro membros da equipe, como pode ser visto na Figura 3, em que possui 48 na coluna `Count` e, consequentemente, é marcado como excessivo.

Além disso, foi detectado a ocorrência de redundância temporal, com os alertas semanticamente equivalentes, como pode ser visto na Figura 4, em que encontrou pelo menos os dois primeiros alertas semanticamente equivalentes no período de 60 segundos,

Com o intuito de ilustrar, a Figura 5 mostra um cenário no qual o número total de alertas foi inferior a 40, os alertas não foram classificados como excessivos e nem mesmo foi classificado como redundante, consequentemente, não houve a detecção do AP.

Além disso, a Figura 6 apresenta o histórico das execuções realizadas. A barra completa representa a quantidade total de elementos do *dataset*; neste exemplo, 1000 alertas. A parte em vermelho corresponde aos alertas problemáticos, totalizando 767 ocorrências, enquanto a parte em verde indica os alertas não excessivos, correspondendo a 233 ocorrências.

A partir da análise dos resultados e das explicações apresentadas, o Observa consegue integrar uma nova fonte de dados sem necessidade de adaptações no *framework*, além dos detectores. O

[6]https://www.suprsend.com/post/alert-fatigue

Alerts

Excessive	Host	Name	Service	Severity	Timestamp	Value
true	app-server-02	TestAlert1	analytics-service	info	2025-12-16T00:40:24Z	7
true	app-server-01	TestAlert3	inventory-service	info	2025-12-16T00:41:26Z	9

Count	Detected	Hour	Reason
48	✅ Yes	2025-12-16T00:00:00+00:00Z	detect_limit_excess

Figura 3: Detalhes de Alertas Excessivos Baseado no Limite

Excessive	Host	Name	Service	Severity	Timestamp	Value
true	app-server-01	TestAlert8	notification-service	info	2025-12-16T01:39:14Z	39
true	app-server-01	TestAlert8	notification-service	info	2025-12-16T01:39:15Z	42
false	app-server-01	TestAlert5	user-service	info	2025-12-16T01:04:56Z	35

Count	Detected	Hour	Reason
33	✅ Yes	2025-12-16T01:00:00+00:00Z	detect_redundancy

Figura 4: Detalhes de Alertas Excessivos Baseado na Redundância

Excessive	Host	Name	Service	Severity	Timestamp	Value
false	app-server-01	TestAlert6	auth-service	critical	2025-12-16T02:58:13Z	83
false	app-server-01	TestAlert0	analytics-service	info	2025-12-16T02:35:48Z	33

Count	Detected	Hour	Reason
36	❌ No	2025-12-16T02:00:00+00:00Z	

Figura 5: Detalhes de Alertas não Detectados como Excessivos

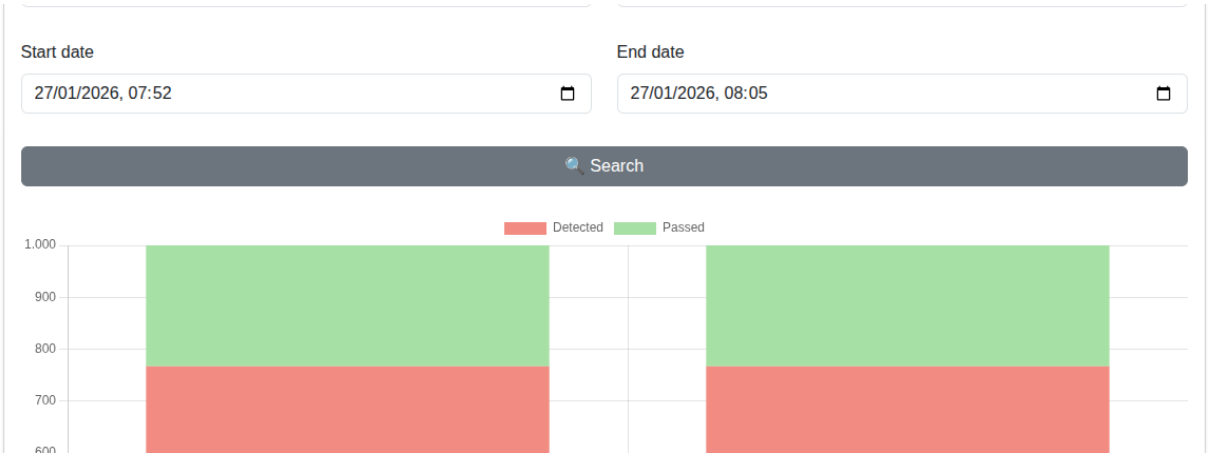

Figura 6: Histórico de Detecções

experimento também demonstrou que os detectores podem ser acionados de forma transparente, mantendo uma interface uniforme, o que reforça a flexibilidade e a extensibilidade do sistema. Além disso, os resultados foram organizados de forma estruturada, com metadados que permitem rastrear sua origem, viabilizando a auditoria das detecções e sua reutilização em análises futuras.

4 Trabalhos Relacionados

Observabilidade: Alguns trabalhos focam no estudo de ferramentas de observabilidade. Janes *et al.* [7] apresentam uma visão geral das principais ferramentas de rastreamento *open-source* disponíveis no mercado por meio de uma comparação crítica. Os autores identificaram 30 ferramentas utilizando um protocolo de Revisão Sistemática Multivocal e realizaram sua classificação. As características analisadas 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 disponibilidade de API e suporte ao OTel.

Giamattei *et al.* [4] apresentam os resultados de uma revisão da literatura cinza conduzida para identificar, classificar e analisar ferramentas de monitoramento voltadas para DevOps e microsserviços. O estudo teve como objetivo fornecer uma visão geral das opções disponíveis e analisar esforços, desafios e direções em andamento para o desenvolvimento de soluções mais eficientes. Ao todo, 71 ferramentas foram analisadas, resultando em um mapeamento das principais características dessas soluções, incluindo benefícios, desvantagens, pressupostos de uso, tipos de informações coletadas, técnicas empregadas para coleta eficiente de dados e formas de apresentação dos resultados. Contudo, esses trabalhos abordam ferramentas gerais de observabilidade, sem foco na detecção de anti-padrões de observabilidade.

Anti-padrões: Alguns trabalhos concentram-se nos anti-padrões e na resolução dos problemas associados. Monsef *et al.* [13] 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.

Gomes *et al.* [6] apresentaram um catálogo de APs que mapeia práticas inadequadas nesse contexto. O catálogo propõe soluções para cada AP, porém apenas de forma teórica, sem implementar mecanismos automáticos para detecção dos problemas. De forma complementar, Min *et al.* [12] 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 pesquisa concentra-se em um único problema, sem contemplar a detecção automática de diferentes APs possíveis.

Nesse contexto, o Observa se posiciona como uma solução de apoio à detecção automática de APs de observabilidade a partir de dados de telemetria.

5 Conclusões e Trabalhos Futuros

Este artigo apresentou o Observa, um *framework* projetado para operacionalizar a detecção automática de APs de observabilidade em aplicações baseadas em microsserviços. Com uma arquitetura modular, extensível e tecnologicamente neutra, o Observa materializa os conceitos teóricos discutidos anteriormente, permitindo a identificação sistemática, reprodutível e baseada em evidências.

Como trabalhos futuros, pretende-se implementar um módulo de solução de APs, possibilitando, além da identificação automática, a mitigação de problemas identificados na observabilidade. Adicionalmente, propõe-se a evolução do *framework* com técnicas de correlação temporal e aprendizado de máquina, viabilizando inferência preditiva e detecção baseada em padrões comportamentais.

Disponibilidade de Artefatos

O código-fonte do Observa, manuais de instalação e uso, informações de licenciamento (*Apache License 2.0*), além de um vídeo explicativo sobre suas funcionalidades, podem ser encontrados no repositório do projeto, disponível em https://github.com/andersonalmada/detect-antipatterns e em https://doi.org/10.5281/zenodo.20387190.

Agradecimentos

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

Referências

[1] S. Chevre. 2024. 7 API Observability Anti-Patterns to Avoid. (2024). https://devops.com/7-api-observability-anti-patterns-to-avoid/ Accessed: March 10, 2025.

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

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

[4] L Giamattei, A Guerriero, R Pietrantuono, S Russo, I Malavolta, T Islam, M Dînga, A Koziolek, Snigdha Singh, Martin Armbruster, et al. 2023. Monitoring tools for DevOps and microservices: A systematic grey literature review. *Journal of Systems and Software* (2023), 111906.

[5] Francisco Gomes, Paulo Rego, and Fernando Trinta. 2024. Rumo a uma Taxonomia de Observabilidade para Aplicações Baseadas em Microsserviços. In *Anais do XXXVIII Simpósio Brasileiro de Engenharia de Software* (Curitiba/PR). SBC, Porto Alegre, RS, Brasil, 234–245. doi:10.5753/sbes.2024.3386

[6] Francisco Gomes, Paulo Rego, and Fernando Trinta. 2025. Rumo a um Catálogo de Anti-padrões de Observabilidade: Uma Revisão da Literatura. In *Anais do XXXIX Simpósio Brasileiro de Engenharia de Software* (Recife/PE). SBC, Porto Alegre, RS, Brasil, 104–114. doi:10.5753/sbes.2025.9779

[7] Andrea Janes, Xiaozhou Li, and Valentina Lenarduzzi. 2023. Open tracing tools: Overview and critical comparison. *Journal of Systems and Software* (2023), 111793.

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

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

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

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

[12] Shengjie Min, Lingfeng Guo, and Guifan Weng. 2023. Alert Fatigue Mitigation in Anomaly Detection Systems: A Comparative Study of Threshold Optimization and Alert Aggregation Strategies. *Journal of Computing Innovations and Applications* 1, 2 (2023), 59–73.

[13] Taravat Monsef and Mehrdad Ashtiani. 2025. Detecting Microservice's Architectural Anti-Pattern Indicators Using Graph Neural Networks. *Software: Practice and Experience* (2025).
[14] Sina Niedermaier, Falko Koetter, Andreas Freymann, and Stefan Wagner. 2019. On observability and monitoring of distributed systems–an industry interview study. In *Service-Oriented Computing: 17th International Conference, ICSOC 2019, Toulouse, France, October 28–31, 2019, Proceedings 17*. Springer, 36–52.
[15] Andy Singleton. 2016. The economics of microservices. *IEEE Cloud Computing* 3, 5 (2016), 16–20.
[16] Johannes Thönes. 2015. Microservices. *IEEE software* 32, 1 (2015), 116–116.
[17] S. Townshend. 2023. Bad Observability. (2023). https://squaredup.com/blog/bad-observability/ Accessed: March 10, 2025.
[18] Muhammad Usman, Simone Ferlin, Anna Brunstrom, and Javid Taheri. 2022. A survey on observability of distributed edge & container-based microservices. *IEEE Access* 10 (2022), 86904–86919.
[19] A. Villela. 2022. Observability Mythbusters: Observability Anti-Patterns. (2022). https://faun.pub/observability-mythbusters-observability-anti-patterns-2b3062405b54 Accessed: March 10, 2025.