Análise da Eficiência do Uso de Recursos de Máquina na Aplicação de Diferentes Práticas de Observabilidade em Ambientes com Hardware Limitado

Gabriel Augusto Souza Borges[1]

[1]Instituto de Ciências Exatas e Informática – PUC Minas
Rua Claudio Manoel, 1162 – Belo Horizonte – MG – Brasil

gabriel.borges@sga.pucminas.br

Abstract. *Observability is an essential practice for monitoring modern systems, but its implementation can generate high hardware resource costs in processing, memory, and storage. The central problem addressed is the inefficient use of these resources, which stems from a lack of guidelines to balance monitoring quality and computational cost. To investigate solutions, this work followed an experimental methodology that compared the efficiency of different observability practices in a controlled, resource-constrained hardware environment. An API was subjected to stress tests while resource consumption was monitored. The results demonstrated that the application of specific observability guidelines — simple sampling, adaptive sampling, and metric cardinality reduction — enabled a reduction in CPU usage by over 40% and decreased the average response time by approximately 45% compared to a non-optimized approach. Based on these findings, the study proposes a set of guidelines for optimizing data collection, offering a balance between resource use efficiency and observability quality in constrained hardware environments.*

Resumo. *A observabilidade é uma prática essencial para monitorar sistemas modernos, mas sua implementação pode gerar custos elevados em recursos de hardware, como processamento, memória e armazenamento. O problema central abordado é a ineficiência no uso desses recursos, que ocorre devido à ausência de diretrizes para equilibrar a qualidade do monitoramento e o custo computacional. Para investigar soluções, este trabalho seguiu uma metodologia experimental que comparou a eficiência de diferentes práticas de observabilidade em um ambiente controlado com hardware limitado. Uma API foi submetida a testes de estresse, enquanto o consumo de recursos era monitorado. Os resultados demonstraram que a aplicação de diretrizes específicas de observabilidade — amostragem simples, amostragem adaptativa e redução da cardinalidade de métricas — permitiu reduzir o uso da CPU em mais de 40% e diminuir o tempo médio de resposta em aproximadamente 45%, em comparação com uma abordagem sem otimizações. Com base nesses dados, o estudo propõe um conjunto de diretrizes para otimizar a coleta de dados, oferecendo um equilíbrio entre a eficiência no uso de recursos e a qualidade da observabilidade em ambientes com restrições de hardware.*

Bacharelado em Engenharia de *Software* - PUC Minas
Trabalho de Conclusão de Curso (TCC)

Orientador de conteúdo (TCC I): João Pedro Batisteli - jp.batisteli@hotmail.com
Orientador de conteúdo (TCC I): Joana Souza - joanasouza@pucminas.br
Orientador de conteúdo (TCC I): Leonardo Vilela - leonardocardoso@pucminas.br
Orientador acadêmico (TCC I): Cleiton Tavares - cleitontavares@pucminas.br
Orientador do TCC II: Danilo Maia - danilomaia@pucminas.br

Belo Horizonte, 23 de Novembro de 2025.

1. Introdução

Para garantir a confiabilidade e o desempenho de aplicações modernas, o conceito de Observabilidade surge como um conjunto de práticas que visa compreender o estado interno de um sistema por meio da análise de seus dados externos [Usman et al. 2022]. Em sistemas distribuídos modernos, onde a complexidade é alta e falhas podem ser difíceis de diagnosticar, a observabilidade torna-se essencial para detecção e resolução ágil de problemas [Usman et al. 2022]. Seus principais pilares são métricas (dados quantitativos sobre desempenho), *logs* (registros de eventos temporais) e *traces* (rastreamento de requisições entre serviços) [Rodrigues et al. 2023].

Entretanto, a implantação de diretrizes de observabilidade, isto é, regras práticas que orientam como coletar, processar e armazenar dados de monitoramento, sem critérios de otimização, alta cardinalidade de métricas, registro completo de *logs* e o rastreamento de todas requisições, em arquiteturas de alta observabilidade pode ser custosa em termos de infraestrutura e tempo humano. Essas práticas exigem maior capacidade de armazenamento, enquanto o processamento desses dados em tempo real pode gerar custos computacionais adicionais [Madupati 2023].

Embora existam trabalhos sobre o impacto negativo nos recursos de máquina durante o uso de ferramentas de observabilidade como o ElasticSearch[1] [Costa and Araújo 2024], **o problema abordado neste trabalho é a ineficiência no uso de recursos de *hardware* causada pela ausência de diretrizes para equilibrar qualidade do monitoramento e custo computacional em práticas de observabilidade.**

Resolver este problema é relevante, pois a escolha inadequada de práticas de observabilidade como a captura excessiva de dados sem restrições pode causar problemas no desempenho do sistema [Picoreti et al. 2018] aumentando custos de infraestrutura e reduzindo a escalabilidade dos *softwares*. Com os gargalos gerados, os sistemas de computador conseguem lidar com menos tráfego novo. Outro aspecto importante é que em cenários de ambientes com restrições de *hardware*, a otimização desses recursos é crítica, sabendo que a performance de ferramentas de observabilidade pode ser diretamente impactada pela saturação dos recursos computacionais das máquinas nas quais a aplicação é executada [Gomes et al. 2024a].

Portanto, **o objetivo geral desse estudo é avaliar como diferentes tratativas de observabilidade afetam a eficiência no uso de recursos de *hardware* em ambientes de produção**. Como objetivos específicos, têm-se: (i) Quantificar o impacto de diferentes

[1] `https://www.elastic.co/elasticsearch`

estratégias de observabilidade no consumo de CPU, memória e latência; (ii) Investigar a relação entre o custo computacional da coleta de dados e o valor em *insights* gerados para a Engenharia de Software; e (iii) Propor diretrizes para otimizar a coleta de dados, como estratégias de amostragem adaptativa. Esses objetivos específicos serão analisados por meio da saturação de um *endpoint* da API (API, do inglês *Application Programming Interface*) construída para esse estudo.

As expectativas de desempenho foram baseadas em evidências preliminares e na literatura existente. [Madupati 2023] observou que a instrumentação de observabilidade pode gerar sobrecarga de aproximadamente um quarto na CPU, sugerindo que estratégias de otimização poderiam reverter significativamente esse impacto. Paralelamente, [Costa and Araújo 2024] alcançaram redução de 80% no armazenamento de dados de observabilidade sem comprometer a qualidade. Neste contexto, espera-se que as estratégias propostas resultem em redução de pelo menos 30% no uso de CPU, 20% no consumo de memória e 15% no tempo médio de resposta em comparação com a abordagem sem otimizações. A amostragem adaptativa, em particular, é esperada como a estratégia mais eficaz por acumular todas as otimizações propostas.

As próximas seções do artigo estão divididas da seguinte forma: Na Seção 2, é apresentado a fundamentação teórica. Na Seção 3, são mostrados os trabalhos relacionados. A Seção 4 traz a metodologia para a realização da pesquisa. Na 5, são apresentados os resultados quantitativos, enquanto na 6, são discutidos os resultados. Na seção 7, são apresentadas as ameaças à validade. Por fim, na seção 8, são expostos conclusões, trabalhos futuros e o pacote de replicação.

2. Fundamentação Teórica

A observabilidade, termo originado da teoria de controle refere-se à capacidade de entender o comportamento de um sistema por meio dos dados que ele gera, facilitando o monitoramento e a análise em tempo de execução [Kalman 1959]. Na computação, seu principal objetivo é reduzir o tempo necessário para diagnosticar problemas. Essa capacidade é sustentada por três pilares fundamentais: *logs*, métricas e *traces* [Rodrigues et al. 2023].

O armazenamento regular de métricas, prática conhecida na literatura técnica como monitoramento, é um dos mecanismos fundamentais para detectar anomalias em sistemas computacionais [Rodrigues et al. 2023]. Esse processo gera séries temporais que, ao serem analisadas, revelam padrões e tendências no comportamento do sistema. Essas métricas são organizadas em diferentes níveis ou categorias, cada um responsável por monitorar aspectos específicos, como desempenho, comportamento e integridade do sistema [Gomes et al. 2024c].

Logs são registros imutáveis de eventos, podendo ser estruturados ou não, coletados de diversas fontes, como aplicações e sistemas operacionais e possuem um nível de detalhamento maior que as métricas [Gomes et al. 2024a]. Os *logs* são coletados diretamente de processos em execução. No entanto, ao contrário das métricas, que são registradas de forma periódica, os *logs* são gerados de maneira não programada, pois dependem da ocorrência de eventos específicos [Rodrigues et al. 2023]. A estrutura dos *logs* pode ser organizados (em formatos como JSON (JSON, do inglês *Javascript Object Notation*)) ou não estruturados (como linhas de texto livre).

Enquanto métricas e *logs* oferecem dados isolados sobre cada serviço, os *traces*

mapeiam o caminho das requisições em um ambiente de *software* [Picoreti et al. 2018]. Sejam esses caminhos internos, passando por classes do sistema ou rastros de outros serviços externos. Assim, os *traces* mostram claramente as relações causais entre requisições, mapeando as complexas interações típicas dos sistemas. Essa capacidade é fundamental para rastrear em tempo de execução como um evento se propaga por toda a arquitetura [Rodrigues et al. 2023].

Além do objetivo principal de reduzir tempo necessário para diagnosticar falhas no sistema sustentado pelo uso de métricas, *logs* e *traces*, outras vantagens podem ser providas pela observabilidade como a ampliação da visibilidade do sistema, tornando mais transparente seu funcionamento interno. Ela vai além do monitoramento superficial, revelando desde métricas de desempenho até padrões operacionais otimizados, permitindo às equipes não apenas identificar problemas [Gomes et al. 2024c].

Além disso, a observabilidade fornece dados estratégicos sobre o comportamento do sistema em produção, permitindo aprimorar continuamente o ciclo de desenvolvimento. Esses *insights* operacionais em tempo real orientam decisões técnicas mais embasadas, desde ajustes pontuais até evoluções arquiteturais mais profundas, garantindo que o sistema se adapte às necessidades reais de uso [Gomes et al. 2024c].

Por fim, a observabilidade proporciona ao cliente (ponta final do *software*) uma melhor experiência com o sistema em questão. Uma boa observabilidade tende a entregar performance superior e maior estabilidade, resultando em uma experiência mais fluida e satisfatória para os usuários finais. Essa excelência operacional não apenas eleva a satisfação do cliente, mas também impulsiona diretamente os resultados de negócio, fortalecendo a reputação da aplicação e aumentando sua taxa de adoção [Gomes et al. 2024c].

3. Trabalhos Relacionados

Os trabalhos relacionados discutidos nesta seção envolvem a análise da experimentação de diferentes práticas e ferramentas de observabilidade em diferentes tipos de infraestutura de *hardware*.

Li et al., (2021) propõem o TraceRCA, uma abordagem não supervisionada para localização de causas-raiz em sistemas de microsserviços por meio da análise de *traces*. O método utiliza três etapas principais: detecção de anomalias em *traces*, mineração de conjuntos suspeitos de microsserviços e ranqueamento baseado em escores de suspeição. Os autores demonstram que o TraceRCA supera abordagens anteriores em mais de quarenta e quatro porcento na detecção do serviço causador da falha, validando sua eficácia em sistemas reais. Embora o foco do artigo seja a acurácia na diagnose de falhas, ele destaca a importância da eficiência computacional ao utilizar técnicas leves (como seleção adaptativa de métricas) para reduzir o espaço de busca. Essa otimização é relevante para o contexto deste trabalho, pois alinha-se ao objetivo de definir estratégias de extração de informação para a observabilidade de sistemas.

Madupati, (2023) explora o uso de observabilidade com microserviços utilizando uma abordagem empírica, aplicando as ferramentas Prometheus para coleta de métricas, o Grafana para visualização e o Jaeger para rastreamento distribuído em sistemas. O autor defende que essa cadeia de ferramentas melhora a detecção de falhas e o monitoramento, embora seu trabalho não apresente métricas quantitativas que dimensionem essa melhoria

(como a redução no tempo médio para resolver incidentes). Por outro lado, o estudo é mais preciso ao quantificar um dos principais desafios: a instrumentação pode gerar uma sobrecarga de aproximadamente 25% no uso da CPU. Essa análise dos custos computacionais oferece uma referência valiosa para a presente investigação sobre o impacto da observabilidade no consumo de recursos.

Rodrigues et al., (2023) fazem uma avaliação da efetividade do uso combinado de dois pilares da observabilidade, métricas e *logs*, para diagnosticar anomalias em um sistema 5G. A metodologia é experimental, baseada na implantação de um ambiente de testes que emula uma rede 5G completa em um *cluster Kubernets* além da condução de testes de conectividade e desempenho sob subdimensionamento de CPU, comparando a detecção de anomalias. Os autores demonstram como a análise combinada de métricas e *logs* pode melhorar a observabilidade de sistemas 5G e ajudar a antecipar a ocorrência de falhas. Já que, no teste de desempenho, os *logs* emitiram alertas anômalos repetidos antes que a métrica de *throughput* indicasse a queda, permitindo uma detecção antecipada em vários segundos. Diante disso, esse estudo se assemelha por também trabalhar com práticas semelhantes que focam em manipulação de métricas. Entretanto, Rodrigues et al, (2023) abordam muito mais a facilidade na detecção de falhas com a aplicação de boas técnicas de observabilidade que na análise do uso de recursos de *hardware*.

Francisco e Vinicius, (2024) analisam o uso da observabilidade para avaliar o desempenho de arquiteturas monolíticas e baseadas em microsserviços, empregando a solução *OpenTelemetry* (OTel) por meio da migração de uma aplicação de referência baseada em microsserviços para um sistema monolítico. O resultado principal indica que o OTel pode impactar negativamente o desempenho da aplicação em ambientes com recursos computacionais limitados. Em que houve uma redução de 0,44% no número de requisições atendidas e um aumento de 0,48% no tempo de resposta para cada solicitação. Este trabalho constrói um ambiente semelhante, uma vez que visa-se limitar propositalmente os recursos disponíveis para uma análise que simula ambientes restritos. Porém o foco não é em restringir o *hardware*, isso faz parte da metodologia. Este estudo foca na manipulação de diretrizes de observabilidade que visam melhorar a performance geral da aplicação.

Breno e Aletéia, (2024) abordam a sobrecarga computacional causada por ferramentas de observabilidade em ambientes de Computação em Névoa, utilizando um cenário prático com o aplicativo *Mobile* IoT-RoadBot. O estudo avalia o impacto de ferramentas de código aberto, como Prometheus, ELK Stack e *OpenTelemetry*, no consumo de CPU, memória e armazenamento em dispositivos IoT e nós de Névoa. Os autores propõem estratégias para reduzir o volume de dados de observabilidade, alcançando uma diminuição de oitenta por cento no armazenamento sem comprometer a qualidade dos dados. Este trabalho relacionado é relevante para o presente estudo, pois quantifica o custo computacional de práticas de observabilidade em um ambiente com restrições de recursos, similar aos cenários abordados aqui. Contudo, os autores focam em otimizar o volume de dados em um contexto específico de Névoa, este estudo explora uma análise em ambiente local com recursos de *hardware* limitados.

4. Materiais e Métodos

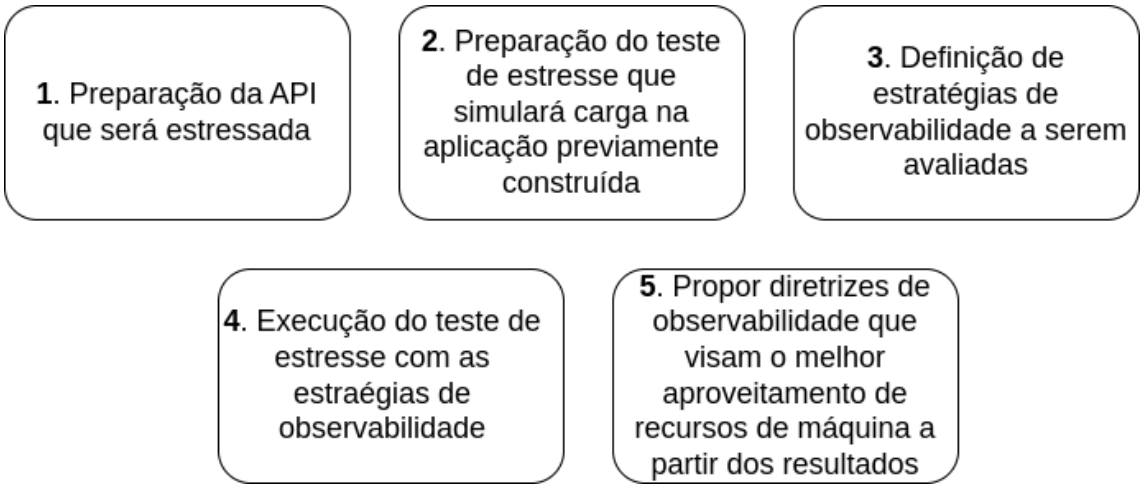

Figura 1. Metodologia de cinco etapas para o desenvolvimento deste projeto.

Este trabalho adota uma abordagem experimental quantitativa e aplicada, com o objetivo de investigar, de forma explicativa, o impacto de diferentes práticas de observabilidade no consumo de recursos de *hardware* através de métricas como o tempo de execução de um *endpoint* da API construída para esse estudo e o consumo computacional das ferramentas. A metodologia foi planejada para traduzir informações em números que podem ser analisados e classificados [Santana et al. 2025] e garantir a obtenção de dados confiáveis, permitindo comparações entre configurações de monitoramento em sistemas de computador. Compreende-se como procedimentos, cinco etapas que serão detalhadas na subseção 4.2.

4.1. Instrumentos Utilizados

A execução da pesquisa utilizará um conjunto integrado de ferramentas de código aberto, selecionadas por sua adequação aos objetivos do estudo. Para a geração de carga de trabalho, a ferramenta Grafana k6 foi escolhida devido à sua arquitetura leve e capacidade de integração nativa com *pipelines* de métricas, permitindo não apenas simular tráfego realista mas também exportar dados de desempenho diretamente para o Prometheus. No k6 será possível analisar a quantidade de requisições feitas, a média de tempo de cada requisição e o p95 de tempo de cada uma.

Complementarmente, a interface do Prometheus será utilizada para analisar graficamente os usos de recursos de *hardware* (CPU e memória) no tempo de execução de cada estratégia de observabilidade. A combinação entre Prometheus e Grafana k6 forma o núcleo da solução de monitoramento, seguindo padrões consolidados do mercado para coleta, armazenamento e visualização de métricas quantitativas em tempo real.

4.2. Procedimentos

A Figura 1 resume as principais etapas, desde a preparação da API que será estressada até a proposição de diretrizes de observabilidade para aproveitamento de recursos de máquina. A seguir, é detalhada cada uma delas.

Construção da API que será estressada

Inicialmente, será construída a API orquestrada por um ambiente com restrições de recurso pelo Docker. Terá um endpoint POST que fará consultas ao banco de dados com duas tabelas relacionais pré montadas antes de fazer inserções em uma terceira tabela e obterá dados no formato JSON. Medir o impacto de operações de escrita e leitura de banco de dados combinadas em um *endpoint*, é uma boa forma para simular o comportamento de aplicações *web* com cargas reais [Guégain et al. 2024].

Configuração do teste de estresse

Nessa etapa, será implementado um teste de estresse para simular condições realistas de carga no sistema. Para isso, utilizaremos a ferramenta k6 para gerar requisições HTTP (HTTP, do inglês *Hypertext Transfer Protocol*) automatizadas, configuradas com parâmetros específicos. Os usuários virtuais que disparam as requisições da API irão variar entre 10 e 200. O teste de estresse será executado para cada estratégia de observabilidade. Serão feitos 30 execuções do teste de estresse para cada estratégia e cada uma delas terá o tempo de vida de 120 segundos. Serão definidas cargas diferentes ao longo de cada período do teste de estresse com a finalidade de simular o volume de tráfego variável de um sistema real.

Implementação das Estratégias de Observabilidade

Na segunda etapa, serão avaliadas três abordagens conceituais de observabilidade, independentes de ferramentas específicas, na seguinte ordem:

- **Redução da cardinalidade de métricas**: Remoção de métricas que possuem um alto número de valores possíveis.
- **Amostragem**: Implementação de um mecanismo que filtra, com um parâmetro fixo, a quantidade de métricas que serão coletadas. Além de aumentar o intervalo de coleta das métricas pelo Prometheus.
- **Amostragem Adaptativa**: Implementação de um sistema dinâmico que ajusta a granularidade do monitoramento conforme a carga do sistema, reduzindo a coleta de dados durante picos de demanda e intensificando-a em condições normais. Aqui também será mantido o aumento no intervalo de métricas coletadas pelo Prometheus.

Vale ressaltar que a cada execução do teste de estresse, o banco de dados estará previamente populado apenas com os dados de configuração inicial.

Coleta e Análise de Dados

A terceira etapa consistirá na execução dos testes de carga para cada cenário de observabilidade. Serão registrados: tempo médio de resposta e p95 do tempo de resposta, utilização de CPU e memória do sistema. Os dados de CPU e memória serão armazenados e analisados no Prometheus enquanto as métricas relacionadas à quantidade e tempo das requisições serão analisados no k6, no output de finalização do teste de estresse.

Formulação de Diretrizes

Com base na análise comparativa dos resultados, serão elaboradas recomendações práticas de observabilidade que visam aproveitar o consumo de recursos de máquina.

4.3. Ambiente Experimental

O estudo será conduzido em ambiente local isolado utilizando contêineres Docker em uma máquina hospedeira que executa qualquer distribuição Linux com Docker Compose para orquestração dos serviços. O ambiente conteinerizado usará uma imagem do Alpine e terá limitações de recursos pré-definidas pelo Docker. Será destinado à API 1 cpu e 128MB (MB, do inglês Mega Bytes) de memória. Enquanto o banco de dados da aplicação possuirá 1 cpu e 512MB de memória.

A API sob teste, será desenvolvida em PHP utilizando o *framework* HyperF, com banco de dados local MySQL. Todos os serviços serão configurados através de um arquivo docker-compose.yml único, com limites de recursos por contêiner, redes *bridge* isoladas e volumes para persistência dos dados coletados.

O presente estudo utilizará recursos limitados para que seja possível aplicá-lo em vários tipos de máquinas que têm como sistema operacional Linux. Além de simular um ambiente com pouco recurso de *hardware* disponível com a finalidade de extrair o máximo possível de desempenho do ambiente isolado.

4.4. Métricas de Avaliação

A seleção das métricas seguiu a abordagem GQM (Goal-Question-Metric) [Basili 1992], que estabelece uma relação sistemática entre objetivos, questões de pesquisa e métricas de avaliação. Neste estudo, o objetivo era avaliar a eficiência de diferentes práticas de observabilidade no consumo de recursos de *hardware*. As questões de pesquisa investigaram: (i) Qual estratégia oferece melhor desempenho em tempo de resposta? (ii) Como cada abordagem impacta o consumo de CPU e memória? (iii) Quais são os *trade-offs* entre qualidade de monitoramento e custo computacional?

Serão coletadas e analisadas quatro métricas principais. Duas delas, o tempo de resposta médio e latência do percentil da API, medido em milissegundos, servirão como indicadores diretos do impacto na experiência do usuário final e performance percebida. Ademais, essas métricas foram escolhidas por serem entregues pela ferramenta de teste de estresse Grafana K6, usada neste trabalho. Já os dados de memória e CPU foram coletados por meio do pacote de observabilidade *hyperf/metric* do *framework* HyperF utilizado para a construção da API. O consumo de recursos de *hardware* será quantificado através do uso de CPU e memória, monitorado pelo Prometheus, que captura o *overhead* computacional introduzido por cada abordagem de coleta de dados.

5. Resultados Quantitativos

Esta seção apresenta os resultados obtidos a partir da análise dos quatro cenários de configuração: Sem Diretrizes, Redução da Cardinalidade, Amostragem Simples e Amostragem Adaptativa. Para visualizar e comparar a distribuição das métricas de desempenho entre os cenários, optou-se pela utilização de gráficos *box plot*. O *box plot* é uma ferramenta robusta para detecção de *outliers* baseada em quartis e na amplitude interquartil

(IQR). Por utilizar medidas resistentes de tendência central (mediana) e dispersão (IQR), é menos sensível a valores extremos, identificando eficazmente observações atípicas. [Mazarei et al. 2025]

A Figura 2 apresenta a distribuição das médias dos tempos de resposta das requisições HTTP entre os quatro cenários. Os cenários de Amostragem Adaptativa e Amostragem Simples apresentaram medianas de $8,645$ ms (milissegundos) e $9,045$ ms, respectivamente, enquanto Redução da Cardinalidade e Sem Diretrizes obtiveram medianas de $14,55$ ms e $14,65$ ms. Em relação à dispersão, o cenário Sem Diretrizes apresentou menor variabilidade, com intervalo interquartil de $0,59$ ms ($Q1 = 14,37$; $Q3 = 14,96$), seguido pela Redução da Cardinalidade com $0,8$ ms ($Q1 = 14,11$; $Q3 = 14,91$). Os cenários de amostragem mostraram maior dispersão, com intervalos interquartis de $0,59$ ms para Amostragem Simples ($Q1 = 8,63$; $Q3 = 9,22$) e $1,59$ ms para Amostragem Adaptativa ($Q1 = 7,86$; $Q3 = 9,45$).

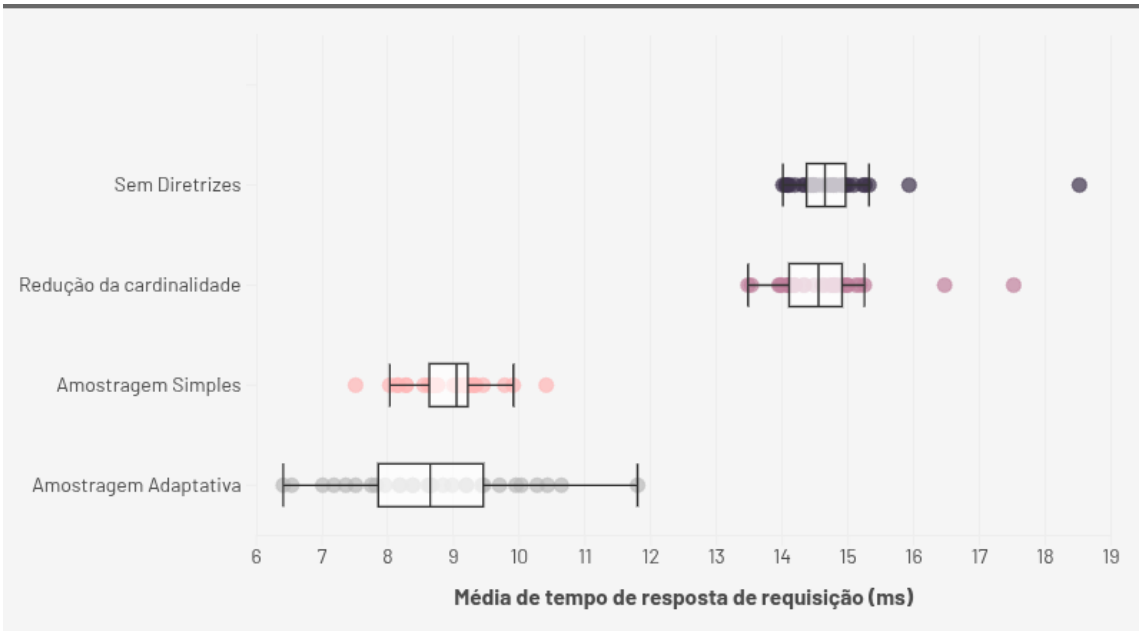

Figura 2. Médias nos tempos de resposta das requisições HTTP.

A Figura 3 mostra a distribuição do percentil 95 ($P95$) dos tempos de resposta. O cenário Redução de cardinalidade apresentou a menor mediana ($17,29$ ms), seguido pelo Sem Diretrizes ($17,56$ ms), Amostragem Simples ($24,59$ ms) e Amostragem Adaptativa ($23,59$ ms). A dispersão foi mais acentuada no cenário de Amostragem Simples, com intervalo interquartil de $10,78$ ms ($Q1 = 19,59$; $Q3 = 30,37$). Observa-se a presença de *outliers* em todos os grupos com exceção da Amostragem Simples, com maior concentração no cenário de Redução de Cardinalidade.

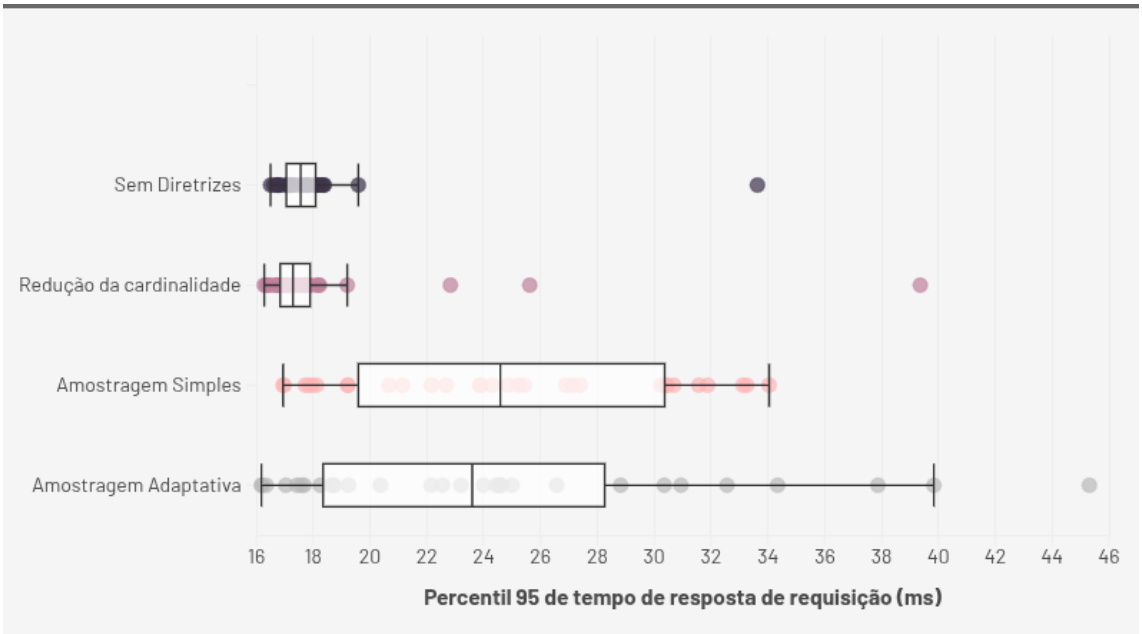

Figura 3. Percentil 95 dos tempos de resposta das requisições HTTP.

A Figura 4 apresenta a distribuição do uso de memória entre os cenários. As medianas variaram de $6,10$ MB (Amostragem Adaptativa) a $6,14$ MB (Sem Diretrizes). Os cenários de amostragem mostraram maior variabilidade, com intervalos interquartis de $0,04$ MB para Amostragem Adaptativa ($Q1 = 6,06$; $Q3 = 6,10$), $0,04$ MB para Amostragem Simples ($Q1 = 6,06$; $Q3 = 6,10$) e $0,04$ MB para Redução de Cardinalidade ($Q1 = 6,13$; $Q3 = 6,17$). Já o cenário Sem Diretrizes apresentou menor dispersão $0,3$ MB ($Q1 = 6,13$; $Q3 = 6,16$) mas, com mais *outliers* (4).

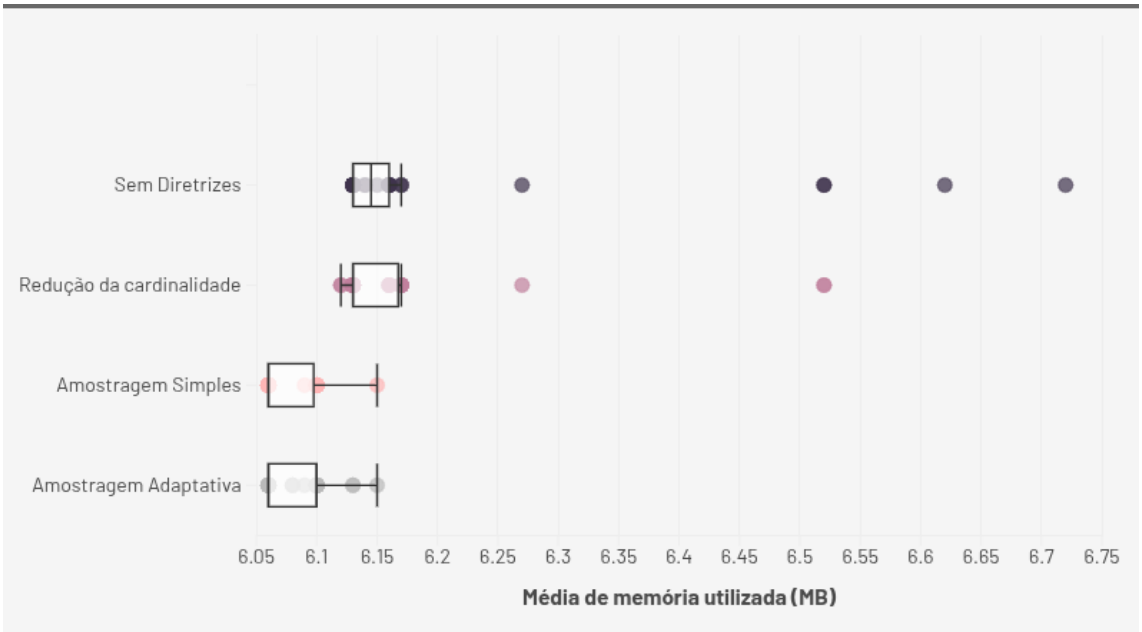

Figura 4. Médias de memória utilizada pelos diferentes cenários.

A Figura 5 exibe a distribuição do uso de CPU medido em microssegundos por segundo (μs/s). As medianas observadas foram de 44744 μs/s (Sem Diretrizes), 47234 μs/s (Redução da Cardinalidade), 25638 μs/s (Amostragem Simples) e 27805 μs/s (Amostragem Adaptativa). Quanto à dispersão, o cenário Amostragem Simples apresentou o maior intervalo interquartil ($7814,75$ μs/s), enquanto a Amostragem Adaptativa mostrou a menor variabilidade (6595 μs/s). Nota-se a presença de valores *outliers* apenas no cenário de Amostragem Simples.

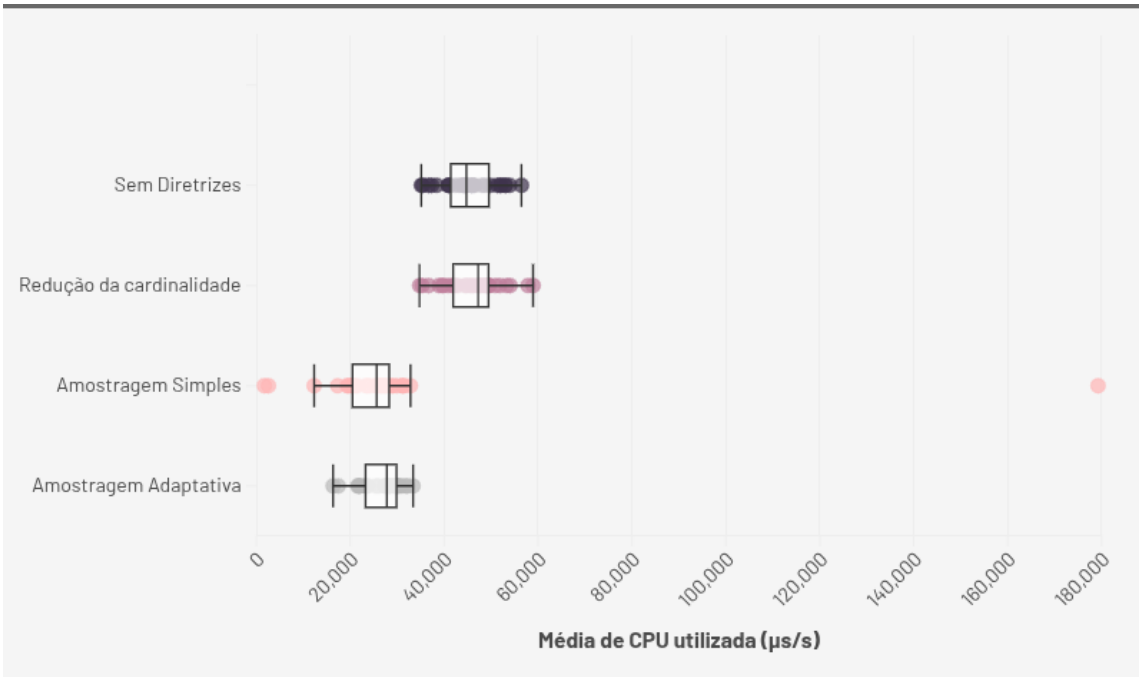

Figura 5. Médias de CPU utilizada pelos diferentes cenários.

6. Discussão dos Resultados

Os resultados quantitativos apresentados na seção anterior revelam um padrão claro de melhoria progressiva à medida que estratégias de otimização de observabilidade são acumuladas. Como primeiro achado, identificou-se que que a combinação de múltiplas técnicas produz ganhos superiores aos alcançados por abordagens isoladas, oferecendo *insights* valiosos para a implementação de observabilidade em ambientes com recursos limitados.

Para facilitar a comparação direta entre os cenários, a Tabela 1 consolida os valores medianos das principais métricas de desempenho. Esta visão sintetizada permite identificar rapidamente os trade-offs e benefícios de cada abordagem.

Tabela 1. Resumo Comparativo das Métricas de Desempenho por Cenário

Métrica	Sem Diretrizes	Red. Card.	Amost. Simples	Amost. Adapt.
Tempo de Resposta Médio (ms)	14,65	14,55	9,045	8,645
Percentil 95 do Tempo de Resposta (ms)	17,56	17,29	24,59	23,59
Uso de Memória (MB)	6,14	6,14	6,10	6,10
Uso de CPU (μs/s)	44.744	47.234	25.638	27.805

6.1. Impacto nas Métricas de Desempenho e Latência

A evolução desde o cenário Sem Diretrizes até a Amostragem Adaptativa mostra uma melhoria consistente e substancial no desempenho do tempo de resposta, conforme ilustrado pela Figura 2. O tempo médio, que começou em 14,65 ms no cenário base, foi reduzido para aproximadamente 8–9 ms nas estratégias mais avançadas – representando uma melhoria de cerca de 45%. Esse ganho progressivo indica que cada camada de otimização contribui para a eficiência geral do sistema, com a combinação de redução de cardinalidade, amostragem básica e ajuste dinâmico demonstrando ser a abordagem mais eficaz para a métrica de tempo médio.

Entretanto, a análise da latência do percentil 95 (P95), apresentada na Figura 3, revela um *trade-off* importante. A Redução de Cardinalidade isolada obteve a melhor mediana de P95 (17,29 ms), sugerindo que esta técnica é eficaz para aplicações que exigem baixa latência consistente. No entanto, à medida que as estratégias de amostragem são incorporadas, observa-se um aumento na variabilidade e nas medianas de P95 (atingindo aproximadamente 23–24 ms). Este comportamento indica que, enquanto a amostragem beneficia o usuário médio, ela pode introduzir uma variabilidade maior que impacta negativamente uma pequena fração das requisições mais lentas. A escolha da estratégia deve, portanto, considerar se o objetivo é otimizar a experiência média ou garantir a consistência da latência para todos os usuários.

Portanto, além de referenciar o objetivo específico i) a resposta para a primeira questão pesquisa é dada. Qual estratégia oferece melhor desempenho em tempo de resposta? Amostragem Adaptativa.

6.2. Eficiência no Uso de Recursos de Sistema

O consumo de CPU, mostrado na Figura 5, apresenta um padrão revelador. Enquanto a aplicação isolada da Redução de Cardinalidade mostrou um consumo ligeiramente superior ao cenário sem otimizações (47.234 μs/s vs 44.744 μs/s), a introdução da Amostragem Simples sobre esta base reduziu drasticamente o uso para 25.638 μs/s – uma economia de aproximadamente 43%. A evolução para a Amostragem Adaptativa manteve ganhos significativos (27.805 μs/s), demonstrando que a combinação estratégica de técnicas é essencial para otimizar recursos computacionais.

Esse resultado é crucial, pois sugere que a Redução de Cardinalidade, quando aplicada isoladamente, pode introduzir *overheads* de processamento que superam seus benefícios. No entanto, quando combinada com estratégias de amostragem, ela contribui para um sistema mais eficiente no uso do processador. Esta descoberta corrobora

observações de Madupati (2023), em que o autor evidencia o aumento de um quarto na sobrecarga da CPU em cenários de coleta detalhada de informações.

Em contraste, o consumo de memória, ilustrado na Figura 4, manteve-se notavelmente estável em todos os cenários—com variações inferiores a 0,1 MB. Este comportamento reflete o padrão de operações da aplicação testada, caracterizado por operações de banco de dados de baixo impacto em memória. O *endpoint* da API implementa uma consulta SELECT que retorna um conjunto de dados pequeno e uma operação INSERT com um *payload* mínimo, não exercitando operações que tipicamente consumiriam grandes volumes de memória, como junções complexas ou agregações de dados massivos.

A análise acima responde a segunda questão pesquisa: Como cada abordagem impacta o consumo de CPU e memória? Além de referenciar o objetivo específico ii).

6.3. Implicações Práticas para Ambientes Restritos

Para contextos práticos como *edge computing* e dispositivos IoT, similares aos investigados por Breno e Aletéia (2024), a abordagem cumulativa demonstra especial valor. A progressão desde técnicas básicas até a Amostragem Adaptativa mostra que é possível alcançar ganhos de eficiência substanciais—como a redução de 45% no tempo de resposta e 43% no uso de CPU—sem comprometer completamente a capacidade de monitoramento. A capacidade de ajustar dinamicamente o volume de dados telemetria (como na Amostragem Adaptativa) é particularmente vantajosa nesses ambientes, onde as condições da rede e a disponibilidade de recursos podem flutuar.

6.4. Considerações sobre Complexidade, Manutenção e *Trade-offs*

Vale ressaltar que a implementação cumulativa dessas estratégias de observabilidade introduz complexidade adicional. Cada camada exige configuração específica, como definir taxas de amostragem adequadas e limiares para dinâmicas adaptativas, além de manutenção contínua para calibrar esses parâmetros. A Amostragem Adaptativa, em particular, requer uma lógica temporal mais sofisticada, utilizando múltiplas taxas baseadas em intervalos pré-definidos. No entanto, os ganhos de performance obtidos justificam essa complexidade incremental em ambientes onde a eficiência de recursos é crítica. O desafio se desloca, portanto, da simples coleta de dados para o projeto inteligente de quando e o que coletar, otimizando o custo-benefício da observabilidade.

Em resposta a terceira questão pesquisa (Quais são os *trade-offs* entre qualidade de monitoramento e custo computacional?), os resultados demonstram um *trade-off* entre a profundidade do monitoramento e a eficiência computacional. A coleta completa de dados, sem otimizações, oferece máxima visibilidade do sistema para debugging, porém consome cerca de 45% a mais de CPU e aumenta o tempo médio de resposta na mesma proporção, representando um overhead significativo.

Em contrapartida, estratégias de otimização como a amostragem e a redução de cardinalidade reduzem drasticamente o custo computacional, mas introduzem limitações na qualidade do monitoramento. A amostragem pode omitir eventos raros críticos, enquanto a redução de cardinalidade simplifica excessivamente as métricas, comprometendo a detecção de anomalias específicas. A escolha ideal depende, portanto, do balanceamento entre a necessidade de detalhamento para análise e a disponibilidade de recursos no ambiente-alvo.

7. Ameças à Validade

Estudos experimentais em Engenharia de Software estão propícios a diferentes ameaças a validade, que necessitam ser postas para deixar a as conclusões confiáveis. Esta seção descreve as principais ameaças à validade identificadas na pesquisa, bem como as estratégias empregadas para mitigá-las.

Validade Interna

No que se refere à validade interna, que avalia se os resultados podem ser efetivamente atribuídos às variáveis manipuladas, uma ameaça considerada foi a possível interferência de fatores diversos nos resultados observados. Como variações inerentes ao ambiente virtualizado e alocação incontrolada de recursos. Para mitigar este risco, o experimento foi conduzido em um ambiente controlado e reproduzível, utilizando contêineres Docker com limites estritos de CPU e memória. Adicionalmente, o banco de dados era limpo antes de cada execução de teste, e os dados utilizados no corpo das requisições eram gerados aleatoriamente, assegurando que cada teste partisse de uma condição igualitária e evitando viés proveniente de dados ou padrões de carga previsíveis.

Validade Externa

A validade externa, diz respeito a a generalização dos resultados para outros contextos. Neste estudo, a principal limitação reside no foco em uma API *backend* típica de ambientes *web*, e executada em uma distribuição Linux. Consequentemente, os resultados podem não ser diretamente generalizáveis para sistemas com características radicalmente distintas, como aplicações de IoT e sistemas operacionais. No entanto, a metodologia adotada, que incluiu a simulação de um ambiente com recursos limitados e a aplicação de cargas de trabalho variáveis e aleatórias, buscou criar um cenário representativo de restrições comuns em ambientes de produção de baixo recurso, aumentando a potencial aplicabilidade dos resultados em contextos análogos.

Validade de Construção

Quanto à validade de construção, que verifica se as métricas e ferramentas usadas realmente medem o que se propuseram a medir, uma preocupação foi a forma como colocamos em prática as diretrizes de observabilidade. Conceitos como ”Amostragem Adaptativa”foram implementados usando regras específicas criadas para esse trabalho, e outras formas de implementar essas mesmas ideias poderiam levar a resultados diferentes. Para aumentar a confiança nas medições, utilizamos ferramentas consolidadas no mercado (Grafana k6, Prometheus e a biblioteca hyperf/metric), garantindo que a coleta de dados sobre consumo de recursos e desempenho fosse feita de maneira padronizada e automática em todos os cenários testados. Assim, assegura-se que quaisquer diferenças observadas nos resultados pudessem ser atribuídas às estratégias de observabilidade em teste, e não a variações na forma de medição.

Validade de Conclusão

A validade de conclusão, se refere à robustez das inferências estatísticas extraídas dos dados. Reconhece-se que usar apenas gráficos de *box plot*, embora eficaz para identificar tendências centrais e valores atípicos, apresenta uma limitação importante: esse método não comprova se as diferenças observadas entre os cenários são estatisticamente significativas ou apenas fruto de variações aleatórias. Para um estudo futuro, seria recomendável complementar essa análise visual com testes estatísticos que poderiam quantificar o nível de confiança nas diferenças encontradas.

8. Conclusões e Trabalhos Futuros

Este trabalho avaliou o impacto de quatro cenários de observabilidade de forma acumulativa (Sem Diretrizes, Redução da Cardinalidade, Amostragem Simples e Amostragem Adaptativa) em uma API com *hardware* controlado. A aplicação com cada diretriz foi analisada encima de quatro métricas: Média de tempo de resposta de requisições HTTP (ms), percentil 95 dos tempos de resposta das requisições HTTP (ms), Média de memória utilizada (MB) e Média de CPU utilizada (μs/s).

Ao avaliar e comparar as métricas, os resultados demonstraram que a aplicação cumulativa de estratégias de observabilidade produz ganhos de desempenho superiores à ausência de práticas otimizadas. Especificamente, o cenário de Amostragem Adaptativa alcançou a maior redução no tempo médio de resposta (aproximadamente 45%) e no consumo de CPU (cerca de 43%) em comparação com o cenário base Sem Diretrizes. Esses ganhos substanciais indicam que a combinação de técnicas é altamente eficaz para otimizar a eficiência operacional em ambientes com recursos limitados.

Além disso, também quanto ao uso de recursos de *hardware*, o consumo de memória manteve-se notavelmente estável em todos os cenários, com variações inferiores a 0,1 MB, refletindo o perfil de baixo impacto das operações da API testada. Em contraste, o consumo de CPU mostrou-se sensível às estratégias adotadas, com a Redução de Cardinalidade isolada introduzindo um pequeno *overhead* adicional que foi drasticamente mitigado pela introdução das amostragens.

Os resultados obtidos ressaltam a importância de um projeto inteligente de observabilidade, no qual o desafio se desloca da simples coleta de dados para a decisão estratégica de quando e o que coletar. A complexidade incremental introduzida pelas estratégias cumulativas —especialmente pela Amostragem Adaptativa— mostra-se justificável pelos ganhos de eficiência alcançados, mas exige configuração e manutenção contínuas para calibrar parâmetros como taxas de amostragem e limiares adaptativos. Todos os *insights* obtidos e discutidos propõem cenários diversos para a aplicação das diretrizes de observabilidades vistas, referenciando o objetivo específico iii).

Para trabalhos futuros, recomenda-se a investigação de mais técnicas de amostragem, como amostragem baseada em cauda, onde toma-se decisões de amostragem após o *trace* ter sido concluído, permitindo decisões mais fundamentadas com base no contexto completo do *trace*. Adicionalmente, estudos em ambientes com hardware mais restrito e com maiores cargas de trabalho. Por fim, a exploração de técnicas de *machine learning* para automação da calibração dos parâmetros de observabilidade representa um caminho promissor para reduzir a complexidade operacional de manutenção dos mesmos.

8.1. Pacote de Replicação

`https://github.com/ICEI-PUC-Minas-PPLES-TI/plf-es-2025-1-tcci-0393100-pes-gabriel-augusto-borges`

Referências

Basili, V. R. (1992). Software modeling and measurement: the goal/question/metric paradigm.

Costa, B. and Araújo, A. (2024). Análise do overhead da observabilidade na computação em névoa. In *Anais da VII Escola Regional de Alto Desempenho do Centro-Oeste*, pages 11–15, Porto Alegre, RS, Brasil. SBC.

Gomes, F., Gabriel, V., Rocha, L., Rego, P., and Trinta, F. (2024a). Observabilidade de desempenho de arquiteturas monolíticas e microsserviços com opentelemetry. In *Anais do LI Seminário Integrado de Software e Hardware (SEMISH)*, pages 121–132, Brasília/DF, Brasil. Sociedade Brasileira de Computação (SBC).

Gomes, F., Gabriel, V., Rocha, L., Rego, P., and Trinta, F. (2024b). Observabilidade de desempenho de arquiteturas monolíticas e microsserviços com opentelemetry. In *Anais do LI Seminário Integrado de Software e Hardware*, pages 121–132, Porto Alegre, RS, Brasil. SBC.

Gomes, F., Rego, P., and Trinta, F. (2024c). Rumo a uma taxonomia de observabilidade para aplicações baseadas em microsserviços. In *Anais do XXXVIII Simpósio Brasileiro de Engenharia de Software*, pages 234–245, Porto Alegre, RS, Brasil. SBC.

Guégain, E., Bonvoisin, A., Quinton, C., Acher, M., and Rouvoy, R. (2024). Exploring performance trade-offs in jhipster.

Kalman, R. (1959). On the general theory of control systems. *IRE Transactions on Automatic Control*, 4(3):110–110.

Li, Z., Chen, J., Jiao, R., Zhao, N., Wang, Z., Zhang, S., Wu, Y., Jiang, L., Yan, L., Wang, Z., Chen, Z., Zhang, W., Nie, X., Sui, K., and Pei, D. (2021). Practical root cause localization for microservice systems via trace analysis. In *2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQOS)*, pages 1–10.

Madupati, B. (2023). Observability in microservices architectures: Leveraging logging, metrics, and distributed tracing in large-scale systems. *Zenodo*. 8 Pages Posted: 22 Jan 2025, Last revised: 10 Mar 2025. Government of the State of Minnesota - Minnesota Department of Corrections.

Mazarei, A., Sousa, R., Mendes-Moreira, J., Molchanov, S., and Ferreira, H. M. (2025). Online boxplot derived outlier detection. *International Journal of Data Science and Analytics*, 19(1):83–97.

Picoreti, R., Pereira do Carmo, A., Mendonça de Queiroz, F., Salles Garcia, A., Frizera Vassallo, R., and Simeonidou, D. (2018). Multilevel observability in cloud orchestration. In *2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress(DASC/PiCom/DataCom/CyberSciTech)*, pages 776–784.

Rodrigues, K. B. C., Bruno, G. Z., Cardoso, K. V., Corrêa, S. L., and Both, C. (2023). Uma investigação empírica sobre observabilidade em sistemas 5g nativos de nuvem. In *Anais do Simpósio Brasileiro de Software (SBSOFT)*, page 1, Goiânia, GO, Brasil. Sociedade Brasileira de Computação (SBC).

Santana, A. C. d. A., Narciso, R., and Fernandes, A. B. (2025). Explorando as metodologias científicas: tipos de pesquisa, abordagens e aplicações práticas. *Caderno Pedagógico*, 22(1):e13333.

Usman, M., Ferlin, S., Brunstrom, A., and Taheri, J. (2022). A survey on observability of distributed edge & container-based microservices. *IEEE Access*, 10:86904–86919.