Observabilidade de Desempenho de Arquiteturas Monolíticas e Microsserviços com OpenTelemetry

Francisco Gomes[1,2], Vinicius Gabriel[2], Lincoln Rocha[2], Paulo Rego[2], Fernando Trinta[2]

[1]Engine Lab – Universidade Federal do Ceará – Crateús – CE – Brasil

[2]GREat – Universidade Federal do Ceará – Fortaleza – CE – Brasil

almada@crateus.ufc.br, vinicius.bg@alu.ufc.br, lincoln@dc.ufc.br, paulo@dc.ufc.br, fernando.trinta@dc.ufc.br

Abstract. *The adoption of microservices architecture emerged in response to the increasing scale and complexity of modern software systems, offering continuous scalability and overcoming complex monolithic codes. With the growing complexity of cloud-native applications, traditional monitoring solutions become inadequate, increasing the risks of failures. The expansion of monitoring to cloud-native applications is called observability. This study evaluates the use of observability to assess the performance of both monolithic and microservices-based architectures, employing the OpenTelemetry (OTel) solution. A migration of a benchmark application based on microservices to a monolithic system was carried out, and experiments were conducted, revealing that, depending on the available resources of the machines, the benefits of using OTel outweigh the overall expenses incurred, resulting in a decrease of 0.44% in the number of requests served and increase of 0.48% in response time for each request.*

Resumo. *A adoção da arquitetura de microsserviços surgiu como resposta à crescente escala e complexidade dos sistemas de software modernos, oferecendo escalabilidade contínua e superando os desafios apresentados pelos códigos monolíticos complexos. Com o aumento da complexidade das aplicações nativas de nuvem, as soluções tradicionais de monitoramento tornam-se inadequadas, aumentando os riscos de falhas. A extensão do monitoramento para aplicativos nativos de nuvem é conhecida como observabilidade. Este estudo analisa o uso da observabilidade para avaliar o desempenho de arquiteturas tanto monolíticas quanto baseadas em microsserviços, empregando a solução OpenTelemetry (OTel). Foi realizada a migração de uma aplicação de referência baseada em microsserviços para um sistema monolítico, e experimentos foram conduzidos, revelando que, dependendo dos recursos disponíveis das máquinas, os benefícios do uso do OTel superam os custos gerais incorridos, resultando em uma redução de 0,44% na quantidade de requisições atendidas e um aumento de 0,48% no tempo de resposta para cada solicitação.*

1. Introdução

Nos últimos anos, os sistemas de software modernos cresceram em escala e complexidade, tornando cada vez mais desafiador incorporar novas funcionalidades. A emergência da arquitetura de microsserviços tem proporcionado uma solução fundamental para esse

problema [Dragoni et al. 2017]. A arquitetura de microsserviços ganhou ampla adoção para contornar bases de código monolíticas excessivamente complexas e dimensionar aplicativos de maneira transparente. Cada vez mais empresas estão fazendo a transição para uma arquitetura de microsserviços, reconhecendo sua capacidade de aprimorar o desempenho e reduzir a mão de obra humana, compensando assim os custos financeiros associados. Infelizmente, não é uma solução única e universal, e introduz novos desafios para os desenvolvedores [Di Francesco et al. 2017].

À medida que os microsserviços e outros componentes arquiteturais inovadores tornam as aplicações nativas de nuvem mais distribuídas, complexas e imprevisíveis, há um aumento nos modos potenciais de falha e o desempenho pode degradar. O diagnóstico torna-se excepcionalmente desafiador quando uma aplicação nativa de nuvem consiste em centenas de microsserviços executando várias instâncias [Dragoni et al. 2017]. Soluções de monitoramento tradicionais falham em fornecer visibilidade completa e identificar proativamente desvios do comportamento esperado antes que interrompam os serviços. Medidas preventivas e detectivas dão origem a um novo conjunto de exigências de monitoramento desafiadoras que requerem um nível mais elevado de monitoramento, conhecido como observabilidade [Kosińska et al. 2023].

Observabilidade pode ser definida como a capacidade de inferir o estado de um sistema complexo com base em suas saídas [Kalman 1960]. Também se refere à propriedade de um sistema que pode ser observada e comparada com o estado esperado ao longo do ciclo de vida do desenvolvimento de software [Karumuri et al. 2021]. Para se aferir a observabilidade, são necessárias ferramentas. Nos últimos anos, a ferramenta com mais ampla adoção na indústria e considerado o *golden standard* para observabilidade é o *OpenTelemetry* (OTel) [Boten and Majors 2022]. OTel é um *framework* de observabilidade de código aberto que fornece um conjunto de SDKs, APIs e ferramentas padronizadas e independentes de fornecedor para ingestão, transformação e envio de dados para um *backend* de observabilidade [Boten and Majors 2022].

Este estudo avalia a utilização da observabilidade com o OTel em uma aplicação com arquitetura monolítica e baseada em microsserviços. Nossa principal descoberta sugere que, dependendo dos recursos disponíveis das máquinas, as vantagens de usar o OTel superam os custos adicionais introduzidos pelas ferramentas de monitoramento, resultando em uma queda de 0,44% na quantidade de requisições atendidas e aumento de 0,48% no tempo de resposta para cada solicitação. As seguintes questões de pesquisa guiaram o estudo:

RQ1 *Qual é a diferença de desempenho entre uma aplicação monolítica e uma aplicação baseada em microsserviços ao usar o OTel?*

RQ2 *Qual é o overhead causado pelas ferramentas do OTel quando diferentes estratégias de configuração e implantação são usadas para aplicações monolíticas e baseadas em microsserviços?*

As contribuições do artigo são as seguintes. Primeiramente, foi migrada uma aplicação de *benchmark* baseada em microsserviços para uma versão monolítica, detalhando os desafios encontrados. Também foi utilizado o OTel e discutido o seu uso, enfatizando o gerenciamento de desempenho e a observabilidade. Foi definida uma carga de trabalho para experimentação e conduzidos experimentos para avaliar o *overhead* causado pelo OTel ao usar a versão monolítica (MO) e a versão de microsserviços (MS). Foram

propostas melhorias para a versão MS da aplicação na análise com o OTel, mostrando um dos benefícios da solução de telemetria. Finalmente, foi gerado um *dataset* público dos experimentos contendo todas as métricas coletadas para pesquisas futuras.

O restante do artigo está organizado da seguinte forma. A Seção 2 apresenta a fundamentação teórica do trabalho. A Seção 3 discute trabalhos relacionados. A migração da aplicação e a integração do OTel são explicadas na Seção 4. A Seção 5 contém os experimentos e resultados. Por fim, a Seção 6 conclui o estudo juntamente com sugestões para pesquisas futuras.

2. Fundamentação Teórica

2.1. Microsserviços

Os microsserviços surgiram da prática, refletindo padrões arquiteturais do mundo real, em que sistemas consistem em serviços colaborativos comunicando-se através de mecanismos leves [Dragoni et al. 2017]. Esses serviços são focados em partes específicas do negócio, implantados de forma automatizada, suportando diversas linguagens de programação e tecnologias de banco de dados. Cada microsserviço possui um limite claro, garantindo uma modularidade prática que é desafiadora em aplicações monolíticas. Isso leva a serviços individuais mais acessíveis para desenvolvimento, compreensão e manutenção. Equipes independentes escolhem tecnologias adequadas para cada microsserviço, permitindo implantações independentes e atualizações sem impactar outros serviços [Baškarada et al. 2018]. Os microsserviços visam a decomposição da aplicação para facilitar o desenvolvimento ágil e a implantação. Essa arquitetura corresponde a um sistema distribuído com comunicação entre processos, mais complexo do que chamadas de métodos no nível da linguagem e desafiador para testar, exigindo testes de integração entre microsserviços. Com mais partes a configurar, implantar, dimensionar e monitorar, é necessário um controle significativo e alto nível de automação [Pahl and Jamshidi 2016].

2.2. Observabilidade

Observabilidade é um termo emprestado da teoria de controle [Kalman 1960]. Na ciência da computação, é definida como uma característica de software e sistemas relacionada às informações que eles geram, permitindo um monitoramento e entendimento mais abrangentes, inclusive em tempo de execução. Um de seus objetivos é reduzir o tempo necessário para entender por que algo não está funcionando como deveria. Além disso, a observabilidade é um conceito que tem sido usado para se referir a funções avançadas de monitoramento no contexto de aplicações baseadas em microsserviços [Marie-Magdelaine et al. 2019]. A observabilidade de um sistema distribuído pode ser aprimorada empregando o que é comumente referido como os três pilares (sinais) da observabilidade: *logs*, métricas e *traces*.

Logs são registros imutáveis de eventos, que podem ser não estruturados ou estruturados. Coletados de várias fontes, como aplicativos e sistemas operacionais, são processados por uma camada de agregação, armazenados e analisados. *Métricas* são números coletados periodicamente que formam uma série temporal e são geradas agregando eventos de uma entidade, permitindo que revelem tendências no comportamento de um sistema. Normalmente, as métricas são usadas para gerar alertas. *Traces* podem

revelar a causalidade de eventos em um sistema, mostrando como um evento interage com outro [Bento et al. 2021].

2.3. OpenTelemetry

OTel é um *framework* e conjunto de ferramentas de observabilidade projetado para criar e gerenciar dados de telemetria, como *traces*, métricas e *logs*. OTel é agnóstico em relação ao fornecedor, o que significa que pode ser usado com uma ampla variedade de *backends* de observabilidade, como Jaeger[1] e Prometheus[2]. OTel é um projeto da Cloud Native Computing Foundation (CNCF) [Boten and Majors 2022]. Para garantir que os dados de telemetria sejam transmitidos da maneira mais eficiente e confiável possível, o OTel definiu o protocolo *OpenTelemetry* (OTLP), que é definido por meio de arquivos *protobuf*. Isso significa que qualquer cliente ou servidor interessado em enviar ou receber OTLP só precisa implementar essas definições para oferecer suporte a ele.

O *OTel* simplifica a gestão de telemetria com seu componente *Coletor*, que oferece uma abordagem independente do provedor para receber, processar e exportar dados. Isso elimina a necessidade de gerenciar vários agentes/coletores. O *Coletor* atua como intermediário entre a fonte de telemetria e o *backend* que armazena os dados para análise. Importante notar que, apesar dos benefícios, são necessários recursos adicionais para a execução, operação e monitoramento do *Coletor*. O componente de entrada de dados, chamado *Receptor*, possibilita a chegada dos dados no *Coletor*, sendo capaz de suportar diversas fontes, formatos e converter para o formato interno. Geralmente, um *Receptor* registra um ouvinte que expõe uma porta no *Coletor* para os protocolos suportados. Após a recepção dos dados, eles passam por *Processadores* para tarefas adicionais, como filtragem ou inserção de atributos. O componente de saída, o *Exportador*, encaminha os dados para um ou mais *backends*/destinos, suportando múltiplas fontes e permitindo a configuração de vários *Exportadores* para destinos distintos, conforme necessário.

3. Trabalhos Relacionados

[Li et al. 2022] realizaram uma pesquisa sobre microsserviços e observabilidade, enfatizando a complexidade de sistemas de microsserviços industriais operando em infraestruturas de nuvem intrincadas com instâncias de serviço criadas e destruídas dinamicamente. Os autores destacam que o desafio das aplicações baseadas em microsserviços reside em entender e diagnosticar problemas dentro dessa arquitetura, como solicitações falhas e latência elevada, dada a participação de inúmeros serviços e o ambiente de execução diversificado. A observabilidade é reconhecida como crucial nesses sistemas. A pesquisa destaca que o rastreamento e análise de microsserviços introduzem desafios de *big data* para engenharia de software.

[Usman et al. 2022] conduzem uma pesquisa abrangente sobre ambientes distribuídos de TI e observabilidade de microsserviços, visando explorar desafios, requisitos, melhores práticas e soluções atuais na observação de sistemas distribuídos. Os pesquisadores compilaram pesquisas relevantes e artigos, apresentando diversos tipos de dados de telemetria cruciais para resolver problemas operacionais. O artigo identifica e discute funcionalidades essenciais para observabilidade, abordando questões de pesquisa em aberto

[1] `https://www.jaegertracing.io/`
[2] `https://prometheus.io/`

relacionadas a infraestruturas heterogêneas e arquiteturas de microsserviços em diferentes casos de uso. Além disso, o estudo observa a influência da cultura organizacional e das mentalidades individuais na adoção de novas ferramentas e práticas de observabilidade.

[Picoreti et al. 2018] destacam que ferramentas automatizadas de orquestração, embora baseadas na observabilidade da infraestrutura, podem ser insuficientes em casos específicos. Aplicações com requisitos desafiadores, como baixa latência, alta largura de banda e capacidade computacional elevada, demandam uma abordagem baseada na observabilidade em vários níveis, coletando dados tanto do nível da aplicação quanto da infraestrutura. Eles desenvolveram uma plataforma que visa aprimorar a orquestração automatizada em ambientes de nuvem, integrando o Prometheus com o Kubernetes. Os autores utilizaram a métrica de aplicação "taxa total de solicitações" para operações de dimensionamento no Kubernetes.

[Tzanettis et al. 2022] enfrentam o desafio de integrar dados de plataformas de orquestração, rastreamento distribuído e ferramentas de registro para simplificar a análise de desempenho de aplicativos distribuídos. A contribuição central é a especificação de um esquema de ligação de dados que suporta a coleta e integração de dados relacionados a recursos de contêiner, métricas de desempenho, rastreamento distribuído e *logs*. O esquema proposto é capaz de mesclar e correlacionar dados de diferentes tipos de sinais, sendo implementado por meio de ferramentas de código aberto.

Em resumo, [Li et al. 2022] e [Usman et al. 2022] abordaram teoricamente a observabilidade como um todo, discutindo os benefícios e desafios para adoção, focando exclusivamente em microsserviços. [Picoreti et al. 2018] utilizou a observabilidade para apresentar alternativas no gerenciamento do dimensionamento da plataforma de orquestração, e [Tzanettis et al. 2022] aborda o desafio da fusão de dados, mas nenhum dos estudos avaliou o impacto de adicionar as ferramentas de observabilidade para analisar as aplicações. Nosso estudo aplica os conceitos de observabilidade por meio do OTel e foca tanto em microsserviços quanto em monolítica em termos de desempenho, a fim de verificar o impacto que a observabilidade pode trazer para a aplicação.

4. Metodologia utilizada para Avaliação da Observabilidade

Esta Seção apresenta a metodologia utilizada neste estudo. Na Seção 4.1, é introduzida a aplicação TeaStore e discutida a migração para versão monolítica. A Seção 4.2 aprofunda-se na integração do OTel com ambas as versões MO e MS da aplicação. Por fim, a Seção 4.3 explica sobre a implantação da infraestrutura.

4.1. Visão Geral do TeaStore e sua Migração para Monolítica

TeaStore [von Kistowski et al. 2018] é uma aplicação de referência e teste de microsserviços para *benchmarks* e testes. Ela simula uma loja web básica para chás e suprimentos de chá gerados automaticamente, e apresenta componentes para geração de banco de dados e redefinição de serviços, além da própria loja.

Teastore é composta por cinco microsserviços e um serviço de registro que se comunicam entre si usando o protocolo REST. O **Registry** é um serviço que mantém as localizações de instâncias de serviço. O serviço **WebUI** é o *frontend* da aplicação e realiza chamadas de API REST para outros serviços. O **Auth** lida com a autenticação dos usuários, enquanto o serviço **Image-Provider** gerencia imagens de produtos, incluindo

armazenamento e redimensionamento. O serviço **Recommender** oferece recomendações de produtos com base nas interações do usuário, e o serviço **Persistence** fornece acesso ao banco de dados de produtos, permitindo ajustes no tamanho do banco de dados para impactar o desempenho da aplicação.

Para migrar o TeaStore para uma aplicação monolítica, os seguintes passos foram executadas: *(i)* No código dos serviços onde chamadas HTTP eram originalmente usadas, foram substituídas essas chamadas por invocações de métodos visando a funcionalidade correspondente. Essa transição possibilitou a execução local de ações anteriormente realizadas por meio de diferentes serviços via requisições HTTP; e *(ii)* Dentro da aplicação monolítica resultante, apenas o serviço **WebUI** passou a operar como um processo independente, servindo como ponto de entrada para solicitações do usuário. O código de todos os outros serviços foi integrado à aplicação como módulos individuais, invocados por meio de chamadas de método.

A migração abrangeu 54 alterações de arquivos, estabelecendo classes de fachada para cada microsserviço (Auth, Image, Persistence e Recommender). Cada classe de fachada contém métodos espelhando a funcionalidade dos pontos de extremidade HTTP do microsserviço. Agora, os clientes utilizam essas classes de fachada para chamadas de métodos locais, eliminando invocações remotas e a necessidade de lógica de balanceamento de carga e registro.

4.2. Integração com OTel

Na etapa seguinte do estudo, foram configuradas e integradas as ferramentas do OTel com as versões MO e MS do TeaStore. Na Figura 1, é fornecida uma visão geral da arquitetura da solução desenvolvida, que opera em um ambiente de *cluster*. O *cluster* é gerenciado pelo Kubernetes, que é um sistema de orquestração de contêineres de código aberto, garantindo que a aplicação escale conforme necessário. A aplicação utilizada aqui é o TeaStore, e foi utilizado o *Coletor* do OTel. Além disso, as ferramentas *backend* de observabilidade são configuradas para exibição e análise de telemetria.

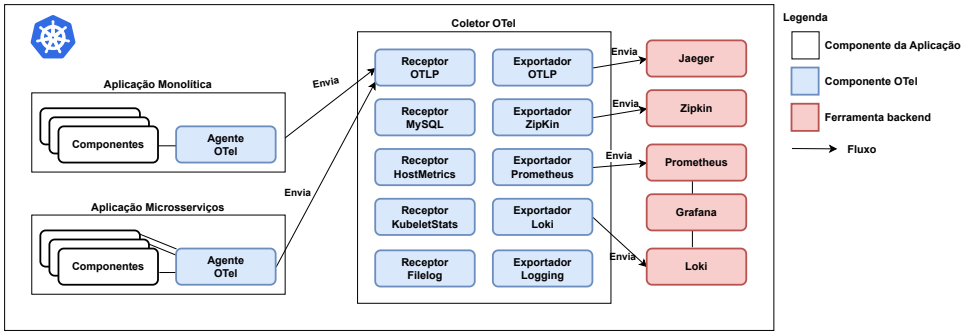


Figura 1. Arquitetura utilizada para Observabilidade dos Experimentos.

Para se ter uma ideia geral do desempenho da aplicação, é necessário configurar receptores para capturar a telemetria de observabilidade como um todo. Assim, foram configurados os receptores que capturam telemetria da aplicação, infraestrutura *host*, *cluster* Kubernetes e banco de dados. No total, dez componentes foram configurados, sendo cinco receptores e cinco exportadores[3].

[3]https://github.com/viniciusbrg/TeaStore-Microservices/blob/

Os seguintes componentes são introduzidos: **Receptor OTLP**, que coleta dados de aplicativos via gRPC ou HTTP no formato OTLP; **Receptor MySQL**, responsável por recuperar métricas do banco de dados consultando o status global do MySQL e tabelas InnoDB; **Receptor HostMetrics**, que gera uma ampla variedade de métricas do sistema hospedeiro, incluindo utilização de CPU, memória, E/S de disco, entre outros; **Receptor KubeletStats**, que obtém métricas de nó, pod, contêiner e volume do servidor API do kubelet; **Receptor Filelog**, usado para coletar logs de arquivos, principalmente para Loki; **Exportador OTLP**, para exportar dados no formato OTLP via gRPC. O Jaeger é um sistema de rastreamento distribuído criado pela Uber e oferece amostragem adaptativa, que possui suporte nativo para OTLP; **Exportador Zipkin**, responsável por enviar dados para um *backend* Zipkin, que é outro sistema de rastreamento distribuído; **Exportador Prometheus**, que exporta dados por *scraping* para um servidor Prometheus; **Exportador Loki**, exporta dados para um sistema de agregação de logs projetado para armazenar e consultar logs de aplicativos e infraestrutura; e **Exportador Logging**, que envia dados para o console.

4.3. Scripts CloudFormation para Facilitar a Reprodutibilidade

Além da migração da aplicação e da integração com o OTel para viabilizar a tecnologia moderna de observabilidade, também foi utilizada uma ferramenta de infraestrutura, o CloudFormation, para facilitar a implementação, experimentação, visualização e análise de dados de telemetria coletados nas versões de microsserviços e monolítica da aplicação TeaStore. O objetivo é preencher a lacuna de não se ter um ambiente que facilite a comparação entre esses tipos de arquitetura, além de ter todas as ferramentas necessárias já configuradas para coletar, visualizar e analisar os dados. Essa solução facilitará a reprodução de trabalhos que desejem usar qualquer versão da aplicação TeaStore e OTel. Além disso, os pesquisadores podem implantar facilmente o ambiente na AWS, uma vez que o serviço CloudFormation fornece e configura esses recursos. Com a configuração do CloudFormation criada, é possível definir o número e o tipo de instâncias tanto da aplicação com a integração, quanto do teste de carga realizado. Essa configuração pode ser vista no link: `https://github.com/viniciusbrg/TeaStore-Microservices/blob/otel-c1/cloudformation.yml`.

5. Experimentos e Resultados

Foram planejados e conduzidos experimentos para avaliar o desempenho das duas versões da aplicação TeaStore e obter *insights* sobre o *overhead* introduzido pelo OTel ao empregar diferentes estratégias de implantação.

5.1. Configuração dos Experimentos

Foram utilizados *scripts* CloudFormation para criar o ambiente de execução da aplicação, implantando um *cluster* Kubernetes na AWS por meio do EKS. As versões em MS e MO foram executadas em três configurações cada. A primeira configuração é sem a observabilidade, para avaliar o desempenho sem a influência do OTel. A segunda configuração é com observabilidade, a fim de analisar o desempenho com a influência do OTel e todas

---

`otel-c1/microservices/examples/kubernetes/otel-manifests/otel-collector-config.yaml`

as ferramentas *backend* no mesmo cluster. A terceira configuração também é com observabilidade, mas com o OTel no mesmo cluster da aplicação e as ferramentas *backend* em outro cluster, a fim de verificar o quanto elas podem influenciar no desempenho. Em relação às réplicas, foram utilizadas 4 WebUI e 4 Persistence da versão MS, pois são os componentes que recebem mais requisições; e 4 réplicas da versão MO. No total, foram definidos 6 cenários para execução:

- *Cenário 1*: MS sem OTel.
- *Cenário 2*: MS com OTel e as ferramentas *backend* no cluster da aplicação.
- *Cenário 3*: MS com OTel e as ferramentas *backend* fora do cluster da aplicação.
- *Cenário 4*: MO sem OTel.
- *Cenário 5*: MO com OTel e as ferramentas *backend* no cluster da aplicação.
- *Cenário 6*: MO com OTel e as ferramentas *backend* fora do cluster da aplicação.

Nos cenários que incorporam o OTel, foram implantados dois ambientes de rastreamento distribuído (Jaeger e Zipkin), proporcionando aos administradores do sistema a flexibilidade para analisar dados usando a plataforma com a qual estão mais familiarizados. Para a análise de métricas recebidas pelos receptores configurados, o Prometheus foi configurado. Além disso, o Loki foi configurado para logs, com resultados exibidos no Grafana, de modo a facilitar a criação de visualizações. Portanto, os usuários que analisam esses dados têm várias opções de ferramentas para cada sinal e podem escolher aquela que se alinha com suas preferências.

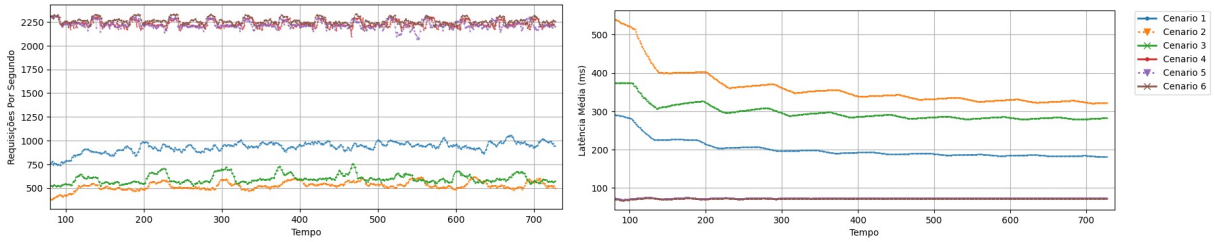
Foi utilizado o Locust[4], pois ele é uma ferramenta escalável, que dá suporte a execução de testes em ambientes distribuídos, para gerar a carga de trabalho. As seguintes operações foram usadas para simular o comportamento do usuário: visitar a página inicial; *login* do usuário; simular comportamento de navegação; simular a compra de produtos no carrinho com dados de usuário; visitar o perfil do usuário; e realizar o *logout* da sessão. A execução dessas operações é realizada variando o número de usuários, usando *ramp-up* e *down*, para simular um cenário real de uso da aplicação. Empiricamente, o número de usuários para o teste foi definido variando de 100 usuários a 200 usuários simultaneamente. Foi utilizada a mesma carga de trabalho por 30 minutos em todos os cenários. O Locust coleta várias métricas, mas focou-se no número de requisições por segundo que a aplicação atendeu e no tempo de resposta para cada solicitação.

5.2. Resultados e Discussões

Os resultados encontrados com os experimentos são apresentados a seguir. Como pode ser visto na Figura 2a, o Cenário 1 possui o maior valor de Requisições por Segundo (RPS), o que indica que o OTel (Cenários 2 e 3), influencia negativamente a capacidade da aplicação de lidar com requisições nas configurações de máquinas que foram utilizadas. Esse impacto negativo foi de 43,4% (comparando os Cenários 1 e 2). Ao analisar os Cenários 2 e 3, é possível ver que, ao remover as ferramentas de *backend* do cluster de execução do OTel, mesmo com a necessidade de mais comunicação de rede, ele pode lidar com mais requisições do que no Cenário 2 (todos os serviços no mesmo cluster Kubernetes), aumentando em 12,75% o RPS. Assim, a degradação está relacionada aos recursos das máquinas utilizadas. Esse comportamento também é refletido no tempo de resposta. Como mostrado na Figura 2b, o Cenário 1 possui o menor tempo de resposta

[4]`https://locust.io/`

em comparação com os outros cenários. Isso indica que o OTel tem um impacto negativo, levando a tempos de resposta mais longos para as requisições dos clientes nas aplicações (aproximadamente 77,08% maior). A mesma análise se aplica aos Cenários 2 e 3, quando as ferramentas *backend* estão fora do cluster, tem uma melhora de 14,94% no tempo de resposta.



(a) Requisições Atendidas pela Aplicação **(b) Tempo de Resposta das Requisições**

Figura 2. Desempenho da Aplicação

Ao avaliar as versões monolíticas, o resultado é diferente. Como visto na Figura 2a, nos Cenários 4, 5 e 6, o RPS é praticamente o mesmo. Examinando os recursos da máquina, foi possível verificar que esse tipo de aplicação, com essas configurações, ainda poderia lidar com mais requisições porque não estava completamente sobrecarregado. Isso mostra que, como ainda havia recursos disponíveis, o OTel teve um pequeno impacto na aplicação (0,44%). O mesmo comportamento pode ser observado na Figura 2b. O tempo de resposta nesses cenários é praticamente o mesmo (aumento de 0,48% com o OTel). Portanto, o resultado principal é que, com base nas configurações de máquinas utilizadas, o OTel pode ter um impacto negativo na aplicação, mas apenas se os recursos da máquina forem limitados. Assim, se o ambiente provisionado tiver recursos suficientes, é provável que a aplicação se beneficie do uso do OTel. Esses resultados contribuem para abordar a **RQ1**, ilustrando que a aplicação monolítica supera sua contraparte baseada em microsserviços por um fator que varia de 2,35 a 4,15 vezes em RPS. Além disso, foram obtidos *insights* para responder à **RQ2**. As ferramentas do OTel precisam de recursos do *cluster* Kubernetes. Portanto, se o *cluster* estiver operando próximo do limite máximo de recursos, as ferramentas podem comprometer o desempenho da aplicação, pois competem por recursos. Assim, é essencial manter uma boa política de elasticidade para o *cluster* e garantir que ele não opere subprovisionado.

Foi aplicado um teste estatístico para avaliar se os valores médios são boas representações dos resultados e, consequentemente, para reforçar as observações apresentadas até agora. Inicialmente, foram divididos os resultados obtidos em três grupos. Cada grupo continha os resultados dos dois tipos de arquitetura (microsserviço e monolítico) para os cenários apresentados anteriormente. Como cada grupo consistia em dois subgrupos de amostras não pareadas, foram realizados os testes necessários para avaliar se seria viável aplicar o teste T-Student. Ao aplicar o teste de Bartlett, foi verificado que as variâncias das amostras não eram iguais, o que torna inviável o uso do teste T-Student. Por causa disso, foi escolhido o teste de Mann-Whitney. No teste de Mann-Whitney, a hipótese nula (H0) indica nenhuma diferença entre as médias avaliadas, ou seja, rejeitar H0 significa que as médias são diferentes. A Tabela 1 mostra os resultados do teste de Mann-Whitney. Foi observado que a hipótese nula foi rejeitada para todos os cenários.

Assim, foi concluído que as médias mostradas na Figura 2a e Figura 2b são estatisticamente diferentes e resumem o comportamento de cada cenário.

Cenários	1 e 4	2 e 5	3 e 6
RPS			
Mann-Whitney	$7.77 \cdot 10^{-275}$	$3.61 \cdot 10^{-282}$	$1.81 \cdot 10^{-282}$
H0 é rejeitada?	✓	✓	✓
Tempo de resposta			
Mann-Whitney	$5.83 \cdot 10^{-280}$	$8.26 \cdot 10^{-291}$	$6.59 \cdot 10^{-289}$
H0 é rejeitada?	✓	✓	✓

Tabela 1. Teste Mann-Whitney relacionado aos resultados

Em relação às diferenças entre as aplicações monolítica e de microsserviços, foi observado que a aplicação monolítica tem um desempenho muito melhor do que a aplicação de microsserviços. Esse impacto ocorre devido à necessidade de E/S de rede na aplicação de microsserviços, enquanto a aplicação monolítica usa chamadas locais. Os cenários de microsserviços que utilizam o OTel também são mais afetados, pois cada serviço terá que criar e registrar *spans*, resultando em um uso maior da rede. Além dos resultados mostrados anteriormente, foi identificado um anti-padrão na aplicação original com a utilização do OTel. Analisando os *traces* fornecidos pelo OTel e visualizados por meio do Jaeger, a Figura 3 mostra que a aplicação emite múltiplas solicitações HTTP GET para o recurso "produto", resultando em acessos a outros microsserviços. Para cada uma das requisições a essa rota, o microsserviço Teastore-Persistence é acessado 25 vezes, o que naturalmente degrada o tempo de resposta da aplicação. Após uma investigação mais aprofundada, foi descoberto que existe um *loop* desencadeando esse comportamento no código. Portanto, uma melhoria envolveria a reimplementação do método para que ele, por exemplo, aceitasse uma lista de parâmetros e retornasse os dados solicitados.

Figura 3. Reconhecimento de um Anti-padrão com o Jaeger

Para realizar uma análise profunda dos *traces*, a fim de compreender os resultados, foi realizado uma instrumentação manual usando a anotação chamada *@WithSpan*. Essa anotação é usada para criar um *span* correspondente a um dos métodos da aplicação quando invocado (ele gera um *span* com sua duração e quaisquer exceções lançadas). No MO, a resposta a uma solicitação depende apenas do mesmo processo, que chama os métodos sequencialmente. Já no MS, a resposta de microsserviços depende de uma série de solicitações a outros microsserviços, o que introduz atrasos na resposta final ao usuário.

Os dados gerados pelo OTel foram organizados em um *dataset* (`https://tinyurl.com/semish2024`) para análises futuras pela comunidade. O conjunto de dados inclui métricas e *traces* coletados. Adicionalmente, o *dataset* inclui métricas de CPU (em %) e rede (em Mb) das instâncias EC2. A Tabela 2 apresenta a média e o intervalo de confiança de 95% da utilização de CPU e rede durante o experimento. Como pode ser observado, a versão MO da aplicação requer menos CPU e utiliza menos tráfego de rede. Pesquisadores podem aproveitar este *dataset* para realizar diversos

estudos, como, por exemplo, análises de desempenho. Além disso, o conjunto de dados fornece um recurso interessante para que os pesquisadores formularem e avaliarem modelos e conduzirem simulações, permitindo uma compreensão mais profunda da dinâmica e comportamento do sistema sob condições variáveis.

Cenário	CPU (%)	Network in (Mb)	Network out (Mb)
Cenário 1	73.21 ± 4.96	1236.21 ± 549.80	1997.32 ± 535.33
Cenário 2	52.07 ± 9.32	719.57 ± 218.36	1159.38 ± 519.24
Cenário 3	60.49 ± 6.54	801.28 ± 376.05	1369.04 ± 341.74
Cenário 4	26.59 ± 5.36	464.37 ± 323.61	965.33 ± 373.93
Cenário 5	40.12 ± 3.73	672.85 ± 490.98	1406.02 ± 514.35
Cenário 6	38.82 ± 1.57	680.95 ± 499.89	1415.96 ± 491.73

Tabela 2. Uso da CPU e da rede do cluster Kubernetes (instâncias EC2)

A partir dos resultados e estudos com o OTel, fica evidente que ele abrange todos os três aspectos do monitoramento de software: *logs*, métricas e *traces*. Tudo isso sem estar vinculado a provedores específicos. Além disso, não é necessário utilizar as bibliotecas de instrumentação do provedor da solução para atender às especificações do cliente, reduzindo o esforço e o custo de desenvolvimento de software. Outra vantagem do OTel é a interoperabilidade, o que significa que vários sistemas podem usar essa forma padrão de instrumentação em todos os serviços para trocar informações. A flexibilidade para alterar as ferramentas de observabilidade do *backend*, sem alterar a instrumentação, é outra vantagem. Por fim, se novas tecnologias forem criadas com as especificações do OTel, elas podem se integrar automaticamente à solução. Em resumo, o OTel oferece vários benefícios significativos para organizações que buscam aprimorar suas capacidades de observabilidade de aplicativos. Apesar desses benefícios, este trabalho foi capaz de avaliar o *overhead* da solução.

6. Conclusão e Trabalhos Futuros

Este trabalho discutiu a comparação de desempenho entre uma aplicação com arquitetura monolítica e baseada em microsserviços, com e sem o uso da observabilidade com o OTel. Para a realização do trabalho, foi realizado o processo de migração de uma aplicação baseada em microsserviços para uma versão monolítica. Os resultados mostraram que a aplicação monolítica superou a de microsserviços. A influência do OTel depende dos recursos das máquinas nas quais a aplicação é executada, e os resultados demonstraram que na configuração monolítica, a aplicação manteve um comportamento consistente, independentemente do uso do OTel. Portanto, conclui-se que o uso do OTel traz benefícios, além do monitoramento de métricas, *logs* e *traces* para entender o comportamento da aplicação, a detecção de problemas na aplicação. Como trabalho futuro, pretende-se utilizar algoritmos de aprendizado de máquina nos dados coletados por meio do OTel para sugerir melhorias no código da aplicação, identificar gargalos, criar componentes para o OTel com métricas personalizadas para uma compreensão mais aprofundada da aplicação e integrar a observabilidade no mecanismo de *auto-scaling* de recursos do Kubernetes.

Referências

Baškarada, S., Nguyen, V., and Koronios, A. (2018). Architecting microservices: Practical opportunities and challenges. *Journal of Computer Information Systems*.

Bento, A., Correia, J., Filipe, R., Araujo, F., and Cardoso, J. (2021). Automated analysis of distributed tracing: Challenges and research directions. *Journal of Grid Computing*, 19:1–15.

Boten, A. and Majors, C. (2022). *Cloud-Native Observability with OpenTelemetry: Learn to gain visibility into systems by combining tracing, metrics, and logging with OpenTelemetry*. Packt Publishing.

Di Francesco, P., Malavolta, I., and Lago, P. (2017). Research on architecting microservices: Trends, focus, and potential for industrial adoption. In *2017 IEEE International conference on software architecture (ICSA)*, pages 21–30. IEEE.

Dragoni, N., Giallorenzo, S., Lafuente, A. L., Mazzara, M., Montesi, F., Mustafin, R., and Safina, L. (2017). Microservices: yesterday, today, and tomorrow. *Present and ulterior software engineering*, pages 195–216.

Kalman, R. E. (1960). On the general theory of control systems. In *Proceedings First International Conference on Automatic Control, Moscow, USSR*, pages 481–492.

Karumuri, S., Solleza, F., Zdonik, S., and Tatbul, N. (2021). Towards observability data management at scale. *ACM SIGMOD Record*, 49(4):18–23.

Kosińska, J., Baliś, B., Konieczny, M., Malawski, M., and Zieliński, S. (2023). Towards the observability of cloud-native applications: The overview of the state-of-the-art. *IEEE Access*.

Li, B., Peng, X., Xiang, Q., Wang, H., Xie, T., Sun, J., and Liu, X. (2022). Enjoy your observability: an industrial survey of microservice tracing and analysis. *Empirical Software Engineering*, 27:1–28.

Marie-Magdelaine, N., Ahmed, T., and Astruc-Amato, G. (2019). Demonstration of an observability framework for cloud native microservices. In *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pages 722–724. IEEE.

Pahl, C. and Jamshidi, P. (2016). Microservices: A systematic mapping study. *CLOSER (1)*, pages 137–146.

Picoreti, R., do Carmo, A. P., de Queiroz, F. M., Garcia, A. S., Vassallo, R. F., 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. IEEE.

Tzanettis, I., Androna, C.-M., Zafeiropoulos, A., Fotopoulou, E., and Papavassiliou, S. (2022). Data fusion of observability signals for assisting orchestration of distributed applications. *Sensors*, 22(5):2061.

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.

von Kistowski, J., Eismann, S., Schmitt, N., Bauer, A., Grohmann, J., and Kounev, S. (2018). TeaStore: A Micro-Service Reference Application for Benchmarking, Modeling and Resource Management Research. In *Proceedings of the 26th IEEE International Symposium on the Modelling, Analysis, and Simulation of Computer and Telecommunication Systems*, MASCOTS '18.