Análise de Desempenho do Spring PetClinic Microservices com Locust

1st Cristina Sousa
Sistemas de Informação
Universidade Federal do Piauí (UFPI)
Picos, Brazil
cristina.sousa@ufpi.edu.br

2nd Francinaldo Barbosa
Sistemas de Informação
Universidade Federal do Piauí (UFPI)
Picos, Brazil
francinaldo.barbosa@ufpi.edu.br

3rd Iago Roberto
Sistemas de Informação
Universidade Federal do Piauí (UFPI)
Picos, Brazil
iago.almeida@ufpi.edu.br

***Abstract*—Este trabalho apresenta uma avaliação de desempenho da aplicação *Spring PetClinic Microservices*, com o objetivo de analisar o comportamento de uma arquitetura de microsserviços sob diferentes níveis de carga. Utilizando a ferramenta Locust, foram definidos três cenários experimentais — leve, moderado e de pico — que permitiram medir métricas essenciais como latência, throughput e taxa de falhas. Os testes foram realizados em ambiente Docker Compose, garantindo reprodutibilidade e isolamento dos serviços. Os resultados demonstraram que o sistema mantém desempenho satisfatório em cargas leves e moderadas, mas sofre degradação significativa em situações de pico. A análise evidenciou o impacto do aumento de usuários simultâneos sobre a estabilidade e a escalabilidade da aplicação. Por fim, o estudo contribui para compreender os limites operacionais e identificar oportunidades de otimização em arquiteturas distribuídas.**

***Index Terms*—Microservices, Performance Evaluation, Load Testing, Locust, Spring PetClinic.**

I. Introdução

A arquitetura de microsserviços tem ganhado cada vez mais espaço no desenvolvimento de aplicações modernas devido à sua capacidade de oferecer escalabilidade independente, maior resiliência e flexibilidade tecnológica [1]. No entanto, essa distribuição funcional entre múltiplos serviços traz novos desafios relacionados ao desempenho [2]. As requisições passam por diferentes componentes, o que pode aumentar a latência e gerar pontos de falha. Processos como descoberta de serviços, comunicação via rede e serialização de dados introduzem overhead que não existe em arquiteturas monolíticas [3]. Dessa forma, garantir que o sistema permaneça eficiente sob carga torna-se essencial. Avaliar o desempenho permite identificar gargalos, prever comportamento em produção e assegurar que os requisitos não funcionais sejam atendidos [4].

Nesse contexto, o Locust surge como uma ferramenta flexível e moderna para testes de carga. Desenvolvido em Python, ele permite definir cenários personalizados com código, o que facilita a simulação de diferentes padrões de uso da aplicação. Além disso, apresenta uma interface web que exibe métricas em tempo real, como tempos de resposta, taxa de falhas e requisições por segundo. Sua arquitetura distribuída possibilita distribuir a carga em múltiplas máquinas, permitindo estressar sistemas mais complexos, incluindo microsserviços [2]. Assim, o Locust se mostra adequado para avaliar como uma aplicação responde ao aumento de usuários concorrentes, especialmente quando envolve múltiplas integrações internas.

O objetivo deste trabalho é avaliar o desempenho do Spring PetClinic em sua versão baseada em microsserviços, utilizando o Locust como ferramenta de teste de carga. O sistema representa um cenário realista de aplicação distribuída, com serviços independentes para gerenciamento de clientes, veterinários e visitas, além de componentes de infraestrutura como API Gateway e banco de dados. Este estudo busca mensurar métricas essenciais, como latência, throughput e taxa de sucesso das requisições em diferentes níveis de estresse. Com isso, pretende-se analisar o comportamento do sistema sob cargas variadas, identificar possíveis gargalos e compreender os limites operacionais da aplicação. Os resultados obtidos servirão como base para recomendações de melhorias e validação da eficiência da arquitetura distribuída adotada [4].

II. Descrição do Sistema Avaliado

O *Spring PetClinic Microservices* é o sistema avaliado neste estudo, sendo amplamente utilizado como aplicação de referência para demonstração e experimentação de arquitetura baseada em microsserviços no ecossistema Spring [5], [6]. Ele simula um ambiente real de gerenciamento de uma clínica veterinária, incluindo funcionalidades de cadastro de proprietários, animais de estimação, atendimentos e informações de profissionais. Cada domínio funcional é implementado como um serviço independente, permitindo que a aplicação seja desenvolvida, escalada e mantida de forma modular. Essa abordagem facilita a evolução contínua da aplicação e contribui para maior resiliência do sistema como um todo [7].

A arquitetura distribuída da aplicação é composta por seis microsserviços principais, que se comunicam via APIs REST [6]. Entre eles, destacam-se: *Customers Service* (8081), responsável por proprietários e seus animais; *Vets Service* (8083), que gerencia os veterinários; *Visits Service* (8082), dedicado aos registros de consultas; e o *GenAI Service* (8084), que fornece funcionalidades de chatbot com inteligência artificial. O *API Gateway* (8080) atua como ponto único de entrada para clientes externos, enquanto o *Config Server* (8888) centraliza configurações e o *Discovery Server* (8761), baseado em Eureka, faz a descoberta dinâmica dos serviços [8].

Cada microsserviço utiliza bancos HSQLDB independentes, garantindo isolamento operacional e maior tolerância a falhas.

Para a execução dos experimentos, a aplicação foi implantada utilizando Docker Compose com a configuração oficial do repositório [6]. O ambiente inclui ainda ferramentas de observabilidade, como Zipkin (9411), Prometheus (9091), Grafana (3030) e Spring Boot Admin (9090), que permitem monitorar o comportamento dos serviços durante os testes. Os contêineres foram limitados a 512 MB de memória cada, buscando padronizar o ambiente e evitar interferências externas nos resultados. Durante os testes de carga, as requisições foram direcionadas diretamente aos serviços funcionais, evitando o uso do API Gateway e possibilitando avaliar o desempenho individual de cada componente da arquitetura distribuída [7], [8].

TABLE I
ENDPOINTS EXERCITADOS DURANTE OS TESTES

Endpoint	Serviço	Distribuição (%)
GET /owners	Customers Service (8081)	40%
GET /owners/{id}	Customers Service (8081)	30%
GET /vets	Vets Service (8083)	20%
POST /owners	Customers Service (8081)	10%

Essa seleção de requisições contempla operações frequentes de consulta (70%) e inserção (10%), permitindo avaliar, de forma equilibrada, tanto o acesso ao banco de dados quanto o processamento das funcionalidades centrais do sistema [6], [7].

III. CENÁRIOS DE TESTE

Para avaliar o desempenho da aplicação Spring PetClinic em uma arquitetura baseada em microsserviços, foram definidos três cenários distintos de carga utilizando a ferramenta de testes de desempenho Locust. Essa abordagem visa compreender de forma abrangente o comportamento do sistema sob diferentes condições de estresse e concorrência, reproduzindo situações próximas às que ocorrem em ambientes reais de produção.

Cada cenário foi cuidadosamente planejado para representar um nível específico de utilização, variando desde um uso leve e estável, caracterizado por poucos usuários simultâneos, até uma carga intensa, com um grande número de requisições concorrentes. Essa variação possibilitou observar o impacto do aumento gradual da demanda sobre métricas fundamentais de desempenho, como latência média e máxima das requisições, throughput (taxa de requisições por segundo) e percentual de sucesso das requisições. Dessa forma, foi possível identificar o comportamento do sistema tanto em condições normais de operação quanto em situações de saturação.

Além da análise global do desempenho, cada microsserviço foi avaliado individualmente, permitindo detectar gargalos específicos de comunicação, processamento e acesso a dados. Essa observação detalhada é essencial para compreender como a arquitetura distribuída da aplicação reage ao aumento da carga, especialmente considerando que a comunicação entre serviços pode introduzir sobrecarga de rede e aumento na latência total das respostas. Assim, os testes permitiram não apenas medir a eficiência do sistema como um todo, mas também avaliar a contribuição e o impacto de cada componente isoladamente.

Os experimentos foram conduzidos de maneira progressiva e controlada, iniciando com um volume reduzido de acessos e evoluindo até atingir o pico de requisições simultâneas. Essa metodologia possibilitou identificar o ponto de inflexão em que o desempenho do sistema começa a se degradar, fornecendo subsídios para futuras otimizações de infraestrutura, ajustes de balanceamento de carga e aprimoramentos na comunicação entre microsserviços.

A Tabela II apresenta as especificações do hardware utilizado durante os experimentos. O ambiente de testes foi configurado em um notebook Acer Nitro 5, equipado com o sistema operacional Ubuntu 24.04.3 LTS, processador AMD Ryzen™ 5 7535HS, 8 GiB de memória RAM e GPU NVIDIA GeForce RTX 3050 Laptop. Essa configuração oferece uma base computacional robusta para a execução dos testes, garantindo condições adequadas para avaliar a escalabilidade, a resiliência e o comportamento dinâmico da aplicação sob diferentes níveis de carga.

Além disso, a escolha de um ambiente de hardware intermediário — semelhante a uma estação de trabalho moderna, mas não de alto desempenho — reflete uma preocupação em reproduzir cenários realistas de implantação, em que recursos computacionais são limitados e a eficiência do sistema depende fortemente da otimização da arquitetura de microsserviços. Assim, os resultados obtidos fornecem informações relevantes não apenas sobre o desempenho bruto da aplicação, mas também sobre sua capacidade de adaptação e manutenção da qualidade de serviço diante do crescimento da demanda.

TABLE II
ESPECIFICAÇÕES DO HARDWARE

Acer Nitro AN515-47	
Sistema Operacional	Ubuntu 24.04.3 LTS
Processador	AMD Ryzen™ 5 7535HS
GPU	NVIDIA GeForce RTX 3050 Laptop GPU
Memória	8,0 GiB

A. Cenário 1 — Carga Leve

- **Usuários simultâneos:** 20
- **Taxa de criação (spawn rate):** 5 usuários/s
- **Duração:** 10 minutos
- **Objetivo:** Estabelecer uma linha de base e identificar possíveis erros iniciais sob baixa carga.

Esse cenário inicial coloca o sistema em um estado semelhante ao uso comum, permitindo verificar se todas as funcionalidades estão acessíveis e operando corretamente. A baixa concorrência contribui para observar uma resposta consistente, sem interferência significativa da fila de requisições. Com isso, é possível registrar os menores tempos médios de resposta e altas taxas de sucesso, servindo como referência para comparação com os demais cenários. Além disso, o

teste auxilia na detecção de falhas de configuração, atrasos inesperados e problemas de comunicação entre microsserviços logo no início da avaliação.

B. Cenário 2 — Carga Moderada

- **Usuários simultâneos:** 50
- **Taxa de criação (spawn rate):** 5 usuários/s
- **Duração:** 10 minutos
- **Objetivo:** Observar o impacto do aumento de usuários na latência e no throughput.

Nesse cenário, a demanda sobre o sistema cresce de forma considerável, simulando horários de maior concentração de acessos. A partir dessa carga, espera-se identificar variações relevantes nos tempos de resposta, especialmente em endpoints mais exigidos. O comportamento dos bancos de dados independentes também passa a ter papel importante, podendo influenciar diretamente a escalabilidade. A análise desse estágio possibilita identificar serviços mais sensíveis ao aumento de requisições e eventuais limitações estruturais que podem se intensificar em situações futuras de pico.

C. Cenário 3 — Pico de Acesso

- **Usuários simultâneos:** 100–200 (dependendo da capacidade do ambiente)
- **Taxa de criação (spawn rate):** 10 usuários/s
- **Duração:** 5 minutos
- **Objetivo:** Simular condições extremas e analisar eventuais falhas e quedas de desempenho.

O terceiro cenário representa uma situação de alto estresse com intensa concorrência no acesso aos serviços. Nessa etapa, espera-se uma degradação perceptível na performance, podendo ocorrer aumentos bruscos na latência e erros das categorias 4xx e 5xx. O teste é fundamental para determinar a capacidade máxima suportada pela infraestrutura atual, contribuindo para decisões relacionadas a dimensionamento e estratégias de escalabilidade futura. Além disso, a análise dos resultados permite identificar possíveis pontos críticos do sistema que exigem otimização para sustentar cenários com grande volume de requisições.

IV. Resultados

A análise dos testes foi realizada a partir dos arquivos exportados pelo Locust, contendo as métricas coletadas em cada execução dos três cenários de carga definidos anteriormente. Para isso, foi desenvolvido um script em Python que agregou e processou os dados experimentais, gerando automaticamente tabelas e gráficos que sintetizam o comportamento da aplicação sob diferentes intensidades de uso. Os resultados obtidos permitem observar como a latência, o throughput e a taxa de falhas evoluem conforme o aumento do número de usuários simultâneos, possibilitando uma avaliação quantitativa da escalabilidade da arquitetura baseada em microsserviços.

A Tabela III apresenta um resumo geral dos resultados obtidos nos três cenários de carga (Leve, Moderado e Pico), incluindo o total de requisições realizadas, a taxa de falhas, o tempo médio de resposta e o throughput (requisições por segundo). Observa-se um aumento progressivo no número total de requisições conforme o nível de carga se intensifica, acompanhado por variações significativas nas demais métricas. A taxa de falhas, em particular, mostra um crescimento acentuado no cenário de pico, enquanto o tempo médio e o throughput demonstram comportamentos distintos entre os cenários, refletindo as diferentes condições de carga aplicadas. Esses resultados fornecem uma visão inicial do comportamento geral do sistema sob diferentes intensidades de uso, sem que, neste momento, sejam exploradas conclusões sobre o desempenho.

TABLE III
RESUMO GERAL DAS MÉTRICAS DE DESEMPENHO COLETADAS NOS CENÁRIOS DE CARGA LEVE, MODERADO E DE PICO.

Cenário	Total Requisições	Taxa de Falhas (%)	Tempo Médio (ms)	Throughput (req/s)
Leve	71156	2.43	8.07	1.41
Moderado	176854	8.9	18.71	3.47
Pico	347250	58.99	16.31	13.65

A Figura 1 apresenta a proporção de requisições bem-sucedidas e com falha em cada cenário de teste. Observa-se um aumento progressivo no volume total de requisições conforme a carga aplicada cresce, acompanhado por uma variação gradual na quantidade de falhas. No cenário leve, predominam as requisições com sucesso, enquanto no cenário moderado já se nota um pequeno incremento de falhas, possivelmente associado ao maior número de acessos simultâneos. No cenário de pico, o volume de requisições atinge seu máximo e a proporção de falhas torna-se mais evidente, refletindo o impacto da alta concorrência sobre a estabilidade do sistema.

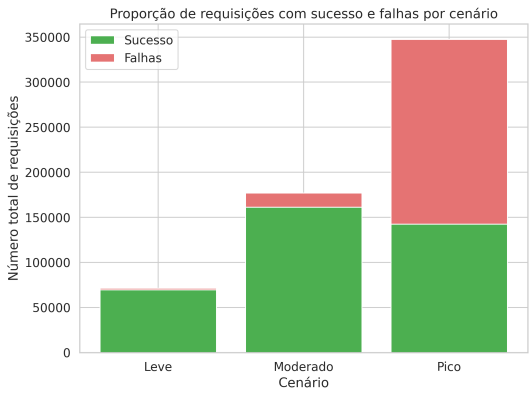

Fig. 1. Proporção de requisições com sucesso e falha por cenário

A Figura 2 apresenta a taxa de falhas percentual observada em cada cenário de teste. Nota-se um crescimento gradual dessa taxa à medida que a carga de usuários aumenta, indicando maior ocorrência de erros sob condições de concorrência mais intensa. No cenário leve, a taxa de falhas mantém-se baixa, em torno de 2,4%, sugerindo estabilidade da aplicação nesse nível de uso. No cenário moderado, o valor sobe para aproximadamente 8,9%, refletindo um impacto

inicial do aumento de requisições simultâneas. Já no cenário de pico, a taxa atinge cerca de 59%, representando uma elevação significativa no número de falhas em relação aos demais testes. Esses valores mostram a variação do comportamento do sistema conforme a intensidade da carga aplicada, sem ainda indicar as causas específicas dessas ocorrências.

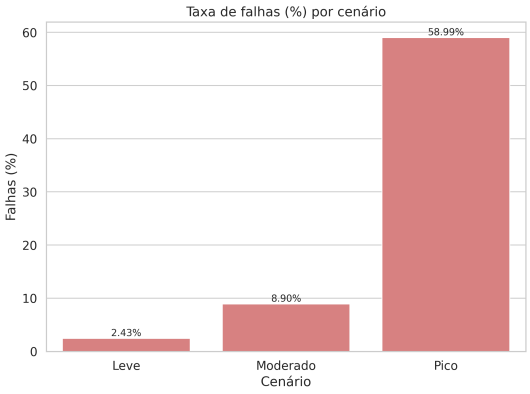

Fig. 2. Taxa de falhas por cenário

A Figura 3 apresenta os percentis de latência (P50, P90, P95 e P99) medidos em cada cenário de carga, ilustrando a variação no tempo de resposta das requisições. Observa-se que, no cenário leve, os valores permanecem baixos e relativamente próximos entre si, com o P99 em torno de 30 ms, indicando respostas rápidas e consistentes. No cenário moderado, há um aumento gradual em todos os percentis, refletindo o impacto do crescimento da carga no desempenho médio e nas requisições mais lentas. Já no cenário de pico, nota-se um comportamento distinto, em que os percentis inferiores (P50 a P95) mantêm valores moderados, enquanto o P99 se eleva de forma expressiva, ultrapassando 100 ms. Essa diferença sugere a ocorrência de picos de latência em uma pequena parcela das requisições sob carga mais intensa, sem que ainda se possam identificar as causas específicas desse comportamento.

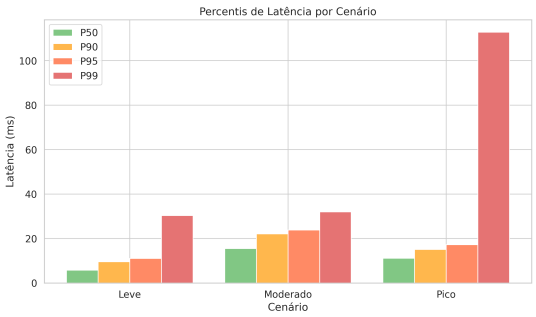

Fig. 3. Percentis de latência por cenário

A Figura 4 apresenta a comparação dos tempos de resposta (mínimo, médio e máximo) obtidos em cada cenário de carga. Observa-se que, no cenário leve, o tempo de resposta médio mantém-se baixo, indicando boa eficiência sob baixa concorrência. À medida que a carga aumenta para o cenário moderado, há um crescimento moderado nos tempos médio e máximo, refletindo a influência da maior quantidade de usuários simultâneos. No cenário de pico, o tempo máximo ultrapassa 900 ms, demonstrando que, sob alta demanda, o sistema enfrenta maior latência e possíveis gargalos de processamento. Esse comportamento revela a limitação do sistema em sustentar tempos de resposta estáveis quando exposto a níveis extremos de carga.

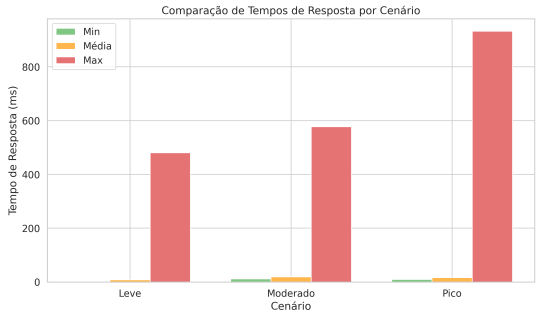

Fig. 4. Comparação de tempos de resposta

A Figura 5 apresenta o throughput médio por cenário, evidenciando o comportamento do sistema sob diferentes níveis de carga. Observa-se um crescimento progressivo da taxa de requisições por segundo, passando de aproximadamente 1,4 no cenário leve para cerca de 3,5 no cenário moderado e atingindo 13,6 no cenário de pico. Esse aumento demonstra que o sistema é capaz de processar um volume maior de requisições conforme a carga se intensifica, mantendo a capacidade de resposta dentro de limites adequados. No entanto, o crescimento acentuado no cenário de pico pode indicar a proximidade de saturação dos recursos, sugerindo que, apesar da boa escalabilidade, há um ponto em que o desempenho pode começar a se degradar sob cargas ainda mais elevadas.

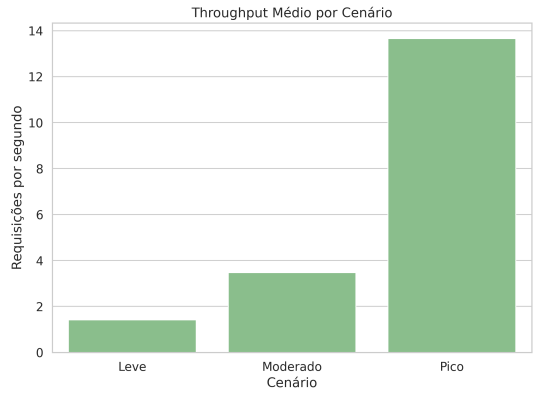

Fig. 5. Throughput médio por cenário

V. Conclusões

Os testes realizados com o *Spring PetClinic Microservices* permitiram avaliar o comportamento da aplicação sob difer-

entes níveis de carga e concorrência. Os resultados evidenciaram que o sistema mantém bom desempenho em cargas leves e moderadas, apresentando tempos médios de resposta estáveis e taxas de sucesso elevadas. Contudo, à medida que a carga aumenta, observou-se uma degradação progressiva, especialmente no cenário de pico, com aumento significativo da latência e crescimento acentuado da taxa de falhas. Essa tendência indica que o sistema possui limites de escalabilidade definidos pelo ambiente e pela capacidade de processamento de cada microsserviço.

De forma geral, o experimento demonstrou que a arquitetura distribuída proporciona bom desempenho sob condições normais de operação, mas exige atenção em cenários de alta concorrência. A sobrecarga observada nos testes de pico pode estar relacionada à limitação de recursos de hardware e à ausência de mecanismos de balanceamento de carga mais sofisticados. Esses resultados reforçam a importância da avaliação de desempenho contínua em aplicações baseadas em microsserviços, permitindo antecipar gargalos e planejar técnicas de otimização antes da implantação em ambientes de produção.

VI. Limitações

Entre as limitações deste estudo, destaca-se o uso de um ambiente de testes restrito, executado em um único equipamento com recursos limitados de memória e processamento. Essa configuração pode ter influenciado diretamente os resultados observados nos cenários de carga mais intensa, especialmente em situações em que o consumo de recursos atingiu o limite físico do sistema.

Outro fator relevante é que a aplicação permite configurar a quantidade de memória alocada para cada contêiner que compõe a arquitetura. Essa parametrização pode impactar significativamente o desempenho geral, uma vez que restrições de memória podem provocar competição por recursos, aumento de latência ou até reinicializações de contêineres sob alta carga. No presente estudo, essa variável não foi explorada de forma sistemática, o que limita a compreensão dos efeitos dessas configurações sobre o comportamento do sistema.

Além disso, o foco da análise concentrou-se apenas nos principais serviços da aplicação, sem medições detalhadas sobre o desempenho interno do banco de dados ou sobre o impacto da comunicação entre contêineres. Esses fatores reduzem a precisão da análise em contextos de maior escala e sugerem a necessidade de experimentos complementares para capturar de forma mais abrangente o desempenho da aplicação em um ambiente distribuído completo.

References

[1] O. Al-Debagy and P. Martinek, "A Comparative Review of Microservices and Monolithic Architectures," *arXiv preprint arXiv:1905.07997*, 2019.

[2] N. Bjørndal, L. J. P. de Araújo, A. Bucchiarone, N. Dragoni, M. Mazzara, and S. Dustdar, "Benchmarks and performance metrics for assessing the migration to microservice-based architectures," *Journal of Object Technology*, vol. 20, no. 2, pp. 2:1–17, 2021. DOI: 10.5381/jot.2021.20.2.a3.

[3] A. Akbulut and H. G. Perros, "Performance Analysis of Microservices Design Patterns," *IEEE Internet Computing*, 2018.

[4] D. K. Pandiya, "Performance Analysis of Microservices Architecture in Cloud Environments," *International Journal on Recent and Innovation Trends in Computing and Communication*, vol. 10, no. 12, pp. 264–274, 2022.

[5] J. P. (Java & Moi), "Architecture Microservices avec Spring Cloud," disponível em: https://javaetmoi.com/2018/10/architecture-microservices-avec-spring-cloud/, 2018. Acessado em: [inserir data].

[6] Spring Projects, "spring-petclinic/spring-petclinic-microservices: Distributed version of Spring Petclinic built with Spring Cloud," disponível em: https://github.com/spring-petclinic/spring-petclinic-microservices, 2025.

[7] R. Cargnelutti, G. Minuzzi, E. V. Agilar, F. P. Basso, and M. Bernardino, "Avaliação de Desempenho de API Gateways para o Gerenciamento de Acessos a Microsserviços," *Anais da Escola Regional de Engenharia de Software (ERES)*, 2024.

[8] Microsoft Corporation, "Microservices assessment and readiness," disponível em: https://learn.microsoft.com/en-us/azure/architecture/guide/technology-choices/microservices-assessment, 2025.