Avaliação de Desempenho do Spring PetClinic (Microservices) com Locust

Alef Cauan Sousa Rodriguês, Clara Emilly Sousa Sátiro, Mateus da Rocha Sousa
Curso de Sistemas de Informação
Universidade Federal do Piauí (UFPI) – Campus Senador Helvídio Nunes Barros
Picos, Piauí, Brasil
{alef.rodrigues, clara.satiro, mateus.sousa}@ufpi.edu.br

***Abstract*—Este trabalho tem como objetivo avaliar o desempenho do sistema *Spring PetClinic* na arquitetura de microsserviços, utilizando a ferramenta Locust como gerador de carga. Foram definidos três cenários distintos de teste: leve, moderado e de pico, a fim de analisar a escalabilidade e a estabilidade do sistema sob diferentes níveis de usuários simultâneos. As métricas observadas incluem o tempo médio e máximo de resposta, o número de requisições por segundo (req/s), o total de requisições atendidas e a taxa de sucesso. Os testes foram executados sobre os principais *endpoints* do sistema (consultas e cadastros de donos e veterinários), com cargas variando entre 50, 100 e 200 usuários virtuais. Os resultados obtidos evidenciam uma tendência de degradação de desempenho conforme o aumento da carga de usuários, oferecendo uma análise quantitativa dos limites operacionais da aplicação e destacando potenciais oportunidades de otimização.**

***Index Terms*—Microsserviços; Avaliação de Desempenho; Teste de Carga; PetClinic; Escalabilidade.**

I. Introduction

A arquitetura de microsserviços tem se tornado uma das abordagens mais adotadas no desenvolvimento de sistemas modernos, principalmente pela sua capacidade de promover modularidade, escalabilidade e manutenção independente entre os componentes. Diferentemente das arquiteturas monolíticas, os microsserviços permitem que cada funcionalidade seja executada de forma isolada, facilitando a implantação contínua e o balanceamento de carga. Contudo, essa flexibilidade também traz desafios relacionados à comunicação entre serviços, à orquestração de containers e ao monitoramento do desempenho em cenários de alta demanda.

Nesse contexto, avaliar o comportamento de sistemas baseados em microsserviços sob diferentes níveis de carga é essencial para compreender seus limites operacionais e antecipar gargalos de desempenho. O sistema PetClinic, amplamente utilizado como aplicação de referência em estudos de engenharia de software, representa um ambiente realista para experimentos de avaliação. Ele combina múltiplos serviços interconectados, o que o torna adequado para investigar aspectos de latência, taxa de erros e estabilidade.

O presente trabalho tem como objetivo analisar o desempenho do PetClinic em cenários progressivos de carga, simulando diferentes quantidades de usuários virtuais e períodos de execução. Por meio de testes controlados, buscou-se observar o comportamento do sistema quanto à sua capacidade de resposta, consumo de recursos e resistência à sobrecarga. Além de mensurar o impacto do aumento de usuários sobre a estabilidade do sistema, pretende-se fornecer subsídios que auxiliem no planejamento e na otimização de arquiteturas baseadas em microsserviços em contextos de produção.

Este trabalho foi desenvolvido como parte da disciplina de Sistemas Distribuídos na Universidade Federal do Piauí (UFPI).

II. Descrição do Sistema Testado

Esta seção apresenta o sistema utilizado para os testes de desempenho, detalhando sua arquitetura e os principais *endpoints* avaliados. As características de cada componente e a função de cada *endpoint* são descritas a seguir.

A. Arquitetura do PetClinic Microsserviços

O *Spring PetClinic* é uma aplicação de referência desenvolvida para demonstrar como o ecossistema *Spring* pode ser utilizado na construção de aplicações baseadas em bancos de dados. Sua versão em microsserviços exemplifica uma arquitetura moderna, modular e distribuída, características ideais para estudos que avaliem o desempenho e a escalabilidade de sistemas baseados em microsserviços. O sistema foi escolhido justamente por ser amplamente utilizado como referência em estudos de testes de desempenho. Sua implementação modular e uso de tecnologias adotadas no mercado permitem a avaliação de comportamento de uma arquitetura distribuída sob diferentes níveis de carga. Cada serviço possui uma responsabilidade específica e comunica-se com os demais por meio de APIs REST, conforme descrito a seguir:

- **Config Server**: oferece o gerenciamento centralizado das configurações de todos os microsserviços;
- **Discovery Server**: responsável pelo serviço de registro baseado no Eureka.
- **API Gateway**: responsável por receber as requisições dos clientes e encaminhá-las ao serviço correspondente;
- **Customers Service**: gerencia os dados dos clientes;
- **Vets Service**: gerencia as informações referentes aos veterinários;
- **Visits Service**: gerencia os registros de visitas dos animais de estimação;
- **GenAI Service**: fornece uma interface de chatbot integrada à aplicação;

Cada microsserviço é implementado utilizando *Spring Boot*, utilizando o *Spring Cloud* para comunicação e descoberta de serviços via *Eureka Server*. O tráfego entre os componentes é gerenciado pelo *API Gateway* citado anteriormente, permitindo a centralização da autenticação, bem como o balanceamento de carga. Essa estrutura modular e distribuída fornece uma base adequada para análises de desempenho, permitindo observar métricas como latência e vazão sob diferentes cargas, além de avaliar a robustez e escalabilidade da arquitetura.

B. Endpoints Avaliados

Os *endpoints* avaliados foram selecionados com o objetivo de representar diferentes tipos de interação entre os clientes e o sistema, baseando-se nas operações comumente executadas por usuários reais. Desse modo, é possível abranger tanto operações de leitura (*GET*), quanto de escrita (*POST*). O método *GET* é responsável por solicitar e retornar informações dos recursos existentes, representando a navegação e consulta de informações. Já o método *POST* envia dados ao servidor para a criação de novos registros, simulando as ações de cadastros realizadas pelos usuários.

As principais rotas utilizadas nos testes de carga foram:

- **GET /owners**: consulta a lista completa de donos de animais cadastrados;
- **GET /owners/id**: obtém os dados de um dono específico a partir de seu identificador;
- **GET /vets**: consulta todos os veterinários cadastrados;
- **POST /owners**: envia os dados de um novo dono para cadastro no sistema.

Esses *endpoints* foram escolhidos por representarem as principais operações executadas pelos usuários da aplicação (como consultas e cadastros), permitindo uma análise mais fiel do comportamento do sistema sob diferentes tipos de requisição e diferentes níveis de carga, aproximando o teste de um cenário real.

III. Metodologia dos Testes

A metodologia adotada consiste na execução de testes de carga e desempenho sobre o sistema Spring PetClinic – Microservices, em sua configuração padrão, com o intuito de medir a resposta do sistema sob diferentes níveis de concorrência de usuários. O processo foi dividido em três etapas principais:

1) Preparação do ambiente de execução
2) Configuração e execução dos testes de carga
3) Coleta e análise dos resultados.

Inicialmente, a aplicação foi implantada via Docker Compose, garantindo que todos os serviços (API Gateway, Customers, Vets e Visits) estivessem operacionais e integrados. Em seguida, o próprio sistema realizou um pré-carregamento do banco de dados com registros mínimos para assegurar que as requisições de leitura retornassem resultados válidos, evitando erros de requisições vazias. As execuções foram realizadas por meio da linha de comando, sem utilizar a interface Web do Locust. Cada cenário foi repetido 5 vezes, com os resultados exportados automaticamente em formato CSV para posterior análise estatística, permitindo também o acompanhamento em tempo real do comportamento das requisições.

Com o objetivo de manter a consistência entre as execuções e garantir que cada teste fosse realizado em condições equivalentes, foi adotado a seguinte abordagem: reiniciar a aplicação *Spring PetClinic* antes do início de cada execução. Essa prática visa mitigar possíveis anomalias decorrentes do acúmulo progressivo de dados gerados pelas diversas requisições *POST* ocorridas durante cada execução, podendo impactar os resultados das requisições *GET* ao aumentar o volume de informações retornadas e elevando consequentemente o tempo médio de resposta.

Visando manter a consistência dos testes entre cada execução, garantindo que cada uma seja realizada em condições semelhantes, foi utilizado a seguinte abordagem: reiniciar a aplicação do PetClinic antes de iniciar cada teste. Desse modo, é possível mitigar possíveis anomalias causadas por problemas como o aumento crescente de dados armazenados por meio das requisições *POST*, fazendo com que as requisições *GET* retornassem mais dados, contribuindo para o aumento do tempo de resposta.

As métricas coletadas de cada execução incluem o tempo médio de resposta (ms), o tempo máximo de resposta (ms), o throughput (req/s), o total de requisições atendidas, o percentual de erros (4xx e 5xx) e o percentual de sucesso. Cada cenário foi repetido 5 vezes para garantir maior confiabilidade estatística, sendo descartado da análise o período inicial de "aquecimento" das instâncias, conforme descrito na subseção de cenários de teste.

A. Ferramenta Utilizada

A ferramenta escolhida para geração de carga foi o Locust, uma solução open-source amplamente utilizada para testes de desempenho em sistemas web. O Locust permite simular múltiplos usuários virtuais realizando requisições HTTP de forma paralela, com controle preciso sobre o volume, o ritmo e o tipo de operação executada.

O arquivo de configuração principal (locustfile.py) foi elaborado com base no seguinte mix de tarefas:

- GET /owners (40%) — consulta à lista de donos de animais;
- GET /owners/id (30%) — detalhamento de um dono específico;
- GET /vets (20%) — consulta à lista de veterinários;
- POST /owners (10%) — cadastro de um novo dono.

Essa combinação representa um comportamento realista de uso do sistema, mesclando operações de leitura e escrita em proporções distintas. Durante as execuções, os resultados foram exportados em formato CSV, permitindo posterior análise e consolidação em planilhas e gráficos comparativos.

B. Cenários de Teste

Foram definidos três cenários de carga, denominados A (leve), B (moderado) e C (pico), com variação no número

de usuários simultâneos e no tempo de execução, expostos na Tabela I:

TABLE I
CENÁRIOS DE TESTE DEFINIDOS PARA AVALIAÇÃO DE DESEMPENHO.

Cenário	Usuários Virtuais	Duração Total	Aumento de usuários
Leve	50	10 min	5/s
Moderado	100	10 min	10/s
Pico	200	5 min	20/s

Durante cada execução, foram registradas as métricas principais de desempenho, conforme descrito na metodologia. O objetivo foi observar o comportamento do sistema ao dobrar progressivamente a quantidade de usuários, avaliando o impacto direto sobre o tempo de resposta, a taxa de requisições e o percentual de sucesso. Cada cenário foi executado repetidamente em condições controladas, sem alteração de configuração no ambiente, a fim de manter a reprodutibilidade dos resultados e reduzir interferências externas.

C. Ambiente de Execução

A Tabela II apresenta as especificações de hardware e software do computador utilizado para a realização dos testes de desempenho. A aplicação estava em um ambiente conteinerizado por meio do *Docker Compose*, garantindo a padronização do ambiente e a reprodutibilidade dos experimentos. Todas as dependências necessárias do sistema (como Java e Spring Boot) estavam encapsuladas nas imagens dos contêineres definidos no arquivo de composição. Dessa forma, assegura-se que a execução ocorra conforme a configuração original do desenvolvedor.

Os testes de carga foram realizados com a ferramenta Locust, executada na mesma máquina que hospedava os contêineres da aplicação. Essa escolha eliminou possíveis problemas das variações externas da rede, assegurando que o desempenho medido refletisse diretamente a capacidade da aplicação em lidar com requisições simultâneas. Durante as execuções, a máquina foi dedicada exclusivamente ao experimento, garantindo o isolamento dos recursos computacionais (como CPU e memória) e a consistência entre as repetições.

TABLE II
HARDWARE E SOFTWARE UTILIZADOS NO AMBIENTE DE EXECUÇÃO DOS TESTES

Componente	Especificação
Modelo	Acer Aspire A515-45
Processador	AMD Ryzen™ 7 5700U with Radeon™ Graphics × 16
Memória RAM	12,0 GiB
Sistema Operacional	Ubuntu 24.04 LTS
Docker	Versão 28.1.1
Docker Compose	Versão 2.31.0-desktop.2
Locust	Versão 2.42.0

IV. RESULTADOS E ANÁLISES

Nesta seção, os resultados dos testes de carga são apresentados e analisados. A avaliação foca-se nas métricas-chave de desempenho que caracterizam a resposta do sistema sob diferentes níveis de estresse. As principais métricas avaliadas incluem a latência média de resposta (ms), o *throughput* (requisições por segundo, ou req/s) e a estabilidade do sistema ao longo do tempo. A análise compara o comportamento da aplicação nos três cenários de carga definidos: Leve (A), Moderado (B) e Pico (C).

A. Métricas Coletadas

Durante as execuções, o Locust gerou relatórios CSV com métricas detalhadas de desempenho, permitindo o cálculo de médias e desvios padrão entre execuções. As principais métricas analisadas estão descritas abaixo:

Tempo médio de resposta (ms)
: Indica o tempo médio necessário para que uma requisição seja processada e respondida com sucesso.

Tempo máximo de resposta (ms)
: Representa o maior tempo de resposta registrado durante os testes.

Requisições por segundo (req/s)
: Corresponde à taxa de processamento de requisições concluídas por segundo.

Total de requisições
: Número total de requisições enviadas durante o cenário.

Erros (4xx / 5xx)
: Quantidade de respostas com falhas.

Taxa de sucesso (%)
: Proporção de requisições bem-sucedidas.

B. Análise de Métricas Agregadas

Os dados agregados dos testes fornecem uma visão sumarizada da capacidade e escalabilidade do sistema.

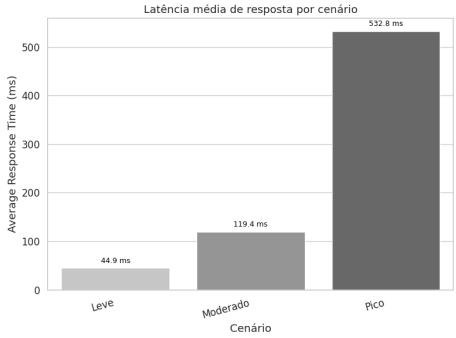

Fig. 1. Comparativo da latência média de resposta (ms) para cada cenário de carga (Leve, Moderado, Pico).

Os resultados consolidados (extraídos da Fig. 1, Fig. 3 e Fig. 2) são a base desta análise quantitativa. Observa-se na Fig. 1 um aumento não linear e pronunciado na latência à medida que a carga de usuários é incrementada. Ao dobrar o número de usuários de 50 (Cenário A) para 100 (Cenário B), o tempo médio de resposta aumentou de 44.9 ms para 119.4 ms, um incremento de aproximadamente 166%.

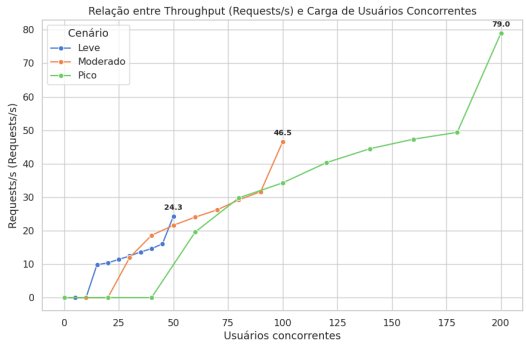

Fig. 2. Relação entre o *throughput* (Requests/s) e o número de usuários concorrentes, demonstrando a escalabilidade do sistema.

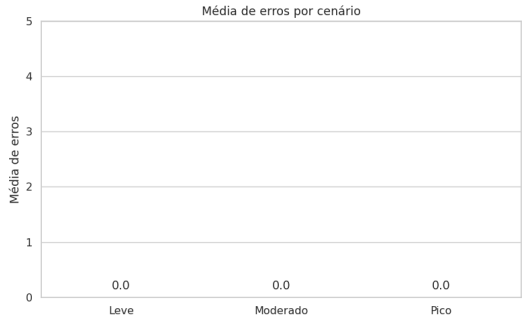

Fig. 3. Relação entre a quantidade de erros obtida para cada cenário de teste.

Uma degradação de desempenho significativamente mais severa ocorre ao dobrar a carga novamente para 200 usuários (Cenário C). Neste cenário, o tempo médio de resposta atingiu 532.8 ms. Isso representa um aumento de 346% em relação ao Cenário B e uma lentidão 11.8 vezes superior à linha de base do Cenário A, indicando que o sistema ultrapassou seu ponto de operação eficiente.

Em termos de *throughput*, a Fig. 2 mostra que o Cenário A (50 usuários) estabilizou em 24.3 req/s, e o Cenário B (100 usuários) alcançou 46.5 req/s. O Cenário C (200 usuários) atingiu um pico máximo de 79.0 req/s, falhando em dobrar a vazão do Cenário B. Isso demonstra uma clara perda de eficiência e a aproximação de um limite de capacidade.

Adicionalmente, a Fig. 3 apresenta a média de erros observada em cada cenário de carga. Nota-se que, em todos os testes, a quantidade média de erros permaneceu igual a zero, indicando que o sistema manteve uma taxa de sucesso de 100% em todas as execuções. Esse resultado reforça a robustez da arquitetura do PetClinic sob diferentes níveis de estresse, demonstrando que, apesar do aumento expressivo na latência e da perda de eficiência em *throughput* no Cenário C, não houve falhas de requisição nem respostas inválidas (*HTTP 4xx/5xx*).

C. Análise de Tendências Temporais

Os gráficos de tendência temporal ilustram o comportamento dinâmico do sistema sob estresse contínuo.

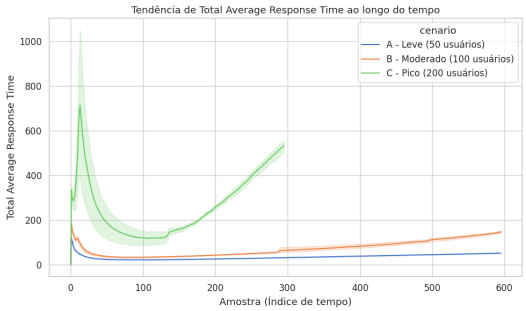

Fig. 4. Tendência temporal da latência média de resposta (Total Average Response Time) por cenário.

Fig. 5. Tendência temporal do *throughput* (Requests/s) por cenário ao longo da execução do teste.

O gráfico da Fig. 4 evidencia a estabilidade do sistema sob cargas leve e moderada. Os Cenários A e B mantêm uma latência de resposta relativamente plana e previsível ao longo dos 10 minutos de teste. O Cenário A permanece consistentemente abaixo de 50 ms. O Cenário B exibe uma leve tendência linear de crescimento, mas sem picos anômalos. Em contraste, o Cenário C (Pico) demonstra instabilidade imediata, com alta variabilidade (representada pela área sombreada) e um crescimento exponencial da latência após a metade do teste (a partir do índice 150), indicando saturação de recursos.

De forma correlata, a Fig. 5 apresenta a vazão do sistema. Os Cenários A e B atingem rapidamente um platô estável (aprox. 25 req/s e 50 req/s, respectivamente), sustentando

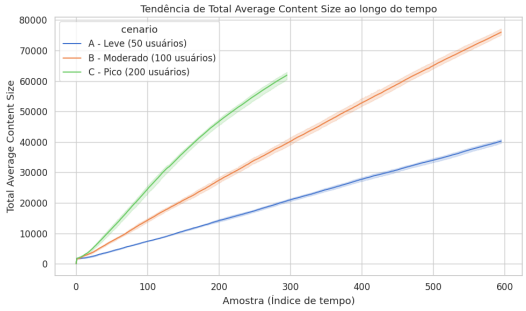

Fig. 6. Tendência do tamanho médio acumulado do conteúdo (Total Average Content Size) transferido ao longo do tempo.

essa taxa. O Cenário C, no entanto, após atingir um pico de vazão (próximo a 100 req/s) durante o *ramp-up*, apresenta uma degradação acentuada. Coincidindo com o aumento exponencial da latência (Fig. 4), o *throughput* do Cenário C decai para níveis próximos de 50-55 req/s, sugerindo que o sistema entrou em contenção.

Adicionalmente, a Fig. 6 ilustra a tendência do tamanho médio acumulado de conteúdo. Observa-se que o volume de dados transferidos cresce de forma constante em todos os cenários, sendo a taxa de crescimento (inclinação da curva) proporcional à carga de usuários. O Cenário C apresenta o crescimento mais acentuado, alinhado com sua maior taxa de requisições, embora o teste tenha sido interrompido prematuramente (próximo ao índice 300), em razão do seu tempo de execução ser metade dos outros cenários.

Finalmente, a "Relação entre Throughput e Carga" (Fig. 2) sintetiza a escalabilidade da aplicação. A relação é quase perfeitamente linear até 100 usuários, indicando que, nessa faixa, o sistema escala eficientemente com a carga. Contudo, a curva do Cenário C (verde) perde inclinação, demonstrando a lei dos rendimentos decrescentes; o sistema não consegue mais processar o dobro das requisições ao dobrar o número de usuários, confirmando que um gargalo foi atingido.

V. Discussão e Conclusões

A análise dos resultados experimentais permite uma avaliação crítica do comportamento da arquitetura de microsserviços do Spring PetClinic sob diferentes níveis de carga. Os resultados demonstram que o sistema mantém estabilidade operacional e previsibilidade de desempenho até aproximadamente 100 usuários simultâneos (Cenário B). Nesta faixa, o *throughput* escala linearmente com a carga e a latência aumentando de forma controlada.

O Cenário C (200 usuários) expõe um ponto de inflexão claro. O comportamento do sistema deixa de ser linear e sofre uma degradação abrupta. A partir de 150 índices de tempo (Fig. 4 e 5), observa-se um aumento exponencial na latência média e uma queda no *throughput* sustentado. Este comportamento é clássico de um sistema que atingiu saturação. O gargalo pode estar localizado no API Gateway, que precisa rotear todo o tráfego, ou em um dos serviços de maior demanda (como *Customers*, que concentra 70% das requisições *GET* e *POST*), onde pode ocorrer exaustão de *threads* de aplicação ou do *pool* de conexões com o banco de dados.

Conclui-se que a arquitetura avaliada, no ambiente de teste configurado, é robusta para cargas leves e moderadas, mas não escala eficientemente para 200 usuários concorrentes. A degradação de desempenho é severa e indica a necessidade de otimização (e.g., *tuning* de JVM, *auto-scaling* de *pods*) ou provisionamento de mais recursos para suportar cargas de pico.

VI. Limitações e Trabalhos Futuros

Os resultados obtidos oferecem uma visão relevante sobre o comportamento do PetClinic sob diferentes níveis de carga, porém algumas limitações devem ser consideradas. Primeiramente, os testes foram realizados em ambiente controlado, o que reduz a influência de fatores externos, como latência de rede real e variações de hardware, mas também limita a representatividade dos resultados em ambientes de produção. Além disso, a análise concentrou-se em um conjunto restrito de métricas, priorizando tempo médio de resposta e taxa de erros, sem incluir aspectos relacionados ao consumo de CPU, memória e rede, que poderiam enriquecer a compreensão sobre o desempenho geral do sistema.

Outra limitação relevante refere-se à ausência de mecanismos de orquestração de microsserviços, como Kubernetes ou Docker Swarm, que poderiam alterar significativamente a forma como o sistema responde a picos de carga. Também não foram aplicadas técnicas de observabilidade contínua, como o uso de Prometheus ou Grafana, que possibilitariam a coleta em tempo real de métricas e logs para análise mais detalhada.

Como trabalhos futuros, propõe-se a expansão da análise para ambientes distribuídos, incluindo a execução de testes em infraestrutura de nuvem, onde fatores de rede e escalabilidade dinâmica possam ser considerados. Recomenda-se ainda a integração de ferramentas de monitoramento para uma observação contínua do comportamento do sistema, além da inclusão de cenários de falhas controladas, com o objetivo de avaliar a resiliência e a capacidade de recuperação dos serviços. Por fim, pretende-se comparar o desempenho do PetClinic com outras aplicações baseadas em microsserviços, buscando identificar boas práticas e estratégias de otimização aplicáveis a diferentes contextos.