AVALIAÇÃO DE PADRÕES DE ARQUITETURA DE SOFTWARE APLICADOS NO DESENVOLVIMENTO DE E-COMMERCES

Renan M. Thomas[1], **Leanderson André** [2]

[1]Bacharelado em Engenharia de Software

[2]Universidade da Região de Joinville (UNIVILLE) – Joinville, SC – Brazil

renan.thomas@univille.br, leandersonandre@univille.br

Resumo. *Este estudo avalia a performance de duas aplicações de e-commerce por meio de testes de carga: uma desenvolvida com arquitetura de microsserviços e outra com arquitetura monolítica. As métricas avaliadas foram tempo de resposta, latência, taxa de transferência e número de pacotes recebidos e enviados. Foram utilizados testes paramétricos por meio do RStudio, para interpretar os dados da simulação e determinar se as diferenças observadas entre os padrões de arquitetura foram estatisticamente significativas. Os resultados mostram que a arquitetura monolítica teve melhor performance nesses aspectos, mas apresentou um percentual de erro maior sob cargas mais altas, embora sem significância estatística.*

Palavras-chave: E-commerce, Arquitetura de microsserviços, Arquitetura monolítica, Testes de carga, Performance

Abstract. *This study evaluates the performance of two e-commerce applications through load tests: one developed with a microservices architecture and the other with a monolithic architecture. The metrics evaluated were response time, latency, throughput and number of packets received and sent. Parametric tests were used using RStudio to interpret the simulation data and determine whether the differences observed between the architecture patterns were statistically significant. The results show that the monolithic architecture performed better in these aspects, but had a higher percentage of errors under higher loads, although without statistical significance.*

Keywords: E-commerce, Microservices architecture, Monolithic architecture, Load tests, Performance

1. Introdução

As aplicações de comércio eletrônico, popularmente conhecidas como e-commerces, são softwares que permitem a realização de transações comerciais pela internet. O comércio eletrônico pode ser definido como a realização de negócios, compras e vendas, usando redes de computadores, incluindo a internet. (KENNETH; GUERCIO, 2019)

Visando garantir que essas aplicações atendam às necessidades crescentes dos usuários, sejam fáceis de manter e evoluir e tenham qualidade satisfatória, é fundamental que essas empresas sigam as melhores práticas da engenharia de software e utilizem uma arquitetura de software adequada. (MAXIM; PRESSMAN, 2021)

Os padrões de arquitetura desempenham um papel crucial na definição das características fundamentais e do comportamento de uma aplicação. Alguns padrões de arquitetura são intrinsecamente adequados para aplicativos altamente escaláveis, enquanto outros são mais apropriados para aplicativos altamente ágeis. (RICHARDS, 2015)

No contexto de um e-commerce, existem requisitos específicos que levam à busca por padrões de arquitetura, desde características explícitas como a necessidade de escalabilidade, elasticidade e performance, até características implícitas como confiabilidade, disponibilidade e segurança. (RICHARDS; FORD, 2020). A performance é um fator crítico para o sucesso de qualquer e-commerce, pois impacta diretamente na satisfação dos usuários finais. (AMJAD et al., 2021)

Entretanto, a seleção de um padrão de arquitetura para uma aplicação de e-commerce pode ser um desafio. Isso se deve ao fato de que diferentes padrões de arquitetura podem ser aplicados a um mesmo sistema, e a escolha do padrão ideal depende das características específicas de cada projeto, da experiência do arquiteto de software e dos requisitos específicos do sistema. (SOMMERVILLE, 2011)

Conhecer as características, os pontos fortes e os pontos fracos de cada padrão de arquitetura é necessário para escolher aquele que atenda às necessidades específicas de seu negócio necessidades e objetivos específicos. (RICHARDS, 2015)

Apesar da existência de diversos estudos que avaliam a performance de diferentes arquiteturas, poucos se concentram nas especificidades do e-commerce. Estudos como os de Cabane e Farias (2022) e Al-Debagy e Martinek (2018) propõem a avaliação da performance entre diferentes padrões de arquitetura, através de testes de carga, usando métricas previamente selecionadas. Essas pesquisas fornecem uma base sólida sobre os benefícios e desafios de cada arquitetura em termos de performance geral. Contudo, esses estudos não abordaram testes de carga focados exclusivamente em e-commerces.

O setor de e-commerce possui características e demandas específicas, como alta variabilidade no tráfego, picos de acesso em períodos promocionais e a necessidade de tempos de resposta extremamente rápidos para garantir uma boa experiência do usuário. (KENNETH; GUERCIO, 2019)

Nesse contexto, foi desenvolvido este artigo, que tem como objetivo avaliar a performance de padrões de arquitetura aplicados a um cenário de e-commerce utilizando testes de performance. Para isso, foram executadas e avaliadas duas aplicações de e-commerces. Uma utilizando uma arquitetura de microsserviços e a outra utilizando uma arquitetura monolítica. O foco deste trabalho e a questão de pesquisa será como avaliar a performance de uma arquitetura de software para sistemas de e-commerce. Além disso, a revisão da literatura apresentará conceitos relevantes. O procedimento metodológico detalhará a elaboração dos cenários de testes, a definição das métricas de performance e a

seleção dos testes estatísticos. A análise dos resultados contemplará os resultados obtidos nos testes, seguidos de discussão. Por fim, na conclusão, será destacada a arquitetura que alcançou os melhores resultados, juntamente com sugestões para melhorias futuras.

2. Revisão da literatura

Nesta seção, discutem-se os padrões de arquitetura em e-commerce, com foco nas arquiteturas de microsserviços e monolítica, destacando suas principais características. Adicionalmente, são abordados os testes de performance, com especial atenção aos testes de carga, utilizados para avaliar a performance das arquiteturas mencionadas. Além disso, apresentam-se os conceitos de testes paramétricos e do teste t, empregados para analisar estatisticamente os resultados obtidos nos testes.

2.1. Padrões de Arquitetura

Os padrões de arquitetura são soluções genéricas e reutilizáveis para problemas recorrentes em projetos de software, que envolvem a estruturação de componentes e subsistemas de um sistema, e são definidos como um conjunto de decisões de design que têm um impacto significativo na performance, escalabilidade, segurança, facilidade de manutenção e outras características de qualidade do sistema. (BASS; CLEMENTS; KAZMAN, 2012)

O design arquitetônico de grandes sistemas sempre foi crucial para seu sucesso, e optar por uma arquitetura inadequada pode resultar em consequências desastrosas. (GARLAN; PERRY, 1995)

Alguns exemplos de padrões de arquitetura comuns são: Arquitetura em camadas, arquitetura monolítica, arquitetura orientada a serviços, arquitetura de microsserviços e arquitetura orientada a eventos. (FORD; PARSONS; KUA, 2017)

2.1.1. Arquitetura de Microsserviços

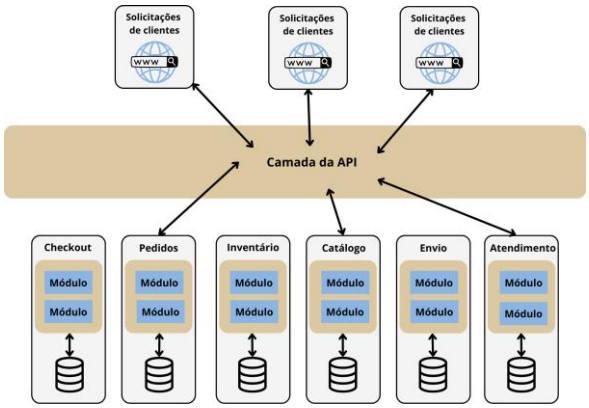
O padrão de arquitetura baseado em microsserviços foi criado com o objetivo de superar os desafios apresentados por aplicações monolíticas e pela arquitetura orientada a serviços (SOA). (RICHARDS, 2015)

A essência dos microsserviços reside na ideia de contexto limitado, onde cada serviço representa um domínio ou processo específico. Cada microsserviço é autossuficiente, contendo todas as classes, subcomponentes e esquemas de base de dados necessários para sua operação. Em arquiteturas monolíticas, é usual que classes comuns sejam compartilhadas entre diferentes partes do aplicativo. Contudo, em microsserviços, busca-se minimizar o acoplamento. Assim, um arquiteto focado em microsserviços pode optar por duplicar classes ao invés de compartilhá-las, para evitar dependências. (RICHARDS; FORD, 2020)

Na Figura 1 está exemplificado um modelo genérico de arquitetura de microsserviços. Nesse modelo, os componentes de serviço, ou apenas serviços, podem variar em granularidade. Em outras palavras, um componente de serviço pode representar um único módulo ou até uma grande parte da aplicação. Esses componentes de serviço podem conter um ou mais módulos com uma função específica como, por exemplo, fornecer informações climáticas de uma cidade. Concomitantemente, outro serviço pode ser res-

ponsável por executar operações relacionadas à colocação de negociações de ações ou determinar as taxas de seguro automotivo. (RICHARDS, 2015)

Figura 01 - Modelo genérico da arquitetura de microsserviços



Fonte: Ford; Parsons e Kua (2017)

Por ser uma arquitetura distribuída, todos os componentes dentro dessa arquitetura são totalmente desacoplados uns dos outros e só podem ser acessados através de algum tipo de protocolo de acesso remoto, como JMS, AMQP, REST, SOAP, RMI, entre outros. (RICHARDS, 2015)

Os microsserviços estruturam uma aplicação como uma coleção de serviços altamente mantíveis e testáveis, que são independentes em termos de implantação e podem ser desenvolvidos por uma pequena equipe. A divisão da funcionalidade de uma aplicação em serviços independentes pode simplificar o processo de implantação e o tempo de inatividade do serviço. (FOWLER, 2014)

2.1.2. Arquitetura Monolítica

Em uma arquitetura monolítica, ou monolito, existe apenas uma única unidade de implementação que engloba todas as funcionalidades do sistema, operando em um único processo e é frequentemente conectada a uma única base de dados. Ainda assim, esse tipo de arquitetura tende a facilitar e acelerar o início de um projeto, por conta de sua baixa complexidade. (RICHARDS; FORD, 2020)

Na Figura 2 está exemplificado um modelo genérico de arquitetura de monolítica. Os sistemas monolíticos destacam-se pela facilidade no desenvolvimento da aplicação, uma vez que dispõem de muitas ferramentas convencionais adaptadas a essa arquitetura. Além disso, oferecem facilidade na implementação, já que consistem em um único conjunto de código com módulos interdependentes. (RICHARDSON, 2018)

Figura 02 - Modelo genérico da arquitetura de monolítica

Fonte: Ingeno (2018)

Por outro lado, em uma arquitetura monolítica as classes costumam ser fortemente acopladas, ou seja, não é possível alterar ou remover uma classe sem entender e alterar muitas outras. Com isso, com o passar do tempo, o sistema se torna uma massa densa que é difícil de aprender, portar e manter. (GAMMA et al., 1993)

Com uma aplicação monolítica, se houver a necessidade de uma nova linguagem de programação, banco de dados ou framework, qualquer mudança impactará uma grande parte do meu sistema. (NEWMAN, 2021)

A maioria das histórias de sucesso de microsserviços começa com um monolito que cresceu muito e, eventualmente, foi dividido em partes menores. (FOWLER, 2015)

2.2. Requisitos de um E-commerce

O comércio eletrônico pode ser definido como transações comerciais habilitadas digitalmente entre organizações e indivíduos (LAUDON; TRAVEL, 2017). Elas se caracterizam por sistemas de software que facilitam transações comerciais por meio da internet, trazendo consigo funcionalidades essenciais para o efetivo funcionamento do comércio online (KENNETH; GUERCIO, 2019). Essa definição abrangente destaca a importância do comércio eletrônico não apenas para varejistas e consumidores, mas também para uma ampla gama de outras atividades comerciais.

Os objetivos de negócios de um site de comércio eletrônico são semelhantes aos de um estabelecimento de varejo típico. A funcionalidade do sistema e as necessidades de informação são onde estão as verdadeiras diferenças. (KENNETH; GUERCIO, 2019)

2.3. Teste de carga

Este é o teste de performance clássico, onde a aplicação é submetida a uma carga até atingir a concorrência desejada, sem ultrapassá-la. O objetivo é alcançar as metas de performance em termos de disponibilidade, concorrência ou taxa de transferência e tempo de resposta. O teste de carga é a abordagem mais próxima do uso real da aplicação e geralmente inclui uma simulação dos efeitos da interação do usuário com o cliente da aplicação. Isso envolve replicar os atrasos e pausas vivenciados durante a entrada de dados, bem como as respostas (humanas) às informações fornecidas pelos servidores da aplicação. (MOLYNEAUX, 2014)

2.4. Testes paramétricos

Um teste paramétrico é aquele que requer que os dados sigam uma das grandes distribuições estatísticas conhecidas. Para que um teste seja considerado paramétrico, certas suposições sobre a distribuição dos dados devem ser verdadeiras. A maioria dos testes paramétricos baseados na distribuição normal tem quatro pressupostos básicos que devem ser atendidos para que o teste seja preciso. Que seriam: Dados normalmente distribuídos, Homogeneidade de variância, Dados de intervalo e In- dependência. (A. FIELD; Z. FIELD; MILES, 2012)

O teste de Shapiro-Wilk verifica a normalidade da distribuição com base na estrutura de covariâncias entre as estatísticas de ordem da distribuição normal. (BECKER, 2015)

Para verificar a homogeneidade das variancias, o teste mais utilizado, estando embutido em quase todos os pacotes estatisticos populares, é o teste de Levene. (BECKER, 2015)

2.4.1. Teste T

O teste t é uma ferramenta estatística versátil usada para várias comparações. Ele pode testar se um coeficiente de correlação é diferente de zero, verificar se um coeficiente de regressão b é diferente de zero, e comparar médias de dois grupos. (A. FIELD; Z. FIELD; MILES, 2012)

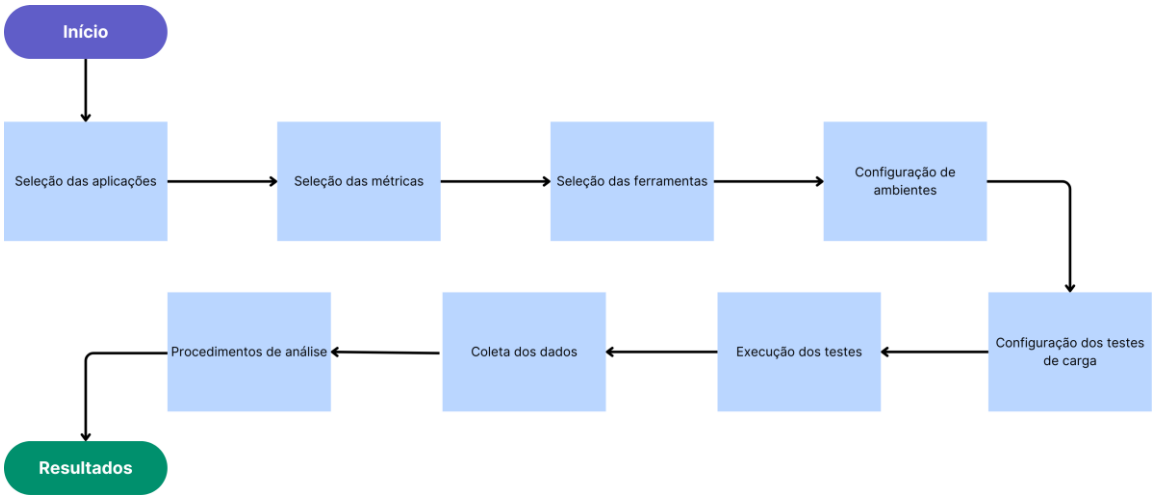
O teste t de Welch não pressupõe a homogeneidade de variâncias, o que o torna mais robusto em situações onde as variâncias dos grupos comparados são diferentes. Este teste é uma alternativa ao teste t de Student quando a suposição de variâncias iguais é violada. O teste t de Welch ajusta os graus de liberdade de forma a compensar as diferenças nas variâncias, proporcionando uma análise mais precisa em tais casos. Ambos os testes t são fundamentais para comparar médias entre dois grupos, sendo o teste t de Student adequado para variâncias homogêneas e o teste t de Welch para variâncias heterogêneas. (A. FIELD; Z. FIELD; MILES, 2012)

3. Procedimentos metodológicos

O trabalho foi desenvolvido seguindo um procedimento metodológico de avaliação comparativa com resultados quantitativos, delineado como uma pesquisa experimental. A metodologia deste estudo foi inspirada no trabalho de Cabane e Farias (2022), que propõe a avaliação da performance entre diferentes padrões de arquitetura através de testes de carga. Da mesma forma que nos estudos mencionados, utiliza-se a ferramenta Apache JMeter para realizar testes de carga, medindo métricas como tempo de resposta, latência, taxa de transferência e número de pacotes recebidos e enviados.

Na Figura 03, detalham-se as etapas da metodologia:

Figura 03 - Etapas da metodologia



Fonte: Cabane e Farias (2022)

3.1. Seleção das aplicações

Para realizar a comparação entre as arquiteturas, foram selecionados dois projetos de e- commerce de código aberto como exemplos representativos: o Northern Mountains,

que adota uma arquitetura baseada em microsserviços, e o eShopOnWeb, que segue uma abordagem monolítica. Ambos os projetos têm seus códigos-fonte disponíveis publicamente no GitHub, uma plataforma popular para compartilhamento de software de código aberto.

O e-commerce eShopOnWeb, que utiliza uma arquitetura monolítica, foi desenvolvido em C# com o framework ASP.NET Core 8.0. O sistema gerenciador de banco de dados utilizado é o SQL Server.

O aplicativo de microsserviços, Northern Mountains, também foi desenvolvido em C# com o framework ASP Net Core 8.0. A funcionalidade do aplicativo é dividida em muitos microsserviços distintos. Há serviços responsáveis pela autenticação e identidade, listagem de itens do catálogo de produtos, gerenciamento das cestas de compras dos usuários e realização de pedidos. Cada um desses serviços separados tem seu próprio armazenamento persistente em um banco de dados PostgreSQL. Não há um armazenamento de dados primário com o qual todos os serviços interagem. Em vez disso, a coordenação e a comunicação entre os serviços serão feitas conforme a necessidade e usando um barramento de mensagens.

3.2. Seleção das métricas

Esta etapa visa selecionar as métricas que permitirão uma análise detalhada dos aspectos de performance de cada aplicativo alvo. As métricas escolhidas para este estudo foram baseadas nas utilizadas por Cabane e Farias (2022), com a exceção do uso de CPU e memória, pois essas métricas não são disponibilizadas nativamente pelo JMeter.

No Quadro 01, constam as métricas e suas respectivas definições utilizadas para avaliação dos testes de carga.

Quadro 01 - Métricas de performance

Métricas	**Definição**
Tempo de resposta	Medida de quão responsivo um aplicativo ou subsistema é para uma solicitação do cliente.
Percentual de erro	Porcentagem de requisições que resultaram em falhas em relação ao total de requisições executadas durante o teste de carga.
Latência	Medida de capacidade de resposta que representa o tempo que leva para concluir a execução de uma solicitação.
Taxa de transferência	Número de unidades de trabalho que podem ser processadas por unidade de tempo.
Pacotes recebidos	Indica a quantidade de pacotes de dados recebidos pelo cliente (JMeter) do servidor.
Pacotes enviados	Representa a quantidade de pacotes de dados enviados pelo cliente (JMeter) ao servidor.

Fonte: Primária (2024)

3.3. Seleção das ferramentas

Para a criação e execução dos testes de carga foi utilizado a ferramenta Apache JMeter. JMeter é uma ferramenta que realiza testes de carga e de estresse em recursos estáticos ou dinâmicos oferecidos por sistemas.

3.4. Configuração de ambientes

Os testes foram conduzidos em um ambiente padronizado para garantir resultados comparáveis e uma análise precisa. Todos os experimentos foram realizados em um único ambiente de teste, no qual foram mantidas as mesmas condições e configurações. As aplicações foram executadas localmente em contêineres por meio do Docker, uma ferramenta que permite a criação e gestão de ambientes isolados para execução de aplicações, garantindo consistência e portabilidade. A máquina utilizada para os testes era equipada com um processador Intel(R) Core(TM) i5-1135G7 com 2.40GHz e com 12GB de RAM, executando o sistema operacional Windows 11 Pro.

3.5. Configuração dos testes de carga

Como o aplicativo de microsserviços e a aplicação monolítica compartilham as mesmas funcionalidades básicas de um comércio eletrônico, foram definidos os seguintes cenários de teste para cada aplicação dentro do JMeter, conforme o Quadro 02:

Quadro 02 - Cenário de teste no Jmeter

1°	Acessar página inicial
2°	Acessar página de login
3°	Logar
4°	Consultar catálogo
5°	Adicionar produto ao carrinho
6°	Consultar carrinho
7°	Finalizar pedido
8°	Consultar pedidos
9°	Fazer logout

Fonte: Primária (2024)

Para simular um cenário de teste mais próximo da experiência real de um e-commerce, cada usuário gerado pelo JMeter foi configurado com credenciais exclusivas, incluindo um nome de usuário e senha únicos. Isso se estende à operação de adicionar produtos ao carrinho, onde cada usuário adiciona um item distinto. Isso garante que as interações reflitam com precisão o comportamento individual dos usuários durante uma sessão de compra.

Foram preparadas seis execuções para cada arquitetura. Cada execução do cenário de teste simula o acesso simultâneo de 5, 15, 25, 50, 100 e 125 usuários. Dessa forma, o comportamento das duas aplicações pode ser observado ao reagir com o aumento do processamento simultâneo.

3.6. Execução dos testes

Nesta etapa, o objetivo foi executar os cenários de teste de cada aplicativo alvo. A execução desses casos de teste permitiu a coleta de dados sobre a performance dos aplicativos, através da ferramenta de teste Apache JMeter.

3.7. Coleta dos dados

Esta etapa teve como objetivo coletar os dados gerados durante a execução dos cenários de teste no JMeter. Para cada execução de teste de carga, foram coletados dados das métricas selecionadas, agrupados por métrica e cenário de teste.

3.8. Procedimentos de análise

Para interpretar os dados da simulação e determinar se as diferenças observadas entre os padrões de arquitetura são estatisticamente significativas, foram utilizados cálculos estatísticos através do RStudio. Especificamente, foram realizados testes de hipóteses, incluindo o teste t de Welch e o teste t de Student, para comparar as médias das métricas de performance entre as arquiteturas de microsserviços e monolítica.

Inicialmente, aplicou-se o teste de Shapiro-Wilk para verificar a normalidade dos dados de cada métrica de performance, que incluem Tempo de Resposta, Percentual de Erro, Latência, Taxa de Transferência, Pacotes Recebidos e Pacotes Enviados. Os resultados revelaram que todos os p-valores foram superiores a 0,05, sugerindo que os dados podem ser considerados normalmente distribuídos.

Posteriormente, utilizou-se o teste de Levene para examinar a homogeneidade das variâncias entre os grupos. Os resultados indicaram p-valores de 0,02866 para Tempo de Resposta, 0,01468 para Percentual de Erro, 0,008135 para Latência, 0,06235 para Taxa de Transferência, 0,001757 para Pacotes Recebidos e 0,02355 para Pacotes Enviados. Exceto pela Taxa de Transferência, que apresentou um p-valor de 0,06235, todas as outras métricas exibiram variâncias heterogêneas (p-valores inferiores a 0,05).

Com base nesses resultados, foram selecionados os testes estatísticos adequados para a análise. Para a métrica de Taxa de Transferência, foi selecionado o teste T de Student, assumindo variâncias iguais. Já para as demais métricas, optou-se pelo teste de Welch, apropriado quando não há pressuposto de igualdade das variâncias entre os grupos.

Além disso, conforme o Quadro 03, foram definidas as seguintes hipóteses para cada uma das métricas de performance:

Quadro 03 - Hipóteses para as métricas de performance

Métrica de Performance	Hipótese Nula (H0)	Hipótese Alternativa (H1)
Tempo de Resposta	Não há diferença significativa nas médias de tempo de resposta entre as arquiteturas de microsserviços e monolítica.	Há diferença significativa nas médias de tempo de resposta entre as arquiteturas de microsserviços e monolítica.
Percentual de Erro	Não há diferença significativa nas médias do percentual de erro entre as arquiteturas de microsserviços e monolítica.	Há diferença significativa nas médias do percentual de erro entre as arquiteturas de microsserviços e monolítica.
Latência	Não há diferença significativa nas médias de latência entre as arquiteturas de microsserviços e monolítica.	Há diferença significativa nas médias de latência entre as arquiteturas de microsserviços e monolítica.
Taxa de Transferência	Não há diferença significativa nas médias de taxa de transferência entre as arquiteturas de microsserviços e monolítica.	Há diferença significativa nas médias de latência entre as arquiteturas de microsserviços e monolítica.
Pacotes Recebidos	Não há diferença significativa nas médias de pacotes recebidos entre as arquiteturas de microsserviços e monolítica.	Há diferença significativa nas médias de pacotes recebidos entre as arquiteturas de microsserviços e monolítica.
Pacotes Enviados	Não há diferença significativa nas médias de pacotes enviados entre as arquiteturas de microsserviços e monolítica.	Há diferença significativa nas médias de pacotes enviados entre as arquiteturas de microsserviços e monolítica.

Fonte: Primária (2024)

4. Análise dos dados e discussão dos resultados

A análise estatística dos resultados presentes no Quadro 04 revela que a arquitetura monolítica parece ter uma performance significativamente melhor em várias métricas em comparação com a arquitetura de microsserviços. O quadro contém todos os resultados obtidos nos testes e está dividido em duas seções principais: a primeira seção mostra os resultados da arquitetura de microsserviços, e a segunda seção apresenta os resultados da arquitetura monolítica. As colunas foram divididas com os diferentes níveis de carga de usuários utilizados, de 5 até 125 usuários. Nas linhas, estão as métricas de performance que foram avaliadas: tempo de resposta, percentual de erro, latência, taxa de transferência, pacotes recebidos e pacotes enviados.

Quadro 04 - Resultados obtidos nos cenários de testes

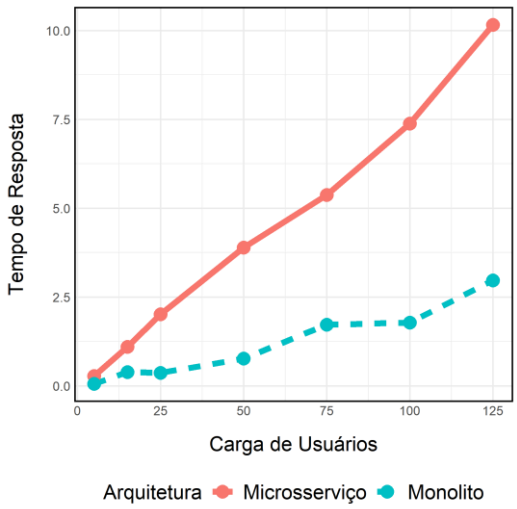
Microsserviços							
Métricas	5 usuários	15 usuários	25 usuários	50 usuários	75 usuários	100 usuários	125 usuários
Tempo de resposta (/s)	0,279	1,098	2,017	3,89	5,37	7,38	10,16
Percentual de erro (%)	0,00%	0,00%	0,00%	0,36%	0,48%	0,37%	1,02%
Latência (s)	0,156	0,555	1,094	1,97	2,48	3,72	4,83
Taxa de transferência (/s)	14,3/s	11,6/s	11,3/s	10,9/s	9,6	9,8/s	8,5/s
Pacotes recebidos (KB/s)	142,56	114,59	111,47	109,2	96,14	98,7	86,47
Pacotes enviados (KB/s)	146,2	119,35	116,26	110,68	97,08	99,59	86,36
Monolito							
Métricas	5 usuários	15 usuários	25 usuários	50 usuários	75 usuários	100 usuários	125 usuários
Tempo de resposta (/s)	0,057	0,386	0,367	0,767	1,72	1,78	2,97
Percentual de erro (%)	0,00%	0,00%	0,00%	2,22%	5,48%	6,46%	11,19%
Latência (s)	0,01369	0,09661	0,08003	0,17	0,505	0,354	0,528
Taxa de transferência (/s)	30,6/s	30,9/s	53,4/s	54,7/s	38,9/s	21,6/s	31,4
Pacotes recebidos (KB/s)	7839,04	7934,12	13685,79	14010,29	9420,26	7055,54	12464,68
Pacotes enviados (KB/s)	956,65	1026,21	1793,55	1852,54	1316,8	723,92	906,6

Fonte: Primária (2024)

O teste t de Welch revelou um valor p de 0,05944 para o tempo de resposta. Embora essa diferença não seja estatisticamente significativa ao nível de 5%, há uma tendência que sugere que a arquitetura de microsserviços tende a ter tempos de resposta maiores em comparação com a arquitetura monolítica. Em outras palavras, a arquitetura monolítica demonstrou tempos de resposta inferiores aos da aplicação de microsserviços, sugerindo uma maior eficiência de performance.

Essas diferenças podem ser atribuídas à forma como os módulos da aplicação se comunicam. Na arquitetura monolítica, todos os módulos estão contidos na mesma aplicação, enquanto na arquitetura de microsserviços, os módulos são implementados como serviços independentes que se comunicam por meio de um barramento de mensagens. Portanto, é possível que essa comunicação entre os módulos na arquitetura de microsserviços esteja aumentando o tempo de resposta da aplicação. A Figura 04 ilustra essa diferença no tempo de resposta:

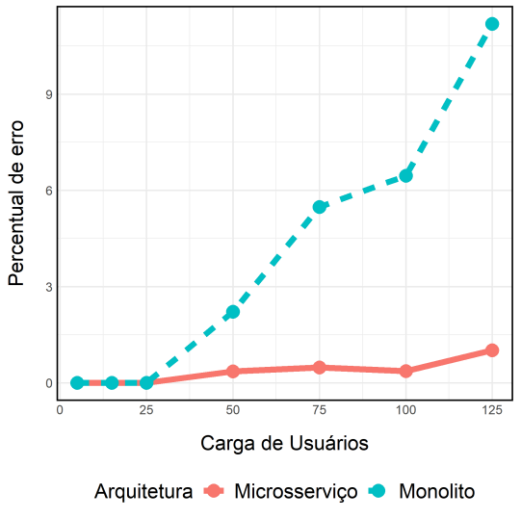
Figura 04 - Gráfico de Tempo de Resposta x Carga de usuários



Fonte: Primária (2024)

O percentual de erro apresentou o valor p de 0,08768, que é maior que 0,05. Portanto, não há evidência suficiente para rejeitar a hipótese nula, indicando que não há diferença significativa no percentual de erro entre as duas arquiteturas, conforme a Figura 05. Contudo, como pôde ser observado nos resultados do Quadro 04, o percentual de erros da arquitetura monolítica apresentou um crescimento maior do que a arquitetura de microsserviços, conforme a carga de usuários foi aumentando.

Figura 05 - Gráfico de Percentual de erro x Carga de usuários

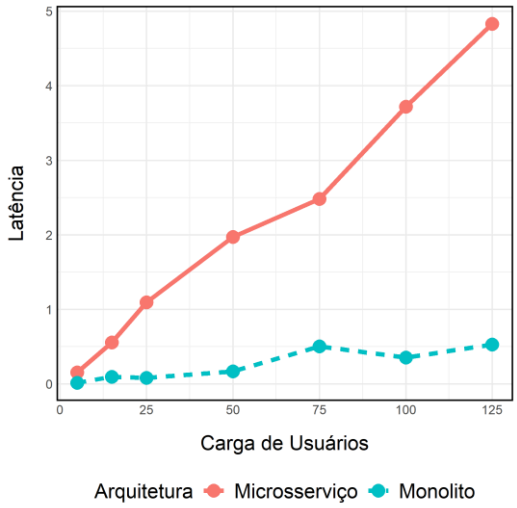


Fonte: Primária (2024)

Com relação à latência, o valor p foi de 0,02731, o que é inferior a 0,05. Portanto, podemos rejeitar a hipótese nula e concluir que existe uma diferença significativa na latência entre as arquiteturas, com a arquitetura de microsserviços apresentando uma latência maior, conforme a Figura 06. Essa disparidade pode ser atribuída à sua natureza distribuída e à comunicação entre os diferentes serviços, que tendem a ter uma latência mais alta. Isso ocorre devido à necessidade dos microsserviços de se comunicarem entre

si por meio de solicitações de rede, o que pode aumentar a latência no processamento dessas solicitações.

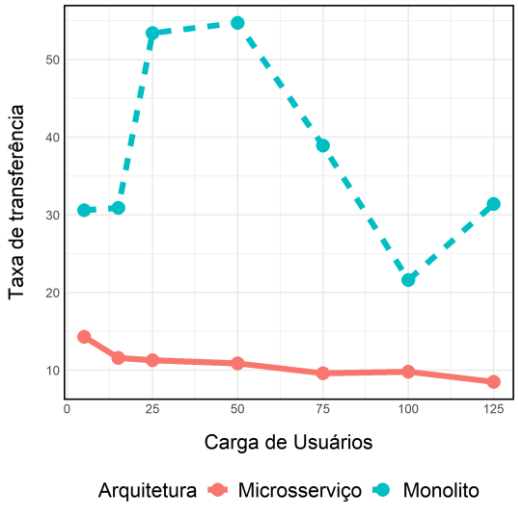
Figura 06 - Gráfico de Latência x Carga de usuários



Fonte: Primária (2024)

A taxa de transferência apresentou o valor p de 0,0001229, que é muito menor que 0,05, indicando uma diferença altamente significativa na taxa de transferência entre as duas arquiteturas, com o monolito apresentando uma taxa de transferência significativamente maior, conforme a Figura 07. Isso pode ser devido à sua natureza centralizada. Na arquitetura monolítica, todas as funcionalidades estão integradas em um único aplicativo, o que pode facilitar a otimização e o gerenciamento de recursos, resultando em uma maior taxa de transferência.

Figura 07 - Gráfico de Taxa de transferência x Carga de usuários



Fonte: Primária (2024)

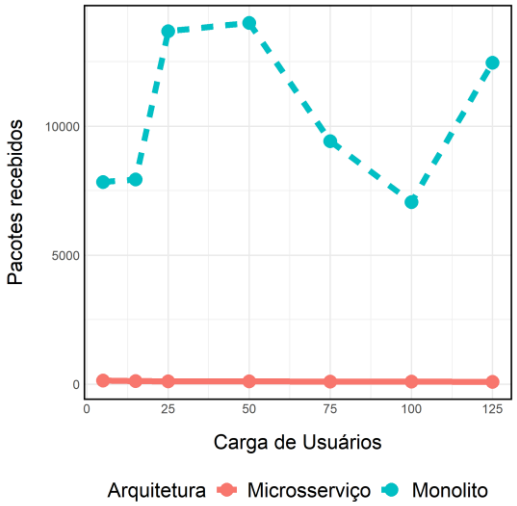
Por fim, a métrica de Pacotes Enviados e Pacotes Recebidos, apresentaram um valor p muito menor do que 0,05, indicando uma diferença altamente significativa no número de pacotes enviados e pacotes recebidos entre as duas arquiteturas, com o monolito

recebendo e enviando significativamente mais pacotes.

Os gráficos presentes nas Figuras 08 e 09 mostraram que a arquitetura monolítica atingiu um pico de performance em termos de pacotes enviados e recebidos com 50 usuários, mas enfrentou uma degradação significativa à medida que a carga aumentou além desse ponto. Isso sugere que a arquitetura monolítica pode ser eficiente até 50 usuários, onde a taxa de transferência também atingiu o seu pico, mas tem limitações de escalabilidade e começa a sofrer gargalos de recursos sob cargas mais altas. Ainda sim, seriam necessários mais testes de carga com mais variações para comprovar esse comportamento, visto que, a carga de pacotes enviados e recebidos volta a aumentar com a carga de 125 usuários simultâneos.

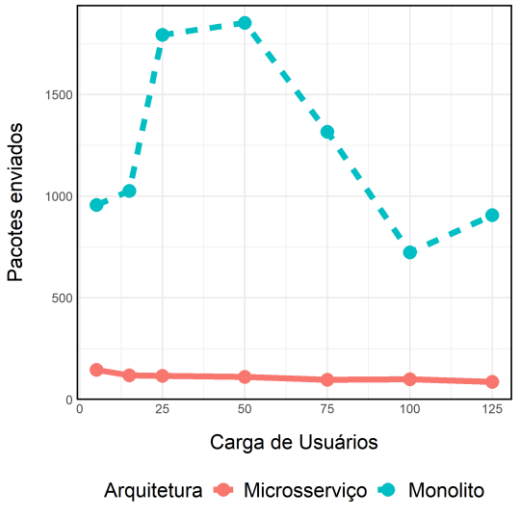
Já a arquitetura de microsserviços apresentou uma redução gradual e constante na quantidade de pacotes enviados e recebidos com o aumento da carga. Isso sugere que os microsserviços distribuem a carga de forma mais uniforme, mas também podem enfrentar limitações na capacidade de processamento por serviço individual, resultando em uma taxa de transferência menor por serviço à medida que a carga aumenta.

Figura 08 - Gráfico de Pacotes recebidos x Carga de usuários



Fonte: Primária (2024)

Figura 09 - Gráfico de Pacotes enviados x Carga de usuários



Fonte: Primária (2024)

5. Conclusão

O objetivo deste estudo foi avaliar a performance das arquiteturas de microsserviços e monolítica em um cenário de e-commerce. Durante o experimento, um e-commerce com arquitetura de microsserviços e outro com uma arquitetura monolítica foram submetidos a testes de carga, utilizando métricas previamente selecionadas para coleta de dados.

Os testes analisaram tempo de resposta, percentual de erro, latência, taxa de transferência, pacotes recebidos e pacotes enviados para observar os efeitos de cada arquitetura. Com base nos resultados dos dados obtidos neste estudo, observou- se que a arquitetura monolítica apresentou melhor performance em termos de tempo de resposta, latência, taxa de transferência, pacotes recebidos e enviados, em comparação com a arquitetura microsserviços. Apesar do monolito apresentar um maior percentual de erro em

cargas mais altas, a diferença não foi estatisticamente significativa.

Estudos como os de Cabane e Farias (2022) e Al-Debagy e Martinek (2018), obtiveram resultados variados ao avaliar a performance de padrões de arquitetura. Cabane e Farias (2022) avaliaram a performance de uma aplicação orientada a eventos e uma aplicação monolítica, concluindo que a arquitetura orientada a eventos teve números de taxa de transferência e tempos de resposta melhores, embora perdesse em uso de CPU e memória para a arquitetura monolítica. Já Al-Debagy e Martinek (2018), compararam a performance entre microsserviços e aplicações monolíticas sob diferentes cargas. Neste experimento, as aplicações apresentaram uma performance semelhante sob cargas normais, porém a arquitetura monolítica mostrou uma performance ligeiramente melhor com menos de 100 usuários, com uma taxa de transferência superior.

Comparando esses resultados, observa-se que a arquitetura monolítica pode ser vantajosa em termos de performance para cenários com cargas moderadas de usuários, enquanto a arquitetura de microsserviços oferece benefícios significativos em termos de escalabilidade, cruciais para ambientes de alta demanda. Esses achados são relevantes para desenvolvedores e arquitetos de software que precisam escolher a arquitetura mais adequada para suas aplicações de e-commerce.

Para trabalhos futuros, recomenda-se uma abordagem mais completa, considerando novas métricas para avaliar outros aspectos relacionados à performance, como uso de CPU e memória. Além disso, aumentar a quantidade de cargas de usuários e escalar as aplicações na nuvem, utilizando serviços como AWS ou Azure, pode ser uma estratégia viável para obter dados mais robustos e realistas, refletindo melhor o ambiente de um e-commerce em produção.

Referências

AL-DEBAGY, Omar; MARTINEK, Peter. **A comparative review of microservices and monolithic architectures**. In: IEEE 18th International Symposium on Computational Intelligence and Informatics (CINTI), 2018, p. 149-154.

AMJAD, Mahfida et al. **Web application performance analysis of E-commerce sites in Bangladesh: an empirical study**. International Journal of Information Engineering and Electronic Business, v. 13, n. 2, p. 47-54, 2021.

APPS, .NET Application Architecture - Reference. **eShopOnWeb**. Disponível em: https://github.com/dotnet-architecture/eShopOnWeb. Acesso em: 05 abr. 2024.

BASS, Len; CLEMENTS, Paul; KAZMAN, Rick. **Software Architecture in Practice**. Addison-Wesley, 2012.

BECKER, João Luiz. **Estatística básica: transformando dados em informação**. Bookman Editora, 2015.

CABANE, Hebert; FARIAS, Kleinner. **On the impact of event-driven architecture on performance: An exploratory study**. Future Generation Computer Systems, v. 153, p. 52-69, 2022.

FIELD, Andy; FIELD, Zoe; MILES, Jeremy. **Discovering statistics using R**. Sage Publications, 2012.

FORD, Neal; PARSONS, Rebecca; KUA, Patrick. **Building evolutionary architectures: support constant change**. O'Reilly Media, Inc., 2017.

FOWLER, Martin. **Monolith First**. Disponível em: https://martinfowler.com/bliki/MonolithFirst.html. Acesso em: 22 mar. 2024.

GAMMA, Erich et al. **Design patterns: Abstraction and reuse of object-oriented design**. In: ECOOP'93—Object-Oriented Programming: 7th European Conference Kaiserslautern, Germany, July 26–30, 1993 Proceedings 7. Springer, p. 406-431.

GARLAN, David; PERRY, Dewayne E. **Introduction to the special issue on software architecture**. IEEE Transactions on Software Engineering, v. 21, n. 4, p. 269-274, 1995.

INGENO, Joseph. **Software Architect's Handbook: Become a successful software architect by implementing effective architecture concepts**. Packt Publishing Ltd., 2018.

KENNETH, C. L.; TRAVER, Carol Guercio. **E-Commerce 2019: Business, Technology and Society**. 15. ed. Prentice Hall, 2019.

MAXIM, B. R.; PRESSMAN, Roger S. **Engenharia de software: uma abordagem profissional**. McGraw-Hill, 2021.

MOLYNEAUX, Ian. **The art of application performance testing: from strategy to tools**. O'Reilly Media, Inc., 2014.

NEWMAN, Sam. **Building microservices**. O'Reilly Media, Inc., 2021.

PLATFORM, .NET. **eShop**. Disponível em: https://github.com/usuario/repositorio. Acesso em: 20 mar. 2024.

RICHARDS, Mark. **Software architecture patterns**. O'Reilly Media, Incorporated, 2015.

RICHARDS, Mark; FORD, Neal. **Fundamentals of software architecture: an engineering approach**. O'Reilly Media, 2020.

RICHARDSON, Chris. **Microservices patterns: with examples in Java**. Simon & Schuster, 2018.

SOMMERVILLE, Ian. **Engenharia de software**. 9. ed. São Paulo, SP, Brasil: Pearson, 2011.