Comparativo entre Arquitetura Monolítica e Arquitetura de Microsserviços

Lucas Gabriel Nerys Pereira, Luciana Mara Freitas Diniz

PUC Minas em Betim
Bacharelado em Sistemas de Informação

lucasnerys10@gmail.com,lucianadiniz@pucminas.br

Resumo. *A definição da arquitetura a ser utilizada em um software é um passo de suma importância, que tem desdobramentos ao longo de todo o ciclo de vida do software, mas essa escolha vai muito além de preferências, é preciso entender o cenário de utilização para decidir de forma assertiva, além das expectativas que o software precisa satisfazer. Buscando clarear esse processo de decisão, uma comparação entre microsserviços e monolito é defendida por meio deste trabalho, através de uma revisão bibliográfica para compreender os conceitos de arquitetura de softwares, arquitetura monolítica e de microsserviços, além de expor as opiniões de especialistas da área, seguida de uma comparação prática entre esses estilos arquiteturais, com o desenvolvimento de dois servidores idênticos, implementados nas diferentes arquiteturas citadas. E por fim, a execução de casos de testes para colher evidências e medir o desempenho de ambas arquiteturas. O resultado obtido permitiu a emissão de uma conclusão sobre a melhor utilização das arquiteturas em questão, baseado nos ambientes criados para os testes e o escopo deste trabalho.*

Palavras-chave: Arquitetura de Softwares, Arquitetura Monolítica, Arquitetura de Microsserviços

1. Introdução.

Na arquitetura de softwares, os sistemas monolíticos dominam o mercado, na maioria dos casos por ser uma arquitetura difundida e consolidada a mais tempo (MACHADO, 2017). Como uma alternativa à arquitetura monolítica, há os microsserviços, definidos como uma arquitetura de softwares e uma estrutura organizacional dos processos relacionados ao desenvolvimento de aplicações distribuídas (AMAZON, 2022).

O caminho natural de desenvolvimento de aplicações começa com uma arquitetura monolítica, por sua facilidade de implementação, e após uma decisão de migração arquitetural, seja ela baseada em problemas advindos da aplicação monolítica ou planejamento, a aplicação adota os microsserviços (FOWLER e LEWIS, 2014). No entanto, se percebe que a migração é um passo existente no ciclo de vida de qualquer aplicação e o planejamento prévio desse processo pode ser a linha divisória entre o sucesso e fracasso.

Mudanças de qualquer natureza em projetos de sistemas computacionais aumentam o custo do projeto exponencialmente de acordo com a fase de desenvolvimento que o sistema se encontra, atingindo o ápice do custo quando ele já está em produção e em algumas situações tornando essa mudança inviável e impraticável (BARTIÉ, 2002). Assim, ao compreender os detalhes de cada arquitetura, os desenvolvedores conseguem entender qual delas é mais adequada para o domínio da

aplicação em questão, evitando escopos de aplicação mal dimensionados, gastos não planejados, retrabalhos e até reestruturação total de uma aplicação.

Tanto a arquitetura monolítica quanto a de microsserviços trazem vantagens e desvantagens no desenvolvimento de software, e é necessário analisar o contexto como um todo antes de tomar qualquer decisão. A escolha da arquitetura a ser utilizada em uma aplicação afeta diretamente a produtividade da equipe e o produto final.

Neste sentido, a escolha de determinada arquitetura possibilita reduzir migrações e redefinições arquitetônicas inesperadas de aplicações pois, migrações de uma arquitetura para outra exigem um grande esforço de gerenciamento do projeto de software, diminuindo drasticamente a produtividade da equipe e do negócio, além de significarem mais gastos e prejuízos em situações críticas onde essa migração se dá por caráter urgente, principalmente partindo de um monólito, haja vista que são aplicações "que possuem um alto grau de acoplamento" (GERVINO, 2020).

Um dado importante levantado por Amaral e Carvalho (2017), é o fato de que a arquitetura de microsserviços é relativamente nova em relação à arquitetura monolítica, assim é percebido que o tema ainda está em seus estágios iniciais, tanto na literatura quanto no mercado de T.I. Mas com o grande avanço das tecnologias de *cloud computing,* as aplicações *serverless* e os novos paradigmas e ferramentas que apoiam o desenvolvimento de microsserviços, este tema vem ganhando força nos últimos anos, impulsionados pelos casos de sucesso da Netflix e Amazon.

Dito isso, ao comparar essas arquiteturas, a justificativa deste trabalho é tornar o processo de escolha arquitetural mais eficiente e tangível, tanto para desenvolvedores de monólito quanto de microsserviços, e também aqueles que ainda estão presos em qual escolha fazer. Um planejamento bem definido da arquitetura utilizada na aplicação, traz eficiência, eficácia e diminuição nos gastos do processo de desenvolvimento.

O objetivo deste trabalho, portanto, é apresentar qual solução é a mais adequada para uma situação de escolha de arquitetura de aplicações de software por meio de uma pesquisa bibliográfica e uma pesquisa aplicada com simulações das duas arquiteturas em questão, que apoiem uma escolha inteligente, competente e produtiva sobre qual arquitetura se faz mais indicada para cada situação.

Este trabalho segue estruturado em Seções, quais sejam: a primeira é a introdução, o referencial teórico na Seção 2. A terceira Seção descreve os trabalhos relacionados aos tipos de arquitetura de softwares. A Seção 4 apresenta a metodologia utilizada. Na quinta Seção são apresentados os resultados da parte prática. A Seção 6 trata da conclusão deste trabalho, seguido das referências bibliográficas.

2. Arquitetura de Softwares.

São decisões que incidem sobre os aspectos centrais de um sistema de software, passando pela definição dos elementos estruturais, as interfaces, o comportamento colaborativo esperado das interações dos elementos do sistema, a integração desses elementos em subsistemas que evoluem progressivamente em tamanho e estilo arquitetural. (NHIMI, 2016).

Arquitetura de Software é o processo de definição de uma solução estruturada que atende a todos os requisitos técnicos e operacionais e ao mesmo tempo otimiza atributos de qualidade padronizados como desempenho, segurança e gerenciamento. Arquitetura envolve uma série de decisões baseadas em uma vasta gama de fatores e cada uma destas decisões pode provocar um impacto considerável no sucesso ou fracasso da aplicação (MAA Guide, 2ª edição).

A arquitetura de softwares pode ser definida como as regras de construção de um determinado software, que delimitam quais funcionalidades e características ele deve satisfazer, assim como características técnicas, como tempo de resposta e poder de processamento. É a estrutura que irá guiar a equipe de desenvolvimento na construção da aplicação, cobrindo os aspectos técnicos e estruturais dos códigos, quais padrões e processos de projetos de softwares serão utilizados, os testes e a implantação, inclusive é aqui que é feita a escolha da arquitetura do software a ser utilizada, seja ela monolítica, de microsserviços, orientada a serviços e entre outras.

2.1 Arquitetura de Microsserviços

É um padrão de desenvolvimento de softwares, onde se objetiva a construção de pequenos serviços para a composição da aplicação. Cada serviço executa seu próprio processo e se comunica através de mecanismos leves, por exemplo uma API[1] de recursos HTTP[2]. São utilizadas máquinas de implantação automatizadas para cada serviço. Os recursos do domínio do negócio são a base para a construção desses serviços, além do gerenciamento ser mínimo e centralizado. E por fim, existe a liberdade para escolher as tecnologias que serão empregadas na aplicação. (FOWLER e LEWIS, 2014).

Desta forma, de acordo com TSERPES (2019), microsserviços são um conceito de arquitetura de software pelo qual a funcionalidade de uma aplicação é decomposta em funções menores e independentes, residindo em recursos próprios de infraestrutura. Assim, segundo o autor, uma característica básica de um ambiente de microsserviços é a replicabilidade, ou seja, um determinado microsserviço pode ser facilmente replicado quantas vezes forem necessárias para alcançar escalabilidade ou simplesmente reutilização, o que implica ao microsserviço a incorporação da lógica da função e o ambiente de provisionamento, por exemplo, um servidor web para expor uma API RESTful[3].

É a divisão granular dos componentes lógicos de uma aplicação, para gerar responsabilidades coesas, baixo acoplamento e o uso de comunicação padronizada entre os componentes, com requisições HTTP/Rest API na maioria dos casos. (BALALAIE, A., 2016). Visando assim, uma maior taxa de atualização dos componentes lógicos da aplicação, já que um componente não tem efeito ou dependência de outro componente e

[1] Application Programming Interface (API) é um conjunto de regras definidas para a comunicação entre aplicações distintas, com uma interface documentada e comunicação padronizada (IBM, 2020).
[2] É um protocolo de comunicação para obtenção de recursos na Web (MDN, 2022).
[3] Representational State Transfer (REST) é um tipo de API que se enquadra nas restrições da arquitetura REST para permitir a interação com serviços Web RESTful (Red Hat, 2020).

uma manutenibilidade maior e menos complexa, já que se lida com componentes coesos e menores.

Portanto, se trata de uma maneira para desenvolver aplicações distribuídas, dividindo cada funcionalidade em pequenos blocos de código. Assim o software é dividido em módulos com especialidades específicas e suas próprias infraestruturas. De acordo com FOWLER, e LEWIS. (2014), essa arquitetura se preocupa com as estruturas do negócio em si, para dividir cada parcela de serviço em um módulo desacoplado e isolado da aplicação, e então integrar os vários serviços para compor a aplicação em sua totalidade. É importante citar que de acordo com a IBM, em um artigo publicado em seu site no ano de 2021, os microsserviços tem grande influência da arquitetura (*Service Oriented Architecture*) SOA.

2.2 Arquitetura Monolítica

Aplicações corporativas geralmente são desenvolvidas em três partes relevantes, que são: a interface com o usuário (páginas HTML com JavaScript sendo executadas no navegador do computador do usuário), um banco de dados (em sua maioria sendo relacionais) e uma aplicação no lado do servidor (essa aplicação é responsável por lidar com as solicitações HTTP, execução da lógica de negócios, processos relacionados ao banco de dados, como recuperação e atualização das informações e preenchimento das visualizações HTML do navegador). Essa aplicação no lado do servidor é um monolito, que contém toda a lógica para lidar com as requisições e é executado em um único processo, assim permitindo o uso de recursos básicos da linguagem de programação para dividir a aplicação em classes, funções e *namespaces* (FOWLER, e LEWIS., 2014).

Se define como sistemas monolíticos, aplicações com um alto grau de acoplamento[4], que concentram em um único código-fonte todos os acessos ao banco de dados, as regras de negócio e os componentes básicos para as funcionalidades. Esse código-fonte é executado em um único processo computacional. (VICERI, 2020). É uma aplicação onde toda a lógica é executada em um único servidor (Bakshi, Kapil, 2017).

São softwares que são criados em sua totalidade em apenas um código. Neste único arquivo de código-fonte são encontradas todas as implementações das funcionalidades da aplicação, desde regras de negócios, até acesso ao banco de dados. É a arquitetura mais utilizada e difundida no mercado, seja por sua facilidade de desenvolvimento, já que não é necessário ter de lidar com questões complexas de aplicações distribuídas, ou por sua maturidade, visto que é o modelo de arquitetura de softwares mais antigo do mercado.

Muitos sistemas legados ainda utilizam essa arquitetura, sendo que em vários casos uma migração é vista como uma tarefa extremamente complexa pela corporação em questão. Apesar dos problemas de acoplamento, manutenção e complexidade, ainda

[4] Em engenharia de software, acoplamento ou dependência é o grau de interdependência entre módulos de software; uma medida de quão intimamente ligadas estão duas rotinas ou módulos (*Systems and software engineering* — Vocabulary, 2010).

é a arquitetura mais indicada para sistemas pequenos e início de projeto de software, onde o escopo do mesmo não é totalmente previsível pela equipe de desenvolvimento.

3. Trabalhos Relacionados.

Este tópico visa apresentar trabalhos relacionados à comparação entre a arquitetura de microsserviços e a monolítica.

Amaral e Carvalho (2017), abordam um comparativo entre as arquiteturas de microsserviços e monolítica. Num primeiro momento apresentam uma revisão bibliográfica levantando pontos positivos e negativos das duas arquiteturas a fim de traçar o contexto de uso de cada uma e elaborar um artigo comparativo, motivados pela falta de conteúdo na literatura sob o tema. Os autores citam as principais características das arquiteturas, como a alta taxa de acoplamento dos módulos, a dificuldade de escalonamento e replicação referentes ao monólito. A independência dos serviços que compõem os microsserviços, e também a falta de ferramentas que apoiem a implementação de uma aplicação baseada em microsserviços. Ao final do embasamento teórico, é iniciado um teste qualitativo e comparativo entre duas aplicações idênticas, porém desenvolvidas nas duas arquiteturas, com o propósito de executar testes de carga de trabalho para coleta e análise dos dados de cada aplicação, a fim de obter uma resposta sobre qual arquitetura teve o melhor desempenho naquela situação. As métricas levadas em consideração nos testes são vazão, latência e uso do hardware. Os resultados são apresentados em gráficos e quadros, e se percebe que a aplicação baseada em monolítico tem um desempenho superior.

O trabalho dos autores Amaral e Carvalho (2017) têm sucesso ao apresentar os dados de forma legível e expor todos os detalhes técnicos das aplicações, com a disponibilização das configurações e dos modelos conceituais de funcionamento das aplicações, bem como listar possíveis trabalhos futuros a partir do mesmo. Porém, os testes realizados foram muito específicos e voltados para o desempenho apenas, assim a aplicação monolítica teve os melhores resultados. Testes comparativos de arquitetura devem levar em conta mais de um contexto de uso e outras métricas de comparação, como tempo de aprendizagem e desenvolvimento de cada aplicação.

Garcia (2021), em seu artigo busca detalhar processos de migração de uma aplicação monolítica para de microsserviços. Após comparar definições de diversos autores sobre as duas arquiteturas e citar conceitos importantes de *cloud computing* e aplicações *serverless*, o trabalho é iniciado com a definição do cronograma das etapas e pontos importantes que formarão a pesquisa desenvolvida. Em seguida, é feita uma contextualização do cenário do mercado de TI. Então, o autor inicia com uma comparação entre os processos de desenvolvimento das duas arquiteturas, como a divisão e estrutura das equipes de desenvolvimento, para então entrar nos detalhes de implementação da parte prática do trabalho, citando os casos de uso e a estrutura da aplicação monolítica que será migrada para microsserviços, e por fim o processo de modularização, decomposição dos serviços e migração.

O trabalho do autor Garcia (2021) foi assertivo em abordar vários detalhes técnicos sobre as duas arquiteturas, e propor um processo migratório bem detalhado e visual, tornando fácil o entendimento do mesmo por parte do leitor. Outro ponto positivo é o fato do autor propor uma comparação justa que busca clarear e exemplificar as duas arquiteturas e o contexto de uso de cada uma delas sem afirmar qual é melhor. Uma melhoria neste trabalho seria abordar mais o processo de migração junto com os desafios e problemas enfrentados, assim tornando este processo mais tangível para o leitor.

Enquanto no trabalho proposto por Amaral e Carvalho (2017), os autores buscam uma comparação quantitativa e mais voltada às métricas colhidas na parte prática para delimitar qual arquitetura de softwares é a melhor no cenário prático proposto, o trabalho do autor Garcia (2021), busca uma comparação qualitativa com foco no esclarecimento das características e peculiaridades das aplicações baseadas em monólito e microsserviços. Explorando aspectos do processo de desenvolvimento e migratório de uma arquitetura monolítica para microsserviços e também, no desenho arquitetônico da aplicação, com diagramas e gráficos para uma melhor visualização.

4. Metodologia.

Para a metodologia deste trabalho, foram definidas duas abordagens. A primeira é uma pesquisa bibliográfica descritiva que visa esclarecer os detalhes, aspectos técnicos e características de aplicações monolíticas e de microsserviços. A segunda abordagem tem um foco em pesquisa e prática aplicada para a simulação das duas arquiteturas abordadas. Nesta etapa, o objetivo é ter uma visão prática do processo de desenvolvimento de aplicações baseadas em microsserviços e monólitos, a fim de estruturar uma comparação mais embasada das arquiteturas em questão.

A parte prática deste trabalho é essencial, e contém o desenho e criação dos diagramas de cada arquitetura, definições de padrões de projetos, escolha das tecnologias que serão utilizadas, desenvolvimento dos códigos, implantação, testagem e coleta dos parâmetros de comparação. Ao final, este trabalho poderá apoiar decisões arquiteturais de aplicações em desenvolvimento, em estágio de definição conceitual ou mesmo aplicações em produção que buscam pontos de melhoria.

Para o desenvolvimento da parte prática, como linguagem de programação foi escolhido TypeScript utilizando o *framework* Nest.JS com a IDE Visual Studio Code. Para a criação de ambientes isolados e automação do processo de execução dos servidores, foi utilizado Docker para a virtualização das imagens dos servidores em containers, o Docker-Compose para a definição, gerenciamento e análise dos containers gerados, e o Docker Desktop como interface visual para o gerenciamento dos containers e imagens geradas de forma mais prática. Com o intuito de possibilitar a comunicação entre os microsserviços, foi utilizado o sistema de mensagens Kafka, e por fim, como banco de dados, foi escolhido o MongoDB. Todas as ferramentas de desenvolvimento necessárias para a execução foram configuradas em um único container para cada servidor, utilizando a *network* interna do Docker, com o objetivo de simplificar a

implantação e os testes. Para a execução dos testes de performance, foi utilizada a ferramenta de benchmarking em protocolo HTTP/1.1, chamada Autocannon.

Os servidores criados para a parte prática são inspirados em um sistema de gestão corporativa, que contém cinco coleções e módulos principais, que são: Clientes, Colaboradores, Departamentos, Feriados e Projetos. Um módulo de gestão interna foi criado para a arquitetura monolítica, para processos onde seriam necessários processamento em mais de um módulo, com objetivo de manter organizada a comunicação interna dos processos. Nos microsserviços, porém, foi necessário a criação de um serviço que atua como *middleware,* para o acionamento dos outros serviços, a publicação das mensagens no Kafka e também como ponto único de entrada para os testes de performance. A Figura 1 representa o desenho arquitetural do monolito e a Figura 2, o desenho arquitetural dos microsserviços.

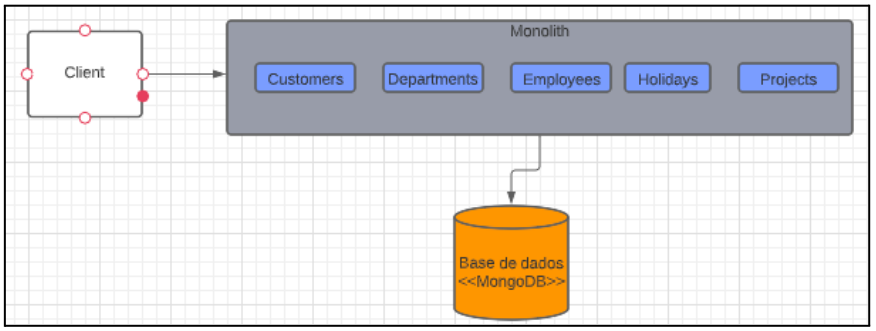

Figura 1. Arquitetura Monolítica

Fonte: Elaborado pelo autor

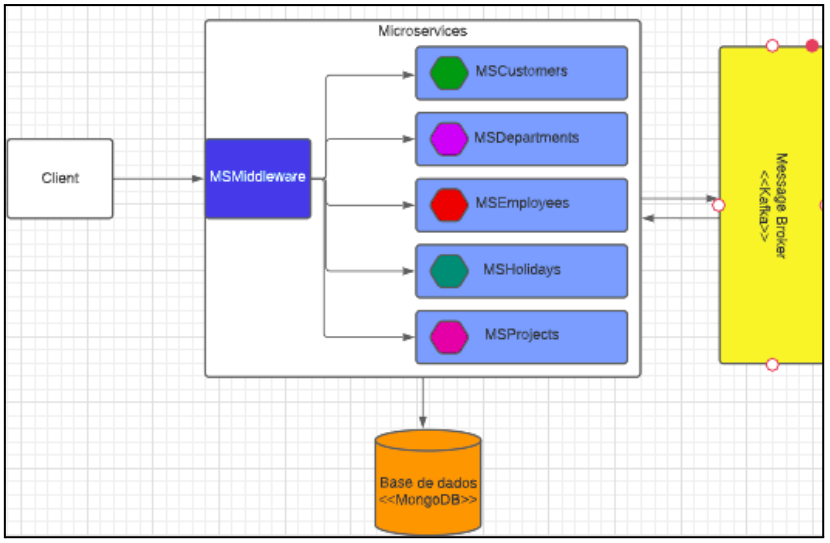

Figura 2. Arquitetura de Microsserviços

Fonte: Elaborado pelo autor

Para efeito de comparação entre as arquiteturas que serão desenvolvidas na parte prática deste trabalho, foram escolhidos cinco parâmetros, que são: Escalabilidade,

complexidade de desenvolvimento, tempo de desenvolvimento, complexidade dos testes e complexidade de implantação.

5. Apresentação de Resultados.

Este tópico tem como objetivo apresentar os resultados da parte prática, onde foram desenvolvidos dois servidores idênticos de gestão corporativa baseados nas arquiteturas de microsserviços e monolítica. Em seguida, foram feitos testes de performance em ambos servidores, com o intuito de coletar e analisar dados para identificar qual arquitetura teve o melhor desempenho nos casos de testes, e por fim, possibilitar avaliar a parte prática a partir dos cinco parâmetros citados no tópico anterior. Os testes foram divididos em três partes: latência de requisições por milissegundos, volume de requisições por segundos e porcentagem de latência.

O Gráfico 1 representa a latência das requisições por milissegundos. Esse teste foi segmentado em três partes:

- 10 conexões HTTP durante 10 segundos, somando 95 mil requisições no total das duas arquiteturas;
- 100 conexões HTTP durante 10 segundos, com 100 mil requisições no total das duas arquiteturas;
- 1000 conexões HTTP durante 10 segundos, com 113 mil requisições no total das duas arquiteturas;

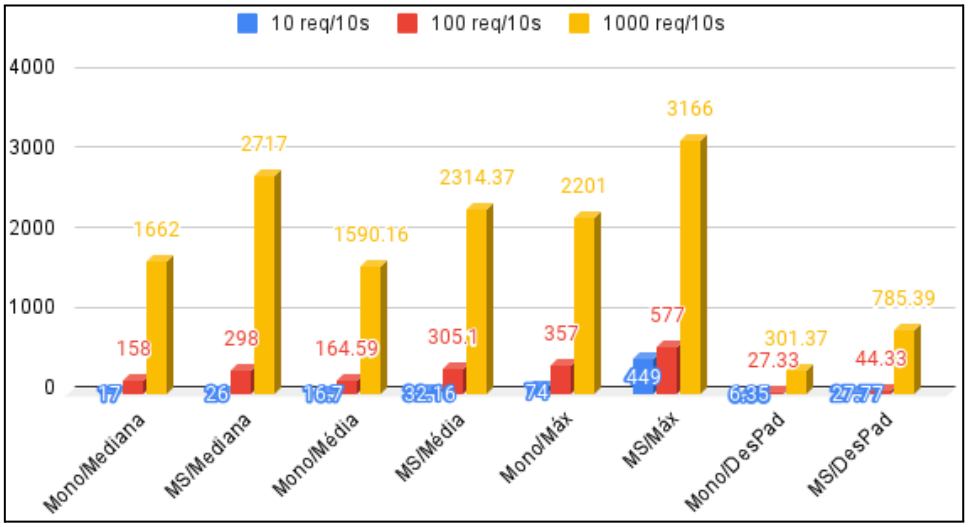

Gráfico 1. Latência de requisições em milissegundos

Fonte: Elaborado pelo autor

Percebe-se que a arquitetura monolítica teve um desempenho superior em todas as faixas de testes. Por conta do valor elevado de requisições, neste caso, os valores que melhor representam esse grupo de valores é a mediana, onde é possível ter uma percepção mais correta da representação desse grupo de valores. No teste com a menor taxa de requisições por segundos, tanto a arquitetura monolítica quanto a de microsserviços tem valores bem próximos. Porém, quanto multiplicado por dez, a

diferença de latência das requisições entre a arquitetura monolítica e de microsserviços chega quase ao dobro. É importante citar a grande diferença entre os valores máximos de latência do intervalo de 10 requisições/10 segundos, onde a arquitetura de microsserviços chega a 449 milissegundos de latência, enquanto a monolítica resultou em 74 milissegundos.

O Gráfico 2 representa o volume de requisições por segundos. Esse teste foi segmentado em três partes:

- 10 conexões HTTP durante 10 segundos, somando 95 mil requisições no total das duas arquiteturas;
- 100 conexões HTTP durante 10 segundos, com 100 mil requisições no total das duas arquiteturas;
- 1000 conexões HTTP durante 10 segundos, com 113 mil requisições no total das duas arquiteturas;

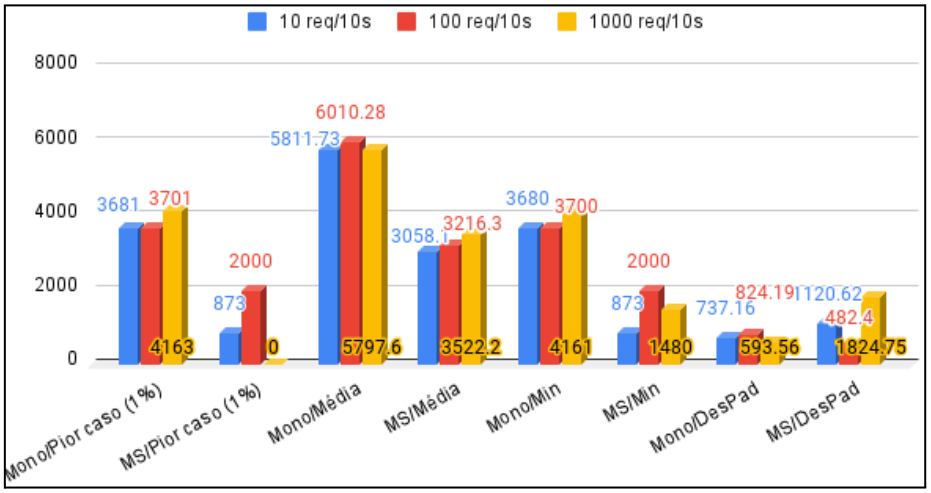

Gráfico 2. Volume de requisições, em requisições/ segundo

Fonte: Elaborado pelo autor

Nos dois primeiros conjuntos de colunas, estão os piores casos para esse teste, que representam cerca de 1% das requisições, as mais lentas de todo o conjunto. Desses conjuntos pode-se analisar o comportamento de ambas arquiteturas no caso mais indesejado. A arquitetura de microsserviços não foi capaz de processar nenhuma dessas requisições no teste de 1000 requisições / 10 segundos nesse caso de teste. Isso pode ser influência da arquitetura de sistemas distribuídos, onde nem sempre é possível garantir que todas as requisições serão recebidas e tratadas pelo servidor, enquanto a monolítica foi capaz de atender 4163 requisições nesse período.

O Gráfico 3 representa a porcentagem de latência das requisições. Esse teste foi segmentado em três partes:

- 10 conexões HTTP durante 10 segundos, somando 95 mil requisições no total das duas arquiteturas;
- 100 conexões HTTP durante 10 segundos, com 100 mil requisições no total das duas arquiteturas;

- 1000 conexões HTTP durante 10 segundos, com 113 mil requisições no total das duas arquiteturas;

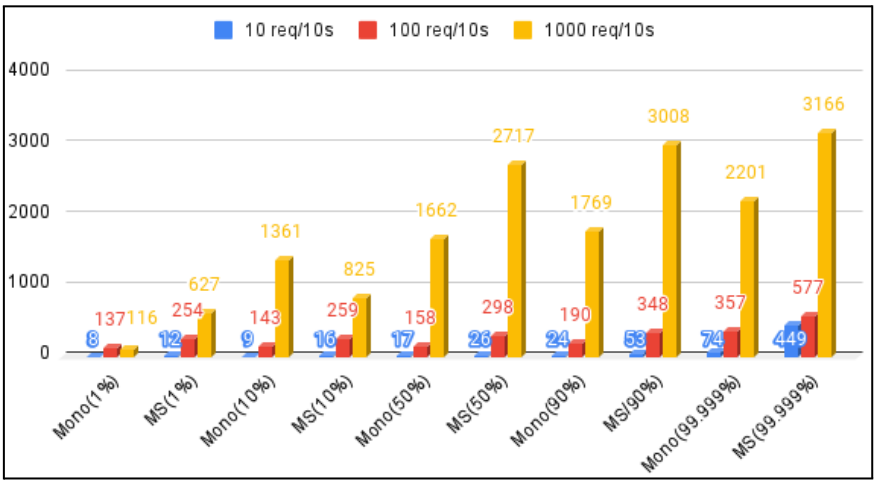

Gráfico 3. Porcentagem de latência em milissegundos

Fonte: Elaborado pelo autor

Enquanto 1% de latência mostra um relativo equilíbrio entre as arquiteturas, ao passo que a porcentagem de latência e o número de requisições por segundo aumenta, a diferença começa a crescer de forma potencial. Aos 90% de latência a diferença dos microsserviços em relação ao monolito quase dobra nas 1000 requisições / 10 segundos. Isso está diretamente ligado ao fato de que existe uma sequência de chamadas de um serviço a outro, fazendo com que exista uma fila de espera para o atendimento de uma requisição.

É notório que a arquitetura monolítica se saiu muito melhor nos casos de testes, e um dos motivos é que em sistemas menores e menos complexos, uma comunicação através de processos internos do servidor é mais eficiente do que chamadas externas a outros serviços, onde se gasta mais tempo processando e esperando respostas às requisições, e existe um gasto mais elevado de capacidade computacional, além da grande complexidade natural de sistemas distribuídos.

O Gráfico 4 representa a avaliação da parte prática tendo os cinco parâmetros definidos no tópico anterior, que são: escalabilidade, complexidade de desenvolvimento, tempo de desenvolvimento, complexidade dos testes, complexidade de implantação.

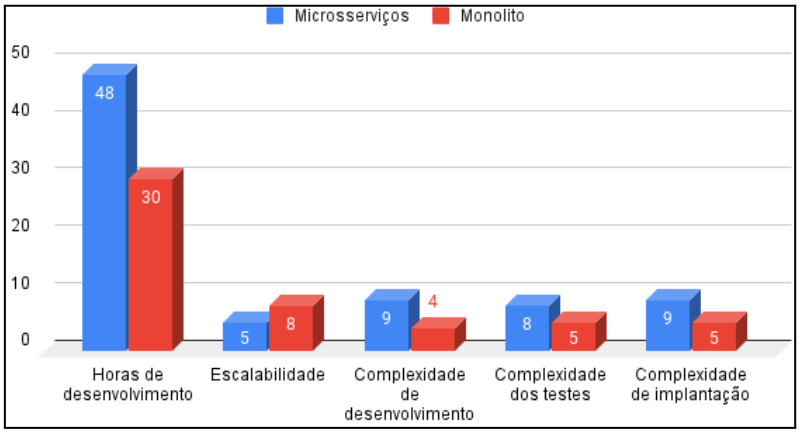

Gráfico 4. Parâmetros de avaliação da parte prática

Fonte: Elaborado pelo autor

Como dito anteriormente, o estilo arquitetural de microsserviços traz consigo um alto grau de complexidade, seja no desenvolvimento, implantação e testes. Esse alto grau de complexidade não compensa em sistemas pequenos, pois estes, podem ser desenhados e implementados em uma arquitetura monolítica, de uma forma mais simples e transparente.

No Gráfico 4, é possível notar que existe uma diferença de 18 horas de desenvolvimento dos microsserviços em relação ao monolito, essa diferença se deu em boa parte na implementação de comunicação entre os microsserviços e infraestrutura necessária para o funcionamento deles.

6. Conclusão.

O objetivo deste trabalho foi uma pesquisa bibliográfica sobre as arquiteturas de microsserviços e monolítica, para uma comparação teórica sobre os fundamentos de cada uma na visão dos autores que formaram a base referencial de pesquisa. Foi possível ter contato com várias opiniões e experiências sobre as duas arquiteturas, e observar pontos de vista que ponderam a escolha da arquitetura baseada nas características que a aplicação precisa satisfazer e outras que foram irredutíveis em apontar determinada delas como a melhor escolha, independente dos detalhes e requisitos.

Em seguida, a parte prática foi iniciada, onde foram definidos e implementados dois servidores idênticos, do ponto de vista funcional, de gestão corporativa em ambas arquiteturas, para a realização de testes de performance e uma comparação baseada nos dados e parâmetros colhidos dos testes.

No ambiente em que foram executados os casos de testes, a arquitetura monolítica teve um desempenho e capacidade de escalabilidade superior, seja na latência das requisições atendidas por milissegundos, no volume de requisições atendidas por segundo e na porcentagem da latência de requisições por milissegundos. Nos parâmetros definidos na Seção 4, também é notório a superioridade do monolito,

onde o tempo, complexidade de desenvolvimento e implementação foram menores que nos microsserviços.

Ao concluir a parte prática, ficou claro que no escopo do trabalho e em sistemas pequenos e menos complexos, a arquitetura monolítica tem vantagem em relação aos microsserviços, onde seu desempenho é melhor e sua complexidade de implementação menor. A arquitetura de microsserviços é possivelmente mais eficiente em sistemas grandes e complexos, onde manter apenas uma base de código se torna um processo trabalhoso e confuso para manutenção e inovação. A divisão dos vários serviços em serviços granulares defendida pelos microsserviços permite uma manutenção mais simples e tangível, além de acelerar o processo de inovação, já que os vários serviços granulares não têm efeito entre si, permitindo vários ciclos de desenvolvimento simultâneos.

Portanto, considerando o escopo menor da parte prática, seria tendencioso apontar que a arquitetura monolítica é superior a de microsserviços em todas as situações. Nos testes aplicados neste trabalho, a arquitetura monolítica teve um desempenho superior aos microsserviços. Dito isso, para um trabalho futuro, a aplicação de uma parte prática com testes em um sistema maior poderá embasar uma comparação mais abrangente de ambas arquiteturas, para a análise de seus comportamentos e desempenhos em um escopo de sistema maior.

Por fim, percebe-se com este trabalho, a importância da análise correta da melhor solução para determinado problema quando se trata de arquitetura de softwares. Um caso de uso de sucesso não significa que esta solução é a melhor em resolver qualquer problema, pois é necessário entender o domínio do negócio e analisar as diversas soluções, procurando a que melhor atende o problema. Redefinições e retrabalhos por conta de uma escolha errada custam caro e afetam a produtividade do negócio.

Referências Bibliográficas.

1. AMARAL, Odravison e CARVALHO, Marcus, (2017): **"**Arquitetura de Micro Serviços: uma Comparação com Sistemas Monolíticos.**"** Disponível em: https://repositorio.ufpb.br/jspui/bitstream/123456789/3235/1/OAJ14062017.pdf . Acesso: 03 de Agosto de 2022.
2. BAKSHI, Kapil, (2017) **"**Microservices-based software architecture and approaches**"** Disponível em: https://ieeexplore.ieee.org/document/7943959 . Acesso: 03 de Agosto de 2022.
3. BALALAIE, Armin, (2016): **"**Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture**"**, paper IEEE Software;
4. BARTIÉ, Alexandre, (2002) **"**Garantia da Qualidade de Software: adquirindo maturidade organizacional.**"** Rio de Janeiro: Elsevier;

5. CARNEIRO, Hugo S. Glauco e MONTEIRO, Miguel, (2019): **"**Towards a Roadmap for the Migration of Legacy Software Systems to a Microservice based Architecture.**"**
6. DEHGHANI, Zhamak, (Abril de 2018.): **"**How to break a Monolith into Microservices**"**: Disponível em: https://martinfowler.com/articles/break-monolith-into-microservices.html . Acesso em: 19 de Maio de 2022.

   Disponível em: https://martinfowler.com/articles/microservices.html#AreMicroservicesTheFuture . Acesso em: 10 de Maio de 2022.
7. FOWLER, Martin, (Agosto de 2019.): **"**Microservice Guide**"**: Disponível em: https://martinfowler.com/microservices/ . Acesso: 10 de Maio de 2022.
8. FOWLER, Martin. LEWIS, James, (Março, 2014): **"**Microservice**"**:
9. GARCIA, Gabriel. B., (2021): **"**Do monolítico ao microserviço.**"** Disponível em: https://suap.ifsp.edu.br/media/edu/projeto_final/TCC_gabriel.pdf . Acesso: 03 de Agosto de 2022.
10. GERVINO, M. T, (2020): **"**MIGRAÇÃO DE SOFTWARE MONOLÍTICO PARA MICRO SERVIÇOS: uma revisão sistemática da literatura.**"** Revista Interface Tecnológica, *[S. l.]*, v. 17, n. 1, p. 17–28. DOI: 10.31510/infa.v17i1.700. Disponível em: https://revista.fatectq.edu.br/interfacetecnologica/article/view/700 . Acesso em: 19 maio. 2022.
11. IBM Cloud Education. (Julho de 2021): **"**O que são microsserviços?**"** Disponível em: https://www.ibm.com/br-pt/cloud/learn/microservices . Acesso em: 03 de Abril de 2022.
12. LOPES, Taylor, (2021): **"**Método de migração de sistemas monolíticos legados para a arquitetura de microsserviços**"**;
13. MACHADO, M. G. Micro Serviços: **"**Qual a diferença para arquitetura monolítica?**"** Disponível em: https://www.opus-software.com.br/micro-servicos-arquietura-monolitica/ . Acesso em: 13 de Maio de 2022.
14. N. Alshuqayran, N. Ali and R. Evans, (2016): **"**A Systematic Mapping Study in Microservice Architecture,**"** 2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA), 2016, pp. 44-51, doi: 10.1109/SOCA.2016.15.

15. NHIMI, Filipe, T. L. R, (2016) : “Princípios e Práticas em Arquitetura de Software” Disponível em: http://www.machado.mg.gov.br/files/concursos/1cf11cf161fe4eb688dfeec880d6b4d9.pdf . Acesso: 10 de Maio de 2022.
16. RICHARDSON, Chris. (Abril de 2019): ‘’What are microservices?” Disponível em: https://microservices.io . Acesso em: 21 de Maio de 2022.
17. TSERPES, Konstantinos, (2019): “A microservices’ methodology for the creation of short, fast-paced, stream processing pipelines”) Disponível em: https://www.sciencedirect.com/science/article/pii/S2405959519301092?via%3Dihub . Acesso em: 03 de Junho de 2022.
18. “Aplicações Serverless: Quais as vantagens de usar esta arquitetura?” Disponível em: https://viceri.com.br/insights/aplicacoes-serverless-quais-as-vantagens-de-usar-esta-arquitetura/ . Acesso em: 2 de Maio de 2022.
19. “Micro Serviços: Qual a diferença para a Arquitetura Monolítica?”, (2021) Disponível em: https://www.opus-software.com.br/micro-servicos-arquietura-monolitica/ . Acesso em: 07 de Junho de 2022.
20. “Systems and software engineering — Vocabulary”, (ISO/IEC/IEEE 24765:2010, ISO/IEC TR 19759:2005, Software Engineering — Guide to the Software Engineering Body of Knowledge (SWEBOK)). Disponível em: https://www.iso.org/obp/ui/#iso:std:iso-iec-ieee:24765:ed-1:v1:en . Acesso: 03 de Agosto de 2022.