Arquitetura de Micro Serviços: uma Comparação com Sistemas Monolíticos

Odravison Amaral, Marcus Carvalho

Universidade Federal da Paraíba (UFPB) – Campus IV – LN
Caixa Postal 58.280-990 – Rio Tinto – PB – Brazil
Centro de Ciências Aplicadas e Educação
Departamento de Ciência Exatas
Curso de bacharelado de Sistemas de Informação

{odravison.amaral, marcuswac}@dce.ufpb.br

Resumo. *Este artigo tem como objetivo comparar aplicações desenvolvidas em diferentes arquiteturas de software (monolíticas e micro serviços) através de experimentos. Através do experimento foram colhidos dados, analisados usando vazão, latência e tempo total de execução dos testes como métricas para definir qual a melhor arquitetura a ser escolhida em diferentes contextos. Ademais, este artigo apresenta o fundamento das duas arquiteturas e realiza uma comparação teórica entre elas.*

Abstract. *This paper has as purpose to compare applications developed with different software architecture (monolithic and microservices) through experiments. Through experiments we collected data and they were analyzed using throughput, latency and test's execution time as metrics to define which is the best architecture to be chosen in differents contexts. Moreover, this paper presents the fundamental aspects of both architectures and perform a theoretical comparison between them.*

1. Introdução

No cenário de desenvolvimento de software estão ocorrendo várias mudanças significativas e com isso o surgimento de novas tecnologias como, por exemplo, novos processos de desenvolvimento, novas linguagens de programação, novos paradigmas de programação e portanto, novos mercados. Sabendo disso, as empresas estão tentando acompanhar essas mudanças migrando os seus serviços para o uso dessas tecnologias. Neste sentido, novas propostas estão sendo feitas na área de arquiteturas de aplicações web. Definir a arquitetura de uma aplicação não é uma tarefa simples e existem vários padrões que podem ser seguidos, sendo difícil escolher o que melhor se adequa ao propósito da aplicação.

Atualmente, a arquitetura monolítica é a mais tradicional e mais utilizada no mercado, devido à sua popularidade [Machado 2017]. Segundo Martins (2007, p. 18), "na arquitetura monolítica o software é construído num único módulo [...]" como, por exemplo, uma aplicação Java que se comunica com o banco através de um driver *JDBC*[1] (Java Database

[1] Conjunto de classes e interfaces (API) escritas em Java que fazem o envio de instruções SQL para qualquer banco de dados relacional;

Trabalho de Conclusão de Curso do discente Odravison Amaral Júnior, sob a orientação do(s) docente(es) Marcus Williams Aquino de Carvalho submetido ao Curso de Bacharelado em Sistemas de Informação da Universidade Federal da Paraíba, Campus IV, como parte dos requisitos necessários para obtenção do grau de Bacharel em Sistemas de Informação.

Connectivity) que vem embutido no módulo da aplicação. Aplicações Monolíticas são planejadas para serem auto-contidas; componentes da aplicação são interconectados e interdependentes [ROUSE, 2016].

Com a chegada da cultura *DevOps*[2] no cenário de desenvolvimento web, uma nova arquitetura de software vem sendo proposta, chamada de Micro Serviços (ou *microservices*, em inglês). Esta arquitetura baseia-se em micro aplicações independentes, diferentemente da que existe na arquitetura monolítica. Micro Serviço é uma parte, específica e independente, de uma aplicação maior, como uma parte de outra aplicação, ela tem responsabilidades únicas dentro da arquitetura. Em outras palavras, a arquitetura de micro serviços é o conjunto de aplicações menores e independentes que juntas formam uma aplicação completa. O termo micro serviço surgiu durante uma conferência de arquitetos de software (ICSAE[3]) em Maio de 2011 e vem sendo bastante difundido desde então na comunidade de arquitetos, engenheiros de softwares e desenvolvedores [Machado 2017].

Atualmente na literatura não existem muitos trabalhos que discutem a comparação da arquitetura de micro serviços com a arquitetura monolítica, tendo em vista que essa arquitetura surgiu há pouco tempo. A falta de informações e de comparações, tanto teóricas quanto práticas, entre essa arquitetura e outros modelos dificulta a escolha por parte dos profissionais dessa área quanto ao modelo de arquitetura mais adequado para uma determinada aplicação web. O desempenho e a complexidade de desenvolvimento são fatores importantes de decisão, mas não há comparações claras entre estes fatores no contexto de aplicações monolíticas e de micro serviços.

Por esta razão, este artigo tem como objetivo auxiliar a tomada de decisões arquiteturais para aplicações web, principalmente de larga escala, fornecendo comparativos teóricos, através da análise comparativa de propriedades da arquitetura de micro serviços e arquitetura monolítica, e práticos, através da comparação de resultados de experimentos com uma aplicação implementada em ambas arquiteturas .

O restante do artigo está estruturado da seguinte maneira. Na seção 2, são abordadas comparações teóricas entre as duas arquiteturas. Na seção 3, está descrita a metodologia da comparação prática, definição das métricas a serem analisadas, definição do ambiente de execução, descrição da aplicação, cargas de teste e instrumentos utilizados, além de todos os detalhes necessário para o seu entendimento. Na seção 4, são apresentados os resultados da execução dos testes, além de uma análise com os dados coletados a fim de obter resultados quantitativos. Na seção 5, são apresentados trabalhos relacionados. Finalmente, na seção 6 é descrita a conclusão das comparações bem como sugestões para trabalhos futuros.

2. Comparação Teórica: arquitetura monolítica vs. Micro serviços

O uso da arquitetura monolítica traz consigo vantagens e desvantagens. Uma das vantagens é a maior facilidade no desenvolvimento da aplicação, pois conta com o suporte que as ferramentas de desenvolvimento convencionais fornecem para esse tipo de arquitetura

[2] Desenvolvedor que tem conhecimento nas três áreas: desenvolvimento, operações e infraestrutura;
[3] International Conference on Software Architecture and Engineering

[Richardson 2016]. O objetivo comum da maioria das ferramentas no mercado hoje é dar suporte a esse tipo de arquitetura. Uma outra vantagem que pode ser destacada é a simplicidade de fazer sua implantação (*deployment*[4], em inglês) pois o uso desse tipo de arquitetura implica na construção e execução de um único programa executável monolítico, que nada mais é do que um conjunto de módulos interdependentes. Porém, as aplicações que usam a arquitetura monolítica tendem a aumentar a sua complexidade com o tempo de desenvolvimento e evolução da aplicação, tornando a leitura e o entendimento do código processos árduos e lentos, principalmente para novos integrantes do time de desenvolvimento [Machado 2017]. Por ser desenvolvida em módulos interdependentes, alterações feitas no código de um módulo da aplicação podem atingir outros módulos desta aplicação.

Comparada com a arquitetura monolítica, a arquitetura de micro serviços veio para aumentar a produtividade do desenvolvimento de software para aplicações web, bem como o seu desempenho em relação a aplicações com arquiteturas monolíticas inseridas no mesmo contexto. Uma forma de aumentar a escalabilidade de servidores web, por exemplo, é usando a replicação, que consiste em espelhar servidores para permitir o balanceamento de carga e alta disponibilidade da aplicação [Sabo 2006]. A replicação de micro serviços é mais eficiente do que a replicação de aplicações monolíticas onde o seu escalonamento só é possível replicando a aplicação inteira.

A seguir, iremos comparar as seguintes propriedade das entre as duas arquiteturas: o acoplamento e independência funcional, escalonamento e replicação, ferramentas de desenvolvimento, complexidade no desenvolvimento.

2.1 Acoplamento e Independência Funcional

Uma das vantagens de se usar a arquitetura de micro serviços é a total independência que os micro serviços têm dentro da aplicação. Cada micro serviço tem o seu próprio banco de dados, sendo este um dos fatores que caracterizam um micro serviço. Entretanto, esta também é a característica que tem o maior custo de planejamento por arquitetos e desenvolvedores de software [Richardson 2016].

Contrário a esse aspecto, a arquitetura monolítica conta com um único banco de dados, onde se concentram todas as informações necessárias para a aplicação. Com isto, ela não demanda de um planejamento tão grande quanto o necessário em micro serviços para se arquitetar as relações entre as tabelas. A desvantagem da arquitetura monolítica está no momento do escalonamento e replicação de dados, já que ela tende a ter um alto acoplamento em comparação a arquitetura de micro serviços. De acordo com Machado (2017), as funções de sistemas monolíticos são tão interdependentes e acopladas que a manutenção em um módulo pode causar inconsistências e comportamentos inesperados.

[4] Executar uma aplicação e deixá-la em produção após o seu desenvolvimento ser concluído;

2.2 Escalonamento e Replicação

Uma das vantagens de se usar arquitetura monolítica é a facilidade de implantação e escalonamento de uma aplicação, comparado a arquitetura de micro serviços onde é necessário fazer implantação de vários serviços. Porém, micro serviços tem vantagem quando necessitamos de consistência e integridade dos dados, pois para aplicações monolíticas essa é uma tarefa difícil. O escalonamento de sistemas monolíticos tem um custo muito alto, pois é necessário replicar todo o sistema, mesmo que apenas algumas funcionalidades sejam necessárias, bem como planejar uma estratégia de integridade dos dados entre as replicações [Machado 2017].

2.3 Ferramentas de Desenvolvimento

Uma das vantagens mais importantes da arquitetura monolítica é ter disponível muitas ferramentas no mercado que dão suporte ao desenvolvimento de aplicações monolíticas [Richardson 2016]. O cenário para arquitetura de micro serviços quanto às ferramentas de desenvolvimento é muito escasso. Recentemente, a equipe de desenvolvimento da Netflix[5] contribuiu para o módulo chamado *Spring Cloud*[6] do Spring Framework, onde é possível ter um ambiente de desenvolvimento Java voltado para aplicações distribuídas em micro serviços. Porém, alternativas como esta ainda são limitadas.

A falta de ferramentas de desenvolvimento para aplicações que usam arquitetura de micro serviços é um dos reflexos causados pela sua recente aparição.

2.4 Complexidade no Desenvolvimento

A dificuldade inicial no desenvolvimento de uma aplicação que usa arquitetura de micro serviços é maior do que uma aplicação de arquitetura monolítica [Fowler].

O custo inicial do desenvolvimento de uma aplicação monolítica é muito inferior do que o de uma aplicação em micro serviços. Atrelado a isso existe a dificuldade com as ferramentas. Entretanto, segundo Fowler (2016), apesar da produtividade inicial para uma aplicação monolítica ser maior que a de micro serviços, no decorrer do desenvolvimento a produtividade para uma aplicação monolítica cai quase que 90% enquanto a de micro serviço permanece variando entre 5% e 10% a menos.

3. Metodologia da Comparação Prática

A metodologia da avaliação prática deste trabalho consiste em fazer uma análise quantitativa do desempenho de uma aplicação desenvolvida com os dois tipos de arquitetura: monolítica e de micro serviços. Para isto, foram utilizados: o ambiente de nuvem computacional da Amazon (LightSail - AWS[7]) para execução das aplicações; o *NGINX*[8] como ferramenta de

[5] http://www.netflix.com/
[6] http://projects.spring.io/spring-cloud/
[7] http://lightsail.aws.amazon.com

proxy-reverso e balanceador de carga de uma das aplicações executadas; a ferramenta de benchmark *JMeter*[9] para gerar e avaliar as cargas de trabalho; e a linguagem de programação *Java*[10] e a ferramenta de desenvolvimento web *Spring Framework*[11] para implementação da aplicação nas duas arquiteturas.

É relevante saber que as aplicações têm propósitos idênticos; a única diferença entre as duas aplicações é a sua arquitetura. Após o desenvolvimento, as aplicações foram executadas, uma por vez. A análise foi realizada da seguinte maneira: a ferramenta *JMeter* foi utilizada para a geração e execução paralela das requisições à aplicação, atendendo a aspectos de cargas de trabalho que serão descritas ainda nesta seção. Terminada a execução, os dados foram coletados e catalogados apropriadamente, juntamente com a descrição da aplicação e do cenário de teste no qual foi executado.

3.1 Métricas

Para realização da análise, foram utilizadas três métricas de performance definidas da seguinte maneira:

- **Vazão:** quantidade de requisições respondidas por segundo;
- **Latência:** diferença entre o tempo de envio de uma requisição e o tempo da chegada da resposta para essa requisição;
- **Uso do hardware:** taxas de uso do CPU e memória RAM.

3.2 Aplicações desenvolvidas

As aplicações foram desenvolvidas com um único e comum objetivo: retornar uma resposta positiva para a requisição. Para isso, elas seguem os seguintes passos:

1. Receber uma requisição (GET);
2. Internamente, fazer ou não uma requisição para outro serviço;
3. Responder a requisição com um *HTTP status code*[12] 200 (OK).

Para reproduzir o comportamento de serviços que se comunicam, os recursos das aplicações foram divididos em níveis. Um recurso que é requisitado e não necessita requisitar outro serviço para atender a requisição é um recurso de nível 1; um recurso que é requisitado e necessita requisitar 1 (um) outro serviço é um recurso de nível 2; e um recurso que necessita requisitar outros 2 (dois) serviços é um recurso de nível 3.

A tabela 1 mostra os recursos (rotas de acesso ou *endpoints*) considerados e seus respectivos níveis.

[8] https://www.nginx.com/
[9] http://jmeter.apache.org/
[10] https://www.java.com/
[11] https://spring.io
[12] Código de resposta para requisições que usam protocolo HTTP;

Tabela 1. Recurso com seu respectivo nível.

Nível	Recurso Micro Serviço	Recurso Monolítico
Nível 1	/ping-data	/lvl1
Nível 2	/ping-data/lvl2	/lvl2
Nível 3	/ping-data/lvl3	/lvl3

3.3 Ambiente de Execução

Para a execução das aplicações foram instanciadas 3 máquinas virtuais. Cada uma com as seguintes configurações: 40GB de espaço de armazenamento (SSD); 2,0GB de memória RAM; e 1 núcleo de processamento virtual (vCPU). A VM1 foi usada como proxy reverso e load balancer enquanto a VM2 e a VM3 foram usadas para instanciar as aplicações onde ficam em execução esperando requisições a serem respondidas.

A figura 1 ilustra a arquitetura de implantação para a aplicação de micro serviços. O fluxo das requisições funciona da seguinte maneira: 1. A ferramenta de benchmark (test application) gera e envia requisições paralelamente para a URL fornecida; 2. Quando a requisição é recebida pela aplicação da VM1, o *gateway-service*[13], API Gateway[14] e load balancer[15] que escolhe através de uma lista prévia de servidores (a escolha é feita através do algoritmo Round-robin [Oliveira, 2011]) fornecida pelo *discovery-service*[16] (Eureka Server) para repassar a requisição; 3. Uma vez que a próxima aplicação, *ping-service* ou *pong-service*, recebe a requisição, ela responde a requisição ou chama outro serviço, dependendo do nível do recurso (endpoint) chamado, através do Ribbon (load balancer); 4. Caso um outro serviço seja chamado e todos os níveis tenham sido percorridos, o recurso responde a requisição feita pelo serviço anterior; 5. Depois que a requisição é processada, uma resposta é retornada para o solicitante.

[13] Serviço responsável por repassar as requisições para os determinados endereços de ip;
[14] Padrão de serviços responsável por repassar chamadas para APIs;
[15] Serviço responsável por distribuir as cargas de requisições;
[16] Serviço responsável por gerenciar os serviços levantados (feito deploy);

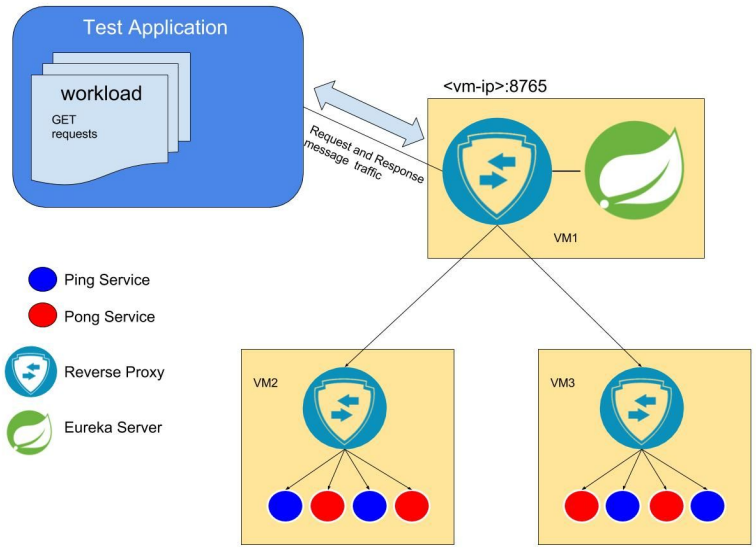

Figura 1. Ilustração da arquitetura de implantação da aplicação usando micro serviços

Não muito diferente desta arquitetura de implantação anterior, a figura 2 ilustra a arquitetura de implantação para a aplicação monolítica. O fluxo de funcionamento é o seguinte: 1. A aplicação de teste (*test application*) executa os testes fazendo requisições para a VM1; 2. A requisição chega na aplicação proxy-reverso (NGINX) da VM1, que por sua vez escolhe a aplicação que irá receber a requisição (a escolha é feita através do algoritmo Round-robin); 3. Dependendo do recurso requisitado, a aplicação faz outra requisição internamente para um outro recurso que por sua vez retorna a resposta ao recurso anterior; 4. O recurso chamado responde ao solicitante.

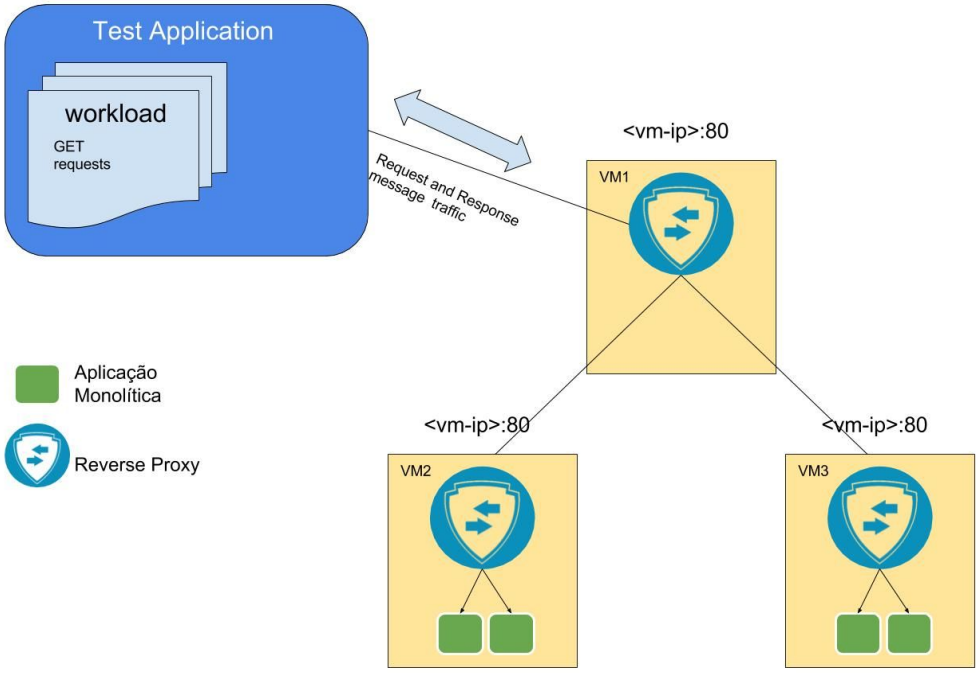

Figura 2. Ilustração da arquitetura de implantação da aplicação monolítica

Como podemos ver, a principal diferença entre as aplicações está na sua arquitetura e no seu Load Balancer. Na VM 2 e 3 da arquitetura de micro serviços, vemos que cada aplicação é na verdade um micro serviço que precisa ser descoberto pelo Eureka Server para que o API Gateway também o conheça e possa fazer requisições para ele. Já na aplicação monolítica, cada aplicação instanciada é o módulo inteiro da aplicação, ou seja, cada aplicação contém o "serviço" ping e pong que se conhecem internamente. Cada VM contém um Load Balancer que está escutando na porta :80 de cada VM. O Load Balancer da VM1 conhece, previamente, o Load Balancer da VM2 e VM3 sendo possível alcançar as aplicações. Todos os Load Balancers usam o algoritmo de escalonamento round-robin.

3.4 Cargas de Trabalho

As cargas de trabalho foram definidas de modo a equilibrar os cenários de requisições geradas para as aplicações. As requisições são enviadas para a aplicação de forma paralela, com 12 requisições simultâneas e a carga de 10 mil requisições.

Para cada nível de recurso, foram executados 3 (três) testes com a carga de trabalho definida acima e calculado a média dos resultados. No total foram coletados 36 resultados de testes; 18 para cada tipo de arquitetura, 9 para cada nível de requisição. Em resumo, as configurações do cenário de teste são orientadas por nível de acesso interno da aplicação (quantas chamadas a serviços internos serão realizadas) e tipo da arquitetura da aplicação.

4. Resultados

Nesta seção, serão apresentados os resultados dos experimentos executados.

Na figura 3, observa-se a vazão (requisições/segundo) das aplicações (monolítica e micro serviços). É possível observar que a vazão da aplicação monolítica se manteve praticamente uniforme durante todos os testes. Isso é previsto, pois a aplicação monolítica contém todos os serviços necessários para atender qualquer requisição que seja solicitada. Contrária a essa performance, a aplicação de micro serviços é um pouco inferior à aplicação monolítica. Ainda na figura 3, observa-se que a aplicação monolítica foi melhor em todos os níveis de requisições. Já a aplicação de micro serviço mostra que seu desempenho é decrescido na medida em que a complexidade da requisição aumenta.

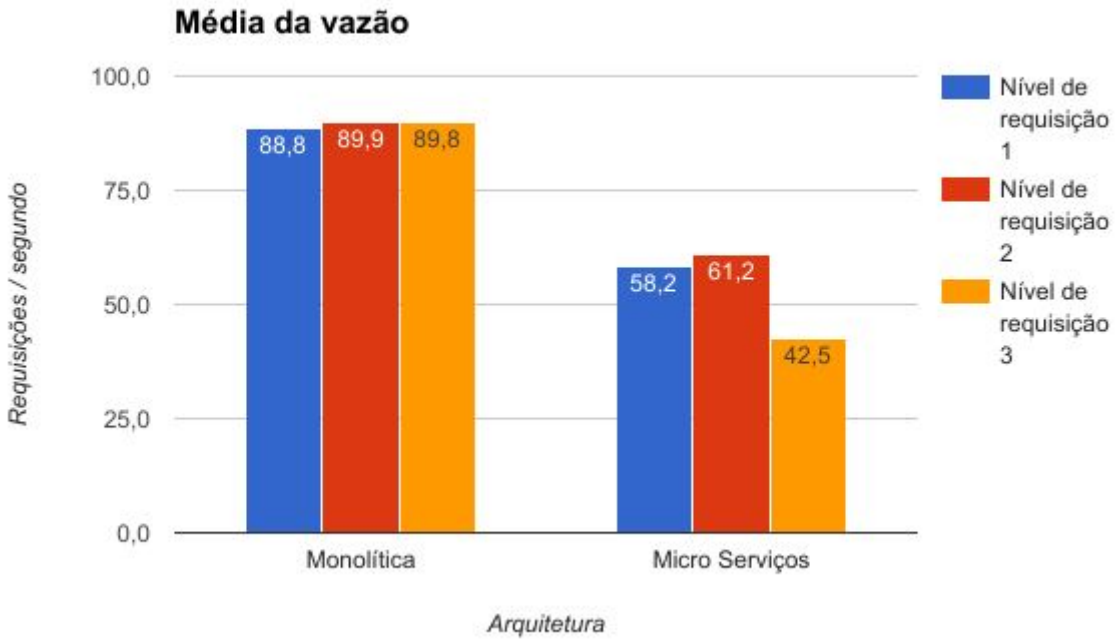

Figura 3. Representação gráfica da média da vazão por arquitetura e nível de requisição

Na figura 4, observa-se a média da latência dos testes realizados. É notável o desempenho superior da aplicação monolítica com a aplicação de micro serviço neste contexto. Enquanto a aplicação de micro serviços teve marcas perto dos 400 ms de latência, a aplicação monolítica obteve, a maior média, 142 ms de latência. O fato da aplicação monolítica está concentrada em um único executável fez com que a sua latência fosse menor do que a aplicação de micro serviços.

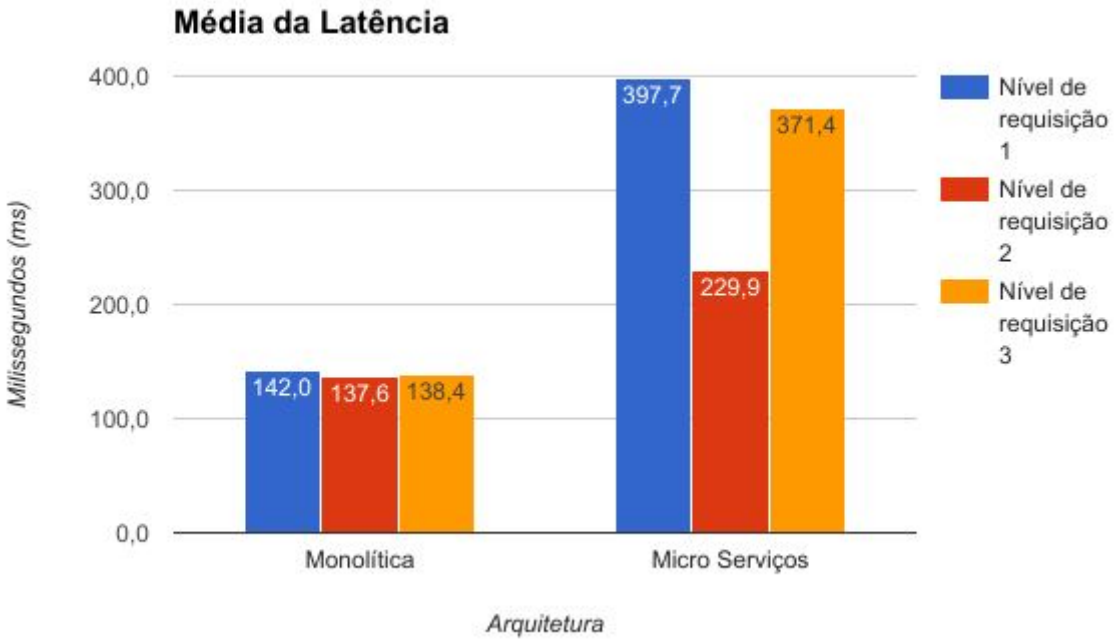

Figura 4. Gráfico que representa a média da latência de cada nível de requisição realizada de cada teste das duas aplicações

As tabelas 4 e 5 apresentam para a arquitetura monolítica e de micro serviços, respectivamente, a média dos indicadores de uso do CPU e Memória RAM para cada tipo de carga. Cada indicador é o resultado da média entre as duas VMs onde as aplicações foram executadas.

Tabela 4. Indicadores médio de CPU e Memória RAM utilizada pelas VM2 e VM3 durante os testes da aplicação monolítica

Aplicação	**Arquitetura Monolítica**								
Requisição	**Nível 1**			**Nível 2**			**Nível 3**		
Teste	**1**	**2**	**3**	**1**	**2**	**3**	**1**	**2**	**3**
CPU	6%	3%	4%	6%	4%	4%	4%	5%	4%
RAM	788MB	789MB	789MB	787MB	789MB	789MB	790MB	789MB	789MB

Tabela 5. Indicadores médio de CPU e Memória RAM utilizada pelas VM2 e VM3 durante os testes da aplicação de micro serviços

Aplicação	Arquitetura de Micro Serviços								
Requisição	Nível 1			Nível 2			Nível 3		
Teste	1	2	3	1	2	3	1	2	3
CPU	35%	25%	30%	33%	23%	29%	33%	25%	24%
RAM	734MB	733MB	734MB	734MB	734MB	735MB	733MB	734MB	735MB

É possível observar na tabela 4 que o uso do processador com a aplicação monolítica é bem inferior comparado ao uso com a aplicação de micro serviços, como podemos ver na tabela 5. Isso ocorre porque a aplicação monolítica executa apenas um processo por aplicação, enquanto a aplicação de micro serviços, por ser um conjunto de serviços, executa N processos, onde N é o número de serviços executados. Enquanto o uso de memória RAM é bem parecido entre as aplicações. Isso ocorre por serem aplicações JAVA que necessitam ser executadas uma máquina virtual, chamada JVM, que aloca um espaço na memória antes da sua execução.

5. Trabalhos relacionados

Miguel et al. (2017) falam a respeito da arquitetura SOA[17], muitas vezes confundida com arquitetura de micro serviços, mas que seria a solução mais próxima de de uma aplicação com arquitetura em micro serviços. Entretanto, o artigo tem como objetivo a comparação das três soluções para cloud computing (Amazon , Oracle e Microsoft) no uso de aplicações com o tipo de arquitetura SOA.

O artigo online, Micro Serviços, por Martin Fowler e James Lewis (2014), dois importantes contribuidores para literatura de micro serviços, comparam teoricamente as propriedade e características as duas arquiteturas (monolítica e micro serviços), além de comentar sobre as ferramentas existentes hoje no mercado para desenvolvimento de aplicações que usam arquitetura de micro serviços. A diferença entre este artigo e o de Martin é que ele não faz nenhum tipo de experimento com aplicações que implementam as arquiteturas.

Silva et al. em seu artigo, propõe a adoção de arquitetura de micro serviços para aplicações corporativas a fim de garantir o ciclo de desenvolvimento de software sustentável. Para isso, o artigo tem como embasamento um levantamento bibliográfico com intenção de avaliar vantagens e desvantagens no uso da arquitetura de micro serviços destacando pontos importantes e desafios que empresas e equipes de desenvolvimento irão encontrar ao optar

[17] **SOA** significa Service-Oriented Architecture, ou Arquitetura Orientada a Serviços, numa tradução livre - https://pt.wikipedia.org/wiki/Service-oriented_architecture;

por esta arquitetura. A diferença em relação a este artigo e o de Silva é que não existe um experimento feito com base no embasamento e das arquiteturas.

6. Conclusões e Trabalhos Futuros

Neste artigo foi realizado uma comparação teórica entre as arquiteturas: monolítica e micro serviços, seguido por um experimento realizado entre duas aplicações idênticas mas com diferentes arquiteturas (monolítica e micro serviço). O objetivo do artigo é de auxiliar profissionais da área na tomada de decisões arquiteturais. Após a execução do experimento, os dados foram coletados, interpretados e analisados, gerando informações com gráficos e tabelas.

Conclui-se que, para aplicações com baixo nível de complexidade, onde necessitam de bom desempenho e que não necessitem ser escalável, o ideal seria a arquitetura monolítica, pois ela tem um maior poder resposta nesse contexto. Para aplicações que necessitam de um alto escalonamento, facilidade na integração dos dados, o ideal seria a arquitetura de micro serviços. Ademais, a arquitetura de micro serviço é ideal caso a alta produtividade de um time de desenvolvimento seja a necessidade, conforme o discutido, o isolamento das funcionalidade de uma aplicação é um dos principais fatores que permitem a alta produtividade de um time. Vale salientar que a aplicação submetida para testes é muito simples e não tem tanta complexidade comparado a uma aplicação real em produção.

Quanto ao uso do hardware, conclui-se que as aplicações que utilizam arquitetura de micro-serviços necessitam de um poder de processamento maior comparação a aplicações monolíticas. Vale salientar que esta conclusão é devido a tecnologia que foi utilizada (JAVA), que necessita de um aparato maior para manter uma aplicação de micro-serviços do que uma aplicação monolítica.

Um possível trabalho futuro seria realizar a mesma análise só que em uma aplicação real, que por exemplo, já esteja em produção há mais de 1 ano e foi convertida em aplicação de micro serviços. Essas aplicações reais podem apresentar resultados diferentes devido à sua complexidade. Outro possível trabalho futuro pode ser adicionando os bancos de dados nessas mesmas aplicações e realizar o mesmo experimento, mas desta vez com persistência e busca em banco de dados (CRUD) para que possa se tomar a decisão de qual o melhor banco para aplicações de micro serviços. Um outro possível trabalho futuro pode ser realizar a análise de diferentes load balancers para aplicações que usam arquitetura de micro serviços.

Referências bibliográficas

Machado, M. G. **Micro Serviços: Qual a diferença para arquitetura monolítica?** <http://www.opus-software.com.br/microservicos-diferenca-arquietura-monoliticas/>. Acessado em: 21 de Março de 2017.

Martins, J. C. C. **Técnicas para Gerenciamento de Projetos de Software,** Brasport, 2007, cap. A, p. 18.

Fowler, M. Microservice Resource Guide. <https://www.martinfowler.com/microservices/>. Acessado em 23 de Maio de 2017.

MIGUEL. C. J.; SANTOS, J. A.; SILVA, P. C.; **COMPARAÇÃO DE SERVIÇOS PARA CLOUD COMPUTING.** Disponível em: <https://revistas.unifacs.br/index.php/rsc/article/download/4623/3067>. Acessado em: 26 de Maio de 2017.

NGINX. <https://www.nginx.com/resources/wiki/>. Acessado em: 31 de Março de 2017.
Oliveira, R. S.; Carissimi, A. S.; Toscani, S. S.; **Sistemas Operacionais - Vol: 11: Série Livros Didáticos,** Informática UFRGS, Rio Grande do Sul, cap 4, p. 117.

Richardson, C. **Pattern: Microservices Architecture.** <http://microservices.io/patterns/microservices.html>. Acessado em: 28 de Fevereiro de 2017.

Rouse, M. (2016) **Monolithic Architecture.** <http://whatis.techtarget.com/definition/monolithic-architecture>. Acessado em: 01 de Maio de 2017.

Sabo, C. P. **Avaliação de Desempenho com Algoritmos de Escalonamento em Clusters de Servidores Web,** São Carlos, 2006, cap. 3, p. 35.

SILVA, S. S. **Adoção da Arquitetura Microservices em Aplicações Corporativas.** Disponível em:<https://www.softplan.com.br/tratadoaprendizagem/wp-content/files_mf/1485519168TCC.pdf>. Acessado em: 26 de Maio de 2017.