INTERFACE TECNOLÓGICA

ARQUITETURAS MONOLÍTICAS E MICROSERVIÇOS AUSENTE DE SERVIDOR: um estudo comparativo

MONOLOTIC ARCHITECTURES AND MICROSERVICES MISSING SERVER: a comparative study

João Henrique Teixeira de Jesus - joao.jesus5@fatec.sp.gov.br
Faculdade de Tecnologia de Catanduva – Catanduva – São Paulo - Brasil

Ronaldo Rodrigues Martins - ronaldoviper@gmail.com
Faculdade de Tecnologia de Catanduva – Catanduva – São Paulo - Brasil

DOI: 10.31510/infa.v18i1.1170
Data de submissão: 17/04/2021
Data do aceite: 09/07/2021
Data da publicação: 30/07/2021

RESUMO

Na última década, os sistemas de informações evoluíram profundamente e modificam as vidas das pessoas e organizações. O surgimento de *smartphones*, a evolução da computação em nuvem e o amadurecimento da computação orientada a serviços permitiu que muitos negócios fortemente estabelecidos se transformassem em novos negócios, modificando a economia e hábitos da vida humana. Acompanhando essa evolução, o desenvolvimento de softwares evoluiu de aplicações monolíticas para aplicações compostas de microsserviços em ambientes ausentes de servidores, permitindo que aplicações sejam mais escalonáveis e flexíveis quanto a manutenção e implantação. Este artigo apresenta um estudo comparativo sobre aplicações monolíticas e de microsserviços em ambientes ausente de servidores em um ambiente de computação em nuvem. Neste sentido, acreditamos que os resultados deste trabalho possam orientar futuros projetos de desenvolvimento de software orientados a serviços além de direcionar futuras pesquisas acadêmicas.

Palavras-chave: Microsserviços, Aplicações Monolíticas, Aplicações Ausentes de Servidores

ABSTRACT

In the last decade, information systems have evolved profoundly and change the lives of people and organizations. The emergence of smartphones, the evolution of cloud computing and the maturation of service-oriented computing has enabled many well-established businesses to become new businesses, changing the economy and habits of human life. Accompanying this evolution, software development has evolved from monolithic applications to applications composed of microservices in server-free environments, allowing applications to be more scalable and flexible in terms of maintenance and deployment. This article presents a

comparative study on monolithic and microservice applications in non-server environments in a cloud computing environment. In this sense, we believe that the results of this work can guide future service-oriented software development projects in addition to directing future academic research.

Keywords: Microservices, Monolithic Applications, Serverless

1 INTRODUÇÃO

Na última década, as pessoas e organizações tornaram-se mais dependentes de tecnologias móveis, inteligentes e distribuídas. Dispositivos inteligentes como *smart phones*, *smart TVs*, *smart watches*, e dispositivos de internet das coisas (do inglês, *Internet of Things* - IoT), aliado as plataformas de computação em nuvem (do inglês, *Cloud Computing*), tem a capacidade de sustentar aplicações flexíveis, onipresentes e multiplataformas (KARWAN, 2019).

A computação em nuvem disponibiliza diversos recursos computacionais como servidores, bancos de dados, serviços de aprendizado de máquina, entre outros. Estes serviços possuem alta disponibilidade e passam a sensação de onipresença aos usuários, aparentando estarem funcionado sem quedas ou falhas na maior parte do tempo (KARWAN, 2019).

Neste contexto a arquitetura orientada a serviços (do inglês, *Service-Oriented Architecture* - SOA), tem se destacado na indústria de desenvolvimento de software. De acordo com Barry e Dick (2013), SOA pode ser definida como uma maneira de projetar, implementar, e montar serviços para dar suporte ou automatizar funções de negócio. Aplicações SOA são disponibilizadas por meio de serviços web (do inglês, *Webservices),* o qual é responsável por permitir o consumo de aplicações clientes

Segundo Fowler e Lewis (2014), softwares orientados a serviços podem ser classificados em monolíticos e microsserviços. Na abordagem monolítica, os serviços são empacotados em um único executável. No entanto, a abordagem baseada em microsserviços é uma evolução das aplicações monolíticas, onde cada função de negócio é empacotada em executáveis individuais permitindo maior flexibilidade de manutenção e implantação.

O desenvolvimento do presente trabalho consiste em uma pesquisa comparativa de aplicações monolíticas e microsserviços ausentes de servidor em uma plataforma de computação em nuvem comercial. Portanto, espera-se que os resultados apresentados possam

orientar e auxiliar futuros trabalhos e pesquisas de desenvolvimento de softwares orientados a serviços.

Este trabalho foi organizado da seguinte maneira: na Seção 1 são apresentados conceitos e definições sobre computação em nuvem, computação orientada a serviços e abordagem de desenvolvimento orientado a serviços; na Seção 2 é apresentado a metodologia, ferramentas e procedimentos da pesquisa comparativa; na Seção 3 são apresentados os resultados e discussões sobre o experimento desenvolvido; e por fim, na Seção 4 são apresentadas as considerações finais.

2 FUNDAMENTAÇÃO TEÓRICA

Esta seção apresenta conceitos e definições sobre computação em nuvem, arquitetura orientada a serviços, arquitetura ausente de servidor e aplicativos de página única. Portanto, apesar da importância dos temas apresentados nesta seção, os mesmos não concentram o foco principal deste artigo, mas servem como suporte para as Seções 2, 3 e 4.

2.1 Computação em Nuvem

A computação em nuvem é uma maneira eficiente de maximizar e flexibilizar recursos computacionais, além disso, permite que seja mantido o funcionamento integral dos recursos computacionais em situação de falhas (TAURION, 2009). Neste sentido, a computação em nuvem disponibiliza recursos computacionais de forma remota na qual pode ser acessado por qualquer dispositivo com acesso à internet (IBM, 2021).

Segundo MELL e GRANCE (2012), a computação em nuvem pode ser classificada em quatro modelos de acordo com a responsabilidade de gerenciamento, sendo eles: (i) **nuvens privadas**: o acesso aos serviços é exclusivo de uma organização, sendo a infraestrutura mantida pela própria organização ou terceiros; (ii) **nuvem comunitária** consiste na disponibilização para um grupo específico que compartilhem um mesmo interesse em comum, podendo ser gerenciada por um ou mais integrantes do grupo ou também podendo ser gerenciado por um terceiro; (iii) **nuvem pública:** consiste em ser disponibilizada para o público em geral, onde a infraestrutura é localizada nas instalações da empresa que provê os recursos; e (iv) **nuvem híbrida:** consiste no conjunto de dois ou mais modelos de nuvem, como por exemplo uma nuvem que pode ser composta de uma parte que trabalhe com dados confidenciais no modelo de nuvem privada e outra parte que seja para uso de um grupo sendo no modelo de nuvem comunitária.

O modo como os serviços podem ser oferecidos por plataformas de computação em nuvem podem ser classificados em três categorias, de acordo com Nakamura (2017), são elas: (i) **infraestrutura como serviço** (do inglês, *Infrastructure as a Service* - IaaS): disponibiliza uma infraestrutura computacional virtual por meio de serviços, como por exemplo processamento, armazenamento, redes de computadores e sistemas operacionais; (ii) **plataforma como serviço** (do inglês, *Platform as a Service - PaaS*): dispõe de recursos para a publicação de aplicações de modo automatizado, disponibilizando linguagens de programação, bibliotecas e frameworks; e (iii) **software como serviço** (do inglês, *Software as a Service* - SaaS): disponibiliza sistemas de software de fácil gerenciamento por meio de interface gráfica de usuário por meio uma página web ou por meio de serviços web podendo ser integrado em outras aplicações.

2.2 Arquitetura Orientada a Serviços

A arquitetura orientada a serviços é um tipo de *design* de software que torna os componentes reutilizáveis por meio de interfaces de comunicação padronizadas, além disso, está presente na maioria das plataformas de computação em nuvem e de acordo com Red Hat (2021). Neste sentido, a SOA pode ser encarada como um paradigma de construção e integração de softwares com elementos modulares chamados de serviços.

De acordo com Fugita (2009), serviços desempenham funções específicas de negócio, podendo ser consumidos por meio de clientes, entretanto, os clientes que consomem os serviços não têm detalhes de seu funcionamento interno. Neste sentido, para Erl (2016), serviço é um software que disponibiliza suas funcionalidades por meio de uma API (do inglês *Application Programing Interface*) publicada.

As principais implementações de serviço web (do inglês, *Web Services*) amplamente adotadas pelo mercado são SOAP e REST. Ambas as implementações trabalham sobre protocolos abertos da internet como por exemplo o HTTP (sigla do inglês, *HyperText Transfer Protocol*), o que permite a comunicação entre serviços locais e serviços externos a organização. Serviços web utilizam padrões abertos para envio de mensagens como XML (do inglês, *EXtensible Markup Language*) e JSON (do inglês, *JavaScript Object Notation*), o que torna os serviços acessíveis e reutilizáveis por qualquer aplicativo (CHASEL, 2006).

Existem uma vasta gama de abordagens orientadas a serviços, entretanto, segundo Fowler (2014), as abordagens orientadas a serviços podem ser classificadas em duas grandes categorias, são elas: (i) **microsserviços**: é uma arquitetura em que uma necessidade de negócio

é dividida em pequenos processos individuais e específicos. Neste sentido, cada processo é um serviço web que pode ser acessado por meio de uma API web. Além disso, cada microsserviço pode ser escrito em diferentes linguagens de programação e utilizar diferentes tecnologias de banco de dados; e (ii) **monolítico**: nesta arquitetura toda regra de negócio é gerenciada por um único processo em memória, sendo que a aplicação é dividida logicamente por meio de técnicas de programação como programação orientada a objetos, funções e *namespaces*.

2.3 Arquitetura Ausente de Servidor

Na arquitetura ausente de servidor (do inglês, *Serverless Architecture*), a infraestrutura de hospedagem e ambiente de execução são terceirizadas para uma plataforma de computação em nuvem. Neste sentido, a abordagem *Serverless* é uma evolução do modelo PaaS, sendo que os desenvolvedores de software se preocupam apenas com a lógica do processamento das requisições feitas por clientes. A monetização nesse tipo de serviço leva em conta o período de execução de cada requisição (ROBERTS, 2021).

Na abordagem *Serverless*, o projeto do desenvolvimento de software é planejado como funções isoladas, independentes e frequentemente granulares que geralmente são executadas em serviços de computação sem estado (SBARSKI, 2017). Portanto, é uma arquitetura de software em que a aplicação é organizada em eventos e funções que podem ser consumidas por meio de uma API web (RAJAN, 2018).

A plataforma de computação em nuvem é responsável por manter a estrutura de hardware, sistema operacional e ambiente de execução. O ambiente de execução é composto por bibliotecas, *frameworks*, softwares servidores, variáveis de ambientes, entre outros. Os desenvolvedores de software implantam os serviços no ambiente e possuem a responsabilidade de implementar e implantar o serviço no ambiente *Serverless* (RAJAN, 2018).

2.4 Aplicativos de Página Única

Em aplicações de página única (do inglês, *Single-Page Application* - SPA), aplicativos web são executados em apenas uma página web. Nesta abordagem, a comunicação entre a aplicação cliente e o servidor acontece de modo assíncrono geralmente por meio de serviços web utilizando o formato de comunicação JSON (SCOTT, 2016). Neste sentido, de acordo com Scoot (2016), a principal vantagem da abordagem SPA em comparação abordagens *Server Side* é que em SPAs não é necessário recarregar arquivos do servidor.

Segundo Guedes (2018), em aplicações SPA, quando um usuário inicia uma aplicação, quase todo o conteúdo *front-end* é carregado para o navegador cliente. Deste modo, em transações subsequentes, o servidor envia para o navegador cliente somente dados de negócio em formato padronizado como JSON (GUEDES, 2018).

3 PROCEDIMENTOS METODOLÓGICOS

Este trabalho tem como objetivo apresentar uma pesquisa comparativa por meio de experimentos entre as abordagens monolítica e microsserviços em arquitetura ausente de servidor. Em uma primeira etapa foi realizado uma pesquisa bibliográfica (Seção 1) sobre os principais conceitos necessários para a implementação dos experimentos. Em uma segunda etapa foi selecionado a plataforma de computação em nuvem Amazon AWS[1] para o desenvolvimento e implantação das abordagens propostas.

3.1 Estudo de caso

Para realizar os experimentos deste artigo foi implementado uma aplicação web para uma cafeteria no qual é possível fazer pedidos para entrega sob delivery. A Figura 1 apresenta o design da aplicação para pedidos on-line de uma cafeteria. A aplicação SPA é executada por meio de navegador web, sendo que ao acessar a aplicação pela primeira vez, são carregados para o navegador cliente arquivos HTML e CSS que são responsáveis pela exibição da aplicação e arquivos de script Angular e TypeScript que são responsáveis pela execução da lógica de negócio no navegador. O processamento da listagem de produtos e pedidos ocorre no ambiente em nuvem no qual são requisitados por meio de requisições REST.

Figura 1 - Estudo de Caso

Fonte: Autoria própria

[1] https://aws.amazon.com/

3.2 Implementação

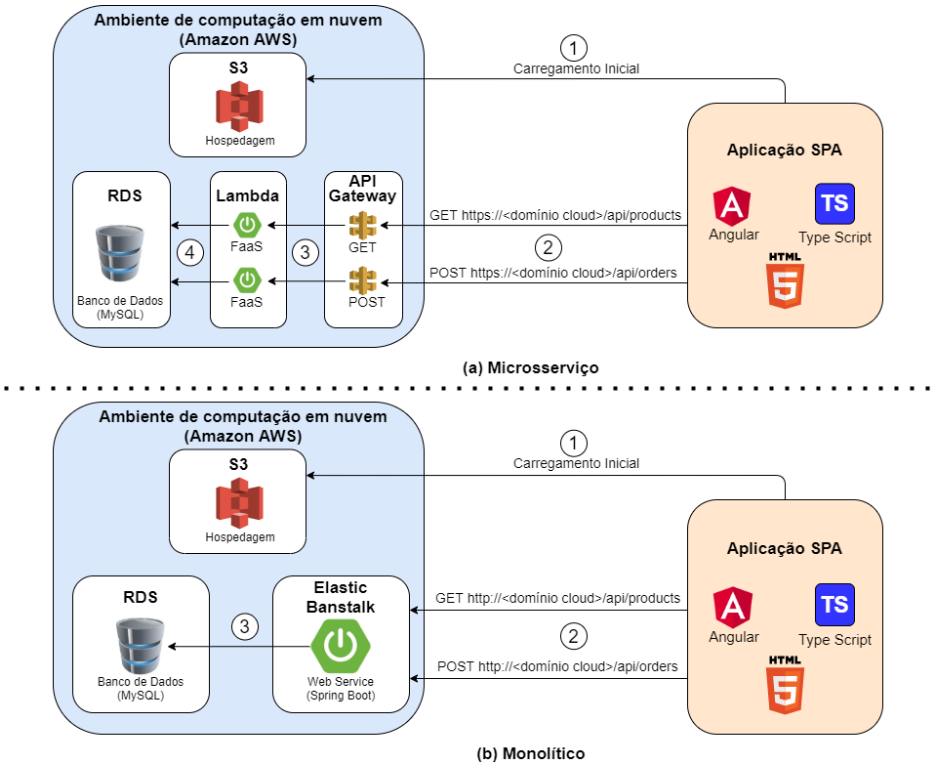
Para implementar as abordagens propostas neste artigo, foi utilizado um ambiente de produção real por meio de serviços disponíveis na plataforma de computação em nuvem *Amazon* AWS. A

Figura 2 ilustra os serviços e tecnologias utilizados na implantação da abordagem monolítica (Figura 2a) e microsserviços (Figura 2b), sendo as tecnologias: (i) **Banco de dados:** em ambas as abordagens, para armazenamento de dados foi utilizado o SBDG MySql[2] por meio do serviço RDS[3] (do inglês, *Relational Database Service*); (ii) **Hospedagem**: em ambas as abordagens foi utilizado uma interface web do tipo SPA, portanto, o servidor de hospedagem não necessita de recursos para gerar conteúdos dinâmicos. Neste sentido foi utilizado o serviço S3[4] que apenas hospeda arquivos estáticos; (iii) **Aplicação SPA**: em ambas as abordagens, foi utilizado o framework de desenvolvimento de aplicações SPA *Angular* que usa a linguagem de marcação HTML5 para o desenvolvimento das estruturas básicas da aplicação e a linguagem de programação Typescript para a implementação das ações que ocorrem no navegador cliente, como por exemplo, consumir os serviços web hospedados na plataforma Amazon; e (iv) **Ambiente de computação em nuvem**: para cada uma das abordagens, monolítica e microsserviços, foram adotados diferentes serviços. Na abordagem monolítica o serviço web REST foi implemento por meio do *framework Spring Boot*[5] e foi implantado no serviço *Elastic Beanstalk*[6]. O serviço *Elastic Beanstalk* é uma implementação PaaS, ou seja, possui ambientes de execução já instalados para diversas plataformas e linguagens de programação. Já na abordagem de microsserviços foram utilizados os serviços *Lambda*[7] *e API Gateway*[8]. O serviço *API Gateway* disponibiliza o serviço web REST no qual direciona as requisições web para o serviço *Lambda*. No serviço *Lambda*, um sistema deve ser decomposto em eventos e funções (conhecido do inglês, *Function as a Service* - FaaS), sendo que o desenvolvedor de software deve modelar cada um dos eventos e as funções relacionadas a cada ação do sistema. Para implementação das funções foi utilizado o *framework Spring Boot Cloud Function*.

[2] https://www.mysql.com/
[3] https://aws.amazon.com/rds/
[4] https://aws.amazon.com/s3/
[5] https://spring.io/
[6] https://aws.amazon.com/elasticbeanstalk/
[7] https://aws.amazon.com/lambda
[8] https://aws.amazon.com/api-gateway

Em ambas as abordagens, monolítica e microsserviços, os fluxos de execuções ocorrem de maneiras semelhantes nos passos 1 e 2 e se diferem nos outros passos. No passo 1, por meio de um navegador web, uma aplicação web é acessada por meio de um endereço URL (do inglês, *Uniform Resource Locator*) e carregada a partir de um serviço de hospedagem estática. A aplicação web possui duas funcionalidades: (i) obter produtos a serem exibidos (`/api/products`) e (ii) fazer pedido de produtos (`/api/orders`). Ambas as funcionalidades consomem serviços web hospedados no ambiente em nuvem, sendo essas funcionalidades representadas pelo passo 2. O processo responsável por disponibilizar os serviços web REST, processar regras de negócio e acessar banco de dados é diferente em ambas as abordagens. Na abordagem de microsserviços, as regras de negócio e disponibilização de serviços web REST são atribuídos a diferentes serviços na plataforma *Amazon* AWS. A disponibilização de serviços web são representadas pelo passo 2 por meio do serviço *API Gateway* e o passo 3 representa o redirecionamento da requisição de serviço web REST para o serviço *Lambda*. Na abordagem monolítica, a disponibilização de serviços web e regras de negócio ocorrem no serviço *Elastic Beanstalk.* O processo de acesso ao banco de dados ocorre de modo semelhante em ambas as abordagens, passo 3 na abordagem monolítica e passo 4 na abordagem de microsserviços.

Figura 2 - Abordagens Monolítica e Microsserviços



Fonte: Autoria própria

Como pode ser observado na abordagem monolítica, os serviços web e regras de negócio são organizados em um único processo em tempo de execução. Entretanto, na abordagem de microsserviços as regras de negócio são organizadas em funções, sendo que cada função ocupa um processo independente em tempo de execução. Neste sentido, no experimento apresentado neste artigo, as funções apresentadas na abordagem de microsserviços compartilham um mesmo banco dados, entretanto, é possível que cada função acesse bancos de dados diferentes.

4 RESULTADOS

Essa Seção tem como objetivo apresentar os resultados das implementações abordadas na Seção 2, além de um estudo comparativo entre as abordagens microsserviços e monolítica.

4.1 Segurança

A segurança em aplicações web é frequentemente negligenciada por desenvolvedores de softwares. Neste sentido, o trabalho de Shenbagam e Salini (2014) apontou que mais de 90% das aplicações possuíam algum tipo de vulnerabilidade sendo que a camada de aplicação recebeu em torno de 75% de ataques maliciosos. De acordo com os autores, a escassez de profissionais na área de segurança contribuía para esta situação. Contudo, de acordo com OWASP (2017), outro ponto que contribui para este cenário é a cultura existente no mercado de software, onde profissionais da área de desenvolvimento deixam de aplicar técnicas de segurança aliadas ao processo de desenvolvimento.

Nas abordagens monolítica e de microsserviços abordadas neste trabalho, foi observado que detalhes de segurança da camada de rede são transparentes para o desenvolvedor de software, ficando a responsabilidade para a plataforma de computação em nuvem. Entretanto, vulnerabilidades da camada de aplicação são de responsabilidade do desenvolvedor de acordo com o relatório "OWASP Top 10 API" produzido pela OWASP[9]. Neste sentido a abordagem de microsserviços trata a defesa de integridade das requisições web de modo transparente, sendo que o serviço *API Gateway* implementa o protocolo HTTPS por padrão. Embora seja possível integrar o protocolo de conexão HTTPS com o serviço Elastic Beakstalk, essa responsabilidade é de responsabilidade opcional do desenvolvedor de software.

[9] http://owasp.org

4.2 Responsabilidades de Configuração

Para a abordagem monolítica foi utilizado o serviço *Elastic Beanstalk* que é classificado como um PaaS. Isso significa que a infraestrutura do ambiente de execução que envolve serviços, frameworks e variáveis de ambiente é de responsabilidade da plataforma de computação em nuvem. Entretanto parâmetros do ambiente de execução podem ser customizados por meio de uma interfase simplificada. Na abordagem de microsserviços, foram utilizados os serviços *Gateway API* e *Lambda.* Nestes serviços a responsabilidade de configuração do desenvolvedor é menor em relação ao *Elastic Beanstalk,* sendo que existem poucos parâmetros que podem ser alterados e os serviços do ambiente de execução são transparentes para o desenvolvedor.

4.3 Monetização

O serviço *Elastic Beanstalk* utilizado na abordagem monolítica exige um gerenciamento mais complexo sobre a monetização de serviços sob demanda em relação ao serviço lambda da abordagem de microsserviços. Isso ocorre devido ao ambiente de execução do serviço *Elastic Beanstalk* possuir diversos serviços associados como instâncias de máquina virtual e ambientes de armazenamento de arquivos. O serviço *Lambda* torna o ambiente de execução invisível para o desenvolvedor, ficando exclusivamente a responsabilidade de implementar as funções e implantar no ambiente e execução. Neste sentido a cobrança do *Lambda* ocorre sobre a quantidade de requisições e tempo de execução de cada função implantada.

4.4 Performance

Uma função *Lambda* quando é requisitada possui um ciclo de execução, sendo as fases: (i) ***init***: essa fase ocorre diante da requisição da primeira chamada ou antes das chamadas das funções subsequentes. Portanto, é criado ou descongelado um ambiente de execução com os recursos previamente configurados, na sequência, ocorre a inicialização de extensões, o ambiente de execução é iniciado, e então o código da função é executado; (ii) ***invoke***: nesta fase um componente chamado invocador de funções é acionado para executar a função atual e possíveis novas funções por um tempo determinado pela plataforma; e (iii) ***shutdown*:** se por um período de tempo a função não é requisitada, o ambiente de execução é removido da memória.

Em sistemas que fazem requisições a serviços web frequentemente, ambas abordagens podem ser semelhantes em relação a performance de execução. No entanto, em sistemas que

fazem requisições a serviços web com baixa periodicidade, a abordagem monolítica pode apresentar melhor performance devido ao seu ambiente de execução estar sempre alocado em memória.

5 CONSIDERAÇÕES FINAIS

A computação orientada a serviços está presente em diversas plataformas de computação modernas, dentre elas destacam-se as aplicações web, computação móvel e internet das coisas. Neste sentido, a computação monolítica que possui um único processo em memória responsável por responder requisições web, evoluiu para aplicações compostas por microsserviços, que são mais flexíveis e capazes de serem reimplantadas ou substituídas em tempo de execução. Neste sentido, o objetivo primário deste artigo é por meio de experimentos em um ambiente de computação em nuvem, avaliar as principais características e similaridades entre as abordagens de softwares monolíticos, e, microsserviços sob a arquitetura ausente de servidores

Em relação a segurança de aplicações, observamos que uma abordagem ausente de servidores pode reduzir possíveis vulnerabilidades de segurança. De acordo com os trabalhos apresentados por Shenbagam e Salini (2014) e OWASP (2017), grande parte das vulnerabilidades existentes em aplicações são em decorrência de erros de configuração e customização de ambientes de execução. Neste sentido a arquitetura ausente de servidores elimina a responsabilidade dos desenvolvedores de configurar e gerenciar ambientes de execução.

Sobre o processo de monetização, a abordagem de microsserviços é mais econômico em relação a abordagem monolítica em modelos de cobrança sob demanda. Isso ocorre devido a abordagem monolítica apresentar serviços dedicados para a execução de uma aplicação, em contrapartida, a abordagem de microsserviços possui infraestruturas computacionais compartilhadas com outras aplicações. Em resumo, a abordagem de microsserviços é monetizada apenas quando ocorre o processamento das requisições, já a abordagem monolítica, a cobrança ocorre mesmo em períodos que não haja requisições aos serviços.

Na avaliação de performance observamos que na abordagem de microsserviços, em uma sequência de requisições, a primeira requisição leva mais tempo de processamento em relação as requisições subsequentes. Isso ocorre devido a mecânica de funcionamento do

serviço *Lambda,* onde em momentos de ociosidade, a função é removida da memória do ambiente de execução. Quando ocorre uma requisição em que a função não esteja na memória, existe um tempo de carregamento da função em memória. Em contrapartida, a abordagem monolítica mantém seu ambiente operacional pronto para receber novas requisições, portando não existem tempo extra de carregamento do serviço em memória.

Acreditamos que este artigo possa direcionar novas pesquisas na área de computação orientada a serviços ou guiar engenheiros de softwares em projetos de softwares. Em trabalhos futuros pretendemos realizar novos experimentos focados em segurança e performance nas abordagens monolíticas e microsserviços. Neste sentido pretendemos explorar outros serviços da plataforma Amazon AWS além de outras plataformas de computação em nuvem como por exemplo Microsoft Azure, Google *Cloud* e IBM *Cloud*.

REFERÊNCIAS

BARRY, D.; DICK, D. **Web Services, Service-Oriented Architectures, and Cloud Computing**-*2.ed.* [S.l.]: Elsevier, 2013.

CHASEL, N. **Understanding web services specifications**. 2006. Disponível em: https://www.ibm.com/developerworks/webservices/tutorials/ws-understand-web-services1/ws-understand-web-services1.html. Acessado em 01/03/2021.

ERL, T. **Service-Oriented Architecture.** Prentice Hall, 2016.

FOWLER, M.; LEWIS, J. **Microservices**. 2014. Disponível em: https://martinfowler.com/articles/microservices.html. Acessado em 01/04/2021.

FUGITA, Henrique Shoiti. **MAPOS: Método de Análise e Projeto Orientado a Serviços**. 2009. Dissertação (Mestrado) - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Computação e Sistemas Digitais, São Paulo, 2009.

GIL, Antônio Carlos. **Como elaborar projetos de pesquisa.** 4ª. ed. São Paulo. Atlas S.A. 2002

GIL, Antônio Carlos. **Métodos e técnicas de pesquisa social**. 6ª. ed. São Paulo. Atlas S.A. 2008

GUEDES, T. **Crie aplicações com Angular - O novo framework do Google**. São Paulo. Casa do Código, 2018.

KARWAN, J.; ABASS, S. **Development History Of The World Wide Web**. International Journal of Scientific & Technology Research 8: 75-79, 2019. Disponível em: https://bit.ly/3kRewHY. Acessado em 21/11/2020.

MELL, P.; GRANCE, T. **The NIST definition of Cloud Computing**. 2012. Disponível em: https://csrc.nist.gov/publications/detail/sp/800-145/final. Acessado em 01/04/2021.

NAKAMURA, L. **Mecanismos de autoconfiguração e auto otimização para arquiteturas virtualizadas que visam a provisão de qualidade de serviço**. 2017. Tese (Doutorado em Ciências da Computação e Matemática Computacional) -Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos, 2017.

OWASP. Code **Review Guide 2.0: Release**. 2017. Disponível em: https://bityli.com/0XrL9. Acessado em 01/04/2021.

OWASP. **OWASP API Security Top 10 2019**. 2019. Disponível em: https://bityli.com/eEgqd. Acessado em 01/04/2021.

RAJAN, R. Arokia Paul. **Serverless Architecture**: **A Revolution in Cloud Computing**. In: *2018 Tenth International Conference on Advanced Computing (ICoAC)*. [S.l.]: IEEE, 2018. Disponível em: https://bityli.com/9nqS8. Acessado em 01/04/2021.

RED HAT, **O que é arquitetura orientada a serviços (SOA)?** Disponível em: https://bityli.com/pi6N5. Acessado em 01/04/2021.

ROBERTS, Mike. **Serverless Architectures.** 2018**.** Disponível em https://bityli.com/2LtA0. Acessado em 03/03/2021.

SBARSKI, PETER. **Serverless Architectures on AWS**. New York Manning Publications Co, 2017.

SCOTT, A. **SPA Design and Architecture**. New York. Manning Publications Co., 2016.

SHENBAGAM, J.; SALINI, P**. Vulnerability ontology for web applications to predict and classify attacks.** In: 2014 International Conference on Electronics, Communication and Computational Engineering (ICECCE). [S.l.]: IEEE, 2014. Disponível em: https://bityli.com/Zgml0. Acessado em 01/04/2021.

WASSON, M. A**plicativos de página única: Crie aplicativos Web dinâmicos e modernos com o ASP.NET**. 2015. Disponível em: https://bityli.com/pIXpw. Acessado em 01/04/2021.