

De *cold starts* a precificação: Uma análise comparativa na execução de aplicação *Node.JS* nas plataformas *AWS Lambda* e *Azure Functions*

From cold starts to pricing: A comparative analysis in executing Node.JS application on AWS Lambda and Azure Functions

Valter Canhizares Filho¹, Renata Spolon Lobato¹, Aleardo Manacero Jr.¹, Roberta Spolon²

¹Departamento. de Ciências de Computação e Estatística - Instituto de Biociências, Letras e Ciências Exatas, São José do Rio Preto - SP

²Departamento de Computação - Faculdade de Ciências, Bauru - SP
Universidade Estadual Paulista “Júlio de Mesquita Filho” – UNESP
Brasil

{valter.canhizares, renata.spolon, aleardo.manacero, roberta.spolon}@unesp.br

Resumo — Nos últimos anos, é observável um notável aumento no interesse tanto comercial quanto acadêmico em computação em nuvem, impulsionando um novo paradigma da computação sem servidor. Esta nova abstração oferece ao desenvolvedor uma maior agilidade na criação de aplicações, redução de custos, escalabilidade e diminuição do tempo para aplicações chegarem até seus usuários. O desenvolvimento deste trabalho se dá pela apresentação do estado da arte da computação sem servidor e comparação de desempenho e custo monetário de uma aplicação em *Node.JS* como função sem servidor e como container em provedoras de computação em nuvem *AWS* e *Azure*.

Palavras-Chave – *computação sem servidor, nuvem, containers, provedoras, funções, Node.JS.*

Abstract — In recent years, there has been a noticeable surge in both commercial and academic interest in cloud computing, driving a new paradigm of serverless computing. This innovative abstraction provides developers with increased agility in application development, cost reduction, scalability, and a significant decrease in the time it takes for applications to reach their customers. The development of this work involves presenting the state of the art in serverless computing and comparing the performance and monetary cost of an application in *Node.JS*, both as a serverless function and as a container, across cloud computing providers such as *AWS* and *Azure*.

Keywords – *serverless computing, containers, serverless providers, functions, Node.JS.*

I. INTRODUÇÃO

Na última década, testemunham-se avanços notáveis na computação em nuvem, evidenciados pelo crescente investimento da indústria e por estudos acadêmicos. De acordo com [1], estima-se que o mercado de computação em nuvem atinja

um volume de revenda próximo a 700 bilhões de dólares até 2025, impulsionado por soluções relacionadas à infraestrutura em nuvem como serviço, inteligência artificial e computação sem servidor.

Atualmente, diversas provedoras oferecem aplicações em nuvem, incluindo gigantes como Amazon, Microsoft e Google com os serviços *AWS*, *Azure* e *GCP*, respectivamente. Todos os serviços distribuem capacidade computacional para consumidores dispostos a pagar pelo consumo.

Novas tendências e direções intensificam a necessidade e popularidade de computação em nuvem. Arquiteturas emergentes, como a computação definida por *software* e a computação sem servidor, estão se consolidando como tendências para as próximas gerações de computação em nuvem. Além disso, é relevante observar o crescimento das tendências em infraestruturas de nuvens heterogêneas [2].

A computação em nuvem vem sendo adotada cada vez mais pelas empresas para otimizar seus processos, seja por meio de infraestrutura como serviço (IaaS), plataforma como serviço (PaaS) ou como *software* como serviço (SaaS). Características como por exemplo, alta disponibilidade, elasticidade e medição do consumo têm colocado a computação em nuvem no centro das atenções da computação contemporânea.

Dessa forma, este trabalho oferece um panorama do estado da arte da computação sem servidor, modelo recente que se encontra como uma abstração de *Softwares* como Serviço na nuvem e apresenta uma aplicação prática em uma aplicação *web* moderna em *Node.JS*.

No contexto da pesquisa, [3] propõem em sua taxonomia que uma das linguagens de programação mais utilizadas no

contexto *serverless* é o *JavaScript*, com *containers* sendo o componente mais utilizado para o desenvolvimento. Ademais, a maioria das aplicações *serverless* opera na camada de aplicação HTTP, sendo que todas essas três tecnologias são empregadas no presente trabalho.

O objetivo deste trabalho é apresentar o estado da arte da computação *serverless* e comparar o desempenho e custo monetário de uma aplicação *web* implementada na nuvem por meio de funções sem servidor e implementada como um *container*, outro modelo de programação nativo da nuvem. As plataformas *Amazon AWS* e *Microsoft Azure* foram utilizadas para o desenvolvimento do presente trabalho.

Busca-se, conseqüentemente, expandir a discussão sobre esse novo paradigma *serverless* em um ambiente inovador da computação em nuvem, além de analisar possibilidades de desenvolvimento mais rápido e barato com implementações em funções e *containers*, o que traz maior benefício para a aplicação *Node.JS*.

Na Seção II são abordadas características técnicas. Na Seção III, apresenta-se a metodologia de desenvolvimento do trabalho. Na Seção IV se discute os resultados e conclusões.

II. CONCEITOS TÉCNICOS

A. Definição de Computação em Nuvem

A computação em nuvem pode ser encontrada como nuvem privada ou nuvem pública, bem como a possibilidade de usar de forma híbrida, a depender do investimento de bens de capital (CAPEX) ou despesas operacionais (OPEX) de uma determinada empresa que quer se beneficiar da tecnologia de processamento e armazenamento de dados. Ao utilizar a computação em nuvem, os usuários podem aproveitar recursos distribuídos provenientes de *datacenters* de fornecedores conhecidos no mercado, como Microsoft e Amazon.

Essas empresas oferecem serviços com rápido provisionamento, possibilitando computação ubíqua sob demanda, com diferentes níveis de configuração para os usuários deste modelo. A computação em nuvem permite aos usuários abstrair toda a gestão de servidores físicos, redes, armazenamento e aplicações, conferindo flexibilidade na personalização desses elementos de acordo com as necessidades individuais [4]. Os mesmos autores destacam a composição do modelo de computação em nuvem por meio de cinco características principais e três modelos de serviço.

Serviço sob demanda, acesso amplo à rede de computadores, aglomerado de recursos computacionais, rápida elasticidade e medição do serviço consumido são características da computação em nuvem. Três modelos, com três níveis de abstração diferentes compõem a aplicação da computação em nuvem [4].

Diferentes modelos de computação em nuvem são distribuídos pelas provedoras, níveis diferentes de controle do cliente sobre sua infraestrutura e aplicações. Diferentes níveis de controle refletem a diferentes métricas e precificação. [4] dividem em SaaS – *Software as a Service*, PaaS – *Platform as a Service* e IaaS – *Infrastructure as a Service*.

A literatura e o mercado elencam muitas possibilidades “como serviço”, *XaaS – Everything as a Service* [5], o presente trabalho utiliza a nomenclatura FaaS – Função como Serviço e CaaS – *Container* como Serviço para se referir a funções *serverless* e provisionamento de *containers*, respectivamente.

B. Definição de computação sem servidor - *serverless*

Diversas aplicações não requerem alocação completa de máquinas virtuais para seu provisionamento. Nos últimos anos, têm surgido conceitos de otimização visando aprimorar a utilização da capacidade computacional em nuvem, proporcionando uma solução mais econômica para um número maior de clientes. Nesse contexto, a abstração da computação sem servidor emergiu como uma alternativa viável, adotando um modelo de precificação diferenciado do convencional, que cobra apenas pela execução da aplicação em si, e não pela infraestrutura completa da máquina virtual utilizada pela aplicação. A terminologia “sem servidor” não implica que a computação seja desprovida de servidores. No contexto atual, significa que uma parte do servidor ou de uma máquina virtual é alugada para a execução de uma aplicação específica, por um período mais breve em comparação com a computação em nuvem convencional [2]. Em outras palavras, na computação sem servidor, o desenvolvedor incorre de custos relacionados à execução de um segmento específico de código, não pela locação de uma máquina virtual ou *container* inteiro.

A abstração sem servidor na computação em nuvem permite que os desenvolvedores executem suas aplicações com um custo granular associado ao uso, eliminando a necessidade de configurar sistemas físicos e lógicos responsáveis pela execução do código [6]. Tais aplicações são executadas com uma abstração eficiente da infraestrutura em nuvem, proporcionando aos desenvolvedores a liberdade de concentrar-se apenas na execução do código, sem preocupações relacionadas ao provisionamento de máquinas virtuais, *bare metal*, ou outras configurações complexas [7].

A computação *serverless* teve seu primeiro modelo introduzido pela Amazon com a plataforma *AWS Lambda* em 2014. Posteriormente, com avanços na indústria e na academia, outras provedoras de computação em nuvem, como por exemplo, *Google Serverless Computing* e *Azure Serverless*, passaram a oferecer esse modelo. Com o objetivo de agilizar o desenvolvimento de soluções, a provedora assume a responsabilidade de provisionar, gerenciar e escalar servidores, máquinas virtuais e *containers*, permitindo que os desenvolvedores foquem apenas no carregamento do código de seus programas. Na Fig. 1, ilustra-se um exemplo do funcionamento de uma função *serverless*, uma tarefa que é executada ao ser acionada por um gatilho.



Figure 1. Exemplo de função *serverless* de redimensionamento de imagens, na plataforma *AWS Lambda*.

Por ser um paradigma emergente na computação em nuvem, a computação *serverless* é aplicada em diversas áreas, incluindo aprendizado de máquina, aprendizagem profunda, computação numérica, processamento de vídeo, Internet das Coisas e análise de *big data* [8]. Um estudo indica que até 2025, 50% das empresas que utilizam a computação em nuvem utilizarão computação sem servidor, gerando um mercado para tais aplicações avaliadas em mais de 22 bilhões de dólares, de acordo com a [9].

As características únicas da computação sem servidor têm atraído a atenção da comunidade de desenvolvedores de software para a criação de aplicações baseadas em *serverless* (*serverless-based applications*). Diversos trabalhos acadêmicos abordam desafios que envolvem a evolução desse paradigma [10], modelagem de aplicativos [11], *frameworks* de programação específicos [12], desenvolvimento de plataformas de nuvem heterogêneas [13], entre outros.

Uma implementação de computação sem servidor são as conhecidas *Function-as-a-Service* (FaaS). O funcionamento dessas funções é acionado por gatilhos ou eventos pré-definidos pelo desenvolvedor [14], que executam pedaços de código. Nesse cenário, a provedora automaticamente provisiona a execução do código de forma escalável e isolada, podendo utilizar as mesmas máquinas virtuais para a execução de múltiplas funções isoladas umas das outras, em *containers*, evidenciando a otimização do modelo pelas provedoras.

É possível encontrar vantagens e desvantagens ao utilizar a computação *serverless* e analisar como os aspectos positivos e negativos se comparam com os demais modelos de computação em nuvem. Como vantagens é possível encontrar, tem-se escalonamento ágil, ausência de despesa financeira pela ociosidade, escalabilidade sem custo adicional, redução de custo operacional, *fast time to market* e uso eficiente dos recursos, de responsabilidade de operadora contratada.

Ilustra-se na Fig. 2 uma comparação entre computação tradicional, computação em nuvem e a computação sem servidor, feita no ponto de vista da alocação de recursos computacionais e tempo de execução.



Figure 2. Computação tradicional versus computação em nuvem versus *serverless* [15]

É notável que a computação em nuvem pode, em alguns casos custar menos capacidade computacional que o modelo

de computação *on premises*, entretanto, a computação *serverless* atinge menos recursos computacionais empregados para determinada aplicação.

Em contrapartida, desvantagens observadas são desafios relacionados a desempenho de aplicações específicas, latência na inicialização de funções, conhecido como *cold start*, aprisionamento da provedora de computação em nuvem, ou *lock-in*, segurança na execução de funções em ambientes compartilhados, precificação na execução de funções e sistemas legado.

O *cold start* pode ser elencado como um dos maiores desafios investigados na academia, por adicionar maior tempo de execução às funções sem servidor, muitos trabalhos buscam mitigar tal desafio. Na Fig. 3 mostra-se um exemplo do *cold start* em relação ao tempo de execução de uma função.

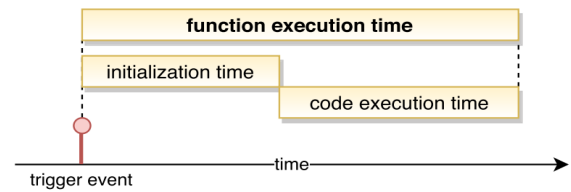


Figure 3. Exemplificação do *cold start* na execução de uma função [14].

Em relação às linguagens de programação suportadas pelas plataformas *serverless*, tem-se: *JavaScript*, *Java*, *C#*, *F#*, *Go*, *Python*, *PowerShell*, entre outras. Cada linguagem pode apresentar desempenho diferente e tamanhos de pacotes de execução diferentes, resultando em custos monetários diversos para cada especificidade.

Por fim, uma comparação pode ser observada na Tabela I, que leva em consideração aspectos como facilidade de desenvolvimento de aplicações, escalabilidade, ciclo de vida de uma função, segurança, resolução de problemas, tolerância a falhas, custo de carga variável carga estática e usuário para qual é voltado.

TABLE I. COMPARAÇÃO ENTRE *SERVERLESS* E COMPUTAÇÃO TRADICIONAL

Característica	Tradicional	Sem Servidor
Desenvolvimento de aplicações	Difícil	Fácil
Escalabilidade	Difícil	Fácil
Ciclo de vida da função	Longa	Rápida
Resolução de Problemas	Fácil	Difícil
Tolerância a falhas	Menos confiável	Mais confiável
Usuário	Administrador e Desenvolvedor	Desenvolvedor

III. DESENVOLVIMENTO

Duas provedoras de computação em nuvem foram usadas para medir o desempenho de uma aplicação previamente migrada de uma plataforma *on premises*. Parte do *backend* de uma aplicação em *JavaScript*, o módulo *sharp* do *Node.js*, é implementada como função no *AWS Lambda* e *Azure Function*, bem como *container* no *AWS Fargate* e *Azure Containers*.

O módulo *sharp* é responsável por fazer o redimensionamento de imagens cadastradas em um *frontend*, na Fig.4 demonstra-se o diagrama na plataforma *AWS*, como exemplo, similar em sua estrutura à plataforma *Azure*, que difere do nome de seus produtos.

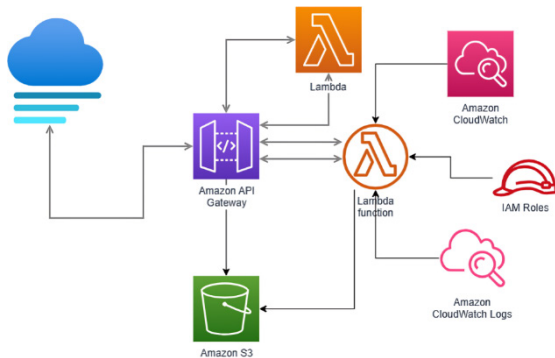


Figure 4. Diagrama da estrutura da aplicação.

Os testes para obter resultados foram conduzidos por meio da ferramenta *Apache JMeter*, responsável por simular cargas de requisições na aplicação. Foram divididos em testes de carga incremental e picos de carga, em horários distintos do dia, em regiões diferentes da nuvem, afetando também a latência e custo monetário da aplicação.

Para os testes, as regiões escolhidas combinam melhor latência e melhor custo: regiões de São Paulo (*sa-east-1*) e Virgínia do Norte (*us-east-1*). Ambas as plataformas permitem configurar quantidade de memória RAM e processadores virtuais que serão alocadas para execução da aplicação, a plataforma *AWS* ainda permite escolher entre processadores *ARM* e *x86*, o que impacta em diferente custo, já o *Azure* não permite tal escolha.

Inicialmente, para execução de função, foi necessário encontrar a melhor quantidade de memória RAM alocada, testes aleatórios com diferentes configurações foram realizados para se obter o melhor custo-benefício na execução da aplicação, com a função empacotada com 14 *megabytes* e apenas um processador virtual alocado.

Na Fig. 5 ilustra-se que o melhor balanço entre tempo de resposta-custo-memória foi 512 *megabytes* alocados na execução, além de um único processador virtual. Paralelamente a isso, alocação em *container* manteve-se com as mesmas configurações. Destaca-se a importância de se manter as menores latências e custos possíveis, com a maior memória possível.

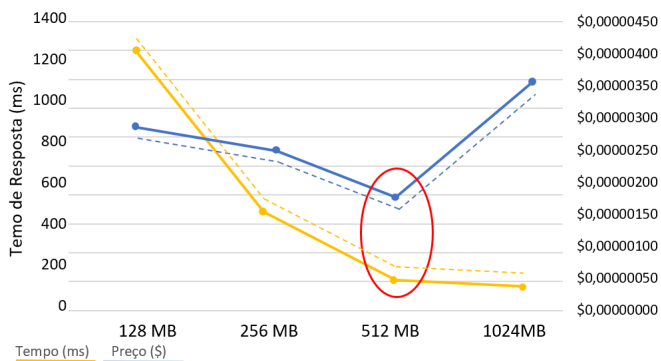


Figure 5. Melhor *setup* para execução *serverless*.

O preço e o tempo de execução são estimativas advindas de ambas as provedoras, baseado no mesmo tamanho de pacote da biblioteca *sharp*, do *Node.JS*, em JavaScript.

Em relação ao custo

Com a premissa de custo granular inerente ao uso, na Fig.6 demonstra-se como o custo monetário se deu com base no comportamento de execução de aproximadamente 10 milhões de execuções, da forma de funções *serverless* e *containers*.

É constatado que a execução na região *us-east-1* é ligeiramente mais barata que a mesma execução na região *sa-east-1*, além disso é visível que arquiteturas *ARM* tendem a ser mais baratas que arquiteturas *x86*.

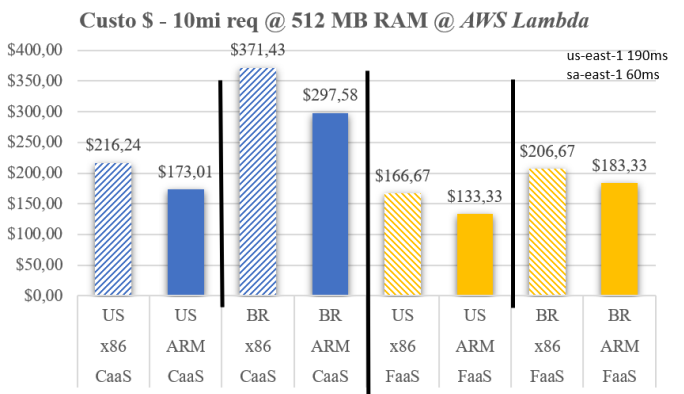


Figure 6. Custo em dólares de 10 milhões de requisições *serverless* (FaaS) e *containers* (CaaS) na *AWS Lambda*.

Em relação às execuções na provedora *Azure*, a plataforma se mostra menos personalizável em relação à arquitetura de processamento das funções, não dando opção de escolha no pacote básico do serviço, apenas customização no pacote *premium*. Na Fig. 7 ilustra-se a distribuição de custo nas mesmas regiões do *AWS Lambda*, destacando-se positivamente pela menor latência, mas, em contrapartida com maior custo monetário para a execução das funções se comparado à *AWS Lambda*, chegando a custar 20 dólares a mais pela mesma configuração e mesma quantidade de execuções, na região *sa-east-1*.

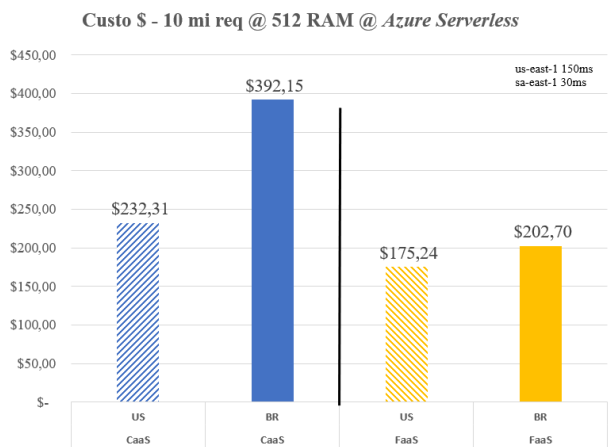


Figure 7. Custo em dólares de 10 milhões de requisições *serverless* (FaaS) e *containers* (CaaS) na *Azure Serverless*.

A região *us-east-1* se mostra mais barata que a região brasileira de execução, apesar do maior custo. É esperado que para cada aplicação se tome a decisão de onde hospedar. Aplicações críticas, por exemplo, requerem menor latência de resposta, pois além da latência de regiões, existe o *cold start* das aplicações *serverless*.

É importante observar microscopicamente como o custo se comporta para cada provedora, região e modalidade de computação. Em seguida, na Fig. 8, é possível observar a variabilidade do custo entre as regiões e provedoras, cada tipo de aplicação pode se comportar de forma diferente no que tange o custo. As informações de custo contidas na Fig. 8 resumizam dados encontrados nas Figuras 6 e 7, com mais detalhes de regiões e arquiteturas utilizadas.

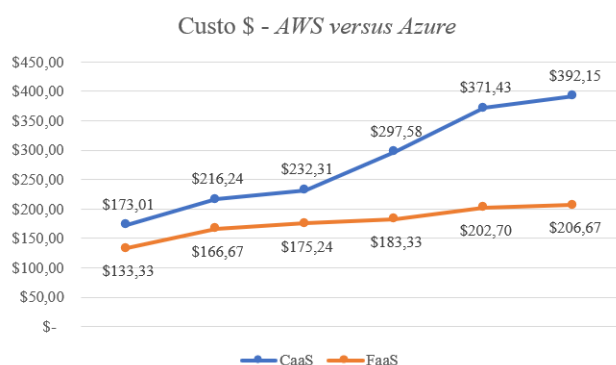


Figure 8. Comparação entre *containers AWS* e *Azure*, em dólares, por região e arquitetura.

Observa-se que, de maneira geral, serviços hospedados na região *us-east-1* são mais baratos do que os hospedados na região *sa-east-1*, a provedora *AWS Lambda* lidera com melhores preços em relação à *Azure Functions* e, arquitetura *ARM*, presente na *AWS* tende a ser mais barato. Também é possível encontrar semelhanças à comparação ao custo de serviços na região de *us-east-1* e a provedora *AWS Lambda*, de maneira geral. Destaca-se a diferença do custo para a execução da mesma aplicação *serverless* e *container*. A execução mais cara *serverless*, de \$206,67 está próxima da segunda mais barata em *container*, de \$216,24.

Em relação ao *cold start*

Outro ponto a ser observado é em relação ao *cold start* e a maneira que as aplicações se comportam com tal característica *serverless*. As provedoras de computação sem servidor oferecem dinamicamente adição e remoção de instancias que executam as funções. Quando uma nova instância trata sua primeira requisição, o tempo de resposta aumenta, o que é chamado de arranque a frio, ou *cold start*. A primeira inicialização a frio acontece quando a primeira requisição chega após o provisionamento da função.

Depois que a solicitação é processada, a instância permanece ativa para ser reutilizada em requisições subsequentes. A estratégia de reutilização difere entre uma provedora e outra. A *AWS* possui políticas para reciclar uma instância ociosa após um período predefinido, já o *Azure* leva em consideração

histórico de invocações anteriores, potencialidade de execução de uma função, por isso se dá a significativa diferença que pode ser observada na Tabela II. Com relação aos serviços de *container*, não há vida útil pré-definida pela plataforma, e sim pelo desenvolvedor que a mantém.

TABELA II. tempo de vida útil de uma instância

Serviço	Vida útil da instância ociosa
<i>AWS Lambda</i>	De 5 a 7 minutos
<i>Azure Function</i>	De 20 a 30 minutos
<i>AWS Fargate</i>	Definida pelo usuário
<i>Azure Containers</i>	Definida pelo usuário

Depois que a primeira requisição é processada, a instância continua ativa para poder ser reutilizada para requisições subsequentes. Nas Fig. 9 e 10 ilustra-se dados empíricos da probabilidade de uma instância ser reciclada após um período de inatividade, nas provedoras *AWS* e *Azure*, respectivamente, representadas pelas linhas contínuas. Além disso, nas imagens é possível encontrar os segundos “quentes” de inicialização, em vermelho e, em azul, os arranques a frio, representadas pela distribuição de pontos.

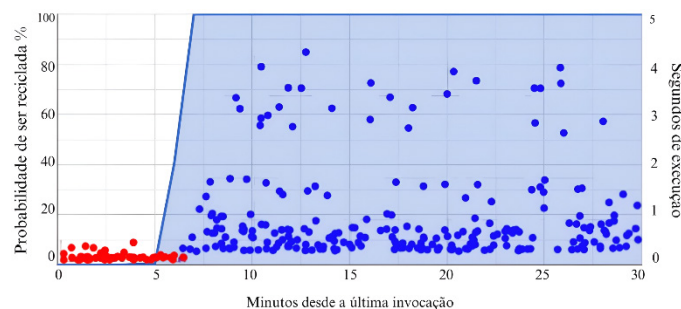


Figure 9. Período de reciclagem de instância e tempos de execução em segundos da plataforma *AWS Lambda*.

Notavelmente as instâncias alocadas no *Azure Functions* tendem a ser recicladas com maior ociosidade quando se comparado ao *AWS Lambda*. É importante constatar também que, é uma alternativa viável concentrar a execução das requisições em um pequeno intervalo de tempo após a alocação da instância para execução de funções.

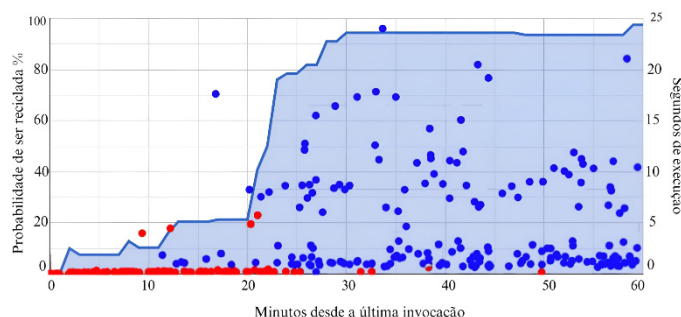


Figure 10. Período de reciclagem de instância e tempos de execução em segundos da plataforma *Azure Functions*.

Ademais, a distribuição gráfica de ambas as plataformas é parecida, mas a distribuição temporal do tempo de execução é distante entre uma e outra, o tempo de execução das funções na plataforma *AWS Lambda* representa cerca de 20% do tempo gasto em relação à plataforma *Azure Functions*.

A partir de um olhar aproximado para a diferença de tempo entre a execução de uma plataforma à outra, observa-se, na Fig. 11 uma amostragem do tempo de execução da mesma função em segundos.

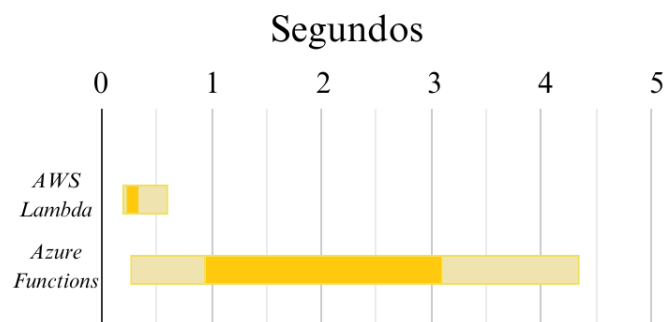


Figure 11. Comparativo entre a execução de funções *Node.JS* nas plataformas *AWS Lambda* e *Azure Functions*.

A plataforma *serverless* da *AWS* oferece tempo de execução inferiores a um segundo, *Azure* atinge mais de 4 segundos para a mesma função, com as mesmas configurações e parâmetros, com instância *Linux*. A parte mais escura representa a maioria das execuções, cerca de 70%.

IV. CONCLUSÕES E TRABALHO FUTURO

Conclui-se que, com base nos testes realizados nas plataformas *serverless*, ambas oferecem suporte para uma ampla gama de linguagens de programação, sendo que o desempenho pode variar conforme a linguagem utilizada. Portanto, cabe ao desenvolvedor realizar testes e determinar qual é a melhor opção para cada especificidade do projeto.

Ao comparar provedoras, observa-se que a *AWS Lambda* oferece maior configurabilidade em relação à instância a ser provisionada, em comparação ao *Azure Functions*. Isso pode ser atribuído, em parte, ao maior tempo de mercado do serviço da *Amazon*, que está mais consolidada e maduro.

Em relação às duas modalidades de computação, os *containers* oferecem ao desenvolvedor um controle mais direto sobre a vida útil do ambiente. Por outro lado, os custos na computação *serverless* podem ser mais variáveis, em contraste com a maior linearidade e previsibilidade dos custos nos *containers*.

Por fim, *containers* são *stateful* e *serverless* são *stateless*, torna-se viável combinar os dois, a depender do tamanho da

aplicação e em como ela pode ser fragmentada em serviços de nuvem.

Como trabalho futuro, é possível explorar testes de desempenho e custos a ser aplicados em outras provedoras de computação sem servidor como *Google Cloud Platform*, *Oracle Cloud*, bem como aplicações *serverless* de código aberto, como *Apache OpenWhisk*, *Fission*, *OpenFaaS*, *Kubeless*, entre outros.

AGRADECIMENTOS

Os autores expressam seus agradecimentos para a UNESP (Universidade Estadual Paulista “Júlio de Mesquita Filho”), FAPESP (“Fundação de Amparo à Pesquisa do Estado de São Paulo”), CAPES (“Fundação Coordenação de Aperfeiçoamento de Pessoal de Nível Superior”) e aos demais envolvidos que contribuíram cientificamente para o sucesso desta pesquisa. O presente trabalho foi parcialmente financiado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES)

REFERÊNCIAS BIBLIOGRÁFICAS

- [1] U. Udunwa, et al. Review of Cloud Computing in Low-Income Countries (LICs): Prospects and Challenges. *International Journal of Applied Engineering Research* (2019), ISSN 0973-4562, p. 3531-3536.
- [2] B. Vaarghese e R. Buyya. Next generation cloud computing: new trends and research directions. 2018. *Future Generation Computer Systems*, 79, 849–861. doi: 10.1016/j.future.2017.09.020.
- [3] G. A. S. Cassel, et al. Serverless computing for Internet of Things: A systematic literature review. In *Future Generation Computer Systems*. 2020. Volume 218, p 299-316. Doi 10.1016/j.future.2021.10.020
- [4] P. Mell e T. Grance, The NIST Definition of Cloud Computing. SP 800-145. 2011. doi: 10.6028/NIST.SP.800-145.
- [5] Y. Duan, G. Fu, N. Zhou, X. Sun, N. Narendra, B. Hu. Everything as a Service (XaaS) on the Cloud: Origins, Current and Future Trends. 2015 IEEE 8th International Conference on Cloud Computing. doi: 10.1109/CLOUD.2015.88.
- [6] E. Van Eyk, L. Toader, et al. Serverless is More: From PaaS to Present Cloud Computing. *IEEE Internet Computing*, 22(5), 8–17. doi:10.1109/mic.2018.053681358.
- [7] P. Casto, V. Ishakian e V. Muthusamy. The rise of serverless computing. *Communications of the ACM*, 62 (12), 44-54, doi 10.1145/3368454.
- [8] J. Wen, et al. Rise of the Planet of Serverless Computing: A Systematic Review. *ACM Transactions on Software Engineering and Methodology*. Doi: 10.1145/3579643.
- [9] Gartner. The CIO’s guide to serverless computing. Disponível em: <https://www.gartner.com/smarterwithgartner/the-cios-guide-to-serverless-computing>.
- [10] D. Taibi, J. Spillner, K. Wawruch. Serverless computing-where are we now, and where are we heading? 2020. *IEEE software* 38, 1, 2531
- [11] S. Eisman, et al. The state of serverless applications: collection, characterization, and community consensus. *IEEE Transactions on Software Engineering*. 2021.
- [12] D. Bermach, et al. AuctionWhisk: Using an auction-inspired approach for function placement in serverless fog platforms. *Software: Practice and Experience* 52, 5. 2022, 1143-1169.
- [13] J. Sampé, et al. Toward multicloud access transparency in serverless computing. *IEEE Software* 38, 1. 2020, 68-74.
- [14] M. Steinbach, et al. TppFaaS: Modeling Serverless Functions Invocations via Temporal Point Processes. *IEEE Access*, v. 10, p. 9059–9084, 2022. doi: 10.1109/ACCESS.2022.3144078.
- [15] J. Schleier-Smith, et al. What serverless computing is and should become. *Communications of the ACM*, 64(5), 76–84. doi:10.1145/3406011.