Avaliação de modelos de implantação de funções serverless no serviço AWS Lambda

Gabriel Duessmann
Programa de Pós-Graduação em Computação Aplicada
Universidade do Estado de Santa Catarina
Joinville, Brasil
gabriel.duessmann@gmail.com

Adriano Fiorese
Departmento de Ciência da Computação
Universidade do Estado de Santa Catarina
Joinville, Brasil
adriano.fiorese@udesc.br

***Resumo*—Com o avanço da computação em nuvem e serviços *serverless*, mais foco essa área vem ganhando nos últimos anos. Provedores de nuvem oferecem serviços relacionados a *serverless*, e em particular a Amazon disponibiliza o AWS Lambda para a criação de funções *serverless* pelos seus clientes. Existem ao menos duas forma de implantação de funções *serverless*. Sendo assim, uma forma encapsula o código fonte e demais arquivos necessários em um arquivo compactado no formato ZIP, e outra onde a própria função executável e demais dependências estão em uma imagem de contâiner. Dependendo da abordagem escolhida, o desempenho, o custo e o tempo de inicialização podem variar. Levando em consideração essas métricas, este trabalho visa compará-las entre as duas abordagens de implantação de funções *serverless* e tem como objetivo descobrir se uma das abordagens apresenta ser mais adequada do que outra. Experimentos conduzidos visando tal comparação demonstram que a criação de funções utilizando arquivo compactado ZIP apresentam vantagens, principalmente no tempo de inicialização da função quando está em modo de partida fria.**

***Index Terms*—Função Serverless, AWS Lambda, Contêiner, Avaliação, Desempenho, Custo, Tempo Inicialização**

I. Introdução

Serverless é um modelo de computação em nuvem na qual os provedores ofertam serviços para provisionar dinamicamente servidores configurados para que seus clientes executem suas aplicações, assumindo a responsabilidade de provisionamento, escalabilidade e segurança das aplicações [**?**]. Isso traz maior praticidade para que desenvolvedores e companhias implementem suas aplicações sem a preocupação em contratação e manutenção de infraestutura necessária para sua execução, mesmo que em nuvem. Cada aplicação implantada nesse modelo, é chamada de função *serverless*, que deve executar independentemente da infraestrutura.

Ao comparar o modelo *serverless* com aplicações monolíticas, as aplicações *serverless* são menores, não precisam de configurações de servidor e são nativamente escaláveis conforme a utilização dos recursos alocados atingem o limite. Já aplicações monolíticas compõem todo o código de uma aplicação, e por isso tendem a ter uma grande base de código, incluindo configurações de servidores e banco de dados. A escalabilidade, além de não ser padrão, pode ser um desafio devido ao maior tamanho da aplicação e necessidade de recursos alocados, visto que toda a aplicação precisa ter mais recursos adicionados independente se o gargalo está em apenas uma pequena parte da aplicação.

Apesar da maior facilidade de desenvolvimento, no sentido da abstração da infraestrutura necessária para a execução e atendimento da demanda elástica da aplicação, a implantação da aplicação no ambiente *serverless* demanda cuidados, pois configurações do ambiente são feitas pelo provedor e os desenvolvedores não tem acesso para alterá-las. Tais cuidados estão relacionados a forma em que a aplicação é instanciada e desativada de acordo com o seu uso e ociosidade. A medida que a aplicação fica ociosa, o provedor desaloca os recursos computacionais da função, e a mesma fica inoperante. Quando uma nova execução é requisitada à função, e a mesma se encontra nesse estado, o serviço a instancia novamente com os recursos alocados; porém é perceptível um tempo de resposta maior na primeira requisição.

No provedor de nuvem AWS, particularmente, há duas maneiras de se implantar uma nova função, que são: a maneira mais tradicional via arquivo compactado, ou através de uma imagem de contêiner, opção introduzida pela empresa em 2020 [**?**].

Na implantação via arquivo compactado, o desenvolvedor precisa compactar em formato ZIP a pasta do projeto, incluindo o código fonte e os arquivos executáveis da aplicação, e deve realizar o *upload* direto para o serviço *serverless*. Com o uso de imagem de contêiner, é necessário um arquivo Dockerfile com as dependências e configurações para realizar o *build* da imagem e executar o projeto, e a partir da imagem gerada no ambiente local do desenvolvedor, esta deve ser publicada no repositório de imagens do provedor. Para criar então a função *serverless*, deve ser selecionada a respectiva imagem disponível no repositório. Independente do modelo de implantação escolhido, o desenvolvedor precisa implementar no código da aplicação uma interface fornecida pela AWS para receber as requisições e parâmetros de entrada. Dessa forma, a função *serverless* sabe qual método do código invocar ao receber requisições.

Sendo assim, este trabalho aborda as duas possibilidades de criar/implantar funções *serverless* na AWS e propõe uma comparação de desempenho, custo e tempo de inicialização entre as abordagens. Por fim, busca-se concluir se há um modelo de implantação mais vantajoso do que outro.

Para chegar nos resultados de comparação, testes foram executados nas funções *serverless* via chamada de API. As métricas analisadas foram de consumo máximo de memória, o custo para implantar as funções e o tempo de inicialização da aplicação. Durante a execução de testes, não houve cobrança nos serviços utilizados, portanto, a comparação de custo se deu com base nos valores informados pela Amazon.

Esse artigo é estruturado da seguinte maneira: A Seção II apresenta o referencial teórico para esclarecer termos relacionados a nuvem e ao provedor AWS, e apresentar características utilizadas na avaliação. A Seção III lista trabalhos relacionados pertinentes ao tema, bem como suas características e como se diferem desse artigo. Na Seção IV, é apresentado a proposta da avaliação, bem como os experimentos realizados e as métricas coletadas. Por fim, a Seção V conclui o artigo.

II. Referencial Téorico

Com o avanço da computação em nuvem, cada vez mais serviços estão sendo migrados ou implementados nesse modelo. Os provedores de nuvem, por sua vez, estão dedicados a criarem e disponibilizarem recursos que melhoram a qualidade e praticidade de seus serviços a fim de atender um maior conjunto de clientes. Um desses serviços é chamado *serverless*, que executa uma aplicação sem que o cliente necessite configurar o servidor e demais infraestruturas. A implantação de uma aplicação em *serverless* é chamada de função *serverless*. Para que a aplicação seja executada como um função, o desenvolvedor precisa adicionar em seu código as dependências disponibilizadas pelo provedor. Em específico, a implementação necessária para a AWS é abordada na Seção II-D.

A. Computação em Nuvem

A computação em nuvem é um modelo com vastos recursos de computação compartilhados, como rede, servidores, armazenamento, aplicações e serviços que podem ser provisionados rapidamente e liberados com baixa configuração e complexidade [**?**]. Os provedores de nuvem são responsáveis por cuidar, controlar e ofertar serviços em nuvem para usuários finais ou entidades. Conforme os clientes utilizam os serviços ofertados, pagam sob demanda de acordo com a utilização dos mesmos ou conforme acordo firmado com o provedor. Com a computação em nuvem, *data centers* e infraestruturas de empresas começaram a ser migrados para nuvem devido a agilidade e praticidade dos serviços ofertados pelos provedores. Como consequência, as aplicações também precisaram ser migradas, o que leva a novas formas de aproveitamento dos recursos computacionais disponíveis, tendo em vista a otimização dos mesmos e do custo, dado o modelo de precificação baseado no pagamento do que é usado sob demanda.

B. Função serverless

Funções *serverless* ou modelo Function as a Service (FaaS) providenciam a abstração dos servidores e demais infraestrutura. Os clientes do modelo, ou seja, os desenvolvedores de aplicações, precisam apenas implementar o código da aplicação, isto é, a função. Ao ter o código implementado, é responsabilidade do provedor instanciar o servidor físico ou virtual, configurar o ambiente e disponibilizar uma Application Program Interface (API) para acesso da aplicação por parte do cliente da mesma.

Uma das principais vantagens desse modelo é que o provedor é responsável pela escalabilidade da aplicação. A escalabilidade nesse caso contempla a instanciação da função para sua utilização pelos clientes da aplicação, de forma a atender a demanda de requisições, bem como o gerenciamento dessas instâncias quando não estão necessariamente atendendo às requisições dos clientes finais. É possível utilizar estratégias de otimização com vistas a reutilização dos recursos para o atendimento da escalabilidade. Uma das estratégias utilizadas para gerenciar o dimensionamento dos recursos é chamada de *cold start*, ou partida lenta. Conforme a função deixa de ser utilizada, os recursos de *hardware* alocados passam a ficar ociosos, e portanto, os provedores desalocam parte desses recursos para obter uma maior economia. Quando a aplicação é novamente invocada, pode necessitar que os recursos sejam reativados e uma nova instância seja criada. Esse processo é chamado de partida lenta ou fria [**?**]. Após ter os recursos reativados, essa instância permanecerá ativa enquanto é utilizada, e por um período de tempo após a sua utilização, processo que é chamado de partida quente, pois possui a instância e os recursos operantes [**?**] para o atendimento de nova requisição, sem que se instancie uma nova instância da função. A função, após ficar alguns minutos ociosa, sem nenhuma execução, tem seus recursos desativados pelo provedor. Esse ciclo continua alternando entre partida fria e partida quente conforme a utilização pelos usuários finais.

Otimizações para diminuir o tempo da partida lenta são estudados para evitar grandes atrasos ao cliente final apesar da dificuldade de alcançar soluções ideais, dado que as configurações acerca do gerenciamento dos recursos e das instanciações são mantidas pelos provedores de nuvem [**?**], [**?**], [**?**], [**?**], e não são disponíveis para experimentações e otimizações por parte dos desenvolvedores da aplicação.

C. Contêineres

Contêineres fornecem às aplicações um ambiente configurado com todas as suas dependências para ser executado. Eles isolam a aplicação de programas e processos externos que executam no sistema operacional hospedeiro. Portanto, pode ser definido como um mecanismo de empacotamento de aplicações [**?**].

A utilização de contêineres se tornou amplamente popular pela facilidade de migrar aplicações de um ambiente para outro sem a ocorrência de problemas ao executar em máquinas diferentes. Sem isso, toda vez que as aplicações migravam para um novo ambiente, havia uma grande chance de ocorrerem erros, pois dois ambientes nunca são totalmente idênticos em questão de *software* e *hardware*, e portanto, podem não ter as configurações e dependências necessárias para a execução da aplicação [**?**]. Sem o uso de contêineres ou alguma técnica

de virtualização, ou seja, em *bare metal*, para cada vez que a aplicação executasse em uma máquina diferente, seria necessário antes disso, instalar suas dependências, programas de terceiro, configurar o ambiente, entre outros.

Para que uma aplicação seja portável entre máquinas com contêiner, é preciso de uma imagem da aplicação com tudo o que é necessário para executá-la. Com a imagem criada, o contêiner pode ser replicado para outras máquinas de diferentes *hardwares* e sistemas operacionais (SOs), mantendo o mesmo funcionamento da aplicação. Isso é possível porque os contêineres são uma forma de virtualização leve, que pode incluir seu próprio SO. [**?**]. Sendo assim, pode-se ter uma imagem de contêiner para cada função *serverless* que se deseja criar e especificar qual imagem utilizar ao criar uma nova função no AWS Lambda.

D. Serviços AWS

Um dos maiores provedores de nuvem da atualidade é a AWS (Amazon Web Service), que oferta centenas de serviços disponíveis na Internet. Dentre os serviços ofertados, este trabalho utiliza três deles: AWS Lambda, API Gateway e AWS ECR.

O serviço de *serverless* da Amazon é chamado de AWS Lambda, e nele, consegue-se criar funções para executar aplicações sem precisar provisionar e gerenciar servidores. O serviço Lambda gerencia toda a configuração computacional, que oferece equilíbrio de memória, CPU, rede e outros recursos necessários para executar o código da aplicação (função). As funções são instanciadas sob demanda conforme requisições feitas por usuários finais, o que faz o serviço alocar parte da máquina virtual para a função. Conforme a função passa a ficar ociosa por alguns minutos, os recursos computacionais alocados são desativados. Isso possibilita que os clientes paguem apenas pelo tempo em que a aplicação está sendo utilizada, com os recursos alocados para a função *serverless* [**?**].

Para implantar uma aplicação em função *serverless*, é necessário que o desenvolvedor importe em seu projeto as bibliotecas da AWS Lambda especificas para cada linguagem de programação e implemente a interface do serviço *serverless* disponibilizada pela biblioteca específica. Ao criar a função, na etapa de configuração, é necessário informar o caminho do método que implementa a interface, pois esse método é usado como ponto de entrada para a função invocar a aplicação e passar os parâmetros de entrada.

Particularmente, na AWS, há duas maneiras de implantar uma função *serverless*, que são: via arquivo compactado do código fonte ou com uma imagem de contêiner. Para a abordagem de arquivo compactado, é necessário compilar os arquivos fontes, compactar a pasta do projeto em formato ZIP e fazer o *upload* direto no serviço AWS Lambda. Com o uso de uma imagem de contêiner, é necessário que o desenvolvedor tenha a imagem publicada na AWS ECR (Elastic Container Registry), que é o repositório de imagens de contêineres, e selecionar a respectiva imagem quando da criação da função.

Além das características citadas acerca do servico AWS Lambda, e seus coadjuvantes relacionados ao modelo *serverless*, a arquitetura de *hardware* onde é executada a função também pode ser escolhida junto ao provedor AWS. Assim, ao criar uma nova função, esta pode ser criada sob arquitetura arm64 ou x86_64. Apesar da arquitetura x86_64 ser a padrão, a arquitetura arm64 se destaca por possui menor custo de execução e atingir melhores resultados de desempenho [**?**].

AWS ECR é um serviço de repositório para armazenar imagens de contêineres. O desenvolvedor deve criar a imagem em sua máquina local a partir de um arquivo Dockerfile e publicá-la no repositório [**?**]. Ao ter a imagem disponível no repositório, esta pode ser usada em outros serviços do provedor, como por exemplo para criar funções *serverless*.

Amazon API Gateway é um serviço para criar, publicar e gerenciar APIs, podendo ser REST, HTTP ou WebSocket. É usado como um ponto de entrada para os serviços AWS, incluindo o AWS Lambda, e as APIs podem ser criadas para acessarem os serviços criados ou dados armazenados. Esse serviço lida com as tarefas de aceitar e processar as requisições, com capacidade de processar centenas de milhares de requisições concorrentes [**?**].

III. Trabalhos Relacionados

Modelos FaaS não são tão recentes, e apesar de trazerem praticidade para implementação de aplicações, há ainda espaço para estudos analisarem pontos de melhoria.

O trabalho [**?**] aborda estratégias para diminuir o *cold start* e compara o impacto no tempo ao instanciar uma função através de arquivo compactado e via imagem de contêiner. Apesar de propor soluções para minimizar o tempo de inicialização, o trabalho citado não aborda custo.

Em [**?**], são investigados os fatores que afetam o desempenho de funções *serverless* e como também são examinados os resultados em opções de contêineres, diferentes linguagens de programação e alternativas de compilações. Porém, os autores do trabalho citado não levam em consideração o custo para executar a função e tempo de inicialização do *cold start*.

Na publicação [**?**], os autores compararam os custos de executar aplicações em monolito, microserviço e função Lambda na AWS. Das três arquiteturas, AWS Lambda obteve o menor custo, reduzindo os custos de infraestrutura em 70%.

A Tabela I compara características de alguns trabalhos relacionados e pontua como esse artigo se difere dos demais. As colunas de arquivo compactado e imagem de contêiner são referentes ao modo de implantação da aplicação. As demais colunas apresentam desempenho, custo e tempo de inicialização como métricas referentes às funções.

IV. Experimentos

Este trabalho propõe avaliar dois métodos de implantação de uma aplicação no serviço *serverless* da Amazon, AWS Lambda. A escolha da linguagem de programação Java para o experimento foi baseada em sua popularidade no meio Web e facilidade de implementar as interfaces requeridas para o serviço na AWS.

TABELA I
COMPARAÇÃO DE TRABALHOS RELACIONADOS

Artigo	Arquivo compactado	Imagem de contêiner	Desempenho	Custo	Tempo de inicialização
Dantas [**?**]	Sim	Sim	Não	Sim	Sim
Elsakhawy [**?**]	Não	Sim	Sim	Sim	Não
Villamizar [**?**]	Não	Não	Não	Sim	Não
Trabalho atual	Sim	Sim	Sim	Sim	Sim

Devido ao aumento do uso de funções *serverless* nos últimos anos, estudos de otimizações e melhorias vem sendo propostos. Sendo assim, visto que que há dois modelos de implantação de funções *serverless* na AWS, este trabalho busca descobrir se uma das abordagens é mais vantajosa do que a outra. Para chegar em tal conclusão, métricas pertinentes ao serviço são analisadas. As métricas levadas em consideração foram de desempenho com base no consumo de memória, de custo e de tempo de inicialização em partida lenta.

Para realizar a comparação das métricas, foi desenvolvido uma aplicação, implantada como função *serverless* na AWS Lambda via arquivo ZIP e imagem de contêiner, e executado os testes via interface disponibilizada pelo próprio provedor AWS.

Primeiro é descrito o ambiente e configurações para simular os testes, e depois são apresentados os dados de comparação entre os modelos de implantação.

A. Ambiente

Para realizar tal avaliação, foi realizada a implementação de uma aplicação simples utilizando a linguagem de programção Java, versão 11. A aplicação possui uma interface de entrada de dados onde deve ser informado uma região e retorna a data e o horário da região informada.

Para implementar a aplicação no serviço da AWS, foi criado uma função utilizando arquivo ZIP compactado e outra função via imagem de contêiner. Para criar a função com o uso da imagem, uma imagem da aplicação foi criada a partir de um Dockerfile configurado para realizar o *build* e publicada no repositória do AWS ECR. Refente a arquitetura escolhida para as funções, foi escolhido a arquitetura arm64 devido ao menor preço.

Para acessar e chamar as funções criadas, foram criadas APIs públicas REST na Amazon API Gateway e configuradas para invocar as respectivas funções. A API de acesso omite qual foi o modelo de implantação utilizado para função, e portanto, os usuários finais não conseguem saber dos detalhes técnicos da implantação.

O ambiente configurado é representado no diagrama da Figura 1. Inicialmente, um usuário faz uma chamada de requisição REST. A requisição é recebida pela API Gateway, que redireciona para a respectiva função *serverless* criada. A função pode executar a aplicação implantada via arquivo compactado ZIP ou com uma imagem de contêiner armazenada no Amazon ECR. A função *serverless* deve então retornar uma mensagem de resposta para o serviço de API Gateway, o qual direciona a mensagem para o usuário.

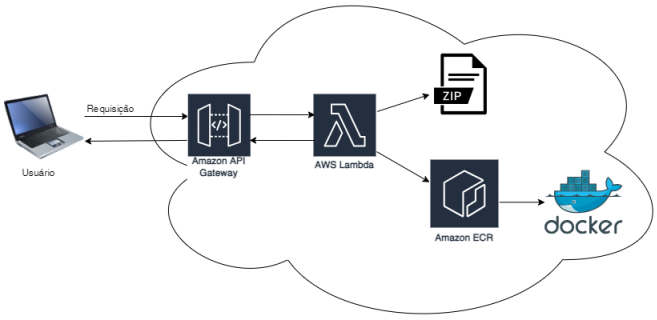

Fig. 1. Diagrama do ambiente de testes

B. Custo

Para executar todos os testes na AWS, não houve incidência de cobrança do provedor por ter sido utilizado apenas serviços e configurações no nível *Free Tier*. Esse nível possibilita que clientes utilizem serviços de forma gratuita, desde que atendam as restrições existentes. Portanto, para comparar o custo entre as duas abordagens de implantação, são usados os valores base de precificação do provedor de nuvem AWS.

O custo para implantar e manter a função *serverless* ativa é o mesmo, independente da abordagem escolhida. Outros fatores, como configuração de *hardware* e localização do servidor podem impactar no valor final, mas estão fora do escopo deste trabalho. Ao fazer o *upload* do projeto compactado, não há nenhuma cobrança para armazenar os arquivos. Porém, para disponibilizar uma imagem de contêiner na AWS ECR, há um custo, que é proporcional ao tamanho da imagem. A precificação na AWS ECR também varia dependendo da região [**?**].

C. Desempenho

Algumas métricas podem estar relacionadas ao desempenho de uma aplicação. Neste trabalho, foi analisado o consumo de memória RAM máximo no ambiente ao executar funções *serverless* que estavam em modo de partida lenta. Essa métrica é coletada no console de saída da AWS Lambda ao fazer uma requisição.

Conforme Figura 2, nota-se que o uso de memória máximo com a abordagem de imagem de contêiner se mantém constante, enquanto via arquivo ZIP, há uma variação de uma execução para outra. A Figura 3 apresenta a média de ambas abordagens, corroborando os resultados da Figura 2, demonstrando que a implantação da função feita a partir da imagem de contêiner obteve melhores resultados em relação

ao consumo de memória, ou seja, consome menos memória para executar a função.

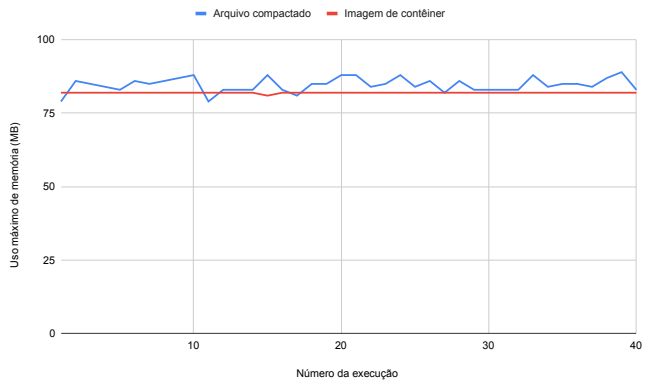

Fig. 2. Gráfico de uso de memória máximo em funções *serverless*

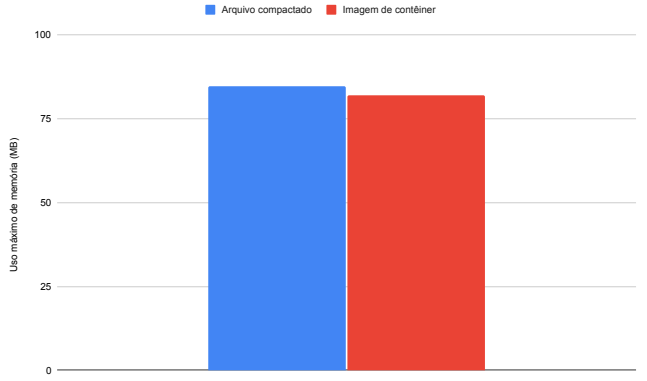

Fig. 3. Gráfico da média de uso de memória máximo em funções *serverless*

D. Tempo de inicialização da partida fria

As Figuras 4 e 5 mostram que o método de implantação via arquivo ZIP possui o menor tempo de inicialização. Particularmente, observa-se na Figura 4 o tempo de inicialização da função implantada via arquivo compactado e via contêiner, para várias inicializações. Cada inicialização ocorreu respeitando tempo suficiente para que os recursos fosses desalocados, obrigando uma nova instanciação da função. A Figura 5 apresenta a média dos tempos de inicialização apresentados na Figura 4.

O tempo de inicialização também impacta no tempo de resposta quando a aplicação está em modo de partida lenta, portanto, funções executando com arquivo compactado obtém melhores resultados.

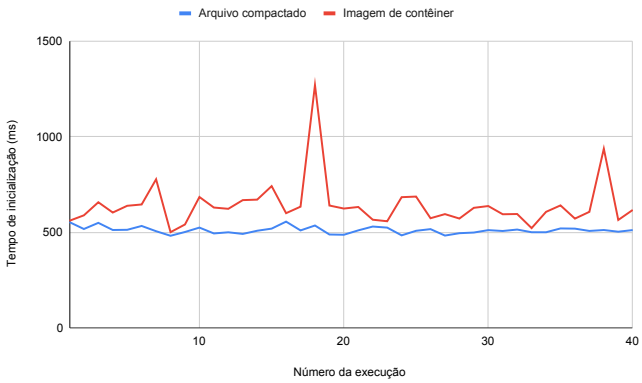

Fig. 4. Gráfico do tempo de inicialização em funções *serverless*

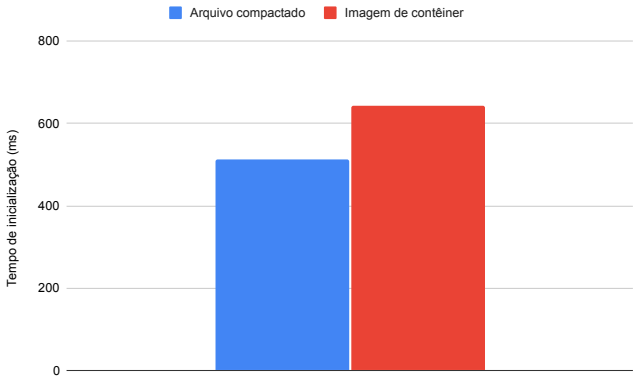

Fig. 5. Gráfico da média do tempo de inicialização em funções *serverless*

V. Conclusão

Esse trabalho avaliou os modelos de implantação de funções *serverless* disponíveis na AWS Lambda. O serviço oferece duas possibilidades: via arquivo compactado no formato ZIP e via imagem de contêiner. Com os arquivos compactados ZIP, o modelo de implantação é mais fácil, pois é necessário apenas fazer-se o *upload* do projeto para o serviço AWS Lambda, enquanto na outra abordagem é necessário configurar um arquivo Dockerfile para executar a aplicação, gerar uma imagem de contêiner e publicá-la na AWS ECR.

Dependendo do modelo escolhido, o custo, o desempenho e o tempo de inicialização da partida lenta podem ser diferentes. Ao criar as funções *serverless* na AWS Lambda nos dois modelos de implantação, ambos não tiverem incidências de custos durante os testes realizados. Porém, quando a abordagem escolhida é com o uso de uma imagem de contêiner, é necessário utilizar outro serviço para armazenar a imagem, nesse caso o AWS ECR, e este pode vir a gerar custos conforme o tamanho da imagem. Ao analisar o uso máximo de memória RAM quando a aplicação está em modo de partida lenta, ambos apresentaram consumo similar, com a vantagem que via imagem de contêiner o uso da memória se manteve constante. A maior diferença se deu no tempo de inicialização

da aplicação em partida lenta, no qual a implantação via arquivo ZIP mostrou ser mais eficiente para alocar os recursos e tornar a função ativa novamente.

Portanto, com base nos dados e resultados obtidos e em sua análise, pode-se concluir que a implantação via arquivo ZIP apresenta vantagens. As principais vantagens comparadas ao modelo de implantação com imagem de contêiner são o custo para implantação, uma vez que não é necessário pagamento para armazenar os arquivos compactados e o tempo de inicialização quando em partida lenta é menor.

Como trabalho futuros, pode-se estender a comparação para as outras linguagens de programação suportadas pelo provedor de nuvem AWS. Outro aspecto a ser comparado é a arquitetura na qual a função *serverless* é executada, x86_64 ou arm64. Neste trabalho, foi utilizado apenas a arquitetura arm64, havendo espaço para tratar da arquitetura x86_64. O escopo da aplicação também pode extrapolado para aplicações maiores ou mais complexas que demandem maior processamento computacional, e que deve impactar no consumo máximo de memória e tempo de inicialização.