v7.n12.2026

REVISTA CIENTÍFICA - RECIMA21 ISSN 2675-6218

FUNÇÕES SERVERLESS, MÁQUINAS VIRTUAIS OU CONTÊINERES PARA FERRAMENTAS INTERNAS DE USO INTERMITENTE: DESEMPENHO, CUSTO E RESPONSABILIDADE OPERACIONAL EM UM ESTUDO DE CASO NA AWS

SERVERLESS FUNCTIONS, VIRTUAL MACHINES, OR CONTAINERS FOR INTERMITTENTLY USED INTERNAL TOOLS: PERFORMANCE, COST, AND OPERATIONAL RESPONSIBILITY IN AN AWS CASE STUDY

FUNCIONES SERVERLESS, MÁQUINAS VIRTUALES O CONTENEDORES PARA HERRAMIENTAS INTERNAS DE USO INTERMITENTE: RENDIMIENTO, COSTO Y RESPONSABILIDAD OPERATIVA EN UN ESTUDIO DE CASO EN AWS

Kaiky Henrique Vieira Barbosa de Moura[1], Ana Paula Abrantes de Castro e Shiguemori[2]

https://doi.org/10.47820/recima21.v7i12.9923

PUBLICADO: 12/2026

RESUMO

A escolha da arquitetura de hospedagem para ferramentas operacionais internas de uso intermitente envolve compromissos entre desempenho, custo e responsabilidade operacional. Este estudo avalia esses compromissos entre *Function as a Service* (FaaS), *Infrastructure as a Service* (IaaS) e *Container as a Service* (CaaS), tendo como caso uma ferramenta de monitoramento e recuperação de tarefas do AWS Database Migration Service (DMS). AWS Lambda e Amazon EC2 foram comparados na execução de um núcleo com uma leitura em lote e 42 atualizações no Amazon DynamoDB; o Amazon EKS foi analisado documentalmente. Três rodadas exploratórias mostraram que corrigir a reinstanciação do SDK reduziu em 83,4% o tempo médio da Lambda. Uma rodada confirmatória, com artefato idêntico, ensaios intercalados e aleatorizados (N = 60 por condição) e três configurações de memória, mostrou que a diferença depende da memória da Lambda: com 128 MB, o tempo médio foi 5,81 vezes o da EC2 t3.micro (1,4956 s *versus* 0,2572 s); com 512 MB, 1,35 vez; com 1.769 MB, não houve diferença estatisticamente significativa. A medição abrange apenas o núcleo instrumentado, sem inicialização, API Gateway nem rede do cliente. No modelo de custos, a Lambda apresentou baixo custo incremental no uso intermitente, enquanto EC2 e EKS mantiveram custos de capacidade provisionada. A análise da recuperação indicou potencial redução das ações humanas, sem constituir medição experimental do *Mean Time to Recovery* (MTTR). A adequação das alternativas depende do perfil de uso, da configuração dos recursos e dos requisitos operacionais, não havendo paradigma universalmente superior.

PALAVRAS-CHAVE: Computação em nuvem. Arquitetura *serverless*. Function as a Service. Infrastructure as a Service. Container as a Service.

ABSTRACT

Choosing a hosting architecture for intermittently used internal operational tools involves trade-offs among performance, cost, and operational responsibility. This study evaluates these trade-offs among Function as a Service (FaaS), Infrastructure as a Service (IaaS), and Container as a Service (CaaS), using as a case a tool for monitoring and recovering AWS Database Migration Service (DMS) tasks. AWS Lambda and Amazon EC2 were compared when executing a core routine comprising one batch read and 42 updates in Amazon DynamoDB; Amazon EKS was assessed through documentary analysis. Three exploratory rounds showed that correcting the SDK reinstantiation reduced Lambda's mean time by 83.4%.

[1]Graduando em Tecnologia em Análise e Desenvolvimento de Sistemas pelo Instituto Federal de Educação, Ciência e Tecnologia de São Paulo (IFSP), campus Jacareí, e engenheiro de software.
[2]Doutora em Computação Aplicada pelo Instituto Nacional de Pesquisas Espaciais (INPE) e professora da área de Computação no Instituto Federal de Educação, Ciência e Tecnologia de São Paulo (IFSP).

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

A confirmatory round, using an identical artifact, randomized interleaved trials (N = 60 per condition), and three memory configurations, showed that the difference depends on Lambda's memory: with 128 MB, its mean time was 5.81 times that of an EC2 t3.micro instance (1.4956 s vs. 0.2572 s); with 512 MB, 1.35 times; with 1,769 MB, no statistically significant difference was found. The measurement covers only the instrumented core, excluding initialization, API Gateway, and client-side networking. In the cost model, Lambda showed low incremental cost under intermittent use, whereas EC2 and EKS retained provisioned-capacity costs. The recovery analysis indicated a potential reduction in human actions without constituting an experimental measurement of Mean Time to Recovery (MTTR). The results indicate that the suitability of the alternatives depends on the usage profile, resource configuration, and operational requirements, with no universally superior paradigm.

KEYWORDS: *Cloud computing. Serverless architecture. Function as a Service. Infrastructure as a Service. Container as a Service.*

RESUMEN
Elegir la arquitectura de alojamiento para herramientas operativas internas de uso intermitente implica compromisos entre rendimiento, costo y responsabilidad operativa. Este estudio evalúa dichos compromisos entre Function as a Service (FaaS), Infrastructure as a Service (IaaS) y Container as a Service (CaaS), tomando como caso una herramienta de monitorización y recuperación de tareas de AWS Database Migration Service (DMS). AWS Lambda y Amazon EC2 se compararon en la ejecución de un núcleo con una lectura por lotes y 42 actualizaciones en Amazon DynamoDB; Amazon EKS se analizó documentalmente. Tres rondas exploratorias mostraron que corregir la reinstanciación del SDK redujo un 83,4% el tiempo medio de Lambda. Una ronda confirmatoria, con artefacto idéntico, ensayos intercalados y aleatorizados (N = 60 por condición) y tres configuraciones de memoria, mostró que la diferencia depende de la memoria: con 128 MB, el tiempo medio fue 5,81 veces el de EC2 t3.micro (1,4956 s frente a 0,2572 s); con 512 MB, 1,35 veces; con 1.769 MB, no hubo diferencia estadísticamente significativa. La medición abarca solo el núcleo instrumentado, sin inicialización, API Gateway ni red del cliente. En el modelo de costos, Lambda presentó bajo costo incremental con uso intermitente, mientras que EC2 y EKS mantuvieron costos de capacidad aprovisionada. El análisis de la recuperación indicó una posible reducción de acciones humanas, sin medir experimentalmente el Mean Time to Recovery (MTTR). La adecuación de las alternativas depende del perfil de uso, la configuración de los recursos y los requisitos operativos, sin paradigma universalmente superior.

PALABRAS CLAVE: *Computación en la nube. Arquitectura serverless. Function as a Service. Infrastructure as a Service. Container as a Service.*

1. INTRODUÇÃO

A adoção da computação em nuvem segue em expansão: a Gartner projetou gastos de usuários finais com serviços de nuvem pública de US$ 723,4 bilhões em 2025, ante US$ 595,7 bilhões em 2024 (Gartner, 2024). Esse crescimento multiplicou os níveis de abstração disponíveis, que variam quanto ao controle operacional delegado ao provedor e afetam desempenho, custo e responsabilidade operacional. Ao mesmo tempo, ferramentas internas de monitoramento, automação e governança consomem parte relevante do trabalho das equipes

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

técnicas: em levantamento de um fornecedor do setor com 2.285 desenvolvedores e líderes técnicos, majoritariamente seus clientes, os respondentes declararam dedicar cerca de 33% do tempo a construí-las (Retool, 2023). Para ferramentas desse tipo com baixo volume de acesso e longos períodos de ociosidade, como o caso analisado, a hospedagem influencia o tempo de resposta, o custo e o trabalho necessário para manter o serviço disponível.

Function as a Service (FaaS), *Infrastructure as a Service* (IaaS) e *Container as a Service* (CaaS) distribuem essas responsabilidades de maneiras distintas. Esses modelos podem ser vistos como um contínuo de abstração, que vai da computação baseada em servidores virtuais, passando pela orquestração de contêineres, até a computação *serverless*, ponto de maior delegação operacional ao provedor (Jonas *et al.*, 2019). Em FaaS, o provedor administra a infraestrutura de execução e cobra por requisições e duração; em IaaS, a equipe controla a máquina virtual e assume responsabilidades sobre o sistema operacional e a capacidade provisionada; em CaaS, a aplicação é empacotada em contêineres e executada sob uma camada de orquestração. A alternativa de menor latência não é necessariamente a de menor custo ou de menor esforço operacional, o que torna a decisão dependente do perfil da carga.

A literatura sobre a escolha entre esses modelos converge em três pontos. No plano econômico, a cobrança por uso favorece arquiteturas *serverless* em cargas irregulares ou de baixo volume: Villamizar *et al.* (2017) estimaram redução de 70% ou mais no custo de infraestrutura de uma aplicação *web* executada em AWS Lambda; Adzic e Chatley (2017) relataram reduções entre 66% e 95% em dois casos industriais; e Allen *et al.* (2024), ao compararem microsserviços em contêineres sobre Amazon EC2 com funções em AWS Lambda, concluíram que as funções são mais baratas quando o número mensal de requisições é limitado. No plano do desempenho, os resultados dependem da plataforma e de sua configuração: no AWS Lambda, a capacidade de CPU é alocada proporcionalmente à memória configurada (Wang *et al.*, 2018), e, em ambientes orquestrados por Kubernetes, plataformas *serverless* auto-hospedadas apresentam desempenhos distintos entre si (Decker; Kasprzak; Kunkel, 2022). No plano metodológico, a revisão de 112 fontes conduzida por Scheuner e Leitner (2020), 51 acadêmicas e 61 da literatura cinzenta, identificou predomínio de *microbenchmarks* de CPU e de sobrecarga da plataforma, divergência entre as configurações testadas por fontes acadêmicas e industriais e falhas que ameaçam a reprodutibilidade dos experimentos.

Esses estudos sustentam uma conclusão já consolidada: não há paradigma universalmente superior, e o resultado depende da carga e da configuração. O que eles não determinam é a magnitude desses efeitos em um perfil pouco representado nas avaliações, o de ferramentas operacionais internas com baixo volume de invocações, longos períodos de

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

ociosidade e um núcleo composto por chamadas a serviços gerenciados. Para esse perfil, permanecem em aberto três questões: quanto a memória configurada altera a diferença de tempo entre FaaS e IaaS; a partir de que volume de uso os custos das alternativas se igualam; e como a existência de infraestrutura de orquestração compartilhada modifica a decisão sobre CaaS. Além disso, a responsabilidade operacional costuma ser discutida em termos conceituais, como em Jonas *et al.* (2019), e não em conjunto com medições de desempenho e custo.

O estudo utiliza como caso o *DMS-Task-Monitor*, uma aplicação que centraliza a consulta do estado de tarefas do *AWS Database Migration Service*, apoia testes de conectividade e permite que um usuário autorizado acione um fluxo de recuperação. Como a ferramenta também apoia a recuperação de falhas, considera-se como referência conceitual o tempo médio para reparo ou recuperação (*Mean Time to Repair* ou *Mean Time to Recovery*, MTTR), métrica relacionada ao tempo necessário para restaurar o serviço (Forsgren; Humble; Kim, 2018; IBM, [s. d.]). Entretanto, neste estudo, a análise da recuperação distingue esforço humano de tempo técnico e não constitui medição experimental de MTTR.

Diante dessa lacuna, este artigo tem por objetivo avaliar os compromissos entre desempenho, custo e responsabilidade operacional associados à adoção de *Function as a Service* (FaaS), *Infrastructure as a Service* (IaaS) e *Container as a Service* (CaaS) na hospedagem de uma ferramenta de uso intermitente. A comparação experimental de desempenho é restrita ao AWS Lambda (FaaS) e ao Amazon EC2 (IaaS), enquanto o Amazon EKS (CaaS) é analisado documentalmente quanto a custos e responsabilidades operacionais.

Os objetivos específicos correspondem a três perguntas de pesquisa. RQ1: como se compara o tempo interno do núcleo nas implementações FaaS e IaaS avaliadas e como a memória configurada na Lambda altera essa comparação? RQ2: como os custos mensais e as responsabilidades operacionais variam nos cenários FaaS, IaaS e CaaS definidos? RQ3: como a ferramenta modifica as ações humanas do processo de recuperação e quais limitações impedem que essa análise seja interpretada como medição experimental de MTTR?

A contribuição deste trabalho não é contestar a conclusão geral da literatura, mas quantificá-la e qualificá-la para o perfil descrito. O estudo mede, com artefato idêntico e ensaios intercalados e aleatorizados, como a diferença entre FaaS e IaaS varia com a memória configurada; mostra o peso de uma decisão de implementação sobre o resultado de um *benchmark* em nuvem; calcula os pontos de equilíbrio de custo e o efeito de um *cluster* dedicado ou compartilhado; e relaciona essas medidas às responsabilidades operacionais de cada alternativa. Com isso, busca subsidiar a decisão arquitetural com valores e condições explícitos, em vez de uma recomendação genérica.

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

2. REFERENCIAL TEÓRICO

2.1. Computação em nuvem

A computação em nuvem permite acesso sob demanda, por meio de rede, a um conjunto compartilhado de recursos computacionais configuráveis, provisionados e liberados com mínimo esforço de gestão (Mell; Grance, 2011). Entre suas cinco características essenciais está o serviço mensurado (measured service), pelo qual o uso dos recursos é monitorado, controlado e reportado, o que fundamenta a cobrança por uso. Os modelos FaaS, IaaS e CaaS diferem no nível de controle oferecido ao usuário e nas responsabilidades operacionais que permanecem sob sua gestão.

2.2. FaaS

No modelo FaaS, o desenvolvedor publica funções executadas sob demanda, enquanto o provedor administra a infraestrutura subjacente de execução, incluindo parte dos mecanismos de provisionamento e escalabilidade. Em levantamento de 89 aplicações *serverless*, Eismann *et al.* (2021) identificaram a redução de custos como a motivação mais frequente (47%), associada ao pagamento por uso em cargas irregulares ou em rajadas, característica de 84% das aplicações analisadas.

No AWS Lambda, a cobrança considera o número de requisições e a duração da execução associada à quantidade de memória configurada (Amazon Web Services, [s. d.]g). A memória definida também determina a quantidade de recursos computacionais, incluindo CPU, disponibilizada à função, o que torna essa configuração um fator relevante para o desempenho (Amazon Web Services, [s. d.]i).

O modelo reduz a administração direta de servidores, mas mantém responsabilidades de configuração, segurança, observabilidade, empacotamento e desenho das integrações. Entre as boas práticas oficiais está inicializar clientes do SDK fora do *handler*, permitindo seu reaproveitamento entre invocações, em vez de recriá-los a cada chamada (Amazon Web Services, [s. d.]h).

Além disso, a primeira invocação em um novo ambiente de execução pode envolver uma etapa adicional de inicialização, conhecida como partida a frio (*cold start*), acrescentando latência à execução (Wang *et al.*, 2018). A partir de agosto de 2025, essa fase passou também a integrar a duração faturada em determinadas condições de execução do AWS Lambda (Gupta; Gebhart, 2025).

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

2.3. IaaS

O modelo de IaaS fornece recursos computacionais virtualizados, como máquinas virtuais, sobre os quais o usuário controla o sistema operacional, o *runtime* e os processos da aplicação. Esse controle favorece configurações específicas e a consolidação de diferentes serviços, mas mantém sob responsabilidade da equipe usuária atividades relacionadas à configuração do sistema operacional, atualizações, segurança, observabilidade e gerenciamento da capacidade provisionada.

No contexto da AWS, o Amazon EC2 representa esse modelo ao permitir o provisionamento de instâncias virtuais com diferentes configurações de processamento, memória, armazenamento e rede. Na modalidade sob demanda (*on-demand*), instâncias Linux são cobradas por segundo, com mínimo de 60 s e sem compromisso de longo prazo (Amazon Web Services, [s. d.]k). Embora as instâncias possam ser iniciadas ou interrompidas conforme a necessidade, aplicações que exigem disponibilidade contínua podem manter capacidade provisionada durante períodos sem utilização efetiva.

2.4. CaaS

O modelo CaaS utiliza contêineres para empacotar aplicações e suas dependências e plataformas de orquestração para gerenciar sua implantação, distribuição, reinicialização e escalabilidade. Contêineres oferecem virtualização mais leve que máquinas virtuais ao compartilhar o núcleo do sistema operacional (Pahl, 2015). Entre as principais tecnologias de orquestração encontra-se o Kubernetes, cuja concepção deriva da experiência do Google com sistemas internos de gerenciamento de cargas de trabalho, como Borg e Omega (Burns *et al.*, 2016).

Neste estudo, o CaaS é representado pelo Amazon Elastic Kubernetes Service (EKS), serviço gerenciado de Kubernetes da AWS. No EKS, a AWS administra o plano de controle do Kubernetes, enquanto ao usuário cabe parte das responsabilidades relacionadas aos *workloads*, políticas, rede, identidade e capacidade computacional utilizada pelas aplicações. Dependendo da configuração, essa capacidade pode ser fornecida por nós do Amazon EC2 ou pelo AWS Fargate (Amazon Web Services, [s. d.]d, [s. d.]l).

O custo da solução pode envolver a tarifa do plano de controle do *cluster* e os recursos computacionais consumidos pelos *workloads*, além de componentes adicionais de rede, armazenamento e observabilidade, conforme a arquitetura adotada (Amazon Web Services, [s. d.]d, [s. d.]f).

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

Em comparação com FaaS e IaaS, CaaS introduz uma camada adicional de orquestração que pode favorecer portabilidade, padronização e gerenciamento de múltiplas cargas, mas também acrescenta componentes e responsabilidades operacionais à arquitetura.

2.5. Proposições de análise

Do referencial decorrem três proposições que orientam a análise dos resultados. P1, alocação de recursos: como a CPU disponível à função acompanha a memória configurada (Amazon Web Services, [s. d.]i; Wang *et al.*, 2018), a diferença de tempo entre FaaS e IaaS deve diminuir com o aumento da memória. P2, modelo de cobrança: como o serviço é mensurado e cobrado por uso em FaaS e por capacidade provisionada em IaaS e CaaS (Mell; Grance, 2011; Adzic; Chatley, 2017), FaaS deve custar menos em uso intermitente, até um volume de equilíbrio. P3, nível de abstração: quanto maior a delegação ao provedor no contínuo de abstração (Jonas *et al.*, 2019), menor a responsabilidade operacional da equipe e menor o seu controle sobre o ambiente. A Seção 4.5 confronta os resultados com essas proposições.

3. METODOLOGIA

3.1. Caracterização da pesquisa

Trata-se de um estudo de caso único, de abordagem predominantemente quantitativa, que combina avaliação experimental de desempenho e análise documental complementar. O DMS-Task-Monitor constitui o caso investigado, enquanto a unidade experimental da avaliação de desempenho corresponde a uma execução do núcleo instrumentado de consulta em lote e atualização de status sob uma condição de execução definida. A variável-resposta primária é o tempo interno do núcleo, expresso em segundos, medido em AWS Lambda (FaaS) e Amazon EC2 (IaaS).

O protocolo teve duas etapas, detalhadas na Seção 3.3: três rodadas exploratórias, com 30 execuções por condição, usadas para identificar fatores de interferência e refinar o procedimento; e uma rodada confirmatória (rodada 4), com 60 execuções por condição. O Amazon EKS não foi implantado; sua análise, documental e econômica, é descrita na Seção 3.6.

O delineamento do estudo considerou as recomendações de Runeson e Höst (2009) para descrição de estudos de caso em Engenharia de *Software* e de Wohlin *et al.* (2012) para a documentação de experimentos. Também foram considerados princípios de avaliação reprodutível de desempenho em ambientes de nuvem, incluindo repetição das medições, descrição do ambiente experimental, controle das condições avaliadas e relato da variabilidade

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

(Papadopoulos *et al.*, 2021), particularmente relevante diante da variabilidade característica de infraestruturas de nuvem (Laaber; Scheuner; Leitner, 2019).

3.2. Estudo de caso

O DMS-Task-Monitor é uma aplicação *web* de página única, com *frontend* desenvolvido em React e *backend* em Python, que utiliza o SDK boto3 para comunicação com serviços da AWS. A infraestrutura da aplicação é definida como código (Infrastructure as Code – IaC) por meio do Terraform.

A ferramenta apoia o suporte e a observabilidade de tarefas do AWS Database Migration Service (DMS), serviço utilizado para migrar bancos de dados e manter processos de replicação (Amazon Web Services, [s. d.]o). Usuários autenticados podem consultar o estado das tarefas, realizar testes de conectividade quando aplicáveis e solicitar uma recuperação controlada.

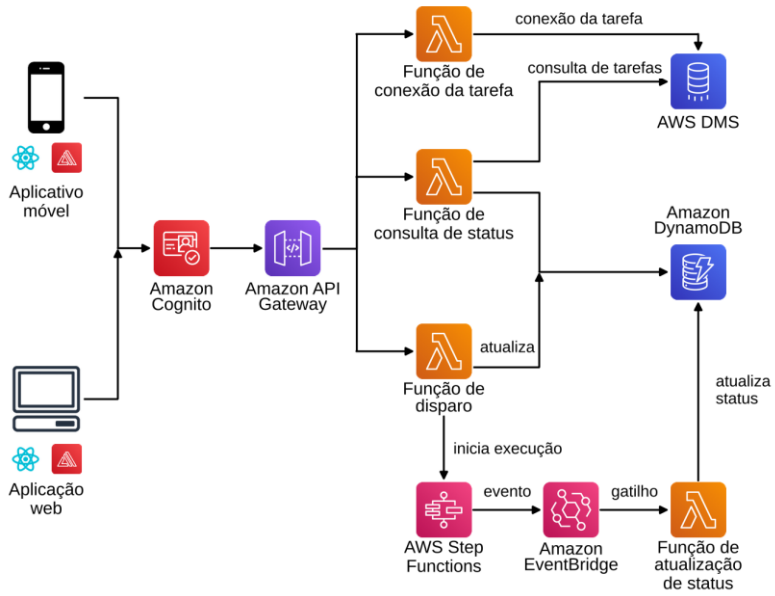
As requisições passam por autenticação e por uma API gerenciada, que aciona funções responsáveis por consultar o DMS, manter o estado da aplicação no Amazon DynamoDB e iniciar uma máquina de estados. Essa máquina coordena os serviços envolvidos no processo de recuperação, registra os resultados das etapas e publica eventos utilizados para atualizar o estado apresentado ao usuário, conforme a Figura 1.

O caso foi selecionado por apresentar três características diretamente relacionadas ao problema investigado: utilização intermitente, integração com múltiplos serviços de nuvem e necessidade de resposta operacional diante de falhas. Essas características permitem analisar, no mesmo contexto, os compromissos entre tempo de execução, custo de capacidade provisionada e responsabilidades operacionais.

Para preservar informações da organização de origem, o caso é apresentado de forma anonimizada. Foram omitidos nomes e identificadores de contas, ARNs, *buckets*, filas, funções, parâmetros, regiões corporativas e outros elementos capazes de revelar a topologia original da solução.

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

Figura 1. Arquitetura lógica anonimizada do DMS-Task-Monitor



Fonte: elaboração própria (2026).

3.3. Ambiente e procedimento experimental

3.3.1. Rodadas exploratórias

Foram comparados AWS Lambda, inicialmente configurado com 128 MB e arquitetura x86_64, e uma instância Amazon EC2 t3.micro, com 2 vCPUs, 1 GiB de memória e Amazon Linux 2. Ambas as condições foram executadas na região us-east-1 e realizaram uma leitura em lote seguida de 42 operações de atualização no DynamoDB. Foram obtidas 30 execuções por condição em cada rodada.

Na rodada 2, a EC2 utilizava Python 3.7.16 e boto3 1.33.13. As versões utilizadas na EC2 na rodada 1 e na Lambda nas rodadas 1 e 2 não haviam sido registradas e, portanto, não são inferidas retrospectivamente. Na terceira rodada, as versões foram padronizadas: Python 3.12.9 na EC2 e Python 3.12.14 na Lambda, com boto3/botocore 1.42.97 em ambas.

As implementações utilizadas nessas rodadas apresentavam assimetrias. Na Lambda, o intervalo medido incluía leitura do evento, montagem da resposta, 84 linhas de log e, mesmo após a correção da rodada 2, a criação de um recurso do SDK a cada invocação, usado na leitura em lote, sobre itens sintéticos contendo seis campos. Na EC2, nas rodadas 2 e 3, o recurso do SDK era criado fora do intervalo cronometrado, os registros continham dois campos e as 30 execuções eram realizadas em um laço dentro do mesmo processo.

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

Além disso, o modo de créditos da instância EC2 não havia sido definido explicitamente no lançamento. Os registros disponíveis indicavam o padrão da conta (*unlimited*) e configuração automática de IPv4 público; entretanto, como as instâncias haviam sido encerradas, essa informação foi tratada como evidência indireta.

3.3.2. Rodada confirmatória

AWS Lambda e Amazon EC2 passaram a executar o mesmo módulo de aplicação, cuja identidade foi verificada por *hash* SHA-256. O recurso do DynamoDB foi criado durante a importação do módulo, as duas plataformas processaram os mesmos 42 itens sintéticos de seis campos e nenhum registro de saída foi realizado dentro do intervalo cronometrado. A medição foi efetuada utilizando time.perf_counter_ns(), relógio monotônico de alta resolução.

O intervalo cronometrado começou imediatamente antes da leitura em lote (BatchGetItem) e terminou após a última atualização (UpdateItem), nas duas plataformas. A variável medida é, portanto, o tempo interno do núcleo, e não a latência ponta a ponta: não inclui a inicialização do ambiente da Lambda, o API Gateway nem a comunicação entre o cliente e a API.

Na AWS Lambda foram avaliadas três configurações de memória: 128 MB, 512 MB e 1.769 MB. A função utilizou Python 3.12, arquitetura x86_64 e boto3/botocore 1.42.97 empacotados com a aplicação. A configuração de 1.769 MB corresponde à alocação aproximada de uma vCPU segundo a documentação da AWS (Amazon Web Services, [s. d.]i).

A condição IaaS utilizou uma instância EC2 t3.micro com Amazon Linux 2023, Python 3.12.15 e boto3/botocore 1.42.97. O modo de créditos foi explicitamente definido como *unlimited*, e a instância utilizou IPv4 público.

O experimento seguiu um delineamento com execuções intercaladas e ordem aleatorizada (Abedi; Brecht, 2017). Em cada bloco, cada uma das quatro condições foi executada uma vez, em ordem determinada aleatoriamente com semente fixa. Antes de cada execução, os 42 registros eram restaurados para seu estado inicial fora do intervalo de medição.

Cada sessão iniciou com três blocos de aquecimento, descartados da análise, seguidos de 30 blocos válidos. Foram realizadas duas sessões em 4 de novembro de 2026 (horário de Brasília, UTC−3), separadas por aproximadamente 6 h 45 min, resultando em 60 medições por condição e 240 medições válidas no total.

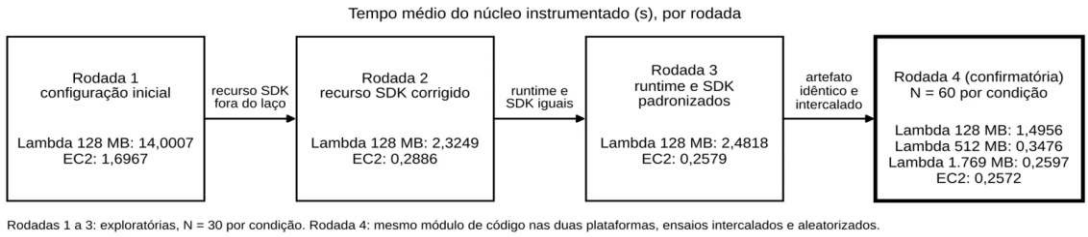
O programa responsável por orquestrar o experimento foi executado na própria instância EC2, localizada na mesma região dos demais recursos, e realizou as invocações da Lambda de forma síncrona.

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

Adicionalmente, foi conduzido um experimento específico para caracterizar partidas a frio da Lambda com 128 MB. Foram forçadas dez partidas a frio, confirmadas pela presença do campo Init Duration no relatório da plataforma. Um primeiro lote foi descartado porque quatro invocações não apresentaram evidência de inicialização de novo ambiente; após a correção do procedimento de espera, um novo lote foi executado. Essas medições foram analisadas separadamente e não integram as 60 observações da rodada confirmatória principal.

O saldo de créditos de CPU da EC2 pôde ser consultado apenas ao final das sessões; não houve cobrança de créditos excedentes durante o experimento. Essa limitação foi considerada na interpretação dos resultados. A Figura 2 resume a evolução do protocolo experimental ao longo das quatro rodadas.

Figura 2. Evolução do benchmark nas quatro rodadas: correção do recurso do SDK, padronização de runtime e SDK e rodada confirmatória com artefato idêntico.



Fonte: elaboração própria (2026).

3.4. Métricas, instrumentos de coleta e análise estatística

A variável primária foi o tempo interno do núcleo, em segundos, medido com time.time() nas rodadas 1 a 3 e com time.perf_counter_ns() na rodada confirmatória; na Lambda, o tempo decorrido foi retornado na resposta da invocação.

Nas execuções em Lambda, a duração registrada pela plataforma, a duração faturada, a memória utilizada e, quando aplicável, as durações de inicialização foram obtidas das linhas REPORT do Amazon CloudWatch Logs. Nas rodadas 1 e 2, essas informações foram recuperadas posteriormente dos registros preservados na conta. Como verificação complementar do instrumento de medição, comparou-se o tempo interno do núcleo com a duração reportada pela plataforma. A diferença média variou de 0,046 s a 0,072 s nas rodadas 1 a 3, com correlações entre 0,998 e 0,9997. Na rodada confirmatória, as diferenças médias foram de 0,019 s para 128 MB e de 0,002 a 0,003 s para 512 e 1.769 MB, com correlações entre 0,971 e 0,999.

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

Para cada condição foram calculados média, desvio-padrão amostral, mediana, mínimo e máximo. Na rodada confirmatória, as observações de um mesmo bloco foram tratadas como pareadas, pois cada bloco continha uma execução de cada condição.

O teste de Friedman foi empregado para avaliar globalmente as diferenças entre as quatro condições, utilizando-se o coeficiente W de Kendall como medida de tamanho de efeito global. Em seguida, foram realizadas três comparações *post hoc* diretamente relacionadas à RQ1 — Lambda 128 MB × EC2, Lambda 512 MB × EC2 e Lambda 1.769 MB × EC2 — por meio do teste de Wilcoxon bicaudal para amostras pareadas. Os valores de *p* foram ajustados pelo procedimento de Holm para as três comparações.

Como medida do tamanho do efeito das comparações pareadas, utilizou-se a correlação bisserial de postos r_{rb}. Neste trabalho, valores positivos indicam maior tempo da Lambda em relação à EC2 e valores negativos indicam menor tempo da Lambda.

Também foram calculadas, para cada bloco, as diferenças entre os tempos da Lambda e da EC2 e, para cada configuração, a razão entre suas médias. Os intervalos de confiança de 95% (IC 95%) das médias, das razões entre médias e das medianas das diferenças foram estimados por *bootstrap* percentil com 10.000 reamostragens de blocos completos. Na análise combinada, a reamostragem foi estratificada por sessão, preservando a estrutura das sessões A e B.

Como análise de sensibilidade à variabilidade temporal, as comparações foram repetidas separadamente para as sessões A e B, cada uma com 30 blocos, utilizando-se novamente o teste de Wilcoxon pareado com correção de Holm. Adotou-se nível de significância de 5%.

3.5. Análise da recuperação operacional

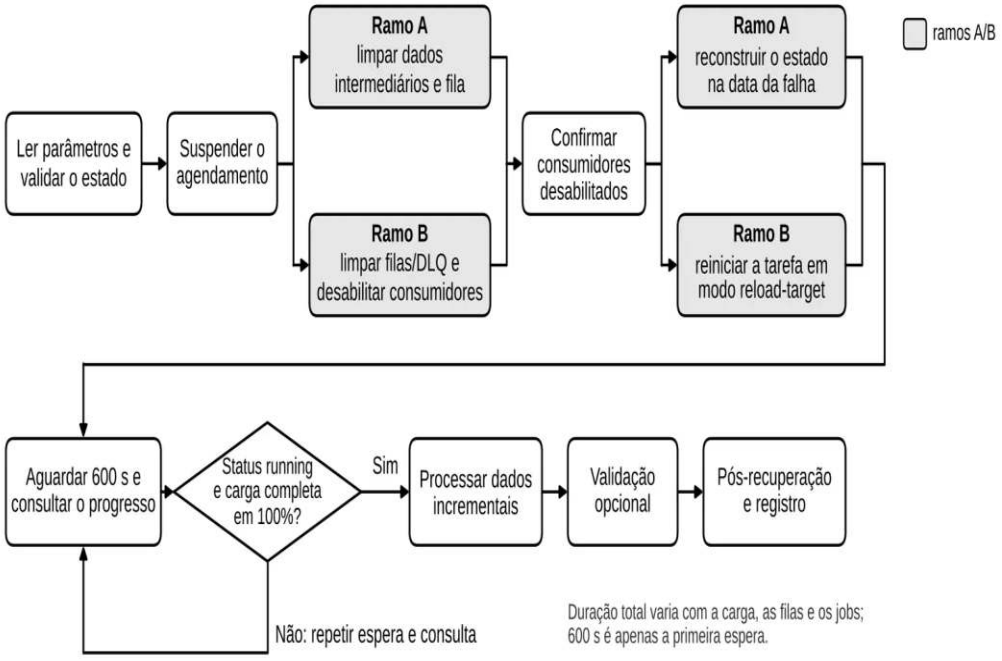
A análise operacional comparou duas estratégias presentes no estudo de caso: o procedimento manual e o fluxo automatizado implementado pelo DMS-Task-Monitor. Como os processos não são equivalentes etapa a etapa, a comparação foi restrita à estrutura do processo e ao esforço humano requerido do operador, não ao tempo total de recuperação.

No procedimento manual, as atividades incluem inspeção de logs e do estado da tarefa, obtenção de *checkpoint* ou definição da posição CDC (Amazon Web Services, [s. d.]j), preparação do reinício e validação posterior. No fluxo automatizado, resumido na Figura 3, a máquina de estados suspende o agendamento, limpa dados intermediários e filas, desabilita consumidores, reconstrói o estado necessário, reinicia a tarefa com recarga do destino, acompanha o progresso, processa dados incrementais, executa validação opcional e aciona o pós-processamento.

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

As estimativas de esforço humano são premissas dos autores, obtidas por decomposição de dois procedimentos. As fontes foram a definição da máquina de estados do fluxo automatizado, da qual se extraíram as macroetapas, a espera fixa de 600 s e o limite de verificações, e a descrição do procedimento manual, apoiada na documentação do AWS DMS sobre replicação contínua (Amazon Web Services, [s. d.]j). Cada procedimento foi decomposto em etapas, e cada etapa foi classificada como ação humana ou como tempo técnico do serviço. Os intervalos das ações humanas foram atribuídos por comparação entre as tarefas exigidas em cada estratégia: na triagem, consulta de estado e logs versus painel e teste de conectividade; no acesso, autenticação, seleção e validação do perfil, tratadas como premissa organizacional; na preparação e no disparo, extração e validação do *checkpoint* e montagem do comando versus acionamento do fluxo; no acompanhamento, inspeção manual de estado e métricas versus consulta do resultado registrado. Não houve cronometragem, entrevistas, uso de histórico de incidentes nem revisão por avaliador independente. Os valores caracterizam, portanto, a potencial redução da intervenção humana e não demonstram redução do MTTR.

Figura 3. Macrofluxo anonimizado da recuperação automatizada.



Fonte: elaboração própria a partir da definição da máquina de estados (2026).

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

3.6. Análise documental de CaaS e modelo de custos

O CaaS foi representado pelo Amazon EKS e analisado sem implantação experimental. A linha de responsabilidade operacional da Tabela 6 é uma síntese qualitativa da documentação oficial da AWS: o modelo de responsabilidade compartilhada, a página de segurança do EKS e as páginas de boas práticas e de configuração de memória do Lambda (Amazon Web Services, [s. d.]h, [s. d.]i, [s. d.]l, [s. d.]m), consultadas em 4 e 5 de novembro de 2026. O critério foi registrar, para cada alternativa, o que essa documentação atribui ao cliente. Não se construiu matriz por camada nem houve dupla verificação; a síntese descreve atribuições formais de responsabilidade e não mede o esforço operacional efetivo.

Foram considerados a região us-east-1, moeda USD e preços públicos da AWS. Para o EKS foram definidos dois cenários. No cenário dedicado, considerou-se um *cluster* em suporte padrão e um *pod* Linux/x86 executado continuamente em AWS Fargate, com capacidade faturada de 0,25 vCPU e 0,5 GB durante 730 h/mês. No cenário compartilhado, considerou-se que o custo do plano de controle já estava alocado a outros *workloads*, atribuindo-se à aplicação apenas o custo incremental da capacidade em Fargate.

No EKS, o Fargate acrescenta 256 MB à reserva de memória de cada *pod* e realiza o arredondamento para a combinação válida seguinte de recursos (Amazon Web Services, [s. d.]n). Assim, a configuração faturada de 0,25 vCPU e 0,5 GB pressupõe solicitação de memória de até 256 MB. A utilização do Fargate no modelo evita contabilizar simultaneamente nós EC2 e capacidade Fargate para o mesmo *workload*.

Como análise complementar, calculou-se também o custo de uma execução equivalente no Amazon ECS com Fargate, utilizando a mesma capacidade de 0,25 vCPU e 0,5 GB. Esse cenário não constitui uma quarta arquitetura experimental, sendo utilizado apenas como referência para estimar o impacto econômico da tarifa do plano de controle do EKS.

A fronteira principal do modelo econômico corresponde à camada computacional. Serviços como API Gateway, DynamoDB, transferência de dados e observabilidade não foram incluídos quando comuns ou dependentes da implementação específica. No cenário EKS/Fargate considerado, os 20 GB de armazenamento efêmero disponibilizados por padrão não geram cobrança adicional; eventual capacidade adicional deve ser incorporada a uma estimativa completa (Amazon Web Services, [s. d.]f). C representa o custo mensal estimado (US$) de cada alternativa.

O custo mensal da AWS Lambda foi modelado por:

$$C_{lambda} = N[P_{req} + (t_{fat\,(s)} \times M_{GB} \times P_{GB-s})] \quad (1)$$

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

em que N representa o número de invocações no período; P_{req}, o preço por requisição; $t_{fat\,(s)}$, a duração faturada de cada invocação; M_{GB}, a memória configurada em gigabytes; e P_{GB-s}, o preço por GB-segundo de computação.

O custo mensal da EC2 foi calculado por:

$$C_{ec2} = 730(P_{t3.micro} + P_{IPv4}) + 8 \times P_{gp3_GB} \quad (2)$$

em que $P_{t3.micro}$ corresponde ao preço horário da instância, P_{IPv4} ao preço horário do endereço IPv4 público e P_{gp3_GB} ao preço mensal por gigabyte de armazenamento EBS gp3. O fator 8 representa o volume de 8 GB adotado no cenário e 730 corresponde ao número de horas consideradas para um mês de disponibilidade contínua.

Para o cenário EKS dedicado, o custo foi estimado por:

$$C_{eks\ dedicado} = 730 \times P_{cluster} + 730 \times C_{Fargate/h} \quad (3)$$

em que $P_{cluster}$ é a tarifa horária do *cluster* e $C_{Fargate/h}$, o custo horário do *pod* Fargate. No EKS compartilhado e no ECS com Fargate, considera-se apenas o *pod* incremental:

$$C_{eks\ compartilhado} = C_{ecs} = (730 \times C_{Fargate/h}) \quad (4).$$

4. RESULTADOS E DISCUSSÃO

4.1. Rodadas exploratórias

Na configuração inicial, o código da Lambda reinstanciava o recurso do SDK boto3 a cada operação de gravação, ou seja, 42 vezes por invocação. Em 30 execuções, a Lambda apresentou tempo médio de 14,0007 s (desvio-padrão, DP = 0,4746 s) (Tabela 1), enquanto a EC2 apresentou 1,6967 s (DP = 0,1375 s). Após mover o recurso utilizado nas gravações para o escopo global do módulo, a média da Lambda foi reduzida para 2,3249 s (DP = 0,1684 s), correspondendo a uma redução de 83,4%. Na mesma rodada, a EC2 apresentou média de 0,2886 s (DP = 0,0101 s), queda de 83,0%. A correção foi aplicada às duas implementações; segundo os registros disponíveis, a EC2 da rodada 1 também criava o recurso dentro do intervalo cronometrado. Esse comportamento é coerente com a recomendação da AWS de inicializar clientes do SDK fora do *handler* para possibilitar seu reaproveitamento entre invocações (Amazon Web Services, [s. d.]h).

Tabela 1. Tempo interno do núcleo nas rodadas exploratórias

Condição	N	Média (s)	Desvio-padrão (s)	Mínimo (s)	Máximo (s)
Lambda — configuração inicial	30	14,0007	0,4746	13,3929	16,2163

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

Condição	N	Média (s)	Desvio-padrão (s)	Mínimo (s)	Máximo (s)
EC2 — configuração inicial	30	1,6967	0,1375	1,5554	2,2719
Lambda — recurso SDK corrigido	30	2,3249	0,1684	2,1925	2,6892
EC2 — recurso SDK corrigido, antes da padronização	30	0,2886	0,0101	0,2763	0,3228
EC2 — *runtime* e SDK padronizados	30	0,2579	0,0151	0,2397	0,2953
Lambda — rodada 3 (código da rodada 2, remedido)	30	2,4818	0,1861	2,3363	2,9166

Fonte: elaboração própria (2026).

Na terceira rodada, a EC2 apresentou média de 0,2579 s e a Lambda, remedida com o código da rodada anterior, 2,4818 s, valor 6,7% superior ao obtido anteriormente pela Lambda. Essa diferença, mesmo utilizando o mesmo código, evidencia variabilidade temporal entre medições realizadas em ambiente de nuvem.

4.2. Rodada confirmatória e efeito da memória

Na rodada confirmatória, todas as 240 medições concluíram as 42 atualizações previstas, sem itens não processados, novas tentativas ou novas conexões HTTP durante o intervalo instrumentado. A Tabela 2 resume a comparação pareada por bloco.

A análise global dos 60 blocos identificou diferenças entre as quatro condições (Friedman: $\chi^2(3) = 160{,}82$; $p < 0{,}001$), com W de Kendall de 0,893. Os postos médios foram 1,600 para EC2, 4,000 para Lambda 128 MB, 2,983 para Lambda 512 MB e 1,417 para Lambda 1.769 MB, em que menores postos correspondem a menores tempos de execução dentro de cada bloco.

Com 128 MB, a Lambda apresentou tempo médio de 1,4956 s, enquanto a média da EC2 foi de 0,2572 s, resultando em razão entre médias de 5,81 [IC 95%: 5,68; 5,95]. A Lambda foi mais lenta nos 60 blocos avaliados, com diferença estatisticamente significativa após correção de Holm ($p < 0{,}001$; $r_{rb} = 1{,}000$).

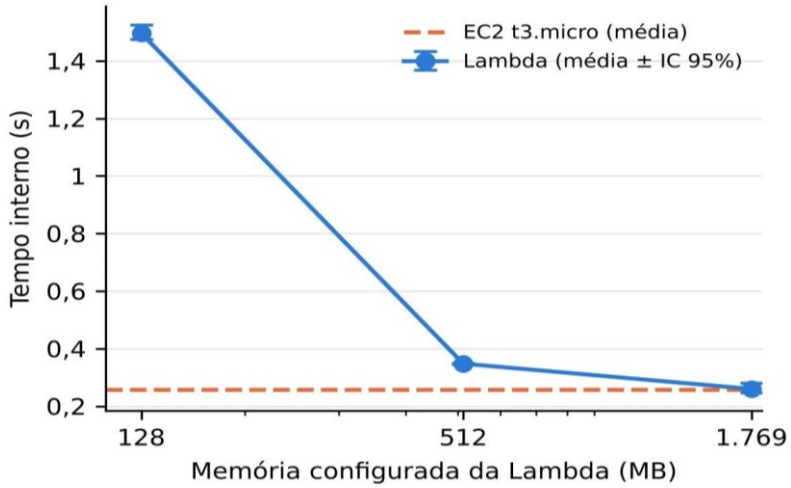
Com 512 MB, a média da Lambda foi reduzida para 0,3476 s, correspondendo a 1,35 vez a média da EC2 [IC 95%: 1,33; 1,37]. A Lambda novamente apresentou maior tempo em todos os 60 blocos ($p < 0{,}001$, após correção de Holm; $r_{rb} = 1{,}000$).

Com 1.769 MB, as médias das plataformas tornaram-se próximas (Figura 4): 0,2597 s para a Lambda e 0,2572 s para a EC2, resultando em razão entre médias de 1,01 [IC 95%: 0,96; 1,09]. No conjunto das duas sessões, o teste de Wilcoxon pareado não identificou diferença

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

estatisticamente significativa após a correção de Holm (p = 0,079; r_{rb} = -0,261). A Lambda apresentou maior tempo em 24 dos 60 blocos.

Figura 4. Tempo médio interno do núcleo da Lambda por memória configurada, comparado à EC2 t3.micro (rodada confirmatória)



Fonte: elaboração própria (2026).

Nota: barras de erro = IC 95% por *bootstrap* de blocos completos, estratificado por sessão (N = 60 blocos por condição).

Embora as médias se aproximem com 1.769 MB, esse resultado não demonstra equivalência entre as plataformas. A análise separada das sessões mostrou inversão na direção do efeito. Na sessão A, a Lambda foi mais lenta em 22 dos 30 blocos (p = 0,010, após correção de Holm; r_{rb} = 0,531), com razão entre médias de 1,04 [IC 95%: 1,00; 1,07]. Na sessão B, a Lambda foi mais rápida em 28 dos 30 blocos (p < 0,001; r_{rb} = -0,867), com razão entre médias de 0,99 [IC 95%: 0,90; 1,13].

A inversão observada entre as sessões envolveu alterações nas duas plataformas. A mediana da EC2 aumentou de 0,2443 s na sessão A para 0,2663 s na sessão B, enquanto a mediana da Lambda com 1.769 MB diminuiu de 0,2539 s para 0,2425 s. Assim, quando os tempos das duas plataformas se tornam próximos, a variabilidade entre sessões passa a apresentar magnitude relevante em relação à própria diferença entre as arquiteturas.

Na sessão B, um dos blocos apresentou diferença de aproximadamente +0,52 s para a Lambda com 1.769 MB em relação à EC2. O valor foi mantido na análise, pois não foi estabelecido critério prévio que justificasse sua exclusão. Sua presença contribui para explicar por que a razão entre médias permaneceu próxima de 1 e seu intervalo de confiança incluiu a

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

unidade, embora a comparação por postos indicasse menor tempo da Lambda na maioria dos blocos daquela sessão.

Em experimento complementar, foram forçadas dez partidas a frio na Lambda com 128 MB. A duração média da inicialização foi de 894 ms (DP = 85 ms), a duração faturada média foi de 2.723 ms e o tempo interno médio do núcleo foi de 1,81 s, superior aos 1,4956 s observados nas execuções já inicializadas.

Tabela 2. Tempo interno do núcleo na rodada confirmatória, considerando os 60 blocos pareados

Condição	**Média (s)**	**DP (s)**	**Mediana (s)**	**Razão L/EC2 [IC 95%]**	**Mediana de L − EC2 (s) [IC 95%]**	**p (Holm)**	r_rb	**Blocos com L > EC2**
EC2 t3.micro	0,2572	0,0191	0,2548	1,00 (referência)	—	—	—	—
Lambda 128 MB	1,4956	0,1025	1,4800	5,81 [5,68; 5,95]	1,2287 [1,2165; 1,2356]	< 0,001	1,000	60/60
Lambda 512 MB	0,3476	0,0112	0,3469	1,35 [1,33; 1,37]	0,0895 [0,0863; 0,0987]	< 0,001	1,000	60/60
Lambda 1.769 MB	0,2597	0,0713	0,2485	1,01 [0,96; 1,09]	-0,0044 [-0,0115; -0,0019]	0,079	-0,261	24/60

Fonte: elaboração própria (2026).

Nota: L = Lambda; N = 60 blocos pareados. Teste global de Friedman: $\chi^2(3) = 160{,}82$; $p < 0{,}001$; W de Kendall = 0,893. Comparações *post hoc* pelo teste de Wilcoxon bicaudal para amostras pareadas, com valores de p ajustados pelo procedimento de Holm (três comparações). r_rb = correlação bisserial de postos pareada; valores positivos indicam maior tempo da Lambda. IC 95% por *bootstrap* percentil com 10.000 reamostragens de blocos completos, estratificadas por sessão. Com 1.769 MB, a mediana das diferenças é negativa e a diferença entre as médias é positiva (+0,0025 s), por efeito de um bloco extremo (+0,52 s) na sessão B; sem a estratificação por sessão, o IC da mediana inclui zero [−0,0129; 0,0004].

4.3. Recuperação operacional

Por decomposição das tarefas, sem replicação de incidentes reais, estimou-se no procedimento manual esforço humano entre 20 e 50 min, distribuído entre triagem, análise, escalonamento ao especialista, obtenção de acesso, preparação do reinício e validação. Com o

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

DMS-Task-Monitor, o esforço humano estimado foi de 3 a 9 min, compreendendo validação da elegibilidade, autenticação, seleção da tarefa, acionamento do fluxo e consulta do resultado (Tabela 3).

O tempo técnico do processo não foi incorporado ao esforço humano. No fluxo automatizado, ele inclui pelo menos a espera inicial de 600 s, recarga dos dados, execução de *jobs*, consultas periódicas e pós-processamento, permanecendo indeterminado na ausência de registros temporais de execuções reais.

Tabela 3. Esforço humano estimado na recuperação operacional: procedimento manual e DMS-Task-Monitor.

Etapa analisada	**Procedimento manual**	**DMS-Task-Monitor**
Triagem e verificação de elegibilidade	5–10 min	1–3 min
Acesso, autorização e preparação	5–15 min	1–2 min
Preparação e disparo da recuperação	5–10 min	0–1 min
Acompanhamento e validação pelo operador	5–15 min	1–3 min
Total de esforço humano estimado	20–50 min	3–9 min
Tempo técnico do serviço	indeterminado	≥ 10 min + recarga, *jobs*, *polling* e pós-processamento

Fonte: elaboração própria (2026).

Nota: intervalos estimados pelos autores por decomposição do procedimento manual documentado e da definição da máquina de estados; não são observações.

O ponto médio dos intervalos corresponde a 35 min no procedimento manual e 6 min com a ferramenta, resultando em redução ilustrativa de 82,9% no esforço humano estimado, que depende integralmente das premissas adotadas e não deve ser interpretada como redução de MTTR.

4.4. Análise de custos

A análise econômica utilizou preços públicos da AWS para a região us-east-1, consultados em 11 de setembro de 2026 e reconferidos em 5 de novembro de 2026. Foram excluídos *Free Tier*, créditos promocionais e descontos contratuais. As tarifas e premissas unitárias utilizadas no modelo são apresentadas na Tabela 4.

No cenário ilustrativo, foram consideradas 1.000 invocações mensais, representativas de uma ferramenta de uso intermitente. Para a Lambda, foram utilizadas as durações faturadas médias da rodada confirmatória: 1.515 ms (128 MB), 352 ms (512 MB) e 263 ms (1.769 MB).

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

Para a EC2, o cenário-base inclui IPv4 público, e um cenário sem IPv4 foi usado como sensibilidade; as demais premissas de EC2 e EKS são as da Seção 3.6. Se todas as invocações com 128 MB ocorressem a frio, com a duração faturada média de 2.723 ms observada no experimento de partidas a frio, o custo de 1.000 invocações seria de aproximadamente US$ 0,0059.

Tabela 4. Premissas e tarifas utilizadas no modelo econômico

Item	Valor adotado	Fonte
Lambda x86 — requisições	US$ 0,20 por 1 milhão	Amazon Web Services ([s. d.]g)
Lambda x86 — computação	US$ 0,0000166667 por GB-s	Amazon Web Services ([s. d.]g)
EC2 t3.micro Linux *on-demand*	US$ 0,0104/h	Amazon Web Services ([s. d.]b)
IPv4 público	US$ 0,005/h	Amazon Web Services ([s. d.]e)
EBS gp3	US$ 0,08 por GB-mês	Amazon Web Services ([s. d.]a)
EKS em suporte padrão	US$ 0,10 por *cluster*-hora	Amazon Web Services ([s. d.]d)
ECS com Fargate	sem tarifa de orquestração	Amazon Web Services ([s. d.]c)
Fargate Linux/x86 — CPU	US$ 0,0000112444 por vCPU-s	Amazon Web Services ([s. d.]f)
Fargate Linux/x86 — memória	US$ 0,0000012347 por GB-s	Amazon Web Services ([s. d.]f)

Fonte: elaboração própria (2026), com tarifas da Amazon Web Services consultadas em 11 set. 2026 e reconferidas em 5 nov. 2026.

Aplicando-se as tarifas da Tabela 4 às Equações (1)–(4), obtiveram-se os custos mensais da Tabela 5.

Tabela 5. Custo mensal estimado no cenário ilustrativo

Cenário	Custo mensal estimado da camada computacional
Lambda 128 MB, 1.000 invocações/mês	US$ 0,0034
Lambda 512 MB, 1.000 invocações/mês	US$ 0,0031
Lambda 1.769 MB, 1.000 invocações/mês	US$ 0,0078
EC2 t3.micro, 730 h + 8 GB gp3 + IPv4 público	US$ 11,88
EC2 t3.micro sem IPv4 (sensibilidade)	US$ 8,23
EKS dedicado + um *pod* Fargate contínuo	US$ 82,01
EKS compartilhado (somente o *pod*)	US$ 9,01
ECS + uma *task* Fargate contínua	US$ 9,01

Fonte: elaboração própria (2026).

Os resultados da Tabela 5 mostram que, para o cenário de 1.000 invocações mensais, o custo estimado da Lambda é várias ordens de grandeza inferior ao das alternativas que

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

mantêm capacidade provisionada continuamente. Entre as configurações da Lambda, 512 MB apresentou o menor custo mensal estimado, de US$ 0,0031, ligeiramente inferior aos US$ 0,0034 obtidos com 128 MB. Esse resultado ocorre porque a redução da duração faturada compensou o aumento da memória configurada. A configuração de 1.769 MB apresentou maior custo entre as três alternativas Lambda, US$ 0,0078, embora ainda permaneça muito abaixo dos valores estimados para EC2 e EKS nas premissas adotadas.

Para a EC2, a remoção do IPv4 público reduziu o custo de US$ 11,88 para US$ 8,23 por mês, mostrando que componentes auxiliares pesam em aplicações de baixo volume. No EKS, o custo caiu de US$ 82,01 para US$ 9,01 quando a tarifa do plano de controle foi considerada absorvida por outros *workloads*, o que torna a infraestrutura preexistente decisiva para o CaaS. O ECS/Fargate teve o mesmo custo incremental do EKS compartilhado, pois usa a mesma capacidade Fargate sem tarifa de plano de controle, o que evidencia o impacto econômico da camada de orquestração.

Mantidas as premissas do modelo e consideradas somente as parcelas incluídas na fronteira econômica, a igualdade de custo entre Lambda e EC2 com IPv4 ocorreria aproximadamente em 3,54 milhões de invocações mensais para Lambda 128 MB, 3,80 milhões para 512 MB e 1,53 milhão para 1.769 MB. Sem IPv4, esses pontos seriam aproximadamente 2,45, 2,63 e 1,06 milhões de invocações mensais, respectivamente. Esses valores constituem pontos de equilíbrio específicos do modelo adotado e não limites universais entre FaaS e IaaS.

Como as durações faturadas foram mantidas constantes, o custo da Lambda cresce linearmente com o número de invocações. A configuração de 512 MB foi a de menor custo em qualquer volume e, mesmo com 1.000.000 de invocações mensais (US$ 3,13), permaneceu abaixo dos US$ 11,88 da EC2 no cenário-base. Aumentar a memória não implica aumento proporcional do custo, pois a maior capacidade computacional reduz a duração: entre 128 e 512 MB, a redução compensou o acréscimo; com 1.769 MB, não.

Esses resultados devem ser interpretados dentro da fronteira econômica definida. Concorrência, duração das execuções, tráfego de dados, observabilidade, banco de dados, API Gateway, créditos de CPU, requisitos de disponibilidade e compartilhamento de infraestrutura podem alterar tanto os custos absolutos quanto os pontos de equilíbrio entre as alternativas.

4.5. Discussão e guia de decisão

A Tabela 6 sintetiza os resultados experimentais, econômicos e documentais. As evidências têm naturezas distintas: o tempo interno do núcleo foi medido apenas para Lambda e EC2; os custos decorrem do modelo da Seção 3.6, com resultados na Seção 4.4; e os aspectos

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

do EKS provêm da análise documental. Nenhuma alternativa apresentou vantagem simultânea em todos os critérios.

Tabela 6. Síntese comparativa das alternativas no caso analisado

Critério	FaaS (AWS Lambda)	IaaS (Amazon EC2)	CaaS (Amazon EKS)
Tempo médio do núcleo	1,50 s (128 MB); 0,35 s (512 MB); 0,26 s (1.769 MB)	0,26 s	não medido
Custo mensal estimado	US$ 0,0031 a 0,0078 (1.000 invocações)	US$ 11,88 (US$ 8,23 sem IPv4)	US$ 82,01 (dedicado); US$ 9,01 (compartilhado)
Custo sem uso	não há duração faturada	instância, volume e IPv4 seguem cobrados	*cluster* e *pod* seguem cobrados no cenário dedicado
Responsabilidade operacional	função, memória, permissões e observabilidade	sistema operacional, *patches*, capacidade e processo	versões, *workloads*, políticas e capacidade; plano de controle gerenciado pela AWS
Cenário mais coerente	acesso esporádico e baixo custo incremental	serviço contínuo que exige controle do ambiente	*cluster* já existente ou vários *workloads* conteinerizados

Fonte: elaboração própria (2026). Nota: tempo medido na rodada confirmatória (N = 60 por condição; ver Tabela 2); custos do modelo da Seção 3.6 (Tabela 5); demais critérios com base na documentação da AWS (Amazon Web Services, [s. d.]m).

A proposição P1 foi confirmada, com uma ressalva. A razão entre os tempos médios da Lambda e da EC2 caiu de 5,81 com 128 MB para 1,35 com 512 MB e 1,01 com 1.769 MB, configuração que corresponde a aproximadamente uma vCPU. O resultado quantifica, para um núcleo composto por chamadas ao DynamoDB, a alocação de CPU proporcional à memória descrita por Wang *et al.* (2018) e pela documentação do serviço. A ressalva é que, próximo da paridade, a direção da diferença se inverteu entre as sessões (razões de 1,04 e 0,99). Laaber, Scheuner e Leitner (2019) observaram coeficientes de variação de 0,03% a mais de 100% em medições repetidas em nuvem pública; na rodada confirmatória, o coeficiente de variação foi de 3,2% na Lambda com 512 MB, 6,9% com 128 MB, 7,4% na EC2 e 27,5% na Lambda com 1.769 MB, este último elevado por um único bloco extremo. Diferenças de 1% a 4% com sinal instável entre sessões, menores que a variação das próprias medições, não podem ser atribuídas à arquitetura.

As rodadas exploratórias acrescentam um segundo achado: a reinstanciação do recurso do SDK elevava o tempo da Lambda de 2,32 s para 14,00 s, efeito de cerca de seis vezes, da mesma ordem da maior diferença medida entre as plataformas. Scheuner e Leitner (2020) apontaram falhas de reprodutibilidade e divergência nas configurações testadas nos estudos de desempenho de FaaS; o presente caso mostra, com valores, como uma assimetria de implementação ou de configuração pode ser confundida com uma diferença entre arquiteturas,

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

o que justifica o delineamento intercalado e aleatorizado adotado na rodada confirmatória (Abedi; Brecht, 2017).

A proposição P2 foi confirmada dentro da fronteira do modelo. Para 1.000 invocações mensais, a Lambda custou menos de US$ 0,01, contra US$ 11,88 da EC2, redução superior a 99,9%, e o equilíbrio só ocorreria entre 1,06 e 3,80 milhões de invocações mensais. A direção coincide com a literatura: Villamizar *et al.* (2017) estimaram redução de 70% ou mais, Adzic e Chatley (2017) relataram de 66% a 95% em casos industriais e Allen *et al.* (2024) encontraram vantagem das funções quando o número mensal de requisições é limitado. A redução maior obtida aqui decorre do volume reduzido do cenário adotado e de o modelo considerar apenas a camada computacional. O perfil também corresponde ao que Eismann *et al.* (2021) identificaram como predominante em aplicações *serverless*: cargas irregulares e redução de custos como principal motivação. O estudo acrescenta dois elementos a esse quadro: a configuração de 512 MB foi mais barata que a de 128 MB, porque a redução da duração faturada compensou o aumento da memória, e o custo do CaaS variou de US$ 82,01 para US$ 9,01 conforme o *cluster* fosse dedicado ou compartilhado.

A proposição P3 foi examinada apenas documentalmente. As responsabilidades da Tabela 6 seguem o contínuo de abstração de Jonas *et al.* (2019): em FaaS restam à equipe a função, a memória, as permissões e a observabilidade; em IaaS somam-se o sistema operacional, as atualizações e a capacidade; em CaaS, mesmo com o plano de controle gerenciado, permanecem versões, *workloads*, políticas e capacidade. O esforço correspondente não foi medido, e o desempenho do EKS não foi avaliado; Decker, Kasprzak e Kunkel (2022) observaram diferenças de desempenho entre plataformas *serverless* executadas sobre Kubernetes, o que reforça a necessidade de medição nesse cenário. Em conjunto, as três proposições sustentam o guia de decisão: FaaS para acesso esporádico, desde que a função seja dimensionada; IaaS quando há necessidade de controle do ambiente ou de serviço contínuo; e CaaS quando já existe *cluster* compartilhado ou vários *workloads* conteinerizados. Quando Kubernetes não é requisito, o ECS com Fargate oferece o mesmo custo incremental sem a tarifa do plano de controle.

5. CONSIDERAÇÕES FINAIS

Este estudo avaliou desempenho, custo e responsabilidade operacional de FaaS, IaaS e CaaS na hospedagem de uma ferramenta operacional de uso intermitente, o DMS-Task-Monitor, com AWS Lambda, Amazon EC2 e Amazon EKS.

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

Em resposta à RQ1, a diferença de tempo entre Lambda e EC2 dependeu da memória configurada: a razão entre os tempos médios foi de 5,81 com 128 MB e de 1,35 com 512 MB; com 1.769 MB, não houve diferença estatisticamente significativa no conjunto das sessões (p = 0,079), resultado que não demonstra equivalência, pois a direção do efeito se inverteu entre elas. As rodadas exploratórias mostraram ainda que corrigir a reinstanciação do recurso do SDK reduziu em 83,4% o tempo médio da Lambda, efeito comparável ao da própria escolha da plataforma.

Em resposta à RQ2, dentro da fronteira do modelo, a Lambda teve o menor custo para 1.000 invocações mensais, com 512 MB como configuração mais barata e pontos de equilíbrio com a EC2 entre 1,06 e 3,80 milhões de invocações mensais. EC2 e EKS dedicado mantiveram custos de capacidade provisionada, e a diferença entre EKS dedicado e compartilhado mostrou o peso da infraestrutura já existente. Quanto às responsabilidades, FaaS delega ao provedor a maior parcela da infraestrutura, IaaS oferece mais controle sobre o ambiente e CaaS acrescenta a gestão da orquestração e dos *workloads*.

Em resposta à RQ3, a análise procedimental indica potencial de redução e padronização da intervenção humana: o esforço estimado passou de 20–50 min para 3–9 min. Trata-se de estimativas, não de medição de MTTR, e o tempo técnico do fluxo automatizado, de pelo menos 10 min, não foi medido.

As limitações incluem o estudo de caso único; a medição restrita ao núcleo instrumentado; uma única região, um tipo de instância, arquitetura x86_64 e uma carga de trabalho; duas sessões confirmatórias no mesmo dia; partidas a frio avaliadas apenas com 128 MB; a execução do orquestrador na própria instância EC2 avaliada; a ausência de *benchmark* do EKS e do ECS; a dependência dos preços vigentes; estimativas de esforço humano baseadas em premissas dos autores, sem replicação de incidentes reais nem revisão independente; e atribuição de responsabilidades por síntese documental, sem matriz por camada. Trabalhos futuros devem medir a latência ponta a ponta sob diferentes cargas e níveis de concorrência, repetir as sessões em dias distintos, avaliar a arquitetura ARM, implantar os cenários EKS e ECS e coletar registros temporais de incidentes reais para estimar o MTTR.

Os resultados confirmam, para o perfil analisado, que não há arquitetura universalmente superior, e acrescentam as condições em que cada uma se justifica: a decisão deve considerar o perfil de uso, a memória configurada, a infraestrutura já disponível, o custo da capacidade ociosa e as responsabilidades operacionais assumidas pela equipe.

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

DECLARAÇÃO SOBRE O USO DE INTELIGÊNCIA ARTIFICIAL

Foram utilizadas ferramentas de inteligência artificial generativa. O Google Antigravity (modelos Gemini 3.8 Flash e Gemini 3.1 Pro) auxiliou, como agente de código, na execução das rodadas 1 a 3 na AWS, na consolidação dos registros e no cálculo de estatísticas descritivas. O Claude Code (Anthropic) executou a rodada confirmatória conforme o protocolo definido pelos autores, recuperou os registros da plataforma e executou as rotinas de análise estatística, modelagem de custos e geração de visualizações. O Claude (modelos Sonnet 5.5 e Opus 5.5) apoiou a revisão crítica, a conferência de fontes, referências, valores e cálculos e a preparação de figuras e tabelas. A concepção, o delineamento, as decisões metodológicas, a interpretação e as conclusões são dos autores, que conferiram as saídas com os dados brutos, o código e as fontes, revisaram integralmente o manuscrito e assumem responsabilidade por seu conteúdo. Nenhuma ferramenta de IA é indicada como autora.

REFERÊNCIAS

ABEDI, Ali; BRECHT, Tim. Conducting repeatable experiments in highly variable cloud computing environments. In: ACM/SPEC INTERNATIONAL CONFERENCE ON PERFORMANCE ENGINEERING, 8., 2017, L'Aquila. **Proceedings** […]. New York: ACM, 2017. p. 287–292. DOI: https://doi.org/10.1145/3030207.3030229.

ADZIC, Gojko; CHATLEY, Robert. *Serverless* computing: economic and architectural impact. In: JOINT MEETING ON FOUNDATIONS OF *SOFTWARE* ENGINEERING (ESEC/FSE), 11., 2017, Paderborn. **Proceedings** […]. New York: ACM, 2017. p. 884–889. DOI: https://doi.org/10.1145/3106237.3117767.

ALLEN, Christopher; LI, Xiaozhou; ABDELFATTAH, Amr S.; CERNY, Tomas; TAIBI, Davide. Comparing cost and performance of microservices and *serverless* in AWS: EC2 vs Lambda. In: HAN, H.; BAKER, E. (ed.). **Next Generation Data Science**: Second Southwest Data Science Conference, SDSC 2023, Waco, TX, USA, March 24–25, 2023, Revised Selected Papers. Cham: Springer, 2024. (Communications in Computer and Information Science, v. 2113). p. 60–72. DOI: https://doi.org/10.1007/978-3-031-61816-1_5.

AMAZON WEB SERVICES. **Amazon EBS pricing**. [S. l.], [s. d.]a. Disponível em: https://aws.amazon.com/ebs/pricing/. Acesso em: 11 set. 2026.

AMAZON WEB SERVICES. **Amazon EC2 T3 instances**. [S. l.], [s. d.]b. Disponível em: https://aws.amazon.com/ec2/instance-types/t3/. Acesso em: 11 set. 2026.

AMAZON WEB SERVICES. **Amazon ECS pricing**. [S. l.], [s. d.]c. Disponível em: https://aws.amazon.com/ecs/pricing/. Acesso em: 5 nov. 2026.

AMAZON WEB SERVICES. **Amazon EKS pricing**. [S. l.], [s. d.]d. Disponível em: https://aws.amazon.com/eks/pricing/. Acesso em: 11 set. 2026.

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

AMAZON WEB SERVICES. **Amazon VPC pricing**. [S. l.], [s. d.]e. Disponível em: https://aws.amazon.com/vpc/pricing/. Acesso em: 5 nov. 2026.

AMAZON WEB SERVICES. **AWS Fargate pricing**. [S. l.], [s. d.]f. Disponível em: https://aws.amazon.com/fargate/pricing/. Acesso em: 11 set. 2026.

AMAZON WEB SERVICES. **AWS Lambda pricing**. [S. l.], [s. d.]g. Disponível em: https://aws.amazon.com/lambda/pricing/. Acesso em: 11 set. 2026.

AMAZON WEB SERVICES. **Best practices for working with AWS Lambda Functions**. [S. l.], [s. d.]h. Disponível em: https://docs.aws.amazon.com/lambda/latest/dg/best-practices.html. Acesso em: 5 nov. 2026.

AMAZON WEB SERVICES. **Configure Lambda Function memory**. [S. l.], [s. d.]i. Disponível em: https://docs.aws.amazon.com/lambda/latest/dg/configuration-memory.html. Acesso em: 5 nov. 2026.

AMAZON WEB SERVICES. **Creating tasks for ongoing replication using AWS DMS**. [S. l.], [s. d.]j. Disponível em: https://docs.aws.amazon.com/dms/latest/userguide/CHAP_Task.CDC.html. Acesso em: 11 set. 2026.

AMAZON WEB SERVICES. **Purchasing On-Demand Instances for Amazon EC2**. [S. l.], [s. d.]k. Disponível em: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-on-demand-instances.html. Acesso em: 4 nov. 2026.

AMAZON WEB SERVICES. **Security in Amazon EKS**. [S. l.], [s. d.]l. Disponível em: https://docs.aws.amazon.com/eks/latest/userguide/security.html. Acesso em: 4 nov. 2026.

AMAZON WEB SERVICES. **Shared responsibility model**. [S. l.], [s. d.]m. Disponível em: https://aws.amazon.com/compliance/shared-responsibility-model/. Acesso em: 4 nov. 2026.

AMAZON WEB SERVICES. **Understand Fargate Pod configuration details**. [S. l.], [s. d.]n. Disponível em: https://docs.aws.amazon.com/eks/latest/userguide/fargate-pod-configuration.html. Acesso em: 5 nov. 2026.

AMAZON WEB SERVICES. **What is AWS Database Migration Service?** [S. l.], [s. d.]o. Disponível em: https://docs.aws.amazon.com/dms/latest/userguide/Welcome.html. Acesso em: 11 set. 2026.

BURNS, Brendan; GRANT, Brian; OPPENHEIMER, David; BREWER, Eric; WILKES, John. Borg, Omega, and Kubernetes. **ACM Queue**, v. 14, n. 1, p. 70–93, 2016. DOI: https://doi.org/10.1145/2898442.2898444.

DECKER, Jonathan; KASPRZAK, Piotr; KUNKEL, Julian Martin. Performance evaluation of open-source *serverless* platforms for Kubernetes. **Algorithms**, v. 15, n. 7, art. 234, 2022. DOI: https://doi.org/10.3390/a15070234.

EISMANN, Simon *et al. Serverless* applications: why, when, and how? **IEEE Software**, v. 38, n. 1, p. 32–39, 2021. DOI: https://doi.org/10.1109/MS.2020.3023302.

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

FORSGREN, Nicole; HUMBLE, Jez; KIM, Gene. **Accelerate**: the science of lean *software* and DevOps. Portland: IT Revolution Press, 2018.

GARTNER. **Gartner forecasts worldwide public cloud end-user spending to total $723 billion in 2025**. [S. l.], 19 nov. 2024. Disponível em: https://www.gartner.com/en/newsroom/press-releases/2024-11-19-gartner-forecasts-worldwide-public-cloud-end-user-spending-to-total-723-billion-dollars-in-2025. Acesso em: 4 nov. 2026.

GUPTA, Shubham; GEBHART, Jeff. AWS Lambda standardizes billing for INIT phase. **AWS Compute Blog**, [S. l.], 29 abr. 2025. Disponível em: https://aws.amazon.com/blogs/compute/aws-lambda-standardizes-billing-for-init-phase/. Acesso em: 4 nov. 2026.

IBM. **Qual é o tempo médio para reparo (MTTR)?** [S. l.], [s. d.]. Disponível em: https://www.ibm.com/br-pt/think/topics/mttr. Acesso em: 4 nov. 2026.

JONAS, Eric *et al.* **Cloud programming simplified**: a Berkeley view on *serverless* computing. Berkeley: University of California, 2019. (Technical Report UCB/EECS-2019-3). Disponível em: https://arxiv.org/abs/1902.03383. Acesso em: 4 nov. 2026.

LAABER, Christoph; SCHEUNER, Joel; LEITNER, Philipp. *Software* micro*benchmark*ing in the cloud. How bad is it really? **Empirical *Software* Engineering**, v. 24, n. 4, p. 2469–2508, 2019. DOI: https://doi.org/10.1007/s10664-019-09681-1.

MELL, Peter; GRANCE, Timothy. **The NIST definition of cloud computing**. Gaithersburg: National Institute of Standards and Technology, 2011. (NIST Special Publication, 800-145). DOI: https://doi.org/10.6028/NIST.SP.800-145.

PAHL, Claus. Containerization and the PaaS cloud. **IEEE Cloud Computing**, v. 2, n. 3, p. 24–31, 2015. DOI: https://doi.org/10.1109/MCC.2015.51.

PAPADOPOULOS, Alessandro Vittorio *et al.* Methodological principles for reproducible performance evaluation in cloud computing. **IEEE Transactions on *Software* Engineering**, v. 47, n. 8, p. 1528–1543, 2021. DOI: https://doi.org/10.1109/TSE.2019.2927908.

RETOOL. **The 2022 state of internal tools**. [S. l.], 1 fev. 2023. Disponível em: https://retool.com/blog/state-of-internal-tools-2022. Acesso em: 4 nov. 2026.

RUNESON, Per; HÖST, Martin. Guidelines for conducting and reporting case study research in *software* engineering. **Empirical *Software* Engineering**, v. 14, n. 2, p. 131–164, 2009. DOI: https://doi.org/10.1007/s10664-008-9102-8.

SCHEUNER, Joel; LEITNER, Philipp. Function-as-a-Service performance evaluation: a multivocal literature review. **Journal of Systems and Software**, v. 170, art. 110708, 2020. DOI: https://doi.org/10.1016/j.jss.2020.110708.

VILLAMIZAR, Mario *et al.* Cost comparison of running web applications in the cloud using monolithic, microservice, and AWS Lambda architectures. **Service Oriented Computing and Applications**, v. 11, n. 2, p. 233–247, 2017. DOI: https://doi.org/10.1007/s11761-017-0208-y.

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

WANG, Liang *et al.* Peeking behind the curtains of *serverless* platforms. In: USENIX ANNUAL TECHNICAL CONFERENCE (USENIX ATC 18), 2018, Boston. **Proceedings** […]. Berkeley: USENIX Association, 2018. p. 133–146.

WOHLIN, Claes; RUNESON, Per; HÖST, Martin; OHLSSON, Magnus C.; REGNELL, Björn; WESSLÉN, Anders. **Experimentation in *software* engineering**. Berlin, Heidelberg: Springer, 2012. DOI: https://doi.org/10.1007/978-3-642-29044-2.

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.