Uma Análise Prática dos Frameworks Serverless AWS SAM, Pulumi e Serverless Framework

Adriano Prado Cavalheiro, Brenda Medeiros Lopes, Claudio Schepke

[1]Laboratório de Estudos Avançados em Computação (LEA)
Universidade Federal do Pampa (UNIPAMPA) – Alegrete – RS – Brazil

{adrianocavalheiro.aluno,brendalopes.aluno,claudioschepke}@unipampa.edu.br

Abstract. *Serverless computing facilitates the development of cloud applications by abstracting the infrastructure. This paper presents a case study comparing AWS SAM, Serverless Framework, and Pulumi in the implementation of a RESTful API on AWS. We evaluate efficiency, flexibility, and integration with cloud services based on observations from an academic course with ten participants, seven of whom completed the evaluation. We analyze aspects such as configuration, documentation, and code simplicity. Pulumi stood out for its flexibility, AWS SAM for its native integration, and Serverless Framework for its extensibility. The study presents practical guidelines for choosing serverless frameworks.*

Resumo. *A Computação Serverless facilita o desenvolvimento de aplicações em nuvem ao abstrair a infraestrutura. Este trabalho apresenta um estudo de caso que compara AWS SAM, Serverless Framework e Pulumi na implementação de uma API RESTful na AWS. Avaliamos a eficiência, flexibilidade e integração com serviços de nuvem com base em observações de uma disciplina acadêmica com dez participantes, dos quais sete concluíram a avaliação. Foram analisados aspectos como configuração, documentação e simplicidade do código. O Pulumi destacou-se pela flexibilidade, AWS SAM pela integração nativa e o Serverless Framework pela extensibilidade. O estudo apresenta orientações práticas para a escolha de frameworks serverless.*

1. Introdução

A computação *serverless* emergiu como um modelo inovador para o desenvolvimento de aplicações em nuvem, oferecendo uma abstração completa dos recursos de *hardware* e reduzindo a necessidade de gerenciamento direto da infraestrutura pelos desenvolvedores [Baldini et al. 2017]. Este modelo tem se expandido, oferecendo novas maneiras de construir e escalar aplicações na nuvem [Jonas et al. 2019, Hellerstein et al. 2018]. A adoção do *serverless* permite que a alocação e a escalabilidade de recursos sejam geridas automaticamente pelo provedor de serviços em nuvem, liberando os desenvolvedores para se concentrarem na lógica de negócios [Roberts 2016, Cloud 2024]. Além disso, essa abordagem melhora a eficiência no desenvolvimento de *software* e reduz os custos operacionais, uma vez que os usuários pagam apenas pelo tempo efetivo de execução das funções [Roberts 2018, Adzic and Chatley 2017].

Entretanto, a escolha do *framework* adequado pode influenciar diretamente a produtividade e a escalabilidade das aplicações desenvolvidas [Eismann et al. 2020]. Com o

crescimento do uso de *frameworks serverless*, torna-se necessário realizar uma análise dessas ferramentas para auxiliar na escolha entre as opções disponíveis [Bille 2023b, Bille 2023a, TatvaSoft 2022, Outsource 2023, Latreille 2022]. Este estudo propõe investigar essas questões e fornecer uma comparação prática dos *frameworks* AWS SAM, Serverless Framework e Pulumi. Esses *frameworks* foram escolhidos com base em sua popularidade e relevância para o desenvolvimento em nuvem, conforme discutido em revisões de mercado e na literatura [Adzic and Chatley 2017, Jonas et al. 2019]. O AWS SAM foi selecionado por ser um *framework* nativo da AWS, permitindo uma comparação direta com soluções de múltiplas plataformas. Serverless Framework e Pulumi foram escolhidos por sua poratabilidade entre provedores de nuvem, conforme relatado no *RightScale State of the Cloud Report* [RightScale 2020] e na documentação do Pulumi [Corp. 2024]. Python foi selecionado por ser compatível com todos os *frameworks* estudados, por seu suporte na AWS, e por ser adequado para a criação de APIs RESTful, conforme evidenciam as documentações de Flask e FastAPI [Projects 2024, FastAPI 2024].

A análise está restrita ao ecossistema AWS, limitando as conclusões ao contexto dessa infraestrutura. Diferentes provedores de nuvem oferecem variações em desempenho, funcionalidades e ferramentas, o que pode influenciar os resultados de uma abordagem *serverless*. Dessa forma, os resultados deste estudo devem ser interpretados considerando as características específicas da AWS.

Dada essa contextualização, a questão de pesquisa central é: "Quais são as principais vantagens e limitações dos *frameworks serverless* AWS SAM, Serverless Framework e Pulumi na implementação de uma API RESTful, considerando a perspectiva do desenvolvedor?" Essa questão norteia a análise prática dos *frameworks* e busca explorar aspectos como facilidade de uso, flexibilidade e adequação à integração com serviços de nuvem.

O estudo busca contribuir com orientações práticas para desenvolvedores e organizações na escolha do *framework serverless* adequado conforme às suas necessidades. Avaliando a facilidade de uso, a curva de aprendizado e os desafios enfrentados durante a implementação, o estudo aprofunda o entendimento das questões práticas relacionadas a esses *frameworks*.

A pesquisa contou com dez participantes que desenvolveram uma *RESTful API* de gestão de tarefas em Python. Embora dez desenvolvedores tenham iniciado, apenas sete concluíram todas as etapas. Os participantes, selecionados por conveniência, receberam tutoriais para nivelar habilidades e conhecimentos prévios. A análise focou em critérios como facilidade de implementação, simplicidade das configurações, documentação, suporte e quantidade de código, oferecendo uma visão qualitativa da experiência dos desenvolvedores.

Os objetivos da pesquisa incluem:

1. Analisar e comparar a facilidade de uso dos *frameworks* AWS SAM[1], Pulumi[2] e Serverless Framework[3] em um contexto prático, focando na experiência do desenvolvedor;

[1]https://aws.amazon.com/pt/serverless/sam/
[2]https://www.pulumi.com/
[3]https://www.serverless.com/

2. Comparar a complexidade de configuração inicial dos *frameworks*;
3. Avaliar a documentação e os recursos de suporte disponíveis para cada *framework*;
4. Medir a eficiência em termos de tempo de desenvolvimento e quantidade de código necessária para a implementação;
5. Avaliar a curva de aprendizado para desenvolvedores iniciantes;
6. Identificar os principais desafios enfrentados durante a implementação com cada *framework*;
7. Propor recomendações para a escolha do *framework* adequado com base nos resultados obtidos.

Esses objetivos são sustentados por pesquisas anteriores sobre o desenvolvimento *serverless*, que destacam a necessidade de uma análise prática e baseada em experiências diretas de uso [Adzic and Chatley 2017].

2. Visão Geral dos Frameworks Serverless

A seguir, discutimos as características dos três *frameworks serverless* selecionados para este estudo: AWS SAM, Serverless Framework e Pulumi. Cada um deles apresenta abordagens distintas para o desenvolvimento e a implantação de aplicações *serverless*, fornecendo opções variadas para diferentes cenários de projeto.

AWS SAM é um *framework* nativo da AWS que facilita a definição e a implantação de aplicações *serverless* por meio de *templates YAML*. Este *framework* é integrado ao ecossistema da AWS[4], oferecendo suporte para serviços como Lambda, API Gateway e DynamoDB [Outsource 2023]. AWS SAM incorporou funcionalidades que o tornam uma opção atraente para desenvolvedores imersos na AWS [Amazon 2024, Adamson 2024].

Serverless Framework é uma ferramenta de código aberto que permite o desenvolvimento e a implantação de aplicações *serverless* em múltiplas plataformas de nuvem, incluindo AWS, Google Cloud Platform[5] e Azure[6]. Este *framework* se destaca por sua comunidade de *plugins*, que permite estender suas funcionalidades conforme necessário [Bille 2023b, Inc. 2024].

Pulumi é um *framework* que permite a definição de infraestrutura utilizando linguagens de programação tradicionais como Python, JavaScript ou TypeScript, facilitando a integração entre o código da aplicação e a infraestrutura [Latreille 2022]. Embora ofereça suporte para múltiplas plataformas de nuvem, Pulumi pode apresentar limitações a funções *serverless*, especialmente em ambientes com uso intensivo de bibliotecas externas. Além disso, a configuração e a gestão de dependências podem exigir um esforço adicional em comparação com *frameworks* mais maduros [Bille 2023a, Corp. 2024].

Esses *frameworks* simplificam a criação e implantação de aplicações, além de proporcionar flexibilidade para lidar com as complexidades específicas de cada plataforma de nuvem [Jonas et al. 2019]. No entanto, estudos indicam que, apesar das vantagens associadas ao *serverless*, o modelo apresenta desafios como limitações de tempo de execução, gerenciamento de estado [Schleier-Smith et al. 2021]. Além disso, problemas como latência em *cold starts* e dificuldades na depuração são desafios adicionais que precisam ser considerados [Eismann et al. 2020].

[4]https://aws.amazon.com/pt/serverless/
[5]https://cloud.google.com/
[6]https://azure.microsoft.com/

3. Metodologia

Este estudo adota uma abordagem qualitativa baseada em um estudo de caso para comparar três *frameworks serverless*: AWS SAM, Serverless Framework e Pulumi. O foco é avaliar a perspectiva do desenvolvedor quanto à facilidade de uso e outros aspectos práticos desses *frameworks* ao implementar uma aplicação padrão de gestão de tarefas na plataforma AWS. Os dados foram coletados por meio de questionários aplicados após os participantes concluírem a implementação da *API* com cada *framework*. Possibilitando a análise qualitativa de suas experiências. A metodologia abrange a seleção dos *frameworks*, o desenvolvimento da aplicação, a configuração do ambiente de testes e a avaliação dos resultados.

Para a comparação, foi desenvolvida uma *RESTful API* de gestão de tarefas, que abrange operações básicas de manipulação de dados e persistência em banco de dados. A aplicação foi projetada utilizando *Python*, uma linguagem suportada por todos os *frameworks* selecionados. O desenvolvimento foi dividido nas seguintes etapas:

- **Configuração Inicial:** inclui a instalação e configuração do *framework*;
- **Desenvolvimento da API:** implementação das funcionalidades básicas da *API*;
- **Teste e Ajustes:** validação do comportamento da aplicação e ajustes necessários.

Os participantes desenvolveram individualmente a aplicação utilizando os três *frameworks serverless* avaliados, com o suporte de tutoriais detalhados fornecidos como parte do estudo.

Os testes foram realizados em um ambiente controlado na plataforma AWS, proporcionando um cenário consistente para a análise dos *frameworks*.

A avaliação dos *frameworks* baseou-se em critérios práticos e relevantes ao desenvolvimento de software, focando em fatores como facilidade de configuração e simplicidade do código [Adzic and Chatley 2017]. Os critérios, alinhados aos objetivos do estudo, foram aplicados por meio de questionários respondidos pelos participantes após o desenvolvimento do estudo de caso, abrangendo:

- **Facilidade de Implementação:** inclui a instalação, configuração inicial, clareza da documentação e a complexidade da definição da infraestrutura. Este critério é importante para avaliar a curva de aprendizado e a rapidez com que os desenvolvedores podem iniciar o uso do *framework*;
- **Simplicidade das Configurações:** avaliada pela facilidade na configuração dos arquivos de definição e ajustes necessários para funcionalidades básicas. A simplicidade na configuração contribui para reduzir erros e aumentar a eficiência no desenvolvimento;
- **Quantidade de Código Necessário:** envolve a análise da extensão do código escrito e a legibilidade do código gerado. Este critério permite avaliar se a solução é eficiente e fácil de manter, considerando que menos código geralmente implica em uma solução simplificada;
- **Desempenho da Aplicação:** analisa como cada *framework* impacta a performance das aplicações em termos de tempo de resposta e escalabilidade. O desempenho é um fator crítico para a experiência do usuário final e para a operação eficiente de sistemas em produção;

- **Documentação e Suporte:** verifica a qualidade e acessibilidade da documentação e suporte técnico. A boa documentação facilita a resolução de problemas e a adoção do *framework*, sendo um aspecto essencial para a experiência do desenvolvedor;
- **Impressão Geral:** inclui dificuldades encontradas, pontos positivos e a recomendação geral dos participantes sobre os *frameworks*. Este critério capta a percepção global dos desenvolvedores, abrangendo aspectos qualitativos não capturados pelos outros critérios.

A participação no estudo foi voluntária, e todos os participantes foram informados dos objetivos, emitindo seu consentimento. Ao todo, dez desenvolvedores aderiram, sem incentivo financeiro ou específico.

Considerações de validade foram identificadas e abordadas, conforme sugerido em literatura de experimentação em engenharia de software [Wohlin et al. 2012]:

- **Tamanho da Amostra:** A pesquisa contou com dez participantes, o que pode limitar a validade externa e a generalização dos resultados, já que amostras pequenas podem não representar uma população maiorr [Wohlin et al. 2012]. Esta limitação foi devido à natureza voluntária da participação, o que restringiu a quantidade de participantes disponíveis;
- **Experiência dos Usuários:** a variação na experiência dos participantes foi mitigada por meio de tutoriais de treinamento, buscando reduzir ameaças à validade interna, especialmente as relacionadas à inconsistência de habilidades entre os participantes [Wohlin et al. 2012]. a adoção desses tutoriais visa garantir que todos os participantes tivessem um nível de conhecimento mínimo antes de iniciar o estudo;
- **Exigências da Plataforma AWS:** a necessidade de criação de contas na AWS e fornecimento de informações de pagamento trouxe preocupações de validade de construto. Para mitigar essa ameaça, foram fornecidas orientações detalhadas sobre o uso do nível gratuito, reduzindo o risco de experiências distintas entre participantes devido a possíveis cobranças não intencionais [Wohlin et al. 2012];
- **Evolução Tecnológica:** a rápida evolução das tecnologias *serverless* pode influenciar a validade de conclusão e limitar a relevância temporal dos resultados [Wohlin et al. 2012]. Como solução, sugere-se que replicações futuras do estudo sejam conduzidas à medida que novas tecnologias ou versões dos *frameworks* sejam lançadas, permitindo comparações contínuas ao longo do tempo.

3.1. Análise Qualitativa dos Dados

As respostas dos questionários foram organizadas e tabuladas para permitir uma análise comparativa entre os *frameworks* avaliados. Em seguida, as frequências das respostas foram calculadas e representadas em gráficos.

4. Resultados

4.1. Facilidade de Instalação e Configuração

Conforme apresentado na Figura 1, a facilidade de instalação e configuração inicial foi majoritariamente classificada como “Neutra” (3 participantes) e “Difícil” (2 participantes). 1 participante avaliou como “Muito Difícil”. E apenas um participante considerou

a instalação e configuração “Fácil”. Não houve respostas indicando que a instalação foi “Muito Fácil”. Esses resultados indicam que a instalação e configuração inicial apresentaram um nível de dificuldade moderado, onde a maioria dos participantes não considerou o processo fácil, e alguns enfrentaram dificuldades importantes, como apontado por aqueles que o avaliaram como “Difícil”e “Muito Difícil”.

4.2. Utilidade dos Tutoriais e Documentação

A maioria dos participantes considerou os tutoriais fornecidos como “Úteis” (3 participantes) e “Muito Úteis” (2 participantes), e 2 como “Neutra”. A documentação também teve avaliações positivas, com 5 participantes a classificando como “Útil”e 1 participante indicou a resposta “Muito Útil”. No entanto, 1 participante indicou uma resposta “Neutra”, o que sugere que, embora a documentação atenda de modo geral, há espaço para melhorias na sua clareza e profundidade, Figura 1.

4.3. Definição de Infraestrutura e Recursos

A definição da infraestrutura e dos recursos necessários, como funções, *APIs* e bancos de dados, foi vista como desafiadora pela maioria. 3 participantes avaliaram essa etapa como “Difícil”, enquanto 4 responderam “Neutra”. Não houve avaliações indicando que essa etapa foi “Fácil”, conforme visualizado na Figura 1, refletindo uma percepção de complexidade no processo.

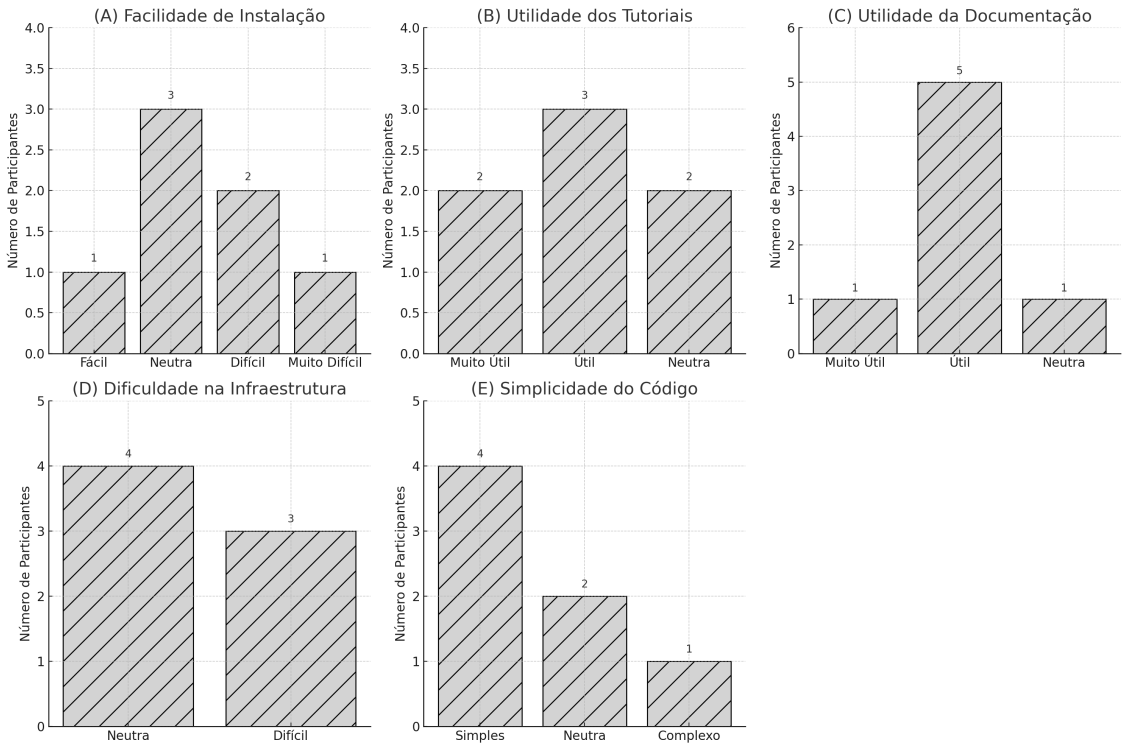

Figura 1. (A) Facilidade de Instalação e Configuração, (B) Utilidade dos Tutoriais, (C) Utilidade da Documentação, (D) Dificuldades na Infraestrutura e (E) Simplicidade do Código

4.4. Simplicidade do Código e Configurações

O código necessário para implementar as funcionalidades da aplicação foi considerado "Simples" por 4 participantes, enquanto 2 a classificaram como "Neutra" e 1 como "Complexa", Figura 1. De modo semelhante, a configuração dos arquivos (YAML, *serverless.yml*, código Pulumi) foi também percebida como "Simples" por 2 participantes, "Neutra" por 4 e 1 avaliou como "Complexa". Sobre as dificuldades ao ajustar as configurações para as funcionalidades básicas, 3 participantes disseram ter encontrado "Poucas Dificuldades", 3 relataram "Muitas Dificuldades", e 1 avaliou como "Neutro", conforme Figura 2.

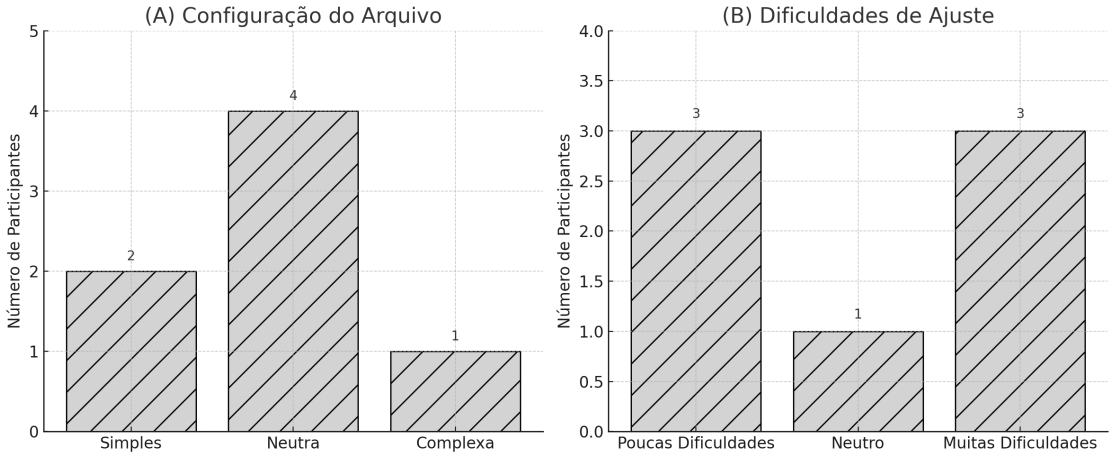

Figura 2. (A) Configuração do Arquivo, (B) Dificuldades de Ajustes nas Configurações

4.5. Avaliação da Quantidade de Código Necessário

A maioria dos participantes considerou a quantidade de código necessário para implementar as funcionalidades como "Concisa" (5 participantes), enquanto 2 a classificaram como "Neutra". Não houve respostas indicando que a quantidade de código foi "Extensa"ou "Muito Extensa". Em relação à legibilidade do código gerado, 5 participantes o consideraram "Neutro", enquanto 2 o consideraram "Legível", conforme mostrado na Figura 3. Esses resultados sugerem que o volume de código é geralmente visto como manejável, mas sua legibilidade apresenta oportunidades para melhorias.

4.6. Recomendações e Impressões Gerais

Entre os *frameworks* avaliados, a maioria dos participantes recomendou o Pulumi, destacando sua flexibilidade e suporte para múltiplas nuvens e diferentes linguagens de programação. O AWS SAM foi preferido por aqueles que já estavam familiarizados com o ecossistema AWS, devido à sua integração nativa com os serviços da AWS. O Serverless Framework foi apreciado por sua extensibilidade e variedade de *plugins*, sendo indicado por desenvolvedores que buscam flexibilidade.

Os principais desafios mencionados incluíram a complexidade inicial na configuração e a ausência de comentários explicativos no código de exemplo. Houve também dificuldades na configuração de credenciais na AWS, considerada uma etapa trabalhosa por alguns participantes. Esses pontos foram levantados ao longo do estudo e estão detalhados nas Figuras 1 e 2.

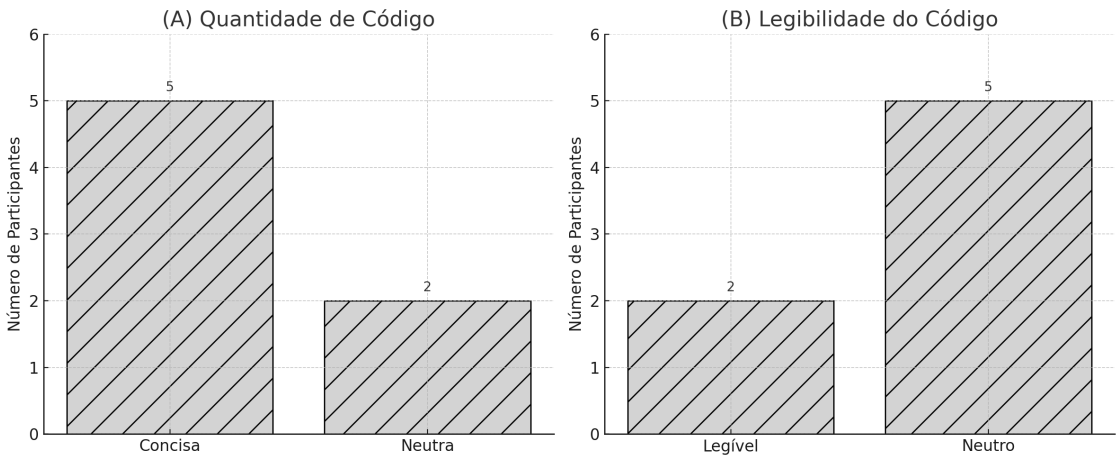

Figura 3. (A) Quantidade de Código, (B) Legibilidade do Código

5. Conclusão e Recomendações

Os resultados deste estudo sugerem que os *frameworks serverless* AWS SAM, Serverless Framework e Pulumi apresentam diferentes níveis de usabilidade e simplicidade no desenvolvimento de aplicações. Todos os *frameworks* foram considerados adequados em termos de simplicidade do código, no entanto, a configuração inicial e a definição da infraestrutura mostraram-se áreas desafiadoras para a maioria dos usuários.

Pulumi foi mencionado pelos participantes como uma solução com maior flexibilidade e suporte para múltiplas nuvens. Além disso, a facilidade de uso de Pulumi em diferentes linguagens de programação foi considerada um aspecto positivo. Participantes com experiência prévia no ecossistema AWS demonstraram preferência pelo AWS SAM devido à sua integração nativa com os serviços da AWS. Serverless Framework foi reconhecido por sua capacidade de extensão, dado o suporte a diversos *plugins* e ferramentas, o que o torna uma escolha viável para desenvolvedores que buscam versatilidade.

Com base nas observações feitas durante o estudo, as seguintes recomendações são propostas:

1. **Aprimoramento da Documentação:** sugere-se que os desenvolvedores dos *frameworks serverless* invistam em uma documentação mais clara e detalhada, com exemplos práticos que abordem tanto cenários comuns quanto situações mais complexas;
2. **Suporte Ampliado para Múltiplas Nuvens:** recomenda-se a ampliação do suporte para múltiplas nuvens e linguagens de programação, o que pode tornar os *frameworks* mais atrativos, especialmente para desenvolvedores que atuam em ambientes diversificados. Estudos anteriores indicam que o suporte a múltiplas nuvens pode aumentar a flexibilidade e a resiliência das arquiteturas *serverless* [Villamizar et al. 2016].

Embora os *frameworks serverless* analisados ofereçam uma base eficiente para o desenvolvimento de aplicações em nuvem, ainda há áreas a serem aprimoradas para melhorar a experiência do desenvolvedor e ampliar sua adoção. O uso dessas tecnologias pode otimizar a eficiência e reduzir custos, mas demanda adaptações na abordagem dos desenvolvedores [Hassan et al. 2021, Cavalheiro and Schepke 2023].

Ademais, este estudo possui limitações, como o número reduzido de participantes e a exclusividade à AWS, o que pode restringir a aplicabilidade dos resultados a outros contextos. Ainda assim, a flexibilidade do Serverless Framework e do Pulumi para múltiplos provedores sugere que esses resultados podem se estender a outros ambientes, como Google Cloud e Microsoft Azure. No entanto, a integração nativa pode variar entre provedores.

A questão de pesquisa abordada neste estudo, sobre como cada *framework* afeta a experiência do desenvolvedor e a eficácia em ambientes *serverless*, foi explorada com foco nos desafios práticos enfrentados e nas oportunidades de melhoria.

Para pesquisas futuras, recomenda-se:

- Expandir a amostra de participantes, incluindo desenvolvedores com diferentes níveis de experiência e que atuem em diversos contextos de desenvolvimento, a fim de obter uma compreensão das vantagens e desafios de cada *framework serverless*;
- Incorporar métricas quantitativas, como tempo de desenvolvimento, desempenho das aplicações e custo operacional, para fornecer uma avaliação mais detalhada e objetiva sobre a eficiência de cada *framework* em cenários reais;
- Explorar a implementação de outras aplicações, além da gestão de tarefas, para avaliar a versatilidade e a adequação dos *frameworks* em diferentes tipos de projetos e domínios;
- Investigar outros *frameworks serverless* e realizar comparações similares, com o objetivo de ampliar as opções disponíveis para desenvolvedores e auxiliar na escolha da ferramenta adequada para necessidades específicas;
- Desenvolver estudos de caso que examinem a adoção desses *frameworks* em ambientes de produção, abordando aspectos como manutenção a longo prazo, escalabilidade e impacto no ciclo de vida do *software*.

Este estudo comparativo oferece uma visão inicial das capacidades e limitações dos *frameworks serverless* AWS SAM, Serverless Framework e Pulumi. Além dos benefícios observados, também identificou desafios, especialmente na configuração inicial e na definição de infraestrutura. Com as recomendações apresentadas, futuras pesquisas poderão explorar mais a fundo essas ferramentas, apoiando o avanço das práticas *serverless* e sua adoção em larga escala.

Referências

Adamson, C. (2024). Serverless Architectures with AWS SAM (Serverless Application Model). Medium. Acesso em: 22 de agosto de 2024.

Adzic, G. and Chatley, R. (2017). Serverless Computing: Economic and Architectural Impact. *arXiv preprint arXiv:1706.05336*.

Amazon (2024). *AWS Serverless Application Model (AWS SAM)*. Amazon Web Services.

Baldini, I., Castro, P., Chang, K., Cheng, P., Fink, S., and Ishakian, V. (2017). Serverless Computing: Current Trends and Open Problems. *Research Advances in Cloud Computing*.

Bille, S. (2023a). Serverless Framework vs SAM vs AWS CDK. DEV Community. Acesso em: 22 de agosto de 2024.

Bille, S. (2023b). Serverless Frameworks for 2023. Elva Group Blog. Acesso em: 22 de agosto de 2024.

Cavalheiro, A. P. and Schepke, C. (2023). Exploring the Serverless First Strategy in Cloud Application Development. In *2023 International Symposium on Computer Architecture and High Performance Computing Workshops (SBAC-PADW)*, pages 89–94.

Cloud, F. (2024). Serverless Computing: O que é?

Corp., P. (2024). *Pulumi Documentation*. Pulumi Corp.

Eismann, S., Scheuner, J., Van Eyk, E., Schwinger, M., et al. (2020). Serverless applications: Why, when, and how? *IEEE Software*, 38(1):32–39.

FastAPI (2024). Fastapi documentation. Acesso em: 14 de outubro de 2024.

Hassan, H. B., Barakat, S. A., and Sarhan, Q. I. (2021). Survey on Serverless Computing. *Journal of Cloud Computing*, 10:1–29.

Hellerstein, J. M., Faleiro, J. M., Gonzalez, J. E., et al. (2018). Serverless Computing: One Step Forward, Two Steps Back. *arXiv preprint arXiv:1812.03651*.

Inc., S. (2024). Serverless Framework Documentation. `https://www.serverless.com/framework/docs/`.

Jonas, E., Schleier-Smith, J., Sreekanti, V., et al. (2019). Cloud programming simplified: A berkeley view on serverless computing. *arXiv preprint arXiv:1902.03383*.

Latreille, L. (2022). My Journey with Pulumi and the Serverless Framework. Level Up Coding. Acesso em: 22 de agosto de 2024.

Outsource, T. (2023). The Best Serverless Frameworks for Developers. TMS Outsource Blog. Acesso em: 22 de agosto de 2024.

Projects, P. (2024). Flask documentation. Acesso em: 14 de outubro de 2024.

RightScale (2020). Rightscale 2020 state of the cloud report. Acesso em: 14 de outubro de 2024.

Roberts, M. (2016). Serverless Architectures. *Mike Roberts Blog*.

Roberts, M. (2018). Serverless Architectures.

Schleier-Smith, J., Sreekanti, V., Khandelwal, A., et al. (2021). What serverless computing is and should become: The next phase of cloud computing. *Communications of the ACM*, 64(5):76–84.

TatvaSoft (2022). Top Serverless Frameworks for Creating Serverless Apps. TatvaSoft Blog. Acesso em: 22 de agosto de 2024.

Villamizar, M., Garcés, O., Ochoa, L., et al. (2016). Infrastructure Cost Comparison of Running Web Applications in the Cloud Using AWS Lambda and Monolithic and Microservice Architectures. In *Proceedings of the 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 179–182. IEEE.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., Wesslén, A., et al. (2012). *Experimentation in software engineering*, volume 236. Springer.