Terraform e CDK: Uma Análise Comparativa entre Tecnologias de Gerência de Infraestrutura

João Victor Tadeu Chaves Frois[1], **Lucas Gabriel Padrão Rezende**[1]

[1]Pontifícia Universidade Católica de Minas Gerais (PUC Minas)
30140-002 – Belo Horizonte – MG – Brasil

{jfrois,lgprezende}@sga.pucminas.br

Abstract. *Infrastructure as Code is a key concept in DevOps that involves automating infrastructure management processes through the use of code. Several tools support this objective, such as Terraform and CDK. In this context, the careful selection of the right tool is crucial for the success of a project. However, there is a lack of clarity about the circumstances in which developers should choose between each of these tools. Therefore, the purpose of this study is to compare the Terraform and CDK tools under four different aspects: abstraction, scalability, maintainability, and performance. As a result, we observe that each tool excels in specific scenarios. For example, Terraform is more suitable for teams with higher levels of experience, focused on rapid implementations. By contrast, CDK is more appropriate for teams with less experience and projects that focus on saving computational resources during implementations.*

Resumo. *Infraestrutura como código é um conceito chave do DevOps que envolve a automação de processos de gerenciamento de infraestrutura por meio do uso de código. Existem diferentes ferramentas que auxiliam esse objetivo, como o Terraform e o CDK. Nesse contexto, a seleção da ferramenta adequada é crucial para o sucesso de um projeto. Contudo, ainda não estão claros quais os principais aspectos que levam os desenvolvedores a escolherem entre cada uma dessas ferramentas. Dessa forma, o propósito deste estudo é comparar as ferramentas Terraform e CDK sob quatro diferentes perspectivas: abstração, escalabilidade, manutenibilidade e desempenho. Como resultado, observa-se que cada ferramenta se destaca em cenários específicos. Por exemplo, o Terraform se destaca em equipes com maior nível de experiência e com foco em implementações rápidas. Por outro lado, o CDK é mais recomendado para equipes com menor experiência e com foco em economizar recursos computacionais durante as implementações.*

Bacharelado em Engenharia de *Software* - PUC Minas
Trabalho de Conclusão de Curso (TCC)

Orientador de conteúdo (TCC I): Cleiton Tavares - cleitontavares@pucminas.br
Orientador acadêmico (TCC I): Laerte Xavier - laertexavier@pucminas.br
Orientador do TCC II: Laerte Xavier - laertexavier@pucminas.br
Coorientador do TCC II: Johnatan Alves de Oliveira - johnatan.oliveira@pucminas.br

Belo Horizonte, 26 de novembro de 2023.

1. Introdução

A cultura *DevOps* engloba um conjunto de práticas e técnicas que visam a entrega rápida e confiável de *software* [Kim et al. 2018]. A automação, um dos pilares dessa cultura, propõe a infraestrutura como código (IaC, do inglês *Infrastructure as Code)*, uma técnica de gerenciamento de infraestruturas em que seus recursos são definidos e gerenciados via arquivos de configuração e códigos [Parnin et al. 2019]. Nesse contexto, a gestão de IaC se encaixa no contexto de *DevOps* como uma prática fundamental para a entrega rápida e confiável de *software* em ambientes de *cloud* por meio da automação de seus *scripts* [Patni et al. 2020]. A gestão desses ambientes na Engenharia de *Software* se tornou relevante devido à necessidade de eficiência e escalabilidade dos recursos [Chappell 2008].

Estudos anteriores avaliam o uso das IaC no processo de desenvolvimento de software [Chappell 2008, Alonso et al. 2023]. Por exemplo, eles evidenciam *code smells* inseridos durante o desenvolvimento de *scripts* de IaC [Alonso et al. 2023] e abordam a utilização das ferramentas de IaC para facilitar a implantação de um modelo arquitetural [Sandobalín et al. 2020]. Além desses estudos, ferramentas para gestão de infraestrutura também foram propostas [Rahman et al. 2019]. Contudo, poucos estudos se propõem a comparar a aplicabilidade de diferentes ferramentas e identificar seus cenários de uso recomendados. Sendo assim, **o problema tratado neste trabalho é a escassez de clareza sobre a eficiência de diferentes ferramentas de suporte para o gerenciamento de infraestruturas como código**.

A escolha das ferramentas de gerenciamento de IaC é tão crucial quanto a seleção de linguagens de programação em contextos específicos. Assim como há estudos comparativos entre linguagens de programação [Bogner and Merkel 2022, Ray et al. 2014], é essencial realizar estudos comparativos entre ferramentas de provisionamento e gerenciamento de infraestrutura. Nesse sentido, torna-se relevante a comparação entre ferramentas de gestão de IaC como o Terraform, ferramenta mais eficiente para implantações *Multcloud* [Gupta et al. 2021], e o *AWS Cloud Development Kit* (CDK), que abstrai o gerenciamento de infraestruturas na principal *cloud*, *Amazon Web Services* (AWS), utilizada atualmente [Vogels 2023]. Portanto, resolver o problema identificado neste trabalho é fundamental para que as equipes de desenvolvimento possam tomar decisões mais assertivas e garantir o sucesso dos projetos na *cloud*.

Tendo em vista o contexto apresentado, o objetivo deste trabalho é **avaliar a eficiência do Terraform e do AWS CDK por meio de uma análise comparativa**. Para atingir esse objetivo, são propostos os seguintes objetivos específicos: (i) avaliar a capacidade de abstração do CDK e Terraform para criação de infraestruturas a partir de desenvolvedores com menos experiência; (ii) avaliar, a partir da visão de desenvolvedores experientes, a escalabilidade e manutenibilidade do Terraform e do CDK; e (iii) avaliar o desempenho do Terraform e do CDK na criação de recursos de infraestrutura na AWS.

Este trabalho apresenta os resultados de uma análise comparativa entre Terraform e CDK, caracterizando suas diferenças em relação à: capacidade de abstração, escalabilidade, manutenibilidade e desempenho. Com isso, são identificados os cenários de uso em que cada ferramenta se destaca. Por exemplo, o Terraform mostrou-se eficaz para equipes experientes que buscam implementações rápidas. Por outro lado, o CDK é mais recomendado para equipes menos experientes que visam economizar recursos computacionais durante as implementações.

O restante deste trabalho está dividido em 7 seções. As Seções 2 e 3 apresentam a fundamentação teórica e os principais trabalhos relacionados, respectivamente. A Seção 4 detalha os materiais e métodos. As Seções 5 e 6 apresentam os resultados e as ameaças à validade, respectivamente. Por fim, a Seção 7 apresenta as conclusões e trabalhos futuros.

2. Fundamentação Teórica

Nesta seção são detalhados os principais conceitos necessários para o entendimento do estudo: *DevOps* e IaC.

2.1. DevOps

O conceito de *DevOps* está relacionado à construção de uma cultura colaborativa entre equipes que são historicamente separadas em: equipes de desenvolvimento e equipes de operações [Loukides 2012]. Essa abordagem foi criada por P. Debois em 2008 na *Agile Conference*, onde foram propostas ideias e processos que visavam criar uma cultura de integração e colaboração entre os times, com o intuito de prover agilidade às demandas de clientes e solucionar empecilhos históricos entre essas equipes [Debois 2008].

Os times de desenvolvimento eram muitas vezes forçados a lançar novas versões de forma rápida, devido a prazos apertados, acarretando que questões relacionadas à manutenibilidade ou estabilidade não fossem priorizadas. Já os times de operações, responsáveis por disponibilizar as novas versões, não conseguiam garantir a estabilidade e a execução dos produtos, o que ocasionava a criação de processos formais complicados que resultavam em menor colaboração entre as equipes [Dörnenburg 2018].

Sendo assim, a cultura proposta por Debois atua como um facilitador, principalmente no contexto de metodologias ágeis, graças ao foco em entregas contínuas de forma rápida, segura e colaborativa [Kim et al. 2018]. Dessa forma, o *DevOps* é uma resposta ao mercado ao possibilitar a implementação de mudanças contínuas em um *software*, possibilitando que os produtos sejam fluidos e passíveis de adaptações dentro de seu ciclo de desenvolvimento [Schlossnagle 2017]. Como consequência, o *software* desenvolvido é implantado rapidamente, de maneira confiável [Mala 2019].

2.2. Infraestrutura como Código

Infraestrutura como código é um conceito chave do *DevOps* que envolve a automação de processos de gerenciamento de infraestrutura por meio do uso de código [Ibrahim et al. 2022]. Na prática, significa que as configurações de infraestrutura, incluindo servidores, redes, bancos de dados, *firewalls* e outros componentes, são definidas e gerenciadas por meio de código. Essa abordagem permite que as equipes de *DevOps* gerenciem as infraestruturas de maneira mais eficiente e escalável, eliminando a necessidade de intervenções manuais e reduzindo a probabilidade de erros humanos [Dalla Palma et al. 2022]. Ao codificar a infraestrutura, as configurações podem ser versionadas, rastreadas e facilmente reproduzidas, o que significa que as equipes podem implementar atualizações e correções com maior rapidez e confiança.

Além disso, a IaC permite que as equipes de *DevOps* criem ambientes de desenvolvimento e teste mais facilmente, o que ajuda a acelerar o ciclo de desenvolvimento. A infraestrutura como código também oferece maior transparência e colaboração entre as equipes de desenvolvimento e operações. Isso ocorre porque os desenvolvedores podem

entender melhor a infraestrutura subjacente e as operações podem se envolver mais cedo no processo de desenvolvimento [Sandobalín et al. 2020]. Como exemplos de plataformas de IaC, destacam-se o Terraform[1] e o CDK[2].

O Terraform é uma ferramenta de orquestração de ambientes *cloud* desenvolvido pela Hashi Corp. Seu principal objetivo é aprimorar o fluxo de implantação de recursos na *cloud* por meio de seus módulos. Nesse aspecto, destaca-se que, para cada serviço da *cloud*, há um módulo específico a ser utilizado para criação de seus recursos [Gupta et al. 2021]. Além disso, o Terraform possui um recurso de planejamento que detalha as mudanças que devem ser implementadas na infraestrutura com base no código que será executado [HashiCorp 2023]. Por fim, destaca-se que o fluxo de trabalho do Terraform consiste em três estágios: Gravar, Planejar e Aplicar.

O CDK oferece uma abstração de alto nível para o *AWS CloudFormation*, permitindo que desenvolvedores usem linguagens de programação populares, como *TypeScript*, *JavaScript*, *Python*, *Java*, *.NET* e *Go*, para definir e implantar infraestrutura na AWS [Vogels 2023]. Essa abstração de alto nível possibilita que equipes de desenvolvimento criem modelos de infraestrutura de forma rápida e com menos esforço, fornecendo flexibilidade e controle para personalizar a implantação conforme as necessidades específicas do projeto [Coulter 2023]. Além disso, a CDK possui uma estrutura de teste incorporada, que permite que os desenvolvedores simulem a implantação, validem os recursos de infraestrutura e executem testes automatizados, garantindo que as definições de infraestrutura estejam funcionando corretamente antes da implantação na AWS [Lovchikova 2023].

3. Trabalhos Relacionados

Os trabalhos relacionados discutidos nesta seção incluem a implementação de recursos em *cloud* de forma dinâmica e ágil, testes para garantir a confiabilidade e a mensuração da qualidade de IaC.

Sandoval et al. (2020) comparam duas abordagens para a implementação de IaC: a *model-driven* e a *code-centric*. Para isso, eles utilizam as ferramentas *Argon* e *Ansible* (outras alternativas de ferramentas de IaC), avaliando cada uma delas em termos de facilidade de uso, eficiência e eficácia. Nesse contexto, os autores buscam resolver o problema da falta de informações sobre a comparação do desempenho das ferramentas. Diante disso, os resultados indicam que não há uma abordagem superior, sendo a escolha dependente das necessidades específicas do projeto. Em contraste, o presente estudo se diferencia ao empregar ferramentas distintas, CDK e Terraform, e ao abordar outras perspectivas, incluindo abstração, escalabilidade, manutenibilidade e desempenho.

Ainda nesse contexto, Dalla Palma et al. (2020) identificam e catalogam métricas de qualidade de software em IaC construídas utilizando *Ansible*. Os autores definiram 46 métricas agrupadas em 3 categorias distintas: (i) métricas orientadas a objetos portáveis para *Ansible* e IaC, (ii) métricas investigadas em trabalhos anteriores sobre IaC portáveis, para o *Ansible*, e (iii) métricas relacionadas às melhores e piores práticas no *Ansible*. Os resultados evidenciaram que as métricas empregadas na avaliação da qualidade do código de software podem ser aplicadas com sucesso na avaliação da qualidade de IaC, incluindo

[1]Página oficial do Terraform: developer.hashicorp.com/terraform
[2]Página oficial CDK: docs.aws.amazon.com/cdk/v2/guide/home.html

a identificação de *smells*. Portanto, embora tenha sido criado para o *Ansible*, esse estudo serve como referência para as conclusões deste trabalho ao exemplificar a avaliação de métricas relacionadas ao IaC.

Gupta et al. (2021) têm como intuito avaliar a eficácia do uso das ferramentas *Ansible* e Terraform na implantação da arquitetura *Hadoop* em um ambiente em *cloud*. Para isso, os autores implementaram *scripts* de automação utilizando as ferramentas de IaC, além de realizar testes de carga para avaliar a escalabilidade e o desempenho da arquitetura implantada. Os resultados mostraram que as tecnologias utilizadas possibilitaram a implantação rápida, eficiente e escalável da arquitetura *Hadoop*. Desse modo, embora essa pesquisa aborde contextos arquiteturais diferentes, é relevante para o artigo em questão, ao fornecer informações e práticas importantes sobre o gerenciamento de infraestrutura, apresentando diferentes abordagens e ferramentas que podem ser úteis. Dessa forma, o estudo conduzido pelos autores se diferencia deste trabalho por não comparar ferramentas de infraestrutura distintas, mas utilizá-las em conjunto.

Sokolowski e Salvaneschi (2023) discutem as melhores práticas para garantir a confiabilidade das infraestruturas. Para garantir a confiabilidade, os autores provêm o uso de uma ferramenta de teste automatizada, o *ProTI* que é baseada em testes de unidade. Essa ferramenta é um *fuzzer*, para programas em Pulumi[3], que adota técnicas modernas para testar programas IaC em diferentes configurações. Os resultados evidenciaram uma diminuição da carga de trabalho manual e um aumento da confiança nos resultados, proporcionando *feedback* rápido e abrangente sobre as infraestruturas. A relação entre esse trabalho e o presente estudo reside no processo de automatização das infraestruturas. Porém, neste trabalho são utilizadas as ferramentas CDK e Terraform, ao invés do Pulumi.

Parnin et al. (2019) auxiliam na prevenção de práticas inseguras de codificação em *scripts* de IaC. Os autores, por meio da análise de 1.726 *scripts* de IaC, identificaram sete *security smells* e desenvolveram a ferramenta *Security Linter for Infrastructure as Code* (SLIC) para realizar análise estática. Os resultados revelaram a presença de 21.201 ocorrências de *security smells*, incluindo 1.326 senhas diretamente codificadas no código. Essas descobertas podem melhorar a qualidade do código, mitigar vulnerabilidades e garantir a eficiência e segurança na gestão da infraestrutura. Essas descobertas oferecem *insights* importantes para a pesquisa atual na detecção de práticas inseguras de codificação em *scripts* de IaC, servindo como um referencial para a criação de padrões sólidos de IaC. Por outro lado, esse estudo se diferencia do presente trabalho ao ter foco na qualidade da infraestrutura gerada e na metrificação de *code smells*, enquanto o trabalho atual enfatiza a comparação da eficiência entre as ferramentas de infraestrutura, sob quatro aspectos distintos: abstração, escalabilidade, manutenibilidade e desempenho.

4. Materiais e Métodos

Este trabalho adota abordagens qualitativas e quantitativas. Para tanto, são realizados dois experimentos controlados, para aplicar intervenções específicas e coletar métricas quantitativas, como o tempo de execução e o consumo computacional das ferramentas. Também é realizada uma entrevista com especialistas em IaC, com o intuito de explorar as

[3]Plataforma de automação de IaC que permite que os usuários criem, implantem e gerenciem infraestrutura em várias nuvens usando linguagens de programação de abstrações de *cloud*

ferramentas avaliadas. O principal objetivo dos métodos utilizados é descrever e comparar as ferramentas com base em quatro aspectos: capacidade de abstração, escalabilidade, manutenibilidade e desempenho.

Especificamente, a comparação entre Terraform e CDK envolve métodos que combinam dados qualitativos e quantitativos. Experimentos práticos são conduzidos com casos de uso reais, permitindo a avaliação da eficiência das ferramentas em situações do mundo real. Esses métodos buscam identificar as diferenças entre as ferramentas e avaliar qual delas é mais eficiente em cada aspecto. Essa combinação de métodos possibilita a análise da eficiência das ferramentas, sendo essencial para tomar decisões informadas sobre a escolha mais adequada para diferentes cenários.

4.1. Avaliação dos Objetivos

Com o intuito de alcançar os objetivos específicos definidos, este estudo avalia as perguntas de pesquisa (RQ, do inglês *Research Question*). São elas:

- **RQ1:** Como o CDK e o Terraform se comparam em relação à capacidade de abstração na criação de IaC's?
- **RQ2:** Como o Terraform e o CDK se comparam em relação à escalabilidade e manutenibilidade?
- **RQ3:** Qual ferramenta possibilita a criação de recursos de infraestrutura na AWS com menor tempo de execução?
- **RQ4:** Qual ferramenta consome mais recursos computacionais, como CPU e memória, durante a criação de recursos?

A RQ1 atende ao primeiro objetivo específico, que visa avaliar a capacidade de abstração das ferramentas em relação a desenvolvedores com menor experiência. Para tanto, utiliza-se o experimento descrito na Seção 4.2. A RQ2 aborda o segundo objetivo específico, que visa avaliar aspectos referentes a escalabilidade e manutenibilidade a partir de *insights* de especialistas da área (estudo descrito na Seção 4.3). Por fim, as RQs 3 e 4 relacionam-se com o terceiro objetivo específico, que visa avaliar o desempenho das ferramentas na criação de recursos na AWS. Para tanto, utiliza-se do experimento descrito na Seção 4.4.

4.2. Estudo 1: Avaliação da Abstração das Ferramentas

Nesta seção, apresenta-se o experimento adotado para avaliar a capacidade de abstração do CDK e do Terraform na criação de infraestruturas por desenvolvedores novatos. Particularmente, busca-se responder a RQ1. Para tanto, o experimento contou com a participação de 30 alunos do sexto período do curso de Engenharia de Software da PUC Minas. O experimento contou com três etapas principais:

1) Seleção dos Participantes: Foram selecionados alunos do curso de Engenharia de Software da PUC Minas, com até o quarto período finalizado. Esse critério foi adotado visto que, até essa etapa do curso, os alunos já possuem os conhecimentos necessários para compreender as ferramentas avaliadas neste estudo.

2) Aula Introdutória: Os alunos foram separados aleatoriamente, divididos em dois grupos, cada um recebendo uma aula sobre Terraform ou CDK. Essas aulas abrangem conceitos fundamentais, sintaxe básica e funcionalidades específicas. Os materiais utilizados para as aulas encontram-se disponíveis no pacote de replicação.

3) Aplicação de Questionário:É aplicado um questionário via Microsoft Forms, composto por questões demográficas, questões sobre as ferramentas (divididas em fáceis, médias e difíceis) e um campo de *Feedback*. Os questionários encontram-se no Apêndice A.

Para avaliar previamente o método proposto, um estudo piloto, em setembro de 2023, com 23 alunos do 5º período foi conduzido. Esse grupo não participou do estudo principal, mas seus *feedbacks* foram utilizados para aprimorar as aulas e os questionários. Após a realização do estudo piloto, foi observada a necessidade de se alterar os trechos de códigos para imagens, dificultando consultas externas que pudessem prejudicar os resultados. Também foi identificada a necessidade de, durante as aulas introdutórias, trabalhar com mais detalhes alguns conceitos, como o fluxo de implementação de cada ferramenta. Essas alterações foram realizadas com base nos resultados do experimento, avaliando as pontuações obtidas pelos alunos e os comentários coletados na seção de *feedback*.

Para investigar diferenças estatísticas nas respostas dos alunos quanto às ferramentas CDK e Terraform, realiza-se o teste de Mann-Whitney [McKnight and Najab 2022]. Esse teste trata-se de um teste não paramétrico que compara a mediana das amostras, permitindo inferir se há diferenças estatísticas significativas entre duas amostras de dados. Adicionalmente, foi utilizado nos testes um nível de significância de 0,05.

4.3. Estudo 2: Avaliação da Escalabilidade e Manutenibilidade das Ferramentas

Nesta seção, são descritas as entrevistas estruturadas, realizadas com três especialistas para avaliar a escalabilidade e manutenibilidade do Terraform e CDK em projetos de IaC. Os especialistas, escolhidos pela rede de contatos dos autores, possuem experiência relevante em ambas as ferramentas. Este método fornece *insights* especializados importantes para a pesquisa em IaC. Esses resultados são utilizados para responder a RQ2.

Um roteiro de entrevista detalhado foi desenvolvido previamente, focando em escalabilidade e manutenibilidade, com perguntas para capturar as visões dos especialistas. Para analisar a escalabilidade definiu-se a seguinte pergunta: *Como você avaliaria a escalabilidade do Terraform e CDK em projetos de infraestrutura de grande porte? Quais são os principais desafios*? Por outro lado, para avaliar a manutenibilidade, definiu-se: *Em termos de manutenibilidade, quais são as práticas recomendadas para organizar e gerenciar código Terraform ao longo do tempo? E para o CDK?* O roteiro da entrevista se encontra no Apêndice B. As entrevistas realizadas em outubro de 2023 via Google Meets permitiram interação direta e gravação para análise posterior, conforme disponibilidade dos participantes. As respostas das entrevistas foram submetidas a uma análise qualitativa para identificar tendências e padrões relacionados às ferramentas estudadas. Os participantes consentiram o uso anônimo dos dados e tinham liberdade para interromper a entrevista a qualquer momento. Cada entrevista durou em média cerca de 40 minutos.

4.4. Estudo 3: Avaliação do Desempenho das Ferramentas na Criação de Recursos

Para avaliar o desempenho do Terraform e do CDK na criação de recursos de infraestrutura na AWS, define-se um contexto que englobe as interações entre serviços e recursos, bem como desafios e limitações típicos do gerenciamento de IaC. A partir desse cenário, são criados *scripts*, com ambas as ferramentas, para provisionar recursos na AWS que se encaixem dentro do *Free Tier*, plano gratuito da AWS. A Tabela 1 apresenta os recursos utilizados neste estudo. Os *scripts* utilizados para implementar os recursos na AWS

encontram-se disponíveis no pacote de replicação. Os resultados desse experimento respondem às RQs 3 e 4.

Tabela 1. Exemplos de recursos da AWS.

Nome do Serviço	Descrição do Serviço	Sigla do Serviço
Elastic Compute Cloud	Máquinas virtuais escaláveis na nuvem.	EC2
Simple Storage Service	Armazenamento de objetos escaláveis e duráveis.	S3
Elastic Container Registry	Registro de contêineres para o ECS.	ECR
Relational Database Service	Serviço de banco de dados relacional gerenciado.	RDS
Simple Notification Service	Serviço de mensagens para aplicativos distribuídos.	SNS
Virtual Private Cloud	Rede isolada na nuvem para hospedar recursos.	VPC

Os *scripts* são desenvolvidos com base nas estruturas e módulos padrões recomendados pelas fontes oficiais de cada ferramenta[4], assegurando, assim, uma avaliação justa e imparcial. Para garantir a replicabilidade do experimento e que ele possa ser executado de forma gratuita, algumas modificações foram realizadas nas características dos recursos, evitando que as configurações excedam o *Freetier* da AWS. Por exemplo, foi necessário ajustar o tipo de instância do EC2 para *T2.micro*. Além disso, para manter um ambiente controlado, planeja-se a montagem do mesmo por meio de código, utilizando cinco instâncias EC2 para executar os casos de teste configurados. A Tabela 2 apresenta a configuração das instâncias utilizadas neste experimento.

Tabela 2. Especificações das Instâncias EC2

Tipo de Instância	T2.micro
Sistema Operacional	Ubuntu 20.04
CPU	1
RAM (GiB)	1,0

Durante o experimento, são coletadas as seguintes métricas: 1) Uso de recurso computacional, que busca avaliar a porcentagem de uso da memória e CPU, possibilitando a avaliação do impacto das ferramentas nos componentes computacionais. 2) Tempo de execução, calculado a partir da média de tempo de cinco execuções para criar o elemento de infraestrutura, registrados em décimos de segundo. Destaca-se que a quantidade de execuções busca evitar medições incorretas dos recursos computacionais, permitindo avaliar a eficiência de cada ferramenta na criação de recursos de IaC.

5. Resultados

Esta seção apresenta os principais resultados obtidos durante a condução do presente estudo. Para tanto, apresenta-se a caracterização dos objetos de estudo, bem como a análise do comportamento das métricas examinadas.

5.1. Estudo 1: Avaliação da Abstração das Ferramentas

Com o intuito de avaliar a capacidade de abstração das ferramentas de IaC, foi conduzido um experimento com desenvolvedores iniciantes. Particularmente, busca-se responder à

[4]registry.terraform.io/providers/hashicorp/aws/latest/docs e https://docs.aws.amazon.com/cdk/api/v2/

primeira questão de pesquisa: *Como o CDK e o Terraform se comparam em relação à capacidade abstração na criação de IaC's?*

Para realização do experimento, foram selecionados 30 alunos do sexto período do curso de Engenharia de Software da PUC Minas. Desses, observou-se que 21 não tinham experiência prévia com IaC. Quanto à experiência profissional, apenas quatro alunos tinham menos de um ano de experiência; 18 tinham de 1 a 3 anos; cinco possuíam mais de 3 anos; e três não tinham experiência na área de tecnologia. Os 16 alunos do grupo do Terraform avaliaram sua familiaridade com as ferramentas como 1, numa escala de 1 a 5, indicando conhecimento limitado. No grupo do CDK, 3 dos 14 alunos classificaram seu nível de familiaridade como 2. Essa uniformidade nas avaliações iniciais aponta para níveis de conhecimento similares entre os estudantes em relação às ferramentas antes do experimento.

A Figura 1 apresenta os resultados obtidos acerca do desempenho dos 30 participantes do primeiro experimento. Na Figura 1(a), observa-se que as notas obtidas pelos participantes do Terraform variam de maneira significativa, com a maior nota alcançando 8 e a menor nota registrando 2, resultando em uma média das notas de 4,56. A mediana das notas foi de 4. Por outro lado, na Figura 1(b), destaca-se uma variação relevante nas notas dos participantes do CDK, com a maior nota alcançando 9 e a menor nota igual a 3, resultando em uma média das notas igual a 6. A mediana é igual a 6.

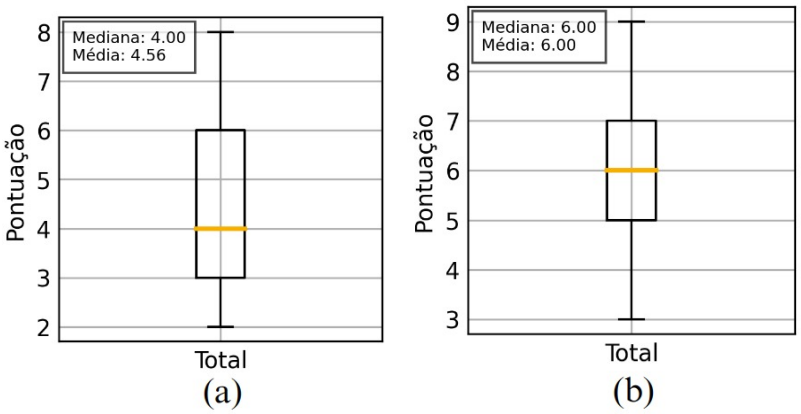


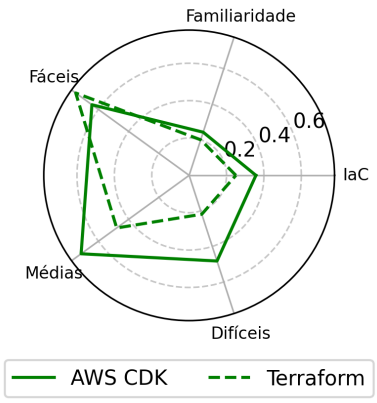
Figura 1. Distribuição das notas nos questionários.

Esses resultados sugerem que, embora o grupo que utilizou o CDK tenha obtido, em média, notas mais altas, a variação nas notas indica uma dispersão maior de desempenho. Por outro lado, o grupo que utilizou o Terraform apresentou notas mais próximas da média, sugerindo uma consistência maior no desempenho. No teste aplicado à pontuação total, a estatística de Mann-Whitney foi de 64, e o valor-p obtido foi aproximadamente 0.048. Dado que esse valor é menor que o limiar convencional de 0.05, conclui-se que há uma sutil diferença entre a mediana das notas totais dos alunos para cada ferramenta.

Para melhor avaliar o desempenho dos participantes, foi medido o percentual de acertos por nível de dificuldade. Na Figura 2, é possível observar as relações entre a familiaridade média dos participantes com a respectiva ferramenta e com IaC no geral, relacionando-se com seu percentual de acertos em cada nível. Em relação ao Terraform,

uma análise aprofundada das respostas revela padrões distintos acerca das questões classificadas como fáceis, médias e difíceis. Nas questões classificadas como fáceis, os participantes obtiveram uma taxa de acerto de 75%, demonstrando um bom domínio dos conceitos teóricos relacionados ao Terraform. Entretanto, nas questões classificadas como médias, a taxa de acerto foi de 47.9%, sugerindo um desafio adicional para os alunos. Já nas questões classificadas como difíceis, a taxa de acerto foi ainda menor, chegando a 21.9%.

Uma análise detalhada das respostas relacionadas ao CDK revela padrões em relação às questões classificadas como fáceis, médias e difíceis. Observou-se que, nas questões classificadas como fáceis, a porcentagem de acertos dos participantes foi de 64.3%. No caso das questões classificadas como médias, houve uma porcentagem mais elevada de acertos, atingindo 71.4%. Por outro lado, nas questões classificadas como difíceis, a porcentagem de acerto foi de 48.2%.



	CDK	Terraform
Fáceis	0.643	0.75
Médias	0.714	0.479
Difíceis	0.482	0.219

Figura 2. Comparação de Desempenho entre as ferramentas em Porcentagem

Nos testes aplicados às questões de dificuldade fácil, a estatística de Mann-Whitney foi de 85.0, e o valor-p obtido foi aproximadamente 0.0695. Dado que este valor excede o limiar convencional de 0.05, conclui-se que não há diferenças estatisticamente significativas nas notas dos alunos entre os dois grupos estudados (CDK e Terraform) para as questões fáceis. Essa ausência de superioridade de um grupo sobre o outro também se manteve nas questões de dificuldade média, cujo valor-p foi de 0.0677. Entretanto, uma observação distinta foi notada nas questões difíceis. Nessa categoria, o valor-p foi significativamente menor que 0.05 (0.0028), indicando uma diferença estatisticamente significativa entre as notas dos alunos para as ferramentas CDK e Terraform. Esse resultado sugere uma variação relevante no desempenho dos alunos entre os dois grupos nas questões de maior dificuldade, com um nível de confiança de 95%.

Os resultados apontam ainda que os alunos apresentaram um bom desempenho nas questões consideradas fáceis, o que reflete uma compreensão eficaz dos conceitos teóricos relacionados ao Terraform. No entanto, as questões de nível médio e difícil mostraram um desempenho relativamente inferior, indicando desafios na aplicação prática de conhecimentos mais avançados e complexos relacionados à ferramenta, bem como na compreensão de sua sintaxe. Isso é especialmente relevante para desenvolvedores

iniciantes, pois sugere a necessidade de foco adicional na prática e na familiarização com aspectos mais complexos do Terraform para um domínio completo da ferramenta.

Além disso, os resultados sugerem que os participantes tiveram um desempenho sólido nas questões consideradas fáceis e médias, demonstrando uma compreensão efetiva dos conceitos relacionados ao CDK. No entanto, nas questões classificadas como difíceis, a taxa de acerto foi ligeiramente inferior, indicando um desafio maior no entendimento e aplicação de conhecimentos mais complexos relacionados à ferramenta. Observa-se ainda que as questões de nível médio tiveram um desempenho especialmente bom, indicando uma aptidão dos participantes para compreender e aplicar os conceitos intermediários do CDK.

Considerando a hipótese de que o CDK busca proporcionar um nível mais elevado de abstração na criação de IaC e sua relação com a facilidade de aprendizado, os resultados parecem indicar uma curva de aprendizado menos íngreme em comparação com o Terraform. Os resultados comparativos sugerem que o Terraform é mais acessível em termos conceituais, mas, à medida que a complexidade aumenta, especialmente em questões práticas relacionadas à aplicabilidade das ferramentas, o CDK se torna mais compreensível devido à sua estrutura programática, tornando a aplicação mais clara. Portanto, quanto ao nível de complexidade, o Terraform pode ser melhor em tarefas mais simples, enquanto o CDK oferece uma curva de aprendizado geralmente mais suave, demandando menos tempo para ser compreendido, o que condiz com o objetivo da ferramenta de abstrair a complexidade e os recursos para facilitar a aplicação prática.

5.2. Estudo 2: Avaliação da Escalabilidade e Manutenibilidade das Ferramentas

Com o intuito de avaliar a escalabilidade e manutenibilidade das ferramentas, foram conduzidas entrevistas com três especialistas, com experiência significativa e certificados pela AWS. As entrevistas foram realizadas via Google Meets e gravadas para uso do estudo, com a autorização de todos os envolvidos. Este estudo visa responder à segunda questão de pesquisa: *Como o Terraform e o CDK se comparam em relação à escalabilidade e manutenibilidade?*

Particularmente, a entrevista conduzida busca entender como as ferramentas de IaC lidam com projetos complexos, tanto em termos de escalabilidade (capacidade de gerenciar infraestruturas em expansão), quanto de manutenibilidade (facilidade de modificar, atualizar e entender o código de infraestrutura ao longo do tempo). Além disso, busca-se obter percepções de especialistas na área, para comparar e avaliar esses dois aspectos críticos, identificando tendências, padrões e características distintas de cada ferramenta em relação à escalabilidade e manutenibilidade em cenários de grande porte. A seguir, são apresentadas as principais conclusões e tendências observadas nas respostas de especialistas:

Vantagens e desvantagens do Terraform: os resultados mostraram que os especialistas consideram que a ferramenta permite a configuração de ambientes híbridos (utilizando diferentes provedores de nuvem), mas apresenta limitações em relação ao gerenciamento do estado e à necessidade de estratégias para trabalho em equipe. Por exemplo, o Entrevistado 1 apontou:

"A principal vantagem do Terraform é a compatibilidade com várias nuvens. A questão do State é um desafio em times grandes."

Vantagens e desvantagens do CDK: os resultados mostraram que os especialistas avaliam que a ferramenta apresenta facilidade no desenvolvimento de recursos para a AWS, principalmente para desenvolvedores iniciantes. Contudo, possui a limitação herdada do CloudFormation na leitura de estados. Por exemplo, o Entrevistado 3 argumentou:

"O CDK é vantajoso para quem está começando, mas herda desafios do Cloud Formation na leitura de estados."

Escalabilidade: para ambas as ferramentas, a organização do código e modularização são desafios, com o uso inadequado de recursos levando a problemas de gestão de estado. Nesse sentido, o Entrevistado 2 disse:

"Os problemas enfrentados basicamente giram em torno da expansão organizada do projeto. Em ambas as ferramentas, se seguirmos práticas de modularização, a expansão se torna simples, porém se feita de uma forma desorganizada sem um padrão claro isso tende a se tornar um monstro. Em específico, no Terraform isso pode também trazer dificuldades no controle do State em times com mais de uma pessoa responsável pela infra, pois podem começar a aparecer conflitos caso existam tarefas sendo feitas em paralelo."

Manutenibilidade: para ambas as ferramentas, a organização do código é crucial, com boas práticas como a modularização e a separação de recursos em repositórios. Por exemplo, o Entrevistado 3 salientou:

"Ter bem separados os escopos é a principal forma de se garantir a manutenibilidade. Ter os projetos separados para cada serviço, ou talvez um diretório. É interessante talvez dividir os ambientes em contas. E pensando na cultura DevOps as pessoas têm que ter conhecimento sobre o que elas estão fazendo e esse conhecimento ser compartilhado."

Curva de aprendizado: o CDK possui uma curva de aprendizado menor, mas é importante aprender o Terraform, que é mais utilizado no mercado. Por exemplo, o Entrevistado 1 expôs:

"Para pessoas que estão entrando no mercado de trabalho o CDK teriam uma curva de aprendizado menor, embora o Terraform tenha uma linguagem que seja bem fácil de se familiarizar, o fato do CDK usar linguagens que as pessoas utilizam na faculdade facilita esse aprendizado por ser um código mais legıvel. Contudo, a longo prazo essa pessoa iria se deparar com o Terraform para lidar com outros ambientes de nuvem, já que é a ferramenta de IaC mais utilizada."

Facilidade de encontrar suporte: o Terraform é mais fácil de encontrar suporte devido à sua longa presença no mercado e comunidade de usuários. Por exemplo, o Entrevistado 1 expressou:

"O Terraform até por ser mais antigo, por ter várias versões, vários fóruns e vários especialistas formando uma comunidade ativa, possibilita você a encontrar essas resoluções mais fáceis. Já o CDK por mais novo continua em fase de teste, as pessoas estão começando a ter as primeiras dúvidas e limitações, continuam se deparando com os primeiros problemas, estão surgindo os primeiros fóruns."

Com base nas entrevistas com especialistas, identificou-se que o Terraform é mais adequado para ambientes *Multcloud*, embora demande atenção especial na gestão de es-

tado e na colaboração entre equipes. Por outro lado, o CDK emerge como uma opção preferencial para desenvolvedores iniciantes que trabalham com a AWS, apesar de suas limitações na gestão de estado advindas do Cloud Formation. Para ambas as ferramentas, uma abordagem modular é essencial para alcançar escalabilidade efetiva. A manutenibilidade depende fortemente da organização adequada e da modularização do código. Esses *insights* reforçam a importância de escolher a ferramenta certa, alinhada às necessidades específicas do projeto e às habilidades da equipe.

5.3. Estudo 3: Avaliação do Desempenho das Ferramentas na Criação de Recursos

Com o propósito de avaliar o desempenho das ferramentas na criação de recursos de infraestrutura na AWS, foi conduzido um experimento para avaliar o tempo de execução. Esse experimento visa responder à RQ3: *Qual ferramenta possibilita a criação de recursos de infraestrutura na AWS com menor tempo de execução?*

Na Figura 3, é possível observar, para cada caso testado no experimento, o tempo total gasto em cada *runner* para que a infraestrutura fosse totalmente criada. O CDK, representado pela linha mais escura, apresentou um tempo de execução maior em todos os casos rodados, gastando mais tempo que o Terraform, representado pela linha mais clara. Portanto, o Terraform demonstrou um desempenho significativamente mais rápido na criação de recursos em comparação com o CDK. Isso pode ser um fator importante para organizações que desejam provisionar recursos de maneira eficiente e rápida.

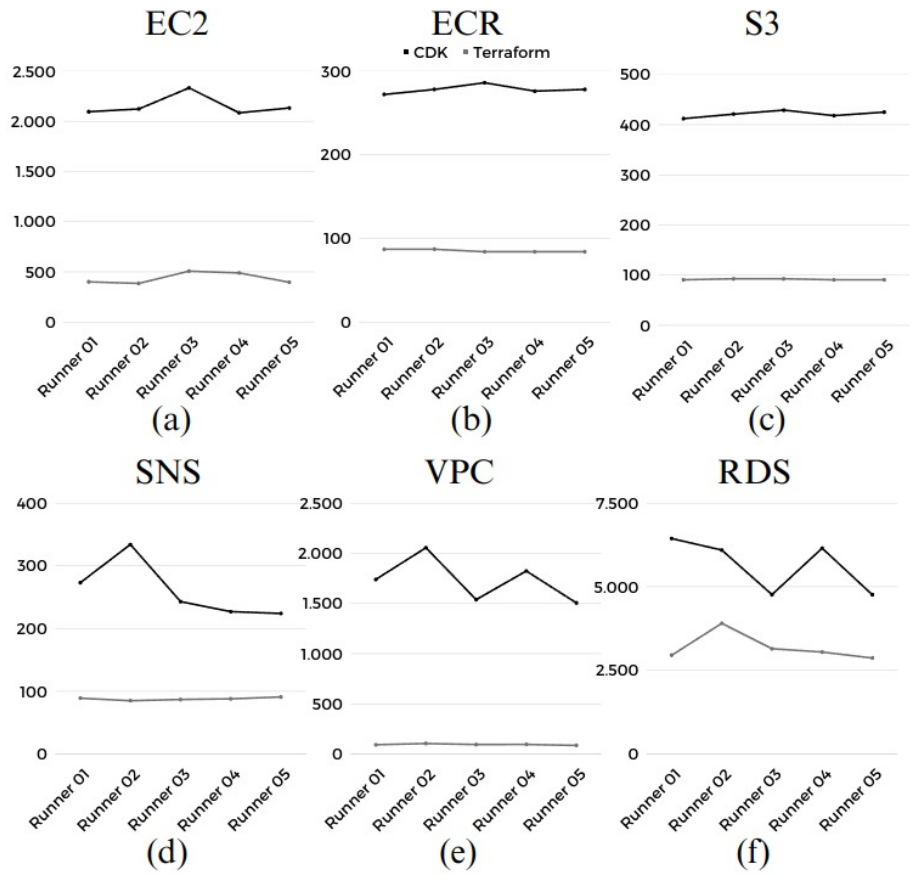


Figura 3. Tempo de execução de cada caso para cada uma das ferramentas.

Para avaliar o desempenho das ferramentas na criação de recursos de IaC, foi conduzido um experimento que busca avaliar métricas de uso de recursos computacionais. Esse experimento visa responder à quarta questão de pesquisa: *Qual ferramenta consome mais recursos computacionais, como CPU e memória, durante a criação de recursos?*

Na Figura 4, é mostrada a média da porcentagem de uso geral CPU, considerando todos os casos, para cada uma das ferramentas. É possível observar que o Terraform, durante todo período de execução, manteve um consumo de CPU acima do CDK. Esse comportamento pode ser explicado através do funcionamento das ferramentas, com o Terraform utilizando mais CPU localmente, já que todas as comparações de estado são feitas no computador onde ele foi executado. Por outro lado, o CDK consumiu menos, uma vez que utiliza a infraestrutura da AWS para parte do processamento.

Adicionalmente, na Figura 5, é apresentada a média da porcentagem de uso geral da memória, considerando todos os casos, para cada ferramenta. Diferente do consumo de CPU, o consumo de memória não mantém o comportamento inicial das ferramentas até o final, sendo possível observar que há variações sobre qual das ferramentas consome mais de acordo com a etapa observada no gráfico. Esse comportamento, por sua vez, pode ser explicado pela forma com que as ferramentas administram os arquivos de estado, com o Terraform manipulando e criando os arquivos na máquina onde é executado e o CDK armazenando esses arquivos na estrutura de *stacks* presente dentro do CloudFormation.

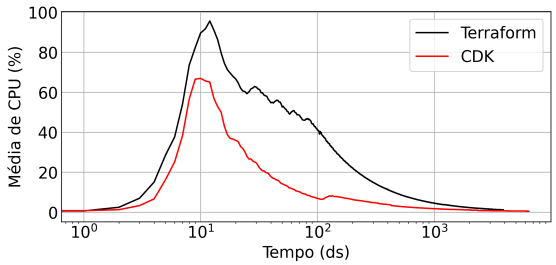


Figura 4. Porcentagem média de uso da CPU.

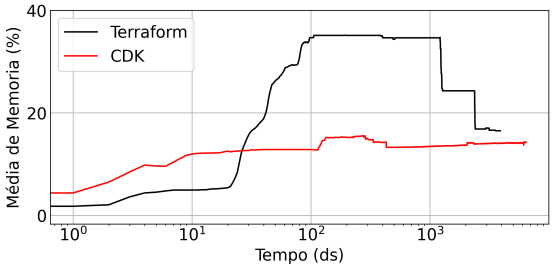


Figura 5. Porcentagem média de uso da Memória.

Ao comparar o Terraform e o CDK, é possível notar uma diferença no uso de recursos. O Terraform requer mais CPU localmente, enquanto o CDK consome menos durante todo processo. Em relação à memória, o Terraform consome menos no início, mas aumenta seu consumo à medida que o processo avança, ultrapassando o consumo do CDK, que era inicialmente maior. Dessa forma, equipes que desejam otimizar seus recursos locais podem se beneficiar dessa diferença ao utilizar o CDK.

6. Ameaças à Validade

Esta seção apresenta e aborda as potenciais ameaças à validade deste estudo, categorizadas como internas, externas e de construção [Wohlin et al. 2012].

As ameaças internas estão relacionadas às possíveis falhas na coleta de dados, particularmente na criação de IaCs na AWS. Para mitigar essas ameaças, foi implementada uma estratégia de repetir a criação dos recursos cinco vezes, calculando a média de tempo de execução para cada tentativa, aumentando a confiabilidade dos resultados.

Em relação às ameaças externas, a principal preocupação é a generalização dos resultados. No Experimento 1, buscou-se uma amostra diversificada de participantes, com diferentes níveis de conhecimento e experiências, visando uma representatividade abrangente. No Experimento 2, os entrevistados foram selecionados por sua experiência comprovada e certificação pela AWS, visando captar *insights* de profissionais qualificados. Adicionalmente, para enfrentar a hesitação de desenvolvedores em discutir práticas inadequadas em suas empresas durante as entrevistas, proporcionou-se um ambiente que incentivava os participantes a compartilhar cenários reais de uso das ferramentas, permitindo assim uma expressão mais autêntica e detalhada de suas experiências.

Quanto às ameaças de construção, identifica-se no terceiro experimento uma limitação relacionada à uniformidade dos recursos, o que pode restringir a profundidade da análise. Para mitigar essa limitação, é implementada uma variedade de recursos diferentes destinados a abranger diversas infraestruturas. Por conseguinte, isso intensifica a análise experimental e simplifica a acessibilidade para replicação do experimento, uma vez que a criação de estruturas mais complexas pode acarretar em custos na AWS. Reconhece-se essa restrição e sugere-se, para estudos futuros, a inclusão de variações nas características dos recursos a fim de permitir uma análise mais ampla e representativa.

7. Conclusão

Este estudo comparou o Terraform e o CDK na gestão de IaC na AWS, avaliando aspectos como capacidade de abstração, escalabilidade, manutenibilidade e desempenho. O CDK mostrou-se mais acessível para iniciantes, com uma estrutura programática, que facilita tarefas complexas, enquanto o Terraform, apesar de acessível, apresenta desafios em cenários mais complexos. Ambas as ferramentas necessitam de uma abordagem modular em grandes projetos, com o Terraform apresentando dificuldades de gerenciamento em equipes maiores e o CDK exigindo organização cuidadosa. O Terraform é mais rápido na criação de recursos, mas consome mais recursos de CPU, ao passo que o CDK é mais eficiente em recursos. A escolha entre eles depende das necessidades do projeto e da familiaridade da equipe.

Como trabalhos futuros, sugere-se duas principais linhas de pesquisa. A primeira é a expansão do estudo atual para explorar a eficiência das ferramentas de IaC em cenários mais complexos, analisando projetos que integram diversos serviços e dependências. Este trabalho visa oferecer uma compreensão mais aprofundada das capacidades e limitações dessas ferramentas na gestão de infraestruturas complexas, alinhando as conclusões com as realidades dos ambientes de produção. A segunda linha de pesquisa propõe um estudo comparativo mais amplo, incluindo ferramentas de IaC como Ansible, Chef, Puppet e CloudFormation, além do Terraform e do CDK. Tal análise comparativa ampliaria o entendimento sobre as características, eficiência e limitações dessas ferramentas, auxiliando as equipes de desenvolvimento e operações na escolha mais adequada e revelando oportunidades de inovação no campo da IaC.

Pacote de Replicação

O pacote de replicação deste trabalho encontra-se disponível em:

`https://github.com/ICEI-PUC-Minas-PPLES-TI/plf-es-2023-1-tci-0393100-pes-lucas-padrao-e-joao-victor-frois.git`

Referências

Alonso, J., Piliszek, R., and Cankar, M. (2023). Embracing iac through the devsecops philosophy: Concepts, challenges, and a reference framework. *IEEE Software*, 40(1):56–62.

Bogner, J. and Merkel, M. (2022). To type or not to type? a systematic comparison of the software quality of javascript and typescript applications on github. In *2022 IEEE/ACM 19th International Conference on Mining Software Repositories (MSR)*, pages 658–669.

Chappell, D. (2008). A short introduction to cloud platforms: An enterprise-oriented view. Technical report, David Chappell & Associates.

Coulter, M. (acessado em 31 de março de 2023). Matt coulter talks aws cdk, meeting werner vogels, silent discos.

Dalla Palma, S., Di Nucci, D., Palomba, F., and Tamburri, D. A. (2020). Toward a catalog of software quality metrics for infrastructure code. *Journal of Systems and Software*, 170:110726.

Dalla Palma, S., Di Nucci, D., Palomba, F., and Tamburri, D. A. (2022). Within-project defect prediction of infrastructure-as-code using product and process metrics. *IEEE Transactions on Software Engineering*, 48(6):2086–2104.

Debois, P. (2008). Agile infrastructure and operations: How infra-gile are you? In *Agile 2008 Conference*, pages 202–207.

Dörnenburg, E. (2018). The path to devops. *IEEE Software*, 35(5):71–75.

Gupta, M., Chowdary, M. N., Bussa, S., and Chowdary, C. K. (2021). Deploying hadoop architecture using ansible and terraform. In *2021 5th International Conference on Information Systems and Computer Networks (ISCON)*, pages 1–6.

HashiCorp (acessado em 31 de março de 2023). Introduction to terraform.

Ibrahim, A., Yousef, A. H., and Medhat, W. (2022). Devsecops: A security model for infrastructure as code over the cloud. In *2022 2nd International Mobile, Intelligent, and Ubiquitous Computing Conference (MIUCC)*, pages 284–288.

Kim, G., Humble, J., Debois, P., and Willis, J. (2018). *Manual de DevOps: como obter agilidade, confiabilidade e segurança em organizações tecnológicas*. Editora Alta Books, Rio de Janeiro, 1a ed. edition.

Loukides, M. (2012). *What is DevOps?* O'Reilly Media, 1a ed. edition.

Lovchikova, N. (acessado em 31 de março de 2023). Aws summit anz 2021 - driving a test-first strategy with cdk and test driven development.

Mala, D. J. (2019). *Integrating the Internet of Things Into Software Engineering Practices*. IGI Global, 1a ed. edition.

McKnight, P. E. and Najab, J. (2022). Mann–whitney u test. *The SAGE Encyclopedia of Research Design*.

Parnin, C., Rahman, A., and Williams, L. (2019). The seven sins: Security smells in infrastructure as code scripts. *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 164–175.

Patni, J. C., Banerjee, S., and Tiwari, D. (2020). Infrastructure as a code (iac) to software defined infrastructure using azure resource manager (arm). In *2020 International Conference on Computational Performance Evaluation (ComPE)*, pages 575–578.

Rahman, A., Mahdavi-Hezaveh, R., and Williams, L. (2019). A systematic mapping study of infrastructure as code research. *Information and Software Technology*, 108:65–77.

Ray, B., Posnett, D., Filkov, V., and Devanbu, P. (2014). A large scale study of programming languages and code quality in github. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2014, page 155–165, New York, NY, USA. Association for Computing Machinery.

Sandobalín, J., Insfran, E., and Abrahão, S. (2020). On the effectiveness of tools to support infrastructure as code: Model-driven versus code-centric. *IEEE Access*, 8:17734–17761.

Schlossnagle, T. (2017). Monitoring in a devops world: Perfect should never be the enemy of better. *Queue*, 15(6):35–45.

Sokolowski, D. and Salvaneschi, G. (2023). Towards reliable infrastructure as code. In *2023 IEEE 20th International Conference on Software Architecture Companion (ICSA-C)*, pages 318–321.

Vogels, W. (acessado em 03 de março de 2023). Werner vogels on the aws cloud development kit (aws cdk).

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., and Wesslén, A. (2012). *Experimentation in software engineering*. Springer Science & Business Media.

A. Apêndice A. Questionário da Avaliação com Alunos

Neste apêndice, são apresentadas as perguntas formuladas para os participantes do experimento descrito na Seção 4.2. Dois questionários foram formulados, um para cada uma das ferramentas, e cada questionário dividido em quatro seções. A primeira seção do questionário contém o termo de consentimento de participação e está descrita no Apêndice A.1. A segunda contém as questões de *background* e está descrita no Apêndice A.2. A terceira contém as questões sobre as ferramentas, incluindo as questões sobre Terraform descritas no Apêndice A.3 e as questões sobre o CDK descritas no Apêndice A.4. Por fim, a quarta seção tem objetivo de coletar *feedbacks* dos participantes e está descrita no Apêndice A.5

A.1. Termo de Consentimento Livre e Esclarecido (TCLE)

O termo de participação criado para o questionário foi elaborado seguindo as normas e recomendações estabelecidas pelo Comitê de Ética em Pesquisa (CEP) da PUC Minas, que aprovou a realização do experimento. O termo criado é apresentado nesse apêndice:

Prezado Sr(a). Por meio deste termo você receberá informações claras e compreensíveis sobre o experimento intitulado "Avaliação da abstração das ferramentas" conduzido por João Victor Tadeu Chaves Frois e Lucas Gabriel Padrão Rezende. Após a leitura do termo você estará ciente dos termos e condições do experimento:

Objetivo do Experimento: O objetivo deste experimento é avaliar a abstração das ferramentas Cloud Development Kit (CDK) e Terraform na criação de infraestruturas por desenvolvedores com pouca experiência.

Participação Voluntária: Sua participação neste experimento é voluntária, tendo o direito de desistir a qualquer momento sem consequências negativas.

Coleta de Dados: Você entende que dados pessoais serão coletados apenas para fins de pesquisa e que sua identidade será mantida em sigilo. Seus dados não serão divulgados publicamente.

Proteção de Dados: Os dados coletados serão armazenados de forma segura e acessados apenas pelos pesquisadores envolvidos no estudo.

Confidencialidade: Qualquer informação que você forneça durante o experimento será tratada com a devida confidencialidade.

Uso dos Resultados: Você compreende que os resultados deste experimento poderão ser utilizados para fins acadêmicos e de pesquisa, mas sua identidade será mantida anônima.

Esclarecimento de Dúvidas: Você terá a oportunidade de fazer perguntas e esclarecer quaisquer dúvidas relacionadas a este experimento antes de concordar em participar.

Contato: Você foi informado sobre como entrar em contato com os pesquisadores responsáveis por este experimento em caso de dúvidas ou preocupações. Sendo os meios de contado os seguintes e-mails - jfrois@sga.pucminas.br ou lgprezende@sga.pucminas.br.

Após a apresentação do termo de consentimento é coletada resposta do participante através de uma pergunta:

Pergunta 01: Aceito, de livre e espontânea vontade, participar deste estudo e concordo com os termos estabelecidos no Termo de Consentimento Livre e Esclarecido (TCLE).

Possíveis respostas:

- Sim
- Não

A.2. Questões de Background

Neste apêndice, são apresentadas as perguntas feitas para analisar o *background* dos participantes do experimento. São elas:

Pergunta 02: Qual seu nome (completo)?

Pergunta 03: Qual seu e-mail?

Pergunta 04: Sua graduação está sendo feita em qual curso?

Pergunta 05: Em qual período da graduação você está?

Possíveis respostas:

- 5º
- 6º
- 7º
- 8º

Pergunta 06: Você já atuou na área de TI?

Possíveis respostas:

- Nunca atuei na área

- Atuei/atuo como Desenvolvedor (Fullstack)
- Atuei/atuo como Desenvolvedor (Back-end)
- Atuei/atuo como Desenvolvedor (Front-end)
- Atuei/atuo como Suporte de TI
- Atuei/atuo como DevOps
- Atuei/atuo na área mas em outra função

Pergunta 07: Há quanto tempo você está atuando na área?

Possíveis respostas:

- Não atuo
- Menos de 1 ano
- Entre 1 e 3 anos
- Mais de 3 anos

Pergunta 08: Antes desse experimento você já teve contato com infraestrutura como código (IaC)?

Possíveis respostas:

- Sim
- Não

Pergunta 09: Qual seu nível de familiaridade com a ferramenta Terraform? 1 - Sem familiaridade e 5 - Muita familiaridade

Possíveis respostas:

- 1
- 2
- 3
- 4
- 5

A.3. Questões de Terraform

Neste apêndice, são apresentadas as perguntas que compõem a avaliação dos conhecimentos acerca do Terraform dos participantes do experimento. São elas:

Pergunta 10: Qual comando o Terraform requer na primeira vez que você executá-lo em um diretório de configuração?

Possíveis respostas:

- terraform import
- **terraform init**
- terraform plan
- terraform workspace

Pergunta 11: Qual é o fluxo de trabalho para implantar uma nova infraestrutura com o Terraform?

Possíveis respostas:

- Execute o *terraform plan* para importar a infraestrutura atual para o arquivo de estado, faça alterações de código e execute o *terraform apply* para atualizar a infraestrutura.

- Escreva uma configuração de Terraform, execute o *terraform show* para exibir as alterações propostas e o *terraform apply* para criar uma nova infraestrutura.
- Execute o *terraform import* para importar a infraestrutura atual para o arquivo de estado, faça alterações de código e execute o *terraform apply* para atualizar a infraestrutura.
- **Escreva uma configuração de Terraform, execute o *terraform init*, execute o *terraform plan* para exibir as alterações de infraestrutura planejadas e use o *terraform apply* para criar uma nova infraestrutura.**

Pergunta 12: Qual é o provedor desse recurso fictício?

Possíveis respostas:

- VPC
- Principal
- **AWS**
- test

Pergunta 13: Quais são as duas etapas necessárias para provisionar uma nova infraestrutura no fluxo de trabalho do Terraform?

Possíveis respostas:

- Init e Plan.
- Plan e Apply.
- **Init e Apply.**
- Import e Apply.

Pergunta 14: Qual das seguintes opções é a maneira correta de passar o valor na variável *num_servers* para um módulo com os servidores de entrada?

Possíveis respostas:

- servidores = num_servers
- servidores = variable.num_servers
- servidores = var(num_servers)
- **servidores = var.num_servers**

Pergunta 15: Terraform pode importar módulos de diferentes fontes. Qual dos seguintes não é uma fonte válida?

Possíveis respostas:

- **Servidor FTP.**
- Repositório GitHub.
- Caminho local.
- Registro do módulo Terraform.

Pergunta 16: O que não é processado ao executar uma atualização de Terraform?

Possíveis respostas:

- Arquivo de estado (State file).
- **Arquivo de configuração (Configuration file).**
- Credenciais (Credentials).
- Provedor de nuvem (Cloud provider).

Pergunta 17: Qual tarefa o *terraform init* não executa?

Possíveis respostas:

- Origina todos os provedores presentes na configuração e garante que eles sejam baixados e disponibilizados localmente.
- Conecta-se ao back-end.
- Origina todos os módulos e copia a configuração localmente.
- **Valida se todas as variáveis necessárias estão presentes.**

Pergunta 18: Qual é a desvantagem de usar blocos dinâmicos no Terraform?

Possíveis respostas:

- Eles não podem ser usados para percorrer uma lista de valores.
- Blocos dinâmicos podem construir blocos aninhados repetíveis.
- **Eles tornam a configuração mais difícil de ler e entender.**
- Terraform ser executado mais devagar.

Pergunta 19: Examine a seguinte configuração do Terraform, que usa a fonte de dados para uma AMI da AWS. Qual valor você deve inserir para o argumento ami no recurso de instância da AWS?

Possíveis respostas:

- aws_ami.ubuntu
- aws_ami.ubuntu
- **data. aws_ami.ubuntu.id**
- aws_ami. ubuntu.id

A.4. Questões de CDK

Neste apêndice, são apresentadas as perguntas que compõem a avaliação dos conhecimentos acerca do CDK dos participantes do experimento. São elas:

Pergunta 10: Qual é o comando utilizado no AWS CDK para sintetizar (gerar) os recursos definidos em sua aplicação de infraestrutura como código?

Possíveis respostas:

- cdk build
- cdk generate
- cdk deploy
- **cdk synth**

Pergunta 11: Qual é a principal vantagem do AWS CDK em comparação com abordagens tradicionais de gerenciamento de infraestrutura como código?

Possíveis respostas:

- **O AWS CDK permite criar e gerenciar infraestruturas usando linguagens de programação familiares, em vez de arquivos de configuração estáticos.**
- O AWS CDK suporta apenas a linguagem de programação Python para definição de infraestrutura.
- O AWS CDK é uma ferramenta exclusiva para provisionar recursos locais, sem integração com a nuvem.

– O AWS CDK oferece uma interface gráfica intuitiva para arrastar e soltar recursos na AWS.

Pergunta 12: Qual é a diferença fundamental entre um "Construct"e um "Stack"no AWS Cloud Development Kit (CDK)?

Possíveis respostas:

– **Um "Construct"é uma unidade básica de recurso, enquanto um "Stack"é uma coleção de "Constructs"que pode ser implantada como uma unidade única.**
– Um "Construct"é uma pilha de recursos virtuais, enquanto um "Stack"representa um único recurso físico implantado.
– Um "Construct"é uma função lambda sem estado, enquanto um "Stack"é uma função lambda com estado.
– Não há diferença entre um "Construct"e um "Stack", sendo termos intercambiáveis no contexto do AWS CDK.

Pergunta 13: Qual é o resultado da execução do código fornecido após a implantação bem-sucedida usando o AWS CDK?

Possíveis respostas:

– Uma instância EC2 será criada, mas não será adicionada como um alvo para o balanceador de carga.
– Um balanceador de carga será criado, mas não será configurado para ser voltado para nenhuma instância.
– **Um balanceador de carga será criado e configurado para ter a instância EC2 como alvo.**
– O código contém erros de sintaxe e não será possível implantá-lo com o AWS CDK.

Pergunta 14: Ao projetar uma tabela global no AWS DynamoDB com requisitos de alta disponibilidade e desempenho, juntamente com a necessidade de proteção contra exclusões acidentais, quais opções de configuração são adequadas para definir as regiões de replicação nas áreas geográficas da Europa Central e Ásia Sudeste, ao mesmo tempo, em que se opta por um modelo de faturamento que exige a especificação prévia de capacidade e habilita a proteção contra exclusões?

Possíveis respostas:

– replicationRegions: ['af-south-1', 'eu-north-1'] billingMode: PAY_PER_REQUEST deletionProtection:false
– **replicationRegions:['eu-central-1', 'ap-southeast-2'] billingMode: PROVISIONED deletionProtection:true**
– replicationRegions:['eu-central-1', 'sa-east-1'] billingMode: ON_DEMAND deletionProtection:true
– replicationRegions: ['ap-northeast-3', 'me-south-1'] billingMode: PAY_PER_REQUEST deletionProtection:false

Pergunta 15: Qual é o objetivo do código fornecido?

Possíveis respostas:

- **Criar um tópico SNS e uma regra do AWS IoT que direciona mensagens para esse tópico quando uma mensagem corresponder ao padrão SQL especificado.**
- Criar um tópico SNS e uma função Lambda que processa mensagens IoT quando recebidas no tópico.
- Criar uma função Lambda que publica mensagens no AWS IoT Topic especificado.
- Configurar um tópico SNS para receber notificações de uma instância do Amazon EC2.

Pergunta 16: Suponha que você está criando um recurso de armazenamento no Amazon Web Services (AWS) usando o AWS Cloud Development Kit (CDK) e deseja configurar um bucket no Amazon S3. Você deseja garantir que o bucket não seja acessível publicamente, os dados dentro do bucket sejam automaticamente criptografados pelo S3, a transferência de dados para o bucket seja sempre feita por SSL, e você também deseja reter o bucket mesmo quando a pilha do CDK for excluída. Qual dos seguintes trechos de código do CDK atingirá esses objetivos?

Pergunta 17: Qual das seguintes afirmações é verdadeira sobre o código abaixo?

Possíveis respostas:

- O código cria um pipeline de entrega contínua que autentica com sucesso no Docker Hub e em um registro personalizado, mas não no ECR.
- O código cria um pipeline de entrega contínua que autentica com sucesso no ECR e em um registro personalizado, mas não no Docker Hub.
- **O código cria um pipeline de entrega contínua que autentica com sucesso no Docker Hub, no ECR e em um registro personalizado.**
- O código contém um erro de sintaxe e não criará um pipeline de entrega contínua.

Pergunta 18: No código fornecido, qual é a finalidade da configuração treatMissingData: cloudwatch.TreatMissingData.IGNORE na definição do alarme do CloudWatch?

Possíveis respostas:

- Garantir que os dados ausentes não sejam considerados em cálculos futuros do alarme.
- Ignorar a criação do alarme se houver dados ausentes no período de avaliação.
- **Suprimir a geração de notificações de alarme em caso de dados ausentes.**
- Configurar o CloudWatch para coletar automaticamente dados ausentes durante falhas.

Pergunta 19: Qual é o objetivo da rota estática criada no trecho de código fornecido?

Possíveis respostas:

- Permitir que o tráfego da sub-rede pública alcance a sub-rede isolada.
- Bloquear todo o tráfego de entrada na sub-rede isolada.
- Redirecionar todo o tráfego da sub-rede isolada para a sub-rede pública.
- **Permitir que o tráfego da sub-rede isolada alcance a sub-rede pública.**

A.5. Questões de Feedback

Neste apêndice, é apresentada a pergunta feita para coletar os *feedbacks* dos participantes do experimento. Sendo ela:

Pergunta 20: Tem algum comentário ou sugestão para melhorarmos esse questionário?

B. Apêndice B. Roteiro da Entrevista com Especialistas

Neste apêndice, é apresentado o roteiro da entrevista com as perguntas realizadas aos especialistas. São elas:

Pergunta 01: Pode nos contar um pouco sobre sua experiência e conhecimento em Infrastructure as Code (IaC), especificamente em relação ao Terraform e ao CDK?

Pergunta 02: Na sua visão, quais as vantagens e desvantagens do Terraform e CDK?

Pergunta 03: Como você avaliaria a escalabilidade do Terraform e CDK em projetos de infraestrutura de grande porte? Quais são os principais desafios?

Pergunta 04: Em termos de manutenibilidade, quais são as práticas recomendadas para organizar e gerenciar código Terraform ao longo do tempo? E para o CDK?

Pergunta 05: Como o Terraform lida com a gestão de estado e quais são os desafios associados à escalabilidade e manutenibilidade nesse aspecto?

Pergunta 06: O CDK introduz abstrações de alto nível e a capacidade de escrever código em linguagens de programação, como TypeScript e Python. Como isso afeta a manutenibilidade e a escalabilidade?

Pergunta 07: Quando você consideraria o uso do Terraform apropriado em vez do CDK e vice-versa? Existem casos de uso específicos para cada uma dessas ferramentas?

Pergunta 08: Como você avaliaria a curva de aprendizado do Terraform em comparação com a do CDK para equipes novas em IaC?

Pergunta 09: No dia a dia de sua equipe quais vocês têm mais facilidade de encontrar suporte (documentação, fóruns, etc.)?

Pergunta 10: Há algum conselho adicional que você gostaria de compartilhar para equipes que estão avaliando tanto o Terraform quanto o CDK?

Pergunta 11: Existe algum caso de uso específico ou cenário em que nenhuma dessas ferramentas seja a melhor opção? Qual ferramenta você usaria? Por quê?

Pergunta 12: Como você enxerga o futuro dessas ferramentas em termos de escalabilidade e manutenibilidade, considerando o desenvolvimento contínuo e as atualizações?