# Terraform and AWS CDK: A Comparative Analysis of Infrastructure Management Tools

João Frois[1], Lucas Padrão[1], Johnatan Oliveira[2], Laerte Xavier[1], Cleiton Tavares[1]
[1]Pontifical Catholic University of Minas Gerais (PUC-MG)
Belo Horizonte, MG, Brazil
[2]Federal University of Lavras (ICTIN-UFLA)
Lavras, MG, Brazil
{joao.victor.frois,padraorz}@gmail.com,johnatan.oliveira@ufla.br,{laertexavier,cleitontavares}@pucminas.br

## ABSTRACT

Infrastructure as Code is a fundamental concept in DevOps that automates infrastructure management processes using code. Several tools, such as Terraform and CDK, support this environment. Selecting the appropriate tool is crucial to a project's success, yet there is ambiguity about the circumstances in which developers should choose between these tools. Therefore, this study aims to compare Terraform and CDK across four aspects: abstraction, scalability, maintainability, and performance. Our findings indicate that each tool performs particularly well in specific scenarios. For instance, Terraform is better suited for experienced teams focused on rapid implementations, while CDK is more appropriate for less experienced teams prioritizing resource efficiency during implementation.

## KEYWORDS

AWS CDK, Terraform, DevOps, Infrastructure as Code

## 1 INTRODUCTION

A wide range of techniques and protocols created to support the reliable and effective delivery of software are included in the DevOps culture [7]. The idea of automation is fundamental to this culture, especially when it comes to using Infrastructure as Code (IaC). IaC uses configuration files and code to manage infrastructure, allowing cloud environments to quickly and securely deploy applications [9]. In order to achieve these aims, this approach uses automated scripts [10]. In software engineering, effective management of these environments is essential to guarantee scalability and resource efficiency [3]. Previous research has evaluated the role of IaC in the software development lifecycle [1, 3].

IaC is receiving more attention in a variety of scenarios, which highlights how it could affect infrastructure management and software distribution [5]. Numerous studies in this emerging sector have found code smells that were introduced when creating IaC scripts [1, 5, 11]. Other studies have suggested creative tools for infrastructure management [11] or investigated the usage of IaC tools to accelerate the deployment of architectural models [13]. Given the extensive use and progress in this field, there are few studies in the literature concerning the relative effectiveness of different infrastructure-as-code support tools.

Terraform is an open-source IaC tool allows users to define and manage their data center infrastructure using a declarative configuration language. Terraform offering multiple service providers, such as AWS, Azure, and Google Cloud Platform[1]. Its state management capabilities and modularity make it a tool for large-scale infrastructure deployments [6]. On the other hand, the AWS Cloud Development Kit (CDK) is an open-source software development platform that allows you to specify cloud architecture using well-known programming languages. With CDK, developers may create infrastructure using languages like TypeScript, Python, Java, and C#, which offers a greater level of abstraction and integration with AWS services[2].

The choice of IaC management tools is as important as the choice of programming languages for software development, which differs depending on certain scenarios. Research that compares different provisioning and infrastructure management solutions should be done, much as studies that compare programming languages [2, 12]. These kinds of studies can help designers determine which tool is best suited for their particular uses. While the AWS Cloud Development Kit (CDK) [15] abstracts infrastructure management on the main cloud, Terraform is the most efficient tool for multi-cloud deployments [6]. For developers to make wise decisions and guarantee the success of projects in cloud environments, closing this comparability gap is essential.

This research aims to provide a comparative analysis to evaluate development using AWS CDK and Terraform. Our goals are structured as follows: (i) assessing the infrastructure development abstraction capabilities of non-experienced developers when using both Terraform and CDK; (ii) evaluating the tools' scalability and maintainability for experienced developers; and (iii) analyzing the performance of these tools in creating AWS infrastructure instances. This paper presents the findings of our comparison between Terraform and AWS CDK, detailing the differences between the two in terms of performance, scalability, maintainability, and abstraction capabilities. The study identifies scenarios in which each tool excels. For instance, Terraform is effective for experienced teams seeking quick implementations, whereas AWS CDK is better suited for less experienced teams aiming to minimize computational resource usage during implementation.

The remainder of this paper is organized as follows. Section 2 present the related work. Section 3 describes the research method. Sections 4 shows the results and discuss the insights. Section 5 outlines the threats to validity and their mitigations. Finally, Section 6 provides the conclusions and future work.

[1]https://www.terraform.io/
[2]https://aws.amazon.com/en/cdk/

## 2 RELATED WORK

Using the Argon and Ansible tools, Sandobalín et al. [13] compares two IaC approaches: i) code-centric and ii) model-driven. To solve the absence of information on the performance comparison of various technologies, they assess each based on its efficacy and convenience of use. According to their findings, there isn't a single strategy that is always better; instead, the best instrument for a given task will depend on its particular requirements. Our research, which differs from theirs, looks at abstraction, scalability, maintainability, and performance from the angles of AWS CDK and Terraform.

Dalla Palma et al. [4] uses Ansible to discover and catalog software quality metrics in IaC. They identified 46 metrics are defined and categorized into three categories: (i) portable object-oriented metrics for Ansible and IaC; (ii) portable metrics from earlier IaC studies to Ansible; and (iii) metrics pertaining to Ansible best and worst practices. Their findings demonstrate that IaC quality may be effectively assessed using software code quality metrics, including the detection of code smells. Even though this study was made specifically for Ansible, it is still a useful resource since it provides an example of how to evaluate IaC metrics and lays the groundwork for future assessments with other tools, like as Terraform and CDK.

Gupta et al. [6] assess how well the Hadoop architecture can be implemented in a cloud environment with the use of Ansible and Terraform technologies. Using these IaC tools, they put automation scripts into place and ran load tests to assess the architecture's performance and scalability. The outcomes showed that the technologies made it possible to install the Hadoop architecture quickly, effectively, and scalable. Their research, in contrast to ours, employs a combination of various infrastructure technologies rather than comparing them side by side.

Sokolowski and Salvaneschi [14] discuss methods for guaranteeing infrastructure dependability. They provide ProTI, an automated testing tool that acts as a fuzzer for Pulumi[3]. ProTI is based on unit testing. This tool uses contemporary methods to evaluate IaC programs across various setups. Their findings demonstrate a decrease in the amount of work that must be done by hand and an increase in trust in the output of the automated testing tool, which offers quick and thorough infrastructure feedback. Their research and ours are related, since both deal with the automation of infrastructures. Instead of concentrating on Pulumi, our study examines AWS CDK and Terraform, offering insights into the functionality and performance of these particular tools.

While previous research offers insightful information about a range of IaC tools and procedures, a thorough comparison of AWS CDK and Terraform is still lacking in the literature. By analyzing these tools from a variety of angles, our research seeks to close this gap and provide developers with useful advice on which tool would be best for their particular requirements.

## 3 RESEARCH METHOD

This study draws on both qualitative and quantitative approaches. To perform this, two controlled experiments were conducted to apply specific interventions and collect quantitative metrics, such as execution time and computational resource consumption of the tools. In addition, interviews were conducted with IaC experts to explore the evaluated tools in depth. The primary goal of these methods is to describe and compare the tools based on four key aspects: i) abstraction capacity, ii) scalability, iii) maintainability, and iv) performance. Practical experiments were performed using real-world use cases, allowing for an evaluation of the tools' efficiency in realistic scenarios. These experiments aimed to identify the differences between the tools and assess their efficiency across various aspects. The combination of these methods provides a detailed analysis of the tools' performance, facilitating informed decision-making regarding the most appropriate choice for different scenarios.

### 3.1 Goal and Research Questions

To achieve the specific goals defined for this study, the following Research Questions (RQs) are assessed:

- **RQ1:** How CDK and Terraform compare in terms of abstraction capability in creating IaC's?
- **RQ2:** How Terraform and CDK compare in terms of scalability and maintainability?
- **RQ3:** Which tool enables the creation of infrastructure resources on AWS with the shorter execution time?
- **RQ4:** Which tool consumes the most computational resources, such as CPU and memory, during the resource creation?

Figure 1 shows the main steps of the studies designed to answer the RQs. Study 1 evaluates the abstraction capacity of the tools in relation to developers with less experience (RQ1). Study 2 assesses scalability and maintainability based on insights from experts in the field (RQ2). Finally, Study 3 examines the performance of the tools in creating resources on AWS, addressing RQs 3 and 4.

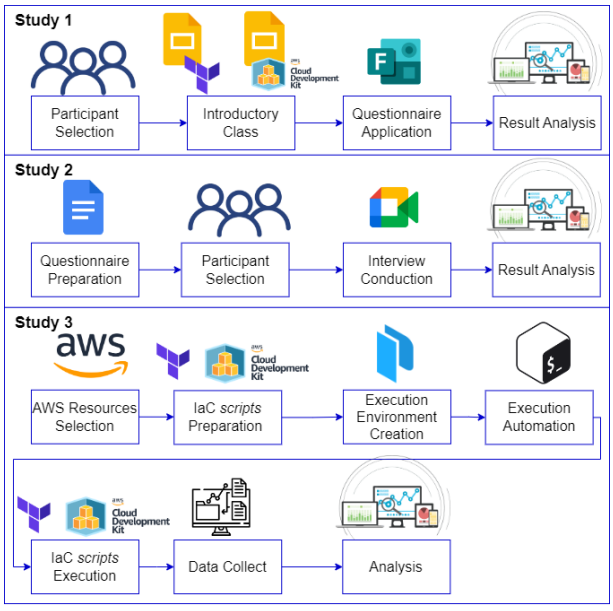



**Figure 1: Study Steps**

[3]IaC automation platform, allowing customers to design, deploy, and manage infrastructure across various clouds.

### 3.2 Study 1: Evaluation of Tool Abstraction

The experiment to evaluate the abstraction capacity of AWS CDK and Terraform in the creation of infrastructure was conducted with 30 students coursing at least one discipline from the sixth period of the Software Engineering in PUC-MG. The stages of the experiment are described as follows:

*Participant Selection*— Students were chosen according to their course of study, with a concentration on those who had finished the fourth period, guaranteeing a minimal but adequate level of understanding of the tools assessed.

*Introductory Class* – Students were randomly divided into two groups, each receiving an introductory class on either Terraform or AWS CDK. These classes covered fundamental concepts, basic syntax, and specific functionalities of each tool.

*Questionnaire Application* – A questionnaire was conducted via Microsoft Forms, including a consent form, eight demographic questions, 10 questions about the tools (easy, medium, and difficult), and a feedback field. To classify the questions, we sought input from three developers in our network.

*Result Analysis* – Statistical differences in student responses regarding AWS CDK and Terraform were analyzed. Given that our data on students' performance do not follow a normal distribution, the Mann-Whitney test [8] is appropriate for assessing the differences between the groups. This is a non-parametric test comparing medians of two samples, with a significance level of 0.05.

*Pilot Study* – A pilot study was conducted in September 2023 with 23 fifth-period students who did not participate in the main study. Feedback from this pilot was used to refine the introductory classes and questionnaires. Adjustments included changes to code snippets to prevent external consultations and a more detailed exploration of concepts such as the implementation flow of each tool.

### 3.3 Study 2: Evaluation of Tools Scalability and Maintainability

To evaluate the scalability and maintainability of AWS CDK and Terraform in IaC projects, we conducted interviews with three experts. This method provides valuable insights from experienced professionals. Each stage of the interview process is described as follows:

*Questionnaire Preparation* – A detailed interview guide was prepared, focusing on scalability and maintainability. Questions aimed to capture expert views on topics such as: *"How would you evaluate the scalability of Terraform and CDK in large infrastructure projects? What are the main challenges?"* and *"In terms of maintainability, what are the best practices for organizing and managing Terraform code over time? And for CDK?"* A total of 12 questions were structured for the interviews.

*Participant Selection* – Three experts were selected through the authors' professional network, chosen for their relevant experience with both tools.

*Interview Conduction* – Interviews were conducted in October 2023 via Google Meet, depending on participants' availability. The sessions were recorded with participants' consent for later analysis, ensuring anonymity and allowing them to withdraw at any time. Each interview lasted approximately 40 minutes.

*Result Analysis* – The recordings and responses were subjected to qualitative analysis, with the coding process inspired by the principles of Grounded Theory, to identify trends and patterns related to the tools' scalability and maintainability.

### 3.4 Study 3: Performance Evaluation of the Tools in Resource Creation

In order to assess Terraform's and AWS CDK's effectiveness in generating infrastructure resources on AWS, we established a context that included resource-service interactions as well as common IaC management constraints and obstacles. We developed scripts in both tools to provide resources on AWS within the Free Tier limitations based on this situation. Table 1 list the resources used in this study.

We developed the scripts using the standard structures and modules recommended by the official sources of each tool[4],[5], ensuring a fair and unbiased evaluation. To maintain the experiment within the AWS Free Tier limits, some modifications were made to the resource configurations. For example, the EC2 instance type was adjusted to *T2.micro* with Ubuntu 20.04, 1 CPU, and 1 GiB RAM. Additionally, a controlled environment was created using code, with five EC2 instances to execute the test cases.

During the experiment, we collected the following metrics: computational resource usage, assessing memory and CPU usage to evaluate the impact of the tools on computational components, and execution time, averaged from five executions to create the infrastructure elements, recorded in tenths of a second. Multiple executions help avoid incorrect measurements, allowing us to accurately evaluate the efficiency of each tool in creating IaC resources.

## 4 RESULTS

### 4.1 Study 1: Evaluation of Tool Abstraction

In order to assess the abstraction capability of IaC tools, an experiment was conducted with novice developers. Specifically, the aim is to address the RQ1: *How do CDK and Terraform compare in terms of abstraction capability in creating IaC's?*

For the experiment, 30 students from the sixth semester of a Software Engineering course were selected and randomly divided into two groups, resulting in 14 for CDK and 16 for Terraform. Among them, 21 had no previous experience with IaC. Regarding professional experience, four students had less than one year, 18 had one to three years, five had more than three years, and three had no experience in the technology field. In the Terraform group, 16 students rated their familiarity with the tool as 1 on a scale of 1 to 5, indicating limited knowledge. In the CDK group, three out of 14 students rated their familiarity level as 2. This uniformity in initial assessments suggests that the students had similar knowledge levels regarding the tools before the experiment.

Figure 2 presents the assertiveness index (score) from the applied questionnaire, composed by 10 questions answered by 30 participants. Figure 2 (left), the scores obtained by Terraform participants vary significantly, with the highest score reaching 8 and the lowest score being 2, resulting in an average score of 4.56 and a median score of 4. Figure 2 (right), the scores of CDK participants also show significant variation, with the highest score reaching 9 and the lowest score being 3, resulting in an average score of 6 and a median score of 6.

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

**Table 1: AWS Resources**

| Service Name | Service Description | Service Acronym |
|---|---|---|
| Elastic Compute Cloud | Scalable virtual machines in the cloud. | EC2 |
| Simple Storage Service | Scalable and durable object storage. | S3 |
| Elastic Container Registry | Container registration for ECS. | ECR |
| Relational Database Service | Managed relational database service. | RDS |
| Simple Notification Service | Messaging service for distributed applications. | SNS |
| Virtual Private Cloud | Isolated cloud network for hosting resources. | VPC |

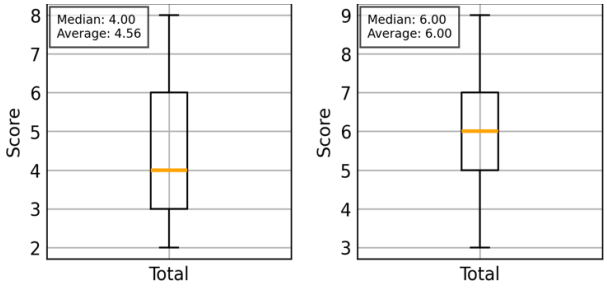
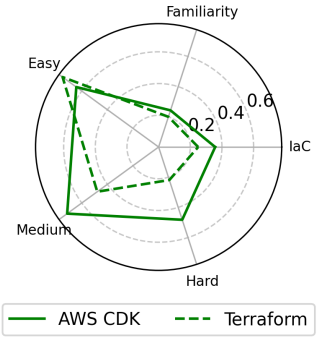



**Figure 2: Terraform (left) and AWS CDK (right)**

The Terraform group showed a median closer to the average, suggesting more consistent performance. On the other hand, CDK group achieved the same average and median scores. In the Mann-Whitney test applied to the total scores, the statistic was 64, and the p-value was approximately 0.048. Since this value is less than the conventional threshold of 0.05, it is concluded that there is a subtle difference between the median total scores of students for each tool.




| | CDK | Terraform |
|---|---|---|
| Easy | 0.643 | 0.75 |
| Medium | 0.714 | 0.479 |
| Hard | 0.482 | 0.219 |

**Figure 3: Performance Comparison between Tools in Percentage**

Figure 3 shows the relationships between participants' average familiarity with the respective tool and their percentage of correct answers at each difficulty level. For easy questions, Terraform participants achieved a 75% accuracy rate, compared to CDK's 64.3%. For medium questions, Terraform's accuracy rate was 47.9%, while CDK's was 71.4%. For hard questions, Terraform's accuracy was 21.9%, and CDK's was 48.2%.

The Mann-Whitney statistic in the tests conducted on simple questions was 85.0, with a p-value of 0.0695. There are no statistically significant differences in the students' scores for easy questions between the two groups (Terraform and CDK) as this value above the traditional threshold of 0.05. With a p-value of 0.0677, the absence of significant difference remained in the medium questions. On the other hand, for the difficult questions, the 95% confidence level p-value was much smaller than 0.05 (0.0028), demonstrating a statistically significant difference in the students' performance between the CDK and Terraform groups.

The findings suggest that CDK has a less steep learning curve than Terraform, despite its goal of offering a higher level of abstraction in the creation of IaC. Particularly to its programmatic nature, CDK gets more comprehensible as task complexity increases, whereas Terraform seems more conceptually approachable for simpler projects. Thus, in keeping with its goal of abstracting complexity and promoting practical use, CDK offers a smoother learning curve for more complicated applications, whereas Terraform may be more appropriate for lesser jobs.

## 4.2 Study 2: Evaluation of Tools Scalability and Maintainability

Three specialists with extensive experience and AWS certifications were interviewed to evaluate the tools' scalability and maintainability. With each participant's permission, the interviews were captured on video via Google Meet. The goal of this study is to answer RQ2: *What is the scalability and maintainability comparison between Terraform and CDK?*

The interviews aimed to understand how IaC tools handle complex projects regarding scalability (managing expanding infrastructures) and maintainability (ease of modifying, updating, and understanding infrastructure code over time). Insights from industry experts were sought to compare these two critical aspects, identifying trends, patterns, and distinct characteristics of each tool in large-scale scenarios. The main conclusions and trends observed in the experts' responses are discussed as follows.

*Terraform Advantages and Disadvantages* – Experts noted that Terraform facilitates the configuration of hybrid environments (using different cloud providers) but has limitations regarding state management and teamwork strategies. For instance, Interviewee 1 pointed out:

> *The main advantage of Terraform is its compatibility with multiple clouds. The issue of state poses a challenge in larger teams.*

*CDK Advantages and Disadvantages* – Experts evaluated CDK as easier for developing AWS resources, especially for novice developers, but it inherits CloudFormation's limitation in state reading. For instance, Interviewee 3 argued:

> *CDK is advantageous for beginners, but inherits challenges from CloudFormation regarding state reading.*

*Scalability* – Both tools suffer from challenges in code organization and modularization, with inconsistent resource usage leading to state management issues. Interviewee 2 stated:

> *The main challenges revolve around organized project expansion. With both tools, modularization practices simplify expansion. However, disorganized implementation without a clear pattern can become problematic. Specifically, in Terraform, managing state can be difficult for teams with multiple people responsible for the infrastructure, as parallel tasks can cause conflicts.*

*Maintainability* – Code organization is crucial for both tools, with best practices such as modularization and separating resources into repositories. Interviewee 3 emphasized:

> *The guarantee of maintainability requires well-separated scopes. Separating projects for each service or into distinct directories is beneficial. Dividing environments into accounts can also be advantageous. In the DevOps culture, it's crucial that people understand their tasks and share this knowledge.*

*Learning Curve* – CDK has a shorter learning curve, but learning Terraform is essential due to its wider industry usage. Interviewee 1 stated:

> *For individuals entering the industry, CDK offers a shorter learning curve, as it uses languages familiar from college, making the code more readable. However, Terraform's language is also easy to learn. In the long run, individuals will likely need to use Terraform for other cloud environments, as it is the most widely used IaC tool.*

*Ease of Finding Support* – Terraform has more support due to its longer presence in the industry and established user community. Interviewee 1 expressed:

> *Terraform, being older with numerous versions, forums, and a community of experts, makes it easier for you to find resolutions. On the other hand, CDK, being newer, is still in a testing phase. People are beginning to have initial doubts and limitations, encountering the first problems, and the first forums are emerging.*

Based on the interviews, we identified that Terraform is more suitable for multi-cloud environments, though it requires special attention to state management and team collaboration. Conversely, CDK is preferred for novice developers working with AWS, despite its state management limitations inherited from CloudFormation. For both tools, a modular approach is essential for effective scalability. Maintainability heavily depends on proper organization and modularization of the code. These insights reinforce the importance of choosing the right tool aligned with specific project needs and the team's skills.

### 4.3 Study 3: Performance Evaluation of the Tools in Resource Creation

To assess the performance of the tools in creating infrastructure resources on AWS, we conducted an experiment to evaluate runtime. This experiment aims to answer RQ3: *Which tool enables the creation of infrastructure resources on AWS with the shorter execution time?*

Figure 4 shows the total time spent to create each infrastructure *runner*. CDK, represented by the darker line, consistently had a longer execution time in all cases compared to Terraform, represented by the lighter line. Therefore, Terraform demonstrated significantly faster performance in creating resources. This is an important factor for organizations that seek efficient and quick resource provisioning.

To evaluate the tools' performance in creating IaC resources, another experiment was conducted to assess metrics related to the utilization of computational resources. This experiment aims to answer RQ4: *Which tool consumes more computational resources, such as CPU and memory, during the creation of resources?*

Figure 5(left) shows the average CPU usage for each tool. Terraform consistently maintained higher CPU consumption throughout the execution period. This behavior can be explained by the operation of the tools, with Terraform performing all state comparisons locally on the machine where it is executed. In contrast, CDK consumes less CPU because it leverages AWS infrastructure for part of the processing.

Figure 5 (right) presents the average memory usage for each tool. Unlike CPU consumption, memory usage varies throughout the execution period. These variations can be attributed to the management of the tools' state files: Terraform manipulates and creates these files locally, while CDK stores them in the CloudFormation *stack* structure. This explains why different tools show variations in resource consumption at different stages of the process.

Comparing Terraform and CDK reveals a perceptible difference in resource usage. Terraform requires more local CPU, while CDK consistently consumes less throughout the entire process. Regarding memory, Terraform consumes less initially but increases its usage as the process progresses, eventually surpassing the consumption of CDK, which starts higher. Teams aiming to optimize their local resources may benefit from using CDK, given these differences.

## 5 THREATS TO VALIDITY

This section presents and discusses the potential threats to the validity of this study, categorized as internal, external, and construct threats [16].

Internal threats relate to potential failures in data collection, particularly when creating IaCs on AWS. To mitigate this threat, the resource creation process was carried out five times, calculating the average execution time to increase the reliability of the results.

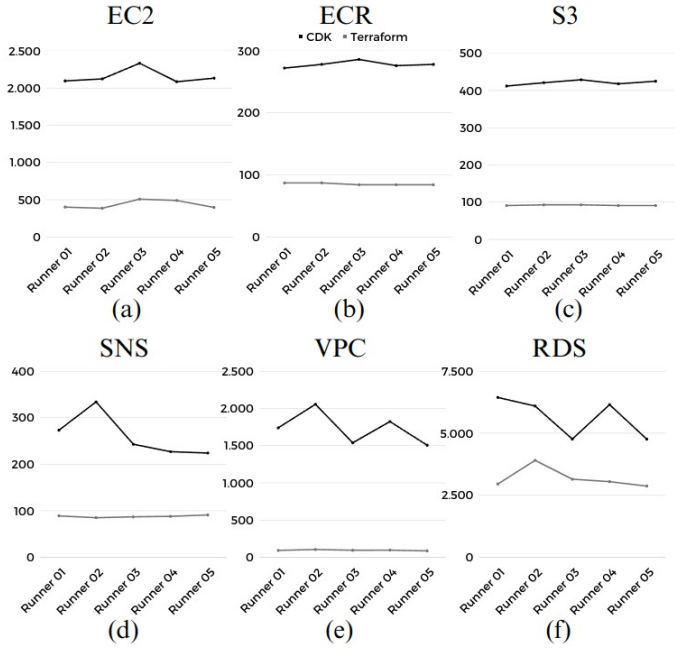



**Figure 4: Runtime of each scenario for each of the tools.**

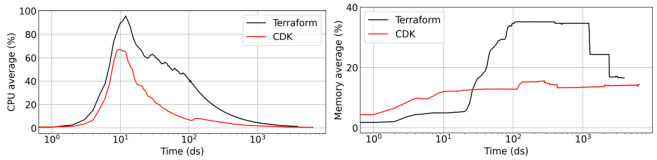



**Figure 5: Average CPU and Memory usage**

External threats concern the generalization of results. In Study 1, a diverse sample of participants with varying levels of knowledge and experience was sought to ensure comprehensive representativity. In Study 2, interviewees were selected based on their proven experience and AWS certification to capture insights from qualified professionals. Additionally, to address developers' hesitation to discuss inappropriate practices at their companies during interviews, an environment was provided that encouraged participants to share real-world scenarios of tool usage, allowing for a more authentic and detailed expression of their experiences.

Construct threats are identified in Study 3, relating to the uniformity of resources, which may constrain the depth of the analysis. To mitigate this limitation, a variety of different resources were implemented to cover multiple infrastructures. This intensifies the experimental analysis and simplifies accessibility for replicating the experiment, as creating more complex structures can incur costs on AWS. This limitation is acknowledged, and it is suggested that future studies include variations in resource characteristics to allow for a broader and more representative analysis.

## 6 CONCLUSION

This study compared Terraform and CDK in managing IaC on AWS, evaluating aspects such as abstraction capability, scalability, maintainability, and performance. CDK proved to be more accessible for beginners with a programmatic structure that facilitates complex tasks, while Terraform, though accessible, presents challenges in more complex scenarios. Both tools require a modular approach in large projects; Terraform faces difficulties in team management for larger teams, while CDK demands careful organization. Terraform is faster in resource creation but consumes more CPU resources, whereas CDK is more resource-efficient. The choice between them depends on the project's needs and the team's familiarity.

As future work, we intend to explore two main lines of research. The first is the expansion of the current study to examine the efficiency of IaC tools in more complex scenarios, analyzing projects that integrate multiple services and dependencies. This will provide a deeper understanding of the capabilities and limitations of these tools in managing complex infrastructures. The second line of research proposes a broader comparative study, including IaC tools such as Ansible, Chef, Puppet, and CloudFormation, in addition to Terraform and CDK. Such a comparative analysis would enhance the understanding of the characteristics, efficiency, and limitations of these tools, assisting development and operations teams in making more informed choices and revealing innovation opportunities in the IaC area.

## REPLICATION PACKAGE

The replication package for this study is available at https://doi.org/10.5281/zenodo.11479674.

# REFERENCES


[1] Juncal Alonso, Radosław Piliszek, and Matija Cankar. 2023. Embracing IaC Through the DevSecOps Philosophy: Concepts, Challenges, and a Reference Framework. *IEEE Software* 40, 1 (2023), 56–62. https://doi.org/10.1109/MS.2022.3212194

[2] Justus Bogner and Manuel Merkel. 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)*. 658–669. https://doi.org/10.1145/3524842.3528454

[3] David Chappell. 2008. *A Short Introduction to Cloud Platforms: An Enterprise-Oriented View*. Technical Report. David Chappell & Associates. http://www.davidchappell.com/CloudPlatforms--Chappell.pdf

[4] Stefano Dalla Palma, Dario Di Nucci, Fabio Palomba, and Damian Andrew Tamburri. 2020. Toward a catalog of software quality metrics for infrastructure code. *Journal of Systems and Software* 170 (2020), 110726. https://doi.org/10.1016/j.jss.2020.110726

[5] MIchele Guerriero, Martin Garriga, Damian A. Tamburri, and Fabio Palomba. 2019. Adoption, Support, and Challenges of Infrastructure-as-Code: Insights from Industry. In *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 580–589. https://doi.org/10.1109/ICSME.2019.00092

[6] Manu Gupta, Mandepudi Nobel Chowdary, Sankeerth Bussa, and Chennupati Kumar Chowdary. 2021. Deploying Hadoop Architecture Using Ansible and Terraform. In *2021 5th International Conference on Information Systems and Computer Networks (ISCON)*. 1–6. https://doi.org/10.1109/ISCON52037.2021.9702299

[7] Gene Kim, Jez Humble, Patrick Debois, and John Willis. 2018. *Manual de DevOps: como obter agilidade, confiabilidade e segurança em organizações tecnológicas* (1a ed. ed.). Editora Alta Books, Rio de Janeiro. 427 pages.

[8] Patrick E. McKnight and Julius Najab. 2022. Mann–Whitney U Test. *The SAGE Encyclopedia of Research Design* (2022). https://api.semanticscholar.org/CorpusID:118856424

[9] Chris Parnin, Akond Rahman, and Laurie Williams. 2019. The Seven Sins: Security Smells in Infrastructure as Code Scripts. *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)* (2019), 164–175. https://doi.org/10.1109/ICSE.2019.00033

[10] Jagdish Chandra Patni, Souradeep Banerjee, and Devyanshi Tiwari. 2020. Infrastructure as a Code (IaC) to Software Defined Infrastructure using Azure Resource Manager (ARM). In *2020 International Conference on Computational Performance Evaluation (ComPE)*. 575–578. https://doi.org/10.1109/ComPE49325.2020.9200030

[11] Akond Rahman, Rezvan Mahdavi-Hezaveh, and Laurie Williams. 2019. A systematic mapping study of infrastructure as code research. *Information and Software Technology* 108 (2019), 65–77. https://doi.org/10.1016/j.infsof.2018.12.004

[12] Baishakhi Ray, Daryl Posnett, Vladimir Filkov, and Premkumar Devanbu. 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* (Hong Kong, China) *(FSE 2014)*. Association for Computing Machinery, New York, NY, USA, 155–165. https://doi.org/10.1145/2635868.2635922

[13] Julio Sandobalín, Emilio Insfran, and Silvia Abrahão. 2020. On the Effectiveness of Tools to Support Infrastructure as Code: Model-Driven Versus Code-Centric. *IEEE Access* 8 (2020), 17734–17761. https://doi.org/10.1109/ACCESS.2020.2966597

[14] Daniel Sokolowski and Guido Salvaneschi. 2023. Towards Reliable Infrastructure as Code. In *2023 IEEE 20th International Conference on Software Architecture Companion (ICSA-C)*. 318–321. https://doi.org/10.1109/ICSA-C57050.2023.00072

[15] Werner Vogels. acessado em 03 de março de 2023. Werner Vogels on the AWS Cloud Development Kit (AWS CDK). https://youtu.be/AYYTrDaEwLs

[16] Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, and Anders Wesslén. 2012. *Experimentation in software engineering*. Springer Science & Business Media.