Evaluating Infrastructure as Code: Key Metrics and Performance Benchmarks

Aditya Gupta
f20190338g@alumni.bits-pilani.ac.in
BITS Pilani, K K Birla Goa Campus
Goa, India

Paras Mittal
f20190183g@alumni.bits-pilani.ac.in
BITS Pilani, K K Birla Goa Campus
Goa, India

Dr. Kunal Korgaonkar
kunalk@goa.bits-pilani.ac.in
BITS Pilani, K K Birla Goa Campus
Goa, India

Abstract

Organizations are increasingly adopting Infrastructure as Code (IaC) to automate the management and provisioning of cloud resources, moving away from manual configurations. Despite its growing usage, there is a lack of comprehensive research evaluating the performance of IaC tools in large-scale, real-world scenarios with complex architectures. This paper addresses this gap by comparing Terraform and AWS CloudFormation, two leading IaC tools, across key performance metrics such as CPU usage, memory consumption, system time, and user time. Using the TrainTicket project, which encompasses 47 microservices, we evaluate both tools in a controlled environment to provide insights into their effectiveness in managing large-scale cloud infrastructures. Our findings offer valuable guidance for organizations choosing between Terraform and CloudFormation in enterprise-scale deployments.

1 Introduction

The adoption of cloud services has simplified resource provisioning but introduced challenges like manually managing cloud infrastructure, minimizing errors, and adapting to fast-evolving technologies. Manual provisioning can result in misconfigurations and slow down processes, especially in large, industry standard systems. To overcome these issues, Infrastructure as Code (IaC) has emerged as a solution, automating and standardizing cloud infrastructure management.

Two widely-used IaC tools are Terraform and AWS CloudFormation, each with distinct features. Terraform, developed by HashiCorp, is cloud-agnostic and supports multi-cloud environments, offering flexibility for organizations operating across multiple providers. AWS CloudFormation, in contrast, is optimized for AWS environments, providing seamless integration and faster performance for AWS-native tasks.

Terraform employs a multi-stage process (initialization of Terraform dependencies, planning updates, and application of updates) that gives users detailed control over infrastructure changes, though this can increase execution time and resource consumption. AWS CloudFormation, on the other hand, uses a simpler, single-stage process focused on speed and integration, which trades off some flexibility for faster task completion in AWS environments.

While both tools adhere to infrastructure immutability principles, their usability and performance differ significantly. Terraform excels in flexibility and multi-cloud support, while CloudFormation offers superior speed within AWS.

Organizations adopting IaC face the difficult decision of selecting the right tool for their specific requirements. While existing studies have explored developers' cost awareness when using tools like Terraform and AWS CloudFormation [2], and others have evaluated simpler applications on AWS CDK and Terraform [3], few have focused on large-scale, industry-relevant architectures. The majority of these studies lack an in-depth evaluation of performance, scalability, and usability metrics that are critical in enterprise environments, where infrastructure needs to scale quickly without sacrificing efficiency. Furthermore, these studies often fail to simulate the complexity and scale typical of real-world systems, leaving a gap in understanding how these tools perform under industrial conditions.

This paper aims to bridge that gap by providing a comprehensive comparison of Terraform and AWS CloudFormation. We evaluate both tools across key performance metrics such as system time, user time, total elapsed time, CPU usage, and memory consumption. Using the TrainTicket project [1], which simulates an industrial microservice system with 47 microservices, we provide insights into how these IaC tools perform in real-world, large-scale scenarios. Our findings offer practical guidance for organizations looking to make informed decisions about their IaC strategy, particularly when balancing the trade-offs between flexibility, performance, and multi-cloud support.

2 Methodology

In this study, we evaluated the performance of Terraform and AWS CloudFormation using the TrainTicket system, a large-scale microservice architecture consisting of 47 microservices. The TrainTicket system was chosen based on its relevance in simulating industry-scale microservice architectures, as highlighted in the work "Benchmarking Microservice Systems for Software Engineering Research" [4]. The infrastructure provisioning tasks, including creation, updates, and deletion, were conducted to reflect realistic usage patterns in large-scale deployments.

The 2nd Workshop on Hot Topics in System Infrastructure (HotInfra'24), co-located with SOSP'24, November 3, 2024, Austin, TX, USA

In this study, we measured several key performance metrics to evaluate the efficiency of Terraform and AWS CloudFormation. System Time refers to the duration spent by the operating system processing system-level calls during infrastructure provisioning, providing insight into computational overhead. User Time captures the CPU time dedicated to executing user-level instructions by the IaC tool. Together, these contribute to the Total Elapsed Time, which reflects the complete duration from the start to the completion of a task, including both system and user time, thereby indicating overall efficiency. We also recorded Peak Memory Usage, which represents the maximum memory consumed during the provisioning process, and CPU Usage (%), indicating the percentage of CPU resources utilized during operations, thus assessing resource efficiency. For AWS CloudFormation, we included the Additional Time to Reflect on Console, a metric that measures the delay between the completion of a command-line interface (CLI) process and the visibility of changes in the AWS console.

We used the Unix's "time" command to measure system time, user time, total elapsed time, peak memory usage, and % CPU used. The additional time for updates to reflect on the AWS console was noted manually for AWS CloudFormation, capturing the delay between the CLI completion and the changes appearing in the AWS Management Console.

It's important to note that these metrics solely focus on the resource provisioning aspect of both tools and do not include any time or performance impacts related to application code execution. The experiments were conducted on the AWS infrastructure, provisioning services like EC2 instances, VPCs, security groups, EBS volumes, IAM roles, CloudWatch log groups, and S3 buckets. The noted times reflect the performance of the CLI processes running on the host system, offering a direct comparison of Terraform and AWS CloudFormation's execution characteristics.

For Terraform, the stages included Initialization, Planning, Applying changes, Planning updates, Applying updates, and Destruction. In contrast, AWS CloudFormation followed a more streamlined approach with the stages being Creation, Updates, and Deletion. By measuring the performance of both tools across these stages, we captured how each tool handles infrastructure provisioning and modification tasks, providing insights into their scalability and resource utilization.

During the creation stage, we provisioned t2.micro EC2 instances for all 47 microservices, using default security groups and VPC settings. In the update stage, these instances were upgraded to t2.small, with additional infrastructure such as 10GB EBS volumes, S3 buckets, and CloudWatch log groups being provisioned, along with the necessary IAM roles. The deletion stage involved the teardown of all these resources.

The experiments were conducted in the ap-south-1 AWS region on a dedicated EC2 instance of the same class (t2.small) to ensure consistency across runs. Terraform and CloudFormation scripts were executed sequentially to avoid interference and ensure accurate metric capture. All metrics were averaged across the 47 microservices to provide a comprehensive comparison of the two tools' performance. This approach allows us to evaluate the overall efficiency of the tools when handling a large-scale, industry-oriented architecture.

3 Results

System Time, User Time, and Total Elapsed Time: Our analysis shows that Terraform consistently has higher system and user times across all operational stages—initial deployment, updates, and destruction. For instance, during initial deployment, Terraform's system time was 1.6 seconds, while CloudFormation's equivalent creation stage took just 0.14 seconds. During updates, Terraform's planning and application stages took 1.122 seconds and 1.026 seconds, respectively, whereas CloudFormation completed its entire update stage in 0.2 seconds. Similarly, Terraform's user time during initial deployment was 7.5 seconds, compared to CloudFormation's 0.84 seconds.

Total Elapsed Time: Terraform's total elapsed time was consistently higher. During initialization, Terraform took 19.21 seconds, while CloudFormation completed a similar task in 1.32 seconds. This difference was even more pronounced during the destruction phase, where Terraform took 86.79 seconds, compared to CloudFormation's 2.08 seconds.

Resource Utilization (CPU and Memory): Significant differences were observed in CPU usage and memory consumption. Regarding memory, Terraform's peak memory consumption during its planning phase was 302 MB, compared to CloudFormation, which required only 69 MB in any of its stages. During the initial deployment, Terraform's CPU usage started at 47% during initialization, dropping to 13% during the application phase, whereas CloudFormation's CPU usage was 72% during creation and spiked to 88% during both the update and deletion stages. The visual representations of this data are provided in Figure 1 and Figure 2. Please note that the values in these figures are averaged and represent the metrics for a single service.

Additional Time to Reflect on the Console (CloudFormation Only): For AWS CloudFormation, we also measured the additional time it took for changes to appear on the AWS Management Console after the CLI process was completed. This delay was most significant during updates, with an additional time of 33 seconds. In contrast, creation and deletion stages took 4.86 seconds and 11 seconds, respectively.

Key Takeaways: Terraform's multi-stage process introduces more overhead compared to CloudFormation's streamlined, single-stage approach. CloudFormation executes changes faster but lacks the planning transparency provided by Terraform. Terraform's longer times reflect the complexity of managing multi-cloud configurations and executing detailed

plans, while CloudFormation leverages AWS optimization to complete tasks quickly, though it sacrifices flexibility outside AWS. Terraform uses less CPU but consumes significantly more memory due to its state management and complex planning. CloudFormation, on the other hand, has higher CPU usage but requires much less memory due to its AWS-optimized, simpler state management. CloudFormation experiences a delay in reflecting changes in the AWS console, particularly during updates, due to the handoff of processes to AWS.

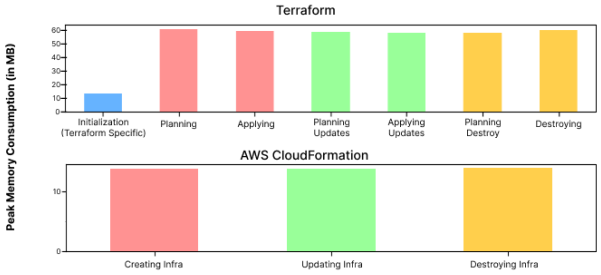

Figure 1. Memory Usage for Terraform and AWS CloudFormation for different stages.

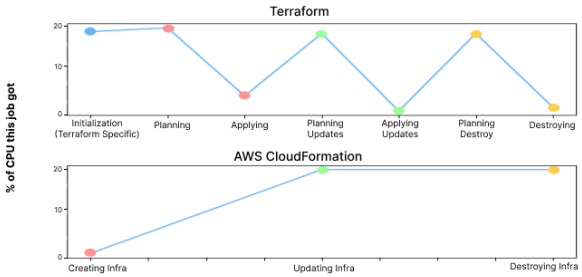

Figure 2. % CPU Usage for Terraform and AWS CloudFormation for different stages.

4 Conclusion and Future Work

This paper presents a data-driven comparison of Terraform and AWS CloudFormation in the deployment, updating, and destruction of 47 microservices on dedicated EC2 instances. The results indicate that CloudFormation consistently outperforms Terraform in speed, completing the initial creation in 1.32 seconds versus 19.21 seconds for Terraform. This trend continues in updates (CloudFormation: 2.118 seconds, Terraform: 122.4 seconds) and destruction (CloudFormation: 2.08 seconds, Terraform: 86.79 seconds), highlighting CloudFormation's efficiency for AWS-native operations.

However, Terraform excels in flexibility and multi-cloud support, offering detailed previews during its 5–6 second planning phases, which is vital for complex environments. Despite Terraform's higher peak memory usage (60,416 KB vs. CloudFormation's 13,906 KB), this trade-off is often justified for teams needing comprehensive execution planning.

In conclusion, organizations should choose CloudFormation for speed and efficiency in AWS-centric operations, while Terraform is preferable for those requiring multi-cloud capabilities and greater control over infrastructure changes. Ultimately, the selection should align with the organization's broader cloud strategy.

Several promising avenues for exploration remain. A critical area is the cost analysis of Terraform and AWS CloudFormation in managing large-scale infrastructures, as costs for API calls and resource updates can increase significantly. Understanding these implications over the long term would aid organizations in optimizing cloud expenditures.

Another focus is state management in Terraform. Our benchmarks suggest that as infrastructure scales, Terraform's state files become more complex, impacting performance and maintainability in collaborative settings. Researching these effects could enhance long-term usability.

Additionally, examining the performance of both tools in serverless and containerized architectures, such as AWS Lambda and Kubernetes, could provide insights into their effectiveness in modern cloud environments.

Comparing Terraform and CloudFormation with emerging IaC tools like Pulumi and AWS CDK could also highlight how new tools meet the demands of agile cloud development.

While this paper offers a detailed evaluation of Terraform and CloudFormation, future research could extend into cost efficiency, security, and the impact of evolving infrastructure patterns. This broader perspective will help organizations make informed choices when selecting their IaC tool.

References

[1] AdKrGu. 2024. GitHub - AdKrGu/train-ticket-benchmarking: This repository serves as a demo for benchmarking IaC tools. Forked from https://github.com/FudanSELab/train-ticket. Retrieved Oct. 15, 2024 from https://github.com/AdKrGu/train-ticket-benchmarking

[2] A.-I. Neamt. 2024. From Terraform to AWS CloudFormation: A Study of Cost Patterns and Antipatterns. *Student Theses Faculty of Science and Engineering* (2024). https://doi.org/33817/1/bCS2024NeamtAI.pdf

[3] A. Pessa. 2023. omparative study of Infrastructure as Code tools for Amazon Web Services. (June 2023). https://doi.org/handle/10024/149567

[4] Xiang Zhou, Xin Peng, Tao Xie, Jun Sun, Chenjie Xu, Chao Ji, and Wenyun Zhao. 2018. Benchmarking microservice systems for software engineering research. In *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings* (Gothenburg, Sweden) *(ICSE '18)*. Association for Computing Machinery, New York, NY, USA, 323–324. https://doi.org/10.1145/3183440.3194991