Comparative Assessment of Multiple IaC Tools for Provisioning Emulated SDN Topologies in Cloud Environments

Vitor Reiel M. Lima[1], **Arthur C. Callado**[1], **Michel S. Bonfim**[1]

[1]Programa de Pós-Graduação em Computação (PCOMP)
Universidade Federal do Ceará (UFC) - Quixadá, CE - Brasil

vitorreiel@alu.ufc.br, {arthur, michelsb}@ufc.br

Abstract. *This paper presents a comparative analysis of multiple Infrastructure as Code (IaC) approaches applied to the provisioning of emulated SDN topologies in cloud environments. The study considers seven widely used tools, evaluated on two SDN topologies with distinct structural characteristics, one symmetric and the other asymmetric. To this end, an experimental evaluation was conducted under controlled conditions, considering metrics related to time, computational resource usage, and stability across executions. The results reveal consistent differences in the operational behavior of the analyzed approaches, indicating that the execution architecture of each tool significantly influences scenario provisioning. This work contributes to expanding the empirical understanding of the use of IaC in container-based SDN emulation environments deployed and executed in the cloud.*

1. Introduction

Infrastructure automation has become a central element in the operation of modern computing environments, especially in scenarios that require repeatability, rapid deployment, and the reduction of manual errors. In computer networks, this need is even more evident, as the growing complexity of architectures, the diversity of components, and the recurrence of configuration changes increase the operational burden imposed on technical teams. In this scenario, practices associated with Infrastructure as Code (IaC) and NetDevOps have been pointed to as alternatives for making the provisioning, configuration, and maintenance of network environments more consistent, traceable, and efficient [Shah and Dubaria 2019, Salazar-Chacón and Parra 2023].

This transformation also changes how infrastructure is managed. By representing resources and procedures through code, the IaC approach promotes versioning, reuse, standardization, and systematic automation [Morris 2020, Bali and Walia 2023]. In the networking domain, this logic is articulated with NetDevOps, incorporating principles already consolidated in DevOps, such as automation, continuous integration, recurring validation, and change control, in contexts where operations require greater agility and less manual intervention [Shah and Dubaria 2019, Salazar-Chacón and Parra 2023]. It becomes even more pronounced in software defined networking (SDN) environments, in which the separation between the control plane and the data plane increases infrastructure programmability and, consequently, reinforces the need for automated mechanisms for the creation, modification, and removal of experimental scenarios [Lima et al. 2026].

As this programmability becomes consolidated in the network development and evaluation cycle, experimentation assumes a more prominent role. In this context, emulation stands out for enabling the construction of scenarios closer to real operation, with lower cost and greater flexibility than complete physical deployments. In studies on SDN, this characteristic makes it possible to instantiate, modify, and remove topologies in a controlled manner, favoring more systematic analyses of architectures, protocols, and management strategies. Using containers in Mininet-based environments has also expanded possibilities for cloud-based experimentation, enabling the construction of reproducible scenarios aligned with the dynamics of programmable environments [Lima et al. 2026].

The consolidation of these emulated environments increases the need to investigate the mechanisms responsible for their automation. This issue was initially addressed in [Lima et al. 2026], a study that investigated the provisioning of cloud-based emulated SDN topologies through a comparison between two IaC tools, with primary emphasis on execution time. Although that study produced important initial evidence, its results also indicated room for further investigation in some aspects: the incorporation of a broader set of metrics, the consideration of more representative structural scenarios, and the expansion of the diversity of evaluated approaches, in order to provide a less restrictive view of automation behavior in network emulation environments.

Building on this gap, this article proposes a comparative analysis of the use of IaC in provisioning cloud-based emulated SDN topologies, expanding the previous study in both scope and depth. The work seeks to provide a broader experimental evaluation capable of observing the operational behavior of different automation approaches across multiple performance dimensions and considering distinct topological scenarios (symmetric and asymmetric). The study aims to produce more consistent evidence on IaC usage in container-based emulation environments with cloud provisioning and deployment.

Based on this context, the study was guided by the following research questions:

RQ1. How do different IaC approaches behave in terms of the total provisioning time of cloud-based emulated SDN topologies?

RQ2. How is the provisioning time distributed across the installation, topology deployment, and convergence phases for each IaC approach?

RQ3. How do different IaC approaches behave in terms of computational resource consumption during provisioning, especially CPU and memory usage?

RQ4. To what extent does the evaluated topology structure influence the total time, resource consumption, and provisioning stability among the analyzed approaches?

The remainder of this paper is organized as follows. Section 2 presents the related work. The theoretical foundation is detailed in Section 3. The proposed methodology is described in Section 4. Section 5 discusses the experimental methods and the results obtained. Finally, Section 6 concludes the paper and presents directions for future work.

2. Related Work

The literature presents different initiatives aimed at infrastructure and network automation, although few of them are focused on the provisioning of cloud-based emulated SDN topologies. [Elradi 2023] discusses the use of Ansible for automating system deployment

and configuration tasks, whereas [Salazar-Chacón and Parra 2023] explore the application of IaC in open networks using Git, Ansible, and Cumulus Linux, highlighting programmability, versioning, and integration with NetDevOps practices. From a broader perspective, [Bali and Walia 2023] analyze IaC tools for efficiency, standardization, and scalability in IT environments. Together, these studies reinforce the importance of automation but do not compare multiple tools in container-based SDN emulation scenarios.

Some studies focus on the context of virtualized networks and cloud infrastructure. [Sicoe et al. 2022] combine Terraform and Ansible in the automated deployment of Cisco virtual routers in OpenStack-based environments, showing operational gains and reduced manual effort. In optical networks, [Dsilva et al. 2025] propose a NetDevOps pipeline with version control, prior validation, automated rollback, and support from a digital twin environment. Even so, these works are focused on device configuration and the operational management of specific infrastructures, rather than on a controlled comparison between IaC approaches in the provisioning of emulated SDN topologies.

There are also studies focused on recent limitations and directions in infrastructure automation. [Qiu et al. 2023] discuss weaknesses in the current IaC ecosystem in stages such as validation, updating, and debugging of cloud infrastructures, while [Bodnar et al. 2024] address challenges related to scalability, orchestration complexity, and resource usage. In another direction, [Shao et al. 2025] present an IaC generation service for the construction of industrial topologies. Closer to the present study, [Lima et al. 2026] compared Ansible and Terraform in the provisioning of cloud-based emulated SDN topologies, with primary emphasis on execution time. Although that work advanced by combining emulation, cloud environments, and tool comparison, its analysis remained restricted to two approaches, smaller topologies, and a reduced set of metrics.

The analyzed studies address infrastructure and network automation in different perspectives with distinct scopes regarding the object under evaluation and the form of analysis (see Table 1). [Lima et al. 2026], in the work closest to this proposal, do not bring together, within the same experimental design, a broader diversity of IaC approaches, more representative topological scenarios, and a multiple-metric evaluation. Unlike these studies, the present article expands this scope with a comparative evaluation focused on operational behavior of analyzed approaches under controlled, reproducible conditions.

3. Theoretical Framework

3.1. Infrastructure as Code

Infrastructure as Code is an automation approach in which infrastructure resources are defined, provisioned, and managed through machine-readable files rather than manual procedures [Morris 2020]. By treating infrastructure in a manner similar to software, IaC promotes standardization, reproducibility, and consistency in environment configuration. In the networking context, this approach is also articulated with NetDevOps practices, which apply automation, versioning, and change control to the management of network infrastructure [Shah and Dubaria 2019, Salazar-Chacón and Parra 2023].

3.2. Emulation of container-based networks

Network emulation makes it possible to build test scenarios with greater flexibility and lower cost than complete physical deployments, which makes it particularly useful in

Table 1. Summary of related work

Study	SDN	IaC	Objective	Main limitation
Elradi (2023)	No	No	Automation with Ansible	No focus on SDN and no IaC comparison
Salazar-Chacón and Parra (2023)	No	Yes	IaC in open networks	Does not compare IaC tools
Sicoe et al. (2022)	No	Yes	Virtual routers	No focus on emulated SDN and no comparison between approaches
Dsilva et al. (2025)	No	No	Pipeline for optical networks	No comparison of IaC tools in SDN scenarios
Qiu et al. (2023)	No	Yes	Infrastructure management	No experimental evaluation in networking
Bodnar et al. (2024)	No	Yes	IaC techniques and challenges	Does not consider emulated SDN topologies
Shao et al. (2025)	No	Yes	IaC generation	Focuses on code generation rather than comparison between approaches
Lima et al. (2026)	Yes	Yes	Ansible vs. Terraform comparison	Restricted to two tools, smaller topologies, and fewer metrics
This work	Yes	Yes	Comparison of IaC approaches	Expands the set of tools, scenarios, and analyzed metrics

SDN studies, where the use of containers expands the ability to create and reconfigure experimental topologies, favoring the execution of reproducible cloud-based environments. Among the solutions used for this type of experimentation, Containernet extends the Mininet model by employing Docker containers as hosts in emulated topologies, making it suitable for SDN studies in virtualization-based scenarios [Lima et al. 2026].

4. Methodology

This study adopts a quantitative experimental approach for comparing the behavior of seven IaC tools in the provisioning of cloud-based emulated software-defined networking environments. The methodology was designed to ensure comparability among the tools, isolation between executions, and reproducible collection of operational metrics. To this end, all tools perform the same task on the same remote environment: preparing a cloud instance, installing required dependencies, and deploying the chosen network topology.

4.1. Experimental environment

The experiments were conducted on AWS. Each iteration was executed on a new EC2 instance, created exclusively for it, and terminated at the end of the experimental cycle. This approach avoided interference between runs, reduced residual effects from previous executions, and increased reliability. The baseline environment for all executions consisted of instances in the `us-east-1` region, of type `c7i-flex.large`, with 2 vCPUs, 4 GB of RAM, Ubuntu 24.04 LTS operating system, 30 GB of storage on a gp3 volume, and open ports for SSH (22), OpenFlow (6653), and the controller interface (8181).

To ensure equivalent conditions across the evaluated tools, the minimum infrastructure required on AWS was created through a shared Terraform template, responsible only for preparing the remote execution environment. This step was not included in the calculation of the comparative metrics, since its purpose was not to evaluate the ability of the tools to provision the cloud provider's infrastructure, but rather to compare their

behavior in provisioning SDN topologies on a previously and uniformly available environment. This choice is also justified by the fact that not all evaluated tools are capable of fully provisioning cloud infrastructure, as several of them are primarily oriented toward the configuration and management of already available environments. For this reason, the measurements focused only on the provisioning of SDN topologies over an infrastructure that had been previously prepared and kept uniform across all scenarios.

The benchmark orchestration used a local machine equipped with an Intel Core i7-14650HX (2.20 GHz), 16 GB of RAM, 2 TB SSD, and an NVIDIA GeForce RTX 5050. This machine was responsible for triggering the main script, managing credentials, monitoring the lifecycle of the instances, and consolidating the results generated remotely into CSV files. It is worth noting that all processing relevant to the experiment, including dependency installation, topology creation, and metric collection, took place on the EC2 instances, thereby minimizing interference from the local environment.

4.2. Tools evaluated

Seven widely used tools were evaluated (see Table 2). The set was defined so as to encompass tools with distinct profiles, including solutions oriented toward resource orchestration and solutions focused on environment configuration. Also, the experiment explicitly recorded versions used to reduce methodological ambiguities and facilitate replication.

Table 2. IaC tools evaluated in the experiments

Tools	Category	Paradigm	Version
Terraform	Orchestration	Declarative (HCL)	1.14.8
OpenTofu	Orchestration	Declarative (HCL)	1.11.5
AWS CloudFormation	Orchestration	Declarative (YAML/JSON)	CLI 2.34.14
Pulumi	Orchestration	Imperative (Python)	3.227.0
Ansible	Config. Management	Declarative (YAML)	2.16.0
Puppet	Config. Management	Declarative (DSL)	7.34.0
Chef	Config. Management	Imperative (Ruby)	Client 18.4.12

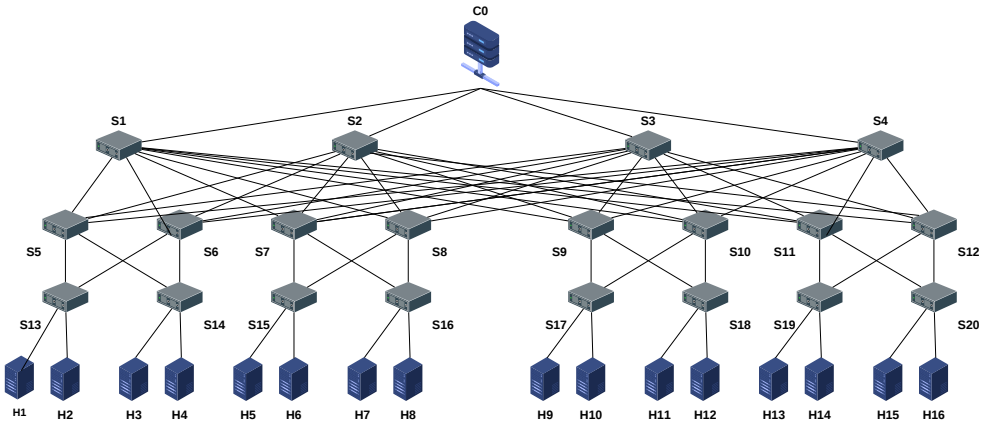
Regardless of the tool, all experiments executed the same sequence of actions on the instance: loading the Open vSwitch module, installing Docker and Docker Compose, transferring the topology artifacts, and initializing the containers corresponding to ONOS and Containernet. This standardization was essential to ensure that the comparison reflected differences in tool behavior rather than variations in the operational objective.

4.3. Topologies evaluated

The study used two SDN topologies with distinct structural characteristics. The first was a symmetric Fat Tree topology with three hierarchical layers, composed of four core switches, eight aggregation switches, eight edge switches, and sixteen hosts (Figure 1).

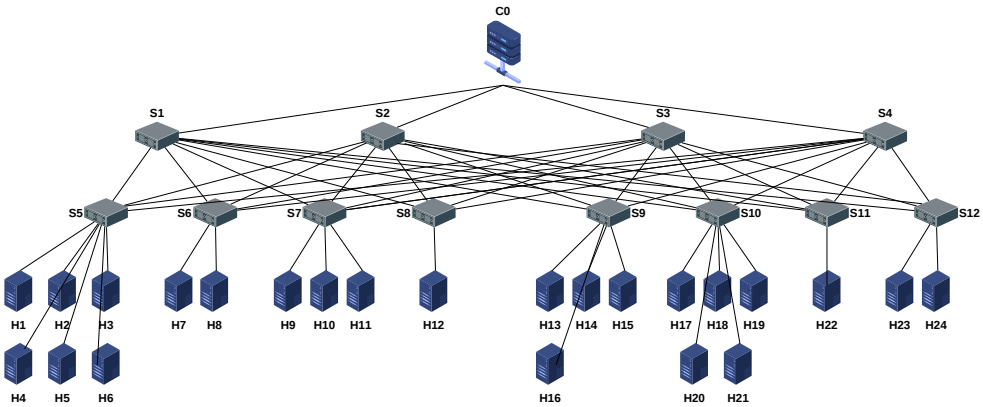
The second was an asymmetric Leaf-Spine topology composed of four spines, eight leaves, and twenty-four hosts unevenly distributed across access nodes (Figure 2).

The choice of these topologies stems from their structural differences. Fat Tree presents a more regular and hierarchical organization, widely associated with data centers and systematic interconnection. Conversely, Leaf-Spine introduces a less homogeneous



Source: Adapted from [Verma et al. 2024].

Figure 1. Fat Tree topology used in the experiments



Source: Adapted from [Huang et al. 2020].

Figure 2. Leaf-Spine topology used in the experiments

arrangement, with uneven distribution of hosts and a less symmetric organization. Thus, evaluation no longer depends on a unique structural configuration and instead observes the behavior of the tools under scenarios with distinct connectivity and complexities.

In both topologies, the ONOS controller ran in a Docker container, with supporting OpenFlow 1.3 applications for network operations and communication. Using the same controller in all scenarios helped unify the emulated network's logical control layer.

4.4. Experimental Procedure

The benchmark was automated by an orchestration script that repeats the execution flow for all tools and topologies. Before starting the measurement window, the remote environment was prepared by creating the instance, obtaining an IP address, checking SSH availability, and waiting for cloud-init completion. For Puppet and Chef, which required prior runtime preparation, it was performed before the measurement window, since the experiment focuses on SDN provisioning workflow rather than initial tool bootstrap cost.

Immediately before executing the tool, a background monitoring process was started on the remote instance. It recorded CPU and memory indicators at one-second intervals and captured initial network and disk counters. Next, the tool's start time was

recorded, after which it executed the dependency installation and topology deployment.

The tool's end time was recorded after the completion of the main command. From that point onward, an additional waiting phase for topology convergence was initiated. This convergence was verified through periodic queries to the ONOS API and was confirmed when the expected number of devices remained stable for three consecutive seconds. This step was included because the completion of the tool execution alone does not guarantee that the topology is fully operational from the controller's perspective.

Each tool has its own execution mechanism (Table 3), including SSH with provisioners, remote commands, SSM, playbooks, *puppet apply*, and *chef-solo*. Instead of eliminating differences, the experiment preserves them precisely because they are part of the way each tool materializes the same provisioning objective.

Table 3. Remote execution mechanisms adopted by each tool

Tool	Remote access mechanism
Terraform	Provisioners `file` and `remote-exec` via SSH
OpenTofu	Same mechanism adopted by Terraform
CloudFormation	AWS Systems Manager Run Command (SSM Run Command)
Pulumi	`pulumi-command`: `CopyToRemote` and `Command` via SSH
Ansible	Push-based model over SSH, agentless
Puppet	SCP and `puppet apply` via SSH
Chef	SCP and `chef-solo` via SSH

4.5. Metrics collected

Each iteration produced one line of results containing time, CPU, and memory metrics. Along the temporal dimension, the analysis considered total provisioning time, dependency installation time, topology deployment time, and convergence time, representing the interval until the emulated topology became operational and the ONOS API accessible. This enabled the decomposition of the temporal cost of each approach. CPU and memory usage were also recorded throughout execution, allowing the analysis of computational resource consumption and its relationship with total provisioning time.

4.6. Experimental scale and comparability

Each tool was executed 100 times on each topology, totaling 1,400 experimental iterations. The complete experiment execution took 64 hours and incurred an estimated cost of 11 US dollars on AWS. It is worth noting that this number of repetitions was adopted to reduce the influence of occasional environmental fluctuations and to enable robust analyses of the median behavior, dispersion, and stability of the results.

Finally, the methodology was designed to isolate the phenomenon of interest as much as possible: the behavior of IaC tools in provisioning emulated SDN topologies. Using a standardized baseline infrastructure, destroying the instance after each iteration, systematically repeating the scenarios, and automating the collection of metrics contributed to strengthening the consistency of the experiment and providing a more solid basis for the discussion of the results. To support transparency and reproducibility, all scripts, configurations, and datasets generated in this study are publicly available at GitHub[1].

[1] `https://github.com/vitorreiel/iac-sdn-provisioning-benchmark.git`

5. Experiments and Results

This section presents the results obtained from the experimental evaluation. The analysis considers three complementary perspectives: the decomposition of total execution time into distinct phases, the average consumption of computational resources, and the relationship between resource usage and total provisioning time. Finally, the threats to validity that may influence the interpretation of the results are discussed.

5.1. Breakdown of total provisioning time

Figure 3 presents the decomposition of total provisioning time into three phases: installation, topology creation, and convergence. In both topologies, Ansible showed the highest average total time, with approximately 302.6 s in Fat Tree and 310.4 s in Leaf-Spine. This behavior is mainly due to the high cost of the installation and topology creation phases, since Ansible executes tasks through remote SSH connections following a mostly sequential workflow, which is more sensitive to the latency accumulated throughout execution.

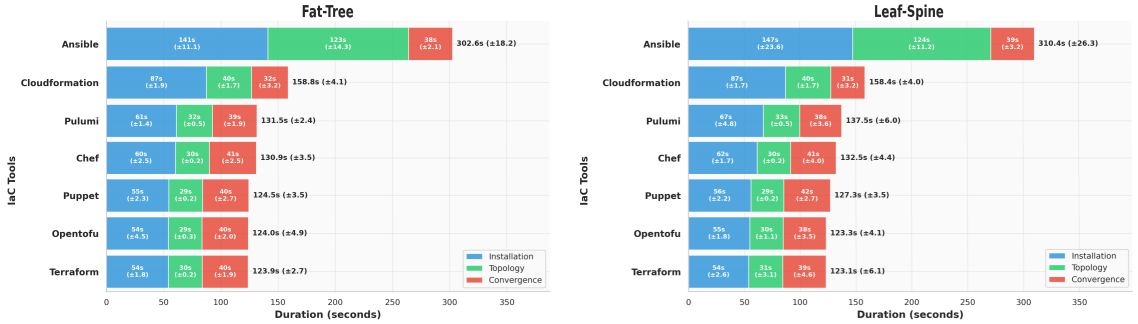


Figure 3. Total provisioning time Breakdown: Fat Tree vs Leaf-Spine topologies.

CloudFormation ranked second-to-last in total time, with about 158.4 s and 158.8 s. The main cost factor was the installation phase, which may be associated with its dependence on communication with the AWS API for operation validation and execution. Despite its declarative approach, part of the additional time stems from the very nature of the managed service, shifting part of the overhead to interaction with the infrastructure.

The shortest durations observed for Terraform, OpenTofu, and Puppet, ranging from approximately 123 s to 127 s, were possibly due to their more direct SSH-based execution flows. Chef and Pulumi formed an intermediate group, with times ranging from approximately 130 s to 137 s. In all cases, the convergence phase showed similar values, varying between about 31 s and 41 s, suggesting that this stage depends more on the stabilization of the SDN controller and the OpenFlow protocol than on the IaC tool.

The difference between topology outcomes was small for most tools, indicating the structural change between topologies was not sufficient to substantially alter total provisioning time. Ansible was an exception, showing greater variability across executions.

5.2. Computational resources

CPU usage (Figure 4) reveals distinct groups: Terraform, OpenTofu, Chef, Pulumi, and Puppet concentrated in the range of approximately 55% to 62% average utilization, whereas CloudFormation remained at an intermediate level, and Ansible showed the lowest consumption, around 23%. It suggests that tools with greater local processing consume

more CPU but take less time to complete provisioning: Terraform and OpenTofu make more intensive use of the control machine to calculate dependencies and plan execution. In contrast, Ansible consumes less local CPU because a large part of its execution depends on the coordination of remote tasks over SSH.

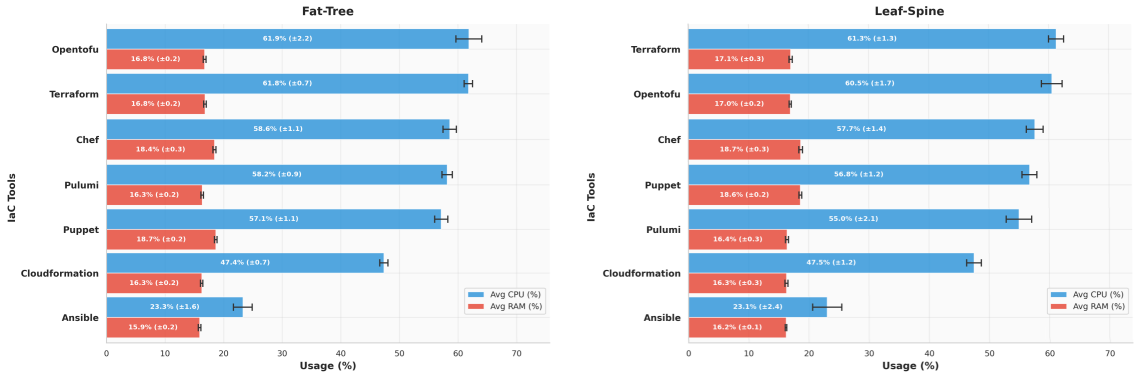


Figure 4. Average CPU and RAM usage during provisioning in the Fat Tree and Leaf-Spine topologies.

The memory usage differences (Figure 4) were much less pronounced. All tools remained approximately between 15% and 19% average RAM usage, indicating that, at the scale of the evaluated topologies, memory consumption did not constitute a decisive factor for differentiating approaches. Chef and Puppet presented slightly higher values, while Ansible and CloudFormation remained in the lower ranges. Even so, the dispersion was small across all tools, suggesting stable and predictable behavior.

An important aspect is that the change in topology did not substantially change CPU or RAM usage. The samples observed for Fat Tree and Leaf-Spine remained close, reinforcing the idea that resource consumption was determined mainly by the tools' execution architecture rather than by the structural differences between the topologies.

5.3. Resource use and total duration

Figure 5 shows the relationship between total duration and average CPU and RAM usage for each individual execution of the experiment. In the panel relating CPU to total time, a very clear negative trend can be observed: tools with higher average CPU usage tend to present shorter total duration. Although it may seem counterintuitive, in practice it indicates that intensive local processing was associated with more efficient execution.

Terraform, OpenTofu, Puppet, Chef, and Pulumi form a cluster concentrated in the region of higher CPU usage and lower execution time, whereas Ansible occupies the opposite region, with low CPU usage and high duration. CloudFormation appears in an intermediate position, reflecting an execution model that combines some local processing with significant dependence on communication with cloud services. Taken together, these clusters suggest that temporal efficiency was more strongly associated with the ability to run planning and orchestration steps locally than with reducing resource consumption.

In the panel relating RAM to total time, on the other hand, no significant correlation can be observed. The tools remain distributed within a relatively narrow memory range, while the execution times vary much more widely. This reinforces the interpretation that RAM was not a limiting factor in this experiment. In this graph as well, the

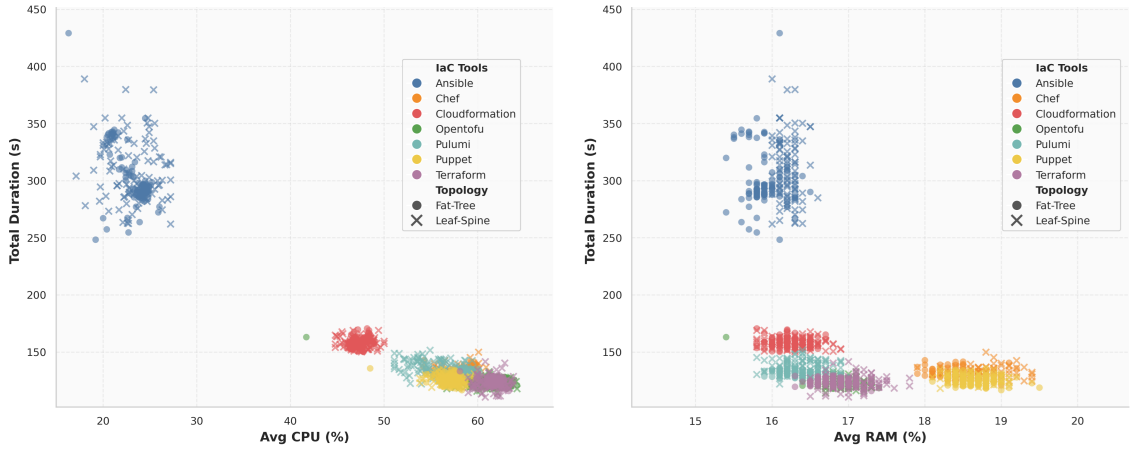


Figure 5. Relationship between resource usage and total provisioning duration in Fat Tree and Leaf-Spine topologies.

points corresponding to the Fat Tree and Leaf-Spine topologies remain strongly overlapped, which confirms that, for the workload considered, the difference between the topologies was not a determining factor for either resource consumption or total duration.

The dispersion of the points also provides indications of stability across executions. Ansible showed the greatest vertical spread, consistent with its higher temporal variability. In contrast, Terraform and OpenTofu formed more compact clusters, indicating greater predictability and more deterministic behavior throughout the repetitions.

5.4. Summary of results

The results make it possible to answer the proposed research questions consistently. Regarding RQ1, it was observed that Terraform and OpenTofu presented the best performance in total provisioning time, combining shorter duration and greater stability across executions. In contrast, Ansible recorded the longest times in both evaluated topologies, while Chef, Pulumi, Puppet, and CloudFormation occupied intermediate positions.

With regard to RQ2, the time decomposition showed that the differences among the tools are concentrated mainly in the installation and topology deployment phases. Convergence exhibited smaller variations across approaches, indicating a greater dependence on the SDN controller and the OpenFlow protocol than on the IaC tool. Ansible's higher total time was associated with the higher cost of the first two phases.

Regarding RQ3, the results indicated that the tools differed more markedly in CPU usage than in memory consumption. Terraform, OpenTofu, Chef, Pulumi, and Puppet exhibited higher average CPU usage, whereas Ansible showed the lowest consumption. In contrast, RAM remained within very similar ranges across the approaches, suggesting that, at the evaluated scale, memory did not constitute a decisive factor for differentiation.

Finally, in response to RQ4, the influence of topology was limited. In general, Fat Tree and Leaf-Spine produced very similar results for most tools, both in total time and resource consumption. This suggests that, in the analyzed scenario, the execution architecture of the tool played a more decisive role than the specific topology structure. The main exception was Ansible, which showed greater variability across executions, indicating higher sensitivity to the operational conditions of the environment.

5.5. Limitations and Threats to Validity

In terms of internal validity, although each execution used a new EC2 instance, variations in the cloud environment, network latency, and repository availability may have influenced part of the collected metrics. In addition, since the executions took place over several days, fluctuations in the load of the accessed services may also have affected the results. Moreover, because Puppet and Chef had their runtime/client installation performed outside the measurement window, their results exclude an overhead that would occur in a full cold-start deployment, which may favor these tools under that interpretation.

Regarding external validity, the generalization of the results is limited by the use of a single cloud provider in a single region, a single instance type, and only two data center topologies. Therefore, it cannot be assumed that the same relative behavior among the tools will be reproduced across other providers, machine profiles, or network scenarios.

From the construct validity perspective, the operation count depends on how tools report their actions, making direct comparisons difficult. Similarly, convergence time was measured based on the stability of the number of devices reported by ONOS, functioning as an indicator of topology readiness, but not guaranteeing end-to-end connectivity.

Finally, regarding conclusion validity, the analysis focused on descriptive statistics and graphical visualizations, without incorporating formal significance tests among the tools. Therefore, the results should be interpreted as comparative evidence of trends rather than as conclusive statistical proof of the observed differences.

6. Conclusions

This article presented a comparative evaluation of seven IaC approaches applied to the provisioning of cloud-based emulated SDN topologies. The results showed that the execution architecture of the tool exerted a decisive influence on total provisioning time, stability across executions, and resource usage profile. In this context, Terraform and OpenTofu presented the best temporal performance and the most stable behavior, while Ansible showed the longest total duration and the greatest variability.

The analysis also showed that the differences among the approaches were concentrated mainly in the installation and topology deployment phases, whereas convergence exhibited smaller variations across the tools. With respect to resource usage, it was found that greater local processing was associated with shorter total provisioning time, while memory consumption remained relatively homogeneous among the evaluated approaches. In addition, the difference between the Fat Tree and Leaf-Spine topologies had a limited impact on the relative behavior of the tools, indicating that, at the considered scale, the architecture of the approach had greater weight than the specific structure of the topology.

Thus, the study contributes to expanding the empirical understanding of the use of IaC in container-based SDN emulation environments executed in the cloud. By comparing multiple approaches across different performance dimensions and in distinct topological scenarios, the article advances beyond previous investigations that were more limited in scope and analytical depth. As future work, noteworthy directions include extending the analysis to other types of scenarios and infrastructure profiles, as well as incorporating inferential statistical analyses to further deepen the comparison among the approaches.

References

Bali, M. K. and Walia, R. (2023). Enhancing efficiency through infrastructure automation: An in-depth analysis of infrastructure as code (iac) tools. In *2023 Intl. Conference on Computing, Communication, and Intelligent Systems (ICCCIS)*, pages 857–863.

Bodnar, L., Bodnar, M., Shulakova, K., Vasylenko, O., Siemens, E., Tsarov, R., Yavorska, O., and Tyurikova, O. (2024). Advanced techniques for iac: Enhancing automation and optimization in cloud-based infrastructure management. In *Proceedings of Internatilnal Conference on Applied Innovation in IT*, volume 12, pages 19–25. Anhalt University of Applied Sciences.

Dsilva, N., Karunakaran, V., Autenrieth, A., Elbers, J.-P., and Bauschert, T. (2025). Network automation using netdevops in optical networks. In *2025 International Conference on Optical Network Design and Modeling (ONDM)*, pages 1–3.

Elradi, M. D. (2023). Ansible: A reliable tool for automation. *Electrical and Computer Engineering Studies*, 2(1):1–11.

Huang, J., Li, W., Li, Q., Zhang, T., Dong, P., and Wang, J. (2020). Tuning high flow concurrency for mptcp in data center networks. *Journal of Cloud Computing*, 9(1):13.

Lima, V., Callado, A., Bonfim, M., and Gonçalves, E. (2026). Comparative performance analysis of iac tools for deploying containerized sdn topologies in cloud environments. In *Anais do XLIV Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, pages 463–476, Porto Alegre, RS, Brasil. SBC.

Morris, K. (2020). *Infrastructure as code*. O'Reilly Media.

Qiu, Y., Kon, P. T. J., Xing, J., Huang, Y., Liu, H., Wang, X., Huang, P., Chowdhury, M., and Chen, A. (2023). Simplifying cloud management with cloudless computing. In *Proceedings of the 22nd ACM Workshop on Hot Topics in Networks*, HotNets '23, page 95–101, New York, NY, USA. Association for Computing Machinery.

Salazar-Chacón, G. and Parra, D. M. (2023). Infrastructure-as-code in open-networking: Git, ansible, and cumulus-linux case study. In *2023 IEEE 13th Annual Computing and Communication Workshop and Conference (CCWC)*, pages 1126–1131.

Shah, J. A. and Dubaria, D. (2019). Netdevops: A new era towards networking devops. In *2019 IEEE 10th Annual Ubiquitous Computing, Electronics Mobile Communication Conference (UEMCON)*, pages 0775–0779.

Shao, M., Liu, Z., Han, W., Gao, C., Liu, J., and Liao, Q. (2025). Intellitopo: An iac generation service for industrial network topology construction. In *2025 40th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 3605–3615.

Sicoe, A.-F., Botez, R., Ivanciu, I.-A., and Dobrota, V. (2022). Fully automated testbed of cisco virtual routers in cloud based environments. In *2022 IEEE International Black Sea Conference on Communications and Networking (BlackSeaCom)*, pages 49–53.

Verma, P. C., Atulkar, M., and Chouhan, R. K. (2024). Performance comparison of ryu controller in different sdn-enabled data center topologies. In *2024 Intl. Conference on Control, Computing, Communication and Materials (ICCCCM)*, pages 93–98.