Contents available at ITME Online Repository

INNOVATIVE TRENDS IN MULTIDISCIPLINARY ENGINEERING

International Peer-Reviewed Journal of the School of Engineering & Technology

K.R. Mangalam University, Gurugram

Journal Homepage: **https://itme.krmangalam.edu.in/**



Infrastructure Automation Using Terraform on Cloud Platforms

Bhavna Nayyer[a,*], **Annika Bajaj**[a], **Karamjot Kaur**[a], **Neha Singal**[a], **Piyush Kukreja**[a], **Preet Millu**[a]

[a] *Computer Science Department, Chandigarh University, Mohali, India*

DOI: 10.68343/ITME.2026.0102.6

ARTICLE INFO

Article history:
Received 13 March 2026
Revised 30 April 2026
Accepted 18 May 2026
Available online DD Month 2026

Keywords:
Infrastructure as Code
Terraform
Multi-Cloud Automation
Jenkins
Cloud Infrastructure Provisioning

ABSTRACT

In this paper, cloud infrastructure provisioning is discussed as an essential component of software systems, considering its complexity and potential for errors when performed manually or using cloud-specific templates. Considering that many organizations seek to deploy multiple cloud infrastructures to increase availability and avoid lock-in with any single cloud vendor, the need for a unified and automated approach to cloud provisioning becomes imperative. Therefore, the paper proposes a Terraform-based approach to the automated provisioning of virtual machine and network infrastructures across multiple heterogeneous cloud environments, including Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform (GCP). Moreover, the proposed approach follows an Infrastructure as Code approach while utilizing a CI/CD pipeline using Jenkins to ensure that the deployed process meets the needs of scalability and repeatability. A modular architecture is proposed using Terraform that enables the abstraction of cloud-specific configurations using cloud-specific Terraform plugins, thus ensuring that a single codebase is able to deploy equivalent infrastructure resources across multiple cloud environments. Experimental evaluation shows a significant improvement in efficiency when using the proposed approach compared to traditional approaches to cloud provisioning.

1. Introduction

The backbone of modern software ecosystems is cloud computing, which is the on-demand provision of computing resources such as computing, storage, and networking. The base layer of cloud computing is the virtual machine and the networking components such as the virtual private cloud, subnet, and security rules. Traditionally, the provisioning of cloud computing resources is done manually through the cloud console or cloud configuration scripts. However, manual provisioning is a tedious and labor-intensive task and is difficult to reproduce in a variety of environments [1]. With the rise of DevOps culture, automation has become an imperative concept to achieve speed and reliability in infrastructure deployment. Infrastructure as Code (IaC) has been recognized as an important concept that allows infrastructure resources to be defined through configuration files, which is helpful for automation and version control [2]. IaC helps to avoid configuration drift and reduces human error by ensuring that infrastructure is provisioned based on specific requirements rather than manual intervention. In recent years, there has been an increase in the use of multiple cloud infrastructures by organizations, which deploy their workloads across multiple cloud providers, including Amazon Web Services, Microsoft Azure, and Google Cloud Platform. This practice has been driven by the need to satisfy business needs such as availability, compliance, cost optimization, and vendor lock-in avoidance [3]. However, managing cloud infrastructures in heterogeneous cloud

* Corresponding author. E-mail: **nayyarbhavna@gmail.com**

© 2026 The Authors. Published by K.R. Mangalam University, Gurugram.
Selection and double-blind peer-review under the responsibility of School of Engineering & Technology.
This is an open-access article distributed under the terms of the Creative Commons Attribution License (CC BY 4.0).

environments is complex due to the fact that each cloud provider has its own API and toolsets [5]. Cloud-specific infrastructure templates have been identified as complicating factors in cloud management. The above issues are addressed by Terraform, an open-source tool for Infrastructure as Code. It provides a model for abstraction in the form of a provider-based model. This model supports cloud-agnostic definitions of infrastructure. With the use of Terraform, a similar syntax can be used for provisioning equivalent resources in different clouds. With the integration of Terraform and continuous integration and continuous deployment tools like Jenkins, the entire process of infrastructure provisioning can be automated. The current study proposes a framework for multi-cloud infrastructure automation using Terraform. The proposed framework focuses on the provisioning of virtual machine and network resources in AWS, Azure, and GCP clouds. It uses a combination of Terraform and a CI/CD tool like Jenkins for the automation of the entire lifecycle of infrastructure deployment. The key contributions of this paper are as follows: design of a unified multi-cloud infrastructure provisioning architecture using Terraform; integration of Jenkins pipelines to automate Infrastructure as Code workflows; implementation and evaluation of VM and network provisioning across AWS, Azure, and GCP infrastructures; and a comparative analysis demonstrating reductions in deployment time and configuration complexity. The rest of the paper is organized as follows: Section 2 discusses related work related to infrastructure as code and multi-cloud automation. Section 3 discusses the problem statement and motivation. Section 4 discusses the proposed multi-cloud automation architecture. Section 5 discusses the implementation of the proposed solution using Terraform. Section 6 discusses the experimental results, and Section 7 concludes the paper with future research directions.

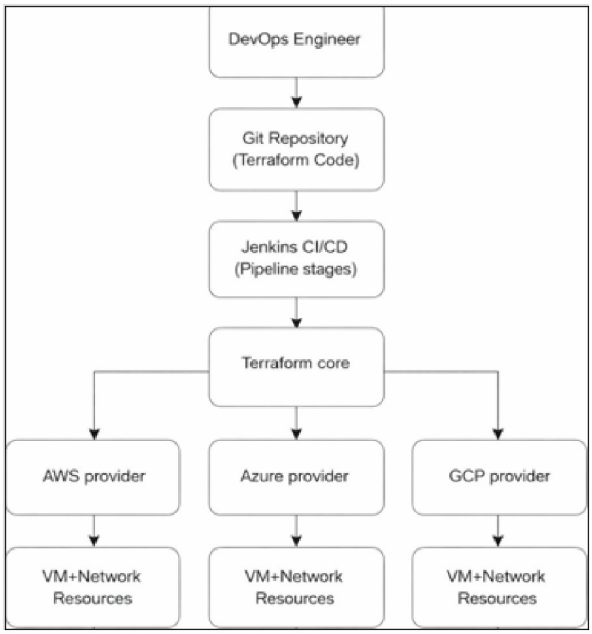


Figure 1. Terraform-based multi-cloud infrastructure automation architecture using Jenkins CI/CD

2. Literature Review

The growing popularity of cloud computing technology has led to increased interest in the importance of infrastructure automation, Infrastructure as Code (IaC), and multi-cloud computing in the literature. This section presents an overview of the available literature on the IaC model, multi-cloud infrastructure management model, and infrastructure automation using a CI/CD pipeline, and the gaps addressed by the proposed system.

2.1. Infrastructure as Code (IaC)

“Infrastructure as Code” refers to a software development practice that includes the definition, delivery, and management of resources through machine-readable code. This concept provides repeatability, version control, and automation, which greatly reduces the chances of human error during the configuration process. In addition, as Morris [2] indicates, IaC, especially through the use of declarative code, provides the ability to define the state of the infrastructure without the need to write any procedural code, thus enabling the

infrastructure to converge into a state that has been defined. Various IaC tools have been proposed and used in practice, including AWS CloudFormation, ARM, and Terraform. Cloud-based IaC solutions like CloudFormation and ARM provide better integration with their respective cloud environments but often result in lock-in due to their cloud-specific syntax and resource definition. Terraform, on the other hand, provides a cloud-agnostic solution using its cloud provider architecture, which supports the provisioning of cloud infrastructure using a unified configuration syntax. Previous studies show that using Terraform's declarative approach provides better consistency and portability of infrastructure, while making environment replication easier across the software development life cycle. However, most studies have been based on single-cloud environments, while little emphasis has been given to the use of Terraform in multi-cloud environments.

2.2. Multi-Cloud Infrastructure Management

Implementing multi-cloud strategies involves deploying infrastructure and workload distribution to multiple cloud providers to ensure higher reliability, minimize vendor lock-in risks, and optimize cost-effectiveness and performance. A detailed survey of cloud migration strategies and multi-cloud challenges is provided in Jamshidi et al. [5], which identifies key challenges as increased complexity, heterogeneous APIs, and configuration complexity. Existing multi-cloud infrastructure management practices heavily rely on cloud-specific orchestration tools or middleware solutions, which sometimes add complexity to infrastructure abstraction layers. Some authors recommend unified cloud management solutions; however, such solutions lack detailed infrastructure resource control or do not support unified resource provisioning for heterogeneous environments [7]. Terraform's abstraction model for cloud providers is identified as a key advantage for managing multi-cloud infrastructure; it allows for applying equivalent infrastructure definitions for multiple cloud platforms [4]. However, limited studies have been conducted to test Terraform-based infrastructure provisioning for multi-cloud environments, specifically for fundamental infrastructure components such as virtual machines or networks.

2.3. CI/CD and Infrastructure Automation

Continuous Integration and Continuous Deployment (CI/CD) pipelines have been recognized as a cornerstone in the application of the DevOps paradigm, enabling the automated testing and deployment of software systems. More recently, research literature has emphasized the application of CI/CD principles in the provisioning of software infrastructures, referred to as "Infrastructure CI/CD" [6]. By integrating Infrastructure-as-Code tools into CI/CD pipelines, software infrastructures are provisioned in a consistent and reproducible manner. Jenkins remains a popular CI/CD tool in the industry owing to its extensibility and plugin capabilities in supporting infrastructure provisioning flows. Research literature has emphasized the benefits of CI/CD-based infrastructure provisioning in terms of reduced deployment time and increased reliability [8]. However, a majority of the literature focuses on the application of CI/CD in the deployment of software applications rather than the provisioning of software infrastructures across multiple cloud platforms. Although industrial research literature has been devoted to the application of Jenkins in integrating with Terraform in the provisioning of software infrastructures, the academic research community remains largely silent on the architectural and performance implications of such an integration in the context of multiple cloud platforms. In this paper, we propose a Jenkins-based CI/CD pipeline integrating Terraform in the automated provisioning of virtual machine and network resources in AWS, Azure, and GCP environments.

2.4. Research Gaps and Motivation

From the surveyed literature, the following gaps were identified:

1. **Lack of infrastructure-level multi-cloud provisioning:** Most of the literature focuses on application orchestration or container-based infrastructure, with less attention given to infrastructure-level resources like virtual machines or networks.
2. **Vendor-specific automation dominance:** Current cloud native IaC tools do not support portability and make users dependent on specific vendors.

3. **Lack of Terraform effectiveness evaluation in multi-cloud environments:** Although Terraform is widely used, its effectiveness is rarely evaluated empirically for multi-cloud environments.
4. **Lack of integration of CI/CD-driven infrastructure workflows:** Very few literature studies propose end-to-end automation of infrastructure provisioning using IaC and CI/CD pipelines.

To address these gaps, this paper proposes a Terraform-based multi-cloud infrastructure automation framework with integration of Jenkins CI/CD pipelines. It focuses on infrastructure provisioning of virtual machines and networks, which is highly relevant to today's DevOps environments.

3. Problem Statement and Motivation

In this study, the concept of infrastructure provisioning is considered an essential part of modern-day cloud computing environments, which is typically managed using manual or configuration management approaches specific to cloud environments. Manual management of virtual machine or network infrastructure provisioning through provider-specific consoles is considered cumbersome, error-prone, and difficult to replicate for multiple environments. Although automated approaches using configuration management tools are available, these are typically specific to individual cloud environments, which makes them less portable. In order to address the challenges of managing multiple environments with multiple cloud service providers, organizations face major challenges in managing heterogeneous environments. Each of the cloud service providers exposes unique APIs, resource definitions, or configuration management mechanisms, which makes it challenging to maintain uniform infrastructure configurations for multiple environments. Such non-standard approaches result in increased complexity and make it challenging to maintain uniform infrastructure configurations for multiple environments. Currently, most of the available automation mechanisms are limited to application or container management, with less attention given to virtual machine and network infrastructure management. Moreover, existing approaches do not support infrastructure management with respect to CI/CD pipelines.

Motivation of Study. The motivation of this study is to address the challenges of managing multiple environments with multiple cloud service providers by adopting a unified Infrastructure as Code management approach for managing multiple environments. Terraform is considered an effective infrastructure management tool that supports multiple cloud environments using its provider abstraction mechanism, which allows infrastructure resources to be provisioned uniformly using a single codebase for multiple environments. Terraform integration with Jenkins-based CI/CD pipelines makes it possible to automate infrastructure management, which is considered scalable and repeatable. By focusing on infrastructure management with respect to virtual machine or network infrastructure provisioning for multiple environments like AWS, Azure, or GCP, it is possible to address challenges with respect to infrastructure management for multiple environments.

4. Proposed Multi-Cloud Automation Architecture

In this section, the architecture of the proposed system for automating the provisioning of multi-cloud infrastructure with the help of Terraform and Jenkins is described. It is designed to provide a unified, scalable, and vendor-neutral mechanism for provisioning virtual machine and network resources.

4.1. Architectural Overview

The architecture proposed in this document uses a DevOps-centric approach to automation, which uses Infrastructure as Code (IaC) to configure infrastructure and a CI/CD pipeline to execute it. There are five major components to this system: the DevOps engineer, version-controlled Terraform configurations, a CI/CD pipeline built using Jenkins, Terraform core, and Terraform provider plugins. The system starts with a DevOps engineer defining infrastructure needs using Terraform configurations written in code. The code is stored in a repository that uses Git, allowing it to have version control. Once the code is committed into this repository, it triggers a CI/CD pipeline that uses Jenkins to run the Terraform code to create infrastructure in a consistent manner without human intervention, ensuring that it is done correctly without any human error.

4.2. Terraform Core and Provider Abstraction

At the heart of this system is the Terraform core, which reads the configuration files. The architecture of Terraform revolves around its abstraction model, which is based on the concept of providers. This allows the user to manage different cloud platforms through plugins for different providers. In the proposed architecture, different providers are configured for different cloud platforms, including Amazon Web Services, Microsoft Azure, and Google Cloud Platform. Each of these providers makes API calls to the respective cloud platforms, which allows the user to create similar resources in different environments. This allows the user to create their own Terraform code, which can be used for the provisioning of different resources, including virtual machines, in different environments.

4.3. Jenkins CI/CD Integration

Jenkins is used as an engine for automating tasks to achieve Infrastructure as Code using its integration with the DevOps model. The purpose of the pipeline in Jenkins is to run Terraform commands to achieve infrastructure initialization, validation, planning, and applying changes to the infrastructure environment. By integrating Terraform execution with a CI/CD pipeline, it ensures that infrastructure delivery is standardized, traceable, and reproducible. In addition, it ensures that Terraform configurations are validated, thus eliminating any possible errors that would be present in the infrastructure environment. This way, it aligns with current-day DevOps practices.

4.4. Multi-Cloud Resource Provisioning

The last layer in the architecture is the provisioned cloud resources. With the help of the proposed framework, virtual machines and network resources such as virtual networks, subnets, and security rules can be created in AWS, Azure, and GCP. This is because of the abstraction provided by the Terraform provider. Figure 1 presents the architecture of the proposed framework in automating multiple cloud platforms. In the figure, the interactions between Jenkins, the Terraform core, and the cloud resources are emphasized. The architecture can be used in the provisioning of scalable and reliable cloud resources.

5. Implementation Using Terraform

In this section, the implementation of the proposed framework for automating the multi-cloud infrastructure using the Terraform tool and Jenkins is discussed. The implementation in this section is mainly based on the automation of virtual machine and network services in AWS, Azure, and Google Cloud.

5.1. Terraform Module Design

To ensure reusability of the code, the infrastructure has been organized through the use of Terraform modules. Separate modules are created for the provisioning of virtual machines and the configuration of the network. These modules are designed in a way that makes the configuration of the same type of infrastructure in different cloud environments almost similar. The input variables are used to parameterize the configuration of the instance type, region, network CIDR block, and rules for the security group. These properties may include the identifiers of the virtual machines deployed in the cloud.

5.2. Provider Configuration for Multi-Cloud Deployment

Terraform provider blocks are defined on a per-cloud basis to support the provisioning of multiple cloud platforms. The configuration of the providers also differs for each cloud, such as AWS, Azure, and GCP. The configuration details include the credentials for the cloud platforms, the region in which the resources are deployed, and the project details for the cloud platforms. The use of Terraform’s provider abstraction ensures the separation of the cloud-specific parameters from the generic infrastructure configuration. This separation is also helpful if the framework is to be expanded to other cloud platforms in the future.

5.3. Jenkins CI/CD Pipeline Integration

The role that Jenkins plays is to automate the execution of the Terraform workflow within the context of the

continuous integration/continuous deployment pipeline. This pipeline is initiated based on changes to the Terraform configuration code repository. This process has various stages, including initialization, validation, planning, and application of changes to infrastructure. In the initialization process, dependencies and providers are set up. In the validation process, Terraform checks the syntax as well as semantic correctness. In the planning process, an execution plan is generated, which allows one to visualize changes to infrastructure that will be deployed. In the last process, resources are created in the cloud environments. This process is automated by Jenkins, thereby ensuring that this process is aligned with the best practices for provisioning of infrastructure, as ensured by the DevOps paradigm.

5.4. Terraform State Management

The management of Terraform State is of critical importance in the management of the current state of the deployed infrastructures. In the proposed solution, a remote backend is utilized for the safe storage of the Terraform State File. The remote backend for the storage of the Terraform State enables collaboration among different users. It also eliminates conflicts that may occur when there are simultaneous changes to the Terraform State. State locking is utilized to ensure that there is only one execution of the pipeline that changes the infrastructure at a given time.

6. Results and Evaluation

In this section, the effectiveness of the proposed Terraform-based multi-cloud infrastructure automation framework is evaluated. The evaluation of the effectiveness of the proposed solution is based on the efficiency of the deployment process, consistency of the deployment process, and portability of the deployment process. The effectiveness of the proposed automated solution is compared to the effectiveness of the traditional manual solution.

6.1. Experimental Setup

To validate the suggested system, various infrastructures were built on AWS, Azure, and GCP. This infrastructure consists of a virtual machine, various network-related components such as virtual network, subnet, security, etc. This infrastructure creation process was done manually through the cloud provider consoles as well as through the automated process using Terraform with Jenkins CI/CD. The time taken to deploy the infrastructure was measured as the time taken from the start of the provisioning process to the time when the virtual machine was available for use. The process was run several times to obtain consistent results.

6.2. Deployment Time Analysis

Table 1 shows the comparison of deployment time between manual provisioning and the proposed method for the three cloud platforms.

Table 1. Deployment Time Comparison

Cloud Platform	Manual Provisioning (minutes)	Terraform-Based Automation (minutes)
AWS	30	5
Azure	35	6
GCP	28	5

The results show that there is a significant reduction in deployment time by using Terraform-based automation. On average, the automated process showed a reduction of 80–85% less time compared to manual methods. This is due to the avoidance of manual repetitive steps and the use of predefined configurations provided by Terraform.

6.3. Qualitative Evaluation

In addition to time taken for deployment, the proposed framework was also qualitatively evaluated in terms

of repeatability, consistency of configurations, and portability. Table 2 shows the qualitative comparison of manual provisioning and automation using Terraform. Manual provisioning often resulted in configuration inconsistencies due to human error and variations in deployment steps. In contrast, Terraform-based automation ensured consistent infrastructure deployment by enforcing declarative configurations and automated execution through Jenkins pipelines.

Table 2. Qualitative Comparison of Provisioning Approaches

Metric	Manual Provisioning	Terraform-Based Automation
Repeatability	Low	High
Configuration Errors	Frequent	Minimal
Portability	No	Yes
Scalability	Limited	High
Auditability	Low	High

6.4. Discussion

As indicated by the evaluation findings, the proposed multi-cloud automation framework effectively addresses the identified challenges associated with the use of traditional manual and cloud-oriented provisioning approaches. For instance, the significant improvement in deployment time highlights the efficiency benefits that can be achieved through the use of Infrastructure as Code and CI/CD-based automation approaches. Moreover, the capability to deploy equivalent infrastructures on AWS, Azure, and Google Cloud environments using a single Terraform codebase highlights the portability and vendor-neutral characteristics of the proposed solution. Despite the successful evaluation findings, there exist some limitations associated with the proposed approach. Firstly, the proposed evaluation process targets a restricted range of infrastructural resources, including virtual machines and networking resources. Moreover, the evaluation process was performed in a controlled environment, which may not reflect the complex environment that may exist in a large-scale enterprise environment.

7. Conclusion and Future Work

The present study proposes a Terraform-centric framework for automating multi-cloud infrastructure provisioning through CI/CD pipelines developed using Jenkins. This approach helps to overcome several issues that arise when using conventional infrastructure provisioning mechanisms, including those for cloud environments. In particular, it helps to ensure consistency in infrastructure configurations, reduces delays in infrastructure provisioning, and minimizes vendor lock-in risks for cloud-based infrastructure provisioning mechanisms. By adopting Infrastructure as Code and Terraform's abstraction mechanism via providers, this approach offers a unified mechanism for infrastructure provisioning for virtual machines as well as network infrastructure in Amazon Web Services, Microsoft Azure, and Google Cloud Platform.

As indicated in the implementation results, it is clear that Terraform-based infrastructure provisioning integrated with Jenkins CI/CD pipelines greatly enhances infrastructure provisioning efficiency and reliability compared to conventional manual infrastructure provisioning methodologies. Experimental results indicate that this approach reduces infrastructure provisioning time as well as errors compared to conventional manual infrastructure provisioning methodologies. Although this approach has several advantages, it is limited to a set of infrastructure components in the current implementation scenario. Future work will focus on extending this approach to accommodate a wider range of infrastructure components for infrastructure provisioning mechanisms. Some possible extensions to this approach include incorporating policy as code for infrastructure provisioning to ensure security as well as cost optimization strategies for infrastructure components, as well as infrastructure component monitoring for dynamic scaling. To conclude, it is clear that this proposed multi-cloud infrastructure provisioning mechanism demonstrates that Terraform, when integrated with Jenkins-based CI/CD pipelines, offers a promising approach for infrastructure provisioning in a multi-cloud environment. This approach helps to minimize infrastructure complexity while offering a unified mechanism for infrastructure provisioning in heterogeneous cloud environments.

References

[1] M. Villamizar, O. Garces, H. Castro, L. Salamanca, R. Casallas, S. Gil, C. Valencia, A. Zambrano, and M. Lang, "Infrastructure as Code: A Systematic Literature Review," *Journal of Systems and Software*, vol. 186, pp. 1–15, 2022.

[2] K. Morris, *Infrastructure as Code: Managing Servers in the Cloud*, 2nd ed. Sebastopol, CA, USA: O'Reilly Media, 2021.

[3] RightScale (Flexera), "State of the Cloud Report," Flexera, Santa Barbara, CA, USA, 2023.

[4] HashiCorp, "Terraform: Infrastructure as Code," HashiCorp Documentation, 2024. [Online]. Available: https://developer.hashicorp.com/terraform

[5] P. Jamshidi, C. Pahl, N. C. Mendonça, J. Lewis, and S. Tilkov, "Cloud Migration Research: A Systematic Review," *IEEE Transactions on Cloud Computing*, vol. 10, no. 2, pp. 1234–1247, Apr.–Jun. 2022.

[6] J. Humble and D. Farley, *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Boston, MA, USA: Addison-Wesley, 2019.

[7] A. Pahl, "Containerization and the PaaS Cloud," *IEEE Cloud Computing*, vol. 7, no. 3, pp. 24–31, May–Jun. 2021.

[8] L. Bass, I. Weber, and L. Zhu, *DevOps: A Software Architect's Perspective*. Boston, MA, USA: Addison-Wesley, 2020.

[9] Amazon Web Services, "AWS Well-Architected Framework," AWS Whitepaper, 2023. [Online]. Available: https://docs.aws.amazon.com/wellarchitected

[10] Microsoft, "Azure Architecture Framework," Microsoft Learn Documentation, 2023. [Online]. Available: https://learn.microsoft.com/azure/architecture

[11] Google Cloud, "Google Cloud Architecture Framework," Google Cloud Documentation, 2023. [Online]. Available: https://cloud.google.com/architecture/framework