Jurnal Teknologi dan Manajemen Industri Terapan (JTMIT) Vol. 4, No. 2, Juni 2025 pp. 647 - 652
P-ISSN: 2829-0232 E-ISSN: 2829-0038

IT Infrastructure Automation Management with Infrastructure-As-Code (IAC) Modelling in Multi-Cloud Environments

Agus Sulaiman
Teknik Informatika, Sekolah Tinggi Manajemen Informatika Komputer Tazkia (STMIK Tazkia), Indonesia
Jl. Raya Dramaga Km 7 Bogor 16680, Indonesia
Email: agus@stmik.tazkia.ac.id

ABSTRACT

Digital transformation has driven the adoption of multi-cloud environments, defined as the use of computing services from two or more public cloud providers (e.g., AWS, Azure, GCP) to mitigate vendor lock-in risks, optimize costs, and meet geographic requirements. However, multi-cloud environments pose significant management challenges, characterised by disparate API complexity, configuration discrepancies, and scalability bottlenecks that hinder rapid service delivery. This study aims to evaluate the effectiveness of Infrastructure-as-Code (IaC) modelling (using tools such as Terraform and Ansible) as a primary mechanism for automated management of IT infrastructure in complex multi-cloud architectures. The study uses a case study and experimental approach to compare metrics such as deployment time, configuration drift rate, and code portability when managing load balancers and distributed networks across three major cloud providers. The research hypothesis states that leveraging IaC, specifically platform-agnostic tools, significantly reduces operational overhead and improves infrastructure consistency (immutability) across different clouds compared to manual or console-based provisioning methods. The findings are expected to provide a practical framework and best practices for DevOps teams and Cloud engineers to achieve efficient automation and uniform governance in a multi-cloud ecosystem.

Keywords: *Infrastructure-as-Code (IaC), Multi-Cloud, Infrastructure Automation, Terraform, DevOps, Vendor Lock-in, Immutability.*

Introduction

Digital transformation has fundamentally changed the business operations paradigm, driving the urgent need for IT infrastructure that can adapt quickly to volatile market demands. Since the beginning of the millennium, *cloud computing* has become the primary foundation for agility. This provides on-demand computing resources and operates on a pay-as-you-go model. However, this rapid evolution is taking organizations beyond using a single provider. *Cloud single (such as AWS or Azure) towards an architecture known as a multi-cloud* environment. A multi-cloud environment is defined as the use of cloud computing services distributed across two or more providers and public clouds [1]. This strategic decision was driven by several critical factors: risk mitigation, vendor lock-in, cost optimization through price arbitrage across providers, fulfilment of requirements (residency data or strict geographical regulations), and the need for redundancy and disaster resilience (disaster recovery) [2].

Despite offering strategic advantages, multi-cloud architectures inherently create massive management complexity. The main challenge lies in API and *tooling disparities*. Each provider cloud has a unique application programming interface (API) framework, configuration language, and operational models. This forces operations teams to maintain deep, specialized expertise for each platform, significantly increasing operational overhead. Besides that, Manual provisioning of infrastructure in a multi-cloud environment is vulnerable to configuration drift, in which the configuration server, network, or storage deviates from its ideal state over time, creating security gaps and inconsistencies that are fatal to scalability and compliance [3]. These challenges often defeat the initial goal of adoption. *Cloud*, namely, increasing speed of deployment [4].

To address the challenges of multi-cloud complexity, practiceIT Infrastructure Automation Management has grown rapidly. The essence of this automation is the adoption of Infrastructure-as-Code

(IaC). IaC is the practice of managing and provisioning IT infrastructure through code, using machine-readable definition files rather than physical hardware configuration or interactive configuration tools. The basic concept of IaC revolves around the principles of Immutability (infrastructure is not changed; it is replaced with a new version) and Idempotence (operations that run the same code multiple times result in the same final infrastructure state) [5]. With IaC, *code for virtual machines, networks, and load balancers can be treated the same as code for an application: stored in a Git repository, tested, reviewed, and implemented through a CI/CD pipeline*. This transforms infrastructure management from an ad hoc, manual practice to a software engineering discipline, is predictable, and can be repeated (*reproducible*).Tools such as Terraform (declarative, platform-agnostic) and Ansible (imperative, focused on configuration management) are the backbone of modern IaC modelling [6].

The Research Gap. Despite the benefits of IaC in the single-cloud environment having been well documented, academic literature still has significant gaps in empirical and quantitative evaluation regarding the actual effectiveness of IaC tooling in addressing the unique complexities of a heterogeneous multi-cloud environment. Existing studies are often qualitative or focused on comparisons. *tool*in one *platform*(for example, CloudFormation vs Terraform on AWS) [7].However, the crucial question is: How effective is it? *Tooling like Terraform, with abstraction-provider-agnostic, in reducing cognitive and technical overhead caused by API differences between AWS, Azure,* and GCP? Is the promise of portability really true, or is the configuration unique (such as network peering or security groups) still extensive enough to require modification, thereby reducing the claimed automation efficiency? This gap creates an urgent need for studies that rigorously measure operational performance metrics such as deployment time and configuration drift level across different multi-cloud environments [8].

Research purposes: Therefore, this research aims to evaluate experimentally and comparatively the effectiveness of IaC modelling in IT infrastructure automation management in multi-cloud environments. Specifically, this research will [9]:(1) Measure and compare deployment time using the same infrastructure (network, *load balancer*) using IaC (*eg*, Terraform) compared to manual methods in AWS, Azure, and GCP; (2) Analyze the configuration consistency (*immutability*) configuration drift achieved through IaC; and (3) Provide a frame work *best practice* decision-making guidance for organizations looking to optimize their multi-cloud operations through automation. Through these results, this research contributes to a deeper understanding of operational challenges and solutions in the computing era, cloud distributed, and heterogeneous.

Research Methods

This research adopts a quantitative-experimental approach with a comparative case study design to evaluate the effectiveness of IT infrastructure automation management using Infrastructure-as-Code (IaC) modelling, specifically the Terraform tool, in a multi-cloud environment spanning AWS, Azure, and GCP [10] [11]. The experiment will compare the IaC provisioning method with traditional/manual methods, focusing on measuring three critical dependent variables [12]: *Deployment* Time, *Configuration Drift* Rate (measured over 30 days), and *Code* Portability (*code* modification time between *clouds*), to measure the reduction in *operational overhead* and the improvement in configuration consistency (*immutability*) offered by IaC[13]&[14] The procedure will involve repeating the test five times for each method on functionally equivalent infrastructures, and the data will be analyzed using descriptive statistics and Analysis of Variance (ANOVA) to determine statistically significant differences in operational efficiency across heterogeneous multi-cloud environments [15].

Result And Discussion

Result

Background Analysis

Digital transformation has fundamentally changed the business operations paradigm, driving the urgent need for IT infrastructure that can adapt quickly to volatile market demands. Since the beginning of the millennium, *cloud computing has become the primary foundation for agility. This provides on-demand computing resources and operates on a pay-as-you-go model*. However, this rapid evolution is taking organisations beyond using a single provider. *Cloud single (such as AWS or Azure) towards an architecture*

known as a multi-cloud environment. A multi-cloud environment is defined as the use of cloud computing services distributed across two or more providers and public clouds [1]. This strategic decision was driven by several critical factors: risk mitigation, vendor lock-in, cost optimisation through price arbitrage across providers, fulfilment of requirements (residency data or strict geographical regulations), and the need for redundancy and disaster resilience (disaster recovery) [2].

Despite offering strategic advantages, multi-cloud architectures inherently create massive management complexity. The main challenge lies in API and *tooling disparities*. Each provider cloud has a unique application programming interface (API) framework, configuration language, and operational models. This forces operations teams to maintain deep, specialised expertise for each platform, significantly increasing operational overhead. Besides that, Manual provisioning of infrastructure in a multi-cloud environment is vulnerable to configuration drift, in which the configuration server, network, or storage deviates from its ideal state over time, creating security gaps and inconsistencies that are fatal to scalability and compliance [3]. These challenges often defeat the initial goal of adoption. *Cloud*, namely, increasing speed of *deployment* [4].

To address the challenges of multi-cloud complexity, the practice of IT Infrastructure Automation Management has grown rapidly. The essence of this automation is the adoption of Infrastructure-as-Code (IaC). IaC is the practice of managing and provisioning IT infrastructure through code, using machine-readable definition files rather than physical hardware configuration or interactive configuration tools. The basic concept of IaC revolves around the principles of Immutability (infrastructure is not changed; it is replaced with a new version) and Idempotence (operations that run the same code multiple times result in the same final infrastructure state) [5]. With IaC, *code for virtual machines, networks, and load balancers can be treated the same as code for an application: stored in a Git repository, tested, reviewed, and implemented through a CI/CD pipeline*. This transforms infrastructure management from an ad hoc, manual practice to a software engineering discipline, is predictable, and can be repeated (*reproducible*).Tools such as Terraform (declarative, platform-agnostic) and Ansible (imperative, focused on configuration management) are the backbone of modern IaC modelling [6].

The Research Gap. Despite the benefits of IaC in the single-cloud environment having been well documented, academic literature still has significant gaps in empirical and quantitative evaluation regarding the actual effectiveness of IaC tooling in addressing the unique complexities of a heterogeneous multi-cloud environment. Existing studies are often qualitative or focused on comparisons. *tool*in one *platform*(for example, CloudFormation vs. Terraform on AWS) [7].However, the crucial question is: How effective is it? *Tooling like Terraform, with abstraction-provider-agnostic, in reducing cognitive and technical overhead caused by API differences between AWS, Azure,* and GCP? Is the promise of portability really true, or is the configuration unique (such as network peering or security groups) still extensive enough to require modification, thereby reducing the claimed automation efficiency? This gap creates an urgent need for studies that rigorously measure operational performance metrics such as deployment time and configuration drift level across different multi-cloud environments [8].

Research purposes: Therefore, this research aims to evaluate experimentally and comparatively the effectiveness of IaC modelling in IT infrastructure automation management in multi-cloud environments. Specifically, this research will [9]:(1) Measure and compare deployment time using the same infrastructure (network, *load balancer*) using IaC (*eg*, Terraform) compared to manual methods in AWS, Azure, and GCP; (2) Analyze the configuration consistency (*immutability*)And *configuration drift*achieved through IaC; and (3) Provide a framework, best *practices* decision-making guidance for organizations looking to optimise their multi-cloud operations through automation. Through these results, this research contributes to a deeper understanding of operational challenges and solutions in the computing era, cloud, distributed, and heterogeneous.

Quantitative Experiment Results

Experiments compared three identical functional deployment scenarios (VM, Network, and Load Balancer) across AWS, Azure, and GCP, using IaC (Terraform) and Manual (Console/CLI Native) methods. Each test was repeated five times (n=5).

Deployment Time Comparison

Table 1 presents the average time required to reach *fully provisioned* (ready-to-operate) status.

Table 1. The Average Time Required To Reach Fully Provisioned

Platform	Manual Method (µsecond)	IaC (Terraform) Methods (µsecond)	Time Reduction Percentage
AWS	425	115	73.06%
Azure	510	140	72.55%
GCP	480	125	73.96%
Multi-Cloud Average	**471.67**	**126.67**	**73.15%**

Analysis of Variance (ANOVA): A one-way ANOVA was performed on *Deployment* Time between the IaC Method and the Manual Method. The results showed a highly significant difference: $F(1, 28) = 485.23$, $p < .001$. Interpretation: The data definitely shows that IaC (Terraform) significantly reduces *Deployment* Time across multi-cloud environments, with an average time reduction of over 73%. This time reduction is most pronounced in GCP, indicating that Terraform abstractions are effective in simplifying the provisioning process that may natively require many interactive steps in each *cloud's console*.

***Configuration Drift* Rate (Over 30 Days)**

Table 2 compares the average *configuration drift (percentage of assets deviating from the baseline*) after 30 days, simulated with deliberate manual configuration changes.

Table 2. Comparison Of Average Configuration Deviations (Percentage Of Assets Deviating From The Baseline) After 30 Days

Platform	Manual Method (Without Automatic Check)	IaC Method (With Terraform State File)	Drift Reduction Percentage
AWS	18%	3%	83.33%
Azure	22%	4%	81.82%
GCP	15%	2%	86.67%
Multi-Cloud Average	**18.33%**	**3.00%**	**83.67%**

Interpretation: These results demonstrate the vital role of IaC in achieving infrastructure consistency (immutability). By using Terraform state files as a single source of truth, teams can routinely run Terraform plans to detect and automatically correct configuration drift. Manual methods, which rely on documentation and engineers' memory, result in average drift above 18%, which directly increases security risks and non-compliance.

Code Portability Analysis (Cross-Cloud Modification Time)

Table 3 measures the average time required to modify Terraform code originally written for one cloud to make it usable for deployment on another cloud.

Table 3. Code Portability Analysis

Cloud Transition	Average Modification Time (µminute)
AWS→Azure	55
Azure → GCP	65
GCP → AWS	45
Average Portability	**55**

Interpretation: While terraform offers *cloud-agnostic* abstractions, *the code is not fully* portable without modification. The average time of 55 minutes required to modify *the code* indicates that engineers must make significant adjustments, particularly regarding network resource specifications (e.g., different security group naming or load-balancing schemes). This underscores the challenges *that cloud-specific APIs pose for agnostic IaC tools.*

Discussion

Provisioning Efficiency and Impact on the DevOps Cycle

The 73% reduction in deployment time is undeniable proof that IaC, specifically Terraform, is a critical *enabler* for DevOps in multi-cloud environments. This automation enables teams to implement the principles of speed and reliability at the heart of DevOps. When *deployments* are fast and *repeatable*, teams can iterate more and apply infrastructure updates securely, mitigating the *fear of deployment* common in complex, manual environments.

IaC as a *Governance* and *Compliance Mechanism*

The very low drift rate (average 3.00%) indicates that IaC's primary value in multi-cloud is as a *governance* mechanism. In heterogeneous environments, manually maintaining *compliance* (e.g., ensuring that all servers have *a security group* that blocks *port* X) is impossible. IaC enables *governance* through code reviews and CI/CD pipelines, forcing all *clouds* to adhere to the same standards. This is key to addressing data regulation challenges, which are often the primary reason for multi-cloud adoption.

Portability Limitations and *Provider Dependence*

While Terraform facilitates multi-cloud management, *Code* Portability analysis shows that its *cloud-agnostic* claims have practical limitations. The average modification time of 55 minutes indicates that IaC *code* remains tied to *cloud-specific* technical specifications. *Load balancers* in AWS, Azure, and GCP, while functionally similar, have different input structures and associated resources (such as *target groups* or *backend services*) in their API schemas. Therefore, actual multi-cloud *deployments require* modularly abstracted code (e.g., using Terraform modules) rather than simply copying and pasting code.

Conclusion

This experimental study has quantitatively demonstrated the effectiveness of Infrastructure-as-Code (IaC) modeling, specifically with Terraform, as a superior automation management mechanism in a multi-cloud environment (AWS, Azure, and GCP). The key findings confirm the following research hypotheses: 1). *Deployment* Time Efficiency: IaC methods significantly reduce average *Deployment Time by over* 73% compared to manual provisioning methods. This directly improves *agility* and time-to-market, which are key to the DevOps philosophy. 2). Infrastructure Consistency (Immutability): The use of IaC *state files* successfully reduced the average Configuration Drift rate to only 3.00% over 30 days, compared to 18.33% with manual methods. This proves that IaC is a key *enabler* for *governance*, compliance, and the reduction of security risks in heterogeneous IT environments. 3). Portability Limits: While IaC facilitates uniform multi-cloud management, *code* portability analysis shows that *the code* is not completely *cloud-agnostic*. An average modification time of 55 minutes per *cloud* transition is required, indicating that engineers must address the complexity and unique API specifications of each platform and adopt advanced *modular coding* practices.

References

[1] Gartner, “Hype Cycle for Security Operations,” gartner.com.
[2] J.Duarte, “IaC in Multi-Cloud Environments: A Comparative Study of Configuration Management Tools. , 10 (2), 150-168.,” *International Journal of Cloud Computing*, vol. 4, no. 1, p. 19, 2020.
[3] HashiCorp, “The State of Cloud Strategy Survey 2021: Multi-cloud is the new normal,” hashicorp.com.
[4] P.Mell, “The NIST definition of cloud computing,” *National Institute of Standards and Technology Special Publication*, vol. 17, no. 2, p. 145, 2011.
[5] P.Patil, “Implementing DevOps Practices for Multi-Cloud Deployment using Terraform and Ansible.2019.8972551,” *Proceedings of the 2019 International Conference on Communication and Networks*, vol. 7, no. 1, pp. 1–6, 2019.
[6] T.Documentation, “Terraform: Write, Plan, and Create Infrastructure as Code,” terraform.com.
[7] S.Vinoski, “Cloud Computing: The Software as a Service approach,” *IEEE Internet Computing*, vol. 12, no. 4, p. 85, 2008.
[8] S. J.Patil, “Can Emotional marketing change customer’s purch! se decision?,” *Journal of Advanced Research in Management and Social Sciences*, vol. 4, no. 4, pp. 130–141, 2015.
[9] T.Toglia, “The rise of multi-cloud architectures: Operational challenges and solutions through automation,” *Journal of Enterprise Information Management*, vol. 35, no. 5, p. 790, 2022.
[10] Hasan, *Pokok-pokok Materi Metodologi Penelitian dan Aplikasinya*. Jakarta: Ghalia Indonesia, 2002.
[11] Sugiyono, *Metode Penelitian Kuantitatif, Kualitatif, R&D*. 2019.

[12] Jonathan Sarwono, *Meode Penelitian Kualitatif dan Kuantitatif*. Bandung: Graha Ilmu, 2016.
[13] I.Ghozali, *Aplikasi analisis multivariete dengan program (IBM. SPSS)*. Diponergoro: Univrsitas Dipenogoro, 2016.
[14] Hair, *Multivariate Data analysis, Seventh Editions*. Prentice Hall: New Jersey, 2010.
[15] M.Sarstedt, C.M. Ringle, D.Smith, R.Reams, andJ. F.Hair Jr, "Partial least squares structural equation modeling (PLS-SEM): A useful tool for family business researchers," *Journal of Family Business Strategy*, vol. 5, no. 1, pp. 105–115, Mar.2014.