Automation of Secure and Compliant Infrastructure Orchestration Utilizing Terraform on AWS

Anusha Singamaneni, Ranjith Bhaskaran, Cristina Hava Muntean and Shaguna Gupta
School of Computing, National College of Ireland, Dublin, Ireland

Keywords: Infrastructure-as-Code, Terraform, Amazon Web Services, Cloud Security, Compliance-as-Code, Identity and Access Management, Virtual Private Cloud, Automation.

Abstract: Secure, compliant cloud provisioning is difficult with manual configuration, where misconfigurations, inconsistent security, and limited auditability often arise. Infrastructure-as-Code (IaC) solves this by defining infrastructure declaratively and enabling repeatable, version-controlled deployments. This paper presents an AWS-focused Terraform approach embedding security-by-design controls into automated provisioning, including least-privilege IAM, network segmentation with public and private VPC subnets, bastion-based administrative access, controlled outbound connectivity via a NAT gateway, and centralized logging and encryption baselines. The implementation is evaluated through a comparative study against manual provisioning using the AWS Management Console. Results show Terraform reduces provisioning time by over 75% across complex networking and access-control scenarios, while improving reliability by increasing success rates from 74% to 96%. Connectivity validation confirms that public resources route traffic through the Internet Gateway, private instances access outbound connectivity only via the NAT gateway, and administrative access to private resources is restricted to the bastion host. Security validation confirms consistent enforcement of baseline controls such as IAM least privilege, subnet isolation, restricted Secure Shell (SSH) ingress, centralized logging, and encryption at rest. These findings demonstrate that Terraform-based Infrastructure-as-Code can simultaneously improve operational efficiency and strengthen security and compliance consistency, offering a practical foundation for repeatable and audit-ready cloud infrastructure deployments, particularly for environments with limited operational overhead.

1 INTRODUCTION

Cloud infrastructure has become the dominant foundation for delivering modern digital services due to its scalability, elasticity, and cost efficiency. However, secure and consistent configuration of cloud environments remains a significant challenge, particularly for organizations with limited dedicated cloud security and operations expertise. Manual provisioning through cloud management consoles is inherently error-prone, difficult to audit, and hard to reproduce, often resulting in misconfigurations, security gaps, and configuration drift over time (Odukoya, 2024). Infrastructure-as-Code (IaC) addresses these challenges by representing infrastructure configurations as declarative, version-controlled code. This approach enables repeatable deployments, systematic change management, and improved collaboration across development and operations teams (Patni et al., 2020; Teppan et al., 2022; Singh et al., 2024; Bhaskaran et al., 2025). Terraform has emerged as a widely adopted IaC tool due to its modular design, provider abstraction, and state management capabilities, allowing infrastructure resources to be provisioned and updated in a predictable and auditable manner (Howard, 2022; Kavas, 2023).

Despite the operational advantages of IaC, automation alone is insufficient to guarantee secure cloud deployments. Security controls, access governance, and compliance mechanisms must be embedded directly into provisioning workflows to ensure that infrastructure is secure by default and remains compliant throughout its lifecycle. This need has been emphasized in DevSecOps research, which advocates integrating security validation and policy enforcement early in the infrastructure deployment pipeline rather than relying on post-deployment hardening (Ibrahim et al., 2022; Bajpai and Lewis, 2022). This challenge is particularly relevant for small and medium enterprises (SMEs), where limited security

Singamaneni, A., Bhaskaran, R., Muntean, C. H. and Gupta, S.
Automation of Secure and Compliant Infrastructure Orchestration Utilizing Terraform on AWS.
DOI: 10.5220/0014873000004039

In *Proceedings of the 16th International Conference on Cloud Computing and Services Science (CLOSER 2026)*, pages 269-276
ISBN: 978-989-758-829-7; ISSN: 2184-5042

expertise and operational resources make manual enforcement of cloud best practices impractical (Alkawsi et al., 2015). In response to these challenges, this paper proposes an AWS-focused Terraform-based approach that codifies security-by-design principles into reusable infrastructure definitions. The proposed solution integrates least-privilege Identity and Access Management (IAM), network segmentation, controlled administrative access via bastion hosts, and audit logging into a standardized and reproducible infrastructure baseline. The approach is evaluated through a quantitative comparison between manual console-based provisioning and Terraform-based automation, focusing on provisioning time, accuracy, and enforcement of security controls.

The paper is structured as follows. Section II reviews related work in Infrastructure-as-Code, DevSecOps, and secure cloud automation. Section III presents the Terraform-based architecture and design objectives. Section IV describes implementation details and modular infrastructure components. Section V evaluates the approach through experimental results and discussion. Section VI concludes and outlines directions for future work.

2 RELATED WORK

Infrastructure-as-Code has become a central practice in modern cloud operations, enabling consistent provisioning and reducing operational overhead. Prior studies highlight the advantages of declarative infrastructure definitions, while identifying common pitfalls such as configuration complexity, insufficient validation, and security misconfigurations. Kumara et al. (Kumara et al., 2021) provide a comprehensive review of IaC practices and emphasize the importance of structured design, modularization, and code quality to prevent infrastructure defects. The integration of security into DevOps workflows has led to the emergence of DevSecOps, which promotes embedding security controls early in the development and deployment lifecycle. Ibrahim et al. (Ibrahim et al., 2022) propose security models that incorporate automated validation and policy enforcement for infrastructure code, demonstrating the need for security-aware provisioning pipelines rather than post-deployment hardening. Terraform has been widely discussed as a practical automation tool for Infrastructure-as-a-Service environments. Howard (Howard, 2022) outlines Terraform's core concepts, including resource graphs, state management, and modular configurations, which support scalable and repeatable infrastructure provisioning. Surveys by Teppan et al. (Teppan et al., 2022) further examine IaC solutions in cloud development, identifying challenges related to governance, maintainability, and compliance across complex cloud environments. While existing work establishes the theoretical foundations of IaC and secure cloud automation, there is a limited body of implementation-oriented research that quantitatively evaluates security-focused Terraform deployments in real-world cloud settings. In particular, few studies provide empirical comparisons between manual cloud provisioning and security-by-design IaC approaches.

Current state-of-the-art research primarily focuses on conceptual models, best practices, and high-level frameworks for Infrastructure-as-Code and DevSecOps adoption, with limited emphasis on end-to-end, security-centric implementations validated through quantitative experimentation. Existing studies often evaluate IaC tools in isolation or discuss security controls at an architectural level, without measuring their operational impact relative to traditional manual provisioning workflows. Moreover, empirical evaluations frequently overlook practical security mechanisms such as bastion-mediated administrative access, enforced network segmentation, controlled outbound connectivity, and centralized audit logging as integrated components of an automated provisioning baseline. As a result, there remains a gap between theoretical security-by-design principles and their measurable effectiveness in real-world cloud deployments. This work advances the state of the art by presenting a reproducible, Terraform-based AWS infrastructure that embeds security and compliance controls into the provisioning process and by quantitatively evaluating its performance against manual console-based deployments in terms of efficiency, accuracy, and enforcement of security controls.

3 PROPOSED ARCHITECTURE

3.1 Design Objectives and Threat Profile

The proposed approach establishes a secure, compliant, and reproducible Amazon Web Services (AWS) infrastructure using Terraform as the primary Infrastructure-as-Code (IaC) tool. The design objectives focus on translating established cloud security best practices into codified and reusable infrastructure definitions suitable for governance-driven cloud adoption (Patni et al., 2020). To satisfy a "security-by-design" methodology, this architecture assumes an **external adversary** with network-probing capabilities attempting unauthorized access, data exfiltration,

or resource compromise. It also considers **insider threats** (unintended misconfigurations by developers or privilege creep). Thus, the design objectives are:

- **Repeatability:** Consistent, deterministic infrastructure across environments using declarative definitions.
- **Security-by-Design:** Default least-privilege access, network isolation, and encryption to reduce attack surface.
- **Auditability:** Centralized logging and configuration history for full traceability and non-repudiation.
- **Operational Simplicity:** Modular Terraform and automation to minimize effort and human error.

These objectives are particularly relevant for small and medium-sized enterprises (SMEs), which often require secure cloud baselines while operating under limited operational and administrative overhead (Alkawsi et al., 2015).

3.2 AWS Infrastructure Baseline

To avoid redundant security features and ensure the architecture is both **necessary and sufficient**, the components are layered following the Defense-in-Depth (DiD) strategy. Each component is selected to close a specific vulnerability without overlapping uncontrollably (Kovacevic and Dicola, 2023; Subhan et al., 2025).

- **Virtual Private Cloud (VPC):** Acts as the primary network isolation boundary for all deployed resources, defining the trust perimeter (Kavas, 2023).
- **Public Subnets:** Host externally accessible components, including bastion hosts. Direct ingress to internal app tiers is strictly prohibited (Kavas, 2023).
- **Private Subnets:** Isolate internal resources such as application servers and databases from direct internet access, neutralizing external scanning (Kavas, 2023).
- **Route Tables:** Regulate traffic flow within the VPC and control internet connectivity paths (Kavas, 2023).
- **Identity and Access Management (IAM):** Fine-grained, role-based access policies (Admin, DevOps, Developer, ReadOnly) enforcing the principle of least privilege, preventing privilege escalation (Ibrahim et al., 2022).
- **Bastion Host:** Provides a single, hardened administrative access point to private subnet resources, eliminating exposed SSH ports on internal databases (Kavas, 2023).
- **Network Address Translation (NAT) Gateway:** Enables controlled outbound internet access for private subnet resources without exposing inbound connectivity, preventing direct external targeting of backend nodes (Kavas, 2023).
- **Security Groups and Network ACLs:** Deployed in tandem to provide stateful (instance-level) and stateless (subnet-level) traffic filtering. This dual-layer prevents internal pivot attacks if one instance is breached (Oulaaffart et al., 2021).
- **VPC Flow Logs:** Capture network traffic metadata for auditing, troubleshooting, and security monitoring (Giamattei et al., 2024).
- **TLS Termination:** Ensures encrypted communication between clients and hosted services, neutralizing Man-in-the-Middle (MitM) attacks (Vakhula et al., 2023).

3.3 Threat Model Mapping and Sufficiency

To evaluate whether this architecture is sufficient to close standard cloud attack vectors, Table 1 maps the components of the proposed baseline to specific threats and adversary capabilities.

Table 1: Threat Model and Component Mapping.

Threat Scenario	Mitigation Mechanism
Brute-force SSH attack	Public Bastion + Private Subnets + SG Rules
Data exfiltration via internet	NAT Gateway egress routing rules only
Lateral movement / Pivot	Stateful Security Groups (Micro-segmentation)
Privilege Escalation	Principle of Least Privilege (IAM custom roles)
Man-in-the-Middle (MitM)	Enforced TLS Termination at the edge
Unchecked backdoor changes	Terraform State files + VPC Flow Logging

3.4 Architecture Overview

Figure 1 presents the Terraform-driven provisioning workflow and the resulting AWS infrastructure layout. The architecture emphasizes network segmentation, controlled administrative access, and integrated monitoring, while Terraform ensures consistent and repeatable deployment through its dependency-aware execution model.

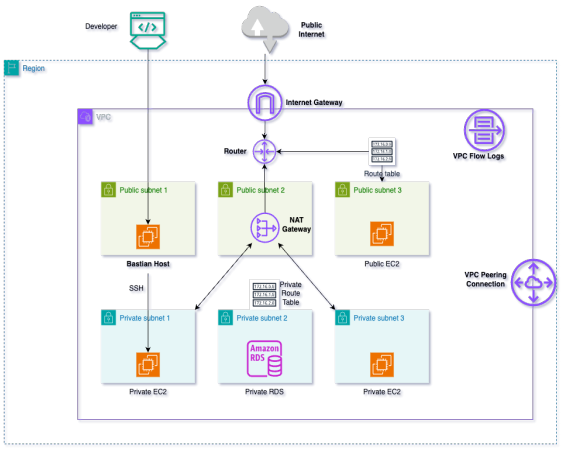

Figure 1: Terraform-driven secure AWS infrastructure architecture.

3.5 Terraform Resource Dependency Model

Terraform manages infrastructure provisioning through a Directed Acyclic Graph (DAG), which automatically determines resource dependencies based on declared configuration relationships. This execution model ensures that infrastructure components are created, modified, and destroyed in the correct sequence.

3.5.1 Resource Provisioning Workflow

The infrastructure provisioning process follows a structured sequence:

- **Resource Definition:** Infrastructure components are declared using HashiCorp Configuration Language (HCL).
- **Dependency Resolution:** Terraform identifies implicit and explicit dependencies among resources.
- **Execution Planning:** A detailed execution plan is generated, outlining all proposed infrastructure changes.
- **Provisioning:** Resources are created according to the DAG, ensuring dependency correctness.
- **State Management:** Terraform maintains a state file to track deployed resources and support updates and drift detection.

3.5.2 Dependency Enforcement Examples

The DAG-based execution model ensures that infrastructure components are created in the correct order. For example, IAM roles and policies are provisioned before being associated with users or services, subnets and route tables are created only after the VPC is successfully established, and security groups must exist before they can be attached to compute resources.

3.6 Modularity and Reusability

The architecture adopts a modular Terraform design in which core components such as networking, IAM, and security controls are encapsulated into reusable modules. This modular approach enables consistent enforcement of security and compliance policies across deployments, simplifies scaling, and minimizes configuration drift. Combined with Terraform's DAG-based execution model, the architecture provides a secure, reproducible, and extensible foundation for automated cloud infrastructure provisioning.

4 IMPLEMENTATION

The proposed system is implemented using Terraform-based Infrastructure-as-Code (IaC) to provision a secure and compliant cloud baseline on Amazon Web Services (AWS). The implementation follows the architectural workflow illustrated in Figure 2, where user Terraform configurations are translated into modular infrastructure components, validated against security and compliance requirements, and deployed as reproducible cloud resources.

4.1 Implementation Workflow and Experimental Setup

The implementation follows a structured IaC lifecycle designed to ensure security-by-design before any resources are instantiated in the AWS cloud. As shown in Figure 2, the workflow is categorized into four distinct phases:

1. **Configuration and Modularization:** Infrastructure intent is expressed through HashiCorp Configuration Language (HCL). Resources are modularized to separate networking, IAM, and compute logic, ensuring security baselines (e.g., encryption-at-rest) are inherited across modules.
2. **Initialization and Dependency Mapping:** Terraform initializes the backend and providers, constructing a Directed Acyclic Graph (DAG) to resolve resource dependencies (e.g., ensuring a VPC exists before a Subnet is provisioned).
3. **Execution Planning (Validation):** The `terraform plan` phase acts as a pre-deployment

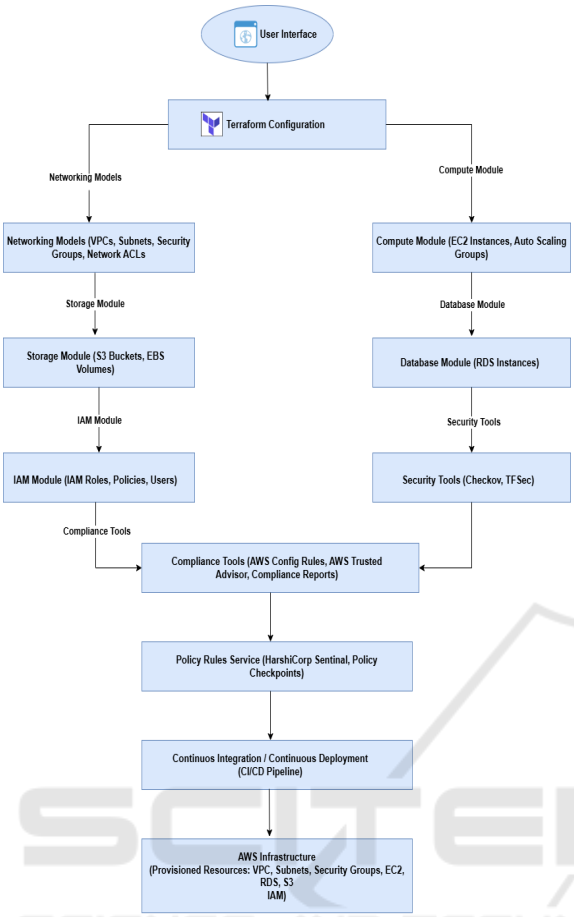

Figure 2: Architectural flow of the proposed Terraform-based infrastructure provisioning.

audit. This allows for the detection of misconfigurations such as overly permissive Security Group rules prior to actual resource creation.

4. **Automated Provisioning:** The `terraform apply` phase executes the API calls to AWS. This process was integrated with a local version-controlled repository to simulate a CI/CD pipeline environment, ensuring that every change is tracked via the Terraform State file.

The experimental execution was conducted on a workstation with the AWS CLI and Terraform v1.5+ installed, targeting the `us-east-1` region. To ensure a fair comparison with manual provisioning, the ”Terraform time” recorded in Section 5 includes the time required for HCL authoring, debugging, and execution. By utilizing this automated workflow, the system minimizes manual intervention, reduces configuration drift, and enables repeatable deployments across multiple environments.

4.2 Terraform Modular Infrastructure Design

To improve maintainability and reusability, the infrastructure is decomposed into logical Terraform modules and configuration files, each responsible for a specific layer of the cloud stack.

- **Networking Layer:** Provisions the Virtual Private Cloud (VPC), public and private subnets, route tables, and the Internet Gateway. Public subnets host externally reachable components, while private subnets isolate internal workloads and backend services.
- **Access Control Layer:** Identity and Access Management (IAM) groups, roles, and policies are defined to represent organizational roles such as Administrator, DevOps, Developer, and Read-Only access. Custom policies enforce least-privilege permissions and simplify audit verification.
- **Secure Administrative Access:** A hardened bastion host is deployed within a public subnet to serve as the single administrative entry point. Security group rules restrict Secure Shell (SSH) access to approved network ranges, ensuring controlled access to private resources.
- **Private Egress Control:** A Network Address Translation (NAT) gateway enables outbound internet connectivity for private subnet resources without exposing them to inbound traffic. Routing rules ensure that all private egress traffic flows exclusively through this component.
- **Logging and Auditing:** VPC Flow Logs are enabled and delivered to encrypted object storage. Supporting IAM roles and policies allow secure log ingestion while preserving the integrity and auditability of network activity.
- **Network Security Enforcement:** Multiple security groups regulate communication between infrastructure tiers, including public-facing services, private application layers, and backend databases. Ingress and egress rules are narrowly scoped to required ports and trusted sources.

4.3 Configuration Management and Parameterization

Infrastructure definitions are authored using HashiCorp Configuration Language (HCL) and parameterized through input variables to support deployment across environments such as development and production. Local values are used to centralize tagging conventions, environment identifiers, and

reusable configuration attributes, enabling consistent governance and cost attribution. Sensitive credentials and access keys are managed programmatically, and outputs are explicitly marked to prevent accidental disclosure during plan and apply operations.

4.4 Terraform Execution and State Handling

Terraform provisions resources using a dependency-resolved Directed Acyclic Graph (DAG) execution model. Prior to deployment, an execution plan is generated to present all proposed infrastructure changes for review. Terraform state files maintain a record of deployed resources, enabling incremental updates, drift detection, and controlled reconciliation toward the declared configuration. This state-driven approach ensures long-term consistency between declared infrastructure and deployed environments.

4.5 Operational Outcome

The deployed infrastructure reflects the architectural flow shown in Figure 2, delivering a secure, segmented, and auditable cloud environment. By embedding access control, network isolation, logging, and encryption directly into the provisioning workflow, the implementation minimizes manual configuration effort and supports repeatable, compliance-ready cloud deployments.

5 EVALUATION AND RESULTS

This section presents quantitative results comparing manual Amazon Web Services (AWS) console provisioning with Terraform-based Infrastructure-as-Code automation. The evaluation considers provisioning time for networking and IAM/security resources, provisioning accuracy, and validation of connectivity and security controls in the deployed infrastructure.

5.1 Experiment 1: Provisioning Time for Networking Resources

Table 2 summarizes the time (measured in minutes) required to provision networking infrastructure across increasingly complex deployment scenarios. In this study, **provisioning time** is defined as the total duration encompassing the authoring of configuration files (or manual console navigation), the execution of the deployment (‘terraform apply‘), and the verification of resource availability. Terraform consistently reduces provisioning time across all configurations, achieving reductions of approximately 78–83%. As shown in Figure 3, the improvement becomes more pronounced as infrastructure complexity increases, indicating better scalability of declarative provisioning compared to manual configuration.

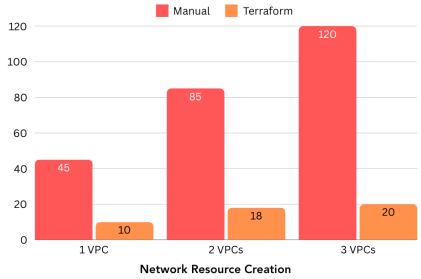

Figure 3: Networking provisioning time comparison between manual configuration and Terraform automation.

Table 2: Provisioning time comparison for networking infrastructure.

Configuration	Manual (min)	Terraform (min)	Reduction (%)
1 VPC (3 subnets)	45	10	77.78
1 VPC + Bastion and NAT	80	17	78.75
2 VPCs (6 Subnets)	85	18	78.82
2 VPCs + Bastion and NAT	120	23	80.83
3 VPCs (9 Subnets)	120	20	83.33
3 VPCs + Bastion and NAT	140	25	82.14

5.2 Experiment 2: Provisioning Time for IAM and Security Resources

Table 3 reports the time required to provision IAM groups, policies, and security-related roles. Terraform significantly reduces repetitive console-based operations such as policy creation and attachment. The observed time includes the logic-design phase where IAM JSON policies are defined within Terraform HCL. Despite the complexity of writing secure policies, automation resulted in time reductions of approximately 66–75%. Figure 4 provides a visual comparison of these durations in minutes.

5.3 Experiment 3: Provisioning Accuracy and Failure Analysis

Provisioning accuracy was evaluated by counting misconfigurations (e.g., incorrect CIDR blocks or missing tags) and failed resource creations observed during 50 resource deployments. As shown in Ta-

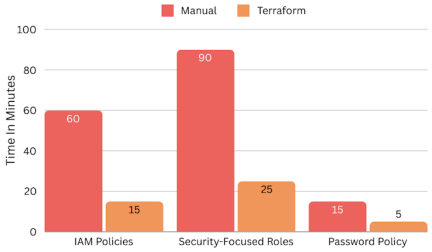

Figure 4: IAM and security resource provisioning time comparison.

Table 3: Provisioning time comparison for IAM and security resources.

Resource Type	Manual (min)	Terraform (min)	Saved (min)
IAM groups and policies	60	15	45
Security-focused roles	90	25	65
Password policy	15	5	10

ble 4, Terraform improves the success rate from 74% to 96%. While Terraform significantly minimizes human error, a 4% failure rate was still observed. Analysis revealed that these failures were not due to syntax errors, but rather **provider-side constraints**, specifically AWS API rate-limiting (throttling) during concurrent resource creation and occasional eventual-consistency delays in IAM role propagation. These findings highlight that while IaC provides superior consistency, infrastructure architects must still account for cloud provider-specific execution limits.

Table 4: Accuracy comparison (50 resource sample).

Metric	Manual	Terraform
Total resources attempted	50	50
Misconfigurations	8	1
Failed resource creations	5	1
Success rate (%)	74%	96%

5.4 Experiment 4: Connectivity Validation

Connectivity validation was performed to ensure that the deployed infrastructure behaves as intended. The results in Table 5 confirm that public instances access the internet via the Internet Gateway, private instances use the NAT gateway for outbound connectivity, and administrative access to private resources is restricted to the bastion host.

Table 5: Network connectivity validation results.

Component	Expected Behaviour	Result
Private instance	Internet via NAT only	Success
Public instance	Direct internet access	Success
NAT gateway	Correct outbound routing	Success
Bastion host	SSH access to private instances	Success

5.5 Experiment 5: Security Feature Validation

Security controls were validated to confirm enforcement of security-by-design defaults. Table 6 summarizes the controls, including least-privilege IAM, network isolation, bastion-mediated access, centralized logging, and encryption. These results demonstrate that Terraform-based automation supports consistent and audit-ready security baseline enforcement.

Table 6: Security feature validation summary.

Security Feature	Status	Remarks
IAM least privilege	Implemented	Group-specific custom policies verified
Network access control	Implemented	Security group and NACL rules enforced
Subnet isolation	Implemented	Private resources accessed via NAT and bastion only
Bastion host access	Implemented	Restricted SSH ingress verified
Monitoring and logging	Implemented	Network activity and change logs validated
Encryption at rest	Implemented	Encryption applied to storage services where applicable
Password policy	Implemented	Strong password requirements enforced

5.6 Limitations and Scalability Analysis

While the proposed Terraform-based framework significantly improves repeatability, certain architectural limitations must be acknowledged for enterprise-scale adoption. First, the current study focuses on a single-region deployment; however, expanding to **multi-region infrastructure** would require advanced Terraform features such as *aliased providers* and global variable mapping to maintain consistency across geographic boundaries. Second, as noted during the 4% failure rate analysis in Experiment 3, **Terraform State management** becomes a critical pivot point in collaborative environments. Without a centralized, locked backend (such as Amazon S3 with DynamoDB locking), concurrent executions can lead to state corruption. Furthermore, while the current ar-

chitecture effectively eliminates common vulnerabilities through layered defense, the ”security-as-code” transition requires ongoing maintenance to ensure that as AWS introduces new services, the Terraform modules are updated to prevent ”shadow IT” or unmanaged resource sprawl.

6 CONCLUSION AND FUTURE WORK

This paper presents a rigorous evaluation of automating secure and compliant AWS infrastructure provisioning using Terraform. By shifting from manual console-based configuration to an Infrastructure-as-Code (IaC) paradigm, security-by-design is embedded directly into the infrastructure's DNA. Our experimental results provide evidence of the benefits: a 75% reduction in provisioning time and an increase in deployment success rates from 74% to 96%. More importantly, the use of a Directed Acyclic Graph (DAG) for resource ordering ensures that security dependencies such as IAM policies and VPC flow logs are active before compute resources are accessible, closing critical windows of vulnerability.

Future work will expand this framework in three main areas. First, it will enable continuous compliance by integrating Terraform into CI/CD pipelines (e.g., GitHub Actions or GitLab CI) with automated security scanning tools like Checkov or tfsec for real-time policy enforcement. Second, it will add multi-region resilience with automated disaster recovery and high-availability setups across AWS regions. Finally, a formal risk analysis will be conducted to measure improvements, particularly in reducing Mean Time to Detect (MTTD) through automated state-consistency checks.

REFERENCES

Alkawsi, G. A., Mahmood, A. K., and Baashar, Y. M. (2015). Factors influencing the adoption of cloud computing in sme: A systematic review. In *International Symposium on Mathematical Sciences and Computing Research*, pages 220–225. IEEE.

Bajpai, P. and Lewis, A. (2022). Secure development workflows in ci/cd pipelines. In *IEEE Secure Development Conference*, pages 65–66. IEEE.

Bhaskaran, R., Muntean, C. H., and Gupta, S. (2025). Ai-driven cloud optimization: Enhancing cost prediction, resource scheduling and fault resilience in cloud environments. In *IEEE Int. Conference on Cloud Computing Technology and Science*, pages 1–8.

Giamattei, L., Guerriero, A., Pietrantuono, R., Russo, S., Malavolta, I., Islam, T., Dînga, M., Koziolek, A., Singh, S., Armbruster, M., Gutierrez-Martinez, J., Caro-Alvaro, S., Rodriguez, D., Weber, S., Henss, J., Vogelin, E. F., and Panojo, F. S. (2024). Monitoring tools for devops and microservices: A systematic grey literature review. *Journal of Systems and Software*, 208:111906.

Howard, M. (2022). Terraform – automating infrastructure as a service. arXiv.

Ibrahim, A., Yousef, A. H., and Medhat, W. (2022). Devsecops: A security model for infrastructure as code over the cloud. In *2nd Int. Mobile, Intelligent, and Ubiquitous Computing Conference*, pages 284–288. IEEE.

Kavas, E. (2023). *Architecting AWS with Terraform*. Packt Publishing.

Kovacevic, B. and Dicola, N. (2023). *Security Orchestration, Automation, and Response for Security Analysts*. Packt Publishing.

Kumara, I., Garriga, M., Romeu, A. U., Nucci, D. D., Palomba, F., Tamburri, D. A., and van den Heuvel, W.-J. (2021). The do's and don'ts of infrastructure code: A systematic gray literature review. *Information and Software Technology*, 137:106593.

Odukoya, O. (2024). The transformative impact of cloud computing on small and medium-sized enterprises (smes). In *International Conference on Smart Applications, Communications and Networking*, pages 1–5.

Oulaaffart, M., Badonnel, R., and Festor, O. (2021). Towards automating security enhancement for cloud services. In *IEEE International Symposium on Integrated Network Management*, pages 692–696. IEEE.

Patni, J. C., Banerjee, S., and Tiwari, D. (2020). Infrastructure as a code (iac) to software defined infrastructure using azure resource manager (arm). In *International Conference on Computational Performance Evaluation*, pages 575–578.

Singh, S., Muntean, C. H., and Gupta, S. (2024). Boosting microservice resilience: An evaluation of istio's impact on kubernetes clusters under chaos. In *Int. Conference on Fog and Mobile Edge Computing*, pages 245–252.

Subhan, F. E., Yaqoob, A., Muntean, C. H., and Muntean, G.-M. (2025). A survey on artificial intelligence techniques for improved rich media content delivery in a 5g and beyond network slicing context. *IEEE Communications Surveys & Tutorials*, 27(2):1427–1487.

Teppan, H., Flå, L. H., and Jaatun, M. G. (2022). A survey on infrastructure-as-code solutions for cloud development. In *IEEE International Conference on Cloud Computing Technology and Science*, pages 60–65.

Vakhula, O., Opirskyy, I., and Mykhaylova, O. (2023). Research on security challenges in cloud environments and solutions based on the security-as-code approach. In *CPITS II*. SCITEPRESS.