Mutation Operators for Terraform

Isadora Pacheco Ribeiro
ribeiroisadora@id.uff.br
Universidade Federal Fluminense
Niterói, RJ, Brazil

Abstract

Mutation testing is a powerful technique used to evaluate the quality of test suites by systematically introducing faults into code and verifying their detection. While extensively applied in traditional software development, its application in Infrastructure as Code (IaC) remains underexplored. This paper presents a novel approach to mutation testing in IaC, focusing on Terraform, a widely used tool for managing infrastructure configurations. We propose a set of mutation operators tailored to Terraform, derived from developer feedback and analysis of common IaC defects. These operators simulate realistic errors to assess the adequacy of test suites in detecting critical configuration flaws. Our experimental results indicate that the proposed operators effectively uncover gaps in existing testing strategies, revealing their inability to address various injected mutants. This study highlights the potential of mutation testing to improve IaC test suite robustness, paving the way for enhanced reliability and security in infrastructure management. Future work will explore automating defect identification through repository mining to refine mutation operator design.

CCS Concepts

• **Software and its engineering → Software testing and debugging**; *Software reliability*; *Software configuration management and version control systems*; • **Social and professional topics →** *Infrastructure.*

Keywords

Mutation Testing, Infrastructure as Code (IaC), Terraform Configurations, Mutation Operators, Test Suite Adequacy,

ACM Reference Format:
Isadora Pacheco Ribeiro. 2024. Mutation Operators for Terraform. In . ACM, New York, NY, USA, 6 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Mutation testing evaluates the quality of test suites by introducing systematic faults (mutations) into code and verifying their detection [2]. Central to this technique are mutation operators, which simulate realistic errors that may occur during development. While extensively studied in traditional software testing, their application to Infrastructure as Code (IaC)—where infrastructure is managed using code-like configurations [5]—remains underexplored.

Terraform[1], a widely used IaC tool, employs declarative configuration files to manage infrastructure resources. Ensuring the correctness of these configurations is critical, as errors can propagate to production systems and cause failures. Current approaches, such as static analysis and integration testing, primarily detect syntax errors or validate outcomes but lack methods to assess test suite robustness, creating a need for techniques like mutation testing.

Research highlights common IaC defects, including misconfigurations, logical flaws, and resource inconsistencies, which can lead to severe consequences [5, 7]. However, existing studies focus on detecting defects rather than evaluating the effectiveness of test suites in addressing such errors, leaving opportunities for mutation testing to improve IaC test adequacy, particularly for Terraform.

This paper introduces a novel set of mutation operators tailored for Terraform, derived from developer surveys and analyses of Terraform's syntactic and semantic features. These operators simulate common defects to evaluate test suite robustness. Controlled testing revealed that none of the unit tests passed when subjected to these mutations, exposing critical gaps in current testing strategies and emphasizing the need for improved methods.

The remainder of this paper is organized as follows: Section 2 outlines the objectives and research questions. Section 3 reviews related work. Section 4 explains the methodology. Section 5 presents the results and their implications. Section 6 discusses threats to validity, and Section 7 concludes with contributions, limitations, and suggestions for future work.

2 Objectives

This work seeks to address the following research questions:

- **RQ1:** What set of mutation operators can be designed for a terraform code?
- **RQ2:** Are these mutation operators effective in a Terraform project?

3 Related Work

The following review synthesizes research on Infrastructure as Code (IaC), mutation testing, and defect analysis. It highlights advancements in testing methodologies, security practices, and automation for IaC, with a specific focus on Terraform and mutation testing.

3.1 Infrastructure as Code and Testing Challenges

Rahman et al. (2019) conducted a systematic mapping of IaC research, highlighting the prevalence of misconfigurations and logical

[1]https://www.terraform.io/

errors in IaC scripts and emphasizing the need for robust testing mechanisms [7]. Hasan et al. (2020) reinforced this by advocating behavior-focused testing practices and modular design to reduce defects and risks, stressing systematic approaches for infrastructure reliability [3].

Teppan et al. (2022) surveyed IaC tools and their integration with CI/CD workflows, emphasizing the need for accessible solutions in resource-constrained environments and highlighting the relevance of on-premise IaC for scalability challenges [13]. Shimizu et al. (2024) proposed a test-suite-guided approach for optimizing least privilege configurations in cloud IaC, reducing operational costs and configuration iterations through iterative permission adjustments [10].

3.2 Mutation Testing in IaC

Mutation testing, while widely used in traditional software engineering, is less explored in IaC. Sokolowski et al. (2023) proposed Automated Configuration Testing (ACT), a framework that uses plugins for test generation and oracles, reducing testing overhead while ensuring accuracy [11]. Rahman et al. (2019) identified syntax-related errors as dominant IaC defects through orthogonal defect classification, recommending static analysis and code inspections for improved detection [8]. Sokolowski et al. (2024) expanded IaC testing with ProTI, integrating fuzzing and automated verification for reliable cloud deployments [12].

3.3 Mutation Testing for Test Suite Evaluation

Mutation testing has proven effective for evaluating test suite adequacy across multiple domains. Jia and Harman (2011) conducted a comprehensive survey, highlighting its role in fault detection within software test suites [4]. Petrovic et al. (2021) extended its scalability to large industrial applications, introducing incremental mutation analysis to enhance computational efficiency during code reviews [6].

Ramler et al. (2017) demonstrated its applicability to safety-critical systems, identifying gaps in unit test suites [9]. Wang et al. (2025) advanced this work by leveraging large language models (LLMs) to create realistic mutants for IaC, outperforming traditional methods in fault detection [14].

3.4 Contribution of This Work

This study addresses the deficit in Infrastructure-as-Code mutation testing by introducing a specialized framework for Terraform. The proposed mutation operators simulate realistic defects, including dependency and configuration errors. Evaluating test suite robustness advances understanding and deployment of mutation testing in IaC.

4 Methodology

This section details the methodological approach adopted in this research, including the design, construction, and analysis of the survey used to achieve the research objectives. The chosen methods aimed to comprehensively explore developer experiences and challenges with Infrastructure as Code (IaC) in Terraform and to inform the design of mutation operators for IaC testing.

4.1 Research Design

The methodology for this study was grounded in a *mixed-methods approach*, integrating both quantitative and qualitative analyses. This approach was chosen to address the research question: *"What set of mutation operators can be designed for Terraform code?"* By combining quantitative data and qualitative insights, the study sought to systematically identify common defects in Terraform configurations and gather developer perspectives to guide the creation of mutation operators.

The framework for this methodology was inspired by established principles for mixed-methods research [1].

4.2 Survey Development and Questions

A survey instrument was developed to collect data directly from Terraform developers, targeting their experiences, challenges, and perceptions of IaC practices. The survey questions were designed to balance technical depth and accessibility, Table 1 outlines the questions, their formats, and response types.

The survey questions adhered to the following design principles:

- **Content Validity:** Each question was aligned with the research objectives, ensuring coverage of both technical and qualitative dimensions of Terraform usage.
- **Clarity and Conciseness:** Questions were written in accessible language tailored to Terraform developers to ensure consistent interpretation.
- **Thematic Organization:** Questions were grouped into logical sections, facilitating smooth navigation through the survey and maintaining participant engagement.

4.3 Data Analysis

The data collected through the survey were analyzed using a systematic approach, integrating quantitative metrics and qualitative thematic coding.

Quantitative Analysis: Likert scale responses and objective questions were analyzed using descriptive statistics to identify trends, such as the frequency of common defects and the relative importance of IaC to operational success.

Qualitative Analysis: Open-ended responses were subjected to thematic analysis, which involved coding the data into recurring themes. Key thematic areas identified included:

- **Dependency Management:** Issues related to the definition and handling of resource dependencies.
- **Authentication Errors:** Challenges arising from misconfigured credentials or access controls.
- **Network Configuration:** Problems involving insecure or incorrect network settings.
- **Resource State Inconsistencies:** Cases where the configuration state diverged from the actual state of resources.

These thematic areas directly informed the design of mutation operators, as detailed in subsequent sections. For example, operators such as `open_sensitive_ports`, addressing insecure network configurations, were categorized under *Network Operator Replacement* (NOR).

4.4 Validation of Mutation Operators

The mutation operators derived from the survey analysis were validated using a custom mutation testing framework developed as part of this research. This framework allowed for the systematic application of mutation operators to Terraform configurations, simulating real-world scenarios to assess the effectiveness of the operators in identifying potential issues. The framework also provided a foundation for evaluating the reliability and security impacts of the identified defects.

4.4.1 *Terramutate: Framework for Mutation Testing.* [2]

The *Terramutate Framework* provides a structured approach to mutation testing specifically tailored for Terraform configurations. By systematically introducing controlled mutations into IaC resources, the framework evaluates whether existing test suites can detect these intentional changes. Through this process, it assesses the robustness and coverage of infrastructure test cases, ultimately informing improvements in test quality.

Framework Structure. The Terramutate Framework is organized into four primary modules, each playing a distinct role in the mutation testing lifecycle:

- **Configuration Module:** Ingests JSON/YAML configuration files to define infrastructure paths and specify mutation operators.
- **Mutation Module:** Applies mutation operators, classified based on their impact—such as altering dependencies, changing authentication settings, or modifying resource properties.
- **Execution Module:** Applies the defined mutations to cloned infrastructure configurations, executes tests via Terratest, and collects results.
- **Reporting Module:** Aggregates test outcomes associated with each mutation and generates a comprehensive report of test performance.

5 Results

Based on the analysis detailed in the *Data Analysis* subsection of the methodology, the following categories of mutation operators for Terraform were identified:

1. **DOR** – Dependency Operator Replacement
2. **AOR** – Authentication Operator Replacement
3. **ROR** – Resource Operator Replacement
4. **NOR** – Network Operator Replacement
5. **SOR** – State Operator Replacement
6. **POR** – Provider Operator Replacement
7. **VOR** – Variable Operator Replacement

These categories were structured similarly to mutation operators used in traditional mutation testing, such as **AOR (Arithmetic Operator Replacement)** and **ROR (Relational Operator Replacement)**, grouping the mutants based on their primary behaviors or effects.

[2] https://github.com/Asunnya/Terramutate

5.1 Categories of Mutation Operators

5.1.1 *1. Dependency Operator Replacement (DOR).* This category includes mutants that modify, add, or remove dependencies between resources, simulating synchronization issues in interdependent resource execution.

DOR Mutants:

- *circular_dependency*: Introduces a circular dependency between resources.
- *duplicate_dependency*: Adds a duplicate dependency.
- *remove_critical_dependency*: Removes a critical dependency.
- *cross_provider_dependency*: Adds a dependency across different providers.
- *non_existent_dependency*: Introduces a dependency on a non-existent resource.
- *remove_depends_on*: Removes explicit dependencies.
- *remove_implicit_dependency*: Removes implicit dependencies.

5.1.2 *2. Authentication Operator Replacement (AOR).* This category focuses on mutants that alter authentication settings and access profiles, simulating authentication issues or incorrect account usage.

AOR Mutants:

- *invalid_authentication*: Substitutes the authentication profile with an invalid account.
- *remove_admin_permission*: Removes critical permissions required for provisioning.
- *access_key_to_role*: Replaces *access_key* authentication with *role*-based authentication.
- *restricted_profile*: Replaces a high-permission profile with a standard user profile.
- *disable_mfa*: Disables multi-factor authentication for a user.
- *expired_token*: Simulates the use of an expired session token.
- *invalid_auth_region*: Changes the authentication region to an incompatible one.
- *remove_write_permission*: Removes write permissions from a profile.

5.1.3 *3. Resource Operator Replacement (ROR).* This category includes mutants that directly alter resource properties, such as timeouts, lifecycle parameters, or other critical configurations.

ROR Mutants:

- *extended_timeout*: Extends the execution timeout.
- *instance_type_t2_to_t3*: Changes the instance type.
- *invalid_instance_type*: Replaces the instance type with an invalid value.
- *remove_create_before_destroy*: Alters the resource lifecycle by removing the create-before-destroy setting.
- *force_destroy*: Enables resource destruction, simulating a security change.

5.1.4 *4. Network Operator Replacement (NOR).* This category focuses on mutants that modify network and security configurations, such as security groups, firewalls, and port settings, simulating resource exposure or security issues.

NOR Mutants:

- *open_security_group*: Removes restrictions on security groups.

- *open_sensitive_ports*: Allows unrestricted traffic on sensitive ports.
- *invalid_network_variable*: Configures invalid values for network variables.
- *invalid_vpc_cidr*: Sets an invalid CIDR for a VPC.

5.1.5 5. State Operator Replacement (SOR). This category involves mutants that alter the infrastructure state, potentially leading to inconsistencies or synchronization issues between operations.

SOR Mutants:

- *skip_terraform_import*: Simulates an inconsistent state without using *terraform import*.
- *disable_state_locking*: Disables state locking, simulating collaboration issues.

5.1.6 6. Provider Operator Replacement (POR). This category focuses on mutants that modify or switch providers, simulating compatibility or resource availability issues.

POR Mutants:

- *provider_aws_to_google*: Replaces the AWS provider with Google.
- *provider_incompatibility*: Forces incompatibility between providers.

5.1.7 7. Variable Operator Replacement (VOR). This category includes mutants that modify or remove essential variables and default values, simulating missing or misconfigured critical variables.

VOR Mutants:

- *remove_required_variable*: Removes variables essential for configuration.
- *extreme_count_value*: Sets extreme values for the *count* parameter.
- *region_us_east_to_eu_west*: Changes the resource region.
- *invalid_region*: Configures an invalid region.
- *invalid_data_type*: Replaces values with incorrect data types.

5.2 Framework Validation

The presented operators were evaluated in an experimental scenario using a pre-existing standard Terraform infrastructure. This project was configured with tests written using the Terratest[3]. Mutants were generated based on the proposed mutation operators, modifying key elements of the infrastructure configuration to assess the framework's effectiveness. In total, 10 mutants were created and distributed among the operator categories. The detailed mutation testing results, including the survival rate of mutants across all categories, are provided in Appendix B to offer a comprehensive view of the framework's validation process.

Due to the limited scope of the Terraform project, not all proposed mutation operators were applied. This small-scale infrastructure configuration ensured that the selected mutations were relevant and meaningful within the project context, allowing for a focused evaluation of the framework's capabilities. Despite the reduced scope, all generated mutants survived, underscoring significant gaps in the test suite's ability to detect the introduced defects. The framework applied each mutation by systematically modifying the Terraform configuration files according to predefined patterns and rules specific to each operator category. These modifications were executed in a controlled environment, ensuring seamless integration with the testing toolchain and enabling an accurate evaluation of the test suite's effectiveness.

[3]https://terratest.gruntwork.io

6 Threats to Validity

This section outlines the potential threats to the validity of this research and the strategies adopted to mitigate them. These threats are categorized into internal, external, and construct validity.

6.1 Internal Validity

Internal validity pertains to the extent to which the results are attributable to the research methodology rather than other factors. In this study, the following threats were identified:

- **Survey Response Bias:** Respondents might have provided socially desirable answers or exaggerated their experiences. To mitigate this, the survey was anonymous, encouraging honest responses without fear of judgement.
- **Self-Reported Data:** As the study is based on self-reported experiences, there is a risk of inaccuracies or incomplete accounts. To address this, open questions were included to allow participants to elaborate on their responses, providing richer context and reducing ambiguity.

6.2 External Validity

External validity refers to the extent to which the findings can be generalized to other contexts or populations. The following issues were considered:

- **Sample Representation:** The survey targeted Terraform developers through professional networks, which might not represent all Terraform users or developers working with other IaC tools. Although efforts were made to reach diverse participants, the findings may still be biased toward active online communities.
- **Context-Specific Results:** As the research is focusing on Terraform, the results may not generalize to other IaC tools. However, the methodology and insights could inform studies on similar tools or practices.

6.3 Survey Construct Validity

Construct validity examines whether the study measures what it intends to measure. The questionnaire was designed following the mixed methods guidelines proposed by Creswell [1], ensuring alignment with the research objectives.

The survey incorporated both quantitative and qualitative elements:

- Quantitative questions, such as Likert scale items, assessed developers' experience and perceptions of IaC challenges.
- Qualitative open-ended questions collected detailed descriptions of common defects, their causes, and their impacts on Terraform configurations.

To enhance validity, the questionnaire was iteratively refined and tested in a pilot with Terraform developers. In addition, the questions were organized thematically to ensure a logical flow, maintaining clarity and relevance to the objectives of the study.

7 Conclusion, limitations and Future Work

The results from this study demonstrates the effectiveness of the proposed mutation operators in simulating defects that existing unit and integration tests failed to detect, emphasizing the need for more robust and comprehensive testing strategies to improve code quality and reliability in Infrastructure as Code (IaC) projects. While the findings offer valuable insights, the research has certain limitations. The reliance on a survey to gather data from Terraform developers may have introduced biases due to the limited availability and willingness of participants to share their experiences. Additionally, the validation was restricted to a single small-scale Terraform project, which constrained the application of all proposed mutation operators and may not fully reflect the diversity and complexity of real-world IaC scenarios.

Future research should address these limitations by shifting from survey-based data collection to mining open-source repositories on platforms such as GitHub to identify recurring defects. Comparing these defects with the proposed mutation operators would enable a more thorough evaluation of their coverage and relevance.

Moreover, validating the operators across multiple and varied IaC projects would provide a more comprehensive assessment of their applicability and effectiveness. These efforts will contribute to advancing testing and validation methodologies, fostering greater reliability, and supporting more secure infrastructure management practices in the IaC domain.

A Survey Questions and Response Types

B Detailed Mutation Testing Results

References

[1] John W. Creswell. 2014. *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches* (4th ed.). SAGE Publications, Thousand Oaks, CA.

[2] Richard A. DeMillo and Fred G. Sayward. 1979. Program Mutation: A New Approach to Program Testing. In *Proceedings of the ACM Annual Conference*. ACM, 488–491. https://doi.org/10.1145/800177.810742

[3] Md Mahadi Hasan, Farhana Afroz Bhuiyan, and Akond Rahman. 2020. Testing Practices for Infrastructure as Code. In *Proceedings of the ACM SIGSOFT International Workshop on Languages and Tools for Next Generation Testing (LTNG)*. ACM, 12–21.

[4] Yue Jia and Mark Harman. 2011. An Analysis and Survey of the Development of Mutation Testing. *IEEE Transactions on Software Engineering* 37, 5 (2011), 649–678.

[5] Kief Morris. 2016. *Infrastructure as code: managing servers in the cloud.* " O'Reilly Media, Inc.".

[6] Goran Petrovic, Marcia Teixeira, Monika Schmidt, and David Schuler. 2021. Practical Mutation Testing at Scale: A View from Google. In *Proceedings of the IEEE International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 165–175.

[7] Akond Rahman, Sarah Elder, Faysal Hossain Shezan, Vanessa Frost, Jonathan Stallings, and Laurie Williams. 2019. Bugs in Infrastructure as Code. *arXiv preprint arXiv:1809.07937v2* (2019). https://arxiv.org/pdf/1809.07937v2 Accessed: November 17, 2024.

[8] Akond Rahman, Sarah Elder, Faysal Hossain Shezan, Vanessa Frost, Jonathan Stallings, and Laurie Williams. 2019. Categorizing Defects in Infrastructure as Code. *Empirical Software Engineering* 24 (2019), 345–372.

[9] Reinhard Ramler, Thomas Wetzlmaier, and Christian Klammer. 2017. Mutation Testing in Safety-Critical Systems. *Software Quality Journal* 25 (2017), 405–428.

[10] Ryo Shimizu, Yuto Nunomura, and Hiroki Kanuka. 2024. Test-Suite-Guided Discovery of Least Privilege for Infrastructure as Code. *Automated Software Engineering* 31 (2024), 523–540.

[11] Daniel Sokolowski and Guido Salvaneschi. 2023. Towards Reliable Infrastructure as Code. In *Proceedings of the IEEE International Conference on Software Architecture Companion (ICSA-C)*. IEEE, 125–134.

[12] Daniel Sokolowski, Dominic Spielmann, and Guido Salvaneschi. 2024. Automated Infrastructure as Code Program Testing. *IEEE Transactions on Software*

Table 1: Survey Questions and Response Types

#	Question	Response Type
1	What is your level of experience with Infrastructure as Code (IaC) in general?	Likert Scale (1 to 5)
2	What is your level of experience with Terraform?	Likert Scale (1 to 5)
3	In which type of organization or sector do you use IaC projects?	Open-ended
4	Have you encountered bugs or unexpected behavior in Terraform configurations?	Objective (Yes/No)
5	If yes, describe the bug or unexpected behavior you encountered in your Teraform configurations.	Open-ended
6	In your experience, what types of bugs are most frequent when implementing a Terraform configuration?	Open-ended
7	What types of bugs do you believe most impact the reliability or security of an IaC project?	Open-ended
8	How important are IaC projects to the success of your operations?	Likert Scale (1 to 5)
9	What tools or methods do you currently use to test and validate your Terraform configurations?	Open-ended
10	Do you think specific tests for Teraform would be useful to identify bugs in your IaC project?	Likert Scale (1 to 5)
11	Would you like to share any other experiences with problems, challenges, or bugs in Terraform configurations that were not addressed in the previous questions?	Open-ended

Table 2: Consolidated Mutation Testing Results Across All Categories

Category	Operator	Result
POR	aws_to_google	Survived
VOR	invalid_region	Survived
DOR	remove_critical_dep	Survived
AOR	remove_admin_permission	Survived

Engineering 50 (2024), 654–670.

[13] Helge Teppan, Lars Håkon Flå, and Martin Gilje Jaatun. 2022. A Survey on Infrastructure-as-Code Solutions for Cloud Development. In *Proceedings of the IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE, 100–109.

[14] Bo Wang, Mingda Chen, Youfang Lin, Mike Papadakis, and Jie M. Zhang. 2025. On the Use of Large Language Models in Mutation Testing. *J. ACM* 1, FSE (2025), 1:1–1:23.