Enhancing LLM-Based IaC Transformation Accuracy Using an AST-Based Approach*

Seohee Kim[1], Mijin Kim[1], Yesol Sim[1], and Seongmin Kim[1]

Sungshin Women University, Seoul, Republic of Korea
{20231068, 220254011, 20221109, sm.kim}@sungshin.ac.kr

Abstract

Recent LLM-driven code conversions, including Infrastructurea-as-code (IaC), often fail to preserve resource dependencies and provider-specific schema consistency, leading to deployment errors. This study explores an AST(Abstract Syntax Tree)-based approach to improve the structural accuracy and deployment stability of LLM-based Terraform code transformation in multi-cloud environments. In experiments converting Terraform configurations from AWS to Azure, the AST-based method maintained hierarchical structures and successfully executed terraform apply, whereas direct LLM conversion resulted in schema mismatches and unsupported attributes. These results demonstrate that incorporating ASTs enables LLMs to capture structural semantics more effectively, improving the practicality of multi-cloud IaC transformation.

Keywords—Infrastructure as Code (IaC), Abstract Syntax Tree (AST), Multi-cloud.

1 Introduction

As cloud-native environments continue to expand, the DevOps paradigm, which unifies development and operations, has become a dominant practice for modern infrastructure management. Within this paradigm, Infrastructure as Code (IaC) has emerged as a key enabler, allowing infrastructure to be defined, deployed, and maintained programmatically. Among various IaC tools, Terraform has established itself as the de facto standard for provisioning resources across major public clouds, including AWS, Azure, and Google Cloud [1] [1]

Despite its multi-cloud capabilities, Terraform faces inherent challenges in code portability due to vendor-specific heterogeneity in resource models, attribute definitions, and naming conventions. To mitigate these issues, recent studies have leveraged Large Language Models (LLMs) for automated IaC generation and conversion [2], establishing criteria for resource configuration and template generation. However, these approaches remain limited to single-cloud contexts and have not yet achieved effective cross-cloud code translation. Because LLMs primarily operate at the natural-language level using user prompts, they often struggle to preserve structural relationships between resources or to align with provider-specific schema definitions.

To address these shortcomings, this study introduces an Abstract Syntax Tree (AST) as an intermediate representation within the LLM-based IaC translation workflow. By incorporating structural abstraction through ASTs, the proposed method aims to enhance translation fidelity and maintain schema consistency across heterogeneous cloud environments.

*Proceedings of the 9th International Conference on Mobile Internet Security (MobiSec'25), Article No. P-32, December 16-18, 2025, Sapporo, Japan.

[1] https://developer.hashicorp.com/terraform/intro, last viewed Oct. 10

2 Proposed Method

The experiments were conducted using Terraform configurations that had been successfully provisioned in the AWS environment. In the proposed AST-based transformation approach, each Terraform configuration was parsed into a JSON-formatted AST using the python-hcl2 module. The resulting structured representation was then provided to the LLM as input. For comparison, a direct transformation approach was also tested as a baseline, in which the original Terraform configuration itself was directly supplied to the LLM. After repeatedly applying both methods to multiple configurations, we found that the direct transformation approach consistently failed during the terraform apply stage, whereas the AST-based transformation successfully executed terraform apply, resulting in proper resource deployment.

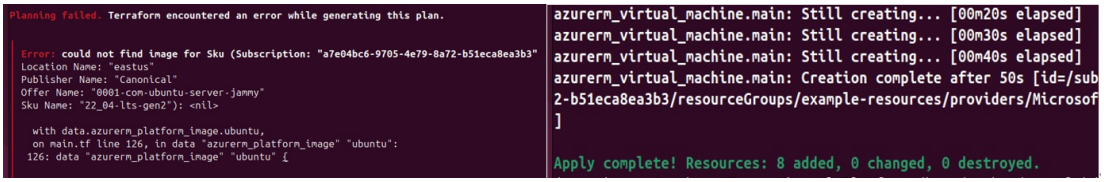

Figure 1: Terraform plan results using the LLM-based direct conversion (left) and the AST-based proposed approach (right)

Figure 1 presents the results of applying the two conversion approaches to AWS Simple Queue Service (SQS) code. In the direct LLM-based conversion, an error occurred during the terraform plan stage because the generated code included an unsupported tags argument. This issue arose from the LLM's incorrect generalization that AWS SQS resources always support individual tags, which it mistakenly applied to the Azure resource. These observations suggest that the LLM relied on surface-level attribute mapping rather than understanding the structural semantics of the code. In contrast, the AST-based conversion successfully created all resources during the terraform apply stage. By incorporating structural information, the AST-based approach overcame the limitation of the direct conversion by correctly reflecting Azure's tagging policy, where tag support varies depending on the resource type.

3 Conclusion

This study proposes an approach that leverages the AST as an intermediate representation to overcome the structural limitations of LLM-based Terraform code transformation. In experiments converting Terraform configurations from AWS to Azure, the direct code input method produced missing resource dependencies and schema mismatch errors. In contrast, the AST-based transformation preserved hierarchical structures and attributes, achieving stable execution. These findings demonstrate that incorporating ASTs helps maintain the structural context of the code, enabling LLMs to more accurately represent cloud resource models. Overall, the AST-based approach enhances the practicality of multi-cloud IaC translation and provides a foundation for future research on LLM-driven IaC transformation that effectively captures code structure and dependencies.

References

[1] HashCorp. Intro To Terraform. Accessed Oct. 10.

[2] Patrick T Kon et al. Iac-eval: A code generation benchmark for cloud infrastructure-as-code programs. *Advances in Neural Information Processing Systems*, 37:134488–134506, 2024.