A Multi-Agent Framework for Autonomous Generation and Validation of IaC and PaC*

Jueun Son[1], Jeongeun Ryu[1], and Seongmin Kim[1]

Sungshin Women's University, Seoul, Korea
{20230378, 20221112, sm.kim}@sungshin.ac.kr

Abstract

The complexity of managing large-scale cloud infrastructure and security policies has made Infrastructure-as-Code (IaC) and Policy-as-Code (PaC) essential. In this study, we propose a multi-agent framework that integrates Retrieval-Augmented Generation (RAG) with verification tools to iteratively generate, validate, and refine IaC and PaC.

Keywords— Infrastructure-as-Code, Policy-as-Code, Multi-Agent, and DevSecOps

1 Introduction

The widespread adoption of cloud computing makes the automation of infrastructure and policy management essential for the effective management of large-scale systems. However, manually managing complex infrastructure and security regulations requires a high level of expertise, and even minor errors or misconfigurations can result in security vulnerabilities. Consequently, IaC and PaC, which define infrastructure and policies as code to automate configuration and policy enforcement, have emerged as key technologies. Because IaC and PaC are managed separately at the code level, the traditional approach, which relies on a single pipeline and manual verification, can easily result in inconsistencies between IaC and PaC [1].

To address the complexity inherent in manual code generation and verification, recent efforts to automatically generate code using Large Language Models (LLMs) have received increasing attention. However, existing approaches [2] either employ a single agent or verify external code, assuming that it is accurate and can serve as a reliable baseline for verification. This study proposes a framework that introduces an iterative loop from generation to verification and automatic refinement by integrating Retrieval-Augmented Generation (RAG) and verification tools within a multi-agent architecture. It generates both IaC and PaC from natural language requirements and regulations while ensuring consistency between IaC and PaC through cross-validation for automated yet proactive error correction.

2 Architecture Overview

We aim to ultimately produce safe and compliant IaC and PaC code through the cyclical stages of *generation, validation*, and *feedback*. Consistency between resources and policies is ensured through iterative cross-referencing of IaC and PaC, while static scanning and policy engine validation prevent errors, such as configuration mistakes and policy violations. Figure 1 illustrates an overview of the architecture.

Generation Layer. When requirements and policies expressed in natural language are provided, enhanced prompts are delivered to the LLM through RAG-G, which utilizes language specifications and official documentation as its knowledge base. Each generator then produces IaC and PaC based on a standardized schema and evidence retrieved via RAG-G.

*Proceedings of the 9th International Conference on Mobile Internet Security (MobiSec'25), Article No. P-4, December 16-18, 2025, Sapporo, Japan.

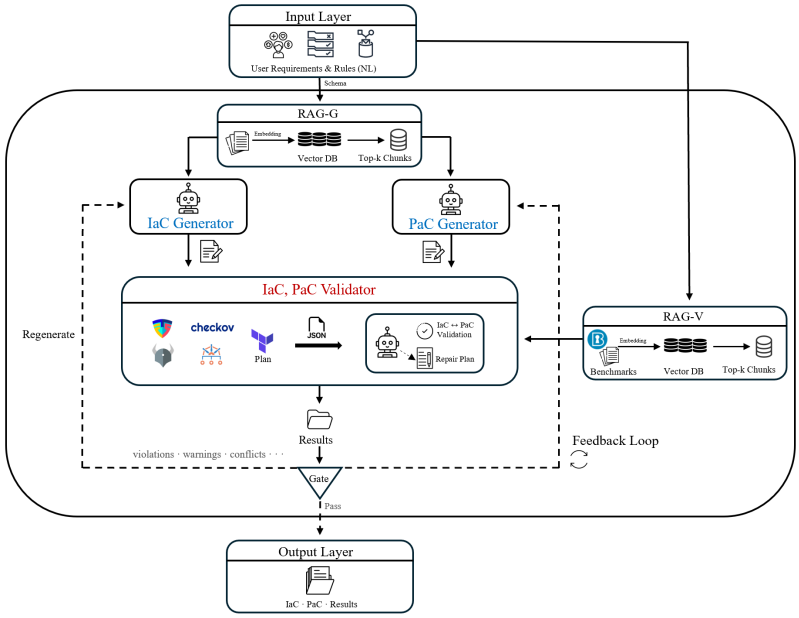

Figure 1: Overall System Architecture

Validation Layer. integrates static analysis and verification into a two-step process. First, IaC is analyzed using verification tools (e.g., Checkov) to detect code vulnerabilities and rule violations, while PaC employs policy engines (e.g., Open Policy Agent) to verify policy compliance and identify violations. Subsequently, the preprocessed infrastructure (e.g., Terraform plan) and policy rules are analyzed by an LLM using RAG-V to assess user intent alignment, validate IaC against PaC, and identify coverage gaps and rule conflicts.

Feedback Loop. Cases that fail the validation layer are refined through a feedback loop. After the validator's suggested modifications are incorporated into the generator, the regenerated code undergoes the same two-step verification process. To handle cases where iteration limits are reached or high-risk configurations persist, the automated loop can be escalated to human review for further intervention. By separating technical evaluation (step 1) from reference-based review (step 2), the system is designed to converge toward consistency between IaC and PaC with fewer iterations through accelerated error detection and correction.

Ultimately, this architecture follows an iterative process to derive IaC and PaC that meet infrastructure and policy requirements expressed in natural language. The generator leverages code patterns and reasoning focused on modular structures, while the validator utilizes policy rules and consistency-oriented reasoning to reduce human error and shorten feedback cycles. Future research will focus on implementing the framework in a real-world system.

References

[1] Sarathe Vijayaraghavan. Policy as code: A paradigm shifts in infrastructure security and governance. *World Journal of Advanced Research and Reviews*, 26:3399–3405, 04 2025.

[2] Akshay Mittal and Vivek Venkatesan. Practical integration of large language models into enterprise ci/cd pipelines for security policy validation: An industry-focused evaluation. In *2025 IEEE International Conference on Service-Oriented System Engineering (SOSE)*, pages 197–203, 2025.