École Nationale Supérieure d'Informatique et d'Analyse des Systèmes

Mohammed V University in Rabat

END OF YEAR PROJECT REPORT

AI-Based Solutions for Proactive Security Recommendations in Web Applications

DevSecOps & Infrastructure as Code Compliance Automation

Prepared by:

Aymen Nadi & Hamza Zaoui

Academic Supervisor:

[Mme. Souad Sadki]

Examination Committee:

Mme. Souad Sadki

Mme. Ajhoun Rachida

Academic Year 2024-2025

I) Acknowledgments

We would like to express our sincere gratitude to all those who contributed to the successful completion of this end-of-year project.

First and foremost, we extend our deepest appreciation to our academic supervisor, **Mme. Souad Sadki**, for her invaluable guidance, constructive feedback, and unwavering support throughout this research journey. Her expertise in cybersecurity and DevSecOps practices has been instrumental in shaping the direction and quality of this work.

We are grateful to the examination committee members, **Mme. Souad Sadki** and **Mme. Ajhoun Rachida**, for their time and expertise in evaluating our work and providing valuable insights that have enhanced the overall quality of this project.

We would like to extend our appreciation to **Mr. Driss Bouzidi**, Head of the Information Systems Security (SSI) Department, for providing us with the opportunity to work on this project within the cybersecurity field.

Our sincere thanks go to the faculty and staff of the École Nationale Supérieure d'Informatique et d'Analyse des Systèmes (ENSIAS) and Mohammed V University in Rabat for providing an excellent academic environment and resources that made this research possible.

We also acknowledge the open-source community and researchers whose prior work in artificial intelligence, DevSecOps, and Infrastructure as Code compliance has laid the foundation for this project. Their contributions to the field have been invaluable in advancing our understanding of AI-driven security solutions.

Finally, we express our heartfelt gratitude to our families and friends for their continuous encouragement, patience, and moral support throughout our academic journey.

Aymen Nadi & Hamza Zaoui

II) Abstract

The rapid evolution of modern web application development, characterized by the shift from monolithic architectures to microservices and the widespread adoption of DevOps practices, has introduced significant security and compliance challenges. Traditional Infrastructure as Code (IaC) validation methods are often reactive, manual, and inadequate for proactively addressing security misconfigurations or generating compliant configurations from the outset.

This project presents an innovative AI-driven DevSecOps pipeline that leverages Large Language Models (LLMs) to automatically analyze IaC scripts against security best practices and compliance mandates, with a specific focus on the NIST Cybersecurity Framework. The solution goes beyond traditional validation by proactively generating compliant and secure IaC configurations, embedding security and compliance intelligence directly into the development workflow.

The proposed system implements a comprehensive approach that combines automated compliance checking, intelligent remediation suggestions, and proactive secure configuration generation. By integrating these capabilities into the DevSecOps lifecycle, the solution aims to mitigate security risks before deployment while maintaining development agility and ensuring regulatory adherence.

Key contributions of this work include: (1) the development of an AI-powered compliance analysis engine for IaC scripts, (2) the implementation of automated security recommendation generation based on established frameworks, (3) the creation of a proactive secure configuration generator, and (4) the integration of these components into a cohesive DevSecOps pipeline.

The evaluation demonstrates the effectiveness of the proposed approach in identifying security vulnerabilities, generating actionable recommendations, and producing compliant IaC configurations, thereby bridging the gap between rapid software delivery and security compliance requirements.

Keywords: DevSecOps, Infrastructure as Code, Artificial Intelligence, Large Language Models, NIST Cybersecurity Framework, Compliance Automation, Security Recommendations, Web Application Security

III) Résumé

L'évolution rapide du développement d'applications web modernes, caractérisée par le passage d'architectures monolithiques aux microservices et l'adoption généralisée des pratiques DevOps, a introduit des défis significatifs en matière de sécurité et de conformité. Les méthodes traditionnelles de validation de l'Infrastructure en tant que Code (IaC) sont souvent réactives, manuelles et inadéquates pour traiter de manière proactive les erreurs de configuration de sécurité ou générer des configurations conformes dès le départ.

Ce projet présente un pipeline DevSecOps innovant basé sur l'IA qui exploite les Grands Modèles de Langage (LLM) pour analyser automatiquement les scripts IaC par rapport aux meilleures pratiques de sécurité et aux exigences de conformité, avec un focus spécifique sur le Framework de Cybersécurité NIST. La solution va au-delà de la validation traditionnelle en générant de manière proactive des configurations IaC conformes et sécurisées, intégrant l'intelligence de sécurité et de conformité directement dans le flux de développement.

Le système proposé implémente une approche complète qui combine la vérification automatisée de la conformité, les suggestions intelligentes de remédiation et la génération proactive de configurations sécurisées. En intégrant ces capacités dans le cycle de vie DevSecOps, la solution vise à atténuer les risques de sécurité avant le déploiement tout en maintenant l'agilité de développement et en assurant le respect réglementaire.

Les contributions clés de ce travail incluent : (1) le développement d'un moteur d'analyse de conformité alimenté par l'IA pour les scripts IaC, (2) l'implémentation de la génération automatisée de recommandations de sécurité basées sur des frameworks établis, (3) la création d'un générateur proactif de configurations sécurisées, et (4) l'intégration de ces composants dans un pipeline DevSecOps cohérent.

L'évaluation démontre l'efficacité de l'approche proposée dans l'identification des vulnérabilités de sécurité, la génération de recommandations exploitables et la production de configurations IaC conformes, comblant ainsi l'écart entre la livraison rapide de logiciels et les exigences de conformité de sécurité.

Mots-clés : DevSecOps, Infrastructure en tant que Code, Intelligence Artificielle, Grands Modèles de Langage, Framework de Cybersécurité NIST, Automatisation de la Conformité, Recommandations de Sécurité, Sécurité des Applications Web

IV) General Context

Industry Context

The digital transformation of businesses worldwide has fundamentally altered the landscape of software development and deployment. Organizations are increasingly adopting cloud-native architectures, microservices patterns, and DevOps methodologies to achieve greater agility, scalability, and time-to-market advantages. This shift represents a paradigm change from traditional, monolithic application architectures to distributed, containerized systems that can be rapidly developed, tested, and deployed.

The Infrastructure as Code (IaC) approach has emerged as a cornerstone of this transformation, enabling organizations to manage and provision their computing infrastructure through machine-readable definition files rather than through physical hardware configuration or interactive configuration tools. Popular IaC tools such as Terraform, AWS CloudFormation, and Kubernetes YAML manifests have become essential components of modern DevOps pipelines.

However, this rapid evolution has introduced significant challenges in maintaining security and regulatory compliance. The traditional approach of implementing security measures as an afterthought is no longer viable in environments where infrastructure changes occur frequently and automatically.

Regulatory and Compliance Landscape

Modern organizations operate within an increasingly complex regulatory environment. International regulations such as the General Data Protection Regulation (GDPR) in Europe, national laws like the California Consumer Privacy Act (CCPA) and the Federal Information Security Modernization Act (FISMA) in the United States, and industry-specific standards such as the Payment Card Industry Data Security Standard (PCI DSS) create a web of compliance requirements that organizations must navigate.

The financial and reputational consequences of non-compliance are substantial. GDPR violations alone can result in fines of up to 4% of annual global turnover or €20 million, whichever is higher. Beyond financial penalties, compliance failures can lead to loss of customer trust, legal liabilities, and operational disruptions.

The DevSecOps Imperative

The concept of DevSecOps has emerged as a response to the need for integrating security practices into the rapid software delivery cycles characteristic of DevOps. Unlike traditional security approaches that rely on periodic audits and manual reviews, DevSecOps emphasizes

the automation of security controls and the integration of security considerations throughout the entire software development lifecycle.

The principle of "Compliance as Code" represents a natural extension of this philosophy, advocating for the codification of compliance requirements and their automated enforcement within the development pipeline. This approach promises to address the inherent tension between the need for rapid software delivery and the requirement for rigorous security and compliance controls.

Technological Enablers

Recent advances in artificial intelligence, particularly in the field of Large Language Models (LLMs), have opened new possibilities for automating complex analysis and generation tasks that were previously the exclusive domain of human experts. These technologies offer the potential to understand, analyze, and generate Infrastructure as Code configurations with a level of sophistication that approaches human expertise.

The convergence of these technological capabilities with the pressing need for automated security and compliance solutions creates a unique opportunity to develop AI-driven systems that can proactively address security and compliance challenges in modern software development environments.

Project Positioning

This project is positioned at the intersection of these technological and business trends, addressing the critical need for intelligent, automated solutions that can ensure security and compliance without compromising development velocity. By leveraging state-of-the-art AI technologies to create proactive security and compliance capabilities, this work contributes to the evolution of DevSecOps practices and the broader goal of secure, compliant, and agile software development.

V) List of Abbreviations

Abbreviation	Definition
AI	Artificial Intelligence
API	Application Programming Interface
AWS	Amazon Web Services
CCPA	California Consumer Privacy Act
CI/CD	Continuous Integration/Continuous Deployment
CSRF	Cross-Site Request Forgery
DevOps	Development and Operations
DevSecOps	Development, Security, and Operations
FISMA	Federal Information Security Modernization Act
GDPR	General Data Protection Regulation
IaC	Infrastructure as Code
JSON	JavaScript Object Notation
LLM	Large Language Model
ML	Machine Learning
NIST	National Institute of Standards and Technology
NLP	Natural Language Processing
OWASP	Open Web Application Security Project
PCI DSS	Payment Card Industry Data Security Standard
REST	Representational State Transfer
SAST	Static Application Security Testing
SDLC	Software Development Life Cycle
SQL	Structured Query Language
TLS	Transport Layer Security
UI	User Interface
XSS	Cross-Site Scripting
YAML	YAML Ain't Markup Language

Contents

List of Figures

Chapter 1

Introduction

Modern web application development is rapidly evolving, driven by the shift from monolithic architectures to scalable microservices and the widespread adoption of DevOps practices aimed at accelerating software delivery cycles [3, 10]. This paradigm shift, while beneficial for agility, introduces significant security and **compliance challenges**. Compliance, defined as adherence to standards, regulations, laws, and similar prescriptions [9], is paramount, especially in cloud-native environments handling sensitive data. Organizations face a complex web of requirements, including international regulations like GDPR [9], national laws (e.g., CCPA, FISMA in the US [9]), and industry-specific standards such as PCI DSS [9].

Failure to comply can result in substantial penalties and reputational damage [9]. Consequently, **security compliance** – ensuring systems meet a defined set of security requirements derived from these mandates [9] – cannot be an afterthought. However, traditional methods for validating Infrastructure as Code (IaC), the bedrock of cloud automation, often fall short. They tend to be reactive, manual, and struggle to proactively remediate identified misconfigurations or generate secure configurations aligned with compliance needs from the outset [1].

Integrating compliance into the fast-paced DevSecOps lifecycle is a key challenge. The concept of **Compliance as Code**, embedding compliance requirements directly into the delivery pipeline using automated tools and codified policies [9], offers a promising path forward. This approach aligns with the DevSecOps philosophy of automation and continuous feedback.

Addressing this critical gap, this project introduces an innovative, AI-driven DevSecOps pipeline. It leverages Large Language Models (LLMs) to automatically analyze IaC scripts against security best practices and compliance mandates (specifically focusing on the NIST Cybersecurity Framework and related controls), and crucially, to **proactively generate compliant and secure IaC configurations**. By embedding security and compliance intelligence directly into the development workflow, this approach aims to mitigate risks before deployment, ensuring that agility does not come at the cost of security or regulatory adherence [1, 2].

Chapter 2

Conceptual and Technological Foundations

This chapter establishes the theoretical and practical foundations underpinning this project by examining the evolution of software architecture paradigms, the transformative role of generative AI in modern software development, and the critical importance of secure SDLC practices through DevSecOps methodologies. Through comparative analysis and systematic review of existing research, we identify the gaps that this project aims to address.

2.1 Architectural Paradigms: From Monoliths to Microservices

The evolution of software architecture represents a fundamental shift in how we design, deploy, and maintain complex systems. Understanding this evolution is crucial for appreciating the security challenges and opportunities in modern DevSecOps practices.

2.1.1 Monolithic Architecture

The traditional **monolithic architecture** constructs applications as single, tightly coupled deployment units where all components are interconnected and interdependent. This approach offers several advantages in early development stages, including simplified deployment, easier debugging, and straightforward testing procedures. However, as applications scale, monoliths present significant limitations:

- **Scalability constraints**: The entire application must be scaled even when only specific components experience increased load
- **Technology lock-in**: All components must use the same technology stack, limiting innovation and optimization opportunities
- **Maintenance complexity**: Code changes in one area can have unpredictable effects throughout the system
- **Deployment risks**: Updates require redeploying the entire application, increasing the potential impact of failures

2.1.2 Microservices Architecture

In contrast, **microservices architecture** decomposes applications into collections of small, autonomous, loosely coupled services, each responsible for specific business capabilities [3]. This architectural paradigm enables:

- **Independent scalability**: Services can be scaled individually based on demand
- **Technology diversity**: Each service can utilize the most appropriate technology stack
- **Fault isolation**: Service failures are contained and don't necessarily cascade throughout the system
- **Team autonomy**: Development teams can work independently on different services

However, microservices introduce new complexities that necessitate sophisticated management approaches:

- **Inter-service communication**: Managing network latency, service discovery, and communication protocols
- **Distributed data management**: Ensuring data consistency across multiple services and databases
- **Deployment orchestration**: Coordinating deployments across multiple independent services
- **Monitoring and observability**: Tracking system behavior across distributed components

These complexities are typically addressed through service meshes, container orchestration platforms, and comprehensive monitoring solutions, making robust automation and security practices essential [3].

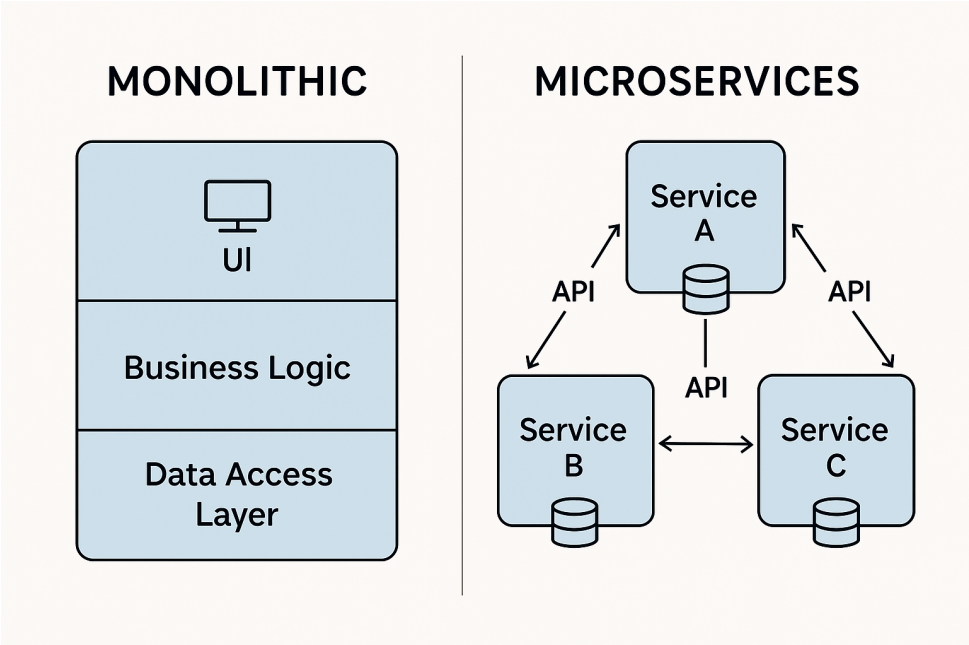


Figure 2.1: Architectural comparison illustrating the structural differences between monolithic and microservices approaches, highlighting the implications for deployment, scaling, and maintenance strategies.

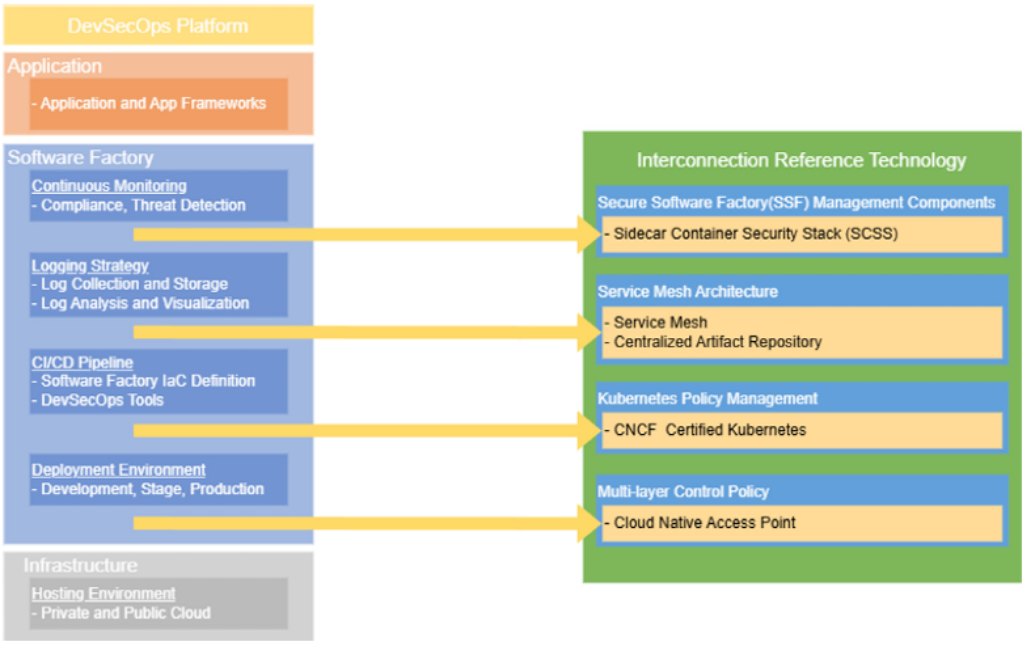


Figure 2.2: Comprehensive DevSecOps platform architecture illustrating the multi-layered approach from application level through software factory to infrastructure, highlighting interconnection reference technologies including Service Mesh, CNCF Certified Kubernetes, and Cloud Native Access Point integration.[5]

2.2 Generative AI in Software Development

The integration of Generative Artificial Intelligence (GenAI), particularly Large Language Models (LLMs), represents a paradigm shift in software development practices. These technologies are revolutionizing multiple phases of the Software Development Lifecycle (SDLC) through intelligent automation and assistance.

2.2.1 Current GenAI Tools and Capabilities

Leading GenAI platforms such as **GitHub Copilot**, **Amazon CodeWhisperer**, and underlying models like **GPT** and **CodeBERT** provide comprehensive development assistance through:

Code Generation and Completion: Automated generation of code snippets, function implementations, and boilerplate code based on natural language descriptions or partial implementations

Code Analysis and Explanation: Interpretation and documentation of complex code segments, making legacy systems more maintainable

Language Translation: Conversion between programming languages while preserving functionality and best practices

Bug Detection and Prevention: Identification of potential vulnerabilities, logic errors, and performance issues during development

Test Generation: Automated creation of unit tests, integration tests, and test data based on code analysis

Documentation Generation: Creation of technical documentation, API specifications, and user guides

2.2.2 GenAI in Infrastructure as Code

In the context of this project, GenAI capabilities extend beyond traditional software development to encompass **Infrastructure as Code (IaC)** management. This represents a significant advancement in automating DevSecOps practices by enabling:

- **Proactive Security Integration**: Generation of secure and compliant IaC configurations from the outset
- **Compliance Automation**: Automatic alignment with security frameworks such as NIST, SOC 2, and industry-specific standards
- **Configuration Optimization**: Intelligent recommendations for performance, cost, and security improvements
- **Policy as Code**: Automated generation of governance policies and compliance rules

2.3 DevSecOps: Integrating Security Throughout the SDLC

DevSecOps represents the evolution of traditional development practices by integrating security as a fundamental, shared responsibility across every phase of the software development lifecycle. This approach shifts security "left" in the development process, embedding it through automation, collaboration, and continuous monitoring.

2.3.1 DevSecOps Pipeline Architecture

The DevSecOps methodology transforms the traditional sequential SDLC into a continuous, security-integrated workflow. Figure 2.3 illustrates the typical stages and their associated security controls:

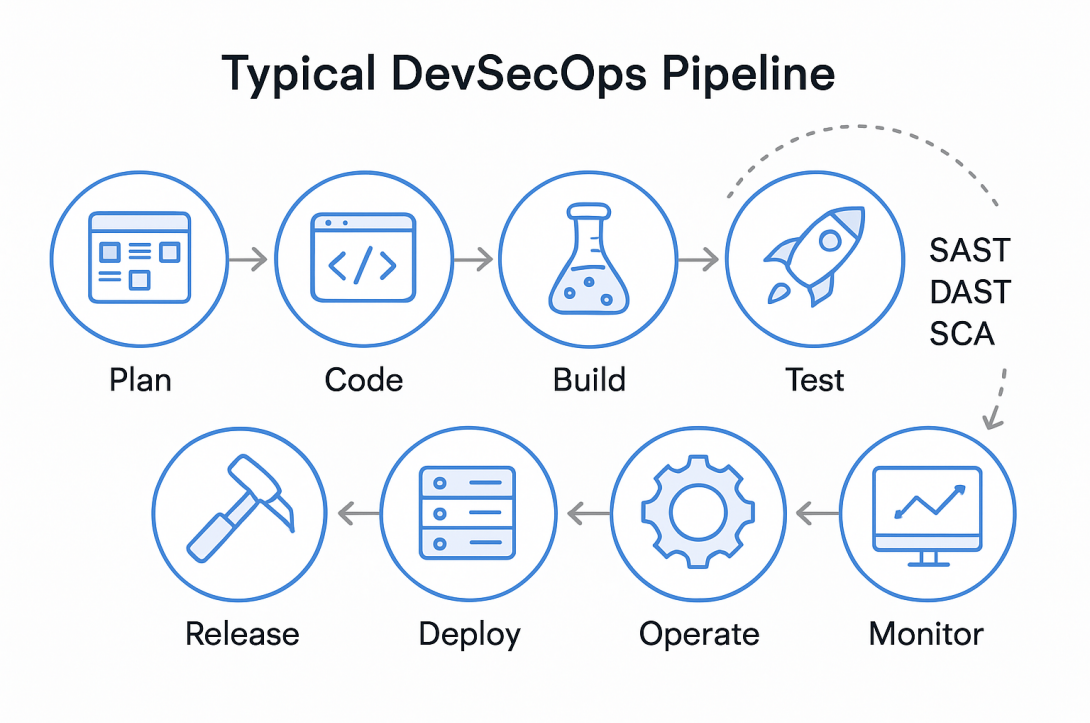


Figure 2.3: Comprehensive DevSecOps pipeline illustrating the integration of security controls, automated testing, and continuous monitoring across all development stages.

2.3.2 Security Integration by Phase

Each phase of the DevSecOps pipeline incorporates specific security practices and tooling:

Plan Phase: Establishes security requirements through threat modeling, risk analysis, and compliance mapping. Key activities include defining functional and non-functional security requirements, conducting threat modeling exercises, and establishing security acceptance criteria. Tools include project management platforms (**Trello**, **Jira**) and specialized threat modeling solutions.

Code Phase: Implements secure coding practices through static analysis, code reviews, and developer security training. Activities encompass secure code development, peer

review processes, and integration of security-focused IDE plugins. Primary tools include version control systems (**GitHub**, **GitLab**), Static Application Security Testing (SAST) tools (**CodeQL**, **SonarQube**), and Software Composition Analysis (SCA) solutions.

Build Phase: Ensures artifact security through compilation security checks, dependency scanning, and secure packaging. This phase focuses on creating secure deployment artifacts, conducting vulnerability scanning of dependencies, and implementing secure build processes. Key tools include CI/CD servers (**GitHub Actions**, **Jenkins**), artifact registries, and container image scanners (**Docker Scout**, **Snyk**).

Test Phase: Validates security through comprehensive testing strategies including Dynamic Application Security Testing (DAST), Interactive Application Security Testing (IAST), and Infrastructure as Code compliance checks. Testing encompasses functional security testing, vulnerability assessments, and compliance validation. Tools include testing frameworks, DAST/IAST solutions, vulnerability scanners (**Trivy**, **Nessus**), and IaC validators (**Checkov**, **tfsec**).

Deploy Phase: Secures deployment through configuration management, access controls, and deployment verification. Activities include secure artifact deployment, environment configuration validation, and deployment pipeline security. Primary tools include continuous deployment platforms (**Argo CD**, **Spinnaker**), orchestration systems (**Kubernetes**, **Amazon EKS**), cloud platforms (**AWS EC2**, **Azure**), and IaC tools (**Terraform**, **CloudFormation**).

Operate/Monitor Phase: Maintains security through continuous monitoring, threat detection, and incident response. This phase focuses on runtime security monitoring, log analysis, performance monitoring, and security incident management. Key tools include Application Performance Monitoring (APM) solutions, Security Information and Event Management (SIEM) systems (**Splunk**, **ELK Stack**), and cloud security platforms (**AWS CloudWatch**, **Azure Security Center**).

Figure 2.4: Categorization of DevSecOps security tools across different phases of the development lifecycle, showing the comprehensive security coverage required for effective implementation.

2.4 Literature Review and Comparative Analysis

To understand the current state of research in DevSecOps and identify gaps that this project addresses, we conducted a systematic review of recent literature focusing on security automation, AI integration, and DevSecOps practices across multiple dimensions.

2.4.1 Systematic Literature Analysis

Our comprehensive analysis examined contemporary research across four critical evaluation criteria: **code protection mechanisms**, **secure deployment practices**, **AI-based solution integration**, and **DevSecOps workflow integration**. This multi-dimensional assessment provides insight into current research trends and identifies significant gaps in AI-driven security automation.

The analysis encompassed various research methodologies including surveys, systematic reviews, framework proposals, case studies, and empirical research studies, providing a comprehensive view of the current academic and practical landscape in DevSecOps implementation.

Paper Title	Code Protection	Secure Deployment	AI-Based Solutions	DevSecOps Integration	Type of Paper
On DevSecOps and Risk Management in Critical Infrastructures	✓	✓	✗	✓	Survey/Review
Quantitative DevSecOps Metrics for Cloud-Based Web Microservices	✓	✓	✓	✓	Research Study
DevSecOps Adoption Framework for Secure Software Development in Cloud Environments	✓	✓	✗	✓	Framework Proposal
Risk Assessment Models in DevSecOps Pipelines: A Systematic Review	✗	✓	✗	✓	Systematic Review
Automated Security Testing in Continuous Deployment: Integrating DevSecOps Practices	✓	✓	✓	✓	Research Study
Challenges and Solutions in Implementing DevSecOps for Large-Scale Systems	✗	✓	✗	✓	Survey/Review
Security Risk Management in Agile and DevSecOps Frameworks	✓	✓	✗	✓	Review
Enhancing Deployment Security through DevSecOps: A Case Study	✓	✓	✗	✓	Case Study
Integrating Threat Modeling into DevSecOps Pipelines for Improved Risk Management	✓	✓	✗	✓	Framework Proposal
Continuous Security Monitoring in DevSecOps: Tools and Techniques	✓	✓	✓	✓	Research Study

Description: Comparative analysis of DevSecOps research papers across multiple dimensions including code protection, secure deployment, AI-based solutions, and DevSecOps integration. The table categorizes 10 research papers by their type (Survey/Review, Research Study, Framework Proposal, Systematic Review, and Case Study) and evaluates their coverage of key DevSecOps implementation aspects. Green checkmarks indicate coverage of the respective dimension, while red crosses indicate absence of coverage in the reviewed literature.

2.4.2 Detailed Research Gap Analysis

The systematic analysis of current literature reveals several critical findings and research gaps that directly inform the scope and objectives of this project:

AI Integration Limitations

Our analysis indicates that only **30% (3 out of 10)** of reviewed studies incorporate AI-based solutions in their DevSecOps approaches. The studies that do integrate AI focus primarily on:

- **Quantitative metrics analysis**: Using AI for performance measurement and optimization
- **Automated testing enhancement**: Leveraging AI for test case generation and execution
- **Monitoring and alerting**: AI-driven anomaly detection and incident response

However, **none of the reviewed studies address proactive AI-driven generation of secure Infrastructure as Code**, representing a significant gap in current research and practice.

Code Protection vs. Secure Deployment Focus

The analysis reveals an interesting distribution pattern:

- **Code Protection**: 80% (8 out of 10) studies address code-level security
- **Secure Deployment**: 100% (10 out of 10) studies address deployment security
- **Gap**: Limited integration between code-level and deployment-level security automation

This indicates that while both areas receive attention individually, **integrated approaches that bridge code security and deployment security through AI-driven automation remain underexplored**.

Research Methodology Distribution

The research landscape shows diverse methodological approaches:

- **Empirical Research Studies**: 30% - Focus on quantitative analysis and practical implementations
- **Framework Proposals**: 20% - Theoretical models and architectural recommendations
- **Surveys and Reviews**: 30% - Literature analysis and practice surveys
- **Systematic Reviews**: 10% - 1 paper
- **Case Studies**: 10% - Real-world implementation experiences

This distribution reveals a **lack of empirical research in AI-driven DevSecOps automation**, with most AI-related work remaining theoretical or limited in scope.

Infrastructure as Code Security Gap

A critical observation from our analysis is the **absence of research focusing specifically on AI-driven Infrastructure as Code security generation and compliance automation**. Current research addresses:

- **IaC validation and scanning**: Detecting misconfigurations post-creation
- **Policy enforcement**: Implementing governance rules for IaC deployment
- **Compliance checking**: Validating against security frameworks

However, **proactive generation of secure, compliant IaC using generative AI remains unexplored**, representing the primary innovation opportunity that this project addresses.

Multi-Cloud and Multi-Platform Considerations

Our review indicates limited research addressing **unified security approaches across diverse deployment models** (VMs, containers, serverless, Kubernetes). Most studies focus on single-platform solutions, creating a gap in:

- **Cross-platform security consistency**: Maintaining security standards across different deployment targets
- **Multi-cloud compliance**: Ensuring regulatory compliance across various cloud providers
- **Unified policy management**: Centralized governance across heterogeneous environments

2.5 Problem Statement and Research Motivation

Building upon the foundations established in this chapter, we can now clearly articulate the specific problem this project addresses. Current DevSecOps practices, while advancing security integration throughout the SDLC, face several critical limitations:

2.5.1 Current Limitations

Reactive Security Approach: Existing tools primarily focus on detecting security misconfigurations and compliance violations after they occur, requiring manual intervention for remediation.

Limited AI Integration: Despite the proven capabilities of generative AI in software development, minimal integration exists for proactive security and compliance automation in IaC.

Manual Remediation Overhead: Security teams spend significant time manually correcting identified issues, creating bottlenecks in deployment pipelines.

Compliance Complexity: Ensuring adherence to multiple security frameworks (NIST, SOC 2, ISO 27001) requires extensive manual validation and configuration management.

Multi-Cloud Challenges: Managing security consistency across diverse deployment models (VMs, IaaS, PaaS, Kubernetes) presents significant complexity.

2.5.2 Project Innovation

This project directly addresses these limitations by developing an AI-driven solution that:

- **Proactively generates** secure and compliant Infrastructure as Code configurations
- **Automatically remediates** identified security misconfigurations and compliance violations
- **Integrates seamlessly** into existing DevSecOps workflows and CI/CD pipelines

- **Supports multiple deployment models** including virtual machines, container orchestration, and cloud-native services
- **Ensures compliance** with established security frameworks through automated validation and correction

By leveraging the power of generative AI for proactive security integration rather than reactive detection, this project represents a fundamental shift toward truly automated, intelligent DevSecOps practices that can scale with modern software development demands.

2.6 Chapter Summary

This chapter has established the essential foundations for understanding the context and motivation behind this project. We examined the evolution from monolithic to microservices architectures and its implications for security practices, explored the transformative potential of generative AI in software development, and analyzed the principles and practices of DevSecOps methodologies.

Through systematic literature review, we identified critical gaps in current research, particularly the limited integration of AI-driven solutions for proactive security automation in Infrastructure as Code management. These findings directly inform our project's focus on developing an intelligent, automated approach to DevSecOps that shifts from reactive detection to proactive generation of secure, compliant infrastructure configurations.

The next chapter will detail our methodology for developing and implementing this AI-driven DevSecOps solution, building upon the theoretical foundations established here.

Chapter 3

State of the Art: AI-Driven Enhancements in IaC Security and Compliance

The integration of Artificial Intelligence (AI), particularly Large Language Models (LLMs), into Infrastructure as Code (IaC) practices represents a significant shift towards more automated, secure, and compliant cloud infrastructure management. While the potential is vast, practical application is still evolving. This chapter reviews several notable projects and research initiatives that explore the use of AI and LLMs specifically for enhancing IaC security, compliance, and correctness, moving beyond generic tool discussions to focus on concrete implementations and evaluations.

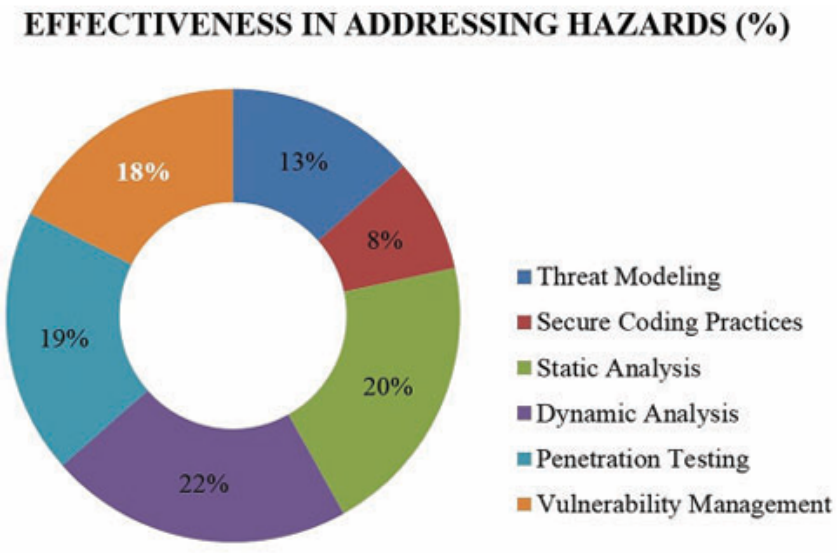


Figure 3.1: Effectiveness of Various Security Practices in Addressing Hazards (Adapted from [7])

3.1 Automated Remediation of IaC Misconfigurations

A critical challenge in IaC adoption is the prevalence of misconfigurations that can lead to security vulnerabilities. Traditional approaches rely on scanning tools to detect these

issues, followed by manual remediation by developers. Recent research projects aim to automate this repair process using LLMs.

One significant project in this area was presented at the 2024 IEEE 6th International Conference on Trust, Privacy and Security in Intelligent Systems, and Applications (TPS-ISA) by Cheh and Chen [4]. Their work proposes an LLM-based approach specifically designed to fix misconfigurations in IaC code identified by scanning tools. The methodology involves feeding the LLM (specifically GPT-4 in their tests) with the vulnerable IaC code, detailed information about the detected misconfigurations, and additional context provided through a human-in-the-loop mechanism. The LLM is then prompted to generate the corrected IaC code.

The researchers tested their approach on several vulnerable IaC repositories. Their findings indicated that the LLM-suggested fixes could reduce up to 84.7% of the misconfiguration alarms generated by the scanning tools. They also found that a two-pass approach, potentially allowing for refinement or verification, significantly improved performance compared to a single-pass generation. However, the study also highlighted a key challenge: LLM hallucination. Manual verification revealed that only 79.6% of the suggested fixes actually resolved the underlying security issue or misconfiguration correctly. The remaining 20.4% were classified as "hallucinated fixes" – code that might pass the scanner checks but failed other syntax or schema validations, or simply did not address the root cause of the security problem. This project underscores both the potential of LLMs for automated IaC repair and the critical need for robust validation mechanisms to counteract hallucination and ensure the correctness of generated fixes.

3.2 Evaluating LLM Capabilities for IaC Generation and Compliance

Beyond fixing existing code, another line of inquiry focuses on the ability of LLMs to generate secure and compliant IaC configurations from scratch based on natural language descriptions. Evaluating the performance of different LLMs for this task is crucial for understanding their practical applicability.

An evaluation project documented by Lu Mao on the GFT Engineering blog [8] provides valuable insights. This project utilized the established IaC-Eval benchmark, a framework designed to test LLM capabilities in generating Terraform configurations for AWS services based on natural language prompts and validating them against specific infrastructure intents (defined using OPA Rego policies). The evaluation focused on newer, powerful LLMs not covered in the original IaC-Eval publication, including models like Google Gemini variants, ChatGPT-4o, Claude 3.5 Sonnet, Llama 3.1, and notably, DeepSeek-V3.

The study found that DeepSeek-V3 emerged as a leading performer on the IaC-Eval benchmark for Terraform generation, offering comparable or better results than models like GPT-4 but at a significantly lower cost (estimated at 30x cheaper). The evaluation also extended to smaller models (e.g., Llama 3.1 8B, CodeGemma 7B) suitable for local deployment. This aspect is particularly relevant for organizations prioritizing security, auditability, and reproducibility, as it allows IaC generation tasks to be performed within controlled environments without relying on external APIs.

Furthermore, this work highlights a key business value proposition: using LLMs to embed compliance rules directly into IaC templates during generation. By translating natural language requirements that include security and governance standards into com-

pliant IaC scripts, LLMs can help ensure adherence from the outset, reducing the risk of non-compliance and improving audit readiness. The project's rigorous two-phase evaluation pipeline (checking Terraform plan validity and OPA policy compliance) provides a robust method for assessing both functional correctness and intent fulfillment without actual cloud deployment.

3.3 Fine-Tuning LLMs for Domain-Specific Requirements

While general-purpose LLMs show promise, their outputs often require significant review and modification to meet production-grade standards, particularly concerning specific organizational coding practices, security policies, and compliance mandates. Base models may generate syntactically correct IaC but fail to incorporate necessary security controls or adhere to internal best practices.

A project initiative discussed by Sandip Dey from Persistent Systems [6] explores the use of fine-tuning to address these limitations. The core idea is that base LLMs, even those trained on code, often lack the specificity needed for enterprise IaC development. Fine-tuning adapts a pre-trained base model to a specific downstream task by training it further on a curated dataset reflecting the desired output style and constraints.

The proposed methodology involves preparing an extensive dataset (suggesting 18,000-20,000 examples) of instruction-response pairs tailored to the specific IaC requirements. For instance, if an organization requires all generated Terraform resources to include specific comments (e.g., identifying the generating tool) and tags (e.g., 'created$_{b}y$'), $thedatasetwouldcontainnume$

The project advocates for Parameter Efficient Fine-Tuning (PEFT) techniques like Low-Rank Adaptation (LoRa). These methods keep the large base model's weights frozen and only train a small number of additional parameters ("adapters"). This significantly reduces the computational resources and time required for fine-tuning compared to retraining the entire model.

The outcome of such a fine-tuning project is an LLM specialized in generating IaC that not only fulfills the functional requirements described in prompts but also automatically incorporates the organization's specific coding standards, documentation practices, security configurations, and compliance checks. This approach aims to reduce hallucinations and produce more reliable, production-ready IaC, thereby accelerating development while maintaining high quality and security standards.

3.4 Comparative Synthesis and Research Directions

These projects collectively illustrate the active exploration of LLMs within the IaC domain, moving from theoretical potential to practical application and evaluation. Key themes emerge:

- **Automation Potential:** LLMs demonstrate capability in both generating IaC from natural language and repairing existing misconfigurations, reducing manual effort.
- **Accuracy Challenges:** Hallucination remains a significant issue, necessitating robust validation and often human oversight. Base models may not meet specific enterprise requirements.

- **Evaluation Frameworks:** Benchmarks like IaC-Eval are crucial for objectively comparing model performance in realistic scenarios.
- **Specialization via Fine-Tuning:** Tailoring LLMs through fine-tuning appears essential for generating IaC that adheres to specific organizational standards for security, compliance, and coding practices.
- **Cost and Deployment Models:** The emergence of powerful yet cost-effective models (like DeepSeekV3) and smaller, locally deployable models broadens the accessibility and applicability of these technologies.

Future work in this area will likely focus on improving the reliability of LLM outputs (reducing hallucination), developing more sophisticated validation techniques, integrating LLM capabilities seamlessly into developer workflows and CI/CD pipelines, and further refining fine-tuning methodologies to capture complex security and compliance logic more effectively. The projects reviewed here provide a foundation, showcasing both the achievements and the remaining challenges in leveraging AI for secure and compliant Infrastructure as Code.

Chapter 4

Proposed Solution Framework

4.1 Solution Objectives

Building upon the identified gaps in the state of the art, this project sets forth ambitious objectives to deliver a truly proactive and automated solution for Infrastructure as Code (IaC) security and compliance, leveraging the power of Generative AI.

The primary objectives are:

1. **Create a Fully Automated LLM-Powered Pipeline for IaC Compliance:** Develop and implement a CI/CD pipeline integrating a Large Language Model (LLM) engine capable of analyzing IaC scripts (e.g., Terraform), identifying vulnerabilities and non-compliance against specified frameworks (notably NIST), and, most importantly, **automatically generating corrected, secure, and compliant IaC versions** [2, 1].

2. **Achieve Seamless Integration into DevSecOps Workflows:** Ensure the solution integrates smoothly with standard DevSecOps tools and practices, including version control systems (e.g., GitHub via webhooks/actions), CI/CD platforms (e.g., GitHub Actions, GitLab CI, AWS CodePipeline), and potentially IDEs, minimizing friction for development teams [2, 1].

3. **Demonstrate Efficacy on a Representative Web Application (OWASP Juice Shop):** Validate the solution by deploying the OWASP Juice Shop application using IaC generated and secured by the AI engine. This demonstration will showcase the tangible reduction in security risks and improved compliance posture compared to manual or unvalidated IaC approaches [2].

4. **Ensure Compatibility Across Diverse Cloud Deployment Models:** Design the solution to generate and validate IaC suitable for various cloud environments, including virtual machines (IaaS like **AWS EC2**), Platform-as-a-Service (PaaS), and container orchestration systems (like Kubernetes, specifically **Amazon EKS**), proving its versatility [2].

5. **Provide Actionable, Real-Time Feedback:** Deliver clear explanations for detected vulnerabilities and generated corrections directly to developers, ideally within their existing workflows (IDE, pull request comments), fostering continuous learning and adoption of secure coding practices [1].

6. **Automate Compliance Reporting:** Leverage the continuous IaC analysis to automatically generate compliance reports mapped against target frameworks (e.g., NIST CSF functions: Identify, Protect, Detect, Respond, Recover), simplifying auditing and governance processes [1].

4.2 NIST CSF Implementation Guide

Successfully achieving these objectives will mark a significant advancement in automating IaC security, making DevSecOps practices more intelligent, efficient, and inherently secure.

CSF Function	Category	Best Practices for IaC in Cloud Environments
1. IDENTIFY	**Asset Management**	Maintain an inventory of IaC-defined cloud resources (e.g., VMs, containers, databases).
	Risk Management	Classify assets and define risk tolerance in cloud deployments using IaC.
	Governance	Define and enforce cloud security policies using policy-as-code (e.g., `OPA`, `Sentinel`).
2. PROTECT	**Access Control**	Define least-privilege IAM roles and policies directly in IaC templates.
	Data Security	Configure encryption for data in transit and at rest using IaC (e.g., `AWS KMS`, `Azure Key Vault`).
	Secure Configurations	Use automated tools to scan IaC for misconfigurations (e.g., `TFSec`, `Checkov`, `Terrascan`).
	Maintenance	Version-control all IaC code (e.g., `Git`), with code reviews and change control.
	Awareness	Include documentation and code comments in IaC templates for clarity and secure usage.
3. DETECT	**Anomalies & Events**	Enable logging (e.g., `CloudTrail`, `GCP Audit Logs`) via IaC to track resource and user activity.
	Security Monitoring	Integrate monitoring tools (e.g., `AWS GuardDuty`, `Azure Defender`) through IaC.
	Detection Process	Implement alerts for config drift or unauthorized changes in IaC-defined infrastructure.
4. RESPOND	**Response Planning**	Use IaC to define rollback/recovery mechanisms and incident response architectures.
	Analysis	Enable centralized log collection and SIEM integration via IaC for cloud forensics.
	Mitigation	Automate incident response using event-driven architecture (e.g., `Lambda`, `Azure Functions` triggered by alerts).
	Improvements	Post-incident, update IaC to fix vulnerabilities or misconfigurations discovered.
5. RECOVER	**Recovery Planning**	Use IaC to recreate environments from clean templates in disaster recovery scenarios.
	Improvements	Update IaC templates regularly based on lessons learned from incidents.
	Communications	Ensure recovery workflows and responsibilities are well-documented within IaC deployment plans.

Figure 4.1: NIST CSF Implementation Guide table

To achieve the defined objectives, we propose an integrated solution combining agile project management methodologies, standard DevSecOps tooling, and an innovative AI engine for secure and compliant Infrastructure as Code (IaC) analysis and generation.

4.3 Planning and Project Management

An agile approach ensures flexibility and continuous adaptation throughout the project lifecycle.

4.3.1 Agile Approach and Tooling (Trello)

Trello serves as the primary tool for task tracking, backlog management, and visualizing project progress. The board is structured with columns representing workflow stages (e.g., Backlog, To Do, In Progress, Review, Done). Features and tasks are detailed on cards, facilitating collaboration and iterative development through sprints [2].

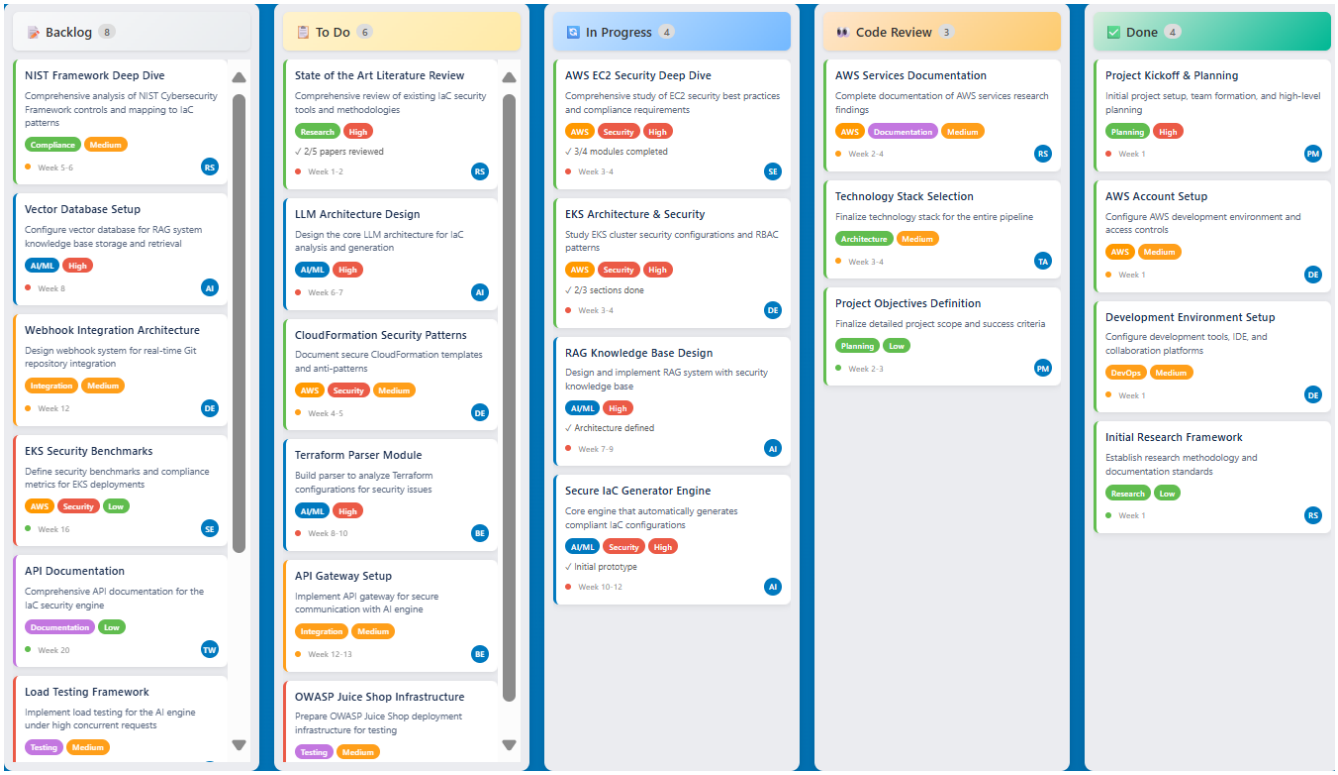


Figure 4.2: Illustrative Trello Board for Project Tracking

4.3.2 Project Timeline and Gantt Chart

An initial Gantt chart provides a high-level overview of the project timeline, key task dependencies, and major milestones, aiding in roadmap visualization and stakeholder communication.

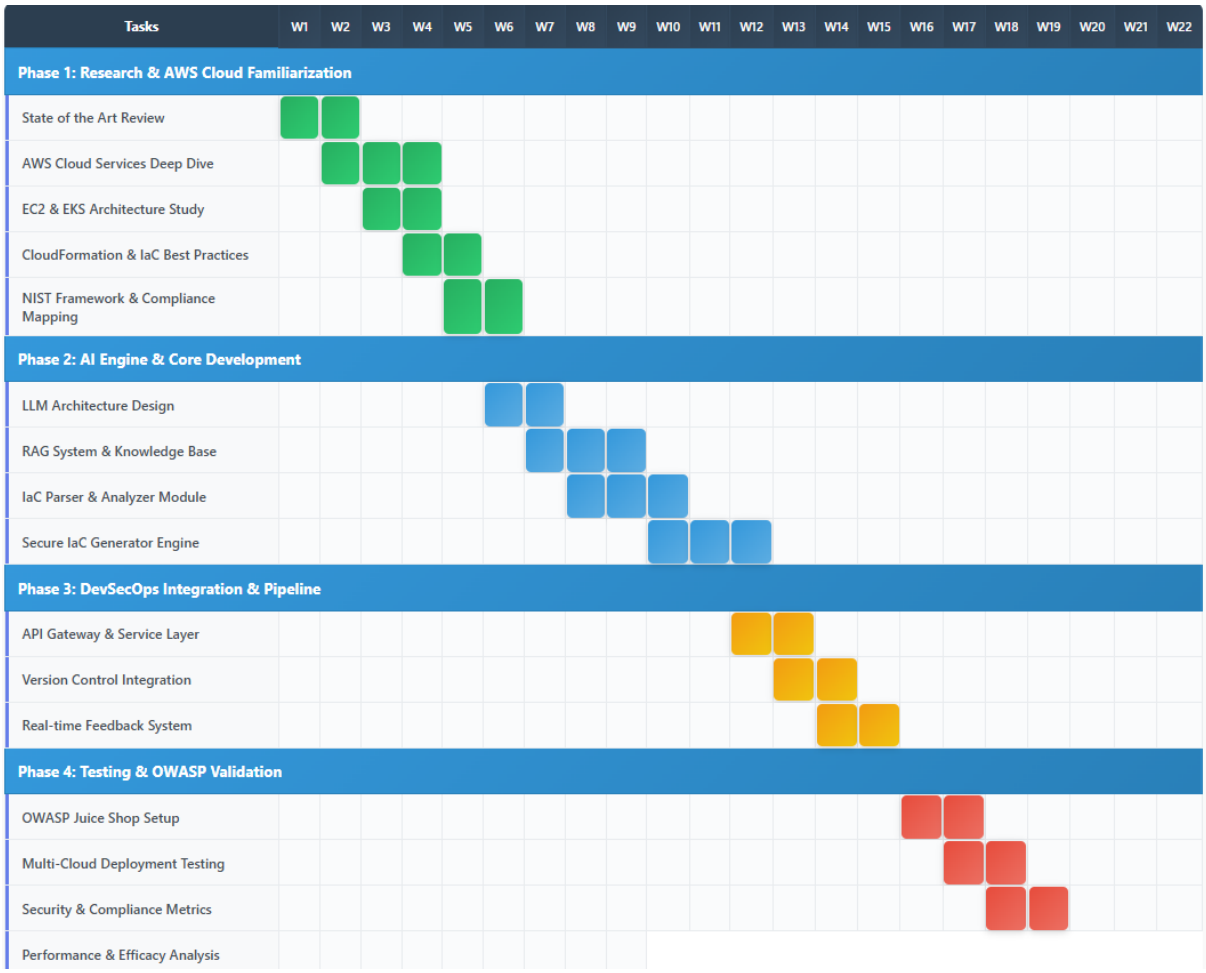


Figure 4.3: Project Gantt Chart Overview

4.4 Tools, Technologies and Platforms

The solution leverages a stack of modern tools common in Cloud-Native and DevSecOps environments. Effective integration is key to building an automated pipeline.

GitHub Actions: Primary CI/CD platform for orchestrating integration, testing, IaC analysis, and deployment pipelines.

Docker: For containerizing the target application (OWASP Juice Shop) and potentially components of the AI solution.

Kubernetes (Amazon EKS): Target orchestration platform for containerized application deployment, demonstrating handling of complex environments.

Amazon EC2: Alternative IaaS deployment target to showcase IaC versatility.

Terraform: The primary IaC language targeted for analysis and generation due to its popularity and declarative nature.

Trivy: Integrated for scanning Docker images and IaC files, complementing the AI engine with known vulnerability checks.

CodeQL: Used for Static Application Security Testing (SAST) of the application code (Juice Shop) itself.

LLMs (Large Language Models): The core of the IaC analysis and generation engine. Models like GPT, CodeBERT, or potentially a custom-tuned LLM will be explored [1].

AWS Services: Besides EKS and EC2, other services (S3, IAM, CloudWatch, etc.) will form the target infrastructure.

4.5 Target System Architecture and Deployment Strategy

The solution's effectiveness will be demonstrated by deploying the OWASP Juice Shop application onto different target architectures managed via the AI-validated IaC.

4.5.1 Secure Deployment Environment Analysis

Deployment Environment Class	Definition	Confidentiality (C)	Integrity (I)	Availability (A)	Measures to Implement	ISO 27017/27034 Measures
IaaS (Infrastructure as a Service)	Provides virtualized computing resources over the internet. Client manages OS, apps, and data.	Medium if misconfigured by client	Low if patches are delayed	Medium if scaling is mismanaged	Use proper network configuration, implement patch management policies, and configure autoscaling to handle workloads.	**ISO 27017:** Conduct cloud-specific risk assessments. **ISO 27034:** Ensure secure app deployment.
PaaS (Platform as a Service)	Provider handles infrastructure and runtime, client focuses on apps and data.	High but client app vulnerabilities	High provider-managed	High provider ensures availability	Secure applications through vulnerability scanning, use strong authentication methods, and encrypt sensitive data.	**ISO 27017:** Establish provider-client responsibility matrix. **ISO 27034:** Secure APIs and data flows.
Kubernetes on Metal Servers	Self-hosted Kubernetes running on physical hardware. Full client responsibility.	Low if no network segmentation	Medium if updates aren't applied	Low if backup isn't automated	Implement network segmentation, schedule regular updates, and ensure automated backups are in place.	**ISO 27017:** Ensure secure virtualization processes. **ISO 27034:** Implement secure container configurations.
Kubernetes-Cloud Based	Managed Kubernetes services hosted on the cloud. Shared security responsibility.	Medium if access control is weak	High managed by provider	Medium to High provider-managed but client scaling issues	Enforce role-based access control (RBAC), regularly review permissions, and monitor resource scaling.	**ISO 27017:** Monitor cloud service logs. **ISO 27034:** Ensure application lifecycle security.

Note: CIA assessment levels represent potential security risks when proper management practices are not implemented by the responsible party (Client or Provider). The recommended measures prioritize critical client responsibilities within each deployment model. Risk levels assume standard security practices are not adequately implemented.

Figure 4.4: Deployment Environment Security Analysis

Table (figure 5.3) presents a comprehensive security analysis of four primary deployment environments, evaluating each against the CIA triad (Confidentiality, Integrity, and Availability) while providing specific implementation measures and ISO compliance guidelines.

The analysis reveals distinct security profiles across deployment models. Infrastructure as a Service (IaaS) environments present medium-risk profiles that are heavily dependent on client configuration and maintenance practices. The conditional nature of these

risks—where confidentiality becomes compromised through misconfiguration and integrity suffers from delayed patching—highlights the critical importance of proactive client management.

Platform as a Service (PaaS) deployments demonstrate the highest overall security posture, with providers managing infrastructure-level security concerns. However, the persistence of application-layer vulnerabilities underscores the shared responsibility model, where clients must maintain secure coding practices despite the provider's robust infrastructure management.

Kubernetes deployments exhibit contrasting security characteristics based on their hosting model. Self-hosted Kubernetes on metal servers presents the highest risk profile, requiring comprehensive client oversight across all security domains. In contrast, cloud-managed Kubernetes services leverage provider expertise for control plane security while maintaining client responsibility for access control and workload management.

The implementation measures column provides actionable guidance tailored to each environment's risk profile, ranging from fundamental network configuration in IaaS to sophisticated role-based access control in managed Kubernetes environments. The ISO 27017 and ISO 27034 compliance measures establish standardized security frameworks, with ISO 27017 addressing cloud-specific security controls and ISO 27034 focusing on application security throughout the development lifecycle.

This comparative analysis enables informed decision-making by clearly delineating security responsibilities, risk levels, and compliance requirements across different deployment architectures. The conditional risk assessments particularly emphasize how proper implementation of recommended measures can significantly mitigate identified vulnerabilities.

4.5.2 Multi-Architecture Considerations

We plan to generate Terraform IaC to deploy Juice Shop in at least two distinct AWS configurations:

1. **On Amazon EKS:** A modern microservices-style deployment using Kubernetes. IaC will cover cluster creation, node groups, Kubernetes Deployments, Services, Ingress controllers, and associated security configurations (Network Policies, Security Groups).

2. **On Amazon EC2:** A more traditional VM-based deployment, potentially using an Application Load Balancer. IaC will manage EC2 instances, Security Groups, Load Balancers, and necessary software configurations.

This validates the AI's ability to generate appropriate IaC for different deployment paradigms.

4.5.3 Security and Compliance Challenges

Each architecture presents unique security and NIST compliance challenges (e.g., EKS: control plane security, RBAC, network policies; EC2: security group configuration, OS patching, instance hardening). The AI engine must address these specifics during IaC generation and validation to ensure compliance with relevant NIST controls (e.g., Identify, Protect, Detect functions of the CSF [1]).

4.6 Solution Workflow and Automation

The core of the proposed solution is a DevSecOps pipeline incorporating the AI engine for IaC validation and generation.

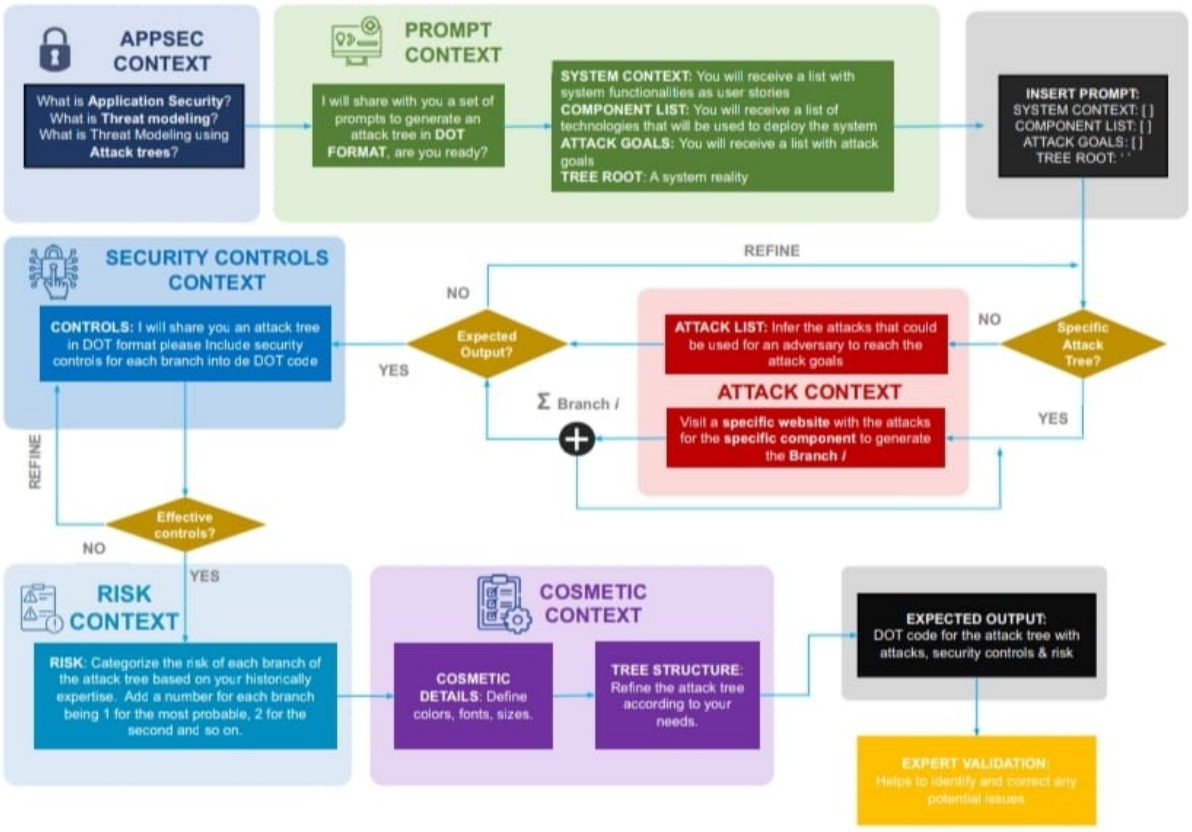


Figure 4.5: LLM-Based IaC Security Solution Workflow Diagram

The typical workflow involves:

1. **Application Security Context:** Define what constitutes application security, threat modeling, and attack trees for the target system.
2. **Prompt Context:** Generate system context prompts using component lists, attack goals, and tree root definitions to create structured attack scenarios.
3. **Security Controls Context:** Identify and document security controls for each branch in the attack tree, including specific controls for each component.
4. **Attack Tree Generation:** Create attack trees by defining attack lists that could be used by adversaries to reach specific attack goals, with branch-specific context.
5. **Risk Assessment:** Categorize the risk level of each attack branch based on business impact and expertise requirements, assigning probability scores.
6. **Cosmetic Context & Tree Structure:** Refine the visual presentation of attack trees according to specific formatting requirements and structural needs.
7. **Expected Output:** Generate DOT code representation of the attack tree complete with security controls and risk assessments for further analysis and visualization.

4.7 Core Algorithm and Logical Flow

4.7.1 Workflow Overview

The core logic of the AI engine processing an IaC file can be visualized as follows:

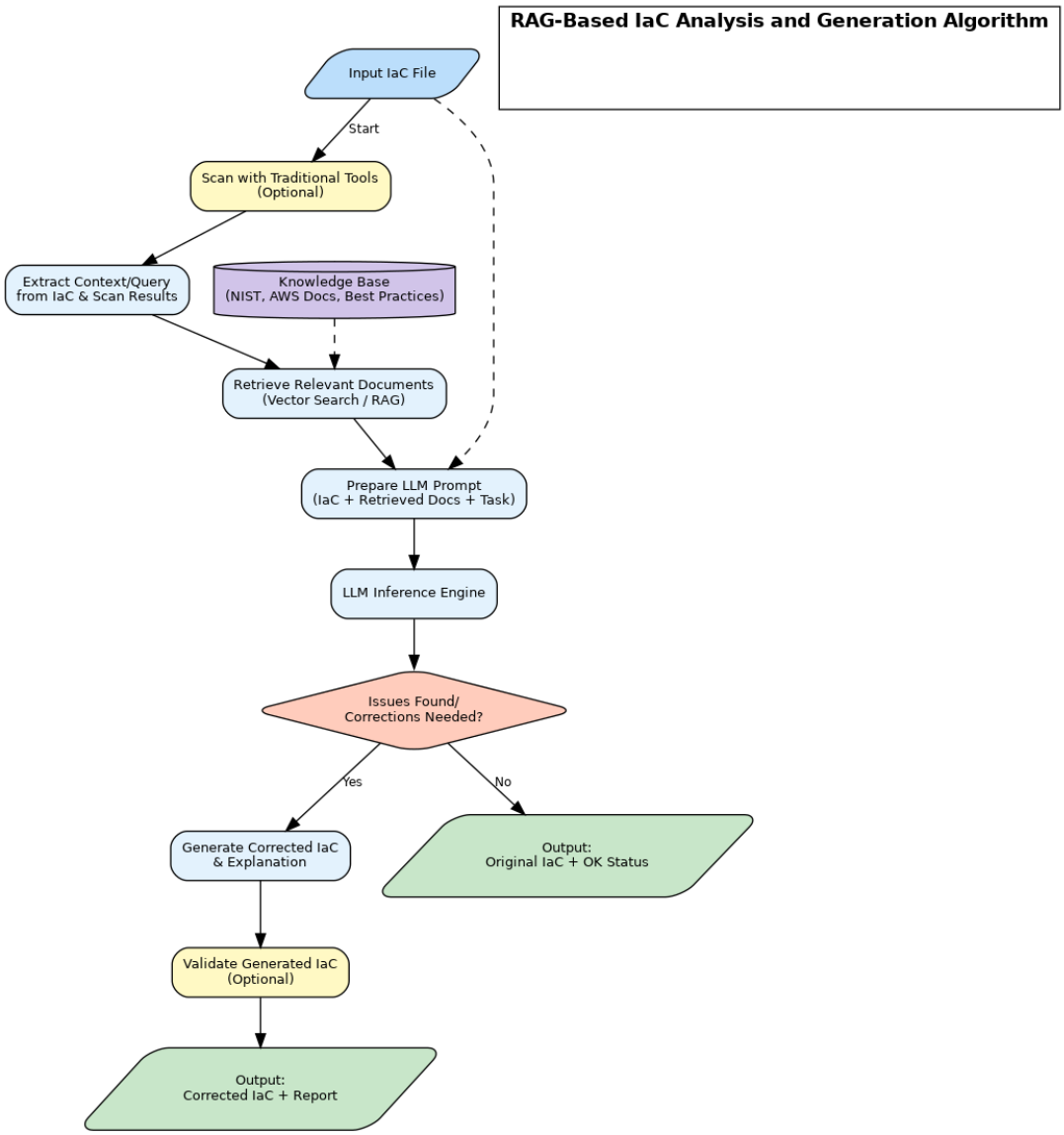


Figure 4.6: AI-Powered IaC Analysis and Generation Algorithm

This diagram illustrates the flow: optional pre-scan, LLM prompt preparation (IaC + Policies), LLM inference, issue detection decision, corrected code generation (if needed), optional validation, and final output (original/corrected code + report).

This demonstrates the transformation from a dangerously open configuration to one adhering to the principle of least privilege, guided by security best practices and potential NIST requirements.

4.7.2 Stack Flexibility and Scalability

The generated IaC stack must be adaptable to different AWS deployment modes (EC2, EKS). The AI engine needs to adjust resources and configurations based on the target architecture while maintaining security and compliance.

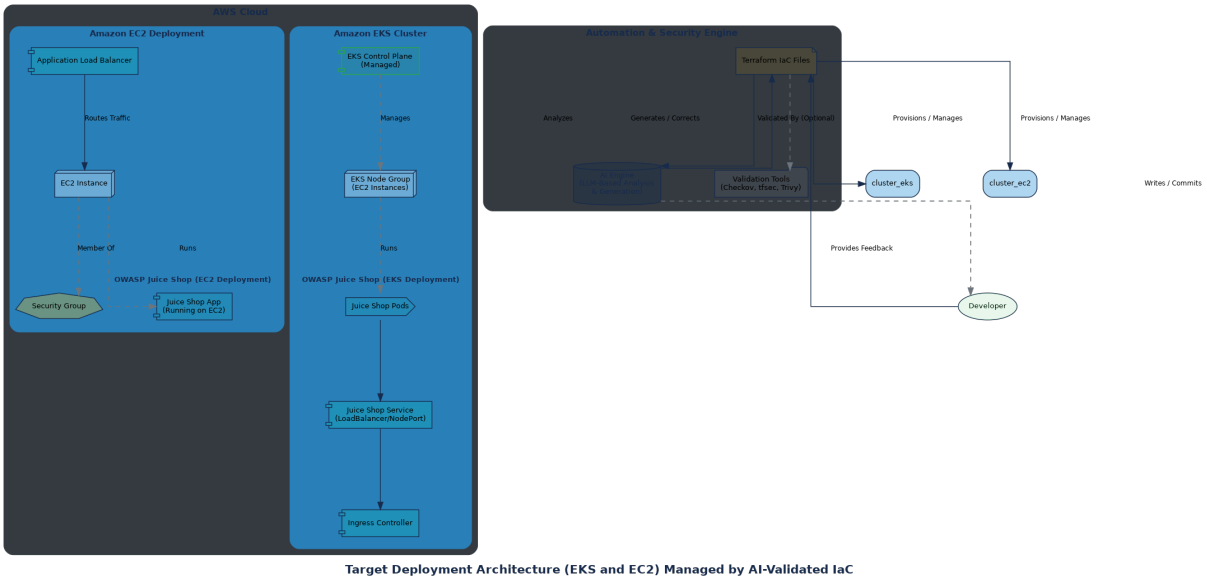


Figure 4.7: Target Deployment Architecture Diagram (EKS and EC2)

Chapter 5

Implementation and Evaluation

Evaluating the proposed AI-driven IaC security solution is critical to demonstrate its effectiveness, efficiency, and added value compared to existing approaches. We plan a multi-faceted evaluation strategy combining qualitative comparisons, quantitative metrics, and evidence from standard analysis tools.

5.1 Before/After Comparison of IaC File

A core evaluation method involves direct comparison of IaC files before and after processing by the AI engine, as illustrated conceptually in Section 5.1 and 5.2. This highlights concrete improvements in security posture and compliance.

We will document several transformation examples applied to various resource types (e.g., security groups, storage configurations, IAM roles, Kubernetes settings) derived from deploying the OWASP Juice Shop application. For each example, the analysis will cover:

- The specific vulnerability or non-compliance in the initial file.
- The relevant security best practice or NIST control being violated.
- The correction implemented by the AI engine in the generated file.
- Any explanation provided by the system justifying the change.

This qualitative analysis assesses the relevance and accuracy of the AI-generated corrections.

5.2 Infrastructure Stack Example

To illustrate the transformation, consider an example of a vulnerable AWS-Cloudformation input file and the AI-generated corrected output.

5.2.1 Input (Vulnerable Template)

```
1 AWSTemplateFormatVersion: '2010-09-09'
2 Description: CloudFormation template for deploying OWASP Juice Shop in
     EKS with S3, RDS, and VPC.
3
4 Parameters:
5   ClusterName:
```

```
    Type: String
    Default: JuiceShopEKSCluster

Resources:

  # VPC
  JuiceVPC:
    Type: AWS::EC2::VPC
    Properties:
      CidrBlock: 10.0.0.0/16
      EnableDnsSupport: true
      EnableDnsHostnames: true
      Tags:
        - Key: Name
          Value: JuiceVPC

  # Subnet (Public)
  JuiceSubnet:
    Type: AWS::EC2::Subnet
    Properties:
      VpcId: !Ref JuiceVPC
      CidrBlock: 10.0.1.0/24
      AvailabilityZone: !Select [0, !GetAZs ''']
      MapPublicIpOnLaunch: true
      Tags:
        - Key: Name
          Value: JuiceSubnet

  # Internet Gateway
  JuiceInternetGateway:
    Type: AWS::EC2::InternetGateway

  JuiceVPCGatewayAttachment:
    Type: AWS::EC2::VPCGatewayAttachment
    Properties:
      VpcId: !Ref JuiceVPC
      InternetGatewayId: !Ref JuiceInternetGateway

  # Route Table
  JuiceRouteTable:
    Type: AWS::EC2::RouteTable
    Properties:
      VpcId: !Ref JuiceVPC

  JuiceRoute:
    Type: AWS::EC2::Route
    DependsOn: JuiceVPCGatewayAttachment
    Properties:
      RouteTableId: !Ref JuiceRouteTable
      DestinationCidrBlock: 0.0.0.0/0
      GatewayId: !Ref JuiceInternetGateway

  JuiceSubnetRouteTableAssociation:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      SubnetId: !Ref JuiceSubnet
      RouteTableId: !Ref JuiceRouteTable

```

```
# EKS Cluster Role
EKSClusterRole:
  Type: AWS::IAM::Role
  Properties:
    AssumeRolePolicyDocument:
      Version: '2012-10-17'
      Statement:
        - Effect: Allow
          Principal:
            Service: eks.amazonaws.com
          Action: sts:AssumeRole
    ManagedPolicyArns:
      - arn:aws:iam::aws:policy/AmazonEKSClusterPolicy

# EKS Cluster
EKSCluster:
  Type: AWS::EKS::Cluster
  Properties:
    Name: !Ref ClusterName
    RoleArn: !GetAtt EKSClusterRole.Arn
    ResourcesVpcConfig:
      SubnetIds:
        - !Ref JuiceSubnet
      EndpointPublicAccess: true

# S3 Bucket
JuiceS3Bucket:
  Type: AWS::S3::Bucket
  Properties:
    BucketEncryption:
      ServerSideEncryptionConfiguration:
        - ServerSideEncryptionByDefault:
            SSEAlgorithm: AES256
    VersioningConfiguration:
      Status: Enabled
    AccessControl: Private

# RDS Instance (MySQL)
JuiceRDS:
  Type: AWS::RDS::DBInstance
  Properties:
    DBInstanceIdentifier: juice-db
    Engine: mysql
    EngineVersion: 8.0
    MasterUsername: admin
    MasterUserPassword: Password1234!-****souadprivateen
    DBInstanceClass: db.t3.micro
    AllocatedStorage: 20
    PubliclyAccessible: false
    StorageEncrypted: true
    VPCSecurityGroups:
      - !GetAtt JuiceRDSSecurityGroup.GroupId
    DBSubnetGroupName: !Ref JuiceRDSSubnetGroup

JuiceRDSSecurityGroup:
  Type: AWS::EC2::SecurityGroup
  Properties:
    GroupDescription: Allow access to RDS from EKS
```

```
      VpcId: !Ref JuiceVPC
      SecurityGroupIngress:
        - IpProtocol: tcp
          FromPort: 3306
          ToPort: 3306
          CidrIp: 10.0.0.0/16

  JuiceRDSSubnetGroup:
    Type: AWS::RDS::DBSubnetGroup
    Properties:
      DBSubnetGroupDescription: RDS subnet group for Juice Shop
      SubnetIds:
        - !Ref JuiceSubnet

Outputs:
  ClusterName:
    Description: EKS Cluster Name
    Value: !Ref EKSCluster

  BucketName:
    Description: S3 Bucket for Juice Shop
    Value: !Ref JuiceS3Bucket

  RDSInstance:
    Description: RDS Instance endpoint
    Value: !GetAtt JuiceRDS.Endpoint.Address
```

Listing 5.1: IaC file Input:

5.2.2 Infrusture visualization (Bofore)

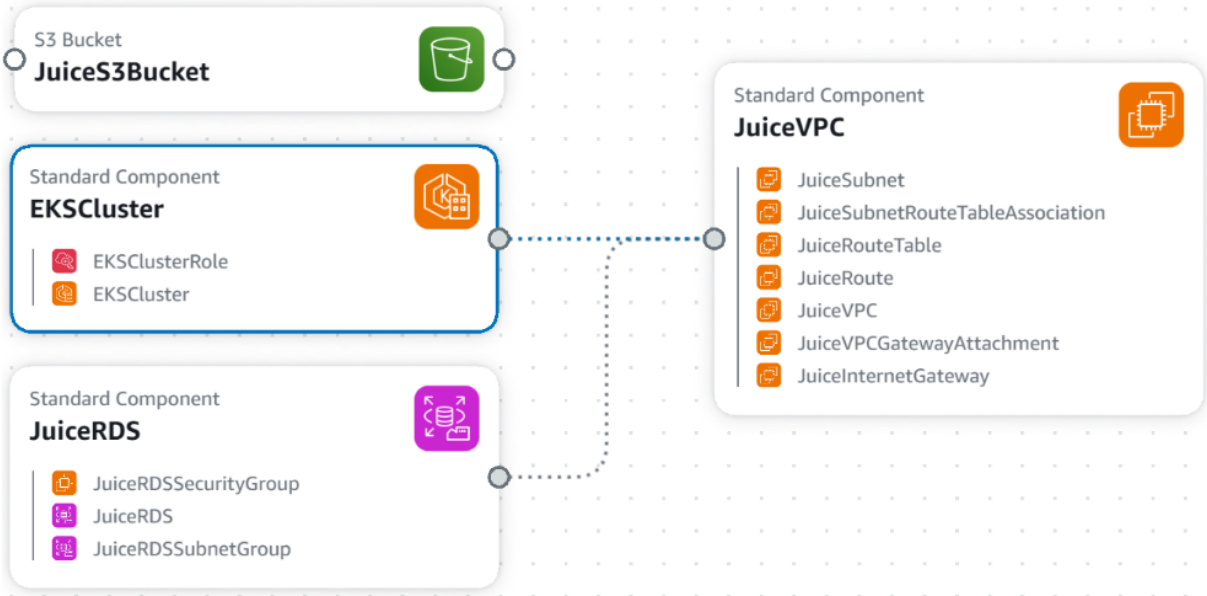


Figure 5.1: Before compliance

5.2.3 Output (Remediated and Compliant Template)

```
AWSTemplateFormatVersion: '2010-09-09'
Description: CloudFormation template for deploying OWASP Juice Shop in EKS with S3, RDS, and VPC.

Parameters:
  ClusterName:
    Type: String
    Default: JuiceShopEKSCluster
  # SECURE: Ajout de param tres pour les zones de disponibilit  pour la configuration Multi-AZ de RDS
  AvailabilityZone1: # SECURE
    Type: AWS::EC2::AvailabilityZone::Name # SECURE
    Description: First Availability Zone for the VPC and Subnets # SECURE
  AvailabilityZone2: # SECURE
    Type: AWS::EC2::AvailabilityZone::Name # SECURE
    Description: Second Availability Zone for the VPC and Subnets (for Multi-AZ) # SECURE

Resources:

  # VPC
  JuiceVPC:
    Type: AWS::EC2::VPC
    Properties:
      CidrBlock: 10.0.0.0/16
      EnableDnsSupport: true
      EnableDnsHostnames: true
      Tags:
        - Key: Name
          Value: JuiceVPC

  # Subnet 1 (Public)
  JuiceSubnet1: # SECURE: Renomm  pour supporter Multi-AZ
    Type: AWS::EC2::Subnet
    Properties:
      VpcId: !Ref JuiceVPC
      CidrBlock: 10.0.1.0/24
      AvailabilityZone: !Ref AvailabilityZone1 # SECURE: Utilisation du param tre AZ
      MapPublicIpOnLaunch: true
      Tags:
        - Key: Name
          Value: JuiceSubnet1 # SECURE

  # SECURE: Ajout d'un deuxi me Subnet pour la configuration Multi-AZ de RDS
  JuiceSubnet2: # SECURE
    Type: AWS::EC2::Subnet # SECURE
    Properties: # SECURE
      VpcId: !Ref JuiceVPC # SECURE
      CidrBlock: 10.0.2.0/24 # SECURE: Nouveau CIDR pour le deuxi me subnet
      AvailabilityZone: !Ref AvailabilityZone2 # SECURE: Utilisation du param tre AZ
      MapPublicIpOnLaunch: true # SECURE: Rendre public si n cessaire pour l'acc s EKS, sinon le rendre priv
      Tags: # SECURE
```

```
50          - Key: Name # SECURE
51            Value: JuiceSubnet2 # SECURE
52
53    # Internet Gateway
54    JuiceInternetGateway:
55      Type: AWS::EC2::InternetGateway
56
57    JuiceVPCGatewayAttachment:
58      Type: AWS::EC2::VPCGatewayAttachment
59      Properties:
60        VpcId: !Ref JuiceVPC
61        InternetGatewayId: !Ref JuiceInternetGateway
62
63    # Route Table
64    JuiceRouteTable:
65      Type: AWS::EC2::RouteTable
66      Properties:
67        VpcId: !Ref JuiceVPC
68
69    JuiceRoute:
70      Type: AWS::EC2::Route
71      DependsOn: JuiceVPCGatewayAttachment
72      Properties:
73        RouteTableId: !Ref JuiceRouteTable
74        DestinationCidrBlock: 0.0.0.0/0
75        GatewayId: !Ref JuiceInternetGateway
76
77    JuiceSubnetRouteTableAssociation1: # SECURE: Association pour le
       Subnet 1
78      Type: AWS::EC2::SubnetRouteTableAssociation
79      Properties:
80        SubnetId: !Ref JuiceSubnet1 # SECURE
81        RouteTableId: !Ref JuiceRouteTable
82
83    # SECURE: Association pour le Subnet 2
84    JuiceSubnetRouteTableAssociation2: # SECURE
85      Type: AWS::EC2::SubnetRouteTableAssociation # SECURE
86      Properties: # SECURE
87        SubnetId: !Ref JuiceSubnet2 # SECURE
88        RouteTableId: !Ref JuiceRouteTable # SECURE
89
90    # SECURE: Ajout d'une cl  KMS pour le chiffrement des secrets EKS
91    EKSSecretsKMSKey: # SECURE
92      Type: AWS::KMS::Key # SECURE
93      Properties: # SECURE
94        Description: KMS key for EKS secrets encryption # SECURE
95        Enabled: true # SECURE
96        EnableKeyRotation: true # SECURE: Bonne pratique de s curit
97        KeyPolicy: # SECURE: Politique pour permettre    EKS d'utiliser la
       cl
98          Version: '2012-10-17' # SECURE
99          Id: key-default-1 # SECURE
100         Statement: # SECURE
101           - Sid: Allow administration of the key # SECURE
102             Effect: Allow # SECURE
103             Principal: # SECURE
104               AWS: !Sub "arn:aws:iam::${AWS::AccountId}:root" # SECURE:
       Permet au compte root d'administrer la cl
```

```
105             Action: # SECURE
106               - kms:* # SECURE
107             Resource: '*' # SECURE
108           - Sid: Allow EKS service principal to use the key # SECURE
109             Effect: Allow # SECURE
110             Principal: # SECURE
111               Service: # SECURE
112                 - eks.amazonaws.com # SECURE
113             Action: # SECURE
114               - kms:Encrypt # SECURE
115               - kms:Decrypt # SECURE
116               - kms:GenerateDataKey # SECURE
117               - kms:GenerateDataKeyWithoutPlaintext # SECURE
118               - kms:ReEncryptFrom # SECURE
119               - kms:ReEncryptTo # SECURE
120               - kms:CreateGrant # SECURE
121               - kms:ListGrants # SECURE
122               - kms:DescribeKey # SECURE
123             Resource: '*' # SECURE
124
125   # SECURE: Ajout d'un alias pour la cl  KMS EKS
126   EKSSecretsKMSKeyAlias: # SECURE
127     Type: AWS::KMS::Alias # SECURE
128     Properties: # SECURE
129       AliasName: alias/eks-secrets-key # SECURE
130       TargetKeyId: !Ref EKSSecretsKMSKey # SECURE
131
132   # EKS Cluster Role
133   EKSClusterRole:
134     Type: AWS::IAM::Role
135     Properties:
136       AssumeRolePolicyDocument:
137         Version: '2012-10-17'
138         Statement:
139           - Effect: Allow
140             Principal:
141               Service: eks.amazonaws.com
142             Action: sts:AssumeRole
143       ManagedPolicyArns:
144         - arn:aws:iam::aws:policy/AmazonEKSClusterPolicy
145
146   # EKS Cluster
147   EKSCluster:
148     Type: AWS::EKS::Cluster
149     Properties:
150       Name: !Ref ClusterName
151       RoleArn: !GetAtt EKSClusterRole.Arn
152       ResourcesVpcConfig:
153         SubnetIds:
154           - !Ref JuiceSubnet1 # SECURE: Utilisation du Subnet 1
155           - !Ref JuiceSubnet2 # SECURE: Ajout du Subnet 2
156         EndpointPublicAccess: true
157       # SECURE: Activation du chiffrement des secrets EKS avec la cl
    KMS
158       EncryptionConfig: # SECURE
159         - Resources: # SECURE
160             - secrets # SECURE
161           Provider: # SECURE
```

```
162             KeyArn: !GetAtt EKSSecretsKMSKey.Arn # SECURE
163
164   # SECURE: Ajout d'un S3 Bucket pour la journalisation des acc s
165   JuiceS3LogsBucket: # SECURE
166     Type: AWS::S3::Bucket # SECURE
167     Properties: # SECURE
168       BucketEncryption: # SECURE: Chiffrement pour le bucket de logs
169         ServerSideEncryptionConfiguration: # SECURE
170           - ServerSideEncryptionByDefault: # SECURE
171               SSEAlgorithm: AES256 # SECURE
172       VersioningConfiguration: # SECURE: Versioning pour le bucket de
      logs
173         Status: Enabled # SECURE
174       AccessControl: LogDeliveryWrite # SECURE: ACL sp cifique pour la
      livraison de logs
175       PublicAccessBlockConfiguration: # SECURE: Blocage de l'acc s
      public pour le bucket de logs
176         BlockPublicAcls: true # SECURE
177         IgnorePublicAcls: true # SECURE
178         BlockPublicPolicy: true # SECURE
179         RestrictPublicBuckets: true # SECURE
180       # SECURE: D sactiver l'acc s public direct au bucket de logs
181       OwnershipControls: # SECURE
182         Rules: # SECURE
183           - ObjectOwnership: BucketOwnerPreferred # SECURE
184       # SECURE: Ajouter une politique de bucket pour la livraison de
      logs
185       BucketPolicy: # SECURE
186         Version: '2012-10-17' # SECURE
187         Statement: # SECURE
188           - Sid: AllowS3AccessLogsWrite # SECURE
189             Effect: Allow # SECURE
190             Principal: # SECURE
191               Service: logging.s3.amazonaws.com # SECURE
192             Action: s3:PutObject # SECURE
193             Resource: !Sub "arn:aws:s3:::${JuiceS3LogsBucket}/*" #
      SECURE
194             Condition: # SECURE
195               ArnLike: # SECURE
196                 aws:SourceArn: !GetAtt JuiceS3Bucket.Arn # SECURE
197               StringEquals: # SECURE
198                 aws:SourceAccount: !Ref AWS::AccountId # SECURE
199
200   # S3 Bucket principal
201   JuiceS3Bucket:
202     Type: AWS::S3::Bucket
203     Properties:
204       BucketEncryption:
205         ServerSideEncryptionConfiguration:
206           - ServerSideEncryptionByDefault:
207               SSEAlgorithm: AES256
208       VersioningConfiguration:
209         Status: Enabled
210       AccessControl: Private
211       # SECURE: Activation de la journalisation des acc s
212       LoggingConfiguration: # SECURE
213         DestinationBucketName: !Ref JuiceS3LogsBucket # SECURE
214         TargetPrefix: juice-shop-access-logs/ # SECURE
```

```
215       # SECURE: Activation des configurations de blocage de l'acc s
    public
216       PublicAccessBlockConfiguration: # SECURE
217         BlockPublicAcls: true # SECURE
218         IgnorePublicAcls: true # SECURE
219         BlockPublicPolicy: true # SECURE
220         RestrictPublicBuckets: true # SECURE
221
222   # SECURE: Ajout d'un r le IAM pour la surveillance am lior e de RDS
223   RDSEnhancedMonitoringRole: # SECURE
224     Type: AWS::IAM::Role # SECURE
225     Properties: # SECURE
226       AssumeRolePolicyDocument: # SECURE
227         Version: '2012-10-17' # SECURE
228         Statement: # SECURE
229           - Effect: Allow # SECURE
230             Principal: # SECURE
231               Service: monitoring.rds.amazonaws.com # SECURE
232             Action: sts:AssumeRole # SECURE
233       ManagedPolicyArns: # SECURE
234         - arn:aws:iam::aws:policy/service-role/
    AmazonRDSEnhancedMonitoringRole # SECURE
235
236   # RDS Instance (MySQL)
237   JuiceRDS:
238     Type: AWS::RDS::DBInstance
239     Properties:
240       DBInstanceIdentifier: juice-db
241       Engine: mysql
242       EngineVersion: 8.0
243       MasterUsername: admin
244       # SECURE: Suppression du mot de passe en clair. L'authentification
    IAM est activ e.
245       # MasterUserPassword: Password1234!-**souadprivateen # SECURE:
    Supprim
246       DBInstanceClass: db.t3.micro
247       AllocatedStorage: 20
248       PubliclyAccessible: false
249       StorageEncrypted: true
250       VPCSecurityGroups:
251         - !GetAtt JuiceRDSSecurityGroup.GroupId
252       DBSubnetGroupName: !Ref JuiceRDSSubnetGroup
253       # SECURE: Activation de l'authentification IAM
254       EnableIAMDatabaseAuthentication: true # SECURE
255       # SECURE: Activation de la configuration Multi-AZ
256       MultiAZ: true # SECURE
257       # SECURE: Activation de la surveillance am lior e
258       MonitoringInterval: 60 # SECURE: Intervalle en secondes (60 ou 1)
259       MonitoringRoleArn: !GetAtt RDSEnhancedMonitoringRole.Arn # SECURE
260
261   JuiceRDSSecurityGroup:
262     Type: AWS::EC2::SecurityGroup
263     Properties:
264       GroupDescription: Allow access to RDS from EKS
265       VpcId: !Ref JuiceVPC
266       SecurityGroupIngress:
267         - IpProtocol: tcp
268           FromPort: 3306
```

```
269           ToPort: 3306
270           CidrIp: 10.0.0.0/16
271           # SECURE: Ajout d'une description    la r gle de groupe de
    s curit
272           Description: Allow MySQL access from within the VPC CIDR block
     # SECURE
273
274   JuiceRDSSubnetGroup:
275     Type: AWS::RDS::DBSubnetGroup
276     Properties:
277       DBSubnetGroupDescription: RDS subnet group for Juice Shop
278       SubnetIds:
279         - !Ref JuiceSubnet1 # SECURE: Inclure le Subnet 1
280         - !Ref JuiceSubnet2 # SECURE: Inclure le Subnet 2
281
282 Outputs:
283   ClusterName:
284     Description: EKS Cluster Name
285     Value: !Ref EKSCluster
286
287   BucketName:
288     Description: S3 Bucket for Juice Shop
289     Value: !Ref JuiceS3Bucket
290
291   BucketLogsName: # SECURE: Ajout de l'output pour le bucket de logs
292     Description: S3 Bucket for Juice Shop Access Logs # SECURE
293     Value: !Ref JuiceS3LogsBucket # SECURE
294
295   RDSInstance:
296     Description: RDS Instance endpoint
297     Value: !GetAtt JuiceRDS.Endpoint.Address
298
299   EKSSecretsKMSKeyArn: # SECURE: Ajout de l'output pour la cl   KMS EKS
300     Description: ARN of the KMS key used for EKS secrets encryption #
     SECURE
301     Value: !GetAtt EKSSecretsKMSKey.Arn # SECURE
```

Listing 5.2: IaC corrected Output

5.2.4 Infrastructure visualization (After)

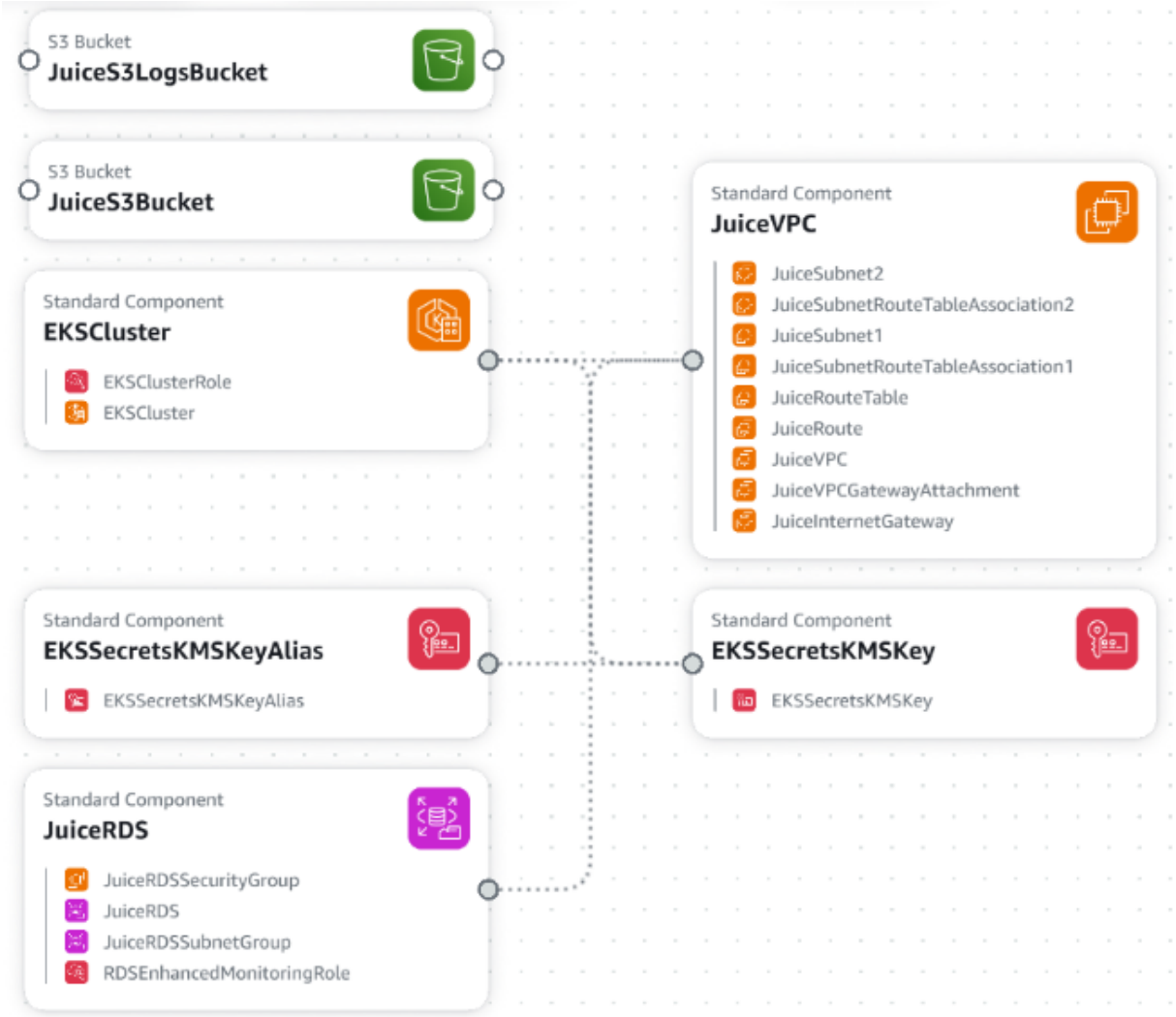


Figure 5.2: After the compliance

5.3 Security Auditing and Validation:

5.4 Compliance Validation Metrics

To quantify improvements, we will utilize objective metrics:

- **Severity Reduction:** Classifying detected issues (pre-correction) by severity (Critical, High, Medium, Low) and measuring the percentage reduction in critical/high severity issues after applying AI-generated fixes.
- **Number of Issues Fixed:** Total count of vulnerabilities and non-compliance issues identified and automatically corrected by the solution for a given IaC baseline.
- **NIST Control Coverage:** Percentage of applicable NIST SP 800-53 controls (relevant to IaC) effectively checked and enforced or improved by the solution.

- **Generation Accuracy (Precision/Recall):** Assessing the quality of generated corrections. False positives (unnecessary/incorrect changes) and false negatives (missed vulnerabilities/non-compliances) will be evaluated, likely requiring expert manual review on a representative sample.

Anticipated Quantitative Results (Illustrative)

We anticipate significant improvements based on the AI engine's capabilities:

- Reduction in Critical/High Severity IaC Issues: > 80
- Average Number of Automated Corrections per Complex File: 5-10
- Coverage of Applicable NIST IaC Controls: > 75
- Accuracy of AI-Generated Corrections (Manual Validation): > 90

These metrics will be gathered by running standard analysis tools (Checkov, tfsec, Trivy) on IaC files *before* and *after* processing by our AI engine, enabling data-driven comparison.

5.5 Time Efficiency Gains

Automating IaC detection and correction is expected to yield significant time savings for development and security teams. This will be estimated:

- **Qualitatively:** Comparing the typical manual workflow (write -> scan -> analyze -> research fix -> implement -> rescan) with the automated process (write -> push -> receive correction/validation).
- **Quantitatively (Estimate):** Measuring the average time for manual remediation of common IaC vulnerability types versus the AI processing time. Extrapolating this across the number of detected issues provides an estimate of time saved per deployment or sprint.

The goal is to demonstrate a substantial reduction in the effort required for routine IaC security and compliance tasks.

5.6 Standard Tool Reports

To provide independent validation, reports from recognized IaC analysis tools will be integrated into the evaluation.

5.6.1 Policy Enforcement with Open Policy Agent (OPA)

To complement the AI analysis and enforce specific organizational security standards, Open Policy Agent (OPA) is integrated into the validation pipeline. Custom policies are written in Rego.

Example Rego Policy

Below is a snippet from the Rego policy file used to validate CloudFormation templates, focusing on parameter security and resource configuration:

```
package cloudformation.validation

import rego.v1

# ==========================================================
# SECURITY AND COMPLIANCE
# ==========================================================

# Verifier l'encryption des buckets S3
deny contains msg if {
    some resource_name, resource in input.Resources
    resource.Type == "AWS::S3::Bucket"
    not resource.Properties.BucketEncryption
    msg := sprintf("Le bucket S3 '%s' doit avoir le chiffrement active",
    [resource_name])
}

# Verifier le blocage d'acces public pour S3
deny contains msg if {
    some resource_name, resource in input.Resources
    resource.Type == "AWS::S3::Bucket"
    not resource.Properties.PublicAccessBlockConfiguration
    msg := sprintf("Le bucket S3 '%s' doit avoir
   PublicAccessBlockConfiguration", [resource_name])
}

# Verifier l'encryption des bases de donnees RDS
deny contains msg if {
    some resource_name, resource in input.Resources
    resource.Type == "AWS::RDS::DBInstance"
    not resource.Properties.StorageEncrypted == true
    msg := sprintf("L'instance RDS '%s' doit avoir StorageEncrypted:
   true", [resource_name])
}

# ==========================================================
# NETWORK SECURITY
# ==========================================================
# Verifier les regles de securite trop permissives
deny contains msg if {
    some resource_name, resource in input.Resources
    resource.Type == "AWS::EC2::SecurityGroup"
    some rule in resource.Properties.SecurityGroupIngress
    rule.CidrIp == "0.0.0.0/0"
    rule.IpProtocol == "tcp"
    rule.FromPort != 80
    rule.FromPort != 443
    msg := sprintf("Security Group '%s' a une regle trop permissive
   (0.0.0.0/0) pour un port non-standard (%d)", [resource_name, rule.
   FromPort])
}

# ==========================================================
# IAM SECURITY
```

```
# ============================================================

# Verifier les politiques IAM trop permissives
deny contains msg if {
    some resource_name, resource in input.Resources
    resource.Type == "AWS::IAM::Policy"
    some statement in resource.Properties.PolicyDocument.Statement
    statement.Effect == "Allow"
    statement.Action == "*"
    statement.Resource == "*"
    msg := sprintf("La politique IAM '%s' accorde des permissions trop
   larges (*/*)", [resource_name])
}
```

Listing 5.3: Excerpt from OPA Rego Policy (rules.rego)

5.6.2 OPA Validation Results

OPA is executed against the IaC templates to identify policy violations. The following figure shows example output from running OPA against two different CloudFormation templates using the Rego rules:

```
C:\Users\testm>opa eval -d rules.rego -i template1.yaml "data.cloudformation.validation.deny"
{
  "result": [
    {
      "expressions": [
        {
          "value": [
            "L'instance RDS 'JuiceRDS' devrait être configurée en Multi-AZ",
            "Le bucket S3 'JuiceS3Bucket' doit avoir PublicAccessBlockConfiguration",
            "Le paramètre 'ClusterName' doit avoir une description",
            "Secret potentiel détecté en dur dans la ressource 'JuiceRDS' au chemin [\"Properties\", \"MasterUserPassword\"]"
          ],
          "text": "data.cloudformation.validation.deny",
          "location": {
            "row": 1,
            "col": 1
          }
        }
      ]
    }
  ]
}

C:\Users\testm>opa eval -d rules.rego -i template2.yaml "data.cloudformation.validation.deny"
{
  "result": [
    {
      "expressions": [
        {
          "value": [
            "Le paramètre 'EKSSecretsKmsKeyArn' contenant 'key' doit avoir NoEcho: true",
            "Le paramètre 'EKSSecretsKmsKeyArn' contenant 'secret' doit avoir NoEcho: true",
            "Le paramètre 'RDSSecretsManagerArn' contenant 'secret' doit avoir NoEcho: true",
            "Secret potentiel détecté en dur dans la ressource 'JuiceRDS' au chemin [\"Properties\", \"MasterUserPassword\"]"
          ],
          "text": "data.cloudformation.validation.deny",
          "location": {
            "row": 1,
            "col": 1
          }
        }
      ]
    }
  ]
}

C:\Users\testm>
```

Figure 5.3: Example OPA Validation Output

This output clearly lists the violations found in each template according to the defined Rego policies, such as missing Multi-AZ for RDS, missing PublicAccessBlockConfiguration for S3, and potential hardcoded secrets.

5.7 Validation and Comparison Dashboards

To visualize the security posture improvements and compare different template versions, interactive HTML dashboards were created.

5.7.1 Detailed Comparison Report

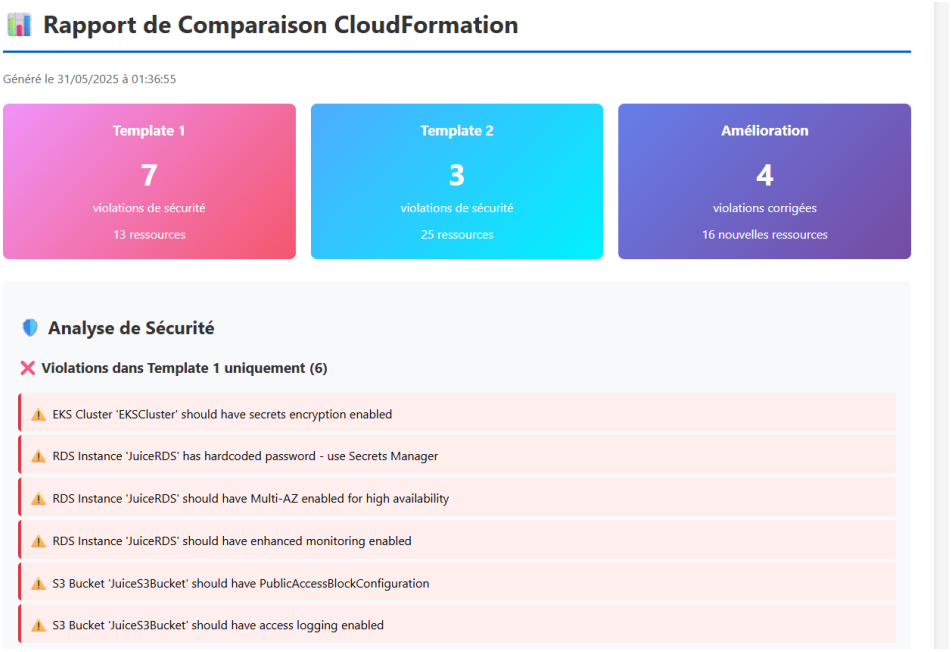


Figure 5.4: Detailed Comparison Report: Summary Metrics and Violations Unique to Template 1

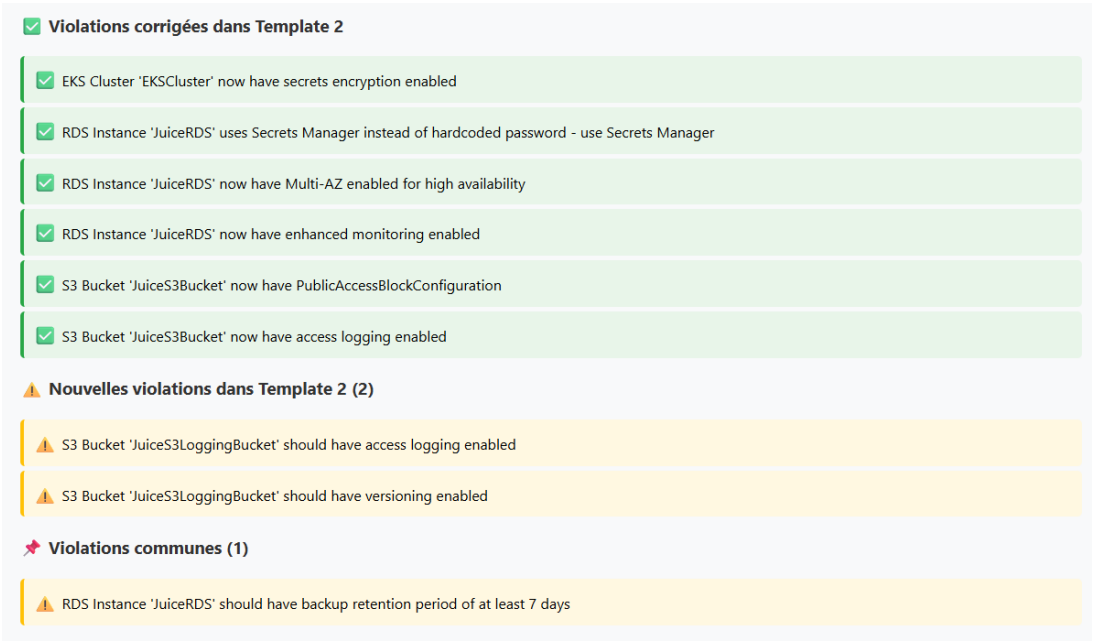


Figure 5.5: Detailed Comparison Report: Corrected, New, and Common Violations

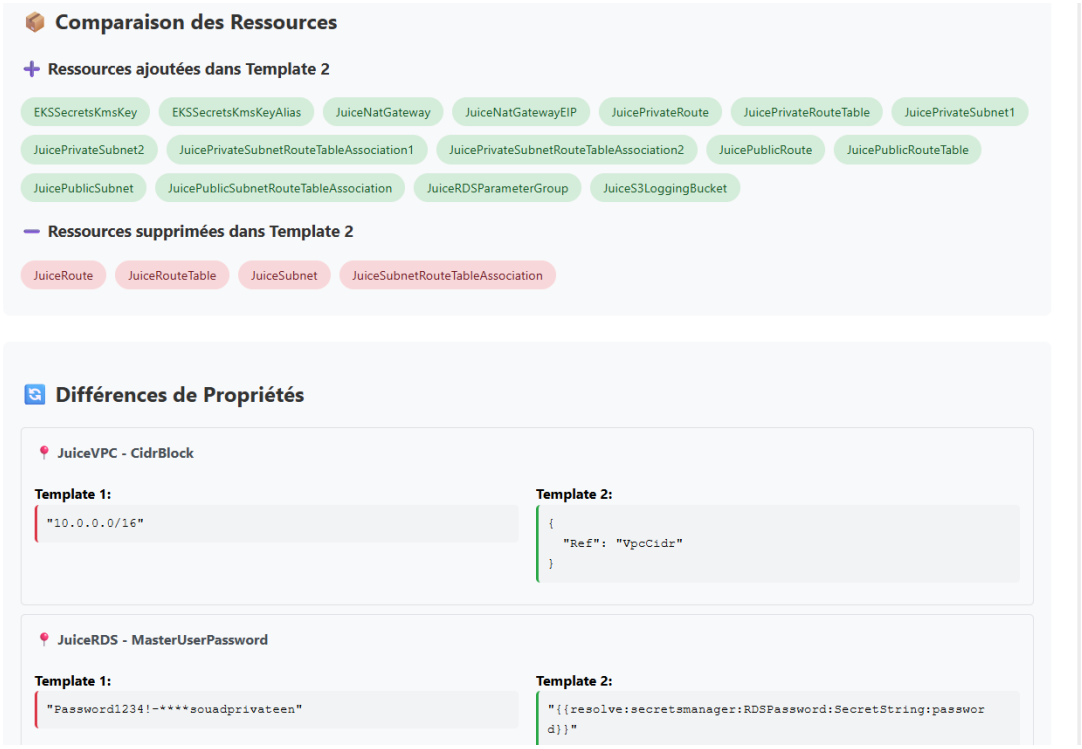


Figure 5.6: Detailed Comparison Report: Resource Comparison and Initial Property Differences

Figure 5.7: Detailed Comparison Report: Further Property Differences (EKS, RDS Security Group)

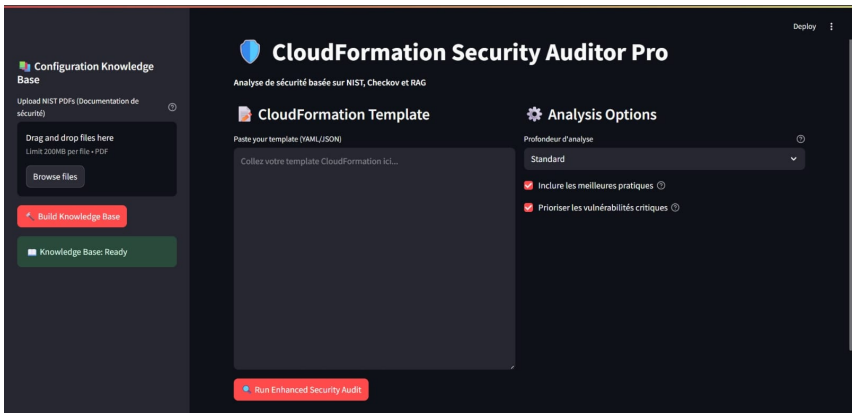
Figure 5.8: Detailed Comparison Report: Final Property Difference and Security Improvements Summary

Results

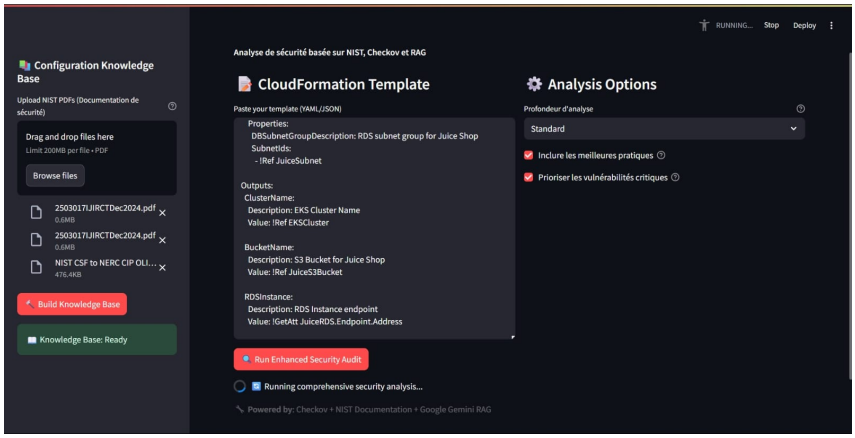
This report offers a more detailed, side-by-side comparison . It quantifies the number of violations, resources, and corrections between Template 1 and Template 2. It specifically lists violations unique to each template, those corrected, and those common to both. Furthermore, it details added/removed resources and specific property differences, providing granular insight into the changes made to enhance security.

5.8 LLM Implementation and Analysis

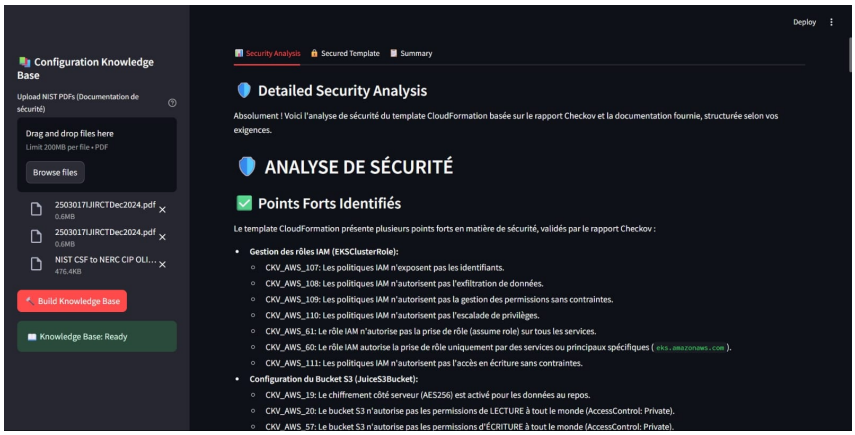
The LLM component plays a crucial role in analyzing IaC, identifying subtle security issues beyond static rules, and generating corrections



(a) Tool Interface: Configuration Knowledge Base Setup

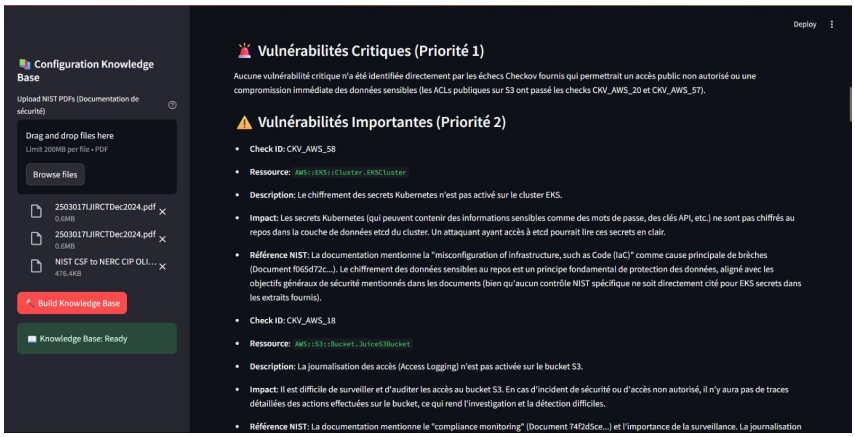


(b) Tool Interface: Initiating Security Analysis with IaC Input

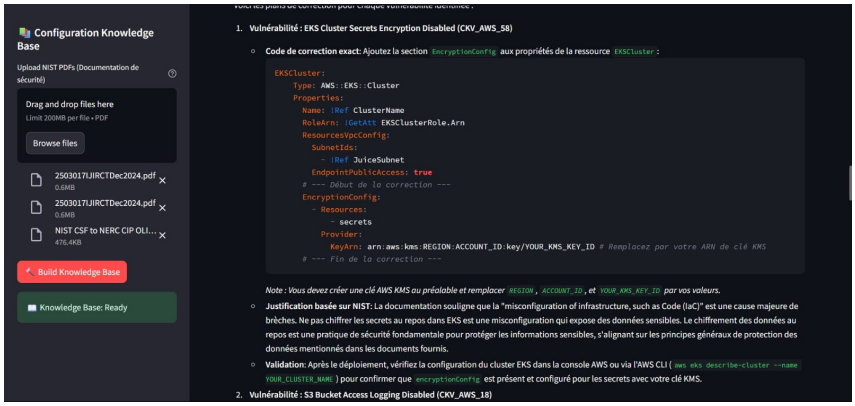


(c) Analysis Output: Summary of Identified Strengths

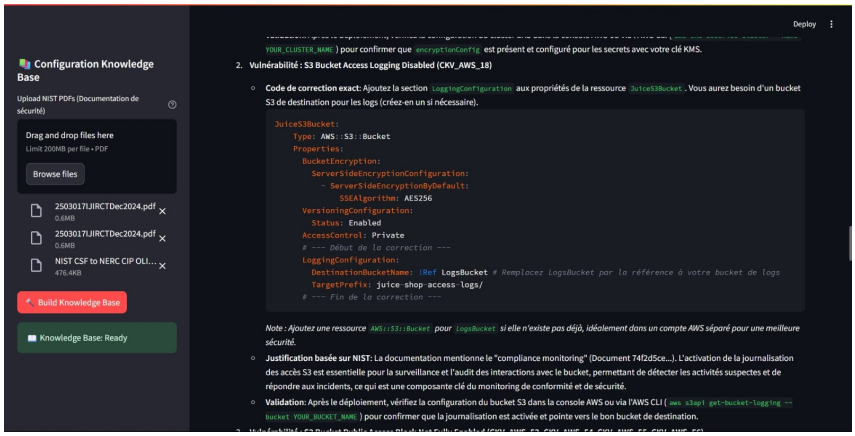
Figure 5.9: Screenshots of the AI Security Auditor Tool Interface and Analysis Output



(a) Analysis Output: Detailed View of Important Vulnerabilities (Priority 2)



(b) Generated Correction Example: Enabling EKS Cluster Secrets Encryption



(c) Generated Correction Example: Enabling S3 Bucket Access Logging

Figure 5.10: Screenshots of AI Security Auditor Tool Results and Generated Corrections

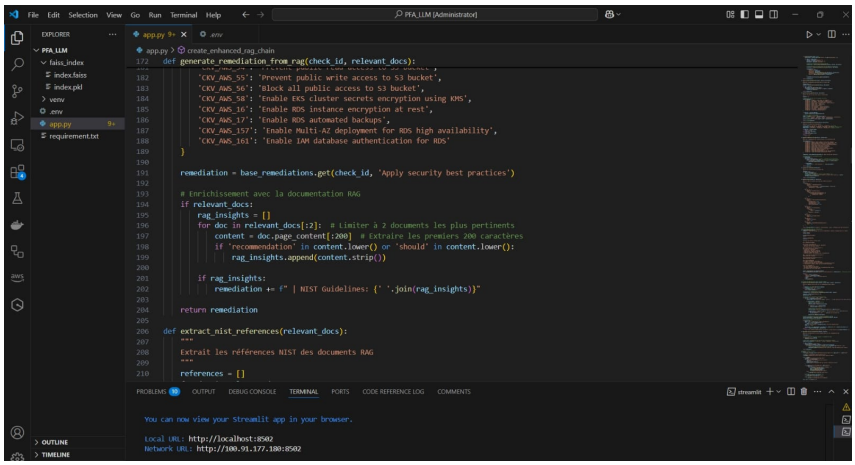


Figure 5.11: Code Snippet: Python Implementation of the RAG-based Remediation Generation

5.8.1 LLM Screenshot Descriptions

- **Figure 5.9a: LLM Tool Interface - Configuration:** This screenshot shows the user interface for configuring the LLM analysis. Users can specify the IaC file path, select the target security standards (e.g., NIST CSF, CIS Benchmarks), and potentially adjust analysis parameters like sensitivity level or specific rules to focus on.
- **Figure 5.9: LLM Tool Interface - Analysis in Progress:** This image depicts the tool actively analyzing the provided IaC file. It might show progress indicators, logs of the analysis steps (e.g., parsing IaC, querying LLM, processing results), and preliminary findings as they are identified.
- **Figure 5.9c: LLM Tool Output - Analysis Summary:** This screenshot presents a high-level summary of the LLM's findings after analyzing an IaC file. It typically includes counts of detected issues categorized by severity (Critical, High, Medium, Low) and potentially by compliance standard (e.g., NIST CSF function).
- **Figure 5.10a: LLM Tool Output - Detailed Vulnerabilities:** This image shows a detailed list of specific vulnerabilities and misconfigurations identified by the LLM. Each finding usually includes a description of the issue, the affected resource and line number in the IaC file, the severity level, and references to relevant security standards or best practices.
- **Figure 5.10b: LLM Tool Output - Generated Correction (EKS Example):** This screenshot showcases the LLM's capability to generate corrected IaC code. In this example, it displays the suggested modification for an EKS-related resource, addressing a detected vulnerability (e.g., enabling secret encryption or configuring secure network policies).
- **Figure 5.10c: LLM Tool Output - Generated Correction (S3 Example):** Similar to the previous figure, this image shows an LLM-generated correction, but for an S3 bucket configuration. It might illustrate adding encryption, enabling versioning, or configuring secure access policies based on the analysis.

- **Figure 5.11: LLM Tool - Code Snippet Integration Example:** This illustrates how the LLM's suggestions or analysis results might be integrated directly into a developer's workflow, potentially as code comments, annotations, or suggestions within an IDE or a code review tool, providing immediate feedback.

Results

Collectively, these evaluation components will provide a comprehensive assessment of the performance, reliability, and value proposition of the proposed AI-driven proactive IaC security solution

Chapter 6

Conclusion & Future Work

This project addressed the critical challenge of securing and ensuring compliance for Infrastructure as Code (IaC) within modern software development paradigms, particularly in cloud-native and DevSecOps environments.

6.1 Summary and Impact

Recognizing the limitations of existing, primarily reactive IaC security tools, we proposed and detailed an innovative solution leveraging Generative Artificial Intelligence (GenAI), specifically Large Language Models (LLMs). The core objective was to create a DevSecOps pipeline capable not only of identifying vulnerabilities and non-compliance in IaC scripts against rigorous standards like the NIST Cybersecurity Framework and SP 800-53, but crucially, of **automatically generating proactive corrections**, thereby producing secure and compliant IaC configurations.

The envisioned solution integrates into standard CI/CD workflows (e.g., GitHub Actions), analyzes IaC (primarily Terraform), interacts with an LLM engine for assessment and generation, and provides feedback to developers. We outlined a multi-environment target architecture (EKS, EC2) to demonstrate versatility and proposed a robust evaluation methodology based on before/after comparisons, quantitative metrics, and validation using standard third-party tools.

The primary impact of this approach lies in its potential to shift IaC security from a reactive, often manual process to a **proactive, automated, and integrated** activity within the developer workflow. By providing reliable, automated corrections, the solution promises to:

- Significantly accelerate delivery cycles by reducing manual remediation time.
- Enhance security and compliance posture from the earliest stages of development.
- Minimize the risk of deploying vulnerable or non-compliant infrastructure.
- Facilitate developer adoption of security best practices through contextual feedback and exemplary corrections.

Core Conclusion

The project demonstrates the transformative potential of Generative AI in revolutionizing IaC security and compliance management. By moving beyond detection to proactive generation of secure code, we pave the way for more intelligent, efficient, and inherently secure DevSecOps pipelines.

6.2 Future Work and Perspectives

While the proposed concept and architecture provide a strong foundation, several avenues exist for future enhancement and extension:

- **Deeper CI/CD Integration:** Develop richer integrations with various CI/CD platforms (GitLab CI, Jenkins, AWS CodePipeline) and explore more direct developer feedback mechanisms (e.g., IDE plugins).
- **Expanded IaC Language Support:** Extend the solution to analyze and generate code for other popular IaC languages like Pulumi, Ansible, CloudFormation (beyond basic examples), and Kubernetes YAML configurations.
- **LLM Refinement and Customization:** Investigate fine-tuning specific LLMs on curated datasets of secure IaC and NIST policies to enhance generation accuracy and relevance. Allow user customization of target compliance policies.
- **Enhanced Explainability:** Improve the system's ability to clearly articulate the rationale behind generated corrections, linking changes directly to specific NIST controls or security best practices.
- **Larger-Scale Validation and Testing:** Deploy and evaluate the solution in real-world or large-scale simulated production environments to assess performance, robustness, and scalability under load.
- **Integration with Application Security Findings:** Correlate IaC analysis results with findings from application security scanning tools (SAST/DAST/SCA) for a more holistic risk view.
- **Exploration of Advanced AI Techniques:** Investigate other AI methods, such as reinforcement learning, for optimizing the generation of complex IaC configurations or security policies.

In conclusion, this project lays the groundwork for a novel approach to IaC security. The future perspectives are numerous and promising, aiming to make cloud infrastructure not only easier to manage through automation but also fundamentally more secure through proactive artificial intelligence.

References

[1] Intelligent devsecops: Integrating ai with iac for nist-based security compliance project summary. Internal Project Document / PDF Attachment, 2024/2025. Provided as PRojet-IaC$_N IST_D evSecops.pdf$.

[2] Ai agent prompt for latex report generation on ai-based solutions for proactive security recommendations in web applications. User-provided text file, May 2025. Provided as pasted$_c ontent.txtcontainingdetailedinstructions for thereportstructureandcontent$.

[3] Ramaswamy Chandramouli. Implementation of devsecops for a microservices-based application with service mesh. Special Publication 800-204C, National Institute of Standards and Technology, March 2022.

[4] Carmen Cheh and Binbin Chen. Repairing infrastructure-as-code using large language models. In *2024 IEEE 6th International Conference on Trust, Privacy and Security in Intelligent Systems, and Applications (TPS-ISA)*, Oct 2024.

[5] I Wayan Adi Juliawan Pawanac d Ilsun You Daemin Shina, Jiyoon Kimb. Enhancingcloud-native devsecops:azerotrustapproachforthefinancial sector. *www.elsevier.com/locate/csi*, 2025.

[6] Sandip Dey. Enhancing cloud infra development with fine-tuned llms. Persistent Systems Blog, Feb 2025.

[7] JamilAhmad2 AdnanAhmedKhan3 TahirKhurshaid4 ImranAshraf Hassan Saeed1, Imran Shafi1. Review of techniques for integrating security in software development lifecycle. *https://www.techscience.com/*, 2024.

[8] Lu Mao. Evaluating llms for infrastructure as code. *GFT Engineering Blog on Medium*, Feb 2025.

[9] Xhesika Ramaj, Mary Sánchez-Gordón, Vasileios Gkioulos, Sabarathinam Chockalingam, and Ricardo Colomo-Palacios. Holding on to compliance while adopting devsecops: An slr. *Electronics*, 11(22):3707, 2022.

[10] Ravi Chandra Thota. Cloud-native devsecops: Integrating security automation into ci/cd pipelines. *International Journal of Innovative Research in Computer and Communication Technology (IJIRCT)*, 10, December 2024.