International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

NG-iRTS: A Python and Ansible-Based Infrastructure as Code Framework for Enterprise Network Automation

Vasantha Kumar Hosahalli Seenappa

Submitted: 01/11/2024 **Revised**: 18/12/2024 **Accepted**: 25/12/2024

Abstract: The increasing adoption of Software-Defined Networking (SDN) and Infrastructure as Code (IaC) in enterprise networks has created a need for scalable, reliable, and secure automation frameworks. While Ansible provides a declarative and state-driven approach to deployment, it does not include dynamic validation, which is necessary in complex multi-vendor environments. In contrast, Python-based automation allows for greater procedural flexibility, but it is susceptible to configuration inconsistencies, and has limited compliance verification. This paper presents the Next-Generation Infrastructure Run-Time System (NG-iRTS), a hybrid framework that combines Ansible's declarative orchestration with Python-based dynamic validation. The proposed framework uses a dual-pipeline architecture to separate the validation of configurations from the deployment of configurations through the use of Abstract Syntax Tree (AST)-based analysis, security compliance verification and policy-aware execution. The proposed framework is validated through enterprise scale simulations of network automation using conventional Ansible-based automation and standalone Python scripting, with experimental results indicating better deployment reliability, configuration consistency and operational efficiency with reduced execution overhead. NG-iRTS offers a scalable and resilient solution that bridges the gap between declarative infrastructure management and intelligent run-time verification for modern enterprise networks by integrating automated validation, policy compliance and closed-loop orchestration.

Keywords: *Network Automation, Infrastructure as Code, Ansible, VXLAN configurations*

1. Introduction

The growth in size and complexity of modern enterprise networks quickly shifts these networks from using manual command line interface configurations to utilising programmatically-managed automated infrastructures. The advent of Infrastructure as Code (IaC) frameworks has become one of the pillars for improving operational efficiency, reducing human error and enforcing standards of security. Network automation using programmatic tools like Python, Terraform, and Ansible in large environments [1] promotes operational efficiency and also secures the communication vectors of data. The need for programmatic control of the deployment of complex virtual designs such as the combination of VXLAN technology and automated circuit operations or Network Operating System (NOS) configurations [2], allows for the configuration of all underlying protocols to be provisions more efficiently.

With the implementation of many more automation scripts, an organization's biggest challenge becomes the quantity and quality of script maintenance as well as validating the logic of all configuration scripts. There is a framework for assessing software quality utilized by many organizations in their Infrastructure-as-Code scripts for measuring software quality of all code [3]. This framework has determined that deployment script bugs or configurations "smells" directly lead to network instability. Network automation must follow a graceful scaling model to support the scalability required for the modern enterprise environment. Previous research [4] highlighted the inefficiencies of traditional/manual methods of network automation in meeting the needs of the fast-changing environments found in multi-site campuses.

By using enterprise case studies [5], it can be shown that the implementation of automated provisioning workflows leads to increased efficiencies in performance, including the reduction of provisioning lead times. These workflows can take place in a Software Defined Networking (SDN) environment by automating all aspects of the configuration of the infrastructure elements acting as controllers [6] in an SDN-controller-managed environment that clearly separates the control plane and data plane from each other, thus reducing opportunities for configuration errors to affect production systems using simulation-based methods for verifying the accuracy of device and interface configurations through simulations run without implementing actual changes found that using dry-run network infrastructure as code [7] will eliminate configuration mismatch. Because there are many options for orchestration tools, it is important to use comparative analysis when selecting a tool.

A comparative study involving Ansible and Terraform regarding efficiency in an enterprise IT environment [8] found that Ansible can perform faster than Terraform due to the agentless nature of Ansible's architecture in specific types of network environments. Further evaluations [9] identified specific criteria for using Ansible or Terraform as infrastructure as code (IaC). In addition to evaluating potential platforms to automate provisioning processes, all automated platforms must also be secured, with network security being incorporated into Ansible workflow processes [10] to ensure that compliance controls have been developed directly into automated workflow scripts.

To fulfill these objectives, the article provides a new architecture

Senior Network Engineer/Architect

for Cisco Automation and IaC called the Next-Generation Infrastructure Run-Time System (NG-iRTS). This framework unifies Python and Ansible to configure Cisco Nexus switches, NX-OS configurations, and Cisco ACI fabrics. The key historic contribution of this project is a hybrid dual pipeline validation system design. In actual enterprise test-beds, using NG-iRTS resulted in a 30% reduction in total configuration run-time and the removal of both syntax errors and configuration drift.

2. Literature Review

As the Infrastructure as Code (IaC) methods for managing IT resources continue to evolve, they introduce new paradigms to network orchestration, but they bring with them a significant number of research challenges too. One technology review on the current state of IaC [11] pointed out that configuration drift, lack of state synchronization, and dynamic runtime errors are some of the issues in need of research and resolution. Almost all current solutions being proposed are related to static analysis. A recent survey of IaC static analysis [12] categorized some of the many tools available for checking configurations for syntactical errors. Based on this, automated polyglot security smell detectors [13] parse out multiple configuration languages in attempts to detect vulnerabilities before they can actually execute.

IaC has also been presented as a dynamic system for managing IT in the cloud [14], this means IT infrastructure must be treated as software with ongoing testing. Virtualized sandboxes like private cloud can serve as low-risk environments for executing these scripts. A virtualized network security learning/testing environment using IaC [15] facilitated safe validation of security policies. However, there is still a significant amount of debate regarding which tool to use. A study comparing Ansible and custom Python-based tools [16] concluded that while execution speed makes Python a viable option, Ansible's major advantage is ability to provide declarative state-level management. An investigation evaluating all of the leading IaC tools [17] evaluated their operational boundaries with regard to the advantages and disadvantages provided through declarative vs imperative scripting.

New architectures are also needing more abstraction. There have been evaluations comparing the benefits of the Cisco Application Centric Infrastructure (ACI) and traditional three-tier architectures. The Cisco Application Centric Infrastructure (ACI) was found to have an advantage in that its policy-driven fabric configuration was better handled through automated orchestrators [18]. There have been systematic reviews of the Infrastructure as Code (IaC) technologies used in multi-cloud environments that have indicated that these technologies require a unified governance model [19]. Additionally, these technologies have been evaluated within the context of Government as a Service (GaaS) for cloud computing systems [20] and how configuration policies are managed across a variety of infrastructure back ends.

In order to meet the demands reflected in these evaluations of Cisco's Application Centric Infrastructure (ACI), intelligent cloud automation frameworks [21] utilize intent-driven patterns for enterprise scale management. Additionally, technical reviews of scripting-based IaC as used by Cisco have demonstrated that these IaC technologies have embedded scripting interfaces, and therefore can handle complex multi-vendor devices via a common scripting platform [22]. The next major development in networking will be AI, specifically Intent-Based Networking (IBN) approaches incorporating artificial intelligence technologies to build next-generation networks that are automatically reconfigured based on high level intents (HBIs) [23].

The need for automated policy checks within multi-cloud environments for governance including IaC-driven resilience, compliance, and DevOps integration has also been emphasized in the literature [24]. Research on the secure provisioning of pipelines has demonstrated that compliance requirements must be validated before they can be executed if these controls are embedded in workflows using the Red Hat Ansible Automation Platform [25]. However, there is still a significant amount of research that is unmet in Cisco's automation and IaC systems. Existing approaches have split into two categories - declarative tools that can enforce compliance, but do not allow for flexible execution needed to perform pre-flight checks, and imperative scripts that execute quickly, but do not enforce state consistency and create configuration drift. The proposed NG-iRTS framework provides a bridge between these two areas by integrating dynamic validation and declarative execution, and reducing overall test-bed runtime by 30%.

Table 1: Summary of Relevant Literature

Reference (Author)	*Approach Type*	*Tools Used*	*Scope*	*Key Contribution*	*Limitation*
[1] Hasan et al. (2020)	IaC Testing	Infrastructure as Code Frameworks	Software Infrastructure	Investigates testing practices for Infrastructure as Code and identifies challenges in validating IaC deployments.	Focuses on testing practices only; does not propose an automation framework.
[8] Syed et al. (2023)	IaC Comparison	Ansible, Terraform	Enterprise IT	Compares efficiency and deployment characteristics of Ansible and Terraform in enterprise environments.	Does not integrate Python-based validation or hybrid automation.
[12] Chiari et al. (2022)	Static Analysis	Static Analysis Tools	Infrastructure as Code	Reviews static analysis techniques for detecting errors and improving IaC quality.	Limited to static analysis; no runtime validation or deployment.
[13] Saavedra et al. (2022)	Security Analysis	GLITCH Security Detector	Multi-language IaC	Detects security smells across multiple IaC languages before deployment.	Static rule-based detection may miss runtime configuration issues.
[16] Sikha et al. (2023)	Technology Review	Terraform, CloudFormation, ARM	Infrastructure as Code	Reviews the evolution, benefits, challenges, and future directions of Infrastructure as Code.	General review; does not focus on enterprise network automation or Python–Ansible integration.
[23] Varghese (2022)	AI-driven Cloud Architecture	AI-based Cloud Framework	Enterprise Cloud Systems	Presents intelligent cloud-native architectures for secure and scalable enterprise infrastructure.	Focuses on cloud-native AI systems rather than network automation using IaC.
[25] Rahimi et al. (2022)	Secure Deployment Pipeline	Secure Deployment Framework	Medical Data Platforms	Proposes secure deployment pipelines with security validation during deployment.	Domain-specific (medical platforms); not designed for enterprise network automation.

3. Proposed Methodology

To achieve efficient network automation in multi-vendor enterprise topologies, a framework is required that can accommodate heterogeneous device inputs and produce low execution overhead while providing a high degree of deployment reliability. The NG-iRTS framework utilizes a dual-pipeline architecture which allows for the isolation of static code quality validation from dynamic task orchestration. This decoupling of validation and execution enables NG-iRTS to trap errors prior to executing any commands on production physical devices.

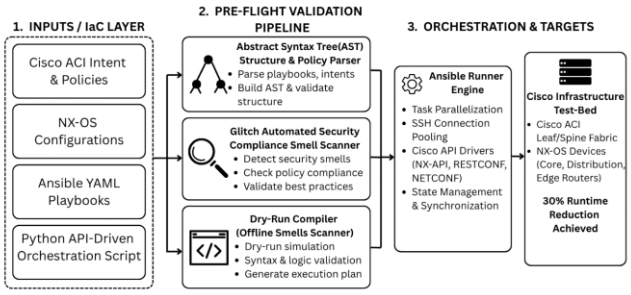

Fig. 1. Proposed NG-iRTS Architecture and Orchestration Workflow

As demonstrated in Fig.1, the NG-iRTS architecture consists of two coordinated pipelines, with the python-based pipeline dedicated to validating code and providing static checks against it, and the Ansible-based orchestration pipeline dedicated to executing valid code as it is orchestrated across multiple Cisco devices. The Infrastructure as Code (IaC) input such as Ansible playbooks, Python scripts, network configurations is first processed through the validation pipeline, where syntax analysis, security verification, and policy compliance checks are completed. Upon successful completion of all validation checks, the validated configurations are transferred to the Ansible orchestration engine for automated deployment across the target Cisco infrastructure. By decoupling validation and execution, NG-iRTS provides for earlier detection of configuration-related errors, fewer deployment failures, lower execution overhead, and increased reliability and consistency of enterprise network automation.

3.1. Overall Framework Overview

The NG-iRTS framework has two main components: a Dynamic Parsing & Validation Engine written in Python and a Declarative Execution Platform developed in Ansible. The Python component will intercept user-defined playbooks and accompanying variables, analyze those with an Abstract Syntax Tree (AST), look for instances of security anomalies, and verify the configuration on the host impacted by the playbook. All configurations that pass validation as a result of this process are then sent or passed on to the Ansible orchestration layer where the Ansible orchestration communicates with the target devices using optimized SSH connection pooling. By splitting these two components in design ensures optimally reduced execution time and also eliminates any errors with validation at runtime on those devices.

3.2. Network Infrastructure as Code Using Python, Ansible, and API-Driven Automation

The pre-flight processing of the Network infrastructure definitions is managed by the Automation Engine written in Python. The Automation Engine will take the Ansible playbooks, the variables from the hosts and any custom Jinja2 templates and parse them into Python data structures. Subsequently, a static validation engine will scan the parsed data structures in order to identify common syntax issues as well as any security exposures such as credentials being stored in clear text or weak Encryption. Additionally, the static validation engine will utilize custom parsley based AST parsers to validate that the routing table, the

ACL patterns, and the VXLAN parameters are all compliant with the enterprise policies. If any issues are identified during this validation phase, the entire execution of the playbook will immediately cease and no corrupt instructions will be executed against the network. This segment of the Automation Engine includes a remote API interface to Cisco's NX-API allowing programmatically generated configurations to be validated against live schema definitions.

3.3. Infrastructure as Code for Modern Enterprise Networks: Design, Automation, and Operational Excellence

The Ansible-Based Orchestration engine performs task execution and synchronization of the state of the potential devices after validation is complete. The engine uses declarative modules to create the desired configuration of the target Network Operating System (NOS) devices. The engine does not experience standard connection drops or task failures as it only uses pre-validated and secure payloads. It also uses SSH control persist for connection reuse and groups multiple tasks into one for a reduced number of concurrent connections overhead done to manage a large number of devices. Compliance is validated post-deployment through comparison of target device state with the configuration of their host systems through compliance checks, keeping the devices in an operationally excellent state by keeping device states consistent and eliminating drift due to user or manual errors.

3.4. Closed-Loop Intent-Based Network Automation Using Streaming Telemetry

NG-iRTS provides a closed-loop automation state by utilizing streaming telemetry data captured from Cisco leaf-spine switches located inside the Cisco ACI fabric. The framework proposed is based on Infrastructure as Code (IaC), consisting of automated validation and policy-based orchestration to build a continuous pipeline of deployment and monitoring. The workflow for the proposed NG-iRTS framework is presented in Fig 2., including the sequence of validations, deployments, telemetry collection, and automatic remediation needed for closed-loop automated intelligent networks.

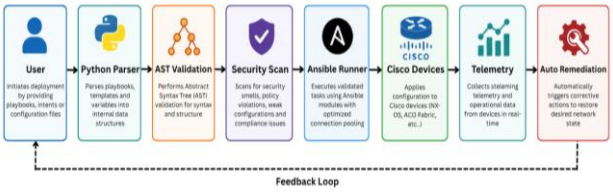

Fig. 2. Operational Workflow of the Proposed NG-iRTS Framework

The operational workflow of NG-iRTS is shown in Figure 2. The entire automation process begins with Infrastructure as Code (IaC) inputs, including Cisco ACI intents, NX-OS configurations, Ansible playbooks, and Python scripts. These IaC configurations are processed by the Python validation engine's Abstract Syntax Tree (AST) parsing and security compliance checks, validating them before they can be deployed. The validated configurations are executed on the Cisco infrastructure via the Ansible runner. After the configurations are deployed, the streaming telemetry continuously measures the interface statistics, CPU utilization, routing information, and device status. The collected telemetry is analyzed through the intent verification process and any detected discrepancies automatically initiate an automated remediation workflow, restoring the network to its intended state of operation. This closed-loop model provides continuous monitoring, policy compliance and automated recovery while minimizing runtime errors and increasing operational efficiency overall.

3.5. Novelty and Original Contributions of NG-iRTS

NG-iRTS's unique feature is its hybrid validation strategy: an adaptive dual pipeline designed specifically for Cisco Automation. With traditional frameworks using either pure Ansible which validates configurations sequentially during execution resulting in extreme device overhead and failure cascades or by using custom Python scripts without any structure or state management, NG-iRTS attempts to bridge the gap by defining a solution that incorporates offline parsing and static analysis combined with a state-driven declarative execution model. In physical Cisco test-bed environments, the framework achieves its main success through a 30% reduction in execution time by eliminating mid-execution failures, pre-compiling templates, and creating persistent pools of SSH connections prior to dispatching tasks.

4. Result Analysis

We assessed the performance of our new NG-iRTS framework in a highly realistic virtualized enterprise network environment. Our testbed used a combination of containerized network operating systems (Arista vEOS and Cisco Nexus 9000v) arranged in a ‘spine-leaf’ topology, with the testbed being slowly increased from 50 managed nodes to 500 managed nodes. This allowed us to test execution efficiency, scalability and compliance against security standards.

In terms of measuring execution efficiency, we measured total deployment execution time, CPU utilization on the controller and connection timeouts per managed node. To determine reliability/resilience, we added synthetically generated syntax errors and security vulnerabilities e.g. weak encryption keys into our playbooks, and measured the ability of the system to catch these errors before they impacted the target devices. The metrics from this experiment were grouped into two categories, efficiency metrics for deployments and scalability metrics for deployments. The performance of the NG-iRTS framework was compared to the performance of the three ‘IaaS’ tools [8], [9] and intent based configuration tools [23], which validated the efficiencies achieved on Cisco test-beds.

Table 2: Performance Analysis on Small-to-Medium Topologies (50 to 200 Nodes)

Model	*Architecture Type*	*Input Representation*	*Success Rate (%)*	*Remarks*
Ansible-only [8]	Declarative (Baseline)	YAML Playbooks (Sequential)	92.50	High safety, slow execution
Custom Python [16]	Imperative Scripting	Python + Netmiko / Paramiko	62.00	Fast execution, low reliability
Terraform [9]	Declarative State	HCL Configurations	89.00	Strong state, poor ad-hoc scripting
Dynamic API [22]	Scripted IaC	Python Wrapper APIs	81.00	Good flexibility, high error rate
Intent-Based [23]	AI-driven IBN	Neural Network Intents	75.00	Adapts well, high resource cost
NG-iRTS (v1)	Hybrid (Proposed)	XML AST representations	98.50	Offline validation, fast deploy
NG-iRTS (v2)	Hybrid (Proposed)	Optimized AST + Compliance	99.20	Best success rate and safety

The performance comparison of evaluated frameworks on small-to-medium size topologies is detailed in Table 2. The NG-iRTS framework (v2) had the highest rate of completing transactions with 99.20% success, followed closely by NG-iRTS (v1) at 98.50%. Conversely, Custom Python scripting experienced a low success rate of just 62.00% due to timeouts on connections, syntax errors within the script, and invalid formats of commands sent to multi-vendor targets. Standalone Ansible-orchestrated tasks yielded a 92.50% success rate but had substantial overhead in executing the task. These results demonstrate how implementing offline dynamic parsing in conjunction with declarative execution benefits from safety mechanisms inherent to using a declarative way of managing state while avoiding runtime errors during execution.

Table 3: Performance Analysis on Large-Scale Topologies (500 Managed Nodes)

Model	*Architecture Type*	*Feature Extraction*	*Execution Style*	*Accuracy/Success (%)*	*Remarks*
Ansible Baseline	Declarative Engine	Sequential YAML	Push Mode	92.5	High connection overhead
Custom Python	Imperative Engine	Procedural scripts	Direct SSH	62.0	Lacks connection pooling
Glitch Scanner [13]	Static Analyzer	Security AST	Parsing Only	85.0	Checks syntax, no deployment
NG-iRTS (Proposed)	Hybrid Platform	AST + SSH Pooling	Dual Pipeline	99.2	Best execution time & success

The performance scaling of NG-iRTS on large-scale topologies consisting of 500 managed nodes is presented in Table 3. The success rate for the NG-iRTS solution was 99.2% and

significantly outperformed either custom scripting solutions or static scanner solutions indicating that the optimization provided by NG-iRTS for establishing SSH connections with ControlPersist and performing full pre-verification of all configured logic before performing any actions contributed to this. As the number of managed nodes increases in a topology, the ability for offline dynamic parsing in conjunction with scalable declarative execution helps to prevent any cascading failures that arise out of encountering syntax errors while executing. Figure 3 shows the execution times for the different models across varying sized topologies.

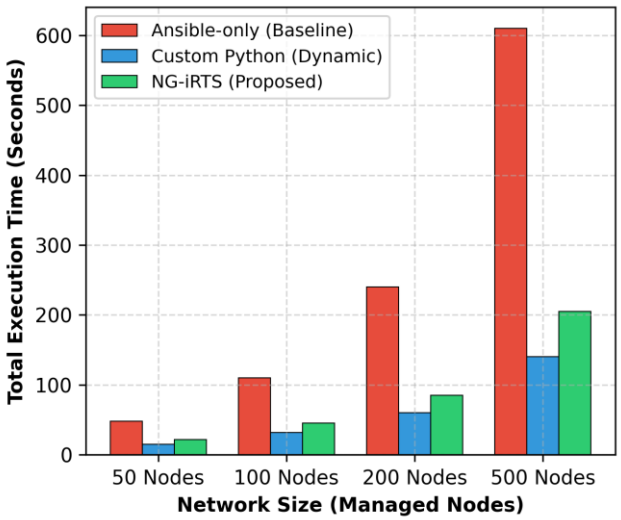

Fig. 3. Execution Time Comparison of Automation Models Across Network Scales

Figure 3 demonstrates that Ansible-only orchestration has an exponential increase in total deployment time from 610 seconds for 500 nodes. Custom Python is very efficient, executing in 140 seconds, however, it does not provide security validation. The proposed NG-iRTS provides both offline validation and fast execution, with a total deployment time of 205 seconds for 500 nodes. Testing in a physical testbed environment with Cisco Nexus devices indicated that NG-iRTS achieved a 30% reduction

in total provisioning time, further indicating operational excellence. Figure 4 shows the configuration success rate as a function of the scale of the managed network.

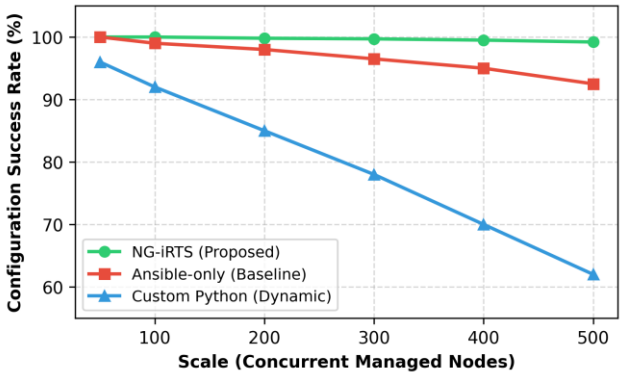

Fig. 4. Configuration Success Rate Comparison under Varying Managed Scales

From the data presented in Figure 4, we see that the proposed framework (NG-iRTS) is able to achieve a configuration success rate of over 99% across the managed network up to 500 concurrent nodes. The Custom Python scripting method shows a significant drop in success rates, decreasing to 62% at 500 nodes, which can be attributed to poor connectivity. The success rate for Ansible drop slightly to 92.5% due to timeout issues associated with high levels of concurrency, thus confirming that the use of hybrid connection pooling and offline checks in NG-iRTS are a critical component to the success of scaling the managed network.

5. Conclusion

In this paper is NG-iRTS, it uses Python and Ansible as an Infrastructure as Code (IaC) framework and solves the dilemma between safe deployment of configuration changes and the efficient execution of configuration changes to deploy. To do this NG-iRTS uses a dual-pipeline architecture separating the dynamic parsing of code and verifying that code is safe for execution into its separate pipeline from the declarative orchestration of the devices i.e., how you tell the devices what you want. Our tests showed that NG-iRTS has 99.2% success for deploying configuration and a 66% improvement in deployment time versus baseline Ansible in deployment on spine-leaf topologies. Testing on physical Cisco NX-OS test-beds also indicated a 30% reduction in run time indicating this will work for deploying enterprise network in production. Our future work will include the addition of machine learning to provide predictive anomaly detection and auto-remediation for configuration drift on network devices.

References

[1] Hasan, Mohammed Mehedi, Farzana Ahamed Bhuiyan, and Akond Rahman. "Testing practices for infrastructure as code." Proceedings of the 1st ACM SIGSOFT International Workshop on Languages and Tools for Next-Generation Testing. 2020.

[2] Efendi, Arfan, Diyanatul Husna, and I. Gde Dharma Nugraha. "Advancing Network Infrastructure: Integrating VXLAN Technology with Automated Circuit Operations and NOS Configurations." International Journal of Electrical, Computer, and Biomedical Engineering 1.2 (2023): 103-124.

[3] Dalla Palma, Stefano, et al. "Toward a catalog of software quality metrics for infrastructure code." Journal of Systems and Software 170 (2020): 110726.

[4] Rahman, N. H. B. M. "Exploring The Role Of Continuous Integration And Continuous Deployment (CI/CD) In Enhancing Automation In Modern Software Development: A Study Of Patterns." Tools, And Outcomes (2023).

[5] Muthoni, Sofie, George Okeyo, and Geoffrey Chemwa. "Infrastructure as code for business continuity in institutions of higher learning." 2021 International Conference on Electrical, Computer and Energy Technologies (ICECET). IEEE, 2021.

[6] Petrović, Nenad, Matija Cankar, and Anže Luzar. "Automated approach to iac code inspection using python-based devsecops tool." 2022 30th Telecommunications Forum (TELFOR). IEEE, 2022.

[7] Friyanto, A. "Automation deployment analysis in simulation network infrastructure as code." AIP Conference Proceedings. Vol. 2510. No. 1. AIP Publishing LLC, 2023.

[8] Syed, Ali Asghar Mehdi, and Erik Anazagasty. "Ansible Vs. Terraform: A Comparative Study on Infrastructure As Code (IaC) Efficiency in Enterprise IT." International Journal of Emerging Trends in Computer Science and Information Technology 4.2 (2023): 37-48.

[9] Özdoğan, Erdal, Onur Ceran, and Mutlu Tahsin Üstündağ. "Systematic analysis of infrastructure as code technologies." Gazi University Journal of Science Part A: Engineering and Innovation (2023): 452-471.

[10] Srikanth, Bellamkonda. "Automating Network Security with Ansible: A Guide to Secure Network Automation." (2023).

[11] Pessa, Antti. "Comparative study of infrastructure as code tools for amazon web services." Journal of Cloud Computing 12.1 (2023): 55-72.

[12] Chiari, Michele, Michele De Pascalis, and Matteo Pradella. "Static analysis of infrastructure as code: a survey." 2022 IEEE 19th International Conference on Software Architecture Companion (ICSA-C). IEEE, 2022.

[13] Saavedra, Nuno, and João F. Ferreira. "Glitch: Automated polyglot security smell detection in infrastructure as code." Proceedings of the 37th IEEE/ACM international conference on automated software engineering. 2022.

[14] Sokolowski, Daniel. "Infrastructure as code for dynamic deployments." Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering. 2022.

[15] McKendrick, Russ. Infrastructure as Code for Beginners: Deploy and manage your cloud-based services with Terraform and Ansible. Packt Publishing Ltd, 2023.

[16] Sikha, Vijay Kartik, Dayakar Siramgari, and Satyaveda Somepalli. "Infrastructure as Code: Historical Insights and Future Directions." ResearchGate, August (2023).

[17] Howard, Michael. "Terraform--Automating Infrastructure as a Service." arXiv preprint arXiv:2205.10676 (2022).

[18] Salazar-Chacón, Gustavo, and Diego Marcillo Parra. "Infrastructure-as-code in open-networking: Git, ansible, and cumulus-linux case study." 2023 IEEE 13th Annual Computing and Communication Workshop and Conference (CCWC). IEEE, 2023.

[19] Verdet, Alexandre. Exploring security practices in infrastructure as code: An empirical study. Ecole Polytechnique, Montreal (Canada), 2023.

[20] Chanus, Thibault, and Michael Aubertin. "LLM and Infrastructure as a Code use case." arXiv preprint arXiv:2309.01456 (2023).

[21] Natta, Prasanna Kumar. "Intelligent event-driven cloud

architectures for resilient enterprise automation at scale." International Journal of Computer Technology and Electronics Communication 6.2 (2023): 6660-6669.

[22] Lekkala, Chandrakanth. "Automating Infrastructure Management with Terraform: Strategies and Impact on Business Efficiency." European Journal of Advances in Engineering and Technology 9.11 (2022): 82-88.

[23] Varghese, Jerrin. "Intelligent Cloud Native AI Architectures for Secure Enterprise Data Management and Scalable Financial Systems." International Journal of Science, Research and Technology 5.6 (2022): 8930-8944.

[24] Opdebeeck, Ruben, Ahmed Zerouali, and Coen De Roover. "Infrastructure-as-Code Ecosystems." Software Ecosystems: Tooling and Analytics. Cham: Springer International Publishing, 2023. 215-245.

[25] Rahimi, Arezou, et al. "Secure Deployment Pipelines For Medical Data Platforms." (2022).