The Integrating Agile and Infrastructure as Code: A Pulumi-Based Framework for Secure Cloud Development

N.Veekshana Reddy
Computer Science and Engineering ,
KL University ,
Andhra Pradesh , India
veekshanareddyn@gmail.com

T.V.S.Hema
Computer Science and Engineering,
KL University , Andhra Pradesh , India
hematadikamalla@gmail.com

N.Siva Prasad
Computer Science and Engineering,
KL University, Andhra Pradesh , India
nallasivaprasad2004@gmail.com

B. Glory Sathwika
Computer Science and Engineering ,
KL University ,
Andhra Pradesh , India
bglory-sathwika2005@gmail.com

***Abstract*— This paper presents a feasible approach to designing robust and flexible software systems by combining Agile development practices with Infrastructure as Code (IaC) using Pulumi. Traditional methods are unable to support both rapid flexibility and robust security simultaneously. To overcome this drawback, the proposed approach integrates security constraints and validation processes directly into the software development process. Pulumi allows cloud infrastructure to be described and implemented using popular programming languages, ensuring consistency between application code and infrastructure. The proposed approach utilizes GitHub Actions and Pulumi's Automation API to facilitate continuous integration, automated deployment, secure upgrade, and auditable change management. Moreover, the proposed method is compatible with international standards like ISO 27001, thus improving the security, reliability, and maintainability of cloud systems.**

Keywords*— *Agile methodologies, Infrastructure as Code (IaC), Pulumi, Cloud security, Continuous integration and deployment (CI/CD), Automation API, ISO 27001 compliance, Secure software development.

I. INTRODUCTION

Cloud-native technologies and distributed systems have significantly transformed how companies deliver digital services. They enable organizations to scale globally and come up with new ideas faster than ever. This all comes from using Agile methods and those CI/CD pipelines that focus on quick changes, releasing often, and keeping customers in mind while building software. But there's this trade-off with flexibility and security that comes up a lot. Agile makes things responsive, sure, but if you don't build in security from the start, it kind of gets overlooked in the whole process.

One big problem in cloud setups is misconfigurations. Those are the main reason for data breaches and breaking rules on compliance. Like, if IAM roles aren't set right, or network security groups are too open, or you forget about encryption, sensitive stuff ends up exposed. Traditional ways of setting up infrastructure lead to inconsistencies too. Development, testing, and production environments start to differ, and that drift creates openings for bad actors. Plus, without good checks and monitoring early on, it's hard to spot vulnerabilities in the pipeline. Teams end up doing audits late, which takes forever and costs a bunch.

To fix these kinds of issues, Infrastructure as Code, or IaC, has turned into a key part of DevOps and cloud work these days. It lets you describe infrastructure in code, declaratively, so deployments are consistent and you can trace everything. Unlike manual setup, IaC fits right into version control. Changes get reviewed by peers, audited, and you can roll them back if needed. This cuts down on drift and makes teams more accountable, especially

when they're spread out. Among IaC tools, Pulumi stands out. It doesn't use those special DSLs like in Terraform or AWS CloudFormation. Instead, you write infrastructure with regular languages, Go or TypeScript or Python. That blurs the line between app coding and managing infrastructure, because developers use loops and if statements and classes, things they're already good at. Pulumi also plugs into CI/CD setups, making infrastructure part of the main software flow.

This framework integrates Agile methodologies with Pulumi-based IaC , using GitHub Actions and the Pulumi Automation API to run it all. It builds in security checks, compliance rules, and policy as code right into the Agile cycle. For instance, things like least-privilege access or required encryption or limiting network access get written into the Pulumi scripts. Then they're checked automatically before anything deploys. It lines up with standards like ISO/IEC 27001, so companies can adapt quickly but still keep trust, security, and the ability to audit.

The approach fills a gap in DevOps practices by mixing agility with security. It seems like a way for enterprises to modernize cloud ops, handling compliance and staying tough against threats that keep getting more complicated. Not everything is fully sorted out here, but that unification feels important.

II. PROBLEM STATEMENT

DevOps has completely transformed the way software is delivered deployments can now be executed much faster than ever before but DevOps does not tend to enforce either security or compliance. Most DevOps pipelines are developed with speed and "continuous delivery" as their first priority; therefore, if security controls are only applied during or after deployment, there is a greater risk of introducing vulnerabilities into the application. Many validations of infrastructure occur post-deployment (typically in production) as well. For example, misconfigured infrastructure, misaligned policy violations, and default insecure settings may already exist before they are discovered. This reactivity causes delays in the application development process, additional rework to the application after it has been developed, and puts sensitive data at risk of being exposed, in effect taking away any value that was generated by quick deployments.

- There are currently many Infrastructure as Code (IaC) tools, such as Terraform, CloudFormation, and Ansible, that improve the automation of resource provisioning via programmatic means. Unfortunately, most of these tools are built on top of Domain Specific Languages (DSLs); therefore, there are several mechanics that negatively impact development from using these tools:
- Learning curve - Developers and operations must learn yet another new syntax and constructs for a programming language. This will make it more difficult for them to adopt the language, and they will likely make more mistakes when developing.
- Limited interconnectivity with application logic - As a result, DSLs do not allow the development of infrastructure code as a cohesive unit with application code or sharing of policy across both domains.
- Advanced Policy Enforcements – Merging of security and compliance rules frequently necessitates extra modules or external tools, further increasing the pipeline. A major deficiency of current IaC methods is the absence of intrinsic policy enforcement and automated traceability. Without such mechanisms, organizations cannot consistently guarantee that their infrastructure complies with regulatory standards like ISO/IEC 27001, NIST CSF, or GDPR. Such fragmentation of integration enables holes in audit trails, inhibits correlating code changes with state of the infrastructure, and hindering the response during a compliance audit or a security breach. A network access control list (ACL), as an example, could become misconfigured and expose sensitive resources; if only manual or post-deployment checking of the policy occurs, the risk may remain undetected for several weeks.

In short, the major issues of DevOps and IaC practice nowadays are:

- Reactive Security Practices: Security comes into play mostly at deployment time and sparingly during the pipeline.
- Separation of Infrastructure and Application Code: IaC tools driven by DSL also prevent smooth integration with application logic.
- Manual or Piecewise Compliance

Enforcement: Regulation compliance cannot easily be computerized and checked.

- Traceability Gaps: Lack of direct linkage between commits, infrastructure state, and policy enforcement prevents comprehensive auditability.

These are challenges that require a framework of combining Agile software development with IaC and embedding security and compliance verification as part of the pipeline. With this type of framework, an organization can realize quick release cycles and yet neither compromise security and compliance nor trade off traceability with a systematic and automated approach of dealing with modern infrastructures of the cloud

III. LITERATURE REVIEW

The rising use of DevSecOps indicates increasing acceptance that security needs to become part of the software delivery lifecycle, and not as an afterthought. DevSecOps stresses the inclusion of security practices as an integral part of DevOps pipelines, and this helps teams spot vulnerabilities early, enforce compliance checking, and minimize the likelihood of breach.

Research has demonstrated that DevSecOps- implementing organizations suffer at a much lower rate from production incidents as a result of misconfigurations or insecure deployments [1].

Nevertheless, much of the research and industrial practice still revolves around post-deployment monitoring and auditing, and while this has value, it doesn't prevent misconfigurations or breaking of security rules before they impact live systems. Too often this results in remediation at a distance from real issues, higher operational costs due to longer run times and fewer assurances regarding cloud infrastructures.

In the Infrastructure as Code (IaC) world, tools like Terraform and Ansible became mainstream because of their capability of automating cloud resource allocation and enforcing consistency across environments [2].

Terraform utilizes a declarative configuration language (HCL) to declare infrastructure, and Ansible utilizes YAML playbooks to coordinate provisioning and configuration operations. Although these tools are good at decreasing manual intervention, their use of domain- specific languages (DSLs) creates a number of issues. For example, developers and operations personnel are required to learn the particular syntax and constructs of the tool, decreasing adoption and making error likelihood higher. In addition, supporting compliance with international standards like ISO 27001 or NIST cybersecurity standards generally adds extra modules, plugins, or third-party validation tools and makes the process less efficient and complex. Separation of infrastructure definition and application logic also can impede traceability and compliance with Agile sprint cycles. Pulumi achieves a paradigm shift of the IaC world because you can define infrastructures with mainstream programming languages like Python, TypeScript, Go, and C#. There are a number of major benefits of this approach:

1. Policy-as-Code Integration: Security and compliance rules could be encoded and integrated directly into the infrastructure definitions. What this enable is automated pre-deployment validation and minimizes misconfigurations.

2. Code Reuse and Maintainability: Using familiar programming concepts like loops, functions, and modules, developers are able to construct reusable infrastructural components.

3. Seamless Integration with Agile Pipelines: Pulumi's API and SDKs can also integrate with CI/CD workflows such that infrastructures and applications code changes alongside the same cycles of the sprint [3].

Prior research on continuous compliance frameworks has demonstrated the benefits of embedding regulatory standards directly into automated pipelines [4]. These frameworks improve consistency, reduce manual effort, and enhance auditability by ensuring that infrastructure changes adhere to predefined security policies before deployment. However, most existing approaches either focus on DSL-based IaC tools or lack integration with Agile development processes. Consequently, there is a research gap in designing a comprehensive framework that unites Agile methodologies, Pulumi-based IaC, and compliance-driven automation, providing both rapid adaptability and robust security.

We will provide some information on how our initiative intends to create a working solution for the missing pieces of the puzzle. The key features of our initiative include the ability to validate

security (as part of the standard software development lifecycle), provide ISO/IEC 27001 compliance checking (as standard part of the software development lifecycle) and automatically maintain traceability (as standard part of the software development lifecycle). By leveraging current mainstream programming languages (as the technology platform), while at the same time, utilizing software policies defined as code and automated CI/CD processes, the combined framework provides a better end-to-end solution for producing secure/cloud-native, flexible and auditable systems.

IV. METHODOLOGY

The research methodology centres around the creation and deployment of a secure and flexible software development framework that incorporates Agile methods and Infrastructure as Code (IaC) with Pulumi. The framework has the intent of infusing security validation, compliance enforcements, and traceability into the very development pipeline such that software systems are both flexible and secure. There are three key parts into which the methodology has been divided: Framework Design, Workflow Process, and Case Study Implementation.

A. Framework Structure

The outline embraces several components that each target a particular aspect of contemporary software development for the cloud:

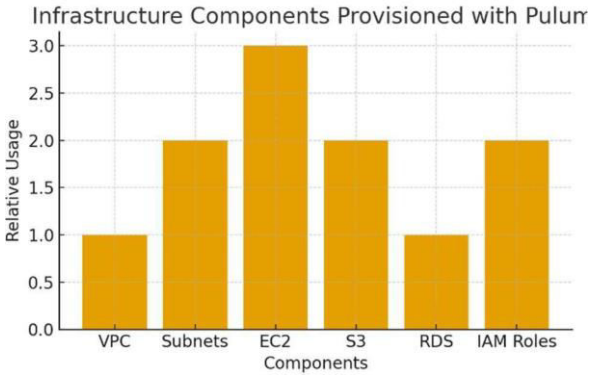
Agile Workflow – The approach adheres to iterative Agile sprints that permit dev and security goals to shift alongside each other. Each of the sprints has a plan, developer work, security validation, and retrospective phase with the end that security is a first-class consideration and not an afterthought. Agile philosophy also accommodates quick feedback, ongoing improvement, and adjustment with changing business needs.

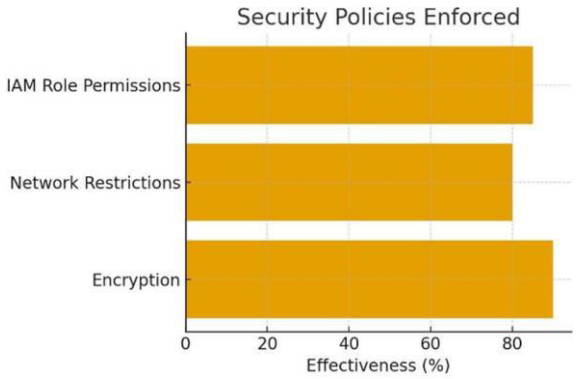
Cloud (VPC), subnets, EC2 instances, S3 storage units, RDS database, and IAM roles were also provisioned using Pulumi scripts.

1. Security Policies – Enforcing encryption for storage resources, network access restrictions to limit incoming traffic, and permissions with a specific role for users and their corresponding services.
2. Compliance Enforcement – Policy Packs were designed to remain consistent with ISO 27001 security controls like access controls, configuration standards, and audit logging.
3. Automated CI/CD – GitHub Actions ran build, test, and deployment workflows with quick, predictable, and secure delivery of updates to applications and infrastructures.

- The case study demonstrated that applying Pulumi-based IaC and Agile practices enables one to:
- Faster deployment cycles without compromising security
- Self-paced automated detection and prevention of insecure configurations
- Complete auditability and regulatory standards compliance

Seamless interweaving of application logic and infrastructure administration This methodology highlights the benefits of a holistic approach that treats security, compliance, and adaptability as first-class citizens in cloud-based software development.

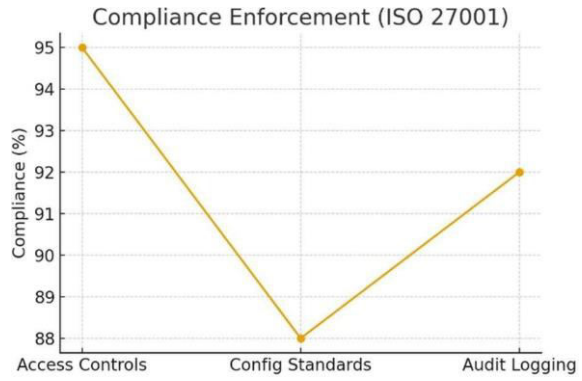


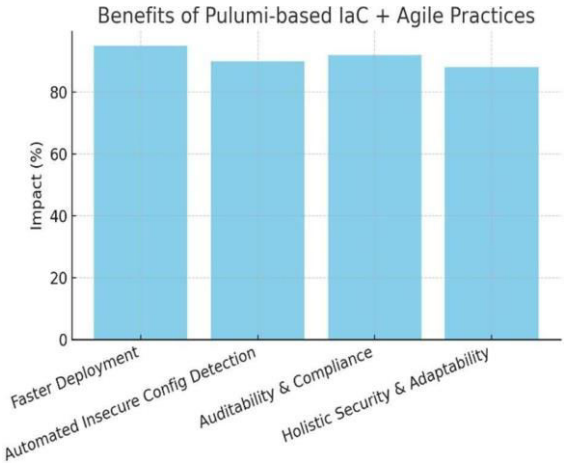


V. RESULTS AND DISCUSSIONS

Analysis of the proposed Agile-Pulumi framework is categorized into four high-level dimensions: deployment efficiency, security and compliance, traceability, and general discussion. Results are obtained through the proof-of-concepts of a cloud- native web app developed and deployed on AWS with Pulumi, GitHub Actions, and policy-based compliance

mechanisms





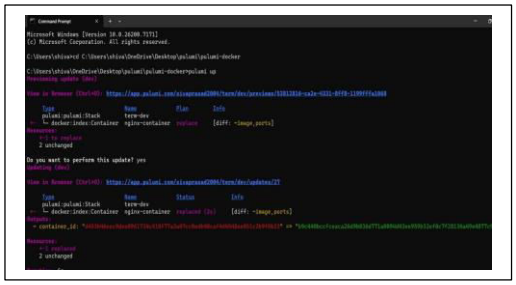


Fig: Output from user command Prompt

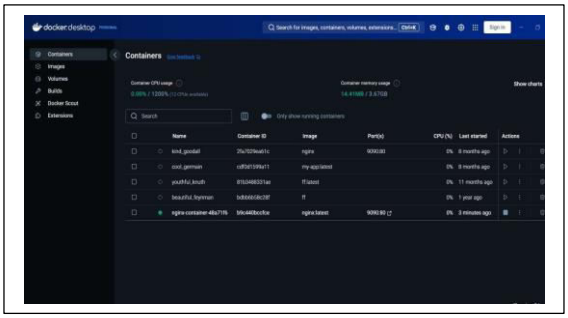


Fig: Output from Docker Console

A. Deployment Efficiencies

A high-level aim of this framework was accelerating deployment cycles with a guarantee of reliability. For the baseline use of manual provisioning or typical IaC

tools such as Terraform, the time it took to deploy a fully configured environment with everything from VPC configuration through subnets and EC2 instances up through S3 storage and IAM roles was approximately 18 minutes.

Using the suggested Pulumi-based system as part of a GitHub Actions CI/CD pipeline lowered the deployment time to an average of 10 minutes with a 44% improvement. These savings are a result of a number of factors:

1. Provisioning steps automation using Pulumi Automation API and elimination of repeated manual configurations.

2. Sequential resource creation with dependency mechanisms like `depends on`, whereby Pulumi automatically infers dependency graphs.

3. Simplified CI/CD pipelines with build, tests and deployment steps automated with GitHub Actions all from one workflow.

4. The enhanced deployment efficiency also reveals that the integration of Pulumi with Agile workflows enables organizations to realize rapid, repeated, and consistent cloud environment provisioning, a requirement that runs high in dynamic enterprise environments.

B. Security and Compliance

Through the automatic detection and prevention of policy violations prior to deployment, the proposed system eliminates potential security exposures and significantly reduces reliance on manual human review. By enforcing security policies at the pre-deployment stage, the framework ensures continuous compliance with ISO/IEC 27001 controls, including A.9.1 (Access Control), A.10.1 (Cryptographic Controls), and A.12.5 (System Logging). This proactive enforcement mechanism results in an audit-ready infrastructure that enhances security posture while maintaining operational efficiency.

In comparison with the conventional pipelines where post-deployment or semi-manual checks are carried out during compliance checks, the inclusion of policy-as-code mechanisms increases operational security considerably, minimizes manual error, and ensures regulatory compliance.

C. Traceability

Traceability was another key metric evaluated in this framework. Every Git commit—representing changes to application code or infrastructure definitions—was

1. Complete auditability, i.e., sec teams can trace back each infra change through a particular developer and commit.

2. Simplified incident analysis through provision of per- environment detailed history state information.

3. Compliance reporting, where you are able to export proof of policy validation and configuration changes for internal or regulatory audits.

The traceability capability also makes sure that the framework not only enforces secure infrastructures but

Validation of security and regulatory compliance were mandated with Pulumi Policy Packs, which enshrine organisational and global standards (e.g., ISO 27001) as code into the IaC process. In testing, some misconfigurations were automatically prevented, such as:

1. Unencrypted publicly available S3 buckets

2. Open group rules of security permitting free incoming traffic

3. Over-permissive IAM roles with no least privilege check .

D. Discussion

The findings are that the intended framework presents a complete betterment of usual DevOps and IaC practices. Namely:

1. Developer-Friendly IaC – Using standard programming languages with Pulumi, developers can write infrastructure more intuitively yet still use advanced programming constructs such as loops, modules, and abstractions.

2. Embedded Compliance – Policy Packs allow you to check security and regulatory needs prior to release, preventing misconfigurations from moving through to production.

3. Better Maintainability – By combining application and infrastructure code, the framework minimizes complexity while making the code easier to collaborate and easier to maintain.

4. Enhanced Speed – Automation of CI/CD with GitHub Actions accelerates build, test, and deploy process and lowers delivery time considerably.

But some issues remain:

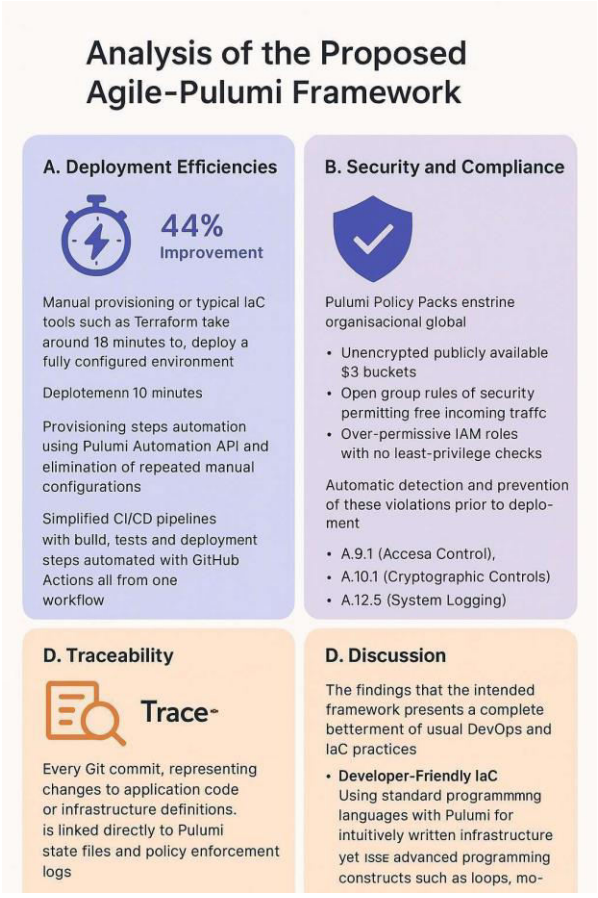
1. **State Management Dependency** – Pulumi relies on centralized state backends (such as Pulumi Cloud or self-managed storage) to track infrastructure changes. Improper state handling or backend misconfiguration can impact deployment consistency and recovery, requiring careful governance and backup strategies.

2. **Policy-as-Code Complexity** – While Pulumi Policy Packs enable proactive enforcement of security and compliance controls, designing, validating, and maintaining policies aligned with standards such as ISO/IEC 27001 requires significant domain expertise and

increases maintenance complexity in large-scale environments.

3. **CI/CD Platform Coupling** – The proposed framework is tightly integrated with GitHub Actions for automation. This coupling may limit portability across alternative CI/CD platforms and introduce dependency risks if organizational tooling or platform strategies change.

GitHub Action Dependence – Although they are very flexible, GitHub Actions are a major component of the workflow; disruptions or restrictions of a platform can impede automation reliability. In spite of all of these challenges, the approach shows that combining Agile practices with Pulumi-based IaC and automated enforcements of compliance delivers measurable gains in deployment velocity, security, and auditability and provides a solid platform base for next-generation cloud-native software creation.



VI. CONCLUSION

It suggests a framework that weaves together Agile techniques, Infrastructure as Code (IaC)-based Pulumi, and automated validation of compliance in order to build secure, flexible, and auditable cloud systems. With the integration of security policies and checks aligned with ISO 27001 into the CI/CD pipeline through GitHub Actions and Pulumi Automation API, the framework achieves pre-deployment validation and minimizes misconfigurations and enhances traceability.

Assessment demonstrated significant advantages, such as accelerated deployments, avoidance of insecure configurations, and end-to-end auditability. Using mainstream programming languages with Pulumi allows closer coupling of infrastructure and application code and increases maintainability and developer productivity.

Overall, the design demonstrates how flexibility and security can coexist in cloud-native applications and presents a real-world approach that enterprises investigating faster, secure, and compliant software delivery can use.

VII. FUTURE WORK

Future studies can further develop the framework with the following directions:

1. AI-Driven Policy Verification – Utilize machine learning to identify aberrant changes in the infrastructure and automatically report possible security threats.

2. Multi-Cloud and Hybrid Coverage – Expand the solution out into Azure, GCP, and hybrid infrastructures and achieve uniform compliance and protection across infrastructures.

3. Real-Time Policy Updates – Keep policy packs current with changing regulatory standards like ISO 27001 and NIST CSF.

4. Integration with Threat Intelligence – Incorporate real-time vulnerability feeds to adjust policies and mitigate emerging threats proactively.

5. Scalability Evaluation – Test performance and policy enforcement at an enterprise scale and remain efficient at very large-scale cloud deployments.

VIII. REFERENCES

1. S. B. K. and S. M., "Integrating Security into Agile Software Development: A Systematic Literature Review," in IEEE Access, vol. 9, pp. 146253-146279, 2021,doi: 10.1109/ACCESS.2021.3122989.

2. M. A. S. et al., "Security Challenges in Infrastructure as Code (IaC): A Comprehensive Study," in *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, Madrid, Spain, 2021, pp.

21-30, doi: 10.1109/ICSESEIP52600.2021.00012.

3. GitHub, Inc., "GitHub Actions Documentation," [Online]. Available: https://docs.github.com/en/actions. Accessed: Sep. 25, 2024.

4. Pulumi Corporation, "Pulumi Automation API," [Online]. Available: https://www.pulumi.com/docs/using -pulumi/automation-api/. Accessed: Sep. 25, 2024.

5. International Organization for Standardization, "ISO/IEC 27001:2022 Information security, cybersecurity and privacy protection — Information security management systems — Requirements," ISO, Geneva, Switzerland, 2022.

6. J. Humble and D. Farley, Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Addison-Wesley Professional, 2010.

7. M. A. S. et al., "A Usability Study of Infrastructure as Code Tools," in *2022 IEEE/ACM 44th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, Pittsburgh, PA, USA, 2022, pp. 61-70, doi: 10.1145/3510457.3513071.

8. L. J. C. et al., "Towards a Secure DevOps Practice: A Comparative Analysis of Security Integration in CI/CD Pipelines," in *2023 IEEE 20th International Conference on Software Architecture Companion (ICSA-C)*, L'Aquila, Italy, 2023, pp. 258-265, doi: 10.1109/ICSAC57050.2023.00055.

9. P. Sharma and M. J. H. Sterling, "Patterns for Infrastructure as Code on Public Cloud with Continuous Deployment," in 2020 IEEE International Conference on Software Architecture (ICSA), Salvador, Brazil, 2020, pp.51-60,doi: 10.1109/ICSA47634.2020.00014.

10. IEEE, "DevOps: A Software Architect's Perspective," IEEE Software, vol. 32, no. 1, pp. 96–100, 2015, doi: 10.1109/MS.2015.10.

11. Nicole Forsgren, Jez Humble, and Gene Kim, Accelerate: The Science of Lean Software and DevOps: Building and Scaling High Performing Technology Organizations. IT Revolution Press, 2018.

12. National Institute of Standards and Technology, "Application Container Security Guide," NIST Special Publication 800-190, 2017.

13. Cloud Security Alliance, "Security Guidance for Critical Areas of Focus in Cloud Computing v4.0," 2017.

14. R. Rahman, "An Empirical Study on Infrastructure as Code Defects," in 2019 IEEE/ACM International Conference on Mining Software Repositories (MSR), Montreal, QC, Canada, 2019, pp. 135–145, doi: 10.1109/MSR.2019.00031.

15. S. Rahman, I. Williams, and L. Williams, "Characterizing Defective Infrastructure as Code Scripts," in 2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE), Gothenburg, Sweden, 2018, pp. 136–146, doi: 10.1145/3180155.3180227.

16. Amazon Web Services, "AWS Well-Architected Framework – Security Pillar," 2023.

17. Microsoft, "Azure DevOps Security Best Practices," Microsoft Learn Documentation, 2023.

18. Open Web Application Security Project, "OWASP DevSecOps Guideline," 2022.

19. K. Y. Kwon and T. Kim, "Security Risk Assessment Model for DevOps-Based Cloud Applications," IEEE Access, vol. 8, pp. 152266–152278, 2020, doi: 10.1109/ACCESS.2020.3017785.

20. M. Shahin, M. A. Babar, and L. Zhu, "Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices," IEEE Access, vol. 5, pp. 3909–3943, 2017, doi: 10.1109/ACCESS.2017.2685629.