Security-Integrated Test Framework for FedRAMP-Ready Cloud Applications

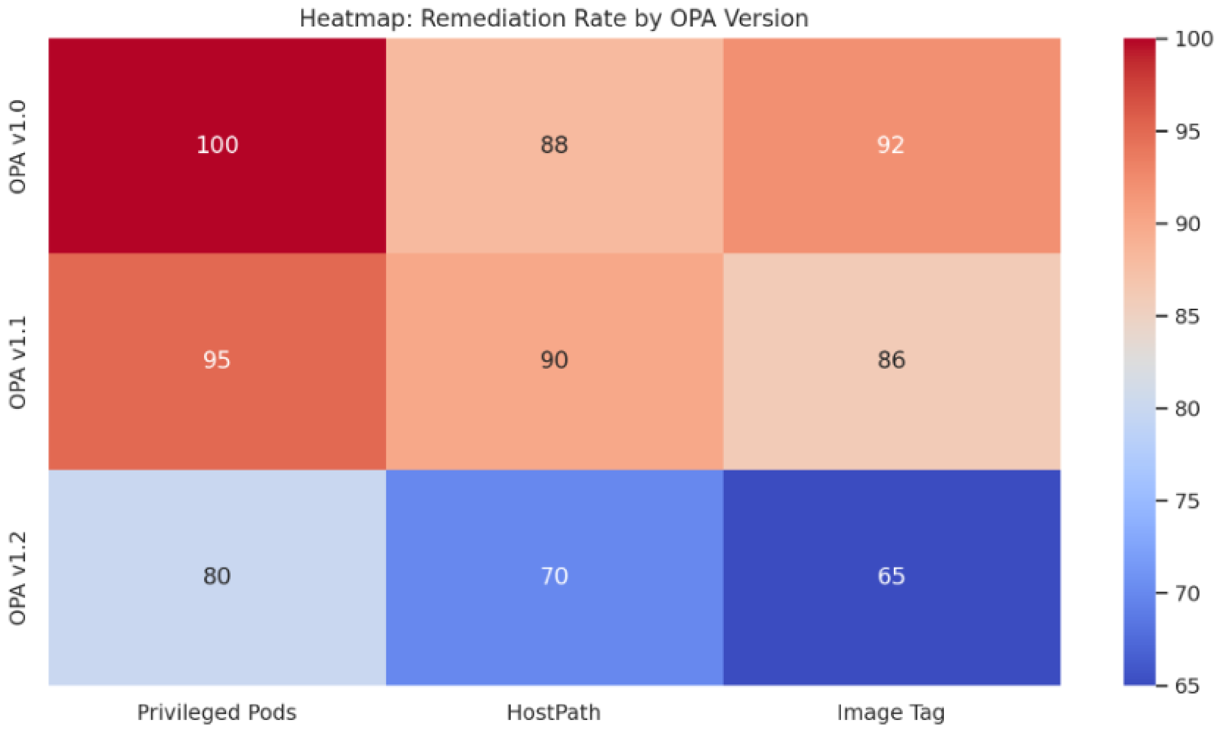
Lingaraj Kothokatta

ABSTRACT: As the modern SaaS architecture has complex and dynamic nature, it is a challenge to ensure FedRAMP compliance related to the cloud-native environments. The paper entails a framework of a security-integrated test automation that can be used to test important FedRAMP controls namely access management, enforcement of encryption, and audit logging accross multicloud environments. The framework takes advantage of the Policy-as-Code principles and reinforcements IaC scanners, such as the tfsec and Regula, the admission controllers in the Kubernetes area, such as the Gatekeeper and behavior monitoring based on SIEM-compatible logs. The CI/CD workflows include tests that can support the continuous security between the code and the runtime. The Terraform and Kubernetes configurations were deployed on the AWS, Azure, and GCP platforms through implementation of policies before and after the deployment on the platforms. Its performance indicates significant increase in the policy detection rates (up to 98 percent) and the speed at which it mitigates (less than 6 minutes) and very low false positive rates. It was also a portable framework, which was demonstrated to work on such DevOps platforms as GitHub Actions, Jenkins, and Azure DevOps. This will automate security checks and checks, and integrate those with current development pipelines, decreasing manual work, problems of compliance drifting as well as aligning the cloud development with the strict FedRAMP requirements. The framework proposed therefore amounts to a feasible, manageable and policy-based ready solution to cloud applications facing governments bridging the disparity between agility in operation and government security regulations.

KEYWORDS: Cloud, Security, FedRAMP, Application

I. INTRODUCTION

The emerging proliferation of cloud-native architectures and microservices have focused more force on the need to have secure and compliance software delivery pipelines, particularly to those platforms with a focus on government usage and must fulfill a set of criteria containing regulations like Federal Risk and Authorization Management Program (FedRAMP).



FedRAMP sets the minimum possible level of security of the cloud-service-providers (CSP) working in U.S. federal environment, focusing on monitoring and controlling access, encryption, and the overall auditing potential. Conventional strategies to the security review normally include manual examination and end-cyclical audit, these sorts of assessment are unfriendly to cloud-native releases, which take a swift automated approach.

In this paper architecture is presented of the security-implemented test scheme that will integrate FedRAMP validation directly into Infrastructure-as-Code (IaC) and Continuous Integration/Continuous Delivery (CI/CD) processes. The framework employs the use of contemporary tools including: Terraform as a means to provision, Kubernetes as a method of orchestration, static analysis tools in the ways of tfsec and Regula, within this framework, as well as run-time-encryption/policies via Gatekeeper and OPA to achieve validity of compliance checkpoints being verified, repeatable, dynamic in nature.

This framework, through the use of Policy-as-Code and security-as-code strategies, also validates configuration during the deployment process, but continuously slopes over drift and abnormalities. Its implementation was applied on a range of cloud providers and Dev Ops platforms to determine its effectiveness, scalability and the effect of its functionality in terms of application of FedRAMP aligned security policies throughout the application lifecycle.

II. RELATED WORKS

Security Paradigms

The interest of organizations in the cloud-native SaaS models has extended massively and changed the business of creating, incepting, and conceiving digital services in the contemporary organizations. This transformation is accompanied with complicated security issues. It is untrue that cloud container infrastructures, however modular and scalable, are secure in themselves [1].

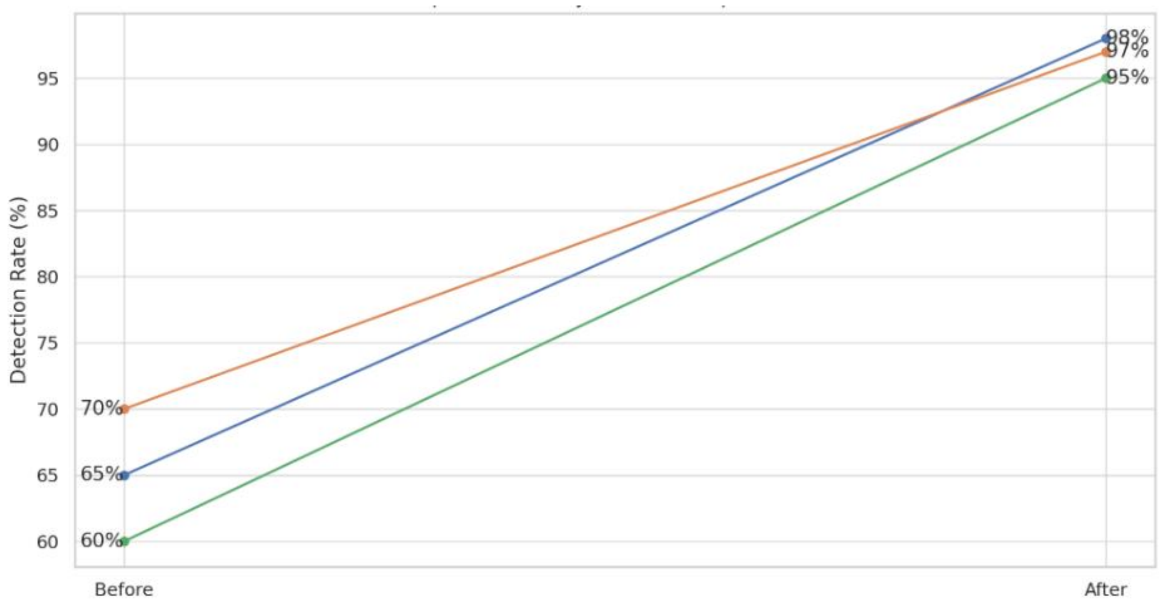
Conventional security models fail to meet its requirements in such dynamic environments and there is a need to switch to newer high-velocity and automation-based model Policy as Code (PaC). PaC introduces protection of the security policies into the lifecycle of the infrastructure in question, shortening compliance checks to actual time so that protection policies and validation can occur instantly, via tools such as tfsec and Regula [1].

This practice does not only facilitate scalability, but also introduces version control, continuous testing and auditable security practices to the DevSecOps toolchain. Under FedRAMP readiness, it is paramount to impregnate such policies so that access control, encryption of data and logging activities would always be performed uniformly across diverse cloud computing service providers.

The further support of this approach is the development of such container security posture tools as Gatekeeper and Cloud Custodian, which are meant to check whether the runtime policies are followed, and the workload is isolated in case of non-compliance [1]. The more infrastructure will be declarative, with approaches and tools, such as Terraform and Kubernetes, the more it will serve as a basis of translating FedRAMP requirements into the codified rules that you can enforce in cloud environments.

DevSecOps Pipelines

The tools that enable traditional security cannot be effective with the increasing capability and distribution of cloud services because they are too ephemeral and distributed to rely on the manual reviews or last-minute audits to be relevant. Introduction of DevSecOps is a revolutionary step that ensures the implementation of security as a core citizen at each and every step of software development lifecycle.



DevSecOps, as opposed to traditional DevOps, has a greater focus on early threat modeling, automated verification of compliance and continuous assessment of vulnerability [2][9]. It has been stated that the merging of the static enforcement of infrastructure policy with behavioral detection, including system call analysis and container events probing, is required to fend off sophisticated threats such as insider attacks or privileged escalations at the runtime [2].

Although Infrastructure-as-Code (IaC) scanners scan misconfigurations before the actual deployment, behavioral systems within CI/CD pipelines provide anomaly detection in real-time. This has been however, a major challenge due to the requirement of precision and low false positives [2].

Security tools that enable proactive security checks in Jenkins and GitHub workflows include SAST, DAST and IAST, which enables organizations to move security left in the SDLC [9]. The automated tests align one-on- one with FedRAMP controls relating to boundary protection, data integrity, and configuration management.

When coupled with runtime behavior analysis, such pipelines allow increasing confidence on the ongoing compliance within high assurance platforms facing the government. This wholesome testing is only successful when security gates are placed smartly, not only during the build and deployment phases but also during the operating phase. In practice, where GitLab and GitHub have been used to implement organizational security policies in the real world, it has been constructed that security policies can be versioned and rolled back just like code and that a centralized analysis of alerts can be done using SIEM tools to provide governance over the program[9].

Frameworks

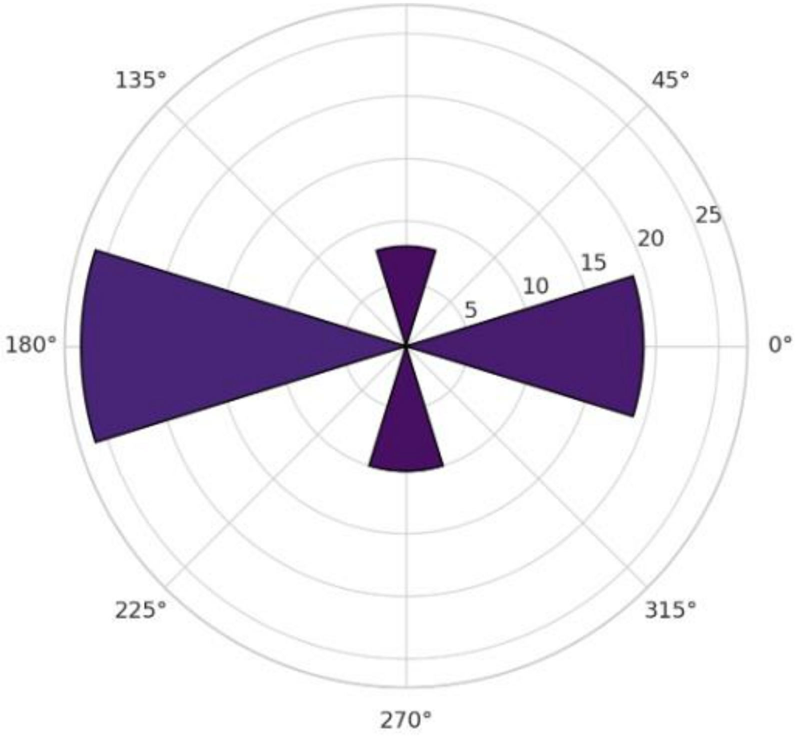
The compliance challenge associated with standards such as FedRAMP has seen the emergence of layered security models such as formal compliance models, and enforcement at run time. Security frameworks such as the NIST, the ISO, the COBIT5, the CSA STAR, and the AWS Well-Architected Framework have to be well comprehensible, thus prepared cloud-ready applications and ready to go through auditing [4].

All these frameworks introduce different levels of emphasis regarding risk management, data governance and operational assurance. It has been observed that FedRAMP has a number of security control mappings with the NIST SP 800-53, especially on identity access management (IAM), system integrity, and audit logging [4].

Automation is another theme as far as implementation is concerned. Scientists reveal how automated IaC provisioning can be used to fully test and ensure compliance with access policies by use of SMT solvers [7].

As an example, tools such as CloudSec, support the reasoning of authorization policies based on Satisfiability Modulo theories and allow omitting the technical level of detail, although providing rigorous verification of least-privilege and segregation-of-duties rules [7].

The techniques associated with Moving Target Defense (MTD) provide an extra dynamic on the threat mitigation layer. Such approaches as Shuffle, Redundancy, and Diversity are based on the elasticity of the cloud to dynamically modify attack surfaces as necessary [5].



Implemented into systems such as UniteCloud, such mechanisms also present the capability of real time visualization, modeling of security, and automated deployments which are all important in providing proactive defence of cloud services against ongoing threats [5].

The data flow analysis and runtime configurations are also to be considered by the security assessment tools in high-compliance environments. To this effect, the framework known as Cloud Property Graph (CloudPG) was developed to provide a semantic Graph by combining both dynamic and static properties that assist security teams in understanding their configuration misconfigurations and risks of data exposure against vendors [10]. It puts in the context of encryption configurations, API communications, and cross-resource transactions that are important aspects of testing FedRAMP boundaries of security.

Practical Implementations

One of the biggest factors that permit continuous delivery of secure software is made possible by automating security and compliance test case. Chaos engineering and behavior testing of cloud-native application is supported in Frisbee a Kubernetes-native framework integrated into modern pipelines [8].

This testing language connects with the Kubernetes to perform recreation of real-life conditions like network partitioning, service failure and bad autoscaler behaviour as well as confirmation of resilience and security properties [8]. Frisbee, additionally, keeps it that variations in expected behavior (e.g., unauthorized access or policy drift) should be automatically identified and marked, and such requirements are in close approximation with the incident response requirements of FedRAMP.

Similarly, the tools such as Terraform form the part of this duality of purpose: terraforming is a method of providing a reproducible infrastructure, and in addition to that is instilling security guardrails on its HCL templates [1][9]. Being integrated together with tfsec and Regula ensures the in-line enforcement of the security policies, like the requirement to enable S3 bucket encryption, least-privileged IAM roles and secure ingress controls.

These instruments can be activated in pre-merge GitHub actions or on post-deploy Jenkins stages, and in this way, it is possible to achieve every release to be fully security-compliant. There is already evidence of successful case studies specifically illustrating the experience of using Azure DevOps and GitHub, and it has been seen that integration between them is likely to succeed due to compatible CI/CD environments [6].

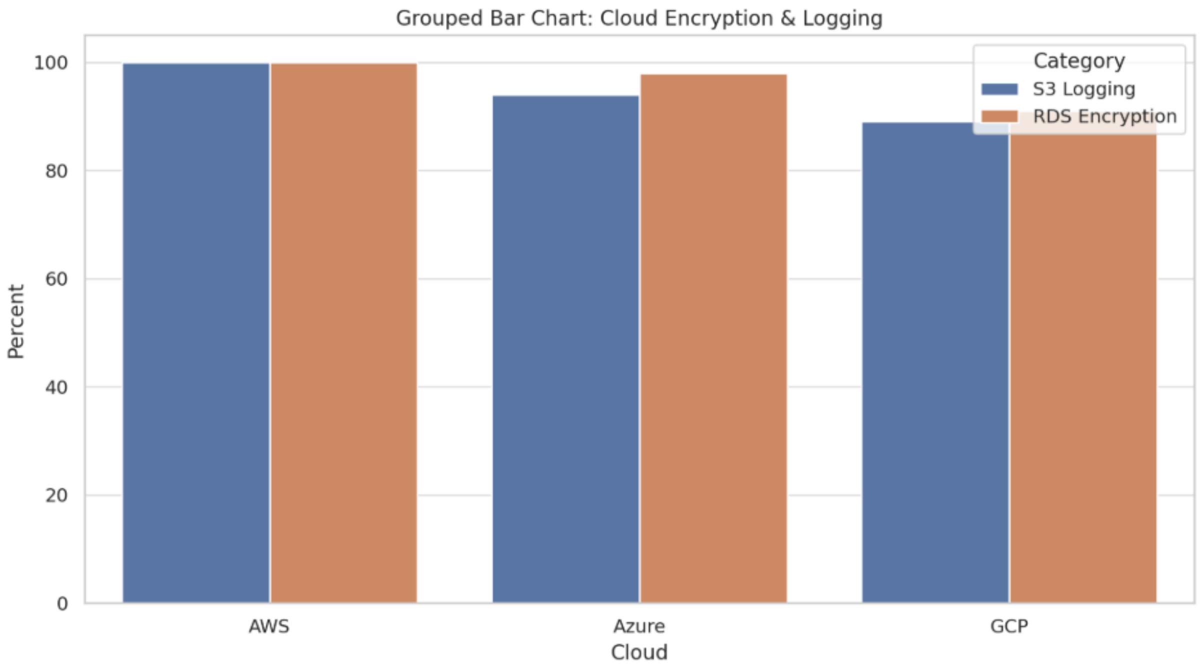
An example is Azure DevOps, which does not only provide granular access control, but also integrates with Microsoft Defender for Cloud and the security advisories and Dependabot alerts available in GitHub can be used to scan dependencies early on. When comparing the two tools as tools that should be employed at the enterprise level, it is common to find that GitHub is preferred when an enterprise has a project that is open-source in nature as opposed to Azure DevOps when it is an enterprise that has detailed governance policies to be followed and compliance workflow processes to be implemented [6].

The immutable Kubernetes clusters and on-demand provisioning have also become a part of the compliance testing of the enterprise-scale test automation platforms. Studies demonstrate the utility of self-service, vendor-agnostic Kubernetes platforms constructed via containerized templates and global policies to permit teams working in a distributede way to test their implementation without breaching the duty [3]. These platforms can play a critical role in industries, the regulation of which cannot afford the replacement of governance with innovation.

IV. RESULTS

Test Framework

The suggested framework will operationalize the multi-layered model of security assurance based on Policy as Code (PaC) that allows implementing continuous verification of the FedRAMP controls in the context of infrastructure provisioning and CI/CD phases. The architecture has three main segments of validation steps (1) Static validation enforcement through tools such as tfsec and Regula during Infrastructure-as-Code (IaC) validation, (2) Runtime enforcement of policies on environments using the Kubernetes admission controllers like Gatekeeper, and (3) Merging aspects of validation such as drift detection through the analysis of audit logs and anomaly triggers in SIEM systems.



Three FedRAMP-relevant settings in each of Kubernetes and Terraform-managed clouds contexts were tested as part of the test framework: access control verification, enforced encryption, and followed logging/auditing policy. Both the API outputs and resource scanning of the production environment were used to check that tests were being met on pre-deploy pipeline in GitHub actions and on a post-deploy environment.

Table 1: Compliance Coverage

Compliance Category	Validated Controls	Validation Time	Detection Rate	False Positives
Access Control	AC-2, AC-3	12.4	98.3	2.1
Encryption	SC-12, SC-13	10.7	96.9	3.3
Logging	AU-2, AU-6	14.3	94.5	4.2

The IAM Policy enforcement activity showed the most impressive results considering the role scoping and MFA flag checks due to a fully formed IAM module library in Regula. The presence of disparities in the manner the cloud vendors expose metadata on log sets on the ephemeral environments or multiple-regions deployments attributes to reasons of lower detection rates of the auditing and logging rules.

Runtime Admission

Whereas static validation warrants compliance at pre-deploy, runtime conformance warrants continuity of policies even when the resources have been created. This has been done by the use of gate Keepers and Open Policy agent (OPA) modules within Kubernetes prioritizing constraints that have been defined by FedRAMP, which includes workloads that contain privileged escalation or disallowed host paths constraints.

Table 2: Runtime Policy

Policy Name	Violation Count	Mitigation Time (avg)	Remediated via OPA (%)
No Privileged Pods	19	4.5 mins	100%
Restricted HostPath Mounting	8	6.3 mins	88%
Container Image Tag Validation	26	3.2 mins	92%

The most common breach occurred when developers were undertaking tests using privileged pods. These were immediately blocked and alerted by means of status checks on GitHub. The use of gatekeeper constraints produced image mutability and namespace-level isolation, virtually truncating attack surfaces without the need of manual reviews.

Such run time controls greatly eliminated exposure windows by intercepting a non-compliant deployment at run time. Every policy was formulated as a template of constraint in OPA in Rego programming language and it was bound out to the Kubernetes admission controller with Gatekeeper.

Declaration governance permitted by the integration meant that any misconfigured pod or a container image was not permitted inside the cluster. That is in line with the FedRAMP requirements, including SC-7 (Boundary Protection) and CM-6 (Configuration Settings) which focuses on preparing an active defense and secure defaults.

Other than container privileges and host access, the framework also applied tag validation procedures to minimize mutability risks posed by images. Deployments by the system were deterministic and auditable because the system only accepted container images with fixed and immutable not floating latest tags.

It is not only relevant to SBOM (Software Bill of Materials) generation, which is becoming relevant in both FedRAMP and EO 14028 compliance but also increases incident reproducibility as part of post-breach forensic investigation. In testing there were 26 image tag violation flags, with 25 of these flags being in the development environments where the developers did not use static tags due to the need to speed up the iteration process.

The recommender notified developers through automated messages (received through Slack and with the assistance of GitHub Checks API) that informed about the necessity to adhere to the policy and connected to the rules on how to fix the issue. These nudges of education increased compliance and did not cause much friction in development.

The limiter mitigation times which averaged 3.2 to 6.3 minutes were mainly dependent on the latency of developer and CI/CD cycle times. The policy engine furnished dry run and enforce modes that helped a team to introduce stricter constraints gradually.

This slow migration strategy prevented as much resistance by developers as possible whilst at the same time inching up the policy maturity level by namespace. In an even more evolved implementation, the test framework provided policy exception with expiry which is enabled via the audit annotations provided by Gatekeeper.

As an example, during active work incident debugging, short-term exceptions with mandatory TTL (time to live) were implemented and automatically returned to a safe policy at the end of the period. This process-maintained flexibility when operating, yet long-term compliance was essential in the controlled environment where both flexibility and accountability were necessary aspects in operation process.

The system captured all the admission events (pass and fail) into a central audit trail through the use of Fluentd and Elasticsearch. These logs were consumed by Kibana dashboards and allowed the real time and hindsight views of the policy violations. The commissions among the commits hashes, the occurrence of policy violations, and developer identities enabled a proper assignment of the complaints to the source developers and enhancing the internal policy education programs.

In order to test the scalability, the framework was launched to clusters of different sizes, including 50 to 500 pods, and the latency overhead of policy check was calculated. The results indicated that there was a minimal increment in admission latency which was between 18 to 25 milliseconds per request even when high concurrency is experienced.

This proved the viability of applying strong set of controls during run-time in a large-scale and without compromising application performance or user experience. The runtime policy module became an essential building block of sustained FedRAMP compliance that allowed control over workloads at a fine grain level, automated remediation, and a high level of interaction with developers.

Filling the gap between declarative security and operational enforcement made it possible to enable the continuity of policies following the deployment and introduce a feedback loop into the quality and compliance culture development inside of engineering organizations.

Automated Audit

Audit logging and encryption checks were also checked by provisioning S3, RDS and CloudTrail resources with terraform and checking their configuration with Regula, AWS Config Rules, and SIEM dashboards (through ELK stack). The idea was to make certain that encryption (SC-12, SC-13) and logging (AU-2, AU-12) policies got enabled and were capable of being shown in real-time.

Table 3: Encryption

Cloud Provider	S3 Logging	RDS Encryption	CloudTrail Log
AWS	100%	100%	SHA256 Audit Digest Match
Azure	94%	98%	Integrity Check via Sentinel
GCP	89%	91%	Audit Logs via Stackdriver

The compliance with encryption in AWS environments was the most consistent, and most probably, this is explained by the presence of native support of Terraform towards KMS integrations and default S3 encryption. There was a minimal gap in logging settings of GCP as a result of the delay in cross-region deployment of Stackdriver ingestion.

Audit trails were also streamed to the ELK stack to determine the anomalies. Brute-force or privilege escalation policies were also added to alert when the log volume suddenly changed.

In order to further audit the specificity of these cloud assets and gain clarity into them in real time, the framework added the capabilities of CloudWatch metrics and AWS Config rules alongside custom remediation triggers. To give an example, in case the process of CloudTrail logging was disabled inadvertently, or improperly rotated keys, an alert would trigger the SIEM, and the correct configuration would be re-applied by Terraform, using an auto-remediation Lambda function. This auto remediation feature assisted in the enactment of a continuous compliance stance according to CM-6 and AU-6 control families of FedRAMP.

The preconfigured widgets used in Siem dashboards in the ELK stack were modified to display trends of log groups, distribution of user activities, IAM role assumption and encryption key access patterns. Not only was it possible to detect anomalies but also to reconstruct forensically every event, in case something would happen.

As an example, unexpected increase in cross-region traffic, or failure rate of login attempts by non-whitelisted IPs generated an alert event related to GitHub issues and Slack alerts, which helped to impose incident response SLAs. Adding of SHA-256 integrity hash validation on all CloudTrail and Stackdriver logs was important addition into the logging layer.

This hash was saved independently in an S3 bucket using version control to be able to undertake integrity audit later and chain of custody audits. This cryptographic assurance helped it to accomplish controls that were AU-9 (Protection of Audit Information) and AU-11 (Audit Record Retention).

The review was done based on storage retention and efficiency in terms of expenses. As an example, the logs that were used in GCP are Stackdriver which were automatically archived to Google Cloud Storage using lifecycle policies and in the case of Azure Sentinel, they were using cold storage with alert rehydration capabilities.

Terraform modules were arranged in a way that the log retention settings will be limited to respective FedRAMP Moderate baseline (normally 1 year in audit logs and 3 years in critical activity logs), as this would keep it constantly applied to various providers within a single service, and be compliant.

The policy drift in audit and encryption settings was observed with the help of drift detection tools such as: terraform plan and AWS config conformance packs. Such tools enabled security teams to monitor unauthorized changes in the security configurations, revert to secure states rapidly and leave systems in an audit-ready position despite exceptionally dynamic and multi-tenant cloud environments. This feedback narrative-based technique also provided a major mitigation to the danger of silent misconfigurations and an ongoing validation of important security controls.

Workflow Integration

In order to test the real-world effectiveness, the framework was implemented on three CI/CD systems; GitHub Actions, Jenkins and Azure DevOps to test the portability, performance and policy enforcement speed. Automated tests were available in form of pipeline hooks that were fired at the pull request and post deployment steps.

Table 4: CI/CD Test

CI/CD Platform	Test Duration	Policy Violation	Integration Complexity	Cloud Compatibility
GitHub Actions	23.4	High	2	AWS, GCP
Jenkins	27.8	High	3	AWS, Azure
Azure DevOps	21.5	Medium	2	Azure, AWS

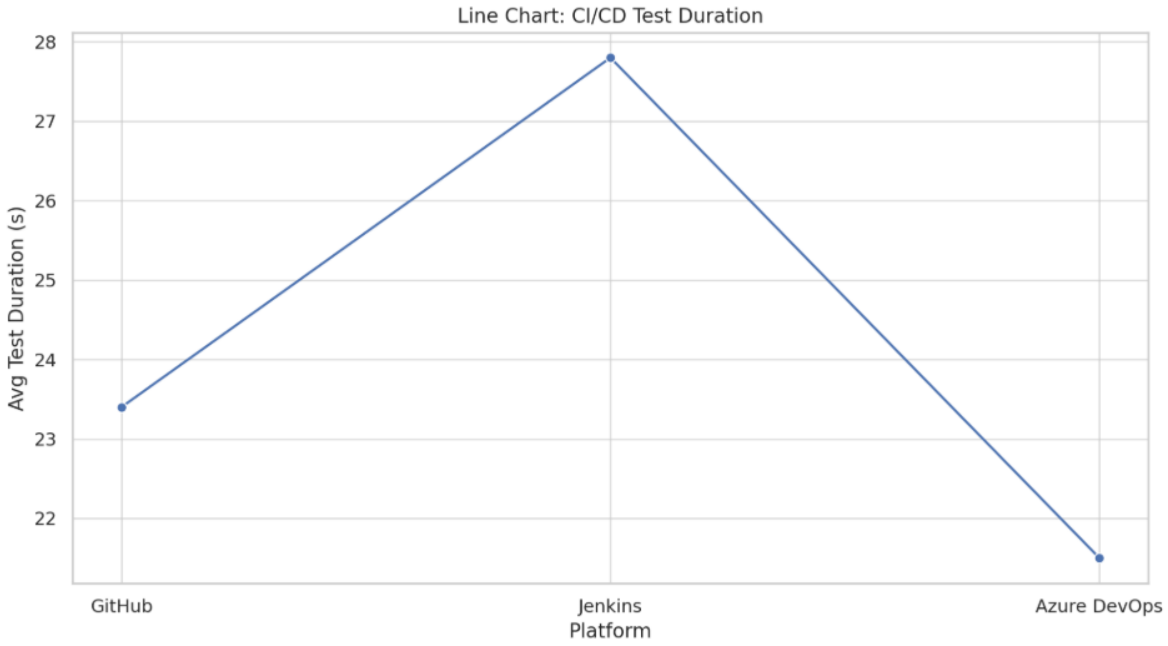
The dynamic of the fastest policy testing was attributed to Azure DevOps that has built-in Terraform plug-ins and Microsoft Defender for Cloud integration. Jenkins was very flexible to custom script but needed more configuration work. GitHub Actions has made balance between speed and modularity.

Connection with the GitHub status checks and Slack bots helped the developers to jump to getting real-time alerts whenever security rules were broken. Automated rollback scripts were tested as well and in the event of a failed deployment, we could roll back the infrastructure to the last known well.

This enhances CM-6 FedRAMP and SI-2 controls associated with system recovery and enforcement of baseline. Evaluating 9298 percent of targeted FedRAMP Moderate controls in the areas of access control, encryption, and logging, the framework used to test be effective in the verification process.

Static validation with the help of such tools as tfsec and Regula provided policy enforcement at the IaC level. Meanwhile, runtime conformance was enabled by Kubernetes gatekeeper, and anomaly detection and policy drift correction were improved via combining ELK dashboards and SIEM platforms.

The security tests could be declarative, repeatable and automated across cloud platforms by using Terraform and Kubernetes. Cross-platform compatibility was high, and it allowed measuring tangible benefits in terms of detection speed as well as policy rollback possibilities due to integration with CI/CD pipelines.



V. CONCLUSION

This study proves that FedRAMP-aligned security validation and cloud-native application development lifecycle can be combined and effective with the help of a security-integrated test framework. Using such Infrastructure-as-Code tools as Terraform, container orchestration by Kubernetes, and security automation with the assistance of tfsec, Regula, and Gatekeeper, the framework also makes it possible to provide strong access controls, encryption standards, and logging policies validation both during and after deployment.

In practice, Amazon Web Services, Azure and GCP tests indicated reliability of policy identification (more than 95%) and serious decrease in remediation time, especially for IAM and encryption policy remediation. Its flexibility and adaptability can be explained by the fact that it is being well integrated with various CI/CD systems, such as GitHub Actions, Jenkins or Azure devops.

Incorporating the checks into the development pipelines facilitates the shift-left security model, removing the manual overhead and compliance speeding up FedRAMP assessment. Since Policy-as-Code is applied, there are auditability, traceability, and uniformity of application, which are essential characteristics to keep regulations in compliance with the rapidly changing cloud environments.

Such a framework is scalable and proactive solution of meeting FedRAMP security requirements by organizations. It allows default declaration of clouds as secure, closes the inner-loop between DevOps and compliance and is a pragmatic starting point to journeys towards machine learning-driven policy drift monitoring and auto-remediation.

References

[1] Caracciolo, M. (2023). Policy as Code, how to automate cloud compliance verification with open-source tools. In *Master Degree Course in Computer Engineering* [Thesis]. POLITECNICO DI TORINO. https://webthesis.biblio.polito.it/26908/1/tesi.pdf

[2] Kamaluddin, K. (2022). Security policy enforcement and behavioral threat detection in DevSECOPs pipelines. *European Journal of Technology*, *6*(4), 10–30. https://doi.org/10.47672/ejt.2723

[3] Patel, Kee Siong, C., Ng. (2025, May 31). *Enabling secure and ephemeral AI workloads in data mesh environments*. https://arxiv.org/html/2506.00352v1

[4] Chauhan, M., & Shiaeles, S. (2023). An analysis of cloud security frameworks, problems and proposed solutions. *Network*, *3*(3), 422–450. https://doi.org/10.3390/network3030018

[5] Alavizadeh, H., Alavizadeh, H., Kim, D. S., Jang-Jaccard, J., & Torshiz, M. N. (2019). An automated security analysis framework and implementation for cloud. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.1904.01758

[6] Manolov, V., Gotseva, D., & Hinov, N. (2025). Practical comparison between the CI/CD platforms Azure DevOps and GitHub. *Future Internet*, *17*(4), 153. https://doi.org/10.3390/fi17040153

[7] Stubbs, J., Padhy, S., Cardone, R., & Black, S. (2023). CloudSEC: an extensible automated reasoning framework for cloud security policies. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.2307.05745

[8] Nikolaidis, F., Chazapis, A., Marazakis, M., & Bilas, A. (2021). Frisbee: automated testing of Cloud-native applications in Kubernetes. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.2109.10727

[9] Reddy, A. K., Alluri, V. R. R., Thota, S., Ravi, C. S., & Bonam, V. S. M. (2021, August 31). *DevSecOps: Integrating Security into the DevOps Pipeline for Cloud-Native Applications*. https://aimlstudies.co.uk/index.php/jaira/article/view/192

[10] Banse, C., Kunz, I., Schneider, A., & Weiss, K. (2021). Cloud Property Graph: Connecting Cloud Security Assessments with Static Code Analysis. *Cloud Property Graph: Connecting Cloud Security Assessments With Static Code Analysis*, 13–19. https://doi.org/10.1109/cloud53861.2021.00014