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

Compliance-as-Code Validation Architectures for Multi-Cloud GxP Platforms

Rajashekar Reddy Gujjula

Abstract

Regulated life-sciences organizations are caught between two systems that were never designed to coexist. Good Practice (GxP) quality frameworks demand a validated state that stays fixed; modern cloud platform engineering is built to change that state continuously, sometimes several times a day. This article argues that the common industry pattern of bolting a compliance-reporting layer onto an already-built Infrastructure-as-Code (IaC) and policy-as-code pipeline cannot validate multi-cloud GxP platforms, because it treats validation evidence as something to be recovered after the fact rather than a property the architecture is designed to produce from the outset. Drawing on operational experience with Cloud Adoption Framework (CAF) landing zones, Terraform-based IaC, Kubernetes cluster engineering, and Azure Policy guardrails, we propose a layered reference architecture in which an enforcement layer, a cross-cloud evidence normalization layer, a validation-artifact generation layer, and an audit and traceability layer are designed concurrently, not sequentially. Evaluation criteria are derived from core GxP validation requirements traceability, attributability, reproducibility, and auditability and the proposed architecture is assessed against them using a practitioner-architecture-evaluation method rather than a controlled experiment. Concurrent design closes gaps that retrofitted reporting layers cannot close by construction, particularly around cross-cloud evidence consistency and installation/operational/performance-qualification (IQ/OQ/PQ) equivalence. GxP cloud governance practice should treat validation-evidence generation as a first-class architectural concern, on par with policy enforcement and infrastructure provisioning; we outline the practical and residual-manual-effort implications of that shift for platform teams operating across multiple cloud providers.

Keywords: *compliance-as-code, GxP validation, multi-cloud governance, policy-as-code, Infrastructure-as-Code, Zero Trust architecture*

1. Introduction

Validated computerized systems in GxP environments are supposed to stay fixed once qualified. Multi-cloud platforms are engineered to do the opposite. That mismatch a validated state that assumes stasis, running on infrastructure that assumes drift is the problem this article addresses. Our position is that compliance-as-code cannot function as a downstream reporting layer stitched onto an existing IaC and policy-as-code pipeline; validation evidence generation has to be architected as a first-class layer, built concurrently with the enforcement layers it is meant to certify.

We encounter this problem directly, not as an abstraction. Platform teams building CAF-aligned landing zones, wiring Terraform pipelines through GitHub and StackGuardian, and running Kubernetes at scale run into the retrofit problem every release cycle: policy-as-code tells you what was blocked, but it rarely tells you, in a form an auditor will accept, why a given configuration was the validated one,

Independent Researcher

who attested to it, and whether the same control produced the same evidence on the second cloud provider as it did on the first. When evidence generation is added after the enforcement layer already exists, the resulting audit trail inherits every gap the enforcement layer's designers never thought to close. That inheritance is the core failure mode this article is written against.

The remainder of the article develops this position across seven sections: regulatory grounding, a tension-based literature review, an evaluation method derived from GxP validation criteria, a four-layer reference architecture, findings from applying that architecture against the criteria, a discussion of what the literature's open tensions imply for practice, and a conclusion. We argue throughout from the platform-engineering and security-architecture side of GxP compliance infrastructure, identity, policy enforcement, and audit-trail mechanics rather than from clinical or pharmacological domains, which sit outside our scope of practice.

2. Background and Regulatory Context

GxP is the umbrella term for Good Practice quality frameworks Good Manufacturing Practice, Good Clinical Practice, Good Laboratory Practice that govern computerized systems supporting regulated processes. At the center of GxP data integrity sits the ALCOA+ principle: data must be Attributable, Legible, Contemporaneous, Original, and Accurate, extended by Complete, Consistent, Enduring, and Available. These are not abstract virtues. Attributability means every configuration change on a GxP platform needs a traceable actor and timestamp; enduring means evidence has to survive infrastructure churn, not just exist at the moment a pipeline runs. For a platform architect, ALCOA+ translates directly into requirements on identity, logging, and immutable evidence storage which is why we treat it as an architectural input, not a checklist applied after deployment.

Validation itself is conventionally demonstrated through Installation Qualification, Operational Qualification, and Performance Qualification IQ/OQ/PQ. IQ confirms a system is installed as specified, OQ confirms it operates within specified parameters, and PQ confirms it performs consistently under real operating conditions. Change control sits underneath all three: any modification to a validated system is supposed to trigger a documented, risk-assessed re-validation path. In a static on-premises system, this model works because change is a rare, scheduled event. Cloud-native platforms break that assumption. Where Terraform applies run continuously and Kubernetes manifests are reconciled on a loop, IQ/OQ/PQ has to be reinterpreted as an ongoing property of the pipeline rather than a one-time gate a reinterpretation most GxP quality systems were never written to anticipate.

The cloud shared-responsibility model complicates this further. Cloud providers assure the compliance of the infrastructure layer; the platform team is left responsible for everything built on top of it network segmentation, RBAC and least-privilege assignment, workload isolation, secure configuration of managed services. In a single-cloud landing zone, that boundary is at least legible: one provider's policy engine, one shared-responsibility matrix, one native evidence format. Multi-cloud GxP platforms multiply the boundary in ways that are easy to underestimate until an audit forces the issue. Each provider's policy-as-code engine expresses guardrails differently, generates logs in different schemas, and offers different native hooks for observability integration none of which were designed with a second provider's evidence format in mind. The validation evidence a GxP quality system needs has to be reconciled across providers before it can be presented as a single, coherent, audit-ready record, and that reconciliation step is

where most of the manual effort in a multi-cloud validation program actually lives.

3. Literature Review

Research on infrastructure code quality and security establishes a strong empirical basis for the enforcement side of compliance-as-code but engages far less directly with the evidence-generation side that GxP validation demands. A systematic mapping study of IaC research identifies testing, defect analysis, and configuration correctness as dominant themes, and notes that empirical treatment of compliance auditability remains comparatively underexplored [1]. This asymmetry recurs across the IaC security literature: the "Seven Sins" study catalogs recurring security smells such as hard-coded secrets in IaC scripts, demonstrating that enforcement tooling is mature enough to detect specific misconfigurations but stops short of producing the kind of attributable, reproducible record a validation auditor requires [2]. Static analysis work on Terraform manifests extends this pattern: alerting tools catch a substantial share of misconfigurations, yet the output they generate is oriented toward remediation rather than evidentiary traceability [9]. The first tension in this literature, then, is that policy-as-code enforcement and validation-evidence generation have been treated as separable problems, when for GxP purposes they are not.

A second tension concerns governance patterns conceived for single-cloud environments and now stretched across multi-cloud estates. NIST SP 800-207 establishes Zero Trust Architecture as a governance model organized around continuous verification, least-privilege access, and segmentation rather than perimeter defense, and its framing has become a reference point for cloud security architecture generally [3]. A recent systematic literature review on Zero Trust implementation challenges contends that organizational and technical barriers to adoption compound significantly in heterogeneous, multi-provider environments, where consistent policy translation across control planes proves difficult in practice [4]. Complementary empirical work on Kubernetes security affirms this point at the workload level: an analysis of security misconfigurations in Kubernetes manifests finds that a nontrivial share of manifests carry weaknesses that policy engines native to one cloud provider may not surface identically on another [5]. A broader empirical study of Kubernetes usage patterns in IT administration and serverless contexts further establishes that operational practices diverge enough across environments to challenge any assumption that a single governance pattern transfers cleanly across clouds [10]. Taken together, this body of work supports Zero Trust as a directional model while leaving open how its guardrails should be operationalized consistently when the underlying policy engines are not the same.

The third tension runs between drift detection and proactive compliance gating. Work on CI/CD pipeline security, including a large-scale characterization of GitHub Actions workflows, documents how frequently misconfigured permissions and exposed secrets persist undetected until a workflow run exposes them illustrating a fundamentally reactive posture toward compliance [6]. Studies of configuration-management tooling extend the same finding to a wider set of ecosystems: task infections in Ansible scripts [7], smelly variables in Ansible infrastructure code [8], replicated security-smell findings across Ansible and Chef [14], and the propagation of security weaknesses through Puppet codebases [15] all establish that misconfiguration, once introduced, tends to persist and spread rather than trigger an immediate gate. Research on developer adoption of security policies in Terraform disputes the assumption that availability equals adoption, since developers often bypass or under-configure the checks provided to them [16], while a broader empirical study of IaC vulnerabilities across ecosystems challenges the assumption that any single tooling layer can catch drift before it

compounds [17]. Adjacent work on continuous auditing, including the MEDINA project's approach to continuous certification of cloud services, argues for moving compliance evidence generation toward real-time evaluation rather than periodic snapshots [13] a direction that intersects meaningfully with GxP's own shift from point-in-time validation toward continuous demonstration of control. Domain-specific applications, blockchain-based data validation in a clinical trial regulatory sandbox [11] and Scrybe's secure audit-trail design for clinical trial data fusion [12], demonstrate that GxP-adjacent industries have already begun building purpose-built evidence layers rather than relying on generic drift detection. That is the direction our proposed architecture extends into the cloud infrastructure layer itself.

4. Methodology / Analytical Framework

We use a practitioner-architecture-evaluation method rather than a controlled experiment or survey instrument. The object under study is an architecture, not a population of subjects, so the usual instruments do not fit. The method proceeds in two stages. First, we derive a criteria set directly from GxP validation requirements rather than from generic cloud-security benchmarks, because the two produce meaningfully different evaluation questions. Second, we assess candidate architectural patterns including the retrofit pattern common in current practice and the layered pattern we propose in Section 5 against that criteria set using structured, evidence-based judgment grounded in operational implementation experience with CAF landing zones, Terraform pipelines (including Terraform Cloud-managed state, where cross-workspace drift proved harder to reconcile than single-workspace deployments), and Kubernetes cluster engineering.

Four criteria anchor the framework, each mapped from an ALCOA+ or IQ/OQ/PQ requirement onto a concrete architectural question. Traceability asks whether every policy decision and infrastructure change can be linked to a specific commit, identity, and timestamp across every cloud in scope, not just the cloud where the change originated. Attributability asks whether the actor behind a change is captured in a form that survives pipeline re-runs and cloud-provider log-retention limits. Reproducibility asks whether a validated state can be reconstructed from stored evidence alone, independent of the live environment the architectural analogue of IQ. Auditability asks whether accumulated evidence can be assembled into an audit-ready package without manual reconstruction. This is where most retrofit approaches show the widest gap.

Each candidate pattern is scored qualitatively against these four criteria along three dimensions: structural sufficiency (does the architecture generate the evidence type by design or only incidentally), cross-cloud consistency (does the evidence look and behave the same across providers), and residual manual effort (how much human reconciliation is required before the evidence is audit-ready). One limitation worth stating plainly: qualitative scoring of this kind depends on the judgment of the practitioners applying it, and a different team with different provider mixes might score the retrofit pattern less harshly on traceability than we do, particularly where a provider's native audit log happens to retain identity metadata longer than the multi-cloud average. We do not claim this method produces a statistically generalizable result. It produces a structured comparison grounded in the same operational vocabulary guardrails, least-privilege RBAC, workload isolation, configuration drift that platform engineers already use to reason about these systems, which we consider a more defensible basis for architectural argument than an abstracted policy framework detached from implementation.

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

Criterion	GxP Requirement Mapped	Architectural Question
Traceability	Change control / commit-to-decision linkage	Can every policy decision and infrastructure change be linked to a specific commit, identity, and timestamp across every cloud in scope?
Attributability	ALCOA+ Attributable	Is the actor behind a change captured in a form that survives pipeline re-runs and log-retention limits?
Reproducibility	Installation Qualification (IQ)	Can a validated state be reconstructed from stored evidence alone, independent of the live environment?
Auditability	Performance/Operational Qualification (OQ/PQ)	Can accumulated evidence be assembled into an audit-ready package without manual reconstruction?

Table 1. Evaluation criteria matrix, derived from GxP validation requirements (author's own analysis).

5. Proposed Architecture

Four layers, designed concurrently rather than sequentially: that is the structure we propose, so that validation evidence is a structural output of the platform rather than a report generated about it after deployment. The first layer is the IaC and policy-as-code enforcement layer itself Terraform modules, Azure Policy definitions, Kubernetes admission controllers the layer most existing tooling already addresses. What differs in our design is that every enforcement point in this layer is required, from the outset, to emit a structured evidence event rather than a pass/fail log line, so the enforcement decision and its evidentiary record are produced by the same code path instead of being derived from it later.

The second layer, cross-cloud evidence normalization, exists specifically because policy engines across providers do not speak the same schema. This layer ingests raw evidence events from each cloud's native tooling and maps them onto a common evidence model, one that captures actor, timestamp, control identifier, and outcome in provider-agnostic form. Without this layer, an organization operating landing zones across two or three cloud providers ends up with two or three incompatible evidence formats, each requiring separate manual interpretation during an audit precisely the retrofit failure mode this article argues against. Normalization has to be built alongside the enforcement layer, because retrofitting a common schema onto years of provider-specific logs after the fact is materially harder than designing the mapping up front.

The third layer, validation-artifact generation, transforms normalized evidence into artifacts that map explicitly onto IQ/OQ/PQ equivalents: installation qualification evidence drawn from Terraform apply records and state reconciliation, operational qualification evidence drawn from policy-as-code enforcement outcomes over a defined observation window, and performance qualification evidence drawn from observability integration data showing the platform operating within specified parameters over time. The fourth layer, audit and traceability, is the layer auditors actually interact with: an immutable, queryable store architected with the same segmentation and

least-privilege access principles applied to production workloads that assembles artifacts from layer three into audit-ready packages on demand, with full lineage back to the original commit and identity. Zero Trust principles apply across all four layers. The evidence layer is itself an attack surface, and a compliance dependency, no less than the workloads it certifies.

Architectural Layer	Enforcement Point	GxP Evidentiary Artifact
Enforcement	Terraform modules, Azure Policy definitions, Kubernetes admission controllers	Structured evidence event emitted at point of policy decision
Cross-cloud normalization	Provider-native log/evidence ingestion	Common evidence model (actor, timestamp, control ID, outcome)
Validation-artifact generation	Terraform apply records, state reconciliation, observability data	IQ/OQ/PQ-equivalent artifacts
Audit and traceability	Immutable, queryable evidence store	Audit-ready package with full lineage to commit and identity

Table 2. Mapping of policy-as-code enforcement points to GxP evidentiary artifacts across the four-layer architecture (author's own analysis).

[Figure 1: Layered Compliance-as-Code Reference Architecture — Enforcement -> Cross-Cloud Evidence Normalization -> Validation-Artifact Generation -> Audit and Traceability, with Zero Trust principles applied across all four layers]

Figure 1. Layered reference architecture (author's own analysis).

6. Results / Findings

Assessed against the traceability criterion, the layered architecture performs materially better than the retrofit pattern, because every enforcement decision carries its evidence event at the point of generation rather than being reconstructed from logs that were never designed with traceability as a requirement. In our qualitative assessment, the retrofit pattern loses commit-to-decision linkage whenever a policy engine's native log rotates, or when cross-cloud correlation depends on matching timestamps across systems with different clock-sync guarantees. The concurrent-design pattern closes that gap by carrying identity and commit metadata through the evidence event itself, rather than reconstructing it from disparate logs afterward.

Cross-cloud consistency shows the sharpest contrast between the two patterns. Retrofit approaches typically produce several separate evidence formats that require manual reconciliation before an audit package can be assembled, since each provider's native tooling was never designed to align with the others. The normalization layer in our proposed architecture collapses this into a single common schema at the point of ingestion, which our qualitative assessment indicates meaningfully reduces audit-preparation effort relative to the retrofit baseline. One caveat: this reduction assumes reasonably consistent provider API behavior over the observation window; a mid-window provider API change (which we observed once, on a policy-log export endpoint, during an actual engagement) forces a normalization-layer update that briefly reintroduces the very manual reconciliation the layer exists to remove. Reproducibility follows a similar pattern to traceability. Validation artifacts in the proposed architecture are generated from stored, normalized evidence rather than reconstructed from live environment state, so a given validated configuration can be rebuilt from the evidence store alone not

reliably true of the retrofit pattern once infrastructure has drifted past the point the original evidence was captured.

Auditability, the criterion GxP quality teams care about most directly, benefits from having the audit and traceability layer designed as a query-ready store rather than an ad hoc log aggregation. In our assessment, this reduces the share of audit findings attributable to missing or unattributable evidence, leaving a residual rate driven mainly by the manual steps discussed in Section 7, rather than by structural gaps in the architecture itself. The overall pattern across all four criteria is consistent: concurrent design does not eliminate every gap, but it converts most gaps from structural impossible to close without redesign into operational. That distinction matters more than it sounds.

7. Discussion

Returning to the first tension identified in the literature review enforcement versus evidence generation treated as separable problems the proposed architecture's central move is to reject that separation at the design stage rather than reconcile it afterward. The IaC security literature is right that enforcement tooling has matured considerably [2] [9], but our findings suggest that maturity in detection does not translate into maturity in evidence generation unless the two are built into the same code path from the start. Where the literature leaves this an open question, our architecture treats it as a design decision: evidence generation is not layered on top of policy-as-code, it is an emission from the same enforcement point.

On the second tension single-cloud governance patterns stretched across multi-cloud estates our findings partly validate and partly complicate the Zero Trust literature. NIST SP 800-207's segmentation and continuous-verification principles [3] translate cleanly onto our cross-cloud normalization layer, but the implementation challenges documented in recent Zero Trust adoption research [4] show up concretely in our architecture as provider-specific policy engine feature gaps. Some cloud providers expose finer-grained policy hooks than others (this is not a minor detail; it is the single largest source of uneven evidence quality we encountered), which means the normalization layer sometimes has to approximate an evidence event that one provider's native tooling captures precisely and another does not. We do not think concurrent design alone resolves this limitation. It is a constraint imposed by the unevenness of provider tooling itself, and organizations adopting this architecture should expect uneven evidence granularity across their cloud estate rather than a uniform result.

The third tension, drift detection versus proactive gating, is where our architecture makes its strongest claim against the retrofit pattern. It is also where residual manual validation steps remain most visible. Continuous auditing approaches such as MEDINA argue for real-time compliance evaluation [13], and our architecture's evidence-at-generation design moves in the same direction, yet full automation of PQ-equivalent evidence still depends on observability integration maturity that varies by workload type. Some validation artifacts, particularly those requiring human judgment about operational risk, resist full automation regardless of how well the underlying evidence layer is designed. We do not claim this architecture eliminates manual validation. We claim it narrows the manual work to genuinely judgment-dependent steps rather than clerical evidence reconstruction a meaningfully different burden for a validation team to carry, and a more defensible one to justify to an auditor.

Limitation	Source	Residual Risk
Provider-specific policy engine feature gaps	Uneven granularity of native policy hooks across cloud providers	Uneven evidence granularity across the cloud estate
PQ-equivalent automation ceiling	Observability integration maturity varies by workload type	Some validation artifacts still require human judgment
Mid-window provider API changes	Native log/export endpoint schema changes during an observation window	Temporary reintroduction of manual reconciliation
Judgment-dependent validation steps	Operational risk assessment requiring human review	Cannot be fully automated regardless of evidence layer design

Table 3. Limitations and residual-risk summary (author's own analysis).

8. Conclusion

The argument here is structural. Validation evidence for multi-cloud GxP platforms has to be produced by the same architectural decisions that produce policy enforcement, not recovered from the exhaust of those decisions after deployment. A compliance-as-code layer bolted onto an existing IaC and policy-as-code pipeline inherits every blind spot the enforcement layer's designers never anticipated needing to prove, and in a multi-cloud estate those blind spots multiply across providers rather than staying contained to one.

The four-layer reference architecture proposed here enforcement, cross-cloud evidence normalization, validation-artifact generation, and audit and traceability is one way to close that gap. Our assessment against traceability, attributability, reproducibility, and auditability suggests it converts most of the retrofit pattern's structural failures into operational ones a platform team can actually manage (though, as Section 7 makes clear, not all of them). It does not remove every manual step, and it does not paper over the unevenness of policy engine capability across cloud providers. What it does is give GxP quality teams and platform engineering teams a shared architectural vocabulary guardrails, least-privilege RBAC, workload isolation, audit readiness for a problem too often split between them, with quality teams inheriting evidence gaps that platform teams never knew they were responsible for closing.

For practice, the implication is direct. GxP cloud governance should stop treating validation evidence as a compliance deliverable produced downstream of platform engineering, and start treating it as a design requirement platform engineering is accountable for from the first architecture decision. That shift changes who is in the room when a landing zone is designed. It changes what "done" means for a policy-as-code control: not merely that it blocks a bad configuration, but that it can prove, on demand and across every cloud in scope, that it did.

Approach	Evidence Generation Timing	Cross-Cloud Consistency	Key Reference
Retrofit compliance reporting	After enforcement, from logs	Low — provider-specific formats	[2], [9]
Zero Trust governance model	Continuous verification, not evidence-specific	Moderate — directional, not operationalized	[3], [4]
Drift detection tooling	Reactive, post-deployment	Low — ecosystem-specific	[6], [7], [8], [14], [15]

Continuous auditing (e.g., MEDINA)	Real-time evaluation	Moderate — standards-based	[13]
Proposed layered architecture	Concurrent with enforcement	High — normalized at ingestion	This article

Table 4. Comparative summary of reviewed governance and compliance approaches against the proposed architecture (author's own analysis, citations as marked).

References

[1] A. Rahman, R. Mahdavi-Hezaveh, and L. Williams, "A systematic mapping study of infrastructure as code research," Information and Software Technology, vol. 108, pp. 65-77, 2019. https://www.sciencedirect.com/science/article/abs/pii/S0950584918302507

[2] A. Rahman, C. Parnin, and L. Williams, "The Seven Sins: Security Smells in Infrastructure as Code Scripts," in Proc. 41st Int. Conf. Software Engineering (ICSE 2019), pp. 164-175, 2019. https://ieeexplore.ieee.org/document/8812041

[3] S. Rose, O. Borchert, S. Mitchell, and S. Connelly, "Zero Trust Architecture," NIST Special Publication 800-207, National Institute of Standards and Technology, 2020. https://www.paloaltonetworks.com/cyberpedia/what-is-a-zero-trust-architecture

[4] M. M. Mushtaq et al., "A Systematic Literature Review on the Implementation and Challenges of Zero Trust Architecture Across Domains," Sensors, vol. 25, no. 19, art. 6118, 2025. https://www.mdpi.com/1424-8220/25/19/6118

[5] A. Rahman, S. I. Shamim, D. B. Bose, and R. Pandita, "Security Misconfigurations in Open Source Kubernetes Manifests: An Empirical Study," ACM Transactions on Software Engineering and Methodology, vol. 32, no. 4, art. 100, 2023. https://dl.acm.org/doi/10.1145/3579639

[6] I. Koishybayev et al., "Characterizing the Security of GitHub CI Workflows," in Proc. 31st USENIX Security Symposium, pp. 2747-2763, 2022. https://www.usenix.org/conference/usenixsecurity22/presentation/koishybayev

[7] A. Rahman, D. B. Bose, Y. Zhang, and R. Pandita, "An empirical study of task infections in Ansible scripts," Empirical Software Engineering, vol. 29, no. 1, art. 34, 2024. https://akondrahman.github.io/files/papers/emse2024-tidal.pdf

[8] R. Opdebeeck, A. Zerouali, and C. De Roover, "Smelly Variables in Ansible Infrastructure Code: Detection, Prevalence, and Lifetime," in Proc. 19th Int. Conf. Mining Software Repositories (MSR '22), pp. 61-72, 2022. https://ieeexplore.ieee.org/document/9796178

[9] H. Hu, Y. Bu, K. Wong, G. Sood, K. Smiley, and A. Rahman, "Characterizing Static Analysis Alerts for Terraform Manifests: An Experience Report," in 2023 IEEE Secure Development Conf. (SecDev), pp. 7-13, 2023. https://ieeexplore.ieee.org/document/10305615

[10] S. K. Mondal, R. Pan, M. H. D. Kabir, T. Tian, and H.-N. Dai, "Kubernetes in IT

administration and serverless computing: An empirical study and research challenges," The Journal of Supercomputing, vol. 78, no. 2, pp. 2937-2987, 2022. https://dl.acm.org/doi/abs/10.1007/s11227-021-03982-3

[11] T. Hirano et al., "Data Validation and Verification Using Blockchain in a Clinical Trial for Breast Cancer: Regulatory Sandbox," Journal of Medical Internet Research, vol. 22, no. 6, e18938, 2020. https://pubmed.ncbi.nlm.nih.gov/32340974/

[12] J. Oakley et al., "Scrybe: A Secure Audit Trail for Clinical Trial Data Fusion," ACM Digital Threats: Research and Practice, vol. 4, no. 2, art. 20, 2023. https://pubmed.ncbi.nlm.nih.gov/37937206/

[13] T. Suhonen and C. Martinez, "Continuous Auditing and Continuous Certification of Cloud Services in MEDINA - Security Auditor's View," Open Research Europe, vol. 3, art. 208, 2023/2024. https://pmc.ncbi.nlm.nih.gov/articles/PMC11196927/

[14] A. Rahman, M. R. Rahman, C. Parnin, and L. Williams, "Security Smells in Ansible and Chef Scripts: A Replication Study," ACM Transactions on Software Engineering and Methodology, vol. 30, no. 1, art. 3, 2021. https://dl.acm.org/doi/10.1145/3408897

[15] A. Rahman and C. Parnin, "Detecting and Characterizing Propagation of Security Weaknesses in Puppet-based Infrastructure Management," IEEE Transactions on Software Engineering, vol. 49, no. 6, pp. 3536-3553, 2023. https://ieeexplore.ieee.org/document/10102545

[16] A. Verdet, M. Hamdaqa, L. Da Silva, and F. Khomh, "Assessing the adoption of security policies by developers in Terraform across different cloud providers," Empirical Software Engineering, vol. 30, no. 3, art. 74, 2025. https://pubmed.ncbi.nlm.nih.gov/40027081/

[17] A. War, A. Diallo, A. Habib, J. Klein, and T. F. Bissyande, "Vulnerabilities in infrastructure as code: what, how many, and who?," Empirical Software Engineering, vol. 30, no. 5, art. 120, 2025. https://link.springer.com/article/10.1007/s10664-025-10672-8