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

Evidence-Graph-Based Continuous Attestation for AWS Platform Foundations

Naga Krishna Reddy Muppidi

Received: 02/6/2024 **Revised**: 17/07/2024 **Accepted**: 5/8/2024

Abstract: Regulated cloud platforms increasingly depend on self-service infrastructure, policy-as-code, and internal developer platforms. A recurring research and operations gap is point-in-time compliance evidence that cannot explain historical platform state. This paper presents a continuous evidence graph for compliance attestation in AWS-centered enterprise platforms. The work models governance decisions as reproducible evidence objects and evaluates them across security, reliability, auditability, and developer-throughput dimensions. The paper contributes a problem formalization, reference architecture, evidence schema, scoring and control-loop design, evaluation methodology, and threats-to-validity analysis. The evaluation design uses controlled multi-account traces and anonymized operational data when available, avoiding unsupported claims about production results.

Keywords: *AWS, cloud governance, regulated financial services, platform engineering, infrastructure as code, policy as code, DevOps, audit evidence, security automation.*

I. Introduction

Financial-services cloud platforms have moved from centralized infrastructure provisioning to self-service operating models. That shift improves delivery speed, but it also distributes control decisions across repositories, pipelines, cloud accounts, identity providers, runtime services, and ticketing systems. The specific issue studied in this paper is point-in-time compliance evidence that cannot explain historical platform state. The issue is difficult because the evidence needed to explain a decision is scattered across systems that were not designed as a single audit or research dataset. A reviewer may see a pull request, a cloud engineer may see a runtime finding, a security analyst may see a control violation, and an auditor may later ask for a durable explanation. Without a model that joins these views, governance becomes a set of manual reconciliations rather than an engineering system. This paper asks how an AWS-centered enterprise platform can turn this recurring problem into a measurable, reproducible, and auditable method. The answer is a continuous evidence graph for compliance attestation. The contribution is a research model that can be implemented with different tooling while preserving a common evidence contract and evaluation methodology.

II. Motivation and Problem Statement

The motivation comes from the interaction between engineering autonomy and regulated control requirements. Developers need fast, predictable paths for infrastructure and platform changes. Security and risk teams need proof that guardrails are effective and exceptions are bounded. Platform teams need operational feedback that shows whether a control improves the system or simply creates friction. The problem domain includes account baselines, CloudTrail events, Config evaluations, control mappings, owners, approvals, and runtime findings. These areas share three properties: they change frequently, they affect multiple teams, and they generate evidence that is only partially visible from any one system. We define a governance decision as a tuple containing a subject, action, resource, environment, policy version, evidence set, decision, owner, and expiration state. A platform is evidence-complete for a decision if another reviewer can reconstruct the decision without relying on human memory. A platform is throughput-aware if the same control loop can measure delivery cost as well as risk reduction. These definitions shape the rest of the paper.

III. Related Work

This work builds on policy-as-code systems, cloud control frameworks, DevOps measurement, infrastructure-as-code, and security governance. AWS Control Tower, AWS Organizations, AWS Config, CloudTrail, and Security Hub provide mechanisms for account governance and detective controls. Terraform and CloudFormation provide declarative infrastructure workflows, while Open Policy Agent and related policy engines provide programmable decision logic. NIST SP 800-53, SP

Independent Researcher
Email: nagamuppidi1015@gmail.com

800-37, and the NIST Cybersecurity Framework describe control, risk, and monitoring concepts. DevOps and SRE literature contributes measurement concepts such as lead time, change failure rate, error budgets, and toil reduction. The gap is that these sources often optimize a single layer. Cloud governance documentation explains available controls; DevOps literature explains delivery performance; audit frameworks explain evidence expectations. Fewer works define a unified empirical model that measures point-in-time compliance evidence that cannot explain historical platform state across control effectiveness, evidence quality, and developer throughput. This paper addresses that gap with a specific evidence schema and evaluation design.

IV. Proposed Model

The continuous evidence graph for compliance attestation has four layers. The signal layer collects events from source control, CI/CD, infrastructure-as-code tools, AWS APIs, identity providers, registries, monitoring systems, and change-management systems. The normalization layer converts those signals into an evidence record with stable fields for owner, scope, environment, policy version, decision, exception state, and remediation link. The evaluation layer applies versioned policy rules and, where appropriate, scoring functions. The action layer routes results to advisory comments, required approvals, deployment gates, remediation queues, or escalation paths. The model separates policy semantics from tool-specific adapters. This separation allows an organization to change CI systems, repository providers, or cloud-account structures without discarding the governance model. It also makes the evaluation clearer because the same model can be tested against multiple baselines.

V. Reference Architecture

The reference architecture contains six components: event ingestion, evidence normalization, policy evaluation, decision storage, workflow routing, and analytics. Event ingestion reads cloud-control-plane events, pipeline results, pull-request metadata, runtime findings, and approval records. Evidence normalization maps those signals into a compact schema. Policy evaluation executes deterministic rules and records the exact policy bundle or version used. Decision storage preserves immutable evidence records. Workflow routing converts decisions into human or automated actions. Analytics computes operational metrics and supports retrospective review. The architecture is designed for AWS-centered platforms but avoids assuming a single tool stack. An organization could implement the evidence store with object storage, a relational database, or a graph database; the research requirement is that records are durable, queryable, and tied to policy versions.

VI. Evidence Schema and Scoring

The evidence schema is the central artifact. At minimum, each record contains: decision identifier, timestamp, account or repository, environment, owner, control domain, resource, source change, policy version, policy result, exception state, expiration, reviewer, and remediation state. The scoring model should be interpretable rather than opaque. For example, a score can combine severity, exposure, age, scope, owner confidence, and evidence completeness. The exact weights should be configurable and evaluated empirically. For Evidence-Graph-Based Continuous Attestation for AWS Platform Foundations, the most important dimensions are evidence freshness, graph completeness, audit query latency, missing owner rate, and audit response time. The model should report both a decision and the reasons for that decision. This is especially important in regulated environments, where an unexplained score is difficult to defend even if it is statistically accurate. Interpretability also helps developers fix the issue without requiring a separate policy consultation.

Core evidence fields for governance decisions

Field	Purpose
Owner	Accountable team or role
Scope	Account, repository, service, or artifact
Policy version	Exact rule bundle used for decision
Evidence	Change, scan, approval, or runtime signal

Field	Purpose
Expiration	Time-bounded exception or review deadline

VII. Evaluation Methodology

The evaluation uses three datasets. The first is a controlled synthetic trace built from representative AWS accounts, repositories, pipelines, and policy decisions. The second is a replay trace generated from realistic infrastructure changes, including approved changes, expired exceptions, unauthorized drift, failed deployments, and emergency remediations. The third is an anonymized historical trace when enterprise data can be used under appropriate controls. Baselines include manual review, binary policy checks, severity-only prioritization, and unstructured ticket-based exception handling. Metrics include evidence freshness, graph completeness, audit query latency, missing owner rate, and audit response time, plus false-positive rate, false-negative rate, reviewer agreement, audit-question response time, and developer lead-time impact. The evaluation includes ablation studies that remove evidence fields, sensitivity analysis for scoring weights, and robustness checks across production, non-production, sandbox, and shared-services scopes.

VIII. Evaluation Outcomes

The evaluation focuses on whether human reviewers spend less time reconstructing context and more time making accountable decisions. The hypothesis is that the model improves evidence completeness when required fields are present, exceptions are time-bounded, and control decisions can be replayed. It may reduce review time for low-risk changes while increasing scrutiny for high-risk changes. The useful outcome is a better allocation of attention rather than a universal reduction in process. Negative results remain important. If a scoring model produces many false positives, the analysis should identify which evidence dimensions caused the problem. If enforcement increases lead time without improving evidence quality, the model should remain advisory.

IX. Discussion

The deeper research question is how regulated platform teams should treat governance as a measurable system. A weak implementation can become another dashboard, but a strong implementation changes operating behavior. Developers get earlier and clearer feedback. Reviewers get context. Risk teams get bounded exceptions. Auditors get reproducible evidence. Platform teams get measurements that show whether a control is healthy. This matters because point-in-time compliance evidence that cannot explain historical platform state is not solved by a single control. It is solved by a feedback loop that can detect drift, explain decisions, and improve policy over time. The model is deliberately conservative: it does not assume perfect data, universal enforcement, or full automation. It assumes that platform governance matures through measured, reviewable increments.

X. Threats to Validity

The primary external-validity threat is that regulated enterprises differ in cloud maturity, control expectations, organizational structure, and tool choice. A second threat is that synthetic traces may not capture the political and social dynamics of production exceptions. A third is that anonymization may remove contextual features that affect reviewer behavior. Construct validity is also a concern: metrics such as lead time and remediation time can be affected by unrelated organizational changes. Internal validity depends on correctly labeling findings, approvals, and outcomes. These threats are reduced by publishing the schema, trace generator, policy examples, scoring functions, and evaluation scripts for the synthetic portion of the study. The analysis distinguishes model benefits from empirically demonstrated results.

XI. Operational Implications

For platform teams, the evidence graph matters only if it changes daily work. The first practical implication is that audit requests shift from artifact collection to evidence replay. Instead of asking individual teams for screenshots, reviewers can query the graph for a control, environment, date range, or owner and retrieve the decision lineage directly. The second implication is that exception management becomes observable. Expired exceptions, missing owners, and stale evidence can be treated as first-class platform defects rather than as side-channel governance debt.

The third implication is organizational. Continuous attestation works best when platform engineering, security engineering, and audit stakeholders share a common evidence contract. The graph does not eliminate policy disagreements, but it makes those disagreements explicit by tying every decision to a visible policy version, evidence set, and owner.

This is a more durable operating model than periodic compliance reconstruction.

XII. Query Workload Examples

An evidence graph is only useful if it answers recurring questions with less effort than manual reconstruction. The most valuable evaluation tasks therefore resemble real audit and operations queries: identify which production accounts lacked a control during a defined window, list which exceptions were active during that window, and show which deployment or policy change caused a control-state transition. These workloads exercise different parts of the graph: temporal joins, ownership lineage, and cross-system control replay.

Table 1 gives a compact query workload that can be used for controlled benchmarking. Query latency is not the only metric. Reviewer completeness, confidence, and number of manual follow-ups matter because an answer that is fast but incomplete still fails the operational goal.

Representative evidence-graph query workloads

Query type	Scope	Expected output
Control replay	Prod accounts	Control state with policy and evidence lineage
Exception audit	Shared services	Owner, reason, expiry, and compensating control
Change trace	Single service	Deployment, approval, and runtime evidence chain
Coverage gap	Org-wide	Missing evidence fields and stale producers

XIII. Conclusion

This paper presented Evidence-Graph-Based Continuous Attestation for AWS Platform Foundations as a model for regulated AWS platform engineering. The contribution is a continuous evidence graph for compliance attestation that treats governance decisions as structured evidence objects connected to delivery and risk metrics. By joining control effectiveness, evidence completeness, and developer-throughput measurements, the approach can make platform governance more auditable and more adaptive. The model supports empirical validation with operational traces, alternative scoring functions, and portability analysis across organizations and cloud-account structures.

XIV. Formal Definitions

Let D denote the set of platform decisions observed during an evaluation window. Each decision $d \in D$ is represented as a tuple (s, a, r, e, p, v, o, x), where s is the subject, a is the requested action, r is the resource, e is the environment, p is the policy identifier, v is the policy version, o is the owner, and x is the exception state. The evidence set E_d contains the source change, scan result, runtime finding, approval record, and remediation state associated with the decision. A decision is reproducible when a reviewer can re-evaluate d using E_d and policy version v without relying on external memory. A decision is stale when its exception state has expired or its owner is no longer accountable for remediation. A control loop is complete when detection, decision, action, and evidence persistence all occur within defined service-level objectives.

We define evidence completeness C_d as the fraction of required evidence fields present for decision d. We define operational burden B_d as a weighted combination of review latency, number of human handoffs, and failed automation attempts. We define residual risk R_d as a weighted combination of severity, exposure, age, and scope. A useful governance model should minimize residual risk and operational burden while maximizing evidence completeness. The objective is not to produce a universal score. The objective is to make the scoring assumptions explicit enough that reviewers can debate, tune, and reproduce them.

XV. Scoring Function Design

The scoring function is interpretable by design. A baseline model can be expressed as

$$Score(d) = w_s S_d + w_e E_d + w_a A_d + w_b B_d - w_c C_d,$$

where S_d is severity, E_d is exposure, A_d is age, B_d is blast radius, and C_d is evidence completeness. The weights w are policy parameters rather than learned constants. This allows a platform governance board to tune the system based on risk appetite, regulatory obligations, and engineering cost. A learned model may be useful later, but a first deployment should prefer transparent

arithmetic. Regulated organizations need to explain why a control blocked a change or allowed an exception. A black-box model may improve ranking accuracy but weaken audit defensibility.

The scoring function should also support mode changes. In advisory mode, the score comments on a pull request, ticket, or finding. In gated mode, a threshold blocks deployment or promotion. In escalation mode, aged exceptions or high-risk decisions notify accountable reviewers. These modes can share the same evidence model while applying different operational consequences. That separation is important because organizations often need to observe signal quality before enforcing policy. Advisory data is therefore not throwaway telemetry; it is the training ground for governance policy.

XVI. Implementation Considerations

A production implementation should begin with one narrow control domain rather than a platform-wide rollout. The first milestone is a stable evidence schema. The schema should be reviewed by platform engineering, security, risk, and developer representatives. Required fields should be minimal: owner, scope, environment, decision, policy version, source evidence, and expiration. Optional fields can be added as the model matures. This prevents the initial rollout from becoming a data governance project before it has proven operational value.

The second milestone is a durable evidence store. The evidence store should preserve immutable decision records, but it does not need to store every raw log line. It can store references to source systems when retention requirements and access controls are clear. Sensitive values should be redacted or tokenized. The store should support queries by account, repository, service, owner, policy, exception state, and time range. Those queries correspond to common audit, incident, and operational questions.

The third milestone is workflow integration. The model should appear where engineers already work: pull requests, CI/CD checks, registry promotion jobs, service provisioning workflows, and remediation queues. A separate dashboard can help leaders, but dashboards rarely change developer behavior by themselves. The decision output should explain what failed, why it matters, who owns it, and what action can resolve it. The explanation is as important as the score.

XVII. Experimental Design

The controlled testbed should include at least 50 simulated accounts and should scale to 500 accounts in stress tests. Accounts should be grouped into production, non-production, sandbox, and shared-services organizational units. The trace should include normal approved changes, unauthorized manual changes, expired exceptions, missing owners, incomplete evidence, and emergency remediations. Each injected condition should have a known ground truth so precision, recall, and reviewer agreement can be measured.

The experiment compares four baselines. The first baseline is manual review using tickets and pull requests without normalized evidence. The second baseline is binary policy-as-code checks that return pass or fail. The third baseline is severity-only prioritization. The fourth baseline is unstructured cloud-security findings without exception ownership. The model is evaluated against each baseline using evidence completeness, response time, remediation time, false escalation rate, and lead-time impact. The analysis reports confidence intervals rather than only point estimates.

A reviewer study uses matched decision cases with and without normalized evidence. The study measures decision time, decision consistency, confidence, and correctness. The hypothesis is that normalized evidence improves consistency and reduces time spent reconstructing context. The study should also test whether too much evidence can reduce decision speed, which motivates a compact summary view and progressive disclosure design.

XVIII. Ablation and Sensitivity Analysis

Ablation analysis should remove one evidence field or scoring dimension at a time. Removing owner should increase unresolved findings. Removing expiration should increase stale exceptions. Removing policy version should reduce reproducibility. Removing environment should increase false prioritization because sandbox and production changes should not be treated equally. Removing source-change links should increase reviewer time. These ablations help show which parts of the model are essential rather than decorative.

Sensitivity analysis should vary scoring weights across severity, exposure, age, blast radius, and evidence completeness. A robust model should not collapse when weights change moderately. If small weight changes produce large changes in prioritization, the model may be too brittle for governance use. Sensitivity analysis also helps expose policy disagreements. For example, security teams may prefer higher severity weights, while platform teams may prefer stronger evidence-

completeness incentives. Making those tradeoffs explicit is part of the contribution.

XIX. Operational Rollout Plan

The rollout proceeds through four phases. Phase one is observation: collect evidence and issue advisory findings without blocking delivery. Phase two is calibration: compare scores with human reviewer decisions and tune thresholds. Phase three is limited enforcement: block only high-confidence, high-risk cases with clear remediation paths. Phase four is expansion: add adjacent control domains and automate recurring remediation. Each phase has exit criteria. Moving to enforcement without measured precision and clear ownership creates resistance and may reduce trust in the platform.

Change management is also part of the rollout. Teams should know which policies are advisory, which are enforced, and how exceptions work. Exception workflows must be time-bounded and visible. A permanent exception is usually a policy failure disguised as risk acceptance. The model should make that visible by reporting exception age and owner accountability. This creates a healthier conversation than periodic manual cleanup campaigns.

XX. Reproducibility Package

A reproducibility package contains a synthetic trace generator, sample evidence records, policy bundles, scoring configuration, and analysis notebooks. The trace generator creates accounts, repositories, resources, owners, approvals, findings, and exceptions. It also labels ground truth so researchers can evaluate precision and recall. The policy bundles include both passing and failing examples. The analysis notebooks reproduce the paper's tables and figures.

The package avoids secrets, real account identifiers, employee names, and organization-specific architecture. Synthetic data is sufficient for reproducibility if it preserves the causal structure of the problem. The analysis separates synthetic-trace results from any operational-trace results.

XXI. Benchmark Definition

The benchmark should not treat every evidence question as identical. Some queries are temporal, such as reconstructing whether a control was effective during a specific incident window. Others are structural, such as proving that a deployment approval, policy version, and runtime observation all relate to the same control objective. The workload should therefore include both timeline reconstruction and lineage reconstruction tasks.

Table 2 shows a compact benchmark layout. Measuring only query latency would miss the larger objective: whether the graph preserves enough relationship structure to reduce follow-up investigation and produce reviewer-consistent answers.

Benchmark dimensions for evidence-graph evaluation

Dimension	Values
Question type	Timeline replay, ownership replay, exception audit, change lineage
Evidence source	CI/CD, control-plane event, runtime finding, approval record
Scope	Single service, account family, production estate, shared service
Answer quality	Complete, partial, stale, unresolved
Reviewer effort	Query-only, query plus one follow-up, manual reconstruction

Disclaimer

The views and opinions expressed in this article are solely those of the author and do not represent or reflect the views, positions, policies, or opinions of the author's employer or any affiliated organization. The content is provided for informational purposes only. The employer makes no representations or warranties regarding the accuracy or completeness of the content and does not endorse or accept responsibility for any statements, conclusions, or materials contained herein.

References

[1] Amazon Web Services, "AWS Control Tower User Guide," 2024.

[2] Amazon Web Services, "AWS Organizations User Guide," 2024.

[3] Amazon Web Services, "AWS Config Developer Guide," 2024.

[4] Amazon Web Services, "AWS CloudTrail User Guide," 2024.

[5] Amazon Web Services, "AWS Well-Architected Framework," 2024.

[6] National Institute of Standards and Technology, "Security and Privacy Controls for Information Systems and Organizations, SP 800-53 Rev. 5," 2020.
[7] National Institute of Standards and Technology, "Risk Management Framework for Information Systems and Organizations, SP 800-37 Rev. 2," 2018.
[8] National Institute of Standards and Technology, "The NIST Cybersecurity Framework 2.0," 2024.
[9] International Organization for Standardization, "ISO/IEC 27001:2022 Information Security Management Systems," 2022.
[10] Open Policy Agent, "OPA and Rego Documentation," 2024.
[11] HashiCorp, "Terraform Documentation," 2024.
[12] B. Beyer, C. Jones, J. Petoff, and N. R. Murphy, Eds., *Site Reliability Engineering*. O'Reilly Media, 2016.
[13] N. Forsgren, J. Humble, and G. Kim, *Accelerate*. IT Revolution, 2018.
[14] J. Humble and D. Farley, *Continuous Delivery*. Addison-Wesley, 2010.
[15] G. Kim, J. Humble, P. Debois, and J. Willis, *The DevOps Handbook*, 2nd ed. IT Revolution, 2021.
[16] Neo4j, "Graph Data Modeling Guidelines," 2024.
[17] Amazon Web Services, "AWS Audit Manager User Guide," 2024.
[18] Amazon Web Services, "Amazon Security Lake User Guide," 2024.
[19] Open Cybersecurity Schema Framework, "OCSF Schema," 2024.
[20] W3C, "PROV-O: The PROV Ontology," 2013.
[21] A. Hogan et al., "Knowledge Graphs," *ACM Comput. Surv.*, vol. 54, no. 4, 2021.
[22] NIST, "Information Security Continuous Monitoring, SP 800-137," 2011.
[23] Amazon Web Services, "Conformance Packs in AWS Config," 2024.
[24] W3C, "RDF 1.1 Concepts and Abstract Syntax," 2014.
[25] Sigstore Project, "Sigstore Documentation," 2024.
[26] NIST, "Secure Software Development Framework, SP 800-218," 2022.
[27] NIST, "Cybersecurity Supply Chain Risk Management Practices, SP 800-161 Rev. 1," 2022.
[28] OpenSSF, "Supply-chain Levels for Software Artifacts (SLSA) Specification v1.0," 2023.
[29] The Linux Foundation, "SPDX Specification v3.0," 2024.
[30] OWASP Foundation, "CycloneDX Specification v1.5," 2023.